Programmation avancée

Répétition 8 : Résolution de problèmes

Jean-Michel Begon

Décembre 2014

Exercice 1

Soit le graphe valué $G = (V, E, W)$, où V est l'ensemble des nœuds (*vertex*) du graphe, $E \subseteq V \times V$ est l'ensemble des arêtes (*edge*) de G et $W : E \to \mathbb{R}_+$ est la fonction de poids de chaque nœud. Par exemple, les nœuds représentent des routeurs dans un réseau informatique et le poids correspond à la capacité du lien. On désire réaliser un arbre couvrant minimum (*Minimum spanning tree*), c'est à dire un arbre (graphe acyclique connexe) qui est tel que la somme des poids des branches est minimum. Proposez un algorithme *greedy* pour solutionner ce problème.

Exercice 2

Soit une expression booléenne composée des symboles `true`, `false`, `and` et `or`. Proposez un algorithme qui détermine le nombre de façons de parenthéser l'expression de sorte qu'elle soit évaluée à `true`.

Exercice 3 (adapté de CLRS, 16.2-4)

Julien participe à une course à pieds qui consiste à relier une ville A à une ville B. Etant donné la contenance de son bidon, il sait qu'il peut parcourir m mètres sans tomber à cours d'eau. Il a également pu obtenir des organisateurs du marathon la liste des endroits où il pourra remplir son bidon et les distances entre ces endroits.

Julien aimerait minimiser le nombre d'arrêts qu'il devra effectuer lors de sa course pour remplir son bidon. Donnez une méthode efficace pour déterminer quels arrêts il devrait faire. Montrez que votre stratégie donne une solution optimale et discutez sa complexité.

Exercice 4

L'ordonnancement des tâches est un problème fréquent en informatique : on dispose d'un ensemble $S = \{s_1, s_2, ..., s_n\}$ de n tâches qu'on doit ordonner. A chaque tâche s_i est associé un temps d'exécution t_i. Une fois l'ordre des tâches fixé, on peut également associer à chaque tâche un temps de complétion, c'est-à-dire, le temps qui s'est écoulé jusqu'à la fin de la tâche.

Par exemple, si on dispose des tâches s_1 et s_2 dont les temps d'exécution respectifs sont $t_1 = 3$ et $t_2 = 5$ et si on exécute les tâches dans l'ordre $< s_1, s_2 >$, alors les temps de complétion seront :

- $C_1 = t_1 = 3$
- $C_2 = C_1 + t_2 = 3 + 5 = 8$

En outre, la somme des temps de complétion sera $C_\Sigma = \sum_i C_i = C_1 + C_2 = 11$.

On souhaite disposer d'un algorithme d'ordonnancement des tâches qui minimise la somme des temps de complétion (C_Σ).

(a) Quelle serait la complexité d'un algorithme *brute-force* pour résoudre ce problème ?

(b) Est-il possible de faire mieux (le problème dispose-t-il de la propriété de sous-structure optimale ?)

(c) Le cas échéant, proposez un algorithme efficace pour résoudre ce problème.

Exercice 5

Dessinez l'arbre de récursion du `MergeSort` pour le problème suivant :

$$A =< 5, 2, 4, 7, 1, 3, 2, 6 >$$

Pourquoi la mémoïsation ne permet-elle pas d'améliorer ce problème ?

Exercice 6 (adapté de CLRS, 15-2)

Proposez un pseudo-code efficace qui permet de déterminer le plus grand palindrome qui existe au sein d'un mot. Par exemple, le mot `caractères` contient un palindrome de taille 5 (`carac`).

Exercice 7 (adapté de CLRS, 15.5)

Version longue :

Supposons que les gens de "dicool.ukfr" vous contactent pour améliorer les performances de leur dictionnaire anglais → français. Avec stupeur, vous vous apercevez que celui-ci utilise des listes-liées. Vous ressortez vite fait votre cours de "Programmation avancée" et décidez de remplacer les listes-liées par un arbre binaire de recherche (c'est déjà mieux !).

Puisque l'arbre ne sera pas modifié une fois créé, on peut l'équilibrer à la construction en choisissant soigneusement l'ordre d'insertion. Il s'avère que le sommet de l'arbre est occupé par le mot "*Smurf*", un mot qui ne doit pas être recherché souvent. Vous réalisez donc qu'il est possible de faire mieux qu'un arbre balancé à condition de connaitre la fréquence de demande des mots. Jour de chance, les gens de "dicool.ukfr" ont archivé toutes les requêtes faites au dictionnaire. Un petit script plus tard, vous disposez d'une probabilité de recherche pour chaque mot du dictionnaire. Au passage vous vous apercevez également que le dictionnaire ne contient pas tous les mots demandés (les gens rentrent vraiment n'importe quoi dans les champs de recherche...).

Toutes ces informations en main, vous commencez à réfléchir à la manière d'optimiser l'arbre de recherche. Quelques minutes d'intense cogitation plus tard, vous baissez les yeux vers le catalogue que vous feuilletiez distraitement et vous apercevez, écrit en grand, "programmation dynamique". En fait, il s'agissait de votre cours préféré, 30 centimètres à droite dudit magazine. Sous le poids des coïncidences, vous décidez de tenter le coup !

Version courte :

Formulez l'optimisation par programmation dynamique d'un arbre binaire de recherche dont la fréquence de recherche des clés est connue.