# CMPE 343 hmw2

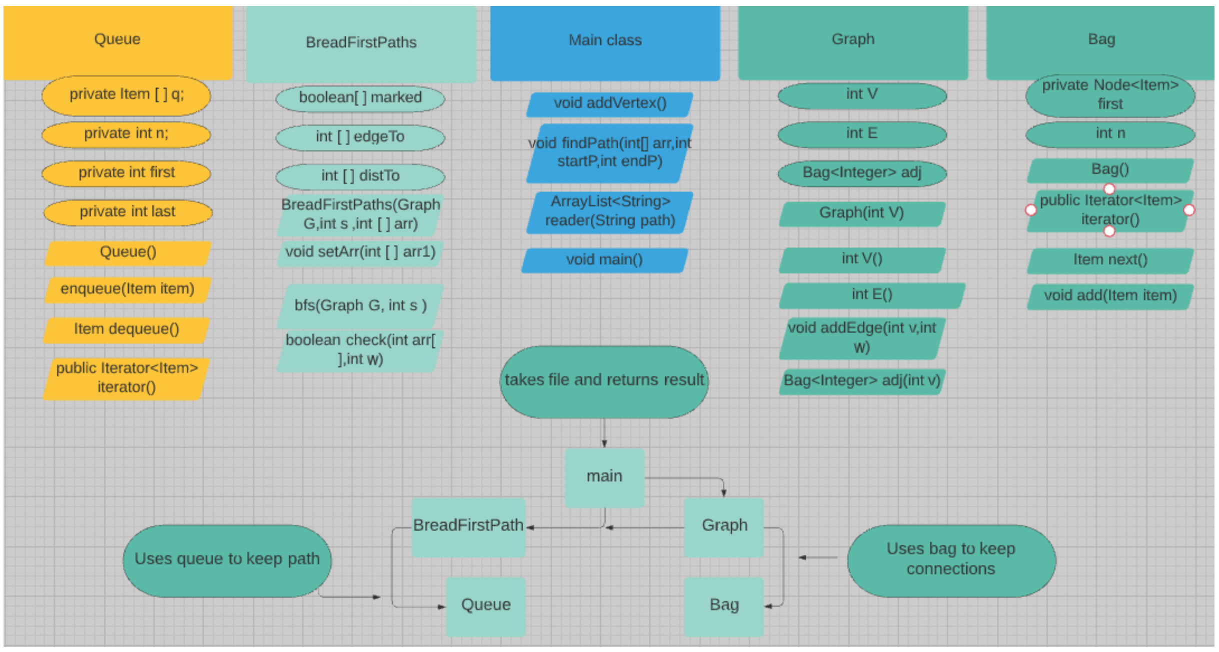
## Information:

**NAME: Abdusselam koç**
**SECTION: cmpe 343- section 2**
**ASSİGNMENT NUMBER: homework 2**

## Problem Statement and Code Design for the first part:

İn the first part the problem is finding shortest path from source number to the target number without passing the forbidden numbers. Firstly, we need to create connection to number for each number from 1-10000. Every number has different connections. For instance, 223 has connection with 224,222,233,213,323,123. The important part of this task is 0s and 9s because they cannot be reduced by one. After creating the graph, we need to find shortest path between two numbers therefore, we need to use bfs to find the lentgh of the shortest path. To find shortest path on without passing on dfs we need to some changes on dfs algorithm.

## Strucuture chart for the first part:



## Implementation and functionality for the first part:

Main class: creates the graph and searches for the shortest path.

addVertex (): adds vertex for each connection to the graph. Reader (): reads the input file and returns required numbers for connections and forbidden array.it sends the graph object to the bfs to get result and returns the result. BFS bfs = new Bfs (Graph g, int startingPoint, int EndPoint) ;

Graph class: creates an adjacency array to keep edges in the graph and adds vertexes for connection to the adj list. E() returns the numbers added to the list. V() returns the maximum length of the

Graph .addEdge : adds Edge to the graph.  public void addEdge (int v, int w) {adj(v).add(w); adj(w).add(v);}

Bag class: it is very useful to keep the graph's connections. It uses nodes to keep the edges.

Iterator<Item> iterator() : returns an iterator that iterates over the item in this bag in arbitrary order.

Add(): adds the item to the bag: public void Add(Item item){Node<Item> oldfirst =first;

first = new Node<Item> ();

first.item=item;

first.next=oldfirst;

n++;}

queue class: it is a helper class that been used by bread first search algorithm. It works in lifo order.

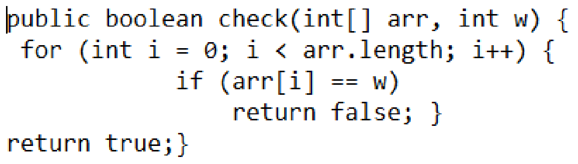
Qeueu(): constructor to initialize the queue class. Enqeueu() : adds items to the queue.

Dequeue (): takes item out of the queue. Dequeue(Item item){ find the last index of the q array and add the item to the q array. }

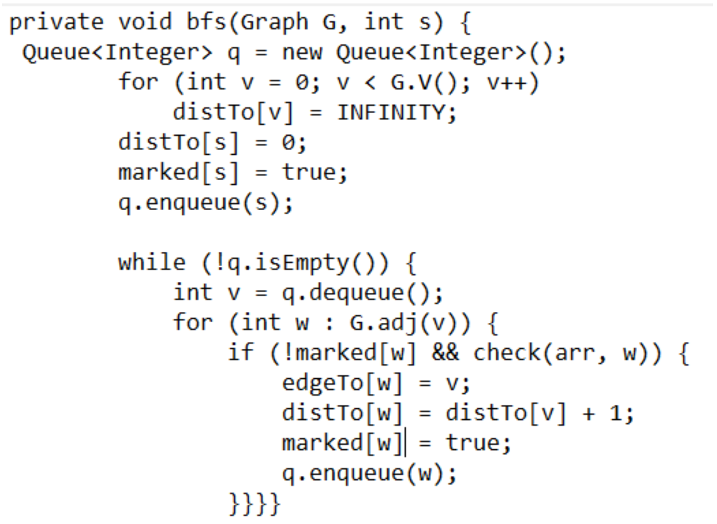
Breadfirstpath class : the class that includes bfs algorithm to calculate the shortest path between sources. BreadfirstPath(): initialize the required Boolean array and integer array according to the Graph that is going to be use in the alghoritm. Bfs () : calculates the shortest path between the edges.

In the modified method it checks whether the current number is forbidden or not than continue if not forbidden.

Check() : checks for the forbidden number and returns a Boolean according to forbidden number array.



```
public boolean check(int[] arr, int w) {
 for (int i = 0; i < arr.length; i++) {
           if (arr[i] == w)
               return false; }
return true;}
```



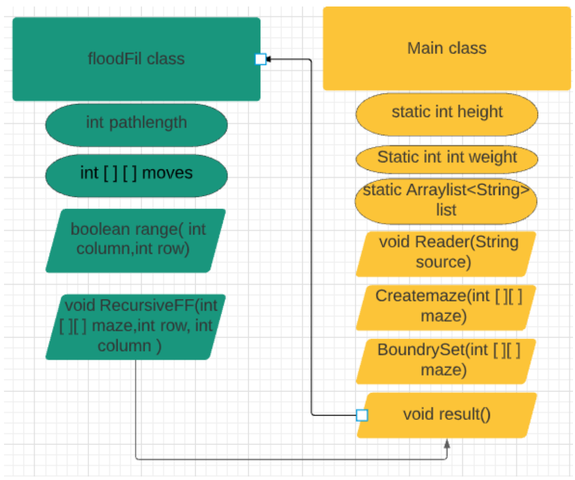
```
private void bfs(Graph G, int s) {
 Queue<Integer> q = new Queue<Integer>();
       for (int v = 0; v < G.V(); v++)
           distTo[v] = INFINITY;
       distTo[s] = 0;
       marked[s] = true;
       q.enqueue(s);

       while (!q.isEmpty()) {
           int v = q.dequeue();
           for (int w : G.adj(v)) {
               if (!marked[w] && check(arr, w)) {
                   edgeTo[w] = v;
                   distTo[w] = distTo[v] + 1;
                   marked[w] = true;
                   q.enqueue(w);
               }}}}
```

## Problem Statement and Code Design for the second part:

In the second part the problem is finding the longest cycle in the maze. We are going to create a maze with slashes and backslashes then try to find all cycles and longest cycles in the maze. The most efficient way to implement this is floodfill algorithm. It checks all possible ways to visit. If there is wall it stops if there is not any wall it continues. We are going to need a 2d array to create our maze with slashes and back slashes. After creating the array, we will try to eliminate the empty corners that there is no possibility to have a cycle. Finally, we are going to apply flood fill algorithm to check all cycles and their lengths.

## Implementation and functionality for the second part:


floodFil class
int pathlength
int [ ] [ ] moves
boolean range( int column,int row)
void RecursiveFF(int [ ][ ] maze,int row, int column )
Main class
static int height
Static int int weight
static Arraylist<String> list
void Reader(String source)
Createmaze(int [ ][ ] maze)
BoundrySet(int [ ][ ] maze)
void result()

Main class: this class reads the file then splits it according to our needs and creates the 2d maze array. After creating 2d array it calls floodfil algorithm and prints the result to the console.

Void Reader(): it reads the text file sets the height and width of the 2d array and fills the String Arraylist that keeps the slashes and backslahes.

Public void Reader(){

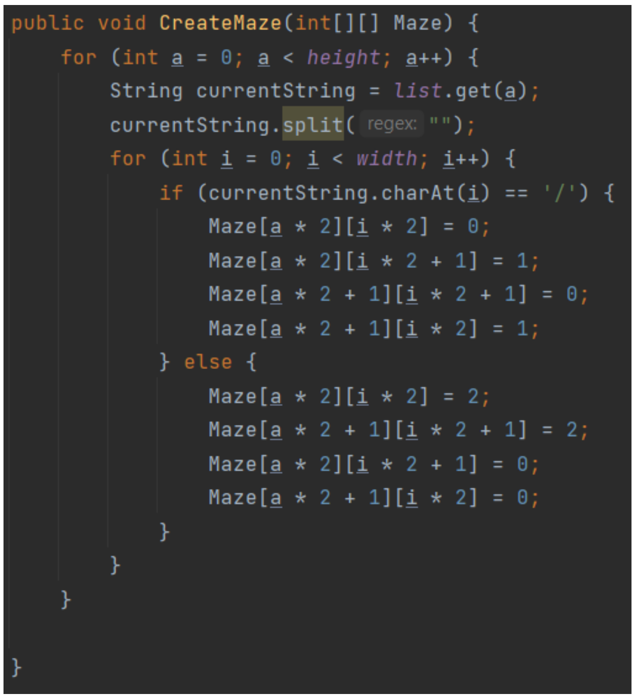
Height = second int;

Width= first int;

While(input not null)

List.add(input) }

Createmaze(int [ ] [ ] maze): takes a 2d array then fills it up with slashes and back slashes according to rules.



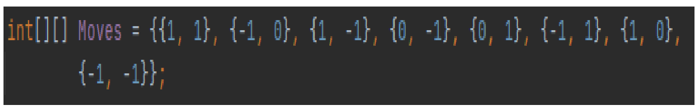
```
public void CreateMaze(int[][] Maze) {
    for (int a = 0; a < height; a++) {
        String currentString = list.get(a);
        currentString.split( regex: "");
        for (int i = 0; i < width; i++) {
            if (currentString.charAt(i) == '/') {
                Maze[a * 2][i * 2] = 0;
                Maze[a * 2][i * 2 + 1] = 1;
                Maze[a * 2 + 1][i * 2 + 1] = 0;
                Maze[a * 2 + 1][i * 2] = 1;
            } else {
                Maze[a * 2][i * 2] = 2;
                Maze[a * 2 + 1][i * 2 + 1] = 2;
                Maze[a * 2][i * 2 + 1] = 0;
                Maze[a * 2 + 1][i * 2] = 0;
            }
        }
    }

}
```

BoundrySet(int [ ][ ]): it checks for the boundaries that not connected and Set them as not connected in the graph. This features a more efficient run the algorithm.
Void Result (): calls the floodfil algorithm and returns the result. If the result is not equals to zero prints the number of the Cycles and length of the biggest cycle. If the result is zero prints, there are no cycles.

FloodFill class: it checks for cycles and if find a cycle counts it lengths. Otherwise, it returns 0.

Int moves( ): it is a 2d integer array and keeps the moves that possible in the 2d maze array.

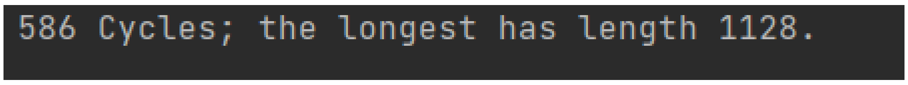


```
int[][] Moves = {{1, 1}, {-1, 0}, {1, -1}, {0, -1}, {0, 1}, {-1, 1}, {1, 0},
        {-1, -1}};
```

Every index represents a direction to go in the array. Since it can go cross the corners, we need include them as well.
Void RecursiveFF(): it takes the maze array and checks for every possible way to go. It works recursively and it is very complex. Range() : checks for the ways whether there are in the maze or not.

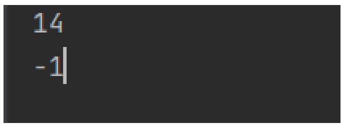
## Testing:

My code provides a tester class for both part1 and part2. İn both tester classes it takes different input apart from givens and returns the result of the inputs. According to the tests that I found on the internet my calculations are true. Since, the results are the same with the given on the internet I can say that my programs working correctly.



```
586 Cycles; the longest has length 1128.
```

my tester class output for the slash maze question.



```
14
-1
```

my tester class output for the wheel question.

## Final Assessments:

The trouble points this assignment was to have two assignments in one assignment. Because it took more time that I would be able to spend during preparation of a homework. It cut my other studies that I must do. Therefore, I would like to say that this homework was very long and very hard for me.

The second part was most challenging part for me. Because I needed to figure out how the mazes working and had to learn flood fill algorithm to solve the question. I still do not understand it properly, but I think as times goes it will better for me.

I like first question of the assignment because it wasn't so hard, and it taught me how to use bfs algorithm.