Algorithmique — M1

Corrigé très succinct et partiel du partiel du 30 novembre 2007, alpha 2

Exercice 1 : diamètre d'un graphe

On considère un graphe $G = (V, E)$ non orienté non pondéré. La *distance* entre deux sommets est la longueur (en nombre d'arêtes) d'un plus court chemin les reliant. Le *diamètre* d'un graphe est la plus grande distance entre deux sommets. Proposez un algorithme de calcul de diamètre du graphe, et analysez sa complexité (dans le meilleur et le pire des cas).

Une solution

1. Donner à chaque arête le poids 1. Ainsi la matrice des poids W coïncide avec la matrice d' adjacence.
2. Appliquer Floyd-Warshall, obtenir la matrice M de distances entre tous les couples de points.
3. Rechercher l'élément maximal de cette matrice. C' est la distance max entre 2 sommets, donc le diamètre.

Analyse de complexité (en supposant que le graphe est donné par ça matrice d'adjacence). Floyd-Warshall : $O(V^3)$. Recherche de max dans une matrice de taille $V \times V$ nécessite encore $O(V^2)$ opération. Total= $O(V^3)$.

Exercice 2 : taux de change optimal

On souhaite convertir de l'argent d'une devise dans une autre. Le problème est que toutes les conversions ne sont pas possibles : pour deux monnaies A et B, on peut parfois convertir de l'argent de A en B, parfois non. On considère donc un *graphe de change* $G = (V, E)$ entre monnaies donnant les conversions possibles. Ce graphe est orienté (parfois on peut convertir A en B mais pas B en A).

La *fonction de change* est une fonction c telle que
- une somme S en monnaie A vaut $S.c(A, B)$ en monnaie B (les taxes éventuelles sont incluses).
- $c(A, B)$ est défini si et seulement si (A, B) est un arc du graphe de change.
- $c(A, B) > 0$

Le *graphe de change étendu* est le graphe $G' = (V, E, c)$ pondéré par la fonction de change. Une *séquence de change* est la conversion d'une monnaie A_1 en monnaie A_k en passant par les monnaies intermédiaires $A_2 ... A_{k-1}$ (en supposant bien sûr toutes ces conversions possibles). Il lui correspond un chemin dans le graphe de change.

Question 1

Quel est le taux de change de A_1 en A_k dans une séquence de change $A_1, A_2, \ldots A_k$?

Réponse

$$\prod_{i=1}^{k-1} c(A_i, A_{i+1})$$

Question 2

Dans quelle condition (exprimée sur G') quelqu'un peut-il devenir *infiniment riche* en changeant de l'argent ?

Réponse

Si et seulement si il existe un cycle $A_1, A_2, ..., A_k$ (avec $A_k = A_1$ dans le graphe, tels que son taux de change (de A_1 vers A_1) est supérieur à 1. C'est-à-dire

$$\prod_{i=1}^{k-1} c(A_i, A_{i+1}) > 1$$

Étant données deux séquences de change différentes de la monnaie A en la monnaie B, la meilleure des deux est celle qui a le taux le plus élevé.

Question 3

Supposons que l'on connaisse une séquence de change S_1 de la monnaie A en la monnaie B, d'une part, et une séquence S_2 de la monnaie A en la monnaie C d'autre part. Supposons que l'arc (C, B) existe, dans le graphe de change. Écrivez une condition de *relaxation* en comparant les taux des séquences S_1 d'une part, S_2 puis (C, B) d'autre part, et gardant la meilleure.

Réponse

Soit $d[X]$ le meilleur taux de change connu à partir de la monnaie A vers la monnaie X. On utilise la couleur rouge pour les éléments différents de l'algo du cours.

```
RelaxTaux(C,B)
        si d[B]<d[C]* c(C,B)alors
                d[B] :=d[C]*c(C,B)
```

Question 4

Écrivez une version modifiée de l'algorithme de Bellman-Ford, utilisant cette condition de relaxation modifiée, donnant les meilleurs taux de change d'une monnaie A vers toutes les autres.

Réponse

```
BFtaux(G,c,A)
        d[V] : réel
        pour tout X dans V    faire
                              d[X]=0
        d[A]=1
        pour i=1 à |V|-1      faire
                              pour tout arc (X,Y) du G faire
                                          RelaxTaux(X,Y)
```

Question 5

Même question, en modifiant l'algorithme de Dijkstra.

Réponse

```
DijkstraTaux(G,c,A)
        d[V] : réel
        pour tout X dans V   faire
                             d[X]=0
        d[A]=1
        Q=FileDePriorité(V,d)
        Tant que Q non vide  faire
                             X=ExtraireMax(Q)
                             pour tout Y dans Adj(X)faire
                                         RelaxTaux(X,Y)
```

Question 6

Dans quelles conditions peut-on utiliser

- Bellman-Ford modifié (de la question 4) ?
- Dijkstra modifié (de la question 5) ?

Donnez des contre-exemples quand ça ne marche pas, des éléments de preuve (en 10 lignes maximum !) quand ça marche.

Réponse partielle

BF modifié : s'il n'y a pas de cycle au produit de fonctions de change >1 (voir question 2).

Dijkstra modifié : si pour toutes le devises $c(A, B) \leq 1$.

Exercice 3 : plus long chemin élémentaire

On considère des graphes orientés. Un chemin *élémentaire* est un chemin qui passe au plus une fois par chaque sommet.

Question 1

Quel est le plus long chemin élémentaire dans le graphe *complet* à n sommets $v_1, \ldots v_n$? Ce graphe est défini de sorte que pour tous sommets $v_i \neq v_j$ il existe une arête (v_i, v_j). Comparez au diamètre de ce même graphe.

Réponse

Diamètre 1 (chaque couple de sommets est relié par une arête). Longueur max de chemin élémentaire : $n - 1$ (pour $v_1, \ldots v_n$).

Question 2

Proposez un algorithme qui calcule le plus long chemin élémentaire dans un graphe. On ne demande pas *le* meilleur algorithme, mais juste un algorithme "naïf" qui marche.

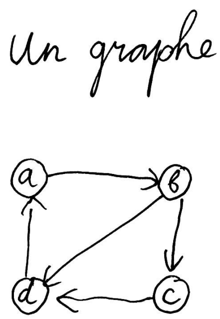
Réponse

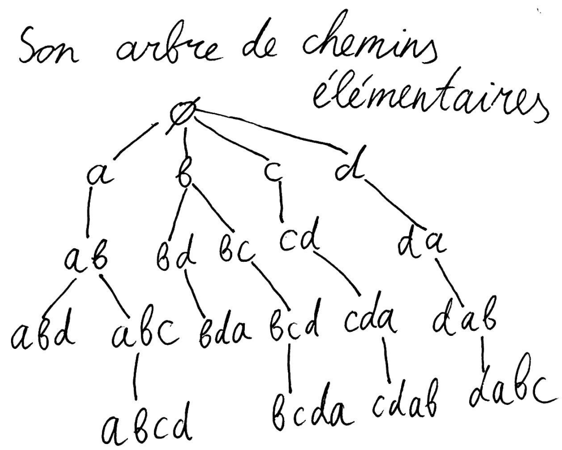
Tous les chemins élémentaires forment un arbre A, où les enfants d'un chemin S de longueur n sont les chemins de longueur $n+1$ de la forme Sv. On remarque que v doit satisfaire les conditions :

– $v \notin S$ (pour que le chemin reste élémentaire ;
– Soit last(S)=u. Il faut que $v \in Adj(u)$ pour que Sv soit un vrai chemin.

La racine est le chemin vide $\emptyset$, ses enfants - les chemins singletons $v_1, \ldots, v_V$.

Exemple





Notre algorithme va construire et au même temps parcourir l'arbre A en appliquant l'approche BFS. Au début on met dans la file tous les chemins singletons. A chaque instant on mémorise le plus long chemin vu auparavant (Champion) et le nombre de sommets dans ce chemin LMax.

```
CheminLong
        Champion=∅; LMax=0
        Q=nouvelle File(V)
        tant que Q non vide faire
                S= Q.défiler()
                si NbSommets(S)>Lmax alors
                        Champion=S
                        LMax=NbSommets(S)
                u=last (S)
                pour tout v dans Adj(u) faire
                        si v n'est pas dans S alors
                                Q.enfiler(Sv)

        retourner Champion
```

Question 3

Analysez la complexité de votre algorithme.

Réponse

On énumère tous les chemins élémentaires. Il y en a au pire cas

– 1 chemin vide
– V chemins de longueur 0 (un seul sommet),
– V(V-1) chemins de longueur 1 (un sommet, puis un autre sommet),

- V(V-1)(V-2) chemins de longueur 2 (un sommet, un autre sommet, et encore un autre),
- ...
- V ! chemins de longueur V (toutes les permutations)

Ce pire cas se réalise pour le graphe complet. On peut majorer la somme par $(V + 1)!$ (pourquoi ?) Notre algorithme visite (construit) chaque chemin une fois. Pour visiter un chemin il faut faire O(V) opérations. On obtient la complexité $O((V + 1)!V)$, ce qui est pire qu'exponentiel en nombre de sommets du graphe.