Homework 7

Thomas Kim tsk389 51835

David Munoz dam2989 51840

CS331 Algorithms and Complexity

Problem Q1(a). Prove or disprove
Claim: Consider an implementation of Ford-Fulkerson algorithm which does not create any backward edges in the residual graph. It is claimed that, there exists a constant $\alpha > 0$, such that for any flow network G, this modified implementation is guaranteed to find a flow of value at least α times the maximum-flow value in G **False**

Proof. A method of constructing a graph in which a particular execution of the above algorithm can produce a "max" flow of 1 whereas the max flow of the graph is an arbitrary positive integer.
First, begin with a graph consisting of 3 nodes, s, a, t, as shown in **Figure A**
The only possible flow of this graph generated by the above algorithm is trivially $f = 1 = \frac{1}{1}$
Next, add an additional edge b, with three additional edges: (s, b) with capacity 1, (b, a) with capacity 1, and (b, t) with capacity 1. This is shown in **Figure B**
The max flow of this graph is $f = 2$. Suppose the above algorithm begins by choosing path (s, b, a, t).
The residual graph would contain only edges $(s, a), (b, t)$, so there exists no path from $s - t$
Therefore, the flow returned by the above algorithm is $f = 1 < 2$
Now, for $i = 2...\infty$ do: Add 3 additional nodes x, y, z. Let the current sink be denoted as w.
Node x will be the new sink t. An edge, (w, x) will be added with capacity i (the previous max flow).
Node y will have edges (w, y) with capacity 2, and (y, z) with capacity 1.
Node z will have edges (s, z) with capacity 1, and (z, y) with capacity 1.
This new graph has a max flow of $i + 1$. Simply take the previous max flow, send i flow over the edge (w, x), and 1 flow along the path (s, z, y, t)
However, suppose the algorithm found a "max" flow of value 1 in the previous iteration (which is trivially possible by induction on i).
Take the path used by this "max" flow and add the edges (w, y) with flow 1, and (y, x) with flow 1 to the path.
This new path, when explored first by the algorithm, causes the residual graph to have no path from s to t, as there exists no path from z to x since (y, x) is removed, and by induction the original graph similarly contains no path from any node $n \neq b$ to w, and adding x, y, z does not create any new paths from nodes in the original graph to w.
2 additional iterations of this construction are shown below in figures **C, D Figure A**

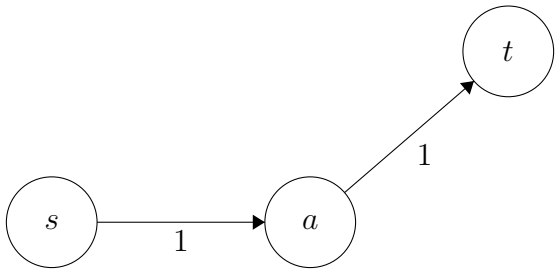


Figure B

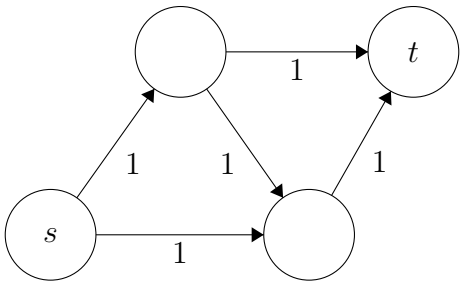


Figure C

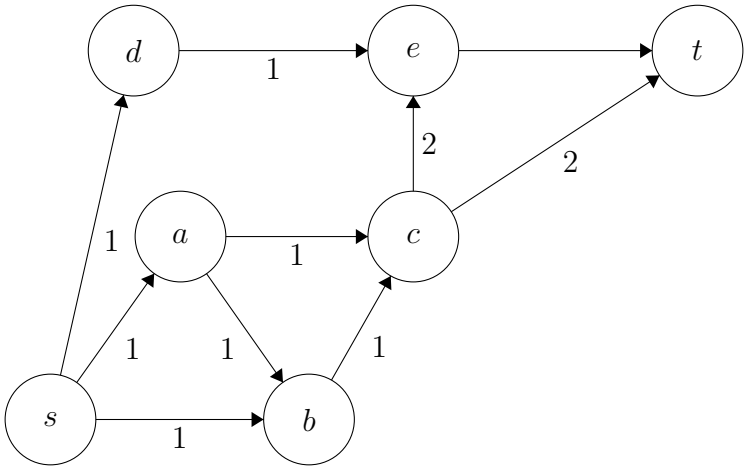
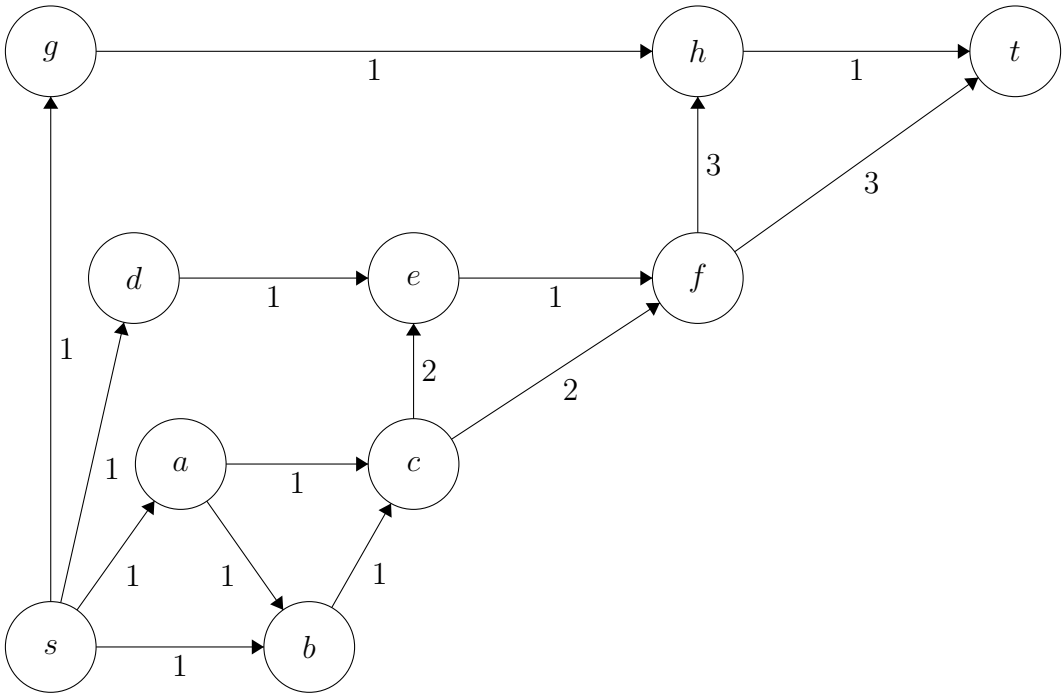


Figure D

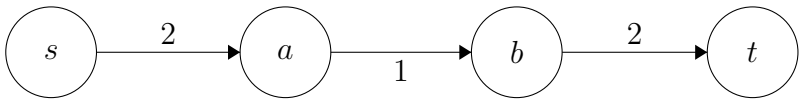


□

Problem Q1(b). Consider a network $G = (V, E)$ with a source s, a sink t, and a capacity c_e on every edge $e \in E$.

Claim: if f is a maximum $s-t$ flow in G, then f either saturates every edge out of s or saturates every edge into t.
False

Proof.

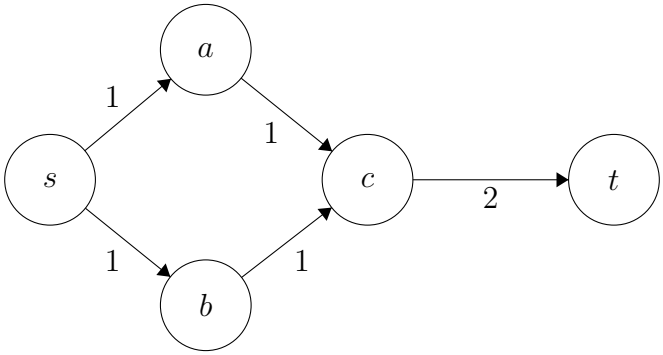


The min cut is $A = \{s, a\}, B = \{b, t\}$. The value of this cut is 1, which implies the max flow $f = 1$. This does not saturate the edge leaving s or the edge entering t.

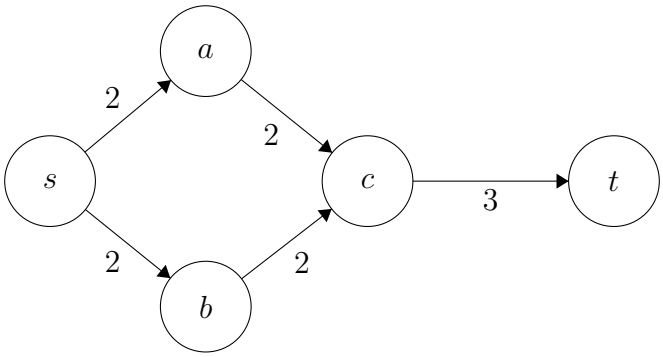
□

Problem Q1(c). Consider a network $G = (V, E)$ with a source s, a sink t, and a capacity c_e on every edge $e \in E$. Let A, B be a minimum $s-t$ cut with respect to the capacities c_e.
Claim: If we add 1 to every capacity, namely $\forall e \in E, c'_e = c_e + 1 \implies A, B$ is still a minimum $s-t$ cut with respect to c'_e
False

Proof. Let G be the graph as pictured below:



Note that a min cut in G is $A = \{s, a, b\}, B = \{c, t\}$ with a value of 2
Now consider G'



Note that the cut $A = \{s, a, b\}, B = \{c, t\}$ now has a value of 4, whereas the cut $A' = \{s, a, b, c\}, B' = \{t\}$ has a value of 3

□

Problem Q2(a). Input: n, $\{b_1, \ldots, b_n\}$, $L_1, \ldots, L_n$

1: Let $G = (V, E)$ be an initially empty graph
2: Suppose the set of buses is denoted $\{b_1, \ldots, b_n\}$
3: Suppose the set of drivers is denoted $\{d_1, \ldots, d_n\}$
4: Suppose the original matching (d_i, b_j) is denoted as $m(d_i) = b_j$
5: $V = \{s, t\}$
6: **for** $\forall 0 < i \leq n$ **do**
7: $\quad V = V \cup \{d_i, m(d_i)\}$
8: $\quad L' = L_i - m(d_i)$
9: $\quad$ Add an edge $e = (s, d_i')$ with cost 0 and capacity 1
10: $\quad$ Add an edge $e = (d_i', m(d_i)$ with cost 0 and capacity 1
11: $\quad$ Add an edge $e = (d_i', d_i)$ with cost 0 and capacity 1
12: $\quad$ **for** $\forall l' \in L'$ **do**
13: $\quad\quad$ Add an edge $e = (d_i, l')$ with cost 1 and capacity 1
14: $\quad$ **end for**
15: **end for**
16: $V = V \cup \{b_{n+1}, d_{n+1}\}$
17: $E = E \cup L_{n+1}$
18: Run any polynomial time algorithm to find a minimum-cost flow for G, denoted f' (ie: Successive Shortest Path)
19: **if** $v(f') = n + 1$ **then**
20: $\quad$ There exists a matching of buses and drivers
21: **else**
22: $\quad$ There does not exist a matching of buses and drivers
23: **end if**
24: **if** $e = (s, m(d_i)) \in f'$ **then**
25: $\quad m(d_i)$ remains matched with d_i
26: **else if** $e = (d_i, b_j), b_j \neq m(d_i) \in f'$ **then**
27: $\quad d_i$ and b_j are now matched, all other matches involving d_i, b_j are invalidated
28: **end if**

Correctness Pf. of Validity

Proof. Note: No driver can be matched with an incompatible bus by construction, since an edge can only exist between a driver and a valid bus.
Claim: No 2 drivers can be matched to the same bus.
Suppose 2 drivers are matched to the same bus.
This means the min-cost flow algorithm A created edges $(d_i, b_k), (d_j, b_k)$
However, the outgoing flow from $b_k = 1$, and the minimum flow along any edge is 1 by construction.
Therefore, no two drivers can be assigned to the same bus.
Claim: No driver can be matched to two different buses.
Suppose one driver is matched to 2 different buses.
This means that either an edge goes from s to $m(d_i)$ or two edges leave d_i
The second case is impossible, as the flow entering d_i is bounded by 1
The first case is impossible, as the flow entering b_i' is bounded by 1

For each driver, he can either stay matched to the same bus or change buses.
This is modeled by the fact that d_i' either sends flow directly to $m(b_i)$ (no change) or d_i (must change by line 8)

Since the algorithm is guaranteed to produce a matching strictly including only matching pairs, it must be valid.

□

Pf. of Optimality

Proof. Suppose the matching generated is non-optimal.
This means there exists some driver d_i which is incorrectly matched to $b_i \neq m(d_i)$
This means the flow generated by the min-cost flow algorithm has some cost $c \geq 1$
This means there exists a valid matching where d_i is matched to $m(d_i)$
To construct an equivalent valid flow to this valid matching, simply take the edge $(d_i', m(d_i))$ for all unchanged pairs, and the edges $(d_i', d_i), (d_i, b_j)$ for pairs which have changed.
This means there exists a valid flow where d_i is matched to $m(d_i)$
By line 10, this valid flow has a cost $c' \leq c - 1$
However, the min-cost flow algorithm is guaranteed to find the lowest cost valid flow.
Contradiction.

□

Runtime

Proof. The construction of G can be done in $O(n^2)$ (polynomial) time.
This is because the algorithm iterates through all n buses/drivers and all n lists, each of which can be up to n long
Solving minimum-cost flow can be done in polynomial time by page 449 in the book.
The final part of the algorithm examines every edge in the flow generated by the aforementioned algorithm.
By construction, there can be up to $3(n + 1)$ edges in the flow, because the max path length is 3 and there are $n + 1$ units of flow leaving s and entering t.

□

Problem Q2(b). Assumptions: No driver has an empty list.

Basic premise:
1. Group together drivers who (transitively) share a bus with another driver in the group.
2. Connect that group with the set of buses that are collectively shared (set union of all their lists).
3. If the group size is larger than the number of buses it maps to, give the drivers GPSes 4. Otherwise put them on the buses which correspond to the group.

To solve (1), simply connect drivers who share a bus with an edge, then take each connected component as a group.
Runtime

Proof. Constructing the graph for step (1) takes $O(n^2)$ time, as there are $n + 1$ lists each with up to $n + 1$ buses on them
Finding connected components can be done in polynomial time using BFS or DFS.
The final assignment of GPSes can be done in $O(n)$ time, as each driver and bus is considered exactly once.
The runtime is therefore polynomial in n.

□

Correctness Validity

Proof. The set union of all groups is equal to the set of all drivers.
Since each group maps to some set of buses, and either the entire group or entire mapped buses get GPS, the matching is valid.

□

Optimality

Proof. Suppose some group is given GPSes but it would be more ideal to give GPSes to their mapped buses.
Note that no other group can be mapped to any of these buses by construction, otherwise this group would be part of another group.
This implies there are fewer buses than people.
This is a contradiction, as step 3 of the algorithm implies there are fewer drivers in the group than mapped buses.
Suppose some mapped buses are given GPSes but it would be more ideal to give GPSes to the group mapped to them.
Note that by the pigeonhole principle, at least one driver will be idle at any given time.
This means giving a GPS to that driver will cause one idle GPS at any time.
However, if all those GPSes were given to the buses, then there would be 0 idle GPSes, as there are more drivers than buses by line 4.

□