Université Paris Diderot – Paris 7 — L3 Informatique
Algorithmique — Année 2012-2013

Devoir à la maison

à rendre pour le 7 janvier 2013

Exercice 1 [Étude d'algorithme : le tri par tas (HeapSort)]

Dans cet exercice, vous pouvez répondre à chaque question indépendemment en supposant démontré l'énoncé des questions précédentes.
On considère l'algorithme du tri par tas. Il est en deux étapes :

1. Construction d'un tas à partir d'un tableau non trié
2. Tri à partir du tas

C'est un tri *en place* : il n'utilise pas de mémoire auxiliaire. À l'étape 1 le tableau non trié est donc *transformé* en tas-max. Puis à l'étape 2 on transforme ce tas-max en tableau. Tout cela en faisant des échanges d'éléments du tableau comme opération élémentaire.

Le tableau T que l'on considère a ses indices qui vont de 1 à n (où n, constant lors de l'algorithme, est le nombre d'éléments à trier). De plus, pour simplifier, on suppose que $n = 2^k - 1$ (n est 1 moins une puissance de 2). Un tas permet d'implémenter un arbre binaire complet : les descendants d'un nœud placé en $T[i]$ sont, rappelons-le, $T[2i]$ et $T[2i + 1]$. Le *sous-arbre de racine i* est l'ensemble des cases correspondant à $T[i]$ et à ses descendants.

Algorithme Tri_par_tas
Données : T, un tableau de n réels (la première case numérotée 1)
Résultat : T est modifié. Ne renvoie rien
// NB : étape 1 : transformation de T en tas
pour i *de* $n/2$ *à* 1 **faire**
  Descente(i,T[1..n])
// NB : étape 2 : tri. On extrait la racine du tas $T[1..i]$ pour la mettre en $T[i]$
pour i *de* n *à* 1 **faire**
  Échanger $T[1]$ et $T[i]$
  // NB : équivaut à supprimer $T[1]$ car i va décroître
  Descente(1,T[1..i-1])

Procédure Descente(x : indice, $T[1..b]$, un tableau de b ésultat T est modifié. Ne re3 1cr2 .73re

Question 3. En déduire qu'à la fin de la première étape de l'algorithme (de la première boucle **pour**) T a été transformé en un tas.

Question 4. On pose que le niveau des feuilles est 1, et celui de la racine est k (le niveau d'un nœud est 1+niveau de ses fils). Montrer que pour un nœud $T[i]$ au niveau a, le nombre de tours de la boucle **tant que** de la procédure *Descente*$(i, T[1..n])$ est au plus a.

Question 5. En observant combien il y a de nœuds à chaque niveau, écrivez une formule qui majore le nombre *moyen* de tours faits par la boucle **tant que** de la procédure *Descente*( ), lors de chacun des appels de la boucle **pour** de la première étape (il faut prendre la moyenne pour tout i entre 1 et n de ce qu'on a calculé à la question précédente).

En utilisant l'identité remarquable (quand $|x| < 1$)

$$\sum_{i=0}^{i=+\infty} i x^{-i} = \frac{x}{(x-1)^2}$$

montrez cette moyenne est plus petite que 2.

Question 6. En déduire que la tranformation d'un tableau non trié en tas-max (étape 1) prend un temps linéaire.

Comparer avec le temps de l'algorithme naïf :

pour i *de 1 à* n **faire**
  $Ajouter\_Tas(T, element[i])$

Question 7. On note $T[a..b]$ le sous-tableau des éléments d'indices entre a et b (inclus). Si $a > b$ alors T est vide. Considérons l'invariant suivant.

invariant 2 $T[1..i]$ est un tas-max de i éléments. $T[i+1..n]$ est un tableau trié dont tous les éléments sont supérieurs ou égaux à ceux de $T[1..i]$.

Montrer que cet invariant est vrai au début de l'étape 2 pour $i = n$. Et que s'il est vrai pour i alors effectuer :

Échanger $T[1]$ et $T[i]$
// NB : équivaut à supprimer $T[1]$ car i va décroître
Descente(1,T[1..i])
fait qu'il devient vrai pour $i - 1$.

En déduire que l'étape 2 transforme un tas-max en tableau trié.

Question 8. Donner le temps de la deuxième étape et le temps total de l'algorithme.

Exercice 2 [Parcours en profondeur] On vous propose d'écrire un algorithme *itératif* qui réalise le Parcours en Profondeur d'un graphe avec l'utilisation d'une *pile* (analogue, mais pas idéntique, à l'utilisation d'une file dans l'algorithme itératif du Parcours en Largeur).

Au moins deux utilisations très différentes de la pile sont possibles pour réaliser un parcours correct, complet et de complexité optimale. Ces deux utilisations effectuent les opérations de push et pop en fonction de critères très différentes basés sur la couleur d'un noeud (blanc, gris, noir). Vous devrez écrire un algorithme correct pour au moins l'une de ces deux utilisations. Vous aurez plus de points si vous fournissez une solution pour chacune des deux.

Avec les trois couleurs {blanc, gris, noir} et les deux états {dans-la-pile, pas-dans-la-pile}, on peut créer six couples. Pour chaque couple, dire si il n'est jamais réalisé par votre algorithme et quelle est sa signification pour un noeud qui se trouve dans cet état.

Pour toute procédure (principale ou auxiliaire) de votre algorithme, vous devrez donner sa complexité et devrez indiquer (par un pseudo-code clair et commenté) comment elle est réalisée

– dans le cas où le graphe est implémenté par listes d'adjacence ;
– dans le cas où le graphe est implémenté par matrice d'adjacence.

Les bornes de complexité pour obtenir la totalité des points de l'exercice (en cas d'algoritme correct) sont :
– $O(n+m)$ en espace et $O(m)$ en temps, dans le cas où le graphe est implémenté par listes d'adjacence ;
– $O(n^2)$ en espace et $O(n^2)$ en temps, dans le cas où le graphe est implémenté par matrice d'adjacence.

Les algorithmes sans commentaires ni explications (concises !) ne seront pas pris en considération.

Exercice 3 [Fiabilité d'un graphe]

On définit qu'un graphe non orienté $G(V, E)$ est k-*fiable* si pour tout sous ensemble F de V de taille inférieure ou égale à k, le graphe $G(V - F, E)$ est connexe.

1. Exhiber un graphe à 4 nœuds qui est 1-fiable et un graphe à 4 nœuds qui n'est pas 1-fiable.
2. Que signifie qu'un graphe est $n-2$ fiable ? Prouvez le.
3. Proposer une procédure qui prend en entrée un graphe G et détermine si G est 1-fiable. Quelle est sa complexité ?
4. Proposer une procédure qui prend en entrée un graphe G et un entier k et détermine si G est k-fiable. Quelle est sa complexité ?