TD 5 Backtracking : corrections

Licence 2 Informatique

Année académique 2024-2025

Exercice 1. soit a un tableau de n entiers strictement positifs notés $a[1]$, $a[2]$,....$a[n]$, et K un entier strictement positif. On cherche à savoir si K est la somme d'éléments de a, chacun ne pouvant être pris qu'au plus une fois. Si c'est le cas, le sous-ensemble d'éléments de a dont la somme donne K doit être représenté dans un tableau binaire x de n éléments, où $x[i] = 1$ si l'élément $a[i]$ fait partie du sous-ensemble et 0 sinon.

1/ Ecrire une procédure de type backtracking nommé $knapsack(x, ..)$ renvoyant vrai si K est la somme déléments de a. L'argument x est aloors un argument de retour permettant de déterminer les éléments du sous-ensemble. Les points représentent d'autres arguments à proposer qui vous semblent nécessaires. La procédure renvoie faux si K n'est la somme d'aucun sous-ensemble. Dans ce cas tous les éléments de x doivent être nuls.

Correction.

```
Algorithme : Knapsack(x,K,S,i)
si S == K alors
    retourner(vrai);
fin
si i > n alors
    retourner(faux);
fin
x[i] = 1;
si Knapsack(x,K,S+a[i],i+1) alors
    retourner(vrai);
fin
x[i] = 0;
si Knapsack(x,K,S,i+1) alors
    retourner(vrai);
fin
retourner(faux);
```

2/ Quelle est la complexité au pire des cas en notation 0 de votre méthode ?

Correction. En $O(2^n)$.

Exercice 2. *On souhaite placer sur un échiquier représenté par une matrice carrée* 8×8, *8 dames de sorte qu'elles ne soient pas en prise entre elles. Deux dames sont en prise si et seulement si elles se trouvent sur la même ligne, la même colonne ou la même diagonale de l'échiquier.*

1/ En vous inspirant de la méthode de backtracking pour le sudoku du cours, écrire le pseudo-code d'un algorithme de backtraking permet de déterminer une solution possible de placement des dames. La méthode renvoie "vrai" dès q'une solution a été trouvée et "faux" si aucune

n'existe.

Correction : L'échiquier est représenté par une matrice carrée binaire $\{E\}_{i,j=1,2,...,8}$ *de dimension* 8×8. $E_{ij} = 1$ *ssi la case* (i, j) *est occupé par une dame, et* 0 *sinon.* E *est une variable globale. On suppose l'existence d'une methode "possible(i,j)" renvoyant "vrai" s'il est possible de placer une dame sur* (i, j)*, sans qu'elle soit en prise avec d'autres déjà placées, et "faux" sinon.*

```
Algorithme : BacktrackingQueens(i)
si i == 9 alors
    retourner(vrai);
fin
sinon
    pour j allant de 1 à 8 faire
        si possible(i,j) == vrai alors
            E[i][j] = 1;
            si BacktrackingQueens(i+1) == vrai alors
                retourner(vrai);
            fin
            E[i][j] = 0;
        fin
    fin
fin
retourner(faux);
```

2/ Ecrire le pseudo-code d'un algorithme de backtraking permet d'afficher toutes les solutions possibles de placement des dames. La méthode ne renvoie rien.

Correction :

```
Algorithme : BacktrackingEchecsAll(i)
si i == 9 alors
    afficher(E);
fin
sinon
    pour j allant de 1 à 8 faire
        si possible(i,j) == vrai alors
            E[i][j] = 1;
            BacktrackingEchecsAll(i+1);
            E[i][j] = 0;
        fin
    fin
fin
```

3/ Peut-on écrire que la complexité du second algorithme est en $O(n!)$.

Correction

L'algorithme proposé positionne les dames colonne par colonne du damier. Il y a au plus n *façons de placer une dame sur la première colonne. Le placement sur la première colonne exclut au moins une ligne entière. Il y reste au plus* $(n-1)$ *façons de placer la seconde sur la deuxième. Ce qui exclut une deuxième ligne entière. Et ainsi de suite. Comme "BacktrackingEchecsAll" teste tous ces placements, il y a au plus* $n * (n-1) * (n-2) .... * 1 = n!$ *tests. D'où la complexité en* $O(n!)$. *Mais celle-ci est une surestimation. car des diagonales sont aussi exlues au fur et à mesure.*

Exercice 3. *Au jeu d'échecs, un cavalier sur la case* (i,j) *(où* $i,j \in \{1,2,...,8\}$ *) de l'échiquier* (8×8) *peut se rendre sur une case* (i',j') *(où* $i',j' \in \{1,2,...,8\}$ *) si et seulement si elle vérifie :*

$$|i-i'|=2 \text{ et } |j-j'|=1$$
$$\text{ou}$$
$$|i-i'|=1 \text{ et } |j-j'|=2$$

Le problème de la tournée d'un cavalier consiste, partant de la case $(1,1)$*, à se déplacer sur toutes les cases de l'échiquier, une et une seule fois, et à revenir en* $(1,1)$*.*

Ecrire une méthode de backtracking permettant de trouver un tel tour et de l'afficher, ou de renvoyer si le tour n'existe pas.

Correction. Un algorithme naïf serait de faire comme suit. Au pire des cas si toutes les possibilités sont énumérées. La complexité serait en $O((n^2)^{n^2})$ *car la double boucle est effectuée à chaque appel récursif.*

On suppose l'existence de la méthode "possible(i,j,k,l)" renvoyant "vrai si le cavalier peut se déplacer de la case (i,j) *à* (k,l) *et "faux" sinon.* E *est la matrice représentant l'échiquier.*

```
Algorithme : BacktrackingKnight(r,c,n)
si (n == 64) et (possible(r,c,1,1) == vrai) alors
|  retourner(vrai);
fin
pour r' = 1,2,...,n faire
|  pour c' = 1,2,...,n faire
|  |  si (possible(r,c,r',c') == vrai) et (E[i][j] == 0) alors
|  |  |  E[r'][c'] = 1 ;
|  |  |  si BacktrackingKnight(r',c',n+1) alors
|  |  |  |  retourner(vrai);
|  |  |  fin
|  |  |  E[r'][c'] = 0 ;
|  |  fin
|  fin
fin
```

On peut réduire cette complexité en utilisant une structure de graphe (des listes d'adjacence). Chaque position est liée dans ce cas à la liste exacte des positions accessibles par un cavalier. Mais même cette amélioration n'est pas efficace en pratique.