Computer Science 2530
April 15, 2020

Happy Wednesday, April 15.

Get to work on assignment 7 as soon as possible. Resolve to finish it early. I have moved the due date to April 26.

Assignment 7

Assignment 7 will initially look difficult. But if you approach it in the right way, ***using successive refinement to break it down into manageable pieces***, you will find that is not nearly as difficult as you initially thought. Here are some hints.

1. You cannot succeed at assignment 7 unless you have a working priority queue module. I will try to get you feedback on it quickly if it does not work so that you can fix it.

2. You have more flexibility to make changes in this assignment than in previous assignments. You can add additional parameters to functions. But do not change the required Graph representation, and write the required functions.

3. Like Assignment 5, Assignment 7 uses weighted graphs. But Assignment 7 uses a different representation of graphs, using linked lists. In addition to the number of vertices and the number of edges, a Graph has an array with a slot for each *vertex*. Each vertex number i has a few pieces of information associated with it (so it is a structure).

   (a) A vertex i stores a real number that is the time at which the first signal reached i. (See below.)

   (b) A vertex i stores the sender of the first signal that reached i.

   (c) A vertex i stores a linked list, called its *adjacency list*, of edges that are incident on vertex i.

4. Think of an edge as a road and a vertex as an intersection of roads. All roads are bidirectional. But think of roads as divided. A road between vertex i and vertex j is thought of as two one-way roads, one from i to j and one from j to i.

You will need an Edge structure that represents a one-way road. Be sure that your Edge structure makes a clear distinction between the start vertex and the end vertex of the one-way road. An Edge structure is used as a cell in an adjacency list, so it need to hold a pointer to the next Edge in the list.

The one-way road from i to j occurs in the adjacency list for vertex i. The one-way road from j to i occurs in the adjacency list of j.

5. You will need readGraph and writeGraph functions, plus helpers for them. But they will not be identical to the corresponding functions in Assignment 5 because the representation of a Graph is different.

   WriteGraph should have one helper function that writes the edges in a particular adjacency list.

   ReadGraph should have two helper functions.

   (a) One helper inserts an edge into a adjacency list. (Make it add the new edge to the beginning of the list.) For example, insertOneWayEdge(L, u, v, w) inserts an edge from u to v with weight w into adjacency list L.

   (b) Another helper function does the entire job of inserting a bidirectional edge into a graph. insertEdge(g, u, v, w) inserts a bidirectional edge between u and v, of weight w, into graph g.

6. The program simulates vertices sending *signals* to one another. This program keeps a collection of *events* by storing them in a priority queue that is called the *event queue*. Each event represents the arrival of a signal, and it holds: the time at which the signal will arrive, the number of the vertex that sent the signal and the number of the vertex that will receive the signal.

   Events are processed chronologically. You will need a function that contains an *event loop*, which repeatedly gets an event out of the event queue and then processes the event.

7. You will need a function that processes an event. If the receiver of the signal has previously received a signal, then processing that event *must do nothing.*

   If the receiver r of the signal has not previously received a signal, then

(a) The time and sender stored with vertex number r are set to the arrival time and the sender of the event.

(b) A signal is sent from r to every vertex i where there is a (one-way) road from r to i. Just look at the adjacency list of vertex number r.

You will need a function to send one signal and another function to send a signal from a given vertex r to every vertex that is adjacent to r. Think about the parameters that those functions need.

8. This program is required to have switchable tracing. Trace the following.

   (a) Whenever a signal is sent, show what is happening in the trace. **Show the current time first**, then which vertex is sending the signal, which vertex will receive the signal and when the signal will arrive.

   (b) Show a signal being received. **Show the current time first**. Then show the vertex that sent the signal, the vertex that received the signal, and whether the signal is ignored.

   (c) If a signal is not ignored, show the time (or distance) that is stored for the receiving vertex.

   (d) Be sure that traces are easy to read and understand. It must be easy to tell the difference between a trace of a signal being sent and a signal being received.

   It must be easy to turn tracing on and off. You main function should have the following heading.

```
int main(int argc, char* argv[])
```

   The operating system passes an array of strings to main. If a program is invoked by command

```
./dijkstra -t
```

   then array argv contains two strings, with argv[0] = "./dijkstra" and argv[1] = "-t". Parameter argc is the number of strings that are in array argv (2 in the example).

   Write a function that checks whether argc $>$ 1 and argc[1] is "-t". Use strcmp to compare strings. If argc[q] is "-t", then

your function should set your global tracing variable to 1. (Make tracing = 0 by default.)

Do not check argv[0]. That is not relevant, and might not be what you think it will be. Programs can be invoked in many ways.

Tracing is an aid to debugging. Put the tracing in as you go so that you can use it to debug what you have. Do not add all of the tracing after everything has already been debugged.

Deletion from a binary search tree

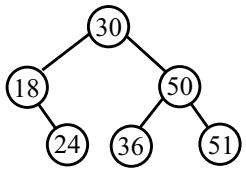
Page **41A** of the notes describes how to delete a value from a binary search tree. There are several cases to deal with. Read page **41A** to get a feel for how deletion is done. I won't ask you to perform a deletion from a binary search tree.

Height-balanced binary search trees

Our goal is to find a way to represent a set of numbers where it takes time $O(\log_2(n))$ to ask whether a number is in the set, to insert a number into the set and to remove a number from the set. As shown at the beginning of page **41B**, binary search trees, as we have done them up to now, do not achieve that goal.

But there is a way to modify the insert and remove functions so that the goal is achieved. We just need to keep trees *balanced*, so that a tree is wide and bushy rather than long and spindly.

To do that, we need four definitions. We use the following tree t_0 for illustration.



Tree t_0

Definition 1. Suppose v is a node in a tree. The *height of node* v is the length of the longest path from v downward to a leaf, were we

count both the start and end vertices in the path. For example, the root of t_0 has height 3. A leaf has height 1.

Definition 2. Suppose t is a binary tree. The *height of tree* t is defined as follows.

(a) If t is an empty tree, then t has height 0.

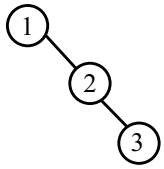
(b) If t is not an empty tree, then t has the same height as its root.

For example, the height of tree t_0 is 3.

Definition 3. Suppose v is a node in a binary tree with left subtree L and right subtree R. Let h_L be the height of tree L and h_R be the height of tree R. Say that **node** v **is height-balanced** if $|h_L - h_R| \leq 1$. That is, node v is height balanced if the heights of its two subtrees differ by no more than 1.

Definition 4. Say that **tree** t **is height-balanced** provided all of its nodes are height-balanced. (An empty tree is height-balanced by definition.)

Tree t_0 is height-balanced. The following tree is not height-balanced. Its root has two subtrees, one of height 0 (an empty tree) and the other of height 2.



Our goal is to modify the insertion and removal algorithms for binary search trees so that they keep the tree height balanced. That is the next topic.