CSC363: Assignment #5

Due on Friday, April 9, 2021 by 11:59pm

Total Marks: 20 $(+4)$

Question 1

(*5 marks*) Let S be a finite multiset of positive integers. A multiset is a set that allows duplicate elements. Define the following language:

$$\begin{aligned}\text{Partition} = \{S : S \text{ is a finite multiset of positive integers and} \\ \exists S_1, S_2 \subseteq S \text{ s.t. } \Sigma S_1 = \Sigma S_2 \text{ and } S_1 \cap S_2 = \varnothing \text{ and } S_1 \cup S_2 = S\}\end{aligned}$$

If a set $S \in$ Partition, then S can be split into two sets S_1 and S_2 where the summation of S_1 is equal to the summation of S_2.
For example, the set $\{6, 1, 4, 4, 3\} \in$ Partition because you can partition the set into $\{6, 3\}$ and $\{1, 4, 4\}$, so that $6 + 3 = 1 + 4 + 4 = 9$. On the other hand, the set $\{3, 2, 4, 2, 1, 5\} \notin$ Partition.

Prove that Partition is NP-complete.

(*Hint: Recall that the Subset-Sum language from lecture is NP-complete.*)

Question 2

(*Designed by Eric, 5 marks*) We shall define the language *MULTI-SAT* as

$$\textit{MULTI-SAT} = \{\phi : \phi \text{ is a boolean formula with } \textit{at least two} \text{ different satisfying assignments}\}$$

Prove that *MULTI-SAT* is NP-complete by reducing from a problem which we already know is NP-complete. *Any solution that does not include a reduction* (*e.g. attempting to replicate the proof of the Cook-Levin theorem*) *will not receive full marks. But you probably wouldn't want to do this anyway...*

Question 3

(*Designed by Paul, 5 marks*) Recall the verifier definition of NP: $A \in$ NP if and only if there exists a poly-time verifier V for A:

$$A = \{w : \text{There exists a string } c \text{ such that } V(w, c) \text{ accepts}\}.$$

We say a language $A \subseteq \Sigma^*$ is **co-NP** if its complement $\bar{A}$ is NP. Using the verifier definition of NP, this means $A \in$ co-NP if and only if there exists a poly-time verifier V such that

$$A = \{w : \text{There does not exist a string } c \text{ such that } V(w, c) \text{ accepts}\}.$$

In other words, we can check whether something does not belong to A in polynomial time.

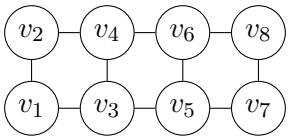
(a) (*1 mark*) Give a reasonable definition for "co-NP-complete".

(b) (*2 marks*) Show that P $\subseteq$ co-NP.

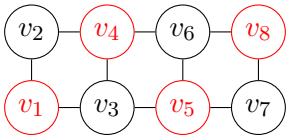
(c) (*2 marks*) Show that P $=$ NP if and only if P $=$ co-NP.

Question 4

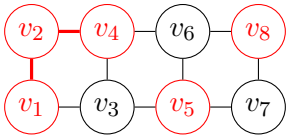
(*Designed by Huzaifa, 5 marks*) Let us define the **Independent Set Problem**, IS. Given a graph $G = (V, E)$ (where V is the set of vertices and E is the set of edges), we say a set of vertices $S \subseteq V$ is an **independent set** if there doesn't exist two vertices $v, w \in S$ such that there is an edge $(v, w) \in E$ between them. For example, let G is the following graph:



Then $S = \{v_1, v_4, v_5, v_8\}$ is an independent set, since there aren't any edges connecting two vertices in S:



$T = \{v_1, v_2, v_4, v_5, v_8\}$ is not an independent set, since there is an edge connecting two vertices in T, such as the edge (v_1, v_2):



In fact, G as defined above has no independent set with 5 or more vertices, so indeed the largest possible independent set in G has size 4.
Define the language IS:

$$\text{IS} = \{(G, k) : G \text{ is a graph that has an independent set with at least } k \text{ vertices}\}.$$

We will show that IS is NP-complete.

(a) (*1 mark*) Show IS $\in$ NP.

(b) (*4 marks*) Show 3SAT $\leq_p$ IS. Conclude that IS is NP-complete.

Hint: There are two ways to do this question.

(1) If you haven't yet, Sipser's book gives a proof that the vertex covering problem is NP-complete, by creating a poly-time reduction from 3SAT to the vertex covering problem. See Theorem 7.34 on page 261. You'll do something similar for (b)).

(2) You may instead choose to show that the vertex covering problem reduces to IS. Then combining 3SAT $\leq_p$ Vertex-Cover and Vertex-Cover $\leq_p$ IS, we will have 3SAT $\leq_p$ IS.

Question 5

(*Designed by **you know who**... 4 bonus marks*) This question is an exercise to apply what you know about computational complexity to a real problem, and determine if it can be solved in polytime or if it is a hard problem.

You have been hired by an MMO game company to analyze the computational complexity of a main feature they are currently working on. The game you are working on is called *Players versus Non-Players*, or *PvsNP* for short (see what we did there?).

Players in *PvsNP* can play the game and obtain **items**. There are finitely many items (some number α) in the game. We can represent all of the items as a set:

$$I = \{i_1, i_2, \ldots, i_{\alpha-1}, i_\alpha\}$$

It is important to note that the game may add or remove items in the future, so α is not fixed and neither is I.

During gameplay, a player may choose to sell some of their items to other players in the game's official **Auction House**, which is the main feature that the company would like to launch soon.

A player can post a **sell order** on the Auction House. A sell order consists of a list of quantity-item tuples, and an *asking price*. Formally, a sell order S can be represented as a tuple over $\mathcal{P}(\mathbb{Z}^+ \times I) \times \mathbb{N}$:

$$S = (I_S, p) \text{ where } I_S = \{(q_1, i_{a_1}), \ldots, (q_j, i_{a_j})\} \text{ so } |I_S| = j > 0, \text{ and } p \in \mathbb{N}$$

For example, if a seller wishes to sell 2 of i_5 and 10 of i_1 for 8 units of currency total, then the sell order would be formatted as $(\{(2, i_5), (10, i_1)\}, 8)$. We consider sell orders that contains the same item more than once to be invalid, so $(\{(2, i_3), (3, i_3)\}, 5)$ is invalid, but $(\{(5, i_3)\}, 5)$ is valid. You also cannot sell 0 of an item, so $(\{(0, i_1)\}, 5)$ is invalid.

Sell orders cannot be partially fulfilled. It must either be not bought at all, or be bought entirely for the asking price (then the buyer will receive every item and their quantity in the sell order, even if it contains items the buyer did not originally want).

The Auction House can be represented as a finite multiset of sell orders. If there are β sell orders, then we can define the following set to be the Auction House:

$$AH = \{S_1, S_2, \ldots, S_{\beta-1}, S_\beta\}$$

Identical sell orders are permitted inside AH, so $AH = \{((2, i_1), 5), ((2, i_1), 5)\}$ is allowed.

And now the problem: The Auction House can be comprised of arbitrarily many sell orders, which may make fulfilling **buy orders** hard to do. A buy order has the same idea as a sell order, but the asking price has been replaced with a maximum budget. Formally, a buy order B can be represented as a tuple over $\mathcal{P}(\mathbb{Z}^+ \times I) \times \mathbb{N}$:

$$B = (I_B, b) \text{ where } I_B = \{(q_1, i_{a_1}), \ldots, (q_j, i_{a_j})\} \text{ so } |I_B| = j > 0, \text{ and } b \in \mathbb{N}$$

Notice it is defined the same way as a sell order.

Whenever a buy order is placed in the Auction House, the server will check if it can immediately fulfill the buy order within the player-defined budget. If it can, then some other server code is invoked to finalize the purchase and notify the necessary players of the sale. If it cannot, then a failure notice is sent to the buyer and the buy order is discarded.

Question 5 continued on next page...

The company is giving you access to two polytime computable functions to make your job easier. Let A be an Auction House, and $|A| = \beta$ so that $A = \{(S_1, p_1), \ldots, (S_\beta, p_\beta)\}$ then let the following functions be defined in the following way:

- $TotalPrice(A) = \sum_{k=1}^{\beta} p_k = p_1 + p_2 + \ldots + p_\beta$. This takes in an Auction House and outputs the total value of the Auction House, so that any buyer with that amount can buy out the entire Auction House.
- $TotalQuant(A, i) = \sum_{k=1}^{\beta} \sum_{j=1}^{|S_k|} q_{k,j}(S_{k,j} == i)$. This takes in an Auction House and an item, and outputs how many of this item is being sold on the Auction House. (You can think of this as a double for-loop, first iterating A then iterating each S to find i)

The language that encapsulates this decision problem is defined:

$$\begin{aligned} Fulfillable = \{&(B, AH, I) : B \text{ is a buy order and } AH \text{ is an Auction House that only sells} \\ &\text{items from } I, \text{ and there exists a } M \subseteq AH \text{ such that} \\ &b \leq TotalPrice(M) \wedge \forall (q, i) \in I_B, q \leq TotalQuant(M, i)\} \end{aligned}$$

If a buy order B can be fulfilled in some Auction House AH, then there exists a subset $M \subseteq AH$ which contains all the items required in the buy order with at least the desired quantity, and it stays below (or is equal to) the budget of the buy order.

Your job within the company is to either show that the problem can be solved in polytime, or show that it is a hard problem. The company personally prefers that you find a polytime algorithm (so that they can launch the new system without issue), but a convincing proof that it is hard will give the company the go-ahead to find a better and faster system. The company has total faith in you ☺.

One last thing, the company has already proven that $Fulfillable \in$ NP. So it is not required that you also show this in your solution as no marks will be awarded for it. You are encouraged to show it for yourself regardless, as it may help with figuring out what to do next.

For full marks, show exactly one of these two claims:

1. $Fulfillable \in$ P
2. $Fulfillable$ is NP-hard (and therefore NP-complete)

(*Hint: If you are trying to show it is NP-hard, restrict your search to four potential candidates for your reduction: Vertex-Cover, 0-1 Knapsack, Subset-Sum, 3SAT*)