CISC220 Project Spring 2013

Project Checkpoint 1 due April 18.
Project Checkpoint 2 due May 2.
Final Project due May 12.
Note that this document and the Sakai assignment only includes part 1. Additional parts will be posted soon if you would like to work ahead.

You may not help other people debug their programs, except in the very limited way described in the syllabus. You may not use any code written by others, whether they are students or not. The TA and the instructor are available for office hours, and by appointment if you can't make hours because of a class conflict.

Groups

You may work with any person from any lab section in groups of 2 for this project. The professor reserves the right to change the group roster at any time. Permission is required from the professor to work in a group of 3 (this permission will not be granted except in extreme circumstances). You are free to work on your own if you prefer.

Overview

The goal of this project is to help with railway planning for a next generation high speed railway in the US. Several assumptions will be made to simplify the domain so that we can directly apply techniques learned in class:

- All rail links in the network are assumed to be straight lines between two endpoints on a cartesian graph, and capacity is not a concern (it is assumed that adding capacity is a much easier task than laying new track and can be done by simply adding railcars or additional trains to an existing route).
- All links are bi-directional (so the resulting graph is undirected).
- All links will originate and terminate at railway stations.
- Each railway station in the rail network provides an "on-ramp" and "off-ramp" for travelers. In this way we do not care about the interface or implementation of regional train/tram networks. Indeed, a traveler may arrive at the railway station by taxi, ride the train to an intermediate destination, and travel to their final destination by car. We are only concerned with the traveler from the point they enter the rail network to the point they depart the rail network.

Using these simplifications, we create a physical representation of the existing network as such:

- S = set of all rail stations
- L = set of all existing links, where each link has endpoints in S

Part 1 - Implementing a Heap Structure

Consider the min ordered BinaryHeap implementation available on Sakai. This implementation uses a C++ STL Vector (http://www.cplusplus.com/reference/stl/vector/) to store BinaryHeapNode instances. We will need a completed implementation of a min ordered BinaryHeap for our algorithms in Part 2 of the project.

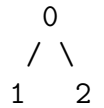
Each BinaryHeapNode has an index (its location in the heap) and a data value, which is a pointer to a Data instance.

Your task is to implement the following functions on a MinBinaryHeap. The headers for the functions are listed in the BinaryHeap.h and BinaryHeap.cpp files. There are various TODO flags placed in the BinaryHeap.cpp where you need to provide implementation:

```
/**
 * Returns the BinaryHeapNode that is the left child of the given node.
 */
BinaryHeapNode * MinBinaryHeap::getLeftChild(BinaryHeapNode * node) const

/**
 * Returns the BinaryHeapNode that is the right child of the given node.
 */
BinaryHeapNode * MinBinaryHeap::getRightChild(BinaryHeapNode * node) const

/**
 * Returns the BinaryHeapNode that is the parent of the given node.
 */
BinaryHeapNode * MinBinaryHeap::getParent(BinaryHeapNode * node) const

/**
 * Updates the position of the given node in the heap.  It is
 * assumed that the position is the same or has increased in heap order.
 *
 * (this is a min heap, so it is a percolate/bubble down operation)
 */
void MinBinaryHeap::increaseKey(Container * container)

/**
 * Updates the position of the given node in the heap.  It is
 * assumed that the position is the same or has decreased in heap order.
 *
 * (this is a min heap, so it is a percolate/bubble up operation)
 */
void MinBinaryHeap::decreaseKey(Container * container)
```

All of the functions listed follow the basic min ordered binary heap rules discussed in class. Keep in mind that you need to maintain correct indices in the underlying vector data structure in addition to the index property of a BinaryHeapNode. Also note that the insert and deleteMin operations of the BinaryHeap are already implemented but depend heavily on your correct completion of the above 5 functions.

For examples please consult the included BinaryHeapTest.cpp. The basic test uses the following min ordered heap:



The values 0, 1, and 2 are placed at indexes 0, 1, and 2 in the Vector. Value 1 is the left child of value 0 because it is at $index * 2 + 1$ where index is 0 for value 0.

Please make sure to consult the BinaryHeapTest.cpp tests if you need clarification for the expected result of each function.

- Download the code from Sakai. Compile/run the BinaryHeapTest.exe to see that the existing implementation does not properly update the nodes to maintain heap ordering.
- Edit BinaryHeap.cpp to implement the above MinBinaryHeap functions. All code changes should be at locations marked with the TODO flags.
- Re-make the test and run it to ensure it is successful.