Exercice 1

Soit $f(n) = \frac{1}{3}n\left(n + \frac{1}{2}\right)(n+1)$. Montrer que $f(n) = \mathcal{O}(n^3)$

Les relations suivantes sont elles vraies ?

i. $f(n) = \mathcal{O}(n^{10})$
ii. $f(n) = \frac{1}{3}n^3 + \mathcal{O}(n^2)$
iii. $\mathcal{O}(n^2) = \mathcal{O}(n^3)$
iv. $\mathcal{O}(n^3) = \mathcal{O}(n^2)$
v. $\frac{1}{3}n^3 + \mathcal{O}(n^2) = \mathcal{O}(n^3)$

Exercice 2

Dessiner le graphe G_1 non orienté décrit par les listes d'adjacence ci-dessous. Donner la matrice d'adjacence et la matrice d'incidence de G_1 (on numérotera les arêtes suivant l'ordre lexicographique sur le couple de sommets *extrémités*).

```
1 : 2, 3
2 : 1, 3, 4, 5
3 : 1, 2, 5
4 : 2, 5
5 : 2, 3, 4
```

Exercice 3

Donner les listes d'adjacence, la matrice d'adjacence et la matrice d'incidence du graphe orienté de la FIG. 1.

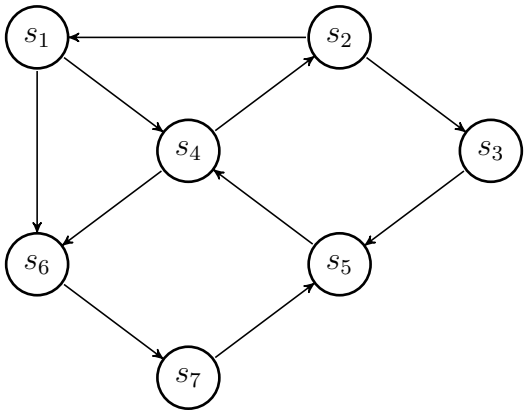

FIGURE 1 – G_2

Exercice 4

Appliquer l'algorithme du ***parcours en largeur*** `PL(G, s)` au graphe G_1 (Exercice 2) à partir du sommet `1` et au graphe G_2 (Exercice 3) à partir du sommet s_1. Chaque fois donner l'ordre d'entrée des sommets dans la file.

```
0   PL(G,s)
1     pour chaque sommet u de X[G] \ {s} faire
2         couleur[u] <- BLANC
3         d[u] <- infini
4         pere[u] <- nil
5     couleur[s] <- GRIS
6     d[s] <- 0
7     pere[s] <- nil
8     Enfiler(F, s)
9     tant que non vide(F) faire
10        u <- tete(F)
11        pour chaque v de Adj(u) faire
12            si couleur[v] = BLANC
13                alors couleur[v] <- GRIS
14                      d[v] <- d[u] + 1
15                      pere[v] <- u
16                      Enfiler(F, v)
17        Defiler(F)
18        couleur[u] <- NOIR
```

Exercice 5

On se donne un graphe orienté à n sommets et m arcs. On suppose que le degré (nombre d'arcs entrant ou sortant) de chaque sommet est au plus d.

Donner l'algorithme et une majoration (la plus basse possible en fonction de n, m, d) pour la complexité de chacune des opérations suivantes :

1. faire afficher tous les successeurs d'un sommet s donné
2. faire afficher tous les prédécesseurs d'un sommet s donné
3. déterminer si un graphe est sans boucle (une boucle est un arc $s \to s$)
4. effectuer le parcours en largeur à partir d'un sommet donné

suivant qu'on utilise, pour représenter le graphe, une matrice d'adjacence `A[i,j]`, une matrice d'incidence `B[u,e]`, ou des listes de successeurs `Adj[u]`.

Exercice 6

On considère un graphe orienté à n sommets et m arcs. On dit que deux arcs e_1 et e_2 sont opposés si $e_1 = (i, j)$ et $e_2 = (j, i)$ avec i et j deux sommets distincts du graphe.

Pour chacune des trois représentations (listes de successeurs, matrice d'adjacence, matrice d'incidence), écrire un algorithme pour déterminer si le graphe contient des arcs opposés et en étudier la complexité.

Solutions (sketch) pour le chargé de TD

Exercice 1

$$f(n) = \frac{1}{3}n(n+\frac{1}{2})(n+1) = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

Il suffit de majorer $f(n)$. Pour tout $n \geq 1$, on a :

$$\frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n \ \leq \ \frac{1}{3}n^3 + \frac{1}{2}n^3 + \frac{1}{6}n^3 \ = \ n^3$$

Donc $f(n) \leq n^3$ pour tout $n \geq 1$, ce qui montre que $f(n) = \mathcal{O}(n^3)$.

i. Vrai.

ii. Vrai.

iii. Vrai.

iv. Faux. Car il faudrait prouver que en fixant c_1 il existe n_0 et une constante c_2 telle que $c_1 \cdot n^3 \leq c_2 \cdot n^2$ pour $n \geq n_0$. On trouverait donc $c_2 \geq c_1 \cdot n$ donc que c_2 ne peut pas être une constante. L'égalité est bien un abus de notation. En effet, $\mathcal{O}(n^2) = \mathcal{O}(n^3)$ mais cela n'implique pas que $\mathcal{O}(n^3) = \mathcal{O}(n^2)$.

v. Vrai.

On peut aussi leur rappeler les relations suivantes :
- $n^\alpha = \mathcal{O}(n^\beta)$, si $\alpha \leq \beta$
- $f(n) = \mathcal{O}(f(n))$
- $\alpha \cdot \mathcal{O}(f(n)) = \mathcal{O}(f(n))$, si α est une constante
- $\mathcal{O}(\mathcal{O}(f(n))) = \mathcal{O}(f(n))$
- $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
- $\mathcal{O}(f(n) \cdot g(n)) = f(n) \cdot \mathcal{O}(g(n))$

ATTENTION ! Il faut se méfier des sommes de $\mathcal{O}(...)$ avec un nombre non bornés de termes, et plus généralement des formules qui utilise la notation grand-O (sommes, produits, exponentielles ...).

En général, $\sum_{i=1}^n \mathcal{O}(f(i)) \neq \mathcal{O}(\sum_{i=1}^n f(i))$. Par exemple, si $f(i) = \deg(i)$ (degré sortant du sommet numéro i dans un graphe orienté à n sommets), alors $\sum_i f(i) = m$, le nombre d'arcs du graphe. Certes $\mathcal{O}(\sum_{i=1}^n f(i)) = \mathcal{O}(m)$, cependant $\sum_{i=1}^n \mathcal{O}(f(i)) \neq \mathcal{O}(m)$ en général. En effet, la notation $C(i) = \mathcal{O}(f(i))$ signifie qu'ils existent α et i_0 tels que, pour tout $i \geq i_0$, le coût pour le sommet i est $C(i) \leq \alpha f(i)$. Peut être que pour $i < i_0$, cela n'est pas vrai. On pourrait avoir $f(i) = \beta$ si $i < i_0$ par exemple. Donc pour tout $i \geq 1$, on peut seulement dire que $C(i) \leq \alpha f(i) + \beta$. Et donc $\sum_{i=1}^n C(i) \leq \sum_{i=1}^n (\alpha f(i) + \beta) = \alpha \sum_i f(i) + \sum_i \beta = \alpha m + \beta n = \mathcal{O}(n+m)$ ce qui est très différent de $\mathcal{O}(m)$, surtout si $m = 1$ et n est très très grand.

Un autre exemple pour illustrer le problème, est la preuve (fausse) de la complexité $\mathcal{O}(n)$ pour la fonction f suivante (on peut faire pareil avec Fibonacci) :

```
f(n):
  si n=1, retourner 1
  sinon, retourner f(n-1) + f(n-1)
```

Si $n = 1$, alors la complexité de $f(n)$ est bien $\mathcal{O}(1)$. Supposons que, pour tout $m < n$, la complexité de $f(m)$ est $\mathcal{O}(m)$. Montrons alors que la complexité de $f(n)$ est $\mathcal{O}(n)$. La complexité de $f(n)$ est $\mathcal{O}(1) + \mathcal{O}(n-1) + \mathcal{O}(n-1)$, ce qui fait $\mathcal{O}(2n-1) = \mathcal{O}(n)$. Or, c'est complètement

faux, $f(n)$ retourne 2^n, nombre qui est calculé en ne faissant que des sommes de 1. La complexité de $f(n)$ est $\mathcal{O}(2^n)$. En posant $C(n)$ comme étant la complexité de la fonction f, et en essayant de réécrire la preuve : $C(m) \leq \alpha \cdot m$ (qui est effectivement vraie au départ), et : $C(n) \leq \mathcal{O}(1) + 2 \cdot C(n-1) = \mathcal{O}(1) + 2\alpha \cdot (n-1)$ ce n'est pas borné par αn ... La récurrence ne marche pas.

Exercice 4 À faire ...

Exercice 5

1. Tous les successeurs du sommet s :
   (a) Matrice d'adjacence A[i,j] :

```
pour j allant de 1 à n faire:
    si A[s,j]=1, afficher(j)
```

   Complexité : $\mathcal{O}(n)$ d'après l'algorithme précédent.
   On peut améliorer dans la pratique l'algorithme précédent avec une version tenant compte du degré sortant.

```
j <- 1
compteur <- 0
tant que j <= n et compteur < d
   faire si A[s,j] = 1
         alors afficher(j)
               compteur = compteur + 1
         j <- j + 1
```

   Malheureusement, la complexité (dans le pire des cas) de ce second algorithme n'est pas amélioré, car même pour d petit, il peut arriver qu'il soit nécessaire de parcourir toute les colonnes de la ligne de s.
   (b) Matrice d'incidence B[i,k] :

```
pour e de 1 à m faire:
    si B[s,e]=1 alors
       pour i allant de 1 à n faire:
           si B[i,e]=-1 alors afficher(i)
```

   Complexité : $\mathcal{O}(m + n\deg(s)) = \mathcal{O}(m + nd)$. En effet, le premier "si" est exécuté m fois. S'il est faux, son coût est $\mathcal{O}(1)$. S'il est vrai, son coût est $\mathcal{O}(n)$. Le test est vrai $\deg(s) \leq d$ fois, d'où la complexité annoncée.
   On peut aussi tenir compte de d pour accélérér la boucle "pour e de 1 à m", mais la complexité de change pas dans le pire des cas.

```
k <- 1
compteur <- 0
tant que k <= m et compteur < d
   faire si B[s,k] = 1
         alors compteur = compteur + 1
               pour chaque i de 1 à n
                  faire si B[i,k] = -1
                        alors afficher(i)
                              sortir   // sort de la boucle pour i
         k <- k + 1
```

   (c) Listes de successeurs :

```
pour chaque v de Adj[s]
    faire afficher(v)
```

Complexité : $\mathcal{O}(\deg(s)) = \mathcal{O}(d) = \mathcal{O}(n)$.

2. Tous les prédecesseurs du sommet s :
   (a) Matrice d'adjacence A[i,j] :
   Pareil que pour 1(a) : remplacer "A[s,j]" par "A[j,s]". Même complexité.
   (b) Matrice d'incidence B[i,k] :
   Pareil que pour 1(b) : échanger le "-1" avec le "+1". Même complexité.
   (c) Listes de successeurs :

```
pour u allant de 1 à n faire:
    pour chaque v de Adj[u] faire:
        si v=s, afficher(u)
```

Complexité : $\mathcal{O}(n+m)$. Le coût pour le deuxième "pour" (le plus imbriqué) est $\mathcal{O}(\deg(u))$, proportionnel au degré de u. Attention, cela n'est pas exactement $\deg(u)$. En particulier lorsque $\deg(u)$ est petit, style 0, le coût est au moins une certaine constante (voir les remarques de Exercice 1). La complexité est : $\sum_u \mathcal{O}(\deg(u)) = \mathcal{O}(n+m)$.
On peut améliorer la temps en utilisant d et en sortant dès que possible. Cependant celà n'améliore pas la complexité dans le pire des cas.

```
i <- 1
compteur <- 0
tant que i <= n et compteur < d
   faire pour chaque v de Adj(i)
             faire si v = s
                   alors afficher(i)
                         compteur = compteur + 1
                         sortir    // sort de la boucle pour v
          i <- i + 1
```

3. Déterminer si un graphe est sans boucle :
   (a) Matrice d'adjacence A[i,j] :

```
pour chaque i de 1 à n
    faire si A[i,i] = 1
          alors retourner FAUX
retourner VRAI
```

Complexité : $\mathcal{O}(n)$.
   (b) Matrice d'incidence B[i,k] :
   la matrice d'incidence n'est pas une réprésentation adaptée pour repondre à cette question : on ne peut pas réprésenter des boucles en utilisant 1 et -1 (il faut éventuellement une troisième valeur non nulle, où coder différemment : mettre un 1 et vérifier qu'il n'y a pas de -1 correspondant. Cependant changer la convention est dangereux, les algorithmes précédents risquant de ne plus fonctionner).
   (c) Listes de successeurs :

```
pour chaque i de 1 à n faire:
    pour chaque v de Adj[i] faire:
        si v=i, retourner FAUX
retourner VRAI
```

Complexité : $\sum_u \mathcal{O}(\deg(i)) = \mathcal{O}(n+m)$.

4. Parcours en largeur à partir d'un sommet s donné : PL(G,s)

   (a) Matrice d'adjacence A[i,j] :
   Il faut remplacer la ligne 11 "pour chaque v de Adj[u]" par l'algorithme de la question 1(a) en ramplaçant s par u et "afficher" par les lignes 12 à 16 de PL.
   Complexité : $\mathcal{O}(n)$ pour les initialisations (lignes 1-8), $\mathcal{O}(n)$ pour les opérations de file (les n sommets sont enfilé/défilé une fois), $\mathcal{O}(n)$ pour le parcours de chaque ligne de la matrice pour déterminer les successeurs d'un sommet. Donc $\mathcal{O}(n+n^2) = \mathcal{O}(n^2)$.

   (b) Matrice d'incidence B[i,k] :
   Il faut remplacer la ligne 11 "pour chaque v de Adj[u]" par l'algorithme de la question 1(b) en ramplaçant s par u et "afficher" par les lignes 12 à 16 de PL.
   Complexité : $\mathcal{O}(n)$ pour les initialisations (lignes 1-8), $\mathcal{O}(m + n\deg(u))$ pour le sommet u, à sommer surtout les sommets u de la file ("tantque"). Cela fait $\mathcal{O}(n) + \sum_u \mathcal{O}(m + n\deg(u))$. Faire très attention ici : il y a une somme infinie avec des grand-O. Il faut borner le coût $\mathcal{O}(m+n\deg(u)) \leq \alpha(m+n\deg(u))+\beta$ et remplacer. Cela donne :

$$\begin{aligned} n + \sum_u (\alpha(m + n\deg(u)) + \beta) &= n + \alpha\sum_u m + \alpha\sum_u (n\deg(u)) + \alpha\sum_u \beta \\ &= n + \alpha nm + \alpha\beta n \;=\; \mathcal{O}(nm) \end{aligned}$$

   car α, β sont des constantes.

   (c) Listes de successeurs :
   Il n'y a rien à modifier, l'algorithme est donné pour les listes d'adjacence (ou successeurs).
   Complexité : $\mathcal{O}(n)$ pour les initialisations (lignes 1-8), $\mathcal{O}(\deg(u))$ pour le sommet u, à sommer surtout les sommets u de la file ("tantque"). Cela fait $\mathcal{O}(n+m)$.

Exercice 6 Matrice d'incidence : $\mathcal{O}(m(n+m))$

```
pour j allant de 1 à m faire:
    pour u allant de 1 à n faire:
        si B[u,e_j]=1 alors i=u
        si B[u,e_j]=-1 alors k=u
    pour p allant de 1 à m faire:
        si B[i,e_p]=-1 et B[k,e_p]=1 alors retourner VRAI
retourner FAUX
```