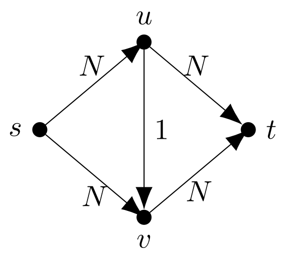
Network flows - First Algorithm

Ford-Fulkerson Algorithm
Input: Network (G, u, s, t).
Output: an s-t-flow f of maximum value

```
1. $f(e) = 0$ for all $e \in E(G)$
2. while $f$-augmenting path $P$ in $G_f$ exists:
3.         compute $\gamma := \min_{e \in E(P)} u_f(e)$
4.         augment $f$ along $P$ by $\gamma$ (as much as possible)
```

1: The algorithm might pick the augmenting path P poorly. At worst, how many iterations might the following network take?



(N is a big integer. Try to trick the algorithm to do many steps by picking unlucky P.)

Solution: Always use the edge uv with capacity 1. The augmentation will be always by one. So $2N$ iterations. The running time depends on u.

2: Show that s-t-flow f is maximum if and only if there is no f-augmenting path. (That is, Ford-Fulkerson algorithm is correct.)

Solution: If there is an augmenting path, then the flow can increase. If there is no augmenting path, then the set of vertices s reachable from s in the reduced graph G_f form a cut. Recall

$$\text{value}(f) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e).$$

Since there is no augmenting path,

$$\sum_{e \in \delta^-(A)} f(e) = 0 \qquad \sum_{e \in \delta^+(A)} f(e) = \sum_{e \in \delta^+(A)} f(e).$$

This proves the maximality of the flow.

The question gives proof to
Theorem (Ford Fulkerson) Maximum value of an s-t-flow equals minimum capacity of an s-t-cut.

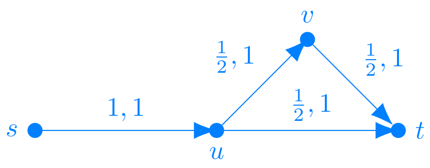
3: If $c : E \to \mathbb{Z}$, is it true that the flow produced from Ford-Fulekrson is integral and that the algorithm finishes in a finite time?

Solution: Yes, the augmenting is always by an integer. Every augmentation raises the value of the flow by at least one. This gives finiteness.

This proves
Theorem Dantzig Fulkerson: If the capacities are integral, then there exists an integral maximum flow.

4: If capacities are integral, is it true that every maximum flow is integral?

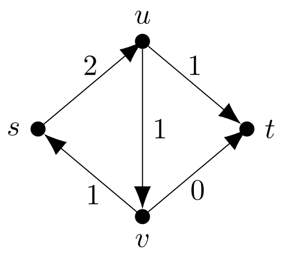
Solution: No - build network that has small cut and 2 paths to/or from it. See the following example with f, u on edges in this order.



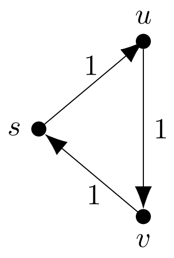
Theorem Gallai; Ford and Fulkerson
Every flow f can be decomposed into s-t-paths $\mathcal{P}$ and circuits $\mathcal{C}$ with weight function $w : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}^+$ such that

- $f(e) = \sum_{e \in P \in \mathcal{P}} w(P) + \sum_{e \in C \in \mathcal{C}} w(C)$
- $value(f) = \sum_{P \in \mathcal{P}} w(P)$.
- $|\mathcal{P} + \mathcal{C}| \leq |E(G)|$.

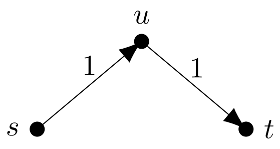
5: Find a decomposition of the following flow into paths and circuits and add find the corresponding weights.



decomposes as



and



Solution: TODO: This needs an update.

6: Prove the theorem. Hint, use induction on number of edges e, where $f(e) > 0$.

Solution: Take the longest circuit or s-t-path W. Assign weight w to W such that $f(h) = w(h)$ for some edge h and for all other edges $f(h) \geq w(h)$. Put W to the collection and apply induction.

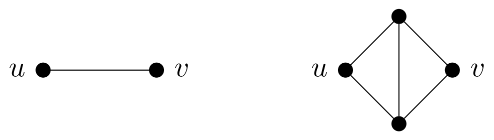
Network flows - Menger's theorem

Theorem Menger: Let G be a graph (directed or undirected), let $s,t \in V(G)$, and $k \in \mathbb{N}$. Then there are k edge-disjoint s-t-paths iff after deleting any $k-1$ edges t is still reachable from s.

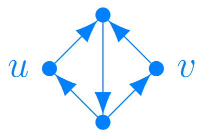
7: Use flows to prove the theorem in directed case.

Solution: Assign capacities one to all edges, find maximum flow and use the previous decomposition theorem. Also notice that the maximum flow is integral.

8: Use the directed case to prove undirected case. Replacing an edge by a pair of opposite directed edges does not work. (Why?) Replace edge uv by a suitable orientation of the following gadget



Solution:



Paths P_1 and P_2 are called *internally disjoint* if they do not share more than the endvertices.

Theorem Menger: Let G be a graph (directed or undirected), let s and t be two non-adjacent vertices, and $k \in \mathbb{N}$. Then there are k pairwise internally disjoint s-t-paths iff after deleting any $k-1$ vertices (distinct from s and t) t is still reachable from s.

9: Prove the directed case. Use the edge case. Hint: Find a gadget for a vertex to prevent using a vertex in multiple paths.

Solution: