CS 3310: Design and Analysis of Algorithms
Final Exam
(5/11/2023)

Name : Loc Nguyen______________________

Last 4 digits of your Student ID #: 2624______________________

Read these instructions before proceeding.

- Closed book. Closed notes. You can use calculator.
- Length: **50 minutes**
- *Important Notes:*
    - *During the exam, all students need to join the Zoom meeting*
    - *No questions will be answered during the exam about the exam questions. Write down your assumptions and answer the best that you can.*
    - *Just in case you have trouble of submitting your exam here @Canvas, alternative way is to submit your completed exam to Prof. Young by emailing*
      gsyoung@cpp.edu

- Two ways to submit your exam:
    - Print out the exam paper. Write your answers on the exam paper. Scan your completed exam papers or take photos of them. Then turn in **one PDF file** here @ Canvas.
    - Read the exam from the computer screen and answer questions on your own white papers (number your answers). Scan your exam answers or take photos of them. Then turn in **one PDF file** here @Canvas.

Q.#1 (18)	Q.#2 (22)	Q.#3 (20)	Q.#4 (20)	Q.#5 (20)	Total (100)

1. (18 pts)

Decide ***<u>"Easy"</u>*** or ***<u>"Hard"</u>*** for each of the following problems that we discussed in the class. **<u>No justifications are required</u>**. (3 pts each)

(a) ***Longest Path*** Problem

Hard

(b) ***Shortest Path*** Problem

Hard

(c) ***0/1 Knapsack*** Problem

Hard

(d) ***Satisfiability (SAT)*** Problem

Easy

(e) ***Shortest-length-First Greedy algorithm for Optimal Storage on Tapes*** Problem

Easy

(f) ***Select algorithm using MM as pivot for Selection*** Problem (as in project 2 of your programming assignment).

Easy

2. (22 pts)

Part I: (10 pts) Let A = {20, 6, 5, 1} be a finite set of coin types. Each type in A is available in unlimited quantity. The coin changing problem is to make up an exact amount C using a minimum total number of coins. C is an integer > 0. A greedy solution to the problem will make change by using the coin types in the order of 20, 6, 5, 1. When a coin type is being considered, as many coins of this type as possible will be given. Give a counterexample to show that this greedy algorithm doesn't necessarily generate solutions that use the minimum total number of coins.

Let C = 24. With a greedy algorithm, it will take one 20 coin and four 1 coins, total up to 5 coins. However, the optimal solution is four 6 coins, total up to 4 coins and adds up to C =24. Therefore, the greedy algorithm will doesn't necessarily generate solutions that use minimum total number of coins.

Part II: Decide ***True*** or ***False*** for each of the followings. You MUST briefly justify your answer. (4 pts each)

(e) (4 pts) (4 pts) It is known that PARTITION is a NP-complete problem. Assume that an O(n^{2023}) deterministic algorithm has been found for the PARTITION problem, then P = NP.

True, because Partition is an NP-complete problem and there exist an algorithm solvable in polynomial time of n^{2023} so P=NP. Note that all other NP-complete problems also are reducible to Partition in polynomial time

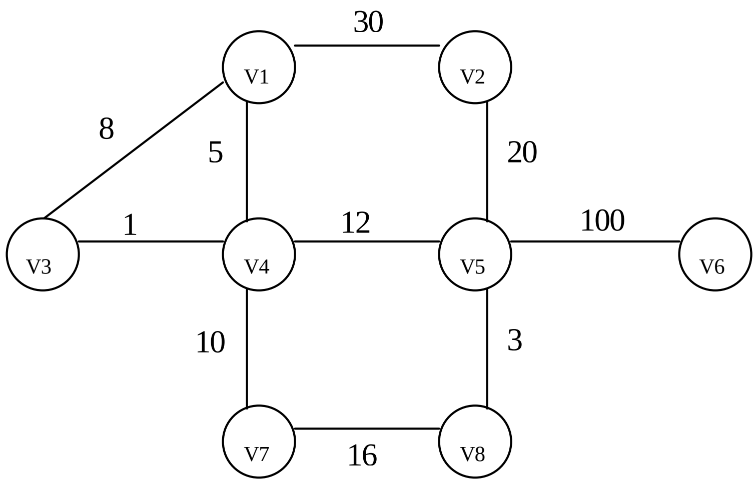
(f) (4 pts) If A is in NP, proving that A is solvable in polynomial time is sufficient to show that P = NP.

False. Since A is solvable in polynomial time, to prove that P = NP, we also needs to prove that it's an NP-complete problem. Since we haven't prove that A is an NP-complete problem, we haven't shown that P = NP

(g) (4 pts) If a decision problem A is NP-complete, proving that A is reducible to B, in polynomial time, is sufficient to show that B is NP-hard.

True. Because NP-complete problems is a subset of NP-hard problems, if A is reducible to B, then B is NP-complete and therefore, B is NP-hard.

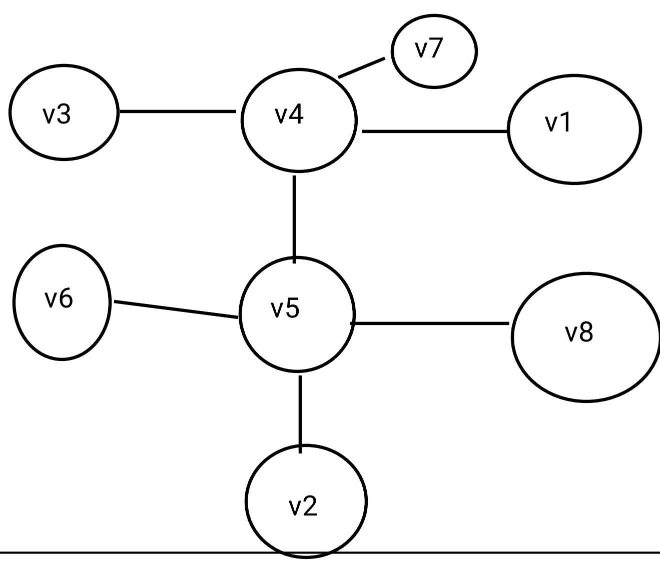
3. (20 points)



Assume that the algorithm starts from v1. Use **Kruskal's Algorithm** to find a minimum spanning tree and the cost of the minimum spanning tree for the above graph. ***Show all steps.***

(v3,v4) = 1	Step 1: Connect v3 and v4. New set
(v5,v8) = 3	Step 2: Connect v5 and v8. New set
(v1,v4) = 5	Step 3: Connect v1 and v4. Since they came from different sets, there's no cycle created
(v1, v3) = 8 X	Step 4: Discard (v1,v3) because a cycle would be created.
(v4, v7) = 10	Step 5: Connect v4 and v7. Since they came from different sets, there's no cycle created
(v4,v5) = 12	Step 6: Connect v4 and v5. Since they came from different sets, there's no cycle created
(v7, v8) = 16 X	Step 7: Discard (v7, v8) because a cycle would be created
(v2, v5) = 20	Step 8: Connect v2 and v5. Since they came from different sets, there's no cycle created
(v1, v2) = 30 X	Step 9: Discard (v1, v2) because a cycle would be created
(v5, v6) = 100	Step 10: Connect v5 and v6. Since they came from different sets, there's no cycle created
	MST completed with 7 edges

Cost of MST = 1 +3 + 5 + 10 + 12 + 20 + 100 = 151



4. (20 pts)

Given M_1 = [3×10], M_2 = [10×2], and M_3 = [2×5]. Use the dynamic programming algorithm (covered in the class) to compute the C(i,j)'s for all i < j and give the solution (i.e. the multiplication order) for this instance of **<u>Chained Matrix Multiplication</u>**..

(Hint: C(i, j), where $i \le j$, represents the minimum cost of computing $M_i \times M_{i+1} \times \dots \times M_j$. In this question, you need to find C(1,3), and then give the solution, the optimal multiplication order. You start from initializing all C(i,j)'s of size 1, and they are C(1,1), C(2,2), and C(3,3). Next is finding all C(i,j)'s of size 2, and so on)

d0 = 3
d1 = 10
d2 = 2
d3 = 5

C[1,2] => k=1 and min{C[1,1] + C[2,2] + d0 x d1 x d2} = 0 + 0 + 3 x 10 x 2 = 60
C[2,3] => k=2 and min{C[2,2] + C[3,3] + d1 x d2 x d3} = 0 + 0 + 10 x 2 x 5 = 100
C[1,3] => k=1 so {C[1,1] + C[2,3] + d0 x d1 x d3} = 0 + 100 + 3 x 10 x 5 = 250
k=2 so {C[1,2] + C[3,3] + d0 x d2 x d3} = 60 + 0 + 3 x 2 x 5 = 90

Solution is (M1 x M2) x M3 with cost of 90 operations. Multiply M1 and M2 first, then M3 by the resultant matrix of M1 and M2.

5. (20 pts)

Consider the **<u>0-1-2-3 Knapsack Problem</u>** obtained by replacing the 0/1 constraint of the 0/1 knapsack problem by x_i = 0 or 1 or 2 or 3 instead (i.e., we assume three copies of the i^{th} object are available, for all i). Description of the **<u>0-1-2-3 Knapsack Problem</u>** is given as follows.

<u>0-1-2-3 Knapsack Problem</u>:

<u>Instance:</u> n objects, $p_1, p_2, \ldots, p_n$ profits, knapsack of capacity M, $w_1, w_2, \ldots, w_n$ weights

<u>Question:</u> To find $x_1, x_2, \ldots, x_n$ s.t. $\sum p_i x_i$ is maximized subject to $\sum w_i x_i \leq M$ where x_i = 0 or 1 or 2 or 3.

For the dynamic programming functional equation to solve the **<u>0-1-2-3 Knapsack Problem</u>**, we define $f_i(X)$ = max profit generated after considering the first i objects (or values of $x_1, x_2, \ldots, x_i$) subject to capacity X.

(a) (3 pts)

$$f_0(X) = \boxed{0}\ ,\ \forall\ 0 \leq X \leq M$$

(b) (12 pts)

We assume that $f_i(X) = -\infty$, $\forall\ 0 \leq i < n$ *and* $X < 0$

$$f_i(X) = \max \left\{ \boxed{\begin{array}{l} f_{i-1}(X) \\ p_i + f_{i-1}(X-W_i) \\ 2p_i + f_{i-1}(X-2W_i) \\ 3p_i + f_{i-1}(X-3W_i) \end{array}} \right\}$$

$$\forall\ 1 \leq i \leq n \text{ and } 0 \leq X \leq M$$

(c) (5 pts) Analyze the complexity of your algorithm in part (c)? **Justify your answer.**

This algorithm first initialize variables weight and value. Then, for each item I and each knapsack weight from 0 to W, and for each k from 0 to 3, we try to find the maximum value that can be obtained by not including an object, include it once, include it twice, or include 3 times. Lastly, we return the max value for knapsack with weight capacity W and no copies of any item left to check.

Time complexity is O(nW) because the algorithm iterate over n, W, and 0,1,2,3 for values of k. Since each iteration of k takes constant time, hence the TC = O(nW). To put it another way, if we were to use a (M+1)(n+1) table with O(1) each operations would yield O(Mn).