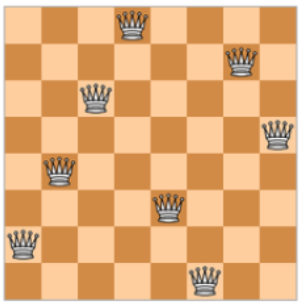
Chapitre 3 : Graphes

3 Exercices

Exercice 3.1 (Composantes connexes)**.** On appelle composante connexe d'un graphe non orienté tout ensemble de sommets qui sont reliés deux à deux par un chemin. Écrire une fonction `composantes_connexes(G)` qui renvoie la liste des composantes connexes de `G` sous forme d'une liste de listes. On considère que `G` est représenté sous forme d'une liste d'adjacence.

Exercice 3.2 (Retour sur trace)**.** Le problème étudié ici consiste à compter et identifier les différentes façons de placer sur un échiquier de 64 cases 8 reines de façon à ce que, relativement aux règles des échecs, elles ne se menacent pas mutuellement. Une reine peut se déplacer de façon rectiligne d'un nombre de cases arbitraire dans toutes les directions ; lorsqu'une reine est placée, la ligne, la colonne et les diagonales sur lesquelles elle se situe sont donc « condamnées ». On va utiliser pour cela un algorithme de retour sur trace.

Voici un exemple de configuration d'une solution :



On ne peut pas placer plusieurs reines sur une même ligne. On peut donc simplifier la modélisation en ne travaillant que sur les colonnes. Ainsi, on cherchera à ne retourner qu'un tableau d'index de colonne, chacun de ces index correspondant à la ligne de son propre index dans le tableau. Par exemple, pour la configuration précédente, en numérotant les lignes et colonnes à partir de `0` (en commençant en haut gauche), la liste correspondante est : `[3,6,2,7,1,4,0,5]`. On veut chercher toutes les configurations possibles.

1. Expliquer pourquoi, une liste de colonnes est une solution si et seulement si pour tout `i` différent de `j`, `c[i] != c[j]` et `(abs(c[i] - c[j]) != abs(i - j))`. En déduire une fonction `Verif_reines` qui prend en argument une liste `liste_reines` et renvoie `True` si `liste_reines` est une solution et `False` sinon.
2. Écrire une fonction `correct(col,L_r)` qui prend en argument un entier `col` et une liste `L_r` et renvoie `True` si la configuration `L_r+[col]` est correcte pour un échiquier de taille `len(L_r)+1`.
3. Écrire une fonction `solve(n)` qui renvoie la liste de toutes les solutions possibles pour un échiquier de taille n. On utilisera une boucle `for` et une liste qui contient toutes les solutions en augmentant la taille de l'échiquier au fur et à mesure. On ajoute les nouvelles solutions si et seulement si la configuration vérifie les conditions des questions précédentes.
4. En quoi ce problème est-il analogue à un parcours de graphe ?