zs2489 Zhou Shen
yx2725 YaoYao Xu

CAofR HW2

COMS 4733
October 18, 2021

Problem 1: Infinite Wall

Consider a point robot on the x-y plane. It is currently sitting at some x ¡ 0, left of an infinitely long vertical wall located at x = 0. The goal is located to the right of the wall at (2, 0).

(a) The goal induces an attractive conic potential function with $\zeta = 1$ everywhere in the plane. The wall induces a repulsive potential function to the left of it with $\eta = 1$ and $Q^* = 1$. Compute the locations of all critical points and show whether each is a local minimum, local maximum, or saddle point (you may either compute the Hessian or rely on an intuitive explanation).

First, the intuitive explanation. Because the potential diagram described above is symmetric along the x axis, the critical points will be on the x axis. So we only calculate the gradient along the x axis. Furthermore, we narrowed down the x range to [-1,0) because only in the junction of attractive potential and repulsive potential there exists the critical points.

Second, compute the value when the gradient expression equals to zero along the x axis in the range of [-1,0) (c = (0,0)):

$$\begin{aligned}\nabla U_t(q_x) = \nabla Ua(q_x) + \nabla U_r(q_x) &= 0\\ \frac{\zeta}{d(q_x,q_{gx})}(q_x - q_{gx}) + \frac{\eta}{d^2(q_x)}\left(\frac{1}{Q^*} - \frac{1}{d(q_x)}\right) * \frac{q_x - c_x}{d(q_x,c_x)} &= 0\\ \frac{1}{\sqrt{(x-2)^2}}(x-2) + \frac{1}{(|x-0|)^2}(\frac{1}{1} - \frac{1}{|x-0|}) * \frac{x-0}{\sqrt{(x-0)^2}} &= 0\\ \frac{1}{x^3} + \frac{1}{x^2} + 1 &= 0\\ x_{real} &\approx -0.68233\end{aligned} \tag{1}$$

So, point(-0.68233,0) is the critical point.

Finally, use Hessian to show whether the critical point is a local minimum, local maximum, or a saddle point. The potential function along the x axis in the range of [-1,0) is:

$$\begin{aligned}U(q) &= U_a(q_x) + U_r(q_x)\\ &= \zeta d(q_x,q_{gx}) + \frac{1}{2}\eta(\frac{1}{d(a_x)} - \frac{1}{Q^*})^2\\ &= \sqrt{(x-2)^2} + \frac{1}{2}(\frac{1}{-x} - \frac{1}{1})^2\end{aligned} \tag{2}$$

Hessian Matrix with respect to x is: $\left(\frac{1}{x^4} - \frac{2(-\frac{1}{x}-1)}{x^3}\right)$ and Hessian determinant is: $\frac{2x+3}{x^4}$. Put the critical point x = -0.68233 in Hessian determinant and the second partial derivatives ($\left(\frac{1}{x^4} - \frac{2(-\frac{1}{x}-1)}{x^3}\right)$), we get $det(H) > 0$ and $\left(\frac{1}{x^4} - \frac{2(-\frac{1}{x}-1)}{x^3}\right) > 0$, so we know that critical point(-0.68233, 0) is local minimum.

(b) The below figure is the path of robot with infinite wall.

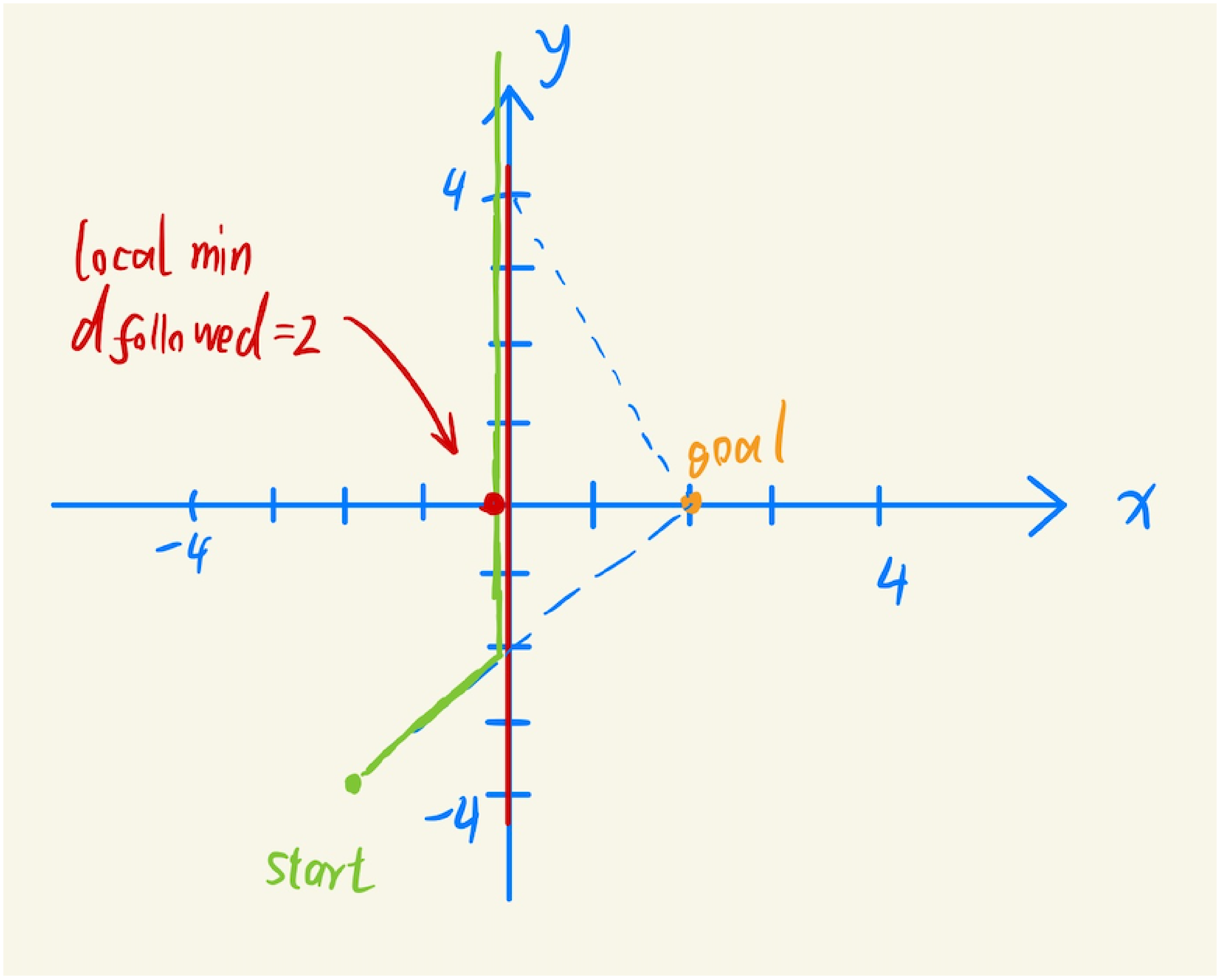


Figure 1: Problem 1b, Path with infinite wall

The red dot in figure above indicates the start of "follow obstacle boundary", which lies just a step above the point(0,0). In that position, the $d(x,n)+d(n,q_{goal}$ increases. Also, the $d_{followed} = 2$ updated, which computes the minimum between the sensed boundary and goal. Because the wall is infinite, the robot will keep moving upward forever, and no more updated $d_{followed}$ any more.

(c) The below figure is the path of robot with infinite wall only in one side.

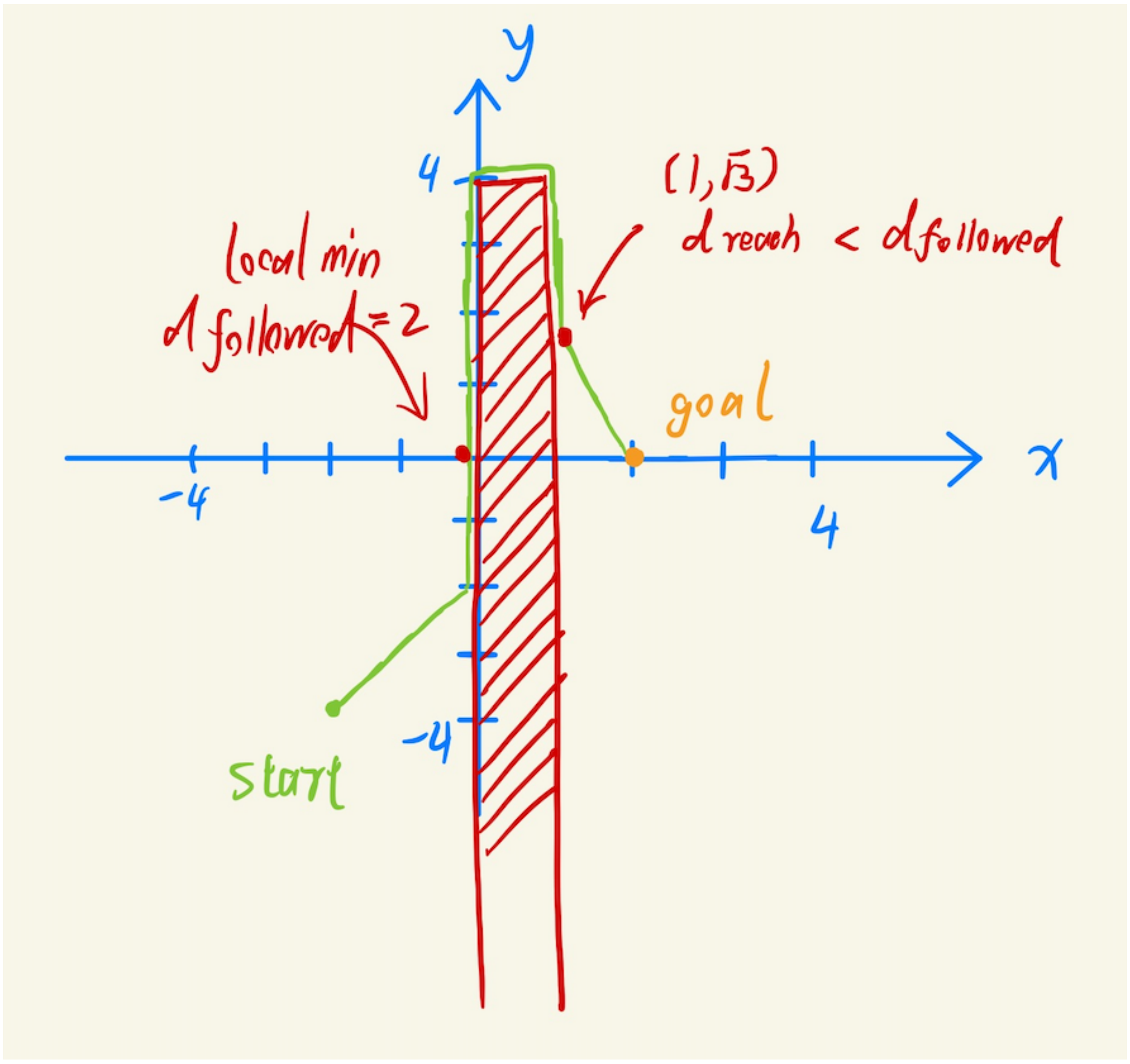


Figure 2: Problem 1c, Path with infinite wall in one side
The left red dot in figure above indicates the start of "follow obstacle boundary", the same position in QUESTION 1b, and $d_{followed} = 2$.Then, the robot will follow the obstacle. The right red dot indicates the the position that $d_{reach} < d_{followed}$, which is $(1, \sqrt{3})$. Then the robot will switch to "Head toward goal" until reach the goal.

Problem 2: Grid Search

(a) The cells that may be expanded and their f-values in the priority queue is,

cell	(4,2)	(3,2)	(2,2)	(3,1)	(0,1)	(1,2)	(4,1)	(3,0)
f	4.8	4.8	4.8	4.8	5.4	5.4	5.4	5.4

Figure 3: Problem 2a, Cells might be expanded

(b) The table shows the open list currently.

Cell	g	rhs	state
(1,2)	2.4	∞	under
(2,2)	3.4	4.8	under
(2,3)	3.8	4.4	under
(4,3)	∞	5.8	over
(4,1)	∞	5.8	over
(3,0)	∞	5.8	over
(4,0)	∞	6.2	over

Figure 4: Problem 2b, Entire open list after UpdatedVertex

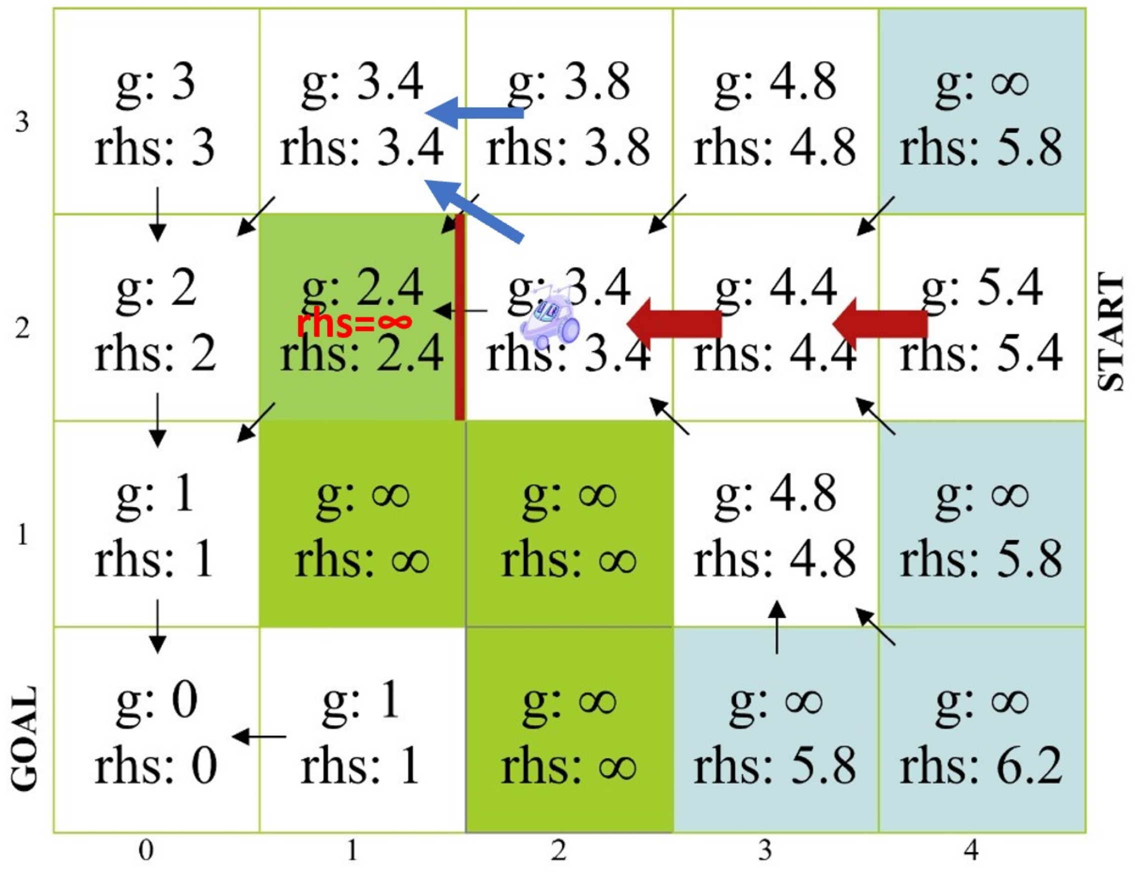


Figure 5: Problem 2b, After UpdatedVertex

(c) The next point which will be popped out is (2,2). Its g is ∞
Updated rhs:

cell	g	rhs	list
(2,3)	3.8	4.4	readd
(3,2)	4.4	5.2	add
(3,1)	4.8	5.4	add

Figure 6: Problem 2c, Recomputed rhs

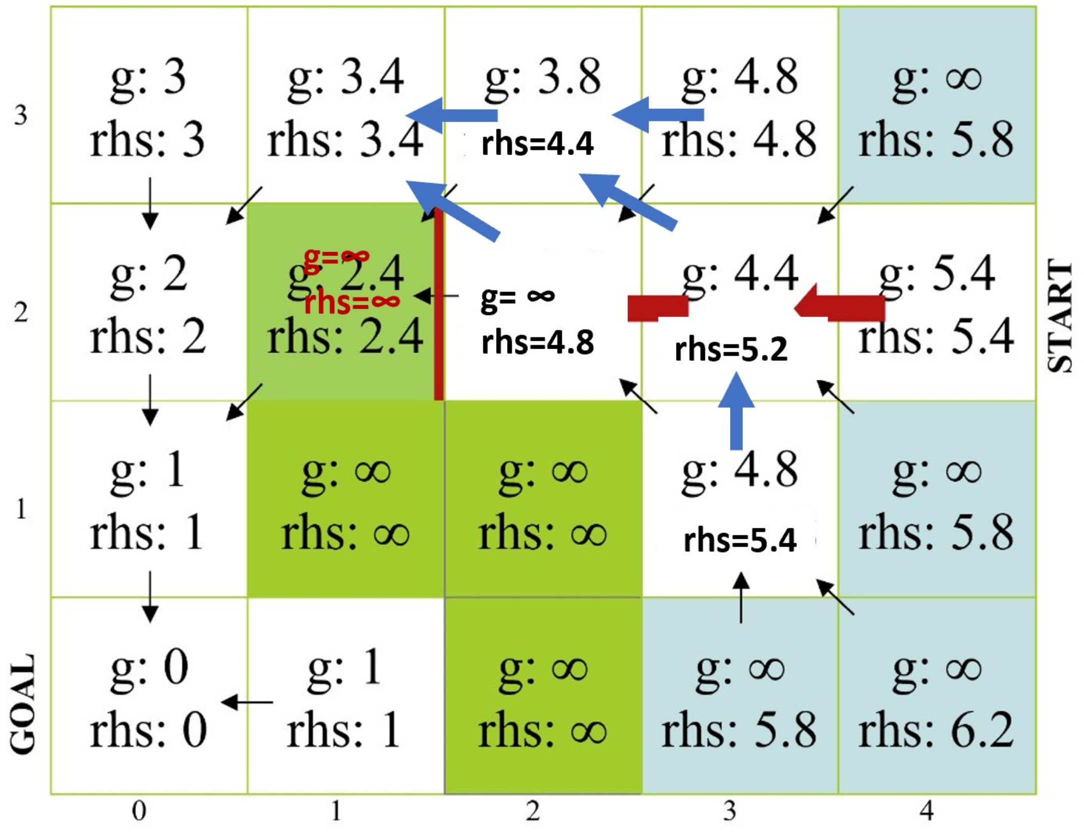


Figure 7: Problem 2c, Current State

cell	g	rhs	state	Real rhs
(2,3)	3.8	4.4	under	correct
(3,2)	4.4	5.2	under	5.8
(3,1)	4.8	5.4	under	6.2

Figure 8: Problem 2d, Underconsistent nodes in the open list

(d) There are 3 nodes in the list that are underconsistant as the table shows.

Problem 3: Decompositions and Roadmaps

(a) It will move along the slice and also move along the obstacle boundary.

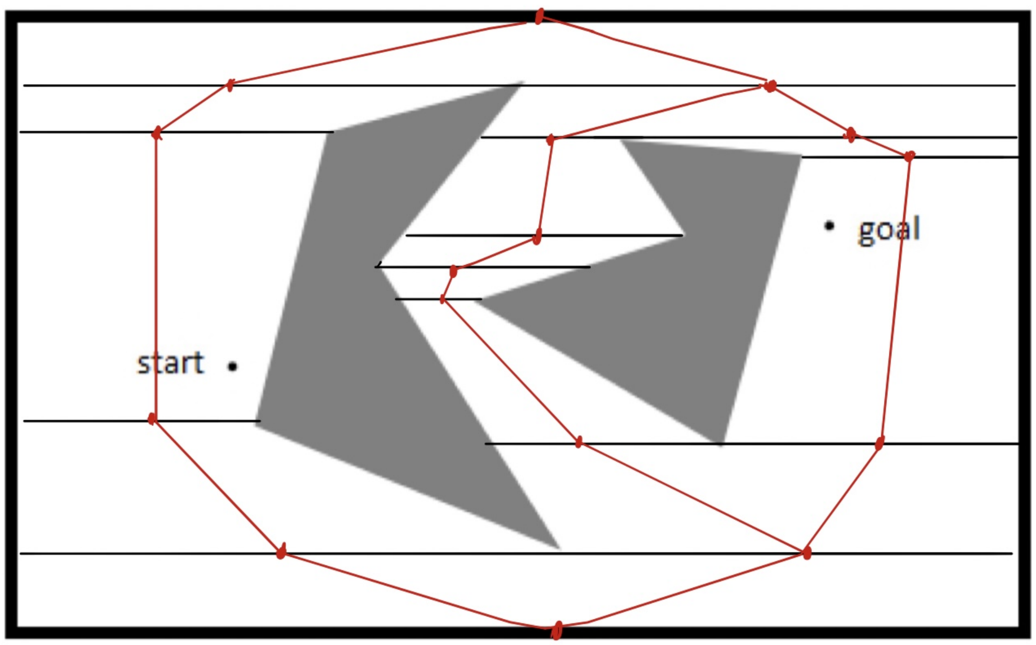


Figure 9: Problem 3a, Trapezoidal decomposition

(b) The roadmap is shown in the figure above

(c) The blue lines represent the supporting lines, and the red lines represent the separating lines.

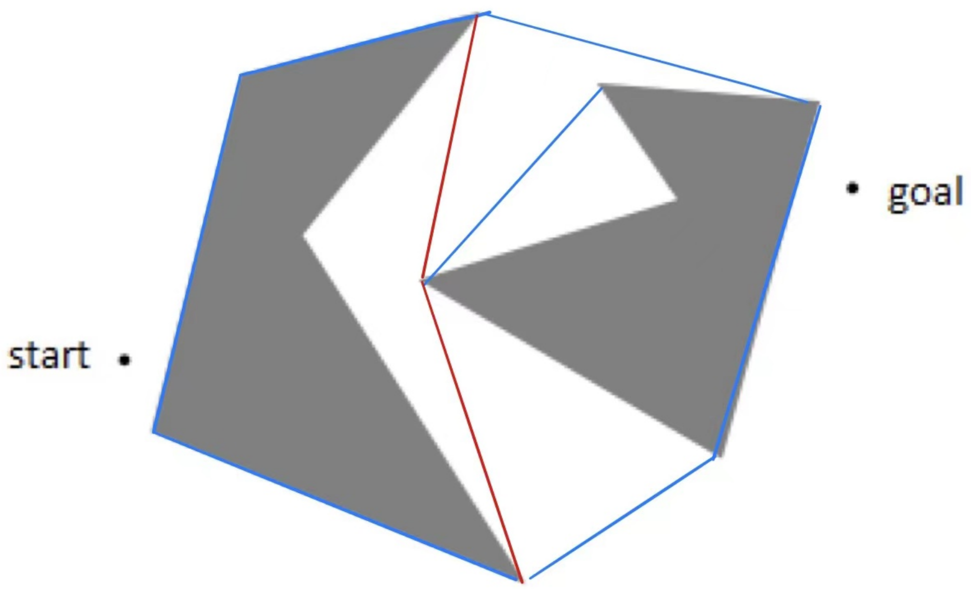


Figure 10: Problem 3c, Visibility graph

(d) 18 individual edge additions or deletions occur throughout its sweep will happen throughout its sweep. The maximum number of edges that any sweep line intersects at one time is 4(begin from the goal configuration).

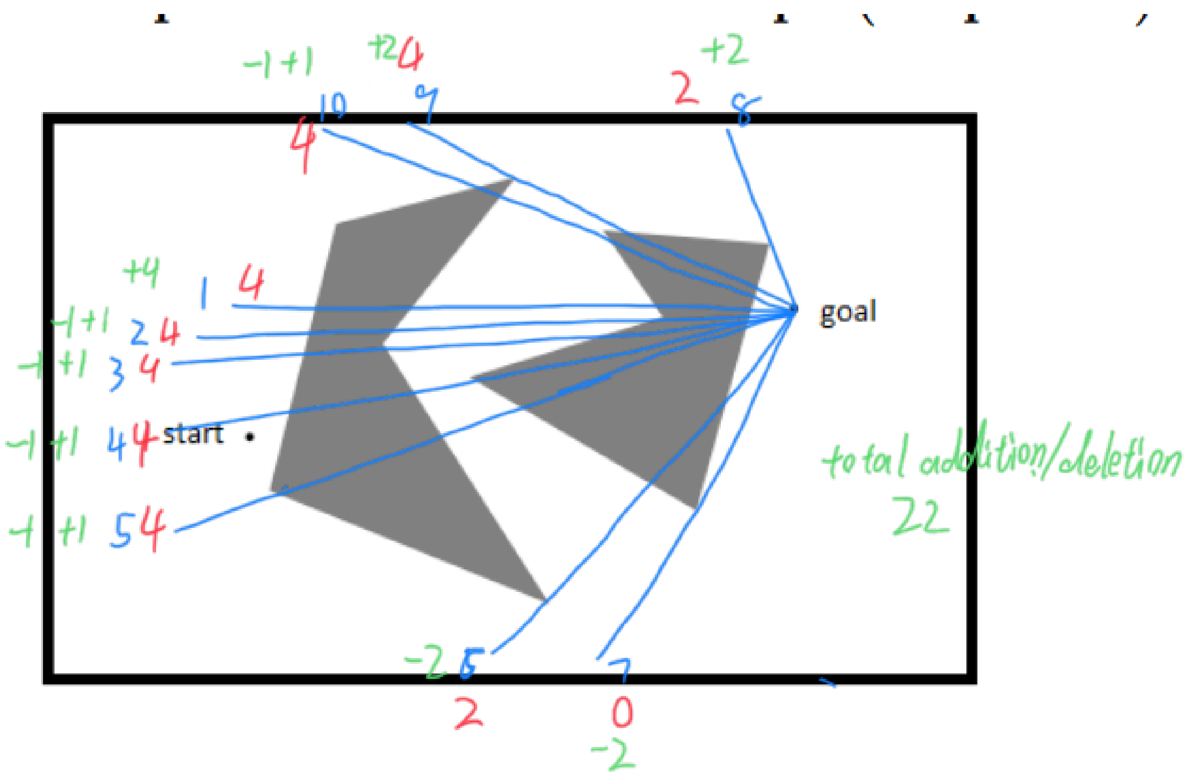


Figure 11: Problem 3d, Sweep line

Problem4: Roadmaps in Non-Euclidean Spaces

(a) In the figure below, because it is a C-space, the path can be extended to the boundary and then connect to the other side's boundary. The two nearest neighbors are indicated by green arrows coming out of each node. The road map is a closed loop actually.

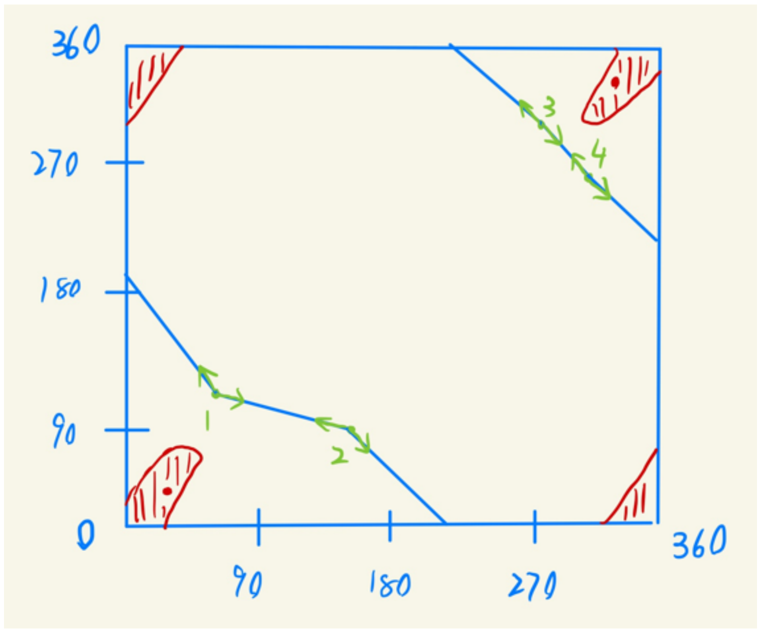


Figure 12: Problem 4a, C-space Road Map and Obstacle

The possible outline for the obstacle is the red wall showing in the figure above. We use the online configuration generator to verify the C-space obstacle.

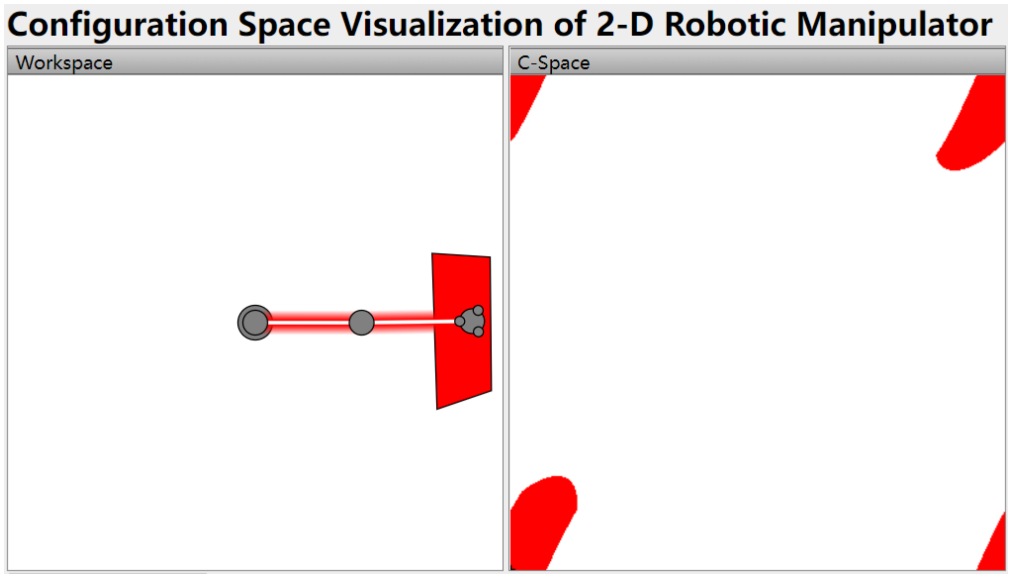

Configuration Space Visualization of 2-D Robotic Manipulator
Workspace
C-Space

Figure 13: Problem 4a, Online Configuration Space

(b) In the figure below, the nearest node: $q_{near} = (150, 90)$ with respect to the node: $q_{rand} = (180, 180)$. Then compute the q_{new}, we get $q_{new} = (153.16, 99.49)$, a distance of

δ away from q_{near} in direction of q_{rand}.

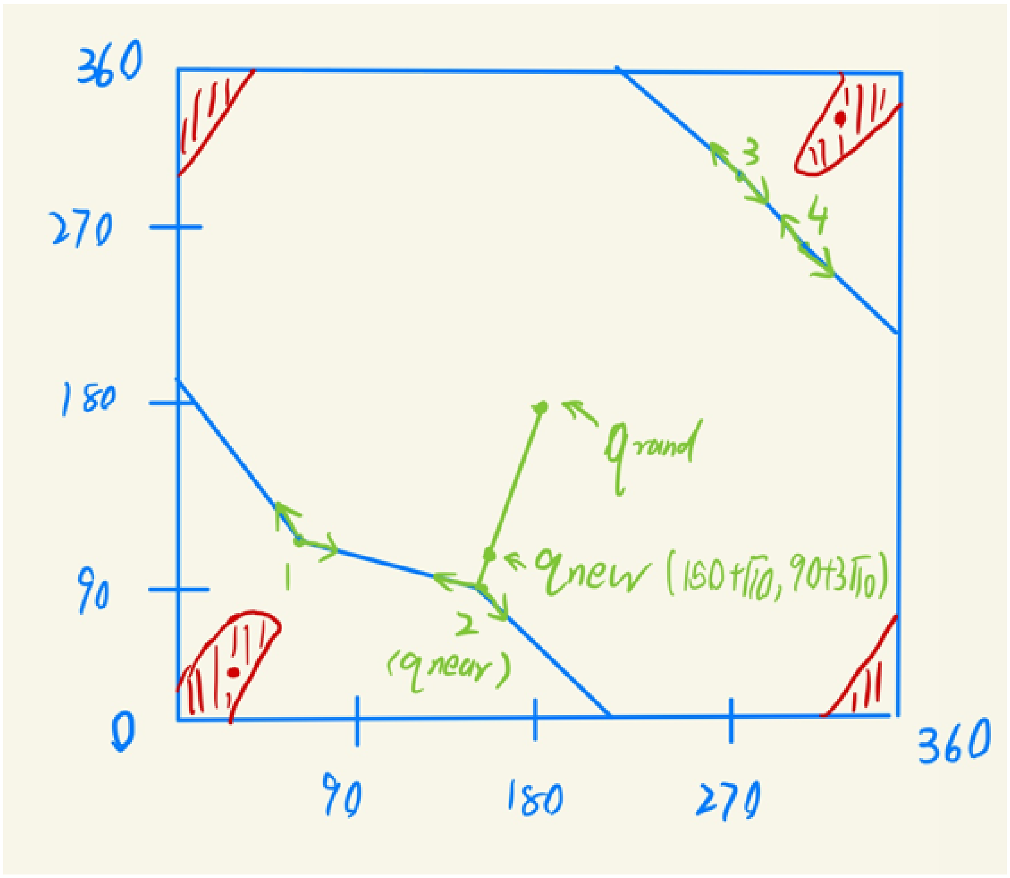
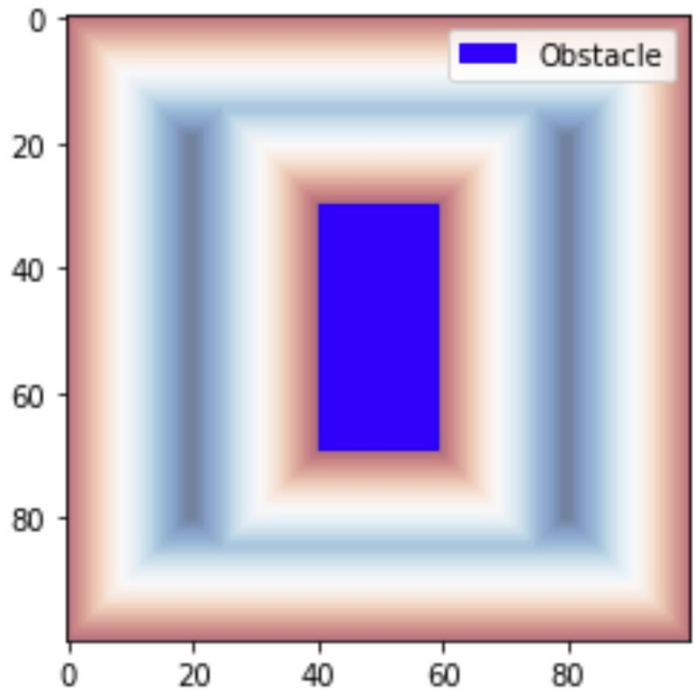


Figure 14: Problem 4b, Rapidly-Exploring Random Tree

Problem 5: GVD Construction

After get girds and add values to represent the environment boundary and obstacle, we store all the obstacle and environment boundary cells in the list. Then each cell in the list will be taken out to calculate the distances of its neighbors until all the cells are filled. If the cell are filled with two different number, we always take the smaller one.
We give a ID to each obstacle and the environment boundary so that it could indicate the original obstacle or environment boundary each grid cell that has been expanded. The cell which are filled with two same number will be add to GVD list, and its neighbors will be added as well for future expansion of GVD to make sure the continuity.
The following figures demonstrate the navigation procedures on each of the provided environments.



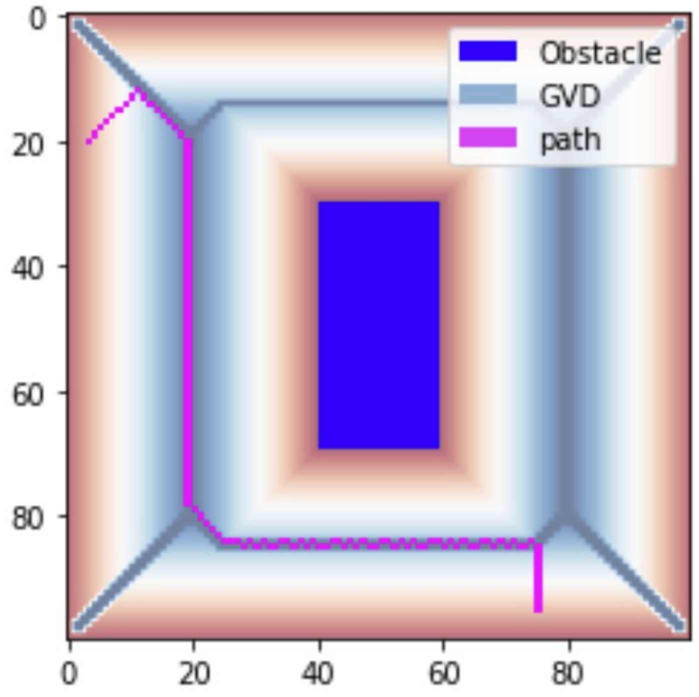


Figure 15: Problem 5, World 1 path

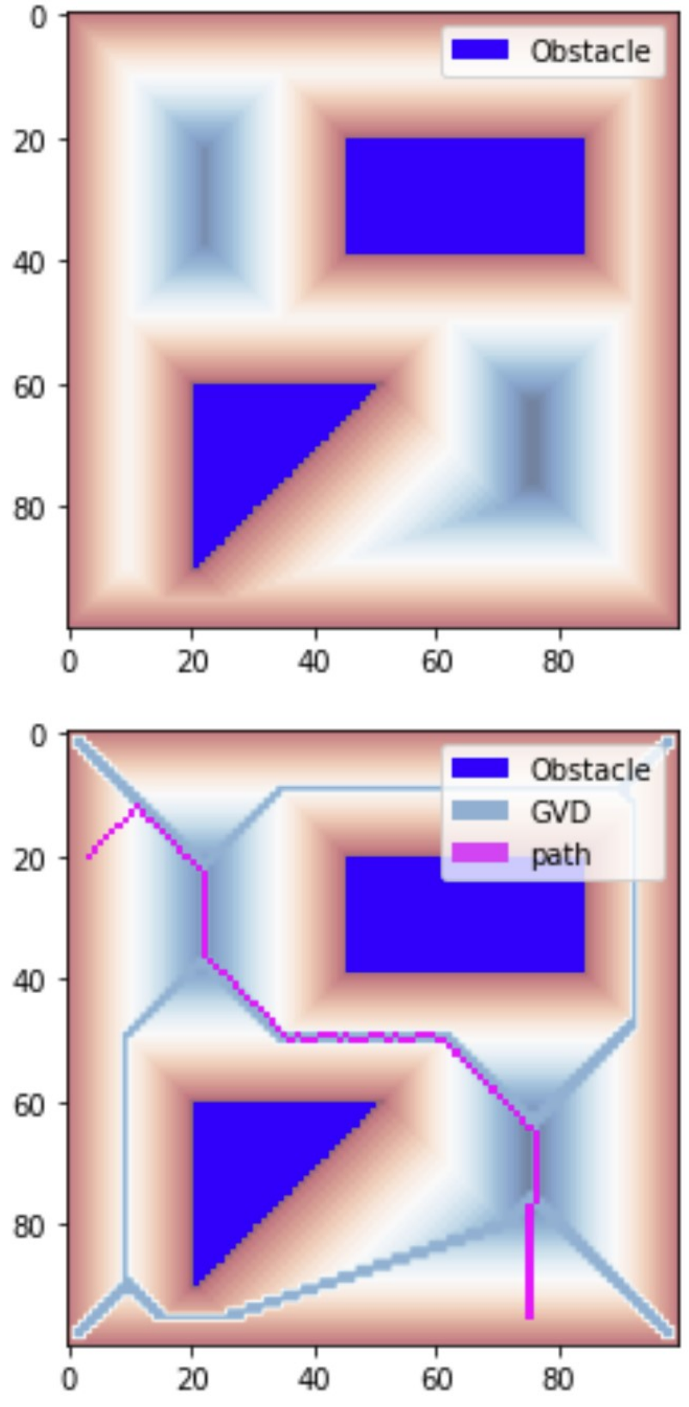
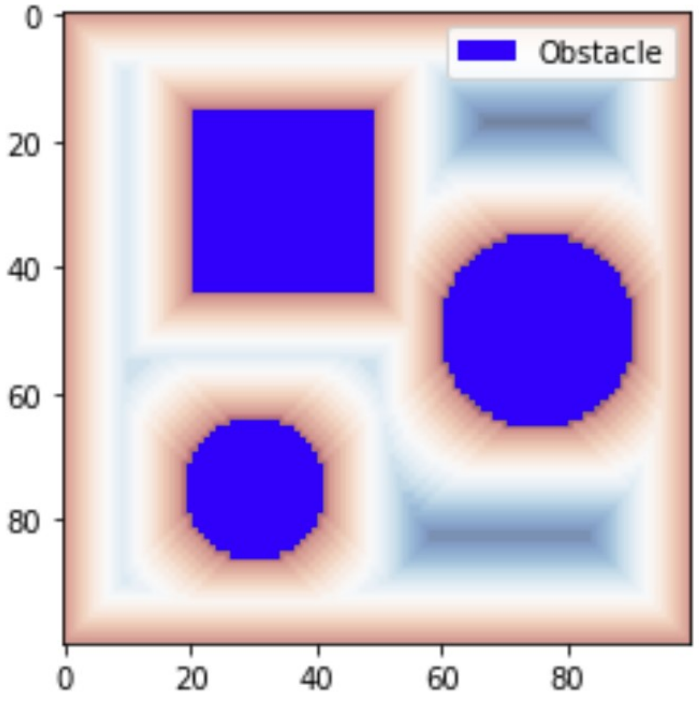


Figure 16: Problem 5, World 2 path



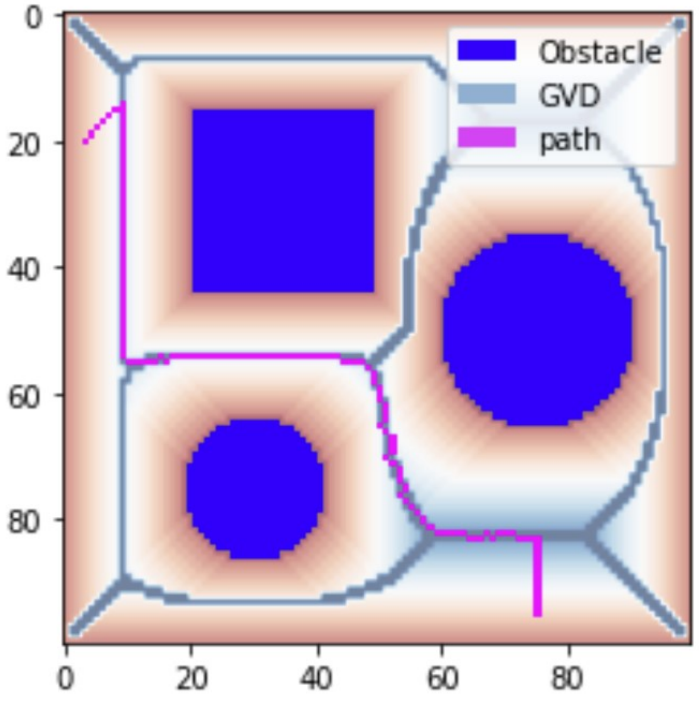
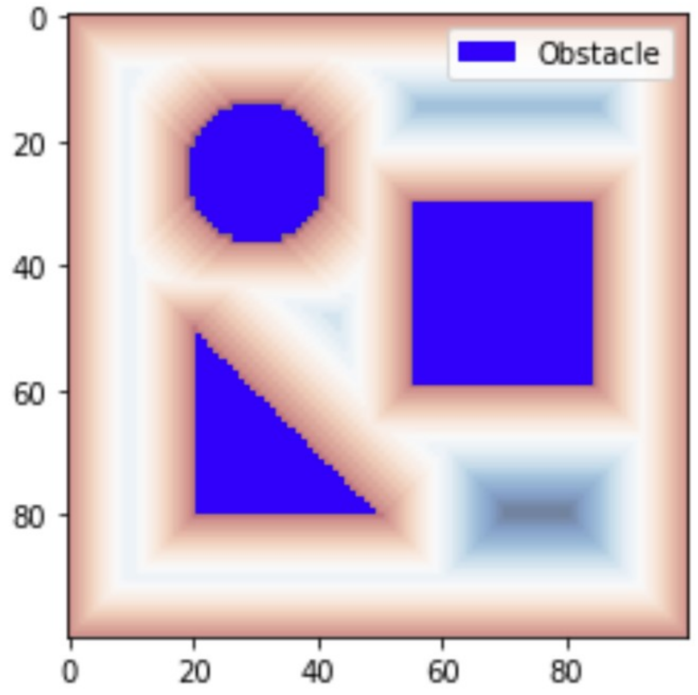


Figure 17: Problem 5, World 3 path



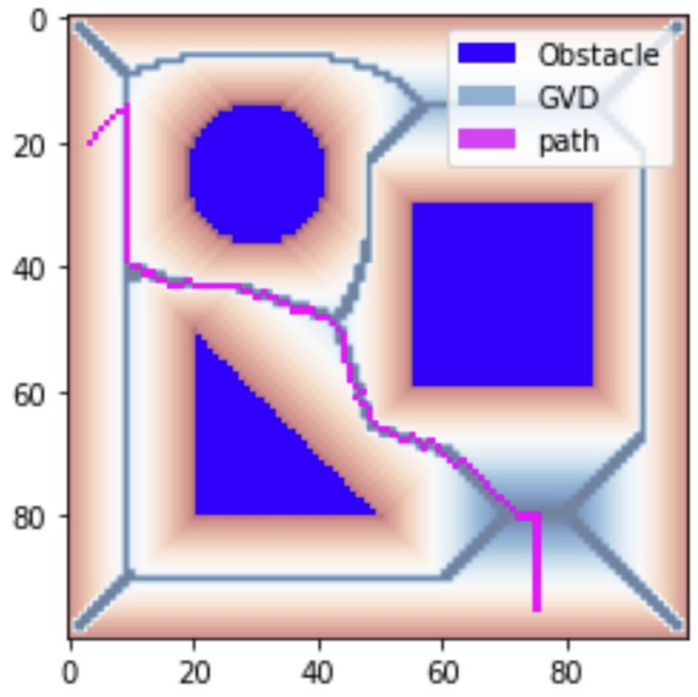


Figure 18: Problem 5, World 4 path

Appendix:

1. Question5 GVD

```
#BrushFrie
import collections
import numpy as np
from ipynb.fs.full.hw2_script import *


def BrushFire(grid):

    temp = np.copy(grid)
    q = collections.deque()
    m, n = len(temp), len(temp[0])

    ### [i,j,c] = [x, y, distance]
    for i in range(m):
        for j in range(n):
            if temp[i][j] < 0:
                continue

            if i == 0 or j == 0 or i == m-1 or j == n-1:
                temp[i][j] = 1


    for i in range(m):
        for j in range(n):
            if temp[i][j] < 0:
                q.append([i,j,0])

            if temp[i][j] == 1:
                q.append([i,j,1])


    direciton = [[1,0],[0,1],[-1,0],[0,-1],[1,1],[-1,1],[-1,-1],[1,-1]]
    while q:
        x, y, dist = q.popleft()

        for dx, dy in direciton:
            i = x + dx
            j = y + dy
            if i < 0 or i >= m or j < 0 or j >= n:
                continue
            if temp[i][j] < 0:
                continue

            if temp[i][j] != 0 and temp[i][j] < dist + 1:
                continue

            if temp[i][j] == 0 or temp[i][j] > dist + 1:
                temp[i][j] = dist + 1
                q.append([i,j,dist + 1])

    return temp



Map,x , y = generate_world_1()
plot_GVD(Map)
Map1BF = BrushFire(Map)
plot_GVD(Map1BF)


Map2,x , y = generate_world_2()
plot_GVD(Map2)
Map2BF = BrushFire(Map2)
plot_GVD(Map2BF)


Map3, x , y = generate_world_3()
plot_GVD(Map3)
Map3BF = BrushFire(Map3)
plot_GVD(Map3BF)


Map4, x , y = generate_world_4()
plot_GVD(Map4)
Map4BF = BrushFire(Map4)
plot_GVD(Map4BF)


def GVDMAP(grid):

    BF = BrushFire(grid)
    temp = np.copy(grid)
    q = collections.deque()
    m, n = len(temp), len(temp[0])
```

```
### [i,j,c] = [x, y, distance]
for i in range(m):
    for j in range(n):
        if temp[i][j] < 0:
            continue

        if i == 0 or j == 0 or i == m-1 or j == n-1:
            temp[i][j] = 1


for i in range(m):
    for j in range(n):
        if temp[i][j] < 0:
            q.append([i,j,0, temp[i][j]])
            continue

        if temp[i][j] == 1:
            if i == 0 and j == 0:
                q.append([i,j,1, -4])
            elif i == m-1 and j == n-1:
                q.append([i,j,1, -5])
            elif i == 0:
                q.append([i,j,1, -4])
            elif j == 0:
                q.append([i,j,1, -7])
            elif j == n-1:
                q.append([i,j,1, -5])
            elif i == m-1:
                q.append([i,j,1, -6])

### BFS
GVD_LIST = set()
direciton = [[1,0],[0,1],[-1,0],[0,-1],[1,1],[-1,1],[-1,-1],[1,-1]]
pointer = collections.defaultdict(set)

while q:
    x, y, dist, ID = q.popleft()

    for dx, dy in direciton:
        i = x + dx
        j = y + dy
        if i < 0 or i >= m or j < 0 or j >= n:
            continue
            if temp[i][j] < 0:
                continue

#             if temp[i][j] != 0 and temp[i][j] < dist + 1:
#                 continue

            if temp[i][j] == 0 or temp[i][j] > dist + 1:
                if (i,j) in GVD_LIST:
                    GVD_LIST.remove((i,j))

                temp[i][j] = dist + 1
                q.append([i,j,dist + 1, ID])
                pointer[(i,j)].add(ID)

            if temp[i][j] != 0 and (temp[i][j] == dist + 1 or abs(temp[i][j] -
                dist - 1) == 1) and len(pointer[(i,j)]) > 0 and ID not in
                pointer[(i,j)]:
                GVD_LIST.add((i,j))
                pointer[(i,j)].add(ID)


    return BF, GVD_LIST


import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Patch


def plotGVD(grid):
    MapBF, GVD_SET = GVDMAP(grid)
    plot_GVD(MapBF)
    GVD_LIST = np.array(list(GVD_SET))
    GVD_x, GVD_y = zip(*GVD_LIST)
    GVD_grid = np.copy(MapBF)
    GVD_grid[GVD_x,GVD_y] = 20
    fig, ax = plt.subplots()
    img1 = ax.imshow(GVD_grid, cmap="RdBu", alpha=0.6)
    obstacles = GVD_grid.copy()
    obstacles[obstacles < 0] = -2.0
    masked_data = np.ma.masked_where(obstacles > 0, obstacles)
    legend_elements = [Patch(facecolor='blue', label='Obstacle')]
    legend_elements.append(Patch(facecolor='#83b1d3', label='GVD'))
    img2 = ax.imshow(masked_data, cmap="bwr")
    ax.legend(handles=legend_elements)
    plt.show()
    return list(GVD_SET)

Map,x , y = generate_world_1()
PATH = plotGVD(Map)



Map2,x , y = generate_world_2()
PATH = plotGVD(Map2)


Map3, x , y = generate_world_3()
PATH = plotGVD(Map3)




def plotPATH(grid, start, end):
    MapBF, GVD_SET = GVDMAP(grid)
    plot_GVD(MapBF)
    GVD_LIST = np.array(list(GVD_SET))
    GVD_x, GVD_y = zip(*GVD_LIST)
    GVD_grid = np.copy(MapBF)
    GVD_grid[GVD_x,GVD_y] = 20

    A = tuple(start)
    B = tuple(end)

    AtoGVD = path_to_GVD(MapBF, list(GVD_SET), start)
    BtoGVD = path_to_GVD(MapBF, list(GVD_SET), end)
    s = AtoGVD[-1]
    BtoGVD.reverse()
    e = BtoGVD[0]
    Route = GVD_path(MapBF, list(GVD_SET), s, e)
    Route = AtoGVD + Route + BtoGVD
    fig, ax = plt.subplots()
    img1 = ax.imshow(GVD_grid, cmap="RdBu", alpha=0.6)
    obstacles = GVD_grid.copy()
    obstacles[obstacles < 0] = -2.0
    masked_data = np.ma.masked_where(obstacles > 0, obstacles)
    legend_elements = [Patch(facecolor='blue', label='Obstacle')]
    legend_elements.append(Patch(facecolor='#83b1d3', label='GVD'))
    img2 = ax.imshow(masked_data, cmap="bwr")

    path_x, path_y = zip(*Route)
    GVD_grid[path_x,path_y] = 40.0
    grid_path = GVD_grid.copy()
    grid_path = np.ma.masked_where(grid_path != 40.0, grid_path)
    img3 = ax.imshow(grid_path, cmap="cool_r", interpolation="nearest")
    legend_elements.append(Patch(facecolor='#e741f6', label='path'))
    ax.legend(handles=legend_elements)

    plt.show()



start = [20,3]
end = [95,75]


Map4, x , y = generate_world_4()
plotPATH(Map4, start, end)




Map3, x , y = generate_world_3()
plotPATH(Map3, start, end)



Map2, x , y = generate_world_2()
plotPATH(Map2, start, end)



Map, x , y = generate_world_1()
plotPATH(Map, start, end)
```