CS566 – Summer 2017
Extra Credit Programming Assignment

Assigned: 7/12
Due: 7/26

This assignment is worth 4% of the course grade.

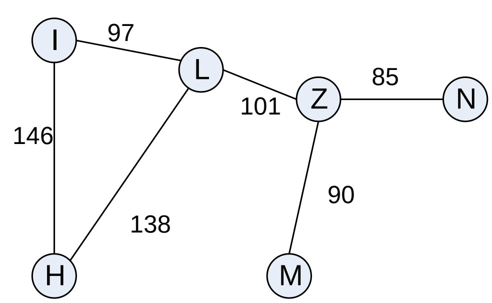
The goal of this assignment is to give students an opportunity to design and implement two heuristic algorithms to find a shortest path in a graph. You need to read the problem description carefully, design algorithms, select appropriate data structures, and write a program to implement the algorithms.

<u>Problem Description</u>

Your program must read a graph information from an input file and store the graph in an appropriate data structure. You can use any data structure to store the input graph.

There are two input files. The first input file, named *graph_input.txt*, contains a textual representation of a graph. The following example illustrates the input file format.

Consider the following graph. Each node has a name (an uppercase character in a circle) and each edge is associated with a number. The number associated with an edge, which is called *weight* of the edge, represents the distance between the two vertices that are connected by the edge. We assume that all distances are positive integers.

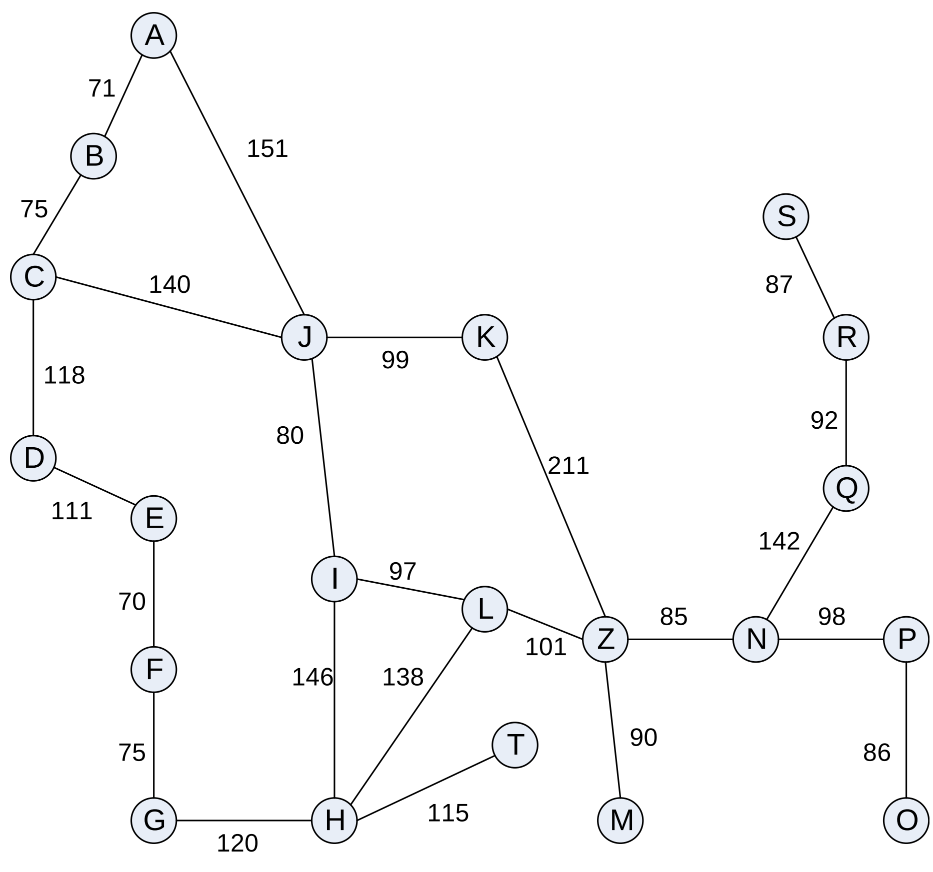



The graph is represented in the input file as follows:

```
        H     I     L     M     N     Z
H       0   146   138     0     0     0
I     146     0    97     0     0     0
L     138    97     0     0     0   101
M       0     0     0     0     0    90
N       0     0     0     0     0    85
Z       0     0   101    90    85     0
```

This representation of a graph is called *adjacency matrix*. In the input file, the vertex names (uppercase letters) and integers are separated by one or more spaces. Le *n* be the number of vertices in a graph. There are *n* + 1 columns and *n* + 1 rows in the input file. The top row and the first column show the names of vertices. The main part of the input file is an *n* x *n* two-dimensional array (or matrix) of integers. The meaning of a non-zero integer at the intersection of row *i* and column *j* is there is an edge connecting the vertex corresponding to row *i* and the vertex corresponding to column *j*, and the integer is the distance between the two vertices. For example, the “138” at the intersection of the first row and the third column in the matrix means there is an edge between the vertex *H* and the vertex *L* and the distance is 138. The entry 0 means that there is no edge connecting two corresponding vertices. For example, the “0” in the sixth row and the first column in the matrix means that there is no edge between the two vertices *Z* and *H*.

Suppose that a network of cities is represented by a weighted, undirected graph as shown below. Each node in the graph represents a city, the edge between two nodes represents a road connecting the two cities, and the weight of an edge represents the distance between the two cities on the connecting road.




The second input file, named *direct_distance.txt*, has the *direct distance* from each node to node *Z*. This direct distance from a node *n* to node *Z* is the distance measured along an *imaginary straight line* (or geographically straight line) from node *n* to node *Z* and is not necessarily the same as the sum of the weights of the edges connecting *n* to *Z*.

The second input file, corresponding to the above input graph, contains:

A 380
B 374
C 366
D 329
E 244
F 241
G 242
H 160
I 193
J 253
K 176
L 100
M 77
N 80
O 161
P 151
Q 199
R 226
S 234
T 92
Z 0

You are required to implement two heuristic algorithms that are described below. Both algorithms try to find a shortest path from a given input node to node Z using heuristic approaches. In a shortest path, a node may appear at most once (i.e., a node cannot appear twice or more in a path).

Both algorithms start with the given input node and iteratively determine the next node in a shortest path. In determining which node to choose as the next node, they use different heuristics.

Let $w(n, v)$ be the weight of the edge between node $n$ and node $v$. Let $dd(v)$ be the *direct distance* from $v$ to the destination node Z.

When choosing the next node from a current node $n$:

> Algorithm 1: Among all nodes $v$ that are adjacent to the node $n$, choose the one with the smallest $dd(v)$.
>
> Algorithm 2: Among all nodes $v$ that are adjacent to the node $n$, choose the one for which $w(n, v) + dd(v)$ is the smallest.

**Example 1**: Start node is J

Algorithm 1:

Current node = J
Adjacent nodes: A, C, I, K
A: *dd*(A) = 380
B: *dd*(C) = 366,
I: *dd*(I) = 193
K: *dd*(K) = 176.
K is selected
Shortest path: J → K

Current node = K
Adjacency node: J, Z
J is already in the path
Z is the destination node. Stop.
Shortest path = J → K → Z
Shortest path length = 99 + 211 = 310

Algorithm 2:

Current node = J
Adjacent nodes: A, C, I, K
A: *w*(J, A) + *dd*(A) = 151 + 380 = 531
C: *w*(J, C) + *dd*(C) = 140 + 366 = 506
I: *w*(J, I) + *dd*(I) = 80 + 193 = 273
K: *w*(J, K) + *dd*(K) = 99 + 176 = 275
I is selected.
Shortest path: J → I

Current node = I
Adjacent nodes: J, H, L
Node J is already in the path
H: *w*(I, H) + *dd*(H) = 146 + 160 = 306
L: *w*(I, L) + *dd*(L) = 97 + 100 = 197
L is selected
Shortest path: J → I → L

Current node = L
Adjacent node: H, I, Z
H: *w*(L, H) + *dd*(H) = 138 + 160 = 298
I is already in the path
Z: *w*(L, Z) + *dd*(Z) = 101 + 0 = 101

Z is selected
Z is the destination. Stop.

Shortest path: J → I → L → Z
Shortest path length = 80 + 97 + 101 = 278

**Example 2**: Start node is G

Algorithm 1:

Current node = G
Adjacent nodes: F, H
F: *dd*(F) = 241
H: *dd*(H) = 160
H is selected
Shortest path: G → H

Current node = H
Adjacent nodes: G, I, L, T
G is already in the path
I: *dd*(I) = 193
L: *dd*(L) = 100
T: *dd*(T) = 92
T is selected
Shortest path: G → H → T

Current node = T
Adjacent node: H
H is already in the path.
**Dead end**.
**Backtrack to H: G → H → T → H**
Node L is selected
Shortest path = G → H → L

Current node = L
Adjacent nodes: H, I, Z
H is already in the path
I: *dd*(I) = 193
Z: *dd*(Z) = 0
Z is selected
Z is the destination. Stop.
Shortest path: G → H → L → Z
Shortest path length = 120 + 138 + 101 = 359

Algorithm 2

Current node = G
Adjacent nodes: F, H
F: $w$(G, F) + $dd$(F) = 75 + 241 = 316
H: $w$(G, H) + $dd$(H) = 120 + 160 = 280
H is selected
Shortest path: G → H

Current node = H
Adjacent nodes: G, I, L, T
G is already in the path
I: $w$(H, I) + $dd$(I) = 146 + 193 = 339
L: $w$(H, L) + $dd$(L) = 138 + 100 = 238
T: $w$(H, T) + $dd$(T) = 115 + 92 = 207
T is selected
Shortest path: G → H → T

Current node = T
Adjacent node: H
H is already in the path
**Dead end**.
**Backtrack to H: G → H → T → H**
L is selected.
Shortest path = G → H → L

Current node = L
Adjacent nodes: H, I, Z
H is already in the path
I: $w$(L, I) + $dd$(I) = 97 + 193 = 290
Z: $w$(L, Z) + $dd$(Z) = 101 + 0 = 101
Z is selected.
Z is the destination. Stop.
Shortest path: Shortest path = G → H → L → Z
Shortest path length = 120 + 138 + 101 = 359

Note that the above examples are just illustration. You need to design your own algorithms and develop pseudoceds based on the above description and the examples.

We assume the followings:

- The number of nodes in an input graph is 26 or smaller.
- Each node is represented by a single uppercase letter.
- The destination node is Z.
- The destination node Z is reachable from all other nodes.
- All distances (edge weights) are positive integers.

Specific Requirements

1. Implement both algorithms in one program. Your program may include multiple files.
2. Your program must prompt the user to enter the start node. Your program must check the validity of the user-entered start node, and if the input is invalid your program must prompt the user to enter an input again.
3. Then, your program must find a shortest path from the input node to node Z using each algorithm and print the output on the screen.
4. Your output must include the following three components for each algorithm: (1) The sequence of all nodes that were initially included in a shortest path. This sequence must include the backtrackings (see the example below), (2) The final shortest path found by the algorithm, (3) Shortest path length.

   Output Example:

   (A) User enters node J as the start node

   Algorithm 1:

   Sequence of all nodes: J -> K -> Z
   Shortest path: J -> K -> Z
   Shortest path length: 310

   Algorithm 2:

   Sequence of all nodes: J -> I -> L -> Z
   Path: J -> I -> L -> Z
   Length: 278

   (B) User enters node G as the start node

   Algorithm 1:

   Sequence of all nodes: G -> H -> T -> H -> L -> Z
   Shortest path: G -> H -> L -> Z
   Shortest path length: 359

   Algorithm 2:

   Sequence of all nodes: G -> H -> T -> H -> L -> Z
   Shortest path: G -> H -> L -> Z
   Shortest path length: 359

Note:

- These two algorithm do not always find a correct shortest path.
- You can use the given graph to test your program. When I grade your program, I may use a different graph. So, you must make sure that your program will work on other graphs.

<u>Deliverable</u>

1. Documentation:
   - You must include a cover page with the course number, assignment name, and your name.
   - You must describe, in detail, how I should run your program.
   - You must indicate which programming language and which version of the programming language you used.
   - You must hardcode input file names in your program. In your documentation, you must clearly state in which part of your program those file names are hardcoded. This is necessary because I should be able to change the input file names when testing your program.
   - You must include the pseudocodes of both algorithms that you used to implement the algorithms. When you write pseudocodes, follow the style of the pseudocodes used in our textbook.
   - You must describe in detail major data structures you used in your program.
   - Name this file *LastName_FirstName_ec.EXT*. Here, *EXT* is an appropriate file extension, such as *pdf* of *docx*.

2. Inline documentation
   - For each method or function in your program, write the description of the input to and output from the method/function as comments.
   - Include sufficient comments within the source code.

3. Source code:
   You must submit all source code files.

   If you used C/C++/C#, you also need to submit an executable file. I will test your program by running the executable file from the command line. So, before submission, you may want to run your executable file from the command line to see that it runs OK.

4. Submission
   Combine the documentation file and source code file(s) into a single archive file. Name the archive file *LastName_FirstName_ec.EXT*. Here, *EXT* is an appropriate file extension, such as *zip* of *rar*.

<u>Grading</u>

The project will be graded on the scale of 100 points.

The documentation and inline comments are worth 30 points.

- If your documentation does not include all required components (for example, if it does not include a pseudocode), points will be deducted.
- If your documentation does not include sufficient inline comments, including the description of I/O of each method/function, points will be deducted.
- If your pseudocodes are not well organized or ambiguous, you will lose points.
- If your description of data structures is not clear or is not detailed enough, you will lose points.

The correctness of your program is worth 70 points. Your program will be tested for two cases: (1) Those that do not involve backtracking (refer to Example 1 above) and (2) Those that involve backtracking (refer to Example 2 above). Each case is worth 35 points.

Your program will be tested with the provided input and it may also be tested on other input graphs. If the outputs are incorrect, points will be deducted.

If I have difficulty running your program, I may ask you to bring your laptop to the class or to my office and demonstrate your program.