منشورات جامعة دمشق
كلية العلوم

الخوارزميات وبنى المعطيات (1)

الدكتور
راكان رزوق

الدكتورة
مادلين عبّود

الدكتور
عمّار خيربك

جامعة دمشق

١٤٣٤ - ١٤٣٥ هـ
٢٠١٣ - ٢٠١٤ م

منشورات جامعة دمشق
كلية الهندسة المعلوماتية

الخوارزميات وبنى المعطيات

(1)

الدكتور رياكان رزوق
الدكتورة مادلين عبّود
الدكتور عمّار خيربك

جامعة دمشق
١٤٣٢ - ١٤٣٣ هـ
٢٠١١ - ٢٠١٢ م

وقل رب زدني علما
جامعة دمشق
Damascus University

مقدمة الكتاب

تعدُّ مادّة الخوارزميّات وبنى المعطيات من الركائز الأساسية في تكوين المعلومياتيّين، وهي موجّهة إلى الدارسين الذي عرفوا المبادئ الأساسية في البرمجة، فيكمّلون معارفهم البرمجية عن طريق تعرّف الخوارزميّات الأساسية، وتعميق منهجيّة حل المسائل البرمجية. الغاية من هذه المادّة تعليم الدارسين اتّباع الطرق المعيارية في حل المسائل البرمجية عن طريق تطبيق متتالية من الخطوات:

- تعريف دقيق للمسألة،
- إيجاد الحل عن طريق متتالية من عمليّات التبسيط،
- تحليل الحل الناتج.

اقتصرت البرمجة في بداية عصر المعلوماتية على مهارة المُبرمجين ممّا جعل الاختلافات في طرق حل المسائل كبيرة بقدر عدد المبرمجين، وأدّى إلى كثرة الأخطاء في الحلول الناتجة. ومع تراكم الخبرات في هذا المجال، وظهور أعمال الأكاديميّين من أمثال Dijkstra وHoare، تحوّلت البرمجة من خبرة ومهارة إلى علم له مبادئه ومنهجيته.

تتناول موضوعات هذا الكتاب منهجيّة إنشاء البرامج وتحليلها، فقد ركّزنا الاهتمام في مفهوم البرنامج كتعبير عن خوارزمية مجرّدة، تتعامل مع معطيات مهيكلة، كما ركّزناه

في الترابط الوثيق بين الخوارزميّات وبنى المعطيات التي تعالجها. وتناولنا بعض المفاهيم الأساسية في البرمجة، نحو توصيف الخوارزميّات، وحساب درجة تعقيدها.

لقد قسمنا موضوعات الكتاب على جزأين غير مستقلين: الجزء الأول يحتوي على أربعة فصول ويلمُّ على المفاهيم الأساسية في الخوارزميات وبعض أنواع الخوارزميات الشهيرة، أمّا الجزء الثاني فيحتوي على خمسة فصول ويلمُّ على بعض بنى المعطيات والعمليات الأساسية عليها.

يتناول الفصل الأوّل مفهوم الخوارزمية، فيعرض المبادئ الأساسية للتعبير عن الخوارزميّات وتطبيق هذه المبادئ في بعض الأمثلة.

أمّا الفصل الثاني فيحاول وضع منهجية صورية مبسطة، من أجل حساب درجة تعقيد الخوارزميات ومقارنة فعالية تنفيذ المسائل البرمجية.

ويعرض الفصل الثالث صفاً متميّزاً من الخوارزميّات، وهو الخوارزميّات العَوديّة، إذ يبيّن حالات يكون فيها التعبير عن الحل بأسلوب عوديّ أمراً طبيعيّاً في حين تكون محاولة إيجاد خوارزمية تكرارية لحل المسألة نوعاً من إضاعة الوقت. يتضمّن هذا الفصل طريقة منهجية لتحويل الخوارزميّات العوديّة إلى خوارزميات تكرارية مكافئة.

ثمّ يعرض الفصل الرابع صفاً آخرَ من الخوارزميات أكثر تعقيداً، وهو الخوارزميات التراجعية، التي تساعد في إيجاد أحد حلول مسألة معيّنة، ومن ثمّ إيجاد الحلول كافّة، وأخيراً إيجاد أفضل حل للمسألة المطروحة.

وفي الفصل الخامس نبدأ دراسة بنى المعطيات بإعطاء فكرة عن نمط المعطيات المجرّد وشرح بنى المعطيات الأساسية.

أمّا الفصل السادس فيلمُّ على بنى المعطيات الخطية كالأشعة والمكدسات والأرتال وتفصيل العمليات الأساسية على هذه البنى.

وفي الفصل السابع نعرض أهم الأفكار حول بنى المعطيات الهرمية ونركّز الاهتمام في الأشجار الثنائية وأهمّ العمليات عليها.

أمّا في الفصل الثامن فسنتعرض لبنى معطيات هرمية أكثر تعقيداً، وهي البيانات، حيث نستعرض أيضاً أهمّ العمليات التي نجريها عليها.

وأخيراً وليس آخراً يعرض الفصل التاسع بنى معطيات بسيطة، وهي جداول التقطيع، لكن ذات فائدة كبيرة في اختصار زمن البحث عن المعطيات، وكذلك أهم طرق التقطيع.

يحوي كل فصل من هذه الفصول، مجموعةً كبيرة من الأمثلة المحلولة على شكل خوارزميات وفق الطرق المتبعة، وترجمة هذه الخوارزميات إلى برامج مكتوبة بلغة باسكال المعيارية. كما يحتوي كل فصل على مجموعة من التمارين المنتقاة غير المحلولة، من أجل توسيع مدارك الطلاب حول مواضيع الكتاب.

في نهاية الكتاب أوردنا مجموعة من المراجع التي ساعدتنا في إنجازه، والتي يستطيع القارئ الرجوع إليها للحصول على معلومات إضافية حول الموضوعات التي يتناولها.

دمشق في 11 أيلول 1999

المؤلفون

وقل رب زدني علما
جامعة دمشق
Damascus University

فهرس المحتويات

وقل رب زدني علما
جامعة دمشق
Damascus University

الجزء الأول

في الخوارزميات

وقل رب زدني علما
جامعة دمشق
Damascus University

مقدمة الجزء الأول

عرف الإنسان منذ القديم الكثير من الخوارزميّات لحل مسائل على الأعداد، مثل خوارزميّة إقليدس لإيجاد القاسم المشترك الأعظم لعددين صحيحين، وخوارزميات حل معادلات الدرجة الثانية. ومع دخول عصر المعالجة الآلية للمعلومات، استُنبطت خوارزميّات تتعامل مع معطيات ليست بالضرورة رقميّة، كخوارزميّات الفرز التي تسمح بترتيب قائمة من الأسماء ترتيباً أبجدياً، أو خوارزميّات البحث عن متتالية من الأحرف ضمن نص، أو ترتيب المهام المطلوب تنفيذها في إطار مشروع معيّن، بما يسمح بتوصيف ترابط هذه المهام الجزئية ضمن الإطار العام للمشروع.

يُعدُّ مفهوم الخوارزميّة من أهم المفاهيم في المعلوماتية، وإذا عدنا تاريخياً نجد أن مفهوم الخوارزميّة قد سبق ظهور الحاسبات والحواسيب بآلاف السنين، فقد وضع البابليون في عهد حمورابي (حوالي 1800 ق.م.) أوّل توصيف صوري لقواعد حلّ بعض أنواع المعادلات. أمّا كلمة خوارزميّة فهي نسبة إلى الرياضي العربي أبي جعفر محمد بن موسى الخوارزمي، الذي عاش في بغداد بين 755 و 847 م وعمل في الرياضيات والجغرافيا والتاريخ. ومن أهم أعماله في الرياضيات كتب "الحساب"، و"الجبر والمقابلة"، و"السند هند الصغير"، وهو كتاب في المثلّثات. وقد شرح في هذه الكتب عدداً من الطرق لحل المسائل. وقد كانت مؤلّفاته مرجعاً للعلماء من بعده.

وفي الثلاثينيّات من هذا القرن قام عدد من الرياضيين من أمثال Godel وChurch وturing وPost بإنشاء توصيف صوري للخوارزميّات وللتوابع الحسوبة (القابلة للحساب) وذلك باستعمال عدّة نماذج مجرّدة كالتوابع الوديّة وآلة Turing ونظام Post والآلات ذات المسجّلات. وتهدف جميع هذه النماذج إلى إيجاد طريقة للحكم على مسألة معيّنة بأنها قابلة للحل بواسطة خوارزميّة أو لا.

سنعرض في هذا الجزء مسائل يمكن حلها بواسطة خوارزميّات، وسنسلّط الضوء على درجة تعقيد هذه الخوارزميّات، كما سنتطرق بالتفصيل إلى نوعين مهمين من الخوارزميات: الخوارزميات الوديّة والخوارزميات التراجعية.

الفصل الأول

مفاهيم أساسية في الخوارزميات

1-1 تعاريف

خوارزميّة (Algorithm) حلّ مسألة، هي توصيف صوري لطريقة الحل على شكل متتالية منتهية من العمليات البسيطة، تنفّذ حسب تسلسل محدّد.

البرنامج (Program) هو توصيف لخوارزميّة حلّ مسألة معيّنة بإحدى لغات البرمجة التي يقبلها الحاسوب.

لغة البرمجة (Programming Language) هي مجموعة من المفردات والقواعد والدلالات المعرّفة التي تسمح بكتابة برنامج يمكن تنفيذه على الحاسوب.

المترجم (Compiler) هو برنامج يفهم البرنامج المكتوب بلغة برمجة معينة، ويحوله إلى برنامج مكافئ مكتوب بلغة المجمع (Assembly Language) الخاصة (أو أحياناً لغة الآلة نفسها) بالمعالج الصغري (Microprocessor) للحاسوب.

1-2 التعبير عن الخوارزميّات

كي تصبح الخوارزميّة قابلة للتنفيذ على الحاسوب يجب ترجمتها إلى تعليمات وفق لغة برمجة متوفّرة في الحاسوب. ويجب تأكيد استقلال الخوارزميّة عن لغة البرمجة

المستخدمة، فخوارزميّة إقليدس لإيجاد القاسم المشترك الأعظم لعددين، تبقى نفسها بأي لغة كُتبت سواء بـ PASCAL أو ADA أو LISP.

نورد فيما يلي بعضَ القواعد في منهجية تحويل الخوارزميّات إلى برامج. وقد استُنبطت هذه القواعد من منهجية عامّة في البرمجة تسمى البرمجة المهيكلة (Structured Programming) وتساعد في إنشاء برامج سهلة الفهم والتعديل والصيانة.

1-2-1 سهولة قراءة البرنامج

يمكن تحسين شكل البرنامج باتّباع قواعد تنسيق التعليمات، وإضافة تعليقات توضّح الهدف من البرنامج. يبيّن الشكل التالي مثالاً لبرنامج كتب بشكلين مختلفين أحدهما أسهل قراءةً وفهماً من الآخر.

```
i := 0; s :=0; while i<n do
begin i:= i+1; if T[i] > 0 then
s:=s+T[i] else s:=s-T[i] end;
```

```
{ يحسب البرنامج التالي مجموع القيم المطلقة لعناصر T }
i := 0;
s := 0;
while i<n do
     begin
          i:= i+1;
          if T[i] > 0 then
                    s := s + T[i]
          else
                    s := s - T[i]
end;
```

1-2-2 تجزئة الخوارزميّة إلى وحدات مستقلّة

من أجل الوصول إلى مرونة في التعديل والصيانة، نجزّئ الخوارزميّة إلى وحدات (Modules) صغيرة ومستقلّة، تؤدي كل منها مهمّة محدّدة. في هذه الحالة يجب تحديد واجهات (Interfaces) الربط بين مختلف الوحدات، وكذلك بين كل وحدة والخوارزميّة المتكاملة. يجري ذلك بتحديد متحوّلات الدخل والخرج لكل وحدة، وكذلك بتحديد المتحوّلات العامّة المستخدمة في كل وحدة.

لإظهار الارتباطات بين الوحدات المختلفة، تحدّد طريقة استخدام هذه الوحدات بعضها بعضاً، وطريقة الاستخدام المشترك للمتحوّلات العامّة.

1-2-3 تجنّب تعليمات التفريع

يبرهَن على إمكان كتابة أي برنامج دون استعمال تعليمات التفريع مثل Goto أو Exit أو غيرها، ويمكن الاستعاضة عن هذه التعليمات بما يكافئها من التعليمات الشرطية أو باستدعاء إجرائيات (Procedures).

1-2-4 الانتباه إلى الخوارزميّات العَوديّة

يسمح التعبير عن طريقة حل مسألة معيّنة بخوارزميّة عوديّة (Recursive Algorithm) باستعمال تقنيات البرهان بالتدرّج، لإثبات صحّة طريقة الحل (أو خوارزميّة الحل) إثباتاً صورياً أنيقاً، كما تسمح البرامج العوديّة بحل مسائل يصعب حلّها باستعمال خوارزميّة تسلسلية (Sequential Algorithm) تقليدية. ولكن يجب الانتباه عند استعمال الخوارزميّات العوديّة، إلى اعتماد منهج يضمن انتهاء المعالجة بعد حدٍ معيّن.

1-2-5 التوصيف الجيّد للخوارزميّات

يهدف توصيف الخوارزميّة إلى شرح ما تقوم به هذه الخوارزميّة دون الإسهاب في طريقة إنجازها للمطلوب.

مثال

سنعرض طريقة توصيف خوارزميّة على ضوء مثال لخوارزميّة تقوم بالبحث التسلسلي لعنصر ضمن قائمة:

```
{ يقوم هذا البرنامج بالبحث عن موضع العنصر X في القائمة L }
{ في حال عدم وجود العنصر X في القائمة تكون النتيجة صفراً }
{ تحوي القائمة n عنصراً}
j := 1;
while ( j <= n ) and ( L[j] <> X ) do
      j:= j+1;
if j > n  then
      j:= 0;
```

أعطينا توصيف الخوارزميّة على شكل تعليقات مكتوبة ضمن أقواس كبيرة في بداية البرنامج. ولا بدّ للقارئ المتيقّظ أن يلاحظ قصور التوصيف المعطى وعدم دقّته، لأنّه لا يجيب عن الأسئلة التالية:

- هل يمكن أن تكون القائمة L فارغة ؟
- ماذا يحدث لو وُجد العنصر X أكثر من مرّة في القائمة L ؟
- ما هي المعطيات (Data) وما هي النتائج ؟

لسدّ هذه النواقص نقترح التوصيف التالي:

- المعطيات هي L وX .
- القائمة L تحوي n عنصراً ويمكن لهذه القائمة أن تكون فارغة (n = 0).
- تبحث الخوارزميّة عن دليل العنصر X ضمن القائمة L.
- عندما تتوقّف الخوارزميّة يحوي المتحوّل j دليل أوّل ظهور لـ X في L.

  وإذا كان X غير موجود في L فإن j يحوي القيمة صفر.

يمكن في بعض الحالات أن يحوي توصيف الخوارزميّة عبارات حسابية، مثل التعبير عن مجموع القيم المطلقة لعناصر القائمة L بـ:

$$S = \sum_{i=1}^{n} |L[i]|$$

أو عبارات منطقية كما في حالة مثالنا:

$(Card(L) = n) \& (n > 0) \& not(X \text{ in } L) \Rightarrow j = 0$
$(X = L[k]) \& (1 \le i \le k \Rightarrow X \neq L[i]) \Rightarrow j = k$

التوصيف الصوري للخوارزميّة هو توصيف يقتصر على عبارات حسابية ومنطقية، ويُعتبر هذا التوصيف أدق أنواع التوصيف، إذ لا يدع أي مجال للالتباس، ويساعد في البراهين الرياضيّة المتعلّقة بانتهاء المعالجة أو بصحّة الخوارزميّة.

1-3 أمثلة

ليس الهدف من الأمثلة التي سنعرضها هنا هو حل المسائل، بقدر ما هو عرض الطريقة المقبولة التي يجب اتباعها في حل مسائل الخوارزميات.

1-3-1 مسألة الفرز بالدمج

ليكن لدينا الجدول T، عناصره من الأعداد الصحيحة غير المرتبة، ونريد ترتيب هذه العناصر ضمن الجدول بطريقة الفرز بالدمج (Merge Sort). وتتلخص هذه الطريقة بما يلي: نقسّم الجدول إلى جدولين جزئيين، ونرتب كل جدول جزئي على حدة، ثمّ نقوم بدمجهما في جدول مرتب واحد. عملية ترتيب كل جدول جزئي هي تطبيق لنفس الطريقة لكن عند مستوى أقل. نتابع هذه العملية وصولاً إلى جداول جزئية من مرتبة خانتين فقط، فهذه يمكن ترتيبها بسهولة.

نحن إذن أمام مسألة عوديّة، نلجأ في حلها إلى الخطوات التالية:

1- بنى المعطيات المستخدمة

نعرّف الجدول بواسطة تسجيلة (Record): يدّل الحقل الأول على عدد الخانات المملوءة فعلاً (بُعد الجدول). أمّا الحقل الثاني فيحوي الجدول نفسه.

```
const
    n = 100;
type
    index = 1..n;
    TableType = record
                  dim   : index;
                  Table : array[index] of integer
                end;
```

2- الخوارزمية

نريد إذن كتابة خوارزمية الفرز بالدمج.

الترويسة

```
procedure MergeSort (low, high : index; var T :
TableType);
```

الدخل

جدول T من الأعداد الصحيحة غير المرتبة. أدلة الحدّ الأدنى والحدّ الأعلى للجدول.

الخرج

نفس الجدول T لكن مرتباً تصاعدياً.

مخطط الخوارزمية

1- مادام شرط التوقف غير محقق

1-2- احسب دليل منتصف الجدول T وضع القيمة في المتحول mid.

1-2- رتّب الجدول الجزئي T[low .. mid] بطريقة الفرز بالدمج.

1-3- رتّب الجدول الجزئي T[mid+1 .. high] بطريقة الفرز بالدمج.

1-4- ادمج الجدولين الجزئيين المرتبين في جدول وحيد مرتب T[low .. high].

ترميز الخوارزمية

لسهولة مخطط الخوارزمية لا داعي لكتابة ترميز لكل مرحلة. لذلك نجد فيما يلي الترميز الكامل للخوارزمية:

```
procedure MergeSort (low, high : index; var T :
TableType);
var
    mid : 1..n;
begin
    if low <> high then  { Stop Condition }
        begin
            mid := (low + high) div 2;
            MergeSort(low, mid, T);
            MergeSort(mid + 1, high, T);
            Merge(low, mid, high, T)
        end
end;
```

لاحظ أن الاستدعاء الأولي لهذه الإجرائية في جسم البرنامج سيكون MergeSort(1,T.mid,T). نلاحظ أيضاً أننا استخدمنا في هذه الخوارزمية الإجرائية الجزئية Merge لدمج جدولين جزئيين مرتبين في جدول مرتب وحيد. هذه الإجرائية الجزئية تحتاج أيضاً إلى توصيف.

خوارزمية الدمج (Merge)

ترويسة الإجرائية

```
procedure Merge(low, mid, high : index; var T : TableType);
```

الدخل

أدلة بداية ونهاية الجدولين الجزئيين في الجدول T والجدول T.

الخرج

الجدول T مرتب.

ملاحظة

نفترض أن الجدولين الجزئيين متجاوران، بمعنى أن دليل بداية الجدول الجزئي الثاني تساوي دليل نهاية الجدول الجزئي الأول + 1.

مخطط الخوارزمية

نستخدم (للتسهيل) جدولاً مساعداً Aux لتخزين العناصر قبل نقلها إلى الجدول T.

نستخدم أيضاً ثلاثة عدّادات:

- i يمسح الجدول الجزئي الأول (نسميه هنا T1 للتبسيط).
- j يمسح الجدول الجزئي الثاني (نسميه هنا T2 للتبسيط).
- k يمسح الجدول المساعد.

1- ما دام لم ينته أحد الجدولين على الأقل:

1-1- إذا كان T1[i] < T2[j]

ننسخ T1[i] في الخانة Aux[k]

نزيد العدّاد i واحداً، ونزيد العداد k واحداً.

1-2- إذا كان T1[i] > T2[j]

ننسخ T1[j] في الخانة Aux[k]

نزيد العدّاد j واحداً، ونزيد العداد k واحداً.

1-3- إذا كان T2[j] = T1[i]

ننسخ T1[i] في الخانة Aux[k] ، وننسخ T2[j] في الخانة Aux[k+1]

نزيد العدّاد i واحداً، والعدّاد j واحداً، والعدّاد k اثنين.

2- إذا انتهى الجدول T1 قبل الجدول T2

ننسخ ما تبقى من الجدول T2 في الجدول Aux

3- إذا انتهى الجدول T2 قبل الجدول T1

ننسخ ما تبقى من الجدول T1 في الجدول Aux

4- ننسخ الجدول Aux في الجدول T.

ترميز الخوارزمية

لسهولة مخطط الخوارزمية لا داعي لكتابة ترميز لكل مرحلة. لذلك نجد فيما يلي الترميز الكامل للخوارزمية:

```
procedure Merge(low, mid, high : index; var T :
TableType);
var
    i, j, k : index;

begin
    i := low; k := low;
```

```
    j := mid + 1;
    while (i <= mid) and (j <= high) do
        if T[i] < T[j] do
            begin
                aux[k] := T[i];
                k := k + 1; i := i + 1
            end
        else if T[i] > T[j] then
            begin
                aux[k] := T[j];
                k := k + 1; j := j + 1
            end
        else
            begin
                aux[k]   := T[i];
                aux[k+1] := T[j];
                k := k + 2;
                j := j + 1; i := i + 1;
            end;

    while j <= high do
        begin
            aux[k] := T[j];
            k := k + 1; j := j + 1
        end;
    while i <= mid do
        begin
            aux[k] := T[i];
            k := k + 1; i := i + 1
        end;
    i := low;
    while i <= high do
        begin
            T[i] := aux[i];
            i := i + 1
        end;
end;
```

1-3-2 مسألة المصفوفات الجوفاء

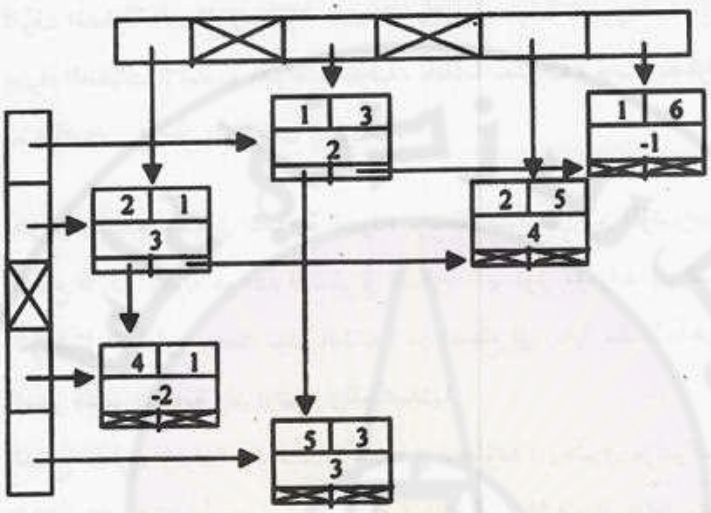
تُعرّف المصفوفة الجوفاء (Sparse Matrix) بأنها مصفوفة تحوي أصفاراً كثيرة. نريد إجراء العمليات الأساسية (قراءة مصفوفة، طباعة مصفوفة، جمع مصفوفتين، ضرب مصفوفتين، ..) على المصفوفات الجوفاء.

نفترض (للتبسيط) تمثيل المصفوفة الجوفاء بواسطة شعاعين من المؤشرات (Pointers):
الشعاع الأول، وطوله هو عدد الأسطر في المصفوفة، يحتوي مؤشرات إلى السلاسل الخطية التي تمثل الأسطر، بحيث تؤشر الخانة i من الشعاع إلى بداية سلسلة خطية من عناصر السطر i غير المعدومة وفق ترتيب أرقام أعمدتها.
الشعاع الثاني، وطوله هو عدد الأعمدة في المصفوفة، يحتوي مؤشرات إلى السلاسل الخطية التي تمثل الأعمدة، بحيث تؤشر الخانة j من الشعاع إلى بداية سلسلة خطية من عناصر العمود j غير المعدومة وفق ترتيب أرقام أسطرها.

أمّا عنصر المصفوفة، فنمثله بتسجيلة من خمسة حقول: رقم السطر، رقم العمود، قيمة العنصر، مؤشر إلى العنصر الذي يلي في نفس السطر، مؤشر إلى العنصر الذي يلي في نفس العمود.

الشكل (1-1) يوضح تمثيل المصفوفة التالية:

$$\begin{bmatrix} 0 & 0 & 2 & 0 & 0 & -1 \\ 3 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \end{bmatrix}$$



الشكل (1-1): تمثيل مصفوفة جوفاء

1- بنى المعطيات المستخدمة

```
const
    nmax  = 1000;
type
    index = 1..nmax;
    pelem = ^elem;          { مؤشر إلى عنصر من المصفوفة }
    elem  = record          { عنصر من المصفوفة }
              l  : index;   { رقم سطر العنصر }
              c  : index;   { رقم عمود العنصر }
              v  : real;    { قيمة العنصر }
              lp : pelem;   { مؤشر إلى العنصر التالي في نفس السطر }
              cp : pelem;   { مؤشر إلى العنصر التالي في نفس العمود }
          end;
```

```
matrix = record
            n  : index;   { عدد الأسطر في المصفوفة }
            m  : index;   { عدد الأعمدة في المصفوفة }
            lv : array [index] of pelem;
                 { مؤشرات إلى سلاسل عناصر الأسطر }
            cv : array [index] of pelem
                 { مؤشرات إلى سلاسل عناصر الأعمدة }
         end;
```

ملاحظات

آ. بنية المعطيات هذه ليست المثلى بالضرورة، لكنّها تسهّل بعض العمليات مثل ضرب المصفوفات. إضافةً إلى كونها قد أعطيت في نص المسألة.

ب. إذا لم تكن بنية المعطيات محدّدة في نص المسألة فيجب، إضافةً إلى شرحها بالتفصيل، تعليل اختيارها من وجهات النظر التالية: اختصار العمليات المكلفة (اختصار زمن) التي نريد إجراءها على هذه المعطيات (انظر الفصل الثاني) ؛ اختصار حجوم التخزين ؛ تسهيل البرمجة.

2- الخوارزمية

نختار الآن إحدى الخوارزميات المطلوبة، ولتكن مثلاً عملية جمع المصفوفات (ونترك بقية العمليات للقارئ كي يحلها بنفس الطريقة).

نريد إذن كتابة إجرائية تجمع مصفوفتين:

ترويسة الإجرائية

```
procedure add2matrices (m1, m2 : matrix;
                        var m : matrix);
```

الدخل

المصفوفتان m1 و m2 اللتان نريد جمعهما.

الخرج

إن كان بالإمكان إجراء عملية الجمع بين m1 و m2 نحصل على مصفوفة نتيجة الجمع: m = m1 + m2، وإلاّ على رسالة خطأ.

مخطط الخوارزمية

1- التحقق من إمكان جمع المصفوفتين

1-1- في حالة عدم إمكان الجمع

كتابة رسالة خطأ

1-2- في حالة إمكان الجمع

2- تهيئة المصفوفة الناتجة m

3- حلقة على أسطر المصفوفتين (أو الأعمدة)

من أجل كل سطر i من المصفوفتين m1 و m2

3-1- ما دامت السلسلتان المرتبطتان بالخانات m1.lv[i] و m2.lv[i] ليستا فارغتين

إضافة العناصر المشتركة (التي لها نفس رقم السطر ورقم العمود) بعد جمعهما إلى المصفوفة الجديدة m. وإضافة العناصر غير المشتركة من كلتا السلسلتين إلى m.

3-2- إذا انتهت السلسلة الأولى m1.lv[i] ولم تنته بعد السلسلة الثانية m2.lv[i]

إضافة العناصر المتبقية من السلسلة الثانية m2.lv[i] إلى المصفوفة الجديدة m.

3-3- إذا انتهت السلسلة الثانية m2.lv[i] ولم تنته بعد السلسلة الأولى m1.lv[i]

إضافة العناصر المتبقية من السلسلة الأولى m1.lv[i] إلى المصفوفة الجديدة m.

ملاحظات

1- إضافة عنصر تعني الأمور التالية: حجز حجم له، ربطه بعناصر نفس السطر، ربطه بعناصر نفس العمود.

2- لم نناقش هنا حالة كون سلسلتي السطرين فارغتين، لأننا لن نقوم بأي شيء بعد أن قمنا بعملية تهيئة المصفوفة الناتجة.

تفصيل الخوارزمية

1- التحقق من إمكان جمع المصفوفتين

```
if (m1.n <> m2.n) or (m1.m <> m2.m) then
    writeln (' the 2 matrices cannot be added:
                dimensions mismatch')
else
     sub-algorithm 2
```

2- تهيئة المصفوفة الناتجة

```
m.n := m1.n; m.m := m1.m;
for i:= 1 to m.n do
    m.lp[i] := nil;
for j:= 1 to m.m do
    m.lc[j] := nil;
```

3- حلقة على أسطر المصفوفتين

يفضل هنا، من أجل برمجة الخطوات 3-1 حتى 3-3، كتابة الإجرائية المساعدة التالية بُغية تسهيل كتابة الخوارزمية: وهي إجرائية لتهيئة عنصر وإعادة مؤشر إليه قبل إضافته إلى المصفوفة:

```
function initelem (l, p : index; w : real) : pelem;
var
    aux : pelem;
begin
    new(aux);
    aux^.i := l; aux^.j := p; aux^.v := w;
    aux^.lp := nil; aux^.cp := nil;
    initelem := aux;
    { يعيد مؤشراً إلى العنصر الذي تمت تهيئته }
end;
```

ما قبل الخطوة (3-1) نحتاج إلى تهيئة المتحولات المحلية التالية:

```
aux1i := m1.lv[i];     { مؤشر يمسح سلسلة عناصر السطر }
aux2i := m2.lv[i];
auxm := nil;
        { مؤشر سيؤشر إلى سلسلة عناصر السطر من المصفوفة الجديدة }
first := true;
```

3-1- نسخ عناصر السطر i من كلتا المصفوفتين إلى المصفوفة الناتجة، آخذين بعين الاعتبار أنه في حالة تطابق رقمي السطر والعمود بين عنصرين من المصفوفتين m1 و m2 يجب جمع قيمهما، وإجراء عملية ربط هذه العناصر المضافة في الوقت نفسه بسلاسل الأعمدة الموافقة.

```
while (aux1i <> nil) and (aux2i <> nil) do
    begin
        if (aux1i^.j < aux2i^.j) then
            auxm := initelem(aux1i^.i, aux1i^.j,
                             aux1i^.v);
```

```
            auxli := auxli^.lp;
        else
            if ((auxli^.j > aux2i^.j) then
                begin
                    auxm := initelem(aux2i^.i,
                     aux2i^.j, aux2i^.v);
                    aux2i := aux2i^.lp;
                end
            else      { نفس رقم العمود للعنصرين فنجمعهما }
                begin
                    auxm := initelem(auxli^.i,
                     auxli^.j, auxli^.v + aux2i^.v);
                    { ننقل كلا المؤشرين في هذه الحالة }
                    auxli := auxli^.lp;
                    aux2i := aux2i^.lp;
                end;
                    { الربط بأول عنصر من نفس السطر للمصفوفة الناتجة }
        if (first) then
            begin
                m.lv[i] := auxm;
                prem := auxm
            end
        else
            begin
                pre^.lp := auxm;
                pre := pre^.lp
            end;
                    { ربط العنصر بسلسلة العمود الموافقة }
        jj := auxm^.j;
        if (m.cv[jj] = nil) then
            m.cv[jj] := auxm
        else
            begin    { الذهاب إلى نهاية السلسلة }
                auxj := m.cv[jj];
                while (auxj^.cp <> nil) do
                    auxj := auxj^.cp;
                auxj^.cp := auxm;
            end
end;
```

3-2- إذا انتهت السلسلة الأولى m1.lv[i] ولم تنته بعد السلسلة الثانية m2.lv[i]

إضافة العناصر المتبقية من السلسلة الثانية m2.lv[i] إلى المصفوفة الجديدة m.

```
while aux2i <> nil do
    begin
        auxm := initelem(auxi2^.i, auxi2^.j,
                         auxi2^.v);
        if (first) then
            begin
                m.lv[i] := auxm;
                prem := auxm
            end
        else
            begin
                pre^.lp := auxm;
                pre := pre^.lp
            end;
        jj := auxm^.j;
        if (m.cv[jj] = nil) then
                m.cv[jj] := auxm
        else
            begin
                auxj := m.cv[jj];
                while (auxj^.cp <> nil) do
                    auxj := auxj^.cp;
                auxj^.cp := auxm
            end ;
        auxi := aux2i^.lp;
    end;
```

3-3- إذا انتهت السلسلة الثانية m2.lv[i] ولم تنته بعد السلسلة الأولى m1.lv[i]

إضافة العناصر المتبقية من السلسلة الأولى m1.lv[i] إلى المصفوفة الجديدة m.

```
while auxli <> nil do
    begin
        auxm := initelem(auxil^.i, auxil^.j,
```

```
                                auxil^.v);
        if (first) then
            begin
                m.lv[i] := auxm;
                prem := auxm
            end
        else
            begin
                pre^.lp := auxm;
                pre := pre^.lp
            end;
        jj := auxm^.j;
        if (m.cv[jj] = nil) then
                m.cv[jj] := auxm
        else
            begin
                auxj := m.cv[jj];
                while (auxj^.cp <> nil) do
                    auxj := auxj^.cp;
                auxj^.cp := auxm
            end ;
        auxi := auxli^.lp;
end;
```

الخطوة الأخيرة (اختيارية إذا كان المطلوب تنفيذ البرنامج على الحاسوب– عادة يستعاض عنها بالـ Listing) هي تجميع كل الرّماز المجزأ في إجرائية متكاملة: توضع فيها التحولات المحلية، التفصيل النهائي للخوارزمية، وتحويل بعض الشروح إلى الرّماز النهائي.

1-4 تمارين

1- نريد حساب وطباعة مثلث باسكال من السطر 1 حتى السطر n باستخدام سلسلة خطية *واحدة فقط* (نمثل فيها العناصر المختلفة عن الواحد فقط). بعد حساب السطر i فإن السلسلة الخطية تحوي فقط عناصر السطر i (المختلفة عن الواحد). من الطبيعي أن حساب عناصر السطر i يعتمد على عناصر السطر 1 - i وفق القاعدة المعروفة:

t[i,j] = t[i - 1,j] + t[i - 1,j - 1] حيث j هو موقع العنصر في السطر i من المثلث t.

أ. اقترح بنى المعطيات المناسبة (دعّم شرحك برسم توضيحي).

ب. اكتب إجرائية تقوم بحساب عناصر السطر i وتخزينه في السلسلة الخطية (لاحظ أن نفس السلسلة تحوي عناصر السطر السابق i-1 قبل استدعاء هذه الإجرائية).

ت. اكتب إجرائية تقوم بطباعة أول n سطراً من مثلث باسكال وفق الشكل النظامي (انظر الرسم التوضيحي الذي يحوي ستة الأسطر الأولى من المثلث).

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

2- نريد إعادة كتابة الإجرائيات الخاصة بالمصفوفات الجوفاء باعتماد بنية معطيات جديدة لتمثيل المصفوفة، فيها اختصار أكبر للحجم الذي يمكن أن تشغله المصفوفة. تعتمد هذه البنية تمثيل الأعمدة غير المعدومة في سلسلة خطية (كل عنصر فيها يحوي رقم العمود ومؤشراً إلى بداية العمود في سلسلة العناصر غير المعدومة ومؤشراً إلى العمود

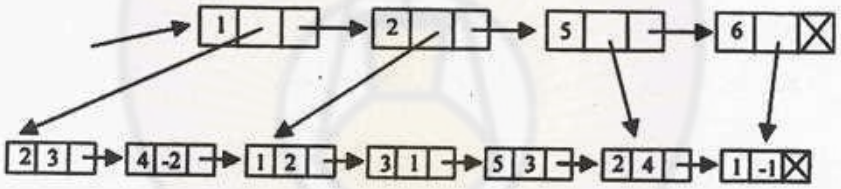
الذي يليه) وتمثيل كل عناصر المصفوفة غير المعدومة في سلسلة خطية أخرى (كل تسجيلة تحوي رقم السطر وقيمة العنصر ومؤشراً إلى العنصر التالي).

مثال

نمثل المصفوفة الجوفاء التالية:

$$\begin{bmatrix} 0 & 2 & 0 & 0 & 0 & -1 \\ 3 & 0 & 0 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \end{bmatrix}$$

بالشكل التالي:



أ. عرف بنية المعطيات Matrix لتمثيل المصفوفة وفق الشكل الجديد.

ب. اكتب الإجرائية ReadMatrix التي تقرأ مصفوفة من النمط Matrix.

ت. اكتب الإجرائية PrintMatrix التي تطبع مصفوفة من النمط Matrix وفق الشكل المعياري لطباعة المصفوفات.

ث. اكتب الإجرائية Add2Matrices لجمع مصفوفتين من النمط Matrix وتخزين المصفوفة الناتجة في المصفوفة الأولى.

ج. اكتب الإجرائية Product2Matrices لضرب مصفوفتين من النمط Matrix وتخزين المصفوفة الناتجة في المصفوفة الأولى.

ح. اكتب الإجرائية IsSymmetric لتحكم على مصفوفة من النمط Matrix: أهي متناظرة بالنسبة إلى القطر الرئيسي أم لا ؟.

الفصل الثاني

تعقيد الخوارزميات

2-1 مقدمة

ولدت دراسة تعقيد الخوارزميات مع استخدام الحواسيب ذات الأداء المتزايد باطراد لتنفيذ التعليمات الصريحة وغير الغامضة للخوارزمية التي تحل مسألة.

نستطيع من الناحية النظرية اللجوء إلى الحاسوب من أجل تنفيذ (Execution) كل مسألة محلولة بواسطة خوارزمية صحيحة. على هذا سيكون من المدهش وجود مسائل لا يمكن تنفيذها على الحاسوب (برغم إمكان حلها بواسطة خوارزميات صحيحة)، لأن تنفيذ هذه الخوارزميات قد يستغرق مئات السنين على أكثر الحواسيب سرعةً.

إذن يكتسب موضوع فعالية الخوارزميات (وحساب زمن تنفيذها) من الناحية العملية أهمية بديهية. لذلك تقوم النظريات الحالية في الحوسبة والتعقيد على طبيعة الحسابات وزمن تنفيذها من أجل توضيح كيفية اعتبار بعض المسائل أكثر صعوبة في التنفيذ من غيرها.

الغرض من هذا الفصل إعطاء فكرة عامة عن منهجية صورية، ولكن عملية لكيفية التعامل مع موضوع تعقيد الخوارزميات من أجل تمييز مفاهيم هامة مثل: الخوارزميات الفعّالة والمسائل السهلة والمسائل الصعبة.

يتطلّب تنفيذ خوارزميّة على حاسوب معيّن استخدام الموارد التي يتيحها هذا الحاسوب، ونقصد بالموارد: المعالج (حجز المعالج لمدة زمنية معيّنة) والذاكرة المركزية والمحيطيّات المتّصلة بالحاسوب. سنركّز اهتمامنا أثناء دراسة تعقيد الخوارزميّات في عاملين أساسيين:

- زمن التنفيذ: ونقصد به الزمن اللازم لتنتهي الخوارزميّة من إنجاز تعليماتها كافّة، يقاس زمن التنفيذ بحساب عدد التعليمات والزمن اللازم لتنفيذ كل تعليمة.
- حجم الذاكرة: اللازمة لتخزين البرنامج والمعطيات التي يعالجها.

إن الهدف من تحليل الخوارزميّات لا يقف عند قياس المقدارين السابقين، بل يفيد في مقارنة خوارزميّات حل مسألة معيّنة، فما نريد الوصول إليه هو حكم من الطراز:

"مهما تكن الآلة التي ستنفّذ الخوارزميّة ومهما تكن لغة البرمجة المستخدمة فإن الخوارزميّة A أفضل من الخوارزميّة B من أجل معطيات ذات حجم كبير"

أو من الطراز:

"إن الخوارزمية A هي مثلى من حيث عدد العمليات الأساسية (أو الكلفة) التي تقوم بها لحل المسألة Q"

سنبين فيما يلي ماذا نقصد بكلمة أفضل وبكلمة أمثلي.

2-2 حساب زمن تنفيذ برنامج

لتكن الخوارزمية A التي تحل المسألة P. إن كتابة برنامج تعني تمثيل A في النموذج الحسابي المعرف بواسطة لغة عالية المستوى (مثل Pascal أو C أو ADA).

يعتمد زمن تنفيذ البرنامج على حاسوب على عوامل عديدة منها:

- معطيات المسألة P الخاصة بتجربة ما.
- جودة الرماز (Code) الذي يولده المترجم من أجل بناء الملف التنفيذي.
- طبيعة وسرعة التعليمات المتوفرة في الحاسوب (في المعالج الصغري).
- جودة البرنامج الذي يكتبه المبرمج.
- *فعالية الخوارزمية A.*

من الملاحظ أن معظم هذه العوامل (عدا الأخير) خاصة: معطيات خاصة، حاسوب خاص، مقدرة المبرمج. لذلك من الطبيعي عدم اللجوء إلى مقارنة البرامج، والاهتمام بمقارنة الخوارزميات التي تحل نفس المسألة (أو اختيار الخوارزمية المثلى التي تحل مسألة ما من صف الخوارزميات التي تحل هذه المسألة). نبحث إذن عن مقياس يعكس جودة الخوارزميات بغض النظر عن تفاصيل تحويلها إلى برامج.

2-3 حساب زمن تنفيذ خوارزمية

لحساب زمن تنفيذ خوارزميّة يجري التركيز في البداية في مسألتين أساسيتين:

الأولى: تحديد بُعد (أو طول أو حجم) معطيات المسألة من أجل كتابة درجة التعقيد بدلالة هذا البعد.

أمثلة عن الخوارزمية:

- في مسائل كثيرات الحدود (Polynoms): يكون البعد هو درجة كثير الحدود أو عدد الأمثال.

- في مسائل المصفوفات (Matrices) من المرتبة (m×n): يكون البعد بدلالة أبعاد المصفوفة مثل max(m, n) أو m+n أو m.n.
- في مسائل البيانات (Graphs): يكون البعد بدلالة عدد العقد (Vertices) أو عدد الأسهم (Edges) أو مجموعها.
- في مسائل الفرز أو الترتيب (Sorting): يكون البعد بدلالة عدد العناصر التي نريد ترتيبها.
- في مسائل التحليل القواعدي (Syntax Analysis): يكون البعد بدلالة طول الكلمة.

الثانية: اختيار نوع أو عدّة أنواع من العمليات، بحيث يتناسب زمن تنفيذ الخوارزمية مع عدد هذه العمليات. يعتمد هذا الخيار على زمن تنفيذ هذه العمليات، إذ تُهمل العمليات البسيطة أمام العمليات التي هي أكثر كلفة.

أمثلة

- في خوارزميّة البحث عن عنصر ضمن سلسلة يجري التركيز في عدد عمليات المقارنة (وهي أكثر كلفة في هذه الحالة) بين العنصر الذي نبحث عنه وعناصر السلسلة.
- في خوارزمية فرز (ترتيب) العناصر في سلسلة يجري التركيز في عدد عمليات المقارنة بين العناصر وعدد عمليات التبديل بين مواقع العناصر.
- في خوارزميّة ضرب مصفوفتين يجري التركيز في عدد عمليات ضرب وجمع الأعداد.

بعد تحديد أنواع العمليات الأساسية لحساب التعقيد الزمني لخوارزمية يحسب عدد العمليات من كل نوع.

لا توجد قاعدة ثابتة أو نظام متكامل يسمح بعد العمليات، إذ يتوقف هذا الأمر على القواعد المتبعة في كتابة الخوارزمية لكننا نورد الملاحظات التالية التي تفيد في حساب التعقيد الزمني لخوارزمية:

آ- عند وجود العمليات في متتالية من التعليمات فإن عددها الكلي هو مجموع عدد العمليات في كل تعليمة. مثلا إذا كان P(X) عدد العمليات الأساسية للتعليمة X فإن:

$$P(X1 ; X2) = P(X1) + P(X2)$$

ب- عند وجود تفريع شرطي (if then else) فإنه من الصعب تحديد أي فرع سيتم تنفيذه، لكن يمكن إيجاد حد أعلى لعدد العمليات.

$$P(\text{if C then A else B}) \le P(C) + \max(P(A), P(B))$$

جـ- عند وجود حلقات فإن عدد العمليات هو:

$$\sum_{i} P(i)$$

حيث i هو المتحول الذي يتحكم بالحلقة، و P(i) عدد العمليات الأساسية المتعلقة بالمرور رقم i في الحلقة.

د- فيما يتعلق بالبرامج الجزئية (التوابع والإجرائيات) التي لا تستخدم العودية، فيكون عدد العمليات الأساسية الموافقة لاستدعاء برنامج جزئي هو عدد العمليات الأساسية الموجودة في هذا البرنامج الجزئي.

هـ- لحساب عدد العمليات اللازمة لتنفيذ برنامج جزئي عودي، يجب إيجاد طريقة لحل معادلة تراجعية. في الحقيقة يمكن التعبير عن عدد العمليات في الاستدعاء العودي

لإجرائية مع قيمة n وليكن T(n) بدلالة T(k) حيث k<n فمثلاً في خوارزميّة إيجاد العاملي لعدد صحيح موجب، إذا اخترنا عملية ضرب عددين صحيحين كعملية أساسية فإن:

$$T(0) = 0$$
$$T(n) = T(n-1) + 1 \qquad \text{من أجل } n \geq 1$$

وحل هذه العلاقة التراجعية البسيطة هو T(n) = n

2-4 التعقيد الزمني الوسطي والتعقيد في أسوأ الأحوال

من الواضح أن زمن خوارزميّة ما يتعلّق بالمعطيات التي تعالجها. وهذه المعطيات تتغيّر وفق منحيين: تغيّر في الحجم وتغيّر في المحتوى، ففي خوارزميّة البحث التسلسلي عن عنصر ضمن قائمة، يمكن أن يتغيّر كلٌ من عدد عناصر القائمة (حجم القائمة) ومحتواها.

سنرمز بـ D_n إلى لمعطيات المسألة ذات الحجم n.

و بـ $Cost_A(d)$ إلى *التعقيد الزمني* للخوارزميّة A من أجل المعطاة d.

نسمي *التعقيد الزمني في أحسن الأحوال* للخوارزميّة A المقدار:

$$Min_A(n) = Min \{ Cost_A(d) ; d \in D_n \}$$

ونسمي *التعقيد الزمني في أسوأ الأحوال* المقدار:

$$Max_A(n) = Max \{ Cost_A(d) ; d \in D_n \}$$

ونسمي *التعقيد الزمني الوسطي* المقدار:

$$Average_A(n) = \sum_{d \in D_n} P(d).Cost_A(d)$$

حيث P(d) احتمال أن تكون معطاة الخوارزميّة هي d، وعندما تكون جميع المعطيات متساوية الاحتمال تصبح العلاقة السابقة:

$$Average_A(n) = \frac{1}{|D_n|} \sum_{d \in D_n} Cost_A(d)$$

حيث $|D_n|$ عدد المعطيات ذات الحجم n .

مثال

سنحاول تحديد عدد المقارنات اللازمة للبحث التسلسلي عن عنصر ضمن قائمة تحوي n عنصراً باستخدام الخوارزمية A المشروحة في الفصل الأول.

من الواضح أن:

$Max_A(n) = n$
$Min_A(n) = 1$

لحساب $Average_A(n)$ يجب معرفة بعض الاحتمالات حول القائمة L والعنصر X:

- ليكن q احتمال أن يكون العنصر X موجوداً ضمن القائمة.
- نفترض أنّه إذا وجد X في السلسلة فإن المواضع كلها متساوية الاحتمال.

من أجل $1 \le i \le n$ ، سنرمز بـ $D_{n,i}$ إلى مجموعة كل القوائم التي طولها n والتي يظهر فيها X أول مرّة في الموقع رقم i، وسنرمز بـ $D_{n,0}$ إلى مجموعة القوائم التي لا يظهر فيها X.

بحسب الفرضيات السابقة:

$$P(D_{n,0}) = 1 - q$$

$$P(D_{n,i}) = \frac{q}{n}$$

من تحليل الخوارزميّة نستنتج أن:

$Cost(D_{n,0}) = n$
$Cost(D_{n,i}) = I$

ومن ثمّ:

$$Average_A(n) = \frac{q}{n}\sum_{i=1}^{n} i + (1-q).n = \frac{q}{n}\frac{n(n+1)}{2} + (1-q).n = \frac{q(n+1)}{2} + (1-q).n$$

فإذا كنّا نعلم سلفاً أن X موجود ضمن L فإن:

$$Average_A(n) = \frac{(n+1)}{2}$$

وإذا كان احتمال وجود X ضمن L هو $\frac{1}{4}$ فإن:

$$Average_A(n) = \frac{3(n+1)}{4}$$

2-5 مقارنة الخوارزميّات

بعد تحديد تعقيد خوارزميّة كتابع لحجم المعطيات، يمكن دراسة سرعة تزايد هذا التابع عندما يزداد حجم المعطيات. تفيد هذه الدراسة في تحديد فعّالية الخوارزميّة من أجل معالجة معطيات كبيرة الحجم، إذ يمكن في بعض الحالات أن نجد فروقاً هائلة بين خوارزميّتين من حيث التعقيد الزمني.

في معظم الحالات نكتفي بتقريب بسيط لتابع التعقيد الزمني، لمعرفة فعّالية الخوارزميّة ولمقارنة خوارزميّتين. فمثلاً عندما تكون n كبيرة يكون من غير المهم أن نعرف، أتحتاج خوارزمية معينة إلى n أم إلى n+5 عملية. ويمكن في معظم الأحوال إهمال الثوابت الضربية في تابع التعقيد. فمثلا إذا كنا نريد مقارنة الخوارزمية A التي تعقيدها الزمني $M_A(n) = n^2$ مع الخوارزمية B ذات التعقيد الزمني $M_B(n) = 4\,n$ فإن الخوارزمية B أفضل من أجل جميع قيم n تقريبا (من أجل n>4).

تؤول التقريبات السابقة إلى إيجاد ما يسمى مرتبة كبر تابع، وتجري مقارنة الخوارزميات على أساس مرتبة الكبر لتوابع التعقيد الزمني.

ولأجل إضفاء منهجية صورية على عمليات مقارنة الخوارزميات في حالة حجوم معطيات كبيرة، لا بد لنا هنا من التذكير ببعض التعاريف الرياضية الأساسية.

تعريف 1

نقول أن التابع: $f : N \to R$ يسعى نحو a (تكتب $f \to a$) عندما تسعى n إلى اللانهاية (تكتب $n \to \infty$) إذا حقق:

$$\forall \varepsilon, \exists n_0 : n \geq n_0 \Rightarrow |f(n) - a| \leq \varepsilon$$

وبنفس الأسلوب نقول إن $f \to \infty$ عندما $n \to \infty$ إذا حقق:

$$\forall K, \exists n_0 : n \geq n_0 \Rightarrow |f(n)| \geq K$$

إن معرفة سعي التابع f إلى نهاية منتهية أو غير منتهية عندما $n \to \infty$ ليست في ذاتها معلومةً دقيقةً بدرجة كافية. لذلك يجب الذهاب أبعد من ذلك من أجل مقارنة التابع f بتوابع ذات سلوك معروف بجوار اللانهاية (مثل: $\log_2 n$ أو n أو n^2 أو e^n).

تعريف 2

نقول عن التابع f إنه مهمل تقاربياً (Asymptotic Neglected) أمام التابع g ونرمز إلى ذلك بـ $f = o(g)$ (أو $f(n) = og(n)$) إذا كان:

$$f / g \to 0 \text{ when } n \to \infty$$

تعريف 3

نقول عن التابع f إنه مكافئ تقاربياً (Asymptotic Equivalent) للتابع g ونرمز إلى ذلك بـ $f \approx g$ (أو $f(n) \approx g(n)$) إذا كان:

$$f / g \to 1 \text{ when } n \to \infty$$

تعريف 4

نقول عن التابع f إنه مُسيطر عليه تقاربياً (Asymptotic Dominated) من قبل التابع g ونرمز إلى ذلك بـ $f = O(g)$ (أو $f(n) = Og(n)$) إذا كان:

$$\exists \text{ constant } c > 0 : |f(n)| \le cg(n) \text{ when } n \to \infty$$

تعريف 5

نقول عن التابع f إنه من نفس مرتبة التابع g ونرمز إلى ذلك بـ $f = \theta(g)$ (أو $f(n) = \theta g(n)$) إذا كان:

$$f = O(g) \text{ and } g = O(f)$$

أمثلة

$f(n) = n^2 + n = O(n^2)$;
$f(n) = 5n^3 = O(n^3)$;
$p(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0, a_m \neq 0 \Rightarrow$
$p(n) = O(n^m)$; $p(n) = o(n^{m+1})$; $p(n) = \theta(n^m)$

ملاحظات

- قد تكون إشارات المساواة في التعاريف السابقة مربكة، لذلك منعاً للالتباس تجدر الإشارة إلى أن الرموز $o(g)$ و $O(g)$ و $\theta(g)$ تعني عملياً:

$o(g) = \{ f : f \text{ asymptotic neglected to } g \}$
$O(g) = \{ f : f \text{ asymptotic dominated by } g \}$
$\theta(g) = \{ f : f = O(g) \text{ and } g = O(f) \}$

لذلك المساواة $f = o(g)$ تعني عملياً $f \in o(g)$

- من الواضح أن: $f = o(g) \Rightarrow f = O(g)$ وأن $f \approx g$ إذا وفقط إذا $f = g + o(g)$
- $f = \theta(g)$ إذا وفقط إذا وجد ثابتان موجبان c_1 و c_2 بحيث عندما $n \to \infty$ يكون $c_1 g(n) \le f(n) \le c_2 g(n)$

خواص

- الانعكاسية والتعدي

$g = O(g)$
$O(O(g)) = O(g)$
$o(o(g)) = o(g)$

- الضرب بثابت

$c.O(g) = O(g)$
$c.o(g) = o(g)$

- جمع وضرب التوابع

$O(g1) + O(g2) = O(Max\ \{g1, g2\})$
$o(g1) + o(g2) = o(Max\ \{g1, g2\})$
$O(g1) . O(g2) = O(g1.g2)$
$o(g1). o(g2) = o(g1.g2)$

- الرفع إلى قوة واللوغاريتم

إذا كان $c>0$ ثابت فإن:

$|O(g)|^c = O(g^c)$ and $|o(g)|^c = o(g^c)$

إذا كان $g(n) \to \infty$ و $|f(n)| \geq c>0$ عندما $n \to \infty$ فإن:

$f = O(g) \Rightarrow \log_2 |f| = o(\log_2 (g))$

ونترك للقارئ برهان الاحتواء التالي:

$0 < \varepsilon < 1 < c$
$O(1) \subseteq O(\log n) \subseteq O(n^\varepsilon) \subseteq O(n) \subseteq O(n . \log n) \subseteq O(n^c) \subseteq O(c^n) \subseteq O(n!)$

نستطيع الآن بعد هذه التذكرة ببعض المفاهيم الرياضية وضع الأساس العملي لحساب تعقيد الخوارزميات:

لتكن A و B خوارزميتين تحلان نفس المسألة ولنفترض أننا استطعنا إيجاد تابعين f و g بحيث:

$Cost_A(n) = O(f(n))$; $Cost_B(n) = O(g(n))$

عندئذ نستطيع مقارنة الخوارزميتين A و B بمقارنة التابعين f و g.

هذه الطريقة تبسط تحليل تعقيد الخوارزميات وذلك بإهمال العوامل الثابتة التي يعود معظمها إلى خواص نموذج الحساب.

أمثلة

المثال الأول

ليكن لدينا الجدول A[1..n] الذي يحوي n عنصرا، ونريد حساب تعقيد خوارزمية الفرز بالفقاعات (Bubble Sort):

الإجرائية

```
procedure BubbleSort (n: integer; A: array[1..n] of
integer);
var
    i, j : integer;
begin
    for i:=n-1 downto 1 do
        for j:=1 to i do
            if A[j+1] < A[j] do
                begin
                    {swap A[j] and A[j+1]}
                    t:=A[j];
                    A[j]:=A[j+1];
                    A[j+1]:=t;
                end
end;
```

الحل

بعد المسألة هو n. ويحتوي البرنامج على حلقتين (واحدة داخل الأخرى)، هناك n-1 مرورا في الحلقة الخارجية. في المرور n-i نجري i عملية مقارنة وi عملية تبديل مواقع ولذا:

$$Cost(n) = \sum_{i=1}^{n} 2i = n(n-1) = \theta(n^2)$$

طريقة أخرى

في المرور الأول نجري n-1 عملية مقارنة و n-1 عملية تبديل مواقع في أسوأ الأحوال. في المرور الثاني نكرر نفس العملية لكن على جدول A[1..n-1] ومن ثم نستنتج العلاقة التراجعية:

```
T1 = 0;
T(n) = T(n-1) + 2(n-1);
```

هذا يؤدي إلى:

$$Cost(n) = \sum_{i=1}^{n} 2(n-i) = \sum_{j=1}^{n-1} 2j = n(n-1) = \theta(n^2)$$

المثال الثاني

نريد حساب تعقيد خوارزمية إعادة تمثيل عدد A من الأساس D إلى الأساس العشري حيث $D \le 10$:

$$A = a_n D^n + a_{n-1} D^{n-1} + \ldots + a_1 D + a_0$$

الإجرائية (الطريقة المباشرة)

```
function Direct (n: integer; A: array[1..n] of
integer): integer;
var
    i, j, C, P : integer;
begin
    C:=A[0];
    for i:=1 to n do
        if A[i] <> 0 then
            begin
                P:=A[i];
                for j:=1 to i do
                    P:=P*D;
                C:=C+P
            end;
    Direct:=C;
end;
```

الحل

بعد المسألة هو n درجة كثير الحدود. العمليات الأساسية هي عمليتي الضرب. في الحلقة الداخلية هناك i عملية ضرب وعملية جمع واحدة. وبالتالي عدد عمليات الجمع sum وعدد عمليات الضرب prod يحسب كالأتي:

$$sum = \sum_{1}^{n} 1 = n$$

$$prod = \sum_{i=1}^{n} i = \frac{n(n-1)}{2} = \theta(n^2)$$

الطريقة الثانية (طريقة هورنر Horner)

نكتب A كما يلي:

$$A = ((...((a_n D + a_{n-1}) \times D + a_{n-2}) \times D + ...) \times D + a_1) \times D + a_0$$

للحصول على C يكفي الحساب في الأساس العشري للمتتالية $(C_i)_{1 \le i \le n}$ حيث:

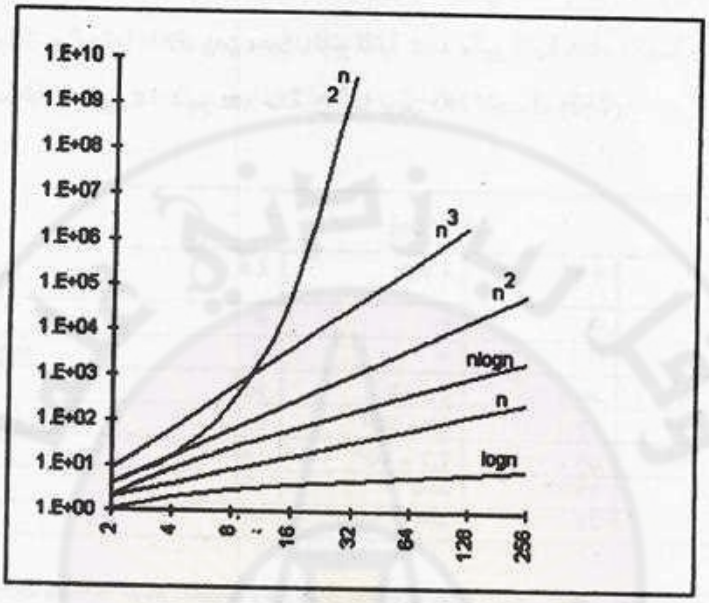
$C_0 = a_n$;

$C_i = C_{i-1} \times D + a_{n-i}$

وباعتبار أن: $C_n = C$ فإننا نجري n عملية ضرب فقط، ويكون التعقيد من رتبة $\theta(n)$ (لاحظ التحسين الذي أجريناه مقارنة مع الطريقة المباشرة).

يبين الشكل (2-1) سرعة تزايد بعض التوابع ذات الأداء "المعروف" بجوار اللانهاية، وهي أكثر استعمالا في تقريب توابع التعقيد الزمني للخوارزميات.

كما يبين الجدول (2-1) تقديرا لزمن تنفيذ خوارزميات يأخذ تابع التعقيد الزمني المرتبط بها أحد الأشكال السابقة، وذلك من أجل قيم مختلفة لـ n حيث افترضنا أن الحاسوب المستخدم يقوم بمليون عملية في الثانية (استعملنا الرمز ∞ عندما تتجاوز القيمة المحسوبة 10^{100}).



الشكل (2–1): سرعة تزايد بعض التوابع
المستخدمة في تقريب التعقيد الزمني للخوارزميات

حجم المعطيات						
1,000,000	100,000	10,000	1,000	100	10	التعقيد
10^{-6} s	10-6 s	10-6 s	10^{-6} s	10^{-6} s	10^{-6} s	1
19.93 ns	16.61 ns	13.29 ns	9.97 ns	6.64 ns	3.32 ns	$\log_2 n$
1000 ms	100 ms	10 ms	1 ms	0.1 ms	0.01 ms	n
19931.57 ms	1660.96 ms	132.88 ms	9.97 ms	0.66 ms	0.03 ms	$n \log n$
11.57d	2.78 h	100.00 s	1.00 s	0.01 s	0.0001 ms	n^2
31710 y	31.71 y	11.57 d	1000.00 s	1.00 s	0.001 ms	n^3
∞	∞	∞	$1. \times 10^{295}$ y	4.02×10^{16} y	1.02 s	2^n

الجدول (2–1): زمن تنفيذ خوارزمية بدلالة التعقيد وحجم المعطيات

من جهة أخرى يمكن دراسة تأثير تعقيد الخوارزميات على الحجم الأعظمي للمعطيات التي يمكن معالجتها خلال زمن معين ،نانو ثانية ns، مكرو ثانية ms، ثانية s، دقيقة mn، ساعة h، يوم d، شهر m، سنة y) كما يبين ذلك الجدول (2-2).

التعقيد	زمن التنفيذ			
	1 s	1 mn	1 h	1 y
1	∞	∞	∞	∞
$log_2 n$	∞	∞	∞	∞
n	10^6	6×10^7	36×10^8	86×10^9
$n \log n$	63×10^3	28×10^5	13×10^7	97×10^8
n^2	10^3	7.7×10^3	60×10^3	26×10^4
n^3	100	360	1500	4400
2^n	19	25	31	36

جدول (2-2): حجم المعطيات التي يمكن معالجتها بدلالة التعقيد الزمني وزمن التنفيذ.

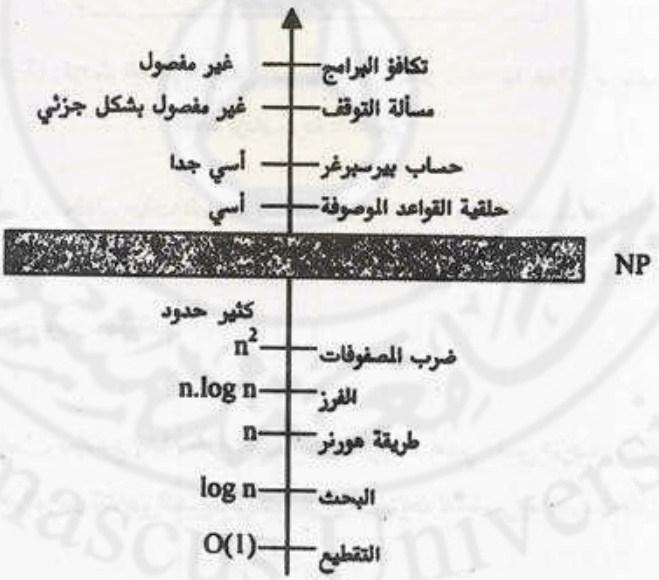
من دراسة الجدولين السابقين يتبين أن بعض الخوارزميات لا تصلح، ويجب عدم توظيفها لحل المسائل باستعمال الحاسوب. يمكن أن نقول إن الخوارزميات الجيدة لحل مسألة معينة هي التي يكون زمن تنفيذها:

- ثابتا مهما كان حجم المعطيات: مثل خوارزميات البحث في جداول التقطيع (Hashing Tables).
- لوغارتميا: كخوارزميات البحث الثنائي.
- خطيا: كخوارزمية البحث المتسلسلي التي عرضناها سابقا.
- من مرتبة $n.\log_2(n)$: كخوارزميات الفرز الجيدة.

يمكن القول بوجه عام، إن الخوارزميات التي يكون زمن تنفيذها من مرتبة n^k (والتي تسمى عادة خوارزميات كثيرات الحدود) حيث: $K>0$ ليست فعالة إلا إذا كانت $K<2$، وعندما تكون $2<K<3$ فإننا نستطيع معالجة معطيات متوسطة الحجم. وعندما يكون $K>3$ فإننا لا نستطيع أن نعالج إلا مسائل ذات حجم معطيات صغير.

أما الخوارزميات ذات التعقيد الأسي (2^n مثلا) فلا تستخدم إلا في حالة معطيات ذات حجم محدود ولهذا وصفناها بأنها غير فعالة.

ولعل الشكل (2-2) يوضح تصنيف المسائل حسب زمن التنفيذ. حيث NP هي مجموعة المسائل التي يمكن إيجاد حلول مساعدة (Heuristic) خاصة لها بواسطة خوارزميات كثيرات الحدود.



الشكل (2-2): تصنيف المسائل حسب زمن التنفيذ

قد يخطر بالبال أن زيادة سرعة معالجة الآلة قد يسمح بتطبيق هذه الخوارزميات. لتوضيح أثر سرعة الآلة (انظر الجدول (3-2)) سنفترض أننا جعلنا سرعة الآلة تزداد بمقدار عشر مرات، ولنحسب ازدياد حجم المعطيات التي يمكن معالجتها خلال زمن معين.

التعقيد	ازدياد حجم المعطيات
1	∞
$\log_2 n$	n^{10}
n	10 n
$n.\log_2 n$	(10-e) n
n^2	3.16 n
n^3	2.15 n
2^n	n + 3.32

الجدول (3-2): ازدياد الحجم الأعظمي للمعطيات التي يمكن معالجتها خلال زمن معين عندما تزداد سرعة المعالجة.

نلاحظ مما سبق أن الخوارزميات الجيدة تستفيد بقدر ممتاز من زيادة سرعة المعالجة، على حين تتحسن الخوارزميات السيئة (ذات التعقيد الأسي مثلا) قليلا جدا عندما تزداد سرعة الحاسوب المستخدم.

نخلص إلى القول إن موضوع إنشاء خوارزميات جيدة وفعالة هي على درجة بالغة من الأهمية، على الرغم من التطور السريع في فعالية التجهيزات الحاسوبية المستخدمة.

2-6 تمارين

1- ضرب كثيري حدود

ليكن كثيرا الحدود A(X) وB(X):

$$A(X) = a_0 + a_1X + \dots + a_{n-1}X_{n-1}$$
$$B(X) = b_0 + b_1X + \dots + b_{n-1}X_{n-1}$$

آ- احسب التعقيد الزمني لحساب: $P(X) = A(X) \times B(X)$ بالطريقة المباشرة:

```
procedure DirectMult(n: integer; A,B: array[0..n-1] of
real);
var
    i, j : integer;
begin
    for i:=0 to 2*n-2 do
        P[i]:=0;
    for i:=1 to n-1 do
        for j:=1 to n-1 do
            P[i+j] := P[i+j]+A[i]*B[j]
end;
```

ب- هل تستطيع إيجاد طريقة ثانية لحساب هذا الجداء بحيث تخفض من قيمة التعقيد الزمني.

2- بفرض x عددا كسريا و n عددا صحيحا موجبا. يمكن حساب x^n باستعمال أحد التابعين power1 و power2 أوجد التعقيد الزمني لهاتين الخوارزميتين من حيث عدد عمليات الضرب.

```
function power1 (x : real;  n : integer) : real;
var
    p : real;
begin
```

```
    p := 1;
    for  i := 1 to  n  do
        p := p * x;
    power1 := p
end;

function power2 (x : real;  n : integer) : real;
var
    factor, res : real;
begin
    res :=1;
    factor := x;
    while n > 0 do
        begin
            if (n mod 2) = 1 then
                res := res * factor ;
            factor:= factor * factor;
            n := n div 2
        end;
    power2 := res
end;
```

3- لحساب جداء مصفوفتين C(m,q) = A(m,n) . B(n,q) يمكن تطبيق الطريقة المعروفة في حساب الحد العام لصفوفة C وذلك في حالة $1 \le i \le m$ و $1 \le j \le q$:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

```
procedure MatMult (A, B: matrix; m, n, q: integer; var
C: matrix);
var
    aux : real;
    i, j, k : integer;
begin
    for i := 1 to m do
        for j := 1 to q do
            begin
                aux := 0;
                for k := 1 to n do
                    aux := aux + A[i,k] * B[k,j];
                    C[i,j] := aux
                end
end;
```

أ. احسب عدد عمليات الضرب وعدد عمليات الجمع اللازمة لحساب جداء مصفوفتين بهذه الطريقة.

ب. بفرض m=n=q وأن عملية الضرب تستغرق 10^{-6} ثانية وأن عملية الجمع تستغرق 10^{-7} ثانية، أوجد الحجم الأعظمي للمصفوفات A و B التي يمكن حساب جداتها خلال دقيقة، ساعة، يوم.

4- سندرس في هذا التمرين طريقة ثانية لحساب جداء مصفوفتين. ستقتصر المناقشة على حالة مصفوفتين مربعتين A(n,n) و B(n,n).

لنحاول أولا إيجاد الجداء في حالة $n = 2$:

- نحسب المعاملات السبعة التالية :

$$
\begin{aligned}
x1 &= (a11 + a22) \cdot (b11 + b22)\\
x2 &= (a21 + a22) \cdot b11\\
x3 &= a11 \cdot (b12 - b22)\\
x4 &= a22 \cdot (b21 - b11)\\
x5 &= (a11 + a12) \cdot b22\\
x6 &= (a21 - a11) \cdot (b11 + b12)\\
x7 &= (a12 - a22) \cdot (b21 + b22)
\end{aligned}
\qquad (I)
$$

نستعمل هذه المعاملات لحساب حدود المصفوفة الناتجة :

$$
\begin{aligned}
c11 &= a11 \cdot b11 + a12 \cdot b21 = x1 + x4 - x5 + x7\\
c12 &= a11 \cdot b12 + a12 \cdot b22 = x3 + x5\\
c21 &= a21 \cdot b11 + a22 \cdot b21 = x2 + x4\\
c22 &= a21 \cdot b12 + a22 \cdot b22 = x1 + x3 - x2 + x6
\end{aligned}
\qquad (II)
$$

نلاحظ أن حساب الجداء يحتاج لسبع عمليات ضرب و 18 عملية جمع (أو طرح).

أ. بفرض $n = 2^k$ كيف يمكن تعميم هذه الطريقة في حال $k > 2$ ؟

ب. بفرض M(k) عدد عمليات الضرب اللازمة في حال $n = 2^k$.

ت. أوجد العلاقة بين M(k) و M(k-1) ثم احسب عدد عمليات الضرب بدلالة n.

ث. بفرض P(k) عدد عمليات الجمع و الطرح اللازمة في حال n = 2. أوجد العلاقة بين P(k) و P(k-1) ثم احسب عدد عمليات الجمع و الطرح بدلالة n .

ج. قارن التعقيد الزمني لهذه الخوارزمية بخوارزمية ضرب المصفوفات المبينة في السؤال السابق.

5- قدر زمن تنفيذ الإجرائيات التالية:

```
procedure Try1;
var
    I, J, K, N, M, P, L : integer;
begin
    readln(N); M:=0; P:=0; L:=0;
    for I:=1 to N-1 do
        for J:=I+1 to N do
            for K:=1 to J do     ……
                begin
                    M:=M+1;
                    P:=P*2;
                    L:=L+3;
                end; {For}
    writeln(M, P, L)
end; {Try1}

procedure Try2;
var
    X, Y, I, J, N : integer;
begin
    readln(N);
    X:=0; Y:=0;
    for I:= 1 to Y do
        if odd(I) then
            begin
                for J:=1 to N do
                    X:= X+1;
                for J:=1 to I do
                    Y:= Y*2;
            end; {if}
    writeln(X, Y)
end; {Try2}
```

```
procedure Try3;
var
    X, Y, I, J, N : integer;
begin
    readln(N);
    X:=0; Y:=0;
    for I:= 1 to N do
        if (I mod 3 = 0) then
            for J:=I to N do
             X:= X+3
        else
            for J:=1 to I do
                Y:= Y*3;
    writeln(X, Y)
end; {Try3}
```

الفصل الثالث

الخوارزميات العوديّة

3-1 مقدمة

نقول عن شيء إنه ذو بنية عوديّة (Recursive) إذا كان مؤلفاً من مَجموعة من المكونات بعضها مُعرّف تعريفَ الشيء الأصليّ. نقول مثلاً إنّ الشجرة هي بنية عوديّة لأنها مؤلفة من مجموعة من الفروع، كلٌ منها يحوي فروعاً جديدة، ومن ثم يكون له بنية الشجرة نفسها أو يحوي مجموعة من الأوراق.

إنّ أكثر من اهتمّ بموضوع العوديّة هم الرياضيون الذين استخدموها أداةً فعّالة في التعاريف الرياضية وخاصةً للمجموعات غير المنتهية. ومن الأمثلة الشهيرة لاستخدام العوديّة في التعاريف نورد ما يلي:

1- تعريف الأعداد الطبيعيّة: تُعرّف الأعداد الطبيعية كما يلي:

- العدد صفر هو عدد طبيعي
- إذا كان n عدداً طبيعياً فإن n+1 عدد طبيعي أيضاً.

2- تعريف بنية الشجرة الثنائية: تُعرّف الشجرة الثنائية كما يلي:

- الشجرة O لا تحوي أي عقدة، نسميها الشجرة الفارغة.

- إذا كانت A و B شجرتين ثنائيتين فإن <X, A, B> هي شجرة ثنائية جذرها X وشجرتها الجزئية اليسارية A وشجرتها الجزئية اليمينية B.

3- تعريف العاملي لعدد صحيح غير سالب: يُعرّف العاملي كما يلي:

- $0! = 1$
- إذا كان $n>0$ فإنّ $n! = n.(n-1)!$

4- تعريف متتالية أعداد فيبوناتشي (Fibonacci): تُعرّف هذه المتتالية بالشكل التالي:

- $Fib_0 = 0$, $Fib_1 = 1$
- إذا كان $n>1$ فإنّ $Fib_n = Fib_{n-1} + Fib_{n-2}$

تكمن أهمية العوديّة في إمكان تعريف مجموعة غير منتهية من الأشياء، بواسطة مجموعة منتهية من التعليمات. بالطريقة نفسها يمكن تعريف عمليات حسابية غير منتهية بواسطة خوارزميّة عوديّة، تكون الخوارزميّات العوديّة أكثر ملاءمة عندما تكون المسألة المطلوب حلها أو بنية المعطيات الواجب معالجتها معرّفة تعريفاً عودياً.

في التطبيقات العملية يجب أن يكون عمق العوديّة منتهياً وصغيراً أيضاً، لأن كل استدعاء عودي يتطلب تخزين وسائط الإجرائية ومتحولاتها الموضعية وحالة الإجرائية، وذلك من أجل استعادة هذه المعطيات عند انتهاء الاستدعاء. على هذا، كلما كان عمق العوديّة صغيراً كان حجم التخزين صغيراً. وتجدر الإشارة إلى أنه في حالات عدم الانتباه لهذا العمق، يؤدي تنفيذ البرنامج العوديّ إلى طفح في مكدس التنفيذ (Stack Overflow) ومن ثم إلى توقف البرنامج فوراً، قبل الانتهاء من التنفيذ.

من المفيد أن نلاحظ أيضا أنه حتى في حالة التعريف العودي للمسألة أو بنية المعطيات، قد لا تكون العودية هي الحل الأفضل.

مثال: أعداد فيبوناتشي

التعريف عودي والحل بواسطة إجرائية عودية مباشر:

```
function fib (n : integer) : integer;
begin
    if n = 0 then
        fib = 0
    else
        if n = 1 then
            fib := 1
        else
            fib := fib(n-1) + fib(n-2)
end;
```

لاحظ أنه باستخدام هذه الإجرائية العودية لحساب fib_n نحتاج إلى 2^{n-1}-1 استدعاء عودي، ولذا ينمو عدد الاستدعاءات أسيا ويعتبر هذا غير فعال من الناحية العملية، على حين نستطيع إيجاد إجرائية غير عودية (تكرارية) خطية للحل:

```
function fib (n : integer) : integer;
var
    i, m : integer;
begin
    i := 1; m := 0;
    while i < n do
        begin
            i := i + 1;
            n := n + m;
            m := n - m;
        end
end;
```

إذن يجب تجنب استعمال العودية عندما يمكن استعمال خوارزمية تكرارية (Iterative Algorithm) مكافئة. ولكن ليس في كل الحالات يمكن تحويل العودية إلى تكرار بسيط

بسهولة (انظر الأمثلة في الفقرة 3-4)، ثم إن عملية التحويل العامة من عوديّة إلى تكرارية قد تكون معقدة (انظر الفقرة 3-5).

3-2 المخطط العام لخوارزميّة عوديّة

يمكن بوجه عام التعبير عن خوارزميّة عوديّة P بواسطة تركيب C لمجموعة عمليات أساسية Si (لا تحوي P) مع P نفسها:

$$P \equiv C[Si, P]$$

إنّ الأداة اللازمة والكافية للتعبير عن برنامج معيّن تعبيراً عودياً هي الإجرائية (Procedure) أو التابع (Function) لأنها تسمح بإعطاء اسم معين لمجموعة تعليمات، وهذا ما يسمح باستدعاء هذه التعليمات استدعاءً عودياً.

يمكن التمييز بين نوعين من الإجرائيات العوديّة:

- *الإجرائيات ذات العوديّة المباشرة*: نقول عن إجرائية P إنها عوديّة مباشرة إذا كانت تحوي استدعاءً صريحاً لنفسها.
- *الإجرائيات ذات العوديّة غير المباشرة*: نقول عن إجرائية P إنها عوديّة غير مباشرة إذا كانت تستدعي إجرائية أخرى Q تستدعي P (بطريق مباشر أو غير مباشر).

3-3 انتهاء تنفيذ إجرائية عوديّة

في كل إجرائية تحوي تكراراً لتعليمات معينة، يجب الانتباه إلى أنّ هذا التكرار سينتهي بعد حد معين. إنّ أولى الاحتياطات الواجب اتخاذها هي جعل الاستدعاء العوديّ

للإجرائية مشروطاً بتحقق قضية معينة. بهذا يمكن أن نعتبر أن المخطط العام لإجرائية عوديّة يأخذ أحد الشكلين:

$$P \equiv \text{if } B \text{ then } C[Si, P]$$

أو

$$P \equiv C[Si, \text{if } B \text{ then } P]$$

للبرهان على انتهاء إجرائية عوديّة يُعرَّف تابع P(X) حيث X مجموعة معاملات الإجرائية. نعرف التابع P(X) بحيث تكون القضية B مكافئة لـ $P(X) > 0$ وفي المرحلة التالية نبرهن أنّ P(X) يتناقص عند كل استدعاء عوديّ.

مثال

لنأخذ إجرائية حساب العاملي:

```
Function fact (n : integer):integer;
begin
     if  n = 0 then
           fact :- 1
     else
           fact := n*fact(n-1)
end;
```

في هذه الحالة مجموعة المعاملات هي $X = \{n\}$، إذا عرفنا التابع $P(n) = n$ فإن شرط الاستدعاء العودي هو $P(n) > 0$.

نلاحظ من نص الإجرائية أن التابع يتناقص بمقدار واحد عند كل استدعاء عودي.

3-4 أمثلة لبرامج عودية

نعرض فيما يلي أمثلة على استخدام البرامج العودية في مسائل قد يكون من المعقد حلها ببرامج تكرارية.

3-4-1 تحويل التعابير الرياضية من مصدَّرة إلى ملحقة

نستطيع كتابة التعابير الرياضية بلا أقواس باستخدام أسلوبين مختلفتين: الكتابة المصدَّرة (Prefixe) والكتابة الملحقة (Postfixe). في الأسلوب المصدَّر تسبق كل عملية (Operator) مباشرة عواملها (Operands) أو متحولاتها مباشرة. على حين في الأسلوب الملحق تتبع كل عملية عواملها مباشرة.

مثال

نظامي (Infix)	مصدّر (Prefix)	ملحق (Postfix)
A+B	+AB	AB+
A+B*C	+A*BC	ABC*+
A*(B+C)	*A+BC	ABC+*
A*B+C	+*ABC	AB*C+
A+B*C+D-E*F	-++A*BCD*EF	ABC*+D+EF*-
(A+B)*(C+D-E)*F	**+AB-+CDEF	AB+CD+E-*F*

إن أفضل طريقة لتعريف هذين الأسلوبين هي باستعمال العوديّة. لنفترض للتسهيل عدم وجود ثوابت في التعابير الرياضية، كما نفترض أيضاً أن جميع العمليات ثنائية.

المخطط الأولي للخوارزمية العوديّة التي تقوم بعملية التحويل من تعبير مصدَّر إلى تعبير ملحق هو كالتالي:

- إذا كان التعبير متحولاً وحيداً، فلا نقوم بأي شيء (تعبيران متكافئان في هذه الحالة).
- لتكن op أول عملية في التعبير المصدّر.
- إيجاد المعامل الأول opnd1 للتعبير المصدَّر وتحويله إلى ملحق وتسميته post1.
- إيجاد المعامل الثاني opnd2 للتعبير المصدَّر وتحويله إلى ملحق وتسميته post2.

دمج التعابير الجزئية opnd1, opnd2, op بالترتيب ووضع الناتج في التعبير الملحق.

لنحدّد أولاً بنى المعطيات المستخدمة:

```
const
    strsize = 80;
type
    string = record
                ch : array[1..strsize] of char;
                length : 0..strsize
             end;
    postype = 1..strsize;
    lentype = 0..strsize;
```

حيث نلاحظ في تعريف سلسلة المحارف string أن ch هي سلسلة المحارف نفسها وlength تمثل الطول الجاري للسلسلة.

نحتاج أيضاً إلى تعريف الإجرائيتين المساعدتين التاليتين:

- الإجرائية concat(a, b, c) التي تقوم بدمج السلسلتين a وb في السلسلة c. فمثلاً الاستدعاء concat("abc", "xy", c) سيضع في c السلسلة "abcxy".
- الإجرائية substr(S1, i, j, S2) التي تضع في السلسلة الجزئية S2 جميع الأحرف التي عددها j من S1 اعتباراً من الموقع i في S1.

ونترك للقارئ مهمة كتابة هاتين الإجرائيتين البسيطتين.

نستطيع الآن كتابة الخوارزمية العودية convert لإجراء عملية التحويل:

```
procedure convert (prefix : string; var postfix :
string);
var
    post1 ,post2, opstr : string;
    opnd1, opnd2, temp : string;
    op : char;
    m, n : lentype;
```

```
begin
    if prefix.length = 1 then
        if letter(prefix.ch[1]) then
            postfix := prefix
        else
            writeln('Illegal prefix string')
    else
        begin
            op := prefix.ch[1];
            m  := find(prefix, 2);
            n  := find(prefix, m+2);
            if not(op in ['+','-','*','/'] or
               (m=0) or (n=0) or
               (m+n+1 <> prefix.length then
                    writeln('Illegal prefix string')
            else
                begin
                    substr(prefix, 2, m, opnd1);
                    substr(prefix, m+2, n, opnd2);
                    convert(opnd1, post1);
                    convert(opnd2, post2);
                    concat(post1, post2, temp);
                    opstr.ch[1] := op;
                    opstr.length := 1;
                    concat(temp, opstr, postfix)
                end
        end
end;
```

نلاحظ في هذه الإجرائية وجود استدعاء للتابع letter(chr) الذي يحدد: أيكون المحرف chr حرفا أم لا. كما نلاحظ استدعاء للتابع find(str, position) الذي يقبل سلسلة محارف str وموقعا position ويعيد طول أطول تعبير مصدر موجود في سلسلة محارف الدخل str التي تبدأ عند الموقع position. مثلا الاستدعاء find('a+cd', 1) يعيد الطول 1، بينما الاستدعاء find('+*abcd+gh', 1) يعيد الطول 5.

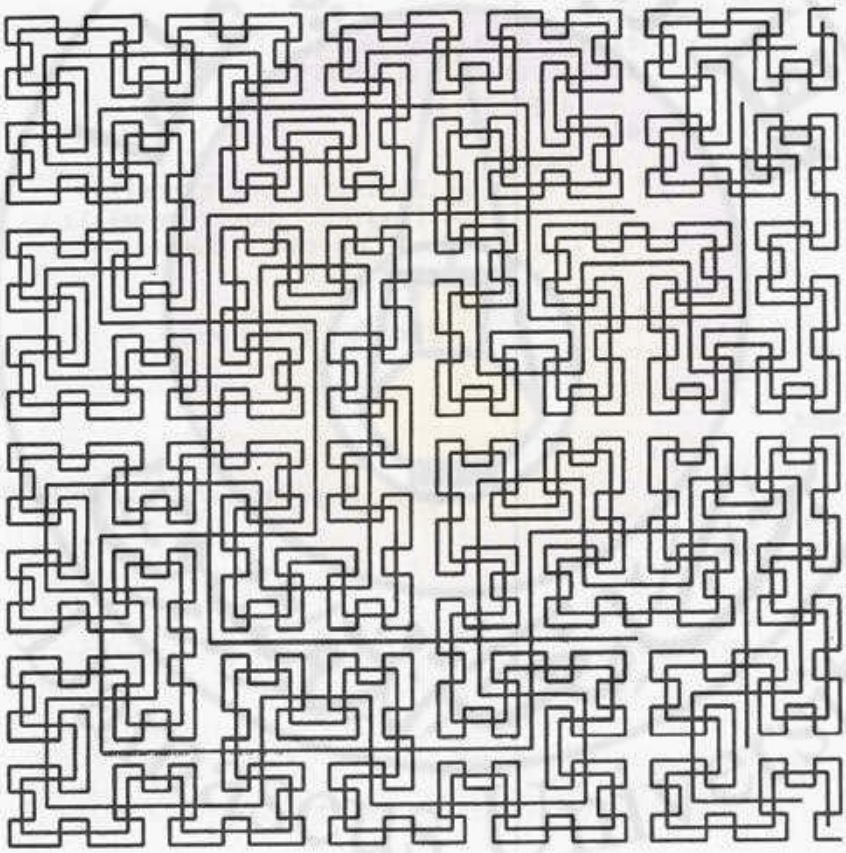
يستعمل هذا التابع لتحديد المعامل الأول والثاني لعملية مصدرة. ولاستكمال العمل لا بد من كتابة هذا التابع:

```
function find(str : string; position : postype) :
lentype;
var
    m, n : lentype;
    first : char;
begin
    if position > str.length then
        find := 0
    else
        begin
            first := str.ch[position];
            if letter(first) then
                find := 1
            else
                begin
                    m := find(str, position+1);
                    n := find(str, position+m+1);
                    if (m=0) or (n=0) then
                        find := 0
                    else
                        find := m + n + 1
                end
        end
end;
```

نلاحظ هنا أيضاً أن التابع find هو تابع عوديّ.

3-4-2 منحنيات هلبرت

إذا نظرنا بإمعان إلى الأشكال الزخرفية المبينة في الشكل (3-1) نرى أنها تتألف من خمسة خطوط منكسرة تتبع منهجاً في الرسم نستطيع برمجته بعد تحديده بدقة.



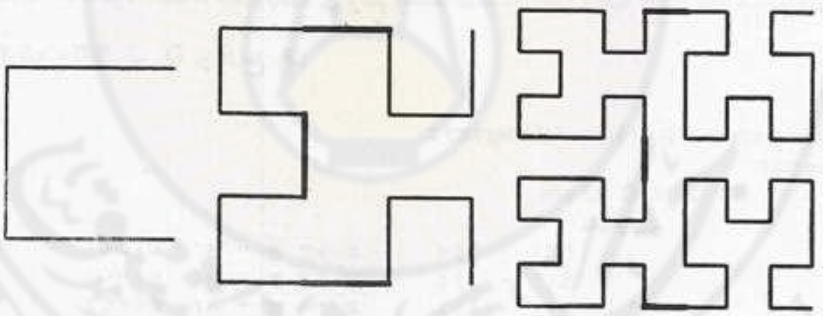
الشكل (3-1): منحنيات هلبرت من H1 إلى H5

لو نظرنا مرة أخرى إلى هذه الخطوط، وحاولنا عزل الثلاثة الأولى منها، نجد أن لها شكلاً مبيناً بالشكل (3-2)، نرمز إلى هذه الأشكال بـ H1، H2 ، H3 على الترتيب.

يُظهر الشكل (3-2) كيف نستطيع الحصول على المنحني Hi+1 بتركيب أربعة منحنيات Hi أصغر حجماً بعد تدويرها باتجاهات معينة، والوصل بينها بثلاث قطع مستقيمة رسمناها بخط عريض..

لنفترض أن الأدوات الأساسية المتوفرة للرسم هي:

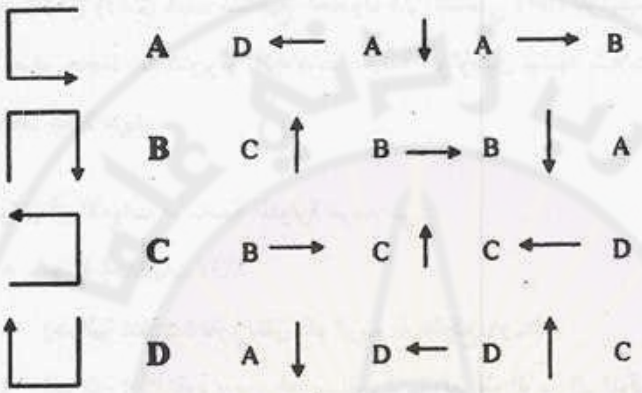
- جملة إحداثيات XOY
- إجرائية Setplot تقوم بنقل قلم الرسم إلى الموقع (x, y)
- إجرائية Plot تقوم برسم خط مستقيم من موقع قلم الرسم إلى الموقع (x, y)



الشكل (3-2): طريقة تركيب منحنيات هلبرت من H1 إلى H3

لما كان كل منحن Hi يتألف من أربعة منحنيات Hi-1 متصلة، فإننا نستطيع التعبير عن إجرائية رسم Hi كتركيب لأربع إجرائيات تقوم كل منها برسم Hi-1 بالاتجاه والقياس

المناسبين. إذا رمزنا إلى هذه الإجرائيات الأربع بالأحرف A, B, C, D وإذا رمزنا بأسهم إلى عمليات رسم الوصلات، فإننا نستطيع التعبير عن المخطط العودي لهذه الإجرائيات كما يلي:



إذا كانت h طول القطعة المستخدمة في رسم الخط المنكسر، فإن كتابة الإجرائية A ستأخذ التركيب التالي لـ D و B مع A:

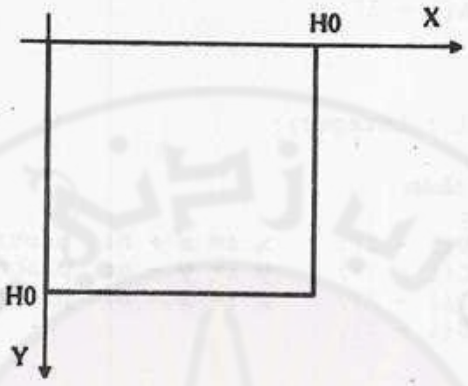
```
procedure A (i : integer);
begin
     if i > 0 then
          begin
               D(i - 1);      x := x - h;  plot;
               A(i - 1);      y := y - h;  plot;
               A(i - 1);      x := x + h;  plot;
               B(i - 1);
          end
end;
```

لتحديد نقطة بداية الرسم سنفترض أن منطقة الرسم عبارة عن مربع طول ضلعه h0، عندئذ سنلاحظ أن رسم الخط Hi يبدأ من الموقع:

$x = h0 + h0 / 2^i$
$y = h0 - h0 / 2^i$

أما طول القطعة المستخدمة في رسم الخط Hi فهو:

$h = h0 / 2^i$



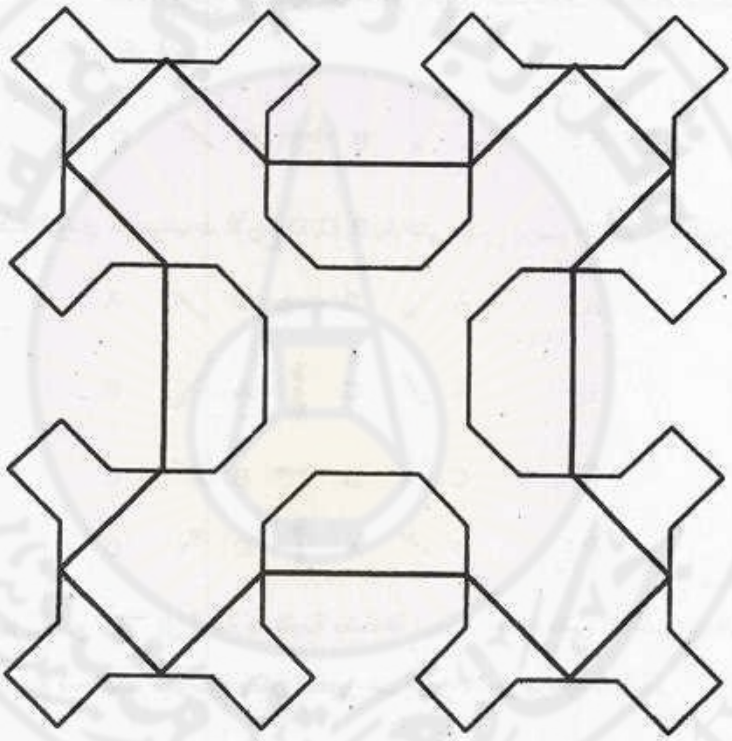
بهذا نكمل تفصيل الخوارزمية ويصبح برنامج رسم منحنيات هلبرت كالتالي:

```
program Hilbert;
const
    n    = 5;
    h0   = 512;
var
    i, h, x0, y0, x, y : integer;

procedure A (i : integer);
begin
    if i > 0 then
        begin
            D(i - 1);     x := x - h;  plot;
            A(i - 1);     y := y + h;  plot;
            A(i - 1);     x := x + h;  plot;
            B(i - 1);
        end
end;

procedure B (i : integer);
begin
    if i > 0 then
        begin
            C(i - 1);     y := y - h;  plot;
            B(i - 1);     x := x + h;  plot;
            B(i - 1);     y := y + h;  plot;
```

```
            A(i - 1);
        end
end;

procedure C (i : integer);
begin
    if i > 0 then
        begin
            B(i - 1);     x := x + h;  plot;
            C(i - 1);     y := y - h;  plot;
            C(i - 1);     x := x - h;  plot;
            D(i - 1);
        end
end;

procedure D (i : integer);
begin
        if i > 0 then
            begin
                A(i - 1);     y := y + h;  plot;
                D(i - 1);     x := x - h;  plot;
                D(i - 1);     y := y - h;  plot;
                C(i - 1);
            end
end;

begin { Program }
    i   := 0;
    h   := h0;
    x0  := h div 2;
    y0  := x0;
    repeat
        i   := i + 1;
        h   := h div 2;
        x0  := x0 + h div 2;
        y0  := y0 - h div 2;
        x   := x0;
        y   := y0;
        Setplot;
        A(i)
    until i = n;
end {Program }.
```

3-4-3 منحنيات سيربنسكي

نعرض في هذه الفقرة مثالا لمنحنيات تشبه من حيث التركيب منحنيات هلبرت، لكنها أكثر تعقيدا وأناقة. يبين الشكل (3-4) المنحنيين S1 و S2.

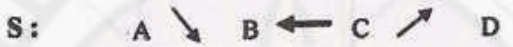


الشكل (3-4) منحنيا سيربنسكي S1 بالغامق و S2.

من ملاحظة هذين الرسمين نجد أننا لا نستطيع الاعتماد على S1 في بناء S2 حتى لو حذفنا أحد أضلاعه. هذا ناتج من كون منحنيات سيربنسكي منحنيات مغلقة، ويجب

اعتماد مخطط عودي لخوارزمية الرسم، بحيث نحصل على منحنيات مفتوحة يؤدي وصلها إلى الحصول على الرسم المطلوب.

سنعتبر أن منحني سيربنسكي من الدرجة n هو تركيب لأربع إجرائيات A, B, C, D تجمعها أربع قطع مستقيمة لا تنتمي إلى هذه المنحنيات ويعطي المنحني المطلوب. إن المنحنيات الأربعة متماثلة وينتج كل منها من الآخر بدوران قدره 90. المخطط الأساسي لرسم منحني سيربنسكي هو:



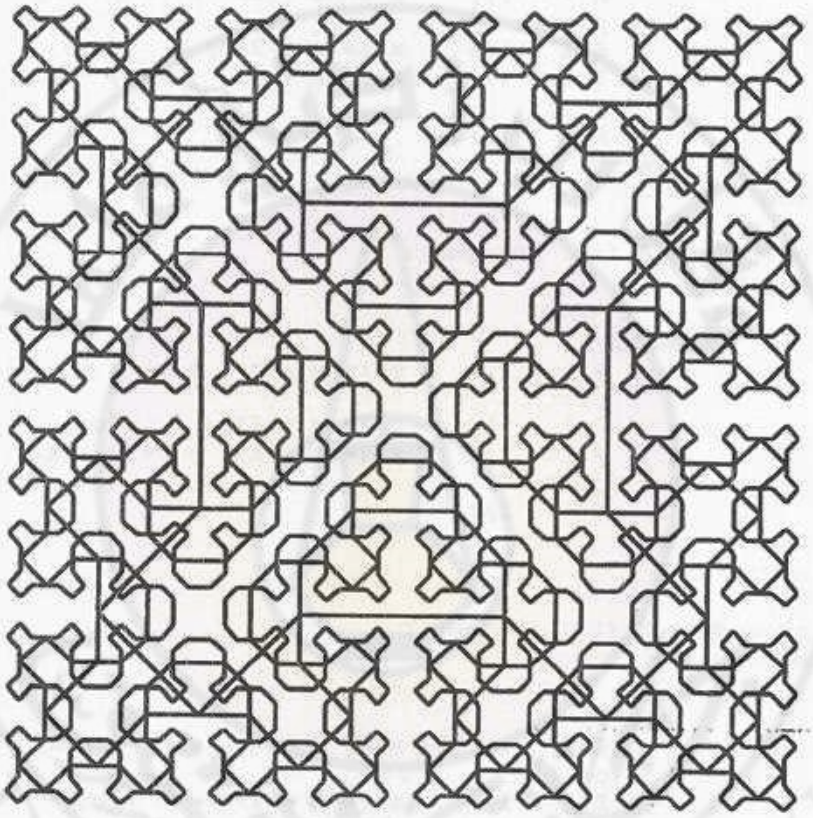
أما المخطط العودي للإجرائيات الأربع A, B, C, D التي تقوم برسم المنحنيات الجزئية فهو:

حيث يرمز السهم ⟶ إلى قطعة مستقيمة مضاعفة الطول. باتباع نفس الخطوات التي أوردناها لرسم منحنيات هلبرت، يمكن التعبير عن الجزء A بالإجرائية التالية:

```
procedure A (i : integer);
begin
    if i > 0 then
        begin
            A(i - 1);    x := x + h;  y := y + h;
            plot;
            B(i - 1);    x := x + 2*h;    plot;
            D(i - 1);    x := x + h;  y := y - h;
            plot;
            A(i - 1);
```

```
        end
end;
```

يبين الشكل (3-5) منحنيات سيربنسكي من S1 حتى S4.



الشكل (3-5) منحنيات سيربنسكي من S1 حتى S4.

```
program Sierpinsky;
const
    n = 4;
    h0 = 256;
var
    i, h, x0, y0, x01, y01, x, y : integer;
```

```
procedure A (i : integer);
begin
    if i > 0 then
        begin
            A(i - 1);     x := x + h;  y := y + h;
            plot;
            B(i - 1);     x := x + 2*h;     plot;
            D(i - 1);     x := x + h;  y := y - h;
            plot;
            A(i - 1);
        end
end;

procedure B (i : integer);
begin
    if i > 0 then
        begin
            B(i - 1);     x := x - h;  y := y + h;
            plot;
            C(i - 1);     y := y + 2*h;     plot;
            A(i - 1);     x := x + h;  y := y + h;
            plot;
            B(i - 1);
        end
end;

procedure C (i : integer);
begin
    if i > 0 then
        begin
            C(i - 1);     x := x - h;  y := y - h;
            plot;
            D(i - 1);     x := x - 2*h;     plot;
            B(i - 1);     x := x - h;  y := y + h;
            plot;
            C(i - 1);
        end
end;
procedure D (i : integer);
begin
    if i > 0 then
        begin
            D(i - 1);     x := x + h;  y := y - h;
            plot;
            A(i - 1);     y := y - 2*h;     plot;
```

```
            C(i - 1);    x := x - h;  y := y - h;
            plot;
            D(i - 1);
        end
end;

begin { Program }
    i   := 0;
    h   := h0 div 4;
    x0  := 2 * h;
    y0  := h;
    x01 := x0;
    y01 := y0;
    repeat
        i   := i + 1;
        x01 := x01 - h;
        h   := h div 2;
        y01 := y01 - h;
        x0  := x01;
        y0  := y01;
        x   := x0;
        y   := y0;
        A(i);    x := x + h;  y := y + h;  plot;
        B(i);    x := x - h;  y := y + h;  plot;
        C(i);    x := x - h;  y := y - h;  plot;
        D(i);    x := x + h;  y := y - h;  plot;
    until i = n;
end { Program }.
```

3-5 تحويل الخوارزميات العودية إلى خوارزميات تكرارية

إن الخوارزميات العودية هي أداة سهلة للتعبير عن طرق حل بعض المسائل التي تقبل تعريفا عوديا لطريقة الحل. ولابد للقارئ من أن يلاحظ أن الخوارزميات العودية واضحة وسهلة الفهم، ويكفي الرجوع إلى الخوارزميات التي عرضناها في الفقرات السابقة ليتأكد له، ولكن البرامج العودية لها بعض المساوئ التي لابد من الإشارة إليها وهي:

- لا تقبل بعض لغات البرمجة (مثل لغة الآلة أو Cobol أو Fortran) التعريف العودي للإجرائيات.
- غالبا ما تكون البرامج العودية مكلفة من حيث زمن التنفيذ، وحجم الذاكرة المطلوبة.

سندرس في هذه الفقرة طريقة لتحويل الإجرائيات العودية إلى إجرائيات تكرارية مكافئة مع بعض الأمثلة التوضيحية. سنبدأ بالحالات البسيطة ثم ننتقل إلى الحالة العامة.

3-5-1 حذف الاستدعاء العودي المتطرف

إذا احتوت إجرائية على استدعاء عودي، وكان هذا الاستدعاء ينهي تنفيذ الإجرائية، فإن هذا الاستدعاء العودي يسمى متطرفا. وبمعنى آخر نقول عن استدعاء عودي إنه متطرف إذا كان لا يلي هذا الاستدعاء تنفيذ أي تعليمة أخرى من الإجرائية. سنوضح مفهوم الاستدعاء العودي المتطرف في ضوء بعض الأمثلة.

أ- إجرائية حل مسألة أبراج هانوي:

```
procedure hanoi (a, b, c : char ; n : integer);
begin
    if n = 1 then
        writeln ('move disk from ', a , ' to ', c)
```

```
    else
        begin
            hanoi (a, c, b, n-1);
            writeln('move disk from ', a , ' to ', c);
            hanoi (b, a, c, n-1);
        end
end;
```

إن الاستدعاء العودي الثاني في هذه الإجرائية طرفي، لأنه غير متبوع بتنفيذ أي تعليمة.

ب- خوارزمية البحث الثنائي (بطريقة تنصيف المجالات) عن عنصر X ضمن قائمة مرتبة T:

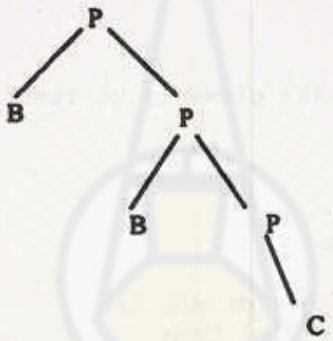
```
procedure dicho (X: element; T: list; g, d: integer;
var res: integer);
var
    m : integer;
begin
    if g <= d then
        begin
            m := (g + d) div 2;
            if X = T[m] then
                res := m
            else
                if X < T[m] then
                    dicho(X, T, g, m - 1, res)
                else
                    dicho(X, T, m + 1, d, res)
        end
    else
        res := 0
end;
```

إن الاستدعاءين العوديين في الإجرائية السابقة طرفيان.

لا علاقة للتعبير "طرفي" بموضع الاستدعاء العودي ضمن نص الإجرائية، وإنما يتعلق بتنفيذ الإجرائية فحسب. هذا ما نلاحظه من شجرة تنفيذ الإجرائية، فمثلا إذا كان مخطط الإجرائية من الشكل:

```
procedure    P;
begin
    if   A then
         begin
              B;
              P
         end
    else
         C
end;
```

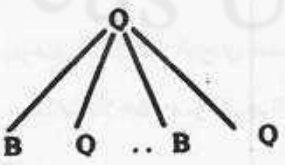
فإن شجرة تنفيذ الإجرائية ستكون من الشكل:



أما إذا كان مخطط الإجرائية من الشكل:

```
procedure Q;
begin
    while A do
         begin
              B;
              Q
         end
end;
```

فإن شجرة تنفيذ الإجرائية تكون من الشكل:



وفي مثل هذه الحالة لا يكون الاستدعاء العودي طرفيا على الرغم من أنه يقع في آخر تعليمة في نص الإجرائية.

بعد هذا التعريف للاستدعاء العودي المتطرف نستطيع تلخيص طريقة حذفه كما يلي:

لتكن الإجرائية العودية:

```
procedure P ( X : ... ; var Y : ...);
begin
        .
        .
        .
    P(E, Y)
        .
        .
        .
end;
```

حيث: X: قائمة متحولات الدخل

Y: قائمة متحولات الخرج

الاستدعاء العودي P(E, Y) متطرف.

إن الإجرائية السابقة مكافئة (من حيث التنفيذ) للإجرائية

```
procedure P (X : ... ; var Y : ...);
label start;
begin
start :     .
            .
            .
            X := E;
            goto start;
            .
            .
            .
end;
```

مثال

تقوم الإجرائية العودية التالية بإيجاد جداء عددين صحيحين:

```
procedure m (x, y : integer; var r: integer);
begin
    if y = 0  then
       r := 0;
    else
       if not odd (y) then
          m( x*2; y div 2; r)
       else
          begin
             m( x*2; y div 2; r);
             r := r +x
          end
end;
```

نستطيع حذف الاستدعاء العودي الأول لأنه طرفي فتصبح الإجرائية السابقة:

```
procedure m (x, y : integer; var r : integer);
label start;
begin
start :
    if y = 0  then
       r  := 0;
    else
       if not odd(y) then
          begin
             x := x*2;
             y := y div 2;
             goto start
          end
       else
          begin
             m(x*2, y div 2, r);
             r := r + x
          end
end;
```

كما نستطيع الاستغناء عن تعليمة goto وتحويلها إلى while فتصبح الاجرائية السابقة:

```
procedure m (x, y : integer; var r : integer);
begin
    while  y <> 0  and  not odd(y) do
        begin
            x :=  x*2;
            y :=  y div 2;
        end;
    if y = 0 then
        r := 0
    else
        begin
            m( x*2, y div 2, r);
            r := r +x
        end
end;
```

3-5-2 الحالة العامة لتحويل الإجرائيات العودية إلى إجرائيات تكرارية

في الحالة العامة تحول الإجرائيات العودية إلى إجرائيات تكرارية باتباع الخطوات التالية:

- حذف المعاملات والمتحولات الموضعية.
- تحويل الإجرائية العودية الناتجة من المرحلة السابقة إلى إجرائية تكرارية مكافئة.

في بعض الحالات يمكن إجراء بعض التعديلات والاختصارات على الإجرائية التكرارية الناتجة، بغية الحصول على إجرائية أكثر فعالية.

3-5-2-1 حذف المعاملات

الغرض من هذه المرحلة هو حذف المعاملات (Parameters) التي نجدها في ترويسة الإجرائية. سنسمي فيما يلي متحولا عاما كل متحول معرف في نفس الكتلة التي تحوي تعريف الإجرائية العودية.

يمكن التمييز بين نوعين من المعاملات: معاملات دخل (Input Parameters) ومعاملات خرج (Output Parameters). فيما يتعلق بمعالجة معاملات الخرج، ستقتصر معالجتنا للحالة عندما تكون معاملات الخرج للاستدعاء العودي هي نفس معاملات الخرج للإجرائية العودية. لحذف معاملات الخرج في هذه الحالة نضع في نص الإجرائية اسم المعامل الفعلي (Effective Parameter) الذي تستدعى الإجرائية معه.

لحذف معاملات الدخل نقوم بما يلي:

- نعرف لكل متحول دخل متحولا عاما يحمل الاسم نفسه (يجب الانتباه إلى عدم وجود متحول بالاسم نفسه).
- نحتفظ بقيم معاملات الدخل قبل الاستدعاء العودي، وذلك باستعمال مكدس (Stack) (انظر التعامل مع المكدسات في الفصل السادس).
- بعد الاستدعاء العودي، نعيد قيم معاملات الدخل إلى ما كانت عليه قبل الاستدعاء العودي.
- نفترض أن الإجرائية ستجد معاملات الدخل في قمة المكدس ومن ثم فإن أول تعليمة ستنفذها هي Pop(X)، ويتحول الاستدعاء الأولي للإجرائية إلى: P ;push(E) حيث E المعاملات الفعلية.

3-5-2-2 حذف المتحولات الموضعية

لحذف المتحولات الموضعية (Local Variables) نقوم بما يلي:

- نعرف لكل متحول موضعي متحولا عاما، يحمل الاسم نفسه.
- نحتفظ بقيم المتحولات الموضعية قبل الاستدعاء العودي وذلك باستعمال مكدس.

• بعد الاستدعاء العودي، نعيد قيم المتحولات الموضعية إلى ما كانت عليه قبل الاستدعاء العودي.

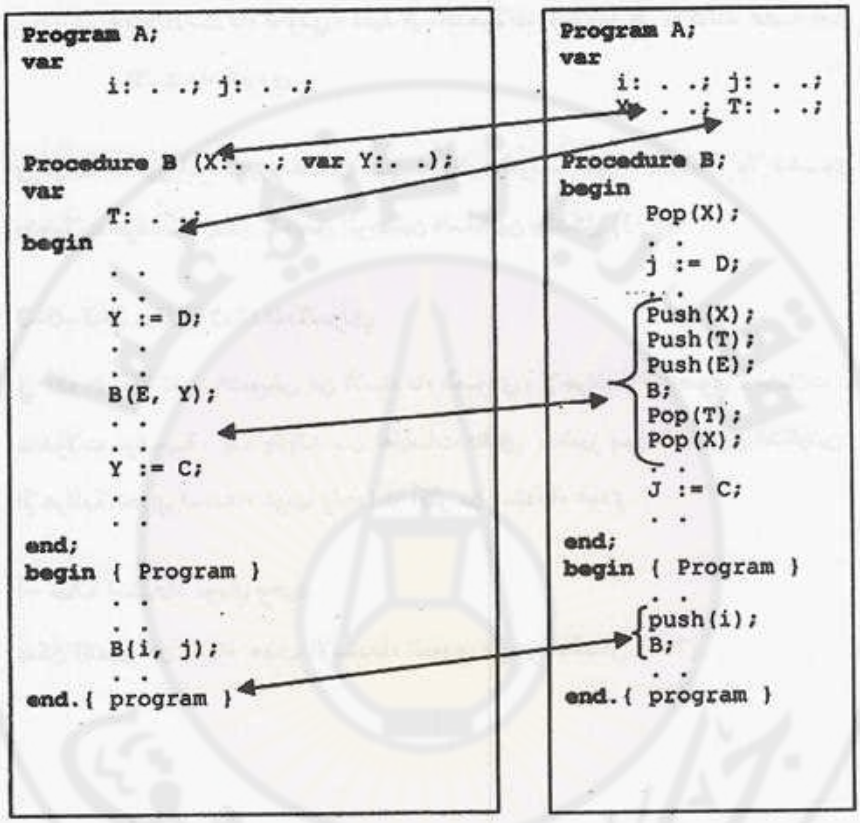
في نهاية هاتين المرحلتين نحصل على إجرائية عودية دون معاملات، ولا تحوي متحولات موضعية. يمكن تلخيص المرحلتين السابقتين بالشكل (3-6).

3-5-2-3 حذف الاستدعاء العودي

في هذه المرحلة نقوم بالتعويض عن الاستدعاء العودي، لإجرائية لا تحوي معاملات أو متحولات موضعية، بما يقابله من تعليمات goto. سنميز بين الحالتين التاليتين: الإجرائية تحوي استدعاء عوديا واحدا أو أكثر من استدعاء عودي.

آ- حالة استدعاء عودي وحيد

يمكن التعبير عن طريقة حذف الاستدعاء العودي الوحيد بالشكل (3-7).

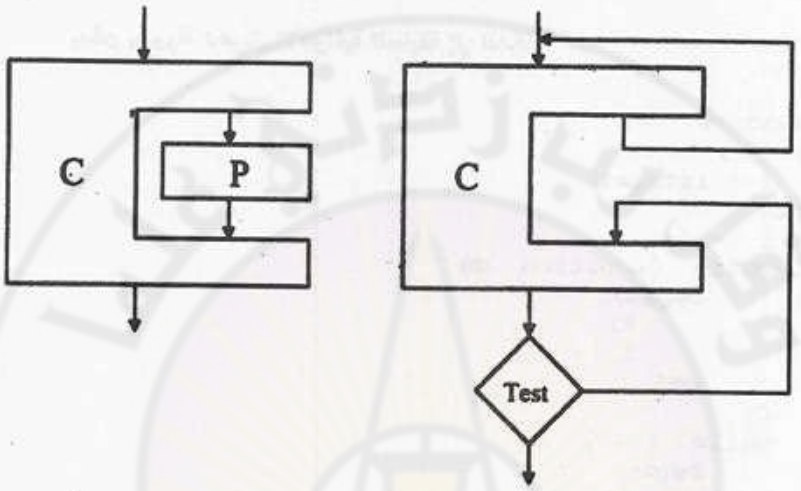


الشكل (3-6): طريقة حذف المعاملات والتحولات الوضعية

يوافق هذا المخطط حالة إجرائية من الشكل:

```
procedure P;
begin
    if Condition then
        begin
            A;
            P;
            B
        end
```

```
    else
        C
end;
```



الشكل (3-7): طريقة حذف الاستدعاء العودي الوحيد

بعد حذف الاستدعاء العودي الوحيد تأخذ هذه الإجرائية الشكل التالي:

```
procedure P;
var
    i : integer;
label start, L1;
begin
    i := 0;
start :
    if Condition then
        begin
            A;
            i := i+1;
            goto start;

L1 :
            B
        end
    else
        C ;
```

```
    if i > 0 then
        goto L1
end;
```

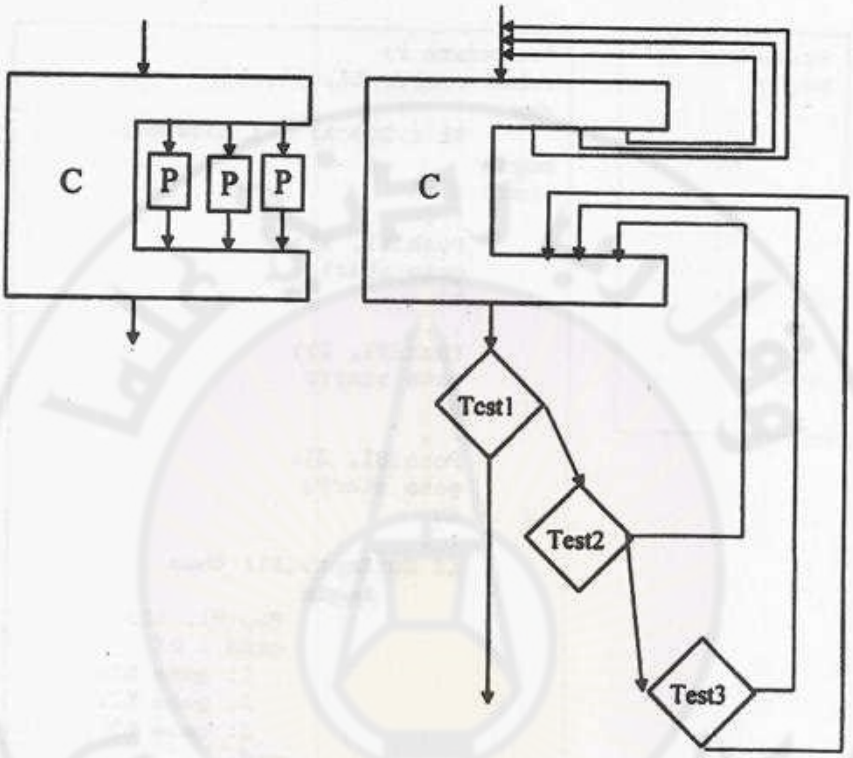
يمكن بسهولة تحويل الاجرائية السابقة إلى الشكل:

```
procedure  P;
var
    i : integer;
begin
    i := 0;
    while  Condition  do
        begin
            A;
            i := i+1
        end;
    C;
    while  i > 0  do
        begin
            B;
            i := i-1
        end
end;
```

ب- حالة أكثر من استدعاء عودي

يمكن التعبير عن طريقة حذف الاستدعاءات العودية بالشكل (3-8).

للتعبير عن الاختبارات $test_1, test_2, ..., test_n$ الموافقة للاستدعاءات العودية نستعمل مكدسا S1 نضع فيه أرقاما تدل على الاستدعاءات العودية، ونستعمل هذه القيم في تحديد آخر استدعاء مؤجل. يمكن التعبير عن طريقة استعمال المكدس بالشكل (3-9).



الشكل (3-8): طريقة حذف الاستدعاءات العودية

```
Procedure P;
begin
        . .
        P;
        A;
        . .
        P;
        B;
        . .
        P;
        C;
        . .
end;
```

```
Procedure P;
label start, L1, 12, 13;
var
      S1 : Stack; i : integer;
begin
start :
        . .
        Push(S1, 1);
        goto start;
L1 :    A;
        . .
        Push(S1, 2);
        goto start;
L2 :    B;
        . .
        Push(S1, 3);
        goto start;
L3 :    C;
        . .
        if NotEmpty(S1) then
              begin
                    Pop(S1, i);
                    case i of
                      1: goto L1;
                      2: goto L2;
                      3: goto L3
                    end
              end
end;
```

الشكل (3-9): استعمال مكدس لحفظ الاستدعاءات العودية

3-6 تمارين

حساب معين مصفوفة مربعة باستخدام طريقة النشر وحساب الـ Minors

نعرف minor لعنصر x من مصفوفة مربعة a(n×n) بأنه المصفوفة الجزئية الناتجة من a بعد حذف كل من السطر والعمود اللذين يحتويان x (أي نحصل على مصفوفة ذات أبعاد n-1×n-1). سنسمي minor العنصر a[i, j] بـ minor(a[i, j]). نعرف معين (Determinant) لمصفوفة a مربعة من الدرجة n عوديا كما يلي:

إذا كانت a(1×1) أحادية فإن: det(a) = a[i, j].

إذا كان بعد a أكبر تماما من 1 فنحسب معين a كما يلي:

نختار أحد الأسطر/و أحد الأعمدة (ومن الطبيعي تفضيل السطر أو العمود الذي يحتوي أكبر عدد من الأصفار) ونحسب لكل عنصر من السطر أو العمود الجداء التالي:

$$(-1)^{i+j} \times a[i,j] \times \det(\text{minor}(a[i,j]))$$

ويصبح معين a في حال اختيار النشر على السطر i كالتالي:

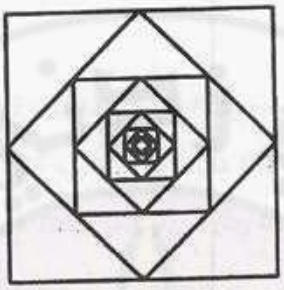
$$\det(a) = \sum_{j=1}^{n} (-1)^{i+j} \times a[i,j] \times \det(\text{minor}(a[i,j]))$$

ويصبح معين a في حال اختيار النشر على السطر j كالتالي:

$$\det(a) = \sum_{i=1}^{n} (-1)^{i+j} \times a[i,j] \times \det(\text{minor}(a[i,j]))$$

المطلوب كتابة إجرائية عودية det(a) تقوم بحساب معين المصفوفة المربعة a(n×n).

2- اكتب إجرائية عودية ترسم الأشكال الزخرفية Pn الناتجة من تراكب عدد من الأشكال من نفس النموذج. يبين الشكل التالي المنحني P5.



يشترط في إجرائية الرسم أن ترسم المنحني المطلوب دون رفع القلم ودون رسم قطعة مستقيمة أكثر من مرة واحدة.

3- اكتب نسخة تكرارية لإجرائية حل مسألة أبراج هانوي.

4- عمم الإجرائية convert (اللازمة لإجراء تحويل التعابير الرياضية المصدرة إلى تعابير ملحقة) وذلك لكي تؤخذ بعين الاعتبار حالة عملية النفي (عملية أحادية المعامل) حيث نرمز إليها بالرمز @.

5- اكتب الإجرائيات: intoprefix لتحويل تعبير نظامي إلى تعبير مصدر، posttoprefix لتحويل تعبير ملحق إلى تعبير مصدر، pretoinfix لتحويل تعبير مصدر إلى تعبير نظامي.

6- اكتب إجرائيتين الأولى عودية والثانية تكرارية لحساب تابع Ackermann المعرف على مجموعة الأعداد الطبيعية كما يلي:

$A(0, n) = n + 1$

$A(m, 0) = A(m - 1, 1)$ for $(m > 0)$
$A(m, n) = A(m - 1, A(m, n - 1))$ for $(m, n > 0)$

قارن بين الإجرائيتين السابقتين من حيث أكبر عدد طبيعي m نستطيع من أجله حساب A(m, m) باستخدام حاسوب معين.

7- لنفترض النمذجة الخوارزمية التالية للعبة Baguenaudier: ليكن لدينا الجدول T[1..N] من القيم البوليانية، في الحالة الابتدائية تكون كل القيم في T مساوية لـ True. الغرض من اللعبة وضع قيمة False في جميع خانات الجدول، مع العلم أنه لا يمكن تبديل قيمة خانة في الجدول إلا في الحالات التالية:

- يمكن دائما تبديل (أو قلب) قيمة T[1].
- يمكن تبديل قيمة T[2] إذا كانت: T[1] = True.
- في حال $j > 2$ يمكن تبديل قيمة T[j] إذا كان:

T[j-1] = True; T[1] = T[2] = ... = T[j-2] = False;

أ. وضح بنى المعطيات المناسبة.

ب. اكتب الإجرائية العودية Bag(m) حيث الحالة الابتدائية:

T[1] =T[2] = ... = T[m] = True

ت. والحالة النهائية المطلوبة هي: T[1] = T[2] = ... = T[m] = False.

ث. اكتب الإجرائية العودية DeBag(m) حيث الحالة الابتدائية:

T[1] = T[2] = ... = T[m] = False

ج. والحالة النهائية المطلوبة هي: T[1] = T[2] = ... = T[m] = True.

ح. اكتب إجرائية تكرارية ItrBag(m) تقوم بنفس عمل الإجرائية العودية Bag(m).

وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل الرابع

الخوارزميات التراجعية

4-1 مقدمة

الخوارزميات التراجعية (Backtracking Algorithms) هي خوارزميات عوديّة تعتمد طريقة "التجريب والخطأ" (Try-and-Error) في حل المسائل.

تتلخص هذه الطريقة ببناء الحل النهائي للمسألة عن طريق مجموعة من الخطوات، في كل خطوة نحدد الإمكانات المتاحة للخطوة التالية، ثم ندرس هذه الإمكانات بأن ننتقي أحدها، ونسجلها على أنها الخطوة التالية في الحل النهائي، ونتابع الخوارزميّة اعتماداً على هذه الخطوة. عندما يتأكد لنا أنّ اختيارنا لا يقود إلى الحل النهائي، أو يؤدي إلى طريق مسدود، نُعدِل عن هذه الخطوة. تشبه هذه العملية بناء سجل أخطاء يفيد في تجنب الوقوع في الخطأ مرتين.

يمكن اعتماد المخطط الخوارزمي التالي لحل المسائل المشابهة:

```
procedure try ;
begin
    initialize selection of candidates;
    repeat
        select next candidate;
        if acceptable then
           begin
               record it ;
```

```
            if solution incomplete then
                begin
                    try next step ;
                    if not successful then
                        cancel recording
                end
            end
    until successful or no more candidates
end;
```

تختلف تفصيلات هذا المخطط باختلاف المسألة المطروحة. سنتداول هذا المخطط في الفقرات التالية لنبين استخداماته المختلفة من خلال عرض حلول بعض أهم المسائل الشهيرة.

4-2 مسألة جولة حصان الشطرنج

لدينا رقعة شطرنج فيها n x n مربعاً، نضع حصاناً في موقع معين <X0, Y0> حيث: $1 \le X0 \le n$ و $1 \le Y0 \le n$ وعلينا إيجاد طريقة لتغطية الرقعة (إذا كان ذلك ممكناً) أي أن نوجد متتالية من الحركات عددها n^2-1 بحيث يحدث المرور بكل مربع مرة واحدة فقط.

لحل المسألة سنهتم فقط بحركة الحصان التالية: سنفترض في كل مرحلة أننا قمنا بعدد من الحركات، ولدينا مجموعة من الإمكانات التي يجب أن نجربها. في كل مرحلة لدينا عدد من الحركات الممكنة، نجرب إحدى هذه الحركات ونناقش أتؤدي إلى حل أم إلى طريق مسدود ؟

يمكن التعبير عن ذلك بالخوارزمية العودية المبسطة التالية والمستوحاة من المخطط الخوارزمي التراجعي العام:

```
procedure try next move ;
begin
    initialize selection of moves;
    repeat
        select next candidate from
```

```
        list of next moves;
            if acceptable then
                begin
                    record move;
                    if board not full then
                        begin
                            try next move;
                            if not successful then
                                erase previous
                                recording
                        end
                end
    until (move was successful) or
          (no more candidates)
end;
```

لتفصيل الخوارزميّة سنحدّد بنى المعطيات المستخدمة ومعاملات الإجرائية. نمثل الرقعة بمصفوفة مربعة h نعرفها بالشكل:

```
const
    n = 8;
type
    index = 1..n;
var
    h : array[index, index] of integer;
```

حيث نعتبر:

- h[X,Y] = 0 عندما لا يحدث المرور في المربع(X, Y)
- h[X,Y] = i عندما يحدث المرور في المربع (X,Y) في الخطوة i

معاملات الخوارزميّة

- الموقع الذي نريد تطبيق الخوارزميّة فيه: (X,Y)
- رقم الخطوة: i
- متحول خرج q يعطي نتيجة المناقشة: نجاح أو فشل

بناءً على ذلك يمكن تبسيط العبارة: "الرقعة ليست ممتلئة" (Board not full) بالشكل:

$i < n^2$

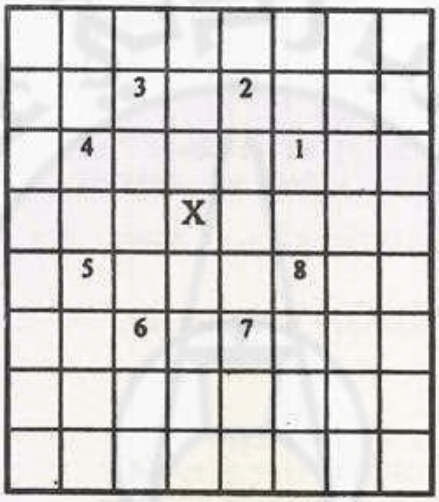
إذا استعملنا متحولين موضعيين u, v لتمثيل أحد المواقع الممكنة (حسب طريقة حركة الحصان المعروفة على شكل L) فإنّ العبارة "مقبول" (Acceptable) يمكن التعبير عنها بالشكل:

$h[u, v] = 0$ and $1 \le u \le n$ and $1 \le v \le n$

أخيراً نُدخل متحولاً موضعياً آخر ql لمراقبة الاستدعاء العوديّ للإجرائية، فتصبح الإجرائية كالتالي:

```
procedure try (i: integer; x, y: index; var q:
boolean);
var
    u,v : integer; ql : boolean;
begin
    initialize selection of moves;
     repeat
        let u,v be the coordinates of the next
        move defined by the rules of chess;
        if  (1<=u<=n) and (1<=v<=n)
            and (h[u,v]=0) then
            begin
                h[u,v] := i;
                if I < sqr(n) then
                    begin
                        try(i+1,u,v,ql);
                        if  ql then
                            h[u,v] := 0
                    end
                else
                    ql := true
            end
    until ql or (no more candidates);
    q :=ql
end;
```

المرحلة الأخيرة من تبسيط الخوارزميّة هي تحديد الحركات الممكنة التي يمكن إجراؤها من موقع معين<X, Y> . نعلم من قواعد لعبة الشطرنج أن الحصان يستطيع في الحالة العامة الانتقال إلى ثمانية مواقع يبينها الشكل (4-1).



الشكل (4-1): إمكانات انتقال الحصان على رقعة الشطرنج

يمكن الحصول على إحداثيات هذه المواقع الجديدة بإضافة فروق إلى إحداثيات موقع الحصان ونخزن هذه الفروق في جدولين a و b يعرفان بالشكل:

```
var
    a, b : array[1..8] of integer;
```

كما نستطيع استخدام عداد k لترقيم إمكانات الانتقال التالي. عند استدعاء الإجرائية نحدد المعاملين <X, Y> اللذين يمثلان الموقع الابتدائي للحصان، ونسند القيمة واحد للموقع h[X, Y].

البرنامج التالي يعطي الحل الكامل لمسألة جولة الحصان على رقعة شطرنج، حيث يبدأ الحصان من الموقع h[1, 1].

```
program knightstour (output);
const
    n = 5; nsq = 25;
type
    index = 1..n;
var
    i,j  : index;
    q    : boolean;
    s    : set of index;
    a,b  : array [1..8] of integer;
    h: array [index, index] of  integer;

procedure try (i: integer; x,y: index; var q:
boolean);
    var
        u,v,k: integer; q1: boolean;
begin
    k := 0;
    repeat
        k := k+1; q1 := false;
        u := x + a[k]; v := y + b[k];
        if (u in s) and (v in s) then
            if  h[u,v] = 0 then
                begin
                    h[u,v] :=i;
                    if i < nsq then
                        begin
                            try (i+1,u,v,q1);
                            if not q1 then
                                h[u,v] := 0
                        end
                    else
                        q1 := true
                end
    until q1 or (k=8);
    q := q1
end {try} ;

begin { program }
    s := [1,2,3,4,5];
    a[1] :=  2; b[1] :=  1;
```

```
    a[2] :=  1; b[2] :=  2;
    a[3] := -1; b[3] :=  2;
    a[4] := -2; b[4] :=  1;
    a[5] := -2; b[5] := -1;
    a[6] := -1; b[6] := -2;
    a[7] :=  1; b[7] := -2;
    a[8] :=  2; b[8] := -1;
    for i := 1 to n do
        for j := 1 to n do
            h[i,j] := 0;
    h[1,1] := 1;
    try(2,1,1,q);
    if q then
        for i :=1 to n do
            begin
                for j := 1 to n do write(h[i,j]:5);
                writeln
            end
    else
        writeln ('No Solution')
end. { Program }
```

يبين الشكل (4–2) حلين حصل عليهما بالمعطيات التالية:

1- n= 5 ; <X0, Y0> = <1, 1>
2- n= 6 ; <X0,Y0> = <2, 2>

6	14	11	16	3	32	9
5	17	30	13	10	23	4
4	12	15	2	31	8	33
3	29	18	27	24	5	22
2	36	1	20	7	34	25
1	19	28	35	26	21	6
	1	2	3	4	5	6

5	25	18	3	12	23
4	8	13	24	17	4
3	19	2	7	22	11
2	14	9	20	5	16
1	1	6	15	10	21
	1	2	3	4	5

الشكل (4–2): حلان لمسألة جولة الحصان

4–3 مسألة الوزراء الثمانية

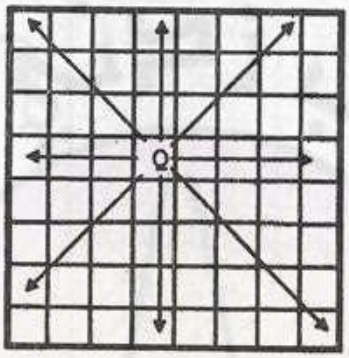
الغرض من هذه المسألة هو توزيع ثمانية وزراء على رقعة شطرنج. بحيث لا يكون فيها أيٌ واحد مهدداً للآخر. بحسب المخطط العام للخوارزميّات التراجعية فإنّ شكل الإجرائية هو:

```
procedure try(i : integer);
begin
    initialize selection of positions
    for i-th queen;
    repeat
        make next selection;
        if safe then
            begin
                setqueen;
                if i < 8 then
                    begin
                        try (i+1);
                        if not successful then
                            remove queen
                        end
                end
    until  successful or no more positions
end { try };
```

نعلم من قواعد الشطرنج أنه يمكن للوزير أن يهدد جميع المربعات الموجودة في العمود أو السطر أو القطر نفسه، كما يبين ذلك الشكل (4–3). لذا يجب وضع وزير واحد في كل عمود، وتؤدي عملية تحديد موقع الوزير رقم i إلى تحديد رقم السطر الذي سيوضع فيه ضمن العمود رقم i .

لحلّ هذه المسألة على الرقعة لابدّ من مناقشة عميقة لبنية المعطيات المستخدمة، بحيث تُحدَّد بنية تساعد في إجراء الاختبارات اللازمة عند تقرير وضع الوزير في مربع معين. في

البداية يمكن تعريف رقعة الشطرنج بأنها مصفوفة مربعة، لكن هـذا التمثيـل سـيؤدي إلى اختبارات صعبة لنعرف أيكون مربع ما آمناً (غير مهدّد) أم لا.



الشكل (4–2): حركات الوزير على رقعة الشطرنج

إنّ ما يهمنا هو موقع كل وزير ضمن العمود الموافـق، وأن نعـرف أيحـوي سـطرٌ معـين أو قطر معين وزيراً أم لا. لذا سنستخدم بنى المعطيات التالية:

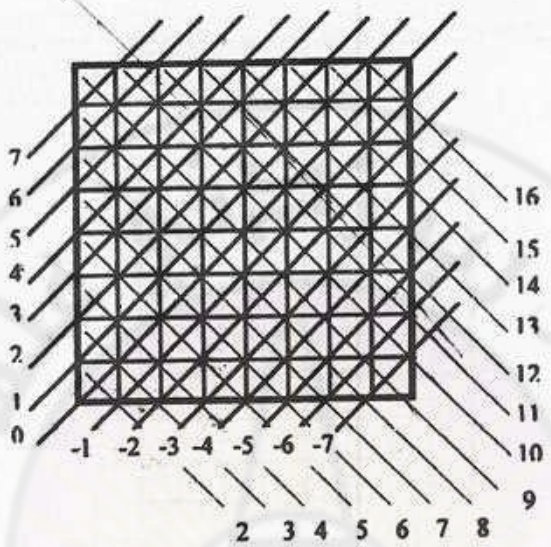
```
x : array[1..8]     of integer;
a : array[1..8]     of boolean;
b : array[2..16]    of boolean;
c : array[-7..7]    of boolean;
```

حيث:

- x[i] تحوي رقم السطر الذي يوضع فيه الوزير رقم i
- a[j] تعني أن السطر j لا يحوي أيَّ وزير
- c[j] تعني أن القطر المباشر رقم j لا يحوي أيَّ وزير

يبين الشكل (4–4) طريقة ترقيم الأقطار

- b[j] تعني أن القطر المتعامد رقم j لا يحوي أيَّ وزير



الشكل (4–4): طريقة ترقيم أقطار رقعة الشطرنج

بعد تحديد بنى المعطيات المُستخدمة نستطيع تفصيل الخوارزميّة، فنلاحظ أن المواقع المكنة للوزير رقم i هي كل مربعات العمود رقم i (عددها ثمانية). لاختبار هذه المواقع نستعمل عدّاداً j يكون في البداية صفراً ويزداد في كل مرة واحداً حتى يصل إلى القيمة 8. تكافئ عملية وضع الوزير رقم i في السطر j تنفيذ التعليمات التالية:

```
x[i]    := j;
a[j]    := false;
b[i+j]  := false;
c[i-j]  := false;
```

وتكافئ عملية رفع الوزير رقم i من السطر j تنفيذ التعليمات التالية:

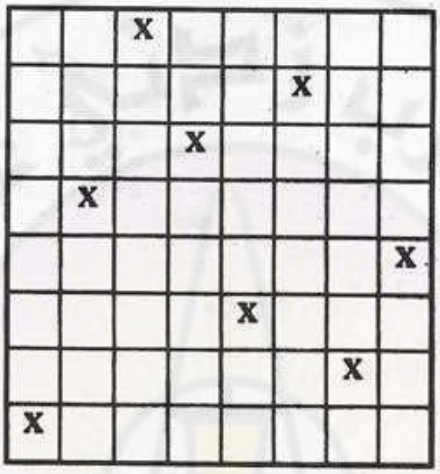
```
a[j]    := true;
b[i+j]  := true;
c[i-j]  := true;
```

ويمكن التعبير عن القضية "أمان" (Safe) بالشكل:

```
a[j] and b[i + j] and c[i - j]
```

تنتهي بذلك عمليات تفصيل الخوارزميّة ونورد فيما يلي النص الكـامل للبرنـامج. يعطـي هذا البرنامج الحل الممثل بالشكل (4–5):

x = (1, 5, 8, 6, 3, 7, 2, 4)



الشكل (4–5): أحد حلول مسألة الوزراء الثمانية

```
program EightQueens (output);
{ find one solution to eight queens problem }
var
    i : integer;
    q : boolean;
    a : array [1..8]  of boolean;
    b : array [2..16] of boolean;
    c : array [-7..7] of boolean;
    x : array [1..8]  of integer;

procedure try (i: integer; var q: boolean);
    var
        j : integer;
begin
    j := 0;
    repeat
        j := j+1; q := false;
        if a[j] and b[i+j] and c[i-j] then
            begin
```

```
                x[i]   := j;
                a[j]   := false;
                b[i+j] := false;
                c[i-j] := false;
                if i < 8 then
                    begin
                        try(i+1,q);
                        if not q then
                            begin
                                a[j]   := true;
                                b[i+j] := true;
                                c[i-j] := true;
                            end
                    end
                else
                    q := true
            end
    until q or (j=8)
end; { try }

begin { Program }
    for i :=  1 to 8     do a[i] := true;
    for i :=  2 to 16    do b[i] := true;
    for i := -7 to 7     do c[i] := true;
    try (1,q);
    if q then
        begin
            for i := 1 to 8  do write(x[i]:4);
            writeln
        end
    else
        writeln('No Solution')
end. { Program }
```

نستطيع الاستفادة من خوارزمية حل مسألة الوزراء الثمانية في تعميم المخطط العام للخوارزميات التراجعية، بحيث نوجد جميع الحلول لمسألة معينة. ولتحقيق ذلك يجب عدم إيقاف تجريب الإمكانات المتاحة عند الوصول إلى حل، وإنما نسجيل هذا الحل ونتابع تجريب الإمكانات الأخرى.

يصبح مخطط الخوارزميات التراجعية التي توجد جميع الحلول كالتالي:

```
procedure try(i: integer);
var
    k : integer;
begin
    for k:= 1 to m do
        begin
            select k-th candidate ;
            if acceptable then
                begin
                    record it ;
                    if i < n then
                        try(i+1)
                    else
                        print solution;
                    cancel recording;
                end
        end
end;
```

في حالة مسألة الوزراء الثمانية تصبح إجرائية البحث عن الحلول كالتالي:

```
procedure try (i: integer);
var
    j: integer;
begin
    for j := 1 to 8 do
        if a[j] and b[i+j] and c[i-j] then
            begin
                x[i]   := j;
                a[j]   := false;
                b[i+j] := false;
                c[i-j] := false;
                if i < 8 then
                    try(i+1)
                else
                    print;
                a[j] := true;
                b[i+j] := true;
                c[i-j] := true;
            end
end;
```

حيث يجري إظهار الحلول بالإجرائية:

```
procedure print;
var
    k : integer;
begin
    for k := 1 to 8 do
        write(x[k]:5);
    writeln
end;
```

بهذا نستطيع إيجاد كافة الحلول وعددها 92 حل، من بين هذه الحلول يوجد اثنا عشر حلا مستقلا (لا ينتج أحدها عن الآخر بتدوير الرقعة أو بالتناظر) يبينها الشكل (4–6).

x1	x2	x3	x4	x5	x6	x7	x8
1	5	8	6	3	7	2	4
1	6	8	3	7	4	2	5
2	4	6	8	3	1	7	5
2	5	7	1	3	8	6	4
2	5	7	4	1	8	6	3
2	6	1	7	4	8	3	5
2	6	8	3	1	4	7	5
2	7	3	6	8	5	1	4
2	7	5	8	1	4	6	3
3	5	2	8	1	7	4	6
3	5	8	4	1	7	2	6
3	6	2	5	8	1	7	4

الشكل (4–6): الحلول المستقلة لمسألة الوزراء الثمانية

4-4 مسألة الخيار الأمثل

يأتي هذا المثال ليبين كيف يمكن أن نستخدم المخطط العام للخوارزميات التراجعية في مقارنة الحلول التي يمكن إيجادها لمسألة معينة. درسنا في المثالين السابقين كيف يمكن استخدام هذا المخطط لإيجاد حل واحد، كما فعلنا في مسألة جولة الحصان، أو في مسألة الوزراء الثمانية، ثم عممنا هذا المخطط لنستطيع إيجاد جميع الحلول لمسألة الوزراء الثمانية. في هذه الفقرة سنتبع المنحى نفسه، لكننا سنحتفظ بحل واحد يكون أمثليا اعتمادا على معايير تسمح بمقارنة الحلول الممكنة.

لإيجاد الحل الأمثل لمسألة معينة يجب إيجاد جميع الحلول الممكنة بحيث نحتفظ دوما بحل واحد يعتبر أمثليا بحسب معيار معين. إذا افترضنا أننا نستطيع مقارنة الحلول بواسطة تابع موجب F(S) حيث (S أحد الحلول)، عندئذ يمكن الحصول على الحل الأمثل بتعديل المخطط العام للخوارزميات التكرارية التي توجد جميع الحلول بحيث نستبدل عملية إظهار الحل بالتعليمات التالية:

```
if f(Solution) > f(Optimal) then
    Optimal := Solution
```

وبذلك نحتفظ في كل لحظة بأفضل حل أمكن الحصول عليه في المتحول Optimal.

سنختار مثالا لمسألة الحل الأمثل مسألة هامة تتلخص بإيجاد الاختيار الأمثل من مجموعة عناصر معطاة. تجري عملية بناء الخيارات الممثلة للحلول المقبولة بالتدريج، عن طريق مناقشة كل عنصر من عناصر المجموعة الأساسية. نناقش في حالة كل عنصر فائدة ضم هذا العنصر إلى الاختيار أو عدم فائدته، ثم ننتقل إلى العنصر التالي عوديا.

عند فحص فائدة عنصر معين نستنتج أحد أمرين: إما أن نضم هذا العنصر إلى الخيار الآني الذي نحن بصدد بنائه، أو نستبعد هذا العنصر ونتابع بناء اختيارنا دونه. ويجب مناقشة هذين الاحتمالين كل على حدة، وهذا مما يجعل خوارزمية الحل تأخذ الشكل التالي:

```
procedure try (i : integer);
begin
    if inclusion is acceptable then
        begin
            include i-th object;
            if i < n   then
                try(i+1)
            else
                check optimality;
            eliminate i-th object
        end;
        if exclusion is acceptable then
            if i < n   then
                try (i+1)
            else
                check optimality
end;
```

تبين دراسة هذه الخوارزمية أن هناك 2^n اختيارا ممكنا (عدد أجزاء مجموعة مؤلفة من n عنصر) لذلك يجب أن نستفيد من معايير القبول في الحد كثيرا من هذا العدد. سنوضح طريقة تقليص هذا العدد في ضوء مثال محدد.

لتكن A = {a1, a2, . . ,an} مجموعة من الأشياء لكل منها وزن W وقيمة V، ولنفترض أننا نريد بناء مجموعة جزئية من A بحيث يكون مجموع أوزان عناصرها أقل من حد معين، ومجموع أسعارها أعظميا. تصادف هذه المسألة أي مسافر يستعد للسفر بالطائرة، ويقوم بإعداد حقيبته، وعليه اختيار مجموعة صغيرة من الأشياء لا يتجاوز وزنها العام الوزن المسموح به في المطار.

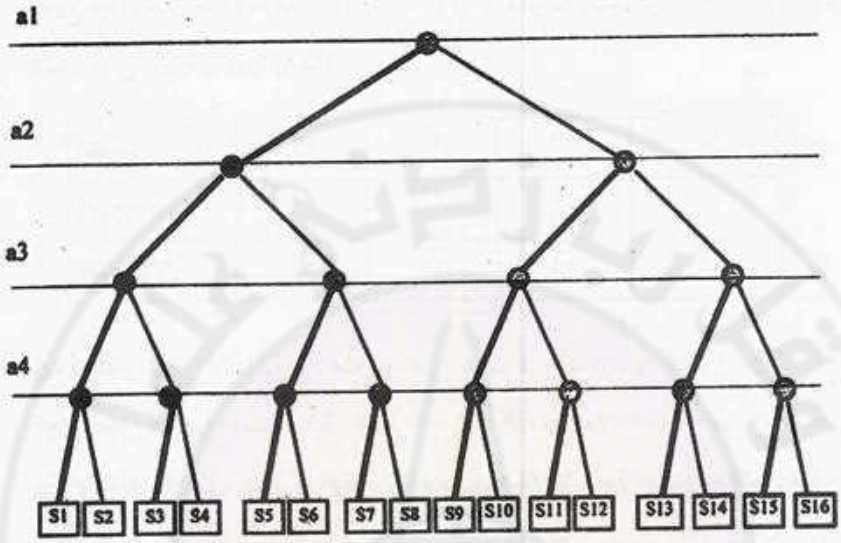
لتوضيح مبدأ العمل نأخذ حالة بسيطة، تحوي المجموعة فيها أربعة عناصر، يبين الجدول التالي وزن كل منها وقيمته.

العنصر	a1	a2	a3	a4
الوزن (W)	10	11	15	17
القيمة (V)	18	20	25	17

ونريد اختيار مجموعة جزئية من هذه العناصر بحيث لا يزيد وزنها الكلي عن 30 كغ وتكون قيمتها أعظمية. يمكن تمثيل تنفيذ خوارزمية البحث عن أفضل اختيار بالشجرة المبينة بالشكل (4–7)، تمثل كل عقدة داخلية في هذه الشجرة عملية مناقشة ضم أو استبعاد عنصر. (رمزنا إلى ناتج ضم العنصر بخط عريض). وتمثل الأوراق الخيارات الناتجة أو المجموعات الجزئية للمجموعة A (عددها 16).

من دراسة هذه الحلول يتبين أنه يجب استبعاد الحلول S1, S2, S3, S5, S9 لأنها لا تحقق شرط الوزن. أما بقية الحلول فيجب حساب قيمة كل منها، ويستطيع القارئ التحقق من أن الحل S10 هو أفضل حل، ويعطي القيمة 45.

لنفكر قليلا في استنتاج طريقة لتقرير ضم عنصر إلى المجموعة الجزئية التي نحن بصدد بنائها أو عدم ضمه. نرقم العقد الداخلية للشجرة السابقة كما في الشكل (4–8). تشبه عملية بناء المجموعات الجزئية التجول ضمن الشجرة السابقة بحيث نبدأ من العقدة رقم 1 ثم إلى العقدة رقم 2 ثم 4 ثم 8 وهكذا.

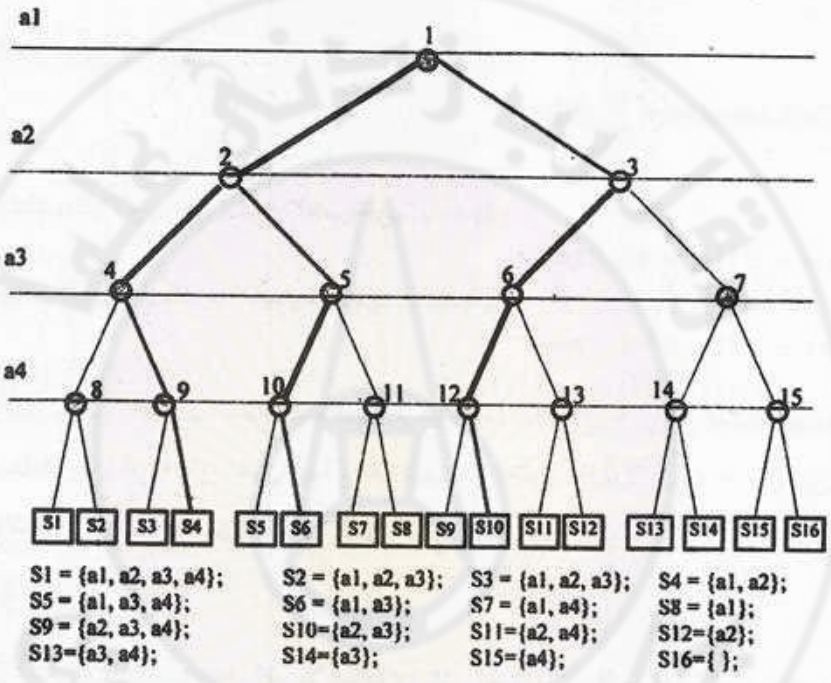


الشكل(4-7): شجرة تنفيذ خوارزمية البحث عن أفضل اختيار

عندما نصل إلى العقدة 4 مثلا، نكون قد أضفنا إلى المجموعة S التي نبنيها العنصرين a1 و a2، ونناقش أنضم العنصر a3 أم لا ؟ في هذه المرحلة نستطيع الاستغناء مباشرة عن الحلين S1 و S2، لأن ضم a3 إلى المجموعة S سيجعل الوزن الكلي لعناصر S يتجاوز الحد الأعلى. بمناقشة مماثلة نستنتج أن الحلول S3, S5, S9 مرفوضة. عندما نصل إلى الحل S4 (وهو أول حل مقبول من ناحية الوزن) نسجل مجموع قيم عناصر وليكن Maxv (في هذه الحالة Maxv = 38).

عندما نصل إلى العقدة 5 تكون S = {a1} ونكون قد قررنا استبعاد العنصر a2، فنجري المحاكمة التالية: إن مجموع قيم العناصر التي ضمت حتى الآن هو 18، وإذا استبعدنا a3 فإن أفضل مجموع قيم سنحصل عليه وهو 18+17=35 أصغر من أفضل قيمة حصلنا عليها حتى الآن (قيمة الحل S4)، ومن ثم فإن استبعاد العنصر a3 مرفوض في هذه

المرحلة. بهذا نكون قد استبعدنا الحلول S7 وS8. بطريقة مشابهة تستبعد الحلول من S11 إلى S16.



الشكل (4–8): استبعاد الحلول غير المفيدة

للتعبير عن طريقة المحاكمة السابقة على شكل خوارزمية نعتبر أننا نقوم ببناء المجموعة S أي ستحوي حلا مقبولا. في كل خطوة سنناقش ضم عنصر واحد إلى S وسنفترض أن العناصر مرقمة من 1 إلى n .

لنفترض tw الوزن الكلي لعناصر S وأن av هو مجموع قيم عناصر S إضافة إلى قيم العناصر التي لم نناقش بعد ضمها إلى S. بافتراض i رقم العنصر الذي نعالجه حاليا.

في البداية i = 1 :

$$Tw = 0; av = \sum_{i=1}^{n} a[i].v$$

عندما تصبح i = n يكون:

$$Tw \le Limw; av = \sum_{a[i] \in S} a[i].v$$

عندئذ يمكن التعبير عن الشرط "الضم مقبول" بالقضية:

```
tw + a[i].w <= limw
```

ويمكن التعبير عن الشرط "الاستبعاد مقبول" بالقضية:

```
av - a[i].v  >  maxv
```

والذي يكافئ القول إن مجموع قيم العناصر الموجودة حاليا في S والتي يمكن ضمها مستقبلا يمكن أن يتجاوز أفضل قيمة وصلنا اليها حتى الآن. عندما تكون i = n ويكون الاستبعاد مقبولا، فإنه من المؤكد أن الحل الناتج أفضل من آخر حل أمكن الحصول عليه.

نجد كل هذه التفصيلات في نص الإجرائية try التي يتضمنها البرنامج العام التالي. يبين الشكل (4–9) نتائج تنفيذ هذا البرنامج على مجموعة تحوي 12 عنصرا، حيث أعطينا الأوزان العظمى قيما تقع بين 10 و 150 كيلوغراما.

```
program Selection (input, output);
const
    n = 12;
type
    index = 1..n;
    objet = record
                v, w: integer
            end;
var
```

```
    i: index;
    a: array[index] of objet;
    limw, totv, maxv: integer;
    w1, w2, w3: integer;
    s, opts: set of index;
    z: array[boolean] of char;

procedure try (i: index; tw, av: integer);
var
    av1: integer;
begin
    if tw + a[i].w <= limw then
        begin
            s := s + [i];
            if i < n then
                try(i + 1, tw + a[i].w, av)
            else if av > maxv then
                    begin
                        maxv := av;
                        opts := s
                    end;
            s := s - [i]
        end;
    av1 := av - a[i].v;
    if av1 > maxv then
        if i < n then
            try(i + 1, tw, av1)
        else
            begin
                maxv := av1;
                opts := s
            end;
end;

begin { program }
    totv := 0;
    for i := 1 to n do
        with  a[i] do
            begin
                read(w,v);
                totv := totv + v
            end;
    read(w1, w2, w3);
    z[true]  := '*';
    z[false] := ' ';
```

```
    write('Weight ');
    for i := 1 to n do
        write(a[i].w : 4);
    writeln;
    write('Value');
    for i := 1 to n do
        write(a[i].v : 4);
    writeln;
    repeat
        limw := w1;
        maxv := 0;
        s := [];
        opts := [];
        try(1, 0, totv);
        write(limw : 5);
        for i := 1 to n do
            write(' ', z[i in opts]);
        writeln;
        w1 := w1 + w2
    until w1 > w3
end. { program }
```

الوزن	10	11	12	13	14	15	16	17	18	19	20	21
القيمة	15	18	15	17	21	20	19	21	23	24	24	25
10	*											
20										*		
30		*								*		
40		*			*	*						
50	*	*			*	*						
60	*	*	*	*	*							
70	*	*			*	*				*		
80	*	*	*		*	*			*			
90	*	*			*	*				*		*
100	*	*		*	*	*				*		
110	*	*	*	*	*	*		*	*			
120	*	*		*	*	*			*	*	*	
130	*	*	*	*	*	*		*	*	*		
140	*	*	*		*	*			*	*	*	*
150	*	*	*	*	*	*		*	*	*		*
160	*	*		*	*	*		*	*	*	*	*
170	*	*	*	*	*	*		*	*	*	*	*
180	*	*		*	*	*	*	*	*	*	*	*
190	*	*	*	*	*	*	*	*	*	*	*	*

الشكل (4–9): مثال لتنفيذ برنامج الاختيار الأمثل

4-5 تمارين

1- مسألة الزواج المستقر

نفترض وجود مجموعتين A و B لهما نفس العدد n من العناصر. المطلوب إيجاد مجموعة من n ثنائية <a,b> بحيث: $a \in A, b \in B$ وتحقق بعض الشروط. يمكن وضع العديد من الشروط على هذه الثنائيات ؛ لكن الشرط الذي يهمنا في هذه المسألة هو "شرط الزواج المستقر".

لتكن A مجموعة من الرجال وB مجموعة من النساء. كل رجل وكل امرأة يستطيع تحديد أفضليات من أجل اختيار الشريك الذي يناسبه. إذا تم اختيار الـ n زوجا (أي ثنائية) وبقي على الأقل رجل وامرأة ويفضل كلاهما الآخر على الشريك الذي اختير بنتيجة الخوارزمية نقول عن الزواج إنه ليس مستقرا. أما إذا لم يوجد مثل هذه الحالة فنقول عن هذا الزواج إنه مستقر.

لاحظ أن هذه المسألة تميز مجموعة هامة من المسائل مثل مسألة المفاضلة في الجامعة أو مسألة فرز الموظفين إلى المؤسسات.

نفترض أيضا أن قوائم أفضليات كل شخص تكون جاهزة قبل تنفيذ الخوارزمية ولا يجوز تعديلها أثناء عمل الخوارزمية.

بنفس الطريقة التي عالجنا فيه المسائل التراجعية:

أ. وضح وناقش بنى المعطيات المستخدمة (نفترض أن عدد الرجال هو 8 وعدد النساء أيضا هو 8 وكل شخص يستطيع تحديد قائمة من 8 أفضليات مرتبة).

ب. اكتب الإجرائية try(m : man) من أجل إيجاد الزوجة المناسبة للرجل m.

ت. اكتب برنامجا كاملا يحل المسألة من أجل إيجاد الثنائيات التي تحقق شرط الزواج المستقر.

2- إحدى شركات القطارات تخدم n محطة : S1, S2, . ., Sn وترغب في تقديم خدمة معلومات لزبائنها. تتلخص هذه الخدمة بإتاحة طرفيات للمستثمرين في محطات القطارات، يستطيع المستثمرون بواسطتها الاستفسار عن كيفية الذهاب من أي محطة Sd إلى محطة أخرى Sa. إجابة برنامج هذه الخدمة يجب أن تكون قائمة بالقطارات التي تصل المحطة Sd بالمحطة Sa (ليس بالضرورة وجود قطار مباشر بين المحطتين، ونفترض أيضا مراعاة زمن كاف للتبديل بين قطارين) بأسرع وقت ممكن.

بنفس الطريقة التي عالجنا فيه المسائل التراجعية:

أ. وضح وناقش بنى المعطيات المستخدمة (نفترض أن عدد محطات الشركة هو 10 والزمن الأصغري للتبديل بين قطارين لا يقل عن 15 دقائق).

ب. اكتب الإجرائية try(Sd, Sa : station) من أجل إعطاء قائمة مرتبة من القطارات التي تصل المحطة Sd بالمحطة Sa.

ت. اكتب برنامجا كاملا يحل المسألة.

وقل رب زدني علما
جامعة دمشق
Damascus University

الجزء الثاني

في بنى المعطيات والعمليات الأساسية عليها

وقل رب زدني علما
جامعة دمشق
Damascus University

مقدّمة الجزء الثاني

لتصميم البرامج الكبيرة والمعقدّة يُعتمد مبدأ التبسيط المتتالي، الذي يمكن تلخيصه بكتابة نسخة أولى من الخوارزمية، تصف المراحل الأساسية للحل دون الخوض في طريقة تحقيقها. تخضع هذه النسخة لمراحل عدّة من التبسيط بحيث يتم تفصيل العمليات التي تمّ وصفها في المرحلة السابقة. بعد مراحل عدّة من التبسيط نصل إلى خوارزمية تكون جميع عملياتها عبارة عن عمليات بسيطة.

لإنجاح هذا المبدأ، لا بدّ من إيجاد طريقة تمكّن من وصف المعطيات توصيفاً مجرّداً دون التطرّق (منذ النسخة الأولى للخوارزمية) إلى كيفية تحقيق هذا التوصيف. سنوضّح مفهوم التوصيف المجرّد للمعطيات، أو ما يسمّى بالأنماط المجرّدة (Abstract Data Types) من خلال المثال التالي:

نستخدم في برامجنا ثوابت ومتحوّلات من النمط الكسري ونقوم بعمليات على الأعداد الكسرية (+ ، - ، * ، ... الخ) دون ذكر خصائص هذه المعاملات لوضوحها. نتعامل مع الأعداد الكسرية دون الحاجة إلى معرفة طريقة التمثيل الداخلي (جزء عشري، قوّة) وفي أحسن الأحوال، يزوّدنا دليل استخدام لغة البرمجة، بالدقّة الواجب توخّيها عند مقارنة عددين كسريين.

يعتبر هذا المنحى في التجريد أبسط أنواع تعريف الأنماط المجرّدة: يتعامل المبرمج مع الأعداد الكسرية على أنها مجموعة من القيم، التي يمكنه أن يطبّق عليها مجموعة

محدّدة من العمليات، وذلك دون الحاجة إلى معرفة كيفية التمثيل الفعلي لهذه القيم أو التحقيق الفعلي للعمليات.

في بعض الأحيان يُعتمد منحى صاعد في التجريد، إذ تُعتمد مجموعة من الأنماط المعرّفة سابقاً في لغة البرمجة، مع العمليات والإجرائيات المطبّقة عليها، بحيث يقتصر التعامل مع هذه الأنماط على مجموعة العمليات والإجرائيات المرتبطة بكل نمط. وتساعد بعض لغات البرمجة مثل ADA وCLU، على اعتماد هذا المستوى من البرمجة.

غير أن المنحى الأكثر استعمالاً هو المنحى التنازلي الذي يسعى لإنشاء الخوارزميات، بحيث تُعرَّف أولاً أنماط المعطيات (مع تحديد مواصفات هذه الأنماط) ثمّ تُصمَّم الخوارزمية على هذا المستوى. بعد ذلك يُعمل على إيجاد التحقيق العملي للأنماط المعرّفة وللعمليات المعرّفة عليها، وصولاً إلى برنامج قابل للتنفيذ.

إن فوائد مبدأ التبسيط المتتالي في البرمجة متعدّدة نذكر منها:

- **سهولة تصميم الخوارزميات وتنفيذها**

  إذ ليس من الضروري أخذ جميع التفاصيل البرمجية بعين الاعتبار منذ البدء بالتصميم، بل يجري ذلك على مراحل، وهذا ما يجعل التركيز يجري في كل مرحلة على نقطة محدّدة.

- **التعريف الموحّد لأنماط المعطيات**

  وهذا ممّا يجعل الاستعمال المشترك لنمط معيّن من قبل الأجزاء المختلفة للبرنامج أسهل.

- **جعل البرامج أكثر هيكلية**

  وذلك عن طريق عزل الإجرائيات، الخاصّة بالتعامل مع أنماط المعطيات المعرّفة، عن باقي إجرائيات البرنامج التي تصبح في هذه الحالة مستقلّة عن التحقيق الفعلي لبنى المعطيات، ومن ثم مستقلّة عن الآلة المستخدمة.

- **سهولة تعديل البرامج وصيانتها ونقلها**

  إذ يكفي لتغيير التحقيق الفعلي لنمط معطيات معيّن، أو للعمليات الممكنة عليه، تعديل جزء بسيط من البرنامج يكون مستقلاً عن بقية أجزاء البرنامج.

في هذا الجزء سنركّز اهتمامنا في هذه المفاهيم من خلال استعراض العديد من بنى المعطيات الهامة المستخدمة في حل المسائل البرمجية.

الفصل الخامس

مفاهيم أساسية في بنى المعطيات

5–1 مقدمة

نمط المعطيات (Data Type) هو مفهوم مجرد يُعرَّف بواسطة مجموعة من الخواص المنطقية. وعندما يُعرَّف نمط المعطيات المجرد والعمليات الممكنة على هذا النمط، نستطيع تنفيذ (Implementation) هذا النمط (أو شكل تقريبي منه). يمكن أن يكون هذا التنفيذ عتادياً (Hardware) أو برمجياً (Software). ما يهمنا هنا هو التنفيذ البرمجي الذي يفترض وجود تعليمات عتادية تسمح بتفسير العمليات البرمجية وإنجازها.

يتضمّن التنفيذ البرمجي لنمط، توصيف كيفية تمثيل الكائنات (Objects أو المتحولات Variables) من هذا النمط بواسطة كائنات أخرى من أنماط أخرى معرفة سابقاً إضافةً إلى توصيف كيفية معالجة هذا الكائن بالتوافق مع العمليات المعرفة من أجله.

عندما يُختار التمثيل المناسب للكائنات من نمط معطيات معين وتُكتب أيضاً البرمجيات الجزئية التي تعالج هذا التمثيل، يصبح بمقدور المبرمج أن يستعمل نمط المعطيات هذا لحل المسائل.

من الأدوات الجيدة لتوصيف الخواص المنطقية لنمط معطيات: *نموذج النمط المجرد*.

5-2 نموذج النمط المجرد

يسمح نموذج النمط المجرد (Abstract Data Type) بتوصيف المعطيات من وجهة نظر خارجية (تركيبية). وهو، على العكس من الأنماط الواقعية التي تسمح بوصف بنى المعطيات كما يتم تنفيذها على الحاسوب (مثل لغة الباسكال)، يصف سلوك المعطيات بتعريف مجموعات رياضية وعمليات وشروط منطقية عليها. ويوضح الجدول (5-1) عملية التنميط (Typing) في لغات البرمجة المشهورة.

Prolog Lisp Assembler	Fortran Algol	Pascal Modula 2	ADA CLU
غير منمطة	منمطة باستخدام أنماط ثابتة	منمطة مع إمكان بناء أنماط جديدة	منمطة مع إمكان بناء أنماط جديدة وتنميط مجرّد

الجدول (5-1): التنميط في لغات البرمجة المشهورة

مثال

النمط المجرد للأعداد الطبيعية

```
Type        NatNumber
Kind        natural
Import      boolean
Operator
    Constructor
        0 : → NatNumber
    Access
        pair : NatNumber → NatNumber
    Transformer
        + : NatNumber × NatNumber → NatNumber
```

Axiom

(1) pair(0) = = true

(2) pair(x+y) = = (pair(x) and pair(y)) or ((not pair(x)) and (not pair(y)))

(3) 0 + x = = x

(4) x + 0 = = x

(5) x + y = = y + x

(6) (x + y) + z = = x + (y + z)

End

نستنتج من ثم أن توصيف نمط مجرد يشتمل على الأشياء التالية:

1- اسم النمط.

2-نوع (Kind) ويعطي مجال تعريف القيم لهذا النمط.

3-أنماط مساعدة أو مستوردة (Imported Types) وتعني التوصيف الخارجي المضاف إلى التوصيف الحالي.

4- مجموعة عمليات: توابع بوسطاء تعطي قيماً ضمن مجالات التعريف. ولها الأنواع الثلاثة التالية:

- الباني أو المولّد (Constructor or Genrator):

$C : S \rightarrow ST$

حيث ST من نوع النمط الذي في قيد التعريف.

الباني يبني قيمة النمط الجديد انطلاقاً من قيمة أخرى من نوع آخر S.

لاحظ أنّ S يمكن أن تكون غائبة (بان ثابت).

- الواصل أو المراقب (Access or Observer):

$A : ST \rightarrow S$

- المحوّل (Transformer):

$T : ST \rightarrow ST$

الذي يحوّل قيمة من النمط الجديد إلى قيمة أخرى.

5- مجموعة شروط قبليّة (Pre-Conditions): تعرّف منطقياً المجموعة الجزئية للمنطلق

6-مجموعة من الحقائق (Axioms): تسمح بتوصيف العمليات، ويجب أن تكون محققة دوماً من أجل القيم الجديدة، ومن ثم تعرف دلالية (Semantics) النمط (أو سلوكه Behavior).

5-2-1 وسطاء الأنماط

يمكن إضافة وسطاء إلى لنمط المجرد من أجل العمومية، وهذه الوسطاء هي وسطاء صورية (Formal Parameters) وتمثل أنماطاً أخرى غير معروفة أثناء التعريف، ويمكن التعويض فيها (Instanciation) أو نسخها أثناء تعريف العمليات.

مثال

يمكن توصيف مكدس (انظر الفصل السادس) من عناصر نمط غير معروف كما يلي (p هي متحول أو نسخة Instance من نمط Stack(T) ، n هي متحول من نمط T):

```
Type        Stack(T)
Kind        Stack
Import      boolean
operator
    Constructor
        EmptyStack : → stack(T)
    Access
        IsEmpty : stack(T) → boolean
        Top : stack(T) → T
    Transformer
        Push : stack (T) x T → stack(T)
        Pop : stack(T) → stack(T)
     Precondition
            (1) precond(Pop(p)) = = not IsEmpty (p)
            (2) precond(Top(p)) = = not IsEmpty(p)
     Axiom
            (1) IsEmpty(Emptystack) = = True
            (2) IsEmpty(Push (p, n)) = = false
            (3) Pop(Push (p, n)) = = p
            (4) Top(Push (p, n)) = = n
```

نلاحظ هنا أنه بتخصيص الوسيط الصوري العام T بأي نمط فعلي (Effective Type) (وليكن مثلاً NatNumber)، نحصل على نمط مجرد آخر يسمح بمعالجة مكدس من عناصر من نمط فعلي (أعداد طبيعية في مثالنا).

5-2-2 الخواص

العمومية كما هي معرّفة في الأنماط المجردة، مناسبة في معظم الحالات لبناء بنى معطيات ذات أنماط بوسطاء. ولكن قد يحدث أحيانا أن تخصيص النمط الوسيط يحتاج إلى توابع أكثر خصوصية لكي تُعرّف على النمط العام.

مثال

نريد تعريف النمط شعاع بوسيط ونعرّف عليه عملية الجمع بين شعاعين.

```
Type Vector(T)
Operator
    Access
        value : integer × vector(T) → T
    Transformer
        ++ : vector(T) × vector(T) → vector(T)

Axiom
        value (i, v1++v2) = = value (i, v1) + value (i, v2)
End
```

حيث التابع value(i, v) يعيد العنصر i من النمط T من الجدول v، والتابع + يقوم بجمع شعاعين.

لاحظ هنا أن + هي عملية على النمط T باعتبار أن value (i, v) يعيد قيمة من النمط T.

في هذه الحالة لا يمكن تخصيص النمط T بنمط آخر لا على التعيين، إذ يجب على النمط الفعلي قبول (قسراً) عملية الجمع. وبُغية تمييز الأنماط التي يمكن تعويضها بالنمط الصوري يجب تعريف خاصة (Property). هذه الخاصة لها اسم وتخصص مجموعة من العمليات باستخدام الحقائق المناسبة.

نقول في هذه الحالة أن النمط العام يتطلب الخاصة المعرّفة على النمط الوسيط. نقول أيضاً إن النمط يمتلك الخاصة P إذا كان تعريفه يتضمن عمليات P مع الحقائق المرتبطة بها.

كما يمكن لنمط عام أن يخصص بنمط آخر، إذا كان يمتلك على الأقل الخاصة التي يتطلبها النمط الآخر.

مثال

في المثال السابق، الخاصة التي يتطلبها T هي Monoide(T) وتعرّف بالشكل التالي:

```
Property Monoide (T)
Operator
    + : T × T → T
    0 : → T
Axiom
    0 + x = = x
    x + 0 = = x
    (x + y) + z = = x + (y + z)
End
```

ويصبح من ثم تعريف النمط شعاع كالتالي:

```
Type Vector(T)
Exigent Monoide(T)
Operator
    Access
        value : integer × vector(T) → T
    Transformer
```

++ : vector(T) × vector(T) → vector(T)

..
..

Axiom

value (i, v1++v2) = = value (i, v1) + value (i, v2)

End

لاحظ أن النمط الذي عرّفناه من قبل: NatNumber يقبل الخاصة Monoide، لذلك يمكن تخصيص T به. على حين النمط Stack(T) لا يقبل هذه الخاصة، ولهذا لا يمكن استبدال Stack(T) في Vector(T) لأن عملية الجمع غير معرفة على المكدسات.

ملاحظة 1

يجب عدم الخلط بين نمط مجرد عام وخاصة. فالأول يعرف مجموعة من الكائنات التي تميزها التوابع التي تعالجها (عمومية بنيوية Structural Generality). أمّا الخاصة فتعرّف مجموعة من التوابع التي تميز مجموعة من الأنماط (عمومية متصاعدة Incremental Generality).

ملاحظة 2

توابع الخاصة يمكن توزيعها في مجموعتين:

- التوابع الأساسية (Fundamental Functions)
- التوابع المشتقة (Derived Functions)

التوابع الأساسية هي التي يجب أن تكون معرّفة في النمط، أمّا التوابع المشتقة فيمكن أن تكون معرّفة في الخاصة فقط.

مثال

Property Order (T)
Import boolean
Fundamental Property
$\leq : T \times T \rightarrow$ boolean
Derived Property
$= : T \times T \rightarrow$ boolean
$\neq : T \times T \rightarrow$ boolean
$< : T \times T \rightarrow$ boolean
$> : T \times T \rightarrow$ boolean
$\geq : T \times T \rightarrow$ boolean
Axiom
(1) $x \leq x$ = = **true** { Reflexive }
(2) $x \leq y$ **and** $y \leq x = = x = y$ { Asymmetric }
(3) $x \leq y$ **and** $y \leq z = = x \leq z$ { Transitive }
(4) $x \neq y$ = = **not** $(x = y)$
(5) $x > y$ = = **not** $(x \leq y)$
(6) $x \geq y$ = = **not** $(x \leq y)$ **or** $(x = y)$

الخلاصة

قد يبدو للوهلة الأولى أن نموذج النمط المجرد يغالي في التجريد، ويبتعد عن الناحية التطبيقية المعروفة في لغات البرمجة. على حين في الواقع، كل لغات البرمجة "الحديثة" تعتمد تنفيذاً (Implementation) خاصاً لهذا النموذج. راجع على سبيل المثال لغة Turbo Pascal المشروحة في كتاب البرمجة (2) وكيفية بناء الوحدات (Units) فيها ومدى التشابه بين النمط المجرد وجزء الواجهة (Interface) الذي ليس إلاّ تنفيذاً للنمط المجرد بقواعد لغة Turbo Pascal. كما لاحظ أيضاً التشابه بين Import في النمط المجرد و uses في بنية الوحدة في لغة Turbo Pascal.

ساهم نموذج النمط المجرد في تأسيس الأفكار التي انتشرت مع ظهور البرمجة الغرضية التوجُّه، التي جرى فيها دمج (أو كبسلة Encapsulation) المعطيات مع سلوكها

(Behavior) أو العمليات التي تجري على هذه المعطيات في بنية الصف. كما يشمل هذا النموذج أيضاً من الناحية النظرية أفكاراً تطبيقية هامة في البرمجة الغرضية التوجه كالوراثة (Inheritance) والصفوف الوهمية (Virtual Classes) وغيرها.

نموذج النمط المجرد هو إذن الإطار النظري المناسب لدراسة بنى المعطيات من طريق توضيح كيفية بنائها وطبيعة العمليات التي يمكن إجراؤها على هذه البنى. سندرس فيما تبقى من الجزء الثاني من هذا الكتاب العديد من بنى المعطيات ضمن هذا المنظور مع الإشارة إلى هذا النموذج عند الضرورة.

3-5 أنماط المعطيات الأساسية

هي مجموعة الأنماط التي توفّرها لغات البرمجة، والتي يمكن استخدامها دون الحاجة إلى تعريفها. تسمح معظم لغات البرمجة بأنماط معيّنة أصبحت شبه معيارية. تشمل هذه الأنماط:

- نمط التعداد (Enumerated Type).
- نمط المجال الجزئي (Subrange Type).
- نمط الأعداد الصحيحة (integer).
- نمط الأعداد الكسرية أو الحقيقية (real).
- نمط المتحوّلات المنطقية (boolean).
- نمط المحارف (char).
- نمط سلاسل المحارف (string).
- نمط المجموعات (set).
- نمط التسجيلات (record).

- نمط الملفات (file).
- نمط المؤشرات (pointer).

سنستعرض فيما تبقى من هذا الفصل بسرعة هذه الأنماط دون التقيد بلغة برمجة معينة وبعض من العمليات المعرّفة عليها التي تزخر بها لغات البرمجة المختلفة.

5-3-1 نمط التعداد

يحوي نمط التعداد (أو النمط المعدود) مجموعة من الثوابت المرتبة مثل أسماء الأشهر، أنواع العملة، أسماء الألوان، ... ويوجد تقابل بينه وبين مجموعة جزئية من الأعداد الطبيعية. من العمليات التي يمكن إجراؤها عليه:

- السابق (Predecessor) لإيجاد الثابت السابق لثابت معين.
- اللاحق (Successor) لإيجاد الثابت التالي لثابت معين.
- الترتيب (Order) لإعطاء ترتيب الثابت ضمن مجموعة الثوابت المعرّفة للنمط

5-3-2 نمط المجال الجزئي

يحوي نمط المجال الجزئي مجالاً جزئياً من نمط متقطع مرتب (Discrete Type) (الأنماط: integer, boolean, char, Enumerated) مثل: 5..25, 'a'..'z' ويوجد تقابل بينه وبين مجموعة جزئية من الأعداد الطبيعية. من العمليات التي يمكن إجراؤها عليه: السابق واللاحق والترتيب المشروحة في الفقرة السابقة.

5-3-3 نمط الأعداد الصحيحة

يحوي نمط الأعداد الصحيحة مجموعة جزئية من مجموعة الأعداد الصحيحة. يختلف حجم هذه المجموعة الجزئية باختلاف الآلة. يجري التعامل مع عناصر هذا النمط

بالعمليات المعروفة على الأعداد الصحيحة وهي الجمع (+)، الطرح (-)، الضرب (*)، القسمة الصحيحة (div) وباقي القسمة (mod).

5-3-4 نمط الأعداد الحقيقية

يحوي نمط الأعداد الحقيقية مجموعة جزئية من مجموعة الأعداد الحقيقية، ويجري التعامل مع هذه الأعداد بالعمليات الأساسية: الجمع (+)، الطرح (-)، الضرب (*)، التقسيم (/). تختلف الدقّة التي تجرى بها هذه العمليات باختلاف الآلة (طريقة تمثيل الأعداد، عدد الخانات التي يمثّل فيها العدد).

من العمليات الأخرى التي يمكن إجراؤها على الأعداد الحقيقية:

- القطع (Truncation) الذي يعطي الجزء الصحيح من العدد الحقيقي.
- التدوير (Round) الذي يعطي أقرب عدد صحيح.

5-3-5 نمط المتحوّلات المنطقية

يحوي نمط المتحوّلات المنطقية قيمتين هما صح (true) وخطأ (false). أمّا العمليات الممكنة على متحوّلات هذا النمط فهي العمليات الثلاث: و (and)، أو (or)، والنفي (not). يبيّن الجدول (5-2) تعريف هذه العمليات.

p	q	p and q	p or q	not p
false	true	false	true	true
false	false	false	true	true
true	true	true	true	false
true	false	false	false	false

الجدول (5-2): العمليات على المتحولات المنطقية

5-3-6 نمط المحارف

يحوي نمط المحارف مجموعة من الحروف الطباعية (تشمل الأحرف الأبجدية اللاتينية والعربية والأرقام وعلامات الترقيم) ومحارف التحكم. لا توجد مجموعة معيارية من المحارف المعتمدة في كل النظم الحاسوبية، ولعلّ أكثر المحارف اعتماداً هي تلك المعرّفة من قبل المنظمة الدولية للمعايير (ISO: International Standard Organization) وخصوصاً النسخة الأميركية المسمّاة ASCII (American Standard Code for Information Interchange). تحوي هذه المجموعة 95 محرفاً طباعياً (منها الحروف اللاتينية الـ 26 والأرقام العربية العشرة وعدداً من الأحرف الطباعية الأخرى كعلامات الترقيم والفراغ) و33 محرف تحكّم (تستخدم بشكل خاص في نقل المعطيات وفي التحكّم بتجهيزات الطباعة والإظهار). إن مجموعة الأحرف الطباعية والأرقام مرتّبة ومتراصة بمعنى :

- x حرف أبجدي لاتيني ⇔ 'A' ≤ x ≤ 'Z' or 'a' ≤ x ≤ 'z'
- x رقم ⇔ '0' ≤ x ≤ '9'

نمط المحارف نمط مرتب حسب جدول ASCII: مثلاً '9' < .. < '1' < '0' < 'a' < 'A'.

من العمليات (التوابع) التي يمكن إجراؤها على نمط المحارف:

- ord الذي يعطي قيمة ترتيب المحرف في جدول ASCII
- chr الذي يعطي المحرف المقابل لعدد ضمن أرقام الترميز في جدول ASCII (التابع العكسي لـ ord).

5-3-7 نمط سلاسل المحارف

هذا النمط لم يصبح سالف التعريف إلاّ حديثاً في لغات البرمجة، فهو يُبنى اعتماداً على تعريف شعاع من المحارف. يحوي هذا النمط سلسلة المحارف نفسها، ومؤشراً إلى نهاية هذه السلسلة. وأدلة المحارف يمكن أن تبدأ عند الصفر أو الواحد، ويمكن أن تنتهي عند طول معين أو تعتمد على مؤشر نهاية السلسلة. من العمليات (التوابع) التي يمكن إجراؤها على سلاسل المحارف:

- الطول (Length) الذي يعطي طول السلسلة الفعلي (أو عدد المحارف بين بداية السلسلة ومؤشر نهاية السلسلة).
- الموقع (Position) الذي يعطي موقع بداية سلسلة محارف جزئية في سلسلة محارف أخرى.
- الدمج (Concatenation) التي تدمج سلسلتي محارف في سلسلة واحدة (وصل نهاية الأولى مع بداية الثانية).
- المقارنة (Comparison) لمقارنة سلسلتي محارف.
- السلسلة الجزئية (Sub-String) لحساب سلسلة جزئية من سلسلة أخرى اعتباراً من موقع معين في السلسلة الثانية ولطول محدّد للسلسلة الجزئية.
- النسخ (Copy) لنسخ سلسلة محارف في سلسلة أخرى.

5-3-8 نمط المجموعات

يوجد نمط المجموعات في بعض لغات البرمجة، وهو يحوي مجموعة من العناصر التي تنتمي إلى أحد الأنماط المنقطعة (integer, boolean, char, Enumerated). من العمليات التي يمكن إجراؤها على نمط المجموعات:

- التقاطع (∩) بين مجموعتين ونرمز إليه بـ *.
- الاجتماع (∪) بين مجموعتين ونرمز إليه بـ +.
- الفرق (/) بين مجموعتين ونرمز إليه بـ -.
- المساواة (=) بين مجموعتين.
- عدم المساواة (≠) بين مجموعتين ونرمز إليه بـ <>.
- الاحتواء الجزئي (⊆) لمجموعة في مجموعة أخرى ونرمز إليه بـ =<.
- الاحتواء الكلي (⊂) لمجموعة في مجموعة أخرى ونرمز إليه بـ <.
- انتماء (∈) عنصر إلى مجموعة ونرمز إليه بالكلمة in.

5-3-9 نمط التسجيلات

يسمح نمط التسجيلات بتجميع عدد من الأنماط الأخرى، وهذا مما يساعد على التعامل معها ككتلة واحدة. ولعّل هذا النمط (مع نمط المؤشرات) من أهمّ الأنماط التي تساعد في بناء بنى معطيات جديدة، والتي سندرس العديد منها بالتفصيل في الفصول المتبقية من هذا الكتاب.

تحوي التسجيلة إذن مجموعة من العناصر التي تنتمي إلى أنماط أخرى (تسجيلات أيضاً) نسميها حقول التسجيلة. العملية الأساسية المعرّفة في هذا النمط هي النقطة (.) التي تسمح بالوصول إلى أي حقل من حقول التسجيلة.

5-3-10 نمط الملفات

يحوي نمط الملفات سلسلة من العناصر التي لها نفس النمط (كالتسجيلات، الأعداد بأنواعها، المحارف). الفرق الأساسي بين هذا النمط والأنماط الأخرى هو التخزين الدائم للملفات في الذاكرة الثانوية (أو القرص الصلب)، لا في الذاكرة الرئيسية كما هو الحال في

الأنماط الأخرى. ويمكن التعامل مع هذه الملفات بواسطة ملفات منطقية تشحن (أو جزء منها) إلى الذاكرة الرئيسية. من العمليات الممكنة على الملفات:

- تأهيل ملف للقراءة منه، ويعني وضع مؤشر الملف في بداية الملف وتجهيزه للقراءة.
- تأهيل ملف للكتابة فيه، ويعني وضع مؤشر الملف في بداية الملف وتجهيزه للكتابة.
- قراءة العنصر الذي يؤشر إليه مؤشر الملف والانتقال إلى العنصر التالي.
- كتابة عنصر في المكان الذي يؤشر إليه مؤشر الملف والانتقال إلى العنصر التالي.
- إغلاق الملف.
- اختبار وصول الملف إلى نهايته (عادة نستخدم الرمز eof).

يمكن في بعض لغات البرمجة تمييز نوع خاص من الملفات وهي الملفات النصية. فرقها الأساسي عن الملفات الأخرى أنها لا تحوي إلا محارف وتكون مخزنة على شكل أحرف، لا بشكل ثنائي (Binary). تبقى العمليات المعرفة على الملفات مطبقة بنفس الطريقة على الملفات النصية، مع إضافة عملية اختبار نهاية السطر في الملف النصي.

5-3-11 نمط المؤشرات

تستعمل بعض لغات البرمجة نمط المؤشرات للتغلب على بعض الصعوبات في التعامل مع بنى المعطيات التقليدية (كالملفات والأشعة) مثل وجوب المعرفة السابقة لحجوم هذه البنى، والكلفة الباهظة التي تسببها عمليات الإضافة والحذف والبحث، بسبب التجاور بين العناصر في الذاكرة.

نعرف المؤشر بأنه بنية معطيات تحوي عنوانا أو دليلا متحولا من نمط معين. ومن ثم يجب اعتبار المؤشر إلى نفس النمط نمطا مستقلا في ذاته. يمكن أن يحوي المؤشر قيمة خاصة nil وتعني أن هذا المؤشر لا يؤشر إلى شيء. من العمليات الممكنة على المؤشرات:

- حجز ذاكرة للمتحول الذي يؤشر إليه المؤشر ((new(p في لغة باسكال).
- تحرير الذاكرة المحجوزة بمؤشر ((dispose(p في لغة باسكال).
- الوصول إلى الخانة التي يؤشر إليها المؤشر (p^ في لغة باسكال).

الخلاصة

حاولنا في هذه الفقرة تلخيص الأفكار المهمة في أنماط المعطيات الأساسية في معظم لغات البرمجة، وإعطاء فكرة موجزة عن أهم العمليات التي تطبق عليها تماشيا مع طريقتنا في الاعتماد النظري على مفاهيم نموذج النمط المجرد في فهم بنى المعطيات.

باستخدام هذه الأنماط الأساسية، نستطيع بناء ما نريد من بنى معطيات جديدة، لتساعد في حل المسائل البرمجية المعقدة بفعالية.

5-4 تمارين

1- اكتب النمط المجرد للأنماط المعرفة في الفقرة 5-3.

2- اكتب تنفيذ الأنماط المعرفة في الفقرة 5-3 بلغات البرمجة التي تعرفها.

3- من بنى المعطيات "البسيطة" التي درستها في السنة السابقة نمط التاريخ الذي يحوي ثلاثة حقول (السنة، الشهر، اليوم).

أ. وسع هذا النمط ليحوي ثلاثة حقول إضافية (الساعة، الدقيقة، الثانية).

ب. عرف بعض العمليات الممكنة على هذا النمط.

ت. اكتب نموذج النمط المجرد لنمط التاريخ موضحا العمليات الممكنة عليه.

4- من بنى المعطيات "البسيطة" التي درستها في السنة السابقة نمط الأعداد العقدية. اكتب نموذج النمط المجرد للأعداد العقدية موضحا جميع العمليات الممكنة عليها.

الفصل السادس

بنى المعطيات الخطية

6-1 مقدمة

تُعدُّ البنى الخطّية أكثر البنى استعمالاً لتنظيم المعطيات، لأنّها تسمح بتنظيم المعطيات التي نريد معالجتها تنظيماً تسلسلياً، وهذا المبدأ في المعالجة هو المطبّق في معظم برامج تنظيم المعطيات وإدارتها.

تعريف 1

بنى المعطيات الخطّية هي البنى التي يوجد بين عناصرها علاقة ترتيب كلي.

تعريف 2

لتكن P مجموعة، نقول إن P مزوّدة بعلاقة ترتيب كلي R إذا تحقّقت الشروط التالية:

- *الانعكاسية*: $\forall x \in P \; x R x$
- *التعدّي*: $\forall x, y, z \in P \quad x R y \text{ and } y R z \Rightarrow x R z$
- *التخالفية*: $\forall x, y \in P \quad x R y \text{ and } y R x \Rightarrow x = y$
- *الشمولية*: $\forall x, y \in P \Rightarrow x R y \text{ or } y R x$

إن موضوع الترتيب في البنى الخطّية هو خاصّة أساسية لهذه البنى، فهي تسمح بإيجاد تقابل بين عناصر البنية ومجموعة جزئية من مجموعة الأعداد الطبيعية. يسمح هذا التقابل بالوصول إلى عنصر من بنية معطيات خطّية، عن طريق العدد الصحيح المقابل أو ما يسمّى عادة دليل العنصر.

نقدّم في هذا الفصل البنى التالية كأمثلة لبنى المعطيات الخطّية:

- الأشعّة (المصفوفات الوحيدة البعد)
- السلاسل
- المكدسات
- الأرتال

6-2 الأشعة

تعتبر الأشعة أو الجداول (Arrays) أكثر بنى المعطيات الخطية انتشارا، بسبب وجود تعريف صريح لها في معظم لغات البرمجة، ومنها اللغات القديمة كلغة FORTRAN.
الشعاع هو بنية متجانسة، بمعنى أنه يتألف من مجموعة مكونات من النمط نفسه. يوصف الشعاع بأنه بنية ذات وصول مباشر، أي إنه يمكن الوصول إلى أي عنصر من شعاع وصولا مباشرا. يعرف الشعاع بثلاثة محددات وهي:

- اسم النمط (Type Name).
- طريقة ترقيم المكونات: أي اعتماد نمط من نوع دليل (Index أو مجال جزئي) يسمح بالوصول إلى مكونات الشعاع.
- نمط مكونات الشعاع.

وبذلك يصبح تعريف بنية الشعاع على كما يلي:

```
type
    T = Array [Index] of TO;
```

حيث TO هو نمط مكونات الشعاع.

مثال

لتعريف جدول من 80 عدداً صحيحاً.

```
type
    Table = Array [1..80] of integer;
```

6-3 السلاسل الخطية

تعتبر السلاسل الخطية (Linear Lists) من أبسط بنى المعطيات وأكثرها استعمالاً. لننظر في الأمثلة التالية:

- أيام الأسبوع: (السبت، الأحد، الاثنين، الثلاثاء، الأربعاء، الخميس، الجمعة)
- الأحقاب الجيولوجية: (كامبري، سيلوري، ديفوني، كربوني، بيرمي، ترياسي، جوراسي، كريتاسي، ...)

إذا أردنا أن نوجد تجريداً للسلسلة نقول إنها إمّا أن تكون فارغة أو من الشكل:

L = (a1, a2, ..., an)

حيث ai عناصر من مجموعة معيّنة S.

يجري التعامل مع السلسلة بمجموعة من العمليات تتضمّن ما يلي:

- إيجاد طول السلسلة.
- قراءة السلسلة من البداية إلى النهاية (أو من النهاية إلى البداية).
- إيجاد العنصر رقم i من السلسلة.

- تخزين عنصر معيّن في المكان رقم i من السلسلة.
- حشر عنصر في مكان معيّن i، بحيث تأخذ العناصر ai, ..., an الأرقام i+1, ... , n+1 بالترتيب.
- حذف العنصر رقم i من السلسلة حيث $1 \le i \le n$. تسبّب عملية الحذف إعادة ترقيم العناصر ai+1, ..., an بالأرقام i, i+1, ..., n-1 بالترتيب.

لتمثيل سلسلة يمكن اعتماد إحدى الطريقتين التاليتين:

- تمثيل باستعمال الأشعة
- تمثيل باستعمال المؤشرات

سندرس فيما يلي كلاً من هاتين الطريقتين مبيّنين مزايا كل منهما ومساوئها.

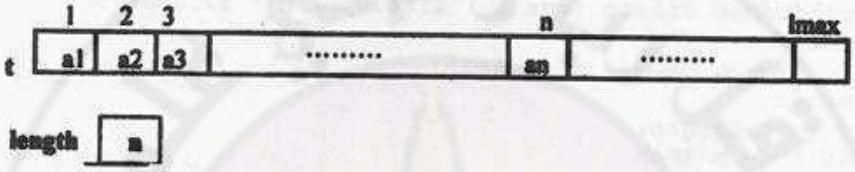
6-3-1 تمثيل السلاسل باستعمال الأشعة

تمثّل السلسلة بشعاع بحيث تحوي الخانة رقم i العنصر رقم i من السلسلة. من الواضح أن طول السلسلة لا يمكن أن يتجاوز طول الشعاع، ولكي نتعامل مع عناصر السلسلة فقط، وليس مع كل مكوّنات الشعاع لابد من معرفة طول السلسلة. تمثّل السلسلة بالتسجيلة: (شعاع من العناصر ذات النمط Element، عدد صحيح يحوي طول السلسلة) كما يُظهر الشكل (6-1).

```
const
    lmax = 100;
type
    ListType  = record
                    t : array [1..lmax] of Element;
                    length :  0..lmax
                end;
```

ليكن L متحوّلاً من نمط ListType فإن:

- L فارغة ⇔ L.length = 0
- L.t [i] يمثّل العنصر ذا الرقم i من السلسلة L.



الشكل (6-1): تمثيل سلسلة خطية باستخدام شعاع

تُجرى العمليات التالية على سلسلة خطية:

أ- إضافة عنصر x في الموقع k من السلسلة الخطية

إذا كان الموقع k ضمن أدلة السلسلة، نزيح العناصر الموجودة في المواقع التي تلي k موقعاً واحداً إلى الأمام ، نضع x في الموقع k ونزيد طول السلسلة 1.

```
procedure insert (var L: ListType; x: Element ; k :
1..lmax;)
var
    n: 0..lmax;
    i: 1..lmax;
begin
    n := L.length;
    if (n <= lmax) and (k <= n + 1) then
        begin
            for i := n downto k do
                L.t[i + 1] := L.t[i];
            L.t[k] := x;
            L.length:= n +1
        end
end;
```

ب- حذف عنصر x من الموقع k في سلسلة خطية

إذا كان الموقع k ضمن أدلة السلسلة، نزيحالعناصر الموجودة في المواقع التي تلي k موقعاً واحداً إلى الخلف وننقص طول السلسلة 1.

```
procedure delete (var L : ListType; k : 1..lmax);
var
    n: 0..lmax; i: 1..lmax;
begin
n := L.length;
if k <= n then
    begin
        for i := k to n - 1 do
            L.t[i] := L.t[i + 1]
        L.length:= L.length- 1
   end
end;
```

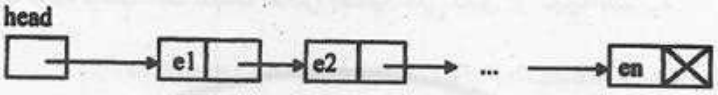
نلاحظ أن عملية حذف عنصر تتطلّب في أسوأ الأحوال n-1 عملية نسب عناصر من النمط Element بسبب الانزياح الواجب القيام به بعد عملية الحذف، وأن عملية الإضافة تتطلّب في أسوأ الأحوال n+1 عملية نسب (أيضاً بسبب الانزياح)، حيث n طول السلسلة.

2-3- تمثيل السلاسل باستعمال المؤشرات

تُستعمل المؤشرات في هذه الحالة لربط العناصر المتتالية في السلسلة، وتتحدّد السلسلة بمؤشر لأول عنصر فيها كما في الشكل (6-2). يمكن تمثيل سلسلة بلغة Pascal كما يلي:

```
Type
    ListP = ^ ListElem;
    ListElem = Record
                  val  :  Element;
                  link : ListP
```

```
End;
```



الشكل (6-2): تمثيل سلسلة خطية باستخدام المؤشرات

ومع أن هذه الطريقة في التمثيل تحتاج إلى ذاكرة إضافية لتخزين المؤشرات، فإنها تسمح بتجاوز مشكلة الطول الأعظمي للسلسلة، وحلّ مشكلة الانزياحات أثناء الإضافة أو الحذف. يمكّن هذا التمثيل من إجراء معظم العمليات على السلاسل بسهولة وسرعة، ولكن بعض العمليات يتطلّب معالجة خاصّة قد تحتاج زمنّ تنفيذ طويلاً نسبياً.

إذا كان L متحوّلاً من نمط ListP فإن:

- L = nil ⇔ السلسلة L فارغة.
- L^.val يمثّل أول عنصر من السلسلة.
- إذا كان P مؤشراً إلى أحد عناصر السلسلة فإن P^.link يؤشر إلى العنصر التالي في السلسلة.

سنعرض فيما يلي إجرائيات التعامل مع السلاسل الممثّلة باستعمال المؤشرات، ونرى من الضروري الإشارة إلى أهمّية توخّي الدقّة عند كتابة البرامج التي تستعمل المؤشرات، إذ تعتبر الإجرائيات التي تتعامل مع المؤشرات من أهم مصادر الأخطاء الشائعة في البرامج. لذا سيلاحظ القارىء استعمالنا المتكرّر لمتحوّلات موضعية في الإجرائيات، لتجنّب ضياع قسم من السلسلة أثناء عملية التجوّل ضمنها. وفيما يلي بعض العمليات على السلسلة الخطية الممثّلة بواسطة المؤشرات:

آ- حساب طول سلسلة خطية

نعرّف عدّاداً نزيده واحداً كلما مسحنا عنصراً إضافياً حتى نصل إلى نهاية السلسلة.

```
function length(head : ListP) : integer;
var
    n : integer;
    P : ListP;
begin
    n := 0;
    P := head;
    while P <> nil do
        begin
            n := n + 1;
            P := P^.link;
        end;
    length := n
end;
```

ب- إعادة عنصر من الموقع k في السلسلة

نمسح السلسلة حتى نصل إلى الموقع المطلوب إذا كان ضمن أدلة السلسلة.

```
procedure ithElem (head : ListP; k: integer; var x:
Element; var error: boolean);
var
    i: integer;
    P: ListP;
begin
    P := head ; error := false;
    i := 1;
    while i < k and not(error) do
        if P = nil then
            error := true;
        else
            begin
                P := P^.link;
                i := i +1
            end;
     if P = nil then
         error := true
     else
```

```
            x := P^.val
end;
```

جـ- حذف عنصر x من الموقع k في سلسلة خطية

نمسح السلسلة حتى نصل إلى العنصر الذي يسبق العنصر المطلوب ؛ نجعل هذا العنصر يؤشر إلى العنصر الذي يلي العنصر المطلوب، ونستعمل لهذه الغاية مؤشرين أحدهما يسبق الآخر بعنصر ؛ أخيراً نحرر الذاكرة المحجوزة للعنصر المطلوب.

```
procedure  Delete   (var head : ListP; k : integer; var
error: boolean);
var
    i: integer;
    P, PP: ListP;
begin
    error := false;
    if head = nil then
          error := true;
    if (k = 1) and not(error) then
        begin
            head := head^.link;
            error := false
        end
    else
        begin
            P := head;
            i := 1;
            while i < k  and not(error) do
                if P = nil then
                    error := true
                else
                    begin
                        PP := P;
                        P := P^.link;
                        i := i+1
                    end;
            if not (error) then
                begin
                    PP^.link := P^.link;
                    dispose(P)
                end
```

```
        end
end;
```

د- إضافة عنصر x في الموقع k من سلسلة خطية

نمسح السلسلة حتى نصل إلى العنصر الذي يسبق موقعه موقع العنصر الذي نريد إضافته ، نولّد عنصراً جديداً ونجعله يؤشر إلى العنصر الموجود في الموقع الذي يلي موقع العنصر الذي نريد إضافته ، أخيراً نجعل العنصر السابق يؤشر إلى العنصر ونستعمل لهذه الغاية مؤشرين أحدهما يسبق الآخر بعنصر.

```
procedure insert (var head : ListP; k: integer; x:
Element; var error: boolean);
var
    i: integer;
    C, P, PP: ListP;
begin
    if k = 1 then
        begin
            new(C);
            C^.val := x;
            C^.link := head;
            head := C;
            error := true
        end
    else
        begin
            P := head;
            error := false;
            i := 1;
            while i < k and not(error) do
                if P = nil then
                    error := true
                else
                    begin
                        PP := P;
                        P := P^.link;
                        i := i+1
                    end;
            if not error then
```

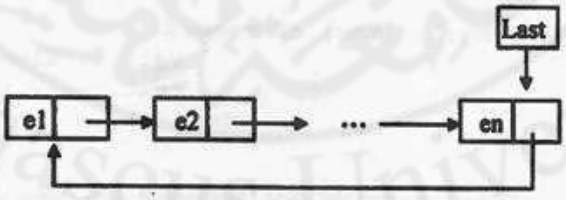
```
                begin
                    new(C);
                    C^.val := x;
                    C^.link := P;
                    PP^.link := C
                end
            end
end;
```

يمكن أن نلاحظ أن كل هذه العمليات تتطلّب في أسوأ الأحوال عدداً من عمليات النسب لمؤشرات (ذات الحجم الصغير) من مرتبة n حيث n هو عدد عناصر السلسلة. وتسمح هذه الطريقة في تمثيل السلاسل، بإضافة عنصر من وسط السلسلة أو حذفه، دون الحاجة إلى إزاحة عناصرها، كما تسمح بوصل سلسلتين دون الحاجة لنسخ عناصر

يمكن إجراء بعض التعديلات على تمثيل السلاسل باستخدام المؤشرات، وذلك بهدف تحسين أداء بعض خوارزميات التعامل مع السلاسل:

- السلسلة الدائرية

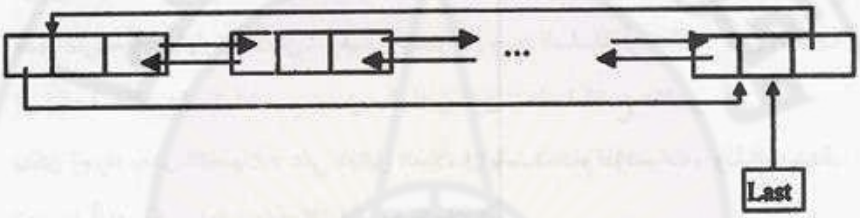
بدلاً من أن يأخذ المؤشر في آخر عنصر في السلسلة القيمة nil نجعل هذا المؤشر يؤشر إلى أول عنصر في السلسلة، ونحدّد السلسلة بمؤشر إلى آخر عنصر فيها كما في الشكل (6-3).



الشكل(6-3): سلسلة خطية دائرية

- **السلاسل المضاعفة الارتباط**

يمكّن التمثيل الذي قدّمناه من مسح السلسلة في اتجاه واحد. في الكثير من التطبيقات نحتاج إلى مسح عناصر السلسلة ابتداءً من آخر عنصر، وانتهاءً بأول عنصر فيها، لذلك نضيف من أجل كل عنصر مؤشراً إضافيا يدلّنا على العنصر السابق في السلسلة ؛ فنحصل بذلك على سلسلة مضاعفة الارتباط، يوضّح الشكل (6-4) سلسلة دائرية مضاعفة الارتباط.



الشكل (6-4): سلسلة خطية دائرية مضاعفة الارتباط

ملاحظة

في بعض لغات البرمجة التي لا تحوي النمط "مؤشر"، يمكن محاكاة مفهوم المؤشر باستعمال مصفوفة واستعمال دليل عوضا عن المؤشر. يبين النمط التالي كيف يمكن تعريف منطقة من الذاكرة لتمثيل السلاسل:

```
type
   Mem = Array[1..lmax] of record
                             val   : Element;
                             next  : 1..lmax
                           end;
```

يمكن إذن تمثيل سلسلة بعدد صحيح يمثل دليل أول عنصر في السلسلة.

إن اختيار حجم كبير للمصفوفة السابقة يسمح بتمثيل عدد من السلاسل في المصفوفة نفسها. في الشكل (6-5) جرى تخزين سلسلتين:

L1 = (e1, e2, e3)
L2 = (e4, e5)

لنتمكن من إدارة الأماكن الفارغة استعملنا السلسلة Empty بحيث تؤدي إضافة عنصر إلى إحدى السلسلتين، إلى حذف أول عنصر من السلسلة Empty.

L1	4	1		7
		2	e3	0
		3	e4	8
L2	3	4	e1	6
		5		0
Empty	1	6	e2	2
		7		5
		8	e5	0

الشكل (6–5): تمثيل سلسلتين خطيتين في جدول واحد

6–4 المكدسات

يعتبر المكدس (Stack) من بنى المعطيات الهامة المستخدمة بكثرة في مجالات البرمجة والاستثمار المختلفة (تذكر استخدامنا للمكدس في تحويل الخوارزميات العوديّة إلى تكرارية). المكدس هو بنية خطية تشبه السلسلة، غير أن عمليات الإضافة والحذف تتم من جهة واحدة ندعوها قمة المكدس كما يُظهر الشكل (6–6).

العمليات على المكدس

- إنشاء مكدس فارغ (EmptyStack)
- إضافة عنصر إلى مكدس (Push)
- حذف عنصر من مكدس (Pop)
- تفريغ المكدس (CleanUp)
- اختبار كون المكدس فارغاً (IsEmpty)

- قراءة العنصر الموجود في قمة المكدس (Top)

إذا كان P مكدسا فإن:

- العمليات Pop(P) و Top(P) تكون معرّفة إذا وفقط إذا كان IsEmpty(P) = false
- Pop(Push(P,e)) = P
  Top(Push(P,e)) = e
- IsEmpty(EmptyStack) = true
  IsEmpty(Push(P, e)) = false

الشكل (6-6): المكدس

تشكل الموضوعات الثلاثة السابقة وصفاً كاملاً لكل العمليات على المكدس (راجع تعريف النمط المجرد للمكدس في الفقرة 5-2-1). يمكن تمثيل المكدس بأكثر من طريقة نعرض منها الطريقتين التاليتين:

6-4-1 تمثيل المكدس باستعمال الأشعة

تمثّل السلسلة بالتسجيلة: (شعاع من العناصر ذات النمط Element، عدد صحيح يحوي دليل قمة المكدس).

```
type
    Stack = Record
                top   : integer;
                Elems : Array[1..lmax] of Element
            end;
```

نستطيع في هذا التمثيل كتابة العمليات التالية على المكدسات:

آ- تأهيل المكدس

```
procedure EmptyStack (var P : Stack);
begin
    P.top := 0;
end;
```

ب- إضافة عنصر إلى قمة المكدس

```
procedure Push (var P : Stack; x : Element);
begin
    if P.top = lmax then
        writeln ( 'error : Stack is full ')
    else
        begin
            P.top := P.top + 1;
            P.Elems[P.top] := x
        end
end;
```

جـ- حذف عنصر من قمة المكدس

```
procedure Pop (var P : Stack);
begin
   if IsEmpty(P) then
        writeln ('error : Stack is empty ')
   else
        P.top := P.top - 1;
end;
```

د- إعادة عنصر القمة في مكدس

```
procedure Top (P : Stack ; var x : Element);
begin
    if IsEmpty(P) then
        writeln ('error : Stack is empty ')
    else
        x := P.Elems[P.top]
end;
```

د- اختبار كون المكدس فارغاً

```
function IsEmpty(P : Stack) : boolean;
begin
    IsEmpty := (P.top = 0)
end;
```

6-4-2 تمثيل المكدس باستعمال مؤشرات

يمثل المكدس في سلسلة خطية بحيث يؤشر إلى بدايتها مؤشر بداية المكدس كما يُظهر الشكل (6-7). وتصبح بنى المعطيات اللازمة لتعريف المكدس كما يلي:

```
type
    Stack = ^ Elem;
    Elem  = Record
              val  :  Element;
              link : Stack
            end;
```

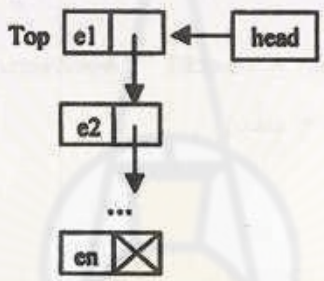
وتصبح إجرائيات التعامل مع المكدس كالتالي:

آ- تأهيل المكدس

```
procedure EmptyStack (var P : Stack);
begin
      P := nil
end;
```

ب- إضافة عنصر إلى قمة المكدس

```
procedure Push (var P : Stack; x : Element);
var
    E : Stack;
begin
    new(E);
    E^.val := x;
    E^.link := P;
    P := E
end;
```



الشكل (6-7): تمثيل مكدس بواسطة سلسلة خطية

جـ- حذف عنصر من قمة المكدس

```
procedure Pop (var P : Stack);
var
    E : Stack;
begin
    E := P;
    if IsEmpty(P) then
        writeln('error. Stack is empty ')
    else
        begin
            P := P^.link;
            dispose(E)
        end
end;
```

د- إعادة عنصر القمة في مكدس

```
procedure top (P : Stack ; var x : Element);
begin
    if IsEmpty(P) then
        writeln ('error. Stack is empty ')
    else
        x := P^.val
end;
```

د- اختبار كون المكدس فارغاً

```
function IsEmpty (P : Stack) : boolean;
begin
    IsEmpty := (P = nil)
end;
```

هـ - تفريغ المكدس

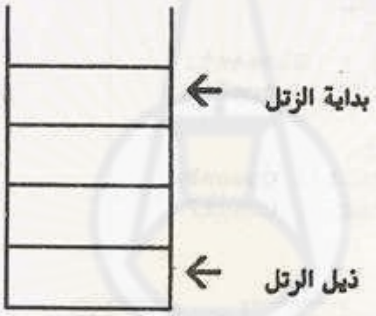
```
procedure CleanUp (var P : Stack);
begin
    if P = nil then
        Writeln ('Stack is already cleaned up');
    while p <> nil do
        pop(P)
end;
```

6-5 الأرتال

من بنى المعطيات الهامة أيضاً الرتل (Queue) وهو بنية خطية تشبه السلسلة غير أن عمليات الإضافة تجري في جهة ندعوها ذيل الرتل. ويجري الحذف والحذف في الجهة المعاكسة التي نسميها بداية الرتل كما يُظهر الشكل (6-8).

العمليات على الرتل

- إنشاء رتل فارغ (EmptyQueue)
- إضافة عنصر إلى رتل (Enqueue)
- حذف عنصر من رتل (Dequeue)
- تفريغ الرتل (CeanUp)
- اختبار كون الرتل فارغا (IsEmpty)
- قراءة العنصر الموجود في بداية الرتل (Top)



الشكل (6-8): الرتل

إذا كان Q رتلا فإن:

- العمليات Enqueue(Q) و Dequeue(Q) تكون معرفة إذا وفقط إذا كان IsEmpty(Q) = false
- IsEmpty(EmptyQueue) = true
  IsEmpty(Enqueue(Q, e)) = false

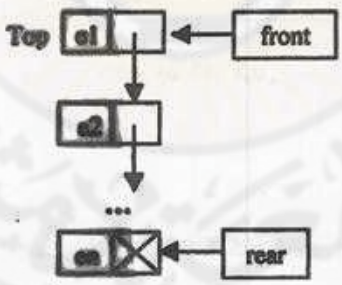
تكون الموضوعات السابقة وصفا كاملا لكل العمليات على المكدس (راجع تعريف النمط المجرد للرتل في آخر هذه الفقرة). يمكن تمثيل الرتل بأكثر من طريقة نعرض منها فقط

طريقة تمثيله بواسطة المؤشرات (ويترك للقارئ تمثيله بواسطة الأشعة وكتابة العمليات على الأرتال في هذه الطريقة):

6-5-1 تمثيل الرتل باستعمال مؤشرات

يمثلّ المكدس في سلسلة خطية بحيث يؤشر إلى بدايتها مؤشر بداية الرتل، ويؤشر إلى نهايتها مؤشر ذيل الرتل كما يُظهر الشكل (6-9). وتصبح بنى المعطيات اللازمة لتعريف الرتل كما يلي:

```
type
    QueueP = ^ Elem;
    Elem   = Record
               val :  Element;
               link : QueueP
             end;
    Queue = record
              front : QueueP;
              rear  : QueueP
            end;
```



الشكل (6-9): تمثيل رتل بواسطة سلسلة خطية

وتصبح إجرائيات التعامل مع الرتل كالتالي:

آ- تأهيل الرتل

```
procedure Make_empty (var Q : Queue);
begin
    Q.front := nil;
    Q.rear := nil
end;
```

ب- إضافة عنصر إلى ذيل الرتل

```
procedure Enqueue (var Q : Queue; x : Element);
var
    E : QueueP;
begin
    new(E);
    E^.val  :=x;
    E^.link := nil;
    if Q.front = nil then
        begin
            Q.front := E;
            Q.rear := E
        end
    else
        begin
            Q.rear^.link := E;
            Q.rear^.link := E
        end
end;
```

جـ- حذف عنصر من بداية الرتل

```
procedure Dequeue (var Q : Queue);
var
    E : QueueP;
begin
    if IsEmpty(Q) then
        writeln ('error. Queue is empty')
   else
        E := Q.front;
        if Q.front^.link = nil then
            begin
```

```
                Q.front := nil;
                Q.rear := nil
            end
        else
            begin
                Q.front := Q.front^.link;
                dispose(E)
            end
end;
```

د- إعادة عنصر بداية الرتل

```
procedure top (Q : Queue ; var x : Element);
begin
   if IsEmpty(Q) then
        writeln ('error. Queue is empty')
   else
       x := Q.front^.val
end;
```

د- اختبار كون الرتل فارغاً

```
function IsEmpty (Q : Queue) : boolean;
begin
    IsEmpty := (Q.front = Q.rear = nil)
end;
```

هـ- تفريغ الرتل

```
procedure CleanUp (var Q : Queue);
begin
    if Q.front = nil then
        Writeln ('Queue is already cleaned up');
    while Q.front <> nil do
        Dequeue(Q)
end;
```

6-5-2 النمط المجرد للرتل

نستطيع تلخيص العمليات السابقة على الرتل وشروطها، بكتابة النمط المجرد للرتل (q هو متحول أو نسخة Instance من نمط Queue(T) ، n هو متحول من نمط T):

```
Type        Queue(T)
Kind        Queue
Import      boolean
operator
    Constructor
        EmptyQueue : ›  Queue(T)
    Access
        IsEmpty : Queue(T) ›  boolean
        Top : Queue(T) ›  T
    Transformer
        Enqueue : Queue (T) × T ›  Queue(T)
        Dequeue : Queue(T) ›  Queue(T)
    Precondition
            (1) precond(Dequeue(q)) = = not IsEmpty (q)
            (2) precond (Top(q)) = = not IsEmpty(q)
    Axiom
            (1) IsEmpty(EmptyQueue) = = True
            (2) IsEmpty(Enqueu (p, n)) = = false
```

6-6 تمارين

1- اكتب الإجرائية التالية:

```
procedure Swap2Elem (p1, p2 : Pelem ; var head : Pelem);
```

التي تبدل مواقع عنصرين من نمط elem والمؤشر إليهما بالمؤشرين p1, p2 ضمن السلسلة الخطية التي يؤشر على بدايتها المؤشر Head (بدون خلق عناصر إضافية ومناقشة جميع الحالات الممكنة).

2- اكتب النمط المجرد للسلسلة الخطية.

3- نريد إعادة كتابة العمليات على السلسلة الخطية حيث نفترض أن النمط Element هو integer والسلسلة مرتبة تصاعديا حسب الحقل val.

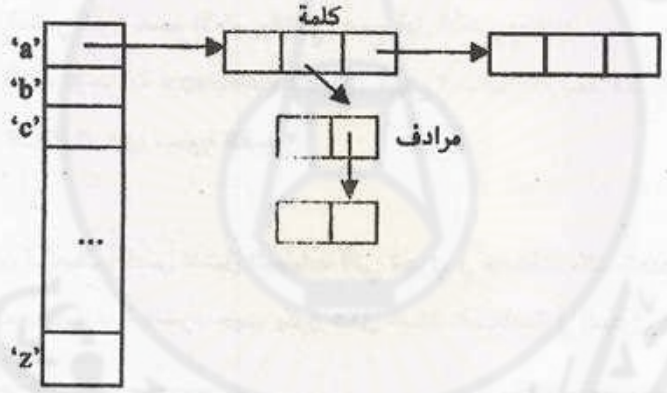
4- استخدام السلاسل الخطية لبناء مكنز (Thesaurus)

نريد بناء مكنز، يحوي مجموعة من الكلمات مع مرادفاتها، بحيث تكون الكلمات والمرادفات مرتبة أبجديا. لتحقيق ذلك نفترض أنه لدينا شعاع من المؤشرات مرقم حسب الأحرف الأبجدية. كل مؤشر يؤشر إلى سلسلة خطية من الكلمات التي تبدأ بنفس الحرف الأبجدي. كل كلمة بدورها تحوي مؤشرا آخر إلى سلسلة خطية تحوي مرادفات هذه الكلمة مرتبة أبجديا كما يوضح الشكل.

أ. وضح بنى المعطيات المستخدمة.

ب. اكتب خوارزميات تحقق العمليات التالية:

- إضافة كلمة إلى المكنز في مكانها الصحيح وفق الترتيب الأبجدي
- إضافة مرادف لكلمة موجودة سلفا في مكانها الصحيح.
- حذف كلمة.
- حذف مرادف.
- البحث عن كلمة.
- البحث عن مرادف.
- طباعة مرادفات كلمة معينة.
- طباعة محتويات المكنز بشكل مرتب.



ت. هل تعتقد أن هذا التمثيل هو الأفضل (لاحظ تكرار الكلمات)، هل بإمكانك اقتراح تمثيلات أخرى لبنى المعطيات بحيث تستطيع تجنب مشكلة التكرار !

5- سنعرف بنية معطيات جديدة نسميها Deque وهي سلسلة خطية عادية نؤشر على بدايتها بالمؤشر Left وعلى نهايتها بالمؤشر Right. يمكن إضافة عناصر إلى السلسلة في البداية والنهاية فقط، كما يمكن حذف العناصر في البداية والنهاية فقط.

أ. اقترح بنى المعطيات المناسبة (دعم شرحك برسم توضيحي).

ب. اكتب الإجرائية RemoveLeft لحذف عنصر من بداية السلسلة.

ت. اكتب الإجرائية RemoveRight لحذف عنصر من نهاية السلسلة.

ث. اكتب الإجرائية InsertLeft لإضافة عنصر في بداية السلسلة.

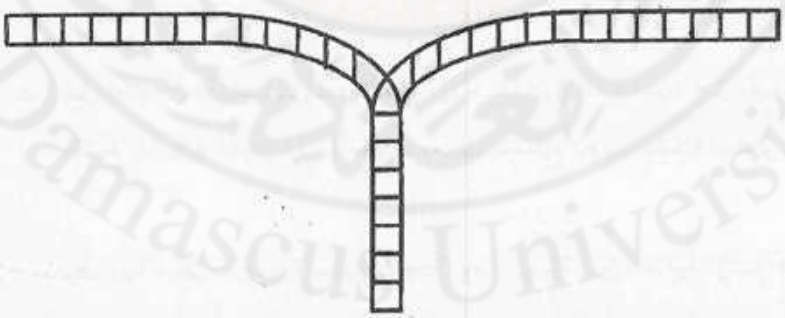
ج. اكتب الإجرائية InsertRight لإضافة عنصر في نهاية السلسلة.

ح. وضح مدى إمكان استخدام هذه البنية الجديدة للاستعاضة عن بنية المكدس (Stack) وبنية الرتل (Queue) معا.

خ. اكتب الإجرائية InverseDeque لعكس ترتيب العناصر في الـ Deque (بمعنى أن العنصر الأول يصبح الأخير والثاني يصبح قبل الأخير وهكذا).

د. اكتب الإجرائية PlainDeque لنتحقق: أيبقى الـ Deque بعد عكسه (بطريقة السؤال السابق) مساويا لنفسه ؟.

6- يمكن استعمال المكدس لتمثيل العمليات التي تجري في محطة السكك الحديدية عند تبديل العربات المكونة لقطار، حيث يكون شكل السكة المستخدمة في التبديل كالتالي:



تصل العربات من اليمين وتدخل الى المكدس ويمكن إخراجها في أية لحظة. مثلا إذا كانت n = 3 فإن العمليات:

إدخال العربة رقم 1 ؛ إدخال العربة رقم 2 ؛ إدخال العربة رقم 3 ؛ إخراج العربة رقم 3 ؛ إخراج العربة رقم 2 ؛ إخراج العربة رقم 1

تعطي الترتيب 3،2،1 للعربات.

أ. من أجل n=3 و n=4 ما هي التباديل التي يمكن الحصول عليها وكيف ؟ هل هناك تباديل لا يمكن الحصول عليها ؟

ب. يمكن تمثيل عملية إدخال عربة بالحرف E، وعملية إخراج عربة بالحرف D تكافئ متتالية الأحرف EEEDDD العمليات اللازمة للحصول على الترتيب 3،2،1 ويمكن التحقق بسهولة أنه ليست كل متتالية من الأحرف E, D تمثل متتالية من العمليات. فالمتتالية EDDEED مثلا ليست صحيحة، والسبب أنه في لحظة معينة يجب حذف عنصر من مكدس فارغ. نقول عن متتالية من الأحرف E وD إنها مقبولة إذا كانت تمثل مجموعة عمليات ممكنة التنفيذ على مكدس. أوجد الشرط اللازم والكافي لتكون متتالية مقبولة.

7- اكتب إجرائيات غير عودية باستخدام المكدسات للقيام بكل عمليات التحويل الممكنة بين التعابير النظامية (Infix Expressions) والتعابير المصدرة (Prefix Expressions) والتعابير الملحقة (Postfix Expressions).

8- اكتب إجرائية لحساب قيمة تعبير مكتوب بالشكل النظامي (أعد السؤال نفسه بالنسبة للشكل المصدر والملحق).

وقل رب زدني علما
Damascus University

الفصل السابع

البنى الشجرية

7-1 مقدمة

درسنا في الفصل السابق بنى المعطيات الخطيّة أو التسلسلية، وسنركّز الاهتمام في هذا الفصل وفي الفصل التالي، في بنى المعطيات غير التسلسلية (Non Sequential Structures)، التي يمكن تعريفها كما يلي:

هي البنى التي لا توجد علاقة ترتيب كلّي بين عناصرها (قد توجد علاقة ترتيب جزئي بين عناصر بعض البنى غير التسلسلية).

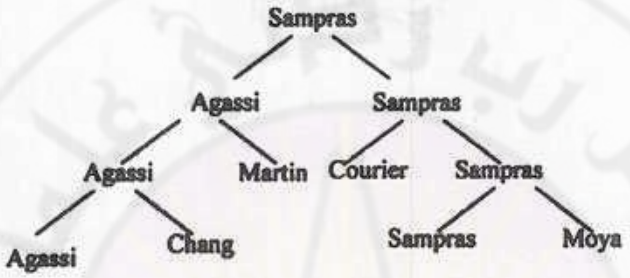
سندرس نوعين هامين من البنى غير التسلسلية: البنى الشجرية (Tree Structures) في هذا الفصل، والبيانات (Graphs) في الفصل التالي.

نعرّف *الشجرة* (Tree) بأنها مجموعة من العناصر نسميها عُقداً (Nodes)، ومرتبطة بعضها ببعض *بروابط* (Links)، ومنظّمة تنظيماً هرمياً (Hierarchical) لا يحوي حلقات (Cycles) مغلقة، أيّ إنّه توجد عقدة مميّزة ووحيدة نسميها *الجذر* (Root)، يمكن بواسطتها الوصول إلى مجموعة من العقد، ومن هذه العقد إلى عقد أخرى وهكذا.

تعتبر البنى الشجرية من البنى المميّزة في مجال المعلوماتية، نظراً لكثرة استعمالها في نظم الاستثمار خاصة، حيث تُنظّم الملفّات في مجلّدات يحوي كل منها مجموعة من المجلّدات الجزئية و مجموعة من الملفّات.

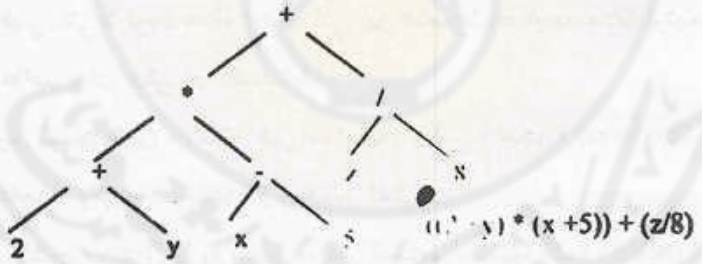
أمثلة على الأشجار

- نتيجة دوري كرة مضرب:



الشكل (7-1): تمثيل دوري كرة المضرب باستخدام شجرة ثنائية

- تمثيل عبارة حسابية:



الشكل (7-2): تمثيل عبارة حسابية باستخدام شجرة

نستعمل في التطبيقات المعلوماتية أنواع عديدة من الشجرات، ندرس منها في هذا الكتاب نوعين هامين: الشجرات الثنائية (Binary Trees)، والشجرات المعممة (Generalized Trees).

7-2 الأشجار الثنائية

وهي حالة خاصة من الأشجار يكون فيها لكل عقدة ابنان على الأكثر.

7-2-1 تعاريف

تعريف 1

نسمي شجرة ثنائية كل بنية B تحقق أحد الشرطين التاليين:

- $B = \Phi$ (الشجرة الفارغة).
- $B = < O, B1, B2 >$ حيث O عقدة نسميها الجذر. B1, B2 شجرتان ثنائيتان منفصلتان. نسمي B1 الشجرة الجزئية اليسارية وB2 الشجرة الجزئية اليمينية.

تعريف 2

نسمي جذر الشجرة الجزئية B1 الابن الأيسر لـ O، وجذر الشجرة الجزئية B2 الابن الأيمن لـ O، ونقول إنه يوجد اتصال يساري بين الجذر والابن الأيسر واتصال يميني بين الجذر والابن الأيمن.

تعريف 3

يمكن أن توجد بين العقد العلاقات التالية: أب، ابن، أخ، سَلَف، حفيد. نقول عن عقدتين إنهما أختان، إذا كان لهما الأب نفسه.

نقول عن عقدة n_i إنها سلف (Ancestor) لعقدة n_j، إذا وفقط إذا كانت n_i أباً لـ n_j، أو كانت n_i سلفاً لأب n_j.

نقول عن عقدة n_i إنها حفيد (Grandchild) لعقدة n_j، إذا وفقط إذا كانت n_i ابناً لـ n_j، أو كانت n_i حفيداً لابن n_j.

تعريف 4

لكل عقدة في شجرة ثنائية ولدان على الأكثر:

- نسمي العُقد التي لها ولد على الأقل عقداً داخلية (Internal Nodes)، أو اختصاراً عقداً.
- نسمي العُقد التي ليس لها أولاد عقداً خارجية (External Nodes)، أو أوراق (Leafs).

تعريف 5

نسمي *طريقاً* (Path) ضمن شجرة ثنائية كل متتالية من العقد، ونسمي فرعاً في الشجرة كل طريق يصل بين الجذر وإحدى الأوراق.

7-2-2 بعض القياسات على الأشجار الثنائية

نبيّن فيما يلي بعض العمليات على الأشجار الثنائية التي تسمح بحساب بعض محدّدات هذه الأشجار، وتفيد في حساب درجة تعقيد الخوارزميات المطبّقة عليها.

نفترض فيما يلي أن B شجرة ثنائية وx عقدة في هذه الشجرة.

تعريف 6

تُعرّف *درجة الشجرة* D(B) بوجهٍ عام، بأنها العدد الأعظمي لأولاد عقدة. من البديهي أن درجة شجرة ثنائية هي 2.

تعريف 7

حجم الشجرة Vol(B) هو عدد عقدها و يمكن تعريفه بالشكل التالي:

Vol(EmptyTree) = 0
Vol(<O, B1, B2>) = 1 + Vol(B1) + Vol(B2)

تعريف 8

ارتفاع عقدة h(x) هو عدد الاتصالات في الطريق الواصل بين هذه العقدة والجذر. ونعرفها بالشكل العودي التالي:

إذا كانت x هي الجذر فإن h(x) = 0.

وإلا فإن h(x) = 1 + h (y) حيث y هو أبو x.

تعريف 9

ارتفاع شجرة H(B) هو أطول ارتفاع عقدة في الشجرة.

H(B) = max { h(x) : x is a node in B}

تعريف 10

عرض الشجرة (L(B)) هو العدد الأعظم للعقد في مستوى ما.

تعريف 11

مسافة التجول ضمن شجرة LC(B) هي مجموع ارتفاعات عقدها.

$$LC(B) = \sum_{x\text{-is-a-node-of-}B} h(x)$$

تعريف 12

مسافة التجول الخارجي ضمن شجرة LCE(B) هي مجموع ارتفاعات أوراقها.

$$LCE(B) = \sum_{x\text{-is-a-leaf-of-}B} h(x)$$

تعريف 13

مسافة التجول الداخلي ضمن شجرة LCI(B) هي مجموع ارتفاعات عقدها الداخلية.

$$LCI(B) = \sum_{x\text{-is-an-internal-node-of-}B} h(x)$$

مثال

لتكن B الشجرة الثنائية المبينة بالشكل التالي:

Root (B) = n1
Left Son (n1) = n2
Right Son (n1) = n4
Leafs = { n8, n10, n5, n9 }
Internal Nodes = { n1, n2, n3, n4, n6, n7 }
h(n1) = 0, h(n2) = h(n4) =1, h(n3) = h(n5) = h(n7) = 2
H(B) = 4
LC(B) = 22
LCI(B) = 9
LCE(B) = 13

7-2-3 بعض الأشجار الثنائية الخاصّة

1- نسمي *شجرة ثنائية خطّية* كل شجرة ثنائية يكون فيها لكل عقدة ولد واحد على الأكثر، كما يُظهر الشكل (7-3).

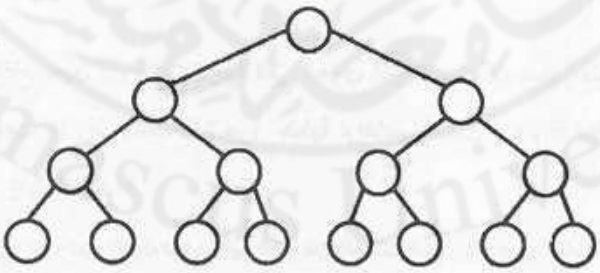
2- نسمي *شجرة ثنائية تامة* (Complete) كل شجرة تحوي عقدة واحدة في المستوى 0، وعقدتين في المستوى 1، وأربع عقد في المستوى 2، ... ، و2^k عقدة في المستوى رقم k، وتكون جميع مستويات مثل هذه الأشجار مملوءة تماما بالعقد، كما يظهر الشكل (7-4).

عدد العقد في شجرة تامة ارتفاعها h هو:

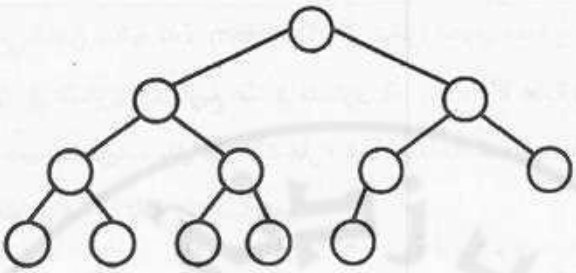
$$1 + 2 + 4 + \ldots + 2^h = 2^{h+1} - 1$$

3- نسمي *شجرة ثنائية كاملة* (Perfect) كل شجرة ثنائية تكون فيها جميع المستويات مملوءة تماما بالعقد، ما عدا المستوى الأخير الذي يمكن أن يحوي فراغات. وفي هذه الحالة تكون الفراغات دوما في جهة اليمين، كما يظهر الشكل (7-5).

الشكل (7-3): شجرة ثنائية خطية



الشكل (7-4): شجرة ثنائية تامة



الشكل (7-5): شجرة ثنائية كاملة

7-2-4 دراسة نظرية لمحددات الأشجار الثنائية

نستعرض في هذه الفقرة بعض النظريات الخاصة بالأشجار الثنائية، بُغية تَعرُّفِ منهجية التعامل مع الأشجار. يجب ألّا ننسى أنه بسبب الطبيعة الوديّة لتعريف الشجرة، ستكون معظم البراهين والخوارزميّات الخاصة بها وديّة.

نظرية 1

لتكن B شجرة ثنائية حجمها n وارتفاعها H، عندئذٍ يكون:

$$\lfloor \text{Log}_2 n \rfloor \le H \le n-1$$

حيث $\lfloor x \rfloor$ هو أكبر عدد صحيح أصغر من x.

البرهان

في حالة ارتفاع معيّن H، فإن الشجرة التي تحوي أقل عدد ممكن من العقد هي شجرة خطّية ارتفاعها H. إذا كانت B شجرة خطّية ارتفاعها H فإن حجم B (عدد عقدها) هو $n = H+1$. إذن $H \le n-1$.

الشجرة التي ارتفاعها H وتحوي أكبر عدد من العقد هي الشجرة التامّة. ولكن حجم شجرة تامّة B ارتفاعها Hهو: $2^{H+1}-1$. إذن، في حالة كل شجرة ثنائية لدينا:

$$n < 2^{H+1} \Rightarrow \text{Log}_2 n < H+1 \Rightarrow \lfloor \text{Log}_2 n \rfloor \le H$$

نتيجة

كل شجرة ثنائية B غير فارغة ذات f ورقة، يكون ارتفاعها H أكبر أو يساوي $\lceil Log_2 f \rceil$.

البرهان

أولاً سنبرهن بالتدريج أنه في حالة كل شجرة ثنائية حجمها n وتحوي f ورقة يكون:

$$f \le \frac{n+1}{2}$$

وهذه الخاصة محققة إذا كان n = 1، إذ يكون f = 1.

لنفترض أن الخاصّة محققة في حالة كل شجرة ثنائية حجمها أقل من n.

لتكن B = <O , B1 , B2> شجرة ثنائية حجمها n، وليكن n1 حجم B1 وn2 حجم B2.

ليكن f1 عدد أوراق B1، وf2 عدد أوراق B2، فبسبب من n = 1 + n1 + n2، نستطيع أن نكتب:

$$f = f1 + f2 \le \frac{n1+1}{2} + \frac{n2+1}{2} = \frac{n1+n2+2}{2} = \frac{n+1}{2}$$

نعود إلى برهان الخاصّة ؛ ولأن التابع Log_2 متزايد على R+، يكون لدينا:

$$Log_2 f \le Log_2\left(\frac{n+1}{2}\right) \Rightarrow$$
$$Log_2 f \le Log_2(n+1) - 1 \Rightarrow$$
$$Log_2 f + 1 \le Log_2(n+1) \Rightarrow$$
$$\lceil Log_2 f \rceil + 1 \le \lceil Log_2(n+1) \rceil = \lfloor Log_2 n \rfloor + 1 \Rightarrow$$
$$\lceil Log_2 f \rceil \le H$$

تطبيق

في دوري أحد الألعاب الرياضية تُمثّل أوراق الشجرة اللاعبين (لعبة تنس) أو الفرق (كرة قدم). فإذا كان لدينا f لاعباً (أو فريقاً)، فإنه يلزم $\lceil Log_2 f \rceil$ مرحلة لإنهاء الدوري.

نظرية 2

لتكن B شجرة ثنائية حجمها n. يمكن حصر مسافة التجوّل ضمن B بالمتراجحة:

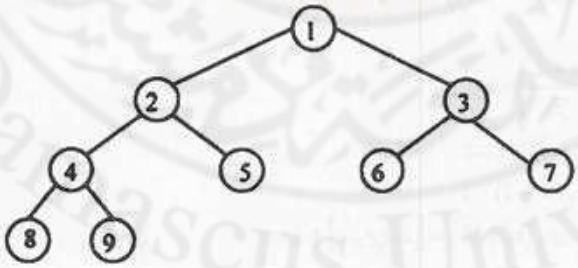
$$\sum_{k=1}^{n} \lfloor Log_2 k \rfloor \le LC(B) \le \frac{n(n-1)}{2}$$

البرهان

1. إذا كانت B خطية فإن مسافة التجول تكون أعظمية ويكون:

$$LC(B) = 1 + 2 + 3 + \ldots + n - 1 = \frac{n(n-1)}{2}$$

2. الشجرة الثنائية التي تجعل مسافة التجول أصغرية، هي التي تكون فيها كل المستويات مليئة ماعدا المستوى الأخير (أي الشجرة الكاملة). يمكن ترقيم العقد بالشكل التالي:



عندئذ يكون ارتفاع العقدة رقم k هو $\lfloor Log_2 k \rfloor$ (يمكن برهان هذه الخاصة بالتدريج على ارتفاع الشجرة). تكون مسافة التجول في هذه الحالة:

$$LC(B) = \sum_{k=1}^{n} \lfloor Log_2 k \rfloor$$

إذن:

$$\sum_{k=1}^{n} \lfloor Log_2 k \rfloor \le LC(B) \le \frac{n(n-1)}{2}$$

نتيجة

مسافة التجوّل في شجرة ثنائية حجمها n، تكون في أحسن الأحوال من مرتبة $n.Log_2 n$ وفي أسوأ الأحوال من مرتبة n^2.

البرهان

$$\sum_{k=1}^{n} \lfloor Log_2 k \rfloor \le \int_{1}^{n} Log_2 x dx = C\int_{1}^{n} \ln x dx = n.Log_2 n \Rightarrow$$

$$\sum_{k=1}^{n} \lfloor Log_2 k \rfloor \le n.Log_2 n$$

نظرية 3

لتكن B شجرة ثنائية تحوي f ورقة، يكون الارتفاع الوسطي للأوراق:

$PF(B) \ge \log_2 f$

نترك برهان هذه النظرية للقارئ كتمرين.

7-2-5 تمثيل الأشجار الثنائية

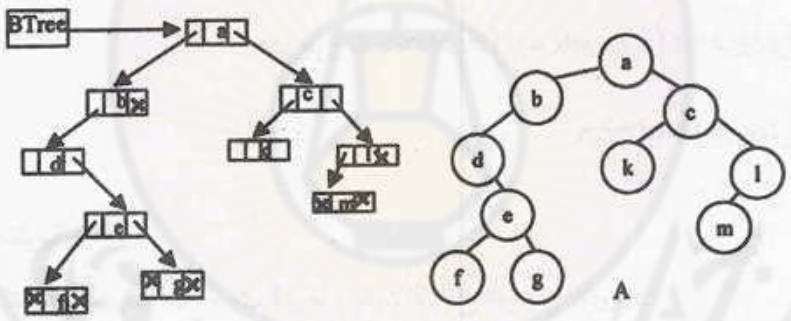
يمكن تمثيل الأشجار بطرق عدّة. لكل طريقة مزاياها التي تتعلق بالعمليات المطلوب القيام بها على الأشجار.

آ- تمثيل الأشجار باستعمال المؤشرات

نعرف العقدة بأنها تسجيلة تحوي إضافة إلى المعلومات المخزنة في العقدة، مؤشرين أحدهما يؤشر إلى الشجرة الجزئية اليسارية، والآخر إلى الشجرة الجزئية اليمينية:

```
type
    BTree = ^ Node;
    Node  = record
              val   : Element;
              L ,R  : BTree
            end;
```

يعطي الشكل (7-6) مثالا لشجرة ثنائية A ممثلة باستعمال المؤشرات.



الشكل (7-6): تمثيل شجرة ثنائية باستخدام المؤشرات

ب- تمثيل الأشجار الثنائية باستعمال المصفوفات

نعرف مصفوفة فيها ثلاثة أعمدة بحيث يحوي كل سطر فيها المعلومات المتعلقة بعقدة. هذه المعلومات هي: معلومات العقدة، رقم الابن اليساري، رقم الابن اليميني. يمكن اعتماد القاعدة التالية في ترقيم العقد: إذا كان n رقم العقدة، فإن 2n هو رقم الابن

اليساري، و2n+1 هو رقم الابن اليميني. لاحظ أنه باعتماد هذه الطريقة نستطيع استنتاج أن أبا عقدة رقمها m هو العقدة رقم m div 2.

```
const
    N = 100;
type
    BTable = array[1..N] of Record
                                val : Element;
                                L,R : 0..N
                            end;
    TabBTree = Record
                    root : 0..N;
                    tab: BTable
                end;
```

مثال

نمثل الشجرة A في الشكل السابق بواسطة المصفوفة المبينة في الشكل (7-7).

root
1

1	a	2	3
2	b	4	-
3	c	6	7
4	d	-	9
5	-	-	-
6	k	-	-
7	l	14	-
8	-	-	-
9	e	18	19
-	-	-	-
14	m	-	-
-	-	-	-
18	f	-	-
19	g	-	-

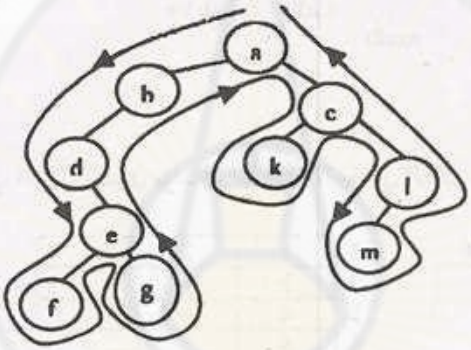
الشكل (7-7): تمثيل شجرة ثنائية باستخدام مصفوفة

7-2-6 التجول ضمن شجرة ثنائية

تهدف عملية التجول إلى المرور بكل عقدة من عقد الشجرة حسب ترتيب معين بهدف إجراء معالجة معينة عليها.

المبدأ العام

التجول في العمق مع الانعطاف يسارا عندما يكون ذلك ممكنا، كما يظهر الشكل (7-8).



الشكل(7-8): التجول في شجرة ثنائية

يحدث الالتقاء بكل عقدة ثلاث مرات:

- عند النزول:

- عند انتهاء التجول في الشجرة الجزئية اليسارية:

• عند انتهاء التجول في الشجرة الجزئية اليمينية:

خوارزمية التجول

نكتب الإجرائية العودية Traversal لإجراء التجول مع إمكان معالجة أي عقدة من الشجرة ثلاث مرات (بواسطة الإجرائيات: Process1 وProcess2 وProcess3) حسب جهة الوصول إليها حيث:

- 1 تعني النزول يسارا
- 2 تعني الصعود من اليسار
- 3 تعني الصعود من اليمين

```
procedure Traversal(A : BTree);
begin
    if A = EmptyTree then
        EndTraversal
    else
        begin
            Process1;
            Traversal(L(A));
            Process2;
            Traversal(R(A));
            Process3
        end
end;
```

حيث EndTraversal هي إجرائية الانتهاء من التجول.

آ- الشجرة ممثلة باستعمال المؤشرات

تصبح الخوارزمية على الشكل التالي:

```
procedure Traversal(A : BTree);
begin
    if A = nil then
        EndTraversal
    else
        begin
            Process1;
            Traversal(A^.L);
            Process2;
            Traversal(A^.R));
            Process3
        end
end;
```

ب- الشجرة ممثلة باستعمال مصفوفة

تصبح الخوارزمية على الشكل التالي:

```
Procedure Traversal (A : TabBTree);
Procedure Trav (i : 0..N)
begin
    if i = 0 then
        EndTraversal
    else
        begin
            Process1;
            Traversal(A.BTable[i].L);
            Process2;
            Traversal(A.BTable[i].R));
            Process3
        end

end;
begin
      Trav(A.root)
end;
```

حسب نوع المعالجات وترتيبها نحصل على طرق التجول التالية:

- التجول حسب الترتيب المصدر (Preorder): Process2 = Process3 = Φ
- التجول حسب الترتيب المتناظر (Inorder): Process1= Process3 = Φ
- التجول حسب الترتيب الملحق (Postorder): Process1 = Process2 = Φ

مثال

إذا أخذنا الشجرة المعروضة في الشكل (7-2)، التي تمثل عبارة حسابية، وأردنا طباعة العبارة الحسابية من الشجرة وفق الشكل النظامي (Inorder) مع إضافة أقواس إلى كل عملية (نفترض أن العمليات ثنائية المعامل)، نطبق خوارزمية التجول حسب الترتيب المتناظر، حيث: Process1= Process3 = Φ و process2 هي عمليا كتابة قيمة العقدة.

```
procedure PrintInorder (A : BTree);
begin
     if Leaf(A) then
        write(A^.val);
     else
         begin
              write('(');
              PrintInorder(A^.L);
              write(A^.val);
              PrintInorder(A^.R);
              write(')');
         end
end;
```

نسخة تكرارية لخوارزمية التجول

لإنشاء خوارزمية تكرارية مكافئة لخوارزمية التجول المودية، نعتمد المبدأ التالي:

- نستعمل مكدسا لتخزين العقد المؤجلة أو التي لم تنته معالجتها.
- ننزل إلى أعمق نقطة مع تخزين العقد التي نمر بها.
- عند تخزين عقدة نخزن علامة N تبين طريقة التلاقي مع عقدة.

```
procedure ItrTraversal (A : BTree);
var
    P : Stack;
    N : 1..3;
begin
    N := 1;
    P := nil;
    repeat
        if N = 1 then
            begin
                while A <> nil do
                    begin
                        Process1;
                        P:= Push(P,(A,1));
                        A := A^.L
                    end
            end
        EndTraversal;
        if not EmptyStack(P) then
            begin
                (A,N) := Top(P);
                Pop(P);
                if N = 1 then
                    begin
                        Process2;
                        P := Push(P,(A,2));
                        A := A^.R
                    end;
                else
                    Process3;
            end;
    until EmptyStack(P)
end;
```

3-7 الأشجار المعممة

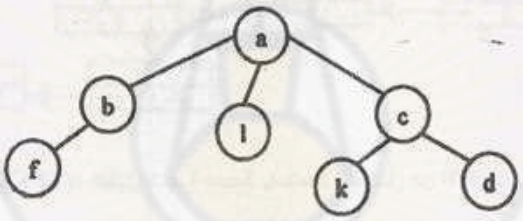
الشجرة المعممة هي شجرة يمكن فيها وجود عدة أبناء لكل عقدة. يمكن تمثيل الأشجار المعممة بطرق عدة، نعرض منها الطريقتين التاليتين:

7-3-1 تمثيل الشجرة باستعمال مصفوفة من المؤشرات

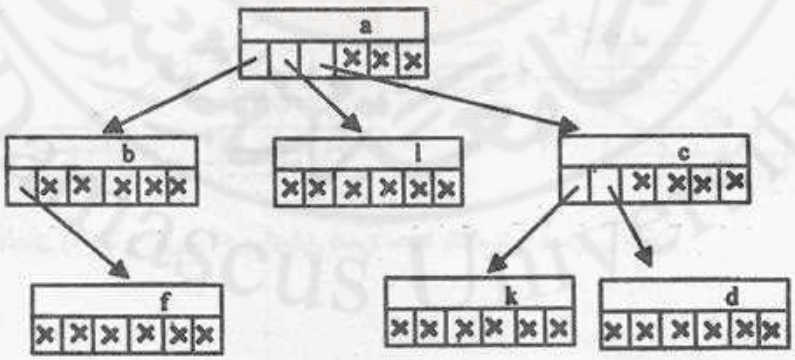
تحوي العقدة مصفوفة من المؤشرات للدلالة على الأولاد، ويعرف عدد أعظمي للمؤشرات (درجة الشجرة) التي تصدر عن كل عقدة. هناك هدر في الذاكرة المحجوزة، ثم إن هذا العدد الأعظمي للأولاد قد يكون غير كاف في بعض الحالات.

مثال

لتكن الشجرة المعممة المبينة في الشكل (7-9). إذا اعتبرنا أن درجة الشجرة (العدد الأعظمي للأولاد) هي 6، يمكن تمثيل الشجرة باستعمال مصفوفة من المؤشرات في الشكل (7-10).



الشكل (7-9): شجرة معممة

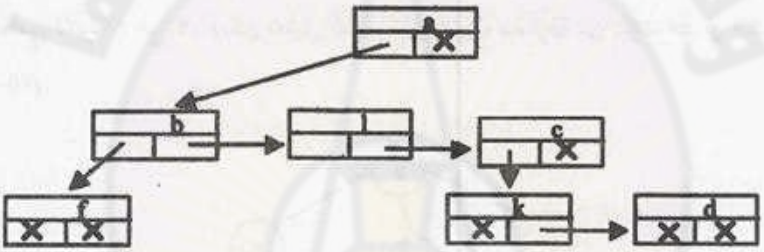


الشكل (7-10): تمثيل شجرة معممة باستخدام مصفوفة مؤشرات

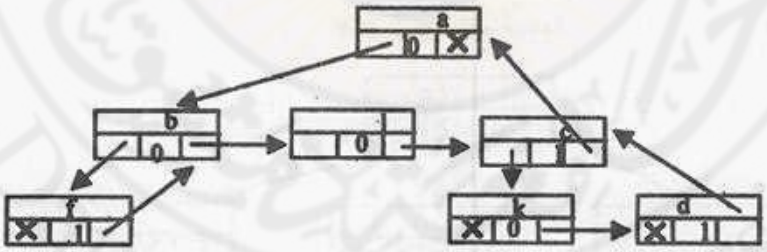
7-3-2 تمثيل الشجرة باستعمال سلاسل من الأولاد

تحوي العقدة مؤشرين يدل الأول على الولد الأول ويدل المؤشر الثاني على الأخ التالي. في هذه الحالة، لا يوجد ضياع في الذاكرة المحجوزة، ويمكن تخزين الشجرة مهما كان عدد الأولاد (كما في الشكل (7-11)).

يمكن تعديل هذه الطريقة في التمثيل بجعل آخر ولد يؤشر إلى الأب (كما في الشكل (7-12)). سنحتاج إذن إلى حقل إضافي لنحدد العقدة التالية: أهي أخ أو أب ؟.



الشكل (7-11): تمثيل شجرة معممة باستعمال سلاسل من الأولاد



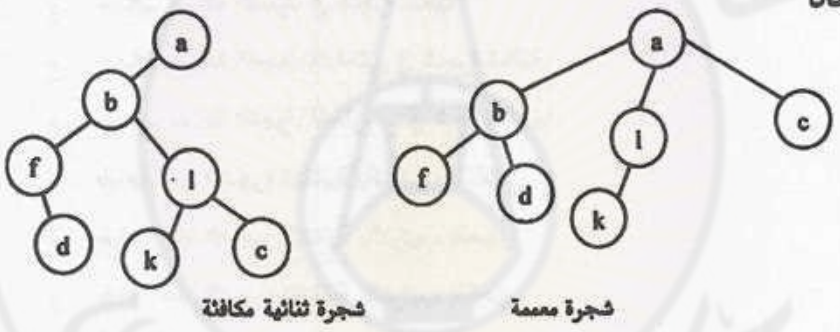
الشكل (7-12): طريقة أخرى لتمثيل شجرة معممة باستعمال سلاسل من الأولاد

ملاحظة

إذا لم يكن هناك ترتيب معرف على أبناء العقد في شجرة معممة، فيمكن تحويلها إلى شجرة ثنائية (الشكل (7-13)) بواسطة الخوارزمية البسيطة التالية:

- **الابن الأول لعقدة في الشجرة المعممة يصبح الابن اليساري لنفس العقدة في الشجرة الثنائية.**
- **أخو عقدة في الشجرة المعممة يصبح الابن اليميني لنفس العقدة في الشجرة الثنائية.**

مثال



الشكل (7-13): تحويل شجرة معممة إلى شجرة ثنائية مكافئة

7- تمارين

1- اكتب خوارزمية لكل ما يلي:

أ. حساب عدد العقد الداخلية في شجرة ثنائية.

ب. حساب عدد الأوراق في شجرة ثنائية.

ت. حساب ارتفاع شجرة ثنائية.

ث. حساب عرض شجرة ثنائية.

ج. حساب مسافة التجول في شجرة ثنائية.

ح. حساب مسافة التجول الداخلي في شجرة ثنائية.

خ. حساب مسافة التجول الخارجي في شجرة ثنائية.

د. طباعة عقد الشجرة الثنائية بالترتيب النظامي.

ذ. طباعة عقد الشجرة الثنائية بالترتيب المصدر.

ر. طباعة عقد الشجرة الثنائية بالترتيب الملحق.

ز. طباعة عقد الشجرة الثنائية الداخلية فقط.

س. طباعة أوراق الشجرة الثنائية فقط.

ش. طباعة طريق عقدة في شجرة ثنائية.

ص. التحقق من كون شجرة ثنائية هي خطية.

ض. التحقق من كون شجرة ثنائية هي تامة.

ط. التحقق من كون شجرة ثنائية هي كاملة.

2- اكتب النمط المجرد (أو جزءا منه) للشجرة الثنائية.

3- نسمي شجرة ثنائية مرتبة، شجرة ثنائية يوجد بين عقدها علاقة ترتيب. نفترض أن كل عقدة تحوي عددا صحيحا، وعلاقة الترتيب بين العقد هي: كل ابن يساري لعقدة هو أصغر تماما من العقدة الأب ؛ كل ابن يميني لعقدة هو أكبر تماما من العقدة الأب.

أ. اكتب الإجرائيات التالية في هذه الشجرة المرتبة:

- إضافة عقدة في مكانها الصحيح.
- حذف عقدة من مكانها الصحيح..
- البحث عن عقدة ما.

ب. قدر التعقيد الزمني للإجرائيات السابقة.

4- نفترض تمثيل التعابير الحسابية بواسطة شجرات ثنائية (العمليات ثنائية فقط).

أ. اكتب إجرائية تحويل تعبير حسابي مع أقواس، إلى شجرة ثنائية بلا أقواس، حيث العمليات هي العقد الداخلية، والمعاملات (Operands) هي الأوراق.

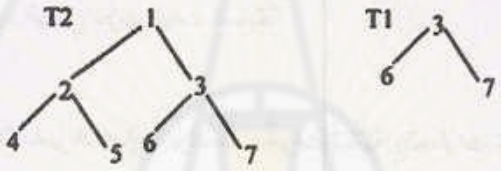
ب. بين أن عمليات التحويل الممكنة بين التعابير النظامية (Infix Expressions) والتعابير المصدرة (Prefix Expressions) والتعابير الملحقة (Postfix Expressions) هي مجرد تجول في الشجرة حسب ترتيب معين.

5- نقول عن شجرتين ثنائيتين إنهما متشابهتان بالتعاكس (Mirror Similar) إذا كانتا فارغتين معا، أو إذا كانتا غير فارغتين والشجرة الجزئية اليسارية من كلتيهما متشابهة بالتعاكس مع الشجرة الجزئية اليمنية من كلتيهما (انظر الرسم التوضيحي). اكتب خوارزمية تحقق من كون شجرتين ثنائيتين، متشابهتين بالتعاكس.

6- نقول عن شجرة ثنائية T1 إنها جزئية من شجرة ثنائية أخرى T2 إذا كانت T2 تحوي كامل T1 (انظر الرسم التوضيحي).

أ. اكتب إجرائية تتحقق من كون شجرة T1 جزئية من شجرة أخرى T2.

ب. اكتب إجرائية تقوم بحساب تكرار وجود T1 في T2.



7- اكتب خوارزمية التحويل من شجرة معممة إلى شجرة ثنائية المشروحة في آخر الفصل.

الفصل الثامن

البيانات

8–1 مقدمة

نركّز في هذا الفصل على بنية معطيات أخرى من البنى غير التسلسلية والمهمة جداً في التطبيقات البرمجية والرياضية. يمكن اعتبار البيانات (Graphs) تعميماً للبنى الهرمية غير التسلسلية من الناحية النظرية، فالأشجار هي حالة خاصة من البيانات (بيانات لا تحوي حلقات). لكننا آثرنا الفصل بينهما في هذا الكتاب بسبب الطبيعة المختلفة للخوارزميات المطبقة على كلٍ منهما.

8–2 تعاريف

تعريف 1

البيان هو ثنائية <V, E> حيث V مجموعة منتهية من العقد (تسمى هنا Vertices)، وE مجموعة منتهية من الثنائيات المرتبة <v1, v2>، حيث v1, v2 ∈ V. تُسمى هذه الثنائيات أسهماً (أو حافات Edges).

نقول عن عقدتين إنهما *متجاورتان* إذا وُجد سهم بينهما.

في البيانات غير الموجهة (Undirected Graphs) تكون الثنائيات التي تمثل الأسهم غير مرتبة لذا: $<v, w> = <v, w>$.

أمّا في البيانات الموجهة (Directed Graphs) فإن هذه الثنائيات تصبح مرتبة لهذا:

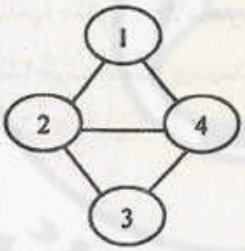
$<v, w> \neq <v, w>$.

لنضع نوعين من الضوابط على البيان:

- إذا كانت $<v, w>$ سهماً في البيان فإن $w \neq v$.
- البيان يمكن أن يحوي سهماً موجهاً في حالة بيان موجه على الأكثر بين عقدتين، وسهماً وحيداً على الأكثر بين عقدتين في حالة بيان غير موجه.

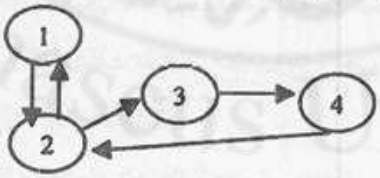
مثال

- في البيان G1 غير الموجه المبين في الشكل لدينا:



$V(G1) = \{ 1, 2, 3, 4 \}$; $E(G1) = \{ <1, 2>, <1, 4>, <2, 3>, <2, 4>, <3, 4> \}$.

- في البيان G2 الموجه في الشكل لدينا:



$V(G2) = \{ 1, 2, 3, 4 \}$; $E(G2) = \{ <1, 2>, <2, 1>, <2, 3>, <3, 4>, <4, 2> \}$

ملاحظة

عدد الروابط الأعظمي في بيان غير موجه يحوي n عقدة يساوي n.(n-1)/2، وفي هذه الحالة نقول عن البيان إنّه تامّ (Complete).

تعريف 2

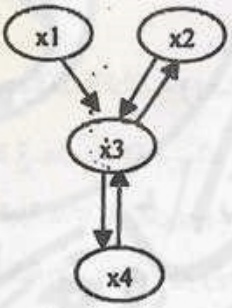
البيان الجزئي هو ثنائية <V', E'> حيث:

- $V' \subseteq V$ and $E' \subseteq E$
- $\langle v, w \rangle \in V' \Leftrightarrow v \in V'$ and $w \in V'$

تعريف 3

درجة عقدة v ونرمز إليها بـ Deg(v) عدد الأسهم الواردة إلى هذه العقدة والصادرة عنها. يمكن التمييز بين الدرجة الواردة $Deg^-(v)$ والدرجة الصادرة $Deg^+(v)$.

مثال



Deg(x3) = 5
$Deg^-(x3) = 3$
$Deg^+(x3) = 2$
Deg(x4) = 2
$Deg^-(x4) = 1$
$Deg^+(x4) = 1$

تعريف 4

نسمي *طريقاً* (Path) ضمن البيان، كل متتالية $v_i, v_{i+1}, \ldots, v_j$ من العقد، بحيث يوجد أسهم بين v_k وv_{k+1} عندما $i \le k \le j-1$. يكون الطريق بسيطاً (Elementary) إذا لم تتكرر أي عقدة فيه مرتين.

تعريف 5

الحلقة (Cycle) هي طريق بسيط بحيث $v_j = v_i$. نسميها حلقة موجهة في حالة بيان موجه.

تعريف 6

يكون البيان *متصل بقوة* (Strongly Connected)، إذا كان في حالة كل عقدتين v و w من البيان، يوجد طريق من v إلى w وطريق من w إلى v.

تعريف 7

المركبة المتصلة (Connected Component) في بيان هي بيان جزئي أعظمي متصل بقوة.

8-3 تمثيل البيانات الموجهة

يمكن تمثيل بيان موجه بطرق عديدة نذكر منها:

8-3-1 تمثيل باستعمال مصفوفة من المتحولات المنطقية (مصفوفة تجاور)

تكون مصفوفة التجاور (Adjacency Matrix) M في هذه الحالة مربعة، الأسطر والأعمدة فيها هي أرقام العقد. نضع في الخانة M[i, j] القيمة true إذا كان يوجد رابط موجه من العقدة v_i إلى العقدة v_j. على حين نضع القيمة false عند عدم وجود رابط بين العقدتين. لاحظ أنه في حالة البيان غير الموجه، تصبح هذه المصفوفة متناظرة بالنسبة إلى القطر الرئيسي.

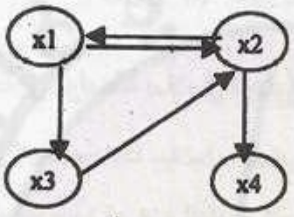
إذن نعرّف بنى المعطيات بالشكل التالي:

```
const
    n = 100;
```

```
type
    MGraph = array[1..n, 1..n] of boolean;
```

مثال

الشكل (8-1) يُظهر بياناً موجهاً وطريقة تمثيله بمصفوفة تجاور.



F	T	T	F
T	F	F	T
F	T	F	F
F	F	F	F

الشكل (8-1): تمثيل بيان بمصفوفة تجاور

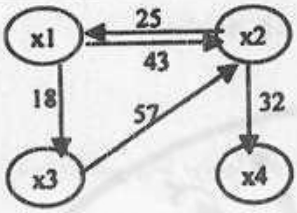
8-3-2 تمثيل باستعمال مصفوفة تعطي كلفة قوس

تكون مصفوفة التجاور M في هذه الحالة أيضاً مربعة، الأسطر والأعمدة فيها هي أرقام العقد. نضع في الخانة M[i,j] قيمة كلفة الانتقال من العقدة v_i إلى العقدة v_j، إذا كان يوجد رابط موجه من العقدة v_i إلى العقدة v_j. على حين نضع القيمة -1 عند عدم وجود رابط بين العقدتين. لاحظ أيضاً أنه في حالة البيان غير الموجه، تصبح هذه المصفوفة متناظرة بالنسبة إلى القطر الرئيسي. بتعديل طفيف على بنى المعطيات السابقة لدينا:

```
const
    n = 100;
type
    MVGraph = array[1..n, 1..n] of real;
```

مثال

الشكل (8-2) يُظهر بياناً موجهاً وطريقة تمثيله بمصفوفة تجاور.



-1	25	18	-1
43	-1	-1	32
-1	57	-1	-1
-1	-1	-1	-1

الشكل (8-2): تمثيل بيان بمصفوفة كلفة أقواس

8-3-3 التمثيل بواسطة سلاسل تجاور

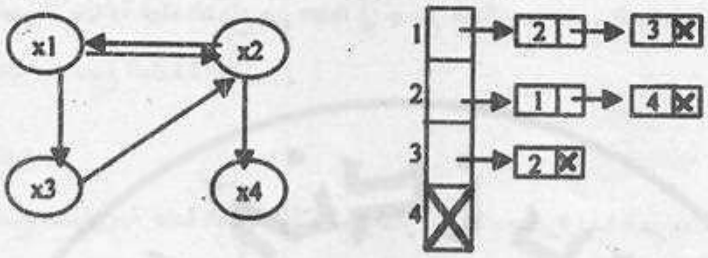
في حالة تمثيل البيان بواسطة سلاسل التجاور (Adjacency Lists)، نمثل البيان بواسطة شعاعٍ من المؤشرات بُعدُه هو عدد العقد في البيان، ومرقّم حسب أرقام العقد. خانة كل عقدة i هي مؤشر إلى بداية سلسلة خطية تحوي جميع العقد j التي ترتبط بـ i وفق الاتجاه i إلى j، وتكون هذه السلسلة مرتبة تصاعدياً حسب أرقام العقد.

نعرّف بنى المعطيات بالشكل التالي:

```
const
    n = 100;
type
    Pvertex = ^Vertex;
    Vertex = record
                 val  : Element;
                 next : Pvertex
             end;
    LGraph = array[1..n] of PVertex;
```

مثال

الشكل (8-3) يُظهر بياناً موجهاً وطريقة تمثيله بواسطة سلاسل تجاور.



الشكل (8–3): تمثيل بيان بواسطة سلاسل تجاور

8–4 بعض العمليات على البيانات

يوجد عدد كبير من العمليات المهمة التي تُجرى على البيانات، نختار في هذا الكتاب بعضاً منها:

8–4–1 مسح البيان غير الموجه

ليكن البيان G = (V, E) غير موجه، ولتكن v عقدة ما في هذا البيان. مسح البيان (Graph Traversal) يعني زيارة جميع العقد في G التي يمكن الوصول إليها بطريق من v. هناك طريقتان للمسح: العمق أولاً (Depth First)، والعرض أولاً (Breadth First).

آ– مسح البيان بالعمق أولاً

الدخل

البيان G = (V, E) غير موجه، وv عقدة البداية.

نعرّف الشعاع المتحول Visited بالشكل التالي:

```
Visited : array[1..n] of boolean;
```

القيمة الابتدائية لهذا الشعاع هي false في جميع الخانات. نغير الخانة Visited[i] إلى true عند مسح العقدة i فقط.

الفكرة

تمسح الخوارزمية عقدة البداية أولا، ثم تختار عقدة مجاورة لها غير ممسوحة بعد، وتستدعي الخوارزمية العودية نفسها عند هذه العقدة. عندما تصل الخوارزمية إلى عقدة ليس لها عقد مجاورة غير ممسوحة، تعود إلى آخر عقدة ممسوحة ولها عقد مجاورة لم تمسح بعد. تنتهي الخوارزمية عند مسح جميع العقد.

الخوارزمية

```
procedure DFS(v)
var
    Visited : array[1..n] of boolean;
    w : Vertex;
begin
    Visited[v] := true;
    for each vertex w adjacent to v do
        if Visited[w] = 0 then
            DFS(w)
end; { DFS }
```

ب- طريقة العرض أولا

الدخل

البيان G = (V, E) غير موجه، وv عقدة البداية. ويبقى تعريف الشعاع Visited كما هو.

الفكرة

تمسح الخوارزمية عقدة البداية أولا، ثم تمسح جميع العقد المجاورة لها غير الممسوحة بعد وتضعها في رتل (Queue).

من أجل كل عقدة باقية في الرتل تمسح الخوارزمية العقد المجاورة لها وتضعها في الرتل وهكذا. تنتهي الخوارزمية عندما يفرغ الرتل.

الخوارزمية

```
procedure BFS(v)
var
    Visited : array[1..n] of boolean;
    w : Vertex;
    Q : Queue;
begin
    Visited[v] := 1;
    Q := EmptyQueue;
    While (1)
        for all vertices w adjacent to v do
            if Visited[w] = 0 then
                begin
                    Enqueue(w,Q);
                    Visited[w] := 1;
                end
        if Q = EmptyQueue. then
            exit
        else
            v := Dequeue(Q)
    end;
end; { BFS }
```

ونترك للقارئ مهمة إعادة كتابة هذه الخوارزميات في جميع حالات تمثيل البيان.

8-4-2 مسح البيان الموجه

لا تختلف طرق المسح وإجراءاتها هنا عن تلك الإجراءات المطبقة في مسح البيان غير الموجه. الفرق الوحيد هو مراعاة اتجاه الأسهم. لذلك عندما يجب أن تمسح الخوارزمية العقدة ز المجاورة للعقدة i التي عندها الخوارزمية، يجب أن يكون السهم بين i وز

بالاتجاه <i,j>، بمعنى آخر يجب أن تنتمي العقدة j إلى $Deg^+(i)$ (والتي نعتبرها هنا مجموعة العقد المجاورة لـ i في حالة بيان موجه).

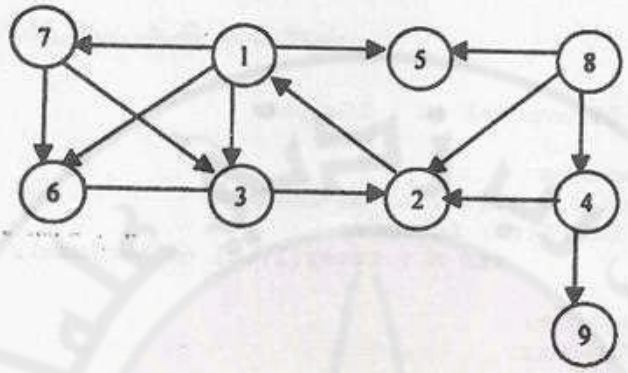
لنأخذ طريقة العمق أولاً ونعيد كتابة الخوارزمية DFS، حيث نستخدم هنا الإجرائية المساعدة OrDFS التي تسمح بالمرور بجميع العقد التي يمكن الوصول إليها من عقدة معينة والتي لم يسبق المرور فيها.

```
procedure DTraversal (G : Graph);
var
    i : integer;
    Visited : array[1..n] of boolean;
    procedure OrDFS(s : integer;
                    var M : array[1..n] of boolean);
    var
        j, v : integer;
    begin
        M[s] := true;
        for j := 1 to Deg+(s) do
            begin
                v := jth adjacent of s;
                if not (M[j]) then
                    OrDFS(v, M)
            end
    end;

begin
    for i := 1 to n do
        Visited[i] := false;
    for i := 1 to n do
        if not (Visited[i]) then
            OrDFS(i, M)
end;
```

مثال

عند تطبيق خوارزمية التجول على البيان G الموضّح في الشكل (8-4) فإن ترتيب المرور في العقد هو:

1, 3, 2, 6, 5, 7,
4, 9,
8



الشكل (8-4): مثال بيان موجه

لنطبق هذه الخوارزمية في الحالتين التاليتين:

- *تمثيل البيان بواسطة مصفوفة تجاور*

```
procedure DTraversal (G : MGraph);
var
    i : integer;
    Visited : array[1..n] of boolean;
    procedure OrDFS(s : integer;
                    var M : array[1..n] of boolean);
    var
        j : integer;
    begin
        M[s] := true;
        for j := 1 to n do
            if G[s, j] and not (M[j]) then
                OrDFS(s, M)
    end;

begin
    for i := 1 to n do
        Visited[i] := false;
    for i := 1 to n do
```

```
        if not (Visited[i]) then
            OrDFS(i, M)
end;
```

- *تمثيل البيان بواسطة سلاسل التجاور*

```
procedure DTraversal (G : LGraph);
var
    i : integer;
    Visited : array[1..n] of boolean;
procedure OrDFS(i : integer;
                var M : array[1..n] of boolean);
var
    j : integer;
    s : PVertex;
begin
    M[s] := true;
    s := G[i];
    while <> nil do
        begin
            j := s^.val;
            { here supposed Number of Vertex}
            if  not (M[j]) then
                OrDFS(j, M);
            s:= s^.next
        end
    end;
begin
    for i := 1 to n do
        Visited[i] := false;
    for i := 1 to n do
        if not (Visited[i]) then
            OrDFS(i, M)
end;
```

3-4-8 الفرز الطبولوجي

يهدف الفرز الطبولوجي (Topological Sort) إلى إيجاد ترتيب بين العقد، بحيث نبني سلسلة من العقد لا تظهر عقدة معينة قبل واحدة من سابقاتها.

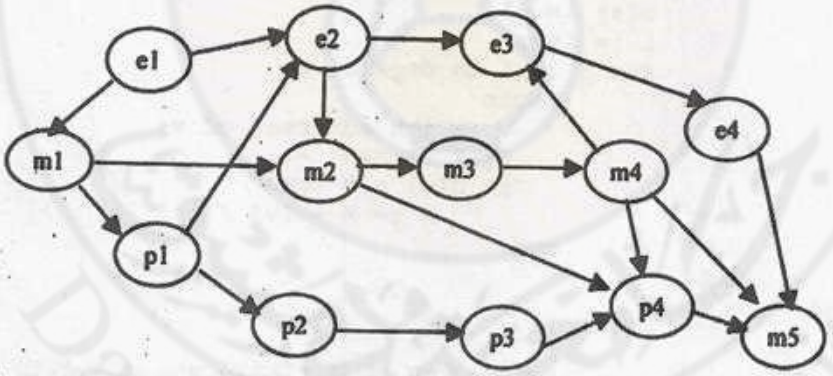
تطبيق

بغية وضع المنهاج السنوي لجامعة، مع مراعاة اعتماد المواد بعضها على بعض، نستعمل بيانا موجها، تمثل عقده المقررات الدراسية، وتمثل الأسهم اعتماد المواد بعضها على بعض، ويظهر الشكل (8–6) مثالا لسنة دراسية فيها 13 مادة. يمكن بنتيجة الفرز الطبولوجي أن نحصل على السلسلة:

(e1, m1, p1, p2, e2, m2, m3, m4, p3, p4, e3, e4, m5)

المبدأ

نستعمل مصفوفة d تحوي في البداية الدرجة الواردة لكل عقدة أو 1- إذا كانت هذه العقدة قد أخذت بعين الاعتبار.



الشكل (8–6): مثال لمواد سنة دراسية يعتمد بعضها على بعض

الخوارزمية

نفترض أن البيان G يحوي n عقدة ولا يحوي حلقات.

```
procedure TopoSort (G: Graph, var L : List);
var
    i, s, v, w : integer,
    d : array[1..n] of integer:
    M : set of 1..n;
begin
    M := [];
    L := EmptyList;
    for s := 1 to n do
        begin
            i :=  deg+(s);
            if  i = 0 then
                M := M+[s];
            d[s] := i
        end;
    i := 1;
    while M <> [] do
       begin
            v := GetElement(M);
            { chooses an element of a set }
            M := M - [v];
            d[v] := -1;
            i := i+1;
            for j := 1 to deg+(s) do
                begin
                    w := jth adjacent of v;
                    d[w] := d[w] -1;
                    if d[w] = 0 then
                        M := M + [v]
                end
       end
end;
```

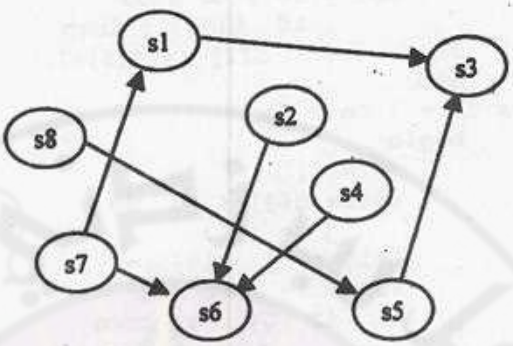
إذا كان البيان ممثلا بمصفوفة تجاور فإن الخوارزمية السابقة تصبح:

```
procedure TopoSort (G : MGraph, var L : List);
var
    i, j, k : integer;
    d : array[1..n] of integer;
begin
    for i := 1 to n do
        begin
            d[i] := 0;
```

```
            for j := 1 to n do
                if  G[j, i] then
                    d[i] := d[i]+1;
        end;
    for i := 1 to n do
        begin
            j := 1;
            while d[j] <> 0  do
                j := j+1;
            L[i] := j; d[j] := -1;
            for k := 1 to n do
                if  G[j, k] then
                    d[k] := d[k]+1;
        end;
end;
```

مثال

يبين الشكل (8-7) مراحل تطبيق خوارزمية الفرز الطبولوجي.



Vertices	s1	s2	s3	s4	s5	s6	s7	s8
d	2	0	2	0	1	3	0	0
L[i] = 2	2	-1	2	0	1	2	0	0
L[i] = 4	1	-1	2	-1	1	1	0	0
L[i] = 7	0	-1	2	-1	1	0	-1	0
L[i] = 1	-1	-1	1	-1	1	0	-1	0
L[i] = 6	-1	-1	1	-1	1	-1	-1	0
L[i] = 8	-1	-1	1	-1	0	-1	-1	-1
L[i] = 5	-1	-1	0	-1	-1	-1	-1	-1
L[i] = 3	-1	-1	-1	-1	-1	-1	-1	-1

الشكل (8-7): مراحل تطبيق خوارزمية الفرز الطبولوجي

8-5 تمارين

1- من الطرق الممكنة لتمثيل بيان غير موجه G=(V,E) طريقة مصفوفة الورود (Incidence Matrix): INC. في هذه المصفوفة كل سطر يمثل عقدة واحدة من البيان وكل عمود يمثل رابطا واحدا من البيان. إذن INC(i,j)=1 إذا كان الرابط j يصل إلى أو ينطلق من العقدة i (ترقيم الروابط يجري من اليسار إلى اليمين ثم من الأعلى إلى الأسفل).

أ. اقترح بنى المعطيات المناسبة (دعم شرحك برسم توضيحي).

ب. أعد كتابة الإجرائية العودية العمق أولا (DFS) لزيارة جميع عقد البيان باستخدام مصفوفة الورود.

ت. أعد كتابة إجرائية العرض أولا (BFS) لزيارة جميع عقد البيان باستخدام مصفوفة الورود.

ث. اكتب الإجرائية IsTree للتحقق من البيان الممثل بهذه الطريقة: أهو عبارة عن شجرة ثنائية أم لا ؟.

ج. إذا كانت ADJ مصفوفة التجاور (Adjacency Matrix) التي درسناها في هذا الفصل والتي تمثل البيان G=(V, E) وكانت INC مصفوفة الورود لنفس البيان. ضمن أي من الشروط يمكن برهان: $\boxed{ADJ = INC \times INC^T - I}$

(× هو ضرب المصفوفات ، INC^T هي المصفوفة المنقولة (Transpose Matrix) لـ INC ، I المصفوفة الواحدية).

2- من خوارزميات مسح البيان (Graph Traversal) درسنا الخوارزميتين التاليتين:

- *العمق أولا* (Depth First Search: DFS)
- *العرض أولا* (Breadth First Search: DFS)

المطلوب إعادة كتابة الخوارزميتين السابقتين في كلتا الحالتين التاليتين: تمثيل البيان بمصفوفة تجاور (Adjacency Matrix)، تمثيل البيان بسلسة تجاور (Adjacency List).

3- سنجري تعديلا طفيفا على تمثيل بيان موجه بواسطة سلاسل التجاور (Adjacency Lists) كما يلي:

نربط بكل عقدة (هي خانة من جدول العقد) سلسلتين خطيتين ؛ الأولى تحوي العقد التي يمكن الوصول إليها بواسطة رابط (موجه طبعا) من هذه العقدة ؛ والثانية تحوي العقد التي منها يمكن الوصول بواسطة رابط إلى هذه العقدة.

أ. اقترح بنية المعطيات المناسبة التي تتضمن هذا التعديل لتمثيل البيان (دعم شرحك برسم توضيحي لمثال على بيان).

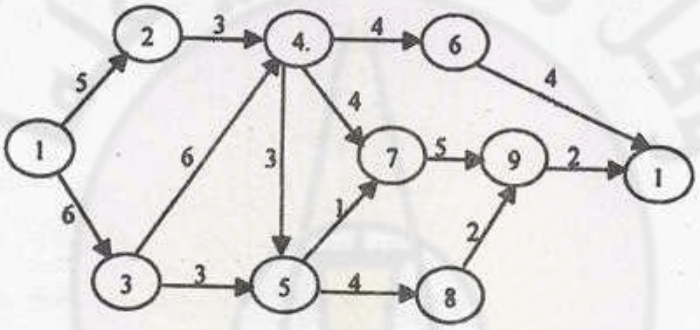
ب. اكتب إجرائية تمثل بيانا موجها بواسطة بنية المعطيات الجديدة، علما أن دخل هذه الإجرائية هو تمثيل نفس البيان بواسطة سلاسل التجاور التقليدية.

ت. اكتب إجرائية تستفيد من طريقة التمثيل الجديدة، وذلك لإيجاد حلقة بدايتها ونهايتها هي عقدة ما من البيان.

ث. هل تعتقد أن الطريقة الجديدة في التمثيل قد غيرت شيئا في كتابة إجرائية إيجاد حلقة ؟.

4- لتمثيل ترابط الأعمال المطلوب إنجازها في إطار مشروع معين، يمكن استخدام بيان موجه، تمثل العقد فيه الأعمال الجزئية، وتمثل الأسهم اعتماد المراحل بعضها على بعض. ويوضع رقم على كل سهم يبين المدة اللازمة بين بداية عمل معين، والعمل الذي يليه.

مثال



نلاحظ أن العمل رقم 2 لا يمكنه البدء به قبل انقضاء 5 أيام على بداية العمل رقم 1.

أ. عرف بنية المعطيات التي تمكن من تخزين مثل هذه البيانات.

ب. اكتب التابع E(i) الذي يحدد أقصر مدة يمكن بعدها البدء بتنفيذ العمل رقم i.

ت. اكتب التابع L(i) الذي يحدد آخر موعد للبدء بتنفيذ العمل رقم i.

ث. اكتب إجرائية تحدد الأعمال الحرجة (الأعمال التي تؤخر تنفيذ المشروع).

ج. اكتب إجرائية تحدد أقصر مدة لازمة لتنفيذ أعمال المشروع كافة.

الفصل التاسع

جداول التقطيع

9-1 مقدمة

تطلب العديد من التطبيقات البرمجية إجراء عمليات الإضافة (Insertion) والحذف (Deletion) والبحث (Search) والتي نطلق عليها عادةً عمليات المعجم (Dictionary Operations).

أمثلة

- يحتاج مترجم لغة برمجة معينة إلى جدول للرموز (Symbol Table) تكون فيه مفاتيح العناصر، سلاسلَ محارف عشوائية تعبر عن المتحولات الممكنة في لغة البرمجة هذه. وأثناء تنفيذ الترجمة (Compilation) يقوم المترجم باستدعاءات عديدة لهذه العمليات.
- يتطلب استخدام مكنز (Thesaurus) في أي تطبيق مكتبي أو نشر إلكتروني بوجهٍ خاص، البحثَ عن المرادفات والكلمات العامة والخاصة لمصطلح معين.

جداول التقطيع (Hash Tables) هي بنى معطيات فعّالة في برمجة التطبيقات التي تحتاج إلى عمليات المعجم.

مع أن البحث عن عنصر في جدول تقطيع، يحتاج إلى زمن تنفيذ من مرتبة O(n) في أسوأ الحالات (مثل الزمن اللازم للبحث عن عنصر في سلسلة خطية)، فإن الزمن المتوقع للبحث عن عنصر في جدول تقطيع هو، ضمن فرضيات مناسبة، من مرتبة O(1).

يعتبر جدول التقطيع نوعاً من التعميم لمفهوم الجدول الاعتيادي. وباستخدام العنونة المباشرة (Direct Addressing) في جدول اعتيادي نستطيع البحث عن موقع ما من هذا الجدول بزمن من مرتبة O(1).

تُطبق عملياً العنونة المباشرة عندما نستطيع منح موقع من الجدول لكل مفتاح (Key) عنصر محتمل. عندما يكون عدد المفاتيح المخزن فعلياً صغيراً بالمقارنة بالعدد الكلي للمفاتيح المحتملة، فإن جداول التقطيع تصبح بديلاً مناسباً لجداول العنونة المباشرة. إذ يستعمل جدول التقطيع عادةً جدولاً ذا حجم متناسب مع عدد المفاتيح المخزنة فعلياً (لا مع عدد المفاتيح المحتملة). لذلك، عوضاً عن استعمال المفاتيح كأدلة مباشرة (أرقام المواقع) للجدول، يمكن حساب هذه الأدلة من المفاتيح مباشرةً باستخدام توابع التقطيع المناسبة.

سنعرض أيضاً في هذا الفصل أفكاراً وتقنيات أخرى وكلها تقوم على القاعدة التالية: *"التقطيع (Hashing) تقنية عملية وفعّالة في إجراء عمليات المعجم بزمن وسطي من مرتبة O(1)."*

9-2 جداول العنونة المباشرة

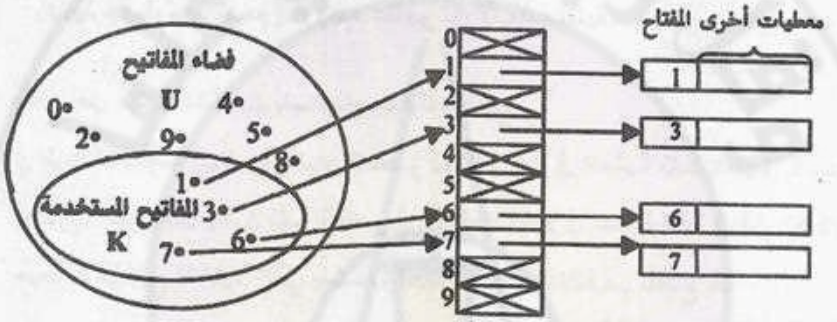
العنونة المباشرة تقنية بسيطة وجيدة عندما تكون مجموعة المفاتيح صغيرة إلى حد معقول. لنفترض أن التطبيق يحتاج إلى مجموعة ديناميكية من العناصر، تقع مفاتيحها ضمن المجال (فضاء المفاتيح):

$U = \{0, 1, \ldots, m-1\}$

m كبيرة جداً. وسنفرض أنه يوجد مفتاح وحيد لكل عنصر.

نستخدم لتمثيل هذه المجموعة الديناميكية جدولاً أو *جدول عنونة مباشرة*: T[0..m-1] بحيث يطابق كل موقع من هذا الجدول مفتاحاً من الفضاء U.

يُظهر الشكل (9-1) تطبيق هذه الطريقة ؛ حيث الدليل k من الجدول يؤشر إلى العنصر الذي له المفتاح k من المجموعة.



الشكل (9-1): تنفيذ مجموعة ديناميكية باستخدام جدول عنونة مباشرة

تصبح عمليات المعجم بديهية في هذه الطريقة:

```
Direct-Address-Search (T, x)
    return T[k]
Direct-Address-Insert (T, x)
    T[key[x]] := x
Direct-Address-Delete (T, x)
    T[key[x]] := nil
```

هذه العمليات سريعة التنفيذ، إذ تحتاج إلى زمن من مرتبة O(1). وتجدر الإشارة إلى أنه في بعض التطبيقات تخزن العناصر في الجدول نفسه، وفي هذه الحالة يصبح دليل العنصر هو مفتاحه، ومن ثم لا تخزن المفاتيح.

9-3 جداول التقطيع

هناك مشكلتان مع جداول العنونة المباشرة:

- عندما تكون مجموعة المفاتيح المستعملة K صغيرة جداً بالنسبة إلى حجم مجموعة المفاتيح الممكنة U، إذ يكون الحجم الضائع من جدول العنونة المباشرة T كبيراً.
- عندما تكون مجموعة المفاتيح الكلية U كبيرة جداً، إذ يصبح من غير الممكن تنفيذ جدول من الحجم |U| (عدد عناصر U) لأسباب تقنية.

يمكن حل هاتين المشكلتين باستخدام جداول التقطيع:

في طريقة العنونة المباشرة يخزن العنصر ذو المفتاح k في الخانة ذات الدليل k من الجدول، على حين يخزن نفس العنصر في جدول التقطيع في الخانة ذات الدليل h(k)، حيث h هو تابع التقطيع الذي يستخدم لحساب دليل الخانة من المفتاح k:

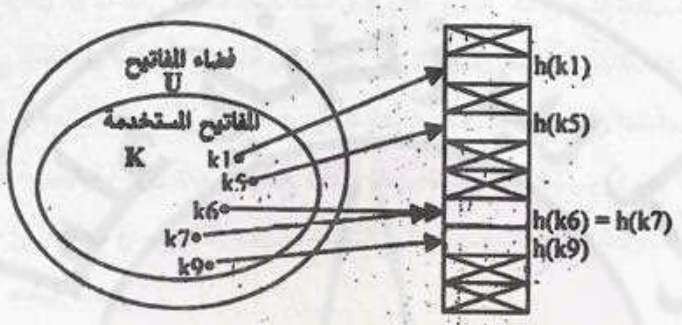
$h : U \rightarrow [0, 1, \ldots, m-1]$

نسمي h(k) بقيمة تقطيع (Hash Value) المفتاح k. يُظهر الشكل (9-2) هذه الفكرة الأساسية.

الهدف من جداول التقطيع إذن هو تخفيض عدد أدلة الجدول (ومن ثم حجم الجدول): يصبح لدينا m دليلاً عوضاً عن |u| دليلاً. مشكلة هذه الطريقة هي وجود عدة عناصر لها نفس قيمة التابع h، وهذا يؤدي إلى تزاحم أو تصادم (Collision) لكن يوجد تقنيات فعّالة لحل هذه المشكلة.

من الطبيعي أن الحل المثالي هو تجنب هذه التصادمات، وذلك باختيار تابع تقطيع مناسب وعشوائي بدرجة كافية، وإذا لم يستطع تجنب التصادمات فإنه يخفف منها

على الأقل. لكن باعتبار أنه في جميع الحالات: $m < |U|$ فإن التصادم سيحدث لا محالة. لذلك لا بدّ من استعراض التقنيات التالية لحل مشكلة التصادم.



الشكل (9-2): تنفيذ مجموعة ديناميكية باستخدام جدول تقطيع

9-3-1 تقنية الربط لحل التصادم

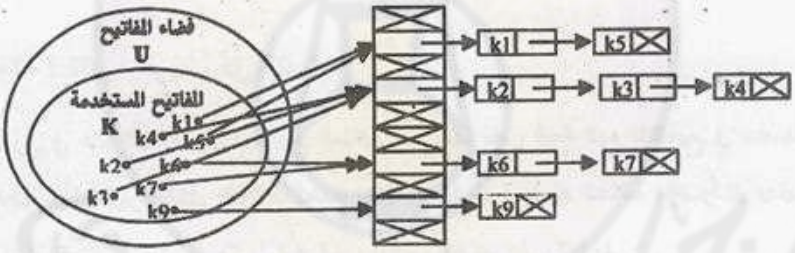
يجري في هذه الطريقة وضع جميع العناصر التي لها نفس قيمة تابع التقطيع في سلسلة خطية كما يُظهر الشكل (9-2). تصبح خانة الجدول في هذه الحالة مؤشراً إلى بداية سلسلة العناصر "المتصادمة". أمّا عمليات المعجم فتصبح على كما يلي:

```
Chained-Hash-Insert (T, x)
    insert x at the head of list T[h[key[x]]
Chained-Hash¬Search (T, x)
    search for element with key k in list T[h[k]]
Chained-Hash-Delete (T, x)
    delete x from the list T[h[key[x]]
```

نلاحظ أن زمن عملية الإضافة هي من مرتبة O(1) في أسوأ الأحوال (الإضافة تحدث دوماً في بداية السلسلة). على حين يتعلق زمن عمليتي الحذف والبحث في أسوأ الأحوال بطول

السلسلة الخطية المحتملة. لذلك نعرّف عامل الخزن (Load Factor) في الجدول بالعلاقة: $\alpha = n/m$ حيث n هو عدد العناصر الكلي وm هو عدد الخانات في الجدول.

مع افتراض أن α تبقى ثابتة عند ذهاب n وm إلى ∞. على هذا تعطينا α الطول الوسطي للسلاسل الخطية المرتبطة بالخانات. من المؤكد أن تقدير الزمن للحذف والبحث في أسوأ الأحوال سيقودنا إلى زمن من مرتبة O(n) (حالة وجود كل العناصر في نفس السلسلة أو بعبارة أخرى تابع التقطيع يعطي نفس القيمة لكل العناصر)، وهو نفس زمن البحث أو الحذف في سلسلة خطية، وبالتالي يفضل تجنب هذه الطريقة واستخدام سلسلة خطية عادية.



الشكل (9-3): حل التصادم بتقنية الربط

إذن لما كان ذلك يتعلق بتابع التقطيع، وجب اختيار توابع تقطيع تحقق تقطيعاً منتظماً (Uniform Hashing) وبسيطاً، أي يكون احتمال وقوع أي عنصر في خانة من جدول التقطيع ثابت القيمة. إذا افترضنا في هذه الحالة أن زمن حساب تابع التقطيع هو من مرتبة O(1)، فإن الزمن الوسطي لإجراء عملية بحث أو حذف هو من مرتبة: $\theta(1 + \alpha)$.

لاحظ أنه عندما تكون: $n = O(m)$ فإن $\alpha = O(1)$ لذا تصبح عمليات البحث والحذف ثابتة. نستنتج من كل هذا النقاش أن اختيار توابع تقطيع "جيدة" هو مسألة أساسية من أجل الاستثمار الجيد لمفاهيم التقطيع.

9-4 توابع التقطيع

9-4-1 ما الذي يجعل تابع التقطيع "جيدا" ؟

يكون تابع التقطيع "جيدا" عندما يحقق على وجه التقريب تقطيعا بسيطا ومنتظما: كل مفتاح له نفس الاحتمال لربطه بأي خانة من $0, \ldots, m-1$.

فإذا افترضنا أن كل مفتاح k يحسب على حدته في توزيع احتمالي P ؛ ومن ثم P(k) هو احتمال وضع k في خانة، وجب أن يحقق التقطيع البسيط والمنتظم العلاقة:

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m}$$

عندما: $j = 0, 1, \ldots, m-1$. لكن من الصعب التحقق دائما من هذا الشرط بسبب كون التوزيع P غير معروف عموما.

مثال

نفترض أن المفاتيح هي أعداد حقيقية وعشوائية وموزعة توزيعا منتظما ومستقلا في المجال $0 \le k < 1$، في هذه الحالة يحقق تابع التقطيع: $h(k) = \lfloor k.m \rfloor$ الشرط السابق.

نستخدم عمليا طرقا مساعدة (Heuristic) لبناء تابع تقطيع "جيد". وقد تكون المعلومات النوعية عن التوزيع P مفيدة في عملية تصميم التابع. لنأخذ على سبيل المثال حالة جدول الرموز في مترجم، حيث يمكن أن تكون المفاتيح -التي تمثل المتحولات في

برنامج- سلاسلَ محارف عشوائية. ومن الشائع استخدام متحولات شبه متشابهة مثل ptr وpts في نفس البرنامج. كي يكون تابع التقطيع "جيداً" في هذه الحالة، عليه أن يقلل من إمكانية تقطيع هذين المتحولين في نفس الخانة.

الطريقة العامة هي اشتقاق قيم التقطيع بحيث تكون مستقلة عن أي عينة (Pattern) يمكن وجودها في المعطيات.

9-4-2 تفسير المفاتيح كأرقام طبيعية

تفترض معظم توابع التقطيع أن فضاء المفاتيح هو مجموعة الأعداد الطبيعية: N={0, 1, 2, ...}، لذلك عندما لا تكون المفاتيح أرقاماً طبيعية، يجب إيجاد طريقة لتحويلها إلى أرقام طبيعية. من الممكن دوماً إيجاد مثل هذه الطرق (لذلك فيما تبقى من الفصل سنفترض أن المفاتيح هي أعداد طبيعية) كما يُظهر المثال التالي.

مثال

عندما تكون المفاتيح سلاسلَ محارف، يمكن تحويلها إلى أعداد طبيعية باستخدام الجذر (Radix) المناسب. فمثلاً إذا كان لدينا السلسلة pt نعبر عنها بالثنائية (112, 116) حيث Ascii(p)=112 ; Ascii(t)=116 وباستخدام الجذر 128 يصبح العدد الطبيعي الذي يمثل السلسلة مساوياً 14452 = 116 + (112.128).

9-4-3 طريقة التقسيم

يُبنى تابع تقطيع باستخدام طريقة التقسيم (Division Method) بربط المفتاح k بالخانة التي تساوي باقي قسمة k على m: $h(k) = k \bmod m$.

فمثلاً إذا كانت m = 12 وk = 100 يكون h(100) = 100 mod 12 = 8. هذه الطريقة سريعة جداً لأنها تتطلّب فقط عملية قسمة واحدة.

باستخدام هذه الطريقة يجب تجنب بعض قيم m مثل $m=2^p$ لأنها تعتمد فقط على الـ p خانة ثنائية الأخيرة ذات القوة الأصغر من k. إن أفضل القيم لـ m هي أعداد أولية بعيدة إلى حد كاف عن قوى 2. يمكن أيضا -على جهة الاحتراز- بعد اختيار m أن نختبر توزيع تابع التقطيع للمفاتيح على خانات الجدول.

مثال

لنفترض أننا نريد تخزين 2000 سلسلة محارف (يمثل كل محرف بـ 8 خانات ثنائية) في جدول تقطيع حيث تحل التصادمات فيه بطريقة الربط. ولا يزعجنا اختبار 3 سلاسل في حالة بحث غير موفق ($\alpha = 3$): $m ? 666.60 \Rightarrow \alpha = 2000/m = 3$.

نختار العدد الأولي $m = 701$ قريبا من 666.6 وبعيدا عن 1024 و512 (قوى 2) وبالتالي يصبح تابع التقطيع: $h(k)=k \bmod 701$.

9-4-3- طريقة الضرب

يبنى تابع تقطيع باستخدام طريقة الضرب (Multiplication Method) بخطوتين:

- نضرب المفتاح k بثابت A في المجال $0<A<1$ ونأخذ الجزء الكسري من k.A،
- نضرب هذه القيمة بـ m ونأخذ الجزء الطبيعي من العدد الناتج.

$h(k) = \lfloor m.(k.(A \bmod 1)) \rfloor$;

حيث $k.(A \bmod 1)$ هي الجزء الكسري من k.A.

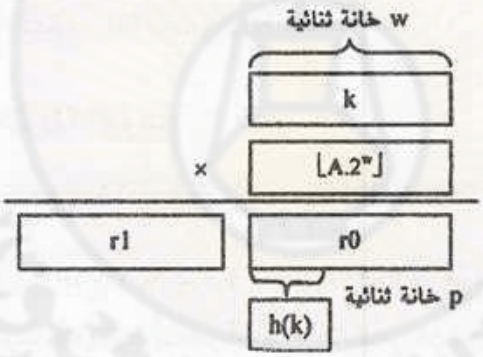
ميزة هذه الطريقة أن اختيار قيمة m يصبح غير حرج، حتى إنه من المنطقي اختيار M من قوى 2: $m = 2^p$ لسهولة تنفيذه على معظم الحواسيب:

لنفترض أن حجم الكلمة في الحاسوب هو w خانة ثنائية ونستطيع تمثيل k في كلمة واحدة. نضرب أولاً k بالعدد الطبيعي $\lfloor A.2^w \rfloor$، جُداء الضرب هو كلمة من 2w خانة وقيمتها $r1.2^w + r0$ حيث r1 هو الجزء الأعلى من الكلمة الجديدة وr0 هو الجزء الأدنى. قيمة التقطيع المرغوبة هي الـ p خانة ثنائية العليا من r0 كما يظهر الشكل (9–4).

لاحظ أنه من المفضل اختيار بعض القيم المناسبة من A. وقد وجد Knuth أن القيمة: $A? (\sqrt{5}-1)/2 ? 0.618034...$ هي قيمة جيدة.

مثال

$$k = 12345, m = 256 \Rightarrow h(k) = \lfloor (256 . (12345 . 0.618034 \bmod 1) \rfloor = 161$$



الشكل (9–4): طريقة الضرب لحساب تابع تقطيع

9–4–4 التقطيع العام

باعتبار أن تابع التقطيع محدّد سلفاً، من الممكن اختيار قيم مفاتيح بحيث تؤدي إلى أسوأ حالة O(n) (لأن قيمة التقطيع تعتمد على طريقة تخزين المفاتيح). الطريقة الوحيدة لتحسين ذلك هو اختيار تابع تقطيع اختياراً عشوائياً ومستقلاً عن طريقة تخزين المفاتيح.

هذه الطريقة تسمى بالتقطيع العام (Universal Hashing) وتعطي وسطياً أداءً جيداً بغض النظر عن طريقة تخزين المفاتيح.

الفكرة الأساسية وراء التقطيع العام هي اختيار تابع عشوائي، وأثناء التنفيذ من صف مصمم بعناية من توابع التقطيع، وهذا مما يضمن معدّلاً جيداً للأداء. وبسبب هذه العشوائية تسلك الخوارزمية أثناء التنفيذ سلوكاً متغيراً حتى بالنسبة إلى مفاتيح متساوية.

لتكن $\mathcal{H}$ مجموعة من توابع التقطيع بحيث (U هي فضاء المفاتيح):

$h \in \mathcal{H} \Rightarrow h : U \rightarrow \{0, 1, ..., m-1\}$

نقول عن $\mathcal{H}$ أنها عامة إذا كان في حالة كل مفتاحين $x, y \in U,\ x ? y$ يكون عدد توابع التقطيع من $\mathcal{H}$ التي تجعل $h(x) = h(y)$ هو تماماً: $|\mathcal{H}| / m$.

لكن كيف يمكن بناء صف عام من توابع التقطيع ؟

لنختر m عدداً أولياً ولنقسّم المفتاح x إلى r+1 بايت (أحرف مثلاً):

$x = <x0, x1, ..., xr>$

شريطة أن تكون أكبر قيمة لبايت xi أصغر من m ولتكن:

$a = <a0, a1, ..., an>$

سلسلة من r+1 قيمة مختارة بشكل عشوائي من $\{0,1, ..., m-1\}$.

نعرّف إذن التابع الموافق:

$$h_a \in \mathcal{H}$$

$$h_a(x) = \sum_{i=1}^{r} a_i x_i \bmod m$$

$$H = \bigcup_a h_a$$

لاحظ أن $\mathcal{H}$ تحوي m^{r+1} عنصراً. ويمكن البرهان أن $\mathcal{H}$ هي مجموعة عامة من توابع التقطيع.

9-5 العنونة المفتوحة

نخزن في العنونة المفتوحة (Open Addressing) جميع العناصر في جدول التقطيع نفسه. لهذا كل خانة من الجدول تحوي إمّا عنصراً وإمّا القيمة nil. نختبر في عملية البحث عن عنصر الخانات حتى نجد العنصر أو حتى يتأكد لنا أن العنصر غير موجود. ومن ثم لا يوجد سلاسل ولا عناصر مخزنة خارج الجدول.

من البديهي إذن أن هذا الجدول يمكن أن يمتلئ، وعندها لا يمكن إجراء إضافات بعد ذلك. لذا، عامل التحميل α لا يمكن أن يتجاوز الواحد. ومن الممكن استخدام المؤشرات هنا ضمن الجدول نفسه، لكن عوضاً عن ذلك نحسب سلسلة الخانات التي يجب اختبارها.

من أجل عملية إضافة عنصر نختبر الخانات حتى نصل إلى خانة فارغة نضع العنصر فيها. لكن عوضاً عن التقيد بالترتيب 0,1, ..., m-1 الذي يتطلب زمن بحث من مرتبة O(n)، نلجأ إلى ترتيب آخر يعتمد على قيمة المفاتيح نفسها. من أجل ذلك نمدّد تابع التقطيع ليحتوي رقم الخانة التي يجب اختبارها:

$h : U \times \{0, 1, ..., m-1\} \rightarrow \{0, 1, ..., m-1\}$

وعلى هذا، في حالة مفتاح k نحتاج إلى ما يسمى بسلسلة الاختبار (Probe Sequence):

$\langle h(k, 0), h(k, 1), ..., h(k, m-1)\rangle$

أيّ تبديل (Permutation) للسلسلة:

$\langle 0, 1, ..., m-1\rangle$

نستطيع الآن كتابة (بلغة الخوارزميات Pseudo Code) عمليات المعجم في جدول التقطيع T كما يلي، حيث نفترض أن العناصر لا تحوي سوى المفاتيح:

آ- إجرائية الإضافة

```
Hash_Insert(T, k)
    i:=0
    repeat
        j := h(k,i)
        if T[j] = nil then
            T[j] := k
            return j
        else
            i := i + 1
    until i = m
error "Hash table overflow"
```

ب- عملية البحث

وتختبر نفس سلسلة الاختبار في عملية الإضافة، مع فارق بسيط هو أن البحث يمكن أن يتوقف (بالفشل طبعاً) عندما يصل إلى خانة فارغة:

```
Hash_Search(T, k)
    i:=0
    repeat
        j := h(k,i)
        if T[j] = j then
            return j
        else
            i := i + 1
    until (T[j] = nil) or (i = m)
return nil
```

جـ- عملية الحذف

أصعب من ذلك، لأنه لا يمكن وضع nil مكان العنصر المحذوف لأن ذلك يؤدي إلى قطع سلسلة الاختبار. الحل هو وضع قيمة أخرى تشير إلى ذلك (Deleted مثلاً)، وتعدّل إجرائية البحث بحيث تستمر ولو وقعت على خانة محذوفة. أما إجرائية الإضافة فتعتبر الخانة Deleted خانة فارغة.

```
Hash_Delete(T, k)
    i:=0
```

```
    repeat
        j := h(k,i)
        if T[j] = j then
            T[j] := "Deleted"
            return j
        else
            i := i + 1
    until (T[j] = nil) or (i = m)
return "Element not found"
```

إنّ فرضية التقطيع المنتظم (Uniform Hashing) تصبح هنا:

كل مفتاح k يستطيع الحصول على أي تبديل ممكن من <0,1, ..., m-1> كسلسلة اختبار خاصة به، علماً أن عدد التباديل الممكنة هو m!. من الصعب إذن إيجاد تقطيع منتظم، لكن توجد بعض التقنيات التقريبية التي يمكن أن تحسب حتى m^2 تبديل في أفضل الحالات. سنعرض فيما يلي ثلاثاً من هذه التقنيات.

9-5-1 الاختبار الخطي

نفترض في الاختبار الخطي (Linear Probing) وجود تابع التقطيع الاعتيادي التالي:

$h' : U \rightarrow \{0, 1, ..., m-1\}$

نعرّف تابع التقطيع الخاص بالاختبار الخطي بالشكل:

$$h(k, i) = (h'(k) + i) \bmod m \; ; \; i = 0, 1, ..., m-1$$

فتصبح سلاسل الاختبار من الشكل:

$$\langle h'(k), h'(k) + 1, ..., h'(k) + m-1 \rangle$$

نلاحظ أن سلسلة الاختبار تعتمد على عنصر البداية فقط، ولذا سيكون هناك m سلسلة اختبار مختلفة فقط !. لذلك لا نعتبر الاختبار الخطي هو تقريباً جيداً للتقطيع المنتظم.

9-5-2 الاختبار التربيعي

نفترض في الاختبار التربيعي (Quadratic Probing) وجود تابع التقطيع الاعتيادي التالي:

$h' : U \rightarrow \{0, 1, ..., m-1\}$

نعرّف تابع التقطيع الخاص بالاختبار التربيعي بالشكل:

$h(k, i) = (h'(k) + c_1.i + c_2.i^2) \bmod m \; ; \; i = 0, 1, ..., m-1 \; ; \; c_1 \neq 0 \; ; \; c_2 \neq 0$

فتصبح سلاسل الاختبار من الشكل:

$\langle h'(k), h'(k) + c_1 + c_2, ..., h'(k) + c_1.(m-1) + c_2.(m-1)^2 \rangle$

نلاحظ أن سلسلة الاختبار تعتمد أيضاً على عنصر البداية فقط، لهذا سيكون أيضاً هناك m سلسلة اختبار مختلفة فقط !. لذلك لن نعتبر أيضاً أن الاختبار التربيعي هو تقريباً جيداً للتقطيع المنتظم.

9-5-3 التقطيع المزدوج

لزيادة عدد سلاسل الاختبار نلجأ إلى التقطيع المزدوج (Double Hashing). نفترض هنا وجود تابعي تقطيع:

$h1 : U \rightarrow \{0, 1, ..., m-1\}$
$h2 : U \rightarrow \{0, 1, ..., m-1\}$

نعرّف تابع التقطيع الخاص بالتقطيع المزدوج بالشكل:

$h(k, i) = (h1(k) + i.h2(k)) \bmod m \; ; \; i = 0, 1, ..., m-1$

تصبح سلاسل الاختبار من الشكل:

$\langle h1(k), h1(k) + h2(k), h1(k) + 2.h2(k), ..., h1(k) + (m-1).h2(k) \rangle$

نلاحظ هنا أن سلسلة الاختبار لا تعتمد على عنصر البداية وإنما على الإنزياحات عنها أيضاً (التي هي غير محدّدة سلفاً بسبب اعتمادها على قيمة h2).

مثال

ليكن لدينا جدولُ التقطيع T[0..12] of integer (أيّ m = 13) وتابعا التقطيع:

h1(k) = k mod 13 ;
h2(k) = 1 + k mod 11

إذا أردنا إضافة المفتاح k = 79 نجد أن سلسلة الاختبار الخاصة به هي:

<1, 4, 7, 10>

باعتبار أنّ الخانة الأولى فارغة فنضع المفتاح 79 في T[1].

سنضيف الآن المفتاح k = 98 فنجد أنّ سلسلة الاختبار الخاصة به هي:

<5, 4, 3, 2, 1, 0, 11, 10, ...>

باعتبار أنّ الخانة الخامسة فارغة نضع المفتاح 98 في T[2].

أخيراً سنضيف المفتاح k = 14 فنجد هنا أن سلسلة الاختبار الخاصة به هي:

<5, 9, 1>

نلاحظ هنا أن الخانة الخامسة حُجزت سلفاً للمفتاح 98، لذلك نضع المفتاح 14 في الخانة التاسعة.

لكن كيف نختار m وتوابع التقطيع h1, h2 المناسبة ؟

من الاختيارات الجيدة:

- m عدد أولي و h1 = k mod m , h2 = 1 + (k mod m')

  حيث m' = m-1 أو m' = m-2

- m من قوى 2 ونتحقق أن h2 يعطي دائماً قيماً فردية.

يحسّن التقطيع المزدوج من أداء التقطيع لأن عدد سلاسل الاختبار تصل في هذه الحالة إلى $\theta(m^2)$، ومن ثم يقترب أداء هذه الطريقة من أداء التقطيع المنتظم.

9-5-4 تحليل التقطيع بالعنونة المفتوحة

يتعلق تحليل العنونة المفتوحة أيضاً بعامل التحميل α (وهنا $1 \leq \alpha$). نناقش هذا التحليل مع افتراض وجود تقطيع منتظم باستعراض النظريات التالية (التي يُترك برهانها للقارئ المتمرس):

نظرية 1

إذا كان لدينا جدول تقطيع بعنونة مفتوحة وكان عامل التحميل $\alpha = n/m < 1$ ، فإن العدد المتوقع للاختبارات في حالة بحث غير موفق يساوي على الأكثر: $1/(1-\alpha)$، بافتراض وجود تقطيع منتظم.

نتيجة

إضافة عنصر إلى جدول تقطيع بعنونة مفتوح يتطلب وسطياً على الأقل $1/(1-\alpha)$ اختباراً، بافتراض وجود تقطيع منتظم.

نظرية 2

إذا كان لدينا جدول تقطيع بعنونة مفتوحة وكان عامل التحميل $\alpha = n/m < 1$ ، فإن العدد المتوقع للاختبارات في حالة بحث موفق يساوي على الأكثر (بافتراض وجود تقطيع منتظم):

$$\frac{1}{\alpha}\log_2\frac{1}{1-\alpha}+\frac{1}{\alpha}$$

9–6 تمارين

1– لنفترض أن طريقة التقسيم تستخدم تابع التقطيع : h(k) = k mod m و m = 2.p –1 حيث k هي سلسلة محارف ممثلة بعدد طبيعي بواسطة الجذر 128. برهن أنه إذا كانت x سلسلة محارف مشتقة من y بتبديل الأحرف فإن كلتا السلسلتين لهما نفس قيمة التقطيع. أعط مثالاً على تطبيق تكون فيه هذه الحالة غير مرغوبة.

2– عدّل خوارزميات الإضافة والبحث المذكورين في الفقرة 9–5 آخذاً بعين الاعتبار قيمة Deleted المذكورة في خوارزمية الحذف.

3– برهن أنه في حالة التقطيع المزدوج تكون سلسلة الاختبار:

<h(k, 0), h(k, 1), ..., h(k, m-1)> تبديل للسلسلة {0, 1, ..., m-1} إذا وفقط إذا كانت قيمة h2 أولية مع m.

4– ليكن لدينا جدول التقطيع T الذي يحتوي m خانة. ولدينا تابع تقطيع h(k) بحيث: h: K → {0, 1, ... m-1} (حيث K فضاء المفاتيح).

تمر عملية البحث عن مفتاح k بالخطوات التالية:

- احسب القيم: `i:= h(k) ; j:= 0 ;`
- ابحث في الخانة i عن المفتاح k. إذا وجدت المفتاح أو وجدت الخانة فارغة فأوقف البحث.
- احسب القيم: `j := j+1 ; i := (i+j) mod m ;`

بفرض أن m من قوى 2.

أ. برهن أن هذا المخطط هو تطبيق للتقطيع التربيعي، وذلك بتحديد الثوابت c_1 و c_2 في المعادلة:

$$h'(k, l) = (h(k) + c_1 l + c_2 l^2) \bmod m$$

ب. اكتب إجرائيات الحذف والإضافة في هذه الحالة.

ت. برهن أن خوارزمية البحث هذه تختبر جميع خانات الجدول في أسوأ الأحوال.

ث. ما مدى جودة تابع التقطيع هذا.

5- لتكن $H = \{h\}$ صفاً من توابع التقطيع، نقول عن H أنها عامة من الدرجة k (k-Universal) إذا كان في حالة كل سلسلة من k مفتاح مختلف: $\langle x1, x2, \ldots, xk \rangle$ ومن أجل كل تابع h من H، تكون السلسلة: $\langle h(x1), h(x2), \ldots, h(xk) \rangle$ أيّ سلسلة (بنفس الاحتمال) من الـ m^k سلسلة ممكنة من $\{0, 1, \ldots, m-1\}$.

أ. برهن أنه إذا كانت H هي عامة من الدرجة 2 فإنها عامة.

ب. برهن أن الصف H المشروح في الفقرة 9-4-4 ليس عاماً من الدرجة 2.

ت. إذا غيرنا تعريف H بحيث يحوي كل تابع ثابتاً b أي:

$$h_{a,b} = a.x + b$$

فإن H تصبح عامة من الدرجة 2.

6- أعد حل مسألة المكنز (المسألة الرابعة في تمارين الفصل السادس) بحيث تستعيض عن جدول الأحرف بجدول تقطيع. ناقش توابع التقطيع المناسبة لاختصار زمن عمليات المعجم على المكنز.

وقل رب زدني علما
جامعة دمشق
Damascus University

المراجع المستخدمة في الكتاب

1. ARSAC J.
Les Bases de la Programmation
Dunod Informatique 1983

2. BAASE S.
Computer Algorithms: Introduction to Design and Analysis
Addison-Wesley 1983

3. BERLIOUX B., BIZARD Ph.
Construction, Preuve et Evaluation des Programmes
Dunod Informatique 1983

4. CORMEN T. H., LEISERSON C. E., RIVEST R. L.
Introduction to Algorithms
The Massachusetts Institute of Technology, 1990

5. FROIDEVAUX C., GAUDEL M. C., SORIA M.
Types de Données et Algorithmes
McGraw- Hill 1990

6. GIROUD X.
Conception par Objet
University of Grenoble I, Thesis 1991

7. HOROWITZ E., SAHNI S.
Fundamentals of Data Structures
Computer Science Press, Inc., 1981

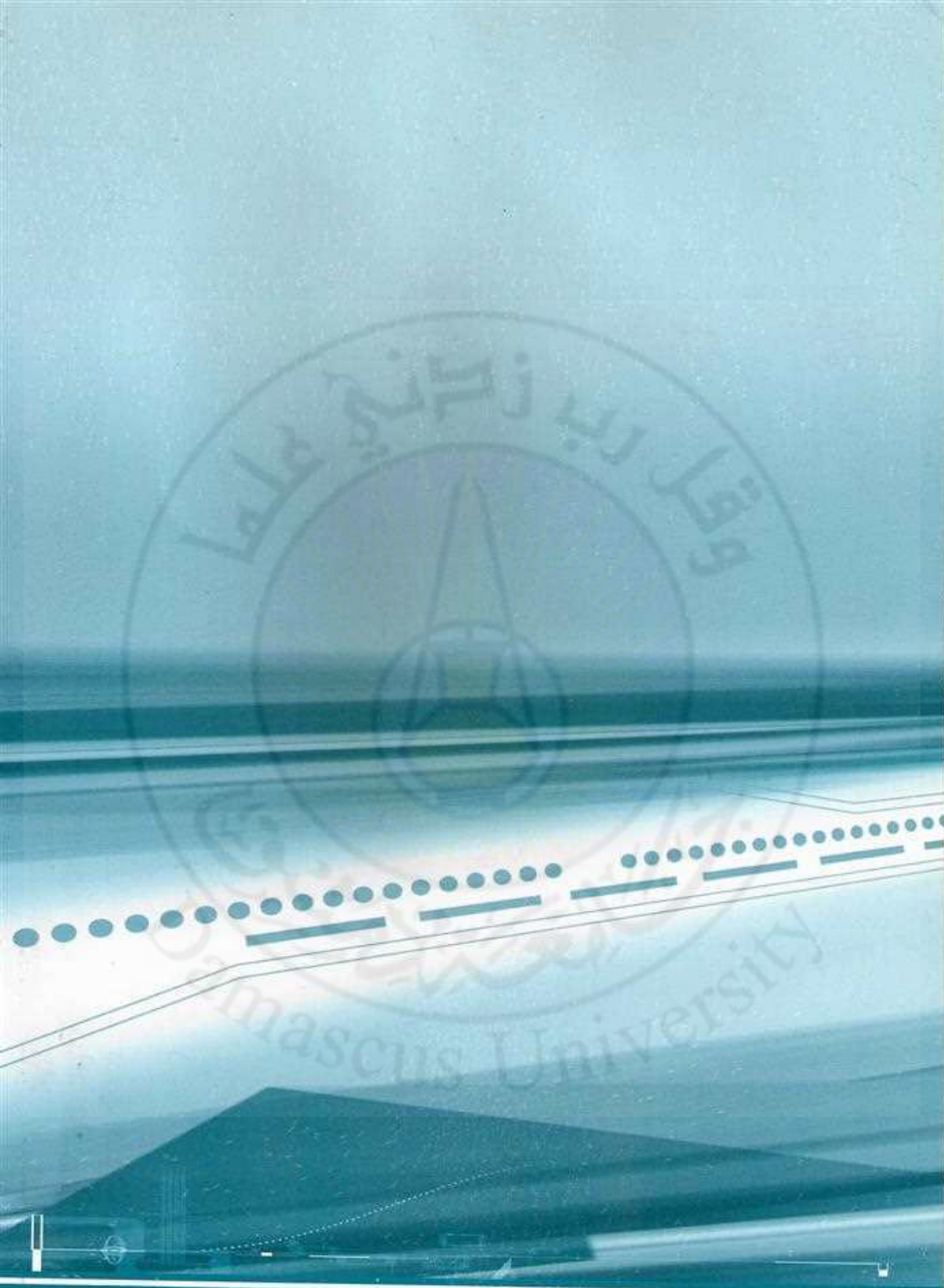
8. KIEBURTZ R. B.
Structured Programming and Problem Solving with Pascal
Prentice-Hall, 1978

9. KNUTH D. E.
The Art of Computer Programming
Volume 1: Fundamental Algorithms
Addison-Wesley 1973

10. TENENBAUM A. M., AUGENSTEIN M. J.
Data Structures Using Pascal
2nd. Edition, Prentice-Hall, 1986

11. WIRTH N.
Algorithms + Data Structures = Programs
Printice-Hall, 1975

12. N. H. XUONG
Mathématiques Discrètes et Informatique
Masson 1992



سعر المبيع : (٢٤٠) ل.س

مطبعة جامعة دمشق