# CS 255: Homework 1

Due September 15 at 11:59 pm

**Instructions:**

1. You may work in groups of up to 3 students.
2. Each group makes one submission on Canvas.
3. Include the name and SJSU ids of all students in the group.
4. You may discuss high level concepts with other groups, but do not copy other's work.

## Problem 1

Let $G(V, E)$ be a flow network with integer edge capacities. Suppose $f$ is a maximum flow in $G$. Describe algorithms for the following operations:

(a) Increment($e$): Increase the capacity of edge $e$ by 1 and update the maximum flow.

(b) Decrement($e$): Decrease the capacity of edge $e$ by 1 and update the maximum flow.

Clearly, you could just recompute a maximum flow, but we want to do this faster. Your algorithms should run in $O(m+n)$ time, where $m = |E|$, and $n = |V|$.

**Solution:**
(a) Let's start with some intuition. If we increase the capacity of a single edge $e$, how could the maximum flow value change? Certainly, it will not decrease. So it either stays the same or increases. Can it increase by more than 1? Intuitively, no it cannot since adding one to the capacity of a single edge only allows for at most one extra $s$-$t$ path. We could prove this directly, but it is a bit tedious. A cleaner argument leverages the max-flow min-cut theorem as follows. The value of maximum flow equals the capacity of the minimum cut. By incrementing (or decrementing) the capacity of a single edge by one, the capacity of minimum cut increases (or decreases) by no more than one.

Suppose we increase the capacity of edge $e$ by 1. Can we increase the maximum flow value? Running a single iteration of Ford-Fulkerson provides the answer. If there is an $s$-$t$ path in the residual graph $G_f$, we augment along the path to increase the maximum flow by one. Otherwise, the current flow is optimal, since there are no $s$-$t$ paths in $G_f$ (this is the termination condition for Ford-Fulkerson and guarantees that current flow is a maximum flow). One iteration of Ford-Fulkerson takes $O(m+n)$ time (it is basically just BFS or DFS in $G_f$ starting from $s$).

(b.) By the argument of part (a), the maximum flow value either stays the same or decreases by one. There are two cases to consider, we reduce the capacity of a saturated edge $f_e = c_e$, or an unsaturated edge $f_e < c_e$.

Decreasing the capacity of any unsaturated edge does not require any changes to the existing flow (the current flow $f$ still obeys capacity constraints and flow conservation). Further, it cannot add any new $s$-$t$ paths in $G_f$ since the capacity of forward edges does not increase, and the capacity of reverse edges stays the same. Therefore, $f$ remains a maximum flow after reducing the capacity of the unsaturated edge $e$.

If the edge $e$ is saturated $f_e = c_e$, then we need to find an $s$-$t$ path with positive flow that uses $e$, and remove one unit of flow from the path. We can find such a path $P$ using BFS or DFS, and decrease the flow on each edge of $P$ by one. The resulting flow $f'$ clearly still obeys capacity and flow conservation constraints. This step runs in $O(n + m)$ time.

Are we done? Not quite. Removing flow from this path changes the residual graph $G_f$. We still need to check if there are $s$-$t$ paths in $G_f$ and augment along any such path. This step is easy. Just run one iteration of Ford-Fulkerson. After at most one iteration there will be no $s$-$t$ paths in $G_f$ and we will have a maximum flow. This entire process runs in $O(n + m)$ time.

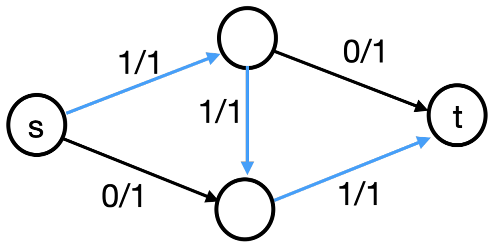
## Problem 2

Your friend suggests a greedy modification to the Ford-Fulkerson algorithm. Instead of maintaining the entire residual graph, just use the forward edges showing the residual capacity $c_e - f_e$ of each edge in the graph $G$ for the current flow $f$. Notice that once we saturate an edge we can just remove it, no reverse edges!

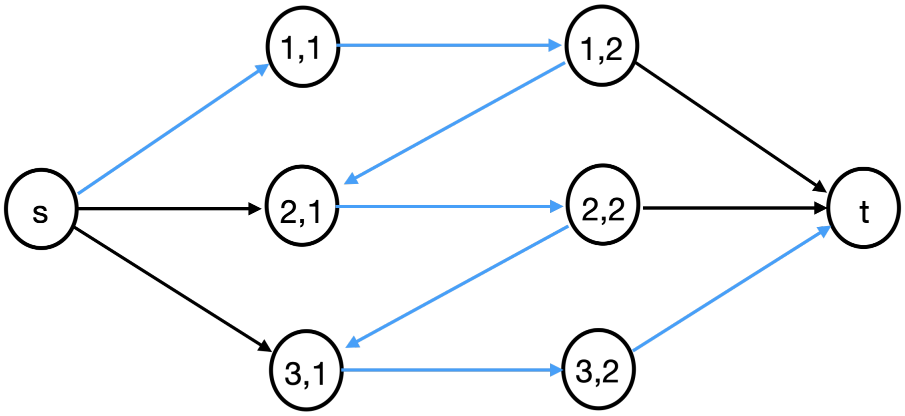
(a) Show that your friend's algorithm is not guaranteed to compute a maximum flow.

(b) Your friend sees the issue, but says not to worry; their greedy algorithm always produces a flow with a value within a constant fraction of the maximum flow value! Unfortunately, this is also wrong. Show that for any constant $\alpha > 1$ there is a flow network $G$ such that the value of the maximum flow is more than $\alpha$ times the value of a flow computed by your friend's greedy algorithm.
[Hint: there are no restrictions on the augmenting paths you can choose, so pick the worst possible one.]

**Solution:**
(a.) The algorithm only using forward edges in the residual graph $G_f$ is the same the naive greedy approach discussed in lecture 3. We can use a similar graph (or the same one) as in lecture. In the figure below, edges are labeled (flow/capacity). The maximum flow value is 2 using the top and bottom $s$-$t$ paths. The flow shown in blue has value 1, and blocks the greedy algorithm's progress.



(b.) Essentially, we want to extend the idea from part (a) to a graph with $\alpha > 1$, $s$-$t$ paths. That is, there should be $\alpha$ distinct 'straight-line' $s$-$t$ paths that can be blocked by single bad choice of flow. One way to achieve this is to have each 'straight-line' $s$-$t$ path consist of 2 intermediate points. Call the left and right points of the $i$th path $(i, 1)$ and $(i, 2)$ respectively; refer to the figure below which shows 3 distinct paths: top, middle, and bottom. In the figure, all edges have unit capacity (omitted to remove clutter). Creating a directed edge from node $(i, 2)$ to node $(i+1, 1)$ gives a way for flow on the $i$th path to block flow on path $i+1$. By chaining this together of over $\alpha > 1$ 'straight-line' $s$-$t$ paths, we ensure a single flow (shown in blue) blocks all other $s$-$t$ paths. Thus, the maximum flow value is $\alpha$, but the greedy algorithm can obtain a flow with value only 1 with the wrong choice of augmenting path.



## Problem 3

This problem shows how to decompose flows into a small number of paths and cycles. Consider a flow network defined by a directed graph $G(V, E)$ with source $s$, sink $t$, and integer capacities $c_e > 0$, $\forall e \in E$. Let $m = |E|$, and $n = |V|$.

(a) A flow is *acyclic* there are no directed cycles with positive flow. Show that for every flow $f$ there is an acyclic flow with the same value as $f$.

(b) Is the Ford-Fulkerson algorithm guaranteed to produce an acyclic maximum flow? Provide a short proof or counter example.

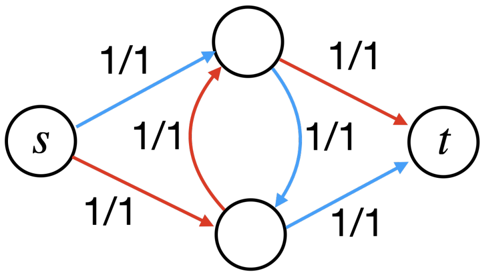
(c) A *path flow* is a simple directed $s$-$t$ path with positive flow on each edge. Prove that every acyclic flow can be written as the sum of at most $m$ path flows.

(d) A *cycle flow* is a simple directed cycle with positive flow on each edge. Prove that every flow can be written as the sum of at most $m$ path and cycle flows.

(e) Describe an algorithm to decompose a flow $f$ into at most $m$ path and cycle flows in $O(mn)$ time.

**Solution:**
(a) Let $C$ be a directed cycle with positive flow in $f$, and let $\Delta = \min_{e\in C} f_e$ denote the minimum amount of flow along any edge of $C$. If we remove $\Delta$ units of flow from each edge $e \in C$, then we eliminate the $C$ (due to our choice of $\Delta$ at least one edge's flow gets set to zero) and we get a new flow $f'$ with strictly fewer cycles. Observe that $f'_e = f_e - \Delta \geq 0$, for each edge $e \in C$. Also, for any vertex $v$ on the cycle, both the flow into $v$ and the flow out of $v$ decrease by $\Delta$. Therefore, $f'$ still obeys capacity and flow conservation constraints, ie., it is a feasible flow. Observe that removing the cycle has no effect on the flow value (flow leaving $s$). Therefore, $f'$ is still a maximum flow. By repeating this process we can eliminate all cycles in $f$ without changing the flow value.

(b) No, consider the example below with two paths: red and blue.



(c) We can decompose the flow $f$ into paths by repeatedly finding *any* $s$-$t$ path $P$ and removing $\Delta = \min_{e\in P} f_e$ units of flow from each edge in $P$. Note that this is just the naive greedy algorithm discussed in lecture 3 used to remove flow instead of adding it. We can find such a path by starting from $s$ and repeatedly following *any* edge out of the current vertex with positive flow. Note that we are guaranteed to reach $t$ since the flow is acyclic. Every simple $s$-$t$ path contains at most $n = |V|$ edges (it passes through every vertex at most once), so finding a path and decreasing the flow on it requires $O(n)$ time. Further, each iteration eliminates flow on at least one edge. Therefore, in $O(m)$ iterations there is no more flow, so $O(nm)$ time in total.

(d) We use essentially the same approach as in part (c), the only difference being that in our search for an $s$-$t$ path we might return a vertex, say $v$, that is in the current path. That is, we found a cycle $C$ that starts at $v$. In this case, we use method described in part (a) to remove the cycle $C$. Otherwise, we find an $s$-$t$ path and use the method of part (c). In either case, we eliminate flow on at least one edge in each iteration, giving a total runtime of $O(mn)$.

(e) As discussed in parts (a), (c), and (d), we can decompose the flow $f$ into at most $m$ paths and cycles in $O(mn)$ time.

## Problem 4

This problem explores a different method to choose augmenting paths to speed up the Ford-Fulkerson algorithm. Recall that the Edmonds-Karp algorithm discussed in lecture #4 selects the shortest augmenting path in the residual graph $G_f$. Suppose that instead, we select the augmenting path on which we can push the most flow. That is, choose the $s$-$t$ path in $G_f$ with the maximum possible minimum residual capacity. (Note this approach was also introduced by Edmonds-Karp.)

(a) Explain how to modify Dijkstra's shortest path algorithm to compute an $s$-$t$ path with the maximum possible minimum residual edge capacity.

(b) Suppose the current flow $f$ has value $F$ and maximum flow value in $G$ is $F^*$. Prove that there is an augmenting path in $G_f$ such that every edge has residual capacity at least $(F^* - F)/m$.
[ Hint: Let $\Delta :=$ the maximum amount of flow that can be pushed on any $s$-$t$ path in $G_f$. Consider the set of vertices reachable from s along paths in $G_f$ with capacity strictly greater than $\Delta$. Relate the value of this $s$-$t$ cut to $F^* - F$. ]

(c) Prove that is variant of Edmonds-Karp terminates within $O(m \log F^*)$ iterations.
[ Hint: Track the algorithm's progress in terms on $F^* - F$ using the bound from (b). The inequality $1 - x \leq e^{-x}$ will be useful. ]

(d) Assume that all edge capacities are integers bounded above by $U$, ie. $c_e \leq U$, $\forall e \in E$. Give an upper bound your algorithm's runtime as a function of $m$, $n$, and $U$.

**Solution.**
(a) Define Bottleneck($v$) as our current estimate of the maximum amount of flow we can send from $s$ to $v$ over a single path in $G_f$. Our modified Dijkstra will greedily pick the unprocessed vertex $u$ with *maximum* Bottleneck, ie., the vertex we can send the *most flow to starting from* $s$, and try to increase the flow we can send to neighboring vertices, say $v$, by sending flow from $s$ to $u$ then to $v$ on the edge $u \to v$. We initialize all estimated Bottleneck capacities at 0 (just assume we cannot send any flow.) The remaining modifications are fairly self-explanatory; an implementation is shown below.

Assuming that we implement our priority queue using a binary max heap, then Insert, GetMax, and IncreaseKey all run in $O(\log n)$ time, where $n = |V|$. Thus, each vertex is popped from the priority queue once, and IncreaseKey is called at most once for each edge. Therefore, the runtime of this modified Dijkstra's algorithm is $O(n + m \log n)$, where $m = |E|$, and $n = |V|$.

(b.) Let $f$ be any flow, and define $F$ as its value and $F^*$ as the value of the maximum flow in $G$. Following the hint, let $\Delta =$ the maximum amount of flow that can be pushed on any $s$-$t$ path in $G_f$. Thus, every $s$-$t$ path in $G_f$ contains at least one edge, say $e$, with capacity $c_e \leq \Delta$, otherwise we would be able to send more flow. Suppose that we run BFS in $G_f$ starting from the source $s$ traveling only on edges with capacity *strictly greater than* $\Delta$. Note that we cannot reach $t$ on such a path due to the definition of $\Delta$. Let $A$ be the set of nodes reached, and $B = V \setminus A$. Then, $(A, B)$ is an $s$-$t$ cut.

```
ModifiedDijkstra($G_f$, $s$, $t$)
    // minimum capacity edge on $s$ to $v$ path for each vertex $v \in V$
    Initialize Bottleneck$[1 \dots n]$ to 0
    // PQ has pairs(vertex $v$, estimated bottleneck capacity to $v$)
    PQ = new PriorityQueue()
    PQ.Insert($s$, $\infty$) // There is no bottle neck edge if we start at $s$
    for each vertex $v \neq s$ do
        PQ.Insert($v$, 0) // guess that we cannot reach $v$
    while PQ is not empty do
        $u \leftarrow$ PQ.GetMax()
        for each edge $u \to v$ in $G_f$ do
            alternatePath $\leftarrow \min\big(\text{Bottleneck}(u),\ c(u \to v)\big)$
            if Bottleneck($v$) < alternatePath then
                Bottleneck($v$) $\leftarrow$ alternatePath
                PQ.IncreaseKey($v$, Bottleneck($v$))
    return Bottleneck($t$)
```

We want to relate the quantity $F^* - F$ to the residual capacity of the edges in the cut $(A, B)$. Since the value of *any* $s$-$t$ flow is less than or equal to the capacity of *any* $s$-$t$ cut:

$$F^* - F \leq \text{capacity}(A, B).$$

Next, we want to upper bound the capacity of the cut. Notice that the residual capacity $c_e \leq \Delta$, $\forall e \in \delta^+(A)$, otherwise we could travel along $e$ in the BFS to reach another vertex, ie. $e$ would not be in the cut. It follows that

$$\text{capacity}(A, B) = \sum_{e \in \delta^+(A)} c_e \leq \sum_{e \in \delta^+(A)} \Delta \leq m\Delta,$$

where $m = |E|$. Substituting this into the previous inequality yields

$$F^* - F \leq \ \text{capacity}(A, B) \leq m\Delta.$$

Rearranging yields the desired result.

(c.) Let $F_t$ = the value of our algorithm's flow $f$ on the $t$'th iteration. We will show by induction that $F^* - F_t \leq F^*(1 - 1/m)^t$. Initially, $F_0 = 0$; so the statement holds for $t = 0$. Assume that this holds for iteration $t$ and consider iteration $t + 1$. Suppose we find the $s$-$t$ path in $G_f$ with maximum minimum residual capacity using our modified Dijkstra algorithm from part (a). Applying the result of part (b), we see that we can send at least $\Delta \geq (F^* - F_t)/m$ units of flow on this path. Thus, $F_{t+1} - F_t \geq (F^* - F_t)/m$. It follows that

$$\begin{aligned} F^* - F_{t+1} &= F^* - F_t - (F_{t+1} - F_t) \\ &\leq (F^* - F_t)(1 - 1/m) \\ &\leq F^*(1 - 1/m)^t(1 - 1/m) = F^*(1 - 1/m)^{t+1}, \end{aligned}$$

where the second inequality uses the induction hypothesis.

After $T$ iterations:

$$F^* - F_T \leq F^*(1 - 1/m)^T < F^* e^{-T/m},$$

where the last step uses the provided inequality $1 - x \leq e^{-x}$. Finally, we let $T = m \log F^*$, to get

$$F^* - F_T < F^* e^{-m \log F^*/m} = 1.$$

Since both the our algorithm's flow and the maximum flow are integer valued, we must have that $F_T = F^*$, ie., it is a maximum flow.

(d) In each iteration, we use our modified Dijkstra's algorithm to find an $s$-$t$ path in $G_f$ with maximum minimum residual capacity, and augment our flow on this path. Dijkstra's runs in $O(m \log n)$ time, and updating the flow takes $O(m)$ time. Thus, $O(m \log n)$ time per iteration. By part (c), the algorithm requires at most $O(m \log F^*)$ iterations. We can bound $F^* \leq O(nU)$, where $U$ is the maximum capacity of any edge. The final runtime is $O(m^2 \log(n) \log(nU))$ time.