

# Algorithmique 2022 - TP 3: Backtracking

Brian Pulfer  
Brian.Pulfer@unige.ch

26.10.2022 (actualisé)

## Remarques:

Veuillez suivre attentivement les spécifications de l'énoncé.

- **Réponses** - Essayez d'être exhaustif dans vos réponses. Les réponses comme "oui" et "non" ne permettent pas d'évaluer vos connaissances.
- **Tracés** - Mettez toujours des étiquettes d'axe et des titres pour les tracés.
- **Implémentation** - Veuillez utiliser exactement les noms fournis pour les fonctions. Vous pouvez utiliser *pytest* avec le script de test donné pour vérifier votre implémentation. N'hésitez pas à créer des fonctions auxiliaires.

- **Soumission**

Merci de télécharger vos devoirs sur moodle:

- **NomPrenom.pdf** avec votre rapport pour tout les exercices
- **tp3.py** avec votre implementations
- *Pas d'autres fichiers* (.pyc, .ipynb, DS\_Store, \_\_pycache\_\_, ...)

**Délai** - 08.11.2022, 23:59 CET

- Pour toute questions et remarques, merci d'utiliser le mail Brian.Pulfer@unige.ch.




## 1 N-Reines (3 Points)

Dans le problème des N-Reines, nous avons un nombre  $n$  de reines à placer sur un échiquier carré de dimensions  $n \times n$  sans qu'aucune reine n'en attaque une autre (une reine peut attaquer toute pièce se trouvant sur la même ligne, même colonne, ou mêmes diagonales qu'elle).

**Notez bien** que cela signifie que  $n$  peut prendre une valeur quelconque, et pas nécessairement égale à 8 comme dans le cas d'un plateau d'échecs.

### 1.1 Implémentation

Implémentez en Python une méthode de backtracking permettant de trouver toutes les solutions pour le problème des N-Reines. En cours, vous avez vu deux code générique pour le backtracking (Algo 10 ou 11), utilisant les trois fonctions  $B(x, k, n)$ ,  $T(x, k, n)$  et  $P(x, k, n)$ . Implémentez ces trois fonctions pour le problème des n-reines. Écrivez une fonction `solve_bt(n)` qui renvoie la liste des solutions pour un échiquier de dimension  $n \times n$ .

Le format de la solution renvoyée par la fonction `solve_bt(n)` est une liste dont chaque élément est un échiquier (liste en deux dimensions) avec symboles ,  et . Notez que les solutions sont renvoyées dans l'ordre où elles ont été trouvées, où nous essayons d'insérer les reines de gauche à droite de haut en bas (en sautant toutes les rangées jusqu'à celle où nous avons placé la dernière reine).

### 1.2 Contrainte

Expliquez à quoi correspondent les fonctions  $B(x, k, n)$ ,  $T(x, k, n)$  et  $P(x, k, n)$  dans le cadre du problème des n-reines.

### 1.3 Complexité

Avec une approche naïve, explorant toutes les combinaisons de reines, la complexité du problème serait de  $O(n^n)$ . L'algorithme de backtracking proposé permet de réduire la complexité à  $O(n!)$ . Tracer les temps d'exécution pour  $n = 1, 2, 3, 4, 5, 6$  pour votre algorithme `solve_bt(n)` et pour l'algorithme naïf donné `solve_naive(n)`.

Quel est la part de l'arbre qui est effectivement parcourue par `solve_bt(n)`? Est-ce que le gain est significatif? Et en pratique, quel est la complexité observée de l'algorithme que vous avez implémenté?

## 2 Sudoku (1 Point)

Le but de cet exercice est de remplir une grille de sudoku quelconque, un carré de 9 cases de côté subdivisé en 9 blocs de 3 cellules par 3 cellules, de sorte que chaque ligne, chaque colonne et chaque bloc contienne une et une seule fois chaque chiffre de 1 à 9.

### 2.1 Implémentation

Écrivez la fonction `solve_sudoku` qui résoud le problème du sudoku avec une stratégie Backtracking. Veuillez retourner uniquement la première solution trouvée (en essayant les valeurs de 1 à 9 dans l'ordre, de gauche à droite, de haut en bas).

### 2.2 Complexité

Dans le pire des cas, où la grille est vide, quel est la taille du domaine à explorer? Comment cela se fait-il que l'algorithme soit si rapide si au contraire on remplit quelques cases initialement (49, comme dans l'exemple donné dans le fichier)?

### 3 Somme d'un sous-ensemble de nombres (1 Point)

Soit  $W = \{w_1, w_2, \dots, w_n\}$  une suite de nombres positifs et soit  $M$  un entier positif.

On souhaite écrire un algorithme qui trouve toutes les combinaisons de  $x_i$  telles que  $\sum_i x_i w_i = M$  et  $x_i = 0$  ou  $1$ . Par exemple pour  $W = \{3, 4, 1, 2, 5, 6\}$  et  $M = 10$ .

#### 3.1 Implémentation

Implémentez en Python un algorithme de backtracking `solve_sum` récursif pour la somme d'un sous-ensemble de nombres. N'hésitez pas à ajouter des paramètres optionnels supplémentaires à la fonction donnée avec des valeurs par défaut.