cours d'algorithmique du 07/03/07

Guillaume Munch, Mathilde Noual

1 Arbres couvrants de poids minimal

1.1 Problème

Etant donné $\mathcal{G} = (V, E)$, un graphe non orienté connexe, et $\omega : E \to \mathbb{R}^+$, on veut trouver un arbre couvrant, $\mathcal{T}$, de $\mathcal{G}$ qui minimise $\omega(\mathcal{T}) = \sum_{x,y \in E} \omega(xy)$.

Rappel : un graphe couvrant de $\mathcal{G} = (V, E)$ est un graphe partiel de $\mathcal{G}$ qui est un arbre et qui contient tous les sommets de $\mathcal{V}$.

1.2 Algorithme de Prim

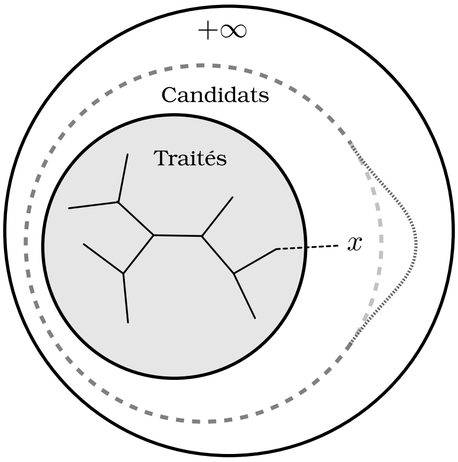
On a :

- *en entrée* : un graphe $\mathcal{G} = (V, E)$ donné par ses listes d'adjacence et une fonction de poids $\omega : E \to \mathbb{R}^+$
- *en sortie* : un arbre couvrant $\mathcal{T}$ de $\mathcal{G}$ de poids minimal

Le *principe* est de faire croître un arbre de poids minimal depuis une racine choisie arbitrairement parmi les sommets de $\mathcal{G}$.

Algorithme 1 : PRIM$(\mathcal{G}, \omega)$

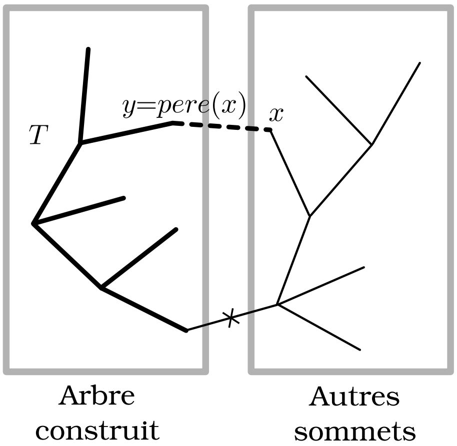
```
début
    pour chaque x ∈ V faire
        poids(x) ← +∞;
        pere(x) ← NULL;
        traite(x) ← NON
    fin
    Candidats ← {r}                  (r : sommet quelconque pris comme racine du calcul)
    poids(r) ← 0;
    SommetsCouverts ← 1;
    tant que SommetsCouverts < n faire
        x ← sommet de poids minimum dans Candidats;
        Candidats ← Candidats \ {x};
        traite(x) ← OUI;
        pour chaque y ∈ N(x) tel que traite(y) = NON faire
            si poids(y) = +∞ alors
                Candidats ← Candidats ∪ {x}
            fin
            si ω(xy) < poids(y) alors
                poids(y) ← ω(xy);
                pere(y) ← x
            fin
        fin
        SommetsCouverts ← SommetsCouverts + 1
    fin
fin
```



Correction
Les invariants de l'algorithme sont :
- I_1 : si $x \in Candidats, poids(x)$ stocke le poids de la plus petite arête qui relie x à un sommet déjà traité.
- I_2 : à tout instant, l'arbre construit peut être prolongé en un arbre couvrant de $\mathcal{G}$ de poids minimum.

Preuve
I_1 est clair d'après les mises à jour de *poids*(.) dans l'algorithme.

Preuve de I_2 :
- L'invariant est vrai à l'étape d'initialisation (puisqu'il n'y a qu'un seul sommet à ce moment)
- On suppose I_2 vrai pour $\mathcal{T}$ jusqu'à l'étape précedant l'ajôut du sommet x. On considère l'arbre de poids minimum, $\mathcal{T}'$, prolongement de $\mathcal{T}$ à tout le graphe. Soit y le père de x dans $\mathcal{T}$.
  Si l'arête $(yx) \notin \mathcal{T}'$, considérons dans $\mathcal{T}'$, une chaîne de x à y et l'arête (zt) de cette chaîne entre $\mathcal{T}' \setminus \mathcal{T}$ et $\mathcal{T}$. D'après le choix de x, on sait que $\omega(zt) \geq \omega(zt)$ (comme t et $y \in \mathcal{T}$, z et $x \in Candidats$. x ayant été choisi, *poids*(x) est minimum). Donc $\mathcal{T}'' = \mathcal{T}' \setminus \{z\} \cup \{(xy)\}$ est un arbre couvrant de poids $\omega(\mathcal{T}'') \leq \omega(\mathcal{T}')$ qui prolonge $\mathcal{T}'$: contradiction, donc $(yx) \in \mathcal{T}'$.



Remarque : on aurait pu utiliser l'invariant I_2' : l'arbre construit est un arbre couvrant de poids minimal des sommets qu'il couvre.

Complexité
On fait :
- $\mathcal{O}(n)$ fois des extractions de minimum,

– $\mathcal{O}(\sum_{x \in V} deg(x)) = \mathcal{O}(m)$ fois des mises à jour de poids,
– $\mathcal{O}(n+m)$ fois des parcours d'arête.

Structures de données

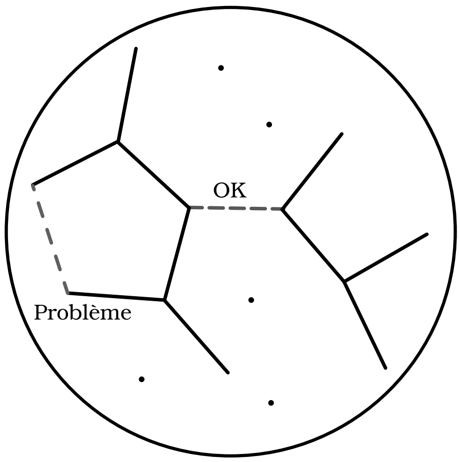
gestion de l'ensemble *Candidats*	liste doublement chaînée	tableau	tas binaire	tas de Fibonacci
complexité totale	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(mlog(n))$	$\mathcal{O}(m+nlog(n))$

Remarque : si la liste est triée en fonction du poids, en terme de compléxité, c'est pire (puisqu'il faut la maintenir triée malgré les modifications de poids).

Remarque : pour $poids(.)$ et $pere(.)$ on peut utiliser des tableaux.

1.3 Algorithme de Kruskal

Principe :
– Trier les arêtes par poids croissant
– ajouter une à une les aretes dans cet ordre sans créer de cycle
→ On obtient une forêt qui croît jusqu'à devenir un arbre couvrant de poids minimal.



Correction
Invariant de l'algorithme : à tout instant, la forêt construite peut se prolonger en un arbre couvrant de poids minimal.

Preuve
– L'invariant est vrai à l'étape d'initialisation (une seule arête, celle de poids minimum)
– On suppose l'invariant vrai pour $\mathcal{F}$ (forêt de parcours) jusqu'à l'étape précedant l'ajoût de l'arête (xy) à $\mathcal{F}$. $\mathcal{F}$ peut se prolonger en $\mathcal{T}$, arbre couvrant de poids minimal.
Si $(xy) \notin \mathcal{T}$, alors $\mathcal{T} \cup \{(xy)\}$ a un cycle qui contient une arête e de $\mathcal{T} \setminus \mathcal{F}$ (x et y ne sont pas déjà connectés dans $\mathcal{F}$ sinon on ne pourrait pas ajoûter (xy), et, $\mathcal{T}$ étant connexe, il contient une chaîne de x à y). Etant donné l'ordre d'ajoût des arêtes, on a :
$$\omega(e) \geq \omega(xy)$$
Mais dans ce cas, $\mathcal{T} \setminus \{(zt)\} \cup \{(xy)\}$ est de poids $\leq \omega(\mathcal{T})$ et contient l'arête (xy) : contradiction, donc $(yx) \in \mathcal{T}$.

Compléxité

Gérer la forêt est équivalent à la gestion de partition pour les opérations *Union*/*Find* :
La forêt correspond à une partition des sommets en ensembles disjoints (connexes et sans cycle).
Le test de création de cycle revient à vérifier si les extrémités de l'arête ajoûtée appartiennent à la même partie (*Find*).
L'ajout d'une arête revient à fusioner les deux parties correspondant à chacune des extrémités de l'arête(*Union*).

Au total : $\mathcal{O}(m.\alpha(n))$ (cf cours de compléxité amortie) et ce, au tri initial des arêtes près (en $\mathcal{O}(m.log(n))$).

2 Exemples d'applications de parcours en profondeur (D.F.S.)

Version Récursive du D.F.S.
(elle facilite l'introduction des "dateurs" par rapport à la version boucle présentée avant)

Algorithme 2 : DFS($\mathcal{G}$)

```
début
    pour chaque u ∈ V faire
        etat(u) ← NONATTEINT;
        pere(u) ← NULL;
    fin
    temps ← 0;
    pour chaque u ∈ V faire
        si etat(u) = NONATTEINT alors
            DFS(G, u)
        fin
    fin
fin
```

Algorithme 3 : DFS($\mathcal{G}, u$)

```
début
    etat(u) ← DECOUVERT;
    temps ← temps + 1;
    d(u) ← temps;
    pour chaque v ∈ N(u) faire
        si etat(u) = NONATTEINT alors
            pere(v) ← u;
            DFS(G, v)
        fin
    fin
    etat(u) ← TRAITE;
    temps ← temps + 1;
    f(u) ← temps
fin
```

Remarque :
$d(u)$= date de découverte de u

$f(u)$= date de fin de traitement de u

Propriétés - Agencement des intervalles $[d(u); f(u)]$
Etant donné $\mathcal{G} = (V, E)$ et les numérotations $d(.)$ et $f(.)$ produites par un D.F.S., soient u et $v \in V$

- – Soit $[d(u); f(u)] \subseteq [d(v); f(v)]$
  - Soit $[d(v); f(v)] \subseteq [d(u); f(u)]$
  - Soit $[d(v); f(v)] \cap [d(u); f(u)] = \emptyset$
- u est un descendant de v dans la forêt de parcours *ssi*
$$[d(u); f(u)] \subseteq [d(v); f(v)]$$
- si (uv) est un arc, alors on ne peux pas avoir
$$d(u) < f(u) < d(v) < f(v)$$

Correction
Invariant de l'algorithme : à tout instant,les sommets dans la pile d'appels à D.F.S. $(\mathcal{G},.)$ (qui sont les sommets découverts mais pas encore traités) forment un chemin de la forêt de parcours depuis une des racines.

Preuve
Supposons $d(u) < d(v)$

- $d(v) \leq f(u)$, alors v est découvert avant que soit traité, donc, à sa découverte, v est au-dessus de u dans la pile d'appels.
  $\rightarrow v$ est un descendant de u (d'après l'invariant dans la forêt de parcours), et
  $\rightarrow v$ est traité avant u (étant donné le comportement de la pile d'appels)
  donc $f(v) < f(u)$ i.e. $[d(v); f(v)] \subseteq [d(u); f(u)]$
- sinon si $d(v) < d(u)$, ...alors $[d(v); f(v)] \cap [d(u); f(u)] = \emptyset$

Principe de l'algorithme de recherche des composantes fortement connexes pour un graphe $\mathcal{G} = (V, E)$ *orienté*

1. faire un D.F.S. sur $\mathcal{G}$ en calculant les numérotations $d(.)$ et $f(.)$
2. calculer $\mathcal{G}^t = (V, E^t)$ (transposé de $\mathcal{G}$ dont les arcs sont inversés par rapport à ceux de $\mathcal{G}$)
3. faire un D.F.S. sur $\mathcal{G}^t$ en prenant pour racine à chaque fois le sommet restant de $f(.)$ maximum.

$\rightarrow$ Les arbres de la nouvelle forêt sont les composantes fortement connexes.