TP4 - Plus courts chemins en C

Pour illustrer l'intérêt de l'algorithme A* par rapport à Dijkstra, on s'intéresse à la recherche de plus courts chemins dans des grilles avec obstacles.

I. Instances du problème

On considère une grille carrée de caractères représentant un environnement, où :

- S : point de départ
- T : objectif à atteindre
- . : case libre
- # : obstacle infranchissable

Exemple de grille :

```
##########
#S.......#
#.####...#
#....#...#
#....#..T#
##########
```

Les déplacements autorisés sont les 8 directions : haut, bas, gauche, droite (coût = 1) et les quatre diagonales (coût = $\sqrt{2}$).
Le but du TP est trouver un plus court chemin de s à t.

Pour pouvoir créer de telles grilles sur la pile, on fixe leur taille 20×20 à l'aide de la directive de préprocesseur `#define N 20`.
Toutes les grilles manipulées auront donc le type `char g[N][N]`.

1. Écrire une fonction `void affiche(char grille[N][N])` qui affiche le contenu d'une grille dans le terminal.

II. Implémentation de Dijkstra

Pour implémenter l'algorithme de Dijkstra, on utilisera la structure suivante :

```
typedef struct {
    int x, y; //coordonées du sommet
    double dist;  // distance connue à la source
    double priorite;  // priorité dans la file
    int pred_x, pred_y; //coordonées du predecesseur du sommet
} sommet;
```

Les variables globales suivantes définissent les déplacements possibles :

```
int dx[8] = {-1, 1, 0, 0, -1, -1, 1, 1};
int dy[8] = {0, 0, -1, 1, -1, 1, -1, 1};
double cout[8] = {1, 1, 1, 1, sqrt(2), sqrt(2), sqrt(2), sqrt(2)};
```


2. Implémenter une fonction `void dijkstra(char grille[N][N], sommet sol[N][N], int s_x, int s_y, int t_x, int t_y)` qui recherche un plus court chemin dans la grille entre les points (s_x, s_y) et (t_x, t_y) de la grille, et remplit pour cela le tableau `sol`. Lorsque le sommet extrait de la file correspond à t, alors on interrompra prématurément l'algorithme (comme dans A*).

3. Pour visualiser quels sommets on été extrait de la file lors de l'algorithme, écrire une fonction `void marque(char grille[N][N], sommet sol[N][N])` qui remplace dans la grille les cases visités par le char `'0'`.

4. Lorsqu'un chemin est trouvé, on remonte les parents pour reconstituer la trajectoire. Écrire une fonction `void chemin(char grille[N][N], sommet sol[N][N], int s_x, int s_y, int t_x, int t_y)` qui remplace dans la grille les cases du chemin de s à t par le caractère `*`.

5. Tester votre fonction sur les instances de tests en affichant les sommets visités ainsi que le chemin trouvé.

V. Algorithme A*

6. L'algorithme A* reprend la même logique que Dijkstra, mais ajoute une *heuristique* h définie par :

$$h((x, y)) = \sqrt{(x - t_x)^2 + (y - t_y)^2}$$

ce qui guide l'exploration vers la cible.

7. Implémenter une fonction `astar` similaire à la fonction `dijkstra` en utilisant cette heuristique.

8. Sur les différentes instances, comparer le nombre de sommets visités lors de l'exécution de Dijkstra et celle de A*.