الفصل الرابع : الخوارزميات التراجعية
Backtracking Algorithms

4-1 مفهوم التراجعية

الخوارزميات التراجعية هي خوارزميات عودية 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 الإمكانيات المتاحة للخطوة التالية, ثم ندرس هذه الإمكانيات 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, نعدل عن هذه الخطوة. تشبه هذه العملية بناء سجل أخطاء يفيد في تجنب الوقوع في الخطأ مرتين.

إن المخطط العام للخوارزميات التراجعية الموضح بالجدول(4-1)(حيث كتب بـــلغة الترميز الزائف pseudo code , أي دون تحديد لغة برمجية معينة, بل تم وضع الشكل العام للخوارزمية مع الشرح باللغة الإنكليزية ودون تفاصيل دقيقة):

الجدول(4-1): المخطط العام للخوارزميات التراجعية

```
void try()
{
 initialize selection of candidates;
 do{
    select next candidate;
    if (acceptable){
       record it;
       if (solution incomplete){
          try next step;
          if (not successful)
            cancel recording;
       }
    }
 }while (not successful)and(remains
      of candidates);
}
```

```
void  try()
{
  تهيئة الإمكانيات الممكن ترشيحها
  do{
       اختيار المرشح التالي
       if (مقبول){
            تسجيل الخطوة الحالية
            if (الحل غير مكتمل){
                 حاول الخطوة التالية
                 if (غير ناجحة)
                      إلغاء تسجيل آخر خطوة
            }
       }
  }while (الحل غير مكتمل)and
            (وجود مرشحين أخرين);
}
```

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

2-4 مسألة جولة حصان الشطرنج - Knightstour :

لدينا رقعة شطرنج فيها n×n مربعاً, نضع حصاناً في موقع معين $<X_0,Y_0>$ حيث إن: $0 \le X_0 \le n-1$ و $0 \le Y_0 \le n-1$ 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 الإمكانيات 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:

```
void  try next move()
{
 initialize selection of moves;
 do{
    select next candidate from list of next moves;
    if (acceptable){
       record move;
       if (board not full){
          try next move;
          if (not successful)
             erase previous recording;
       }
    }
}while (board not full)and(there are more candidates/moves yet);
}
```

لتفصيل الخوارزمية، سنحدد بنى المعطيات المستخدمة والمعاملات الإجرائية. نمثل الرقعة بمصفوفة مربعة h نعرفها بالشكل:

```
const int n=5;
int   h[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 الخوارزمية:

- الموقع الذي نريد تطبيق الخوارزمية 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 إحدى المواقع الممكنة (حسب طريقة حركة الحصان المعروفة على الشكل L) فإن العبارة "مقبول" (acceptable) يمكن التعبير عنها بالشكل:

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

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 (4-1) 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[8] , b[8];

		3		**2**			
	4				**1**		
			X				
	5				**8**		
		6		**7**			

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

كما نستطيع استخدام عداد k لترقيم إمكانيات 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 الشطرنج, حيث أنه يبدأ الحصان من الموقع h[x,y] .

```
#include <iostream.h>
#include <iomanip.h>   // --> setw()

const int n=5;     //  number of lines/columns
int  i,j,x,y;     //  x,y = start coordinates
char q='n';       // no = the solution is incomplete
```

```
int  a[8],b[8];
int  h[n][n];

void try1(int i,int x,int y,char &q);
/*************************************************************
***************/
void main()
{
  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;
  cout<<"Enter x="; cin>>x; cout<<"Enter y="; cin>>y;

  for (i=0;i<n;i++)
          for (j=0;j<n;j++)
             h[i][j]=0;
  h[x][y]=1;       // first step (i=1) is fixed
  try1(2,x,y,q);
  if (q=='y')
     for (i=0;i<n;i++){
         for (j=0;j<n;j++)
              cout<<setw(5)<<h[i][j];
     cout<<endl;
     }
  else
     cout<<"No solution !\n";
}
/*************************************************************
***************/
void try1(int i,int x,int y,char &q)
{
  int u,v,k=-1;



 do{
```

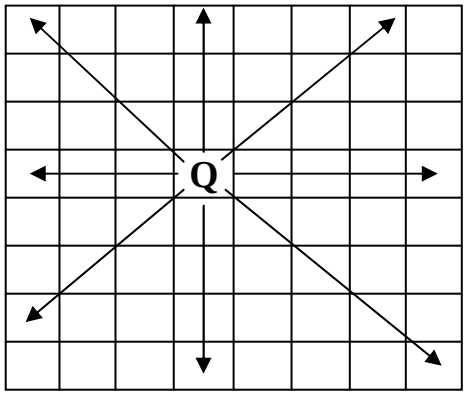
```
        k++;
        u=x+a[k];  v=y+b[k];
        if (u>=0 && u<n && v>=0 && v<n && h[u][v]==0){
       h[u][v]=i;
          if (i<n*n){
                try1(i+1,u,v,q);
                if (q=='n')
                   h[u][v]=0;
          }
          else
         q='y'; //  yes = the solution is complete
        }
 }while (q=='n' && k<7);
}
```

3-4 مسألة الوزراء الثمانية - Eight-Queens

الغرض من هذه المسألة هو توزيع ثمانية وزراء على رقعة شطرنج, بحيث لا يكون فيها أي واحد مهدداً للآخر. بحسب المخطط العام للخوارزميات التراجعية فإن شكل الإجرائية هو:

```
void  try(int i)
{
 initialize selection of positions for i-th queen;
 do{
    make next selection;
    if (safe){
      setqueen;
      if (i<7){
        try (i+1);
        if (not successful)
          remove queen;
      }
    }
 }while(not successful && there are more positions yet);
}
```

نعلم من قواعد الشطرنج أنه يمكن للوزير أن يهدد جميع المربعات الموجودة في العمود أو السطر أو القطر نفسه, كما يبين ذلك الشكل (4-2). 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 . لحل هذه المسألة على الرقعة لا بد من



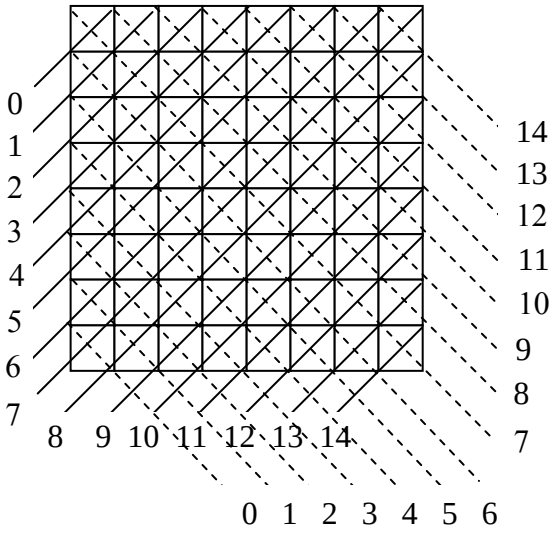
الشكل(4-2): حركات الوزير على رقعة الشطرنج

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 سطر معين أو قطر معين وزيراً أم لا. لذا سنستخدم بنى المعطيات التالية:

```
int   x[8];        //    i تحوي رقم السطر الذي يوضع فيه الوزير رقم
                          x[i]
char  a[8];        //                تعني أن السطر j لا يحوي أي وزير
                                    a[j]=='y'
 char  b[15];      //      تعني أن القطر المباشر رقم j لا يحوي أي وزير
                                    b[j]=='y'
char  c[15];       //       تعني أن القطر المتعامد رقم j لا يحوي أي وزير
                                    c[j]=='y'
```

يبيّن الشكل(4-3) طريقة ترقيم أقطار رقعة الشطرنج:



الشكل(4-3): طريقة ترقيم أقطار رقعة الشطرنج

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 7.

تكافئ عملية وضع الوزير رقم i في السطر j تنفيذ التعليمات التالية:

```
x[i]=j;
a[j]='n';
b[7+i-j]='n';
c[i+j]='n';
```

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

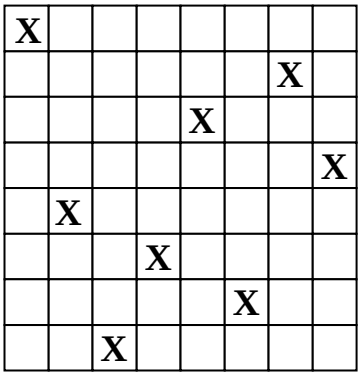
```
a[j]='y';
b[7+i-j]='y';
c[i+j]='y';
```

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

```
(a[j]=='y' && b[7+i-j]=='y' && c[i+j]=='y')
```

يعطي البرنامج التالي الحل للمسألة الممثل بالشكل(4-4):

x = [0 , 4 , 7 , 5 , 2 , 6 , 1 , 3]



الشكل(4-4): إحدى حلول مسألة الوزراء الثمانية

```
#include <iostream.h>
#include <iomanip.h>

int  i;
char q='n';       //  no, there is no solution
char a[8];        //  lines
char b[15];       //  direct diagonals
char c[15];       //  indirect diagonals
int  x[8];        //  solution = lines

void try1(int i,char &q);
void print();
/**************************/
void main()
{
 for (i=0;i<8;i++)  a[i]='y';  // all lines are safe
 for (i=0;i<15;i++) b[i]='y'; //all direct diagonals are safe
 for (i=0;i<15;i++) c[i]='y';//all indirect diagonals are safe
  try1(0,q);
  if (q=='y')
     print();
```

```
 else
   cout<<"No solution !\n";

}
/************************************/
void try1(int i,char &q)
{
 int j=-1;
 do{
        j++;
        if (a[j]=='y' && b[7+i-j]=='y' && c[i+j]=='y'){
           x[i]=j;
           a[j]='n';
           b[7+i-j]='n';
           c[i+j]='n';
           if (i<7){
                  try1(i+1,q);
                  if (q=='n'){
                         a[j]='y';
                         b[7+i-j]='y';
                         c[i+j]='y';
                  }
           }
        else
           q='y';
        }
 }while(q=='n' && j<7);
}
/************************************/
void print()
{
 for (int k=0;k<8;k++)
        cout<<setw(5)<<x[k];
 cout<<endl;
}
/************************************/
```

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 الإمكانيات المتاحة عند الوصول إلى حل, وإنما نسجل هذا الحل ونتابع تجريب الإمكانيات الأخرى.

يصبح مخطط الخوارزميات التراجعية التي توجد جميع الحلول كالتالي:

```
void try(int i)
{
 for (int k=0;k<m-1;k++){
     select k-th candidate;
     if (acceptable){
        record it;
        if (i<n)
           try(i+1);
        else
           print solution;
     }
 }
}
```

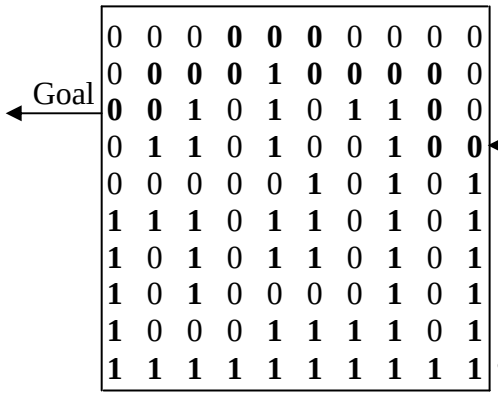
في حالة مسألة الوزراء الثمانية تصبح إجرائية البحث عن الحلول كالتالي:

```
void try2(int i)
{
 for (int j=0;j<8;j++){
     if (a[j]=='y' && b[7+i-j]=='y' && c[i+j]=='y'){
            x[i]=j;
            a[j]='n';
            b[7+i-j]='n';
            c[i+j]='n';
            if (i<7)
           try2(i+1);
            else
           print();
            a[j]='y';
            b[7+i-j]='y';
            c[i+j]='y';
         }
 }
}
```

بهذا نستطيع إيجاد كافة الحلول وعددها 92 حلاً, من بين هذه الحلول يوجد 12 حلاً مستقلاً (لا ينتج أحدها عن الآخر بتدوير الرقعة أو بالتناظر).

4-4 مسألة المتاهة

يمثل الشكل (4-5) متاهة, حيث إن الواحدات تمثل جدران المتاهة والأصفار تمثل الممرات الممكن سلوكها عبر المتاهة.



وبالتالي يمكن الانتقال ضمن المتاهة من موضع إلى موضع آخر يوجد فيه صفر.

من أجل إيجاد المخرج من هذه المتاهة ننطلق من المدخل Start ونقوم بالسير قدماً دون أن نرفع يدنا فوق الأصفار, ونستمر بالمشي حتى نصل إلى المخرج Goal.

لنكتب برنامجاً يعتمد على الخوارزمية التراجعية try1 والتي تقوم بإيجاد أول طريق واصل ما بين المدخل والمخرج, حيث إنه يجب في نهاية البرنامج طباعة الشكل السابق نفسه لكن مكان الأصفار الواقعة على الطريق يجب وضع رقم الخطوة الحالية لمعرفة تسلسل الخطوات.

الشكل (4-5): متاهة

ثم نعمم المسألة لإيجاد جميع الحلول الممكنة بتابع ثانٍ try2 , ومن ثم الحل الأمثل فقط في تابع ثالث try3.

```
#include <iostream.h>
#include <conio.h>

const int n=10;    //  number of lines/columns in labyrinth

int h[n][n]={{0,0,0,0,0,0,0,0,0,0}, {0,0,0,0,1,0,0,0,0,0},
        {0,0,1,0,1,0,1,1,0,0}, {0,1,1,0,1,0,0,1,0,0},
        {0,0,0,0,0,1,0,1,0,1}, {1,1,1,0,1,1,0,1,0,1},
        {1,0,1,0,1,1,0,1,0,1}, {1,0,1,0,0,0,0,1,0,1},
        {1,0,0,0,1,1,1,1,0,1}, {1,1,1,1,1,1,1,1,1,1}};
int optS[n][n];  // optimal solution
```

```
int a[4]={-1,0,1,0};
int b[4]={0,1,0,-1};
int X_start,Y_start,X_goal,Y_goal,minL=0; // minL= length of optimal
solution (=shortest way)
char q='n',c; // q=='n' → no, the solution is incomplete

void try1(int i,int x,int y,char &q);  // one solution
void try2(int i,int x,int y);         // all solutions
void try3(int i,int x,int y);        // optimal solution
void print(int h[n][n]);
void reset();    // reset labyrinth for other solutions
/*********************************************/
void main()
{
  cout<<"Enter X-Start = "; cin>>X_start;
  cout<<"Enter Y-Start = "; cin>>Y_start;
  cout<<"Enter X-Goal = "; cin>>X_goal;
  cout<<"Enter y-Goal = "; cin>>Y_goal;
  do{
     reset();
     h[X_start][Y_start]=2;
          cout<<"One Solution      -->  Press 1\n";
          cout<<"All Solutions     -->  Press 2\n";
          cout<<"Optimal Solution  -->  Press 3\n";
          cout<<"Exit Program      -->  Press 4\n";
          c=getche();

          if (c=='1'){
        try1(3,X_start,Y_start,q);
        if (q=='y')
           print(h);
        else
           cout<<"No solution";
     }
     else
     if (c=='2')
        try2(3,X_start,Y_start);
     else
     if (c=='3'){
        try3(3,X_start,Y_start);
```

```
        print(optS);
    }
    getch();
  }while(c!='4');
}
/*********************************************/

void try1(int i,int x,int y,char &q)  // only one solution
{
  int u,v,k=-1;
  do{
    ++k;
    u=x+a[k];  v=y+b[k];
    if (u>=0 && u<n && v>=0 && v<n && h[u][v]==0){
       h[u][v]=i;
       if (u==X_goal && v==Y_goal)
           q='y';
       else{
           try1(i+1,u,v,q);
           if (q=='n')
                          h[u][v]=0;
       }
    }
  }while(q=='n' && k<3);
}
/*******************************************/

void try2(int i,int x,int y)      // all possible solutions
{
  int u,v,k=-1;
  do{
    ++k;
    u=x+a[k];  v=y+b[k];
    if (u>=0 && u<n && v>=0 && v<n && h[u][v]==0){
       h[u][v]=i;
       if (u==X_goal && v==Y_goal){
          print(h);
          getch();
       }
       else
          try2(i+1,u,v);
```

```
        h[u][v]=0;
    }
  }while(k<3);
}
/*********************************************/
```

```
void try3(int i,int x,int y) // optimal solution=shortest way
{
  int u,v,k=-1;
  do{
     ++k;
     u=x+a[k];  v=y+b[k];
     if (u>=0 && u<n && v>=0 && v<n && h[u][v]==0){
        h[u][v]=i;
        if (u==X_goal && v==Y_goal){
           if (minL==0 || i<minL){
              minL=i;
              for (int w=0;w<n;w++)
                 for (int t=0;t<n;t++)
                    optS[w][t]=h[w][t];
           }
        }
        else
           try3(i+1,u,v);
        h[u][v]=0;
     }
  }while(k<3);
}
/*********************************************/

void print(int h[n][n])
{
  for (int i=0;i<n;i++){
     for (int j=0;j<n;j++)
        cout<<h[i][j]<<"    ";
     cout<<endl;
  }
}
```

```
/**********************************************/

void reset()
{

  h[0][0]=0; h[0][1]=0; h[0][2]=0; h[0][3]=0; h[0][4]=0; h[0][5]=0;
  h[0][6]=0; h[0][7]=0; h[0][8]=0; h[0][9]=0; h[1][0]=0; h[1][1]=0;
  h[1][2]=0; h[1][3]=0; h[1][4]=1; h[1][5]=0; h[1][6]=0; h[1][7]=0;
  h[1][8]=0; h[1][9]=0; h[2][0]=0; h[2][1]=0; h[2][2]=1; h[2][3]=0;
  h[2][4]=1; h[2][5]=0; h[2][6]=1; h[2][7]=1; h[2][8]=0; h[2][9]=0;
  h[3][0]=0; h[3][1]=1; h[3][2]=1; h[3][3]=0; h[3][4]=1; h[3][5]=0;
  h[3][6]=0; h[3][7]=1; h[3][8]=0; h[3][9]=0; h[4][0]=0; h[4][1]=0;
  h[4][2]=0; h[4][3]=0; h[4][4]=0; h[4][5]=1; h[4][6]=0; h[4][7]=1;
  h[4][8]=0; h[4][9]=1; h[5][0]=1; h[5][1]=1; h[5][2]=1; h[5][3]=0;
  h[5][4]=1; h[5][5]=1; h[5][6]=0; h[5][7]=1; h[5][8]=0; h[5][9]=1;
  h[6][0]=1; h[6][1]=0; h[6][2]=1; h[6][3]=0; h[6][4]=1; h[6][5]=1;
  h[6][6]=0; h[6][7]=1; h[6][8]=0; h[6][9]=1; h[7][0]=1; h[7][1]=0;
  h[7][2]=1; h[7][3]=0; h[7][4]=0; h[7][5]=0; h[7][6]=0; h[7][7]=1;
  h[7][8]=0; h[7][9]=1; h[8][0]=1; h[8][1]=0; h[8][2]=0; h[8][3]=0;
  h[8][4]=1; h[8][5]=1; h[8][6]=1; h[8][7]=1; h[8][8]=0; h[8][9]=1;
  h[9][0]=1; h[9][1]=1; h[9][2]=1; h[9][3]=1; h[9][4]=1; h[9][5]=1;
  h[9][6]=1; h[9][7]=1; h[9][8]=1; h[9][9]=1;
}
```

مثال

ليكن لدينا مصفوفة مربعة بعدها n , والمطلوب إيجاد الحل الأمثل للمسألة التالية مستخدماً مفهوم الخوارزميات التراجعية: يجب تلوين جميع عناصر هذه المصفوفة, أي يجب إعطاء لون معيّن لكل عنصر, بحيث إنه لا يوجد عنصران متجاوران من اللون نفسه (لا بالسطر نفسه ولا العمود نفسه ولا بالقطر المباشر نفسه أو غير المباشر نفسه). وعلماً أن العدد الأعظم للألوان المسموح استخدامها يساوي إلى 8 لون مختلف. أي يكون الحل أمثلياً في

حال استخدامنا للتلوين أصغر عدد ممكن من الألوان. ثم بعد استدعاء التابع التراجعي يجب طباعة "العدد الكلي للحلول الممكنة".

مثال: ليكن لدينا n=5 والألوان المتاحة هي: أحمر, أزرق, أخضر, أصفر, أسود, أبيض, بني, رمادي.

الحل: الأمثل يستخدم 4 ألوان فقط من أصل 8 وهو على الشكل التالي:

```
The number of all possible solutions=325061760
The minimum number of used colors in the optimal solution=4
   black     red   black     red   black
    blue   green    blue   green    blue
   black     red   black     red   black
    blue   green    blue   green    blue
   black     red   black     red   black
```

الحل:

```
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>

const n=5;

char* colors[8]={"black","red","blue","green","yellow",
"white","brown","gray"};
char used[8]={'n','n','n','n','n','n','n','n'};
int solution[n][n]={-1};
int optS[n][n]={-1};
int minC=9;//minimum number of colors in optimal solution
int col=0;  //  colors used until now
int solNum=0; // number of all possible solutions

void color(int i,int j);
void print(int s[n][n]);
//---------------------------------------------

int main(int argc, char* argv[])
{
   color(0,0);
```

```
   if (minC>0){
     cout<<"The number of all possible solutions=" << solNum<<endl;
    cout<<"The minimum number of used colors in the optimal
solution="<<minC<<endl;
       print(optS);
   }
   else
      cout<<"No solution!";
   getch();
   return 0;
}
//---------------------------------------

void color(int i,int j)
{
   int k=-1;
   char restoreColor;
   do{
      ++k;
      if (i>0 && solution[i-1][j]==k)
         continue;
      if (j>0 && solution[i][j-1]==k)
         continue;
      if (i>0 && j>0 && solution[i-1][j-1]==k)
         continue;
      if (i>0 && j<n-1 && solution[i-1][j+1]==k)
         continue;

      solution[i][j]=k;
      restoreColor='n';
      if (used[k]=='n'){     // first use of this color
         ++col;
         used[k]='y';
         restoreColor='y';  // only restore if its the first use!!
      }

      if (j<n-1)
         color(i,j+1);
      else
      if (i<n-1)
         color(i+1,0);
```

```
      else{
         ++solNum;
         if (col<minC){
            minC=col;
            for (int a=0;a<n;a++)
               for (int b=0;b<n;b++)
                  optS[a][b]=solution[a][b];
         }
      }

      solution[i][j]=-1;
      if (restoreColor=='y'){
         --col;
         used[k]='n';
      }
   }while(k<n-1);
}
//---------------------------------------------

void print(int s[n][n])
{
   for (int i=0;i<n;i++){
      for (int j=0;j<n;j++)
         cout<<setw(10)<<colors[s[i][j]];
      cout<<endl;
   }
}
```

4-5 تمارين الفصل الرابع

تمرين1: لتكن لدينا مجموعة من أعداد صحيحة مختلفة, والمطلوب تشكيل سلسلة من هذه الأرقام بحيث يكون الفرق ما بين كل عددين متتاليين إما زيادة بواحد أو ناقصاً بواحد.

مثال: لتكن لدينا الأعداد التالية: 1 , 1 , 2 , 2 , 3 , 4 , 4 , 5 , 5 , 6

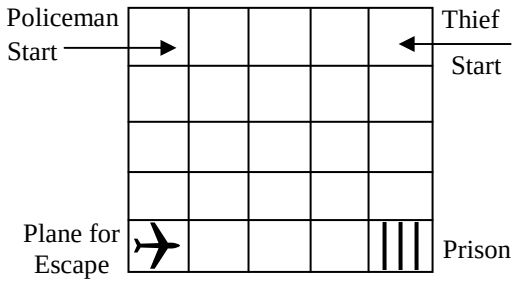
إحدى الحلول الممكنة يكون على الشكل التالي: 1 , 2 , 1 , 2 , 3 , 4 , 5 , 4 , 5 , 6

أما مجموعة الأعداد التالية مثلاً فليس لها حل: 1 , 1 , 2 , 2 , 3 , 4 , 4 , 5 , 5 , 7

و المطلوب:

اكتب برنامجاً لإيجاد جميع الحلول الممكنة لهذه المسألة اعتماداً على مفهوم الخوارزميات التراجعية.

تمرين 2: لتكن لدينا اللعبة التالية:



يوجد مجرم Thief هارب من الشرطي Policeman , لكل واحد منهما مربع بدء مختلف عن الآخر (كما هو موّضح بالشكل المعطى). والغاية من اللعبة هي وصول المجرم إلى المربع الحاوي على الطائرة Plane التي تمّكن المجرم من الهروب بها إلى مكان آمن, وهنا تنتهي اللعبة بنجاح. حركات المجرم دوماً مربع واحد بإحدى الاتجاهات الأربعة: إما إلى الأعلى أو الأسفل أو اليمين أو اليسار حيث لا يمكن المجرم المرور على المربع الواحد أكثر من مرة واحدة. وكلما تحرك المجرم خطوة واحدة, يتحرك الشرطي أيضاً ولكن بالاتجاه المعاكس إن أمكن ذلك, أي إذا تحرك المجرم إلى الأعلى مثلاً فيتحرك الشرطي إلى الأسفل, المجرم للأسفل فالشرطي للأعلى, المجرم لليمين فالشرطي لليسار, المجرم لليسار فالشرطي

لليمين. تكون حركة المجرم مقبولة إذا كان الموضع الجديد مقبولاً وإلا يجب عليه اختيار حركة أخرى, أما بالنسبة للشرطي فإذا لم يكن المربع الجديد مقبولاً بالنسبة له فيبقى واقفاً بمكانه الحالي. تنتهي اللعبة بخسارة عندما يلتقيان الشرطي والمجرم على المربع نفسه أو عندما يقف المجرم على المربع الحاوي على السجن Prison . **والمطلوب:**

اكتب برنامجاً اعتماداً على مفهوم الخوارزميات التراجعية لإيجاد حل لإنهاء هذه اللعبة بنجاح (أي وصول المجرم إلى الطائرة دون أن يتم القبض عليه أو يسجن).

تمرين 3: ليكن لدينا عدد من السجادات n =12 وكل سجادة لها طول معيّن مقيس بالمتر. وليكن لدينا طريق طوله 1000 متر مثلاً. **والمطلوب:**

اكتب خوارزمية تراجعية تقوم بإيجاد حل لمد عدد معيّن من السجادات بحيث إنه يغطى الطريق كاملاً (أي من دون زيادة ولا نقصان). أطوال السجادات ممثلة بالشكل التالي مثلا:

100 , 250 , 450 , 300 , 50 , 600 , 400 , 200 , 50 , 10 , 30 , 175

تمرين 4:

لتكن لدينا لعبة الحظ التالية: لدينا 25 مربعاً, بعض المربعات تمثل ماء (▨) , فإذا وقف عليها اللاعب يغرق فيها ويخسر. أما الوقوف على بقية المربعات فيربّح اللاعب جائزة مادية معينة عند كل مربع, فمثلاً يمكن للوقوف على مربع ما أن يربّح 100 ل.س أو 500 ل.س أو 0 ل.س ..... الخ, وعلماً أنه لا يمكن الوقوف على المربع الواحد أكثر من مرة.يتم تحديد كل من مربع البدء ومربع النهاية, وعلى اللاعب أن يبدأ بجمع الجوائز بدءاً من مربع البدء وحتى يصل إلى مربع النهاية دون أن يغرق. **والمطلوب:**

اكتب خوارزمية تراجعية تقوم بإيجاد الحل الأمثل بالنسبة للاعب, أي هو يريد أن يعلم, كيف يمكنه أن يمشي بحيث إنه يربح أكبر مبلغ ممكن.

يمكن توزيع الجوائز بالشكل التالي مثلاً :

10	▨	0	15	100
300	50	20	150	5
25	0	400	15	350
250	80	800	▨	500
▨	15	5	200	80

تمرين 5: إحدى شركات القطارات تخدم n محطة : $S_1, S_2, \ldots, S_n$ و ترغب في تقديم خدمة معلومات لزبائنها . تتلخص هذه الخدمة بإتاحة طرفيات للمستثمرين في محطات القطارات ، يستطيع المستثمرون بواسطتها الاستفسار عن كيفية الذهاب من أي محطة S_d إلى محطة أخرى S_a . إجابة برنامج هذه الخدمة يجب أن تكون قائمة بالقطارات التي تصل المحطة S_d بالمحطة S_a (ليس بالضرورة وجود قطار مباشر بين المحطتين ، و نفترض أيضاً مراعاة زمن كاف للتبديل بين قطارين) بأسرع وقت ممكن . و **المطلوب :**

1. وضح و ناقش بنى المعطيات المستخدمة (بفرض أن عدد محطات الشركة هو 10 و الزمن الأصغري للتبديل بين قطارين لا يقل عن 15 دقيقة)
2. اكتب إجرائية تقوم بإعطاء قائمة مرتبة من القطارات التي تصل المحطة S_d بالمحطة S_a .

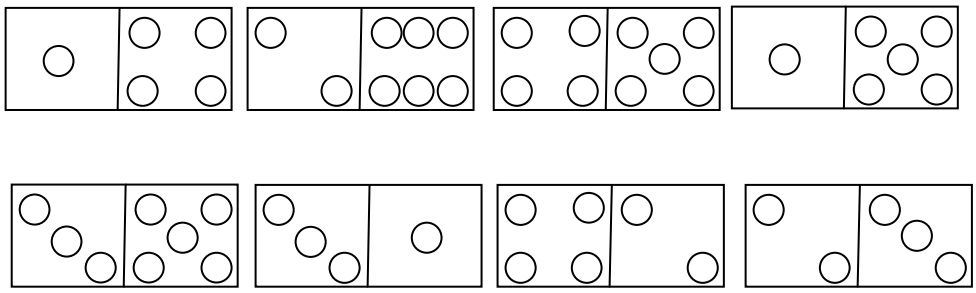
تمرين 6: لتكن A مجموعة من الرجال و B مجموعة من النساء كل رجل و كل امرأة يستطيع تحديد أفضليات من أجل اختيار الشريك الذي يناسبه . إذا تم اختيار الـــ n زوجا (أي ثنائية) و بقي على الأقل رجل و امرأة و يفضل كلاهما الآخر على الشريك الذي اختير بنتيجة الخوارزمية نقول عن الزواج إنه ليس مستقرا" . أما إذا لم يوجد مثل هذه الحالة ، فنقول عن هذا الزواج إنه مستقر. نفترض أن قوائم أفضليات كل شخص تكون جاهزة قبل تنفيذ الخوارزمية و لا يجوز تعديلها في أثناء عمل الخوارزمية .

1. وضح و ناقش بنى المعطيات المستخدمة (بفرض أن عدد الرجال هو 8 و عدد النساء أيضا" هو 8 و كل شخص يستطيع تحديد قائمة من 8 أفضليات مرتبة)
2. اكتب دالة إجرائية من أجل إيجاد الزوجة المناسبة للرجل m .
3. اكتب برنامجاً كاملاً يحل المسألة من أجل إيجاد الثنائيات التي تحقق شرط الزواج المستقر .

تمرين 7: لنفترض أنه لدينا n حجرة دومينو (Domino) ، حيث إنه لكل حجرة رقمان من 1 إلى 6 . نربط أحجار الدومينو بعضها ببعض لكي تعطي سلسلة من الأحجار المتتالية علماً أنه يكون الربط بالشكل التالي:

رقم بداية كل حجرة جديدة يجب أن يكون مطابقاً مع رقم نهاية الحجرة السابقة. ويسمح عند اختيار الأحجار الجديدة تدويرها 180° إن احتجنا إلى ذلك (أي إذا كان الرقم المطلوب في الجهة المعاكسة).

أمثلة على أحجار دومينو:



والمطلوب:

اكتب خوارزمية تراجعية تقوم بعدّ وطباعة جميع السلاسل التي يمكن تشكيلها من أصل n حجرة دومينو حيث إنه يجب أن تبدأ جميع السلاسل بالرقم 1 وتنتهي بالرقم 3 (طول السلاسل الناتجة لا يهم).

اكتب التعليمة (التعليمات) اللازمة لاستدعاء الخوارزمية السابقة.

مثال عن إحدى الحلول الممكنة للأحجار المرسومة بالأعلى: