Instructions:-

1) File is GraphTraversal.py
2) Command to run is

```
Python GraphTraversal.py <input_text_file> <output_solution_file>
```

3) To verify output

```
Python verifyGraph.py <input_text_file> <output_text_file>
```

Answer (a):

Graph : directed cyclic weighted graph

Vertices : nodes are either R or B

Edges : directed edges with unit weights as N,S,E,W,NW,NE,SW,SE

Algorithm : Depth first search algorithm with backtracking

Answer (b):

Pseudocode

Input : input file text

Output : path traversed

Algorithm: GraphTraversal

```
Global list Path_taken  , color_taken
Function isvalid(i,j,m,n)
        If i,j <0 or >= (m,n)
                Return false
        Return true
Function get_next_Step(direction,I,j)
        if(direction=='N'):
                return i-1,j
        elif(direction=='E'):
                return i,j+1
        elif(direction=='S'):
                return i+1,j
        elif(direction=='W'):
            return i,j-1
        elif(direction=='NE'):
            return i-1,j+1
        elif(direction=='SE'):
            return i+1,j+1
        elif(direction=='SW'):
            return i+1,j-1
        else:
            return i-1,j-1
function traversal(input_mat, visited, i, j, current_colour, current_direction, m, n)
        global variables path_taken,color_taken
        if input_mat[i][j] == 'O':
            return True
        found_path = False
        while True
            i, j = get_next_step(current_direction, i, j)
            if isvalid(i,j,m,n) then
                get cell value at i,j
                if cell value = ‘O’ then
                    return true
                new_color,new_direction = cell[0],cell[1]
                if new_color not equal to current_color and not visited[i][j]
                    set visited[i][j] to true
                    append new_direction to path_taken list
                    append new_color to color_taken list
                    ## recursive calls to traversal algorithm
                    found_path = traversal(input_mat, visited, i, j, new_colour, new_direction, m, n)
                    if found_path:
                        return found_path
                    else
                        while (path_taken[-1] != current_direction or colour_taken[-1] != current_colour)

                    remove last value from from path_taken and color_taken lists if different direction or color
                ##if color equal or visited node already
                append current_direction and current_color to path_taken and color_taken list
                else
                    append current_direction and current_color to path_taken and color_taken list
            else
                return False not a valid a direction
        return found_path

main function():
        open input file text and load contents to input_matrix of size given in text file mxn
        initialize visited matrix of size mxn with zeros since no nodes are visited at beginning
        initialize indices i,j =0
        split contents of input_matrix[i][j] to cell[0], cell[1]
        initialize current_color, current_direction to cell[0],cell[1]
        initialize path_taken list and color_taken list with current_direction,current_color resptively
        ##call traversal function with below variables
        found_path = traverse(input_mat, visited, i, j, current_colour, current_direction,
         m, n)
        ##formatted path taken as required
        path_taken_formatted = ""
        cur_direction = ''
        cur_colour = ''
        counter = 0
        for direction, colour in zip(path_taken, colour_taken):
            if cur_direction == '':
                cur_direction = direction
                cur_colour = colour
            elif (cur_direction != direction or cur_colour != colour):
                path_taken_formatted += ' ' + str(counter) + cur_direction
        counter = 0
                cur_direction = direction
                cur_colour = colour
            counter += 1
        if counter >= 1:
            path_taken_formatted += ' ' + str(counter) + cur_direction
        else:
            path_taken_formatted += ' ' + cur_direction

        write path_formatted to output text file
```

Answer (c):

Lines	Time complexity
Function isvalid	$\Theta(1)$
Function getnextstep	$\Theta(1)$
Function Traversal	$mnT(n-1) + \Theta(1)$
Main function	$\Theta(n^2) + T(n) + \Theta(n)$
Total complexity	$mnT(n-1) + 2* \Theta(n^2) + \Theta(n) = mnT(n-1) + \Theta(n^2)$
Estimated time complexity	$n^2T(n-1) + \Theta(n^2)$