Math 182: Problem Set 6
Kenny Guo

Question 1: Suppose G is a network which has weights on vertices as well as edges. Design an efficient algorithm to solve the maximum network flow problem in this new setting.

More formally: you are given a network $G = (V, E, s, t, w)$, as well as a function $c: V \to \mathbb{N}$ assigning a natural number to each vertex. A flow f on G is defined as usual except that it must also satisfy an extra constraint: for any vertex $v \in V$, we must have

$$\sum_{\{u:(u,v)\in E\}} f(u, v) \leq c(v).$$

The size of f is defined as usual; you need to design an algorithm to find a flow of maximum size subject to this extra constraint.

Hint: Modify G to turn this into a normal network flow problem. You may use the standard Ford-Fulkerson algorithm for maximizing network flow without needing to provide details.

Idea: Ignoring s, we simply split each weighted vertex v into unweighted vertices v_{in} and v_{out}, where all edges entering v enter into v_{in} only and all edges exiting v exit out of v_{out} only. We install an edge (v_{in}, v_{out}) with weight $c(v)$ into the new graph, run Ford-Fulkerson, and then the flow on the edges of the original graph is our solution.

```
WeightedVertexFF(G=(V,E,s,t,w,c)):
    f = flow function
    V', E' = empty sets
    w' = weight function

    // Create normal network
    for vertex v in V:
        Create vertices v_in, v_out
        Add v_in, v_out to V'
        Add (v_in, v_out) E'
        Set w'(v_in, v_out) = c(v)
    for edge (u,v) in E:
        Add (u_out, v_in) to E'
        Set w'(u_out, v_in) = w(u,v)
    s' = s_out
    t' = t_out

    f' = FordFulkerson(G=(V', E', s', t', w'))

    // Translate back to original network
    for edge (u,v) in E:
        f(u,v) = f'(u_out, v_in)

    return f
```

This algorithm runs in $O(|V|(|V|+|E|)^2)$ time.

Proof. Creating the new network G' takes $O(|V|+|E|)$ to loop over all vertices and edges and add new vertices and edges into an adjacency list. The new network has $2|V|$ vertices and $|V|+|E|$ edges. Thus, Edmonds-Karp runtime for max-flow is $O(|V|(|V|+|E|)^2)$. Translating back to the original network is $O(|E|)$. Thus, total runtime is $O(|V|(|V|+|E|)^2)$. □

This algorithm produces a max-flow subject to the vertex capacity constraints.

Proof. We show that flows in G and G' correspond to each other.

Consider a flow f in G satisfying edge and vertex capacities. Define a flow f' in G' where for every transformed original edge, $f'(u_{out}, v_{in}) = f(u,v)$ and for every installed edge, $f'(v_{in}, v_{out}) = \sum_{\{u:(u,v)\in E\}} f(u,v) \leq c(v) = w'(v_{in}, v_{out})$, where the inequality follows since f obeys vertex capacities. Thus, f' is a valid flow on G', with the same size as f. Namely, this means the max-flow in G' must be at least that of the max-flow in G.

Now, consider a flow f' in G' satisfying edge capacities. Define a flow f in the original graph by $f(u,v) = f'(u_{out}, v_{in})$. The edge capacities in G are obeyed because each edge (u_{out}, v_{in}) has capacity $w(u,v)$. Also, for every vertex v, all flow entering v_{in} must pass through the edge (v_{in}, v_{out}), whose capacity is $c(v)$. Thus, $\sum_{\{u:(u,v)\in E\}} f(u,v) \leq c(v)$ and f satisfies the vertex-capacity constraint. Thus, f is a valid flow on G, with the same size as f'. Namely, this means the max-flow in G must be at least that of the max-flow in G'. Thus, the max-flows in both graphs are equal, and maximizing the ordinary flow in G' using Ford-Fulkerson also maximizes the flow in the original network subject to the vertex capacities. □

Question 2: Suppose (G, w) is a network such that $w(e) = 1$ for all edges of G. Let k be a positive integer with $k \leq |E|$. Design an efficient algorithm to select k edges so that when these edges are removed the maximum flow in the new graph is as small as possible.

Idea: We use the Max Flow-Min Cut Theorem. Use Ford-Fulkerson to find a max flow, and deduce the minimum cut using the residual graph, whose size must equal that of the max flow. Delete k edges from that cut, or arbitrarily elsewhere if the cut has less than k edges.

```
RemoveEdges(G=(V,E,s,t,w), k):
    f, G_res = FordFulkerson(G)
    A = vertices reachable from s in G_res
    B = V \ A

    C = empty set (crossing edges)
    for edge (u,v) in E:
        if u in A and v in B:
            add (u,v) to C

    R = empty set (removed edges)
    for each edge e in C:
        if |R| < k:
```

```
        add e to R
for each edge e in E:
    if |R| < k and e is not in R:
        add e to R
return R
```

This algorithm runs in $O(|V||E|^2)$ time.

Proof. Running Ford-Fulkerson/Edmonds-Karp is $O(|V||E|^2)$ time. Assembling the cut requires finding all vertices reachable (i.e. using a BFS method), which is $O(|V|+|E|)$ time. Finally, finding the crossing edges and edges to remove are $O(|E|)$ time. $O(|V||E|^2)$ dominates. □

This algorithm chooses k edges that when removed, reduces the size of the max flow as much as possible.

Proof. Let $size(f)$ be the size of the original max flow in G. Thus, by the Max Flow-Min Cut Theorem, the minimum cut has $size(f)$ size. Furthermore, since each edge has weight 1, the minimum cut in fact has $size(f)$ crossing edges.

Thus, removing up to k edges from that cut reduces its size to $\max(0, size(f) - k)$, which is the minimum size possible, and thus, the algorithm produces the lowest possible max flow by Max Flow-Min Cut. □

Question 3: Recall our initial flawed attempt to write a greedy algorithm to solve the max flow problem:

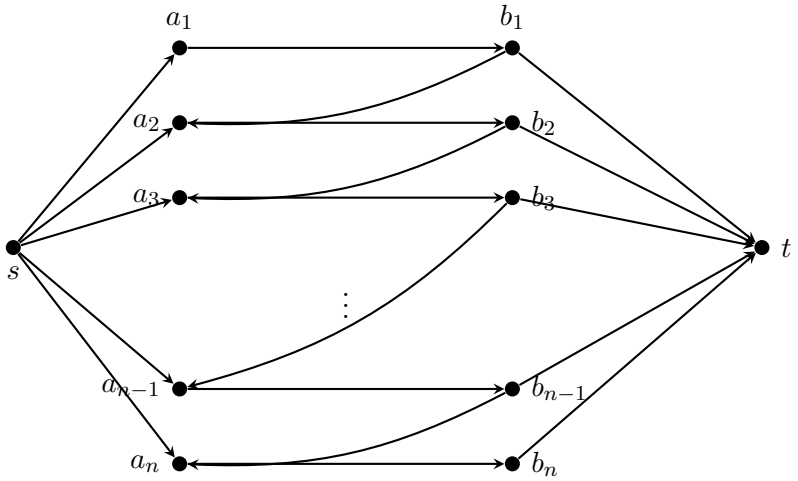
1. Find an $s - t$ path.
2. Push as much flow through as possible.
3. Reduce the edge-weights along the path by the amount of flow we used, deleting an edge when the weight hits zero.
4. Repeat until there are no more $s - t$ paths.

We showed that this will not always produce an optimal graph. But sometimes greedy algorithms that aren't correct will nonetheless always produce a good approximation.

Prove or disprove the following claim: There is a fixed constant $A > 1$ independent of the network and the method used to determine the path) so that the greedy algorithm will always produce a flow with value at least $1/A$ times the actual max-flow value.

The claim is false.

Proof. Suppose for contradiction there is some $A > 1$ such that the claim is true. Choose n such that $n > A$ and construct the following graph with all edges having weight 1:



Then if the greedy algorithm first chooses the path

$$(s, a_1, b_1, a_2, b_2, \ldots, a_n, b_n, t),$$

this shuts down all other $s - t$ paths. The flow produced is 1, but the maximum flow is n (use the cut $A = (s, a_1, \ldots, a_n)$). By our choice of n,

$$\frac{1}{n} < \frac{1}{A},$$

our contradiction. □