Université de Genève

-

Sciences Informatiques

Algorithmique - TP 03

Noah Munz (19-815-489)

Novembre 2022

Contents

TP 03 - Backtracking

1 N-Reines

Implémentez en Python une méthode de backtracking permettant de trouver toutes les solutions pour le problème des N-Reines.

1.1 Implémentation Contrainte

Comme autorisé dans le mail, l'algorithme implémenté suit les conventions ainsi que la méthode définie en cours pour trouver toutes les solutions au problème des N-Reines.

La version vue en cours nous donne que (donnée la taille du board `n`, nos conditions implicites `B`, conditions explicites/espace de solutions `T` et l'algorithme 11):

```
1  # Algorithm 11: Backtring - recursive implementation
2      def rBacktrack (x, k, n) :
3          for y in T(x, k, n) :
4              x[k] = y
5              if B(x, k, n) :
6                  if P(x, k, n) : print Sol(x, k, n)
7                  rBacktrack (x , k+1,n)
```

- On itère (récursivement) sur les lignes du board en mettant à chaque fois 1 reine, ligne par ligne.
  $\Longrightarrow$ On a donc au maximum 1 reine par ligne.

- Les solutions sont donc les (indices des) colonnes du board,
  i.e. une solution $\mathbf{x}$ est un vecteur de $[\![0,\ n-1]\!]^n$, tel que la i^{eme} reine se trouve à l'index $(i, \mathbf{x}[i])$ du board. On a donc que le board est une matrice $M = (a_{ij}) \in \mathbb{M}_n$ telle que :

$$(a_{ij}) = \begin{cases} 1 & j = \mathbf{x}[i] \\ 0 & sinon \end{cases}$$

  C'est ceci qui va donc nous définir nos contraintes **explicites**, celles exprimées par l'évolution de `T(x, k, n)`. Nos contraintes explicites vont être, ici, "Pas 2 reines sur la même colonne".
  C'est à dire que à l'étape `k`, `T(x, k, n)` va nous retourner l'espace des solutions que peut prendre x_{k+1}, i.e. "Quelles colonnes sont libres ?". On va simplement enlever les anciens x_i des prochains T l'ensemble $([\![0,\ n-1]\!] \setminus \{x_0, ..., x_k\})$

  `T(x, k, n)` se charge d'imposer les contraintes **explicites** (mises à jour à chaque `k`) et on itère dessus avec la boucle `for` de la ligne 3 (Algo 11)
  $\Longrightarrow$ On a donc maximum 1 reine par colonne.

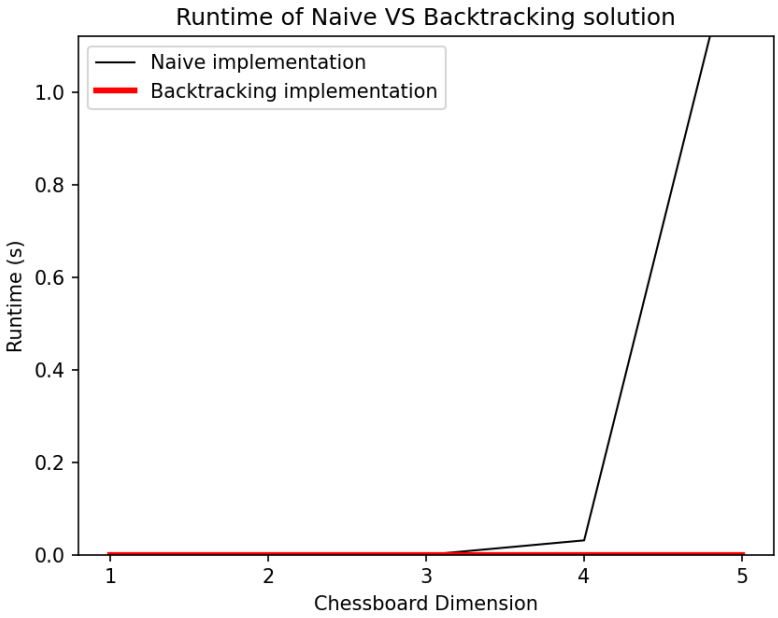
- Les B sont les conditions **implicites**, celles qui vont vérifier que "pas 2 reines sont sur la même diagonale (ou anti-diagonale)". Dans les lignes 5 à 7 de l'algo 11, on ne passe au prochain `k` seulement si `B(x, k, n)` return `True` i.e. seulement si les conditions implicites sur les diagonales sont respectés. C'est donc bien ce test qui va nous faire reculer si jamais on arrive à la fin de la boucle `for`.

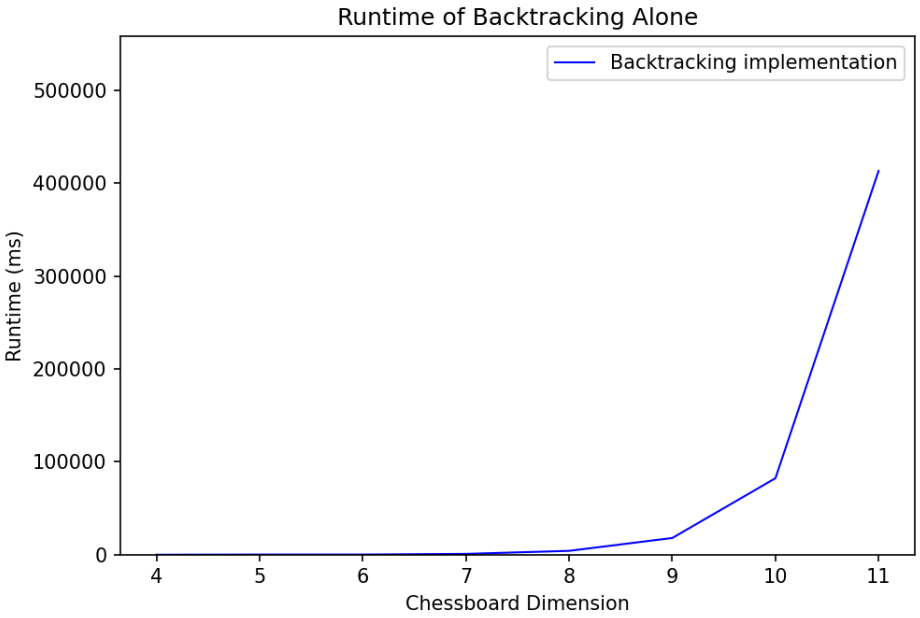
  $\Longrightarrow$ On a donc maximum 1 reine par diagonale (ou anti-diagonale).

- `P(x, k, n)` quant à elle, vérifie si jamais **x** est une solution "finale" du problème, i.e. donne la condition d'arrêt (total) sur la solution courrante. `P(x, k, n)` détermine si on a fini de chercher ou non. C'est à dire l'ensemble des solutions S est défini comme $S := \{x | P(x, k, n)\}$.

En combinant les contraintes implicites et explicites on a bien que notre alogirthme donne une solution valide au problème des N-Reines.

1.3 Complexité





Comme d'habitude on voit que le runtime de la fonction naive est écrasant devant celui de la version backtrack. Tellement que l'on a l'impression que la complexité en $O(n!)$ du backtrack serait presque bonne.
EC'est pourquoi un plot du runtime de la version backtrack seule à ajouté, en effet,
même si on a que $n!$ $o(n^n)$, $n!$ reste énorme. On montre ici qu'il explose avec le "chessboard dimension" qui augmente.

La partie qui est effectivement parcourue par `bt_solve` est celle qui ne mène pas un deadlock, i.e. celle qui ne mène pas une branche du graph (graph des path possibles que prend notre programme) qui ne fera jamais partie d'une solution.

`bt_solve` abandonne rapidement les choix "inutiles" qui sont faux sans continuer pendant longtemps à construire une solution qui était fausse depuis le départ (comme fait la fonction naive).

De plus la complexité de notre algo (ici pour $n = 1, ..., 6$) est déjà réduit par le fait qu'il n'existe pas de solution pour $n < 4$