Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

# Introduction à l'informatique : TD 7

**Code UE : S14IN1I1**

Année 2019-20

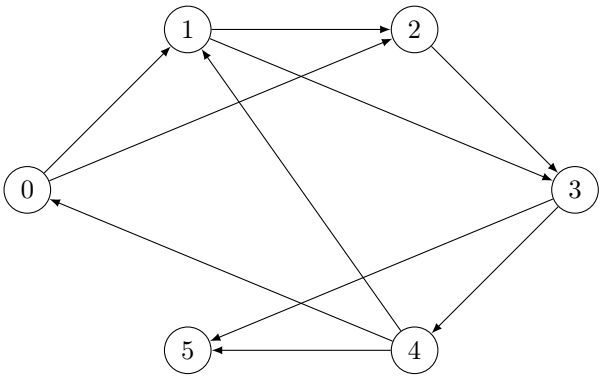
**Parcours de graphes**

**Exercice 1 (Parcours en largeur)** En cours, on a décrit l'algorithme de parcours en largeur dans un graphe orienté $G$ à partir d'un sommet $s$ :

```
fonction parcours_en_largeur(G, s) :
    n := nombre_sommets(G)
    H := graphe_vide(n)          #  graphe sans arcs
    F := ∅                       #  file vide
    couleur := tableau de longueur n rempli de 'blanc'
    couleur[s] := rouge
    enfiler(F, s)
    Tant que F ≠ ∅ faire
        u := défiler(F)
        Pour v de 0 à n − 1 faire
            Si (G[u,v] = 1 et couleur[v] = blanc) alors
                couleur[v] := rouge
                H[u,v] := 1
                enfiler(F, v)
            FinSi
        FinPour
        couleur[u] := vert
    FinTantQue
    retourner(H)                 #  graphe des chemins minimaux
```

Cet algorithme retourne le graphe $H$ qui ne contient que les arcs traversés en parcourant un chemin de $s$ à chacun des autres sommets accessibles.

1. Exécuter l'algorithme de parcours en largeur sur le graphe $G$ représenté ci-dessous à partir du sommet $s = 1$, en montrant toutes les étapes de l'algorithme (avec la coloration des sommets et l'état de la file dans les configurations intermédiaires). Représenter aussi le graphe $H$ retourné par la fonction.




2. Écrire en pseudo-code une fonction `calculer_chemin`$(H, s, t)$ qui prenne en argument le graphe des chemins minimaux $H$ construit par la fonction `parcours_en_largeur`$(G, s)$ et affiche le chemin en $H$ qui commence par le sommet source $s$ et se termine avec le sommet $t$. Pour simplifier, on pourra afficher les sommets du chemin en ordre inverse : par exemple, si le chemin entre $s$ et $t$ est $s \to u \to v \to t$, on pourra afficher « $t$ $v$ $u$ $s$ ».

**Exercice 2 (Le loup, le mouton, le chou et la bergère)** Du temps de Pagnol, une bergère veut traverser la rivière Durance avec un chou, un mouton et un loup. Malheureusement, pas de

pont à l'horizon et elle ne possède qu'une minuscule barque dans laquelle elle peut naviguer avec un seul de ses « compagnons » d'aventure. En sa présence, le mouton n'ose pas manger le chou, pas plus que le loup n'ose manger le mouton, mais ils n'hésiteraient pas à satisfaire leurs appétits si la bergère tournait le dos. Comment doit-elle s'y prendre pour amener tout le monde de l'autre côté de la rivière ? Utilisons un graphe pour l'aider !

1. Quels sont les configurations possibles de cette aventure ? On pourra les décrire en observant les personnages qui peuvent se trouver sur la rive de départ et la rive d'arrivée.
2. Modéliser alors le problème sous la forme d'un graphe dont les sommets sont les configurations possibles et les arêtes représentent l'évolution possible de l'aventure.
3. Résoudre le problème à l'aide d'un parcours du graphe précédent. Décrire à la bergère les allers-retours qu'elle doit faire. Combien de traversées la bergère doit-elle faire ? Existe-t-il plusieurs solutions avec ce nombre minimal de traversées ?

**Exercice 3 (Plus court chemin)**

On a vu en cours un algorithme permettant de calculer des plus courts chemins dans des graphes pondérés. On décrit un graphe (orienté) pondéré à l'aide d'une matrice (un tableau bi-dimensionnel) d'adjacence $G$ avec autant de lignes et de colonnes qu'il y a de sommets dans le graphe, et dont le coefficient pour $u$ et $v$ deux sommets vaut

$$G[u,v] = \begin{cases} +\infty & \text{s'il n'y a pas d'arcs de } u \text{ à } v \\ \text{poids de } u \to v & \text{sinon} \end{cases}$$

Le poids d'un chemin $u_0 \to u_1 \to u_2 \to \cdots \to u_{n-1} \to u_n$ du graphe (où $u_0 \to u_1$, $u_1 \to u_2$, …, $u_{n-1} \to u_n$ sont des arcs du graphe) est égal à la somme $G[u_0,u_1]+G[u_1,u_2]+\cdots+G[u_{n-1},u_n] = \sum_{i=0}^{n-1} G[u_i,u_{i+1}]$. Un plus court chemin de $u$ à $v$ est un chemin menant de $u$ à $v$ dont le poids est minimal parmi tous les chemins possibles.

En cours, nous avons exécuté l'algorithme de Dijkstra. En voici une description sous forme de pseudo-code (noter la ressemblance avec le parcours en largeur, même si nous n'avons plus de graphe $H$ mais deux nouveaux tableaux...) :

```
fonction dijkstra(G, s) :
    n := nombre_sommets(G)
    distance := tableau de taille n rempli de +∞    # aucun prédécesseur
    prédécesseur := tableau de taille n rempli de ⊥
    F := ∅                          #  file de priorité vide
    couleur := tableau de longueur n rempli de 'blanc'
    couleur[s] := rouge
    distance[s] := 0
    enfiler_priorité(F, s, 0)
    Tant que F ≠ ∅ faire
       (u, p) := défiler_prioritaire(F)
        Pour v de 0 à n − 1 faire
            Si (couleur[v] = blanc et G[u,v] ≠ +∞) alors
                      couleur[v] := rouge
                      prédécesseur[v] := u
                      distance[v] := p+G[u,v]
                      enfiler(F, v, p+G[u,v])
            Sinon Si (couleur[v] = rouge et p+G[u,v] < distance[v]) alors
                      mettre_à_jour_priorité(F, v, p+G[u,v])
                      prédécesseur[v] := u
                      distance[v] := p+G[u,v]
            FinSi
        FinPour
        couleur[u] := vert
    FinTantQue
    retourner(prédécesseur)         #  graphe des chemins minimaux
```

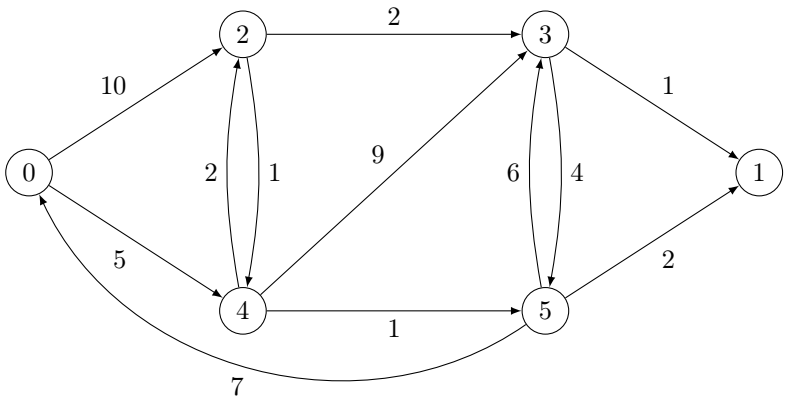
La file $F$ est une file de priorité, c'est-à-dire que chacun de ses éléments est associé à une priorité (un entier positif ici) :

— lorsqu'on insère un nouvel élément $s$ dans la file, on précise sa priorité $p \in \mathbf{N}$ à l'aide de la fonction `enfiler_priorité(F, s, p)` ;

— l'élément prioritaire est celui de **plus petite priorité** : on peut récupérer cet élément ainsi que sa priorité à l'aide de la fonction `(u, p) := défiler_prioritaire(F)` ;

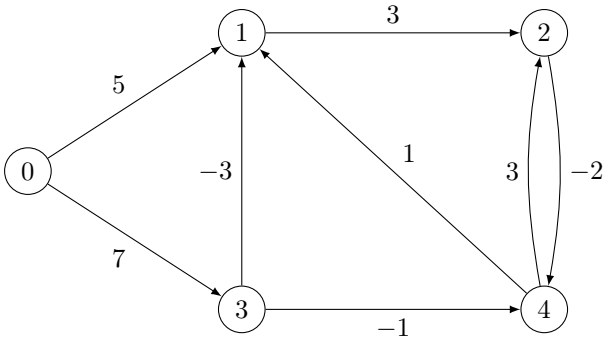
— on peut mettre à jour la priorité d'un élément $s$ de la file pour lui attribuer la nouvelle priorité $p$ dans $F$, à l'aide de la fonction `mettre_à_jour_priorité(F, s, p)`.

1. Exécuter l'algorithme de Dijkstra à partir du sommet 0, sur l'exemple ci-dessous :




2. En déduire un plus court chemin du sommet 0 au sommet 1.

3. En cours, nous avons étudié uniquement des graphes pondérés avec des poids entiers positifs ou nuls : pourtant, on pourrait imaginer des graphes avec des poids négatifs, par exemple si le poids de l'arc représente des échanges d'argent (vente ou achat de produits). Exécuter l'algorithme de Dijkstra sur l'exemple ci-dessous où plusieurs arcs ont des poids négatifs :




4. Qu'en déduisez-vous sur l'algorithme de Dijkstra ?