

# Brute Force & Backtracking

La solution brute force à un problème de recherche / optimisation consiste à tout tester :

```
S:list # liste finie de solutions

# Problème de recherche
for s in S:
    if is_solution(s):
        return True
return False

# Problème d'optimisation
opt:int # = +infini ou -infini
for s in S:
    if is_better(score(s), opt): # si s est mieux que l'ancien mieux
        opt = score(s)
return opt
```

La difficulté est d'énumérer les éléments de  $S$ . C'est l'espace de recherche (space search). Cela revient à construire une bijection entre  $S$  et  $[0, \#S - 1]$

## I - Structures ensemblistes usuelles et énumération

On en dénombre 4 classiques :

- Intervalle de  $\mathbb{N}$
- Produit cartésien
- Sous-ensemble d'un ensemble fini  $E$
- Permutations

Mais il en existe également des moins classiques, il faudra s'adapter.

On reprend les exemples du chapitre 17.

### 1. Recherche d'un nombre premier dans $[a, b]$

Énumérer l'intervalle  $[a, b] \subseteq \mathbb{N}$

for i = a to b do ...

Remarque : S'il y a solution, le plus petit est trouvé d'abord, donc on résout aussi le problème d'optimisation associé.

### 2. Somme d'une tranche d'un tableau

Soit  $a$  un tableau d'entiers et  $s = \text{len}(a)$

On cherche  $(i, j) \in [0, n - 1] \times [1, n]$  avec  $i < j$

Produit cartésien

De manière générale, pour énumérer  $A \times B$  on effectue une double boucle telle que celle-ci :

```
for a in A:
    bar()
    ...
    for b in B:
```

foo()

...

### 3. Subset Sum // sac à dos

\$\rarr\$ Énumérer les sous-ensembles  $F \subseteq E$

Solution classique :

Généralement  $E$  est représenté par un tableau de longueur  $n = \#E$

On encode alors un sous-ensemble  $F$  par un tableau  $T_F$  de booléens de longueur  $n$  tel que  $\forall i \in [0, n-1] : T_F[i] = \text{True} \Leftrightarrow i \in F$

Comment énumérer les  $T_F$  ?

\$\rarr\$ Par ordre lexico-graphique, ce qui correspond par ailleurs à énumérer les entiers de 0 à  $2^n - 1$  en binaire.

### 4. Les N-dames

Nouvelle difficulté : identifier  $S$

Une première idée : On cherche à placer  $N$  dames sur  $N^2$  cases.

\$\rarr\$ On numérote les cases de 0 à  $N^2 - 1$ . Un candidat est alors une façon de choisir  $N$  cases parmi les  $N^2$ .

$\Rightarrow$  Il y a  $\binom{N^2}{N}$  candidats, ce est égal à :  $\frac{N^2!}{N!(N^2-N)!}$

Une meilleure idée :

- Il y aura exactement 1 dame par colonne donc on appelle dame  $n^\circ i$  celle qui sera placée dans la colonne  $i$ .
- Un candidat est alors une fonction  $f : [0, N-1] \rightarrow [0, N-1]$  et qui indique “la dame  $n^\circ i$  est sur la ligne  $n^\circ f(i)$ ”.
- De plus  $f$  est injective puisqu’on ne peut pas avoir 2 dames sur la même ligne.
- Donc elle est bijective par cardinal.

$\Rightarrow f$  est une permutation

Comment les énumérer ? Avec l’ordre lexico-graphique.

### 5. Problème du Cavalier

Solution naïve : Les candidats sont des suites de cases,  $f : [0, N^2 - 1] \rightarrow [0, N^2 - 1]$  et  $f$  est une permutation  $\Rightarrow N^2!$  candidats !

Bonne solution : Le cavalier a 8 déplacements possibles à chaque étape et il y en a  $N^2$ .

Une course de cavalier se décrit par une suite de  $N^2 - 1$  déplacements parmi  $\{A, B, C, D, E, F, G, H\}$ .

$\Rightarrow \{A, B, C, D, E, F, G, H\}^{N^2-1}$

$\Rightarrow 8^{N^2-1}$  candidats.

	A		B	
H				C
		C		
G				D
	F		E	

## II - BackTracking (retour sur trace)

On s'intéresse dans un premier temps aux problèmes de recherche. L'idée du backtracking est d'organiser l'espace de recherche  $S$  sous forme d'un arbre afin que les feuilles de l'arbre correspondent de manière bijective aux éléments  $s \in S$ .

Les nœuds représentent alors des "candidats partiels" de telle sorte qu'un candidat partiel ait quelque chose en commun avec tous les candidats  $s \in S$  qui sont ses feuilles dans l'arbre.

Pour énumérer  $S$ , on effectue un parcours en profondeur de l'arbre.

### Illustration sur le cavalier

Dans les nœuds on écrit la position actuelle du cavalier. Chaque nœud a 8 enfants qui correspondent aux sauts possibles A, B,...H.

Dès lors qu'un nœud se situe en dehors de l'échiquier, on arrête de l'explorer, on ne construit pas le sous-arbre de ce nœud (idem si case déjà visitée).

Le backtracking consiste à effectuer le parcours en profondeur de cet arbre sans le construire.

### Deuxième exemple : Subset Sum

Il s'agit d'énumérer  $P([0, n - 1])$ . Comment organiser  $S$  dans un arbre ? Il faut éviter la redondance.

Une bonne idée serait alors de créer un arbre de décision. Dans l'ordre de 0 à  $n-1$ , on se pose la question "je prends ou pas ?".

### Résumé

Si la façon de représenter  $S$  est pertinente, on peut s'épargner la recherche de certains sous-arbres. Dans le cas des optimisations, on va pouvoir ne pas explorer les sous-arbres qui ne respectent plus la contrainte mais il faudra bien trouver toutes les solutions pour déterminer l'optimum (ouverture sur le Branch and Bound).