الخوارزميات التراجعية

1. مقدمة

الخوارزميات التراجعية (Backtracking Algorithms) هي خوارزميات عودية تعتمد طريقة "التجريب والخطأ" (Trial-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 الخوارزمية اعتماداً على هذه الخطوة. عندما يتأكد لنا أن اختيارنا لا يقود إلى الحل النهائي، أو يؤدي إلى طريق مسدود، نَعدِل 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.

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 إذا كانت هذه الحركة تؤدي 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 الخوارزمية سنحدد بنى المعطيات المستخدمة ومعاملات الإجرائية. نمثل الرقعة بمصفوفة مربعة h نعرفها بالشكل:

```
Int n=8 ;
int[][] h = new int[n][n];
```

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 الخوارزمية فيه: (X,Y)
- رقم الخطوة: i
- متحول خرج q يعطي نتيجة المناقشة: نجاح أو فشل

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

i < n2

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

```
((1<=u)&&(u <= n) && (1<=v)&&(v <= n)) && (h[u][v] == 0)
```

أخيراً نُدخل متحولاً موضعياً آخر q1 لمراقبة الاستدعاء العودي للإجرائية، فتصبح الإجرائية كالتالي:

```
procedure try (i: integer; x, y: index; var q: boolean);
var
     u,v : integer; q1 : 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,q1);
                              if  q1 then
                                   h[u,v] := 0
                         end
                    else
                         q1 := true
               end
     until q1 or (no more candidates);
     q :=q1
end;
```

المرحلة الأخيرة من تبسيط الخوارزمية 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 التالي.

		3		2			
	4				1		
			X				
	5				8		
		6		7			

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

```
int[] a=new int [n];
int[] b=new int [n];
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;
```

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 فيها 25 مربعاً، حيث يبدأ الحصان من الموقع h[1, 1].

```
package Backtracking;
import java.*;
import java.io.*;
import java.math.*;
public class knightstour
{
      public knightstour() {
      }
      public static boolean try1( int i,int x, int y ,boolean q)
      {
  int u,v,n=8 ; boolean q1=false ;
  {

    int[] s = { 1, 2, 3, 4, 5};
    int[][] h = new int[n][n];
    int[] a = new int[n];
    int[] b = new int[n];
    for (int k=0;k<8;k++)
    {
      if (q1 )
        break;

      q1 = false;
      u = x +a[k];
      v = y+b[k] ;

      if  ((1<=u)&&(u <= 5) && (1<=v)&&(v <= 5))
        if (h[u][v] == 0) {
          h[u][v] = i;
```

```
          if (i < 25)
          {
           q1= try1(i + 1, u, v,q1);
            if (!q1)
              h[u][v] = 0;
          }
          else
            q1 = true;
        }
    }
    q = q1;
  }
      return q;
      }

public static void main (String argv[])
{
 int[] s ={1,2,3,4,5};
 int n=8;
 int[][] h=new int [n][n];
int[] a=new int [n];
int[] b=new int [n];
boolean q=true;
a[0] =  2; b[0] =  1;
a[1] =  1; b[1] =  2;
a[2] = -1; b[2] =  2;
a[3] = -2; b[3] =  1;
a[4] = -2; b[4] = -1;
a[5] = -1; b[5] = -2;
a[6] =  1; b[6] = -2;
a[7] =  2;b[7] = -1;

for (int i = 0 ;i<n ;i++)
  for (int j = 0 ;j<n ;j++)
      h[i][j] = 0;
h[1][1] = 1;
boolean solution=false;
solution =try1(2,1,1,solution);
if (solution)
  for (int i=1 ;i< n ;i++)
  {
   for (int j = 1; j < n; j++)
     System.out.print(h[i][j] + " ");
   System.out.println();
  }
else
  System.out.println ("No Solution");

}
}
```

يبين الشكل التالي 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

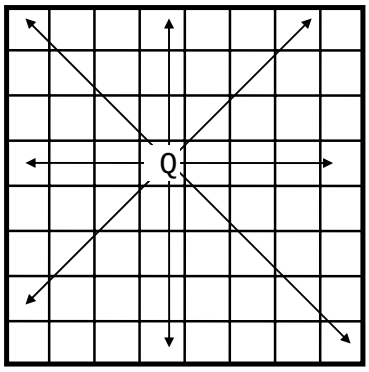
5	25	18	3	12	23
4	8	13	24	17	4
3	19	2	7	22	11
2	14	9	20	5	16
1	1	6	15	10	21
	1	2	3	4	5

3. مسألة الوزراء الثمانية

الغرض من هذه المسألة هو توزيع ثمانية وزراء على رقعة شطرنج، بحيث لا يكون فيها أيٌّ واحد مهدداً للآخر. بحسب المخطط العام للخوارزميات التراجعية فإن 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 التالي. 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 لابد من مناقشة عميقة لبنية المعطيات المستخدمة، بحيث تُحدد بنية تساعد في إجراء الاختبارات اللازمة عند تقرير وضع الوزير في مربع معين. في البداية يمكن تعريف رقعة الشطرنج بأنها مصفوفة مربعة، لكن هذا التمثيل سيؤدي إلى اختبارات صعبة لنعرف أيكون مربع ما آمناً (غير مهدد) أم لا.

إن ما يهمنا هو موقع كل وزير ضمن العمود الموافق، وأن نعرف أيحوي سطرٌ معين أو قطر معين وزيراً أم لا. لذا سنستخدم بنى المعطيات التالية:

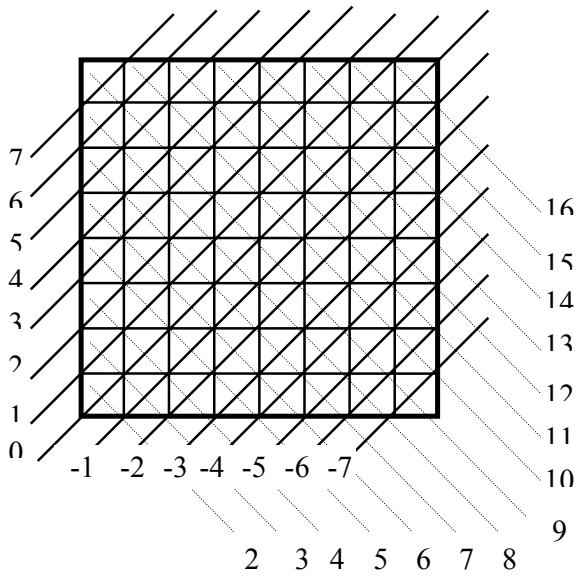
```
x : array[1..8]       of integer;
a : array[1..8]       of boolean;
b : array[2..16]      of boolean;
c : array[-7..7]      of boolean;
```

حيث:

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

يبين الشكل (4) طريقة ترقيم الأقطار

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



بعد تحديد بنى المعطيات المستخدمة نستطيع تفصيل الخوارزمية، فنلاحظ أن المواقع الممكنة للوزير رقم i هي كل مربعات العمود رقم i (عددها ثمانية). لاختبار هذه المواقع نستعمل عداداً 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:

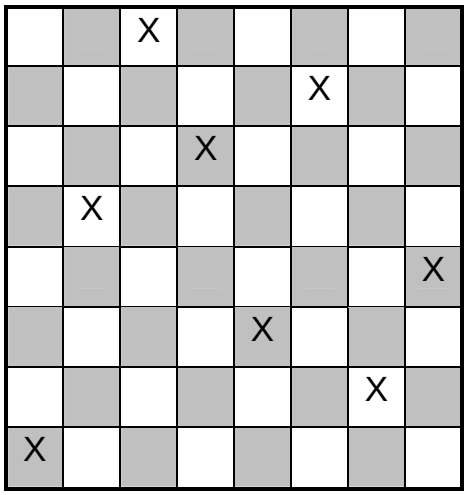
```
a[j]  := true;
b[i+j]:= true;
c[i-j]:= true;
```

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

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

تنتهي بذلك عمليات تفصيل الخوارزمية ونورد فيما يلي النص الكامل للبرنامج. يعطي هذا البرنامج الحل الممثل بالشكل التالي:

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



```
package Baktaking;
import java.*;
import java.io.*;
import java.math.*;


public class Queen {
  public Queen() {
  }

  int i;
  boolean q;
  boolean[] a = new boolean[7];
  boolean[] b = new boolean[16];
  boolean[] c = new boolean[14];
  int[] x = new int[7];

  public static boolean try1(int i)
  {
  boolean q1=false;
  boolean[] a = new boolean[7];
  boolean[] b = new boolean[16];
  boolean[] c = new boolean[14];
  int[] x = new int[7];
    for (int j=0;j<7;j++)
    {
      boolean q = false;
      if (a[j] && b[(i + j)-2] && c[(i - j)+7])
      {
        x[i] = j;
        a[j] = false;
        b[i + j] = false;
        c[i - j] = false;
        if (i < 8) {
          q = try1(i + 1);
          if (!q) {
            a[j] = true;
            b[i + j] = true;
            c[i - j] = true;
          }
```

```
        }
        else
          q = true;
      }

      q1=q;
      if (q1) break;
    }
    return q1 ;
  }

  public static void main(String argv[])
  {
  int i;
  boolean q;
  boolean[] a = new boolean[7];
  boolean[] b = new boolean[16];
  boolean[] c = new boolean[14];
  int[] x = new int[7];

    for (i=0; i <7; i++)
      a[i] = true;
    for (i = 2; i < 16; i++)
      b[i] = true;
    for (i =0; i <=13; i++)
      c[i] = true;
    q = try1(1);
    if (q)
        {
      for (i =0 ;i< 16 ;i++)
        System.out.print(x[i]+" ");
      System.out.println();
    }
    else
        System.out.println("No Solution");
 }
}
```

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 نسجل 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:

```
public static boolean try1(int i)
  {
  boolean q,q1=false;
  boolean[] a = new boolean[7];
  boolean[] b = new boolean[16];
  boolean[] c = new boolean[14];
  int[] x = new int[7];
    for (int j=0;j<7;j++)
    {
      q = false;
      if (a[j] && b[(i + j)-2] && c[(i - j)+7])
      {
        x[i] = j;
        a[j] = false;
        b[i + j] = false;
        c[i - j] = false;
        if (i < 8) {
                        q = try1(i + 1);
                    }
          else
                    {
                        for (i =0 ;i< 16 ;i++)
                              System.out.print(x[i]+" ");
                        System.out.println();
                    }
        a[j] = true;
        b[i + j] = true;
        c[i - j] = true
      }
      q1=q;
    }
    return q1 ;
  }
```

بهذا نستطيع إيجاد كافة الحلول وعددها 92 حل، من بين هذه الحلول يوجد اثنا عشر حلاً مستقلاً (لا ينتج أحدها عن الآخر بتدوير الرقعة أو بالتناظر) يبينها الشكل التالي.

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

4. مسألة الخيار الأمثل

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

لإيجاد الحل الأمثل لمسألة معينة يجب إيجاد جميع الحلول الممكنة بحيث نحتفظ دوماً بحل واحد يعتبر أمثلياً بحسب معيار معين. إذا افترضنا أنّنا نستطيع مقارنة الحلول بواسطة تابع موجب F(S) حيث (S أحد الحلول)، عندئذ يمكن الحصول على الحل الأمثل بتعديل المخطّط 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 مثالاً لمسألة الحل الأمثل مسألة هامة تتلخّص بإيجاد الاختيار الأمثل من مجموعة عناصر معطاة. تجري عملية بناء الخيارات الممثّلة 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، ثُم ننتقل إلى العنصر التالي عودياً.

عند فحص فائدة عنصر معين نستنتج أحد أمرين: إما أن نضم هذا العنصر إلى الخيار الآتي الذي نحن بصدد بنائه، أو نستبعد هذا العنصر ونتابع بناء اختيارنا دونه. ويجب مناقشة هذين الاحتمالين كلٍ 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 اختياراً ممكناً (عدد أجزاء مجموعة مؤلّفة من n عنصر) لذلك يجب أن نستفيد من معايير القبول في الحد كثيراً من هذا العدد. سنوضح طريقة تقليص هذا العدد في ضوء مثال محدد.

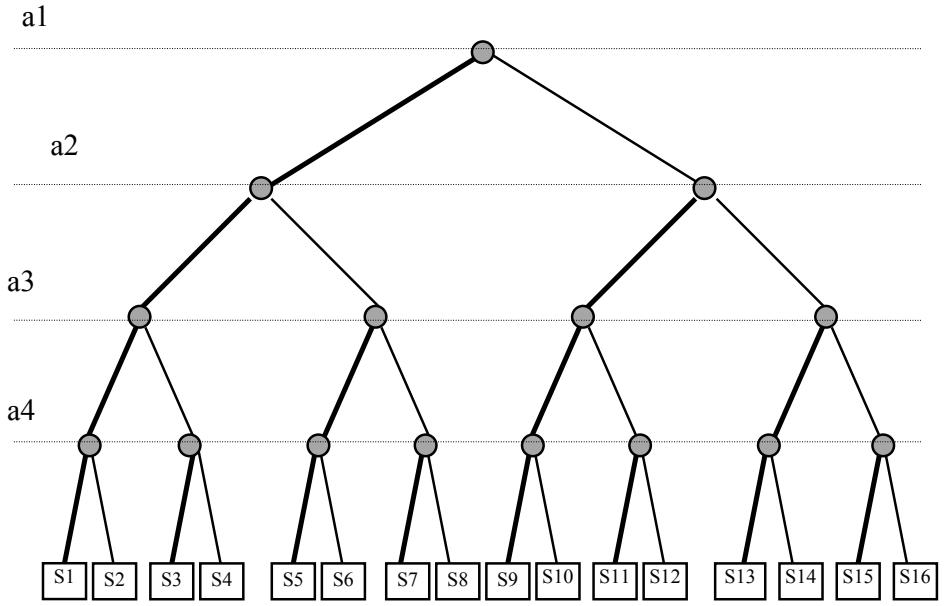
لتكن A = {a1, a2, . . ,an} مجموعة من الأشياء لكل منها وزن W وقيمة V، ولنفترض أننا نريد بناء مجموعة جزئية من A بحيث يكون مجموع أوزان عناصرها أقل من حد معين، ومجموع أسعارها أعظمياً. تصادف هذه المسألة أي مسافر يستعد للسفر بالطائرة، ويقوم بإعداد حقيبته، وعليه اختيار مجموعة صغيرة من الأشياء لا يتجاوز وزنها 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 الكلّي عن 30 كغ وتكون قيمتها أعظمية.

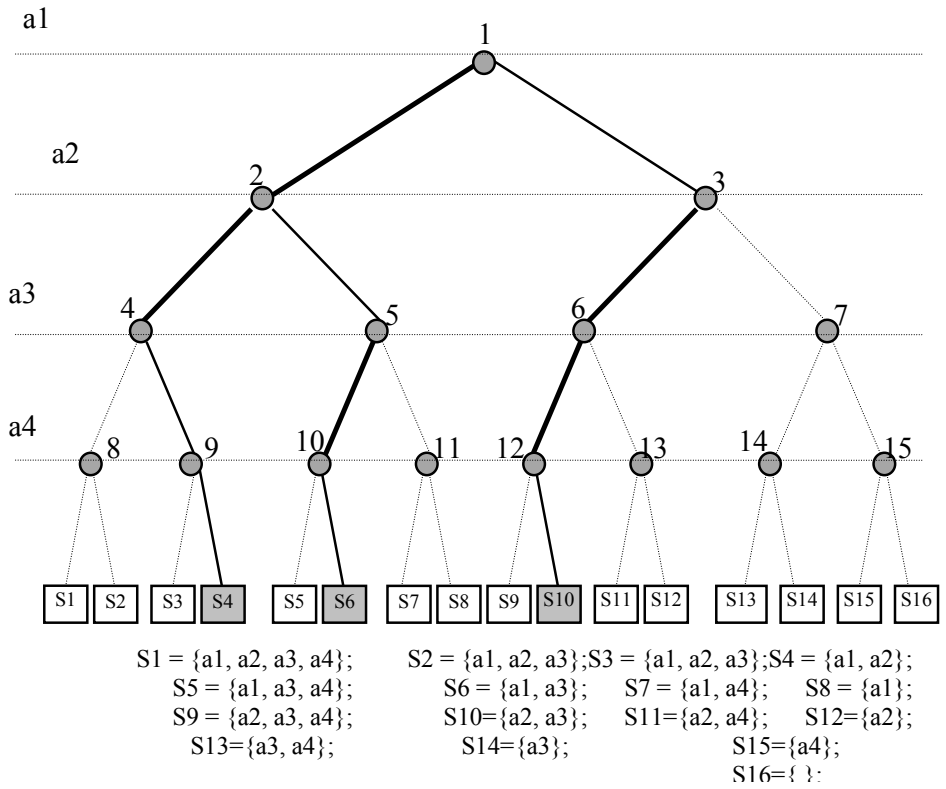
يمكن تمثيل تنفيذ خوارزمية البحث عن أفضل اختيار بالشجرة المبينة بالشكل التالي، تمثّل كل عقدة داخلية في هذه الشجرة عملية مناقشة ضم أو استبعاد عنصر. (رمزنا إلى ناتج ضم العنصر بخط عريض). وتمثّل الأوراق الخيارات الناتجة أو المجموعات الجزئية للمجموعة A (عددها 16).



من دراسة هذه الحلول يتبين أنّه يجب استبعاد الحلول S1, S2, S3, S5, S9 لأنها لا تحقّق شرط الوزن. أما بقية الحلول فيجب حساب قيمة كل منها، ويستطيع القارئ التحقق من أن .

هو أفضل حلّ، ويعطي القيمة 45.

لنفكّر قليلاً في استنتاج طريقة لتقرير ضم عنصر إلى المجموعة الجزئية التي نحن بصدد بنائها أو عدم ضمه. نرقّم العقد الداخلية للشجرة السابقة كما في الشكل التالي. تُشبه عملية بناء المجموعات الجزئية التجول ضمن الشجرة السابقة بحيث نبدأ من العقدة رقم 1 ثم إلى العقدة رقم 2 ثم 4 ثم 8 وهكذا.



عندما نصل إلى العقدة 4 مثلاً، 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 الحلّين S1 و S2, لأن ضم a3 إلى المجموعة S سيجعل الوزن الكلّي 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 تُستبعد الحلول من S11 إلى S16.

للتعبير عن طريقة المحاكمة السابقة على شكل خوارزمية نعتبر أنّنا نقوم ببناء المجموعة S أي ستحوي حلاً مقبولاً. في كل خطوة سنناقش ضم عنصر واحد إلى S وسنفترض أن العناصر مرقّمة من 1 إلى n .

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

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 حالياً في S والتي يمكن ضمها مستقبلاً يمكن أن يتجاوز أفضل قيمة وصلنا اليها حتى الآن. عندما تكون i = n ويكون الاستبعاد مقبولاً، 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 التالي نتائج تنفيذ هذا البرنامج على مجموعة تحوي 12 عنصراً، حيث أعطينا الأوزان العظمى قيماً تقع بين 10 و 150 كيلوغراماً.

```
package optimalsolution;
import java.*;
import java.io.*;
public class OptimalSolution {
  int n=12;int i; object[] a=new object[12];
  int  index ,limw=0, totv, maxv,w1, w2, w3;
```

```
int [] opts=new int[12];
int [] s=new int[12];
char [] z=new char[2];
private static Object v;
public class object
   {
   int v;
   int w;
   }

public OptimalSolution() {
}

public static void try1(int i,int tw,int av )
{
    object[] a=new object[12];
    int n=12;
    int [] opts=new int[12];
    int [] s=new int[12];
    int  index ,limw=0, totv, maxv=0,w1, w2, w3;
    int av1 ;
    if ((tw + a[i].w )<= limw)
      {
       s[i]=i;
       if (i < n)
          try1(i + 1, tw + a[i].w, av);
       else
          if ( av > maxv )
           {
             maxv = av;
           opts = s;
         }
       s[i]=0;
      }
    av1 = av -a[i].v;
    if (av1>maxv)
      if(i<n)
         try1(i + 1, tw, av1);
      else
       {
         maxv = av1;
         opts = s;
       }
}
/*********************/
/*      MAIN         */
/*********************/
public static void main(String argv[])
{
  try {
    char [] z=new char[2];
    int n = 12;
    int w1,w2,w3;
    int totv = 0;
    int [] opts=new int[12];
```

```
      int [] s=new int[12];
      object[] a = new object[12];
      for (int i = 0; i < n - 1; i++)
      {
        a[i].w = System.in.read();
        a[i].v = System.in.read();
        totv = totv + a[i].v;
      }
      try{
         w1 = System.in.read();
         w2 = System.in.read();
         w3 = System.in.read();


        char c='*';
        char q=' ';
        z[0] = q; z[1] = c;



           System.out.print("Weight ");
           for (int i =0 ;i< n ;i++)
           System.out.print( a[i].w +"  ");
           System.out.println();
           System.out.print("Value");
           for(int i =0;i<n;i++)
           System.out.print(a[i].v +"   ");
          System.out.println();
           while(w1<w3)
           {
           int limw = w1;
           int  maxv = 0;
           for (int i =0 ;i< n ;i++)
           {
             s[i]= 0;
             opts[i] = 0;
           }
           try1(1, 0, totv);
           System.out.print(limw +"    ");
            for (int i =0 ;i< n ;i++)
              if (opts[i]==i)
            System.out.print(" ");
           System.out.println();
           w1= w1 + w2;
           }
         }catch (Exception err) {
           }


    }
    catch (Exception err) {
           }
    }

}
```

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	*	*	*	*		*	*		*	*	*	*

5. تمرينات

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 زوجاً (أي ثنائية) وبقي على الأقل رجل وامرأة

ويفضل كلاهما الآخر على الشريك الذي اختير بنتيجة الخوارزمية نقول عن الزواج إنه ليس مستقراً. أما إذا لم يوجد مثل هذه الحالة فنقول عن هذا الزواج إنه مستقرٌ.

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

نفترض أيضاً 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 أيضاً هـ و 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 برنامجاً كاملاً يحل المسألة من أجل إيجاد الثنائيات التي تحقق شرط الزواج المستقر.

2- مسألة محطات القطار

إحدى شركات القطارات تُخدم 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 أيضاً 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 هـ و 10 والـ زمن الأصغري للتبديل بين قطارين لا يقل عن 15 دقائق).

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

ت. اكتب برنامجاً كاملاً يحل المسألة.