TD d'algorithmique numéro 9
Natacha Portier et Daniel Hirschkoff
semaine du 29 novembre au 3 décembre 1999

Question 1 ⋆ composantes fortement connexes d'un graphe ⋆

On travaille sur des graphes *orientés*, que l'on suppose représentés par des *listes d'adjacence* : à deux ensembles S et A de sommets et arcs respectivement, on associe autant de listes $Adj[u]$ qu'il y a de sommets u dans S, chaque élément v de $Adj[u]$ dénotant le fait qu'il existe un arc de u à v.

Question 1.1 ⋆ parcours en profondeur ⋆

Le parcours en profondeur d'un graphe G fait usage des constructions supplémentaires suivantes :

- À chaque sommet du graphe est associée une "couleur" : au début de l'exécution de la procédure, tous les sommets sont blancs. Lorsqu'un sommet est rencontré pour la première fois, il devient gris. On noircit enfin un sommet lorsque l'on est en fin de traitement, et que sa liste d'adjacence a été complètement examinée.
- Chaque sommet est également *daté* par l'algorithme, et ceci deux fois : pour un sommet u, $d[u]$ représente le moment où le sommet a été rencontré pour la première fois, et $f[u]$ indique le moment où l'on a fini d'explorer la liste d'adjacence de u. On se servira par conséquent d'une variable entière "temps" pour la datation.
- La manière dont le graphe est parcouru déterminera aussi une relation de paternité entre les sommets, et l'on notera $\pi[u] = v$ pour dire que v est le père de u (selon l'exploration qui est faite).

On définit alors le parcours en profondeur (**PP**) de la manière suivante :

PP(G) :=
  `pour chaque` sommet u de S *Initialisation*
    `faire` couleur$[u] \leftarrow$ BLANC
      $\pi[u] \leftarrow$ NIL
  temps$\leftarrow 0$
  `pour chaque` sommet u de S *Exploration*
    `faire si` couleur$[u]$=BLANC
      `alors` **Visiter_PP**(u)

Visiter_PP(u) :=
  couleur$[u] \leftarrow$ GRIS
  $d[u] \leftarrow$temps $\leftarrow$temps+1

```
  pour chaque v ∈ Adj[u]
    faire si couleur[v]=BLANC
            alors π[v] ← u
                  Visiter_PP(v)
  couleur[u] ← NOIR
f[u] ← temps← temps+1
```

Quel est le temps d'exécution de PP?

Que peut-on dire de $d[u]$, $f[u]$, $d[v]$ et $f[v]$, pour deux sommets u et v du graphe?

Montrer que v est un descendant de u si et seulement si à $d[u]$, il existe un *chemin blanc* de u à v.

Question 1.2 — ⋆ composantes fortement connexes ⋆

On définit la relation binaire R des sommets d'un graphe orienté par : xRy si et seulement s'il existe un chemin orienté de x à y et un chemin orienté de y à x. Montrer que c'est une relation d'équivalence.

Les classes d'équivalence sont appelées *composante fortement connexe* du graphe orienté. Ce sont les sous-ensembles maximaux du graphe tels qu'il existe un chemin allant d'un sommet u à un sommet v pour tous sommets u et v de l'ensemble.

On appelle *sous-graphe de liaison* d'un parcours en profondeur le graphe $G_\pi = (S, A_\pi)$, où

$$A_\pi \;=\; \{(\pi[v], v),\, \pi[v] \neq \text{NIL et } v \in S\}\,.$$

on remarque que G_π forme une forêt en profondeur, composée d'*arborescences en profondeur.*

Montrer que lors d'un parcours en profondeur, tous les sommets appartenant à la même composante fortement connexe sont dans la même arborescence en profondeur.

Pour tout sommet u, et pour une exécution d'un parcours en profondeur, on appelle $\phi(u)$ le sommet w tel que $u \rightsquigarrow w$ (i.e. il existe un chemin de u à w dans le graphe) et $f[w]$ est maximal. On appelle $\phi(u)$ l'*aïeul de u.*

Montrer que $\phi(u)$ est un ancêtre de u.

Remarque : par conséquent, u et $\phi(u)$ sont dans la même composante fortement connexe, et plus globalement deux sommets sont dans la même composante fortement connexe, si et seulement si ils ont le même aïeul.

Question 1.3

On considère l'algorithme suivant :

- Exécuter le parcours en profondeur de G pour calculer f sur l'ensemble des sommets.

- Calculer le graphe ${}^tG$ qui a les mêmes sommets que G mais des arcs en sens inverse.

– Exécuter le parcours en profondeur de ${}^tG$ en appelant dans la boucle principale les sommets par f décroissant. Les composantes fortement connexe de G (et aussi de ${}^tG$) sont les arborescences ainsi obtenu.

Monotrer la correction de l'algorithme précédent en montrant par récurrence sur le nombre d'arborescences déjà calculées qu'une composante fortement connexe, et donc l'ensemble des sommets de même aïeul selon G est une arborescence.

Donner la complexité de l'algorithme.

Question 2 ⋆ 2-SAT ⋆

Le problème 2-SAT est défini de la manière suivante : on se pose la question de la satisfiabilité d'une formule booléenne en *Forme Normale Conjonctive* d'ordre 2 (2-FNC). Une telle formule est une *conjonction* de *clauses*, chaque clause étant elle-même constituée d'une *disjonction* de deux *littéraux*. Un littéral dans une formule booléenne est une variable ou sa négation.

Exemple :

$$(x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

est une 2-FNC, satisfiable en mettant par exemple x_1 à vrai et x_2 à faux.

Question 2.1 ⋆ condition nécessaire ⋆

Représenter la formule comme un graphe orienté. Donner une condition nécessaire sur le graphe pour que la formule soit satisfiable.

Nous allons montrer que cette condition est suffisante.

Question 2.2

Montrer qu'une valuation v qui satisfait la formule F est constante sur une composante fortement connexe du graphe associé G.

Question 2.3 ⋆ graphe des composantes ⋆

On construit un graphe orienté H associé au graphe G de la manière suivante :

– Les sommets de H sont les composantes fortement connexes de G.

– Il y a un arc de X à Y dans H s'il existe un arc d'un sommet de x vers un sommet de y dans G.

Montrer que H est sans cycle orienté.

Question 2.4 ⋆ tri topologique ⋆

Les arcs induisent un ordre partiel sur les sommets du graphe sans cycle orienté H et cet ordre peut-être complété en un ordre total. L'algorithme du tri topologique est défini de la manière suivante : le graphe est parcouru en profondeur et les sommets classés par $f[u]$ décroissant en les insérant au fur et à mesure au début d'une liste chaînée.

Montrer que l'ordre total ainsi obtenu étend bien l'ordre partiel.

Question 2.5

On définit la valuation v de la manière suivante : si la variable x est dans la composante X et $\neg x$ dans la composante Y alors on pose $v(x) = faux$ si $X < Y$ et $v(x) = vrai$ sinon.

Montrer que v satisfait F et conclure.

Question 3 ⋆ SAT-n ⋆

On appelle SAT-n le problème SAT restreint aux formules qui n'ont pas plus de n occurences de la même variable.

Question 3.1 ⋆ SAT-3 ⋆

Montrer que SAT-3 n'est pas plus facile que SAT, et donc pas plus facile que SAT-n.

Question 3.2 ⋆ SAT-2 ⋆

Soit x une variable apparaissant dans la formule F de SAT-2. Trouver une formule équivalente à F et dans laquelle la variable x n'apparaisse plus.

En déduire un algorithme polynomial pour SAT-2.

Question 4 ⋆ Hachage ⋆

La technique du hachage (en anglais *hashing*) est utilisée pour implémenter un dictionnaire lorsque l'espace des clés est considérablement plus petit que le nombre de clés effectivement manipulées (et il serait alors maladroit de gérer des tableaux comportant une case par entrée possible).

L'idée est de travailler avec une table de taille m, inférieure au nombre n de clés possibles. Une *fonction de hachage* permet alors de calculer la case où un élément doit être inséré. Bien entendu, on peut alors observer des *collisions*, plusieurs éléments de clés différentes étant hachés dans la même case. Le *facteur de remplissage* α d'une table est défini par n/m, i.e. le nombre moyen d'éléments ayant été stockés dans une case.

Afin de limiter le nombre de collisions, on s'attache à ce qu'une fonction de hachage répartisse aussi uniformément que possible les éléments. On dit qu'une fonction de hachage respecte l'hypothèse de *hachage uniforme simple* si chaque élément a la même chance d'être haché dans l'une quelconque des cases de la table, indépendamment de l'endroit où les autres éléments sont tombés.

Question 4.1 ⋆ Hachage avec chaînage ⋆

On fait ici l'hypothèse d'un hachage uniforme simple. Une manière simple de gérer les collisions engendrées par la fonction de hachage est d'associer à chaque case de la table une liste chaînée d'éléments dont les clés partagent la même valeur de hachage. On insère alors un élément à la fin de la liste d'éléments déjà insérés dans la case correspondante.

Montrer qu'une recherche qui échoue prend en moyenne un temps en $O(1+\alpha)$. Montrer qu'il en est de même pour une recherche réussie.

Ces propriétés indiquent que si le nombre de cases de la table de hachage est au moins proportionnel au nombre d'éléments de la table, on a $n = O(m)$, et α est en $O(1)$, ce qui montre que la recherche se fait en temps constant en moyenne.

Question 4.2 ⋆ Adressage ouvert ⋆

En adressage ouvert, on ne manipule plus de pointeurs (ce qui a pour avantage en principe de libérer de la place pour des cases mémoire supplémentaires) : tous les éléments sont conservés *dans la table*, dont les cases soit sont vides soit contiennent un seul élément, et on *calcule* une suite de cases à examiner.

Pour insérer un élément dans une table en adressage ouvert, on calcule une case où insérer la clé ; si la case est prise, on passe à la case suivante selon le procédé de hachage employé, et ainsi de suite jusqu'à trouver une case vide où l'insertion peut être effectuée. Symétriquement, la recherche consiste en général en un parcours d'une séquence de cases, jusqu'à trouver l'élément ayant la bonne clé ou une case vide (cas d'échec de la recherche).

Remarque : en adressage ouvert, α ne doit jamais être supérieur à 1.

On s'intéresse à trois techniques permettant de mettre en œuvre le hachage ouvert (ci-dessous, h', h_1 et h_2 sont des fonctions de hachage ordinaire) :

- *Sondage linéaire* $h(k, i) = (h'(k) + i) \bmod m$
- *Sondage quadratique* $h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$

  On doit fixer c_1 et c_2 ...
- *Double hachage* $h(k, i) = (h_1(k) + i h_2(k)) \bmod m$

  Remarque : pour tout k, le nombre $h_2(k)$ doit être premier par rapport à la taille m de la table de hachage.

Quelle est la meilleure méthode ? La pire ? Lesquelles de ces méthodes vérifient l'hypothèse de hachage uniforme ?