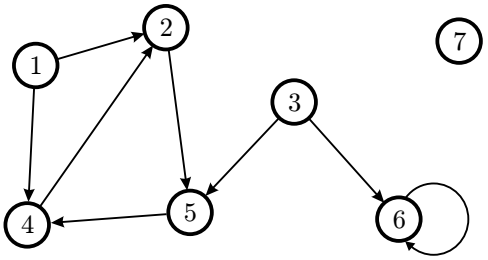
IN 101 - TD 13
17 décembre 2010

Énoncé

ENSTA ParisTech

Nous allons nous intéresser à des algorithmes travaillant sur la représentation des graphes par matrices d'adjacence (ou par liste de successeurs pour les plus forts). Rappelons que la *matrice d'adjacence* d'un graphe G à n sommets est la matrice carrée d'ordre n telle que $M_{i,j} = 1$ si (i, j) est un arc de G et $M_{i,j} = 0$ sinon.



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

Exercice 1 : Fermeture transitive d'un graphe

Le but du calcul de la fermeture transitive d'un graphe est de déterminer pour tout couple de sommets s'il existe ou non un chemin reliant le premier au second. Un algorithme efficace (en $\Theta(n^3)$) est le suivant :

ROY-WARSHALL(G)
  soit n le nombre de sommets de G
  soit M la matrice d'adjacence de G
  soit H un graphe à n sommets de matrice d'adjacence N (graphe retourné)
  pour $i \leftarrow 1$ **à** n, **faire**
    pour $j \leftarrow 1$ **à** n, **faire** // copie de M dans H
      si $i = j$, **alors** // avec ajout de la réflexivité.
        $N_{i,j} \leftarrow 1$
      sinon
        $N_{i,j} \leftarrow M_{i,j}$
  pour $k \leftarrow 1$ **à** n, **faire**
    pour $i \leftarrow 1$ **à** n, **faire**
      pour $j \leftarrow 1$ **à** n, **faire**
        $N_{i,j} \leftarrow N_{i,j} \vee (N_{i,k} \wedge N_{k,j})$
  renvoyer H

On définit le type `graph` de la façon suivante :

```
1 typedef struct {
2   int n;
3   int** mat;
4 } graph;
```

On rappelle que la création dynamique d'un tableau à deux dimensions s'effectue en commençant par l'allocation du tableau des pointeurs vers les lignes, suivie de l'allocation de chacune des lignes du tableau. Au passage, on initialise la matrice à 0 : le graphe n'a aucune arrête au départ. On obtient le code C suivant :

```
1  graph* init_graph (int nb) {
2    int i,j;
3    graph* G = (graph*) malloc(sizeof(graph));
4    int** matrix = (int**) malloc(nb * sizeof(int*));
5    for (i=0; i<nb; i++) {
6      matrix[i] = (int*) malloc(nb * sizeof(int));
7      for (j=0; j<nb; j++) {
8        matrix[i][j] = 0;
9      }
10   }
11   G->n = nb;
12   G->mat = matrix;
13   return G;
14 }
```

1.a] Programmez l'algorithme de Roy-Warshall tel qu'il est décrit ci-dessus.

1.b] Programmez une fonction ayant pour signature `void print_graph(graph* G)` qui affiche la matrice d'adjacence d'un graphe. Dans la fonction `main` calculez la fermeture transitive du graphe dessiné en première page (il faut entrer les coefficients de la matrice d'adjacence à la main) et faites afficher sa matrice d'adjacence. Vérifiez « à la main » que le résultat est correct.

1.c] En vous inspirant de la fonction `init_graph` ci-dessus, écrivez une fonction dont le prototype est `graph* read_graph(char* filename)` qui permet de lire la matrice d'adjacence d'un graphe dans un fichier (dont le nom est `filename`). Le fichier contenant les données à lire contient un entier correspondant au nombre n de sommets du graphe, puis une suite de n^2 entiers correspondants aux coefficients de la matrice d'adjacence du graphe : soit des 0 et des 1 pour une matrice d'adjacence simple, soit des entiers pour des graphes pondérés. Par exemple, il peut ressembler à cela :

```
3
0 1 1 0 0 0 1 1 0
```

1.d] Utilisez votre fonction précédente pour lire le graphe qui se trouve (sur les machines ENSTA) dans le fichier `"~finiasz/graph"` et vérifiez que vous obtenez bien le bon graphe et la bonne fermeture transitive.

1.e] Programmez une fonction `void print_successors(graph* G)` qui affiche un graphe sous forme de "liste de successeurs" : une ligne par sommet contenant le numéro du sommet et l'ensemble des successeurs de ce sommet. Affichez ainsi le graphe et sa fermeture transitive.

Exercice 2 : Recherche de plus courts chemins dans un graphe

Considérons maintenant le problème de la recherche de chemins les plus courts dans le cas d'un graphe dont chaque arc a une *longueur*, c'est-à-dire un poids associé. Pour cela, on utilise l'algorithme de Aho, Hopcroft, Ullman rappelé ci-dessous avec comme structure algébrique

$\mathcal{S}(\sqcup, \odot, \emptyset, \varepsilon) = R^+ \cup \infty(\min, +, \infty, 0)$ (pour tout x, $x^* = 0$) [voir poly, p. 104] ; (à noter qu'avec cette structure algébrique, l'algorithme est aussi connu sous le nom d'*algorithme de Floyd-Warshall*). La notion de matrice d'adjacence est généralisée de la manière suivante : s'il existe un arc reliant le sommet i au sommet j, l'entrée correspondante de la matrice prend comme valeur le poids de cet arc. S'il n'y a pas d'arc, l'entrée de la matrice prend la valeur ∞.

AHO-HOPCROFT-ULLMAN(G)
  soit n le nombre de sommets de G
  soit M la matrice d'adjacence (généralisée) de G
  soit H un graphe à n sommets, de matrice d'adjacence N
  pour $i \leftarrow 1$ **à** n, **faire**
    pour $j \leftarrow 1$ **à** n, **faire**
      si $i = j$, **alors**
        $N_{i,j} \leftarrow 0$
      sinon
        $N_{i,j} \leftarrow M_{i,j}$
  pour $k \leftarrow 1$ **à** n, **faire**
    pour $i \leftarrow 1$ **à** n, **faire**
      pour $j \leftarrow 1$ **à** n, **faire**
        $N_{i,j} \leftarrow \min(N_{i,j}, (N_{i,k} + N_{k,j}))$
  renvoyer H

2.a] Programmez l'algorithme de Aho-Hopcroft-Ullman ainsi qu'une nouvelle fonction d'initialisation `init_graph2` qui initialise la matrice du graphe avec ∞ dans chaque case (on peut prendre la convention $\infty = -1$ car ici les poids sont toujours positifs). Utilisez ces deux fonctions pour calculer les plus courts chemins dans le graphe de la première page. Affichez le résultat et vérifiez que c'est bien correct.

2.b] Calculez à partir des nombres de stations indiqués dans le tableau ci-dessous le nombre minimal de stations de metro permettant d'aller d'une station à l'autre. Les données de ce tableau sont contenues dans le fichier `"~finiasz/metro"` que vous pourrez lire avec votre fonction `read_graph`. La convention est que -1 correspond à l'infini (pas de chemin). ⚠ Il s'agit ici d'un graphe non orienté, il faut donc commencer par symétriser la matrice d'adjacence !

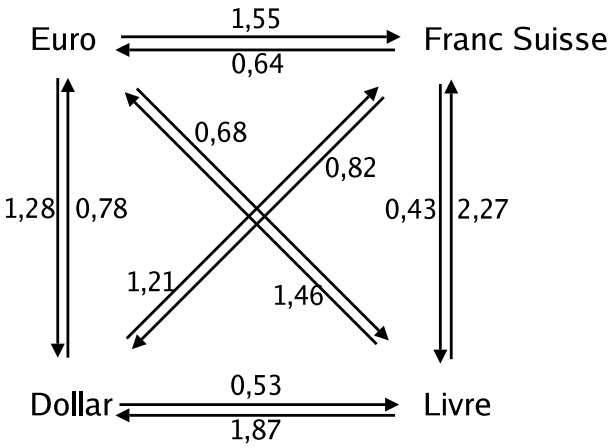
	Nation	Chatelet	Étoile	Montparnasse	Concorde	Motte-Piquet	Balard	République	Opéra	Place d'Italie	Bastille
Nation								6		9	3
Chatelet				7	3						3
Étoile					4	7					
Montparnasse					7	4				7	
Concorde						4			2		
Motte-Piquet							5				
Balard											
République									5		4
Opéra											
Place d'Italie											5
Bastille											

2.c] Dans le poly (pages 105-106) est expliqué comment modifier l'algorithme de Aho-Hopcroft-Ullman pour, en plus de calculer le coût du chemin optimal, permettre de retrouver facilement ce chemin optimal. Modifiez l'algorithme précédent afin de savoir par quelles correspondances passe le plus court chemin allant de Balard à Place d'Italie (par exemple).

2.d] (Pour ceux qui vont vite) Programmez une structure de liste de successeurs (avec des arcs pondérés) et une fonction permettant de convertir une matrice d'adjacence en cette structure. Ensuite, programmez l'algorithme de Dijkstra vu en cours (vous pouvez réutiliser la structure de tas du TD 12) pour calculer les plus courts chemins entre Balard et toutes les autres stations. Vérifiez que les distances trouvées et les correspondances sont bien les mêmes qu'à la question précédente.

Exercice 3 : Spéculation monétaire

Considérons une application bien plus attrayante de l'algorithme précédent. On forme un graphe dont chaque sommet représente une devise (Euro, Dollar US, Livre, Franc Suisse, ...). Ce graphe est complet, c'est-à-dire que chaque paire de sommets est reliée. Chaque arc est de plus pondéré par le taux de change entre les deux monnaies correspondantes.



3.a] Écrivez un programme recherchant une chaîne de conversion entre monnaies permettant de s'enrichir (vous devez pour cela récrire la fonction Aho-Hopcroft-Ullman pour qu'elle fasse des multiplications au lieu des additions et prenne un max au lieu d'un min, et aussi qu'elle prenne en compte les boucles : la fonction `Star` dans le version générale de AHU présentée dans le poly). Notez que la structure `graph` doit être changée pour gérer des matrices de `double` maintenant. Appliquez le programme au graphe précédent. Remarquez qu'une petite erreur de conversion permet de trouver un moyen de s'enrichir infiniment (augmentez pour cela très légèrement l'un des taux de change).