CS570 Spring 2019: Analysis of Algorithms Exam III

	Points		Points
Problem 1	20	Problem 5	16
Problem 2	16	Problem 6	16
Problem 3	16		
Problem 4	16		
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Closed book and notes
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.

1) 20 pts

Mark the following statements as **TRUE, FALSE, or UNKNOWN** (if unknown is given as a choice). No need to provide any justification.

[[✓ TRUE]/FALSE/UNKNOWN]
Let ODD denote the problem of deciding if a given integer is odd. Then ODD is polynomial time reducible to Vertex Cover.

[TRUE/[✓ FALSE]]
If we find a quadratic time solution to an NP-Complete problem, then all NP-Complete problems can be solved in quadratic time.

[[✓ TRUE]/FALSE/UNKNOWN]
Independent Set problem has a polynomial run time on certain types of graphs.

[[✓ TRUE]/FALSE]
Consider two decision problems Q1, Q2 such that Q1 reduces in polynomial time to Hamiltonian Cycle and Hamiltonian Cycle reduces in polynomial time to Q2. Then it is possible for both Q1 and Q2 to be in NP.

[[✓ TRUE]/FALSE]
The problem of determining whether there exists a cycle in an undirected graph is in NP.

[TRUE/[✓ FALSE]]
If the edge weights in a connected undirected graph G are NOT all distinct, then there will be more than one minimum spanning tree in G.

[TRUE/[✓ FALSE]]
Dijkstra's algorithm works correctly on graphs with negative-weight edges, as long as there are no negative-weight cycles.

[TRUE/FALSE/[✓ UNKNOWN]]
`If Integer Linear Programming` $\leq_p$ A, then A cannot be solved in polynomial time.

[[✓ TRUE]/FALSE]
Suppose that the capacities of the edges of an s-t flow network are integers that are all evenly divisible by 3. (E.g., 3, 6, 9, 12, etc.) Then there exists a maximum flow, such that the flow on each edge is also evenly divisible by 3.

[TRUE/[✓ FALSE]]
Suppose that G is a directed, acyclic graph and has a topological ordering with $v_1 v_1$ the first node in the ordering and $v_n v_n$ the last node in the ordering. Then there is a path in G from $v_1 v_1$ to $v_n v_n$.

Rubric: -2 for each incorrect answer

2) 16 pts
Recall the Interval Scheduling Problem. A set of n requests is given, each with a given start and finish time, $[s_i , f_i]$. The objective is to compute the maximum number of activities whose corresponding intervals do not overlap. In class we presented a greedy algorithm that solves this problem. We will consider some alternatives here.

(a) Earliest Activity First (EAF): Schedule the activity with the earliest start time. Remove all activities that overlap it. Repeat until no more activities remain. Give an example to show that EAF is not optimal. Your example should show not only that it is not optimal, but its approximation ratio can be arbitrarily high. (6 pts)
EAF can be arbitrarily bad. Consider an instance with $n - 1$ non-overlapping intervals and one interval that covers all of them. This covering interval starts first, and so it will be selected. The optimum algorithm would select the other $n - 1$ intervals, so the performance ratio is $(n - 1)/1$, which is unbounded as n tends to infinity.

Rubric:
Showing an example to show EAF is not optimal: 3 pts
Showing that the ratio can be as high as (n - 1)/1: 3 pts
Proving that it is worse that 0.5 approximation: -2 pts
If you analyze the time instead of the number of tasks to prove that the ratio can be as high as possible: -3pts

(b) Shortest Activity First (SAF): Schedule the activity with the smallest duration $(f_i - s_i)$. Remove all activities that overlap it. Repeat until no more activities remain. Give an example to show that SAF is not optimal. Show that it is a ½ approximation. (10 pts)

SAF is not optimal. Consider an instance consisting of three activities consisting of two nonoverlapping intervals and a shorter interval that overlaps both. It is possible to schedule the two non-overlapping intervals, but greedy will select the shorter overlapping interval. 1 We can prove that SAF has a performance ratio of at most 2. Here is a sketch.
Each time greedy selects an interval there are two possibilities:
(1) Opt also selects this interval, or
(2) Opt does not choose this interval. In case (2), Opt can select at most two intervals that overlap the interval chosen by greedy. This is because if there were three such intervals in Opt, the middle one would need to be even shorter than the greedy choice, contradicting the fact that greedy picks the smallest interval.
Since each greedy choice is balanced against at most two optimum choices, it follows that $|G| \geq |O|/2$.

Rubric:
Showing an example to show SAF is not optimal: 5 pts
Proving it is 0.5 approximation: 5 pts
Proving the 0.5 approximation for a worst case scenario without proving that it is the worst case: -3pts
Claiming that it is not optimal but not proving 0.5 approximation: -3 pts
Saying that the chosen task removes 2 other tasks, but not proving that it removes 2 optimal tasks that can be chosen together: -2 pts (because it can remove an arbitrarily high number of tasks, but it is important to say that the optimal solution can only choose 2 of those tasks)

3) 16 pts

There are n piles of dirt scattered over a large construction site. Pile number i weighs w_i tons and is at position p_i in the plane. The dirt in these piles has to be moved into m new piles at positions q_j (for $j = 1, \ldots, m$) with weights v_j (also in tons).
The good news:
You have a bulldozer.
The bad news:
You have to figure out the cheapest way to move the dirt.
The cost of moving t tons of dirt from position p to positions q is $t \times dist(p, q)$ where $dist(p, q)$ is the distance from p to q. Note that t need not be an integer. Your goal is to minimize the total cost of moving the dirt from its current locations to the new ones. The input consists of the location (X and Y coordinates) of points $p_1, \ldots, p_n, q_1, \ldots, q_m$ and weights $w_1, \ldots, w_n, v_1, \ldots, v_m$. You may ignore the cost of driving around the construction site when not carrying dirt; you only need to account for the cost of moving dirt from pile to pile. Also, because no nuclear reactions will be involved, you may assume that mass is conserved, that is, $\sum_{i=1}^{n} w_i = \sum_{j=1}^{m} v_j \sum_{i=1}^{n} w_i = \sum_{j=1}^{m} v_j$.
Give a linear program that minimizes the cost. No need to turn it into the standard form.

Variables:
For every pair i, j variable x_{ij} indicates the weight of the dirt moved from position p_i to q_j

Constraints:
For each pair i, j, we have $x_{ij} \geq 0$.
For each i, we have $\sum_{j=1}^{m} x_{ij} \sum_{j=1}^{m} x_{ij} = w_i$ (since we can use at most w_j tons from location p_i).
For each j, we have $\sum_{i=1}^{n} x_{ij} \sum_{i=1}^{n} x_{ij} = v_j$ (since we need v_j tons at location q_j).
Cost function to be minimized: $\sum_{ij} x_{ij} \sum_{ij} x_{ij} \times dist(p_i, q_j)$

Rubric:

2: Specifying what the decision variable signifies

4: Objective function

2: x_{ij}>=0

4+4 for the remaining two constraints.

The solution must be in LP form and be equivalent to the answer provided.

4) 16 pts

Consider an instance of the Satisfiability Problem, specified by clauses C1, . . . , Cm, over a set of Boolean variables x1, . . . , xn . We say that the instance is **monotone** if each term in each clause consists of non-negated variables; that is, each term is equal to x_i , for some i, rather than $\neg x_i$ (not x_i). For example, suppose we have the three clauses: ($x_1 \vee x_2$), ($x_1 \vee x_2 \vee x_4$), ($x_3 \vee x_4$) $(x_1 \vee x_2), (x_1 \vee x_3), (x_2 \vee x_3)$. This is monotone since we don't have any negated terms, and obviously the assignment that sets all three variables to 1 satisfies all the clauses. But we can observe that this is not the only satisfying truth assignment; we could also have set x_1 and x_4 to 1, and x_2 and x_3 to 0. Indeed, for any monotone instance, it is natural to ask how few variables we need to set to 1 in order to satisfy it.

Given a monotone instance of Satisfiability, together with a number k, the problem of Monotone Satisfiability with Few True Variables (MSFTV) asks: Provided m clauses, is there a satisfying assignment for the instance in which at most k variables are set to 1?

(1) Prove that the MSFTV problem is in NP. (4 pts)

(2)

Solution + Rubric:

Certificate: Assignment for variables (1 pt) which has poly-length/size (0.5 pt)

Certifier: Check for all clauses to be true (1 pt) and at most k variables to be true in the given assignment (1 pt). This take O(mn) steps, i.e. Poly-time (0.5 pt).

Hence this is in NP by definition.

Common mistake: Checking for non-negated variables in clauses etc. - which means checking if the input is a valid instance, which is not what certification is about.

(2) Prove that the MSFTV problem is NP-complete. (12 pts)

Solution + Rubric:

Construction (5 points)

--- From Vertex Cover: Starting with an instance of the VC problem, we formulate an MSFTV instance by representing each node v in the graph with a variable x_v and each edge (u, v) with a clause (x_u V x_v). (Some have attempted a more complicated way which is to consider the set of edges as a union over cliques (edges are a special case as 2-cliques), and construct a clause for each such clique.)

--- From Set Cover: Starting with an instance of the SC problem with a set of elements S and a collection of subsets, we formulate an MSFTV instance by representing each subset S_i with a variable x_i and each element j in S by a clause c_j as a boolean OR over all x_i's such that S_i contains j.

--- **From Dominating Set:** Starting with an instance of the DS problem, we formulate an MSFTV instance by representing each node v with a variable x_v and for each node u, construct a clause x_u with a clause as a boolean OR over all x_v's such that v is u or its neighbor. (Many have attempted this while calling the problem VC which it's not.)

Rest of the reduction (3 points)

Having constructed the clauses, consider a boolean AND (1 point) with the same k as the input. This expression is monotone (0.5) and thus a valid MSFTV instance. Then return YES IFF the MSFTV blackbox returns yes (1 point). This takes polynomially many steps and just 1 call to the blackbox, thus a valid reduction (0.5 points)

Proof of correctness (3 points: at most 2 points if only one direction)

iIf the blackbox returns YES, there's an assignment with k true vars, such that each clause must be true. Hence, choosing a vertex to be in the cover whenever it's set to 1 in MSFTV, covers the corresponding edge by construction and yields a k-sized VC. (Analogous argument for Set cover/ Dominating set constructions).

Similar argument for the other direction, but needs to be explicit.

Conclusion (1 point)

Thus, VC (or SC/DS) problem $\leq_p$MSFTV. Since VC NP-complete, MSFTV is NP-hard. Since it's also in NP by part 1), it is NP-complete.

Common mistakes:

Unsuccessful attempts to reduce from other problems (e.g. Ind set/ SAT) could get up to 30% partial credit (0.5 - 3.5 points) if they demonstrate understanding of how the reduction works to some degree.

Constructing the other way, e.g., "Construct a graph G with an edge for every clause" etc. will lose at least half the construction points, and more if it is carried throughout.

5) 16 pts

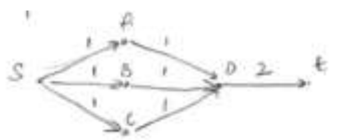
Given a flow network (G, c, s, t), John wants to solve the Max-cut problem, i.e. find an s-t cut with maximum capacity. He proposes to do this by reducing it to the Min-cut problem. So, he first finds the edge with maximum capacity w. Then he considers a transformed network (G', c', s, t) where, $c'(e) = w - c(e)$ for every edge e. Thus, this new network has all non-negative capacities, and the edges having small capacities in the new network, must originally have large capacities, and vice versa. John claims that a min-cut in the new network corresponds to a max-cut in the original network. Prove or disprove whether this is a valid reduction.

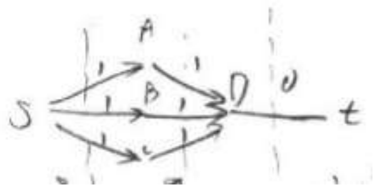
Standard answer:

The method is wrong.

Disprove by using a flawless counter-example:

G G'





Min cut is {S,A,B,C,D},{T} in G'; But the cut {S,A,B,C,D},{T} in G is not an max-cut, since {S},{A,B,C,D,T} has larger capacity than this one. Therefore disprove the method.

According to the discussion agreed upon the instruction team:

a) 0 marks to those who try to prove the statement to be True.
   +6 points for claiming that the reduction is wrong
b) For people get 6 point in a):
   +0 points for no counter-example or example extremely wrong
   + 4 , +8, depends on the level of effort (+4 for obvious mistakes or multiple small mistakes, + 8 for small mistakes such as no arrows in a flow chart)
   +10 points for a valid counter-example with no errors.

Therefore, the student can only get points of 5 levels:

0, 6, 10, 14, 16

Common mistakes including:

1.

If you use proof to disprove, you still have to give an example for any statement such as "XX may(can) not be XXX", "There might be another", "Cannot guarantee". This is the paradigm of proof, not negotiable.

2.

The result of ford-fulkerson cannot disprove min cut in G' doesn't necessarily equal to max-cut in G

3.

Didn't show which is the min cut and which is the max cut clearly. Mentioning the capacity value of these two cuts don't mean nothing.

4.
No arrows to indicate directions of flow edges
5.
Did’t correctly set w as the largest capacity for a single edge
6.
Wrong min cut in G’ or max cut in G
7.
Some cut is not a cut, in a cut, one sides can reach s, nodes on another side can reach t
8.
w is the largest capacity of an edge in G, not a random value.
9.
Try to improve the method but don’t explain why it’s invalid by using a counter example.
10.
Min cut in G’ can be zero, it’s not a correct reason to disprove the method.

6) 16 pts

Consider a weighted complete bipartite graph G, with each partition having *n* vertices in it. (In a complete bipartite graph, there is an edge between each node in one partition to all nodes within the other partition, and vice versa). Let M be a maximum matching in G, i.e. a matching containing n disjoint edges.
We want to find a spanning tree in G that contains the given matching M (i.e. all the edges in M) and has the minimum weight among all the spanning trees that contain M. Find an efficient algorithm for this and analyze its complexity.

Solution:

1. Start off by adding edges in M to T (spanning tree) (7 points)

2. For the remaining edges, sort them in increasing weights and apply Kruskal's. (7 points)

3. Complexity is the same of Kruskal O(ElogE) (2 points)

Rubric:

1. If Prim's is used, you get 9 points, provided you got the runtime of Prims correctly. If not, you get 7 points. Prim's will not work, as Prim starts of with any vertex and covers the minimum edge. Prim does not guarantee that all edges in M will be included.
2. If you describe Kruskal's algorithm then you must describe the cycle property. You get a -2 if the cycle property is not mentioned.
3. If you have modified the initial graph to make sure that the weight of edges in M is the smallest, the solution is OK. However, the wrong weight allocation will give you -4 points. Wrong weight allocations include (not exhaustive!)
   a. Allocating weight of edges in M as 0. There could be other edges in E-M with lower weights
   b. Allocating weight of edges in M as 1. Same reasoning
   c. Allocating weight of edges in M to be equal to the lowest weight in E-M. It may occur that the other edge will be included in the spanning tree instead of an edge from M