Practical work 3 documentation

def minimum\_cost\_walk(self, s, t, prev):

e = self.number\_of\_edges

v = self.number\_of\_vertices

# d[x][k]=the cost of the lowest cost walk from s to x and of length at most k, where s is the starting vertex

d = [[1 << 27 for x in range(e + 1)] for y in range(v)]

d[s][0] = 0

for k in range(1, e + 1):

for i in range(v):

# d[i][k]=min(d[i][k],min(d[j][k-1]+cost(j,i)), where j belongs to the set of inbound edges of i

d[i][k] = d[i][k - 1]

for j in self.dict\_in[i]:

if d[j][k - 1] + self.dict\_cost[(j,i)] < d[i][k]:

d[i][k] = d[j][k - 1] + self.dict\_cost[(j,i)]

# i is now the direct predecessor of j

prev[i] = j

# check for negative cost cycles

for i in range(1, v):

min\_cost = d[i][e]

for j in self.dict\_in[i]:

if d[j][e] + self.dict\_cost[(j, i)] < min\_cost:

raise Exception("Negative cycle !")

# the lowest cost found

return d[t][e]

This is the UI function . We call the minimum\_cost\_walk function from the graph and then we build the walk using the prev array :

def minimum\_cost\_walk(self):

s = int(input("Please insert the start vertex:"))

t = int(input("Please insert the end vertex:"))

prev = [-1] \* self.\_\_graph.number\_of\_vertices

try:

cost = self.\_\_graph.minimum\_cost\_walk(s, t, prev)

print("Cost of lowest cost path: " + str(cost))

res = "Lowest cost path: "

#Going backwards in order to build the minimum cost walk

stack = []

while prev[t] != -1:

stack.append(t)

t = prev[t]

stack.append(s)

#Displaying the walk on the screen

while len(stack) > 1:

res = res + str(stack.pop()) + "->"

res = res + str(stack.pop())

print(res)

except Exception as ex:

print(ex)