Homework 8

Out: 11.28.16
Due: 12.7.16

1. [Single-Source Shortest Path, 20 Points]
   a. Graph G represents a computer network, where vertices represent switches and edges represent communication lines joining pairs of switches. Each edge has an associated bandwidth. The bandwidth of a path is defined as the bandwidth of the lowest bandwidth edge in the path. Give an algorithm that, given a graph and two switches *a* and *b*, outputs the maximum bandwidth of a path between *a* and *b*. What is the runtime of your algorithm?
   b. G(V,E) is a weighted, directed graph with no negative-weight cycles. We define m as the maximum number of edges on any shortest path from the source s to a destination v. Describe a modification to the Bellman-Ford algorithm that allows it to terminate in m+1 passes, even if m is not known in advance.

2. [Dynamic Programming, 20 Points]
   a. You are given a currency system with coins of decreasing value $c_1, c_2, \ldots, c_N$ cents. Give a dynamic programming algorithm that computes the minimum number of coins required to give K cents in change.
   b. The longest common subsequence problem is as follows: Given two sequences $A = a_1, a_2, \ldots, a_M$, and $B = b_1, b_2, \ldots, b_N$, find the length, k, of the longest sequence $C = c_1, c_2, \ldots, c_k$ such that C is a subsequence (not necessarily contiguous) of both A and B. For example, if $A = d, y, n, a, m, i, c$, and $B = p, r, o, g, r, a, m, m, i, n, g$, then the longest common subsequence is a, m, i, and has length 3. Give a dynamic programming algorithm to solve the longest common subsequence problem. Your algorithm should run in O(MN) time. Explain.

3. [All-Pairs Shortest Paths, 10 Points]
   Give an algorithm to find the negative weight cycle with the minimum number of edges in a graph. Your algorithm should return the number of edges in such a cycle, or declare that no such cycle exists. Provide the runtime of your algorithm.

4. [Binary Search Trees, 50 points]
   Consider two binary search trees that contain the same set of unique keys, possibly in different orders.
   a. Devise and analyze an efficient algorithm that will transform any given binary search tree into any other binary search tree (with the same keys) using only ZIG and ZAG rotations.
   b. The provided *BST.h*, and *BST.cpp* files contain a BST class, implementing a binary search tree, and a Rotation class, which stores a rotation.
   Implement a new derived class of BST, MyBST, which extends the binary search tree with a *transform* method, which implements your algorithm from part (a). You should also implement a main function, which receives two files, *T1.txt* and

T2.txt (you may assume that these will be the names of the input files), containing one integer per line, to be inserted in order into two binary search trees. The *transform* method should receive the BSTs generated from T1 and T2, and return a vector of rotations required to transform T1 into T2, which is then printed out. You may not modify *BST.h* and *BST.cpp*, but you may add methods to *MyBST.cpp* as you see fit (and add a *MyBST.h* file).

Assuming that the required rotations are a ZIG rotation on pivot=3, followed by a ZAG rotation on pivot=8, your output should be as follows:

ZIG on 3
ZAG on 8

To get full credit your solution must work on any two binary search trees that contain exactly the same set of keys.

Submit your solution, consisting of your MyBST.cpp, MyBST.h, main.cpp, and possibly a text file with description, along with a modified *makefile* that was used to compile it on the lab computers. No additional files should be submitted.

Write your name in a comment at the top of the main file of the program, along with an explanation of your approach (part a).