

## TD 2 : Recherche exhaustive et *backtrack*

### Exercice 1.

Somme de sous-ensemble

On considère un ensemble  $X$  d'entiers (qui peuvent être positifs ou négatifs) et une cible  $t \in \mathbb{Z}$  et on veut écrire un algorithme pour savoir si il existe un sous-ensemble de  $X$  dont la somme des éléments vaut  $t$  exactement. Les éléments de l'ensemble  $X$  sont stockés dans un tableau  $T$ . Par exemple, si  $T = [-5, 2, 7, -3, 2, -6]$  et  $t = 5$ , alors il suffit de prendre  $[2, 7, 2, -6]$  dont la somme bien 5. Par contre si  $t = 10$  ou  $t = -13$ , il n'y a aucune solution.

1. On veut parcourir tous les sous-tableaux de  $T$ . Proposer une façon de les représenter.
2. Écrire un algorithme pour résoudre le problème et analyser sa complexité.

On suppose maintenant que  $T$  ne contient que des entiers positifs.

3. Écrire un algorithme, de type *retour sur trace*, pour le problème modifié et analyser sa complexité. Indication. Écrire un algorithme qui prend en entrée non seulement  $T$  et  $t$  mais aussi  $i$ , et détermine s'il existe un sous-tableau de  $T_{[0,i]}$  dont la somme vaut  $t$ . Expliciter l'appel initial.
4. Faire tourner l'algorithme précédent sur  $T = [1, 3, 7, 11, 2]$  avec  $t = 12$ .

### Exercice 2.

Parcours d'objets

Les algorithmes de recherche exhaustive nécessitent de parcourir différents types d'objets. Le but de cet exercice est d'écrire quelques uns de ces parcours.

Pour chacun des ensembles donnés par la suite, effectuer les tâches suivantes :

- déterminer un ordre de parcours de l'ensemble, avec en particulier un début et une fin ;
- écrire un algorithme SUIVANT qui étant donné un élément renvoie le suivant pour l'ordre déterminé ;
- analyser la complexité de l'algorithme SUIVANT ;
- si possible, compter le nombre total d'élément dans l'ensemble.

Les ensembles à traiter sont les suivants :

1. Ensemble des  $n$ -uplets d'entiers entre 0 et  $k-1$ .
2. Ensemble des  $n$ -uplets d'entiers croissants entre 0 et  $k-1$ . Par exemple, lorsque  $n = k = 3$  l'ensemble des triples croissants sur  $\{0, 1, 2\}$  est  $[0, 0, 0]$ ,  $[0, 0, 1]$ ,  $[0, 0, 2]$ ,  $[0, 1, 1]$ ,  $[0, 1, 2]$ ,  $[0, 2, 2]$ ,  $[1, 1, 1]$ ,  $[1, 1, 2]$ ,  $[1, 2, 2]$  et  $[2, 2, 2]$ .
3. Ensemble des combinaisons de  $k$  éléments de  $\{0, \dots, n-1\}$ , c'est-à-dire des sous-ensembles de  $k$  éléments de  $\{0, \dots, n-1\}$ .

### Exercice 3.

Cryptarithme

Un *cryptarithme* est un casse-tête logique qui consiste en une équation mathématique où les lettres remplacent des chiffres à trouver. Par exemple résoudre le cryptarithme OASIS + SOLEIL = MIRAGE consiste à assigner des chiffres entre 0 et 9 à chacune des lettres afin que l'équation résultante soit valide. Chaque chiffre étant assigné à une lettre différente. Ici, il faut trouver  $73858 + 876456 = 950314$ , c'est-à-dire  $O = 7$ ,  $A = 3$ , etc. On note  $n$  le nombre de lettres total apparaissant dans le cryptarithme, et on suppose toujours que  $n \leq 10$ .

1. Comment décrire toutes les solutions possibles ?
2. Écrire un algorithme qui prend en entrée les trois mots  $a$ ,  $b$  et  $c$  d'un cryptarithme, et renvoie un dictionnaire qui à chaque lettre apparaissant dans un des mots associe un entier entre 0 et  $n-1$ . Si il y a moins de 10 lettres différentes dans  $a \cup b \cup c$ , on complétera avec des lettres n'apparaissant ni dans  $a$ , ni dans  $b$ , ni dans  $c$ , afin d'avoir 10 lettres distinctes. En supposant que toutes les opérations du dictionnaire se font en temps  $O(1)$ , analyser la complexité de l'algorithme.
3. Écrire un algorithme qui étant donné un mot  $m$ , un dictionnaire tel que celui de la question précédente, et une permutation  $\pi$  de  $\{0, \dots, 9\}$  représentée par un tableau, renvoie la valeur du mot en remplaçant chaque lettre par l'entier  $\pi(i)$  où  $i$  est la valeur associée à la lettre par le dictionnaire. Analyser sa complexité.
4. Écrire un algorithme de résolution de cryptarithme et analyser sa complexité.

### Exercice 4.

Huit reines

Le problème des huit reines demande s'il est possible de placer 8 reines sur un échiquier  $8 \times 8$ , sans qu'elles se menacent l'une l'autre : deux reines se menacent si elle sont sur la même ligne, la même colonne, ou la même diagonale (ou anti-diagonale).


C'est en effet possible, et il y a même plusieurs solutions. Ce qui nous intéresse ici est de calculer toutes les solutions possibles. De plus, on généralise le problème au cas des  $n$  reines : on cherche à placer  $n$  reines sur un échiquier  $n \times n$ , avec toujours les mêmes conditions.

1. 1. Montrer que le problème n'admet aucune solution pour  $n = 2$  et  $n = 3$ .  
2. Démontrer que dans toute solution, chaque ligne de l'échiquier contient exactement une reine.  
On représente une solution par un tableau  $Q$  de taille  $n$ , tel que  $Q_{[i]} = j$  s'il y a une reine en case  $(i, j)$  de l'échiquier.
2. 1. Quelle propriété de  $Q$  est impliquée par le fait deux reines ne peuvent pas appartenir à la même colonne ?  
2. Écrire une fonction qui teste si  $Q$  est une solution valide et analyser sa complexité.  
3. En déduire un premier algorithme de recherche exhaustive et analyser sa complexité.
3. Écrire un algorithme de type *retour sur trace* pour le problème. *Indications.* Écrire un algorithme plus général, qui prend en entrée  $Q$  et  $r$ , suppose que les  $r$  premières reines de  $Q$  ne se menacent pas mutuellement, et cherche toutes les façons de compléter  $Q$ .

#### Exercice 5.

*Sac-à-dos*

Étant donné  $n$  objets  $(t_0, v_0), \dots, (t_{n-1}, v_{n-1})$  et une taille  $S$ , on cherche un sous-ensemble  $I \subset \{0, \dots, n-1\}$  qui maximise la valeur totale  $\sum_{i \in I} v_i$  tout en respectant la contrainte  $\sum_{i \in I} t_i \leq S$ .

 Généraliser l'algorithme de type retour sur trace de l'exercice 1 au problème du sac-à-dos et analyser sa complexité.

#### Exercice 6.

*Codes de Gray*

Un code de Gray est une énumération de tous les mots binaire de longueur  $n$  dans un ordre particulier, afin que pour passer d'un mot au suivant, seul un bit ait besoin d'être modifié. Par exemple, l'énumération suivante est un code de Gray sur 2 bits : 00, 01, 11, 10. Les codes de Gray permettent une énumération très efficace des mots binaires, et donc des sous-ensembles d'un ensemble donné.

1. Écrire un code de Gray sur 3 bits.
2. Quelle est la longueur d'un code de Gray ?
3. Soit  $T$  un tableau contenant un code de Gray sur  $n$  bits. On définit les tableaux  $T^0$  et  $T^1$  par  $T^0_{[i]} = 0T_{[i]}$  et  $T^1_{[i]} = 1T_{[i]}$  : on concatène soit 0 soit 1 en tête de  $T_{[i]}$ .  
  1. Montrer que tous les mots binaires sur  $n+1$  bits sont dans  $T^0 \cup T^1$ .
  2. Montrer comment construire *facilement* un tableau  $T^+$  contenant un code de Gray sur  $n+1$  bits à partir des tableaux  $T^0$  et  $T^1$  (s'inspirer possiblement des cas  $n=2$  ou  $n=3$ ).
  3. En déduire un algorithme de construction d'un code de Gray sur  $n$  bits et analyser sa complexité.