Algorithmique

Examen du 12 novembre 2009

durée 3 heures

Le polycopié du cours est le seul document autorisé.

Les exercices sont indépendants.

Toutes les réponses devront être correctement justifiées. La rigueur des raisonnements, la clarté des explications, et la qualité de la présentation influeront sensiblement sur la note.

Le chiffre en regard d'une question est une indication sur sa difficulté ou sa longueur.

1 Complexité

[3] **a)** Donner un algorithme pour trouver le minimum et le maximum de $n \geq 2$ valeurs contenues dans un tableau t en faisant au maximum $\lceil \frac{3}{2}n - 2 \rceil$ comparaisons.

On considère un tableau t contenant n entiers (positifs ou négatifs) et vérifiant $t[1] < t[2] < \cdots < t[n]$.
On souhaite trouver, pourvu qu'il existe, un indice $1 \leq i \leq n$ tel que $t[i] = i$.

[2] **b)** Donner un algorithme pour résoudre ce problème avec une complexité au pire en $\mathcal{O}(\log n)$.

[2] **c)** Montrer qu'on ne peut pas faire mieux en utilisant uniquement des comparaisons entre des éléments du tableau et des entiers.

2 Ensemble

On définit un type abstrait T par :

- Donnée : un sous-ensemble X de $\{1, \ldots, N\}$.
- Opérations :

$$\begin{aligned} \texttt{créer} &: \quad \text{Int} \to \text{T} \\ \texttt{prendre} &: \quad \text{T} \to \text{T} \times \text{Int} \\ \texttt{mettre} &: \text{T} \times \text{Int} \to \text{T} \end{aligned}$$

- `T(N) X` crée le sous-ensemble $X = \emptyset$ de $\{1, \ldots, N\}$.
- `X.prendre()` supprime un élément arbitraire de X et le retourne.
  L'ensemble X doit être non vide.
- `X.mettre(i)` ajoute l'élément i à l'ensemble X.
  On *doit* avoir $i \in \{1, \ldots, N\}$ et on *peut* avoir $i \in X$.

[3] **a)** Donner une implémentation complète de ce type abstrait pour laquelle les opérations `prendre` et `mettre` sont réalisées en temps constant.
Attention à bien gérer le fait qu'on peut appeler `X.mettre(i)` alors que $i \in X$.

3 Partitions

On définit une variante du type abstrait `partition` comme suit :

- Donnée : une partition P de $\{1,\ldots,N\}$, chaque classe $X \in P$ étant désignée par un entier de $\{1,\ldots,M\}$, *son nom*.
  Attention : le nom d'une classe n'est pas nécessairement un élément de la classe et deux classes distinctes ne peuvent avoir le même nom.
- Opérations :

$$\begin{aligned} \texttt{créer} &: \text{Int} \times \text{Int} \to \texttt{Partition} \\ \texttt{find} &: \texttt{Partition} \times \text{Int} \to \text{Int} \\ \texttt{union} &: \texttt{Partition} \times \text{Int} \times \text{Int} \times \text{Int} \to \texttt{Partition} \\ \texttt{renommer} &: \texttt{Partition} \times \text{Int} \times \text{Int} \to \texttt{Partition} \end{aligned}$$

- `Partition(N,M) P` crée une partition P composée des singletons de $\{1,\ldots,N\}$ et dont l'ensemble de noms est $\{1,\ldots,M\}$.
  Il faut $N \leq M$ et initialement, le nom du singleton $\{i\}$ est i.
- `P.find(i)` retourne le nom de la classe de P qui contient i.
- `P.union(x,y,z)` fusionne les classes de noms x et y en une classe de nom z.
  Ne fait rien s'il n'y a pas de classe de nom x ou pas de classe de nom y ou s'il y a déjà une classe de nom z et que $z \neq x$ et $z \neq y$.
- `P.renommer(x,y)` change le nom de la classe x en y.
  Ne fait rien s'il n'y a pas de classe de nom x ou s'il y a déjà une classe de nom y.

[5] **a)** Décrire une structure de données concrète pour implémenter très efficacement le type abstrait `partition` et donner les algorithmes pour les 4 opérations sur ce type. Les opérations `union` et `renommer` devront s'exécuter en temps constant. L'opération `créer` devra être en temps $\mathcal{O}(N+M)$. Le coût d'une suite d'opérations comportant $n-1$ `unions` et m `find` sera en $\mathcal{O}(n+m\alpha(n))$ où $\alpha(n)$ est l'inverse de la fonction d'Ackerman vue en cours.
Remarque : on peut bien sûr utiliser les théorèmes du cours.

On souhaite maintenant résoudre le problème des minima hors-ligne. La donnée est une suite d'entiers 2 à 2 distincts et d'instructions `extraire-min`, notées E dans la suite. Par exemple : $\sigma = 5, 4, 8, 2, E, 7, E, 1, 6, E, E, 3$.
Chaque instruction E doit afficher le plus petit entier qui précède et qui n'a pas déjà été affiché. Pour l'exemple ci-dessus on affichera 2, 4, 1, 5.
On cherche à résoudre ce problème avec un algorithme *hors-ligne*, i.e., on peut lire *toute* la donnée σ *avant* de commencer à afficher le résultat.

[5] **b)** Donner un algorithme pour résoudre le problème des minima hors-ligne lorsque les entiers de la suite σ forment une permutation de $\{1,\ldots,N\}$ et que σ contient au plus N instructions `extraire-min` (E). L'algorithme devra s'exécuter en temps $\mathcal{O}(n\alpha(n))$.
Indication : on pourra commencer par créer une partition dont la partie de nom $i \geq 1$ contient les entiers qui se trouvent entre le $(i-1)$-ème E et le i-ème E. Avec l'exemple ci-dessus, nous aurons les parties $X_1 = \{5,4,8,2\}$, $X_2 = \{7\}$, $X_3 = \{1,6\}$, $X_4 = \emptyset$, et $X_5 = \{3\}$.

Une question supplémentaire pour ceux qui s'ennuieraient.

[5] **c)** Montrer que la complexité d'une suite de n unions par taille *suivie* d'une suite de m find avec compression des chemins est au pire en $\mathcal{O}(n+m)$.
Remarque : il est important pour cette question de faire *toutes* les unions *avant* les find.