Aix*Marseille université

Domaine Sciences et Technologies
Portail 1 René Descartes

Introduction à l'informatique : TD 7

Code UE : S14IN1I1

Année 2018-19

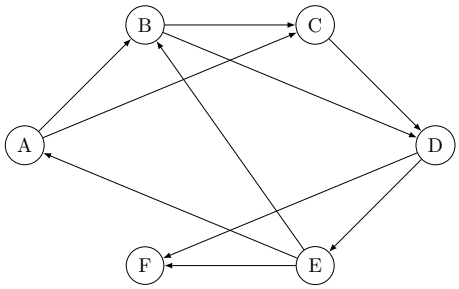
Parcours de graphes

Exercice 1 (Parcours en largeur) En cours, nous avons décrit l'algorithme de parcours en largeur d'un graphe orienté G à partir d'un sommet s :

```
Fonction parcours_largeur(d G[1..n, 1..n]: tab_entier, d s: entier): tab_entier
    i: entier;    H: tab_entier;
    F: file_entier;    couleur[1..n]: tab_couleur;
Début
    Pour i de 1 à n faire    couleur[i] := blanc;    Fin faire
    couleur[s] := vert;    F := ∅;
    enfiler(F, s);
    Tant que (F ≠ ∅) faire
       u := défiler(F);    couleur[u] := rouge;
       Pour v de 1 à n faire
          Si (G[u, v] = 1 et couleur[v] = blanc) alors
             couleur[v] := vert;
             H[u, v] := 1;
             enfiler(F, v);
          Fin si
       Fin faire
       couleur[u] := doré;
    Fin faire
    retour H;
Fin
```

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

1. Exécutez l'algorithme de parcours en largeur sur le graphe G représenté ci-dessous à partir du sommet $s = A$, 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. Écrivez en pseudo-code une procédure `Procédure calculer_chemin(d` $H[1..n, 1..n]$`: tab_entier, d` s`,` t`: entier)` qui prend en argument le graphe des chemins minimaux H construit par la fonction `Fonction parcours_largeur(d` $G[1..n, 1..n]$`: tab_entier, d` s`: entier): tab_entier` et affiche à l'écran le chemin de H qui commence par le sommet s et se termine par 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, la chèvre, le chou et la bergère) Du temps de Pagnol, une bergère veut traverser la rivière Durance avec un chou, une chèvre 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, la chèvre n'ose pas manger le chou, pas plus que le loup n'ose manger la chèvre, 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. Quelles 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. Modélisez 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. Résolvez le problème à l'aide d'un parcours du graphe précédent. Décrivez 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

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

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 (notez la ressemblance avec le parcours en largeur, bien que nous n'ayons plus de graphe H mais deux nouveaux tableaux stockant respectivement les distances et les prédécesseurs) :

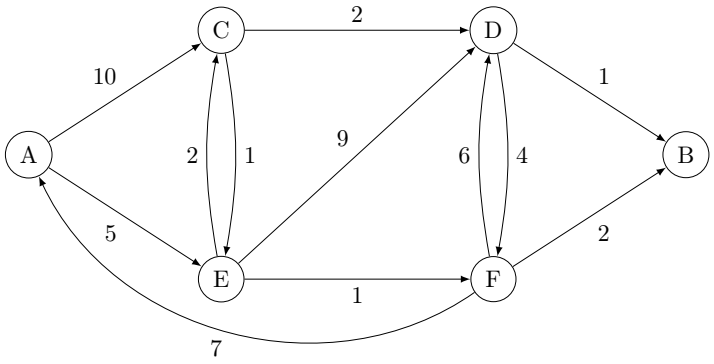
```
Procédure dijkstra(d G[1..n, 1..n]: tab_entier, d s: entier, r dist[1..n], pred[1..n]:
tab_entier)
    coul[1..n]: tab_entier;
    F: file_entier;
    u, v, p: entier;
Début
    F := init_file();                          crée une file de priorité vide de taille n
    coul := init_couleur();          crée un tab_couleur de taille n avec blanc partout
    dist := init_dist();                crée un tab_entier de taille n avec ∞ partout
    pred := init_pred();               crée un tab_entier de taille n avec −1 partout
    coul[s] := rouge;
    dist[s] := 0;
    enfiler_avec_priorite(F, s, 0);
    Tant que (F ≠ ∅) faire
       (u, p) := defiler_avec_priorite(F);
       Pour v de 1 à n faire
          Si (coul[v] = blanc et G[u, v] ≠ ∞) alors
             coul[v] := rouge;
             pred[v] := u;
             dist[v] := p + G[u, v];
             enfiler_avec_priorite(F, v, p + G[u, v]);
          Sinon si (coul[v] = rouge et p + G[u, v] < dist[v]) alors
             pred[v] := u;
```

```
            dist[v] := p + G[u, v];
            mettre_a_jour_priorite(F, v, p + G[u, v]);
          Fin si
        Fin faire
        coul[u] := vert;
    Fin faire
Fin
```

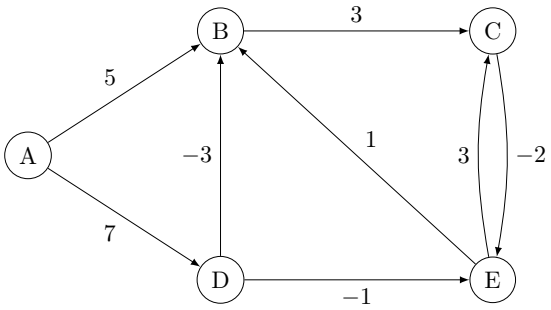
La file F est une file de priorité, c'est-à-dire que chacun de ses éléments est associé à une priorité (un entier positif ici) :

- 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_avec_priorite`(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 $(u, p) :=$ `defiler_avec_priorite`(F) ;
- on peut mettre à jour la priorité d'un élément v de la file pour lui attribuer la nouvelle priorité p *si celle-ci est inférieure à la priorité initiale* de v dans F, à l'aide de la fonction `mettre_a_jour_priorite`(F, v, p).

1. Exécutez l'algorithme de Dijkstra en partant du sommet A sur l'exemple ci-dessous :



2. Déduisez-en un plus court chemin du sommet A au sommet B.
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). Exécutez l'algorithme de Dijkstra sur l'exemple ci-dessous en partant du sommet A où plusieurs arcs ont des poids négatifs :



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