CS570 Fall 2022: Analysis of Algorithms Exam III

	Points		Points
Problem 1	27	Problem 4	20
Problem 2	16	Problem 5	20
Problem 3	17		
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Open book and notes. No electronic devices or internet access.
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) 27 pts total (3 pts each)
Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification.

[TRUE/FALSE]
Linear Programming is polynomial-time reducible to Maximum-Flow.

[TRUE/FALSE]
If Hamiltonian Cycle is polynomial time reducible to Interval Scheduling, then P = NP.

[TRUE/FALSE]
Not every decision problem in P has an efficient certifier.

[TRUE/FALSE]
The problem of deciding whether a graph has a vertex cover of size k is NP-hard.

[TRUE/FALSE]
If $Y \leq_P X$, and there exists an efficient 2-approximation algorithm for X, then there must exist an efficient 2-approximation algorithm for Y.

[TRUE/FALSE]
Let X and Y be decision problems in NP, and assume P $\neq$ NP. If $X \leq_P Y$ and $Y \leq_P X$, then both X and Y are NP-complete.

[TRUE/FALSE]
If A is a problem in NP and if A can be solved deterministically in polynomial time, then P = NP.

[TRUE/FALSE]
Let TWO denote the problem of deciding whether a given integer is even. Then TWO is polynomial time reducible to 3-SAT.

[TRUE/FALSE]
A vertex that is part of a Minimum Vertex Cover can never be part of a Maximum Independent Set

2) 16 pts

Circle ALL correct answers and only correct answers (no partial credit when missing some of the correct answers). No need to provide any justification.

I. If there exists a strongly polynomial-time algorithm for the decision version of the Traveling Salesman problem, which of the following statements is necessarily true? (4 pts)

a) There exists a strongly polynomial-time algorithm for the decision version of 0/1 Knapsack.
b) NP-hard problems are at least as hard as problems in P.
c) $P \neq NP$.
d) All decision problems can be solved efficiently.

II. Consider an undirected graph *G*. Which of the following statements are true? (4 pts)

a) The problem of determining whether there exists a simple cycle in G is in NP.
b) The problem of determining whether there exists a simple cycle of length k in G is in NP-complete.
c) The problem of determining whether there exists a simple cycle that visits all nodes in G is in NP-complete.
d) The problem of determining whether there exists a simple cycle that visits all nodes in G is in NP-hard.

III. Which of the following statements are known to be true? (4 pts)

a) It has been proven that $P \neq NP$
b) There are problems in NP that cannot be solved in polynomial time.
c) There are problems in NP that can be solved in polynomial time.
d) There are NP-complete problems that cannot be solved in polynomial time.

IV. Given a directed graph *G*, two distinct nodes *S* and *T* in *G*, and a number *k*, which of the following decision problems are NP-complete? (4 pts)

a) Determine whether there exists a simple path of length at most *k* from *S* to *T*.
b) Determine whether there exists a simple path of length at least *k* from *S* to *T*.
c) Determine whether there exist at least *k* simple edge-disjoint paths from *S* to *T*.
d) Determine whether there exist at least *k* simple node-disjoint paths from *S* to *T*.

3) 17 pts

A chef is cooking meals for a number of food critics. The chef keeps n different ingredients in the kitchen, where n is a positive integer. Let a_i be the amount of ingredient i the chef has stored in the kitchen measured in grams, where a_i is a positive integer and $1 \leq i \leq n$. The chef knows how to cook m different recipes. Each recipe requires a different amount of each ingredient. Let b_{ij} be the amount of ingredient i required to cook recipe j measured in grams, where b_{ij} is a non-negative integer and $1 \leq i \leq n$ and $1 \leq j \leq m$. In order to maximize the variety of meals presented to the critics, the chef will make at most one of each recipe.

Consider the following problem:

Given the number of available ingredients n, *the number of possible recipes* m, *the available ingredient amounts* a_i $(1 \leq i \leq n)$, *and the recipe requirements* b_{ij} $(1 \leq i \leq n, 1 \leq j \leq m)$, *what is the maximum number of meals the chef can make without making the same meal more than once?*

Follow the steps below to express this problem as an instance of Integer Linear Programming.

a) What are the variables used by your integer linear program? What are the domains of your variables, i.e., what are the possible values for your variables? In plain English, describe what your variables represent. (6 pts)

x_j $(1 \leq j \leq m)$, where $x_j \in \{0,1\}$ and $x_j = 1$ if the chef cooks recipe j and $x_j = 0$ otherwise.

Rubric:

2 points for saying there's one variable per recipe i.e. x_j for $1 \leq j \leq m$

2 points for saying they are binary, i.e. $x_j \in \{0,1\}$ (No penalty if mentioned in part c)

2 points for explaining the meaning, i.e. $x_j = 1$ indicates the chef cooks recipe j

b) What objective does your integer linear program optimize? Specify whether your integer linear program maximizes or minimizes the objective. (4 pts)

$max \sum_{1 \leq j \leq m} x_j$

Rubric: 1 point for max; 3 points for 'sum of all x_j'

c) What are the constraints of your integer linear program? (7 pts)

$\sum_{1 \le j \le m} b_{ij} x_j \le a_i$ for all ($1 \le i \le n$)

Rubric:

3 points for correctly writing the amount of ingredient i used $\sum_{1 \le j \le m} b_{ij} x_j$

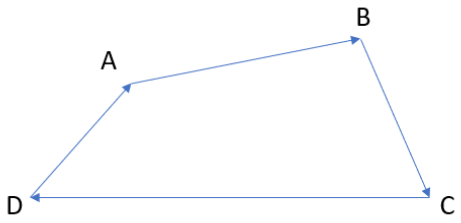
2 points for applying the constraint that amount used $\le a_i$ (available)

2 points for ensuring the constraint is for all ingredients (for all ($1 \le i \le n$))

4) 20 pts

Consider the following problem: Given a directed graph *G*, remove some edges to turn *G* into a Directed Acyclic Graph (DAG) of maximum size (i.e. with maximum number of edges).

For example, for the following graph, removing any one of the edges, e.g. AB will result in a maximum size DAG of size 3.



Our goal is to find a ½-approximation to this problem. In other words, we want to end up with a DAG that includes at least ½ the number of edges in the optimal DAG.

a) Describe an algorithm to achieve a ½ approximation for this problem (10 pts)
Hint 1: Remember topological ordering of nodes in a DAG
Hint 2: As a first try, see if a random ordering of the nodes can help you identify which edges to remove in order to achieve this approximation ratio.

1. Choose an arbitrary ordering of the nodes. (1 point)

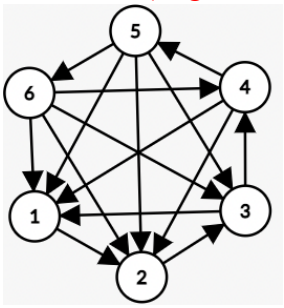
2. Delete edges that don't follow the order (i.e. delete edge (i, j) if and only if j < i) so that the remaining graph is a DAG described by the chosen ordering as its topological ordering. (4 points)
3. If the resulting DAG has size at least m/2, return this DAG; (2 points)
4. If the resulting DAG has size less than m/2, then reverse the ordering and follow the same rule as above, which will end up in the complement of the previously obtained DAG and will have size at least m/2. (3 points)
   (An equivalent way to obtain the complement DAG is to use the same ordering and keep all edges (i, j) such that j < i.)

Alternative solution: https://stackoverflow.com/a/1944536 (Partial credit of 5 if didn't recurse on the sets)

Refined Rubric:

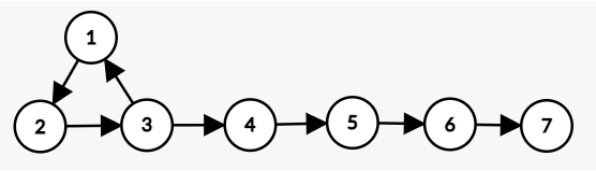
- For algorithms that are almost correct, 1-2 point penalty for each small error in each step.
- For approaches that do not work, 4 points if returning a valid DAG, 0-3 point if not.
  Examples of incorrect approaches that cannot guarantee ½ approximation:
  1. Repeatedly finding a cycle in G and removing a node/edge from the cycle (may remove more than half of the edges).
  2. Performing DFS/topological sorting on G and deleting all back edges found (count of back edges could be larger than half).
  3. Starting from an empty set, repeatedly adding edges which don't form a cycle (may not get more than half of the edges).
  4. Repeatedly picking a node with the smallest indegree, deleting the incoming edges and picking the outgoing edges; or picking a node with the largest outdegree and deleting the outgoing edges. (the correct way is to pick a node with outdegree>=indegree).

  Counterexample for repeatedly adding/removing edges: n nodes labeled 1 through n, one "forward edge" for each consecutive pair (i, i+1), one "backward edge" for every other pair (j, k) where j>k. So there are O(n) forward edges and O(n^2) backward edges. An incorrect approach could choose all forward edges / delete all backward edges so that it won't be a half approximation. Graph is for n=6 (larger n too cluttered to show):



  Counterexample for picking node with smallest indegree: all nodes have

the same indegree=1, if picking node 7 first, then edge (6,7) will be removed, and same for all the edges along the long chain. (Reverse all edges for a counterexample for picking node with largest outdegree)



b) Prove that your solution in part a achieves a ½ approximation (10 pts)

1. For any arbitrary ordering of nodes, there will be k edges consistent with one ordering and m-k edges consistent with the reverse ordering. So either k >= m/2 or m-k >= m/2. (7 points)
2. The optimal DAG with p edges naturally satisfies p <= m. (2 points)
3. Thus, we are guaranteed that either k/p >= ½ or (m-k)/p >= ½. (1 point)

Partial credit for attempts at proof, when incorrect approach given in part a:

- +2 points if mentioning optimal value p <= m.
- +1 point if the ½ ratio is calculated between the correct quantities in the right direction (mentioning size/p >= ½ or size/m >= ½).
- 0 point if arguing that the number of deleted edges is at most twice the optimal number of deleted edges. (Counterexample: m=9, p=5, k=2)

5) 20 pts

Consider a special case of the Traveling Salesman Problem called TSP379 where each edge in the graph has a weight of either 3, 7, or 9. Prove that TSP379 is NP-Complete.

a) Prove that TSP379 is in NP. (5 pts)

Certificate: A TS tour v1, …, vn (2 points)
Certifier: Check that all the nodes are visited once. Check that length of the tour is at most k. Both can be done in poly-time. (1 point for each of the three statements).

b) Prove that TSP379 is NP-hard (15 pts)

Reduction from HC:
Given G, an instance of HC, construct G' with

- Same set of nodes as G (1 point)

- Includes all edges in G with weight 3 (3 points)
- Make G' a fully connected graph by adding all the edges that are not in G with a weight of 7 or 9(3 points)

Set k = 3n (1 points)

Claim: G' will have a TS tour of at most 3n iff G has a HC (1 point)

Proof of reduction:

If G has a HC, since they have a weight of 3 in G' by construction, they form a TS tour of G' with weight at most 3n. (3 points)
If G' has a TS tour of at most 3n, all the edges in the tour must have a weight 3 (since there are n edges in that cycle and min possible weight is 3). By construction, these edges must be present in G, hence they form a HC of G. (3 points)

Additional Space

Additional Space