457A Assignment 1

Problem 1:

Initial state: the initial state is all 64 discs stacked correctly on one pole, say the left most pole.

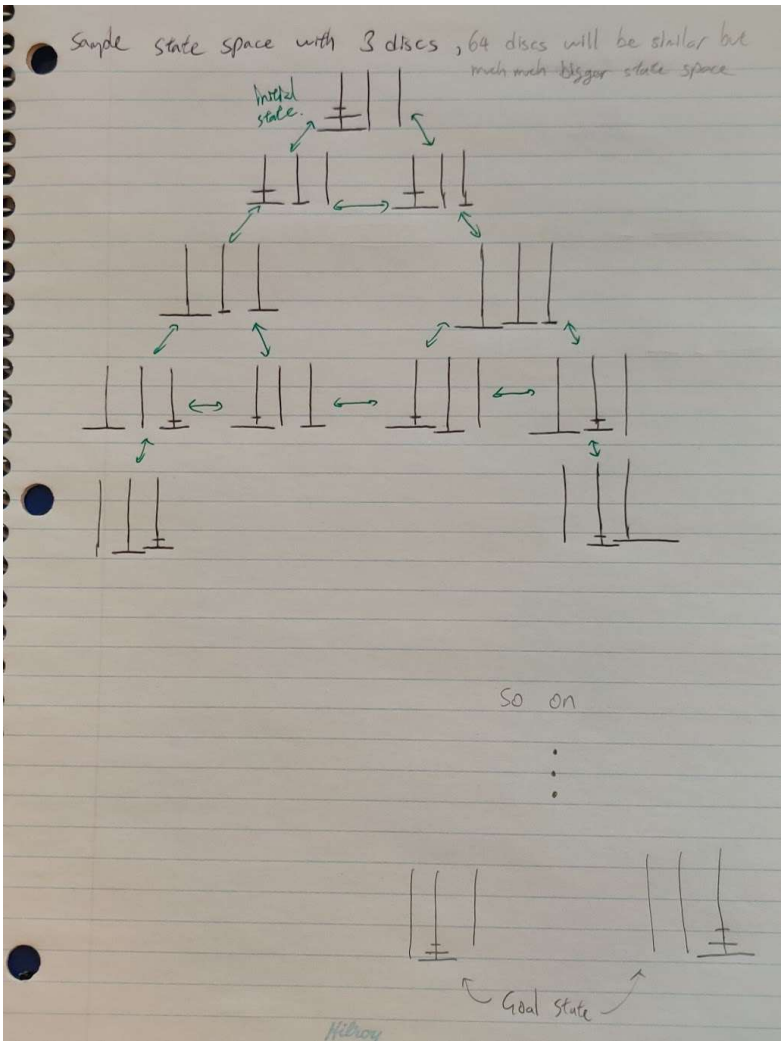
Goal state: the goal state is to have all 64 discs stacked correctly on one of the three poles that is not the starting pole.

Restrictions imposed by problem: A disc may be stacked only on top of a larger disc or onto an empty pole.

States: arrangement of 64 discs on the 3 poles

Possible actions: move a disc onto an empty pole or move a disc onto a pole where the disc beneath is larger.

State space diagram and sequence of actions that takes from initial state to goal state:



Problem 2:

Initial state: Using *a*, and *b* to represent the amount of water in the 4-gallon jug and 3-gallon jug, respectively. Where *0 <= a <= 4 and 0 <= b <= 3*
The initial state is *a = 0, b = 0*

Goal state: to get values a, b where b = 2, *0 <= a <= 4*

Restrictions imposed by problem: Neither jug has measuring marks on it, water can be poured between jugs or onto the ground.

States:

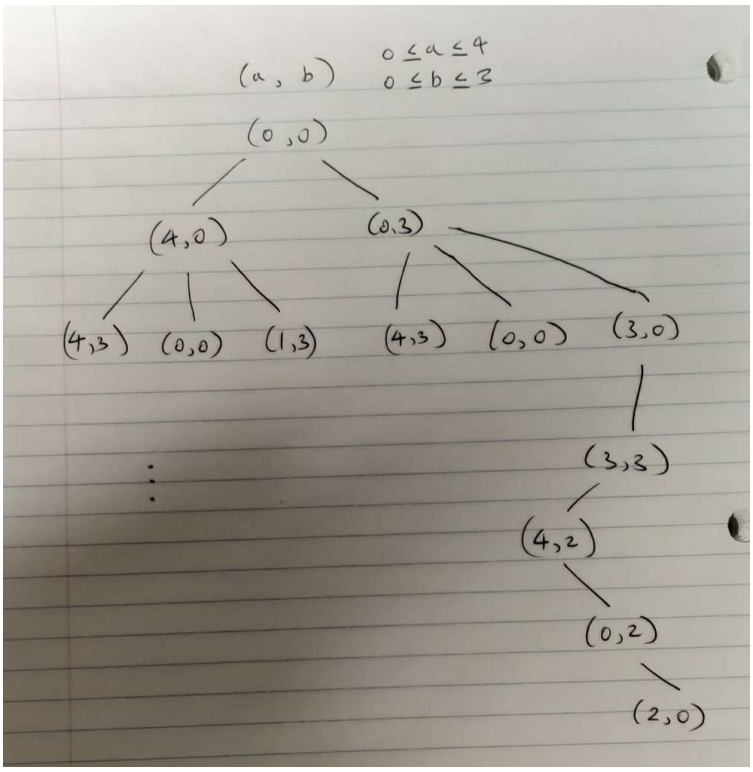
0 <= a <= 4 and 0 <= b <= 3,

a and b are whole numbers, where a represent the 4-Gallon jug and b the 3-Gallon jug.

Possible actions:

- Fill 4-G jug
- Fill 3-G jug
- Pour all of 4-gallon jug into 3
- Pour 3 into 4-gallon jug
- Empty 4-G jug
- Empty 3-G jug

State space diagram and sequence of actions that takes from initial state to goal state:



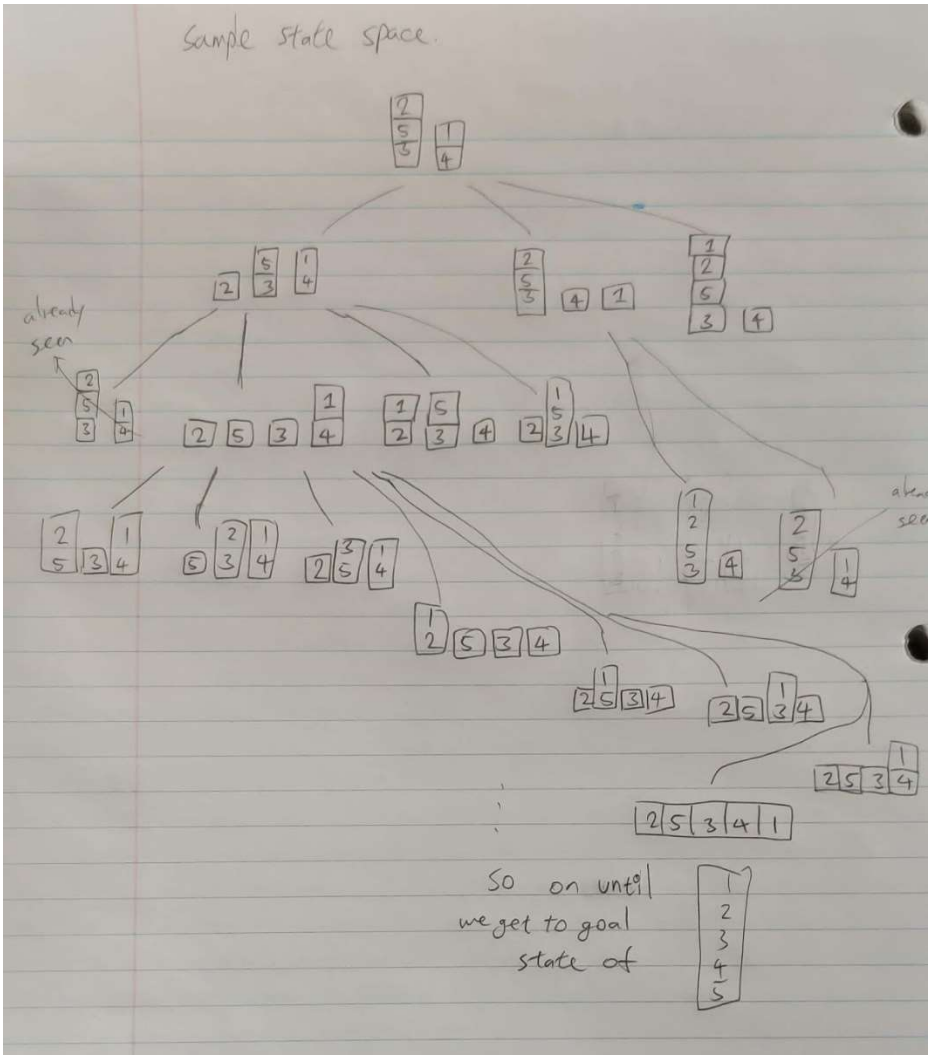
Problem 3:

Goal state: to arrange blocks in a single pile in ascending order.

Restrictions imposed by problem: only blocks at the top of the current stack can be moved, either to the table or atop another pile

States: n piles of blocks where 1 <= n < = 5 and any distribution of 5 blocks within the piles.

State space diagram and sequence of actions that takes from initial state to goal state:



A: problem formulation: Blind search algorithm

Initial state: initial configuration of 5 blocks in several piles *n*, *1 <= n <= 5*

Possible actions: move one block at a time to the table or atop another pile of blocks.

Goal Test: Check if a stack of 5 in ascending order (top to down) exist

Cost: Each block-action account for 1 cost

//Psuedo code:

```
//assume node has access to blocks a &b
Stack_op (node)
        put block a on b
        if (a < b & VisitedStates doesn't contain new formation)
                update current state

PutDown_op (node)
        put block a on table
        If (VisitedStates doesn't contain new formation)
                update current state

Set initialState;
Set goalState;
Var currentState = []
VisitedStates = {}

Queue of operations= []
Queue.add (blocks at top of each stack)
visitedState
```

BFS:

```
While (queue !empty) {
        Curr = queue poll()
        Update currentState(Curr):
        visitedState add(currentState)

        If(currentState = goalState)
                Return
        If (visitedState.contains(currentState))
                Continue

        //add block from the top of each pile after state update (the 'neighbor nodes')
        Queue.add(Stack_op(Curr))
        Queue.add(PutDown_op(Curr))
}
```

DFS:

```
Traverse (Operation,  node) {
        Update currentState with (node)
        If (VisitedStates.contains(currentState))
                return
        Traverse(Stack_op(node))
        Traverse(PutDown_op(node))
}
```

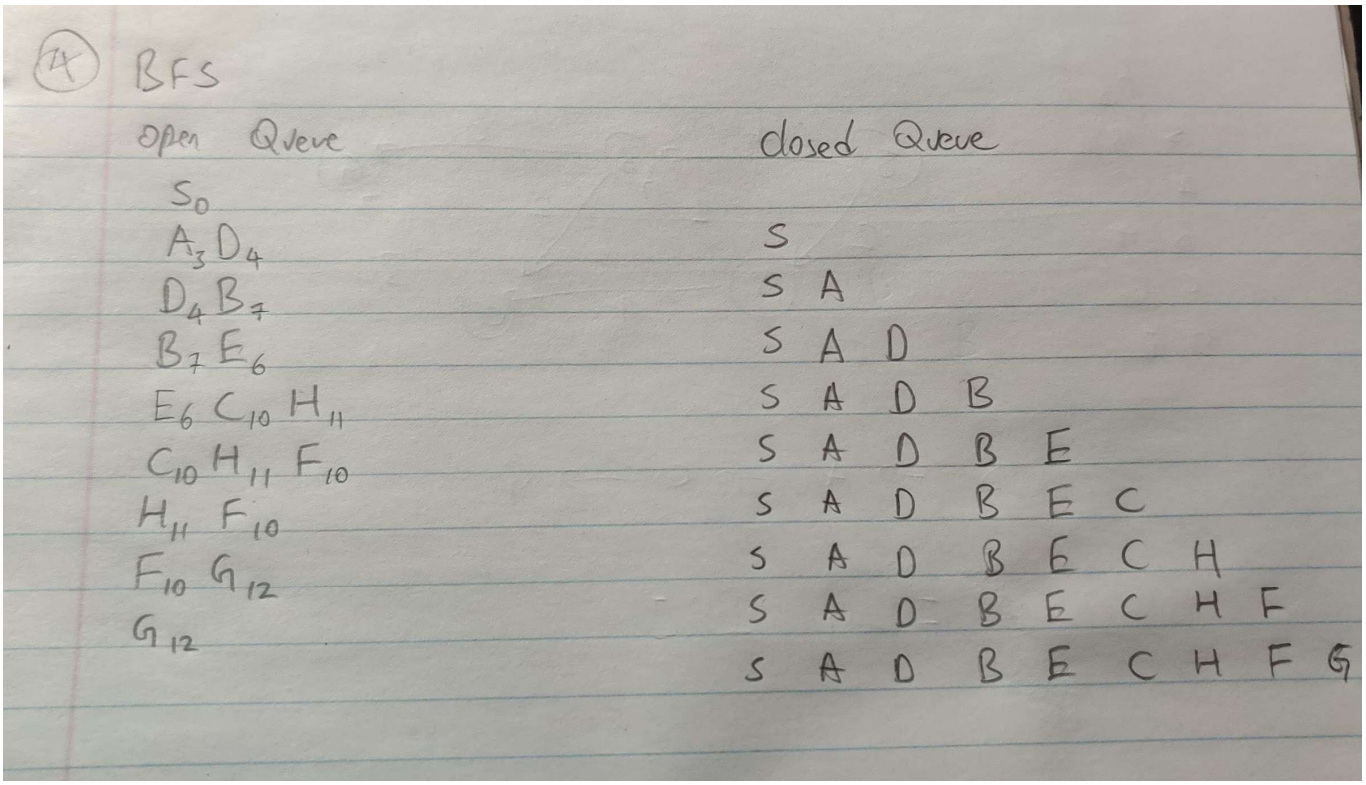
BFS starts at the root (initial state) and traverses all nodes along the width of the graph, or level order traversal. It stops when all nodes have been visited, and doesn't consider the best way to the result.
DFS starts at the root and traverses nodes along with the depth of the graph. It searches a branch as deep as possible. When an edge is explored, the search traces back to the start of the edge of that node. DFS continues until all nodes have been visited.

Problem 4:

BFS:

Sequence of nodes visited: S, A, D, B, E, C, H, F, G

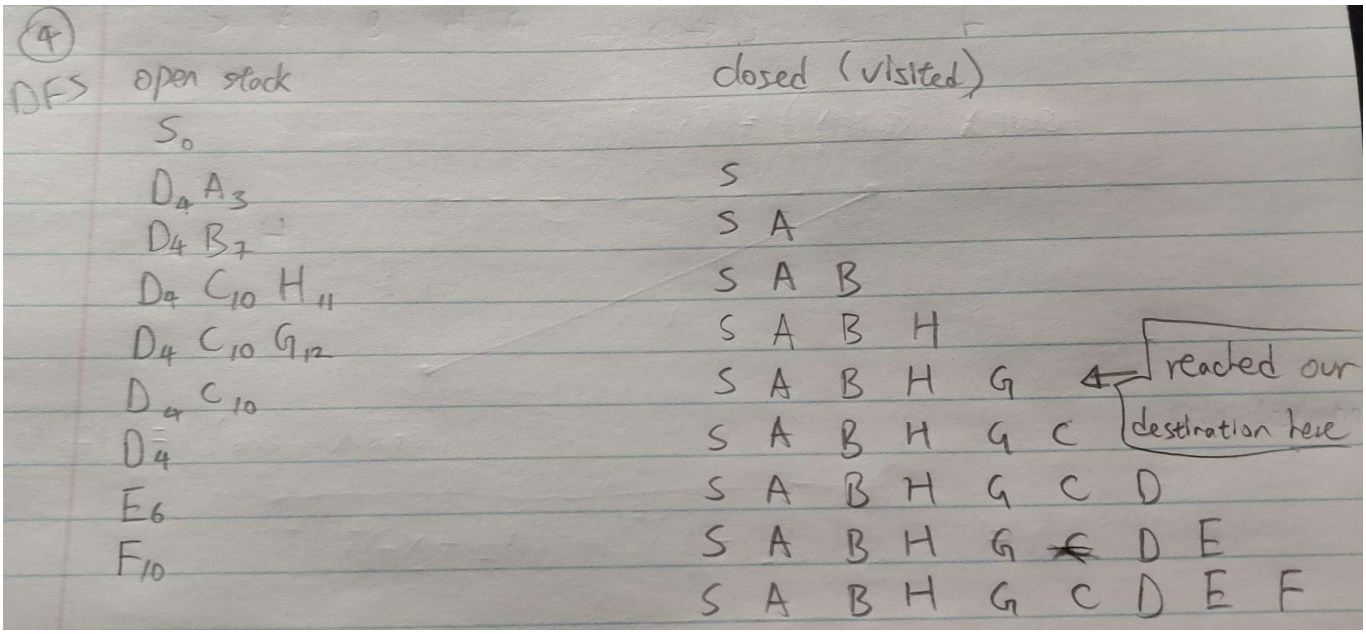
Path returned: S A B H G (cost of 12)



DFS:

Sequence of nodes visited: S, A, B, H, G, C, D, E, F

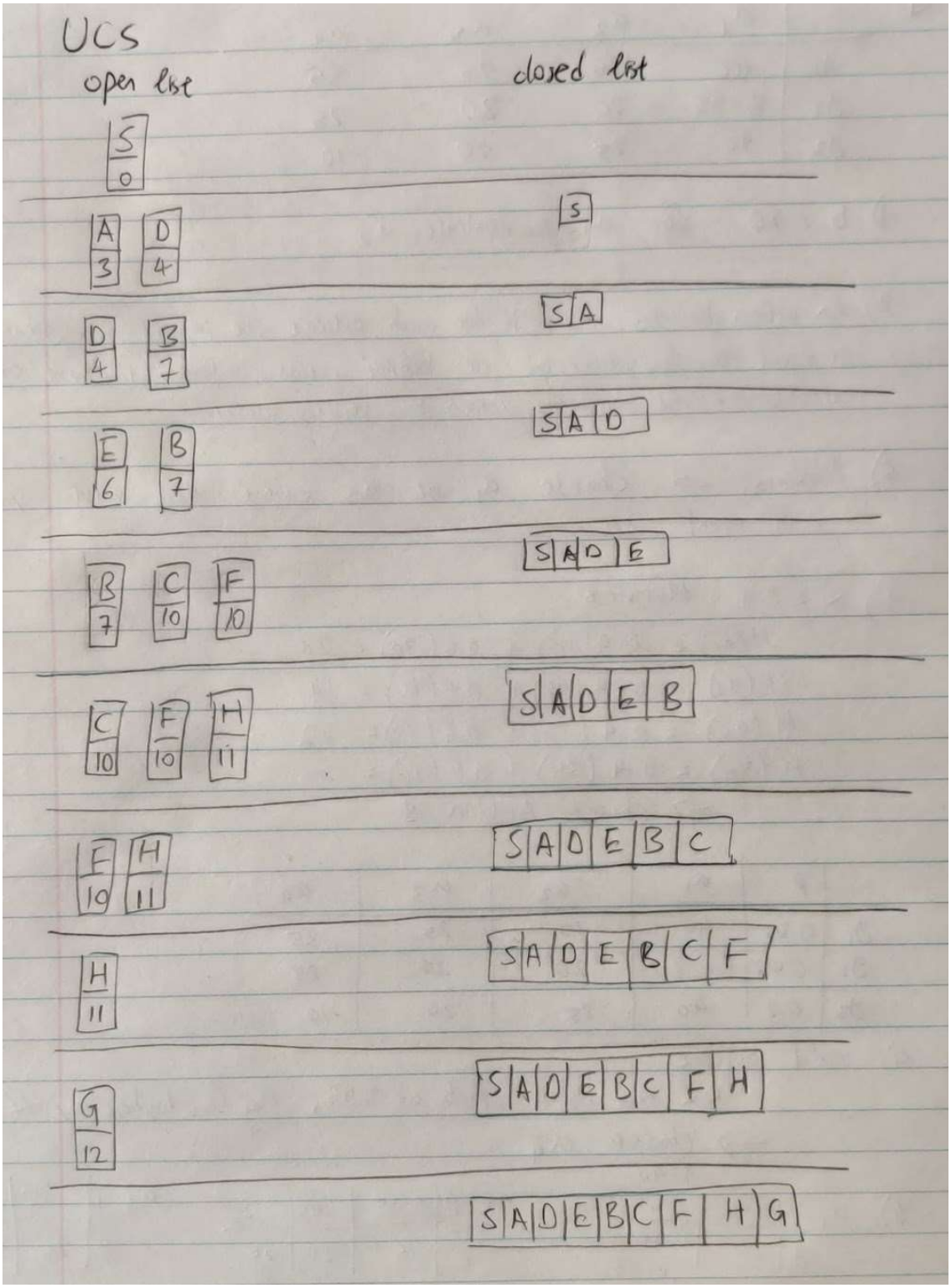
Path returned: S,A,B,H,G (cost of 12)



UCS:

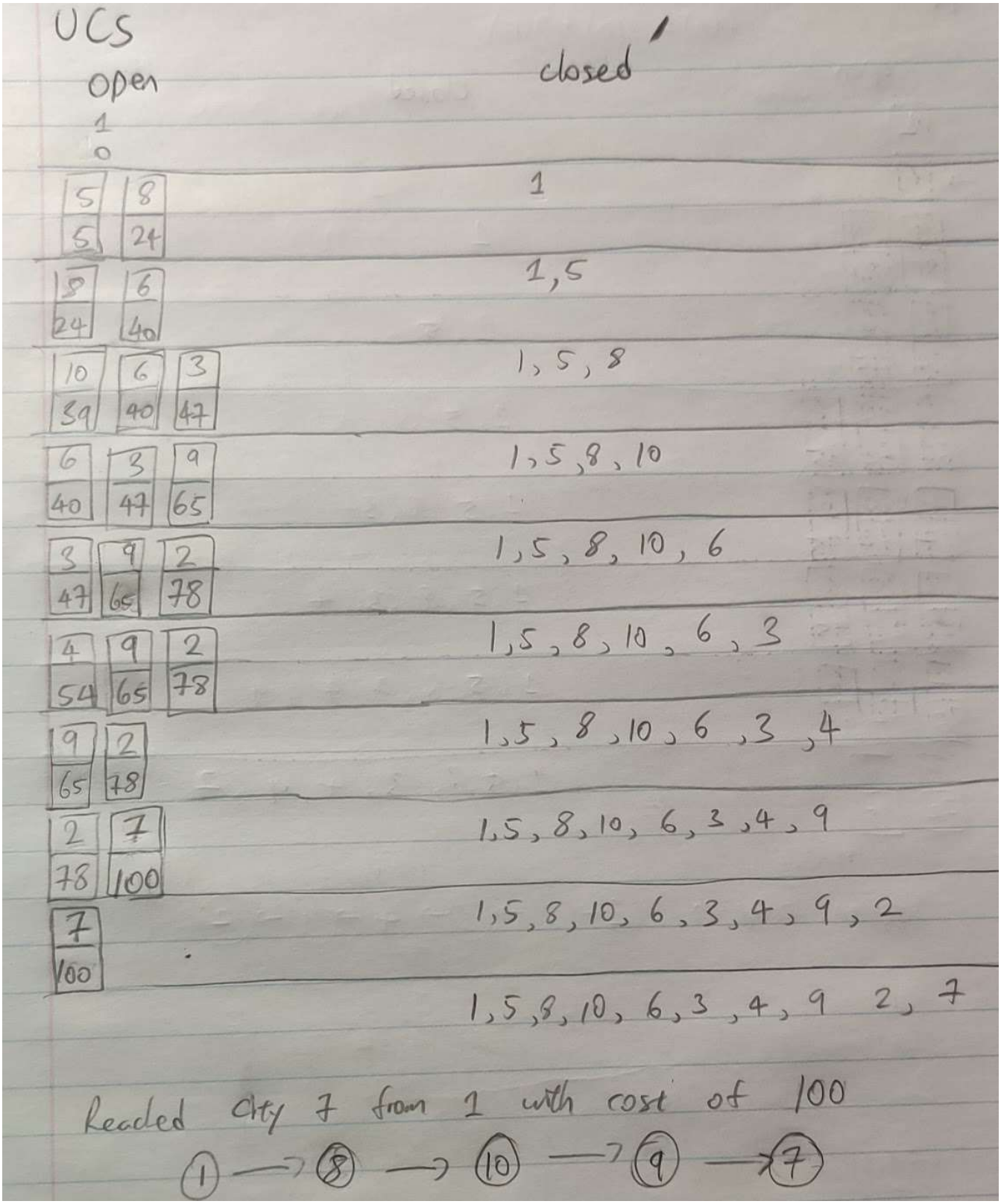
Sequence of nodes visited: S, A, D, E, B, C, F, H, G

Path returned: S,A,B,H,G (cost of 12)

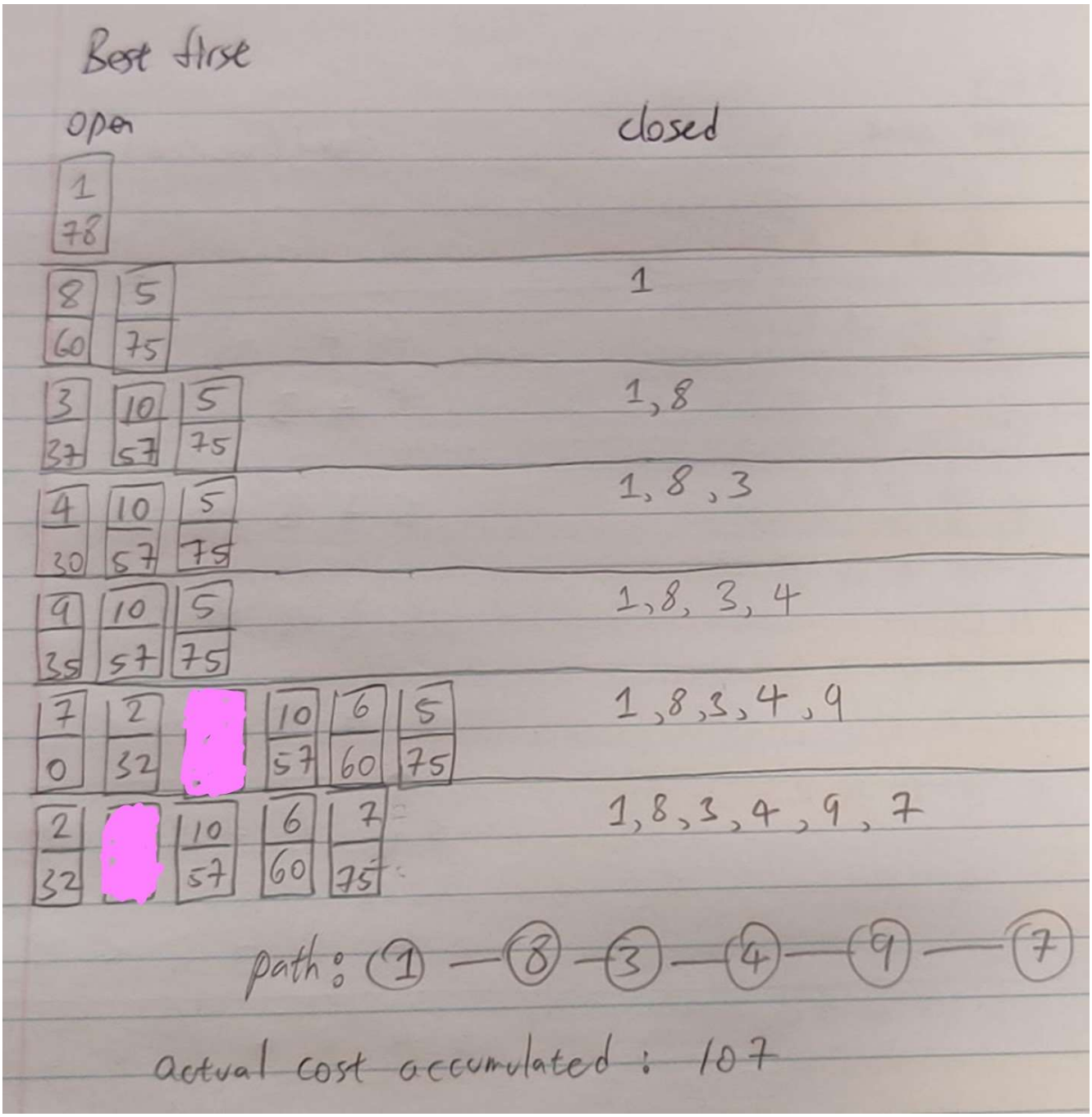


Problem 5:

UCS:

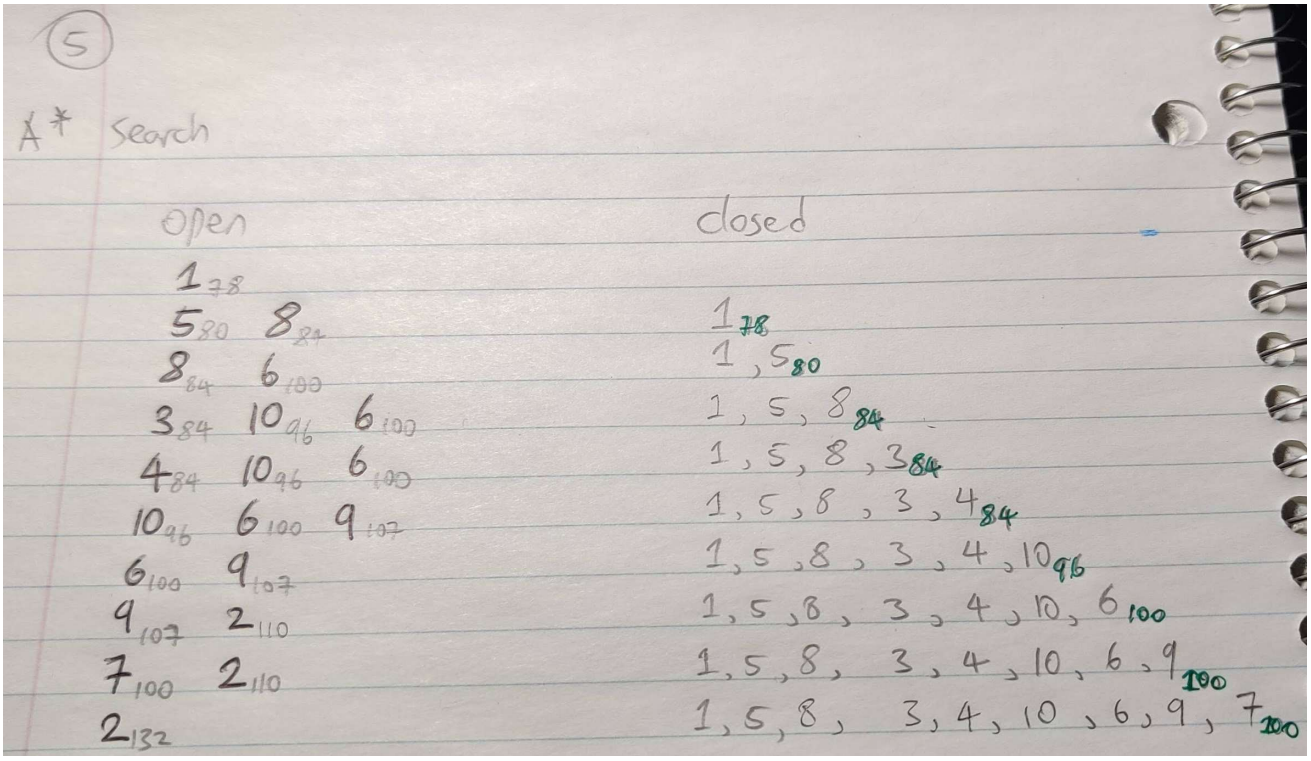


Best first Search:



A* Search:

Path: 1, 8, 10, 9, 7 (distance of 100)



Problem 6:

BFS code:

```
public class BFS {
    public int[][] directions= {{0,1},{0,-1},{1,0},{-1,0}};
    public int BFSsearch (int[][] grid, int startI, int startJ, int endI, int
endJ, List<int[]> pathList) {
        int nodesExplored = 0;

        Set<String> visitedSet = new HashSet();
        // queue that holds next x and y pairs:
        Queue<int[]> queue = new LinkedList();
        queue.offer(new int[] {startI, startJ});

        while (!queue.isEmpty()) {
            int[] currCoord = queue.poll();
            int i = currCoord[0], j = currCoord[1];
            String currCords = i+"~"+j;

            if (visitedSet.contains(currCords))
                continue;

            //reached end:
            if (i == endI && j == endJ) {
                System.out.println("found target after exploring "+nodesEx-
plored+ " nodes");
            }

            //check out of bounds or if it's wall
            if (i < 0 || j < 0 || i >= grid.length || j >= grid[i].length ||
grid[i][j] != 0)
                continue;

            visitedSet.add(currCords);
            nodesExplored++;
            pathList.add(new int[] {i, j});

            for(int k = 0; k < directions.length; k++) {
                int r = i + directions[k][0], c = j + directions[k][1];
                queue.offer(new int[]{r, c});
            }
        }
        return nodesExplored;
    }
}
```

Results of running BFS (order of node coordinates visited):

S -> E1

```
found target after exploring 428 nodes

 finished BFS to E1, total nodes explored: 447
[13, 2][13, 3][13, 1][14, 2][13, 4][14, 3][13, 0][14, 1][12, 1][15, 2][13, 5][14, 4][15, 3][14, 0][12, 0][15, 1][11, 1][16, 2]
[13, 6][14, 5][15, 4][16, 3][15, 0][11, 0][16, 1][10, 1][17, 2][13, 7][14, 6][15, 5][16, 4][17, 3][16, 0][10, 0][17, 1][9, 1]
[13, 8][14, 7][15, 6][16, 5][17, 4][17, 0][9, 0][9, 2][8, 1][13, 9][14, 8][12, 8][15, 7][16, 6][17, 5][18, 0][8, 0][9, 3][8, 2]
[7, 1][13, 10][14, 9][12, 9][15, 8][11, 8][16, 7][17, 6][19, 0][7, 0][9, 4][8, 3][7, 2][6, 1][13, 11][14, 10][12, 10][15, 9]
[11, 9][16, 8][10, 8][17, 7][20, 0][6, 0][9, 5][8, 4][6, 2][5, 1][13, 12][14, 11][12, 11][11, 10][16, 9][10, 9][17, 8][9, 8]
[20, 1][21, 0][5, 0][9, 6][8, 5][5, 2][4, 1][13, 13][14, 12][12, 12][11, 11][10, 10][17, 9][9, 9][9, 7][8, 8][20, 2][21, 1][22, 0]
[4, 0][8, 6][5, 3][4, 2][3, 1][13, 14][14, 13][12, 13][11, 12][10, 11][9, 10][17, 10][8, 9][8, 7][7, 8][20, 3][21, 2][22, 1][23, 0]
[3, 0][5, 4][4, 3][3, 2][2, 1][13, 15][14, 14][11, 13][9, 11][8, 10][17, 11][7, 9][6, 8][20, 4][21, 3][22, 2][23, 1][24, 0][2, 0]
[5, 5][4, 4][3, 3][2, 2][1, 1][13, 16][14, 15][12, 15][9, 12][8, 11][7, 10][17, 12][6, 9][5, 8][21, 4][22, 3][23, 2][24, 1][1, 0]
[5, 6][4, 5][3, 4][2, 3][1, 2][0, 1][13, 17][14, 16][12, 16][15, 15][11, 15][9, 13][8, 12][7, 11][6, 10][17, 13][5, 9][5, 7][22, 4]
[23, 3][24, 2][0, 0][4, 6][3, 5][1, 3][0, 2][13, 18][14, 17][12, 17][15, 16][11, 16][16, 15][8, 13][7, 12][6, 11][5, 10][17, 14]
[4, 9][22, 5][23, 4][24, 3][3, 6][0, 3][13, 19][12, 18][15, 17][11, 17][16, 16][17, 15][7, 13][6, 12][5, 11][4, 10][18, 14][3, 9]
[23, 5][24, 4][13, 20][14, 19][12, 19][11, 18][16, 17][10, 17][17, 16][18, 15][7, 14][6, 13][5, 12][3, 10][19, 14][2, 9][24, 5]
[13, 21][12, 20][15, 19][11, 19][17, 17][9, 17][18, 16][19, 15][7, 15][6, 14][5, 13][4, 12][3, 11][2, 10][19, 13][20, 14][1, 9]
[24, 6][13, 22][12, 21][11, 20][16, 19][18, 17][9, 16][19, 16][20, 15][7, 16][8, 15][6, 15][5, 14][4, 13][3, 12][2, 11][1, 10]
[19, 12][20, 13][21, 14][1, 8][0, 9][24, 7][13, 23][12, 22][11, 21][17, 19][19, 17][9, 15][8, 16][20, 16][21, 15][7, 17][6, 16]
[5, 15][4, 14][3, 13][2, 12][1, 11][0, 10][19, 11][20, 12][1, 7][0, 8][24, 8][23, 7][13, 24][14, 23][12, 23][11, 22][10, 21][18, 19]
[20, 17][21, 16][22, 15][7, 18][5, 16][4, 15][3, 14][2, 13][1, 12][0, 11][19, 10][20, 11][1, 6][0, 7][24, 9][23, 8][22, 7][14, 24]
[12, 24][15, 23][11, 23][10, 22][9, 21][19, 19][21, 17][22, 16][7, 19][6, 18][4, 16][3, 15][2, 14][1, 13][0, 12][19, 9][20, 10]
[0, 6][24, 10][23, 9][22, 8][21, 7][15, 24][11, 24][16, 23][10, 23][9, 22][20, 19][21, 18][22, 17][23, 16][7, 20][6, 19][5, 18]
[3, 16][2, 15][1, 14][0, 13][19, 8][20, 9][24, 11][23, 10][22, 9][20, 7][16, 24][10, 24][17, 23][21, 19][22, 18][23, 17][7, 21]
[6, 20][5, 19][4, 18][2, 16][1, 15][0, 14][19, 7][20, 8][23, 11][22, 10][9, 24][18, 23][21, 20][22, 19][23, 18][24, 17][7, 22][6, 21]
[4, 19][3, 18][1, 16][0, 15][23, 12][22, 11][8, 24][19, 23][21, 21][22, 20][23, 19][24, 18][6, 22][3, 19][2, 18][0, 16][22, 12]
[7, 24][19, 24][20, 23][21, 22][22, 21][23, 20][24, 19][5, 22][2, 19][1, 18][6, 24][20, 24][21, 23][22, 22][23, 21][24, 20][2, 20]
[1, 19][0, 18][21, 24][22, 23][23, 22][24, 21][2, 21][1, 20][0, 19][22, 24][23, 23][24, 22][1, 21][0, 20][23, 24][24, 23][0, 21][24, 24]
```

S -> E2

```
found target after exploring 132 nodes

 finished BFS to E2, total nodes explored: 447
[13, 2][13, 3][13, 1][14, 2][13, 4][14, 3][13, 0][14, 1][12, 1][15, 2][13, 5][14, 4][15, 3][14, 0][12, 0][15, 1][11, 1][16, 2]
[13, 6][14, 5][15, 4][16, 3][15, 0][11, 0][16, 1][10, 1][17, 2][13, 7][14, 6][15, 5][16, 4][17, 3][16, 0][10, 0][17, 1][9, 1]
[13, 8][14, 7][15, 6][16, 5][17, 4][17, 0][9, 0][9, 2][8, 1][13, 9][14, 8][12, 8][15, 7][16, 6][17, 5][18, 0][8, 0][9, 3][8, 2]
[7, 1][13, 10][14, 9][12, 9][15, 8][11, 8][16, 7][17, 6][19, 0][7, 0][9, 4][8, 3][7, 2][6, 1][13, 11][14, 10][12, 10][15, 9][11, 9]
[16, 8][10, 8][17, 7][20, 0][6, 0][9, 5][8, 4][6, 2][5, 1][13, 12][14, 11][12, 11][11, 10][16, 9][10, 9][17, 8][9, 8][20, 1][21, 0]
[5, 0][9, 6][8, 5][5, 2][4, 1][13, 13][14, 12][12, 12][11, 11][10, 10][17, 9][9, 9][9, 7][8, 8][20, 2][21, 1][22, 0][4, 0][8, 6]
[5, 3][4, 2][3, 1][13, 14][14, 13][12, 13][11, 12][10, 11][9, 10][17, 10][8, 9][8, 7][7, 8][20, 3][21, 2][22, 1][23, 0][3, 0][5, 4]
[4, 3][3, 2][2, 1][13, 15][14, 14][11, 13][9, 11][8, 10][17, 11][7, 9][6, 8][20, 4][21, 3][22, 2][23, 1][24, 0][2, 0][5, 5][4, 4]
[3, 3][2, 2][1, 1][13, 16][14, 15][12, 15][9, 12][8, 11][7, 10][17, 12][6, 9][5, 8][21, 4][22, 3][23, 2][24, 1][1, 0][5, 6][4, 5]
[3, 4][2, 3][1, 2][0, 1][13, 17][14, 16][12, 16][15, 15][11, 15][9, 13][8, 12][7, 11][6, 10][17, 13][5, 9][5, 7][22, 4][23, 3]
[24, 2][0, 0][4, 6][3, 5][1, 3][0, 2][13, 18][14, 17][12, 17][15, 16][11, 16][16, 15][8, 13][7, 12][6, 11][5, 10][17, 14][4, 9]
[22, 5][23, 4][24, 3][3, 6][0, 3][13, 19][12, 18][15, 17][11, 17][16, 16][17, 15][7, 13][6, 12][5, 11][4, 10][18, 14][3, 9][23, 5]
[24, 4][13, 20][14, 19][12, 19][11, 18][16, 17][10, 17][17, 16][18, 15][7, 14][6, 13][5, 12][3, 10][19, 14][2, 9][24, 5][13, 21]
[12, 20][15, 19][11, 19][17, 17][9, 17][18, 16][19, 15][7, 15][6, 14][5, 13][4, 12][3, 11][2, 10][19, 13][20, 14][1, 9][24, 6]
[13, 22][12, 21][11, 20][16, 19][18, 17][9, 16][19, 16][20, 15][7, 16][8, 15][6, 15][5, 14][4, 13][3, 12][2, 11][1, 10][19, 12]
[20, 13][21, 14][1, 8][0, 9][24, 7][13, 23][12, 22][11, 21][17, 19][19, 17][9, 15][8, 16][20, 16][21, 15][7, 17][6, 16][5, 15]
[4, 14][3, 13][2, 12][1, 11][0, 10][19, 11][20, 12][1, 7][0, 8][24, 8][23, 7][13, 24][14, 23][12, 23][11, 22][10, 21][18, 19]
[20, 17][21, 16][22, 15][7, 18][5, 16][4, 15][3, 14][2, 13][1, 12][0, 11][19, 10][20, 11][1, 6][0, 7][24, 9][23, 8][22, 7][14, 24]
[12, 24][15, 23][11, 23][10, 22][9, 21][19, 19][21, 17][22, 16][7, 19][6, 18][4, 16][3, 15][2, 14][1, 13][0, 12][19, 9][20, 10]
[0, 6][24, 10][23, 9][22, 8][21, 7][15, 24][11, 24][16, 23][10, 23][9, 22][20, 19][21, 18][22, 17][23, 16][7, 20][6, 19][5, 18]
[3, 16][2, 15][1, 14][0, 13][19, 8][20, 9][24, 11][23, 10][22, 9][20, 7][16, 24][10, 24][17, 23][21, 19][22, 18][23, 17][7, 21]
[6, 20][5, 19][4, 18][2, 16][1, 15][0, 14][19, 7][20, 8][23, 11][22, 10][9, 24][18, 23][21, 20][22, 19][23, 18][24, 17][7, 22][6, 21]
[4, 19][3, 18][1, 16][0, 15][23, 12][22, 11][8, 24][19, 23][21, 21][22, 20][23, 19][24, 18][6, 22][3, 19][2, 18][0, 16][22, 12][7, 24]
[19, 24][20, 23][21, 22][22, 21][23, 20][24, 19][5, 22][2, 19][1, 18][6, 24][20, 24][21, 23][22, 22][23, 21][24, 20][2, 20][1, 19]
[0, 18][21, 24][22, 23][23, 22][24, 21][2, 21][1, 20][0, 19][22, 24][23, 23][24, 22][1, 21][0, 20][23, 24][24, 23][0, 21][24, 24]
```

(0,0) -> (24, 24)

```
found target after exploring 446 nodes

 finished BFS to (24,24), total nodes explored: 447
[0, 0][0, 1][1, 0][0, 2][1, 1][2, 0][0, 3][1, 2][2, 1][3, 0][1, 3][2, 2][3, 1][4, 0][2, 3][3, 2][4, 1][5, 0][3, 3][4, 2][5, 1]
[6, 0][3, 4][4, 3][5, 2][6, 1][7, 0][3, 5][4, 4][5, 3][6, 2][7, 1][8, 0][3, 6][4, 5][5, 4][7, 2][8, 1][9, 0][4, 6][5, 5][8, 2]
[9, 1][10, 0][5, 6][8, 3][9, 2][10, 1][11, 0][5, 7][8, 4][9, 3][11, 1][12, 0][5, 8][8, 5][9, 4][12, 1][13, 0][5, 9][6, 8][8, 6]
[9, 5][13, 1][14, 0][5, 10][6, 9][4, 9][7, 8][8, 7][9, 6][13, 2][14, 1][15, 0][5, 11][6, 10][4, 10][7, 9][3, 9][8, 8][9, 7]
[13, 3][14, 2][15, 1][16, 0][5, 12][6, 11][7, 10][3, 10][8, 9][2, 9][9, 8][13, 4][14, 3][15, 2][16, 1][17, 0][5, 13][6, 12]
[4, 12][7, 11][8, 10][3, 11][2, 10][9, 9][1, 9][10, 8][13, 5][14, 4][15, 3][16, 2][17, 1][18, 0][5, 14][6, 13][4, 13][7, 12]
[3, 12][8, 11][9, 10][2, 11][1, 10][10, 9][1, 8][0, 9][11, 8][13, 6][14, 5][15, 4][16, 3][17, 2][19, 0][5, 15][6, 14][4, 14]
[7, 13][3, 13][8, 12][2, 12][9, 11][10, 10][1, 11][0, 10][11, 9][1, 7][0, 8][12, 8][13, 7][14, 6][15, 5][16, 4][17, 3][20, 0]
[5, 16][6, 15][4, 15][7, 14][3, 14][8, 13][2, 13][9, 12][1, 12][10, 11][11, 10][0, 11][12, 9][1, 6][0, 7][13, 8][14, 7][15, 6]
[16, 5][17, 4][20, 1][21, 0][6, 16][4, 16][7, 15][3, 15][2, 14][9, 13][1, 13][0, 12][11, 11][12, 10][13, 9][0, 6][14, 8][15, 7]
[16, 6][17, 5][20, 2][21, 1][22, 0][7, 16][3, 16][8, 15][2, 15][1, 14][0, 13][11, 12][12, 11][13, 10][14, 9][15, 8][16, 7]
[17, 6][20, 3][21, 2][22, 1][23, 0][7, 17][8, 16][2, 16][9, 15][1, 15][0, 14][11, 13][12, 12][13, 11][14, 10][15, 9][16, 8]
[17, 7][20, 4][21, 3][22, 2][23, 1][24, 0][7, 18][9, 16][1, 16][0, 15][12, 13][13, 12][14, 11][16, 9][17, 8][21, 4][22, 3][23, 2]
[24, 1][7, 19][6, 18][9, 17][0, 16][13, 13][14, 12][17, 9][22, 4][23, 3][24, 2][7, 20][6, 19][5, 18][10, 17][13, 14][14, 13]
17, 10][22, 5][23, 4][24, 3][7, 21][6, 20][5, 19][4, 18][11, 17][13, 15][14, 14][17, 11][23, 5][24, 4][7, 22][6, 21][4, 19]
[3, 18][11, 18][11, 16][12, 17][13, 16][14, 15][12, 15][17, 12][24, 5][6, 22][3, 19][2, 18][11, 19][12, 18][11, 15][12, 16]
[13, 17][14, 16][15, 15][17, 13][24, 6][5, 22][2, 19][1, 18][11, 20][12, 19][13, 18][14, 17][15, 16][16, 15][17, 14][24, 7]
[2, 20][1, 19][0, 18][11, 21][12, 20][13, 19][15, 17][16, 16][17, 15][18, 14][24, 8][23, 7][2, 21][1, 20][0, 19][11, 22][12, 21]
[10, 21][13, 20][14, 19][16, 17][17, 16][18, 15][19, 14][24, 9][23, 8][22, 7][1, 21][0, 20][11, 23][12, 22][10, 22][13, 21]
[9, 21][15, 19][17, 17][18, 16][19, 15][19, 13][20, 14][24, 10][23, 9][22, 8][21, 7][0, 21][11, 24][12, 23][10, 23][13, 22][9, 22]
[16, 19][18, 17][19, 16][20, 15][19, 12][20, 13][21, 14][24, 11][23, 10][22, 9][20, 7][12, 24][10, 24][13, 23][17, 19][19, 17]
[20, 16][21, 15][19, 11][20, 12][23, 11][22, 10][20, 8][19, 7][13, 24][9, 24][14, 23][18, 19][20, 17][21, 16][22, 15][19, 10]
[20, 11][23, 12][22, 11][20, 9][19, 8][14, 24][8, 24][15, 23][19, 19][21, 17][22, 16][19, 9][20, 10][22, 12][15, 24][7, 24][16, 23]
[20, 19][21, 18][22, 17][23, 16][16, 24][6, 24][17, 23][21, 19][22, 18][23, 17][18, 23][21, 20][22, 19][23, 18][24, 17][19, 23]
[21, 21][22, 20][23, 19][24, 18][19, 24][20, 23][21, 22][22, 21][23, 20][24, 19][20, 24][21, 23][22, 22][23, 21][24, 20][21, 24]
[22, 23][23, 22][24, 21][22, 24][23, 23][24, 22][23, 24][24, 23][24, 24]
```

DFS code:

```
import java.util.HashSet;
import java.util.List;

public class DFS {
    public int DFSSearch(int[][] grid, int startI, int startJ, int endI,
int endJ, List<int[]> pathList) {
        return dfs(grid, startI, startJ, endI, endJ, pathList, new
HashSet(), 0);
    }

    private int dfs(int[][] grid, int i, int j, int destI, int destJ,
List<int[]> pathList, HashSet<String> visitedSet, int explored) {
        if (visitedSet.contains(i+ "~" +j))
            return 0;

        if (i < 0 || j < 0 || i >= grid.length || j >= grid[i].length ||
grid[i][j] != 0)
            return 0;

        if (i == destI && j == destJ) {
            System.out.println("found target after exploring " + explored
+ " nodes");
            return 1;
        }

        pathList.add(new int[] {i, j});
        visitedSet.add(i+"~"+j);

        explored = 1 + dfs(grid, i+1, j, destI, destJ, pathList, visit-
edSet,explored+1) +
        dfs(grid, i-1, j, destI, destJ, pathList, visitedSet, explored+1)
+
        dfs(grid, i, j+1, destI, destJ, pathList, visitedSet, explored+1)
+
        dfs(grid, i, j-1, destI, destJ, pathList, visitedSet, explored+1)
        return explored;
    }
}
```

Results of running DFS (order of node coordinates visited):

S -> E1

```
found target after exploring 131 nodes

 finished DFS to E1, total nodes explored: 448
[13, 2][14, 2][15, 2][16, 2][17, 2][17, 3][16, 3][15, 3][14, 3][13, 3][13, 4][14, 4][15, 4][16, 4][17, 4][17, 5][16, 5][15, 5][14, 5]
[13, 5][13, 6][14, 6][15, 6][16, 6][17, 6][17, 7][16, 7][15, 7][14, 7][13, 7][13, 8][14, 8][15, 8][16, 8][17, 8][17, 9][16, 9][15, 9]
[14, 9][13, 9][12, 9][11, 9][10, 9][9, 9][8, 9][7, 9][6, 9][5, 9][4, 9][3, 9][2, 9][1, 9][0, 9][0, 10][1, 10][2, 10][3, 10][4, 10]
[5, 10][6, 10][7, 10][8, 10][9, 10][10, 10][11, 10][12, 10][13, 10][14, 10][14, 11][13, 11][12, 11][11, 11][10, 11][9, 11][8, 11][7, 11]
[6, 11][5, 11][5, 12][6, 12][7, 12][8, 12][9, 12][9, 13][8, 13][7, 13][6, 13][5, 13][4, 13][3, 13][2, 13][1, 13][0, 13][0, 14][1, 14]
[2, 14][3, 14][4, 14][5, 14][6, 14][7, 14][7, 15][8, 15][9, 15][9, 16][8, 16][7, 16][6, 16][5, 16][4, 16][3, 16][2, 16][1, 16][0, 16]
[0, 15][1, 15][2, 15][3, 15][4, 15][5, 15][6, 15][7, 17][7, 18][6, 18][5, 18][4, 18][3, 18][2, 18][1, 18][0, 18][0, 19][1, 19][2, 19]
[3, 19][4, 19][5, 19][6, 19][7, 19][7, 20][6, 20][6, 21][7, 21][7, 22][6, 22][5, 22][2, 20][1, 20][0, 20][0, 21][1, 21][2, 21][9, 17]
[10, 17][11, 17][12, 17][13, 17][14, 17][15, 17][16, 17][17, 17][18, 17][19, 17][20, 17][21, 17][22, 17][23, 17][24, 17][24, 18][23, 18]
[22, 18][21, 18][21, 19][22, 19][23, 19][24, 19][24, 20][23, 20][22, 20][21, 20][21, 21][22, 21][23, 21][24, 21][24, 22][23, 22][22, 22]
[21, 22][21, 23][22, 23][23, 23][24, 23][24, 24][23, 24][22, 24][21, 24][20, 24][19, 24][19, 23][20, 23][18, 23][17, 23][16, 23][15, 23]
[14, 23][13, 23][12, 23][11, 23][10, 23][10, 24][11, 24][12, 24][13, 24][14, 24][15, 24][16, 24][9, 24][8, 24][7, 24][6, 24][10, 22]
[11, 22][12, 22][13, 22][13, 21][12, 21][11, 21][10, 21][9, 21][9, 22][11, 20][12, 20][13, 20][13, 19][14, 19][15, 19][16, 19][17, 19]
[18, 19][19, 19][20, 19][12, 19][11, 19][11, 18][12, 18][13, 18][23, 16][22, 16][21, 16][20, 16][19, 16][18, 16][17, 16][16, 16][15, 16]
[14, 16][13, 16][12, 16][11, 16][11, 15][12, 15][13, 15][14, 15][15, 15][16, 15][17, 15][18, 15][19, 15][20, 15][21, 15][22, 15][21, 14]
[20, 14][19, 14][18, 14][17, 14][17, 13][17, 12][17, 11][17, 10][19, 13][20, 13][20, 12][19, 12][19, 11][20, 11][20, 10][19, 10][19, 9]
[20, 9][20, 8][19, 8][19, 7][20, 7][21, 7][22, 7][23, 7][24, 7][24, 8][23, 8][22, 8][22, 9][23, 9][24, 9][24, 10][23, 10][22, 10][22, 11]
[23, 11][24, 11][23, 12][22, 12][24, 6][24, 5][23, 5][22, 5][22, 4][23, 4][24, 4][24, 3][23, 3][22, 3][21, 3][20, 3][20, 4][21, 4][20, 2]
[21, 2][22, 2][23, 2][24, 2][24, 1][23, 1][22, 1][21, 1][20, 1][20, 0][21, 0][22, 0][23, 0][24, 0][19, 0][18, 0][17, 0][16, 0][15, 0]
[14, 0][13, 0][12, 0][11, 0][10, 0][9, 0][8, 0][7, 0][6, 0][5, 0][4, 0][3, 0][2, 0][1, 0][0, 0][0, 1][1, 1][2, 1][3, 1][4, 1][5, 1][6, 1]
[7, 1][8, 1][9, 1][10, 1][11, 1][12, 1][13, 1][14, 1][15, 1][16, 1][17, 1][9, 2][8, 2][7, 2][6, 2][5, 2][4, 2][3, 2][2, 2][1, 2][0, 2]
[0, 3][1, 3][2, 3][3, 3][4, 3][5, 3][5, 4][4, 4][3, 4][3, 5][4, 5][5, 5][5, 6][4, 6][3, 6][5, 7][5, 8][6, 8][7, 8][8, 8][9, 8][10, 8]
[11, 8][12, 8][9, 7][8, 7][8, 6][9, 6][9, 5][8, 5][8, 4][9, 4][9, 3][8, 3][14, 14][13, 14][13, 13][14, 13][14, 12][13, 12][12, 12][11, 12]
[11, 13][12, 13][0, 12][1, 12][2, 12][3, 12][4, 12][3, 11][2, 11][1, 11][0, 11][0, 8][1, 8][1, 7][0, 7][0, 6][1, 6]
```

S -> E2

```
found target after exploring 212 nodes

 finished DFS to E2, total nodes explored: 450
[13, 2][14, 2][15, 2][16, 2][17, 2][17, 3][16, 3][15, 3][14, 3][13, 3][13, 4][14, 4][15, 4][16, 4][17, 4][17, 5][16, 5][15, 5][14, 5]
[13, 5][13, 6][14, 6][15, 6][16, 6][17, 6][17, 7][16, 7][15, 7][14, 7][13, 7][13, 8][14, 8][15, 8][16, 8][17, 8][17, 9][16, 9][15, 9]
[14, 9][13, 9][12, 9][11, 9][10, 9][9, 9][8, 9][7, 9][6, 9][5, 9][4, 9][3, 9][2, 9][1, 9][0, 9][0, 10][1, 10][2, 10][3, 10][4, 10][5, 10]
[6, 10][7, 10][8, 10][9, 10][10, 10][11, 10][12, 10][13, 10][14, 10][14, 11][13, 11][12, 11][11, 11][10, 11][9, 11][8, 11][7, 11][6, 11]
[5, 11][5, 12][6, 12][7, 12][8, 12][9, 12][9, 13][8, 13][7, 13][6, 13][5, 13][4, 13][3, 13][2, 13][1, 13][0, 13][0, 14][1, 14][2, 14]
[3, 14][4, 14][5, 14][6, 14][7, 14][7, 15][8, 15][9, 15][9, 16][8, 16][7, 16][6, 16][5, 16][4, 16][3, 16][2, 16][1, 16][0, 16][0, 15]
[1, 15][2, 15][3, 15][4, 15][5, 15][6, 15][7, 17][7, 18][6, 18][5, 18][4, 18][3, 18][2, 18][1, 18][0, 18][0, 19][1, 19][2, 19][3, 19]
[4, 19][5, 19][6, 19][7, 19][7, 20][6, 20][6, 21][7, 21][7, 22][6, 22][5, 22][2, 20][1, 20][0, 20][0, 21][1, 21][2, 21][9, 17][10, 17]
[11, 17][12, 17][13, 17][14, 17][15, 17][16, 17][17, 17][18, 17][19, 17][20, 17][21, 17][22, 17][23, 17][24, 17][24, 18][23, 18][22, 18]
[21, 18][21, 19][22, 19][23, 19][24, 19][24, 20][23, 20][22, 20][21, 20][21, 21][22, 21][23, 21][24, 21][24, 22][23, 22][22, 22][21, 22]
[21, 23][22, 23][23, 23][24, 23][24, 24][23, 24][22, 24][21, 24][20, 24][19, 24][19, 23][20, 23][18, 23][17, 23][16, 23][15, 23][14, 23]
[13, 23][12, 23][11, 23][10, 23][10, 24][11, 24][12, 24][13, 24][14, 24][15, 24][16, 24][9, 24][8, 24][7, 24][6, 24][10, 22][11, 22]
[12, 22][13, 22][13, 21][12, 21][11, 21][10, 21][9, 21][9, 22][11, 20][12, 20][13, 20][13, 19][14, 19][15, 19][16, 19][17, 19][18, 19]
[19, 19][20, 19][12, 19][11, 19][11, 18][12, 18][13, 18][23, 16][22, 16][21, 16][20, 16][19, 16][18, 16][17, 16][16, 16][15, 16][14, 16]
[13, 16][12, 16][11, 16][11, 15][12, 15][13, 15][14, 15][15, 15][16, 15][17, 15][18, 15][19, 15][20, 15][21, 15][22, 15][21, 14][20, 14]
[19, 14][18, 14][17, 14][17, 13][17, 12][17, 11][17, 10][19, 13][20, 13][20, 12][19, 12][19, 11][20, 11][20, 10][19, 10][19, 9][20, 9]
[20, 8][19, 8][19, 7][20, 7][21, 7][22, 7][23, 7][24, 7][24, 8][23, 8][22, 8][22, 9][23, 9][24, 9][24, 10][23, 10][22, 10][22, 11][23, 11]
[24, 11][23, 12][22, 12][24, 6][24, 5][23, 5][22, 5][22, 4][23, 4][24, 4][24, 3][23, 3][22, 3][21, 3][20, 3][20, 4][21, 4][20, 2][21, 2]
[22, 2][23, 2][24, 2][24, 1][23, 1][22, 1][21, 1][20, 1][20, 0][21, 0][22, 0][23, 0][24, 0][19, 0][18, 0][17, 0][16, 0][15, 0][14, 0]
[13, 0][12, 0][11, 0][10, 0][9, 0][8, 0][7, 0][6, 0][5, 0][4, 0][3, 0][2, 0][1, 0][0, 0][0, 1][1, 1][2, 1][3, 1][4, 1][5, 1][6, 1][7, 1]
[8, 1][9, 1][10, 1][11, 1][12, 1][13, 1][14, 1][15, 1][16, 1][17, 1][9, 2][8, 2][7, 2][6, 2][5, 2][4, 2][4, 3][5, 3][5, 4][4, 4][3, 4]
[3, 5][4, 5][5, 5][5, 6][4, 6][3, 6][5, 7][5, 8][6, 8][7, 8][8, 8][9, 8][10, 8][11, 8][12, 8][9, 7][8, 7][8, 6][9, 6][9, 5][8, 5][8, 4]
[9, 4][9, 3][8, 3][3, 3][2, 3][1, 3][0, 3][0, 2][1, 2][2, 2][14, 14][13, 14][13, 13][14, 13][14, 12][13, 12][12, 12][11, 12][11, 13]
[12, 13][0, 12][1, 12][2, 12][3, 12][4, 12][3, 11][2, 11][1, 11][0, 11][0, 8][1, 8][1, 7][0, 7][0, 6][1, 6]
```

(0,0) -> (24, 24)

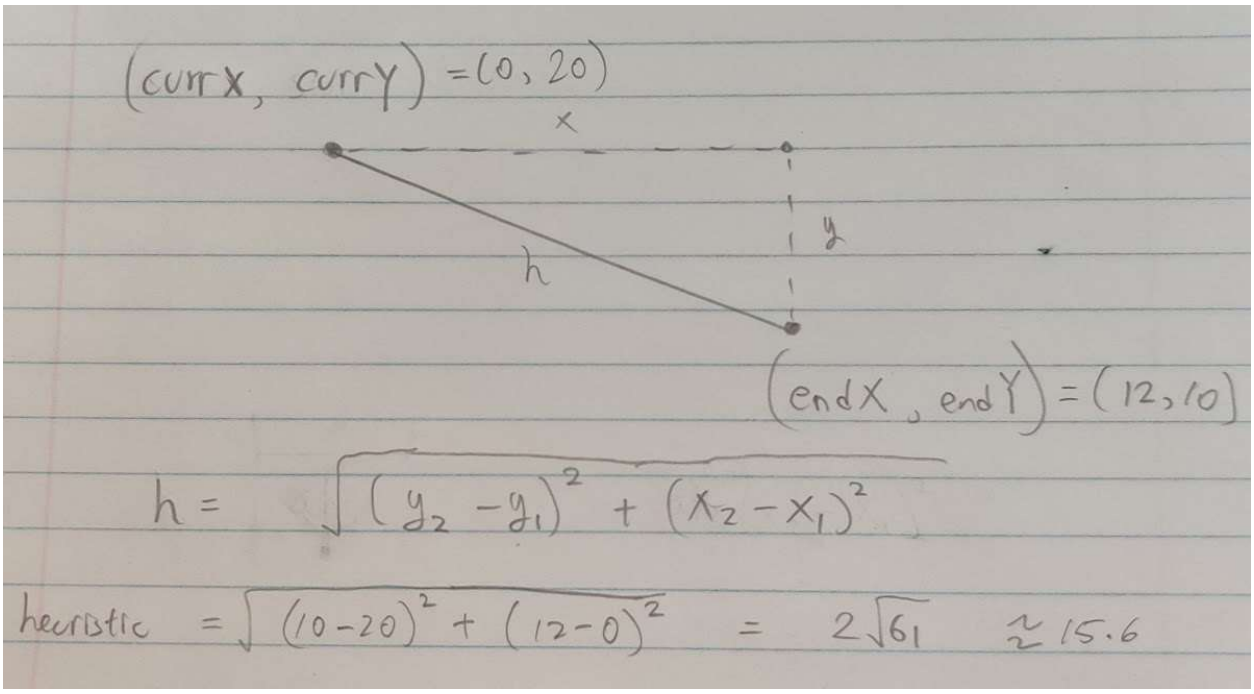
```
found target after exploring 98 nodes

 finished DFS to (24,24), total nodes explored: 448
[0, 0][1, 0][2, 0][3, 0][4, 0][5, 0][6, 0][7, 0][8, 0][9, 0][10, 0][11, 0][12, 0][13, 0][14, 0][15, 0][16, 0][17, 0][18, 0][19, 0]
[20, 0][21, 0][22, 0][23, 0][24, 0][24, 1][23, 1][22, 1][21, 1][20, 1][20, 2][21, 2][22, 2][23, 2][24, 2][24, 3][23, 3][22, 3][21, 3]
[20, 3][20, 4][21, 4][22, 4][23, 4][24, 4][24, 5][23, 5][22, 5][24, 6][24, 7][23, 7][22, 7][21, 7][20, 7][19, 7][19, 8][20, 8][20, 9]
[19, 9][19, 10][20, 10][20, 11][19, 11][19, 12][20, 12][20, 13][19, 13][19, 14][20, 14][21, 14][21, 15][22, 15][22, 16][23, 16][23, 17]
[24, 17][24, 18][23, 18][22, 18][21, 18][21, 19][22, 19][23, 19][24, 19][24, 20][23, 20][22, 20][21, 20][21, 21][22, 21][23, 21]
[24, 21][24, 22][23, 22][22, 22][21, 22][21, 23][22, 23][23, 23][24, 23][23, 24][22, 24][21, 24][20, 24][19, 24][19, 23][20, 23]
[18, 23][17, 23][16, 23][15, 23][14, 23][13, 23][12, 23][11, 23][10, 23][10, 24][11, 24][12, 24][13, 24][14, 24][15, 24][16, 24]
[9, 24][8, 24][7, 24][6, 24][10, 22][11, 22][12, 22][13, 22][13, 21][12, 21][11, 21][10, 21][9, 21][9, 22][11, 20][12, 20][13, 20]
[13, 19][14, 19][15, 19][16, 19][17, 19][18, 19][19, 19][20, 19][12, 19][11, 19][11, 18][12, 18][13, 18][13, 17][14, 17][15, 17]
[16, 17][17, 17][18, 17][19, 17][20, 17][21, 17][22, 17][21, 16][20, 16][19, 16][18, 16][17, 16][16, 16][15, 16][14, 16][13, 16]
[12, 16][11, 16][11, 17][12, 17][10, 17][9, 17][9, 16][8, 16][7, 16][6, 16][5, 16][4, 16][3, 16][2, 16][1, 16][0, 16][0, 15][1, 15]
[2, 15][3, 15][4, 15][5, 15][6, 15][7, 15][8, 15][9, 15][7, 14][6, 14][5, 14][4, 14][3, 14][2, 14][1, 14][0, 14][0, 13][1, 13][2, 13]
[3, 13][4, 13][5, 13][6, 13][7, 13][8, 13][9, 13][9, 12][8, 12][7, 12][6, 12][5, 12][4, 12][3, 12][2, 12][1, 12][0, 12][0, 11][1, 11]
[2, 11][3, 11][3, 10][4, 10][5, 10][6, 10][7, 10][8, 10][9, 10][10, 10][11, 10][12, 10][13, 10][14, 10][14, 11][13, 11][12, 11][11, 11]
[10, 11][9, 11][8, 11][7, 11][6, 11][5, 11][11, 12][12, 12][13, 12][14, 12][14, 13][13, 13][12, 13][11, 13][13, 14][14, 14][14, 15]
[15, 15][16, 15][17, 15][18, 15][19, 15][20, 15][18, 14][17, 14][17, 13][17, 12][17, 11][17, 10][17, 9][16, 9][15, 9][14, 9][13, 9]
[12, 9][11, 9][10, 9][9, 9][8, 9][7, 9][6, 9][5, 9][4, 9][3, 9][2, 9][1, 9][0, 9][0, 10][1, 10][2, 10][0, 8][1, 8][1, 7][0, 7][0, 6]
[1, 6][5, 8][6, 8][7, 8][8, 8][9, 8][10, 8][11, 8][12, 8][13, 8][14, 8][15, 8][16, 8][17, 8][17, 7][16, 7][15, 7][14, 7][13, 7][13, 6]
[14, 6][15, 6][16, 6][17, 6][17, 5][16, 5][15, 5][14, 5][13, 5][13, 4][14, 4][15, 4][16, 4][17, 4][17, 3][16, 3][15, 3][14, 3][13, 3]
[13, 2][14, 2][15, 2][16, 2][17, 2][17, 1][16, 1][15, 1][14, 1][13, 1][12, 1][11, 1][10, 1][9, 1][8, 1][7, 1][6, 1][5, 1][4, 1][3, 1]
[2, 1][1, 1][0, 1][0, 2][1, 2][2, 2][3, 2][4, 2][5, 2][6, 2][7, 2][8, 2][9, 2][9, 3][8, 3][8, 4][9, 4][9, 5][8, 5][8, 6][9, 6][9, 7]
[8, 7][5, 3][4, 3][3, 3][2, 3][1, 3][0, 3][3, 4][4, 4][5, 4][5, 5][4, 5][3, 5][3, 6][4, 6][5, 6][5, 7][13, 15][12, 15][11, 15][7, 17]
[7, 18][6, 18][5, 18][4, 18][3, 18][2, 18][1, 18][0, 18][0, 19][1, 19][2, 19][3, 19][4, 19][5, 19][6, 19][7, 19][7, 20][6, 20][6, 21]
[7, 21][7, 22][6, 22][5, 22][2, 20][1, 20][0, 20][0, 21][1, 21][2, 21][22, 8][23, 8][24, 8][24, 9][23, 9][22, 9][22, 10][23, 10][24, 10]
[24, 11][23, 11][22, 11][22, 12][23, 12]
```

A*:

The heuristic function h(n) chosen is the Euclidean distance between two nodes on the plane. At the starting node, the heuristic (distance) between two nodes is supposedly the highest the algorithm would see, and at the ending node, heuristics would be 0.
at each step, f(n) = g(n) + h(n) where h(n) is the distance between two points and g(n) = 1 as we are only taking one step at a time.

By using the above heuristic function, we'd have a rough estimate of the remaining distance to the end, and that estimate value gets closer and closer to zero when we are moving in the right direction.



In my A star algorithm, I am choosing the next node to explore from the BFS fringe by ranking each of the neighbor nodes via the heuristic function
f(n) = g(n) + h(n)
since g(n) AKA path cost would be the same for any of the neighbor node we take, f(n) is directly correlated with h(n)
For the 'greedy' part when selecting the next node to explore, I have used a priority-queue with a custom comparator. The comparator compares two nodes by their Euclidean distance to the destination coordinate. So at each node, as we put the neighboring nodes into the priority-queue, they will be sorted in ascending order based on their h(n) value. This will ensure that the next element popped off the priority-queue will be the node with the lowest heuristic value from the fringe.

A* code:

```
public class AStar {
    public int[][] directions= {{1,0},{-1,0},{0,1},{0,-1}};

    public int AStar (int[][] grid, int startX, int startY, int endI, int endJ,
List<int[]> pathList) {
        int nodesExplored = 0;
        Comparator<Node> customComparator = new Comparator<Node>() {
            @Override
            public int compare(Node s1, Node s2) {
                //our heuristic comparing 2 nodes decides their order in the
fringe: (x-endX)^2 + (y-endY)^2
                double h_s1 = Math.pow(Math.abs(s1.I - endI), 2) +
Math.pow(Math.abs(s1.J - endJ), 2);
                double h_s2 = Math.pow(Math.abs(s2.I - endI), 2) +
Math.pow(Math.abs(s2.J - endJ), 2);
                return (int)(h_s1 - h_s2);
            }
        };

        PriorityQueue<Node> pq = new PriorityQueue(customComparator);
        pq.offer(new Node(startX, startY));

        while (!pq.isEmpty()) {
            Node currNode = pq.poll();
            int i = currNode.I, j = currNode.J;
            if (grid[i][j] == -1)
                continue;

            if (i == endI && j == endJ)
                System.out.println("found target after exploring "+nodesExplored+
" nodes");

            if (i < 0 || j < 0 || i >= grid.length || j >= grid[i].length)
                continue;

            //reached end, quit:
            if (i == endI && j == endJ)
                return nodesExplored+1;

            grid[i][j] = -1; //mark as visited
            nodesExplored++;
            pathList.add(new int[] {i, j});

            for(int k = 0; k < directions.length; k++) {
                int r = i + directions[k][0], c = j + directions[k][1];
                pq.offer(new Node(r, c));
            }
        }
        return nodesExplored;
    }
}

class Node {
    int I, J;
    Node (int i, int j) {
        this.I = i;
        this.J = j;
    }
}
```

Results of running A* (order of node coordinates visited):

S -> E1

```
found target after exploring 29 nodes

 finished AStar to E1, cost of: 30
[13, 2][13, 3][13, 4][13, 5][13, 6][13, 7][13, 8][13, 9][13, 10][13, 11][13, 12][13, 13][13, 14][13, 15][12, 15][12, 16][11, 16]
[11, 17][10, 17][10, 18][9, 18][9, 19][8, 19][8, 20][7, 20][7, 21][6, 21][6, 22][5, 22]
```

S -> E2

```
found target after exploring 10 nodes

 finished AStar to E2, cost of: 11
[13, 2][12, 2][11, 2][10, 2][9, 2][8, 2][7, 2][6, 2][5, 2][4, 2]
```

(0, 0) -> (24,24)

```
found target after exploring 48 nodes

 finished AStar to (24,24), cost of: 49
[0, 0][1, 0][1, 1][2, 1][2, 2][3, 2][3, 3][4, 3][4, 4][5, 4][5, 5][6, 5][6, 6][7, 6][7, 7][8, 7][8, 8][9, 8][9, 9][10, 9][10, 10]
[11, 10][11, 11][12, 11][12, 12][13, 12][13, 13][14, 13][14, 14][15, 14][15, 15][16, 15][16, 16][17, 16][17, 17][18, 17][18, 18]
[19, 18][19, 19][20, 19][20, 20][21, 20][21, 21][22, 21][22, 22][23, 22][23, 23][24, 23]
```

Seems like the heuristic I chose for A* algorithm worked pretty well since there are less nodes visited and we waste less time exploring before finding the target