Rapport de TP 4MMAOD : Génération d'ABR optimal

MILAN Guillaume (SLE) KOUVTANOVITCH Geoffrey (SLE)

14 avril 2017

Préambule

- *Compléter ce patron de rapport en supprimant toutes les phrases en italique : elles ne doivent pas apparaître dans le rapport pdf.*
- *Il sera attribué* **1 point** *pour la qualité globale du rapport : présentation, concision et clarté de l'argumentation.*

1 Principe de notre programme (1 point)

L'objectif du programme est d'obtenir l'arbre de parcour optimal sur l'ensemble des éléments. Pour cela nous avons donc chercher à obtenir la racine optimal de notre arbre. Mais par l'expression évoqué dans le rapport précédent liant un arbre optimal a ses sous arbres droite et gauche. Ainsi pour trouver la racine de optimal l'arbre contenant les éléments i à j il suffit de parcourir tout les éléments k entre i et j et en choisissant k comme racine de l'arbre. Cet arbre a donc pour profondeur moyenne d'après la formule du rapport précédent :
$P_k(i,j) = P(i,k-1) * P_{tot}(i,k-1) + P_{tot}(k,k) + P(k+1,j) * P_{tot}(k+1,j)P_{tot}(i,j)$
on en déduit que pour cela il faut connaitre les arbre optimaux de $(i,k-1)$ et $(k+1,j)$ pour tout k. Ceci implique de le calculer avant. Nous avons donc fait le choix d'organiser notre programme avec une structure de donné contenant la profondeur moyenne d'un arbre optimal, sa racine et son poids total. Cette structure est alors utilisé pour fabriquer une matrice (cf Figure :1) diagonal supérieur qui stocke les information des arbre optimaux. Ici $M(i,j)$ contient les données de l'arbre optimal entre i et j.
Pour des raison de clarté et d'homogénéité au programme, la matrice a ses indices qui commence à 0
Une fois cette matrice calculé il suffit de construire l'arbre optimal en partant de la racine $M(0,n-1)$ avec n la taille de la matrice en respecatant le principe de construction d'arbre optimal. C'est à dire que tout sous arbre est optimal. (cf Figure :2 et Figure :3).

2 Analyse du coût théorique (2 points)

Donner ici l'analyse du coût théorique de votre programme en fonction du nombre n d'éléments dans le dictionnaire. Pour chaque coût, donner la formule qui le caractérise (en justifiant brièvement pourquoi cette formule correspond à votre programme), puis l'ordre du coût en fonction de n en notation Θ de préférence, sinon O.

2.1 Nombre d'opérations en pire cas :

Le nombre d'opération dans le pire cas est $\Theta(n^3)$ avant optimisation et en $\Theta(n^2)$ après.

Justification : Pour effectuer le calcul de chaque sous arbre nous parcourons les $(i-j)$ possibilité. Or il y a $n(n+1)2$ sous arbres. Ansi le produit de l'imbrication de ces boucle dans le résulats :
$\Theta(n^3)$

Optimisation Cependant nous utilisons aussi la méthode de Knuth qui en moyenne fait passer ce coût à (n^2)

2.2 Place mémoire requise :

La place en mémoire requise est un tableau de n élément comprenant les poids du fichier. Puis un tableau de n élement représentant l'arbre sous la forme demander. Et enfin un tableau $n + n * n2$

Justification : Pour stocker toutes les valeurs indispensable des sous arbre optimaux, il faudrait une matrice de taille $n \times n$. Mais dont le triangle inférieur n'est pas utilisé. Deplus les valeurs de la dignonale de cette matrice sont facilement retrouvable. Nous avons donc décidé de déplacer les valeur supérieur à la moitié dans le triangle inférieur. (cf figure :3)
Deplus pour éviter les défaut de cache un maximum nous faison en sorte que $M(i, k-1)$ soit proche de $M(k+1, j)$ grace à une fonction de relocalisation simple. Finalement pour que ces valeurs soit proche en mémoire nous indiçons les ligne sur j et les collone sur i.

2.3 Nombre de défauts de cache sur le modèle CO :

Justification : Grâce à ma fonction qui est censé optimiser le nombre de defaut de cache, je ne vois pas comment calculer. En effet, si on suppose que notre cache à une taille d, et que l'on regarde les recherche de valeur dans notre marice de valeur, on obtient que la probabilité que $k < n2 < j$ n'est pas négligeable. Mais je ne sais pas comment la claculer.

3 Compte rendu d'expérimentation (2 points)

3.1 Conditions expérimentaless

Décrire les conditions permettant la reproductibilité des mesures : on demande la description de la machine et la méthode utilisée pour mesurer le temps.

3.1.1 Description synthétique de la machine :

Tous les tests ont été effectués sans qu'il y ait d'éventuels programmes parasites qui tournent sur la machine. Grace à la commande *cat /proc/cpuinfo*, on trouve les données de la machine. Celle-ci est dotée de l'OS Ubuntu. Son processeur est un Intel Pentium qui tourne a 2.20 GHz, avec 2 Mo de cache.

3.1.2 Méthode utilisée pour les mesures de temps :

L'ensemble des mesures effectuées sur le temps d'exécution a été faite à l'aide de la commande time d'UNIX. Les 5 exécutions ont été faite consécutivement et avec aucun programme lancé simultanément (on limite de ce fait le risque d'un potentiel ralentissement du temps de calcul).

3.2 Mesures expérimentales

	temps min(ms)	temps max(ms)	temps moyen (ms)
benchmark1	< 1	< 1	< 1
benchmark2	< 1	< 1	< 1
benchmark3	56	60	57.6
benchmark4	228	268	238.4
benchmark5	544	556	550.4
benchmark6	1908	1988	1945.6

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

3.3 Analyse des résultats expérimentaux

Les résultats expérimentaux sont en accord avec l'analyse théorique. En effet on obtient bien une croissance du temps en $\Theta(n^2)$. Et pour les défauts de cache expérimentaux on obtient environ 1000 defauts de cache sur 5000.