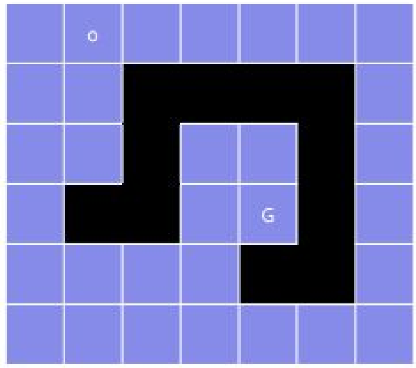
DFS and A* Search

We consider a maze under a windy condition as shown in the following figure. We assume that the wind



comes from the south and the cost of one step for the agent is defined as follows: 1 for moving northward; 2 for moving westward or eastward; 3 for moving southward. We assume that the square labeled with 0 is the starting square and the goal square is labelled with "G" and all dark-shaded squares and all edges are obstacles.

Depth-First Search We consider the graph-search version, i.e., using explored set to void redundant paths.

A* Search We use a modified Manhattan distance used in class as the heuristic function $h(n)$ by considering the windy situation. For example, for the start node, the agent has to move at least 3 steps eastward and 3 steps southward in order to reach the goal. Therefore, we have $h(n) = 3 * 3 + 3 * 2 = 15$ at the start node.

We use a label we did in class to indicates the order of choosing the corresponding unlabeled square and adding it to the frontier. To break tier for unlabeled squares (expanding children nodes), use this order: first westward; then northward; then eastward; then southward. To break tier for labeled squares (picking one child node to expand), the smallest label is picked first.

Follow the same way as done in the class to show the search steps with labels inside circles for the following search algorithms: Depth-first search and A* search (ignoring the subscripts which we use in class). Your outcome should be displayed as these:

DFS:

01	00	02	26	25	24	23
04	03	##	##	##	##	22
06	05	##	28		##	21
07	##	##	27	29	##	20
08	09	13	15	##	##	19
10	11	12	14	16	17	18

A*

01	00	02	04	07	09	11
05	03	##	##	##	##	13
08	06	##	25		##	16
10	##	##	23	26	##	19
12	14	17	20	##	##	22
15	18	21	24			

DFS Algorithm

```
bool alreadyVisited(std::vector<std::pair<int,int>> vec, int row, int col)
{
    std::vector<std::pair<int, int>>::iterator it;

    it = std::find(vec.begin(), vec.end(), std::make_pair(row, col));
    if (it != vec.end()) return true;
    else return false;
}
```

- Checks explored set to avoid infinite loop

```
            //Is it possible to move down?
            if (downBound < row && grid[downBound][currY] != "##" && !downPath)
            {
                dfsVisited.push_back(std::make_pair(downBound, currY));
                grid[downBound][currY] = std::to_string(dfsVisited.size() - 1);
                draw();
                if (currX == dfsFrontier.back().first && currY == dfsFrontier.back().second)
                {
                    dfsFrontier.pop_back();
                }
                dfsFrontier.push_back(std::make_pair(downBound, currY));
                dfsFrontierUpdated = true;
            }
            //No possible move from this position, remove it from frontier
            else if(dfsFrontierUpdated == false)
            {
                dfsFrontier.pop_back();
            }
            dfsFrontierUpdated = false;
```

- Checks possible moves from current location (Order: left, up, right, down)
- If there are no possible moves, remove the node from the frontier (stack)

Microsoft Visual Studio Debug Console

```
===== DFS Start =====

--      0       --      --      --      --      --

--      --      ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
1       0       --      --      --      --      --

--      --      ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
1       0       2       --      --      --      --

--      --      ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
1       0       2       --      --      --      --

--      3       ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
```

Microsoft Visual Studio Debug Console

```
****************************************************
1       0       2       26      25      24      23

4       3       ##      ##      ##      ##      22

6       5       ##      28      --      ##      21

7       ##      ##      27      GG      ##      20

8       9       13      15      ##      ##      19

10      11      12      14      16      17      18

****************************************************
1       0       2       26      25      24      23

4       3       ##      ##      ##      ##      22

6       5       ##      28      --      ##      21

7       ##      ##      27      29      ##      20

8       9       13      15      ##      ##      19

10      11      12      14      16      17      18

****************************************************
===== DFS End =====
```

A* Algorithm

```cpp
int EstManhattanDistance(int row, int col)
{
    //The estimated path cost the goal from current position
    int h = 0;
    int xCost = 0, yCost = 0;

    //Horizontal distance
    //We're not in the right column, so we need to move left or right
    if (col != goal.second)
    {
        xCost = abs(col - goal.second) * 2; //Path cost of 2 for both left and right case
    }
    //We are in the correct column, no left/right movement necessary
    else
    {
        xCost = 0;
    }

    //Vertical distance
    //Wind condition is 1 for upward movement, but 3 for downward movement
    //Current vertical position on the grid is lower than the goal and must move upward
    if (row > goal.first)
    {
        yCost = (row - goal.first); //Path cost of 1
    }
    //Current vertical position on the grid is higher than the goal and must move downward
    else if (row < goal.first)
    {
        yCost = (goal.first - row) * 3; //Path cost of 3
    }
    //Current vertical position is on the same row as goal, no upward/downward movement necessary
    else
    {
        yCost = 0;
    }

    h = xCost + yCost;

    return h;
}
```

- Heuristic h(n) for implemented A* algorithm

```cpp
bool lowestTotalEstCost(std::vector<int> v1, std::vector<int> v2)
{
    //Sort by the lowest total estimated path cost
    if (v1[2] != v2[2])
    {
        return v1[2] < v2[2];
    }
    //If total estimated path costs are equal, sort by stepRank
    else
    {
        return v1[5] < v2[5];
    }
}
```

- Additional compare function when used with std::sort
- Always sort by the lowest total estimated path cost - lowest f(n)
- If there are equal costs, sorts those by the lowest step rank (whichever step was queued up first)

Microsoft Visual Studio Debug Console

```
===== A* Start =====

--      0       --      --      --      --      --

--      --      ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
1       0       --      --      --      --      --

--      --      ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
1       0       2       --      --      --      --

--      --      ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
1       0       2       --      --      --      --

--      3       ##      ##      ##      ##      --

--      --      ##      --      --      ##      --

--      ##      ##      --      GG      ##      --

--      --      --      --      ##      ##      --

--      --      --      --      --      --      --

****************************************************
```

Microsoft Visual Studio Debug Console

```
****************************************************
1       0       2       4       7       9       11

5       3       ##      ##      ##      ##      13

8       6       ##      --      --      ##      16

10      ##      ##      23      GG      ##      19

12      14      17      20      ##      ##      22

15      18      21      24      --      --      --

****************************************************
1       0       2       4       7       9       11

5       3       ##      ##      ##      ##      13

8       6       ##      25      --      ##      16

10      ##      ##      23      GG      ##      19

12      14      17      20      ##      ##      22

15      18      21      24      --      --      --

****************************************************
1       0       2       4       7       9       11

5       3       ##      ##      ##      ##      13

8       6       ##      25      --      ##      16

10      ##      ##      23      26      ##      19

12      14      17      20      ##      ##      22

15      18      21      24      --      --      --

****************************************************
===== A* End =====
```