# Sujet E3A 2019-Mines 2012 : correction

**Exercice 3**
**Question 15** Le diamètre de $G_1$ est 3 et un chemin maximal est $(0, 2, 3, 4)$. Le diamètre de $G_2$ est 3 et un chemin maximal est $(0, 1, 2, 3)$.
**Question 16**
a. Le graphe de sommets $\{0, 1, 2, 3, 4\}$ d'arête $\{0, 1\}$, $\{1, 2\}$, $\{2, 3\}$, $\{3, 4\}$, $\{4, 5\}$ est connexe non orienté de diamètre maximal 4.
b.

```
let diam_max n = let g=Array.make n [] in g.(0)<-[1]; g.(n-1)<-[n-2];
    for i=0 to n-1 do g.(i)<-[i-1;i+1];
    g;;
```

**Question 17**
a. Le graphe de sommets $S = \{0, 1, 2, 3, 4\}$ et d'arêtes entre tous les sommets, à savoir
$A = \{\{0, 1\}, \{0, 2\}, \{0, 3\}, \{0, 4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$, est connexe non orienté et a diamètre 1.
b.

```
let diam_min n = let g=Array.male n [] in
    for i=0 to n-1 do for j=0 to n-1 do if j<>i then g.(i)<-j::g.(i) done done;
    g;;
```

**Question 18** Cet algorithme a en entrée un graphe pondéré non orienté et un sommet $a$. En sortie, il renvoie les distances pondérées $d(a, b)$ entre $a$ et tout autre somme $b$ du graphe. En considérant un graphe connexe non orienté comme un graphe pondéré de pondération unitaire (à savoir $p(a) = 1$ pour tout $a \in A$, $p(a) = 0$ pour tout $a \notin A$), cet algorithme renvoie toutes les distance $d(a, b)$, pour $a$ fixé. Il suffit ensuite d'exécuter cet algorithme sur tous les sommets $a$ et de trouver le maximum de toutes ces distances pour obtenir le diamètre du graphe.
**Question 19** Le parcours en largeur permet de calculer toutes les distances entre un sommet fixé et les autres sommets.
**Question 20**L'algorithme de Dijkstra nécessite d'ordonner les sommets à parcourir en fonction des distances des sommets depuis lesquels ils sont découverts. Il est donc plus complexe que le simple parcours en largeur. On lui préférera donc le parcours en largeur.
**Question 21**

```
let a=Noeud( 0 ,
Noeud( 1 ,
Noeud( 2, Noeud( 4, Feuille, Feuille), Feuille),
Noeud( 3, Noeud( 5, Feuille, Feuille), Noeud( 6, Feuille, Feuille))),
Feuille);;
```

Le diamètre est 4. Ses chemins maximaux sont $(4, 2, 1, 3, 5)$ et $(4, 2, 1, 3, 6)$.
**Question 22** $r = n - 1$.
**Question 23**

```
let rec nb_noeuds a = match a with
    | Feuille -> 0
    | Noeud(_,g,d) -> 1 + (nb_noeuds g) + (nb_noeuds d);;
```

**Question 25**

```
let arbre_vers_graphe a = let n=nb_noeuds a in let g=Array.make n [] in
    let rec aux arbre i = match arbre with
        | Feuille -> ()
        | Noeud(x,g,d) -> g.(i)<- x :: g.(i); g.(x) <- i:: g.(x); aux g x; aux d x in aux a; g;;
```

**Question 26** On applique l'algorithme de parcours en largeur au graphe associé à l'arbre binaire considéré. La fonction de création du graphe est de complexité $O(n)$. L'algorithme de parcours est de complexité $O(n)$ pour un seul sommet donc de $O(n^2)$ pour tous les sommets. La complexité totale est de $O(n^2)$.
**Question 27** Un chemin passant par la racine de longueur maximale de $a = Noeud(x, g, d)$ est est un chemin de la plus grande profondeur de $g$ à la racine concaténé avec un chemin de la racine à la plus grande profondeur de $d$. Il aura donc longueur $h(g) + h(d)$.
**Question 28**

```
let diam_arbre a =
    let rec haut_diam a = match a with
        | Feuille -> 0,0
        | Noeud(_,g,d) ->
        let (hg,dg)=haut_diam g and (hd,dd)=haut_diam d in
        let h=1+(max hg hd) in (h,max (max dg dd) (hg+hd)) in let (h,d)=haut_diam a in d;;
```

**Problème**

**Question 5** $C_0 = \{\{0_A, 1_B\}, \{1_A, 3_B\}, \{2_A, 0_B\}\}$ est un couplage de cardinal 3.

**Question 6** On ne peut coupler à la fois $1_A$ et $3_A$ car ils ne sont voisins que d'un seul sommet, $3_B$. On ne peut donc coupler que $0_A$, $2_A$ et l'un des sommets parmi $1_A$ et $3_A$. Au maximum, on pourra coupler trois couples de sommets. Il n'existe pas de couplage de cardinal quatre.

**Question 7**

```
let verifie g c = let n=Array.length c and b= ref true and tab=Array.make n true in
    for i=0 to n-1 do if c.(i)<>-1 then begin b:= !b && tab.(c.(i)) && g.(i).(c.(i)); tab.(c.(i))<-false end done;
    !b;;
```

Il y a des instructions de complexité $O(1)$ et une boucle de taille $n$ dont chaque passage a un test et au plus une instruction de complexité $O(1)$. La complexité totale est donc $O(n)$.

**Question 8**

```
let cardinal c= let n=Array.length c and card= ref 0 in
    for i=0 to n-1 do if c.(i)<>-1 then card:= !card +1 done;
    !card;;
```

Il y a des instructions de complexité bornée, une boucle de taille comprenant un nombre borné d'instructions de complexité bornée. La complexité totale de la fonction cardinal est de $O(n)$.

**Question 9** La première arête choisie sera $\{1_A, 3_B\}$ ou $\{3_A, 3_B\}$. Choisissons $\{1_A, 3_B\}$. On retire $\{2_A, 3_B\}$, $\{3_A, 3_B\}$.
On choisit ensuite $\{2_A, 2_B\}$ et on retire $\{2_A, 0_B\}$, $\{2_A, 1_B\}$, $\{0_A, 2_B\}$.
On pourra ensuite choisir $\{0_A, 0_B\}$ et retirer les arêtes $\{0_A, 1_B\}$, $\{0_A, 2_B\}$. Le graphe n'ayant plus d'arêtes, l'algorithme s'arrête.
Le couplage obtenu est $\{0_A, 0_B\}$, $\{2_A, 2_B\}$, $\{1_A, 3_B\}$.

**Question 10** La première arête choisie est $\{3_A, 2_B\}$. On retire ensuite $\{2_A, 2_B\}$ et $\{3_A, 3_B\}$. Le nouveau graphe, sans les sommets $3_A, 2_B$, possèdent deux composantes connexes : $\{0_A, 1_A, 2_A, 0_B, 1_B\}$ et $\{4_A, 5_A, 3_B, 4_B, 5_B\}$. Chacun de ces deux graphes ne pourra avoir de couplage de cardinal strictement supérieur à deux. Au maximum, le couplage obtenu sera de cardinal $1 + 2 + 2 = 5$. Pourtant le couplage $\{i_1, i_B\}$ pour tout $i \in [\![0, 5]\!]$ est de cardinal 6. Ainsi, le couplage obtenu par l'algorithme ne renvoie pas un couplage de cardinal maximum.

**Question 11**

```
let arete_min g a = let n=Array.length g let d= ref n+n and degA=Array.make n 0 and degB=Array.make n 0 and couple= ref (-1,-1) in
    for i=0 to n-1 do for j=0 to n-1 do if g.(i).(j) then begin degA.(i)<-degA.(i)+1; degB.(j)<-degB.(j)+1 done done;
    for i=0 to n-1 do for j=0 to n-1 do
        let x=degA.(i)+degB.(j) in if x < !d then begin d:=x; couple:=(i,j) end done done;
    if !couple=(-1,-1) then false
    else begin let (i,j)= !couple in a.(0)<-i; a.(1)<-j; true end;;
```

Dans cette fonction se trouvent deux boucles imbriquées de taille $n$ dont chaque instruction est en nombre borné et de complexité bornée. La complexité totale est donc $O(n^2)$.

**Question 12**

```
let supprimer g a = let n=Array.length g in for j=0 to n-1 do g.(a.(0)).(j)<-false done;
    for i=0 to n-1 do g.(i).(a.(1))<-false done;
    g;;
```

Cette fonction utilise deux boucles de taille $n$. La complexité de cette fonction est de $O(n)$.

**Question 13**

```
let algo_approche g = let a=[|-1;-1|] and g0=dupliquer_matrice g and b= ref true and c=Array.make n -1 in
    while !b do
        b:= arete_min g a;
        c.(a.(0))<-a.(1);
        supprimer g a done;
    c;;
```

La boucle "tant que" aura une taille au plus de $n$. Chaque passage de boucle a une complexité de $O(n^2)$. La complexité totale est donc $O(n^3)$.

**Question 14**

```
let une_arete g a= let n=Array.length g and i= ref 0 and j= ref 0 and b= ref true in
    while !b && !i<n do
        j:=0;
        while !b && !j<n do
            if g.(!i).(!j) then begin b:=false; a.(0)<-!i; a.(1)<-!j end;
            j:= !j +1 done;
        i:= !i +1 done;
    not(!b);;
```

**Question 15**

```
let meilleur_couplage g = let a=[|-1;-1|] and n=Array.length g in
    let rec aux g c =
        let g0=dupliquer_matrice g and g1=dupliquer_matrice g in
        if not(une_arete g0 a) then c
```

```
        else begin
        let c0=dupliquer_matrice c in g0.(a.(0)).(a.(1))<-false;
        c.(a.(0))<-a.(1); supprimer g1 a;
        let c0=aux g0 c0 and c=aux g1 c in
        let n1=cardinal c0 and n2=cardinal c in
        if n1>n2 then c0 else c end in let c=Array.make n -1 in aux g c;;
```

**Question 16** Le seul sommet libre de $A$ est $2_A$ ; le seul sommet libre de $B$ est $4_B$. Une chaîne alternée augmentante relativement à $C_1$ est $(2_A, 2_B, 3_A, 3_B, 4_A, 4_B)$.
**Question 17** Soit $C'$ l'ensemble des arête d'une chaîne alternante. Soit $C_0 = C\Delta C' = (C \cup C')\backslash(C \cap C')$.
Une arête de $C'$ qui n'est pas dans $C$ possède deux extrémités. Si cette extrémité est adjacente à une arête de $C$, $C$ étant un couple, cette arête est unique ; il s'agit donc d'une arête du chemin constitué par $C'$ qui est donc dans $C' \cap C$ donc qui n'est pas dans $C_0$. Ainsi, dans $C_0$, toute arête de $C'$ n'est adjacente d'aucune autre arête de $C_0$. Comme $C$ est un couplage, toute arête de $C_0 \cap C$ n'est adjacente à aucune arête de $C$.
En conclusion, $C_0$ est bien un couplage. Enfin, $C'$ a sa première et sa dernière arête qui ne sont pas dans $C$ ; $C'$ a donc $n$ arêtes dans $C$ et $n+1$ non dans $C$. Ainsi, $\mathrm{card}(C_0) = \mathrm{card}(C) + \mathrm{card}(C') - 2\,\mathrm{card}(C \cap C') = \mathrm{card}(C) + 2n + 1 - 2n = \mathrm{card}(C) + 1$.
**Question 18** La chaîne est $(1_A, 0_B, 0_A, 1_B, 2_A, 2_B, 3_A, 3_B)$. Elle est bien alternée, commence et termine par deux sommets libres. Elle est donc augmentante.
**Question 19**

```
let actualiser c r mA mB = let n=Array.length c and j= ref 0 in
    while c.(!j)<>-1 || mB.(!j)=-1 do j:= !j+1 done;
    let b= ref true in
    while !j<>-1 do
        if !b then begin r.(!j)<-mB(!j); c.(mB(!j))<-!j; j:=mB(!j) end
        else j:=mA(!j);
        b:=not(!b) done;;
```

**Question 20** Les sommets qui peuvent être atteints sont : $1_A$, $3_B$, $4_B$, $3_A$, $4_A$. $1_A$ possède la marque $-1$, $3_B$ la marque 1, $3_A$ la marque 3, $4_B$ la marque 3, $4_A$ la marque 4. Une chaîne alternée augmentante doit aboutir en $1_B$ qui ne peut être atteint. On ne peut donc pas trouver de chaîne alternée augmentante relativement à $C_2$.
**Question 21** La récursivité croisée est un couple de fonctions récursives tel que chacune de ces fonctions fait appel à l'autre fonction pour son propre calcul.

```
let let rec chercherA g c r mA mB i = let n=Array.length g and f= ref -1 in
    for j=0 to n-1 do
        if g.(i).(j) && c.(i)<>j then begin mB(j)<-i; f:=chercherB j end
    done;
    !f;
and chercherB g c r mA mB j = if r.(j)=-1 then j else begin mA(r.(j))<-j; chercherA r.(j) end;;
```

**Question 22**

```
let chaine_alternee g c r mA mB = let n=Array.length g and fin=ref -1 and i=ref 0 in
    while !i<n && !fin = -1 do
        if c.(!i)=-1 then fin:=chercherA !i done;
    !fin;;
```

**Question 23**

```
let algorithme_hongrois g = let n=Array.length g in let c=Array.make n -1 and r=Array.make n -1
and mA=Array.make n -1 and mB=Array.make n -1 in
    let numero=ref chaine_alternee g c r mA mB in
    while !f<>-1 do
        actualiser g c r mA mB !numero;
        numero:=chaine_alternee g c r mA mB done;
    c;;
```