UNIVERSITÉ LAVAL

TP2 - Question 3

réalisé par Claire Bouttes - 536793110
IFT-3001

Table des matières

Introduction **2**

1 Analyse en pire cas **3**

2 Calcul du temps d'exécution amorti **3**

Conclusion **3**

6 décembre 2020

Introduction

A partir d'un algorithme donnée par la question 3, nous allons analyser la fonction Suivant premièrement en pire cas, puis nous l'analyserons en analyse amorti en utilisant la méthode du comptable.

On prendra l'opération élémentaire (car s'effectue en temps constant) $S.pop()$ comme opération de base.

Algorithm 1: Init(racine)

Initialise une pile vide S;
$S.push(1)$;
$n \leftarrow racine$;
return n;

Algorithm 2: Suivant()

while $S \neq \emptyset \wedge S.top() > c_n$ **do**
    $n \leftarrow p_n$;
    $S.pop()$;
end
if $S = \emptyset$ **then**
    return nil;
end
$i \leftarrow S.pop()$;
$S.push(i+1)$;
$S.push(1)$;
$n \leftarrow E_n[i]$;
return n;

1 Analyse en pire cas

En pire cas, on se retrouve au bout d'un arbre avec une seule branche (de profondeur k donc) et on doit remonter jusqu'à la racine pour finalement retourner nil. On a alors :
$C_{WORST}(n) = \sum_{i=0}^{n} 1$
$C_{WORST}(n) = n$

Ce pire cas s'effectue après avoir été executé k fois avec un temps d'exécution de 1. Il survient quand on retourne en arrière dans l'arbre. Or initialement, il a fallut avancé au moins k fois pour pouvoir reculer k fois sachant qu'au premier appel de *Suivant* dans un programme, on part de la racine.

On est ainsi en $\theta(k)$ avec k correspondant au nombre d'exécution de *Suivant* pour k noeuds.

2 Calcul du temps d'exécution amorti

Ce pire cas s'effectue après avoir été executé k fois avec un temps d'exécution de 1. Il survient quand on retourne en arrière dans l'arbre. Or initialement, il a fallut avancé au moins k fois pour pouvoir reculer k fois sachant qu'au premier appel de *Suivant* dans un programme, on part de la racine.

C'est-à-dire pour une fonction *ParcourArbre* parcourant un arbre entier.

Pour $C_{amorti}(k) = 2$
Soit B_i la banque de temps après avoir ajouté i éléments.
$B_0 = 0$
Pour la première exécution de la procédure *Suivant*, on est forcément dans un temps d'exécution de 1 (ou 0 si l'arbre est vide) étant donné qu'on ne peut reculer depuis la racine.
$B_1 = B_0 + C_{amorti}(0) - C_{reel}(0) = 0 + 2 - 1 = 1$

Ainsi, on a $B0 \geq 0$ et $B1 \geq 0$.

Supposons que $B_i \geq 0$ pour $0 \leq i \leq k$.

- Si n, le noeud sur lequel on se trouve, a un enfant vers lequel itéré, le temps d'exécution sera de 1. Ce k+1 ième appel ajoute 2 unités de temps à la banque de temps. $B_{k+1} = B_k + 2 - 1 \geq 1$.
- Si n n'a pas d'enfant sur lequel itéré, le temps d'exécution sera alors égal au nombre de noeuds sur lesquels remonter. Or on a pu stocker 1 k fois. Pour rappel, en pire cas, nous somme en $\theta(k)$. Ainsi $B_{k+1} = B_k + 2 - k$ donne $B_{k+1} = k + 2 - k \geq 2$

Conclusion

Nous avons démontré que charger 2 unités de temps est suffisant pour ne pas mener la banque de temps en faillite.
Puisque $C_{amorti}(n) \leq 2$ pour tout n, nous avons $C_{amorti}(n) \in O(1)$.
Puisque $C_{amorti}(n) \geq C_{best}(n) \in \omega(1)$, nous avons $C_{amorti}(n) \in \theta(1)$.