Summer 2020 - Exam 2

Problem 1 (30 pts)

T/F NP-completeness and LP (randomly select 5)

[TRUE/FALSE]
Assuming that we have found a correct optimal solution to an LP problem, if someone claims to have found a different optimal solution, we know that that solution must be incorrect.

[TRUE/FALSE]
Suppose that we find a polynomial-time solution to a problem in NP. This means that we can also solve subset-sum on polynomial time.

[TRUE/FALSE]
Every problem with a polynomial-time solution is polynomial-time reducible to Hamiltonian Cycle.

[TRUE/FALSE]
If the optimization version of the Traveling Salesman Problem can be solved in polynomial time, then the decision version of Independent Set will be solvable in polynomial time.

[TRUE/FALSE]
Say problem Y is reducible to Hamiltonian Cycle, and Hamiltonian Cycle is reducible to problem X, then at least one of the two problems (X or Y) is in NP-hard.

[TRUE/FALSE]
Suppose we have already solved an LP problem. Now we add two more constraints to the constraint set. This will cause the feasible region to become smaller.

[TRUE/FALSE]
The two objective functions (Maximize 5X + 7Y, and Minimize 5X - 7Y) will produce the same solution to a linear program.

[TRUE/FALSE]
If A is polynomial-time reducible to B and C is polynomial-time reducible to B, then A is polynomial-time reducible to C.

[TRUE/FALSE]
In order to have efficient certification, one must have both a polynomial length certificate and a polynomial-time certifier.

T/F Network flow (randomly select 5)

[TRUE/FALSE]
In a flow network with a total capacity of edges going out of source = C, if we round up all edge capacities to the closest factor of 10, then the maximum number of iterations required for the Ford-Fulkerson algorithm to find max flow in the new network will be at most C/10.

[TRUE/FALSE]
At the moment when the Ford-Fulkerson algorithm finds max flow f, it is possible to find a path from T to S in the residual graph Gf.

[TRUE/FALSE]
In a unit capacity network, i.e., all edges have a capacity of 1, all min cuts must have the same number of edges that cross the min-cut.

[TRUE/FALSE]
Let (u,v) be an edge that crosses the min-cut closest to S and the min-cut closest to T in flow network G. Then G must have a unique min-cut.

[TRUE/FALSE]
In a flow network, if the min-cut is not unique, then decreasing the capacity of an edge that belongs to a min-cut may not result in decreasing the value of the maximum flow.

[TRUE/FALSE]
In a flow network G, if we increase the capacity of one edge by a positive amount x and we observe that the value of max flow also increases by x, then that edge must belong to every min-cut in G.

[TRUE/FALSE]
In a circulation network with lower bounds l_e on each edge e, as long as we have edge capacities that are at least equal to Max (l_e) for all edges, and if total demand = total supply, we are guaranteed to find a feasible circulation.

T/F Other topics (randomly select 5)

[TRUE/FALSE]
Finding the minimum element in a binary max heap of n elements takes O(logn) time.

[TRUE/FALSE]
Suppose we must solve a special version of the interval scheduling problem where all intervals are of size 2. Then the earliest start time approach will provide an optimal solution to the problem.

[TRUE/FALSE]
Recall the closest pair of points in 2D problem that we solved in class. Since we were able to reduce the cost of the merge operation to O(n), then the only reason why we ended up with O(nlogn) complexity was because of the initial sorting of the arrays by X and Y coordinates. Otherwise, the divide and conquer part of the solution runs in O(n) time.

[TRUE/FALSE]
The following recurrence equation T(n) = 3T(n/3) + 0.1n has the solution: T(n) = $\Theta(\log(n^{2n}))$.

[TRUE/FALSE]
Runtime complexity of a dynamic programming algorithm is equal to the number of unique sub-problems.

[TRUE/FALSE]
Amortized analysis is used to determine the worst-case runtime complexity of a sequence of operations.

[TRUE/FALSE]
$7^{\log(n)} = O(n^3)$ **Note:** assume log base 2 by convention.

Problem 2 (20 pts)

Regina decided to pursue her bachelor's degree in CS at USC. She knows that as a first-year student she has to share an apartment with someone else. Fortunately, the apartment that she will be staying in for the Fall 2020 comes fully furnished. But, unfortunately, this set of n furniture items are not meant to be shared by two people, so she has to split the furniture items between herself and her roommate. Since she arrives on campus earlier than her roommate, she gets to decide which furniture items she should keep and which to give to her roommate. She then assigns each furniture item i (1<=i<=n) a usefulness value Ui. To be fair to her roommate, she wants to know if she can split the set of n furniture items into two disjoint sets of furniture items A and B with the same exact total usefulness value. Let's call this problem a Fair Roommate Furniture Sharing (FRFS) problem.

Now, Suppose that FRFS is the only NP-complete problem you know of. We want to prove that Subset Sum is NP-complete.

Note: The decision version of Subset Sum asks whether there is a subset of items with a total weight within the range M and W.

a). Show that Subset Sum is in NP. (5pts)

Solution:

Subset Sum is in NP

1. Certificate - A subset of the input set.
2. Certifier - add up the weight of all items in the subset and make sure it's between M and W. Linear time certifier.

3 pts if either one is correct.
5 pts if both are correct.
-1 pt if not mentioning M and W or incorrect total weight.

b). Show that Subset Sum is NP-complete. (10pts)

Solution:

FRFS <=p Subset Sum

Reduction:

Input to RFRS: a single set of furniture items with usefulness value of item i being U_i

Input to Subset Sum: a single set of items with weights w_1,...,w_n, M and W

Transform input to FRFS into input to Subset Sum:

The input set S to Subset Sum will contain n items with weights w_1,.....,w_n, where n is the number of furniture items in the input to FRFS.
Set for each i to n, w_i = U_i
Set M = W = (Sum(U_i) for i from 1 to n)/2

3 pts if show FRFS <=p Subset Sum.
2 pts if set w_i to U_i.
5 pts if set M and W correctly.
0 pt if use other NPC problems.

c). Prove that your reduction is correct. (5pts)

Solution:

Proof of correctness for the reduction step:

A. If there is a subset of values in Subset Sum that adds to exactly half of the sum of the usefulness value in FRFS, then the remaining values will add to the exact same thing → proving the existence of a valid FRFS solution. If no such subset can be found, then no such FRFS solution can be formed.

B. If there is a solution for FRFS, then there is a solution for subset sum with M, W = half of the sum of the usefulness value in FRFS.

3 pts if either one is correct.
5 pts if both are correct.
-1 pt if not mentioning M, W = half of the total sum.

Problem 3 (20 pts)

The USC Marching Band is scheduled to perform during a football game in San Francisco (SF) next week. The band needs to ship m musical instruments through several means of transport (e.g., by plane, train, bus) to SF. There are four units of each type of musical instrument, and there are five means of transport that the team can use. Suppose that the five means of transport can take a maximum of 4, 4, 5, 8, and 8 units of musical instruments, respectively. Using Max Flow, we want to determine if there is a way that the musical instruments can be shipped to SF with no two units of the same instrument in the same means of transport.

a). Describe the construction of a flow network to solve this problem. Describe what each node and edge will represent and what the edge capacities are. (15pts)

- Source node S to draw flow from (1 pts)
- Sink node T to absorb flow (1 pts)
- A set of m/4 nodes X to represent musical instrument types (2 pts)
- A set of 5 nodes Y to represent the means of transport (2 pts)
- Edges with a capacity of 4 from S to each node in X, limiting the number of instrument used from each type to 4 (2 pts)
- Edges with capacity of 1 from each node in X to each node in Y, limiting the use of each means of transport for a given instrument type to 1 (so no two instruments of the same type can get on the same means of transport) (2 pts)
- Edges with a capacity of 4, 4, 5, 8, and 8 units from the 5 nodes representing the means of transport to T, limiting the capacity of each means of transport (2 pts)

Run max flow. If the value of max flow comes out to be m then we have a feasible transportation solution. (3 pts)

(-2 pts) If you use circulation instead of maxflow

b). Prove that your solution is correct. (5pts)

Part a) If we have a flow f of value m in the network, we can find a feasible transportation solution.

We can identify the edges going from X to Y that are carrying flow. These edges identify the assignment of musical instruments to means of transportation. All constraints required for transportation are met since

- the flow is of value m, so all instruments are assigned to means of transportation
- edges from X to Y have a capacity of 1 so no more than one instrument type is assigned to a means of transport
- edges from Y to T have capacities of 4, 4, 5, 8, and 8 units. So, no capacities on means of transport have been violated.

(2 pts) missing anything in the proof: -1 pts

Part b) If we have a feasible transportation solution, we can find a (max) flow of value m in the network. Here is how:

- For each instrument assigned to a means of transport we will add a flow of 1 between the corresponding nodes in X and Y
- Since there are 4 instruments of each type we have 4 units of flow going out of each node in the set X, so we now bring 4 units of flow into those nodes from S. This saturates all edges going out of S
- Since the transportation constraints are met then the flows into nodes in the set Y are no more than 4,4,5,8, and 8. There is adequate capacity on edges from Y to T to carry this flow into T, and for each of these edges we will add a flow of value equal to the number of instruments being carried by that means of transport.
- This flow is valid since it meets all capacity and conservation of flow conditions
- It is a max flow since all edges out of S are saturated

(3 pts) missing anything in the proof: -1 pts

For partial proof you only get 2 or 3 pts from 5 pts

Problem 4 (10pts)

We are given a set of n cities on a map and a set of roads connecting them. We are also given the following information:

- All distances of road segments connecting cities. Note: not every two cities are connected directly by a road segment.
- (longitude, latitude) position of each city on the map.
- No two cities share the same longitude.

We would like to find a shortest path that starts with the westernmost city W, goes through the city C, and then reaches the easternmost city E. We also want this path to always go east, i.e., the longitude of the cities visited on the path will always increase.

a). Design an $O(n^2)$ time algorithm to compute the length of the shortest path given n cities. Be sure to argue why your algorithm takes $O(n^2)$. (10pts)

Idea:

Find the shortest path from W to C and shortest path from C to E. Since this is a DAG the shortest path can be found in $O(n^2)$ using Dynamic Programming for both segments.

Construction:

We have a connected graph with cities as nodes and the roads connecting them as edges. Here, the road distances are the edge weights. We will change this graph into a directed graph by creating the edge direction from city with lower longitude to city with higher longitude. Furthermore, this is a Directed Acyclic Graph (DAG) as no cities have the same longitude. We'll now use topological sorting and dynamic programming to compute the shortest path between city W and C and between city C and E. We use the following algorithm for both:

Algorithm:

1.Initialize distance between all nodes as INFINITY, i.e. dist[] = INF and ZERO for the the source city, i.e. dist[s] = 0
2. Create a topological order of all vertex
3. For every vertex u in the topological order:

For every adjacent vertex v of u:
if (dist[v]>dist[u]+weight(u,v)):
dist[v] = dist[u] + weight(u,v)

Time Complexity:

Time complexity of topological sorting O(V+E). After finding topological order, the algorithm processes all vertices and for every vertex, it runs a loop for all adjacent vertices. Total adjacent vertices in a graph is O(E). So the inner loop runs O(V+E) time. Therefore, overall time complexity of this algorithm is O(V+E), i.e. O(E). For dense graphs E is of the order n^2, hence time complexity : $O(n^2)$.

Construction of DAG – 2 pts
Algorithm – 4 pts
Recurrence Equation – 1 pt
Correct Time complexity – 1.5 pts
Correct Time Explanation – 1.5 pts

Alternate Solution:

Construction + Algorithm:

We have a connected graph with cities as nodes and the roads connecting them as edges. Here, the road distances are the edge weights. We will change this graph into a directed graph by creating the edge direction from city with lower longitude to city with higher longitude. Furthermore, this is a Directed Acyclic Graph (DAG) as no cities have the same longitude.
On this directed graph, we run Dijkstra's algorithm **using Fibonacci heap** from W to get the shortest path from W to C. Then we run Dijkstra's **using Fibonacci heap** from node C to get the shortest path from C to E.

Time Complexity:

n extract min operations using Fibonacci heap = O(nlogn)
m decrease key operations using Fibonacci heap = O(m)
For dense graph $O(m) = O(n^2)$
Time complexity of Dijkstra using fibonacci heap = $O(n\log n + n^2) = O(n^2)$

Algorithm – 6 pts
Fibonacci heap – 1 pt
Correct Time complexity – 1.5 pts
Correct Time Explanation – 1.5 pts

Problem 5 (10pts)

Jacques, a CS570 student, knows three ways to study for Exam 2. He estimates that each hour reading the textbook will get him 1.5 points on the exam, each hour of doing practice problems will get him 2 points on the exam, and each hour of watching past lecture videos will get him 3.5 points on the exam. He wants to spend at most 2.5 hours reading the textbook and is required to spend at least 3 hours watching lecture videos. He knows that he can pass the course if he can get a total of at least 50 points. Assume that the exam is out of 100.

How should he split his time in order to pass the course while spending the least amount of time studying? Give a linear programming solution to this problem. **Note:** you are not required to solve it numerically.

a). Describe the variables. (2pts)

3 variables: one for hours spent doing practice problems, one for hours spent reading the textbook, and one for hours spent watching lectures.

For example,

1. R = hours spent reading the textbook.
2. P = hours spent doing practice problems.
3. L = hours spent watching lecture videos.

-1 if missing one variable or more
-1 if you did not describe the variables.
-0.5 if you didn't specify a unit of time (in hours) or mention time.
-0.5 if you didn't really "describe" variables.

b). Give an objective function. (4pts)

Minimize the sum of hours spent doing practice problems, hours spent reading, and hours spent watching lectures.

Hence, this can be denoted as *minimize (P+R+L)*

-2 if didn't mention minimize.
-2 if not using the variables you specified in part a).
-2 if incorrect objective function. You use one of the constraints instead (e.g., min(1.5R+2P+3.5L) or 50<=1.5R+2P+3.5L<=100, max(1.5R+2P+3.5L), etc.)

c). Give the constraints. (4pts)

Constraints:

- 0<=R<=2.5
- 0<=P
- 3<=L
- 50 <= 1.5R + 2P + 3.5L <= 100

+1 for 0<=R<=2.5
    -0.5 if missing 0<= (lower bound)
+1 for 0<=P
+1 for 3<=L
    -0.5 if you have another constraint that says 0<=L
+1 for 50 <= 1.5R + 2P + 3.5L <= 100
    - 0.5 if missing 50 <=
    -0.5 if missing <=100
-0.25 for any use of incorrect sign, sign flip, etc. (e.g., =, <, >) for each constraint

Problem 6 (10 pts)

Use the Master Theorem to solve each of the following recurrences by giving tight Θ-notation bounds, or describe why the Master Theorem does not apply to that particular recurrence. If the Master Theorem is applicable for the particular recurrence, be sure to state **which case** and **show all your work**.

a). $T(n) = 2^nT(n/2) + n^n$

2^n is not constant, so master theorem does not apply.

b). $T(n) = 16T(n/4) + n!$

$\Theta(n!)$

c). $T(n) = 4T(n/2) + cn$

$\Theta(n^2)$

d). $T(n) = 2T(n/2) + n\log n$

$\Theta(n \log^2 n)$

e). $T(n) = 64T(n/8) - n^2\log n$

$-n^2\log n$ is not an asymptotically positive function, so the master theorem does not apply.

2 points for each item. No partial credit will be given.
For cases where Master Theorem applies, credit is given if the Θ-notation bound is correct. For other cases, it should be mentioned why Master Theorem is not applicable.