Session #9 10/09/19 - Parcours de Matrice

Proposition du format de la séance :

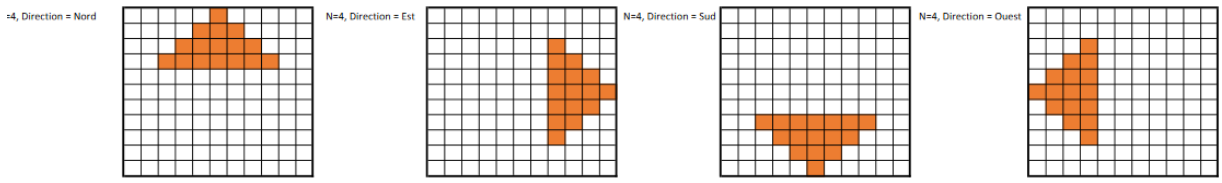
- **Petit rattrapage sur la boite à outils 'Matrice' - 20'**
- **Pour chaque exercice :**
  - **lecture seule de l'énoncé : 10'**
  - **je cherche les fonctions *utils* pour ma boite à outils**
  - **en méta-code, un ou plusieurs volontaire.s propose.nt une implémentation : 15'**

Pour ceux qui le souhaitent, ils le coderont en dehors et iront mettre leur version dans la page solution.

Boite à Outils :

- Rotation de matrice :

  Etant donné un tableau : coder la fonction qui effectue une rotation de 90° (peu importe le sens).



tiré du concours MDF: "Concours Credit Agricole 2018" - Flèche orientable (le reste de l'exo n'est pas très intéressant)

Les autres outils ont déja été abordés dans la Session #6 - thème 3 :

- **Trouver toutes les positions** d'un tableau ayant une valeurs donnée : Session #6 19/08/19 : Ma boite à outils
- **Trouver toutes les positions continues** ayant la meme valeur à partir d'un point de départ :Session #6 19/08/19 : Ma boite à outils

Exercices autours des matrices :

- Meilleur Dev de France 2018 - Session 18:30 : Exercice Barrage fluvial
- Meilleur Dev de France 2018 - Session 12:00 : Exercice Petri
- Concours de sélection de La Poste - Septembre 2018 : Exercice Fresque réfléchie
- Meilleur Dev de France 2017 (Session 2) : Exercice Démineur

Solutions

Rotation de matrice de 90° vers la gauche

Java

```
static void rotateLeft(int nbTurn, String[][] mat) {
    int N = mat.length;
    for (int i = 0; i < nbTurn; i++) {
        for (int x = 0; x < N / 2; x++) {
            for (int y = x; y < N - x - 1; y++) {
                String temp = mat[x][y];
                mat[x][y] = mat[y][N - 1 - x];
                mat[y][N - 1 - x] = mat[N - 1 - x][N - 1 - y];
                mat[N - 1 - x][N - 1 - y] = mat[N - 1 - y][x];
                mat[N - 1 - y][x] = temp;
            }
        }
    }
}
```

Code pour Barrage

Python

```
import datetime
import debug
import sys
from math import sqrt,ceil
from copy import deepcopy

def dist(tup_1, tup_2):
    return sqrt((tup_1[0]-tup_2[0])**2 + (tup_1[1]-tup_2[1])**2)


def neighbor_tab(tab, i, j):
    # Index Voisin en croix
    n = len(tab)
    m = len(tab[0])

    l = []
    for k in {-1, 1}:
        if (i + k) % n == i + k:
            l += [(i + k, j)]

        if (j + k) % m == j + k:
            l += [(i, j + k)]
    return l


def identifie(tab):
    new_table = deepcopy(tab)
    n = len(tab)
    for i in range(n):
        for j in range(n):
            if tab[i][j] == '#':
                for tup in neighbor_tab(new_table,i,j):

                    if tab[tup[0]][tup[1]] in {'1','2'}:
                        new_table[i][j] =  tab[tup[0]][tup[1]]
    return new_table


def elem_tab(tab, elem):
    "Ensemble des index des élement elem dans tab"
    p = list()
    n = len(tab)
```

```
    for i in range(n):
        for j in range(n):
            if tab[i][j] == elem:
                p.append((i,j))
    return p

def main():

    # Lecture de Table
    N = int(input())
    table = list()
    for i in range(N):
        table.append(list(input()))

    table[0][N-1] = '1'
    table[N-1][0] = '2'

    while True :
        new_tab = identifie(table)
        if all([new_tab[i] == table[i] for i in range(N)]):
            break
        table = new_tab

    tab1 = elem_tab(table, '1')
    tab2 = elem_tab(table, '2')
    print(ceil(min([min([dist(te, ta) for te in tab1]) for ta in tab2])))
```