CS4102 - ALGORITHMS - SPRING 2022

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

D1 BASIC(YLW4SJ)

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

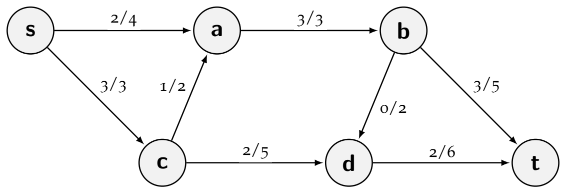
2. In a standard network flow-graph, under certain conditions a vertex v that is not the source or the sink can have total in-flow that has a different value than its total out-flow.
   Solution: False

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:

   (a) (s, a): forward edge = 2 , backward edge = 2

(b) (s, c): forward edge = 0 , backward edge = 3
(c) (a, b): forward edge = 0 , backward edge = 3
(d) (a, c): forward edge = 1 , backward edge = 1
(e) (c, d): forward edge = 3 , backward edge = 2
(f) (b, d): forward edge = 2 , backward edge = 0
(g) (b, t): forward edge = 2 , backward edge = 3
(h) (d, t): forward edge = 4 , backward edge = 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: Originally, to reduce from Vertex-Disjoint Paths to Edge-Disjoint Paths we create make two copies of each node (one connected to incoming nodes and one connected to outgoing) with a single edge connecting the in-node and out-node. To modify this, I would have two edges between each in-node and out-node. This restricts each vertex to two edges, meaning each vertex can be used a max of two times.

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 $C \in NP$-Hard
b) $A \in NP$-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: False

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: To reduce Element Uniqueness to Closest Pair of Points, we can pass in each element in the list for Element Uniqueness as a pair of points. For example, the element 1 would be passed in as (1, 1). Next, Closest Pair of Points will solve for the distance between the closest points. If it returns 0, that means there are are least two of the same points and therefore at least two of the same element. Therefore, we can return False for Element Uniqueness. If it returns anything else, we can return True for Element Uniqueness since there are no points with the same coordinates.

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.