جامعة العربي بن مهيدي - أم البواقي
كلية العلوم الدقيقة، علوم الطبيعة والحياة
قسم الرياضيات والإعلام الآلي
سنة أولى بنى موزعة
إمتحان في مادة الطرق الخوارزمية - تصحيح

اللقب : .....
الإسم : .....
الإمضاء : .....

1. (6 نقاط) أكتب خوارزمية Strassen مع التمييز بين المراحل الثلاث لطريقة D&C و معتمدا على الإتفاقيات التالية :
   - يمكن استعمال العاملان "+" و"-" على المصفوفات
   - يمكن تعيين جزء من مصفوفة على نحو : `M[a..b, c..d]`
   - يمكن استعمال العامل " <<- " لنسخ مصفوفة أو جزء من مصفوفة نحو : `M1 [a..b, c..d] <<- M2 [a..b, c..d]`

تذكير بالصيَغ :

$$\begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix} X \begin{bmatrix} B11 & B12 \\ B21 & B22 \end{bmatrix} = \begin{bmatrix} P1+P4-P5+P7 & P3+P5 \\ P2+P4 & P1+P3-P2-P6 \end{bmatrix}$$

where

- $P1=(A11+A22)\times(B11+B22)$
- $P2=(A21+A22)\times B11$
- $P3=A11\times(B12-B22)$
- $P4=A22\times(B21-B11)$
- $P5=(A11+A12)\times B22$
- $P6=(A11-A21)\times(B11+B12)$
- $P7=(A12-A22)\times(B21+B22)$

```
Strassen (A[1..n,1..n], B[1..n,1..n]) → Array[1..n,1..n]
      Var   A11, A12, A21, A22,
            B11, B12, B21, B22,
            C11, C12, C21, C22,
            P1, P2, P3, P4, P5, P6, P7 : Array[1..n/2,1..n/2]
            C : Array[1..n,1..n]

      If not isPowerOf(n,2) Return null   //n is power of 2. If not, we could add lines/columns of 0s
      If n = 1 Return A[1,1] x B[1,1]
      //divide ------------------
      A11 <<- A[1..n/2, 1..n/2]
      A11 <<- A[n/2..n, 1..n/2]
      A11 <<- A[1..n/2, n/2..n]
      A11 <<- A[n/2..n, n/2..n]
      B11 <<- B[1..n/2, 1..n/2]
      B11 <<- B[n/2..n, 1..n/2]
      B11 <<- B[1..n/2, n/2..n]
      B11 <<- B[n/2..n, n/2..n]
      //conquer ---------
      P1 <<- Strassen (A11+A22, B11+B22)
      P2 <<- Strassen (A21+A22, B11)
      P3 <<- Strassen (A11, B12-B22)
      P4 <<- Strassen (A22, B21-B11)
      P5 <<- Strassen (A11+A12, B22)
      P6 <<- Strassen (A11-A21, B11+B12)
      P7 <<- Strassen (A12-A22, B21+B22)
      //combine ---------
      C11 <<- P1+P4-P5+P7
      C12 <<- P3+P5
      C21 <<- P2+P4
      C22 <<- P1+P3-P2-P6
      C[1..n/2, 1..n/2] <<- C11
      C[n/2..n, 1..n/2] <<- C12
      C[1..n/2, n/2..n] <<- C21
      C[n/2..n, n/2..n] <<- C22

      Return C
End
```

1 point

1 point

2 points

2 points

2. (6 نقاط) أكتب خوارزمية على طريقة الخوارزميات الشرهة (Greedy algorithms) لحل مشكل تلوين الرؤوس في مبيان (Graph vertices) زيادة على القيد المعتاد (لونين مختلفين لرأسين متجاورين)، نريد الحد من عدد الألوان المستعملة إلى قيمة ما m.

As in course example, let's use :

- an adjacency matrix: A[i,j]=1 => vertex i and vertex j are adjacent.
- sequential numbers as ids of colors and an array (C) to store solution (series of selected colors)

2.5 points

```
Greedy_Graph_Col (A[1..n, 1..n], m) → Array[1..n]
        // n: nbr of vertices. A: Adjacency matrix.
        // m: max nbr of authorized colors
        Var C[1..n] ← [0]    // C: selected colors numbers (initially no one)
        luc ← 0              // luc: last used color
        For i ← 1..n         // equivalent to C ← C \ {e} as per template notation
                (scn, luc) ← Select_Color(A, C, i, luc, m)    //scn : selected color number.
                if scn > 0
                        C[i] ← scn    // equivalent to S ← S U {e} as per template notation
                Else
                        Return C
                End if
        End for
End
```

3.5 points

```
Select_Color (A[1 .. n, 1..n], C[1..n], i, luc, m) → (integer, integer)
        // i: index of vertex to be colored.
        For k ← 1..n
                If A[i,k] ≠ 1 & C[k] ≠ 0      //k^th vertex not adjacent to i^th and has a color
                        Return (C[k], luc)    // then reuse it (+ remember last used color luc)
                End if
        End for
        If luc < m           //last used color number less than authorized max
                Return (luc+1, luc+1)    // then pick an additional new color and remember it
        End if
        Return -1
End
```

3. - (6 نقاط) أكتب خوارزمية على طريقة التراجع (Backtracking) تسمح بالخروج من متاهة (maze) .

مخطط المتاهة تمثله مصفوفة M حيث يشير الرقم 1 إلى أنّ الخلية مسدودة و 0 أنها متاحة. مسار الخروج من المتاهة تمثله سلسلة خلايا مُرتّبة في مصفوفة ثانية P . أنظر للبيان.

قواعد التنقل :

- خليتا الدخول والخروج من المتغيرات
- لا يمكن المرور عبر نفس الخلية مرتين
- يمكن التحرك عموديا أو أفقيا فقط

- (2 نقاط) البدأ بتنظيرالمشكلة من خلال متغيرات أخذ القرارات و القيود (decision variables and constraints)

Path matrix P

	↓2	3	4	5	6	7	8
1	1	0	17	18	19	20	21	22
2	2	0	16	0	0	0	0	23
3	3	0	15	14	13	12	0	24
4	4	0	0	0	0	11	0	25
5	5	6	7	8	9	10	0	26
6	0	0	0	0	0	0	0	27
7	0	0	0	0	0	0	0	28
8	0	0	0	0	0	0	0	29 ↓

Maze configuration matrix M

	↓2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	0
2	0	1	0	1	1	1	1	0
3	0	1	0	0	0	0	1	0
4	0	1	1	1	1	0	1	0
5	0	0	0	0	0	0	1	0
6	0	1	0	0	0	1	1	0
7	0	1	0	1	0	0	1	0
8	0	0	0	1	0	0	1	0 ↓

Formalisation:

2 points

- P = {x_{ij}, i=1..n, j=1..m}. n*m decision variables, one for each cell of a maze of n*m size
- D = {0..n*m}. x_{ij} = 0 => cell is not part of the path. x_{ij}=k => cell is the k^{th} of the path
- C1: ∀ i,j P[i,j] ≠ 0 then M[i,j] ≠ 1 → empty cells only

  C2: ∀ i≠k ∀ j≠l, P[i,j] ≠ P[k,l] → same cell only once

  C3: ∀ x_k=P[l_k,c_k]≠0, if ∃ x_{k+1} = x_k+1, then (l_{k+1}=l_k & c_{k+1}=c_k±1) or (l_{k+1}=l_k±1 & c_{k+1}=c_k) → H or V move

1 point

```
Appelant_proc()
        Init (M[1..n,1..m])        // Maze configuration matrix. n=m=8 accepted
        P[1..n,1..m] ← [0]         // Init Path matrix to 0s
        (le, ce) ← (1, 1]          // line and column numbers of entry cell
        (lx, cx) ← (n, n)          // line and column numbers of exit cell
        BackTrack_Maze (M, P, le, ce, lx, cx, 1)
        // result: P where ≠ 0     // by reference
End
```

5 points

```
BackTrack_Maze ( M[1..n,1..m], P [1..n,1..m], lc, cc, lx, cx, i)
        //(lx,cx): exit cell. (lc, cc): current cell. i: number of the next cell of the path

        If (lc, cc) = (lx, cx)
                Return          // arrived to exit
        End if

        For (dl, dc) in {(+1, 0), (-1, 0), (0, +1), (0, -1)} // dl: delta L & dc: delta C. C3
                l ← lc + dl
                c ← cc + dc
                If ( 1 ≤ l ≤ n  &  1 ≤ c ≤ m )                // definition of x_ij vars
                        If ( M[l,c] ≠ 1  &  P[l,c] = 0 )      // C1 & C2
                                P [l,c] ← i
                                BackTrack_Maze (M, P, l, c, lx, cx, i+1)
                        End if
                End if
        End for
End
```