ME5413-Homework3 Group number :16

LI BO ： A0269522Y XUE HAOYAN: A0268454U

Task 1: Graph Search Algorithms

1.1 Image preprocessing:

1.1.1 Add location information to the map

Mark multiple locations on a 1000x1000 pixel map and label them. In this process, we first use the matplotlib library to add, but found that using matplotlib will cause the pixels and color channels of the picture to change. So we implement the method as follows：

1. Imported the Image, ImageDraw and ImageFont modules of the PIL library. Then,
2. Read 'vivocity.png' through Image.open() method.
3. Define a dictionary locations, which contains information such as names, coordinates, marker colors, and marker sizes of multiple locations.
4. Define a font for adding text to the label. Next,
5. Draw multiple circles and labels on the image through the ImageDraw.Draw() method and the for loop.
6 Save the marked image to the 'vivocity_locations.png' file through the Image.save() method.

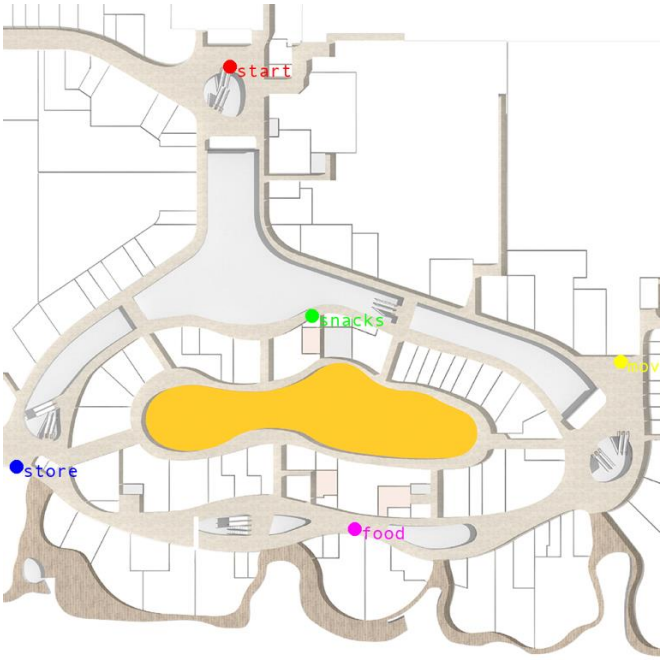

Figure 1.1 vivocity_with_locations

1.1.2 inflated map

Mentioned in the title "as a human, are expected to have a circular footprint of no less than 0.3m radius. "There are two ideas at the beginning, the first is to judge obstacles during the pathfinding process and then give The position of 1 grid pixel near the obstacle is also marked as an obstacle, but this increases the burden of the algorithm, so we finally use preprocessing to expand the map. The specific steps are as follows:

1. Use the cv2.imread function to read a binarized image, and set the pixels whose threshold is greater than or equal to 254 to 255, and the rest to 0 to obtain a binarized image.

2. Create a blank image with the same size as the original image, and the pixel value type is np.uint8.

3. For each pixel of the original image, determine whether there is a pixel with a pixel value of 0 in the eight surrounding pixels. If there is, it means that the pixel is an edge, and its pixel value is set to 0, otherwise its pixel value is set to 255.

4. Use the cv2.imwrite function to save the processed image to the specified path.

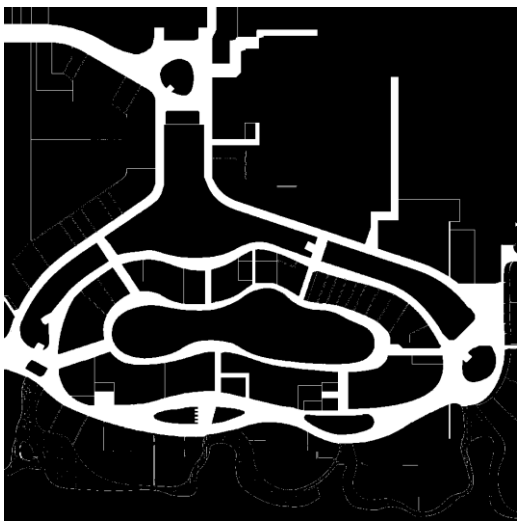

Figure 1.2 Before inflating

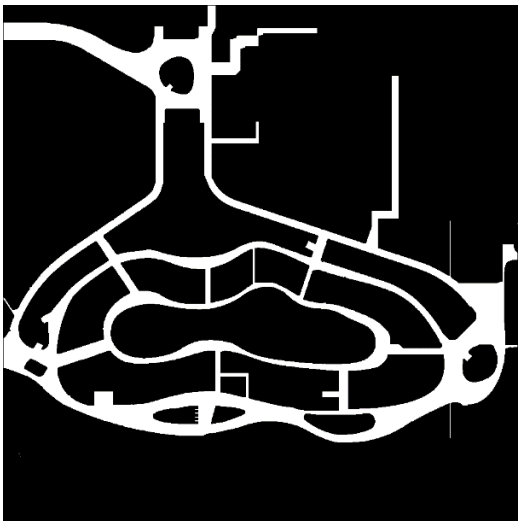

Figure 1.3 After inflating

1.2 A* planning algorithm.

A* algorithm is a heuristic search algorithm, which is often used in path planning problems. Our processing flow is as follows:

1. The calculated path first selects the expanded map, and the planned route is depicted on the original map.

2. Initialization: add the starting point to the open list openList, and set the g value of the starting point to 0.

3. Iterative pathfinding: Repeat the following steps until the end is reached or the open list is empty.

   a. Select the node with the smallest f value in the openList as the current node, delete it from the openList, and add it to the closed list closedList.

b. Expand the current node: Calculate the f value of the nodes around the current node (here contains eight directions), and add it to the openList.

c. Sort the nodes in the openList according to the f value to ensure that the nodes with the smaller f value are expanded first.

d. If the end point is in the openList, the search is complete.

4. Reverse path finding: starting from the end point, according to the parent pointer of each node, trace back to the starting point in turn to obtain the path.

(The heuristic function H uses the Manhattan distance, and G uses the Manhattan distance from the starting point to the point. During the search process, the step size is gradually increased from 4, and the step size is reduced until the search is close to the end point.)

To\From	start	snacks	store	movie	food
start	0.0	156.3	171.9	183.2	335.3
snacks	146.6	0.0	120.7	108.6	189.3
store	161.1	124.4	0.0	219.0	136.0
movie	181.9	139.2	248.2	0.0	119.7
food	322.6	136.3	120.2	118.5	0.0

1.2.1 Comparison of different heuristic functions

We also tried the Chebyshev distance, and the heuristic function we set. Through comparison, it is found that the Manhattan distance is more suitable for representing the distance between pixels, while the Chebyshev distance is more suitable for representing the distance between two rectangles. In path planning, the Manhattan distance is more suitable for expressing the distance between two points in a grid map, while the Chebyshev distance is more suitable for expressing the distance between two points in an octave map.

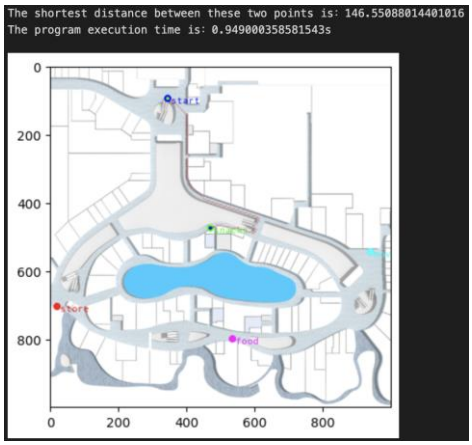

Figure 1.4 Harman Kardon

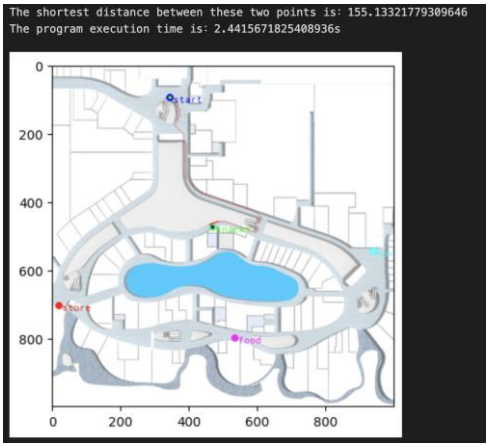

Figure 1.5 Chebyshev

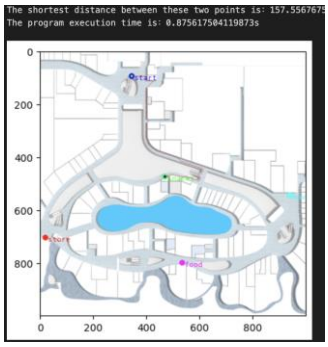

Figure 1.6. H = (dx2 + dy2) + 1.586 * min(dx2, dy2)

1.2.2 Degenerate the A* algorithm to Dijkstra's Algorithm

To degenerate the A* algorithm into Dijkstra's algorithm, we remove the heuristic component of the algorithm, which means that the cost of each node is only the cost to get from the origin to that node, regardless of the estimated cost to reach the goal node. Here are the steps we took to design Dijkstra's algorithm:

1. Initialize the start node with a cost of 0 and all other nodes with a cost of infinity.

2. Add the start node to the priority queue.

3. When the priority queue is not empty, perform the following operations:

a. Take the node with the lowest cost from the priority queue.

b. For each neighbor of the removed node, calculate the cost of reaching the neighbor via the removed node. This cost is the cost of getting to the neighbors from the fetched node plus the cost of the edges between the fetched node and the neighbors.

c. If the calculated cost is less than the neighbor's current cost, update the neighbor's cost with the calculated cost and add the neighbor to the priority queue.

4. Repeat step 3 until the target node is taken out or the priority queue is empty.

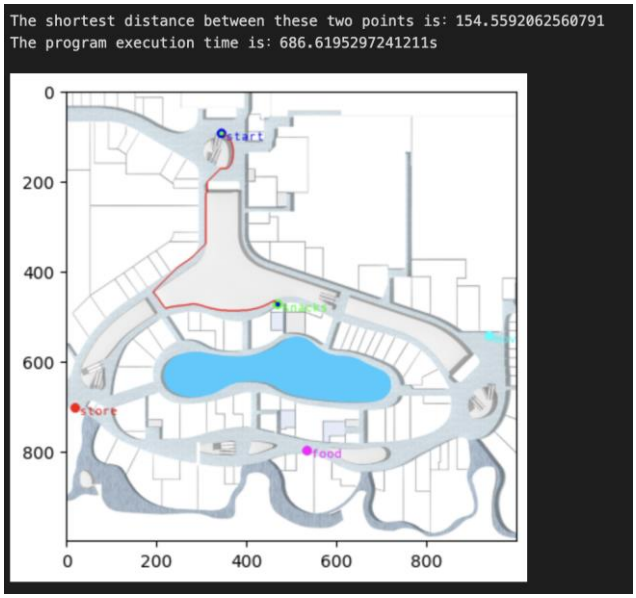

Figure 1.7 **Dijkstra's Algorithm**

Task bonus

2.1 Traveling Salesman Problem:

First of all this problem can be seen as a 5*5 distance table problem, which has nothing to do with finding the shortest path on a map, so it becomes a mathematical problem also known as a TSP problem.

The TSP (Traveling Salesman Problem) is the problem of finding the shortest path for a traveler who needs to travel to each of a given number of cities in turn and eventually return to the city of origin.

2.1.1 Iterating through all cases in a loop:

Since in the case of this task, the number of cases from the starting point and back to the starting point is relatively small, the first step is to use the enumeration method, recording all paths and their corresponding distances, and continuously updating the shortest path and shortest distance in a loop until the end of the traversal to obtain the final result.

```
Shortest path: start, store, food, movie, snacks
Shortest distance: 665.9
The program execution time is: 0.0008338000000094326s
```

2.1.2 Dynamic programming solutions:

```
Shortest distance: 665.9
The program execution time is: 0.0002071000000114509s
```

可以看出动态规划的方法和所有情况遍历一遍的方法最后求出的最短路径结果相同，但是动态规划在时间上要优于遍历的方法。

2.2 Show the final shortest route on the map: