GOLYGON

Batch-6 Members

G.Priyanka 21B01A0552 CSE
D.Devisree 21B01A0540 CSE
M.Kanthi 21B01A0462 ECE
D.Anshu 21b01a0414 ECE
A.Likhitha 21b01a0101 CE

SVECW

January 28, 2023

Intoduction About The Project

- The target is to construct all possible routes for a tourist visiting a grid-like city,taking into account blocked intersections and incresing step lengths.
- Input includes the number of cities to be visited, the length of the longest edge of the golygon for each city, and the coordinates of any blocked intersections.Output includes a list of possible golygons for each city and the number of solutions found.

Example of sample input

```
1
8
2
-2
0
6
-2
```

- The above values are given as the input, then the output produces the directions to be followed and the number of Possible golygon(s).

Output

```
wsenenws
Found 1 golygon(s)
```

Approach

- The problem can be solved by using a Depth-First Search (DFS) algorithm
- To implement this in code, we can generate a recursie function that takes in the current position, edge length, and direction as parameters, and uses a stack to keep track of the path. The function should return the list of all possible golygons for a given city.

Learnings

- The way of solving a problem that involves finding all possible paths on a grid.
- The good understanding of graph theory and pathfinding algorithms in order to implement a solution.
- The way of handling input and output in a specific format.
- Package: NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python

Challenges

- **Generating all possible golygon configurations:** There could be a large number of possible golygon configurations that need to be considered, making it computationally expensive.
- **Meeting all the restrictions:** Ensuring that all the intersections that are blocked are avoided in the final solution could be a complex task.
- **Finding the optimal solution:** Given the number of blocked intersections, the developer needs to find the best solution that satisfies all the restrictions and maximizes the size of the golygons.
- **Optimizing the performance:** As the number of blocked intersections increases, the number of possibilities to check will also increase, making the algorithm very slow. The developer need to make sure that the algorithm is optimized for performance.

Challenges

- **Handling multiple cities:** The program should be able to handle multiple cities, each with its own set of blocked intersections and maximum golygon size.
- **Handling large input:** The program should be able to handle large inputs such as a large number of cities, a large number of blocked intersections, and a large maximum golygon size.
- **Output format:** The program should be able to output the solutions in the required format, which includes the sequence of characters and the number of solutions found for each city.

Statistics

Number of Lines of code

132

Number of Functions used

7

Functions Used

def direction(x, y, pos)

This Function is used to check if the current position is the endpoint and if it is then it will increment the number of possible paths and add the direction of the path to a string variable result.

def is visited(xx, yy)

This Function is used to track if the current cell has been visited or not

def mark as visited(xx, yy)

This Function is used to mark the cell as visited

def mark as unvisited(xx, yy)

This Function is used to mark the cell as unvisited

Functions Used

def dfs(x, y, pos, dir)

This Function is the main function that is called recursively to navigate through the grid. It takes in four parameters: the current x and y coordinates, the current position in the path, and the current direction.

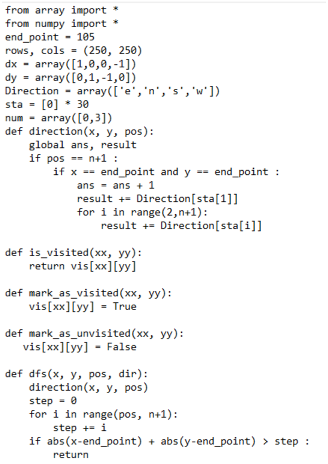
def create grid()

This Function is used to create the grid

def create visited grid()

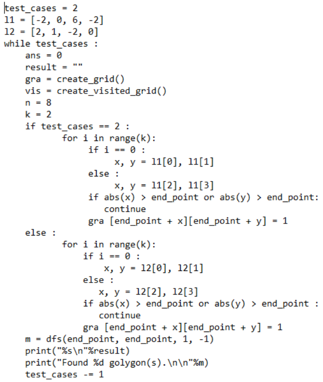
This Function is used to create the visited grid.

Demo of the Project
Screen Shots of the Project

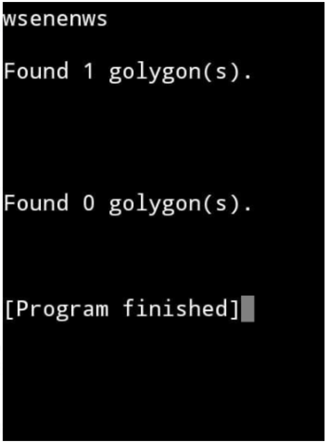


```
from array import *
from numpy import *
end_point = 105
rows, cols = (250, 250)
dx = array([1,0,0,-1])
dy = array([0,1,-1,0])
Direction = array(['e','n','s','w'])
sta = [0] * 30
num = array([0,3])
def direction(x, y, pos):
    global ans, result
    if pos == n+1 :
        if x == end_point and y == end_point :
            ans = ans + 1
            result += Direction[sta[1]]
            for i in range(2,n+1):
                result += Direction[sta[i]]

def is_visited(xx, yy):
    return vis[xx][yy]

def mark_as_visited(xx, yy):
    vis[xx][yy] = True

def mark_as_unvisited(xx, yy):
   vis[xx][yy] = False

def dfs(x, y, pos, dir):
    direction(x, y, pos)
    step = 0
    for i in range(pos, n+1):
        step += i
    if abs(x-end_point) + abs(y-end_point) > step :
        return
```

```
if dir == -1 :
        for i in range(4):
            xx = x + pos*dx[i]
            yy = y + pos*dy[i]
            if is_visited(xx, yy):
               continue
            k = 0
            if 1 <= i and i < 3 :
                for k in range(min(y, yy),max(y, yy)+2):
                    if gra[xx][k]:
                        break
                if k == max(y, yy) + 1 :
                    sta[pos] = i
                    mark_as_visited(xx, yy)
                    dfs(xx, yy, pos+1, 1)
                    mark_as_unvisited(xx, yy)
            else:
                for k in range(min(x, xx), max(x, xx)+2):
                    if gra[k][yy] :
                        break
                if k == max(x, xx) + 1 :
                    sta[pos] = i
                    mark_as_visited(xx, yy)
                    dfs(xx, yy, pos+1, 0)
                    mark_as_unvisited(xx, yy)
else:
        if dir == 0 :
            for i  in range(1,3):
                xx = x + pos*dx[i]
                yy = y + pos*dy[i]
                if is_visited(xx, yy):
                    continue
                    k = 0
                    for k in range(min(y, yy), max(y, yy)+2):
                        if gra[xx][k] :
                           break
                    if k == max(y, yy) + 1 :
                        sta[pos] = i
                        mark_as_visited(xx,yy)
                        dfs(xx, yy, pos+1, 1)
                        mark_as_unvisited(xx, yy)
            elif dir == 1 :
                for j  in range(2):
                    i = num[j]
                    xx = x + pos*dx[i]
                    yy = y + pos*dy[i]
                    if is_visited(xx, yy):
                       continue
                    k = 0
                    for k in range(min(x, xx), max(x, xx)+2):
                        if gra[k][yy]:
                           break
                    if k == max(x, xx) + 1 :
                        sta[pos] = i
                        mark_as_visited(xx, yy)
                        dfs(xx, yy, pos+1, 0)
                        mark_as_unvisited(xx, yy)
    return ans
def create_grid():
    a = [0] * 62500
    return reshape(a, ( 250, 250))

def create_visited_grid():
    b = [False] * 62500
    return reshape(b, (250, 250))
```



```
test_cases = 2
l1 = [-2, 0, 6, -2]
l2 = [2, 1, -2, 0]
while test_cases :
    ans = 0
    result = ""
    gra = create_grid()
    vis = create_visited_grid()
    n = 8
    k = 2
    if test_cases == 2 :
           for i in range(k):
                if i == 0 :
                     x, y = l1[0], l1[1]
                else :
                     x, y = l1[2], l1[3]
                if abs(x) > end_point or abs(y) > end_point:
                   continue
                gra [end_point + x][end_point + y] = 1
    else :
           for i in range(k):
               if i == 0 :
                   x, y = l2[0], l2[1]
               else :
                  x, y = l2[2], l2[3]
               if abs(x) > end_point or abs(y) > end_point :
                  continue
               gra [end_point + x][end_point + y] = 1
    m = dfs(end_point, end_point, 1, -1)
    print("%s\n"%result)
    print("Found %d golygon(s).\n\n"%m)
    test_cases -= 1
```



```
wsenenws

Found 1 golygon(s).



Found 0 golygon(s).


[Program finished]
```

Thank You