7,2/9

Devoir 1 IFT2015

latex +1

Catherine Larivière 0955948

Dominique Vigeant 20129080

2 juin 2021

1 Partie pratique

Remit sous la forme ListeCirculaire.java

2 Partie théorique

0,8/1

1. Je vous ai donné des définitions formelles de T_{best}, T_{avg} et T_{worst}. Démontrez formellement que $T_{best}(N) \leq T_{avg}(N) \leq T_{worst}(N)$.

$T_{avg}(m_j) = \frac{1}{n}\sum_{i=1}^{n} T_i(m_j, N) \geq \frac{1}{n}\sum_{i=1}^{n} minT_i$. Cette deuxième partie est égale à $\sum_{i=1}^{n} minT_{best}$, ce qui est égal à T_{best}.
Ensuite, $T_{avg}(m_j) = \frac{1}{n}\sum_{i=1}^{n} T_i(m_j, N) \leq \frac{1}{n}\sum_{i=1}^{n} maxT_i$. Cette deuxième partie est égale à $\sum_{i=1}^{n} maxT_{worst}$, ce qui est égal à T_{worst}.
Et donc $T_{best}(N) \leq T_{avg}(N) \leq T_{worst}(N)$.

manque un peu de rigueur -0,2

0,8/1

2. Si vous êtes plus à l'aise avec cela, vous pouvez remplacer le symbole epsilon par un signe d'égalité.

a) Écrivons $f < g$ pour noter que $f(N) \in O(g(N))$ mais $g(N) \in O(f(N))$. Trouvez trois fonctions f, g et h telles que

$$f < g < h$$

et trois fonctions T_{best}, T_{avg} et T_{worst} telles que

$$T_{best}(N) \leq T_{avg}(N) \leq T_{worst}(N),\ N \geq 0$$

et

$$T_{worst} \in ?(f(N)),\ T_{avg} \in ?(g(N)), T_{best} \in ?(h(N)).$$

Vous avez le droit de remplacer chaque point d'interrogation par le symbole O ou Ω,mais vous ne pouvez pas faire le même choix trois fois.

Posons :

$$f(N) = 1 \quad g(N) = N \quad h(N) = N^2$$

et aussi :

$$T_{best}(N) = logN \quad T_{avg}(N) = NlogN \quad T_{worst}(N) = N^2logN$$

Nous avons bien, comme demandé,

$$f < g < h$$

et

$$T_{best}(N) \leq T_{avg}(N) \leq T_{worst}(N),\ N \geq 0$$

On peut maintenant remplacé dans la questions les ?.

$T_{worst}(N) \in \Omega f(N)$, puisque f(N) est une borne inférieure de T_{worst}

$T_{avg}(N) \in \Omega g(N)$, puisque g(N) est une borne inférieure de T_{avg}

$T_{best}(N) \in \mathcal{O}h(N)$, puisque h(N) est une borne supérieure de T_{best}

✓

b) Trouvez trois fonctions f, g et h, et une combinaison quelconque de O et Ω pour remplacer les trois points d'interrogation pour définir les trois classes $?(f(N))$ $?(g(N))$ et $?(h(N))$, mais tel qu'il n'est pas possible de trouver T_{best}, T_{avg} et T_{worst} telles que

$$T_{best} \in ?(f(N)),\ T_{avg} \in ?(g(N)), T_{worst} \in ?(h(N))\ et$$

$$T_{best}(N) \leq T_{avg}(N) \leq T_{worst}(N),\ N \geq 0$$

L'exemple d'impossibilité doit impliquer les deux fonctions T_{avg} et T_{worst}, et il faut donner la preuve de l'impossibilité.

Posons :

$$f(N) = 2^N \quad g(N) = N^2 \quad h(N) = NlogN$$

Avec :

$$T_{best}(N) \in \Omega f(N), \quad T_{avg}(N) \in \Omega g(N) \text{et} \quad T_{worst}(N) \in \mathcal{O}h(N)$$

Ceci est impossible, parce que $T_{worst} < T_{avg}$. Prenons un exemple pour prouver par contradiction. Nous avons que $T_{avg}(N) \in \Omega g(N) = N^2$. Donc on peut supposer que $T_{avg} = n^3$. De même, $T_{worst}(N) \in \mathcal{O}h(N) = NlogN$, on peut supposer que $T_{worst} = n$. Et on voit facilement que $T_{worst} < T_{avg}$.

un exemple n'est pas une preuve !

-0,2

1,4/2

3.

a) Suppose $T_1(N) = O(f(N))$ and $T_2(N) = O(f(N))$. Which of the following are necessarily true ?

$$\frac{T_1(N)}{T_2(N)} = O(1)$$

Faux, nous trouvons facilement comme contre-exemple $T_1(N) = 2^N$ et $T_2(N) = 2^{2N}$, alors que $f(N) = 2^N$.

non, $2^{2N} \notin O(2^N)$ -0,1

et pourquoi est-ce un contre-exemple ici ? -0,5

b) An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 if the running time is the following (assume low-order terms are negligible) :

a) en temps linéaire, on trouve que $\frac{500}{100} = 5$, donc 5 fois plus de temps. Nous trouvons $5 * 0,5 = 2,5ms$
b) en temps $O(NlogN)$, on calcul $\frac{500log500}{100log100} \approx 6,75$, donc $6,75 * 0,5 = 3,375ms$
c) en temps quadratique, $\frac{500^2}{100^2} = 25$, donc $25 * 0,5 = 12,5ms$
d) en temps cubique, $\frac{500^3}{100^3} = 125$, donc $125 * 62,5ms$
Il est intéressant ici de voir comment le rythme de croissance des fonctions fait rapidement augmenter le temps d'exécution.

4. For each of the following six program fragments, give an analysis of the running time (Big-Oh will do).

(2)
```
sum = 0;
for ( i = 0; i < n; i++)
    for ( j = 0; j < n; j++)
        sum++;
```

On parcours la boucle for de i $n-1$ fois, et la boucle imbriqué j $n-1$ fois, nous avons donc un temps d'exécution de l'ordre de N^2.

(3)
```
sum = 0;
for ( i = 0; i < n; i++)
    for ( j = 0; j < n * n; j++)
        sum++;
```

On parcours la boucle for de i $n-1$ fois, et la boucle imbriqué j $(n-1)*(n-1)$ fois, nous avons donc un temps d'exécution de l'ordre de N^3.

(4)
```
sum = 0;
for ( i = 0; i < n; i++)
    for ( j = 0; j < i; j++)
        sum++;
```

On parcours la boucle for de i $n-1$ fois, et dans le pire cas, la boucle imbriqué j $n-2$ fois, nous avons donc un temps d'exécution de l'ordre de N^2.

(5)
```
sum = 0;
for ( i = 0; i < n; i++ )
    for ( j = 0; j < i * i; j++ )
        for ( k = 0; k < j; k++ )
            sum++;
```

On doit regarder le temps d'exécution dans le pire cas. Donc, on parcours la boucle i $n-1$ fois, la boucle j $(n-1)^2$ fois, la boucle k $n-3$, et donc nous avons un temps d'exécution de l'ordre de N^5.

Par besoin de faire en pire cas
En fait c'est un peu moins précis

5. Consider the following algorithm (known as Horner's rule) to evaluate $f(x) = \sum_{i=0}^{N} a_i x^i$:

```
poly = 0;
for ( i = n; i >= 0; i--)
    poly = x * poly + a[i];
```

a) Show how the steps are performed by this algorithm for $x = 3$, $f(x) = 4x^4 + 8x^3 + x + 2$.

On commence avec $n = 4$ puisque le degré de notre polynôme est 4 et $x = 3$:

```
poly = 0;
for ( i = 4; i >= 0; i--)
    poly = x * poly + a[i];
```

1^{ere} itération :

```
poly = 0;
for ( i = 4; i >= 0; i--)          // 4 >= 0
    poly = 3 * 0 + a[4];           // poly = 3 * 0 + 4 = 4
```

2^{e} itération :

```
poly = 0;
for ( i = 3; i >= 0; i--)          // 3 >= 0
    poly = 3 * 4 + a[3];           // poly = 3 * 4 + 8 = 20
```

3^{e} itération :

```
poly = 0;
for ( i = 2; i >= 0; i--)          // 2 >= 0
    poly = 3 * 20 + a[2];          // poly = 3 * 20 + 0 = 60
```

4^{e} itération :

```
poly = 0;
for ( i = 1; i >= 0; i--)          // 1 >= 0
    poly = 3 * 60 + a[1];          // poly = 3 * 60 + 1 = 181
```

5^{e} itération :

```
poly = 0;
for ( i = 0; i >= 0; i--)          // 0 >= 0
    poly = 3 * 181 + a[0];         // poly = 3 * 181 + 2 = 545
```

6^{e} itération :

```
poly = 0;
for ( i = -1; i >= 0; i--)         // -1 < 0
    poly = x * poly + a[i];
```

poly = 545 (Résultat final)

b) Explain why this algorithm works.

En utilisant la récursion, l'algorithme s'assure de calculer tout les degrés de polynômes, puisque la longueur du polynômes à calculer peut être variable.

c) What is the running time of this algorithm ?

Nous avons une simple boucle for, alors l'algorithme est de l'ordre de $O(n)$.

6. A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree.

Prouvons, par induction, que le nombre de noeuds complets plus un est égal aux nombres de feuilles dans un arbre binaire non vide.
Posons

$$k = \text{nombre de noeuds}$$
$$n = \text{nombre de noeuds pleins}$$

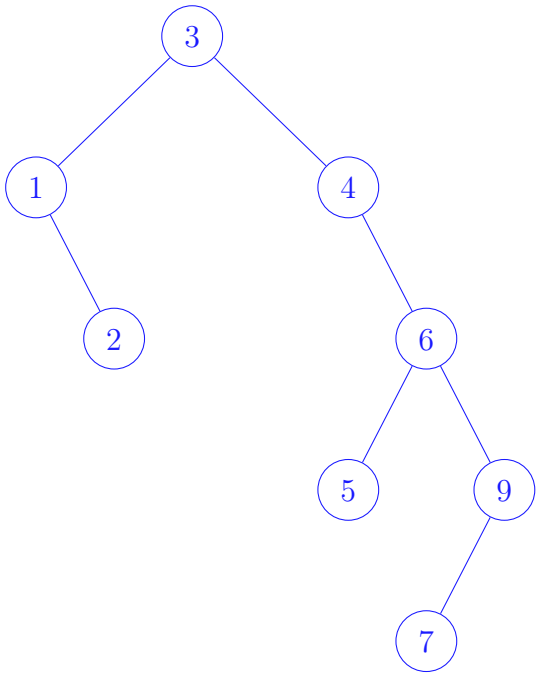
On veut prouver que pour tout k dans les naturels, $P(k)$, un arbre binaire avec k noeuds, a n noeuds pleins et $n+1$ feuilles.
Cas de base :
lorsque $k = 1$, nous avons $P(1) = 0 + 1 = 1$. Un arbre binaire avec un seul noeud n'a pas de noeud plein, et a une seule feuille, lui-même.
Pas d'induction : on prouve que $P(k) \implies P(k+1)$
Si on ajoute un noeud a une feuille, alors le nombre de noeud plein et le nombre de feuille reste inchangé.
Si on ajoute un noeud a un noeud qui a un enfant, l'arbre qui avait n noeud plein contient maintenant $n+1$ noeud plein. Puisque le parent devient un noeud complet, alors le nombre de feuille augmente de un. Le nombre de feuille devient donc $(n+1)+1 = n+2$. Le nombre de noeud plein est excatement un de moins que le nombre de feuille, et donc $P(k+1)$ est vraie.
Donc, puisque $P(k+1)$ est vrai, alors on peut conclure que $P(k) \implies P(k+1)$ est vraie, et donc que le nombre de noeud pleins plus un est égal aux nombres de feuilles dans un arbre binaire non vide. [1]

tout est bon, mais vu que c'est pris du net je vais enlever 0,5/2
(Sans la source vous auriez eu 0)

7.

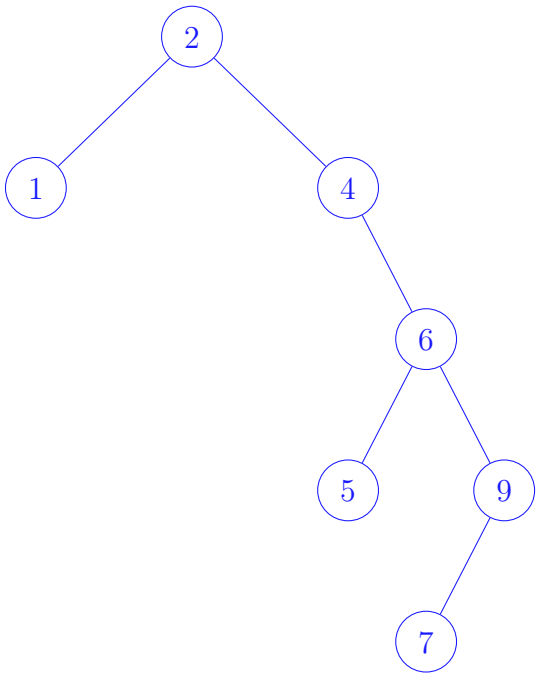
a) Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.

On construit l'arbre, toujours du haut vers le bas. Si l'élément à ajouter est plus grand que le noeud, on le place à droite, sinon à gauche. À chaque nouvel élément, on refait le chemin a partir du haut. On obtient :

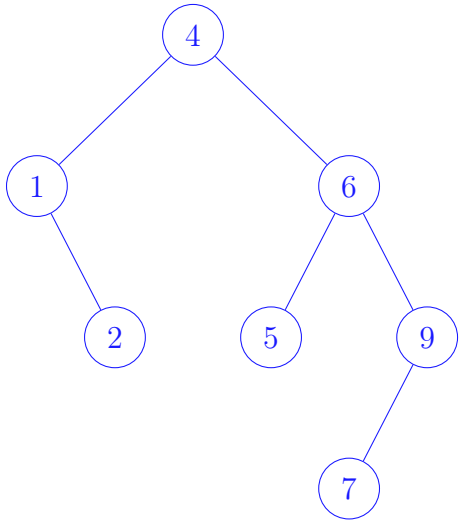


b) Show the result of deleting the root.

Pour ce qui est d'enlever la racine par échange avec le prédécesseur, on veut trouver le plus grand élément dans le sous-arbre gauche à partir de la racine. On prend donc la branche gauche, puis on descend à droite jusqu'à trouver un noeud avec un pointeur null à droite. Ce qui nous donne :



Pour enlever la racine par un échange avec le successeur, même principe, mais on trouve le plus petit élément dans le sous-arbre de droite. On descend la branche de droite, puis on descend à gauche jusqu'à trouver un noeud avec un pointeur null à gauche. Résultat :



Références

[1] CHERRY$_D$*eveloper*. *Is my proof that, the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree, correct ?* URL : `https://math.stackexchange.com/questions/2196027/is-my-proof-that-the-number-of-full-nodes-plus-one-is-equal-to-the-number-of-le`.