L3-INFO Algorithmique

Quick 2 – 10 novembre 2023 – durée 1h

Les documents et les téléphones (incluant smartphone, tablettes,... tout ce qui contient une interface réseau) sont interdits. Les calculettes sont autorisées.
Seuls les dictionnaires pour les personnes de langue étrangère sont autorisés.
Toutes les réponses doivent être justifiées. Le barème est indicatif.

---

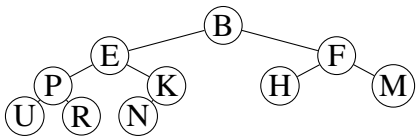
1 Premiers nœuds dans un tas (*6 points*)

Dans cet exercice, on considère des tas-min (où chaque nœud contient une clé **inférieure** à celles de ses fils), représentés sous forme de tableaux indexés à partir de 0, comme vu en TD.

Aucune opération d'arbre n'est autorisée, on ne travaille que sur le tableau et ses indices.

Les différentes opérations sur les tas **ne sont pas supposées implémentées**, il faudra donc les réécrire vous-mêmes si vous pensez qu'elles sont utiles.
Pour simplifier, on pourra supposer que tous les éléments du tas sont distincts.

Pour tous les exemples donnés dans l'énoncé, on utilisera le tas :



Question 1 : *(1 pt)*

On cherche à déterminer le 2è plus petit élément dans un tas (contenant au moins 2 éléments).

Dans notre exemple, il faudrait renvoyer E.

Donnez un algorithme qui résout ce problème en temps constant.

Réponse 1 :

Il suffit de comparer les fils gauche et droit de la racine et de renvoyer le plus petit des deux. Attention au cas où il n'y a qu'un seul fils.

si *n==2* **alors**
  renvoyer *T[1]*
sinon
  renvoyer *min(T[1], T[2])*

Question 2 : *(2 pts)*

On cherche à déterminer le 3è plus petit élément dans un tas. On supposera pour cette question que le tas comporte largement plus de 3 éléments (au moins 10 par exemple).

Dans notre exemple, il faudrait renvoyer F.

Donnez un algorithme qui résout ce problème en temps constant.

Réponse 2 :

Attention, les fils du plus petit fils de la racine sont aussi des candidats. Il faut donc comparer :

— le plus grand fils de la racine

— les deux fils du plus petit fils de la racine.

si *T[1] < T[2]* **alors**
  renvoyer *min(T[2], T[3], T[4])*
sinon
  renvoyer *min(T[1], T[5], T[6])*

Pour les cas où le tas contient moins de 7 éléments il faudrait ajouter des tests supplémentaires pour ne pas considérer les cellules hors bornes.

Question 3 : *(3 pts)*

Donnez un algorithme qui renvoie le plus **grand** élément dans un tas.
Dans notre exemple, il faudrait renvoyer U.
Donnez un algorithme qui résout ce problème en strictement moins de $n-1$ comparaisons.

Réponse 3 :

Cette fois l'aspect ordonné des tas ne nous aidera pas beaucoup. On peut cependant se limiter à parcourir les feuilles du tas, c'est-à-dire les $\lfloor n/2 \rfloor$ derniers éléments du tableau.

Max(T, n)
Données : T : un tas binaire sous forme de tableau, n : nombre d'éléments dans le tas
Résultat : le maximum de T
m := $T[n-1]$
pour $i := n-2$ **to** $\lceil n/2 \rceil$ **faire**
  si *T[i] > m* **alors**
    m := $T[i]$
  renvoyer m

Coût en $n/2$, soit une complexité linéaire.

2 Preuve d'algorithme *(14 points)*

Soit un tableau T (indexé de 0 à $n-1$). On cherche à inverser l'ordre des éléments présents dans T. Par exemple, pour $T = [E, K, S, G, N, P]$, l'objectif est de transformer ce tableau pour que $T = [P, N, G, S, K, E]$.

On propose l'algorithme suivant :

INVERSER(T, n)
Données : Un tableau T de n éléments quelconques
Résultat : Le tableau T est modifié pour inverser l'ordre de ses éléments

```
1 i := 0
2 j := n − 1
3 tant que i < j faire
4 |   x := T[i]
5 |   T[i] := T[j]
6 |   T[j] := x
7 |   i := i + 1
8 |   j := j − 1
```

Question 4 : *(2 pts)*

Justifiez que cet algorithme se termine.

Réponse 4 :

Le variant $j - i$ convient : il est bien entier, positif, et strictement décroissant (il diminue de 2 à chaque itération).

Question 5 : *(2 pts)*

On appelle T le tableau initial, et T' le tableau obtenu en fin d'exécution de l'algorithme.

a. Représentez sous forme de schéma la postcondition qui exprime que notre algorithme réalise la tâche souhaitée.
b. Exprimez également cette postcondition en faisant référence explicitement à des indices des tableaux T et T' (soit par une phrase en français, soit par une formule de la forme $\forall i \ldots T'[i] \ldots$).

Réponse 5 :

$\forall 0 \leqslant i < n,\ T'[i] = T[n-1-i]$

Autrement dit, tout élément d'indice i dans le tableau résultat T' correspond à l'élément d'indice $n-1-i$ dans le tableau initial T.

Question 6 : *(3 pts)*

a. Exprimez un invariant vérifié par la boucle Tant Que de notre algorithme qui permettra de démontrer cette postcondition (encore une fois, en français ou comme une formule, du moment que les indices concernés sont explicites).
b. Présentez également votre invariant sous forme d'un schéma.

Réponse 6 :

$\forall 0 \leqslant k < i,\ T'[k] = T[n-1-k]$ et de même pour tout $j < k < n$.

Autrement dit tous les éléments à gauche de i et à droite de j (strictement) ont déjà été inversés comme voulu.

Question 7 : *(1 pt)*

Justifiez que cet invariant est vérifié après la ligne 2 de l'algorithme.

Réponse 7 :

Comme $i = 0$ et $j = n - 1$ il ne concerne aucune cellule de T', il n'y a donc rien à vérifier.

Question 8 : *(3 pts)*

Démontrez que la boucle Tant Que maintient effectivement cet invariant.

Réponse 8 :

La boucle ne modifie (échange) que les éléments d'indices i et j, et à toute itération on a $j = n - 1 - i$.

La propriété $\forall 0 \leqslant k < i,\ T'[k] = T[n - 1 - k]$ reste donc vérifiée (éléments inchangés). Comme $i' = i + 1$ il nous faut également vérifier qu'en fin d'itération $T'[i] = T[n - 1 - i]$, ce qui est assuré par l'échange de $T[i]$ et $T[j]$.

De même pour $j' < k < n$.

Question 9 : *(2 pts)*

Démontrez enfin la postcondition à l'aide de l'invariant en sortie de boucle.

Réponse 9 :

En sortie de boucle $i = j$ (tableau de taille impaire) ou $i = j + 1$.

Dans le cas pair $0 \leqslant k < i$ et $j < k < n$ recouvrent exactement $0 \leqslant k < n$, donc on a bien inversé la position de tous les éléments $T[i]$ comme voulu.

Dans le cas impair l'élément $T[(n - 1)/2]$ n'est pas concerné par l'invariant, mais comme il est central dans le tableau on n'a pas à le déplacer.

Question 10 : *(1 pt)*

Si la condition de la boucle Tant Que était $i < n$, notre algorithme serait incorrect. Expliquez quelle étape de la démonstration que nous venons d'écrire ne serait plus valable.

Réponse 10 :

L'invariant cesserait d'être vérifié à partir de l'itération $i = n/2 + 1$ puisqu'on replacerait des éléments à leur position originale.