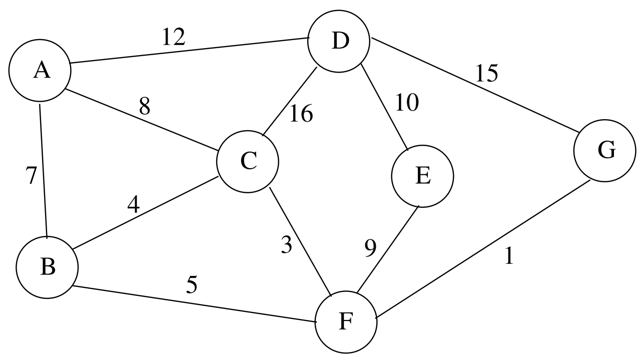
# Homework 7

CS6033 Design and Analysis of Algorithms I
Fall 2024
(Sec. B, Prof. Yi-Jen Chiang)

**Due: Wed. 12/11 by 1pm**
**(submit online on NYU Brightspace; one submission per group)**
**Maximum Score: 113 points**

*Note: This assignment has 3 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 $D$, 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:10, b:7, c:22, d:46, e:32, f:38.
```

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: Decision problem of integral knapsack with no repetition**:
Same as **Version 1**, except that no repetitions are allowed, i.e., each item $i$ can be taken **at most once**. Moreover, we only want to know whether there is a subset of the $n$ items that can fully fill up the knapsack, i.e., whose total weight is **exactly** $W$.
(**Note:** This is a **decision problem** — the task is to report either **yes** together with the desired subset of items, or **no**.)

**(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 $d$ ($d > 0$) 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. (20 points)**
Given a connected, undirected, weighted graph $G = (V, E)$ where each edge $e$ has a positive weight $w(e)$, a **bottleneck spanning tree** $T$ is a spanning tree of $G$ whose largest edge weight is minimum over all spanning trees of $G$, i.e., for each spanning tree of $G$ look at its largest edge weight, and $T$ is the one whose largest edge weight is the smallest among all spanning trees of $G$. Prove that a minimum spanning tree of $G$ is a bottleneck spanning tree of $G$.
(**Hint:** Prove by contradiction using a "swapping" argument, by applying the idea of cut edges (i.e., edges across a cut). Such a "swapping" argument can also be found in the proof of Theorem 21.1 in the Textbook.)

**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 $< 0$), a vertex $s \in V$ and an integer $k \in (1, V - 1)$, design and analyze a dynamic

programming algorithm to compute the lengths of the **longest paths** from $s$ to all other vertices, where each longest path uses **at least** $k$ **edges** (if a vertex $v$ cannot be reached from $s$ at all or can only be reached from $s$ with fewer than $k$ edges, then report "No Such Path" for $v$ instead of a path length).

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 $O(V(V+E))$ worst-case time.