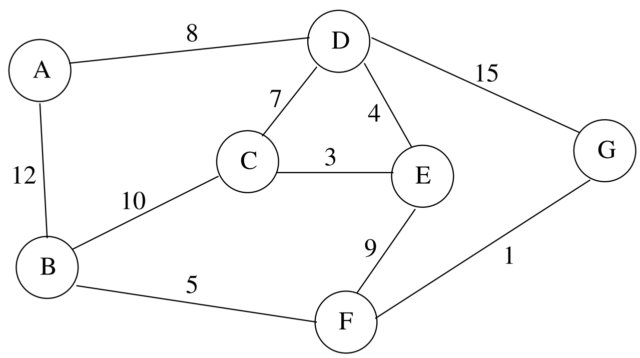
Homework 7

CS6033 Design and Analysis of Algorithms I
Fall 2023
(Sec. B, Prof. Yi-Jen Chiang)

Due: Wed. 12/13 by 1pm
(submit online on NYU Brightspace; one submission per group)
Maximum Score: 111 points

Note: This assignment has 2 pages.

1. (10 points)
Consider the undirected connected weighted graph below:



(a) What are the FIRST FIVE (5) edges added to the minimum spanning tree by the Kruskal's algorithm, in the order they are added? Name an edge by its endpoints, e.g., AB. **(5 points)**

(b) What are the FIRST FIVE (5) edges added to the minimum spanning tree by Prim's algorithm, started at vertex A, in the order they are added? **(5 points)**

2. (10 points)
What is an optimal Huffman code for the following set of symbols and their corresponding frequencies?

```
a:18, b:5, c:30, d:24, e:42.
```

Show the tree after each iteration, and give the final Huffman code for each symbol. Use the convention that the frequency of the left child is **no larger than** the frequency of the right child.

3. (40 points)
This question considers the knapsack problem discussed in the Textbook Section 15.2.
(Recall that there are n items; each item i has value v_i and weight w_i. The total weight of the items that are put into the knapsack cannot exceed the knapsack capacity W. Also, each w_i, as well as W, is a **positive integer**.)

(a) Prove that the fractional knapsack problem has the greedy-choice property (where the greedy choice is as discussed in Section 15.2). **(10 points)**

(b) Now we consider the following two versions of the **integral** knapsack problem:

Version 1: Integral knapsack with repetition:
There are unlimited quantities of each item available. Each item i can be taken **multiple** times (including 0 time), but it **cannot be taken partially**.

Version 2: Integral knapsack with at most one repetition:
Same as **Version 1**, except that at most one repetition is allowed, i.e., each item i can be taken **at most twice**.

(1) Give a dynamic programming algorithm to solve **Version 1** in $O(nW)$ worst-case time. **(12 points)**

(2) Give a dynamic programming algorithm to solve **Version 2** in $O(nW)$ worst-case time. **(18 points)**

4. (15 points)
In the **art gallery guarding** problem, we are given a line L that represents a long hallway in an art gallery. We are also given a set $X = \{x_0, x_1, \cdots, x_{n-1}\}$ of real numbers that specify the positions of n paintings in this hallway. Suppose that a single guard can protect all the paintings within distance at most 1 of their position on both sides. Design and analyze a greedy algorithm to find a placement of guards that uses the minimum number of guards to protect all the paintings at positions in X. You need to prove the correctness of your algorithm by proving the greedy-choice property and optimal substructure.

5. (18 points)
You are given a connected, undirected, weighted graph $G = (V, E)$ where each edge e has a positive weight $w(e)$, and a minimum spanning tree T of G. Now suppose some edge (u, v) of G has its weight $w(u, v)$ **increased** to some new value w'. Call this updated graph G'. Design and analyze an algorithm to perform the necessary update(s) to T, if any, so that it is a minimum spanning tree of G'. Your algorithm should run in $O(V + E)$ worst-case time.

6. (18 points)
Given a directed acyclic graph (DAG) $G = (V, E)$ where each edge e has a weight $w(e)$ ($w(e)$ can be > 0 or < 0) and a vertex $s \in V$, design and analyze a dynamic programming algorithm to compute the lengths of the **min-bottleneck paths** from s to all other vertices, where a **min-bottleneck path** from s to a vertex v is a path from s to v whose **longest edge (i.e., the bottleneck edge)** along the path is the **shortest** possible (and such edge length is the length of the min-bottleneck path).
Define your cost function $d()$ for each vertex v, derive a recursive solution for $d()$, and explain how to compute $d()$ for all vertices v. Be careful to explain in what order the computation takes place, and analyze the running time. Your algorithm should run in $O(V + E)$ worst-case time.