Algorithmique
Partiel n° 2

Info-Spé – Epita

D.S. 313424.3 BW (4 mai 2010 - 09 :00)

Consignes (à lire) :

□ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.

– Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
– Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
– Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
– Aucune réponse au crayon de papier ne sera corrigée.

□ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.

□ **Les algorithmes :**

– Tout algorithme doit être écrit dans le langage Algo (pas de C, Caml ou autre).
– Tout code Algo non indenté ne sera pas corrigé.
– Tout ce dont vous avez besoin (types, routines) est indiqué en **annexe** (dernière page) !

□ Durée : 3h00

Exercice 1 (Dénombrement et Graphes non orientés – *6 points*)

1. Déterminer **(vous justifierez votre réponse)** le nombre d'arêtes d'un graphe simple non orienté complet de n sommets dans les deux cas suivants :
   (a) Si le graphe n'admet pas d'arêtes réflexives.
   (b) Si le graphe admet des arêtes réflexives.
2. Peut-on construire un graphe non orienté simple (aucune arête réflexive et pas plus d'une arête entre deux sommets) ayant :
   (a) 4 sommets et 7 arêtes ?
   (b) 5 sommets et 9 arêtes ?
   (c) 10 sommets et 46 arêtes ?
3. En déduire une règle entre le nombre de sommets n et le nombre d'arêtes p permettant de savoir s'il est possible de construire un graphe non orienté simple de n sommets et p arêtes.

Exercice 2 (ARM et autres... – *4 points*)

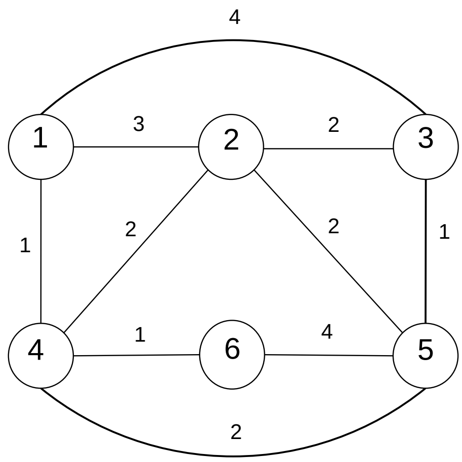

Fig. 1 – Graphe non orienté valué

1. Qu'est-ce qu'un arbre de recouvrement d'un graphe G non orienté connexe ?
2. Combien d'arbres de recouvrement minimum possède le graphe de la figure 1 ? Justifiez.
3. Tracer un arm du graphe de la figure 1.
4. Rappeler pourquoi tout graphe non orienté valué connexe admet au moins un arbre couvrant minimum.
5. Donner un exemple d'application au problème des arbres de recouvrement de poids minimum.

Exercice 3 (Plus court chemin et parcours profondeur – *17 points*)

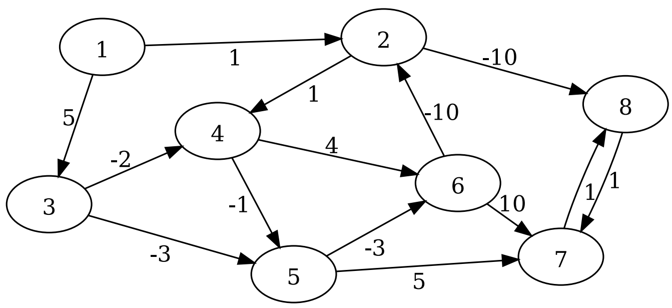

Fig. 2 – Graphe G_1

Nous allons nous intéresser à la recherche de plus court chemin dans des graphes orientés avec coûts quelconques. Notre problème ici est qu'il n'y a aucune garantie sur l'absence de circuit dans les graphes considérés. Plutôt que de chercher un algorithme résistant aux circuits, nous préférerons, pour le problème considéré, *éliminer* les circuits. Éliminer arbitrairement les circuits ne garantit pas forcément les meilleurs chemins vers tous les sommets du graphe, mais assure l'existence d'une solution satisfaisante même en présence de circuits absorbants.

En raison des coûts négatifs, l'algorithme retenu sera l'algorithme de Bellman. Il existe plusieurs stratégies pour mettre en oeuvre cet algorithme, nous retiendrons celle qui exploite la construction par un parcours profondeur d'une pile de sommets à traiter. Les sommets sont empilés *en suffixe* lors du parcours profondeur, on profite également de ce parcours pour *numéroter* les sommets avec les ordres suffixes et préfixes de rencontre afin d'identifier les circuits.

L'algorithme de Bellman que nous utiliserons répond donc au principe suivant :

– À l'aide d'un parcours profondeur, on trie les sommets dans une pile en ordre suffixe de rencontre ;
– puis, pour chaque sommet de cette pile, on relâche les arcs sortants (selon la méthode habituelle) en ignorant les arcs en arrière (ceux à l'origine des circuits).

Dans la suite, les applications sur le graphe d'exemple se feront toujours ne considérant les sommets *triés* par ordre croissant (aussi bien dans la liste de sommets que dans les listes d'adjacences). On précise également que lorsqu'il fait mention de l'ordre préfixe ou suffixe il s'agit de l'ordre de rencontre dans le parcours profondeur établit avec un compteur *unique*.

1. Soit `op` et `os` les tableaux indiquant l'ordre préfixe et d'ordre suffixe de rencontre dans le parcours profondeur du graphe, rappeler la condition suffisante pour que l'arc $s \rightarrow sa$ soit un arc *en arrière* (indiquant un circuit).

2. Soit la procédure d'appel du parcours profondeur :

```
algorithme procedure pprof
  parametres locaux
    t_graphe_d        g
    entier            src
  parametres globaux
    t_vect_entiers    op, os
    t_pile            p
  variables
    t_listsom         ps
    entier            i
debut
  pour i ← 1 jusqu'a g.ordre faire
    op[i] ← 0
    os[i] ← 0
  fin pour
  p ← pile_vide()
  i ← 0
  pprof_rec(recherche(src, g), i, op, os, p)
fin algorithme procedure pprof
```

   Écrire la procédure récursive `pprof_rec(ps,cpt,op,os,p)` qui effectue le parcours profondeur depuis le sommet `ps` et numérote les sommets (en ordre préfixe de rencontre) dans `op` et (en ordre suffixe de rencontre) dans `os` à l'aide du compteur `cpt`. Enfin, la procédure empilera les sommets en ordre suffixe dans la pile de pointeur (de type `t_listsom`) `p`.

3. Écrire la procédure `bellman_prof(g, src, op, os, p, pere, dist)` qui calcule l'ensemble des *plus courts* chemins (en ignorant les arcs en arrière) depuis le sommet `src` dans le graphe `g` vers tous les autres sommets. La pile `p`, ainsi que les vecteurs `op` et `os` ont été obtenus par le parcours profondeur du graphe `g` à partir du même sommet source `src` (par la procédure `pprof`). L'algorithme remplit le vecteur de pères (`pere`) et le vecteur de distances (`dist`) avec les chemins calculés.

4. Appliquer le parcours profondeur depuis le sommet 1 sur le graphe de la figure 2 pour remplir les tableaux d'ordre préfixe (`op`) et suffixe (`os`). Vous remplirez également le tableau figurant la pile remplie par le parcours (le sommet de pile étant en première case à gauche).

5. À partir des tableaux remplis à la question précédente, quels sont les arcs qui seront ignorés par la procédure `bellman_prof` ?

6. Appliquer la procédure `bellman_prof` sur le graphe de la figure 2 depuis le sommet 1 en utilisant la pile et les vecteurs construits à la question 4 et remplir le vecteur de pères et le vecteur de distances.

7. Quel est le chemin obtenu pour aller du sommet 1 au sommet 2 (donner le chemin et le coût total) ?

8. Existe-t-il dans le graphe de la figure 2 un meilleur chemin (que celui trouvé) du sommet 1 au sommet 2 ? Si oui, lequel (donner le chemin et le coût total) ?

Exercice 4 (Graphe réduit – *3 points*)

Soit G un graphe orienté admettant p composantes fortement connexes : C_1, C_2, ..., C_p. On définit le *graphe réduit* de G (noté G_R) par $G_R = < S_R, A_R >$ avec :

- $S_R = \{C_1, C_2, \cdots, C_p\}$
- $C_i \rightarrow C_j \in A_R \Leftrightarrow$ Il existe au moins un arc dans G ayant son extrémité initiale dans la composante fortement connexe C_i et son extrémité terminale dans la composante fortement connexe C_j.

Écrire un algorithme qui construit le graphe réduit G_R d'un graphe G (Les deux graphes sont en représentation statique). L'algorithme prendra en entrées le vecteur des composantes fortement connexes cfc, qui contient pour chaque sommet le numéro de la composante fortement connexe à laquelle il appartient, ainsi que le nombre de composantes fortement connexes du graphe G.

Annexes

Piles de sommets

Types
`t_pile`

Operations
`pile_vide() : t_pile` /* *Renvoie une nouvelle pile vide* */
`est_vide(t_pile p) : booleen` /* *Renvoie vrai si la pile p est vide* */
`empiler(t_listsom ps, t_pile p) : pile` /* *Empile ps sur la pile p et renvoie la nouvelle pile* */
`depiler(t_pile p) : t_listsom` /* *Depile un element et le renvoie, la pile est modifiee* */

Les graphes : Représentations

Représentation statique

```
constantes
   Max_sommets = 100
types
   t_mat_adj  = Max_sommets × Max_sommets  entier
   t_graphe_s = enregistrement
       booleen     orient
       entier      ordre
       t_mat_adj   adj
   fin enregistrement t_graphe_s
```

Représentation dynamique

```
types
    t_listsom = ↑ s_som       /* liste des sommets */

    t_listadj = ↑ s_ladj      /* liste d'adjacence */

    s_som      = enregistrement       /* un sommet */
       entier      som
       t_listadj   succ
       t_listadj   pred
       t_listsom   suiv
    fin enregistrement s_som

    s_ladj      = enregistrement       /* un successeur (ou prédécesseur) */
       t_listsom   vsom
       entier      nbliens
       reel        cout
       t_listadj   suiv
    fin enregistrement s_ladj

    t_graphe_d  = enregistrement       /* le graphe */
       entier     ordre
       booleen    orient
       t_listsom  lsom
    fin enregistrement t_graphe_d
```