Rapport de TP 4MMAOD : Génération d'ABR optimal

ARGENSON Benjamin (ISSC)
MAJOU Steve (ISSC)

14 avril 2017

1 Principe de notre programme (1 point)

Notre programme se base principalement sur deux fonctions récursives. La fonction *Bellman* qui comme son nom l'indique, implémente l'équation de Bellman qui permet de calculer le nombre minimum moyen de comparaisons pour accéder à un élément donné de l'arbre. Cette fonction stocke également dans un tableau dans un tableau pour chaque sous arbres l'élément à placer aux sommets de façon à ce qu'il soit optimal ainsi que le cout correspondant.

La fonction *bstree* permet de construire l'arbre en question à partir des résulats de la fonction *Bellman* stockés dans des tableaux. Cette fonction se base sur le fait que pour qu'un arbre soit optimal il faut que la structure de ses deux sous-arbres gauche et droite soient également optimale. A noter que nos deux fonctions récursives utilisent la mémoïsation grâce à des tableaux qui stockent le résutat de chaque appel. Nous stockons également les résultats de la somme des probabilités de i à j dans un tableau afin de minimiser les calculs. Ainsi, on a quatre tableaux pour stocker les racines, le nombre de comparaisons, les calculs de somme et un tableau correspondant à l'arbre optimal recherché qui résulte de la fonction *bstree*

2 Analyse du coût théorique (2 points)

2.1 Nombre d'opérations en pire cas :

Justification : Equation de Bellman :

$$\phi(i,j) = \min_{k\in\{i+1,\dots,j-1\}} [\phi(i,k-1) + \phi(k+1,j)] + \sum_{n=i}^{j} p_n$$

Pour le calcul de la somme des i,j, il faut **j-i+1** opérations. Pour le nombre d'appels récursifs à partir de $\phi(i,k-1)$, il faut, pour que k-1 = i, **i-k** additions. Pour le nombre d'appels récursifs à partir de $\phi(k+1,j)$, il faut, pour que j = k+1, **k-j** additions. Ainsi, le nombre total de comparaisons des appels récursifs de la fonction de Bellman çi-dessus peut être modélisé par :

$$\sum_{k=1}^{n}\sum_{i=1}^{k}\sum_{j=k+1}^{n}(k-i)(j-k) = O(n^3)$$

2.2 Place mémoire requise :

Justification : On stocke les différents résultats dans 4 tableaux de n^2 cases. La place mémoire est donc en O(n^2).

2.3 Nombre de défauts de cache sur le modèle CO :

Justification :

3 Compte rendu d'expérimentation (2 points)

3.1 Conditions expérimentaless

3.1.1 Description synthétique de la machine :

Machine : Macbook Pro **2,7 GHz** Intel Core **i7**, **16 Go 1600 MHz** DDR3, El Capitan. Tests effectués sur une machine virtuelle CentOS a partir de VMware Fusion avec aucun autre programme en cours et une seule session active.

3.1.2 Méthode utilisée pour les mesures de temps :

Nous avons utilisé la fonction

$$gettimeofday(structtimeval * tv, structtimezone * tz)$$

avant et apres le processus, puis enfin nous avons soustrait les deux valeurs apres mise en forme dans le bon format. Les tests ont été effectués en microsecondes (us), puis nous avons convertis en secondes pour les benchmark 3,4,5,6. Les mêmes tests ont été effectués 5 fois a la suite.

3.2 Mesures expérimentales

Remarque : le coût du patch a été calculé à partir des *probabilités* et non des fréquences. Ainsi cela représente le nombre de comparaisons moyen pour rechercher un élément de l'arbre.

	coût du patch	temps min	temps max	temps moyen
benchmark1	2.114286	21 us	30 us	25.4 us
benchmark2	2.568627	31 us	39 us	33.8 us
benchmark3	8.465934	3,740 s	3,798 s	3,765 s
benchmark4	9.453828	43,163 s	43,862 s	43,554 s
benchmark5	10.045390	167,189 s	168,903 s	167,964 s
benchmark6	10.518968	858,143 s	862,754 s	860,903 s

FIGURE 1 – Mesures des temps minimum, maximum et moyen de 5 exécutions pour les 6 benchmarks.

3.3 Analyse des résultats expérimentaux

Après avoir tracé le temps d'exécution en fonction du nombre d'éléments du dictionnaire, on voit que les résultats expérimentaux sont en accord avec nos résultats théoriques : le temps d'exécution forme une portion de parabole qui correspond bien à $O(n^3)$.

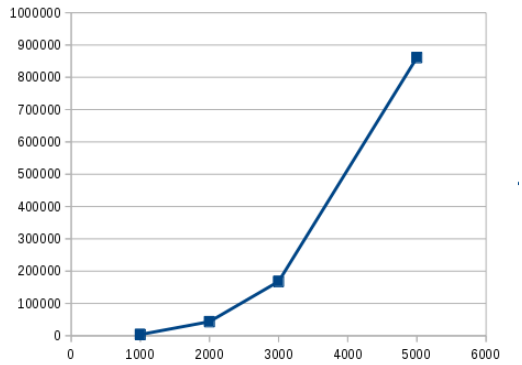

FIGURE 2 – Temps d'exécution en fonction du nombre d'éléments du dictionnaire.

4 Question bonus : programme `mystere.c`(2 points)

4.1 Que fait le programme mystère et dans quel but ? (1.5 point)

Dire brièvement ce que fait le programme mystere et quel est l'impact lors de l'exécution (i.e. lors de recherches avec le dictionnaire) de ce post-traitement.

4.2 Qu'en pensez-vous ? (0.5 point)

Répondre à l'argumentation en justifiant : soit que le programme mystere est (presque) optimal (justifier les hypothèses) ; soit qu'il n'est pas optimal en justifiant comment faire encore mieux.