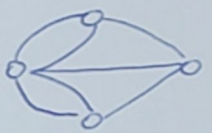
7 - Accessibilité et chemins dans un graphe. Applications.

Prérequis : Programmation C et OCaml Niveau : MP2I

Introduction :
- Omniprésence des graphes (cartes GPS, groupes sociaux dans les réseaux sociaux, ...)
- Énigme introductive : ponts de Königsberg
  ↳ Existe-t-il un chemin eulérien ?



I) Graphes non pondérés :

1.) Graphes non orientés :

Définition 1 : On appelle graphe non orienté un couple (V,E) tel que V est un ensemble fini et $E \subseteq \mathcal{P}_2(V)$.
Les éléments de V sont appelés sommets et ceux de E arêtes.

Définition 2 : On appelle voisin d'un sommet u d'un graphe $G=(V,E)$ tout sommet $v \in V$ tel que $\{u,v\} \in E$.
On appelle alors degré de u le nombre de ses voisins.

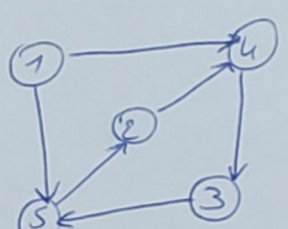
2) Graphes orientés :

Définition 3 : On appelle graphe orienté un couple (V,E) tel que V est un ensemble fini, et $E \subseteq V \times V$.
Les éléments de V sont appelés sommets et ceux de E arcs.

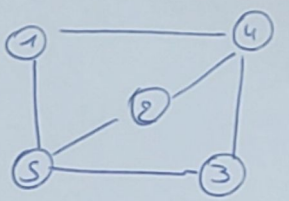
Définition 4 : On appelle voisin d'un sommet u d'un graphe $G=(V,E)$ tout sommet $v \in V$ tel que $(u,v) \in E$.
On appelle degré entrant de u le cardinal de $\{v \in V \mid (v,u) \in E\}$ et degré sortant de u le cardinal de $\{v \in V \mid (u,v) \in E\}$.

3.) Représentations et implémentations :

Exemple 5 :



Graphe G_1 orienté



Graphe G_2 non orienté

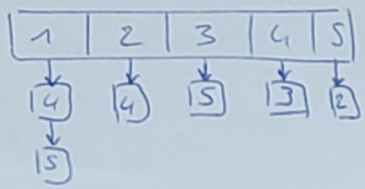
Proposition 6 : On peut représenter les graphes par des matrices d'adjacence telles que : pour tout graphe $G=(V,E)$, la matrice d'adjacence M de G est une matrice carrée indicée par V où $m_{i,j} = \begin{cases} 1 \text{ si } (i,j) \in E \\ 0 \text{ sinon.} \end{cases}$

Proposition 7 : On peut représenter les graphes par des listes d'adjacence telles que : pour tout graphe $G=(V,E)$, la liste d'adjacence q de G est un tableau indicé par V où q_i est la liste des voisins de i dans G.

Exemple 8 :

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Matrice d'adjacence de G_1



Remarque 9 : Tout graphe G non orienté peut être associé à un graphe orienté G' où $V = V'$ et $E' = \{(u,v), (v,u) \mid \{u,v\} \in E\}$.

Activité 10 : Implémentation en OCaml des fonctions usuelles de manipulation des graphes sous les deux représentations présentées.

4.) Accessibilité :

Définition 11 : Un **chemin** de s à t est une suite finie non vide de sommets $s_0, \dots, s_n$ de V tels que : $s_0 = s$, $s_n = t$ et $\forall i \in [\![0; n-1]\!]$, $(s_i, s_{i+1}) \in E$.

Définition 12 : On dit qu'un graphe non orienté G est **connexe** si pour tout $u, v \in V$ distincts, il existe un chemin de u à v dans G.

On dit qu'un graphe orienté est **fortement connexe** si pour tout $u, v \in V$ distincts, il existe un chemin de u à v et un chemin de v à u dans G.

Définition 13 : Un chemin $(s_0, \dots, s_n)$ est dit **élémentaire** si les (s_i) sont deux à deux distincts.

Lemme 14 de König : On peut extraire un chemin élémentaire de tout chemin.

Définition 15 : On dit qu'un sommet t est **accessible** depuis s s'il existe un chemin de s à t.

5.) Parcours :

Définition 16 : On appelle **parcours** d'un graphe G connexe à partir de l'un de ses sommets s une liste de ses sommets L telle que :
- s est le premier sommet de L ;
- tout sommet de G apparaît une unique fois dans L ;
- pour tout sommet v de L différent de s, il existe un sommet u tel que u est placé avant v dans L et $\{u, v\} \in E$.

Algorithme 17 : Parcours générique de G à partir de s

Entrées : graphe G et sommet initial s_0 de G.
a_traiter ← {s_0}
ouverts ← [false, ..., false]
ouverts [s_0] ← true

Complexité : $O(|V|+|E|)$

```
Tant que a_traiter est non vide :
    s ← sortir un élément de a_traiter
    Pour tout voisin t de s :
        Si non ouverts [t] :
            a_traiter ← a_traiter ∪ {t}
            ouverts [t] ← true
```

Propriété 18 : Prendre les sommets par ordre de traitement dans le parcours générique constitue un parcours de G à partir de s.

Remarques 19 : · En implémentant la variable a_traiter dans le parcours générique par une pile, on obtient le parcours en profondeur, et en l'implémentant par une file, on obtient le parcours en largeur.

6.) Applications :

DEV 1 : Tri topologique pour les graphes orientés sans circuit

Algorithme 20 : Algorithme de Kosaraju de calcul des composantes fortement connexes d'un graphe orienté

DEV 2 : Lien entre composantes fortement connexes et le problème 2-SAT.

II.) Graphes pondérés :

1.) Généralités :

Définition 21 : On appelle graphe pondéré tout graphe $G=(V,E)$ muni d'une fonction $c : E \to \mathbb{R}$ appelée valuation. Le coût/poids d'un arc/d'une arête e est alors $c(e)$, et le coût d'un chemin $\gamma = (s_0, s_1, \dots, s_n)$ est $c(\gamma) = \sum_{i=0}^{n-1} c(s_i, s_{i+1})$.

Problème 22 : Problème du plus court chemin
Entrées : graphe pondéré G, s et t deux sommets de G
Sortie : un chemin de coût minimal de s à t.

Propriété 23 : Soient s et t deux sommets d'un graphe pondéré G, et $s=s_0, s_1, \dots, s_n=t$ un plus court chemin de s à t. Alors, pour tout $i \in [\![1, n-1]\!]$, $(s_0, \dots, s_i)$ est un plus chemin de s à s_i, et $(s_i, \dots, s_n)$ est un plus court chemin de s_i à t.

Remarque 24 : On remarque que s'il existe un circuit accessible de poids strictement négatif (appelé absorbant), alors un tel chemin de poids minimal n'existe pas. On se place dans l'hypothèse où il n'y a pas de tel cycle.

Propriété 25 : Si G n'a pas de circuit absorbant, alors pour tous s et t sommets de G tels que t est accessible depuis s, il existe un chemin de coût minimal de s à t.

2.) Plus courts chemins depuis un sommet :

Algorithme 26 : Algorithme de Dijkstra de calcul des plus courts chemins depuis un sommet vers tous les autres sommets accessibles d'un graphe G pondéré positivement sur tous les arcs/arêtes.

Propriété 27 : L'algorithme de Dijkstra est correct et a une complexité de $O(|E| \times \log(|V|))$ en utilisant une structure de tas-min.

3.) Plus courts chemins pour tous couples :

Algorithme 28 : Algorithme de Bellman-Ford de calcul des plus courts chemins pour tous les couples (s,t) d'un graphe pondéré G.

Propriété 29 : L'algorithme de Bellman-Ford est correct et a une complexité temporelle de $O(|V| \times (|V|+|E|))$.