EC330 Applied Algorithms and Data Structures for Engineers
Fall 2018

Homework 7

Out: November 21, 2018
Due: November 30, 2018

This homework has a written part and a programming part. Both are due at 8:59 am on November 30. You should submit both parts on Blackboard. For the written part, you should submit a single PDF file containing either typeset answers or scanned copies of hand-written answers. Make sure you write your answers clearly. For the programming part, your code should be easy to read and understand, and demonstrate good code design and style. Your program must compile and run on the lab computers.

1. Dijkstra's Algorithm [10 pt]

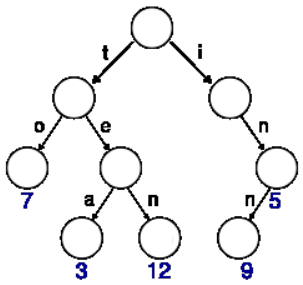
In class, we have discussed an implementation of Dijkstra's Algorithm using min-heap. Analyze the worst-case running time of an implementation of this algorithm using unordered linked-list (as the data structure for d(v), the upper bound on the shortest distance from source s to v). Give your answer in Θ. Justify your answer (and state any assumptions you make).

2. Programming [90 pt]

For the programming part (part b), make sure to write your name and BU ID in a comment at the top of the program, along with your collaborator's name and BU ID, if any. To use the compiler on the lab computers, run "*module load gcc*" first.

a. Implement a **trie-based map**. A map stores data in the form of (key, value) pairs where every key is unique and each key maps to a value. You can assume that the keys are strings composed of lowercase letters only, and the values are positive integers. Implement the `insert`, `search`, and `remove` methods in `trie.h`.

For example, below is the result trie after the sequence of insertion ("to", 7), ("tea", 3), ("ten", 12), ("in", 5), ("inn", 9). Note that the size of the resulting map should be 5 and the size of the resulting tree should be 9.



You are encouraged to create your own main with test cases. **Submit only the modified trie.h file**. **[40 pt]**

b. Imagine that you are managing a supercomputer that runs jobs for your clients. A client can submit a set of computing jobs, numbered from *1* to *n*, and the dependencies among them. The dependencies are expressed as pairs. For example, (1, 2) means job 2 depends on job 1, i.e. job 2 can only start after job 1 finishes. Write a program that determines if it is possible to finish all the jobs without actually running them. Below are some example inputs and their expected outputs.

Input: number of jobs = 2, dependencies = [(1, 2)]
Output: true
Explanation: We can run job 1 first followed by job 2.

Input: number of jobs = 2, dependencies = [(1, 2), (2, 1)]
Output: false
Explanation: We need job 1 to finish before we can run job 2, and job 2 to finish before we can run job 1. It is clearly impossible.

Implement the method canFinish in job.h. In the written part of your answer, state and justify both the time and space complexity of your algorithm. **Submit both the modified job.h file and the written complexity analysis. [50 pt]**