Mehmet Barış Yaman
150130136

ANALYSIS OF ALGORITHMS PROJECT 1 REPORT

Analysis of the BFS and DFS algorithms

When the program runs with the input file of 'blocks.txt', the results are:

For BFS,

```
Number of nodes created for bfs is: 6
(BFS) Maximum number of nodes can be kept in memory is: 1389
Run Time for BFS is: 0.041986 seconds
```

For DFS,

```
(Number of nodes created for dfs is: 46
(DFS) Maximum number of nodes can be kept in memory is: 302
Run Time for DFS is: 0.006051 seconds
```

By inspecting the results, we see that number of nodes generated in BFS algorithm is less than DFS algorithm, since BFS algorithm always guarantees the shortest path in the graph if the edges have no weights. On the other hand, DFS algorithm always finds the path but the path can not be always the shortest. This situation depends on the implementation of the move parts in the graph.

It is also seen that maximum number of nodes kept in memory is more in BFS. Due to the fact that BFS works layer by layer, which means traversing all paths between the layers before starting to traverse the next layer. This is accomplished by using single list data structure that works as a queue or multiple lists in BFS algorithms. On the other hand, dfs algorithms generally traverse only one path between the nodes and directly goes to the next state without considering the other paths. Therefore, dfs do not generate the nodes in all layers, just traverses the node in the next level. This is accomplished with stack data structure. Therefore DFS keeps less node in memory in general.

Moreover, by inspecting the results we see that run time in Dfs is less than BFS. This is usually true but not always. Depending on the implementation of moves DFS can last longer than BFS. For example if we consider the nodes generated by moving the blocks down different than up, the run time of DFS algorithm changes accordingly.

Mehmet Barış Yaman
150130136

Algorithms Pseudocode and Work Structure

```
BFS(Node s)
    Initialize discovered array
    Initialize queue and push s
    While queue is not empty                                    O(n)
            Assign the first node to s
            If s is not discovered                              O(n)
                    Add s to te node vector
                    Make the node as discovered
                    Generate nodes with respect to the moves in s
                    Check that the generated node has clear path to the exit
                    If the node has clear path to the exit
                            Stop the algorithm and create output file
            Pop a node from queue
```

Considerig the discovered array iteration and queue traversing (with pop and top functions), the complexity of my BFS implemented algorithm is O(n^2)

```
DFS(Node u)
    Initialize explored array
    Initialize stack and push u
    While stack is not empty                                    O(n)
            Assign the topmost node to u
            Pop the topmost node from stack
            If u is not discovered                              O(n)
                    Add u to te node vector
                    Make the node as discovered
                    Generate nodes with respect to the moves in u
                    Check that the generated node has clear path to the exit
                    If the node has clear path to the exit
                            Stop the algorithm and create output file
```

Considerig the discovered array iteration and stack traversing (with pop and top functions), the complexity of my DFS implemented algorithm is O(n^2)

Mehmet Barış Yaman
150130136

Basically BFS algorithm works with discovered array for cycle checking, node vector to store the nodes and their return numbers and a queue to clarify the first in first out style for pathfinding.

DFS algorithm works with explored array for cycle checking, node vector to store the nodes and their return numbers and a stack to clarify the last in first out style for pathfinding.

Classses and Methods

In this project, I have three classes namely Block, Node and Graph.

Block class holds the structure of each blocks in the environment. It has 5 properties namely xCoordinate, yCoordinate, length, direction and blockNumber which is used in the representation of blocks.

Node class is for the nodes of the graph. Its properties are environement vector(used for hoding blocks of the environment), representation array, nodeNumber, backNumber(used for returning the starting node after a path was found), terminate (used for terminating the algorithms). It has also three methods which are addBlock(used for adding a block to the environment vector), popBlock and represent (used for printing the output).

Graph class is needed for DFS and BFS algorithms. It two properties property namely nodes vector and numberOfNodes which is used for assigning new nodeNumbers to new nodes. Additionally it has 7 seven properties namely bfs, dfs, addNodes, checkIfNodeWasDiscovered(used for cycle checking), createOutput, pushNodesForQueue, pushNodesForStack. pushNodesForQueue and pushNodesForStack functions used for making a single move in the node, generate a new one and push to the list/stack.

Extra Complexity in Cycle Search

The function responsible for cycle checking in my implementation is named checkIfNodeWasDiscovered. In this function discovered/explored array is traversed to find that if a node is visited or not. The complexity of that function is O(n). Therefore the extra complexity with that function is O(n).

In my dfs and bfs function, each list/stack nodes and the visited array node is visited. Therefore if we have n vertices in this graph the complexity of bfs and dfs will be O(n^2). Without cycle checking the complexity will be O(n).

If Adjacency List is Used

In my implementation, nodes are created at run time. Therefore adjacency list can only be used for cycle checking. Before adding the node, adjacency list is used for checking whether there is a cycle or not by traversing the array part(in other words, first list node of an array index). Thus, the additional complexity of using adjacency list is O(n) which is same as my algorithm used in cycle checking. Therefore, the algorithm complexity of my implementation will stay O(n^2).

Mehmet Barış Yaman
150130136