Arbres binaires de recherche optimaux

Référence : Cormen, p.357 de la seconde édition

2011-2012

On s'intéresse ici à optimiser les arbres binaires de recherche quand on connait la probabilité pour chaque clef d'être recherchée.

On se donne n clefs distinctes triées dans l'ordre croissant $k_1 < \cdots < k_n$, où pour tout i, k_i a une probabilité p_i d'être recherchée.

On rajoute aussi $n+1$ clefs factices $d_0, \ldots, d_n$ représentant les recherches qui ne sont aucun des k_i.

d_0 représente toutes les valeurs inférieures à k_1, d_n celle supérieures à k_n, et d_i celles comprises entre k_i et k_{i+1}. On se donne de plus une probabilité q_i que la recherche soit dans d_i.

Les nœuds internes de l'arbre seront les k_i, et les feuilles les d_i.

On a donc

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1.$$

Le coût moyen d'une recherche dans notre arbre sera

$$1 + \sum_{i=1}^{n} \mathrm{Prof}(k_i) p_i + \sum_{i=0}^{n} \mathrm{Prof}(d_i) q_i.$$

Un arbre binaire de recherche sera dit *optimal* si ce coût moyen est minimal. On abrège en "ABRO".

La recherche d'un ABRO en listant tous les arbres possibles a une complexité bien trop grande, on va donc essayer un algorithme de programmation dynamique.

On remarque pour cela que tout sous-arbre de notre arbre *doit* être optimal : sinon, on le remplace par un optimal, faisant ainsi baisser le coût total.

Le sous-problème à résoudre est donc, étant donnés $i \geqslant 1$ et $i - 1 \leqslant j \leqslant n$ un ABRO contenant les clefs $k_i, \ldots, k_j$.

Soit $e[i, j]$ le coût d'un arbre contenant $k_i, \ldots, k_j$.

Si $j = i - 1$, alors il n'y a que la clef factice d_{i-1} dans l'arbre : le coût moyen est q_{i-1}.

Sinon, $j \geqslant i$. On choisit un nœud k_r, $i \leqslant r \leqslant j$, et on construit l'arbre de racine k_r. Le sous-arbre gauche de k_r contiendra les clefs $k_i, \ldots, k_{r-1}$, et le sous arbre droit les clefs $k_{r+1}, \ldots, k_j$.

Quand on fait "descendre" un sous-arbre, on augmente la profondeur de ce sous-arbre de 1, on donc on augmente le coût de

$$w(i, j) = \sum_{k=i}^{j} p_k + \sum_{k=i-1}^{j} q_k.$$

On obtient donc la formule :

$$e[i,j] = p_r + e[i,r-1] + w(i,r-1) + e[r+1,j] + w(r+1,j).$$

On remarque que

$$w(i,j) = w(i,r-1) + p_r + w(r+1,j),$$

et donc

$$e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j).$$

Cette formule est valide si on a choisi la bonne racine pour notre sous-arbre. On optimise donc :

$$e[i,j] = \begin{cases} q_{i-1} & \text{si } j = i-1 \\ \min_{i \leqslant r \leqslant j} e[i,r-1] + e[r+1,j] + w(i,j) & \text{si } i \leqslant j \end{cases}$$

Pour garder une trace de notre construction, on définit $racine[i,j]$ comme l'indice de la racine optimale pour le sous-arbre contenant $k_i, \ldots, k_j$.

On va maintenant utiliser la programmation dynamique pour calculer $e[1,n]$.

Remarque – Plutôt que de calculer tous les $w(i,j)$ à chaque fois, on crée un tableau défini par

$$\begin{cases} w(i,i-1) = q_{i-1} \\ w(i,j) = w(i,j-1) + p_j + q_j \end{cases}$$

On a donc l'algorithme suivant :

Algorithm 1 ABRO

Préconditions: p, q probabilités, n nombre de clefs

```
pour i = 1 à n + 1 faire
        e[i, i − 1] ← q_{i−1}
        w[i, i − 1] ← q_{i−1}
ruop
pour ℓ = 1 à n faire
        pour i = 1 à n − ℓ + 1 faire
                j ← i + ℓ − 1
                e[i, j] ← ∞
                w[i, j] ← w[i, j − 1] + p_j + q_j
                pour r = i à j faire
                        t ← e[i, r − 1] + e[r + 1, j] + w[i, j]
                        si t < e[i, j] alors
                                e[i, j] ← t
                                racine[i, j] ← r
                        is
                ruop
        ruop
ruop

retourner e et racine
```

Les trois boucles imbriquées nous donnent une complexité en $\mathcal{O}(n^3)$.