Corrigé du TD 1

Guillaume Hanrot

26 janvier 2004

1. Il suffit de considérer la suite d'objets suivants : (poids, intérêt)= $(2,3),(1,2),(1,2)$, et un sac de capacité 2,3,3 pour les trois question respectivement.

2. Dans tout l'exercice, on maintient en parallèle une représentation de l'ensemble E sous forme d'un tableau de pointeurs vers la structure de partition ; cela permet d'accéder en temps $O(1)$ au sous-ensemble de la partition contenant un élément x ; toute la difficulté de l'opération de recherche consiste alors à retrouver dans cette structure le représentant canonique.

i. On représente chaque sous-ensemble par une liste chaînée, en gardant un pointeur sur les deux extrémités de la liste ; le représentant d'un ensemble est l'élément de queue de la liste chaînée ; la recherche se fait alors en suivant la bonne liste chaînée, soit un coût $O(|E|)$; l'union consiste juste à raccorder une liste chaînée à une autre, ce qui coûte $O(1)$.

Par exemple, si l'une des deux listes est représentée par (p_1, p_2) et l'autre par (p'_1, p'_2), l'union consiste à faire pointer p_2 vers p'_1 et à renvoyer (p_1, p'_2).

ii. On représente l'ensemble par un tableau T indexé par les éléments de E, $T[x]$ donnant le représentant du sous-ensemble auquel appartient x. La recherche se fait en temps $O(1)$, pour l'union, si $E_k \leftarrow E_l \cup E_j$, où x est le représentant de E_l et y le représenant de E_j, il faut parcourir T et remplacer par exemple toutes les occurrences de x par y.

iii. L'union se fait en choisissant l'un des deux arbres A_1 pour être sous-arbre de l'arbre A_2, et consiste uniquement à attribuer comme père à la racine de A_1 la racine de A_2, et coûte donc $O(1)$.

La recherche consiste à suivre le pointeur permettant d'accéder à l'élément x, et à suivre les pointeurs remontants jusqu'à la racine (que l'on détecte en remarquant que la racine pointe sur elle-même). On renvoie alors l'étiquette de la racine. Son coût est en O(hauteur de l'arbre).

Montrons par récurrence que la taille d'un arbre de hauteur h est au moins 2^h. Initialement, les arbres sont tous des singletons, et c'est évident. Si l'on suppose l'assertion vraie à une certaine étape du calcul où l'on réalise l'union de E_i et de E_j, correspondant à des arbres de hauteurs h_i et $h_j \geq h_i$, on crée alors un arbre de taille $|E_i| + |E_j|$. Deux cas :

- Si $h_i = h_j = h$ alors l'arbre résultant a hauteur $h+1$ et taille $|E_i| + |E_j| \geq 2^{h_i} + 2^{h_j} = 2^h + 2^h = 2^{h+1}$ et l'assertion est encore vraie.
- Si $h_i > h_j$, alors l'arbre résultant a hauteur h_i et taille $> |E_i| \geq 2^{h_i}$ et l'assertion est encore vraie.

L'assertion est donc prouvée par récurrence. En particulier, le rang de tous les arbres construits est toujours dominé par $\log_2(n)$, et le coût de la recherche est dans le cas le pire égal au rang de l'arbre concerné, donc $O(\log n)$.

iv. On en déduit une implantation de l'algorithme de Kruskal en temps $O(m \log m + m \log n)$.

3. Triangle de Pascal

On constate facilement que programmer récursivement le calcul de C_n^k conduit à évaluer les coefficients binomiaux intermédiaires un grand nombre de fois (jusqu'à C_1^1 qui est évalué C_n^k fois). Il est beaucoup plus efficace d'utiliser la méthode habituelle du triangle de Pascal, en partant de C_1^1, et en calculant les C_j^i par la formule $C_j^i = C_{j-1}^{i-1} + C_{j-1}^i$ avec un tableau `C[i][j]`. Noter que l'on n'a en fait besoin que de calculer les C_j^i tels que $j - i \leq n - k$. Noter aussi qu'on peut en fait se contenter d'un espace mémoire linéaire.

Une alternative consiste, naturellement, à utiliser l'identité $C_n^k = n(n-1)\dots(n-k+1)/k!$.

4. Insertion d'espaces dans un texte.

On considère la suite de booléens définie par $T[j] = 1$ si et seulement si l'on a une décomposition valide de $x_1 \dots x_j$. Pour $j > 0$, $T[j]$ vaut alors 1 s'il existe $k \leqslant j$ tel que $T[k] = 1$ et $x_{k+1}..x_j$ est dans le dictionnaire.

On peut donc écrire

$$T[0] = 1, T[j] = \bigvee_{\ell_{\max} \leq slant k \leqslant j} T[k] \wedge \mathrm{Dict}(x_{k+1}..x_j),$$

où $\ell_{\max}$ est la longueur maximum d'un mot du dictionnaire. Noter que si la recherche d'un mot dans le dictionnaire (prédicat Dict) se fait en temps $O(1)$, la complexité de l'algorithme est alors en $O(n\ell_{\max})$.

Le pseudo-code s'écrit de la manière suivante, où l'on maintient un tableau C pour garder trace des choix de coupure :

1. $T[0] \leftarrow 1$.
2. Pour j de 1 à n faire
   - 2.1. Pour l de $j-1$ à $j - \ell_{\max}$ faire
     - 2.1.1. Si $T[l] = 1$ et $\mathrm{Dict}(x_{l+1}..x_j)$ alors $T[j] \leftarrow 1$, $C[j] \leftarrow l$.
3. Renvoyer $T[n]$.

Pour afficher la décomposition, on procède de la façon suivante :

Affiche(C, x, j)

1. Si $j > 0$
   - 1.1. Affiche$(C, x, C[j])$
   - 1.2. print(" ") ; print$(x_{C[j]+1}..x_j)$.

et on exécute Affiche(C, x, n)

5. Organisation d'un pot.

On va appliquer une méthode de programmation dynamique. À chaque sous-arbre A de l'arbre des relations hiérarchiques, on associe la somme maximale $T[A]$ pouvant être obtenue avec des éléments de A si on prend la racine, et la somme maximale $U[A]$ pouvant être obtenue avec des éléments de A si on ne prend pas la racine.

Il est alors facile de voir que si l'arbre A a pour fils les arbres A_i, la relation de récurrence suivante a lieu :

$$T[A] = n(a) + \sum_i T[A_i],$$

$$U[A] = \sum_i \max(T[A_i], U[A_i]),$$

où $n(a)$ est la note de convivialité de la racine a de A. En effet, si a est invité, il faut choisir les invités parmi les subalternes de ses propres subalternes immédiats.

On peut donc recourir à une technique de programmation dynamique qui fournira la solution optimale. Le calcul se fait de bas en haut dans l'arbre. Comme d'habitude, il faut garder trace du choix fait à une certaine étape.

Comme un choix est fait à chaque noeud de l'arbre et que le nombre de noeuds de l'arbre est exactement égal aux nombres de personnes dans l'entreprise, la complexité est linéaire.

Pour garantir que la PDG est invitée, il faut assurer qu'à l'extrême fin de l'algorithme, le second choix sera fait dans le max. Cela peut se faire simplement à partir de $T[A]$.

6. Algorithme de Viterbi

i. Il suffit de parcourir le graphe en largeur ; partant du sommet v_0, on suit toutes les arêtes d'étiquette s_1, et on construit la liste des sommets d'arrivée ; pour chacun de ces sommets, on suit alors les arêtes d'étiquette s_2, etc.

Si le graphe a s sommets, la complexité est en $O(ns^2)$.

ii. Soit $c = v_0, v_1 \dots v_n$ un chemin le plus probable partant de v_0 d'étiquette s ; alors le chemin $v_1 \dots v_n$ est un chemin le plus probable partant de v_1 et d'étiquette $s_2 \dots s_n$.

Posons alors $f(v,i)$ le chemin le plus probable issu de v et dont l'étiquette est $s[i..n]$. On a dans ce cas la relation de récurrence

$$f(v,i) = \max_{u;\text{étiq}(v,u)=s_i} p(v,u) f(u,i+1),$$

avec la condition initiale $f(v,n+1) = 1$ pour tout v. La complexité est dans ce cas $O(nS^2)$, car le calcul d'un terme de la table (qui est de taille $O(nS)$) prend un temps $O(S)$.