Rapport : Coût minimal d'un arbre recherche binaire

M. Migne*A. El Ahmar†

20 octobre 2023

*Matthieu.Migne@grenoble-inp.org>
†Anas.ElAhmar@grenoble-inp.org

Table des matières

1 Contexte du problème

Un arbre de recherche binaire optimal est un arbre de recherche binaire qui minimise le coût de recherche attendu. Dans un arbre de recherche binaire, le coût de la recherche est le nombre de comparaisons nécessaires pour rechercher une clé donnée.

Dans ce type d'arbre, chaque nœud se voit attribuer un poids qui représente la probabilité de la clé recherchée. Le coût de recherche attendu d'un nœud est la somme du produit de sa profondeur et de son poids, et du coût de recherche attendu de ses enfants selon la formule suivante :

$$W_{i,j} = \sum_{k=i}^{j} p_k + \min_{r=i}^{j}(W_{i,r-1} + W_{r+1,j})$$

L'idée de la formule ci-dessus est simple, nous essayons un par un tous les nœuds comme racine (r varie de i à j dans le deuxième terme). Lorsque le rième nœud devient la racine, nous calculons récursivement le coût optimal de i à r-1 et de r+1 à j. Nous ajoutons la somme des fréquences de i à j (voir le premier terme de la formule ci-dessus). Le résultat final correspond à $W_{0,n-1}$.

Alors comment on peut implémenter cela pour calculer le coût optimal de recherche dans un arbre binaire ? on donne en entrée une liste des clés avec leurs fréquences respectives et on veut en sortie le coût de recherche minimal. On cherche aussi à savoir comment reconstruire l'arbre optimal une fois on a calculé le coût optimal.

2 Présentation de l'alogrithme force brute

Dans cet algorithme on utilise une approche de force brute pour explorer toutes les possibilités d'arborescence, en divisant le problème en sous-problèmes jusqu'à ce qu'il atteigne les cas de base (donc récursivité + diviser le problème).

L'algorithme commence par définir un coût initial comme étant l'infini, puis explore chaque point de partition possible. Pour chaque partition, il calcule récursivement le coût de la partie gauche de la partition et le coût de la partie droite de la partition, puis additionne ces coûts avec la somme des fréquences de la partition actuelle. Le coût total de chaque partition est comparé au minimum actuel, et si le coût de la partition est plus faible, il devient le nouveau coût minimum. L'algorithme répète ce processus pour toutes les partitions possibles.

L'algorithme explore toutes les possibilités de partition. Dans ce cas, le nombre total d'appels récursifs pour résoudre le problème est exponentiel. Cette approche n'est pas optimale pour des grandes valeurs de n en raison de sa complexité exponentielle, car elle nécessiterait un temps de calcul excessif.

3 Présentation de l'alogrithme avec programmation dynamique

Pour réduire la complexité temporelle, on doit envisager d'autres approches comme la programmation dynamique qui permet de passer à une complexité de $O(n^3)$ (Pour notre première implémention qu'on n'aborde pas ici). Cependant, cette complexité est toujours grande pour nous, nous avons choisi d'implémenter l'algorithme de programmation dynamique de Knuth qui permet d'avoir un meilleur temps d'exécution.

Notre problème satisfait les conditions d'optimisation de Knuth et donc permet de gagner en complexité puisque pour calculer $W_{i,j}$, on a plus besoin de tester les valeurs de k entre i et j et on teste seulement entre $W_{i,j-1}$ et $W_{i+1,j}$.

Grâce à cette transformation, on voit notre complexité passer de $O(n^3)$ (implémention dynamique naïve) à $O(n^2)$.

4 Reconstruction de l'arbre optimal

Une fois on a trouvé le coût minimal, on aura les 2 matrices suivantes remplies :

Matrice **cost** : $cost_{i,j}$ stocke le coût minimum de construction d'un arbre binaire de recherche optimal contenant les clés allant de keys[i] à keys[j]. Chaque élément $cost_{i,j}$ représente le coût d'un arbre contenant un sous-ensemble des clés de l'ensemble d'origine. Le coût d'un arbre est calculé en tenant compte des fréquences des clés. Initialement, les valeurs sur la diagonale principale de la matrice **cost** sont définies en fonction des fréquences individuelles des clés.

Matrice **opt** : $opt_{i,j}$ stocke l'indice de la clé choisie comme racine pour l'arbre binaire de recherche optimal contenant les clés allant de keys[i] à keys[j]. Chaque élément $opt_{i,j}$ est l'indice de la racine optimale pour l'arbre dans la plage spécifiée par i et j. Cette matrice est essentielle pour la construction de l'arbre en suivant l'algorithme de Knuth.

Nous pouvons utiliser la matrice **opt** pour construire l'arbre binaire de recherche optimal. La construction de l'arbre se fait de manière récursive en utilisant une fonction qui prend en compte les indices i et j de la plage actuelle et utilise les valeurs de **opt** pour choisir les racines optimales à chaque niveau de l'arbre.

5 Sitographie

KNUTH OPTIMIZATION : `https://cp-algorithms.com/dynamic_programming/knuth-optimization.html`