Structures de données et algorithmes

Répétition 7 : Résolution de problèmes

Jean-Michel Begon

Avril 2015

Exercice 1 : Sudoku

Proposez une solution *brute-force* pour résoudre un Sudoku.

Exercice 2 : Les huit dames

On désire placer huit dames sur un échiquier de 8x8 cases de manière à ce qu'aucune dame ne puisse en attaquer une autre. Autrement dit, il n'y a jamais deux dames sur une même ligne, colonne ou diagonale.

Proposez une solution *brute-force* pour ce problème.

Exercice 3 : SlowSort

Soit l'algorithme de tri suivant : on génère toutes les permutations possibles du tableau à trier. On teste chacune afin de voir si elle correspond au tableau trié. On renvoie ce dernier.

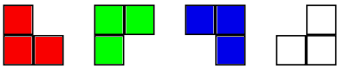
(a) A quelle catégorie appartient l'algorithme proposé ?

(b) Donnez le pseudo-code de cet algorithme.

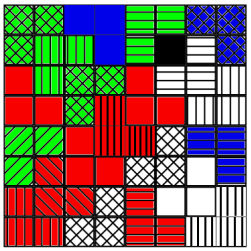
(c) Quelle est la complexité de cet algorithme ?

Exercice 4 : Carré de triminos

On souhaite recouvrir un carré de NxN ($N = 2^m, m > 0$) à l'aide des quatre triminos, à l'exception d'une case vide dont les coordonnées sont données.



Les triminos de base



Carré 8x8 recouvert

Proposez un algorithme pour résoudre de problème.

Exercice 5 : Les 12 pièces

Soit le problème suivant :

> *"Vous disposez de 12 pièces dont une est fausse. Celle-ci possède un poids moindre. Comment déterminer la fausse pièce le plus rapidement possible à l'aide d'une balance à plateaux ?"*

Dans le cadre général où on dispose de N pièces :

(a) Proposez une approche par force brute pour résoudre ce problème.

(b) Proposez une approche *divide-and-conquer* pour résoudre ce problème. Peut-on donner une borne sur le nombre de pesées ?

Exercice 6

On désire passer d'un tableau à une liste tout en renversant ses éléments ($[a_1, a_2, ...a_{n-1}, a_n] \rightarrow [a_n, a_{n-1}, ...a_2, a_1]$).

(a) Proposez une approche directe pour ce problème. Quelle est sa complexité ?

(b) Proposez une approche *divide-and-conquer* pour ce problème. Quelle est sa complexité ?

(c) Qu'adviendrait-il de la complexité de la solution *divide-and-conquer* si on voulait garder un tableau en sortie ?

Exercice 6 3/4

On souhaite déterminer la sous-séquence non vide contigüe de somme maximale d'un tableau de nombres A.

(a) Proposez une approche *brute-force* pour résoudre ce problème.

(b) Proposez un algorithme *divide-and-conquer* pour ce problème.

Exercice 7

Soit un ensemble $P = \{p_1, p_2, ..., p_n\}$ de points répartis dans un plan et tels que $p_i = (x_i, y_i)$. Proposez un algorithme qui détermine la distance (euclidienne) entre les deux points les plus proches.

Exercice 8

Dans une répétition précédente, nous avions solutionné l'exercice :

> *Soit un ensemble de N valeurs entières distinctes. Ecrivez une fonction calculant le nombre d'arbres binaires de recherche distincts qu'il est possible de construire à partir de ces N valeurs ($N \geq 1$).*

par le pseudo-code suivant :

```
NbTree(B)
1  if N ≤ 1
2        return 1
3  Nb = 0
4  for i = 1 to N
5        Nb = Nb + NbTree(i − 1) ∗ NbTree(N − i)
6  return Nb
```

dont la complexité est importante à cause des appels à des sous-problèmes déjà résolus.

(a) Dessinez le graphe des sous-problèmes pour une taille de $N = 4$.

(b) Utilisez la mémoïsation pour faire baisser la complexité de l'algorithme (donnez le pseudo-code d'une version ascendante et d'une version descendante).

Bonus

On considère une application biomédicale dont le but est de segmenter les cellules (c'est-à-dire de retrouver le polygone qui correspond à chaque cellule) sur des images de grandes dimensions (de l'ordre du gigapixel). Malheureusement, on est contraint de découper les images en tuiles rectangulaires (de mêmes dimensions) afin de permettre des temps de traitement raisonnables. [1]

La segmentation traite donc chacune des tuiles séparément et identifie dans chacune d'elles les polygones des cellules.

Pour garantir un bon fonctionnement de l'application, il faut cependant éviter qu'une cellule à cheval sur plusieurs tuiles soit considérée comme plusieurs cellules. Pour ce faire, on dispose d'un prédicat `touch` qui permet de vérifier si deux polygones se touchent ainsi qu'une fonction `merge` qui permet de fusionner ceux-ci.

Une approche *brute-force* en $\Theta(n^2)$ pour ce problème serait de passer chaque polygone avec tous les autres à la fonction `touch` et de fusionner ceux qui se touchent. Proposez une solution plus efficace.

1. Le même procédé est utilisé dans Google Maps, par exemple.