# AU 332 Artificial Intelligence: Principles and Techniques

By: Xiaocheng Wang(517030910326)

HW#: 1

September 23, 2019

## I. INTRODUCTION

Recently we have learnt about some searching algorithms, such as BFS, DFS, UCS and A * search. In this lab, it's required to apply these algorithms to several particular graphs, both in thinking and coding. And the performance also needs consideration, such as their complexity and consistency of heuristic function.

## II. SOLUTIONS

1. Expanding trees:

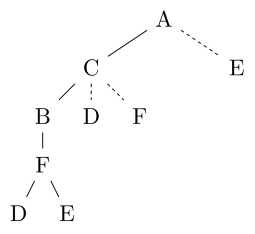


FIG. 1: DFS expanding tree(dashed line means edge in fringe list but node not expanded)

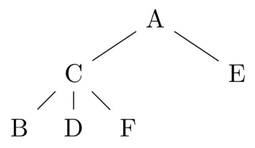


FIG. 2: BFS expanding tree

Complexity:

First talk about time complexity. In the worst case, goal is located at the deepest layer and farest branch, then all nodes are visited and all edges are checked. Therefore, time complexity for both DFS and BFS is equal to number of edges and nodes, which is $O(nd + n) = O(nd)$.

Space complexity differs. In terms of DFS, since the given constraints, nodes number $n$ and maximum degree $d$, can hardly reveal any information about the shape of the graph, I just assume the worst situation occurs, when on each layer of the expanding tree, only one node has child nodes, like the figure below:

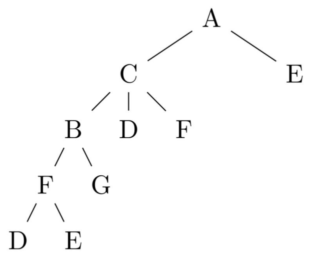


FIG. 3: a worst-case for DFS

Here, when goal node locates at the bottom, almost all edges will be stored, so the complexity is $O(nd)$.

When it comes to BFS, the worst situation is that every layer has only one node except the bottom layer as shown below. Since only last layer has more than one node, the storage requirement is exactly equal to the degree of the node on last but one layer, which must be the maximum degree $d$. So the complexity is $O(d)$.

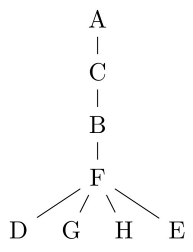


FIG. 4: a worst-case for BFS

Conclusion:

DFS: time: $O(nd)$ space: $O(nd)$

BFS: time: $O(nd)$ space: $O(d)$

2. Uniform cost search takes accumulated step costs into consideration when expanding new nodes. Here is step by step update process of the fringe list and closed list.

   Step1: Fringe: {A-C=3,A-B=10,A-D=20}, Close: {A}

   Step2: Fringe: {C-B=5,A-B=10,C-E=18,A-D=20}, Close: {A,C}

   Step3: Fringe: {A-B=10,B-D=10,C-E=18,A-D=20}, Close: {A,C,B}

   Step4: Fringe: {A-B=10,C-E=18,A-D=20,D-E=21}, Close: {A,C,B,D}

   Step5: Fringe: {C-E=18,A-D=20,D-E=21}, Close: {A,C,B,D}

   Step6: Fringe: {A-D=20,D-E=21}, Close: {A,C,B,D,E}

   The shortest path is A-C-E.

3. In A star search, a heuristic function $h$ is designed, along with accumulated step cost $g$, to calculate $f(n) = g(n) + h(n)$ as the rule of expanding.

   In this problem, an obvious choice of Heuristic function is Manhattan distance, i.e. the sum of the absolute values of differences between $x$s and $y$s: $h(n) = |(x_1 - x_2)| + |(y_1 - y_2)|$

   Belowing maze is labeled by heuristic function.

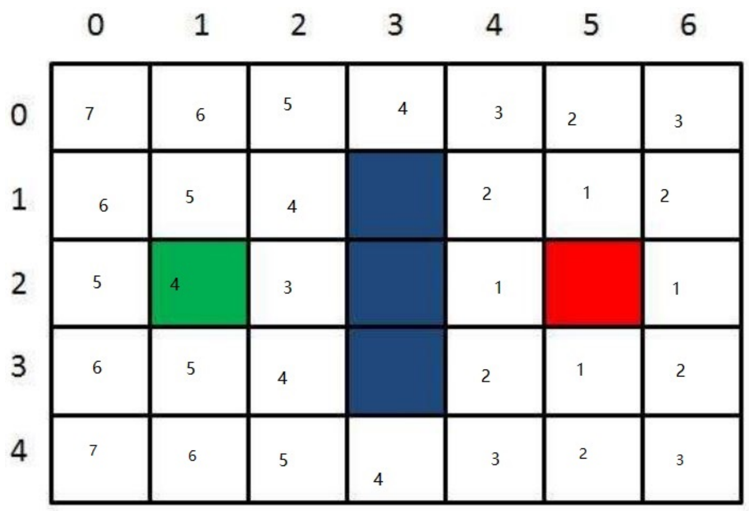
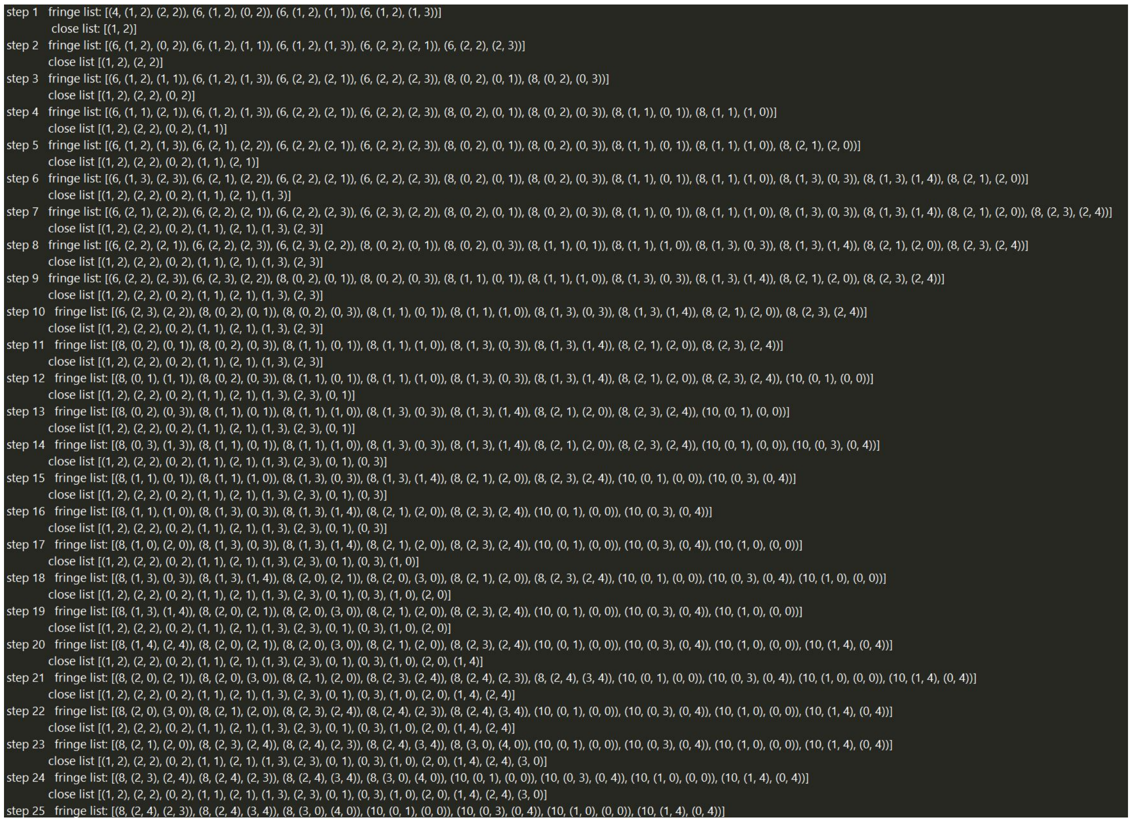
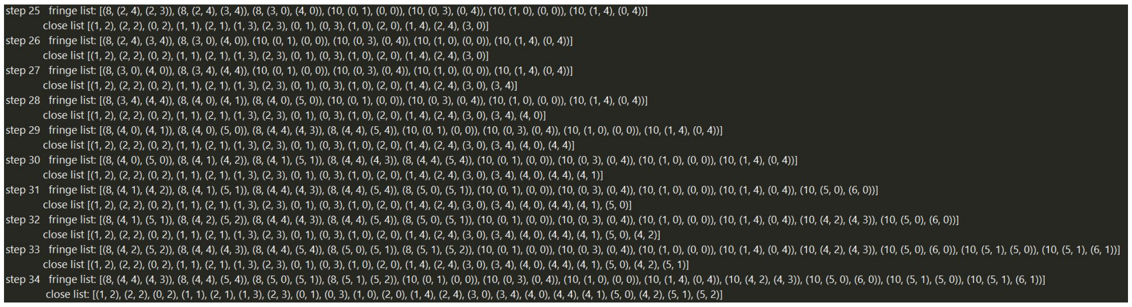


FIG. 5: heuristic on maze

Using A star search algorithm, the complete path finding process on how fringe list and closed list changes is shown below.



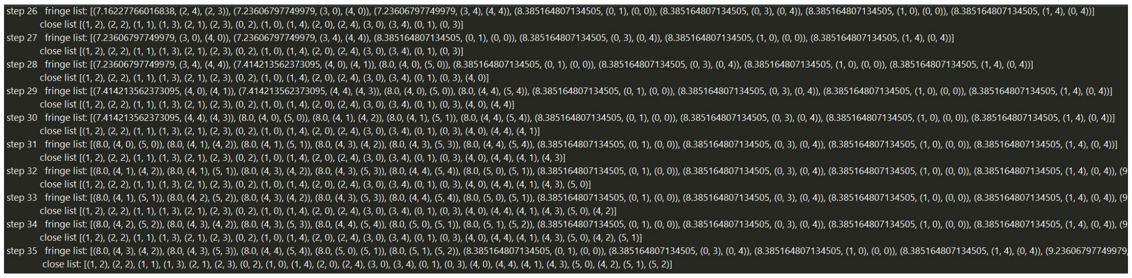
```
step 1   fringe list: [(4, (1, 2), (2, 2)), (6, (1, 2), (0, 2)), (6, (1, 2), (1, 1)), (6, (1, 2), (1, 3))]
          close list: [(1, 2)]
step 2   fringe list: [(6, (1, 2), (0, 2)), (6, (1, 2), (1, 1)), (6, (1, 2), (1, 3)), (6, (2, 2), (2, 1)), (6, (2, 2), (2, 3))]
         close list [(1, 2), (2, 2)]
step 3   fringe list: [(6, (1, 2), (1, 1)), (6, (1, 2), (1, 3)), (6, (2, 2), (2, 1)), (6, (2, 2), (2, 3)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3))]
         close list [(1, 2), (2, 2), (0, 2)]
step 4   fringe list: [(6, (1, 1), (2, 1)), (6, (1, 2), (1, 3)), (6, (2, 2), (2, 1)), (6, (2, 2), (2, 3)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1)]
step 5   fringe list: [(6, (1, 2), (1, 3)), (6, (2, 1), (2, 2)), (6, (2, 2), (2, 1)), (6, (2, 2), (2, 3)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (2, 1), (2, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1)]
step 6   fringe list: [(6, (1, 3), (2, 3)), (6, (2, 1), (2, 2)), (6, (2, 2), (2, 1)), (6, (2, 2), (2, 3)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3)]
step 7   fringe list: [(6, (2, 1), (2, 2)), (6, (2, 2), (2, 1)), (6, (2, 2), (2, 3)), (6, (2, 3), (2, 2)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3)]
step 8   fringe list: [(6, (2, 2), (2, 1)), (6, (2, 2), (2, 3)), (6, (2, 3), (2, 2)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3)]
step 9   fringe list: [(6, (2, 2), (2, 3)), (6, (2, 3), (2, 2)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3)]
step 10   fringe list: [(6, (2, 3), (2, 2)), (8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3)]
step 11   fringe list: [(8, (0, 2), (0, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3)]
step 12   fringe list: [(8, (0, 1), (1, 1)), (8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1)]
step 13   fringe list: [(8, (0, 2), (0, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1)]
step 14   fringe list: [(8, (0, 3), (1, 3)), (8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3)]
step 15   fringe list: [(8, (1, 1), (0, 1)), (8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3)]
step 16   fringe list: [(8, (1, 1), (1, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3)]
step 17   fringe list: [(8, (1, 0), (2, 0)), (8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0)]
step 18   fringe list: [(8, (1, 3), (0, 3)), (8, (1, 3), (1, 4)), (8, (2, 0), (2, 1)), (8, (2, 0), (3, 0)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0)]
step 19   fringe list: [(8, (1, 3), (1, 4)), (8, (2, 0), (2, 1)), (8, (2, 0), (3, 0)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0)]
step 20   fringe list: [(8, (1, 4), (2, 4)), (8, (2, 0), (2, 1)), (8, (2, 0), (3, 0)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4)]
step 21   fringe list: [(8, (2, 0), (2, 1)), (8, (2, 0), (3, 0)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (8, (2, 4), (2, 3)), (8, (2, 4), (3, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4)]
step 22   fringe list: [(8, (2, 0), (3, 0)), (8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (8, (2, 4), (2, 3)), (8, (2, 4), (3, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4)]
step 23   fringe list: [(8, (2, 1), (2, 0)), (8, (2, 3), (2, 4)), (8, (2, 4), (2, 3)), (8, (2, 4), (3, 4)), (8, (3, 0), (4, 0)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0)]
step 24   fringe list: [(8, (2, 3), (2, 4)), (8, (2, 4), (2, 3)), (8, (2, 4), (3, 4)), (8, (3, 0), (4, 0)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
         close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0)]
step 25   fringe list: [(8, (2, 4), (2, 3)), (8, (2, 4), (3, 4)), (8, (3, 0), (4, 0)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
```



```
step 25   fringe list: [(8, (2, 4), (2, 3)), (8, (2, 4), (3, 4)), (8, (3, 0), (4, 0)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0)]
step 26   fringe list: [(8, (2, 4), (3, 4)), (8, (3, 0), (4, 0)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0)]
step 27   fringe list: [(8, (3, 0), (4, 0)), (8, (3, 4), (4, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4)]
step 28   fringe list: [(8, (3, 4), (4, 4)), (8, (4, 0), (4, 1)), (8, (4, 0), (5, 0)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4), (4, 0)]
step 29   fringe list: [(8, (4, 0), (4, 1)), (8, (4, 0), (5, 0)), (8, (4, 4), (4, 3)), (8, (4, 4), (5, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4), (4, 0), (4, 4)]
step 30   fringe list: [(8, (4, 0), (5, 0)), (8, (4, 1), (4, 2)), (8, (4, 1), (5, 1)), (8, (4, 4), (4, 3)), (8, (4, 4), (5, 4)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4), (4, 0), (4, 4), (4, 1)]
step 31   fringe list: [(8, (4, 1), (4, 2)), (8, (4, 1), (5, 1)), (8, (4, 4), (4, 3)), (8, (4, 4), (5, 4)), (8, (5, 0), (5, 1)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4)), (10, (5, 0), (6, 0))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4), (4, 0), (4, 4), (4, 1), (5, 0)]
step 32   fringe list: [(8, (4, 1), (5, 1)), (8, (4, 2), (5, 2)), (8, (4, 4), (4, 3)), (8, (4, 4), (5, 4)), (8, (5, 0), (5, 1)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4)), (10, (4, 2), (4, 3)), (10, (5, 0), (6, 0))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4), (4, 0), (4, 4), (4, 1), (5, 0), (4, 2)]
step 33   fringe list: [(8, (4, 2), (5, 2)), (8, (4, 4), (4, 3)), (8, (4, 4), (5, 4)), (8, (5, 0), (5, 1)), (8, (5, 1), (5, 2)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4)), (10, (4, 2), (4, 3)), (10, (5, 0), (6, 0)), (10, (5, 1), (5, 0)), (10, (5, 1), (6, 1))]
          close list [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4), (4, 0), (4, 4), (4, 1), (5, 0), (4, 2), (5, 1)]
step 34   fringe list: [(8, (4, 4), (4, 3)), (8, (4, 4), (5, 4)), (8, (5, 0), (5, 1)), (8, (5, 1), (5, 2)), (10, (0, 1), (0, 0)), (10, (0, 3), (0, 4)), (10, (1, 0), (0, 0)), (10, (1, 4), (0, 4)), (10, (4, 2), (4, 3)), (10, (5, 0), (6, 0)), (10, (5, 1), (5, 0)), (10, (5, 1), (6, 1))]
           close list: [(1, 2), (2, 2), (0, 2), (1, 1), (2, 1), (1, 3), (2, 3), (0, 1), (0, 3), (1, 0), (2, 0), (1, 4), (2, 4), (3, 0), (3, 4), (4, 0), (4, 4), (4, 1), (5, 0), (4, 2), (5, 1), (5, 2)]
```

FIG. 6: complete path

Actually I also have tried Euclidean Distance as the huristic function, which takes one more step to reach the the goal.



```
step 26   fringe list: [(7.16227766016838, (2, 4), (2, 3)), (7.23606797749979, (3, 0), (4, 0)), (7.23606797749979, (3, 4), (4, 4)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3)]
step 27   fringe list: [(7.23606797749979, (3, 0), (4, 0)), (7.23606797749979, (3, 4), (4, 4)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3)]
step 28   fringe list: [(7.23606797749979, (3, 4), (4, 4)), (7.414213562373095, (4, 0), (4, 1)), (8.0, (4, 0), (5, 0)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0)]
step 29   fringe list: [(7.414213562373095, (4, 0), (4, 1)), (7.414213562373095, (4, 4), (4, 3)), (8.0, (4, 0), (5, 0)), (8.0, (4, 4), (5, 4)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0), (4, 4)]
step 30   fringe list: [(7.414213562373095, (4, 4), (4, 3)), (8.0, (4, 0), (5, 0)), (8.0, (4, 1), (4, 2)), (8.0, (4, 1), (5, 1)), (8.0, (4, 4), (5, 4)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0), (4, 4), (4, 1)]
step 31   fringe list: [(8.0, (4, 0), (5, 0)), (8.0, (4, 1), (4, 2)), (8.0, (4, 1), (5, 1)), (8.0, (4, 3), (4, 2)), (8.0, (4, 3), (5, 3)), (8.0, (4, 4), (5, 4)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4))]
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0), (4, 4), (4, 1), (4, 3)]
step 32   fringe list: [(8.0, (4, 1), (4, 2)), (8.0, (4, 1), (5, 1)), (8.0, (4, 3), (4, 2)), (8.0, (4, 3), (5, 3)), (8.0, (4, 4), (5, 4)), (8.0, (5, 0), (5, 1)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4)), (9
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0), (4, 4), (4, 1), (4, 3), (5, 0)]
step 33   fringe list: [(8.0, (4, 1), (5, 1)), (8.0, (4, 2), (5, 2)), (8.0, (4, 3), (4, 2)), (8.0, (4, 3), (5, 3)), (8.0, (4, 4), (5, 4)), (8.0, (5, 0), (5, 1)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4)), (9
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0), (4, 4), (4, 1), (4, 3), (5, 0), (4, 2)]
step 34   fringe list: [(8.0, (4, 2), (5, 2)), (8.0, (4, 3), (4, 2)), (8.0, (4, 3), (5, 3)), (8.0, (4, 4), (5, 4)), (8.0, (5, 0), (5, 1)), (8.0, (5, 1), (5, 2)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4)), (9
          close list [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0), (4, 4), (4, 1), (4, 3), (5, 0), (4, 2), (5, 1)]
step 35   fringe list: [(8.0, (4, 3), (4, 2)), (8.0, (4, 3), (5, 3)), (8.0, (4, 4), (5, 4)), (8.0, (5, 0), (5, 1)), (8.0, (5, 1), (5, 2)), (8.385164807134505, (0, 1), (0, 0)), (8.385164807134505, (0, 3), (0, 4)), (8.385164807134505, (1, 0), (0, 0)), (8.385164807134505, (1, 4), (0, 4)), (9.23606797749979,
           close list: [(1, 2), (2, 2), (1, 1), (1, 3), (2, 1), (2, 3), (0, 2), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4), (0, 1), (0, 3), (4, 0), (4, 4), (4, 1), (4, 3), (5, 0), (4, 2), (5, 1), (5, 2)]
```

FIG. 7: another heuristic

4. To realize BFS, DFS, UCS, and A*S. Just use different data structure to store edges when expanding new nodes. BFS uses queue, DFS uses stack (in python, it's LifoQueue). UCS and A*S both use PriorityQueue, but the stored costs are differed by whether containing heuristic function.

5. (a) To check whether start and end nodes in graph, since the storage form is like a linked list, I have to traverse keys and values of the dictionary storing edges. Here I ignore the case when a node is isolated.

```python
flag = [0]*2
if start in graph.edges.keys():
    flag[0] = 1
if goal in graph.edges.keys():
    flag[1] = 1
for e in graph.edges.values():
    if start in e:
        flag[0] = 1
    if goal in e:
        flag[1] = 1
if 0 in flag:
    if flag.count(0) == 2:
        print("Start and goal not included in graph, please check your setting.")
    else:
        print("%s not included in graph, please check your setting."%["Start","
                                                    Goal"][flag.index(0)])
    return None
```

(b) To verify consistency of heuristic, use the triangle inequality relation $h(n) \leq c(n, a, n') + h(n')$ on every nodes.

```
def consistency_verification(graph, heuristic, goal):
  for current_node in graph.edges:
        for kid in graph.neighbors(current_node):
            if heuristic(graph, current_node, goal) > graph.get_cost(current_node,
                                                        kid) + heuristic(graph,
                                                        kid, goal):
                print("this heuristic is not consistent!")
                return False
  return True
```

## III. DISCUSSION & CONCLUSION

In this lab, I applied BFS, DFS, UCS, A*S to several graphs, did some search on them step by step, calculated the complexity of BFS and DFS, write codes in python to realize these algorithms on computer, and did some verification work on whether nodes included and the consistency of heuristic. All of this improve my ability in these searching strategies.