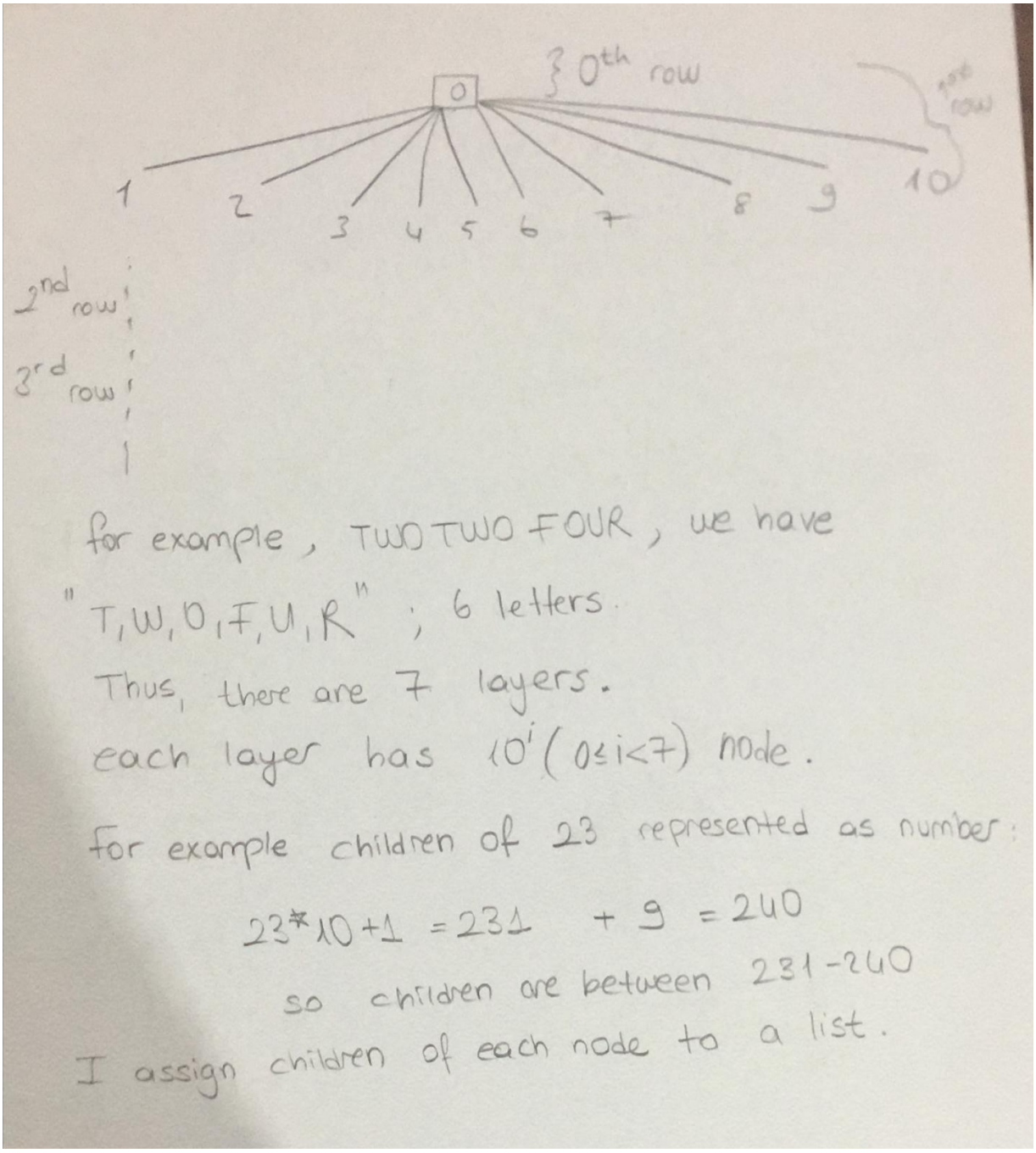
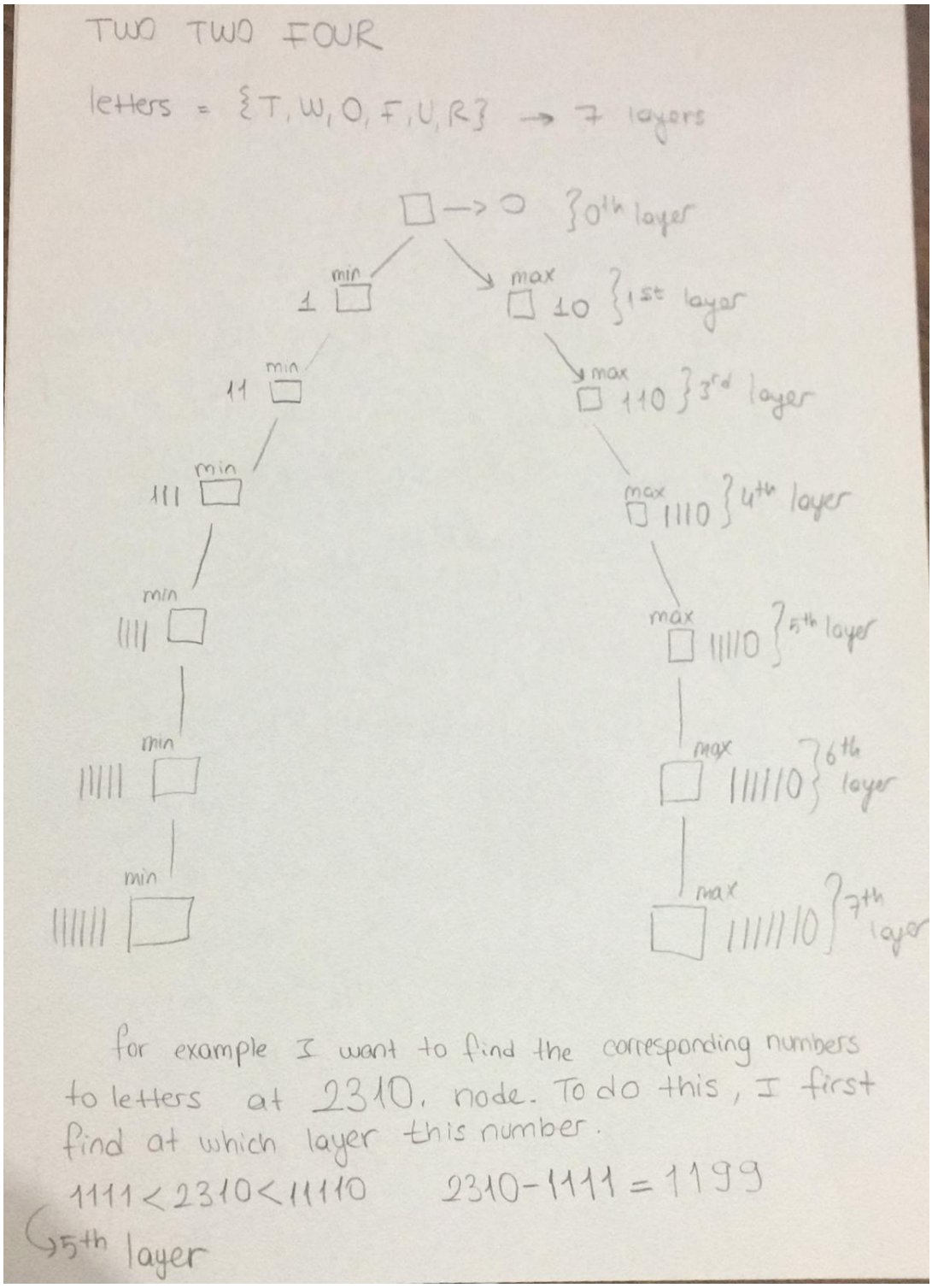
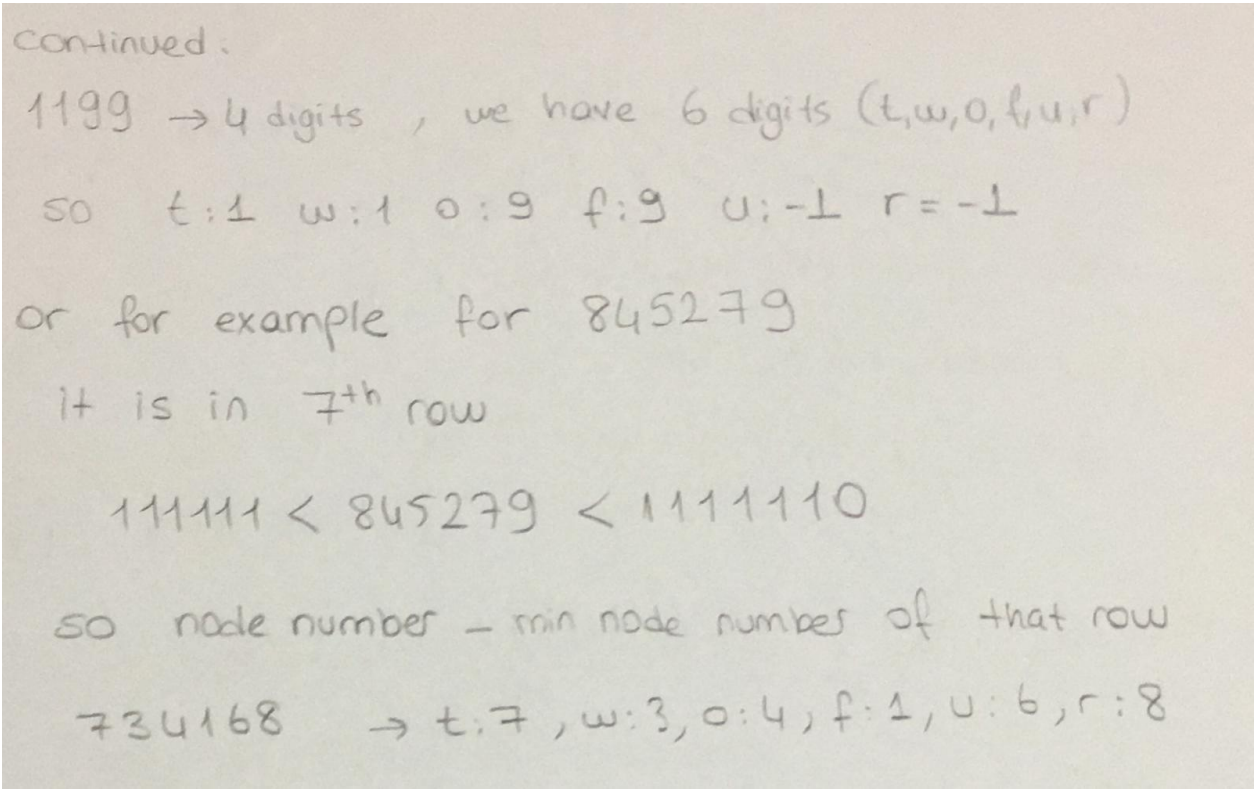
1.a) First of all, I created a class named Graph and created a variable named aNode in map type that holds a list for each element in this class. Then I added nodes to aNode according to the number of letters in the main function. Each map node keeps a list, each node has 10 children (they are kept as list). Here I thought of the tree as rows. For example, a row between 1-10 is,or a row between 11-110, my code hovers between the members of these rows and adds 10 children to all of them. I kept the nodes numerically instead of creating objects. For example, the children of the 2nd node are between 21-30. The children of the 10th node are 101-110, so by multiplying the node number by ten and adding 1, we can find the first child of that node, my code works with this logic. While creating the nodes in the code, I also do a check process, at the same time, for example, for the word 'TWO', 'T' can not be 0, checking these kind of constraints, I avoid memory-related problems by reducing the number of nodes.



I did not keep the numbers corresponding to the letters in every node, instead I used a different method. I kept the numbers of the nodes at the beginning and the end of each row in an array, for example below I explained the logic for two two four inputs.





1.b)

DFS

```
if(node is not found)
        node visited is true;
        create list named node;
        for(childs of the node)
                if(node is not visited before)
                        a = check if the node fits to searching criteria;
                        if( return value of a is 2)
                                increase the number of nodes visited by 1;
                                DFS(child, words array)
                        else if( return value of a -1 )
                                do nothing;
                        else
                                node is found
                                return;
```

BFS

```
if(node is not found)
        create list named list;
        mark the input node as visited
        push the node to the list;
        create list named node;
        while(there are still nodes to visit)
                assign the node to the front of the list;
                remove the front of the list;
                for(children of the node)
                        if(node is not visited before)
                                a = check if the node fits to searching criteria;
                                if( return value of a is 2)
                                        add node to the list;
                                        increase the number of nodes visited by 1;
                                        mark node as visited;
                                else if( return value of a -1 )
                                        do nothing;
                                else
                                        node is found
                                        return;
```

1.c) In DFS as can be seen in my pseudo code if the node fits the searching criteria it goes directly to its children without going to the sideways as it is in the BFS algorithm. We know that complexity is dependent on the number of vertices and edges. In DFS if the node does not comply with the searching criteria it does not branch in the tree towards its children, it goes back in the tree and visits the remaining. However, in BFS it does not go back and visits all nodes by traversing sideways, so more vertices and edges are created and visited, which increases the time complexity.

2.a) When the number of letters decreases, number of visited nodes also decreases.

According to my code results in BFS number of visited nodes is larger than it is in DFS.

2.b) Maximum number of nodes kept in the memory is same for DFS and BFS for the same input because I put some constraints when creating nodes, so in both search operations maximum number of nodes

remain the same, but with the increase in the letter number, maximum number of visited nodes also increases.

2.c) The time it takes to make DFS for the same input is less than BFS according to my results. As the number of letters in input increases, the search time increases for both search methods.

3) To keep track of the nodes that we visited, and preventing visiting repetitions we should maintain a list of discovered nodes.