CS 477/677 Analysis of Algorithms

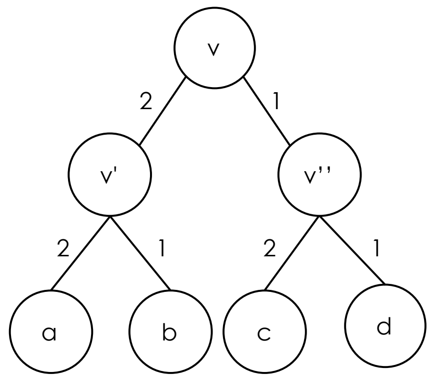
Homework 7

Due December 3, 2019

1. [50 points] (U&G-required)

The binary tree below represents a model of a timing circuit for VLSI design. The tree is a **complete**, **balanced** binary tree, with *n* leaves (*n* is a power of two). Each tree edge has a length l_e (always positive). We define the *distance* from the root of the tree to a leaf as the sum of all the lengths of all the edges on that path from the root to the leaf. The root node generates a *clock signal* which travels along the edges toward all the leaves. The time necessary for the signal to reach a leaf is proportional with the *distance* between the root and the leaf. A problem with the VLSI design is if the leaves do not have the same distance from the root, because the signal will not reach all the leaves at the same time. In a properly designed circuit the leaves are completely synchronized, meaning that they receive the signal at the same time and the tree has *no skew*. To ensure this synchronization, we have to *increase* the lengths of some edges (we are not allowed to reduce edge lengths). Given a tree representing a VLSI timing circuit, write pseudocode for an efficient algorithm that increases the lengths of some edges such that the resulting tree has *no skew* and the total edge length (sum of all distances for all leaves) is *minimized*.

Note: to prove that your greedy strategy yields the optimal solution, you have to prove that the problem has the *greedy-choice property*.



Example: in the shown tree, the optimal solution would increase all the edges with length 1 to 2. This will result in a tree with no skew and a total edge length of 12.

2. [50 points] (U&G-required)

A search engine company needs to do a significant amount of computation every time it recompiles its index. For this task, the company has a single large supercomputer, and an unlimited supply of high-end PCs.

They have broken the overall computation into n distinct jobs, labeled $J_1, J_2, \ldots, J_n$, which can be performed completely independently of one another. Each job consists of two stages: first it needs to be *preprocessed* on the supercomputer, and then it needs to be *finished* on one of the PCs. Let's say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC.

Since there are at least n PCs available on the premises, the finishing of the jobs can be performed fully in parallel – all the jobs can be processed at the same time. However, the supercomputer can only work on a single job at a time, so the system managers need to work out an order in which to feed the jobs to the supercomputer. As soon as the first job in order is done on the supercomputer, it can be handed of to a PC for finishing; at that point in time a second job can be fed to the supercomputer; when the second job is done on the supercomputer, it can proceed to a PC regardless of whether or not the first job is done (since the PCs work in parallel); and so on.

Let's say that a *schedule* is an ordering of the jobs for the supercomputer, and the *completion time* of the schedule is the earliest time at which all jobs will have finished processing on the PCs. This is an important quantity to minimize, since it determines how rapidly El Goog can generate a new index.

Give a polynomial-time algorithm that finds a schedule with as small a completion time as possible.

Note: to prove that your greedy strategy yields the optimal solution, you have to prove that the problem has the *greedy-choice property*.

4. [20 points] (G-required)

Exercise 16.1-2 (page 422).

5. [20 points] (Extra credit)

Exercise 16.3-3 (page 436).