

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

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 .

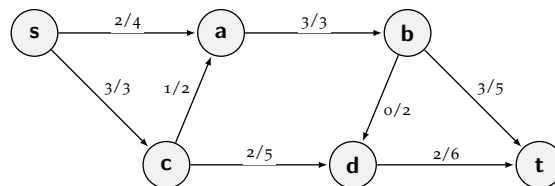
Solution: True

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:

$s \rightarrow c : 0$

$c \rightarrow s : 3$

$s \rightarrow a : 2$
 $a \rightarrow s : 2$
 $c \rightarrow a : 1$
 $a \rightarrow c : 1$
 $c \rightarrow d : 3$
 $d \rightarrow c : 2$
 $a \rightarrow b : 0$
 $b \rightarrow a : 3$
 $b \rightarrow d : 2$
 $d \rightarrow b : 0$
 $b \rightarrow t : 2$
 $t \rightarrow b : 3$
 $d \rightarrow t : 4$
 $t \rightarrow d : 2$

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

Solution: $s \rightarrow a \rightarrow c \rightarrow d \rightarrow t$

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: The original reduction is to make two copies of each node, one connected to incoming edges, the other to outgoing edges when converting an instance of the vertex-disjoint paths problem into an instance of edge-disjoint paths. Now each vertex can be used at most twice so we need to make two copies of each node, one connected to incoming edge and the other to outgoing edge. However, we need two edges between them to limit their edge capacity to 2. The original version only has one edge between the incoming node and outgoing node.

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: b

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

Solution: b

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: False

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

3. If we find a polynomial solution to a problem in NP, then this proves that $P = NP$.

Solution: False

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: True

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

Solution: True (we don't know but most computer scientists believe it so)

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: We can make a list of points and each point's x value is a number in the input list of the *Element Uniqueness* problem, with y value as 0. In this way, we get a list of points $(x, 0)$, and we use *Closest Pair of Points* problem to handle this input. If the return value is 0, then we return False to the *Element Uniqueness* problem. Otherwise, we return True to the *Element Uniqueness* problem.

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.