Programming exercise 2, due Fri, Mar 6, 11.59pm (on Gradescope)

Dijkstra's Algorithm for Shortest Path in a Graph

In this exercise you will create an implementation (in Python 3 or Java or C++) of Dijkstra's Algorithm for finding the shortest paths in a directed graph with weighted edges. This problem and algorithm are described in Section 4.4 of *Algorithm Design.* Your code must run in $O(m \log n)$ time, where m is the number of edges and n is the number of vertices. We've set rough upper bounds on runtime through the autograder; if your code exceeds these, you will see the limit and your code's running time.

Collaboration policy

On this assignment, no collaboration is permitted. The solution must be your own, you must write and test it alone. (Consulting a classmate on some programming detail is OK.) Using any program or program part from some other source, like some Internet site is emphatically not permitted.

Submission Format

On Gradescope submit a file (either `Dijkstra.py` or `Dijkstra.java` or `Dijkstra.cpp`). If your program needs some supporting files (say in Java) you can upload those, too, but one of your uploaded files must have the exact name shown here.

Your program reads a text file from standard input, and writes to the standard output. If, say, input file `input2` and output file `output1` are in the same directory as your program then for example your Python program would be called *from the command line* as

```
> python3 Dijkstra.py < input2 > output1
```

(If file `output1` already exists then one may have to write `>|` or `>!` to overwrite it, depending on the command line interpreter used.)

If it is in Java then the autograder would call

```
> javac Dijkstra.java
> java Main < input2 > output1
```

This requires that your program file defines a class `Main`, in which you defined the `main(String[] args)` method.

If it is in C++ then

```
> g++ -std=c++17 Dijkstra.cpp
> a.out < input2 > output1
```

See the instructions on Homework 1 on how to read/write standard input/output (the signs `<,>`).

Since the focus of this exercise is on the algorithm, you can use a priority queue available from the libraries of Python, Java or C++ implemented as a heap, if you don't want to use your own heap.

Input

The vertices are indexed $0, 1, \ldots, n-1$. The weights will all be integers. The file `input` contains $m+1$ lines.

- First line: integers n, m, s separated by spaces: the number of vertices, number of edges, and the index of the source vertex.
- Every line after that will specify an edge in the following format:
  index of starting vertex, space, index of end vertex, space, weight of the edge.

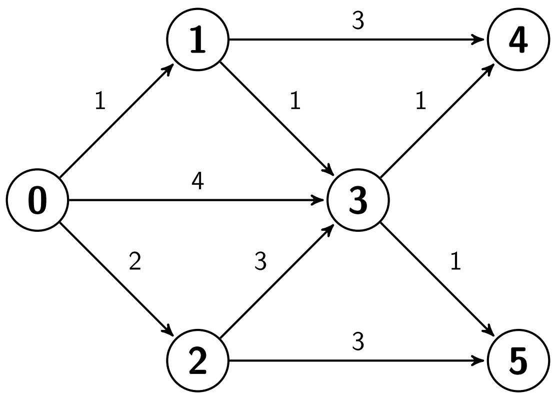
We have uploaded files corresponding to the visible test cases on Gradescope. They may help you test and debug locally. Note: passing these test cases does not guarantee your code is correct!

Output

The output must contain one line for each vertex different from s and reachable from s (in increasing order of the vertex indices), with the following information. If for example node 4 has distance 9 and its parent in the tree of shortest paths is 13 then the line is `4 9 13`.

Example

Consider the following directed graph:



For calculating the paths from vertex 2, here is a possible input file (note that the edges may come in any order).

```
6 9 2
0 1 1
0 3 4
0 2 2
2 3 3
2 5 3
3 4 1
1 4 3
1 3 1
3 5 2
```

Output:

```
3 3 2
4 4 3
5 3 2
```