CS4102 - ALGORITHMS - SPRING 2022

DUE TUESDAY, MAY 3, 2022 AT **11:30 pm** VIA GRADESCOPE

D1BASIC(MST3K)

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: list collaborators's computing IDs

Sources: Cormen, et al, Introduction to Algorithms. *(add others here)*

PROBLEM 1 *True or False. (You don't have to explain this in your submission, but you should understand the reason behind your answer.)*

1. When the *Ford-Fulkerson* algorithm completes, each back-flow edge from v back to u in the residual graph G_f represent the final flow values for edge (u, v) in the flow-graph G.
   Solution: True. It represents the flow that can be pushed back or the current flow that is going from u to v.

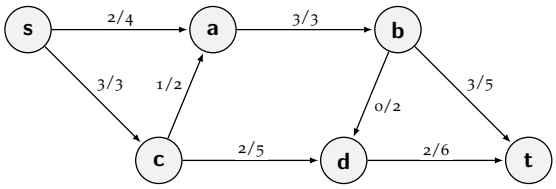
2. In a standard network flow-graph, under certain conditions a vertex v that is not the source or the sink can have total in-flow that has a different value than its total out-flow.
   Solution: False.

3. In *Ford-Fulkerson*, for a pair of vertices u and v connected in the residual capacity graph G_f, the sum of the values for the back-flow and the residual capacity edges between that vertex-pair must always equal the capacity of the edge between them in the flow-graph G.
   Solution: True.

4. When using Ford-Fulkerson, if there is no augmenting path in G_f, then there exists a cut in G whose capacity equals f, the max-flow value for graph G.
   Solution: True.

PROBLEM 2 *Max Flow*

Given the following Flow Network G:



1. Find the Residual Graph G_f for G by listing all the edges in G_f and the numeric value associated with each edge.
   Solution:

Node From \ Node To	S	A	B	C	D	T
S		2		0		
A	2		0	1		
B		3			2	2
C	3	1			3	
D			0	2		4
T			3		2	

2. Find an augmenting path in the graph G_f. List the nodes in the path you found in order (e.g., $s \to a \to b \to t$).
Solution: $s \to a \to c \to d \to t$

3. Find the min cut of the graph. List the nodes below on each side of the cut.

S **(one side)**	$V - S$ **(other side)**
s a	b c d t

4. What is the maximum flow of this graph?
Solution: 6

PROBLEM 3

We used *Ford-Fulkerson* to solve the *Vertex-Disjoint Paths* problem in class by reducing *Vertex-Disjoint Paths* to *Edge-Disjoint Paths* and *Edge-Disjoint Paths* to *Max Flow*. Recall that a set of vertex-disjoint paths is a set of edge-disjoint paths where each node is used at most once. How would we modify the reduction from *Vertex-Disjoint Paths* to *Max Flow* if we want to compute a set of edge-disjoint paths where each vertex is used at most *twice*? Briefly describe our original reduction and the change(s) you would make. **Note:** If you prefer, you may just modify the reduction from *Vertex-Disjoint Paths* to *Edge-Disjoint Paths*.
Solution: To solve edge disjoint paths, we simply computed max flow given that each edge had a capacity of 1. From a solution to max flow, we could generate a solution to edge disjoint paths. To solve vertex disjoint paths, we converted all the nodes in our graph to a pair of connected nodes, assigning all the inflow to the prior node to the "left" node post-transformation and the beginning of the outflow edges to the "right" node post-transformation. This converts our vertex-disjoint path problem into an edge-disjoint path problem. Now, we can use the reduction from edge-disjoint paths to max flow and compute accordingly. Additionally, we do not place an effective limit on non-node edges when reducing. For all edges that do not traverse a pair of nodes that represent a split node, we do not place a limit on flow as our "limiting-reagent" in the vertex-disjoint path problem are the number of times we can go through edges not the number of times we can go through edges.

In the reduction from vertex-disjoint paths to edge-disjoint paths, we converted each node into a pair of connected nodes with max flow of 1. If instead we converted the nodes into a pair of connected nodes into a pair of connected nodes with max flow of 2 instead, we would be able to use each vertex twice. Just like the prior reduction, we place no limit on max flow for edges that do not connect split nodes.

PROBLEM 4 *NP Completeness*

1. We know that C is NP-Hard and that $A \in NP$. Which of these show that A is NP-Complete?
   a) A reduces to B and B reduces to C
   b) C reduces to B and B reduces to A

   Solution: From the information that we're given, we already know that $A \in NP$. All that's left to prove is that $A \in NP - Hard$. To do so, we can use the definition from lecture to prove that something C NP-Hard. Namely, we have to demonstrate that for all $A \in NP$, $A \leq_p C$. In our case, we have to demonstrate that all NP problems reduce to A. By definition, NP-Hard problems are problems that all NP problems reduce to, and we have one of those in C. So all we have to do is show that C reduces to A because then by transitivity, all NP problems reduce to A. Statement b shows this.

2. Which shows that $P = NP$, given that A reduces to B and B reduces to C?
   a) $A \in P$ and $C \in NP$-Hard
   b) $A \in NP$-Hard and $C \in P$

   Solution: Statement b shows that $P = NP$, given that A reduces to B and B reduces to C. Given statement b, we know that all NP-hard problems reduce to a P problem. The definition of NP-hard is that all NP problems reduce to this given problem. And if all NP problems reduce to A, and A reduces to a Polynomial Time problem C, then by transitivity all NP problems reduce to a Polynomial Time problem and therefore, P = NP as all NP problems can be solved in polynomial time.

PROBLEM 5 *True or False. (You don't have to explain this in your submission, but you should understand the reason behind your answer.)*

1. In our reduction of an instance G of the *vertex-disjoint path* problem to an instance G' for *edge-disjoint path* problem, if two paths in G' share a vertex, then the two paths that correspond to those in G must share an edge.
   Solution: If two paths in G' share a vertex, then those paths share an edge in the G' path but not necessarily in the corresponding G path. Consider the following two paths in G: $a \to b \to c$ and $d \to b \to e$. Both share a vertex b in G. But in the corresponding G' paths, both share an edge specifically, the edge from bl to br, the decomposition of b that occurs during the reduction from a vertex-disjoint path problem to a edge-disjoint path problem. The path's in G' become: $al \to ar \to bl \to br \to cl \to cr$ and $dl \to dr \to bl \to br- > er \to el$. So I think False, but the wording is also very vague here.

2. In our example illustrating bipartite matching, we can tell if every hard-working TA was matched with exactly one adorable dog and *vice versa* by transforming the bipartite graph to a network flow problem and checking if the max-flow $|f| = V$, the total number of vertices in bipartite graph.
   Solution: False. Not the total number of vertices in the bipartite graph. The flow should be equal to the number of TA-dog pairings, or the total number of TAs or total number of dogs.

3. If we find a polynomial solution to a problem in NP, then this proves that $P = NP$.
   Solution: False. We just show that there is a problem in P that is also in NP. That's kind of the point because $P \subseteq NP$. But if this problem is NP-Complete, then this would be True.

4. If someone proves that a given problem X in NP has an exponential lower bound, then no problem in NP-complete can be solved in polynomial time.
   Solution: I think True. This is a little roundabout. But here goes. If a problem in NP has an exponential lower bound, and we know that all NP problems reduce to NP-Hard problems, then we know that all NP-Hard problems share the same exponential lower bound. Because NP-Complete is a subset of NP-Hard, we know that all NP-Complete problems are also lower-bounded exponentially and cannot be solved in polynomial time.

5. It is not possible that an algorithm that solves the 3-CNF problem in polynomial time exists.

   Solution: False, we've only proven that it is NP-Hard. It is still unknown whether or not a polynomial time solution exists or not.

PROBLEM 6 *Create a Reduction*

Reduce *Element Uniqueness* to *Closest Pair of Points* in $O(n)$ time. Element Uniqueness is defined as: given a list of numbers, return true if no number appears more than once (i.e., every number is distinct). Closest Pair of Points is defined as: given a list of points (x, y), return the smallest distance between any two points.
Solution: Lets just create a point from every number. So the list of numbers [1, 2, 3, 4, 5, 6] would become (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6) a pair of points with a distance of 0 or if Closest Pair of Points were to return 0 on our input list (converted list from original list) then we would return True that we have a duplicate element. Otherwise, we would return False.

PROBLEM 7 *Gradescope Submission*

Submit a version of this `.tex` file to Gradescope with your solutions added, along with the compiled PDF. You should only submit your `.pdf` and `.tex` files.