BLG336E – SPRING 2022

ANALYSIS OF ALGORITHM II

PROJECT 1 REPORT

Fatma Sena AKÇAĞLAYAN 150170085

1)

a) Firstly, instead of defining a matrix to store game board coordinates, states are stores as numbers in a single list. In this way, it was easier to control the states. Bit masking method was used to convert the coordinates to integers. For example, for the coordinate (3, 2), the numbers 3, binary 11, and 4, binary 10, are combined. As a result, 1110, that is, the 14th index, has specified the coordinate (3, 2). This process is formulated in code. The index of the coordinate read was calculated with the formula size_of_board * x + y.

A gameboard list is defined for each player. Ships index are marked as true on the list. Then a graph is created for each player holding the gameboard. Visited_counter is defined for visited nodes in the graph, and memory_counter is defined for nodes kept in memory. A dynamic list was determined as long as the number of nodes to hold the neighbor of each node. Top, left, bottom and right neighbors for each node is added to this list, so the edges is drawn. Then, a hit sequence is occured according to the given algorithm. It is determined which player would win by comparing the list of the rival gameboard.

b)

```
O(N) BFS(Graph G, Node start_point, Rival Board rv)
        O(1) Q  = new Queue;
        O(1) Push start_point  to Q
        O(1) memory_counter++;
        O(N) While Q is not empty
                O(1) Let current_point = Q.front();
                O(1) Q.pop();
                O(1) If current_point not visited before
                        O(1) Mark current_point as visited
                        O(1) visited_counter++;
                        O(1) If hits ship on rival board rv
                                O(1) ship counter--;
                                O(1) if (ship counter == 0) return True;
                        O(4) For each neighbor i is not visited before
                                O(1) Push neighbor i to Q
                                O(1) memory_counter++;
        O(1) return False;
```

```
O(N) DFS(Graph G, Node start_point, Rival Board rv)
    O(1) S  = new Stack;
    O(1) Push start_point  to S
    O(1) memory_counter++;
    O(N) While S is not empty
        O(1) Let current_point = S.top();
        O(1) S.pop();
        O(1) If current_point not visited before
            O(1) Mark current_point as visited
            O(1) visited_counter++;
            O(1) If hits ship on rival board rv
                O(1) ship counter--;
                O(1) if (ship counter == 0) return True;
            O(4) For each neighbor i is not visited before
                O(1) Push neighbor i to S
                O(1) memory_counter++;
    O(1) return False;
```

c) As seen in psuedo code, the time complexity depends on the number of nodes "n" and the number of edges on "m". In our program, the number of edges always 4. Therefore, it can be said that the time complexity is O(n).

2)

a) The number of visited nodes increased to 42 in game2 and 48 in game3. So the DFS algorithm worked better in this game. It is more difficult for the BFS algorithm to reach the points close to the corners than for the DFS algorithm. Because after visiting all the neighbors first, it moves to the next layer. DFS reaches edges and corners with fewer moves.
b) The number of nodes kept in memory has increased to 78 in game2 and 85 in game3. In other words, the DFS algorithm consumes less memory. Since BFS visited more nodes, it had to keep more nodes in memory.
c) Both were very close to each other. The DFS algorithm mostly worked a little longer.

3) When visited nodes are not kept, the program will be able to go to visited neighbors again and again. This will cause an infinite loop and the traverse process cannot be terminated. Therefore, keeping visited nodes is very important.

4) When the number of ships remains the same and their coordinates do not change, only the size board is increased. In this case, the BFS algorithm gave better results than the DFS algorithm and used less memory. Because DFS reached the front edges first, it traversed more when the sizeboard increased.