

Programmation avancée

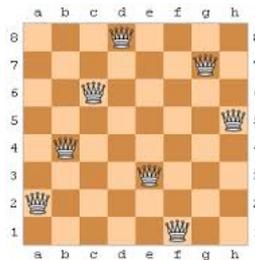
Répétition 7 : Résolution de problèmes

Jean-Michel BEGON

Décembre 2014

Exercice 1

Dans le problème des 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 2

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 quelle catégorie appartient l'algorithme proposé ?
- Donnez le pseudo-code de cet algorithme.
- Quelle est la complexité de cet algorithme ?

Exercice 3

Le problème des 12 pièces possède plusieurs variantes. Considérons celle-ci :

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

Dans le cadre général où on dispose de N pièces :

- Proposez une approche par force brute pour résoudre ce problème.
- Proposez une approche *divide-and-conquer* pour résoudre ce problème. Peut-on donner une borne sur le nombre de pesées ?

Exercice 4

Soit un ensemble $P = \{p_1, p_2, \dots, p_n\}$ 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 5

Dans la répétition 5, nous avons 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  $N \leq 1$ 
2      return 1
3   $Nb = 0$ 
4  for  $i = 1$  to  $N$ 
5       $NbTREE(i - 1) * NbTREE(N - i)$ 
6  return  $Nb$ 
```

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

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