**Longest Path in a DAG Problem**

*Find a longest path between two nodes in an edge-weighted DAG.*

**Input:** An edge-weighted graph, a source node *source*, and a sink node *sink*.

**Output:** The length of the longest path from *source* to *sink*, followed by a longest path.

A *Directed Acyclic Graph (DAG)* is a graph that does not contain any directed cycles. The length of a path in an edge-weighted graph is given by the sum of its edge weights. Given nodes *source* and *sink*, our goal is to find a longest path from *source* to *sink*. We assume that all nodes of the graph have integer labels and ordering the nodes in ascending represents a *topological order*, i.e., all edges of the graph connect a node with smaller label to a node with a larger label.

**Input Format.** The first line of the input contains an integer representing *source* that has the smallest label. The second line of the input contains an integer representing *sink* that has the largest label. Each of the remaining lines represents an edge in the graph *G(V,E)* with node-set *V* and edge-set *E*, where each line is in the format *u*->*v*:*w* denoting an edge from node *u* to node *v* with weight *w*.

**Output Format.** The first line of the output should contain a number representing the length of the longest path from *source* to *sink*. The second line of the output should be a longest path in the format *source*->*a*->*b*->*c*->...->*sink*, where each of the items delimited by -> is a node in *G*. (If multiple longest paths exist, you may return any one.)

**Constraints.** |*V*| ≤ 50; |*E*| ≤ 100

**SAMPLE DATASET:**

Input:

0

4

0->1:7

0->2:4

2->3:2

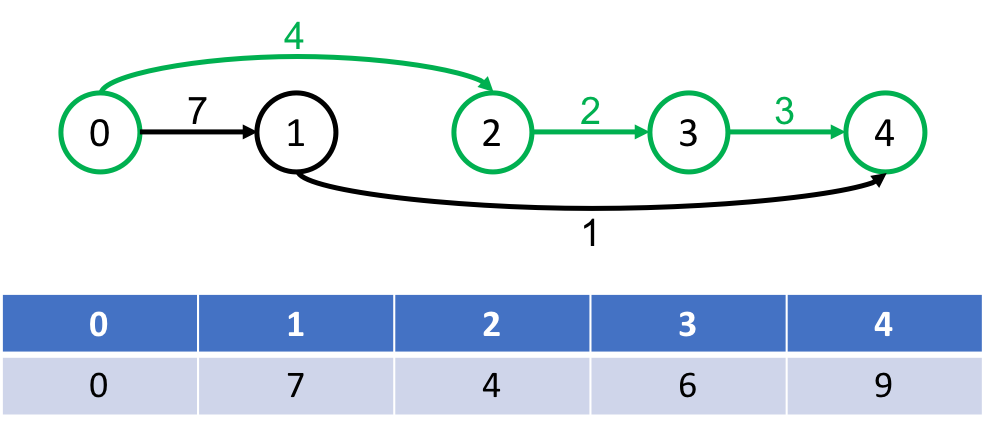
1->4:1

3->4:3

Output:

9

0->2->3->4

A longest path from node 0 (*source*) to node 4 (*sink*) is 0→2→3→4 with length 4 + 2 + 3 = 9. 

**TEST DATASET 1:**

Input:

0

3

0->1:1

0->3:10

1->2:1

2->3:1

Output:

10

0->3

This test makes sure that your code is actually finding the *longest* path in a DAG not the *shortest* path in a DAG. The shortest path in this DAG goes 0->1->2->3 with path length of 3 while the longest path goes 0->3 with a path length of 10. If your code outputs a path length of 3 it is likely that you are finding the shortest path instead of the longest path.

**TEST DATASET 2:**

Input:

0

3

0->1:2

0->2:1

1->3:3

2->3:3

Output:

5

0->1->3

This test makes sure that your code correctly parses the input. An input line of the form A->B:C represents an edge from node A to node B with a weight of C. A common mistake is parsing the input so that A->B:C represents an edge from node A to *node C with weight B*. If your code outputs a path length of 4 with a path of 0->2->3 it is likely that you are making the described parsing error.

**TEST DATASET 3:**

Input:

0

3

0->1:1

0->2:5

1->3:10

2->3:1

Output:

11

0->1->3

This test makes sure that your code isn’t an implementing of a greedy approach to this problem. A simple greedy approach that chooses paths at branching points based on edge weights at that point is not guaranteed return the correct output for this problem. If your code outputs a path weight of 6 and a path of 0->2->3 then it’s possible that you are using a greedy approach and fall into a non-optimal solution immediately by choosing the 0->2:5 edge.

**TEST DATASET 4:**

Input:

1

4

1->2:1

1->3:5

2->4:10

3->4:1

Output:

11

1->2->4

This test makes sure that your code does not rely on the source node being labeled 0. This dataset is the same as Test Dataset 3, except that each node label is incremented by one. If your output doesn’t match the correct output then your code likely assumes that the source node is labeled 0. Make sure that your implementation uses the source node label from the input instead of making any assumptions about the label of the source node.

**TEST DATASET 5:**

Input:

1

10

1->2:1

2->3:3

3->10:1

Output:

5

1->2->3->10

This test makes sure that your code can correctly parse inputs in which there are double digit node labels. All previous datasets only have nodes with single digit labels, so if your code relies on node labels only having single digits it will likely fail on this dataset. If your output doesn’t match the correct output make sure that your code can handle nodes that have double digit labels.

**TEST DATASET 6:**

Input:

0

4

0->4:7

Output:

7

0->4

This test makes sure that your code can correctly handle inputs that only contain one edge. If your output doesn’t match the correct output make sure that your implementation doesn’t contain an off-by-one error that prevents the use of the only edge in the graph.