CS570 Fall 2018: Analysis of Algorithms Exam III

	Points		Points
Problem 1	20	Problem 5	10
Problem 2	8	Problem 6	16
Problem 3	14	Problem 7	12
Problem 4	20		
	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** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
Let X be a decision problem. If we prove that X is in the class NP and give a poly-time reduction from X to 3-SAT, we can conclude that X is NP-complete.

[TRUE/FALSE]
Let A be an algorithm that operates on a list of n objects, where n is a power of two. A spends $\Theta(n^2)$ time dividing its input list into two equal pieces and selecting one of the two pieces. It then calls itself recursively on that list of n/2 elements. Then A's running time on a list of n elements is O(n).

[TRUE/FALSE]
If there is a polynomial time algorithm to solve problem A then A is in NP.

[TRUE/FALSE]
A pseudo-polynomial time algorithm is always slower than a polynomial time algorithm.

[TRUE/FALSE]
In a dynamic programming formulation, the sub-problems must be non-overlapping.

[TRUE/FALSE]
A spanning tree of a given undirected, connected graph G = (V, E) can be found in O(E) time.

[TRUE/FALSE]
Ford-Fulkerson can return a zero maximum flow for flow networks with non-zero capacities.

[TRUE/FALSE]
If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, then $h(n) = \Theta(f(n))$

[TRUE/FALSE]
There is a polynomial-time solution for the 0/1 Knapsack problem if all items have the same weight but different values.

[TRUE/FALSE]
If there are negative cost edges in a graph but no negative cost cycles, Dijkstra's algorithm still runs correctly.

2) 8 pts

Let G = (V, E) be a simple graph with n vertices. Suppose the weight of every edge of G is one.
Note: In a simple graph there is at most one edge directly connecting any two nodes. For example, between nodes u and v there will be at most one edge uv.
(a) What is the weight of a minimum spanning tree of G? (1 pt)

(b) Suppose we change the weight of two edges of G to 1/2. What is the weight of the minimum spanning tree of G? (2 pts)
(c) Suppose we change the weight of three edges of G to 1/2. What is the minimum and maximum possible weights for the the minimum spanning tree of G? (2 pts)
(d) Suppose we change the weight of k < V edges of G to 1/2. What is the minimum and maximum possible weights for the the minimum spanning tree of G? (3 pts)

1. Any spanning tree of G has exactly V − 1 edges, since all the weights are one its total weight is V − 1.
2. To answer this question, consider running Kruskal on G, it will first pick the two lighter edges as they cannot form a cycle (the graph is simple), then V − 3 other edge each with weight one. Therefore, the total value is V − 3 + 2(1/2) = V − 2.
3. Use the similar approach as (b). Kruskal first selects two light edges. There are two possibilities for the third edge: (i) it forms a cycle with the first two edges or (ii) it does not form a cycle with the first two edges not. In case (i), Kruskal selects two edges of weight 1/2 and V − 3 edges of weight 1, therefore, the total weight is V − 3 + 2(1/2) = V − 2. In case (ii), Kruskal selects three edges of weight 1/2 and V − 4 edges of weight 1, therefore, the total weight is V − 4 + 3(1/2) = V − 5/2.
4. Similar to (c), the minimum weight happens when the k light edges do not form any cycle. In this case, MST contains k edges of weight 1/2 and n − 1 − k edges of weight 1. Therefore, its weight is k/2 + n − 1 − k = n − k/2 − 1. The maximum weight happens if the k edges span as few vertices as possible. This happens when the k edges are part of an almost *complete graph*. Let h be a variable denoting the number of nodes in this subgraph such that the number of edges in a complete graph comprised of these edges nodes exceeds k. In particular, define h to be the smallest number such that, the binomial coefficient Binomial[h,2] > k, ie., (h choose 2) > k. Accordingly, we can pack all k light edges between h vertices. Consequently, the MST has h− 1 edges of weight 1/2 and n − 1 − (h − 1) edges of weight 1. Therefore, its weight is n − h/2 − 1/2.

Rubric:
For each part (a,b,c,d), No partial marking.
This also applies to the parts where minimum and maximum is asked.
If either is wrong then there is no partial marks for that part.

Q3. 14 pts

Recall that in the discussion class we showed that the Bipartite **Undirected** Hamiltonian Cycle problem is NP-complete. Now in the Bipartite **Directed** Hamiltonian Cycle problem, we are given a bipartite directed graph and asked whether there is a simple cycle which visits every node exactly once. Note that this problem might potentially be easier than Hamiltonian Cycle because it assumes a directed bipartite graph. Prove that Bipartite Directed Hamiltonian Cycle is in fact still NP-Complete.

Solution:

i) This problem is NP. Given a candidate path, we just check the nodes along the given path one by one to see if every node in the network is visited exactly once and if each consecutive node pair along the path are joined by an edge. (3)

ii) To prove the problem is NP-complete, we reduce Directed Hamiltonian Cycle (DHC) problem to Bipartite Directed Hamiltonian Cycle (BDHC) problem. (2)

Given an arbitrary directed graph G, we split each vertex v in G into two vertex v_in and v_out. Here v_in connects all the incoming edges to v in G; v_out connects all the outgoing edges from v in G. Moreover, we connect one directed edge from v_in to v_out. After doing these operations for each node in G, we form a new graph G'. (3)

Here G' is bipartite graph, because we can color each v_in "blue" and each v_out "red" without any coloring conflict.

If there is DHC in G, then there is a BDHC in G'. We can replace each node v on the DHC in G into consecutive nodes v_in and v_out, and (v_in, v_out) is an edge in G'. Then the new path is a BDHC in G'. (3)

On the other hand, if there is BDHC in G', then there is a DHC in G. Note that if v_in is on the BDHC, v_out must be the successive node on the BDHC. Then we can merge each node pair (v_in, v_out) on the BDHC in G' into node v and form a DHC in G. (3)

In sum, G has DHC if and only if G' has a BDHC. This completes the reduction, and we confirm that the given problem is NP-complete

Alternate solution:
We can also use Undirected Hamiltonian Cycle for the reduction.
Prove that it's NP as before. (3)
Note that you are using Bipartite Undirected Hamiltonian Cycle for reduction. (2)
Given an arbitrary undirected bipartite graph G, convert G to G' so the vertices in G' stay the same, but all undirected edges are converted to directed edges in **both directions**. G' remains bipartite. (3)

If there is an undirected Hamiltonian cycle in G, you can use the corresponding edges(in either direction) to find a Directed Hamiltonian cycle in in G'. (3)
On the other hand, if there is a directed Hamiltonian cycle in G', you could convert the corresponding edges to undirected edges and find the equivalent cycle in G. (3)

Common mistakes:

Major mistake in checking for NP: -2
Checking for edges instead of vertices: -1
Missing the NP check altogether: -3

Mistake in the conversion step: -2

Missing the two-way proof: -4
Missing either side of the proof: -3
Incomplete explanation as to why the proof holds: -2

If you are proposing to reduce a certain NP-Complete problem, but are going with a solution corresponding a different NP-Complete problem, this shows misunderstanding of the concept: -7

If you are proposing to reduce an incorrect (and irrelevant) NP-complete problem: -8

4) 20 pts.
Suppose you want to sell n roses. You need to group the roses into multiple bouquets with different sizes. The value of each bouquet depends on the number of roses you used. In other words, we know that a bouquet of size i will sell for v[i] dollars. Provide an algorithm to find the maximum possible value of the roses by deciding how to group them into different-sized bouquets.
Here is an example:
Input: n = 5, v[1..n] = [2,3,7,8,10] (i.e. a bouquet of size 1 is $2, bouquet of size 2 is $3, … , and finally a single bouquet of size n=5 is $10.
Expected Output = 11 (by grouping the roses into bouquets of size 1, 1, and 3)

a) Define (in plain English) subproblems to be solved. (4 pts)
OPT(j) is the maximum value of j roses.

b) Write the recurrence relation for subproblems. (6 pts)
OPT(j) = MAX (OPT(j), {OPT(j-i) + v[i]} for all sizes 0 <= i <= j)

i.If iteration of i from 0 to j is missing => -1 points
ii.If OPT(j) is missing in the MAX term => -1 points
iii.0 points for wrong recursion

c) Using the recurrence formula in part b, write pseudocode to compute the maximum possible value of the n roses. (6 pts)
Make sure you have initial values properly assigned. (2 pts)

```
OPT[0] = 0
OPT[1] = v[1]
For (int j=2; j < n; j ++ )
     max = 0
          For (int i=1; i < j; i++)
               If (i<=j && max < OPT[j-i]+v[i] )
                    max = OPT[j-i]+v[i]
     OPT[j]=max
```

i) Missing one loop => -1 point
ii) Missing two loops => -2 points
iii) If in b) you got x points, you get x points in the pseudo code. Apart from the case where you missed the iteration in b, where you will get x+1 points in this question
iv) 0 points for no pseudocode or any modification from b.

d) Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not (2 pts)
e) d) O(n^2)

5) 10 pts
Give a tight asymptotic upper bound (O notation) on the solution to each of the following recurrences. You need not justify your answers.

$T(n) = 2T(n/8) + n$

Solution: $(n^{1/3} \lg n)$ by Case 2 of the Master Method.

$T(n) = T(n/3) + T(n/4) + 5n$

Solution: Use brute force method
$T(n) = cn + (7/12)\, cn + (7/12)^2\, cn + \ldots$
$= (n)$

Rubric:
No partial marks. 5 points for each correct answer.

6) 16 pts

There are *n* people and *m* jobs. You are given a payoff matrix, C, where C_{ij} represents the payoff for assigning person i to do job j. Each applicant has a subset of jobs that he/she is interested in. Each job opening can only accept one applicant and a job applicant can be appointed for only one job. You need to find an assignment of jobs to applicants that maximizes the total payoff of your assignment. Write an **Integer Linear Program** (with discrete variables) to solve this problem.

Solution

We are looking for a perfect matching on a bipartite graph of the minimal cost.

Let $x_{ij} = \begin{cases} 1, & if\ edge(i,j)\ is\ in\ matching \\ 0, & otherwise \end{cases}$

Objective: $\sum_{\substack{1<i<m \\ 1<j<n}} c_{ij}.x_{ij}$

Subject to:

1. $\sum_j x_{ij} \leq 1 \quad , where\ i = 1,2,\dots,n$
2. $\sum_i x_{ij} \leq 1 \quad , where\ j = 1,2,\dots,n$
3. $x_{ij} \in \{0,1\} \quad , \forall\ i,j$
4. $x_{ij} = 0$, for all jobs j that applicant i is not interested in.

Rubric:

-4 if objective function is incorrect.

-6 if you mention equal to instead of less than equal to sign for constraints 1 and 2

-2 if constraint 3. is not mentioned, but only is defined to take values 0 or 1.

-2 if discrete variable is not defined.

0 points awarded if you use a max-flow, min-cut formulation

7) 12 pts

The Maximum Acyclic Subgraph is stated as follows: Given a directed graph find an acyclic subgraph of that contains as many arcs (i.e. directed edges) as possible.

Give a 2-approximation algorithm for this problem.

Hint: Consider using an arbitrary ordering of the vertices in G.

Solution:

1. Pick an arbitrary vertex ordering This partitions the arcs into two acyclic subgraphs and . Here A1 is the set of arcs where and is the set of arcs where . Thus at least one of or contains half the arcs of G. The maximum acyclic subgraph contains at most the total number of arcs in G, so we have a factor 2-approximation algorithm (12)
2. Divide the set of nodes into two nonempty sets, A and B. Consider the set of edges from A to B and the set of edges from B to A. Throw out the edges from the smaller set and keep the edges from the larger set (break ties arbitrarily). Recursively perfomr the algorithm on A and B individually (12)

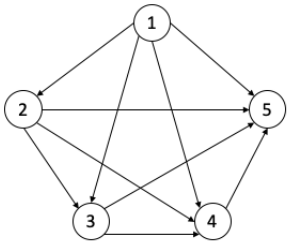
Rubric:

If you are only considering the forward or backward edges (-2)
If you start with one node and add the forward/backward edges to/from neighbors, your answer won't work for disconnected graphs (-2)
If your final answer is not 0.5 approximation (-6)
If your answer is not acyclic (-6)
If your algorithm is vague and not implementable (for example if you said, "we find max independent set", or "we find the topological order") (0)
If your algorithm has conceptual flaws (for example if you are adding an edge to the sub graph and also deleting it from the sub graph) (0 pts)

Some of the answers that do not work:

1. If your final answer is a tree (a DFS or a BFS or any other kinds of trees):

   You can imagine an acyclic complete directed graph such as below. Your algorithm should return at least half of the edges (n * (n-1) / 4) but a tree can have n-1 edges. Thus it is not 0.5 approximation (6 pts).



2. If you start adding/removing edges to/from the graph, until it is acyclic. Your algorithm can choose an edge that is repeated in many cycles and end up removing all the other edges in those cycles.
   You can try your method on the following graph (All l(i) nodes are connected to all r(j) nodes). If your algorithm chooses all the (r(j), X) and (X, l(i)) edges (7 edges), you cannot add any (l(i), r(j)) edges (12 edges). Your final answer (7) is less than half of the best answer (16/2 = 8) so it is not 0.5 approximation.