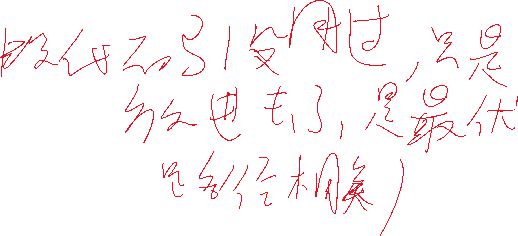
from \_\_future\_\_ import print\_function



import numpy as np

import copy

import Queue

from pylab import \*

import time

from PIL import Image

from scipy.misc import imread, imresize, imsave

# a = [[[[] for i in range(6)] for j in range(6)] for k in range(6)]

INF = 100000000.0

eps = 0.000001

dx = [1, 0, -1, 0, 1, 1, -1, -1]

dy = [0, 1, 0, -1, -1, 1, 1, -1]

edge = [1.0, 1.0, 1.0, 1.0, np.sqrt(2.0), np.sqrt(2.0), np.sqrt(2.0), np.sqrt(2.0)]

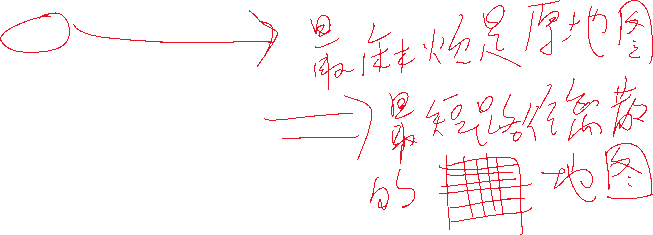
unit = 10

limit = 1 \* unit

def dcmp(x):

if np.fabs(x) < eps:

return 0



if x > 0:

return 1

return -1

class spfa():

def \_\_init\_\_(self, \_mp, \_n, \_m, \_s\_x, \_s\_y, \_e\_x, \_e\_y):

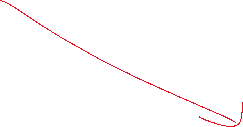
self.mp = \_mp

self.n = \_n

self.m = \_m

self.checkmp = np.zeros((self.n, self.m))

self.s\_x = int(\_s\_x \* unit)

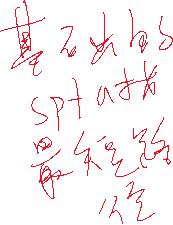


self.s\_y = int(\_s\_y \* unit)

self.e\_x = int(\_e\_x \* unit)

self.e\_y = int(\_e\_y \* unit)

self.dist = np.zeros((self.n, self.m))



self.inq = np.zeros((self.n, self.m), dtype=int32)

self.fax = np.zeros((self.n, self.m), dtype=int32)

self.fay = np.zeros((self.n, self.m), dtype=int32)

self.q = Queue.Queue(maxsize = self.n \* self.m)

self.ans = []

for i in range(self.n):

for j in range(self.m):

if self.mp[i][j] == 1:

for kx in range(int(-0.3\*unit), int(0.3\*unit)):

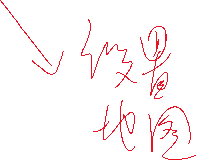
for ky in range(int(-0.3\*unit), int(0.3\*unit)):

if i+kx >= 0 and i+kx < self.n and j+ky >= 0 and j + ky < self.m:

self.checkmp[i + kx][j + ky] = 1

# x = np.random.random((self.n, self.m, 3))

# for i in range(self.n):



# for j in range(self.m):

# if self.mp[i][j] == 1:

# print('111111')

# x[i][j][0] = 1.0

# else:

# x[i][j][0] = 0.0

# plt.imshow(x)

# plt.show()

# self.showmp(self.mp)

# time.sleep(10000000)

for i in range(self.n):

for j in range(self.m):

self.dist[i][j] = INF

if i == self.s\_x and j == self.s\_y:

self.dist[i][j] = 0

self.inq[i][j] = 1

self.q.put([i, j])

def showmp(self, a):

tmp = np.zeros((self.n, self.m))

for i in range(self.n):

for j in range(self.m):

tmp[i][j] = a[self.n - 1 - i][j]

plt.imshow(tmp)

plt.show()

def check(self, x, y):

if x >= 0 and x < self.n and y >= 0 and y < self.m:

if self.mp[x][y] == 0 and self.checkmp[x][y] == 0:

return True

return False

else:

return False

def run(self):

print('start')

tim = 0

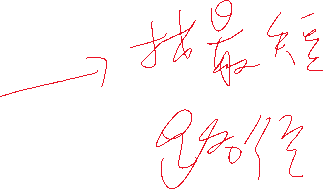
while(self.q.empty() == False):

tim += 1

# print('now', str(tim))

t = self.q.get()

x = t[0]



y = t[1]

# print("??", x, y)

self.inq[x][y] = 0

for k in range(8):

nx = x + dx[k]

ny = y + dy[k]

# print('????', x, y, nx, ny)

# time.sleep(0.5)

d = edge[k]

if self.check(nx, ny) == False:

continue

if dcmp(self.dist[x][y] + d - self.dist[nx][ny]) < 0:

self.dist[nx][ny] = self.dist[x][y] + d

# print('1111', type(x), x, y, nx, ny, self.fax[nx][ny], self.fay[nx][ny])

self.fax[nx][ny] = x

self.fay[nx][ny] = y

# print('fffuck', type(x), x, y, nx, ny, self.fax[nx][ny], self.fay[nx][ny])

# time.sleep(1)

# if y < 0:

# print('?????????')

if self.inq[nx][ny] == 0:

self.inq[nx][ny] = 1

self.q.put([nx, ny])

t\_x = self.e\_x

t\_y = self.e\_y

print("over")

print('start', self.dist[self.e\_x][self.e\_y], self.e\_x, self.e\_y, self.s\_x, self.s\_y, self.fax[t\_x][t\_y], self.fay[t\_x][t\_y])

print('????')

while((t\_x == self.s\_x and t\_y == self.s\_y) == False):

# print(t\_x, t\_y, self.dist[t\_x][t\_y])

# time.sleep(1)

self.ans.append([t\_x, t\_y])

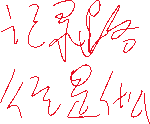
n\_x = self.fax[t\_x][t\_y]



n\_y = self.fay[t\_x][t\_y]

t\_x = n\_x

t\_y = n\_y



print('another over')

def getKey(self):

if self.checkmp[self.s\_x][self.s\_y] == 1 or self.checkmp[self.e\_x][self.e\_y] == 1:

return 1.0 \* ((self.s\_x - self.e\_x) \* (self.s\_x - self.e\_x) + (self.s\_y - self.e\_y) \* (self.s\_y - self.e\_y)) / unit, 1.0 \* self.e\_x / unit, 1.0 \* self.e\_y / unit

self.run()

# self.showPath()

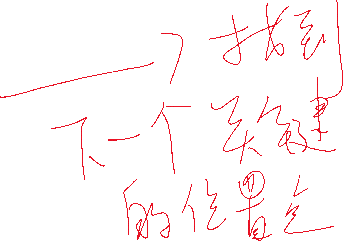
g\_x = 1.0 \* self.e\_x / unit

g\_y = 1.0 \* self.e\_y / unit

for i, var in enumerate(self.ans):

t\_x = var[0]

t\_y = var[1]



if self.dist[t\_x][t\_y] <= limit:

g\_x = 1.0 \* t\_x / unit

g\_y = 1.0 \* t\_y / unit

break

return 1.0 \* self.dist[self.e\_x][self.e\_y] / unit, g\_x, g\_y

def drawBigPoint(self, data, x, y, num):

for k in range(8):

nx = x + dx[k]

ny = y + dy[k]

if nx >= 0 and nx < self.n and ny >= 0 and ny < self.m:

data[nx][ny] = num

def showPath(self):

print('draw')

data = np.zeros((self.n, self.m))

for i in range(self.n):

for j in range(self.m):

data[i][j] = self.mp[i][j]

self.drawBigPoint(data, self.s\_x, self.s\_y, 3)

for i, var in enumerate(self.ans):

t\_x = var[0]

t\_y = var[1]

self.drawBigPoint(data, t\_x, t\_y, 2)

self.showmp(data)

# plt.imshow(data)

# plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

N = int(6.6 \* unit + 10)

M = int(9.9 \* unit + 10)

mp = np.zeros((int(6.6 \* unit + 10), int(9.9 \* unit + 10)))

for i in range(int(4.5\*unit),int( 6.60\*unit + 1)):

for j in range(int(0\*unit),int( 1.90\*unit + 1)):

mp[i][j] = 1

for i in range(int(6.00\*unit),int( 6.60\*unit + 1)):

for j in range(int(2.80\*unit),int( 5.00\*unit + 1)):

mp[i][j] = 1

for i in range(int(2.30\*unit),int( 3.70\*unit + 1)):

for j in range(int(3.15\*unit),int( 3.45\*unit + 1)):

mp[i][j] = 1

for i in range(int(3.70\*unit),int( 4.10\*unit + 1)):

for j in range(int(9.10\*unit),int( 9.90\*unit + 1)):

mp[i][j] = 1

for i in range(int(1.60\*unit),int( 3.40\*unit + 1)):

for j in range(int(8.50\*unit),int( 9.90\*unit + 1)):

mp[i][j] = 1

for i in range(int(0\*unit),int( 1.00\*unit + 1)):

for j in range(int(0\*unit),int( 3.30\*unit + 1)):

mp[i][j] = 1

for i in range(int(0\*unit),int( 1.00\*unit + 1)):

for j in range(int(4.40\*unit),int( 5.80\*unit + 1)):

mp[i][j] = 1

for i in range(int(0\*unit),int( 1.40\*unit + 1)):

for j in range(int(6.10\*unit),int( 7.90\*unit + 1)):

mp[i][j] = 1

myspfa = spfa(mp, N, M, 3, 1.5, 3, 7)

myspfa.run()

myspfa.showPath()