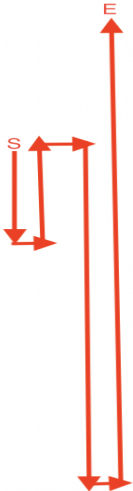


TEST5 : APPLICATION SERVEUR 2

Zayed Herma

```
DDDRUUURDDDDDDDDDDRUUUUUUUUUUUUUUUUUUU
#####

#.....#.....#
#.....#.....#
#.....#####.##
#.....E.....#F.....#
#.....U.....#.....#
#.....U.....#.....#
#####.U.....#####
#...SURU.....#.....#
#...DUDU.....#.#.....#
#...DUDU.#####
#...DRDU.....#.....#
#####DU.....#.....#
#.....DU.....#.....#
#.....DU.....#.....#
#.....DU.....#.....#
#.....DU.....#.....#
#E...DR.#####.##
#####
```



1 Présentation du problème

Trouver une trajectoire permettant au personnage d'atteindre une sortie sans brûler (sans passer par une case voisine du feu).

2 Utilisation en ligne de commande

Le code est dans le répertoire **test5**. Vous pouvez afficher, dans le terminal, une trajectoire possible de votre problème en ligne de commande. Le fichier **grid** correspondant doit se trouver dans le répertoire **test5/grids**. Une fois entré au fichier **test5** à travers la ligne de commande, l'affichage de la solution se fait en exécutant la ligne de commande **python test5.py grids/nomGridFile.txt**.

Par exemple la commande suivante :

```
[(base) zayed@MacBook-Pro-de-Zayed test5] % python test5.py grids/grid3.txt
```

A la suite de cette commande, on doit avoir une solution possible affichée dans le terminal, sous forme de séquence de lettres.

Pour plus de clarté, vous pouvez décommenter les lignes 132 et 133 pour voir la trajectoire sur la grille **grid**.

3 Présentation de la solution

3.1 bibliothèques et outils

- Le module **sys** : Pour lire les arguments donnés en ligne de commande.
- La méthode **open** : Pour lire les fichiers txt.

3.2 Formulation algorithmique du problème :

3.2.1 Présentation de l'algorithme

L'idée est de transformer notre problème en un problème de parcours de graphe et donc d'utiliser certains algorithmes élémentaires sur les graphes pour le résoudre.

Notre grille peut être vue comme un graphe non orienté $G=(P,A)$, avec **P** l'ensemble des points de notre grille (**#**, **.**, **E**, **F**) et **A** l'ensemble des arrêtes de notre grille (**(#,#)**, **(#,.)**, **(.,.)**, **(E,.)**.....etc).

Pour trouver une solution, l'algorithme utilisé est l'algorithme de parcours en profondeur (**depth-first search**), qui applique l'algorithme de rebroussement (**backtracking**) : tant qu'on peut progresser en suivant un arc, on le fait, et sinon on fait marche arrière. Une fois un point est visité, on le marque pour ne pas tourner infiniment dans un cycle.

Dans notre problème particulier, qui vise à trouver un chemin entre **"S"** et **"E"**, l'algorithme doit s'arrêter une fois tombé sur un point **"E"** et donc retourner une solution. Dans le cas contraire (l'algorithme a visité tous les points dans le graphe sans tomber sur **"E"**), le problème n'admet pas de solution.

Voici le pseudo-code de l'algorithme implementé dans **test5.py** :

Algorithm 1: Depth-first search

```
Input : Grid comme une matrice adjacente représentant la grille en question, le point de
départ S, path  $\leftarrow \emptyset$  (comme variable globale).
if  $S == E$  then
    //le point de départ est le point de sortie "E".
    return true
end
marquer neighbour comme visité ("V")
for neighbour in neighbours of S do
    while (neighbour is not visited "V") and (neighbour is not a tree "#") and (neighbour is
        not neighbour of "F") and (there is path from neighbour to "E") do
        ;
        //we add to path "D for down" or "U for up" or "R for right" or "L for left", to
        indicate the. direction of the choice.;
        path.add(direction from S to neighbour)
    end
end
return false;
```

Il y a deux cas possibles à la sortie de cet algorithme :

- L'algorithme renvoie **flase**, alors dans ce cas la variable globale est vide. Pas de chemin possible entre "S" et "E".
- L'algorithme renvoie **true**, alors dans ce cas le variable globale contient un chemin entre "S" et "E". Le problème admet alors une solution.

3.2.2 Complexité de l'algorithme en temps et mémoire :

- **Complexité en temps** : la complexité en temps est au pire des cas $\mathcal{O}(P + V)$. Car, chaque point du graphe est mis dans la pile d'exécution au plus une fois et donc examiné au plus une fois, et chaque arc est considéré au plus une fois lorsque son origine est examiné.
- **Complexité en mémoire** : la complexité en mémoire est au pire des cas $\mathcal{O}(V)$. Car, chaque point du graphe peut être présent sur la pile d'exécution au plus une fois .

4 Le temps passé sur ce test

Le temps passé sur ce test est environ 4 heures.