Arbres Binaires de Recherche Optimaux

Julien Devevey

2018-2019

Ref : Cormen - Introduction to Algorithms, p. 397

Remarque 1. En utilisant des arbres binaires particuliers, comme les arbres rouge-noir, il est possible d'effectuer des recherches dans un arbre en temps au plus $O(n\log(n))$. Mais dans certains cas, par exemple la traduction d'un texte de l'anglais vers le français, on connaît la probabilité qu'un mot soit recherché, et dans ce cas, on sait que si l'arbre est mal construit, ce pire cas se réalisera souvent. Les arbres de recherche optimaux permettent de construire des arbres dont le temps de recherche pour chaque clé est optimal, étant donné la probabilité qu'on recherche une clé. On va alors voir comment construire un tel arbre.

Théorème 2. Etant donné $k_1, \dots, k_n$ des clés, $p_1, \dots, p_n$ les probabilités qu'on recherche une clé donnée, et $q_0, \dots, q_n$, les probabilités qu'on recherche un élément x tel que $k_i \leq x \leq k_{i+1}$ (on a alors $\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$), il existe un arbre binaire de recherche T qui minimise la quantité

$$\mathbb{E}(\text{coût d'une recherche dans } T) = \sum_{i=1}^{n} \text{prof}_T(k_i)p_i + \sum_{i=0}^{n} \text{prof}_T(d_i)q_i$$

On étiquettera les feuilles d'un tel arbre avec $d_0, \dots, d_n$ qui correspondront aux cas où la recherche ne donne rien.

Démonstration.

Commençons par une simple observation : si on a un tel arbre, alors pour tout sous-arbre T', T' contient un sous-ensemble de clés contingentes : $k_i, \dots, k_j$, car on a un arbre binaire de recherche. Mais de plus les recherches dans T' doivent être optimales : si on avait un meilleur arbre T'', on pourrait remplacer tout T' par T'' dans T et obtenir un arbre binaire de recherche plus performant : alors T' est un arbre binaire de recherche optimal pour le sous-problème $k_i, \dots, k_j$, de probabilités associées $p_i, \dots, p_j$ et $\sum_{k<i}(q_k + p_k) + q_i, q_{i+1}, \dots, q_{j-1}, q_j + \sum_{j>k}(q_k + p_k)$. On voit alors qu'on peut résoudre le problème de manière récursive, ce qu'on va faire.

L'objectif est donc de construire $e[i, j]$ qui est le coût moyen d'une recherche du sous-arbre contenant uniquement les clés $k_i, \dots, k_j$. Au final, le but est de calculer $e[1, n]$. De plus, on pose par convention que $e[i, i-1] = q_{i-1}$ car par

convention un arbre optimal est juste la feuille d_{i-1}. Quand $j \geq i$, on sélectionne une racine k_r et on prend deux sous-arbres optimaux pour $k_i, \ldots, k_{r-1}$ et $k_{r+1}, \ldots, k_j$ qu'on utilise comme fils gauche et droit respectivement pour construire un nouvel arbre binaire. Alors une recherche dans cet arbre coûte 1 de plus (sauf si on cherche k_r, où le coût est 1). Etant donné qu'un arbre binaire de recherche optimal est de cette forme d'après le premier paragraphe, il reste à calculer le min des coûts de recherche suivant la racine r choise. On pose $w(i,j) = \sum_{i=i}^{j}(p_i + q_i) + q_{i-1}$:

$$e[i,j] = \min_{i \leq r \leq j} p_r + (e[i,r-1] + w(i,r-1) + (e[r+1,j] + w(r+1,j))$$
$$e[i,j] = \min_{i \leq r \leq j} e[i,r-1] + e[r+1,j] + w(i,j)$$

Si en plus lors de chaque calcul on retient quel racine a été choisie, on construit ainsi à la fin un arbre binaire de recherche : en effet, si la racine est r, alors ses fils gauche et droit sont respectivement les racines des sous-arbres $1, \ldots, r-1$ et $r+1, \ldots, n$, etc. De plus, le coût moyen d'une recherche est $e[1,n]$ et on sait qu'un tel arbre est optimal, car à chaque étape, on construit un arbre dont le temps de recherche est optimal. □

Algorithme 3. L'algorithme suivant permet de construire un arbre binaire optimal en $O(n^3)$.

Algorithm 1 Calcul dynamique d'un ABR optimal

Entrée: p, q, n les probabilités et la taille de l'arbre
Sortie: un arbre binaire optimal

1: $e \leftarrow$Tableau vide indexé par $[\![1, n+1]\!] \times [\![0, n]\!]$
2: $w \leftarrow$Tableau vide indexé par $[\![1, n+1]\!] \times [\![0, n]\!]$
3: $root \leftarrow$Tableau vide indexé par $[\![1, n]\!] \times [\![1, n]\!]$
4: **pour** $i = 1$ à $n + 1$ **faire**
5: $\quad e[i, i-1] \leftarrow q[i-1]$
6: $\quad w[i, i-1] \leftarrow q[i-1]$
7: **fin pour**
8: **pour** $l = 1$ à n **faire**
9: $\quad$ **pour** $i = 1$ à $n - l + 1$ **faire**
10: $\quad\quad j \leftarrow i + l - 1$
11: $\quad\quad e[i, j] \leftarrow \inf$
12: $\quad\quad w[i, j] \leftarrow w[i, j-1] + p[j] + q[j]$
13: $\quad\quad$ **pour** $r = i$ à j **faire**
14: $\quad\quad\quad t \leftarrow e[i, r-1] + e[r+1, j] + w[i, j]$
15: $\quad\quad\quad$ **si** $t < e[i, j]$ **alors**
16: $\quad\quad\quad\quad e[i, j] \leftarrow r$
17: $\quad\quad\quad\quad root[i, j] \leftarrow r$
18: $\quad\quad\quad$ **fin si**
19: $\quad\quad$ **fin pour**
20: $\quad$ **fin pour**
21: **fin pour**
22: **retourner** e et $root$