Algorithmique TD7
L3 Informatique – ENS Cachan

3 septembre 2019

Exercice 1

Le problème du minimum différé propose de maintenir à jour un ensemble dynamique T d'éléments du domaine $\{1, 2, \ldots, n\}$ pouvant subir les opérations d'insertion et d'extraction du minimum. Soit une séquence S de n appels à l'insertion et m appels à l'extraction du min, où chaque clé $i \in \{1, \ldots, n\}$ est insérée exactement une fois. On souhaite déterminer la clé retournée par chaque appel à l'extraction du min. Plus précisément, il s'agit de remplir un tableau extrait$[1, m]$ où pour tout $i = 1, 2, \ldots, m$, extrait$[i]$ est la clé retournée par le i-ième appel à l'extraction du min. Le problème est "différé" au sens où il est possible de traiter entièrement la séquence S avant de déterminer n'importe laquelle des clés retournées.

1. Dans l'instance ci-dessous du problème du minimum différé, chaque opéation d'insertion est représentée par un nombre et chaque extraction par E :

   $$4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5\,.$$

   Remplir le tableau extrait avec les valeurs adéquates.

2. Pour développer un algorithme, on découpe la séquence S en sous-séquences homogènes. Autrement dit, on représente S par

   $$I_1, E, I_2, E, I_3, \ldots, I_m, E, I_{m+1}\,,$$

   où chaque I_j est une suite (éventuellement vide) d'insertions. On commence par placer les clés de chaque I_j dans un ensemble K_j, qui est vide si I_j est vide. On exécute la procédure suivante :

   for i from 1 to n **do**
   | Déterminer j tel que i appartient à K_j;
   | **if** $j \neq m + 1$ **then**
   | | extrait$[j] \leftarrow i$;
   | | $\ell \leftarrow \min\{k > j \mid K_k \text{ exists}\}$;
   | | $K_\ell \leftarrow K_j \cup K_\ell$; détruire K_j;
   | **end**
   end

   Démontrer que cette procédure résout le problème.

3. Expliquer comment implémenter cet algorithme efficacement et donner sa complexité.

Exercice 2

Un *tas 2-3-4* est un arbre dont les clés sont uniquement enregistrées dans les feuilles. Chaque feuille x possède une unique clé $c(x)$ et un pointeur vers son père $p(x)$. Chaque noeud interne x possède :

— Un nombre de fils $n(x)$ qui est égal à 2, 3 ou 4.
— Un pointeur vers chacun de ses fils.

— Un pointeur vers son père.
— Une valeur $s(x)$ qui est égale à la plus petite clé contenue dans le sous-arbre enraciné en x.

La racine contient également une valeur $h(x)$ égale à la hauteur du tas. Toutes les feuilles ont la même profondeur. Pour toutes les questions, on demande d'écrire la procédure et de donner sa complexité.

1. Écrire une fonction MINIMUM(t) qui retourne un pointeur vers la feuille de clé minimale du tas t.
2. Écrire une fonction DIMINUE(t, x, k) qui diminue la valeur de la clé de la feuille x de t à k (on suppose que t ne contient pas la clé k).
3. Écrire une fonction INSERTION(t, k) qui insère la clé k dans t (on suppose que t ne contient pas la clé k).
4. Écrire une fonction SUPPRIME(t, x) qui supprime une feuille x donnée de t.
5. Écrire une fonction UNION(t_1, t_2) qui unit les tas t_1 et t_2 en les détruisant.

Exercice 3

On considère une pyramide de nombres, qu'on va ici représenter comme une matrice triangulaire : la première ligne (d'indice 0), contient un élément à la colonne 0, la deuxième ligne (d'indice 1), contient deux éléments numérotés 0 et 1, ... Un chemin est une suite d'indices $(u_i)_{i\in\mathbb{N}}$ tels que $u_i \leq u_{i+1} \leq u_i + 1$. Donner un algorithme qui calcule un chemin qui maximise la somme des nombres contenus dans les case parcourues.

Exercice 4

On s'intéresse ici à l'application d'une permutation miroir, que l'on définit comme suit. Prenons en entrée un tableau A de taille $n = 2^k$, avec $k \geq 0$. On peut voir chaque indice de A, comme un entier binaire de taille k que l'on écrira $< a_{k-1}, \ldots, a_1, a_0 >$, soit $a = \sum_{i=0}^{k-1} a_i 2^i$. On définit maintenant :

$$\begin{aligned}\text{rev}_k(a) &= \text{rev}_k(< a_{k-1}, \ldots, a_1, a_0 >) \\ &=< a_0, a_1, \ldots, a_{k-1} > \\ &= \sum_{i=0}^{k-1} a_{k-i-1} 2^i\end{aligned}$$

On s'intéresse donc à créer le tableau B défini par $B[i] = A[\text{rev}_k(i)]$.

1. Donner un algorithme qui permet d'effectuer une permutation miroir en temps $O(nk)$.
2. On se propose maintenant d'implémenter la permutation miroir en se servant de la fonction incr définit comme : $\text{incr}(a) = \text{rev}_k(\text{rev}_k(a)+1)$. Montrer comment implémenter la fonction incr de telle sorte que le calcul de la permutation miroir soit en temps $O(n)$.