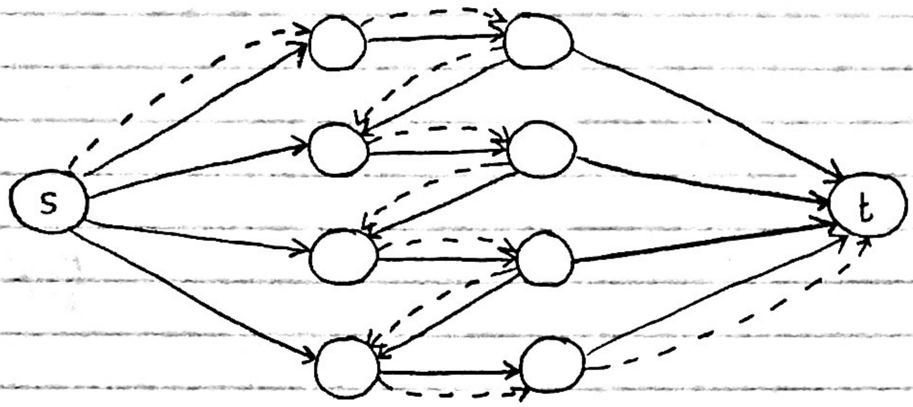
Stewart Dulaney
SID: 904-064-791
CS 180

HW 7

7.11 False. Counterexample:



Suppose all edges have capacity 1. Note that for a graph with 2k vertices besides s and t, the max flow is k.
However, if the path along the dotted lines is the first path chosen by the Forward-Edge-Only (FEO) Alg, it finds a max flow of 1 b/c all potential augmenting paths require a backwards edge. Hence for any k there exists a graph where FEO Alg. find a max flow that is $\frac{1}{k}$ times the optimal max flow.
k depends on the # of vertices and is not an absolute constant.
QED.

7.14 a) We construct a flow network graph $G' = (V', E')$:

- assign capacity 1 to all existing edges between X and S
- add source node s and connect to each node in X w/ a directed edge of capacity 1
- add sink node t and connect each node in S w/ a directed to t of capacity $|X|$ (all routes may end at the same node)

Then calculate max-flow using Ford-Fulkerson Alg. in $O(|E'| \cdot |X|)$.

Claim: max-flow = $|X|$ iff required routes exist

Pf: If required routes exist, max-flow = $|X|$.

Use the $|X|$ capacity 1 edges from s to each of the nodes in X. From there, we have a path from each node in X to some node in S using unique edges. ∴ the $|X|$ units of flow can reach nodes in S from which they have $|X|$ capacity paths to t.

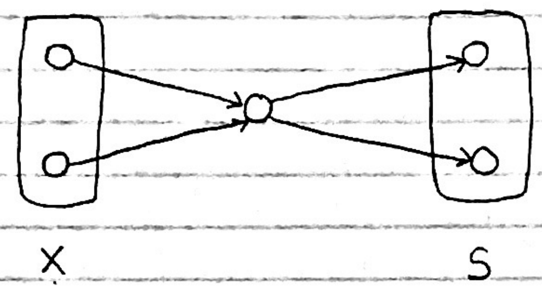
Pf: If max-flow = $|X|$, required routes exist.

If max-flow = $|X|$, each node in X is receiving capacity 1 flow from s. Since all this flow is reaching t and each edge between X and S is capacity 1, all of this flow must be using unique edges. ∴ we have a unique path from every node in X to some node in S.

b) Before constructing the flow network graph, replace each vertex v of G by v_1 and v_2, add an edge between v_1 and v_2, let all in-edges of v point to v_1 and all out-edges of v go out from v_2 as shown below:

s is connected to v_1 of all vertices in X and the v_2 of all vertices in S are connected to t. The capacities are constructed the same as part (a) and we run the same alg. from part (a). This runs in $O(|V| + |E'| \cdot |X|)$ time. Since this gives us paths on unique edges, no two paths can share the "internal edges" e_v. ∴ by "shrinking" the pairs (v_1, v_2) to reverse the 1st step, we obtain paths where no two paths share each vertex v.

Example with yes for part (a) but no for part (b):



Let $n = |V|$, $m = |E|$

7.17 Find-Unreachable-Set (G)

```
Find-Unreachable-Set (G)
    Declare empty set B
    While there is an edge-disjoint s-t path P not considered
        v = Find-First-Unreach (P)
        add v as well as the nodes that come after it to B
    return B
```

```
Find-First-Unreach (P)
    if P has size 1
        return the only vertex
    mid = 1 + (size of P - 1) / 2
    x = ping (mid)
    if x = no s-mid path
        return Find-First-Unreach (s-mid path)
    else
        return Find-First-Unreach (mid-t path)
```

Since there are k edges in a minimum s-t cut (the ones destroyed), by Menger's Thm there are at most k edge-disjoint s-t paths and $\therefore$ the while loop runs at most k iterations. Since each path P contains at most n vertices and Find-First-Unreach uses binary search to locate the first unreachable one, Find-First-Unreach calls ping at most $\log n$ times. $\therefore$ The algorithm Find-Unreachable-Set uses $O(k \log n)$ pings.

Note that preprocessing to compute all s-t disjoint edge paths can be done using Ford-Fulkerson Alg. in $O(mC)$ where $C = \sum_{e \text{ out of } s} c_e$.

Algorithm

7.29 ALG: Let $n = |V|$, $m = |E|$

We can solve this by constructing a directed flow network graph $G = (V, E)$ finding its max-flow, computing the minimum cut (A, B), and returning the set $A - \{s\} = S$

Construction

- Let there be nodes in G labeled v_i for each software application $i \in \{1, \dots, n\}$.
- For each expense x_{ij} let there be two edges (v_i, v_j) and (v_j, v_i) both w/ capacity x_{ij}, except for any edges that originate at v_1 (will never incur this expense)
- Let there be a source node s and connect it to each node v_i w/ a directed edge of capacity b_i
- Let v_1 be the sink node as it has to remain on the old system

Then we run ALG.

Proof of correctness

The goal is to maximize sum of benefits - sum of expenses of moving applications in S. This is just:

$$C - \sum_{v_j \in B} b_j - \sum_{\substack{v_i \in A, \\ v_j \in B}} x_{ij} \quad \text{where} \quad C = \sum_{i=1}^{n} b_i$$

Consider the capacity of the min-cut, $c(A, B)$. Edges that go from A to B must either be from $v_i \in A$ to $v_j \in B$, or from s to $v_j \in B$. Each edge (v_i, v_j) has capacity x_{ij} and each edge (s, v_j) has capacity b_j.

So $c(A, B) = \sum_{v_j \in B} b_j + \sum_{\substack{v_i \in A, \\ v_j \in B}} x_{ij}$ and we've found a minimum $c(A, B)$.

Since C is a constant, our desired quantity is maximized. QED

Time Complexity

compute max-flow Ford-Fulkerson $O(mW)$ where $W = \sum_{e \text{ out of } s} c_e$

compute min-cut $O(m)$

Total $= O(mW + m)$