Structures de données et algorithmes

Répétition 8.2: Résolution de problèmes - Exercices supplémentaires

Jean-Michel Begon

Avril 2014

Exercice 1 (B. Boigelot, 2009)

On considère un terrain de jeu rectangulaire possédant $n \times m$ cases organisées en n lignes et m colonnes. Des sommes d'argent sont initialement placées sur ces cases. Un joueur traverse le terrain à partir du coin supérieur gauche jusqu'au coin inférieur droit. A chaque étape de ce trajet, le joueur collecte la somme d'argent placée sur la case où il se trouve, et a ensuite le choix de se déplacer d'une case soit vers la droite, soit vers le bas (les déplacements vers la gauche, vers le haut ou en diagonale sont interdits).

(a) Ecrire un algorithme capable de calculer une suite de mouvements qui permet au joueur d'atteindre la case inférieure droite en ayant accumulé le plus d'argent possible.

(b) Calculer la complexité en temps et en espace de l'algorithme proposé.

Exercice 2

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

Exercice 3

L'*arbitrage* est une opération financière qui consiste à exploiter des écarts temporaires constatés sur différents taux de change de sorte à assurer un gain positif. Par exemple, il peut il y avoir un (court) laps de temps pendant lequel 1 dollar américain s'échange pour 0.75 livres anglaises, 1 livre anglaise s'échange 2 dollars australiens et 1 dollar australien s'échange pour 0.7 dollar américain. Un trader peut donc échanger 1 dollar américain et terminer avec $0.75 \times 2 \times 0.7 = 1.05$ dollars américains, et ainsi faire un profit de 5%.

Supposons n monnaies $c_1, ..., c_n$ et une table R telle que $R[i, j]$ représente le taux de change de la monnaie c_i vers la monnaie c_j. On souhaiterait concevoir un algorithme qui permette de trouver la valeur maximale de :

$$R[c_1, c_{i_1}] \times R[c_{i_1}, c_{i_2}] \times ... \times R[c_{i_{k-1}}, c_{i_k}] \times R[c_{i_k}, c_1]$$

Exercice 4

Soit un tableau $N \times N$ de booléens (`0` ou `1`). Proposer un algorithme pour trouver le plus grand sous-tableau contigu contenant uniquement des valeurs `1`.

Exemple : Le tableau suivant contient un sous-tableau 4×4 contigu ne contenant que des `1`.

```
1 0 1 1 1 0 0 0
0 0 0 1 0 1 0 0
0 0 1 1 1 0 0 0
0 0 1 1 1 0 1 0
0 0 1 1 1 1 1 1
0 1 0 1 1 1 1 0
0 1 0 1 1 1 1 0
0 0 0 1 1 1 1 0
```

Exercice 5

Soit un tableau $N \times N$ de booléens (`0` ou `1`). Proposer un algorithme pour trouver le plus grand sous-tableau contigu contenant autant de valeurs `0` que de valeurs `1`.

Exercice 6 (P.-A. de Marneffe, 2006)

Soit un tableau $B[0..M-1, 0..N-1]$ de M lignes et N colonnes, avec $0 < M$ et $0 < N$. Les élémetns du tableau sont des booléens.

$B[i, j]$ est à l'origine d'une λ-figure(p), avec $p \geq 1$ si les conditions suivantse sont vérifiées, $\forall k : 0 \leq k < p$, :

- $B[i-k, j-k]$ et $B[i-k, j]$ existent dans le tableau B ;
- Si k est pair, alors $B[i-k, j-k]$ et $B[i-k, j]$ sont `true` ;
- Si k est impair, alors $B[i-k, j-k]$ et $B[i-k, j]$ sont `false`.

Note : si $k = 0$, alors les deux éléments $B[i-k, j-k]$ et $B[i-k, j]$ sont confondus en $B[i, j]$.

Appelons $pmax(i, j)$ la valeur maximale de p telle que $B[i, j]$ est origine d'une λ-figure(p). Si $B[i, j]$ est `false`, alors $pmax(i, j) = 0$ par convention.

$B[i, j]$ est origine d'une λ2-figure(s) si les trois conditions suivantes sont vérifiées :

- $B[i, j]$ est origine d'une λ-figure(s) avec $s = pmax(i, j)$ et $s \geq 3$;
- $B[i-2, j-1]$ existe dans le tableau et est origine d'une λ-figure(q) avec $q = pmax(i-2, j-1)$ et $q \geq 1$;
- $q \leq s - 2$.

On demande de concevoir un algorithme qui détermine dans une variable `smax` la valeur maximale de s pour les λ2-figure(s) qui existent dans B.

Exercice 7 : Arbre de recherche optimal (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.