CSE 6010 / CX 4010 Assignment 5
Total Amazement

Ting Liao, GTID: 903278773 | Chin Wang, GTID: 903506612

10/15/2020

For this Maze assignment, we tried to use the idea similar as divide and conquer. The main reason is that we tried to assign the division of labor across the team evenly. Also, it is much easier for us to focus on a small part without being distracted by other code.

Let’s first talk about our function. We implement the DFS to generate the maze and the BFS to solve the maze. Also, we modified the Linked List Queue file from HW2 to add the predecessor item in the Qnode and adjust the LLQ_Search function to return the value of the node’s predecessor. After using BFS, we utilized the LLQ_Search function to help us find the predecessor from each node and store in the array, BFS_modified, in order to print out the final solution.

Description of our Approach

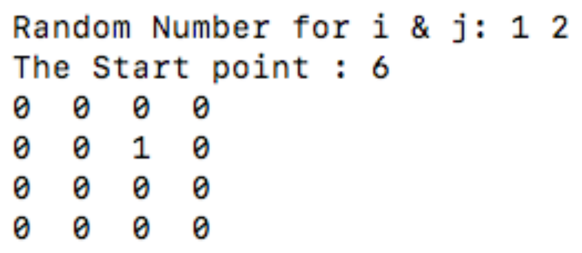
- For generating the maze, we randomly chose a node as start point and used depth-first search to connect the remaining available points. To backtrack the route, we set a variable to record the predecessor of a point. Hence, when the one route came to a node with all visited neighbors, the route had the way to go back to search new route. How about visited nodes? When the nodes were connected with each other, they would be labeled as visited by setting a variable for identifying whether the node was visited. By recursively calling depth-first search algorithm, we created a maze which was outputted in adjacency list.
- For solving the maze, we had to pass the above adjacency list into breadth first search algorithm. At first, our BFS algorithm was designed to pass other representation of maze rather than adjacency list. Therefore, we tried to transform adjacency list into the array which was more understandable for designing BFS algorithm. Then we utilized the property of queue to find the exact route in the maze. For ensuring that the route could backtrack after finding the end point, we recorded the predecessor of a point just like in DFS algorithm to complete the solution.

Future improvement

- When passing the adjacency list to the BFS algorithm, we transformed the maze into a new array rather than adjacency list. And this process can be skipped for better performance such as better space complexity. Therefore, the performance may be improved if we try to use adjacency list directly.
- In BFS algorithm, we put too many loops and if statements in the functions. The reason was that we did not use adjacency list directly. If we conceive a way for passing adjacency list into algorithm, the space and time complexity might be significantly decreased.

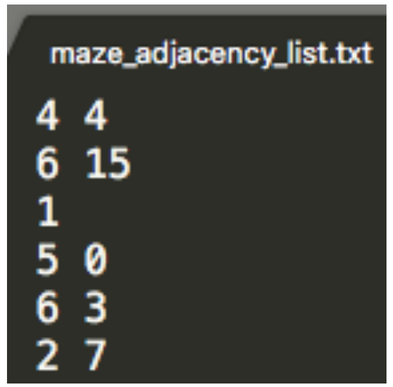
Evidence of Correct Operation

- Let’s first show the evidence with a small maze. As you can see in fig.1, we successfully pick the start point on the maze randomly. Then, in fig.2, it can randomly choose the direction to go and now we can guess that the finish point should be 15. In fig.3, it shows that we successfully output the adjacency list file in the correct format. In fig.4, it represents that our BFS function is correct and finally print out the result.
- Now we implement a 10x10 maze. In fig.5, we successfully generates the adjacency list in the correct format. In fig.6, it shows that our maze_solve program can provide the correct solution even with a 10x10 maze.
- As a result, both of our two programs work and can provide with correct and sufficient output.



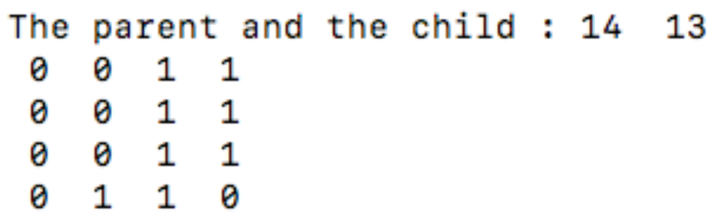
```
Random Number for i & j: 1 2
The Start point : 6
0  0  0  0
0  0  1  0
0  0  0  0
0  0  0  0
```

Fig.1



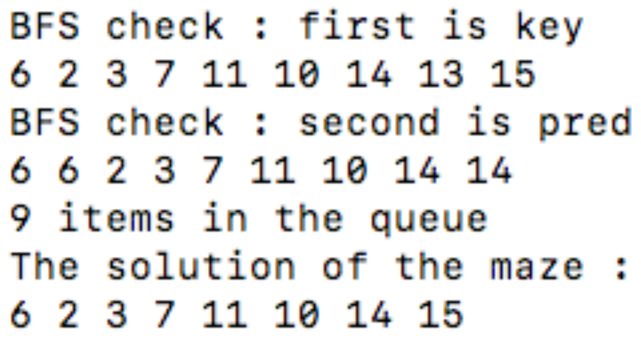
```
maze_adjacency_list.txt
4 4
6 15
1
5 0
6 3
2 7
```

Fig.3



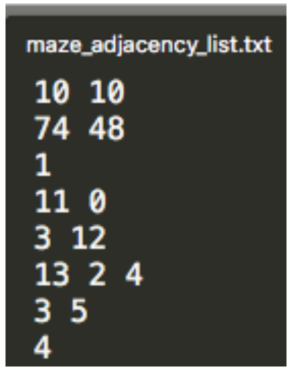
```
The parent and the child : 14  13
 0  0  1  1
 0  0  1  1
 0  0  1  1
 0  1  1  0
```

Fig.2



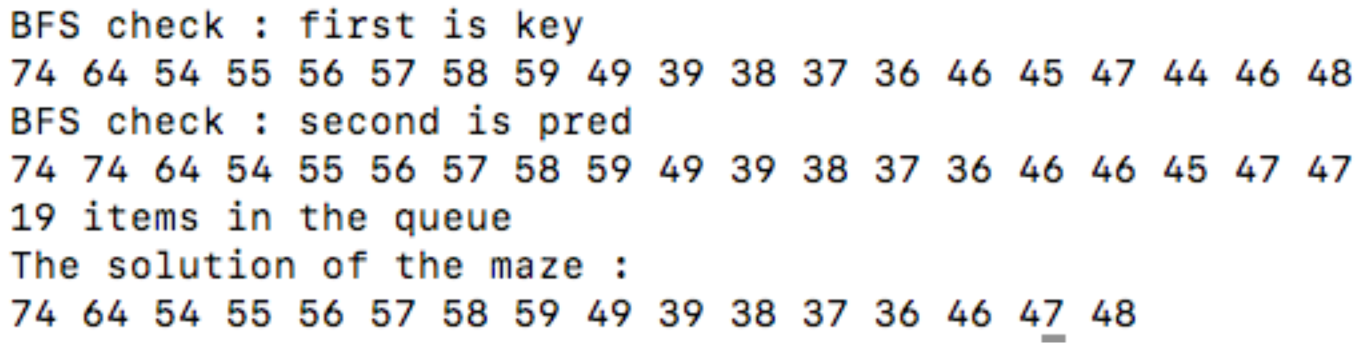
```
BFS check : first is key
6 2 3 7 11 10 14 13 15
BFS check : second is pred
6 6 2 3 7 11 10 14 14
9 items in the queue
The solution of the maze :
6 2 3 7 11 10 14 15
```

Fig.4



```
maze_adjacency_list.txt
10 10
74 48
1
11 0
3 12
13 2 4
3 5
4
```

Fig.5



```
BFS check : first is key
74 64 54 55 56 57 58 59 49 39 38 37 36 46 45 47 44 46 48
BFS check : second is pred
74 74 64 54 55 56 57 58 59 49 39 38 37 36 46 46 45 47 47
19 items in the queue
The solution of the maze :
74 64 54 55 56 57 58 59 49 39 38 37 36 46 47 48
```

Fig.6

Division of Labor

- At first, we developed the maze generation independently. In this process, we both presented other ideas and found out that Ting's way was more efficient and understandable. Therefore, the remaining algorithms were based on the data structures that Ting came up with. After outputting the results, Chin organized the main structure of this assignment and presented it in the slide.
- Ting Liao: DFS, BFS, most algorithms in main().
- Chin Wang: DFS, most functions for constructing queue.