Option Informatique
Arbres couvrants minimaux

Sujet

16 décembre 2005

1 Graphes

Informellement, un graphe est un ensemble de points (des sommets) dont certains sont reliés par un trait (une arête). Pour les distinguer, on numérote les sommets (voir figure 1).

Formellement, un graphe non-orienté est un couple (V, E) ou V est l'ensemble des sommets (vertices) et $E \subseteq \mathcal{P}_2(V)$ est l'ensemble des arêtes (edges). E est un ensemble de paires de sommets.

Deux sommets sont dits voisins s'ils sont reliés par une arête. Une arête est dite adjacente à un sommet e si ce sommet est l'une de ses deux extrémités. On peut pour certains problèmes associer un poids (ou un coût) à chaque arête.

La représentation habituelle des graphes se fait par listes d'adjacence : un graphe est représenté par un tableau de longueur $|V|$ dont la case i contient la liste des voisins du sommet i. Une autre représentation classique mais plus coûteuse en mémoire est la matrice d'adjacence : un graphe est alors représenté par une matrice $M \in \{0,1\}^{|E|\times|E|}$ telle que $m_{i,j} = 1$ si et seulement s'il y a une arête entre i et j. Nous utiliserons ici la représentation *forward star* qui est proche de la représentation par listes d'adjacence. La seule différence est que toutes les listes d'adjacence sont concaténées dans une liste. Pour chaque sommet on stocke la position dans ce tableau du début de sa liste d'adjacence. On a donc :

```
  (* Une arête est un triplet contenant les deux extrémités et le poids. *)
type arete = Edge of int*int*int ; ;
type graphe =  Sommets : int list ; Aretes : arete list ; ;
```

2 Problème de l'arbre couvrant

On considère ici des graphes non orientés valués : $G = (E, A, v)$ où E est un ensemble fini (ensemble des sommets), $A \subset \mathcal{P}_2(E)$ est l'ensemble des arêtes et $v : A \to \mathbb{R}$ est la fonction de poids. Le poids de G est $\Sigma_{\alpha\in A} v(\alpha)$. Un sous-graphe de G est un graphe de la forme $G' = (E', A', v_{|A'})$ où $E' \subset E, A' \subset \mathcal{P}_2(E') \cap A$. Si $E = E'$, G' est couvrant.

Un chemin de a à b, $a, b \in E$ est une suite finie de sommets $(x_1, \ldots, x_n)$ avec $a = x_1$, $b = x_n$. On dit qu'un chemin est simple si la suite est injective. On dit que G est connexe s'il existe un chemin entre deux sommets quelconques (il y a alors un chemin simple). Si, de plus, il y a unicité du chemin simple, alors on dit que G est un arbre (ou graphe simplement connexe, graphe connexe sans cycle). Remarquons qu'un arbre possède $\sharp E - 1$ arêtes.

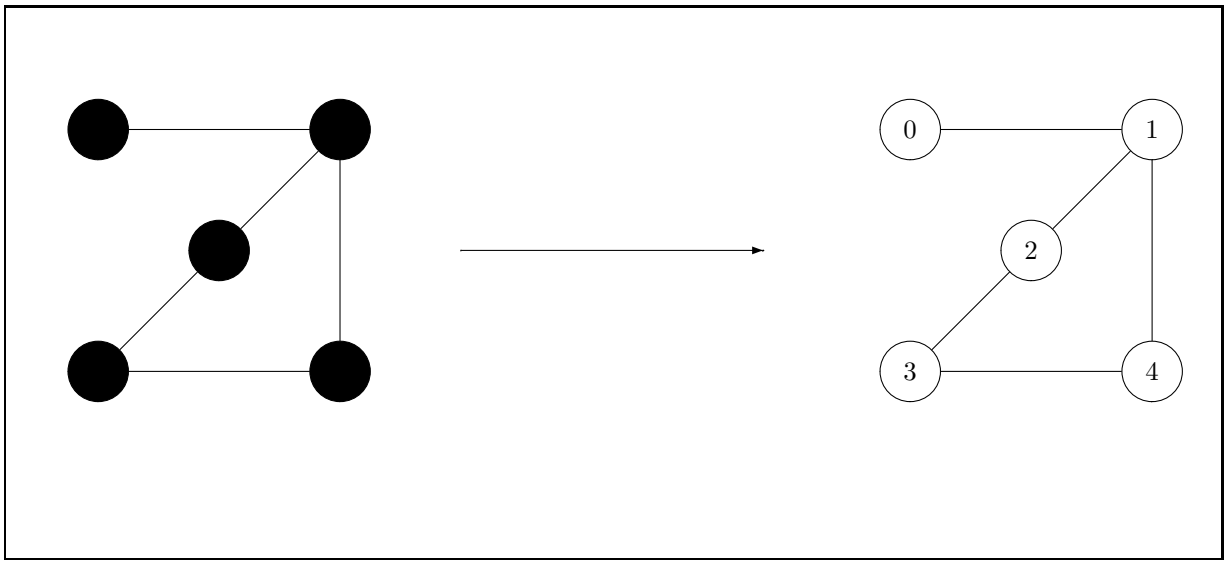

FIG. 1 – Numérotation des sommets

On se pose le problème suivant : supposant G connexe, trouver un sous-graphe couvrant G' qui soit un arbre, et de poids minimal. Evidemment, cela revient à chercher un sous-ensemble de A. On identifie $F \subset A$ au sous-graphe (E', F) contenant les sommets extremités d'arêtes de F.

Questions

3 Algorithme de Prim

On construit ici un sous-graphe connexe que l'on fait croître (voir figure 2).

Algorithme 1 (Prim)

1. $F \leftarrow \emptyset$ *(ensemble d'arêtes, qui définit un sous-arbre)*
2. *Tant que F n'est pas couvrant, lui rajouter une des arêtes de poids minimal qui le relie à un sommet extérieur.*

Questions

1. Écrire une fonction `tri_aretes` de type `arete list` → `arete list` effectuant le tri d'une liste d'arêtes par ordre de poids croissant.
2. Écrire une fonction `poids` de type `graphe` → `int` calculant le poids d'un graphe.
3. Écrire une fonction `rnd_elt` tirant au hasard un élément d'une liste. Quel est son type ?[1]
4. Écrire une fonction `cherche_areteP` de type `int list` → `arete list` → `arete` qui renvoie l'arête de poids minimum reliant un ensemble de sommets à un autre sommet du graphe.
5. En déduire une implémentation de l'algorithme de Prim
   (fonction `spanningtree_Prim`, type `graphe` → `graphe`).
6. Démontrer que l'algorithme de Prim renvoie bien un arbre couvrant minimal.

[1] La documentation de Caml indique : Pour les essais, il est souvent pratique de disposer d'un générateur de nombres aléatoires. La bibliothèque random propose la fonction `random__int` qui renvoie un nombre aléatoire compris entre 0 inclus et son argument exclus. `random__float` est analogue mais renvoie un nombre flottant. La fonction `random__init` permet d'initialiser le générateur avec un nombre entier. Par exemple :`random__init(int_of_float (sys__time())) ; ;`

4 Algorithme de Kruskal

On réunit au fur et à mesure des composantes connexes (voir figure 3).

Algorithme 2 (Kruskal)

1. *Trier les arêtes par poids croissant* $v(\alpha_1) \leq v(\alpha_2) \leq \ldots$
2. $F \leftarrow \emptyset$ *(ensemble d'arêtes)*
3. *Parcourir les arêtes et pour chacune, la rajouter à* F *si elle relie deux composantes connexes distinctes du graphe défini par* F

Questions

1. Écrire une fonction `fusion` réalisant la fusion par l'arête a de deux graphes g_1 et g_2 d'une forêt (liste de graphes), de type `graphe list` → `graphe` → `graphe` → `arete` → `graphe list`.
2. Écrire l'explosion d'un graphe en un ensemble de ses sous-graphes élémentaires (`explose : graphe` → `graphe list`).
3. Écrire `cherche_arbre`, de type `int` → `graphe list` → `graphe`, qui à tout élément `x` et à toute forêt `f` associe le graphe de `f` auquel appartient `x`.
4. Écrire la recherche de l'arête de poids minimal reliant deux arbres d'une forêt (`cherche_areteK : graphe list` → `arete list` → `graphe * graphe * arete`).
5. En déduire une implémentation de la recherche d'arbre couvrant minimal par l'algorithme de Kruskal (`spanningtree_Kruskal : graphe` → `graphe`).
6. Prouver le lemme suivant :

   Lemme 1 *Si* F_1 *et* F_2 *sont deux sous-arbres couvrants et si* $\alpha \in F_1, \alpha \notin F_2$*, alors il existe une arête* $\beta \in F_2$ *telle que* $F_1 \backslash \{\alpha\} \cup \{\beta\}$ *soit un sous-arbre couvrant.*
7. En déduire une preuve de la validité de l'algorithme de Kruskal.

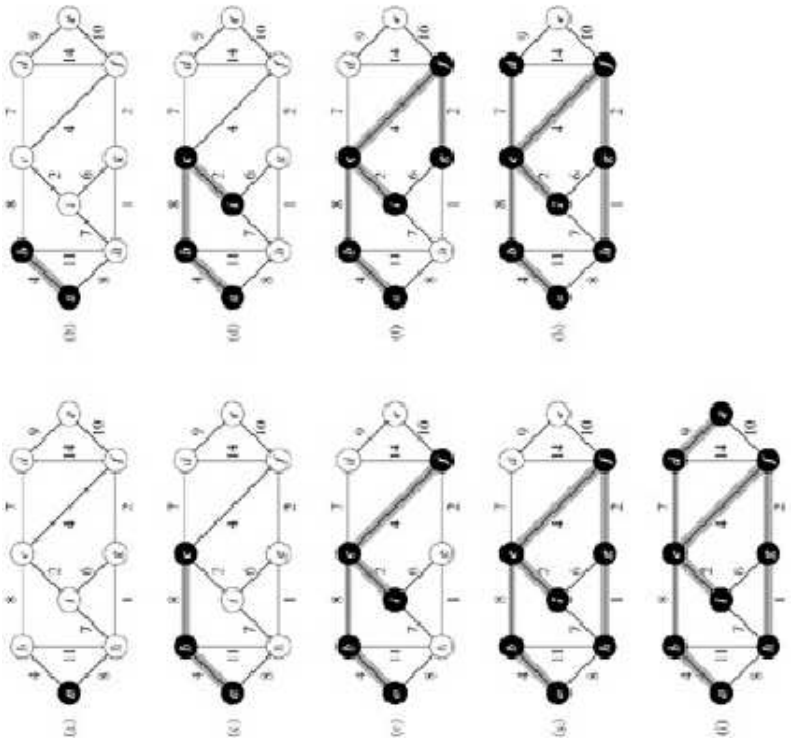

FIG. 2 – Algorithme de Prim

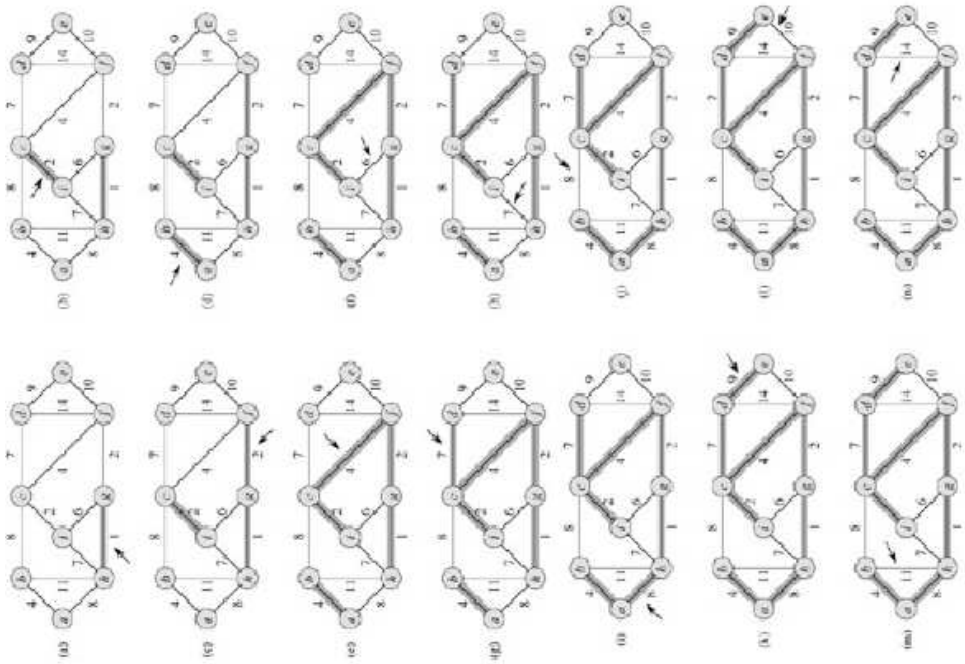

FIG. 3 – Algorithme de Kruskal