Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 7

Code UE : SPO1U57

Année 2023–2024

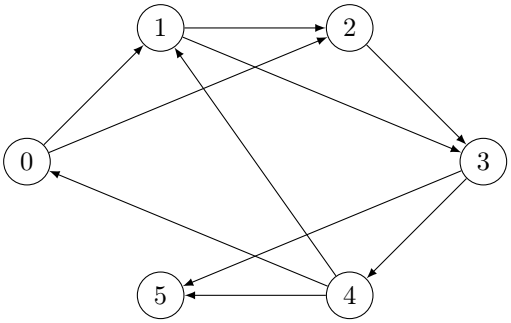
Parcours de graphes

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 :

```
def parcours_en_largeur(G, s):
    n = len(G)
    H = nouveau_graphe(n)          # graphe vide
    F = nouvelle_file()            # file vide
    couleur = ['blanc'] * n        # n cases blanches
    couleur[s] = 'rouge'
    enfiler(F, s)
    while not est_vide(F):
        u = défiler(F)
        for v in range(n):
            if G[u][v] == 1 and couleur[v] == 'blanc':
                couleur[v] = 'rouge'
                enfiler(F, v)
                H[u][v] = 1
        couleur[u] = 'vert'
    return 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 $s = 0$, 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 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 (Plus court chemin) Lorsque les graphes sont équipés de poids (sur les arcs), il peut être intéressant de calculer des plus courts chemins entre des paires de sommets. On décrit un graphe (orienté) pondéré à l'aide d'une matrice d'adjacence M 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

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

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

$$M_{u_0,u_1} + M_{u_1,u_2} + \cdots + M_{u_{n-1},u_n} = \sum_{i=0}^{n-1} M_{u_i,u_{i+1}}$$

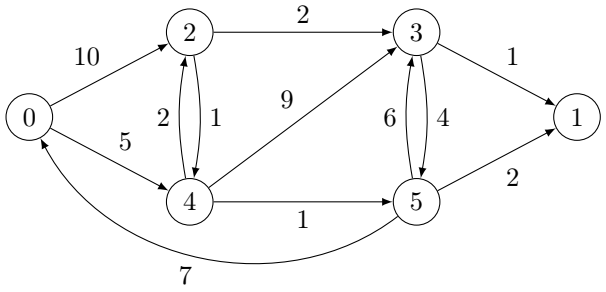
Un plus court chemin de u à v est un chemin menant de u à v dont le poids est minimal parmi tous les chemins possibles.

Voici une description de l'algorithme de Dijkstra :

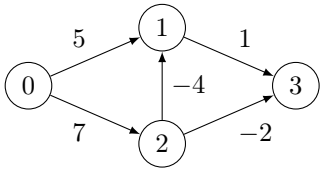
— colorier le sommet de départ en rouge et les autres sommets en blanc
— attribuer au sommet de départ la distance provisoire 0
— attribuer aux autres sommets la distance provisoire $+\infty$
— attribuer à chaque sommet le prédécesseur nul
— créer une file de priorité (dont la priorité est la distance provisoire)
— insérer le sommet de départ dans la file
— tant que la file n'est pas vide :
  — extraire de la file le sommet u ayant la distance minimale
  — pour chaque sommet v adjacent à u :
    — si v est blanc alors colorier v en rouge et enfiler v dans la file
    — si la distance de u + le poids de (u, v) est inférieur à la distance de v alors colorier v en rouge, mettre u comme prédécesseur de v et attribuer à v la distance de u + le poids de l'arc (u, v)
— renvoyer la liste des prédécesseurs

La différence notable entre le parcours en largeur et l'algorithme de Dijkstra réside dans l'utilisation d'une file de priorité qui nécessite de maintenir les estimations des distances et de mettre à jour les arcs rouges : pour cela, plutôt que de maintenir un graphe `H` comme précédemment, on associe à chaque sommet v rouge son prédécesseur dans le graphe `H`, c'est-à-dire l'unique sommet u tel que (u, v) est un arc rouge de `H`. Initialement, on choisit d'attribuer à tout sommet 0 comme prédécesseur.

1. Exécuter l'algorithme de Dijkstra sur l'exemple ci-dessous en partant de la source $s = 0$:



2. En déduire un plus court chemin du sommet 0 au sommet 1.
3. Jusque-là, 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 ?