EC330 Applied Algorithms and Data Structures for Engineers
Spring 2022

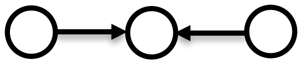
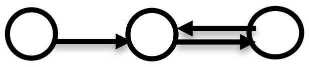
Homework 6

Out: April 9, 2022
Due: April 20, 2022

This homework has a written part and a programming part. Both are due at 11:59 pm on April 20. You should submit both parts on Gradescope.

This is an individual assignment. See course syllabus for policy on collaboration.

1. **Celebrity [10 pt]**
   We model a group of $n \geq 3$ people as a directed graph G with n nodes (one node per person) and an edge from node u to node v if person u knows person v. A person x is called a *celebrity* if everybody else knows x, but x does not know any other person in the group. For instance, the middle node in the left graph below is a celebrity, whereas there is no celebrity in the graph on the right.

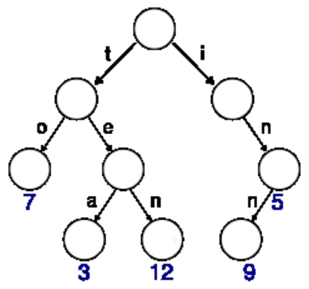
   If the group has a celebrity, can the graph G have an *Eulerian cycle* (a cycle that visits every edge exactly once)? Can G have a *Hamiltonian cycle* (one that visits every node exactly once)? Justify your answers.

2. **Bipartite Graph [10 pt]**
   Prove that if a graph is bipartite, then it cannot contain any cycle with an *odd* number of distinct vertices.

3. **Trie [30 pt]**
   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 *delete* methods in trie.cpp. For *delete*, if you remove a leaf node, you do not need to remove redundant ancestors if there are any. **[30 pt]**

   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.



Submit the *trie.cpp* file on Gradescope.

4. **Job Scheduling as Graph Problems [50 pt]**
Imagine that you are managing a supercomputer that runs jobs for your clients. A client can submit a set of 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.

   a) Write a program that determines if it is possible to run all the jobs without actually running them (assuming each job takes finite time to run). **[20 pt]**

      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. This is clearly impossible.

      Implement your algorithm in the method *canFinish* in *job.cpp*.

   b) Write a program that checks if a given job *j* can be run in the *i*th time slot on a sequential computer. You can assume that *i* ranges from 1 to *n*, each job can finish in one time slot, and you can run no job in a slot. You can also assume the answer to part a) is true for the input given in this part. **[30 pt]**

      Below are some example inputs and their expected outputs.

      Input: number of jobs = 2, dependencies = [(1, 2)], j = 2, i = 1
      Output: false
      Explanation: We have to run job 1 first before we can run job 2, so job 2 cannot be run as the 1st job.

Input: number of jobs = 2, dependencies = [(1, 2)], j = 1, i = 2
Output: true
Explanation: we can run no job in the 1st time slot, and then run job 1 in the 2nd time slot.

Implement your algorithm in the method *canRun* in *job.cpp*.