WE LOVE PYTHON

Shortest path identification

Team Name： 名字取得最好的组

Team member

张黄灏、吴思澜、章甜、郑宜暄

CONTENT

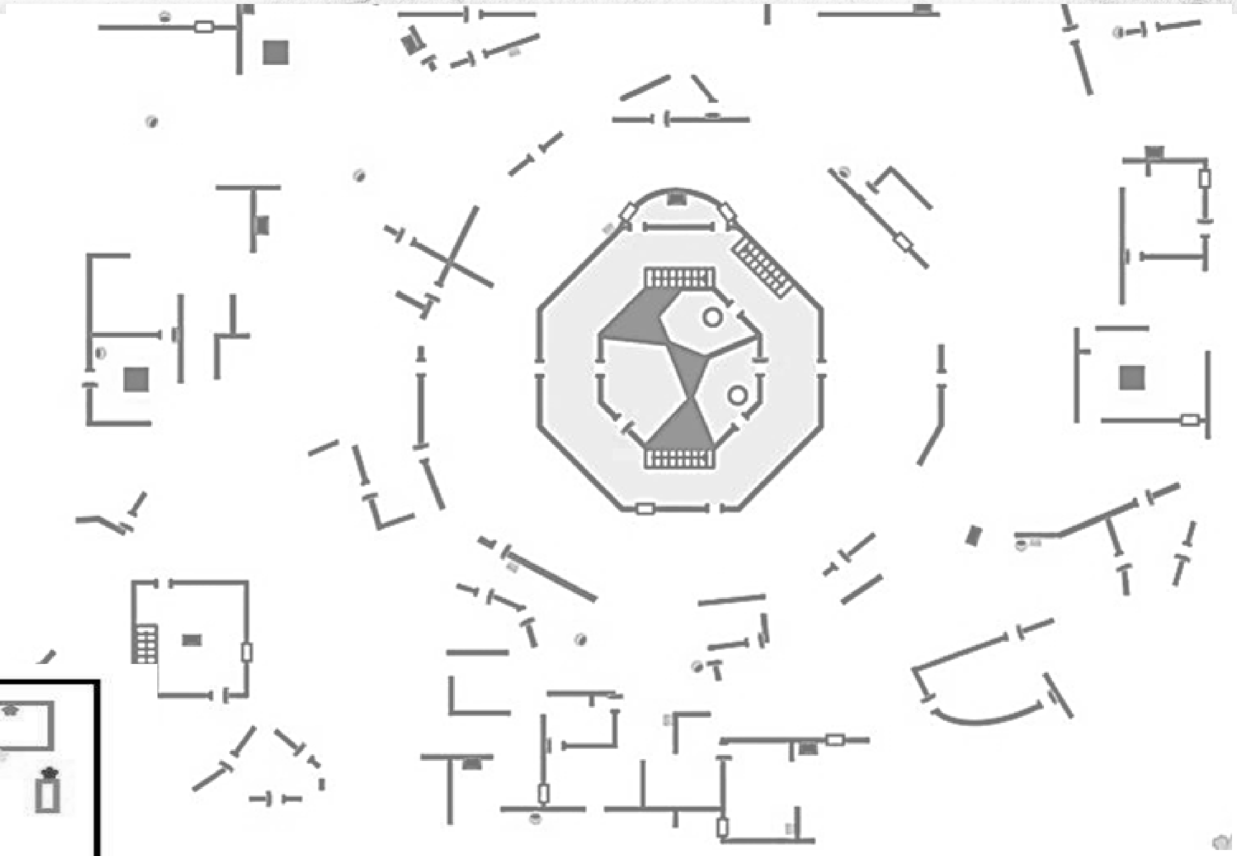
01

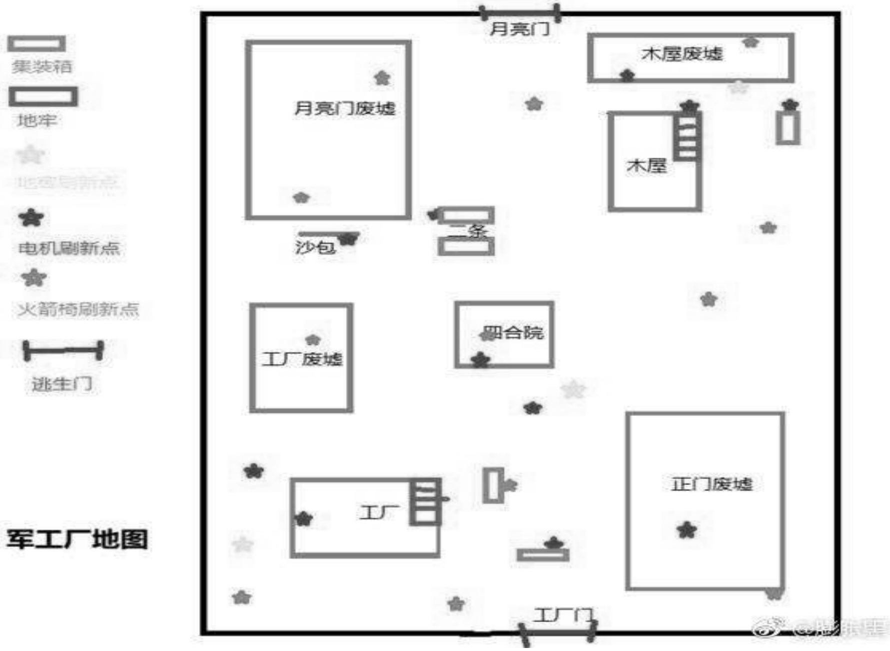
Background

Background

When we use map apps like baidu, gaode, etc, they will automatically recommend the shortest route for us. However, this kind of map apps can’t cover some special area, especially small indoor environment. For example, we can use Gaode to seek a route to a shopping mall. It is a pity that after we enter the mall, these apps can’t help us to find specific stores any longer. Their feasible function stop here.

Game style





Besides, seeking the shortest route for winning a game with fixed map is also **beyond** these apps' function.

But the maps are so **complicated** that it is hard for us to figure out the shortest escape route immediately.

Our Goals

So we'd like to develop a program that can make up for the defect we talked before, which means you can input a map like the specific locations of stores in a big mall, and ask the program to identify the shortest path for you.

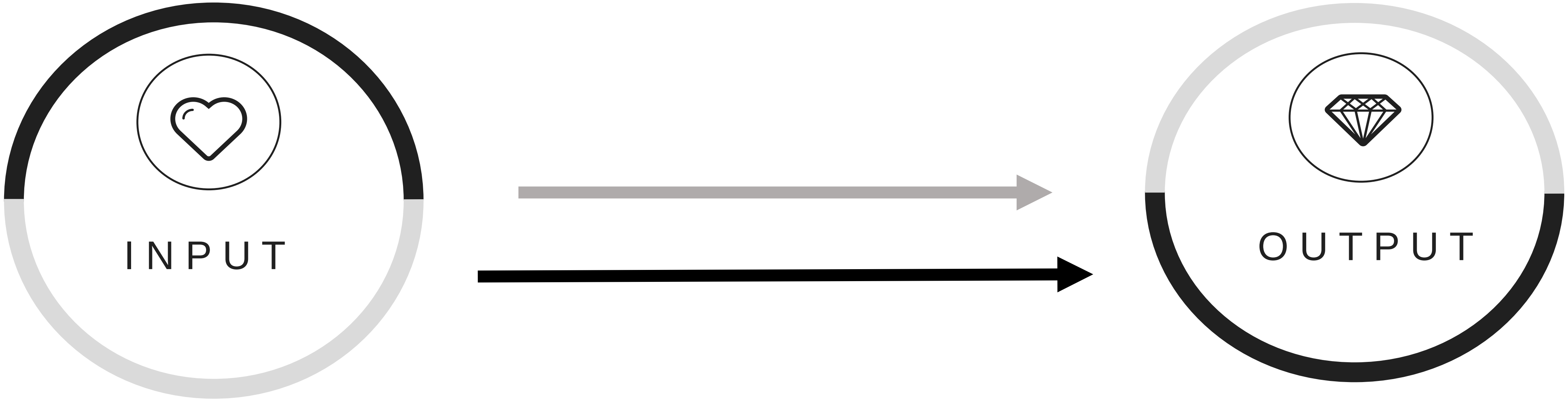
What's more, except for the use we just mentioned, this program can also benefit a lot in the virtual world.

With this program, all you have to do is to prepare the specific map in advance and input your location when you want. Then you will get the best way leading to the gate as soon as possible.

02

Description

Descripition



01 INPUT

INPUT

The input to the program is a text file. It contains comments, a start position, a goal position and a 2D map. Sentences begin with a dollar character($) are comment lines. The first non-comment line of that text file encodes the start position of the robot and the second non-comment line is the goal position. Those are two integer numbers separated by spaces. They encode x, y, in that order.

The second part of the input is the map. It consists of either '.' (a free cell) or 'X' (occupied cell = wall). The bottom left cell in the text file is located in the origin of the coordinate system (0, 0). The x-axis follows this line to the east (right) while the y-axis follows this column to the north (up).

The width of the map is defined by the number of characters in the line. The height of the map is defined by the number of lines in the map part.

02 OUTPUT

OUTPUT

The program has to output the map with the found path marked with the letter P. The format follows exactly the input map format, except that the found path, including the start point and the goal point, are marked with the letter ’P’ instead of a ’.’

03

Explanation Of the code

Simplify the map

```
COMMENT_MARKER = '$'
ROAD_MARKER = '.'
WALL_MARKER = 'X'
PATH_MARKER = 'P'
```

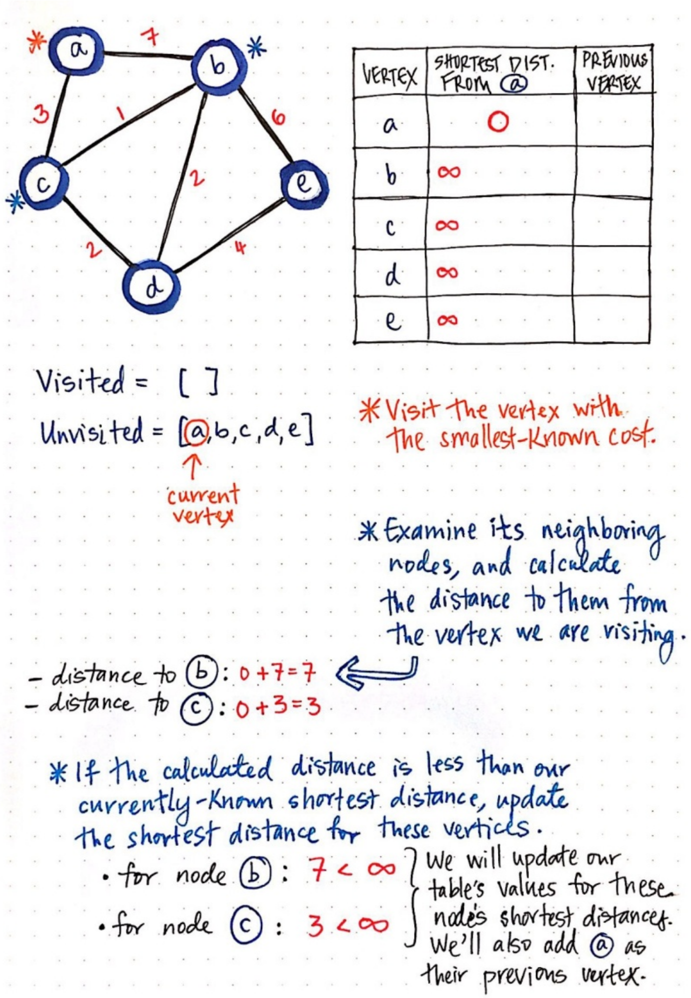
```
$
$ Start position, goal position and map
$
$ Start position: x, y
$ Coordinate systen starts with ., . in lower left corner
$
$ Now the actual start position:
$
6 4
$
$ Now the goal position:
$
7 6
$
$ Now follows the map (consists of only . or X)
$
..................
..................
..................
...XXXXXXX........
...X.....X........
...X.....X........
...X.....X........
..................
..................
```

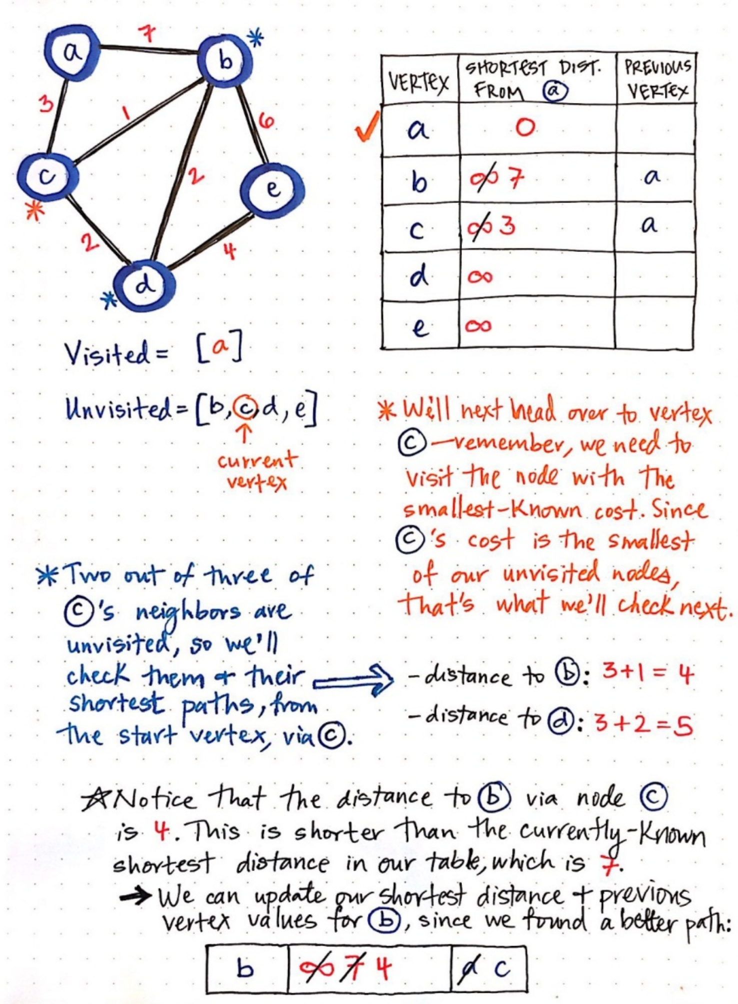
```
..................
..................
..................
...XXXXXXX........
...X.....X........
...X.....X........
...X.....X........
..................
..................
```

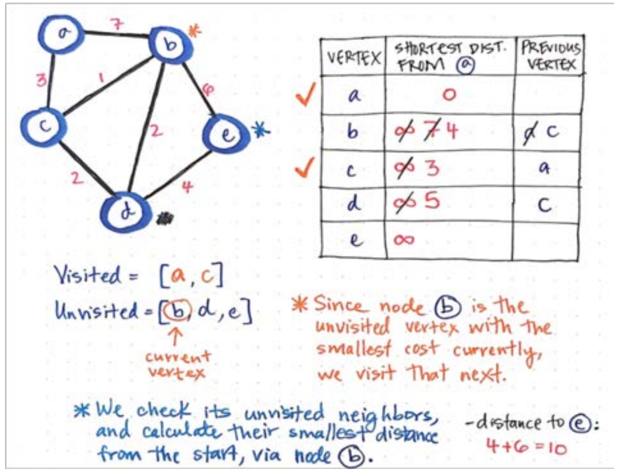
↓

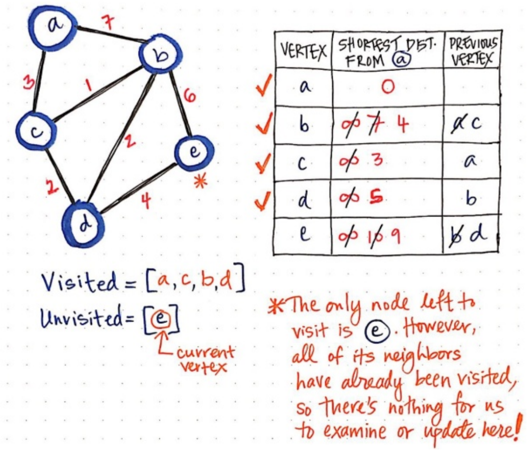
```
..................
..................
.......PPP........
...XXXXXXXP.......
...X..P..XP.......
...X...P.XP.......
...X....PXP.......
.........P........
..................
```

Core: Dijkstra’s algorithm





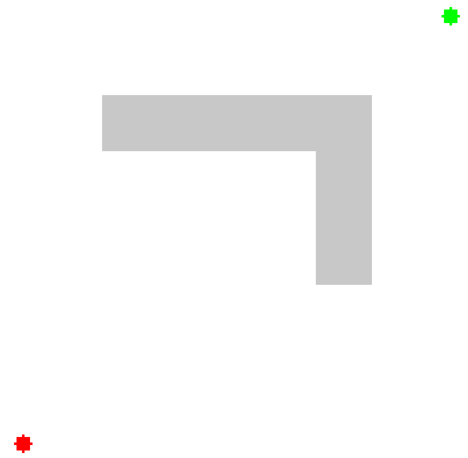




[a,c,b,d,e]

Pseudocode

```
Initialize all map cells with infinite distance
Initialize the start cell with 0 distance
Add start cell to wavefront (active set)
While wavefront not empty:
    select the first node
        remove node from the wavefront
        if node == goal:
            break
        for all 8 neighbors of the node:
            if the neighbor is valid and not a wall:
                neighDist = distance saved in node +
distanceToNeighbor (1 or sqrt(2))
                if neighDist < distance saved in neighbor:
                    update distance saved in neighbor with nieghDist
                    append neighbor to the wavefront (active set)
Create the path:
       starting from the goalCell select the neighbor with the lowest
distance saved
              add the selection to the path and continue with the
selection to save it’s lowest neighbor till you reached the start
point
Print the map with the path
```



Optimization

```
# Use 8 directions

BORDERING_DIRECTIONS = [
    Direction(dx=1, dy=0),  # Up
    Direction(dx=0, dy=1),  # Right
    Direction(dx=-1, dy=0),  # Down
    Direction(dx=0, dy=-1)  # Left
]

BORDERING_DIRECTION_TRAVEL_COST = 1

DIAGONAL_DIRECTIONS = [
    Direction(dx=1, dy=1),  # Upper right
    Direction(dx=-1, dy=1),  # Bottom
right
    Direction(dx=-1, dy=-1),  # Bottom
left
    Direction(dx=1, dy=-1)  # Upper left
]

DIAGONAL_DIRECTION_TRAVEL_COST = sqrt(2)
```

```
# Wavefront algorithm
wavefront(active set) = [start cell]
# To get the eight points around the middle point
def get_neighbour_point(point, direction):
    ......
# Check if the point is in the map
def point_is_in_map(point, map):
    ......
# Check if the point is "wall"
def point_is_free(point, map):
    ......
# Label points of the determined path on the map
def set_path_on_map(row_index, column_index, map):
    ......

......
```

04

Extended Examples

INPUT

1 **Start position**

6 4

2 **Goal position**

7 6

3 **Map**

```
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
. . . . . . .   . . . . . . . . . .
. . . X X X X X X X . . . . . . . .
. . . X . .   . . X . . . . . . . .
. . . X . . . . . X . . . . . . . .
. . . X . . . . . X . . . . . . . .
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
```

OUTPUT

```
. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .
. . . . . . . P P P . . . . . . . .
. . . X X X X X X X P . . . . . . .
. . . X . . P . . X P . . . . . . .
. . . X . . . P . X P . . . . . . .
. . . X . . . . P X P . . . . . . .
. . . . . . . . . P . . . . . . . .
. . . . . . . . . . . . . . . . . .
```

INPUT

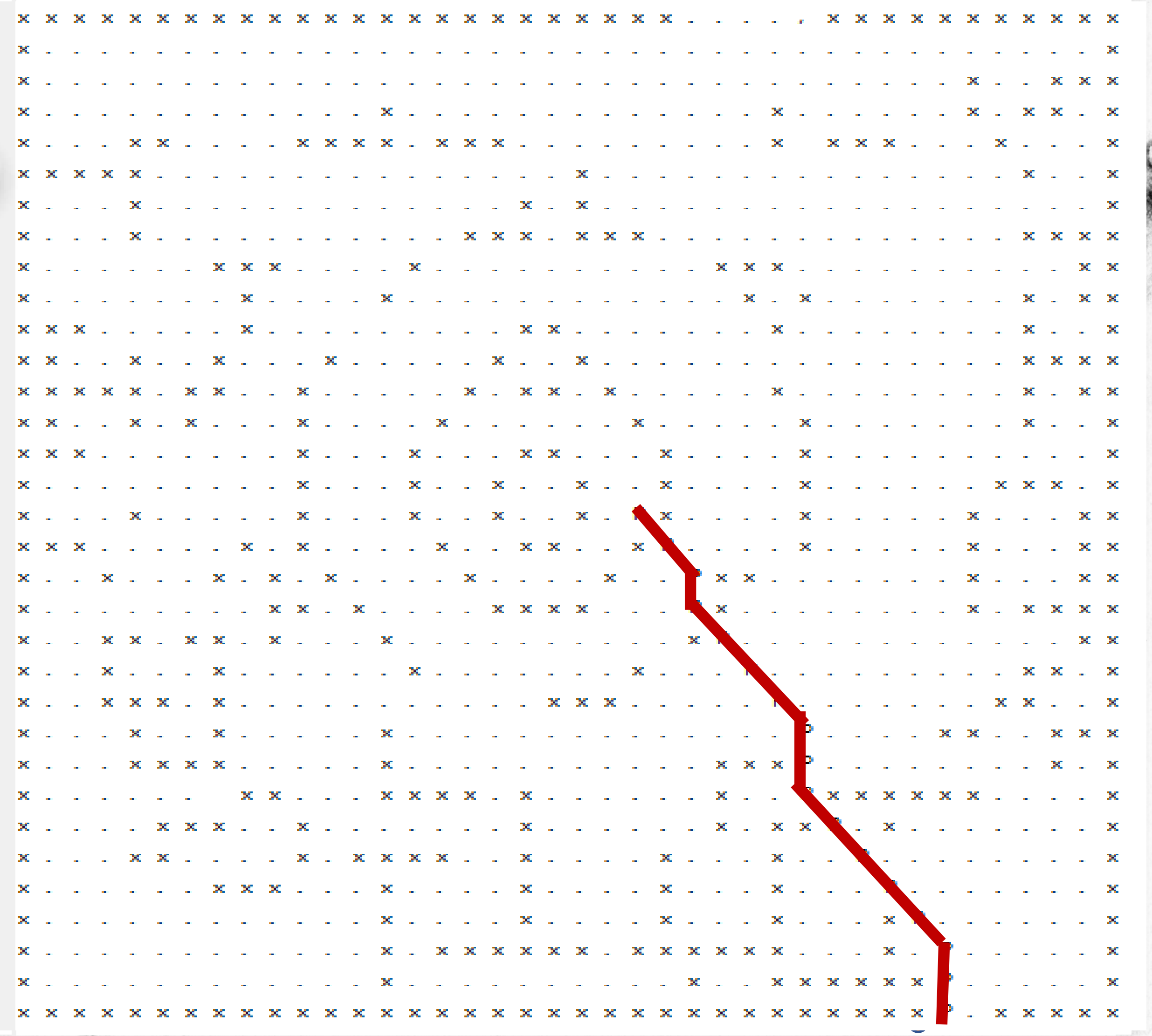
1 **Start position**

22 16

2 **Goal position**

33 0

3 **Map**



05

Reflection& Conclusion

THIS PROGRAM HELPS US:

•Offer point to point path planning ability for places that can’t be covered by the electronic map.

•Can be used in virtual world.

DIFICIENCY:

It must start with a manual entry map.

The position also need to be manually input.

WE LOVE PYTHON

Thanks for listening

Team

名字取得最好的组