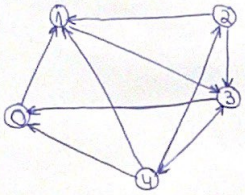
Manual execution

Graph



0	1	1
1	3	2
2	3	3
3	4	-2
4	0	10
4	3	4
2	1	-5
3	0	3
4	1	1
4	2	8

Graph

findMinimumCostWalk (0 , 4)

- MinimumCostWalk Using Dynamic Programming (0, 4)
- ReconstructWalk (4, costMatrix, beforeVertices)

- MinimumCost Walk Using Dynamic Programming

I costMatrix

	0	1	2	3	4
0	0	∞	∞	∞	∞
1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1

II

	0	1	2	3	4
0	0	∞	∞	∞	∞
1	0	1	∞	∞	∞
2	0	1	∞	3	∞
3	0	1	∞	3	1
4	0	1	9	3	1

— length

| vertices

before

keys	
0	-1
1	-1
2	-1
3	-1
4	-1

| vertices

keys	
0	-1
1	0
2	4
3	1
4	3

| vertices

Reconstruct Walk before : {-1, 0, 4, 1, 3}

I walk = [4]
current_vertex = 4

→ II ~~current_vert~~
walk = [4, 3]
current_vertex = 3

→ III walk = [4, 3, 1]
current_vertex = 1

→ IV walk = [4, 3, 1, 0]
current_vertex = 0

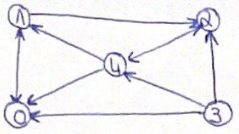
— stop

walk = [0, 1, 3, 4]

the cost = costMatrix [length(walk-1)][end_vertex] = costMatrix [3][4] = 4

Graph

Find Minimum Cost Walk (1 , 3)



- Minimum Cost Walk Using Dynamic Programming (1,3)
- Reconstruct Walk (3, cost matrix, before vertices)

0	1	4
1	2	5
1	0	1
2	4	-1
3	4	-4
3	2	-3
3	0	3
4	0	8
4	1	10
4	2	2

Graph

- Minimum Cost Walk Using Dynamic Programming

I cost Matrix

	0	1	2	3	4
0	0	∞	∞	∞	∞
1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1

II

	0	1	2	3	4
0	∞	0	∞	∞	∞
1	1	0	5	∞	∞
2	1	0	5	∞	4
3	1	0	5	∞	4
4	1	0	5	∞	4

— length

vertices

before

0	-1
1	-1
2	-1
3	-1
4	-1

vertices

0	1
1	-1
2	1
3	-1
4	2

vertices

if cost matrix[m.of vertices - 1][target - vertex] == ∞ ⇒ there is no walk between 4 and 3

cost matrix[4][3] = ∞