Problem Set 3 for CS 170

Note

When asked for an algorithm you must give (1) a brief informal description of the algorithm, (2) a precise description using pseudo-code, (3) an informal argument for termination and correctness of the algorithm, and (4) an analysis of the running time of the algorithm. Be clear about what the input to the algorithm is, how you measure the size of the input, and what constitutes a "step" in your running-time analysis.

Problem 0. [Any questions?] (5 points)

What's the one thing you'd most like to see explained better in lecture or discussion sections? A one-line answer would be appreciated.

(Sometimes we botch the description of some concept, leaving people confused. Sometimes we omit things people would like to hear about. Sometimes the book is very confusing on some point. Here's your chance to tell us what those things were.)

Problem 1. [Amalgamated Networks] (30 points)

Amalgamated Networks is a new company started by your friends. They've bought spare capacity in fiber-optic networks and hope to become a major telecommunications player. They promise to deliver quality service, and want you to analyze their network to make sure that they have enough redundancy in place. Specifically, they're concerned about damage to wires because of backhoes. They want to avoid the situation where they can't transmit data from one part of their network to another.

They hand you a list of k cities $c_1, \ldots, c_k$ and a list of connections between the cities where they have links. Their network is initially connected. You may assume that links are bi-directional. Your task is to design an algorithm which can find a *critical-link* in their network. A link is critical if the removal of that link eliminates the ability for some city in the network to communicate with some other city.

Problem 2. [Instruction Scheduling] (30 points)

It's the year 2017 and computer hardware is highly advanced. Recent breakthroughs from Berkeley have created an architecture which supports infinite parallelism. This means that the CPU can execute *any number* of instructions in one cycle. Programs still take longer than 1 cycle since some instructions take the results of other instructions as their input. Such instructions can only run after any instructions that they are dependent on complete.

A *program* is a list of instructions, $I_0, I_1, \ldots, I_k$. The program also contains a list of dependencies. For each I_j, there is a dependency set D_j which contains a list of instructions that must be completed before I_j can run.

A *valid scheduling* for a program is an ordered list of sets of statements which honors the dependencies for that program. For example, a program can be specified as $\langle \{I_0, I_1, I_2, I_3, I_4\}, D_0 : \{I_1\}, D_2 : \{I_0, I_1\} \rangle$

And a valid schedule:

$$\{I_1, I_3, I_4\}$$
$$\{I_0\}$$
$$\{I_2\}$$

Design an algorithm which takes a program (the list of instructions plus the dependencies for each instruction) and outputs the shortest valid CPU scheduling for that program. The length of a schedule is the number of cycles that it takes to execute the schedule.

Problem 3. [Almost Connected] (35 points)

Let's define a directed graph $G = (V, E)$ to be $1-$*almost strongly connected* if adding a well placed edge makes it a strongly connected graph.

Construct an algorithm that checks whether a graph is 1-almost strongly connected. If it is 1-almost strongly connected, output an example of an edge whose addition would make the graph strongly connected.