

Algorithmique Avancée - TD 4 : corrigé - 8 novembre 2011

Université Paris 7 — BioInformatique — M1

Arbres couvrants de poids minimum

Exercice 1 Il faut mettre toutes les arêtes de poids 1. Pour celles de poids 2 il faut les prendre toutes, sauf l'une des 3 arêtes du cycle 2,4,5. Trois possibilités donc 3 arbres.

Exercice 2 Un arbre couvrant de $G = (V, E)$ est un sous-ensemble d'arêtes $A \subset E$. Comme E est fini, il y a un nombre fini 2^m de sous-ensembles, donc d'arbres couvrants. Tout ensemble fini admet un minorant, CQFD.

Exercice 3 Si on ne s'est pas trompé le poids total devrait être 25

Exercice 4 KRUSKAL(G)

```
A : ensemble vide.
Trier les arêtes de E par ordre croissant de poids.
Initialiser la structure de données
Pour toute arête (u,v) de E prise dans l'ordre croissant faire
  Si Find(u,v)
    ajouter (u,v) à A
    Union(u,v)
  Fin du si
Fin du pour
Retourner A
```

Exercice 5 m Il y a exactement m appels à Find et $n - 1$ appels à Union. On peut donc déjà dire que la complexité est $O(mf + nu)$ où f et u sont les temps d'exécution de Find et Union.

Exercice 6 Une implémentation "naïve" est de donner un numéro de composante à chaque sommet. Find se fait alors en temps constant mais Union demande de renuméroter tous les sommets d'une composante, en temps $O(n)$.

Une autre implémentation "naïve" est de faire des listes chaînées de sommets de chaque composante. Là c'est Union qui se fait alors en temps constant mais Find en temps $O(n)$.

La bonne implémentation est d'utiliser des arbres. Chaque sommet d'une composante pointe vers un père (qui peut être lui-même) et deux sommets sont dans la même composante si et seulement si ils sont dans des arbres de même racine. Find se fait en $O(h)$: il suffit de remonter à la racine pour les deux. Union se fait aussi en $O(h)$: on remonte aux deux racines, puis le père de la racine de v devient la racine de u . Noter que cela augmente de 1 la hauteur de v mais pas celle de u . Si l'on choisit l'arbre de plus petite hauteur pour devenir fils de l'arbre de plus grande hauteur, alors on peut montrer que $h = O(\log n)$. Donc Union et Find tournent en temps $O(\log n)$.

Exercice 7 Avec la première solution naïve : $O(m+n^2) = O(n^2)$. Avec la deuxième : $O(mn+n) = O(mn)$ ce qui est pire. Avec la troisième solution : $O((n+m)\log n)$. Comme on suppose le graphe connexe, cela fait $O(m\log n)$ ce qui est meilleur (c'est la même complexité que Prim).

Il existe une implémentation de l'Union-Find qui tourne avec une complexité de $O(m \log^* n)$ en tout, où $\log^*(n)$ est le nombre de fois où il faut itérer le log pour tomber sur une valeur négative (formellement, $\log^*(n) = \max\{k | \log^{(k)}(n) \text{ est défini}\}$). Or si $n < 2^{65536}$, $\log^*(n) < 6$. Et 2^{65536} est nettement plus grand que le nombre d'atomes de l'univers observable et est donc une bonne limite pour la taille d'une mémoire d'ordinateur... Donc on peut considérer que, en pratique, l'algorithme est linéaire, même si asymptotiquement ce n'est pas vrai. **Voir cours!**

Exercice 8 1. Y'a qu'à...

2. Là encore on utilise un tas (une file de priorité). Trouver la clé minimale se fait en temps constant ; modifier les clés (y compris extractions) se fait en $O(\log n)$.
3. On remarque que les marques ne sont pas nécessaires : la présence dans la file suffit. L'algo modifié devient :

PRIM(G, s)

Clé : tableau d'entiers indexé par V initialisé à $[+\infty, +\infty, \dots, +\infty]$

Pere : tableau de sommets indexé par V initialisé à $[\text{null}, \text{null}, \dots, \text{null}]$

Clé[s] = 0

F : file de priorité initialisée avec les valeurs de Clé

Tant que la file F est non-vidée faire

 u = extraire-min(F)

 Pour tout sommet v voisin de u faire

 Si v est dans la file et $p(u, v) < \text{Clé}[v]$

 Père[v] = u

 Clé[v] = $p(u, v)$

 modifier-clé(F, v, $p(u, v)$)

 Fin du si

 Fin du Pour

Fin du tant que

Retourner Pere

4. On note n le nombre de sommets et m le nombre d'arêtes. Créer la file prend un temps $O(n)$ (la longueur de la file). A chaque tour dans la boucle **Tant que** on extrait un sommet de la file : il y a exactement n tours. Chaque tour dans cette boucle examine l'adjacence de v . Donc chaque arête (u, v) est vue exactement deux fois : une fois lors du tour de u et une fois lors du tour de v . Le **Si** dans la boucle **Pour tout** sont donc faite exactement $2m$ fois au cours de l'algorithme. Or ce **Si** contient un test et deux affectations en temps constant, plus un appel à **modifier-clé** en temps $O(\log n)$. On a donc en tout une complexité $O(n + m \log n)$. Comme on suppose le graphe connexe, cela fait $O(m \log n)$
5. C'est mieux de Kruskal avec l'implémentation "simple" de l'Union-Find (listes des sommets de composantes) mais moins bien qu'avec l'implémentation subtile

Exercice 9 [Arbre couvrant de poids max.] Il existe au moins deux solutions. En réalité le problème de l'arbre couvrant de poids maximum et l'ACPM sont "identiques". Nous pouvons transformer le problème maximum en problème d'ACPM. Il suffit pour cela de changer toutes les poids des arêtes par leur inverse (i.e. $p(e') = \frac{1}{p(e)}$). Nous pouvons montrer que T est un arbre couvrant de poids maximum ssi T' obtenu en changeant les valuations est un arbre de poids minimum.

L'algorithme est identique à celui de Prim, il faut simplement changer le choix min par max.

Exercice 10 [Arbre couvrant de poids min, \times .] Il suffit de voir que

$$a \times b \times c = e^{\ln(a \times b \times c)} = e^{\ln(a) + \ln(b) + \ln(c)}.$$

Par conséquent l'algorithme est le même, on peut utiliser soit Kruskal soit Prim.

Algorithme et complexité similaire à celle de l'exercice précédent.