BLG336E, Analysis of Algorithms II, Spring 2017 Project Report 1

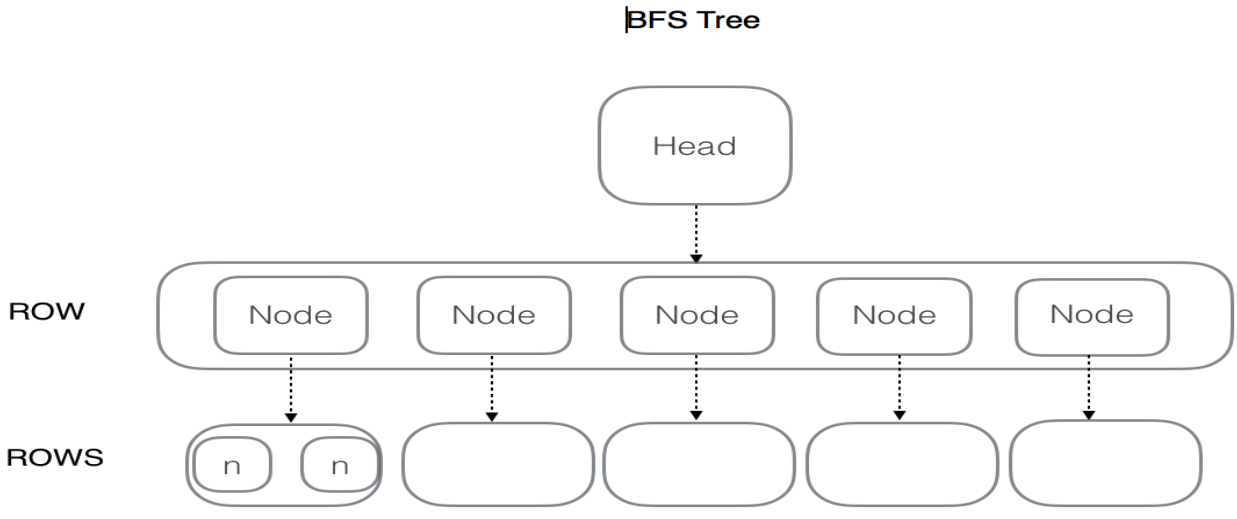
Name: Emre Özdil

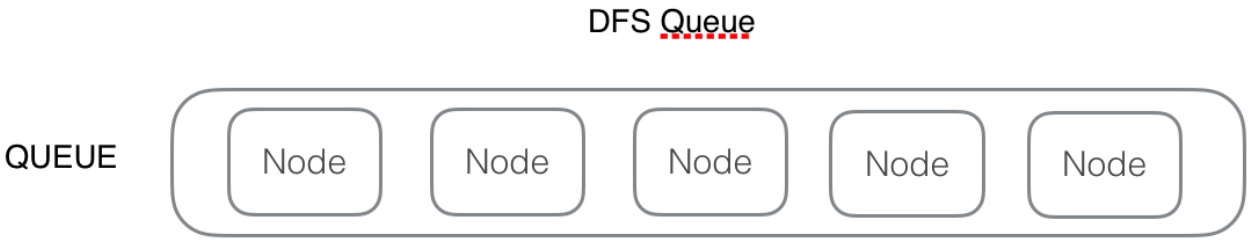
Student ID: 150120138

Part A. My Program (20 points)

1) My algorithm work

- Read console.
- Read input and select algorithm.
- **If BFS is selected.**
  - Head Node's maze is equal to input maze
  - All Nodes set insert this maze for control cycle
  - **While is solution find**
    - **Traverse bfs**
    - Traverse maze
    - Try go to right, left, up, or down
    - Each loop row nodes vector insert a new node which include new maze
    - Going to right we control to find the solution
    - If new maze is existed continue
    - Print output head to solution node
- **If DFS selected**
  - Head Node's maze is equal to input maze
  - All Nodes set insert this maze for control cycle
  - **While is solution find**
    - **Traverse dfs**
    - Traverse maze
    - Try go to right, left, up, or down
    - Each loop new node inserted to queue
    - Going to right we control to find the solution
    - If new maze is existed continue
    - Print output head to solution node





Complexity = O (Nlog N) **N-> The number of nodes to find the solution**

Constant numbers are ignored like a 6x6 maze.

2) Classes and Methods

```
// Car struct include car inputs
struct Car
{
    int id;
    char direction;
    int length;
};
// Maze struct include maze[6][6] matrix
struct Maze
{
    vector< vector<int> > maze;
};
// In BFS tree include node and nodes
class Node
{
public:
    Node();
    ~Node();
    vector< vector<int> > maze;
    Nodes* below;
    Node* above;
};
// In BFS each node has nodes child
struct Nodes
{
    vector<Node*> nodes;
    Node *above;
};


void readFile(string, string, string); // read txt and select algorithm
void addBFSNode(Maze*, string); // create BFS tree and find solution
void addDFSNode(Maze*, string); // create DFS queue and find solution
```

Part B. Complexity of my algorithm (20 points)

1) Extra complexity that is caused by the cycle search

I use set for cycle control. All nodes insert this set. **O (log N)** to search for an individual element in <set>.[1] This algorithm search for all elements; therefore, complexity is equal to O (Nlog N).

2) Adjacency list representation

Matrix will be $O(V^2)$.

Adjacency List is O(V + E).

E-> node

V-> the number of connection all nodes

How to compile

- g++ *.cpp
- ./a.out bfs blocks.txt bfs.txt
- ./a.out dfs blocks.txt dfs.txt

[1] http://www.cplusplus.com/reference/set/set/find/