Assignment No.3

Introduction to Algorithms

Path planning using wavefront Algorithms

By:

Ahmed Mohmed Bn: 6

Eman Emad Bn: 13

Delivered to:

Eng. Yara El-shamy

Prof. Inas Yassine

Contents

Introduction

Our code implements a utility that can be used to create an environment (map) with its obstacles, goal, and boundaries, but there are two basic functions and capable of placing a destination (goal) and find the shortest path from any starting point on the map if possible.

Our code consists of mainly one class (Environment) that wraps in all the needed functions and attributes. There are two main functions that contribute to the path planning algorithm. (start_wave, get_shortest_path). And their explanation is below.

Note: there is a .py file has the class implementation and a Jupiter notebook has the visualization with the report.

All the code are provided here also: github.com/AhmedXAlDeeb/wavefront_task3-algorithms

Files:

pathGenerator.py: have the main class and the testcase function

plotting.ipynb: has the plotting examples and the big maze test case with visualization

wavefrontGUI.py is not completed as it was planned to be like the website provided in the statement

Problem Analysis

We divided the problem into several steps and made it as general as possible, so we created the class at first which make a map (2d matrix) and place obstacles in it. Then we worked on the algorithm that fills the map which took a little bit of time.

To finish the function of the map filling (wavefront) we revisited graph traversal algorithms (bfs, dfs) also revisited some tutorials about path planning and shortest path algorithms. Also, we learned for the first time about using the directions vector.

Then we implemented the shortest path finder function which simply iterates over the cell and follow the minimum values.

Obstacles Faced

- Understanding the problem and differentiating between the wavefront algorithm and other algorithm.
- Forget to take obstacles in consideration while wave propagation
- Implementing the wavefront algorithm wasn't straight forward as we needed to enhance our implementation several times and use the direction vector.
- Silly mistakes like initializing the environment(map) with wrong obstacle positions making it generate different values. Forget to set up the default values.
- The time complexity of the mat file which needed to run locally.

Main Functions Explanation

Start wave function:

The start wave function is a crucial part of the wavefront algorithm, which is used in path planning for grid-based environments. It computes distances from a goal position and determines the shortest path to the goal.

Importance of start wave:

- Environment filling: The function assigns values to each cell in the grid based on its distance from the goal. The farther a cell is from the goal, the higher its value.
- Avoiding obstacles: The wavefront propagates around obstacles, effectively marking them as impassable areas in the grid. This ensures the shortest path computation respects environmental constraints.
- Pathfinding: The wavefront values guide the shortest path computation. A path from a starting position to the goal can be traced by following the gradient of decreasing values.

How it works:

- The function starts by ensuring that a goal is set in the environment and initialize it with value of two.
- a queue is initialized with the goal position Each cell in the queue propagates its wave value to its neighbors in all eight possible directions (cardinal and diagonal directions).
- For each neighbor cell the algorithm checks its bounds and if it is free.
- Valid neighbors are updated with the incremented wave value and added to the next queue.
- The wave value is incremented after checking all neighbors in the current queue.
- The queue is updated to next queue.
- The function ends when the queue becomes empty, meaning all reachable cells have been assigned wave values.

Big-O complexity:

Size = Rows*columns

O(size)

Pseudocode:

```
Function StartWave( ):

  if Goal-flag is False:
      raise "Goal not set"

  wave-value ← 2
  env[goal-pos] ← wave-value
  queue ← [goal-pos]

  while queue is not empty:

      next queue ← [ ]

      for each element in queue:
          for each direction in directions:
              neibour ← element + direction
              if neibor in bounds and Free cell:
                  neibour ← wave-value + 1
                  next queue ← [neibour]
      wave-value ← wave-value + 1
      queue ← next queue
```

Get shortest path function:

Function is a critical part of the wavefront algorithm that computes the optimal path from a given starting point to the goal. Its able to leverage the wave values generated by the start wave function to efficiently backtrack and construct the shortest path.

Importance:

- Path Extraction: After wave propagation, the grid contains distance values from the goal. This function uses those values to extract the shortest path by following the gradient of decreasing values.
- Optimal path: It ensures that the path found is the shortest possible route to the goal while avoiding obstacles.
- Dynamic environment: The function works well in environments where obstacles or grid configurations can change, if start wave is re-run before pathfinding.

How it works:

- The function first ensures the starting point is valid (not an obstacle) and lies within the propagated wavefront (value > 1).
- The path starts with the given start position.
- A variable current is initialized with start to track the current cell being processed.
- Check all possible neighbors (cardinal and diagonal directions).
- Identify the neighbor with the smallest wave value, but greater than one.
- Add the neighbor with the smallest wave value to the path.
- Update current to this neighbor.
- The loop terminates when current reaches the goal position. At this point, the path is complete.
- The function returns the constructed path as a list of grid coordinates from the start to the goal.

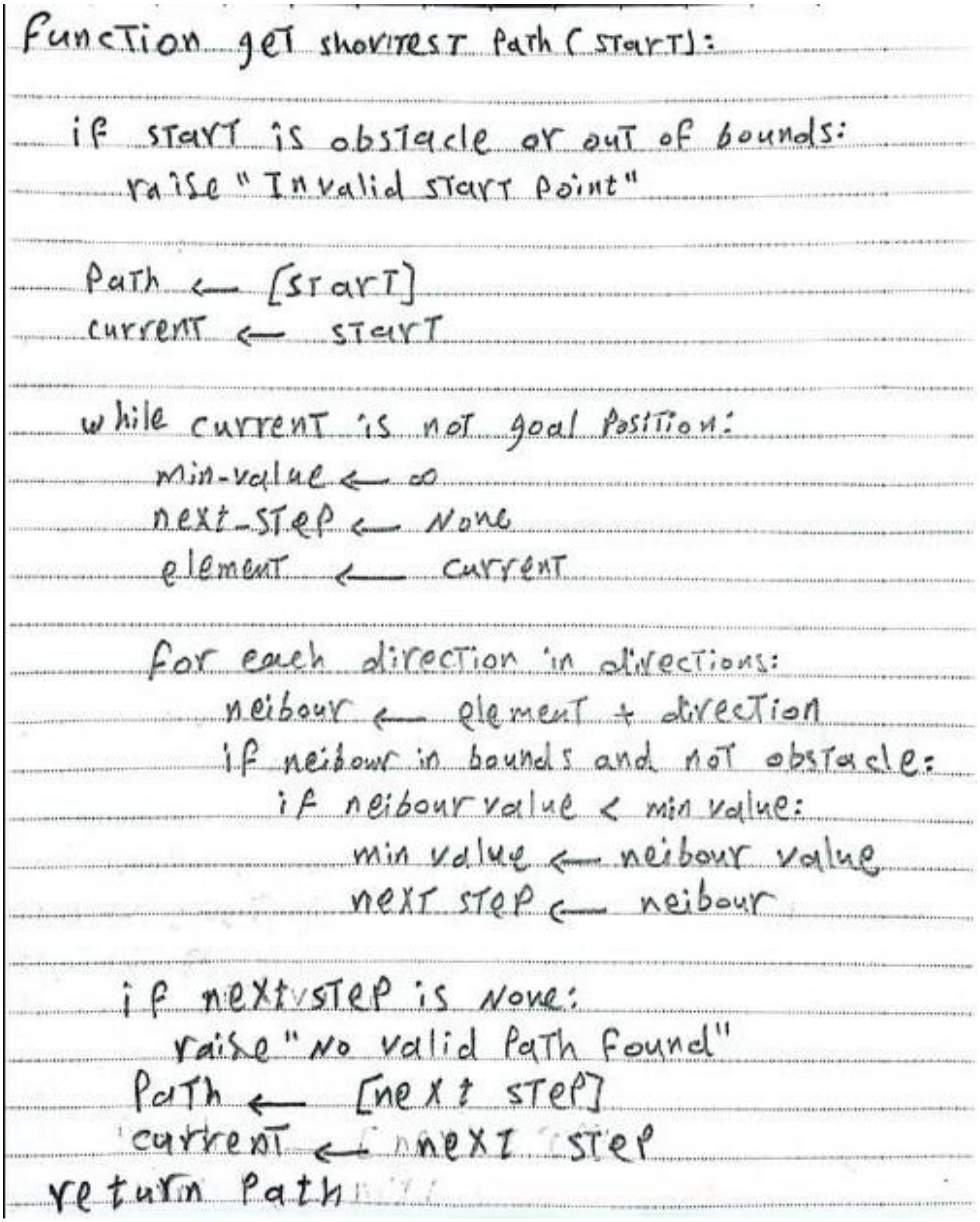
Big-O complexity:

P (path length)

O(P)

The Total Big-O of algorithm is O(size)

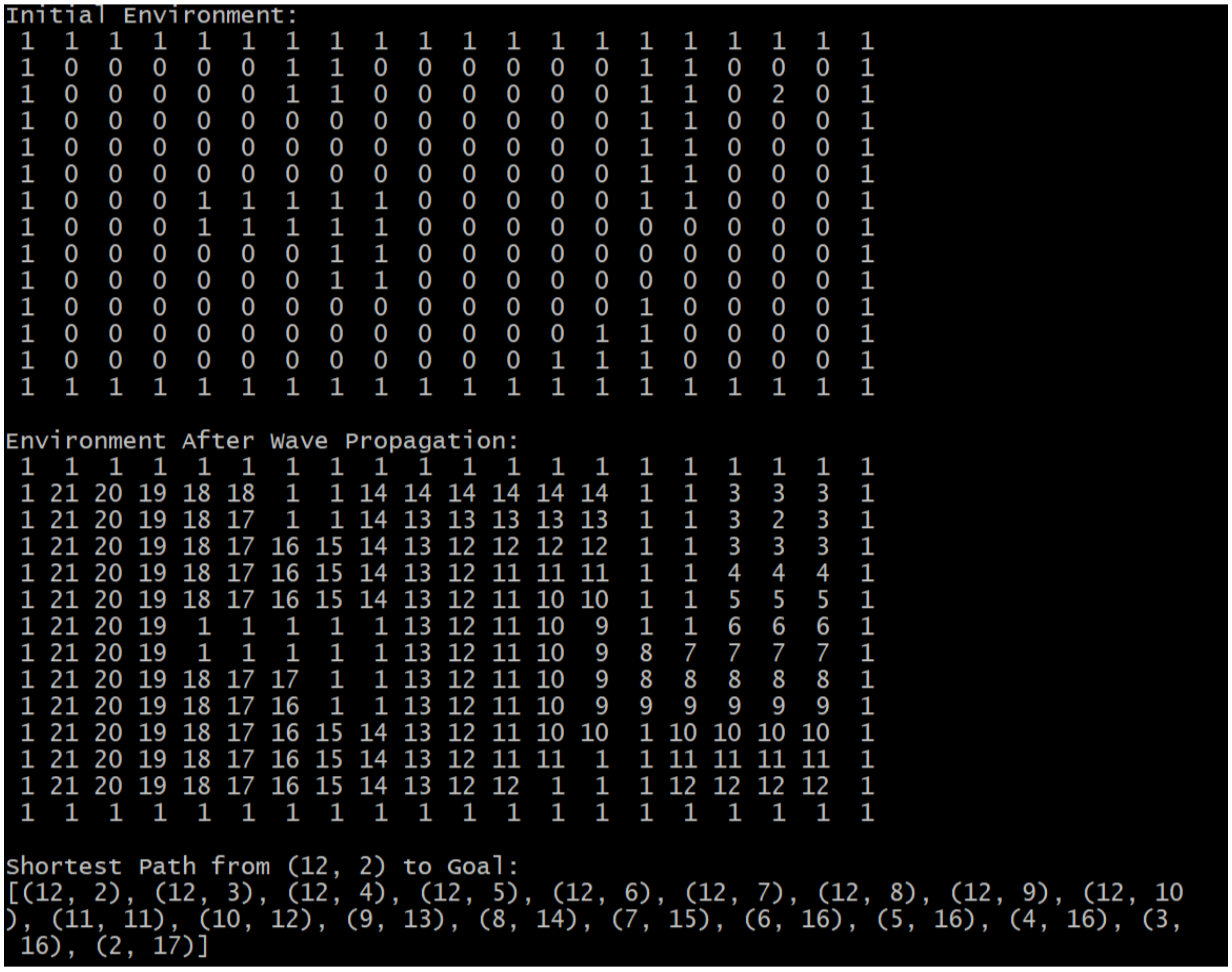
Pseudocode:



```
function get shortest path(start):

  if start is obstacle or out of bounds:
      raise "Invalid start point"

  path ← [start]
  current ← start

  while current is not goal position:
      min-value ← ∞
      next_step ← None
      element ← current

      for each direction in directions:
          neibour ← element + direction
          if neibour in bounds and not obstacle:
              if neibour value < min value:
                  min value ← neibour value
                  next step ← neibour

      if next_step is None:
          raise "No valid path found"
      path ← [next step]
      current ← next step
  return path
```

Testcases figures and visualizations

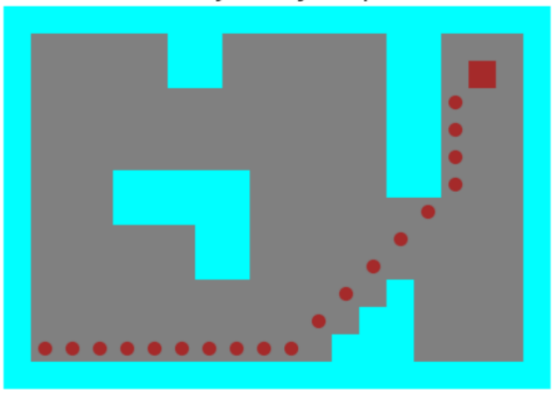
Original test case (20x14):



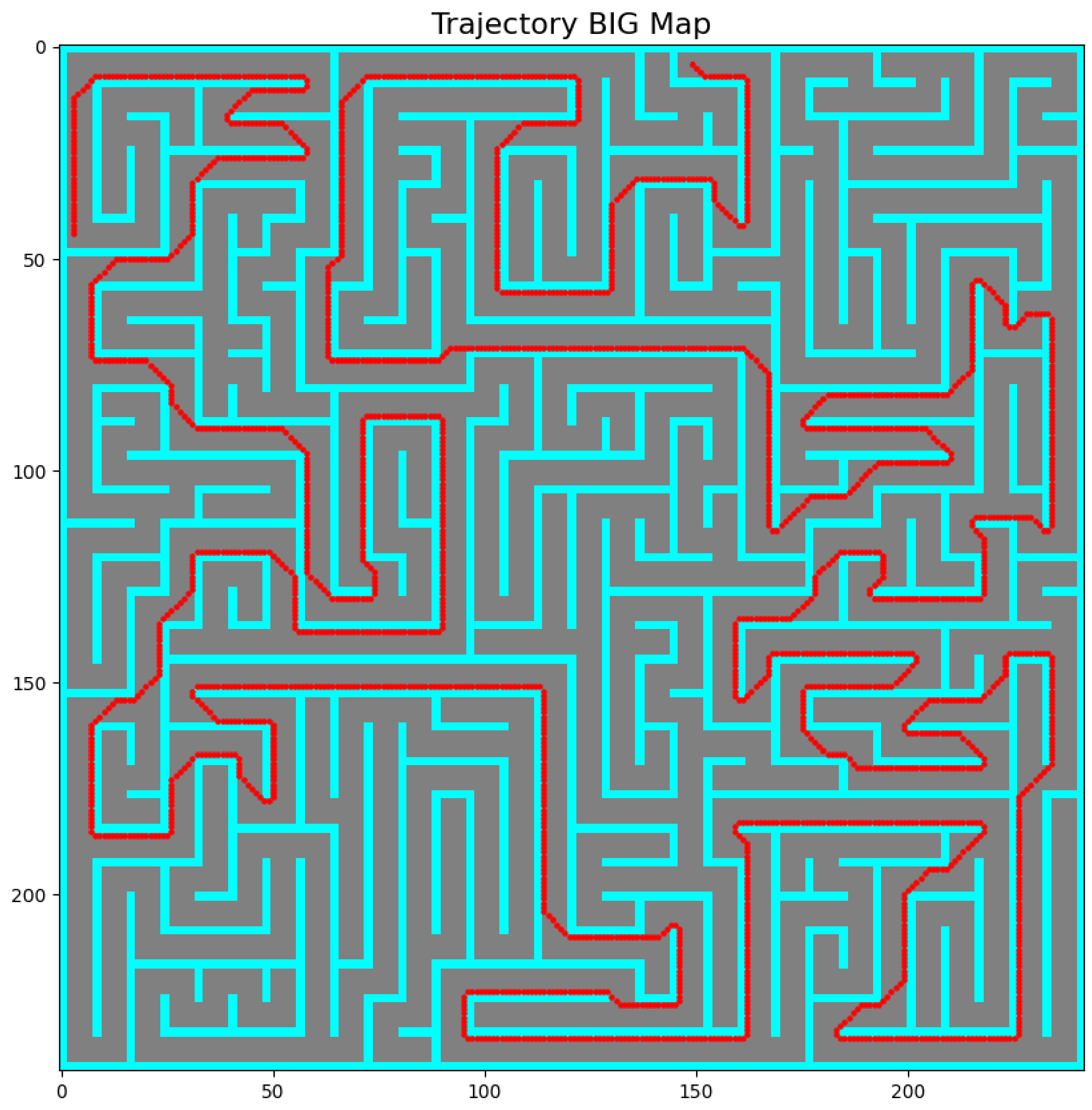
```
Initial Environment:
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  0  0  0  0  0  1  1  0  0  0  0  0  0  1  1  0  0  0  1
  1  0  0  0  0  0  1  1  0  0  0  0  0  0  1  1  0  2  0  1
  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  1
  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  1
  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  1
  1  0  0  0  1  1  1  1  1  0  0  0  0  0  1  1  0  0  0  1
  1  0  0  0  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  1
  1  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  1
  1  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  1
  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  1
  1  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  1
  1  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1

Environment After Wave Propagation:
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1 21 20 19 18 18  1  1 14 14 14 14 14 14  1  1  3  3  3  1
  1 21 20 19 18 17  1  1 14 13 13 13 13 13  1  1  3  2  3  1
  1 21 20 19 18 17 16 15 14 13 12 12 12 12  1  1  3  3  3  1
  1 21 20 19 18 17 16 15 14 13 12 11 11 11  1  1  4  4  4  1
  1 21 20 19 18 17 16 15 14 13 12 11 10 10  1  1  5  5  5  1
  1 21 20 19  1  1  1  1  1 13 12 11 10  9  1  1  6  6  6  1
  1 21 20 19  1  1  1  1  1 13 12 11 10  9  8  7  7  7  7  1
  1 21 20 19 18 17 17  1  1 13 12 11 10  9  8  8  8  8  8  1
  1 21 20 19 18 17 16  1  1 13 12 11 10  9  9  9  9  9  9  1
  1 21 20 19 18 17 16 15 14 13 12 11 10 10  1 10 10 10 10  1
  1 21 20 19 18 17 16 15 14 13 12 11 11  1  1 11 11 11 11  1
  1 21 20 19 18 17 16 15 14 13 12 12  1  1  1 12 12 12 12  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1

Shortest Path from (12, 2) to Goal:
[(12, 2), (12, 3), (12, 4), (12, 5), (12, 6), (12, 7), (12, 8), (12, 9), (12, 10
), (11, 11), (10, 12), (9, 13), (8, 14), (7, 15), (6, 16), (5, 16), (4, 16), (3,
 16), (2, 17)]
```

Plotting:

Trajectory Map



The matfile



Resources used:

www3.cs.stonybrook.edu/~bghai/docs/Wave Front Method Based Path Planning Algorithm for Mobile Robots.pdf opus.lib.uts.edu.au/bitstream/10453/30533/1/quinACRA2014.pdf

https://www.youtube.com/watch?v=GMC4M14gTOA&t=250s&pp=ygUXd2F2ZWZyb250IHBhdGggcGxhb
m5pbmc%3D

https://www.youtube.com/watch?v=GMC4M14gTOA&t=250s&pp=ygUXd2F2ZWZyb250IHBhdGggcGxhb
m5pbmc%3D