EC330 Applied Algorithms and Data Structures for Engineers Spring 2020

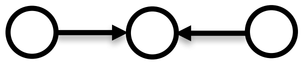
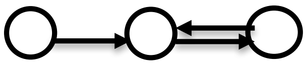
Homework 7

Out: April 14, 2020
Due: April 21, 2020

This homework has a written part and a programming part. Both are due at 11:59 pm on April 21. You should submit both parts on Gradescope.
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, you should make sure your code compile and run on Gradescope. You should also try to create your own test cases to test and debug your code. Make sure you write your name and BU ID in a comment at the top of the program, and use clear comments to explain the key steps in your code.

1. **Bipartite Graph [20 pt]**
   Describe a linear-time algorithm for determining whether a given undirected graph is bipartite. You can write your algorithm in pseudo code similar to the ones I gave in recent lectures, or write actual C++ code. Justify your answer (make sure you describe the data structures that you use).

2. **Celebrity [10 pt]**
   We model a group of *n* 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.

3. **Dijkstra's Algorithm [10 pt]**
   Analyze the worst-case running time of an implementation of Dijkstra's 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).

4. **Programming [60 pt]**

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.

a) 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. 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. 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*.