CS570 Summer 2019: Analysis of Algorithms Exam I

	Points		Points
Problem 1	20	Problem 5	15
Problem 2	15	Problem 6	10
Problem 3	15	Problem 7	10
Problem 4	15		
	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]
In a flow network, a flow *f* is a max flow if and only if there exists no cut with the same capacity as *v(f)*

[TRUE/FALSE]
If we replace each directed edge in a flow network with two directed edges in opposite directions with the same capacity and connecting the same vertices, then the value of the maximum flow remains unchanged.

[TRUE/FALSE]
It is possible for a circulation network to not have a feasible circulation even if all edges have unlimited capacity.

[TRUE/FALSE]
If subset sum can be solved in polynomial time, then 3SAT can be solved in polynomial time.

[TRUE/FALSE]
If problem A is NP-complete and $A \leq_p B$, then problem B is also NP-complete.

[TRUE/FALSE]
Some problems in the set P do not have efficient certifiers.

[TRUE/FALSE]
If an NP-hard problem can be solved in polynomial time then P=NP

[TRUE/FALSE]
A dynamic programming algorithm always uses some type of recurrence relation

[TRUE/FALSE]
The main difference between divide and conquer and dynamic programming is that divide and conquer solves problems in a top-down manner whereas dynamic-programming does this bottom-up.

[TRUE/FALSE]
A dynamic programming solution will always have a polynomial running time.

2) 15 pts

Let G= (V, E) be an undirected, bipartite graph.

a- Prove that the following algorithm is a ½ approximation to the maximum matching in G. (10 pts)

Start with M=Null (an empty set)
While there are more edges in G
    Select a random edge *e* in G and add it to M
    Remove *e* and all edges that share a node with *e* from G
Endwhile
Return M

Solution

A matching M in G is a set of pairwise non-adjacent edges, none of which are loops, that is, no two edges share a common vertex.

A maximum matching is a matching which has the highest number of edges possible from a graph G and is still a matching. It is simply the maximum cardinality matching possible in a graph.

We call the matching obtained by our algorithm M*

Every edge in the maximum matching must share a vertex with at least one edge in the matching found by this algorithm (since we exhaust all the edges, if an edge could be included in the set without violating the property of matching, it would have been included already). So each edge in M* can share vertices with at most two edges in the maximum matching.

Every edge in M* will have two vertices. The extreme case would be if each vertex was represented by two different edges in the maximum matching. Thus at worse, every edge in M* can account for 2 edges in the maximum matching. Thus, |M*|/|Maximum matching| = ½.

The algorithm of course terminates because of the condition that we go on till there are more edges present in the G to remove and we keep removing edge from G after selecting and adding random edges into the solution M*. So it terminates.

Rubric:

10- if the argument is presented correctly in generalized form for all possible graphs as explained in the solution provided.

6- If a correct example is used to explain and captures the main concept of why this is a half approximation.

Common mistakes:
These are **some wrong assumptions** made in multiple solutions:

- Removal of edges in G goes on only when the number of edges in G is more than the number of edges in M*
- Removal of edges in G goes on only when the number of edges in G is more than the ones removed from it previously.
- The graph being bipartite will lead to the half approximation factor.
- Matching consists of vertices, not edges.
- Edges are same as matchings.
- Maximum matching includes all the edges in a graph.

b- Show an example of a graph with no more than 6 edges where this algorithm could produce a matching that is exactly half the size of its maximum matching. (5 pts)

Solution: M*={B,C} and maximum matching={A,C},{B,D}

Any correct example showing the M* and the maximum matching such that M* is EXACTLY half the size of the maximum matching gets full credits. Also, maximum matching does not mean all edges are included in it.

Rubric:
5- A correct graph with M* and maximum matching found and written down to show the exact half approximation factor between them.
3- If only a correct graph is drawn

3) 15 pts

For each of the following problems, sketch an efficient algorithm or give an NP-completeness proof. You may draw on results from class or the textbook. (Note: In this problem, path length is the number of edges).

a- Longest path in a directed acyclic graph: Given vertices s and t, find a maximum length path from s to t. (5 pts)

Compute a topological sort of the vertices. Then assign to each vertex a distance from s that is one plus the maximum distance of all of its predecessors to s. An alternate (but less efficient approach), is to assign all edges a distance of -1, and then use the Bellman-Ford algorithm to compute shortest paths.

Rubric:

For both a and b if several algorithms are provided and at least one is wrong: 0 points
(Because otherwise every algorithm could be tried for all problems)

First variant
Topological sort: 2 points
Assign distance: 2 points
Runtime: 1 point

Second variant
Assign all edges a distance of -1: 2 points
Bellman-Ford: 2 points
Runtime: 1 point

b. Path of length K: Given a directed graph, vertices s and t, and an integer K, find a path of length exactly K between s and t. (5 pts)

This can be done with a dynamic programming algorithm, which records that there is a path of length j from s to v if there is a path of length j−1 from s to u and an edge from u to v. K phases of the Bellman-Ford algorithm (with edges assigned a length of one) does not work for this, since a single phase of Bellman-Ford can extend a path by more than one edge.

Rubric:

Dynamic Programming mentioned: 2 points
Correct application to the problem: 2 points
Runtime: 1 point

c. Longest simple path. Given a directed graph and vertices s and t, find the longest simple path between s and t. (Recall that a simple path is a path with no repeated vertices.) (5 pts)

This is NP-Hard. The reduction is from the Hamiltonian Path problem.

Rubric:

NP-Complete mentioned: 1 point
Hamiltonian Path or cycle mentioned: 2 points
Correct reduction: 2 points

4) 15pts

Let G= (V, E) be a directed graph with distinguished vertices s and t. Describe an algorithm using N/W flow to compute a minimum sized set of vertices to remove to separate s and t. Your algorithm should identify the actual vertices to remove (and not just determine the minimum number of vertices that could be removed).

Solution:

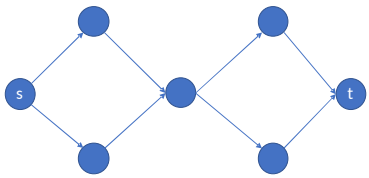
1. We build a flow graph and then use the minimum cut in the graph to determine the set of vertices V to remove.
2. We then split each vertex, other than s or t into vertices v_{in} and v_{out} with a unit capacity edge from v_{in} to v_{out}.
3. The vertex s is replaced by s_{out} and t is replaced by t_{in}.
4. The edge (u, v) is replaced by an edge (u_{out}, v_{in}) with infinite capacity (we don't want any of the existing edges in the min-cut).
5. We compute a maximum flow between s_{out} and t_{in}.
6. Let S be the set of vertices reachable from s_{out} in the residual graph by paths of positive capacity.
7. We say that a vertex v is a cut vertex if v_{in} is in S but v_{out} is not in S.
8. The cut vertices are a minimum set of vertices to remove to separate the graph.

Rubrics:

- Use a minimum cut to find the vertices to be removed. (3 points).
- Vertices to be removed are on one end of the edges of a min-cut. (3 points).
- The min-cut should be the one with vertices having the most incident edges (3 points), except for s and t. (2 points).
- Split the vertices, except s and t (3 points).
- Capacity for edges added to split vertices not defined to one unit (-2 points).
- Capacity for edges other than the split vertices not defined to infinity capacity (-2 points).
- The solution does not use Network Flow algorithm (-8 points).
- N/W flow algorithm that works for all cases (15 points).

Common mistakes:

- Assuming that the max-flow is the size of the set of vertices to be removed (considering all edges have one-unit capacity). Counter example:



- Running BFS from s in the residual graph and identify the leaves as vertices to be removed.

5) 15 pts

A bus rental company has to solve the following problem daily and need an efficient algorithm to help with their planning. They have n buses, m drivers, and they need to plan for k trips. Each bus is suitable only for certain types of trips. For example, a trip to the mountains may require shorter and more powerful buses. They also know that not every driver can drive every bus. Note: a driver and a bus are assigned to a trip for the whole day. n>k, m>k

a- Describe a NW Flow based solution to determine, given k trips (with type of trip specified) if they can satisfy all k trips for that day. (10 pts)

Solution:

S → Driver → Buses → Trip → T

- Create a source s and connect it to the m drivers with capacity 1 (each driver can only be assigned to one bus and one trip for the whole day).
- Create a sink t and have each trip connect to t with capacity 1 (so that we know that we can satisfy k trips or not).
- Create a link from drivers to buses. The link should exist only if the driver can drive that bus. Each link should have a capacity 1.
- Create a link from buses to trips. The link should exist only if the bus is suitable for that trip. Each link should have a capacity 1.

Run maxflow algo. If the max flow == k, then they can satisfy all k trips for that day.

Rubric:

- **Have source and sink (1pt)**
- **Add source to all drivers (1pt)**
  - **The edge between the source to all drivers should have a capacity of 1 (1pt)**
- **Add edges between drivers to *the buses that he can drive* (1pt)**
  - **The above edges should have a capacity of 1 (1pt)**
- **Add edges between buses and the trip *that the bus is suitable for (1pt)***
  - **The above edges should have a capacity of 1 (1pt)**
- **Add edges from all trips to sink (1pt)**
  - **The above edges should have a capacity of 1 (1pt)**
- **Run maxflow algorithm and clearly state that if we obtain the max flow = k, then we can satisfy all k trips for that day. (1pt)**

- **The opposite order also works. (S->trips->buses->drivers->T)**
  - **(the grading is still the same)**

***Note students have to mention the underlined information in order to get a full point for that item.**
***Some students just drew the graph without "describing". We will give points based on what they described or what is obvious.**

b- If they cannot meet all the demand for trips on a given day, they want to know whether it is due to the fact that they don't have an appropriate set of drivers or a suitable set of buses, or both. Describe how they can make that determination. (5 pts)

Solution:

The idea is to find the bottleneck in the graph.
Find min-cut closest to S.

- If S involves only edges between buses and trips, then the problem is with not having the right buses for the trip.
- If it involves edges between buses and trips as well as drivers and buses, the problem is with both.
- If it involves **only** edges between drivers and buses, the problem is with not having the right drivers for the bus.
  - But it also may be related to not having the right buses. In this case, find the min-cut closest to T. If this cut involves any edges between buses and trips, there is also a problem with not having the right buses.

Rubric

- **Any mentioning of finding a min-cut, if flow < k between sections in the graph to solve this, or solving this based on identifying a bottleneck in the graph (2pt)**
- **Only Buses and trips min-cut → problem is with buses (1pt)**
- **Only Drivers and buses min-cut → problem with the drivers (1pt)**
- **Min-cut for both buses and trips as well as drivers and buses, the problem is with both (1pt). ← clearly state the case for both to get the point for this.**

6) 10 pts

In the MERGE-SORT algorithm we merge two sorted lists into one sorted list in O(n) time. Describe an O(n log k)-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists. Be sure to explain why your algorithm runs in O(n log k)-time.

Solution:

First, we remove the smallest element from each sorted list, and we build a min-priority queue (using a min-heap) out of these elements in O(k) time. Then we repeat the following steps: we extract the minimum from the min-priority queue (in O(log k) time) and this will be the next element in the sorted order. From the original sorted list where this element came from we remove the next smallest element (if it exists) and insert it to the min-priority queue (in O(log k) time). We are done when the queue becomes empty and at this point, we have all the numbers in our sorted list. The total running time is O(k + n log k) = O(n log k).

Priory queue also works instead of heap in the above solution.

Doing pairwise merge between lists also works. We merge k/2 pair of lists with O(n/k) elements, then k/4 pair of lists with O(2n/k) elements, …., so k/2*n/k + k/4*2n/k+…. = n/2 + n/2+ … = n/2 * log k

Rubric:

-10 pts for the wrong algorithm,
-5 pts for wrong time complexity argument
-7, partially wrong algorithm like not using a heap and ending up with O(nk)

7) 10 pts

Duckwheat is produced in Kansas and Mexico and consumed in New York and California. Kansas produces 15 shnupells of duckwheat and Mexico 8. Meanwhile, New York consumes 10 shnupells and California 13. The transportation costs per shnupell are $4 from Mexico to New York, $1 from Mexico to California, $2 from Kansas to New York, and $3 and from Kansas to California.

Write a linear program that decides the amounts of duckwheat (in shnupells and fractions of a shnupell) to be transported from each producer to each consumer, so as to minimize the overall transportation cost.

Solution:

We will use concatenation of the first letters of the two cities for the shnupells of duckwheat transported between those cities (i.e. MN for the quantity of shnupells between Mexico and New York etc.)

The linear program will be the following:

- minimize 4MN +MC +2KN +3KC
- MN + KN = 10
- MC + KC = 13
- MN + MC = 8
- KN + KC = 15
- MN,MC,KN,KC ≥ 0

The constraints must be equalities and inequalities are not accepted as correct solutions.

Rubric:

Full credit on defining all the constraints, and objective function.

5- objective function

1- For each constraint (Total +5):

For the last constraint, all variables used should be marked as >=0

Additional Space

Additional Space