CS570 Summer 2022: Analysis of Algorithms   Exam II

	Points		Points
Problem 1	20	Problem 5	12
Problem 2	9	Problem 6	16
Problem 3	12	Problem 7	16
Problem 4	15		
	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) 20 pts

Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification.

[TRUE/FALSE]
If we can use a 2-approximation algorithm for vertex cover to find a 0.5-approximation algorithm for independent set in polynomial time, then P = NP.

[TRUE/FALSE]
In class we showed that the weighted vertex cover problem is polynomial time reducible to linear programming.

[TRUE/FALSE]
Breadth first search is an example of a divide-and-conquer algorithm.

[TRUE/FALSE]
Suppose f is a max flow in a flow network G. Increase the capacity of an edge in G by 1 unit. Then, updating the flow f to reflect the new max flow in G can be done in linear time.

[TRUE/FALSE]
It is not known whether P $\subseteq$ NP.

[TRUE/FALSE]
In the scaled version of the Ford Fulkerson algorithm, choice of augmenting paths cannot affect the number of iterations.

[TRUE/FALSE]
Maximum value of an s-t flow could be less than the capacity of a given s-t cut in a flow network.

[TRUE/FALSE]
If the runtime of an algorithm is bounded by a polynomial in the number of bits in its input then this algorithm is considered to be efficient.

[TRUE/FALSE]
Let G be an arbitrary flow network with a source S and a sink T, and every edge has a positive capacity. Let (P, Q) be a minimum S-T cut, if we increase the capacity of every edge by 1, (P, Q) will remain a minimum S-T cut in G.

[TRUE/FALSE]
The problem of finding out whether a given flow f in a flow network G is a maximum flow or not is in the class NP.

2) 9 pts (for each question, full score is given if only all correct answers for that question are selected, zero points otherwise)

I. Assume P ≠ NP. If X is polynomial time reducible to Y then select all true statements. (3 pts)

a) If Y belongs to NP but not NPC then 3SAT is not polynomial time reducible to X

b) If X belongs to P then Y necessarily belongs to NP

c) If Y is the decision version of the vertex cover problem then X can be any problem
in NP-Hard

d) None of the above

II. If a problem Z is in NP, which of the following statements are true? Circle all correct answer(s). (3 pts)

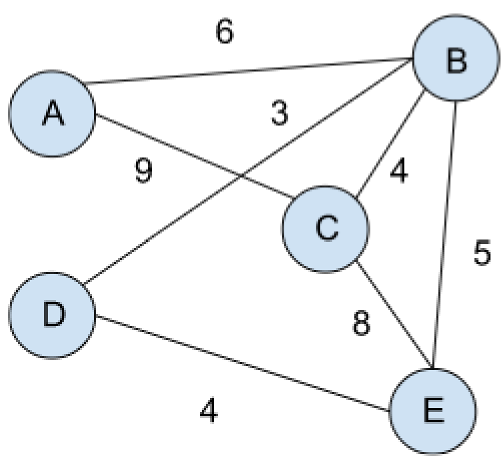
a) If Z can be solved in polynomial time, then P = NP

b) If the Independent Set problem is polynomial time reducible to Z, then Z is NP-complete

c) If there is a polynomial time solution to 3 SAT, then there is a polynomial time solution to Z

d) All of the above

III. Which of the following edge sets forms an MST? (3 pts)



a) {(d,b,), (c,b), (d,e), (a,c)}

b) {(a,b), (d,b,), (d,e), (c,b)}

c) {(d,b,), (d,e), (c,b)}

d) {(c,b), (d,e), (b,e), (a,b)}

3) 12 pts

Given an edge-weighted undirected graph and a list of edges in its Minimum Spanning Tree (MST), describe an efficient algorithm (instead of running Kruskal/Prim's algorithm again) for updating the MST when the graph is modified in the following ways. (You don't need to describe how to re-implement common graph operations such as BFS/DFS and finding a cycle)

A) Update the MST when the weight of an edge e = (u, v) that was not part of the MST decreases. (3 pts)
B) Update the MST when the weight of an edge e = (u, v) that was part of the MST decreases. (3 pts)
C) Update the MST when the weight of an edge e = (u, v) that was part of the MST increases. (3 pts)
D) Update the MST when the weight of an edge e = (u, v) that was not part of the MST increases. (3 pts)

- (a) Add the updated edge to the MST, which will create a cycle. Find all the edges in this cycle, loop over them and find the one with maximum weight. Remove that edge and restore an MST

(b) Do nothing since it does not change the MST.

(c) Intuitively, removing the edge e from the MST disconnects the MST into two trees. We need to find a new edge with the minimum weight who reconnects the two trees to construct a new MST.

To mark vertices into two sets corresponding to two trees, we can run DFS/BFS on the original MST from the vertex u and v, which takes O(|V|+|E|). Then looping over all edges connecting the two sets (two endpoints cannot be in the same set - analogous to a general step in Kruskal's), we find the one e' with the minimum weight among them (in O(|E|) time). Collecting the edge e' with the two trees can make a new MST.

(d) Do nothing since it does not change the MST.

Rubric:

a) 1 pt for adding the edge, 2 pts for finding the largest edge in the cycle and removing it
1 pt if comparing and replacing the decreased edge with all larger edges in MST
b) -3 pts for any algorithm provided
c)
   i) 1 pt for removing the modified edge, 2pts for correctly mentioning how to combine the two sets
   ii) -1pt if they mention that by removing the modified edge only a single node will be disconnected rather than a component

iii) 1 pt if comparing and replacing the increased edge with all smaller edges not in MST

d) -3 pts for any algorithm provided

4) 15 pts

Problem Statement: You are given an *m x n* matrix. Jack wants to traverse from the northwest corner cell *[0][0]* to the southeast corner cell *[m-1][n-1]*.
At any cell, he can only make either of the 3 moves and for each move, he loses some energy before landing on the destination cell.

1. Go east by one cell - This decreases energy by 10 units.(eg. If his energy was x units when he moved from cell *[1,2]* to *[1,3]*, his energy becomes *x-10* units before he lands on *[1,3]*)
2. Go south by one cell - this decreases energy by 10 units.
3. Go diagonally south-east by one cell - this decreases energy by 15 units.

You cannot make a move that takes you outside the matrix.

For all $0 \leq i < m$ and $0 \leq j < n$, *G[i][j]* is an integer representing the factor by which Jack gains energy when he lands on that cell. E.g. if he had *x* amount of energy when he landed on cell *[3][4] with G[3][4]=4*, his energy becomes *4x*. He always starts at cell *[0][0]* with energy = 20. Also, *G[0][0] =1* and *G[m-1][n-1] = 1*

Note : if you are moving from cell A to B, the energy deduction of the move takes place before you land on B and the energy factor is applied after you land on cell B

Objective : Find an optimal path from the starting cell to the target cell that maximizes your final energy at cell *[m-1][n-1]*.

(a) Define the subproblem in your own words (3 pts)

OPT(i,j) is defined as the optimal value or the maximum value of energy achieved by traveling from [0,0] to [i, j]

(b) Write an efficient algorithm/ pseudocode to find the optimal value of the solution, addressing the base conditions, recurrence relation, and edge cases. (6 pts)

1. Initialize a 2-D OPT array of size m X n
2. OPT[0,0] = 1
3. For j in 1, 2, 3 .. n-1, OPT[0,j] = (OPT[0, j-1]-10) * G[0, j]
4. For i in 1, 2, 3 … m-1, OPT[i,0] = (OPT[i-1, 0]-10) * G[i, 0]
5. For i in 1, 2, 3 … m-1

   For j in 1, 2, 3 … n-1

   OPT[i][j] = max(OPT[i-1][j]-10, OPT[i][j-1]-10, OPT[i-1][j-1]-15) * G[I][j]

Return OPT[m-1, n-1]

(c) Where will the optimal value of the solution lie and how will you obtain the optimal solution (path to traverse)? (3 pts)

Optimal value of the solution lies at OPT[m-1, n-1]
To obtain an optimal solution or optimal path, we traceback from OPT[m-1, n-1] to OPT[0, 0].
We add OPT[m-1, n-1 to our solution and then -
At each OPT[I, j], we see which of the 3 arguments gave the highest result for the recurrence ->
max(OPT[i-1][j]-10, OPT[i][j-1]-10, OPT[i-1][j-1]-15) * M[I][j]
We select the arguments with max value, for a tie, we can choose either of the values.
We continue this till OPT[0, 0]

(d) What is the time complexity of your algorithm? Is this an efficient solution? (3 pts)

Time complexity O(mn)
Yes, efficient

5) 12 pts
In class we showed that undirected Hamiltonian Cycle is polynomial time reducible to undirected Hamiltonian Path. Prove that directed Hamiltonian Cycle is also polynomial time reducible to directed Hamiltonian Path.

Solution copied and modified form discussion problem solutions

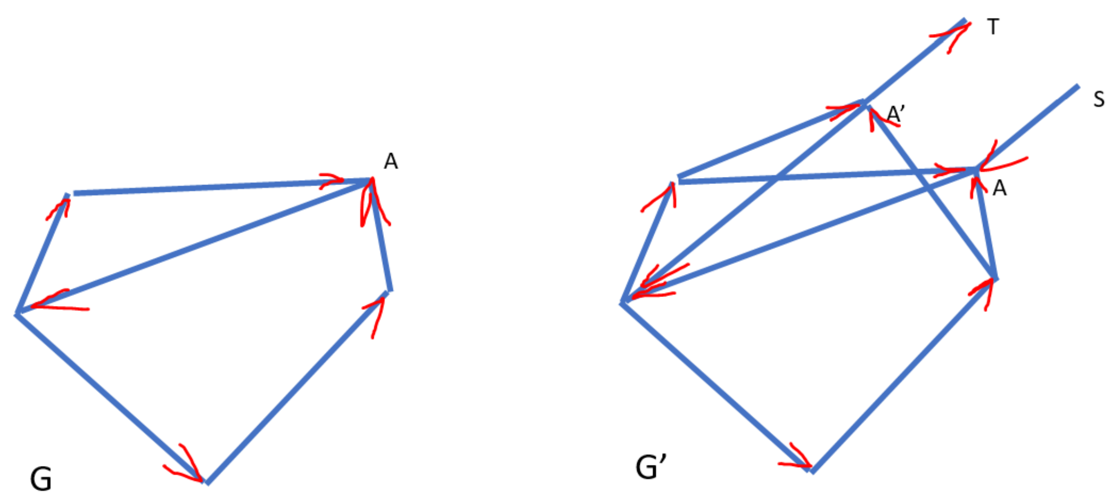
Showing HP to be in NP - not asked for and has no credit.

Approach 1: Single call to HP black-box

Claim (1 point - only if a reasonable construction is presented):
Given graph G—an instance of the Directed Hamiltonian Cycle problem, we will construct G’ (as described below) such that G’ has a Hamiltonian Path iff G has a Hamiltonian Cycle.

Construction of G’ (6 points):
We consider a node in G, say A and split it into Nodes A’ and A” -i.e. remove A and add A’ and A”. Let A’ have all the incoming edges of A and let A” have all the outgoing edges of A. (Call this Option 0)

1) Optionally (i.e. not necessary), also add two nodes S and T with edges A’T and SA”.
2) Further optionally, (**<u>but only if option 1 has been opted for</u>**), make A’, A” identical copies of A, i.e. have both incoming/outgoing edges that A did. This has been the most common approach. The resulting transformation has been shown in the figure below (the A in the right figure should be A”)



Proof (5 points):

A- If we are given a Hamiltonian Path P in G', since A' and A'' have only incoming and outgoing edges respectively (in option 0), or, T and S have only incoming and outgoing edges respectively (in options 1,2) **<u>the path P must go from S to T</u>**. Then, Ignoring the two new edges in the respective options, this path will give us a Hamiltonian Cycle in G by replacing A' and A'' with A to close the loop.

B- If we are given a Hamiltonian Cycle in G, we can create a Hamiltonian Path in G' by splitting the Cycle at node A and creating a path by following the construction in the respective options.

Approach 2: Multiple calls to HP black-box

Construction (6 points): For each edge (u,v), create an HP instance G'(u,v) by adding nodes s,t and edges (u,s) and (t,v). Okay to remove or not remove the edge (u,v). (A slightly more *compact* reduction does this for all outgoing edges (u,v) for a particular u - or all incoming edges (u,v) for a particular v - instead of ALL edges in G)

Claim (1 point - only if a reasonable construction is presented):
Given graph G—an instance of the Directed Hamiltonian Cycle problem, we construct instances G'(u,v) (as described above) such that **<u>at least one of them</u>** has a Hamiltonian Path iff G has a Hamiltonian Cycle.

Proof (5 points):

A- If we are given a Hamiltonian Path P in G'(u,v) for **<u>some</u>** (u.v), P must go from t to s for the same reason as described above in approach 1 proof. Then ignoring the new edges, the path from v to u with the edge (u,v) gives a HC in G.

B- If we are given a Hamiltonian Cycle C in G, **<u>pick any edge (u,v) on C</u>** (can't be an arbitrary edge), then G'(u,v) has the HP t -> v -> trace C -> u -> s.

Common <u>fundamental</u> mistakes (which seriously affect the correctness of the solution):

1) Specific to the approaches - as highlighted in bold and underline.
2) General mistake - incoherent solution flow: Assuming a HC in G or specifics thereof when coming up with the construction, OR, doing constructions when proving either side of the claim.

Partial credit offered in these cases depending on the relative correctness of the rest of the solution.

6) 16 pts

A company has n software applications. Each application i has F(i) unique features (or functionalities), i.e. features are not shared across different applications. Each feature j requires one database (DB) connection from a given subset of databases D(j). For example: An application i with F(i)=3 features could have these database requirements: {{1, 3}, {2, 4, 5}, {6}}. This means that the first feature requires access to either database 1 or 3, the second feature requires access to databases 2, or 4, or 5, and the third feature requires access to database 6. Note that a database may be included in more than one D(j) sets.

Assuming that each database k can only accommodate C(k) connections at a given time, design a network flow solution to determine if there is an assignment of features to databases in which each application will at most have one feature without a DB connection. In other words, each application i should have at least F(i)-1 of its features up and running.

a) Describe the complete construction of your network. (12 pts)

Solution 1: Constructing a max-flow based solution

[+1] Create source node S and sink node T

[+1] Create n nodes representing the n applications

[+2] Connect S to each node representing application i with capacity F(i)-1

[+1] Create F(i) nodes for each application i representing its feature set

[+2] Connect the node representing application i to each of its feature nodes with capacity of 1

[+1] Create a node for each database (i.e. one for each DB in the superset of D(j) of all features of all applications)

[+2] Connect each feature node j to the subset of the DB's it can use i.e. D(j) with capacity 1

[+2] Connect an edge from each database k to T with capacity C(k)

Solution 2: Constructing a circulation-based solution

[+1] Create source node S and sink node T. Create a node for each database (superset of all D(j))

[+1] Create n nodes representing the n applications. Create F(i) nodes for each application i representing its feature set

[+2] Connect S to each node representing application i with capacity F(i)-1

[+2] Set the demand of source node S to -∑(F(i) - 1) and the demand of sink node T to ∑(F(i) - 1) (otherwise, connect the sink to source node with unlimited capacity.)

[+2] Connect the node representing application i to each of its feature nodes with capacity of 1

[+2] Connect each feature node to the subset of the DB's it can use with capacity 1

[+2] Connect an edge from each database k to T with capacity C(k)

b) Which problem will you solve in this network and what algorithm will you use to solve it? (2 pts)

[+1] Find Max flow (If the solution for (a) is Max Flow) / Find the feasible circulation (If the solution for (a) is Circulation)
[+1] Scaled version of Ford Fulkerson or Edmonds Karp etc

c) Describe how the solution to your network flow problem can be used to determine whether the given problem has a solution (i.e., each application can have at least all but one of its features up and running.) No proof is necessary. (2 pts)

For solution 1: Max-flow based
[+2] We have a solution, i.e., we can have all but one features in all applications up and running if and only if value of max flow = ($\sum$ (F(i)-1) for all i) in the network above

For solution 2: Circulation-based
[+2] …. If and only if a feasible circulation can be found

6) 16 pts

Suppose you are the same professor from exam 1 who is obsessed with giving algorithm books as gifts to students because they got good grades. So again, there are *n* students sitting in a line. The i^{th} student in the line has grades *g(i)*. There are *m* different book types (titles) in your possession, with several copies of each type (title). A single copy of the j^{th} type has a price of *p(j)* and you own *c(j)* copies of that type.

You will give rewards to the students based on the following new set of criteria:

1. Each student receives at least one book.
2. For any two students *i* and *i'* sitting next to each other (i.e. $|i - i'| = 1$):
   a. If *g(i) > g(i')* then number of books given to i must be greater than that given to *i'*.
   b. If *g(i) = g(i')* then the same number of books should be given to *i* and *i'*
3. No student should get multiple copies of the same book type.
4. Any two students sitting next to each other, should not get copies of the same book type.

Write an integer linear program to find the minimum total price of books you must distribute.

a) Describe what your discrete variables represent (2 pts)

Variables: X_ij is 1 if student i gets a copy of book j, 0 otherwise.

b) Write your linear constraints, and for each linear constraint mention which of the problem constraints they enforce (12 pts)

Constraints:

Sum_j X_ij >= 1	for all i	… Const. 1 above
Sum_j X_ij >= Sum_j X_(i+1)j + 1	for all i with g(i) > g(i+1)	… Const 2a
Sum_j X_ij <= Sum_j X_(i+1)j - 1	for all i with g(i) < g(i+1)	… Const 2a
Sum_j X_ij = Sum_j X_(i+1)j	for all i with g(i) = g(i+1)	… Const 2b
X_ij + X_(i+1)j <= 1	for all i, j	… Const 4
Sum_i X_ij <= c(j)	for all j	… No. of copies owned
X_ij \in {0,1}	for all i,j	… Var defn + Const 3

point distribution Const 2a (combined) : 3 points, Last one : 1 point, others : 2 points

Lack for all quantifiers. -0.5 points

c) Write your linear objective function (2 pt)

Objective: Min Sum_i Sum_j p(j) X_ij

Additional Space

Additional Space

Additional Space