CS570 Spring 2024: Analysis of Algorithms Exam III

	Points		Points
Problem 1	20	Problem 5	14
Problem 2	9	Problem 6	18
Problem 3	15	Problem 7	12
Problem 4	12		
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Closed book. No electronic devices or internet access. One 8.5x11 cheat sheet allowed.
2. If a description to an algorithm or a proof is required, please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure, so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.
8. This exam is printed double sided. Check and use the back of each page.

1) 20 pts

Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification.

[TRUE/FALSE]

3-SAT and Max 3-SAT are both NP-hard

[TRUE/FALSE]

For optimization problems X and Y, if $X \leq_p Y$ and there exists an α-approximation algorithm for Y, then there exists an α-approximation algorithm for X.

[TRUE/FALSE]

If we want to prove that an optimization problem X is NP-hard, it's sufficient to prove that X is polynomial-time reducible to 3-SAT.

[TRUE/FALSE]

If P = NP, then there exists a polynomial-time algorithm for at least one NP-hard problem.

[TRUE/FALSE]

Assuming that P≠NP, if there is a 5-approximation algorithm for the general Traveling Salesman Problem, then there exists a polynomial-time algorithm for 3-SAT.

[TRUE/FALSE]

The decision variant of Vertex Cover is NP-hard.

[TRUE/FALSE]

Linear Programming is polynomial-time reducible to Max-Flow.

[TRUE/FALSE]

The linear program representing a Max Flow problem on a graph with integer capacities has at least one optimal solution where all the variables have integer values.

[TRUE/FALSE]

If X is an optimization problem, then there must not exist an efficient certifier for X.

[TRUE/FALSE]

If P = NP, then the Shortest Path problem is NP-complete.

9 pts (3 pts each)
Circle ALL correct answers (no partial credit when missing some of the correct answers). No need to provide any justification.

i- Consider the following problem: Given a graph G, is the largest independent set in G of size k? This problem is in:

a) NP

b) NP-hard

c) NP-complete

d) Undecidable

ii- Which of the following statements about linear programming are true?

a) Linear programming can be used to solve problems with linear objective functions and linear constraints.

b) The feasible region of a linear program is always a convex region.

c) There always exists an optimal solution at a vertex of the feasible region.

d) The Simplex algorithm is a polynomial-time algorithm for solving linear programs.

iii- If P ≠ NP, then which of the following statements must be true?

a) There does not exist a polynomial-time algorithm for 3-SAT.

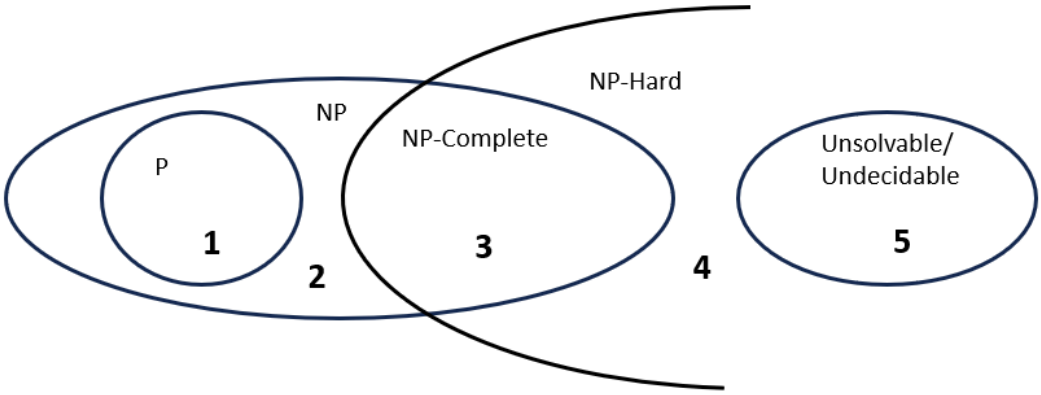
b) Independent Set $\leq_p$ 3-SAT

c) 3-SAT $\leq_p$ Max-Flow

d) Max-Flow $\leq_p$ 3-SAT

3) 15 pts

For each of the regions 1 through 5 marked in the complexity class diagram below, give an example of one problem that currently belongs to that region and specify whether the problem is a decision problem or an optimization problem. Assume P≠NP.



a) Region 1:

Is there a path from A to B in G? (decision)

b) Region 2:

Are graphs G and G' isomorphic? (decision)

c) Region 3:

decision version of 3-SAT

d) Region 4:

Optimization version of Independent Set

e) Region 5:

The halting problem (decision)

4) 12 pts

Recall the 0/1 Knapsack Problem: Given a knapsack capacity W and n items, where each item $i = 1, \ldots, n$ has non-negative value v_i and non-negative weight $w_i \leq W$, select a subset of items $S \subseteq \{1, \cdots, n\}$ with maximal total value $\sum_{i \in S} v_i$ subject to the constraint that the total weight of the selected items does not exceed the knapsack capacity, i.e., $\sum_{i \in S} w_i \leq W$.

Consider the following greedy approximation algorithm for the 0/1 Knapsack Problem: Initialize the knapsack of selected items S as an empty set. Sort the items in decreasing order of the ratio v_i / w_i (breaking ties arbitrarily). Iterate over the items $i = 1, \ldots, n$. Add item i to the knapsack if doing so does not violate the knapsack capacity constraint. Otherwise, stop iterating and the total value of the current knapsack with v_i, the value of the item that could not be added to the knapsack without violating the knapsack capacity constraint. If v_i is larger, return $\{v_i\}$, otherwise return S. In other words, if the total weight of items $1, \ldots, k$ is less than or equal to the knapsack capacity but the total weight of items $1, \ldots, k + 1$ exceeds the knapsack capacity, then return $S = \{1, \ldots, k\}$ if $\sum_{i = 1, \ldots, k} v_i \geq v_{k+1}$, otherwise return $S = \{k + 1\}$.

Show that this is a ½-approximation algorithm. That is, prove that this algorithm returns a knapsack with at least half as much value as an optimal knapsack.

Pseudocode for algorithm in problem #4

1. $S = \{\}$ // Initialize solution to empty set.
2. $W_{\text{filled}} = 0$, $V_{\text{filled}} = 0$.
3. Sort the items in decreasing order of the ratio v_i / w_i (breaking ties arbitrarily). 4. For $i = 1, \ldots, n$:

    If ($W^{\text{filled}} + w^i \leq W$):

        S = S U $\{i\}$. // Add i to the knapsack if it fits

        $W^{\text{filled}} = W^{\text{filled}} + w^i$

        $V^{\text{filled}} = V^{\text{filled}} + v^i$

    Else: break.
5. If (|S| = n): Return S. // Everything fit, so return that.
6. Else:

    If ($V_{\text{filled}} \geq v_i$): Return S

    Else: Return $\{i\}$ // i was the last item that did not fit

Solution:

Let OPT be the optimal solution. If we can put all items in the knapsack, then it is clearly an optimal solution. Otherwise, suppose we cannot add item i' to the knapsack.
Let $x = W$ - $\sum_{i = 1 \ldots i' - 1} w_i$ be the residual capacity when trying to add item i'.
Clearly, if i' did not fit, x < w_{i'}
Now,

$$\sum_{i = 1 \ldots i'} v_i >= \sum_{i = 1 \ldots i' - 1} v_i + v_{i'} * x / w_{i'}$$
$$= \sum_{i = 1}^{i' - 1} (v_i / w_i) * w_i + (v_{i'} / w_{i'}) * x$$
$$>= OPT ..$$

The last inequality follows since we are using items in decreasing order of v_i / w_i and exhausting all capacity W.
since \sum_{i = 1}^{i'} p_i >= OPT as shown above, hence, either \sum_{i = 1}^{i' - 1} p_i >= 1/2 OPT or p_{i'} >= 1/2 OPT, so we do output a set of items with total value >= 1/2 OPT.

5) 14 pts

For the Trojan Candy Company, there are three products, *X*, *Y*, and *Z*. The company has a limited amount of resources, including production time, labor, and materials. The goal is to maximize profits subject to resource constraints. The following information is available:

- Each unit of product *X* requires 2 hours of production time, 3 hours of labor, and 2 units of raw materials.
- Each unit of product *Y* requires 3 hours of production time, 2 hours of labor, and 3 units of raw materials.
- Each unit of product *Z* requires 2 hours of production time, 4 hours of labor, and 1 unit of raw materials.
- The company has 300 hours of production time, 400 hours of labor, and 200 units of raw materials available.
- The profit earned per unit of product *X* produced is $20, the profit earned per unit of product *Y* produced is $30, and the profit earned per unit of product *Z* produced is $15.
- The company has to supply the canteen with at least 50 units of product *X*, 75 units of product *Y*, and 25 units of product *Z*.

Follow the steps below to formulate this problem as an instance of Linear Programming:

a) What are the variables of your linear program?
b) What is the objective that your linear program maximizes?
c) List all the constraints of your linear program.

Solution:

a) variables x, y, z for #units of the 3 products resp.
b) max(20x + 30y + 15z)
c) subject to the following constraints:

- 2x + 3y + 2z <= 300
- 3x + 2y + 4z <= 400
- 2x + 3y + z <= 200
- x >= 50
- y >= 75
- z >= 25

6) 18 pts

Consider the following two variants of the Independent Set problem:

(i) Connected Independent Set: Given a connected graph $G = (V, E)$ and an integer k, determine whether there exists an independent set S of size k in G such that after deleting each of the vertices in S from G, the graph still stays connected.

(ii) Independent Set On Trees: Given a graph $G = (V, E)$ that is a tree, find the largest independent set in G.

One of these problems is NP-complete, and the other has an efficient solution.

a) State which of them is NP-complete, and which one is not. No justification needed for this part. (3 pts)

b) For the problem you identified as NP-complete in part a), prove the problem is NP-complete by following the steps below.

b-1) prove that the problem is in NP (5 pts)

See next page for part b-2

b-2) prove that the problem is in NP-hard (10 pts)

Hint: Use a reduction from the Independent Set problem.

Answer:

a) `ConnectedIndependentSet` is NP-complete.
b) A set of nodes S as certificate. Certifier checks it has size at least k, no edge between each pair of nodes, and after removing S from G, the residual graph can be checked to be connected using BFS/DFS . All the steps run in poly-time.
c) We reduce from IndSet problem. There are several possible reductions, but the easiest is to just add a new node w to the graph G (fron the arbitrary IS instance) and connect to all the existing nodes - to obtain G'. Then, G has an independent set of size k iff G' has an ind. set of size k, which when removed, ensures the residual graph is connected.
   proof)
   Suppose G has an Ind Set S of size k. S is also an ind. set in G'. If S is removed from G', the resultant graph is connected due to w as per the construction (w clearly remains after S is removed.
   On the other hand, suppose G' has an Ind Set S of size k (which when removed, the residual graph is connected). S cannot have w (unless k =1) since it is adjacent to all other nodes (for completeness: k=1 case can be easily handled directly on the IS instance without invoking the blackbox - OKAY to miss this caveat). Thus, S only has nodes from G, making it an ind. set of size k in G.

Rubrics: Refer to Gradescope.

7) 12 pts

A filmmaker is looking to make a movie with a set of actors $\{1, \ldots, m\}$ available to be cast. Assume that all the actors are versatile enough to play any role and actor j charges a fee of f_j to play any role they are cast in. The filmmaker has some flexibility in the story to introduce new roles, but at the very minimum, she needs p actors. She has a budget B for the actors' fees. To maximize the audience engagement, she resorts to a survey with a set of people $\{1, \ldots, n\}$ where each person i reports a set L_i of actors they like and a set D_i that they dislike. A person will go to watch the movie if it has at least one of the actors they like, and none of the actors they dislike. Complete the integer linear program below to determine which actors she should cast given her constraints, to maximize the number of people who will watch the movie (w.r.t. the survey)

a) Our integer variables will be:

Binary xj's to denote if actor j is cast.
(xj=1 indicating that actor j is cast, xj=0 indicating otherwise)

Binary yi's if person i will watch the movie.
(yi=1 indicating that person i will watch the movie, yi=0 indicating otherwise)

b) Objective function: (2 pts)
Maximize \sum yi

c) Constraints (10 pts):

$\sum$ xj $\geqslant$ p	…. At least p actors
$\sum$ fj xj $\leqslant$ B	…. At most B budget for fees
$\forall$ i: $\sum_{j \in Li}$ xj $\geqslant$ yi	…. i won't watch if none from Li cast
$\forall$ i: $\forall$ {j $\in$ Di}: 1 – xj $\geqslant$ yi	…. i won't watch if any from Di cast
xj and yi $\in$ {0,1}	…. Okay to miss since mentioned as binary in 'a'

Additional Space

Additional Space