Homework 6: Minimum Spanning Trees

Assigned Friday, April 13, 2012. Due Friday, April 27, 2012. Electronic submissions should be sent to fatalay@sju.edu and nph87903@gmail.com by midnight on the due date. No hardcopy required. You are allowed to do this homework individually, or in groups of two or three. Each group will return a single homework and will get the same grade. But a group is NOT allowed to share anything written with another group.

Overview:

In this project you will implement a number of different data structures (graphs, multiway trees, and heaps) to compute and store a minimum spanning tree (MST) of a connected undirected graph with weighted edges.

Graph Input:

Assume the input is in a file called "input.txt" and output to the standard output. The input begins with a description of the graph. This description starts with the number of vertices, followed by a list of vertex identifiers, one per line. Each vertex has an associated string *identifier*. You may assume that each string identifier consists of at most 20 characters. For example, here is the input for three vertices.

```
3                          (number of vertices)
V128.8.10.14               (vertex identifier)
V128.8.128.423
V128.8.10.268
```

To specify the edges, the number of edges is given followed by a list of edges, each consisting of the identifiers of the endpoints and the edge weight (float). You may assume that all weights are positive.

```
2                                        (number of edges)
V128.8.10.14   V128.8.10.268   43.56     (endpoints and edge weight)
V128.8.10.268  V128.8.128.423  170.36
```

You may assume that the first part of the input follows these specification (that is, you do not have to check for input format errors). You may assume that the vertex identifiers are unique, and that the edges are distinct (no multiple edges) and the endpoints are valid vertices.

MST by Prim's Algorithm:

You may assume that the input graph is connected. After inputting the graph, compute its MST using Prim's algorithm. For consistency, you should use the first vertex input (V128.8.10.14 in this case) as the starting vertex for the algorithm. Output the sequence of edges that have been added to the tree and their associated weights.

Directives:

The rest of the input consists of a sequence of directives, each indicating an operation to be performed on the graph.

Print the MST: Print the contents of the current MST(s), given a particular vertex v as the root. Details on the output format are provided later in the document, but basically involve printing the MST according to a preorder traversal of the tree.

Path: Given two vertex identifiers, compute and print the path between them in the MST.

Here is the format of the directives.

```
print-mst       u       Print the MST using u as the root
path            u v     Print the path from u to v in the MST
quit                    This is the last directive.
```

Implementation Requirements:

You are required to implement at least three separate data structures for the assignment: (1) an undirected (weighted) graph, (2) a multiway tree, and (3) a heap priority queue (It is okay to use the posted HeapPriorityQueue class). The undirected graph must be implemented using an adjacency list. The MST is to be stored in the multiway tree, using the firstChild and nextSibling representation. (If you prefer another representation, that will befine too, but in your documentation make sure to explain it well.) It will also be necessary to have a parent pointer for each node for path finding operations. The binary heap is used in Prim's algorithm.

You are not required to provide an efficient method for mapping a vertex identifier to a vertex object. You may use the built-in Hashtable class in java.util, but you can just do a linear search if you like.

You are allowed to use simple, linear data structures from the Java libraries (e.g. strings, linked-lists, arraylists, stacks, queues). You are not allowed to use anything more sophisticated though (no trees, dictionaries, priority queues, etc.). The only exception is you may use any data structure you like for mapping vertex identifiers to actual vertex objects, as mentioned in the previous paragraph.

The binary heap must support an operation `decreaseKey()`, which decreases the key value of an entry in the heap, and rearranges the heap contents accordingly. Note that the hard part of this operation is locating where the item is stored in the heap. There is no fast method for searching a heap for a given key other than checking every value in the heap (think about this). You should provide a cross-reference mechanism by which each vertex can find its corresponding node in the heap in $O(1)$ time.

Major Data Structures:

The main data structures consist of the graph, which is represented using an adjacency list representation and a multiway tree. The graph data structure will likely consist of at least two types of nodes: *vertices* (shaded in the figure below) and *edges*. Each vertex node contains information such as the vertex identifier and a pointer to the adjacency list. Each edge node contains information about each edge, including the neighboring vertex, the edge weight, and the cross link to the twin

edge (not shown in the figure). In the figure the edge neighbor is given as an identifier, but this would actually be a pointer (or index) to the vertex node. In addition there will be a link for accessing the next element in the adjacency list.

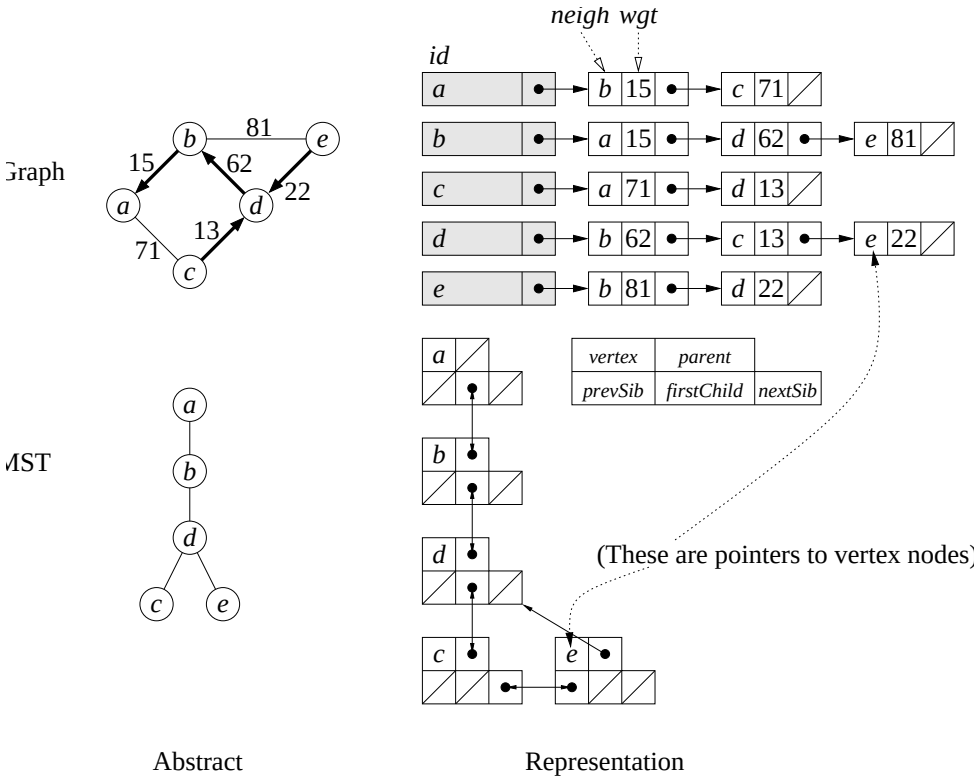

Figure 1: Overview of major data structures.

When Prim's algorithm runs, the link from each vertex u to its predecessor corresponds to a link in the MST. Each MST is a multiway tree. Each node of this data structure contains a reference to the associated vertex in the graph. (In the figure this is given as the vertex identifier, but actually this would be a pointer to the vertex node.) In addition the node might contain pointers to the first child, next sibling, previous sibling, and parent in the multiway tree.

The multiway tree will need to provide a number of operations to support reorganization of the tree to make an arbitrary node the new root (which is required for the print-MST operation), or to compute the path between two vertices in the tree.

Evert:

The evert operation is not a requirement of the project, but it is a very useful operation, and it simplifies many aspects of the project. Given an MST (represented as multiway rooted tree) and given any node u in this tree, the operation `evert(u)` produces a modified tree which has the same edge structure but in which u is made the root of the tree. The figure below shows an example of the evert operation, on the abstract multiway tree (left) and on the firstChild-nextSibling representation (right). We have shown parent links as dotted lines.

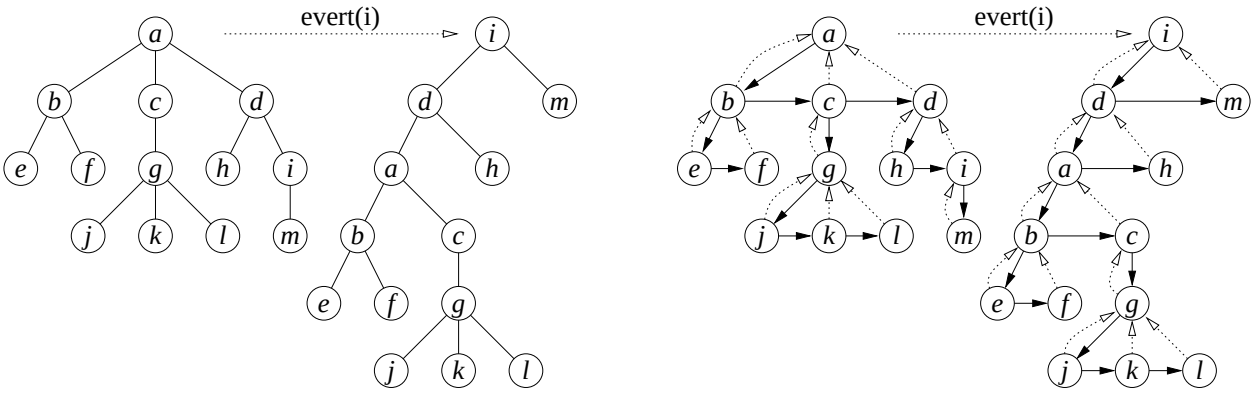

Figure 2: Evert operation on a multiway tree.

As a hint to implementing evert, you may want begin by implementing two utility operations, `cut(u)`, which cuts a node u and its associated subtree out of a tree by deleting the link to its parent in the multiway tree, and `link(u,v)`, which assumes that u is the root of a tree which has been cut, and links this subtree rooted at u as a new child of node v. The figure below show the result of applying `cut(c)` and `link(c,d)`. Evert can be implemented as a sequence of cuts and links.

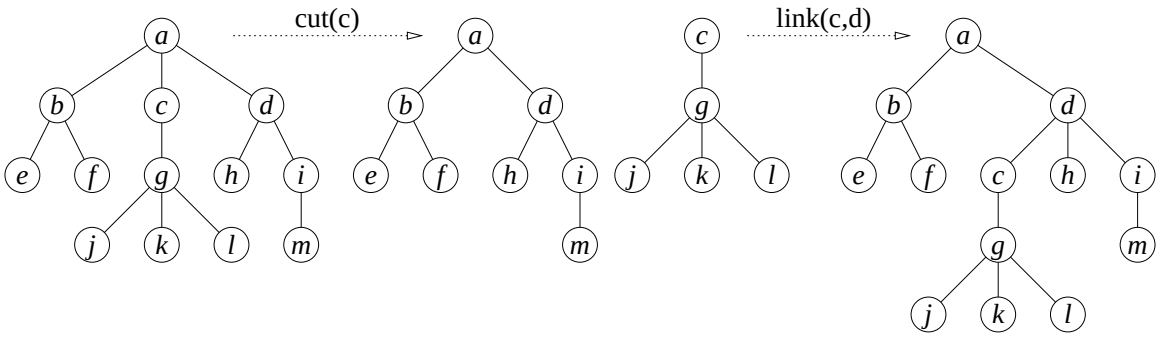

Figure 3: Application of `cut(c)` and `link(c,d)`.

Output Format: For each directive output the directive, for the path and print-mst directives print the associated output, and print any error message, if needed.

Each MST should be printed in preorder. Start each line with a string ". . . " to indicate depth. For consistency of output, the children of each node should be listed in increasing order by the string identifier. One easy way to implement this ordering is that whenever a node u is added as a child to a another node v, insert u in sorted order among v's children.
For example, for the tree shown in Figure 2, the operation "`print-mst i`" would produce the output shown on the right. This can be done by invoking `evert(i)`, and then applying a preorder traversal of the tree.

```
Directive-----------------> print-mst i
  i
  . d
  . . a
  . . . b
  . . . . e
  . . . . f
  . . . c
  . . . . g
  . . . . . j
  . . . . . k
  . . . . . l
  . . h
  . m
```

Test Data: I posted test data on the class webpage.

Submission Details: Your submission will consist of an encapsulation of files which you will email directly to me and our TA (nph87903@gmail.com). Please read the following instructions carefully, since significant deviations can result in the loss of points. If you are not using Java, your program should run on one of the lab machines for me to run it. If I am not able to run your program I cannot grade it. (It doesn't matter how it runs on your PC.) The encapsulation must include the following items:

README: There must be a file called `README`, which contains any helpful information on how to compile and run your program. This includes:

Your name: (Very important.)

How to compile

How to run it Explain how to execute your program on a given input file.

Known Bugs and Limitations: List any known bugs, deficiencies, or limitations with respect to the project specifications.

File directory: If you have multiple source or data files, (other than those created by the compiler), please explain the purpose of each.

Source Files: All the source files and header files.

Input Files (if applicable): If you would like me to run your your program on specific test data (e.g. because your program has limitations that prohibit me from using our test data) please include that.

DO NOT INCLUDE: Please delete all executable and object (.o or .class) files prior to submission.

To submit, store everything in a directory whose name is lastname (or something easily identifiable) e.g. `smith`. In particular, DO NOT send directories with names like `project5`. Be sure to delete any unnecessary files (executable or ".o" files).

Then zip the directory and email to me. If you discover an error in an earlier submission, you may repeat your submission until the deadline. But in consideration to our time and disk quota, please keep the number of submissions to a minimum and name your submission identical to the previous ones (previous ones will be overridden). I will grade the last submission I receive. Server errors are rare but can occur. Be sure to save your final submission somewhere safe (very important). I reserve the right to deduct points for people that deviate significantly from these instructions.