Algorithmique et Programmation

Projet : Plus proche ancêtre commun *

Ecole normale supérieure
Département d'informatique
td-algo@di.ens.fr

2014-2015

Le problème du Plus Proche Ancêtre Commun (PPAC) est le suivant : Soit T un arbre. Etant donnés deux sommets u et v de T, il s'agit de trouver leur ancêtre commun qui est le plus éloigné possible de la racine de T. Il existe un algorithme quasi-trivial qui résout la question en temps $\mathcal{O}(h)$ où h désigne la hauteur de l'arbre. Le but [2] du projet est de programmer un algorithme résolvant le problème en temps $\mathcal{O}(1)$, après avoir au préalable fait un précalcul *linéaire* en la taille de l'arbre.

1 Réduction à Minimum d'Intervalle

Le problème Plus Proche Ancêtre Commun est lié à un autre problème, Minimum d'Intervalle. Soit A un tableau contenant n entiers. Etant donnés deux entiers i et j, il s'agit de trouver un entier $\mathrm{MI}_A(i,j)$ entre i et j tel que $A[\mathrm{MI}_A(i,j)] = \min_{i \leq k \leq j} A[k]$. En effet, PPAC. se réduit à MI de la façon suivante.

Précalcul. On double chaque arête de T, puis on effectue un parcours Eulérien du graphe résultant en partant de la racine. On stocke le i-ème noeud rencontré dans un tableau $A[i]$, et on stocke sa hauteur dans un autre tableau $B[i]$. Comme chaque arête est traversée une fois en descendant et une fois en montant, A et B ont tous deux taille $2n-1$. On note que deux entrées contigües de B diffèrent exactement d'une unité. Dans un troisième tableau C, on stocke la première occurrence de i dans A : $C[i] = \min\{k : A[k] = i\}$.

Requête $\{u, v\}$. Étant donné une paire de sommets u et v, il est clair (et on admettra) que leur plus petit ancêtre commun est visité entre u et v dans la tournée Eulérienne. Il s'ensuit que si c'est le ℓ-ième noeud visité, alors $C[u] \leq \ell \leq C[v]$, et clairement $B[\ell]$ est minimal dans l'intervalle. On a donc :

$$\mathrm{PPAC}_T(u,v) = A\Big[\mathrm{MI}_B\big(C[u], C[v]\big)\Big]$$

Pour avoir les complexités souhaitées, il suffit donc de savoir générer les trois tableaux A, B et C en temps linéaire en la taille de l'arbre (ce qui est laissé à la sagacité de l'auteur du projet), et de savoir, après un précalcul linéaire en la taille du tableau, répomdre aux requêtes de MI en temps $\mathcal{O}(1)$.

2 Minimum d'Intervalle

2.1 Avec précalcul $\mathcal{O}(n \log n)$

Dans cette section on veut résoudre le problème MI sur un tableau A de n éléments. Tout d'abord, voyons comment on peut répondre aux requêtes de Minimum d'Intervalle en temps $\mathcal{O}(1)$, après un précalcul en $\mathcal{O}(n \log n)$.

Précalcul. L'idée est de précalculer, dans un nouveau tableau T, les valeurs de $T[i,k] = \mathrm{MI}_A\left(i, i + 2^k - 1\right)$ pour tout $i, 1 \leq i \leq n$, et tout k tel que $1 \leq i + 2^k - 1 \leq n$. Ceci peut se faire en temps $\mathcal{O}(n \log n)$ par programmation dynamique.

*Conception : Charles Bouillaguet. Révisé par Claire Mathieu 9/2013.

Requête $\{i, j\}$. Une fois qu'on a ce tableau T, on peut s'en servir pour calculer MI_A en temps constant. Notons 2^k la plus grande puissance de deux telle que $i + 2^k - 1 \leq j$. On observe que l'intervalle $[i; j]$ est l'union (pas forcément disjointe) des deux intervalles $[i; i + 2^k - 1]$ et $[j - 2^k + 1; j]$. Il s'ensuit que :

$$\mathrm{MI}_A(i,j) = \begin{cases} T\left[i, i + 2^k - 1\right] & \text{si } A\big[T\left[i, i + 2^k - 1\right]\big] < A\big[T\left[j - 2^k + 1, j\right]\big] \\ T\left[j - 2^k + 1, j\right] & \text{sinon} \end{cases}$$

2.2 Avec précalcul $\mathcal{O}(n)$

Dans cette section on veut améliorer l'algorithme précédent pour résoudre, avec moins de précalcul, le problème MI sur un tableau A de n éléments dont deux entrées contigües diffèrent d'exactement une unité.

On utilise la "méthode des quatre russes" [3] (il s'avère en fait qu'un seul de ces auteurs est russe, mais le nom est resté). L'idée générale est la suivante [1] :

1. Précalcul : on découpe le tableaux A en $2n/\log n$ blocs de taille $\frac{\log n}{2}$. On définit alors un tableau A' de taille $n' = 2n/\log n$ tel que $A'[i]$ contient le mimium du i-ème bloc de A. De la même manière, on définit $B'[i]$ comme étant l'indice dans le i-ème bloc où le minimum $A'[i]$ est atteint.
2. Précalcul : on fait le précalcul de l'algorithme précédent, mais appliqué au tableau A'.
3. Précalcul : on stocke toutes les valeurs possibles de $\mathrm{MI}_A(i, j)$ quand i et j sont dans le même bloc (cf. ci-dessous).
4. Requête $\{i, j\}$: on observe que si i et j n'appartiennent pas au même bloc, alors $\mathrm{MI}_A(i, j)$ est l'indice du minimum des trois valeurs suivantes :
   (a) Le minimum de l'intervalle qui s'étend de i à la fin du bloc le contenant
   (b) Le minimum de l'intervalle formé de tous les blocs compris entre celui de i et celui de j
   (c) Le minimum de l'intervalle qui s'étend du début du bloc contenant j à j lui-même.

   La deuxième valeur est fournie par l'algorithme précédent appliqué au tableau A', tandis que la première et la troisième sont déjà toutes prêtes parce qu'elles ont été précalculées.

Il reste à déterminer comment précalculer toutes les valeurs possibles de MI_A à l'intérieur d'un même bloc en temps $\mathcal{O}(n)$. Pour cela, on observe que si on prend un bloc quelconque, et qu'on ajoute une constante à chacun de ses éléments, on va modifier la valeur absolue du minimum du bloc, mais pas sa position dans le bloc. On peut donc se ramener à des blocs c'est-à-dire dont le premier élément vaut zéro. Comme chaque entrée ne diffère de la précédente que d'une unité, il n'y a que $2^{(\log n)/2} = \mathcal{O}(\sqrt{n})$ blocs normalisés différents de taille $(\log n)/2$. On peut donc, pour chaque bloc normalisé possible, précalculer assez naïvement l'ensemble des réponses à l'intérieur du bloc. Le prétraitement de l'ensemble des blocs normalisés est possible en temps et en espace $\mathcal{O}\left(\sqrt{n}\log^2 n\right)$.

Un dernier détail : déterminer la forme normalisée d'un bloc demande a priori un temps $\mathcal{O}(\log n)$, donc ne peut pas être fait "en-ligne". Il faut donc précalculer à l'avance pour chaque bloc quelle est sa forme normalisée.

3 Travail demandé

Programmez l'algorithme pour MI avec précalcul en $\mathcal{O}(n \log n)$, puis avec précalcul en $\mathcal{O}(n)$ décrit section 2.2, puis programmer l'algorithme pour PPAC. Tester sur des petites entrées et sur des entrées aléatoires. Vous devez définir une structure de donnée pour stocker l'arbre, puis créer deux fonctions : PRECALCUL et PPAC. Votre programme doit prendre en argument le nom d'un fichier qui décrira un arbre. Le format de ce fichier est simple : `()` désigne un noeud seul, `((((()))))` désigne un peigne de longueur 4, `(()()()()())` désigne une racine avec 5 fils, et `((()())(()()))` désigne l'arbre binaire complet à 7 noeuds. On va supposer que les noeuds sont numérotés à partir de zéro dans l'ordre du parcours préfixe. Votre programme devra effectuer le précalcul puis afficher "READY" sur la sortie d'erreur. A partir de ce moment-là, votre programme devra lire sur l'entrée standard une paire de deux entiers, et écrire sur la sortie standard le numéro du plus petit ancêtre commun aux deux sommets en question.

Par exemple, si l'arbre est `(((()())(((())(()()))))`, alors la requête `7 9` doit provoquer la réponse `5`. La procédure doit s'arrêter lorsqu'une ligne vide est entrée.

Références

[1] Michael A. Bender, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Finding least common ancestors in directed acyclic graphs. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 845–854, 2001.

[2] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2) :338–355, 1984.

[3] M Kronrod, V Arlazarov, E Dinic, and I Faradzev. On economic construction of the transitive closure of a direct graph. In *Sov. Math (Doklady)*, volume 11, pages 1209–1210, 1970.