College of Computer & Information Science
Northeastern University
Spring 2019

CS5800
Algorithms
8 April 2019

Sample Solutions to Midterm 2

Problem 1. (5 points) Directed Acyclic Graphs

Give an algorithm that takes as input a directed acyclic graph $G = (V, E)$ and returns a path that visits every vertex of the graph; if no such path exists, then your algorithm should indicate so.
State the worst-case running time of your algorithm, in terms of the number of vertices and edges of G.
Your grade for this question will be determined on the basis of the correctness of your algorithm and its efficiency, given by its worst-case running time. Partial credit may be given for non-optimal algorithms provided they are correct and well explained.

Answer:

1. Find topological ordering of G. Let this order be $v_1, v_2, \ldots v_n$.
2. Check if there is an edge (v_i, v_{i+1}) for each $1 \leq i < n$. If this is true for all i, return the path $v_1 \to v_2 \to \cdots \to v_{n-1} \to v_n$. If this is not true, then indicate that no desired path exists.

The first step takes $\Theta(n + m)$ time where n is the number of vertices and m the number of edges in G. The second step takes $\Theta(n)$ time. The total time is $\Theta(n + m)$.

Problem 2. (4 + 3 = 7 points) MSTs and shortest paths

For each of the following claims, indicate whether it is true or false. If true, provide a short proof. If false, provide a counterexample.

(a) Let G be a given weighted undirected graph. Let T_1 and T_2 be two minimum spanning trees of G that differ in only one edge. Let e_1 be the edge in T_1 that is not in T_2 and e_2 be the edge in T_2 that is not in T_1. Prove or disprove the following statement: The cycle obtained when e_1 is added to T_2 contains e_2.

Answer: True. If the cycle obtained when e_1 is added to T_2 does not contain e_2, then this cycle also exists in T_1, a contradiction. Note that this problem applies to any spanning trees T_1 and T_2.

(b) Let G be a weighted directed graph and s be a vertex in G. Let p denote a shortest path from s to t.
Prove or disprove: If we increase the weight of every edge in the graph by 1, then p remains a shortest path from s to t.

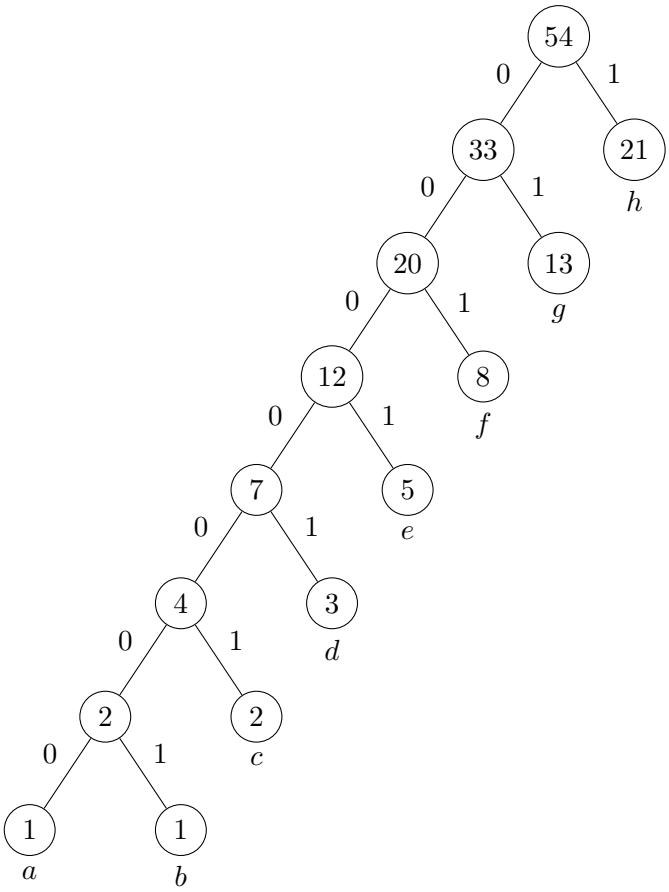
Answer: False. Consider four vertices s, a, b, t, with edges (s, a), (a, b), (b, t), and (s, t) of weights 1, 1, 1, and 4, respectively. The shortest s-t path is $s - a - b - t$ of weight 3. When we add weight 1 to every edge, this path is no longer shortest.

Problem 3. (4 + 2 = 6 points) Huffman Encoding

(a) What is an optimal Huffman code for the following set of frequencies based on the first 8 Fibonacci numbers? Draw the binary tree and provide the corresponding code.

$$a : 1, b : 1, c : 2, d : 3, e : 5, f : 8, g : 13, h : 21$$

Answer: The binary tree corresponding to the Huffman encoding of the first 8 Fibonacci numbers is given below.



The following table summarizes the code and code length for each symbol.

Symbol	Code	Code Length
a	0000000	7
b	0000001	7
c	000001	6
d	00001	5
e	0001	4
f	001	3
g	01	2
h	1	1

(b) Generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers.

Answer: The optimal code for a set of symbols whose frequencies are the first n Fibonacci numbers will be $n - 1$ for the first two symbols, and then reducing in length by 1 for each successive symbol with a code length of 1 for the symbol with the highest frequency.

Problem 4. (6 points) Scheduling activities in two lecture halls We are given a set S of n activities that we would like to schedule among two lecture halls. For each activity, we have a start time s_i and a finish time f_i. Two activities that overlap in time may not be scheduled in the same lecture hall; they may, however, be scheduled separately in the two lecture halls.

Design a greedy algorithm to select a maximum-size set of activities from S that may be scheduled among the two lecture halls. State the worst-case running time of your algorithm.

Answer: This can be solved by a simple extension of the greedy algorithm for activity scheduling. Maintain a current set X of unprocessed activities and set Y of selected activities. Initially, X is S and Y is $\emptyset$. Repeatedly:

(i) Find an activity i in X with earliest finish time;

(ii) If i can be scheduled in any of the two lecture halls, add i to Y and place it in the lecture hall with largest finish time of a scheduled activity;

(iii) delete i from X.

Return Y .

The worst-case time complexity for this algorithm is $\Theta(n)$ as it requires a single scan through the set S of activities.

Problem 5. (6 points) Optimal location of stores

You are opening a chain of stores, and plan to choose from n potential locations, all of which are located in a straight line. These locations are numbered 1 through n. The distance between location i and $i + 1$ is given by d_i miles, for all $1 \leq i < n$. At any location i, at most one store can be opened, and a store at location i is valued at vi dollars. You would like to open stores of maximum total value, subject to one constraint: any two opened stores are at least k miles apart. Give a polynomial-time algorithm for determining the maximum total value that can be achieved by an optimal solution for locating the stores. If your algorithm is based on dynamic programming, present a recurrence followed by pseudocode. State the running time of your algorithm (no analysis needs to be presented). You need not prove the correctness of your algorithm. Your algorithm need not compute the actual store locations in your solution.

Answer: Let $M[i]$ denote the maximum total value attainable by placing stores in a subset of locations 1 through i. For convenience, we set $M[0] = 0$. We then obtain the following recurrence.

$$M[i] = \max\{M[i-1], v_i + M[j_i]\}.$$

where j_i is the largest index less than i such that the distance between j_i and i is at least k; if there is no such index, then set $j_i = 0$.

The pseudocode just consists of first finding j_i for each i. A naive implementation takes times $O(n^2)$. Then, a linear-time pass to calculate $M[i]$ for each i, as i increases from 1 to n. There is also a faster way of computing j_i, for each i, in a separate linear scan through the list, which would lead to a running time of $O(n)$.