CSC373 Week 4 Notes

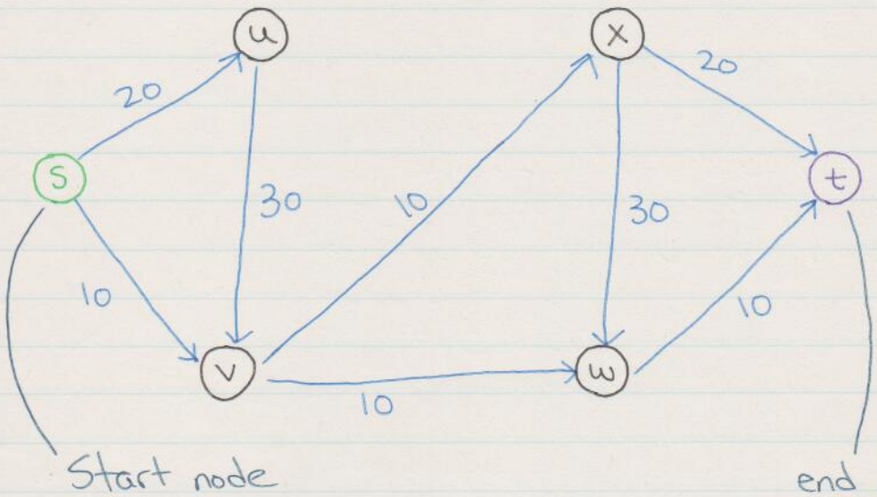
Introduction to Network Flow:

- Input:
  1. A directed graph $G = (V, E)$.
  2. Edge capacities that are non-negative.
  3. A source node, s, and a target node, t.

- Output: Maximum "flow" from s to t.

- Assumptions:
  1. No edges enter s
  2. No edges leave t
  3. Each edge capacity is non-negative (≥ 0)

- E.g.



Note: Treat the edge capacities like the max amount of stuff/weight that can go on the edge at a single time.

- The flow is the amount of stuff/weight carried over an edge.

- The flow is always between 0 and the capacity of the edge, inclusive.

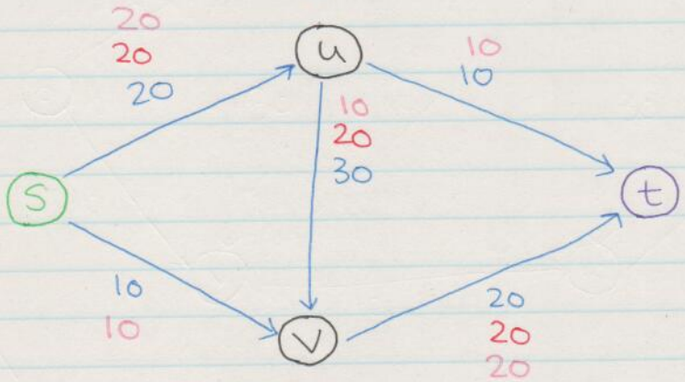
  I.e. $0 \leq f(e) \leq c(e)$, $\forall e \in E$

  ($f(e)$: flow; $c(e)$: edge capacity)

- For all nodes except for s and t, the sum of the flow entering that node equals the sum of the flow leaving it.

  I.e. $\forall v \in V \setminus \{s, t\}$, $\sum_{e \text{ entering } v} f(e) = \sum_{e \text{ leaving } v} f(e)$

- Consider the graph below:



I will use red to denote the numbers for the first example and pink for the second example.

Blue numbers mean the max capacity of that edge.

In the first example, we put 20 on the edge (s,u), then 20 on the edge (u,v) and 20 on the edge (v,t). However this is not the most optimal solution.

Note: The most optimal soln is the min between the total capacity of all the edges flowing from s or the total capacity of all the edges flowing into t.

In my second example, we do get an optimal solution. This is because the total amount flowing into t is 30, which is the same as both the total capacity of edges flowing out of s or the total capacity of edges going into t.

- We can use a residual graph to reverse bad decisions.

- In the example above, the pink numbers (2nd soln) was the residual graph.

Residual Graph:
- Let c(e) denote the capacity for edge e.
- Let f(e) denote the flow of edge e.
- Suppose the current flow is f.
  Let G_F be the residual graph of f.
  G_F has the same vertices as f
  For each edge e=(u,v) in f, G_F has at most 2 edges:
  1. Forward Edge
     - e = (u,v)
     - Capacity = c(e) - f(e)
     - How much more flow we can send on this path

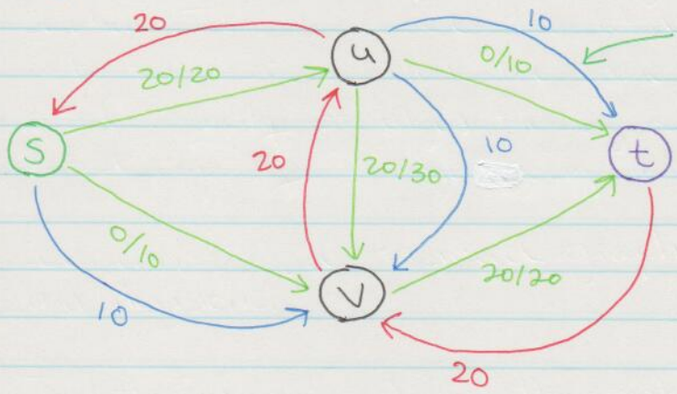
2. Reverse Edge

- $e^{rev} = (v, u)$
- capacity = $f(e)$
- This is the max reverse flow. Since the original flow along edge e is $f(e)$, the max we can send back is $f(e)$.

I.e. Say that the flow from S to A is 3. Then, the reverse flow from A to S is 3. →

Note: We only add edges where the capacity is greater than 0.

- E.g.



The red lines are the reverse edges and the blue lines are the forward edges.

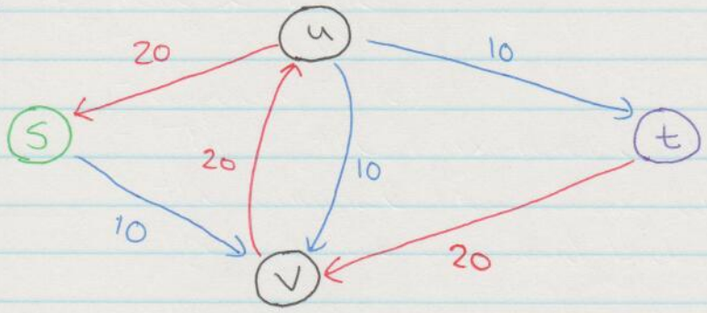
Notice that some edges only have a forward edge or a reverse edge. This is bc the other edge has a capacity of ≤ 0, so we don't add it.

- Let P be an S-T path in G_F.
- let bottleneck(P, f) be the smallest capacity across all edges in P.
- We can augment f by sending bottleneck(P, f) units of flow along P.

Note: Sending x units of flow along P means:

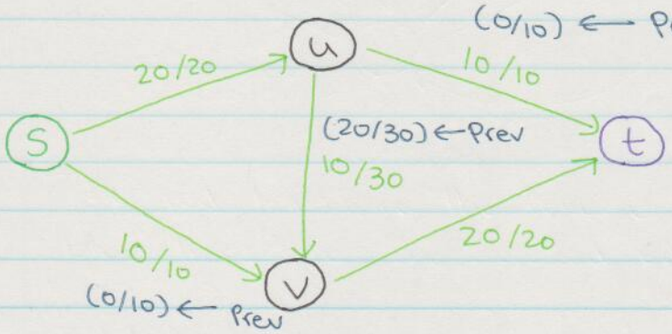
1. For each forward edge in P, inc the flow by x.
2. For each reverse edge in P, dec the flow by x.

- E.g.



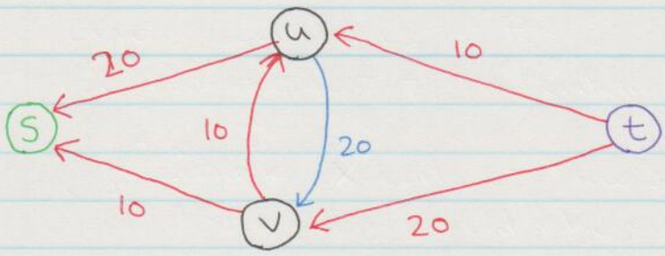
We see that (S,v) and (u,t) are bottlenecks as they have the smallest capacity (10). Hence, we will increase the flow by 10 along all forward edges and decrease the flow by 10 along all reverse edges.

New flow f:



We added/increased the flow from (S,v) and (u,t) by 10 and decreased the flow from (u,v) by 10.

New Residual Graph Gf:



Notice there's no S-T path in Gf anymore.

- Now, we'll prove that the new flow is valid:

1. Capacity Constraints:
   - If we increase flow on e, we can do so by at most the capacity of the forward edge e in G_F which is $c(e) - f(e)$.

     $f(e) + (c(e) - f(e)) = c(e)$ ← Max new flow

   - If we decrease flow on e, we can do so by at most the capacity of the reverse edge e^{rev} in G_F, which is $f(e)$. Hence, the new flow is at least $f(e) - f(e)$ or 0.

2. Flow Conservation:
   - Each node except for s and t has 2 incident edges.

   - If they are both forward/both reverse, that means one is incoming, one is outgoing. The flow is increased on both or decreased on both. The net flow is 0.

   - If one is forward and the other reverse then we get both incoming or both outgoing. Flow is increased on one and decreased on the other. The net flow is 0.

Ford-Fulkerson Algorithm:

```
Max Flow (G):

    // Initalize
    set f(e) = 0 for all e in G

    // while there's an s-t path in GF:
    While P = FindPath(s, t, Residual(G, f)) != None:
        f = augment(f, P)
        UpdateResidual(G, f)

    return f
```

- Running time: $O((m+n) \cdot C)$ (Has $2m$ edges; Has n vertices)

The max flow / max num of augmentations is at most $C = \sum_{e \text{ leaving } s} c(e)$

For path P in G_F, because bottleneck(P, f) ≥ 1, each augmentation increases flow by at least 1. That's why the max flow is at most $\sum_{e \text{ leaving } s} c(e)$.

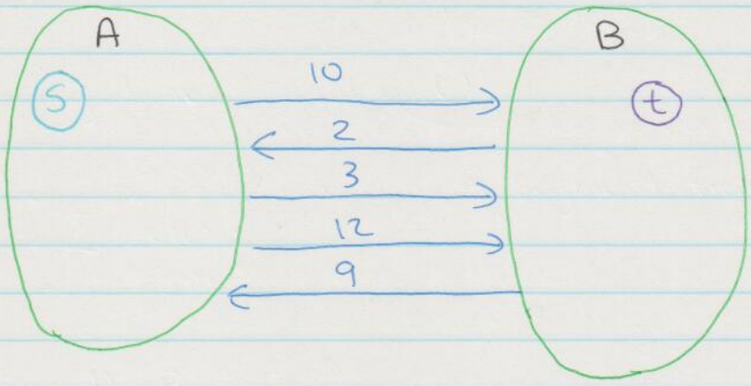
The time it takes to perform an augmentation is $O(m+n)$. This is bc G_F has n vertices and at most $2m$ edges. So finding P, computing bottleneck(P, f) and updating G_f takes $O(m+n)$ time.

Proof of Optimality:

- Let (A, B) be an s-t cut. This means we partition the nodes into 2 groups, A and B. Furthermore, $s \in A$ and $t \in B$.
  $A \cup B = V$
  $A \cap B = \emptyset$

- Let $cap(A, B)$ be the sum of the capacities of the edges leaving A.

- E.g.



$$Cap(A, B) = 10 + 3 + 12$$
$$= 25$$

- Thm: For any flow f and any s-t cut (A, B),
  $v(f) = f^{out}(A) - f^{in}(A)$

Note: $v(f) = f^{out}(s) = f^{in}(t)$

Only edges that connect nodes in both partitions can deliver the flow to t.

Proof:

Edges that connect nodes in the same partition cancel each other out.

I.e. Consider nodes x and y s.t. they're in the same partition. Suppose the flow from x to y is e. That means y has to "release" e amount of flow. If it's to other nodes in the same partition, it doesn't directly lead to t, so it cancels out.

Assume only x goes into y. →

So, we're only interested in the edges that span both partitions.

Therefore, we need to sum the capacity of the edges that go from A to B and subtract the capacity of the edges that go from B to A.

We need to do the subtraction because the edges that go from B to A "return" some of the flow.

- Thm: For any flow f and any s-t cut (A,B), $v(f) \leq cap(a,b)$

Proof:

$v(f) = f^{out}(A) - f^{in}(B)$

$\leq f^{out}(A)$

$= \sum_{e \text{ leaving } A} f(e)$

$\leq \sum_{e \text{ leaving } A} c(e)$

$= cap(A,B)$

Hence, $\max_f v(f) \leq \min_{(A,B)} cap(A,B)$

I.e. The max value of any flow ≤ min capacity of any s-t cut

Implications:

1. Max flow = Min cut
2. Ford-Fulkerson generates max flow

- Thm: Ford-Fulkerson returns / finds the max flow

Proof:
f = flow returned by F-F
A^* = nodes reachable from s in G_F
$B^* = V \setminus A^*$
Note: We look at the residual graph G_F but define the cut in G
Note: The F-F algo terminates when there are no paths from s to t. (See pg. 7)

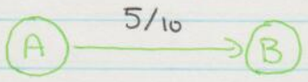
Claim: (A^*, B^*) is a valid cut.
$s \in A^*$ by definition
$t \in B^*$ because there's no path from s to t in G_F. Hence, $t \notin A^*$

Consider the following:
1. Each edge (u,v) going out of A^* in G must be Saturated ($c(e) = f(e)$). Otherwise, G_F would have its own forward edge (u,v) and $v \in A^*$.

   Recall: If edge (u,v) in G has a flow less than its capacity, then in G_F, there will be a forward edge with capacity $c(e) - f(e)$.

2. Each edge (v,u) coming into A^* in G must have 0 flow. Otherwise G_F would have its reverse edge (u,v) and then $v \in A^*$.

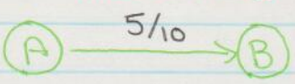
E.g. Suppose $A \in A^*$ and $B \in B^*$



G

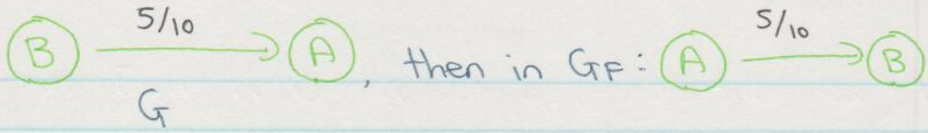
, then in G_F:

Forward Edge



This means we can reach B from s, which means $B \in A^*$, contradicting our

Reverse Edge



This means that we can reach B from S, contradicting our assumption.

Based on 1 and 2, we conclude:

$f^{out}(A^*) = cap(A^*, B^*)$

$f^{in}(A^*) = 0$

$\therefore v(f) = f^{out}(A^*) - f^{in}(A^*)$

$= cap(A^*, B^*)$

Edmonds-Karp Algo:

- A modification on Ford-Fulkerson algo.
- Instead of finding any s-t path in Gf, like the Ford-Fulkerson algo does, it finds the shortest s-t path.
- Algo:

```
MaxFlow(G):
// Initalize
Set f(e) = 0  ∀e∈G

// Find the shortest s-t path in GF and augment
while P = BFS(s,t, Residual(G,f)) != None:
    f = augment(f, P)
    UpdateResidual(G, f)

return f
```

- Let $d(v)$ be the shortest dist to node v from s in the residual graph G_f.

- Lemma 1: During the execution of the algo, $d(v)$ does not decrease for any v.

Proof:
Suppose the augmentation $f \to f'$ dec $d(v)$ for some v.
Choose the v with the smallest $d(v)$ in $G_{f'}$.
Say $d(v) = k$ in $G_{f'}$ and so, $d(v) \geq k+1$ in G_f.
Note: G_f is the residual graph for f and $G_{f'}$ is the residual graph for f'.

G_f is the original graph.
$G_{f'}$ is its augmented graph.

Recall that f', the augmented graph of f, is created by finding the bottleneck in the s-t path in G_f and then sending that many units along that path. (See pg 4)

Let node u be the node just before v on the shortest path $s \to v$ in $G_{f'}$.
Since $d(v) = k$ in $G_{f'}$, $d(u) = k-1$ in $G_{f'}$.
Furthermore, since $d(u)$ didn't decrease, $d(u) \leq k-1$ in G_f.
Note: Since we start with G_f and get $G_{f'}$ later, by the lemma, $d(u) \geq d(u')$. However, because of our assumption, $d(u) < d(u')$.

Here's a table to summarize the values:

	$d(u)$	$d(v)$	
G_f	$\leq k-1$	$\geq k+1$	Before
$G_{f'}$	$k-1$	k	After

However, in G_f, (u,v) must be missing.
This is because in G_f, $d(u) \leq k-1$ and $d(v) \geq k+1$.
If there is an edge (u,v), then
$d(v) = d(u)+1$, but it doesn't.
Since (u,v) doesn't exist in G_f, but exists in G_f, we must have added it by selecting (v,u) in the augmented path P. However, P is a shortest path so it cannot have an edge (v,u) with $d(v) > d(u)$.

Note: Consider $d(t)$. We know that it will always increase as the execution continues and that eventually, $d(t) = \infty$ because there is no path from s to t.

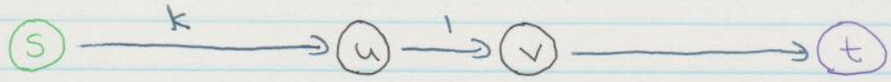
- Let edge (u,v) be critical in an augmentation step if it's part of the augmented path P and its capacity is equal to bottleneck(P,f) and the augmentation step removes e and adds e^{rev}, if it's missing.

- Lemma 2: Between any 2 steps in which (u,v) is critical, $d(u)$ increases by at least 2.

Proof:
If (u,v) is critical, then that means its capacity is the lowest along path P. Furthermore, during the augmentation step, we send that many units of flow down path P and therefore, (u,v) is saturated. Hence, the residual graph will remove (u,v) and replace it with (v,u).

Suppose that $d(u) = k$ in G_f. Since (u,v) is a shortest path, $d(v) = k+1$ in G_f.

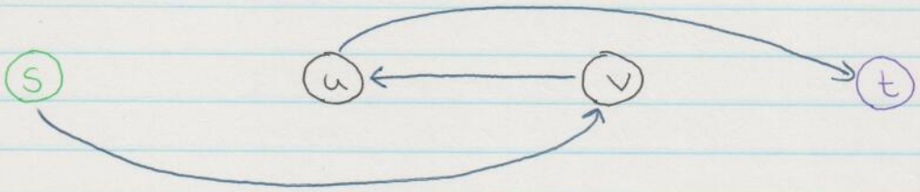
I.e.



For (u,v) to be critical a second time, it must have been added back at some point.

Since the residual graph removed (u,v) and added (v,u) in its place, to get (u,v) back, the augmented path must have selected (v,u).

I.e.



This path is still a shortest path.

Hence, in G_f', $d(u) = d(v) + 1$ ← The +1 comes from (v,u).
$\geq (k+1) + 1$
$\geq k+2$

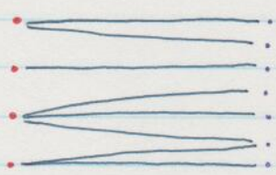
- Proof of Running Time:
- Each $d(v)$ can go from 0 to n. (Lemma 1)
- So, each edge (u,v) can be critical at most $\frac{n}{2}$ times. (Lemma 2)
- So, there can be at most $\frac{mn}{2}$ augmentation steps.
- Since each augmentation takes $O(m)$ time, there are $O(m^2n)$ operations in total.

- Applications of Network Flow:
- Integral Theorem: If edge capacities are integers, then the max-flow computed by Ford-Fulkerson and its variants are also integral, meaning that the flow on each edge is an integer.

- Bipartite Matching:
  - Problem: Given a bipartite graph $G = (U \cup V, E)$, find a max cardinality matching.

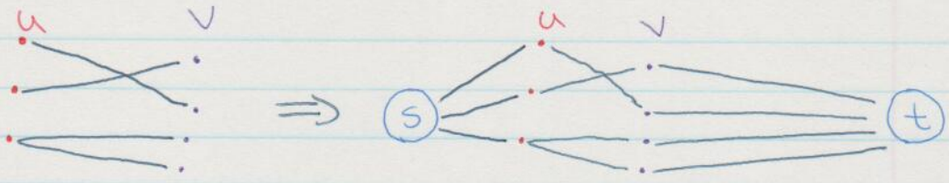
    Note: A bipartite graph is a set of vertices decomposed into 2 disjoint sets s.t. no 2 graph vertices within the same set are adjacent.

    E.g.



    Note: A matching is a set of pairwise non-adjacent edges s.t. no 2 edges share common vertices.

  - Consider this:



    We can "reduce" this problem to max flow by adding a starting node that connects to each node in u and an end node that connects to each node from v.

- Furthermore, let the capacity of each edge be 1. This way, we can "mimic" choosing an edge as 1 and not choosing an edge as 0.

- Note: There is a 1-1 correspondence btwn matchings of size k in the original graph and flows with value k in the corresponding flow network.

Proof:
Note: Similar to proving an iff, we must prove both directions.

Matching $\Rightarrow$ Flow:
Take a matching M, $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ of size k.
Construct the corresponding flow f_M where:
- Edges $s \to u_i \to v_i \to t$ have flow 1, $\forall i \in 1 \ldots k$
- All other edges have flow 0.
Hence, the total flow is k.

Flow $\Rightarrow$ Matching:
Take any flow f with value k.
The corresponding unique matching M_f equals to the set of edges from U to V with a flow of 1.
Since k units of flow come from s, it must go into k vertices in U and that will go to k distinct vertices in V and finally, all k flows will go to t.

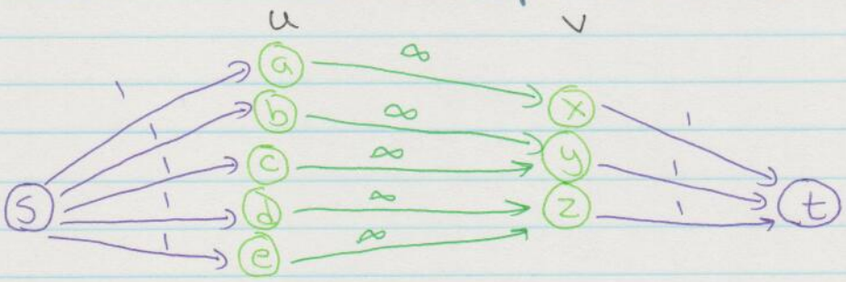
- Note: A perfect matching occurs when we have a flow with value n where $n = |U| = |V|$

A perfect matching is a matching that covers every vertex of the graph.

- Hall's Marriage Thm:
- We know that a bipartite graph has a perfect matching when the corresponding flow network has a value of n.
- However, can we interpret this condition in terms of edges of the original bipartite graph? → yes we can
- For $S \subseteq U$, let $N(S) \subseteq V$ be the set of nodes that is in V and is adjacent to some node in S.
- Observation: If G has a perfect matching, then $|N(S)| \geq |S|$ for each $S \subseteq U$. This is bc each node in S must be matched to a distinct node in N(S).
- We'll consider a slightly different network flow, which is still equivalent to bipartite matching. Now, all U→V edges have ∞ capacity. All s→U and V→t edges still have a capacity of 1.
- Hall's Thm: G has a perfect matching iff $|N(S)| \geq |S|$ for each $S \subseteq U$.

Proof (Reverse Direction):

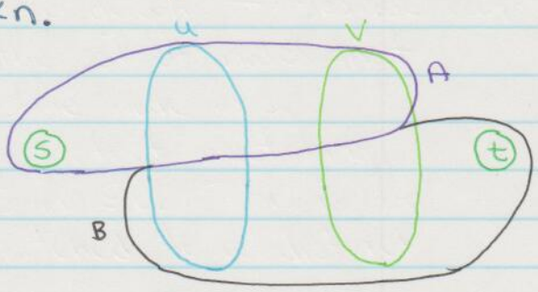
First, I'll draw an example of our network.



Note: We want to prove: if $|N(S)| \geq |S|$ for each $S \subseteq U$, then G has a perfect matching."

We will prove the contrapositive: If G does not have a perfect matching then for some $S \subseteq U$, $|N(S)| < |S|$.

Suppose that G doesn't have a perfect matching. Then max-flow = min-cut $< n$.
Let (A, B) be a min-cut →
Note: There can't be any edges going from $U \cap A$ to $V \cap B$. This is because, since the capacity of all (u,v) edges are infinity, if such an edge exists, then $cap(A,B) = \infty$, but we know that the maxflow $< n$.



$$\begin{aligned} Cap(A,B) &= |U \cap B| + |V \cap A| \\ &< n \\ &= |U| \end{aligned}$$

$$\left.\begin{aligned} |U \cap B| + |V \cap A| &< |U| \\ |V \cap A| &< |U| - |U \cap B| \\ &= |U \cap A| \end{aligned}\right\} \; |V \cap A| < |U \cap A|$$

Furthermore, since we know that no edges between $(U \cap A)$ and $(V \cap B)$ exist (see above), we know that: $N(U \cap A) \subseteq V \cap A$ and that $|N(U \cap A)| \leq |V \cap A|$.

Hence, $$\begin{aligned} |N(U \cap A)| &\leq |V \cap A| \\ &< |U \cap A| \end{aligned}$$

- Edge-Disjoint Paths:
- Problem: Given a directed graph $G = (V, E)$, 2 nodes s and t, find the max number of edge-disjoint $s \to t$ paths.
  Note: 2 $s \to t$ paths P and P' are edge-disjoint if they don't share an edge.
- Soln: Assign 1 as the capacity for all edges
- Thm: There is a 1-1 correspondence btwn sets of k edge-disjoint s-t paths and integral flows of value k.

Proof:

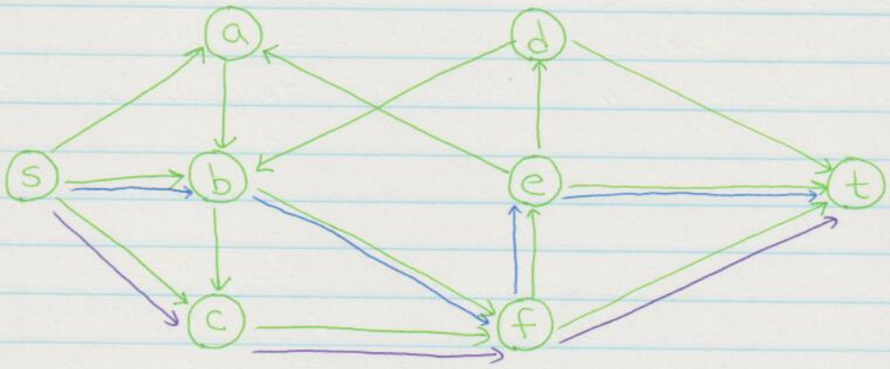
1. Path → Flow:
   Let $\{P_1, \ldots, P_k\}$ be a set of k edge-disjoint s-t paths.
   Define flow f where $f(e) = 1$ whenever $e \in P_i$ for some i and 0 otherwise.
   Since paths are edge-disjoint, flow conservation and capacity constraints are fulfilled/satisfied.
   There is a unique integral flow of value k.

2. Flow → Path:
   Let f be an integral flow of value k.
   That means that there are k outgoing edges, all of which have unit flow.
   Pick 1 edge (s, v_i):
   - By flow conservation, v_i must have an outgoing flow of value 1, that we haven't used yet.
   - Pick such an edge and build a path until we reach t.

   Repeat for all other $k-1$ edges.

- Menger's Thm: In any graph, regardless if it's directed or undirected, the max number of edge-disjoint s-t paths equal the min number of edges whose removal disconnects s and t.

E.g. Consider the graph below



The blue and purple lines each represent a s-t path and they're edge disjoint.

If we remove the edges (b, f) and (c, f), we disconnect s and t.

Notice that the number of edge-disjoint s-t paths equals to the min number of edges removed to disconnect s and t.

- Multiple Source / Sinks:
- Problem: Given a directed graph $G=(V,E)$ with edge capacities $c: E \to N$, sources $s_1, \ldots, s_k$ and sinks $t_1, \ldots, t_\ell$, find the max total flow from sources to sinks.

- Soln:
  - Add a new source s and connect s with each node $s_1, \ldots, s_k$ s.t. the edges have infinite capacity.
  - Add a new sink t s.t. t is connected with each node $t_1, \ldots, t_\ell$ and the edges all have infinite capacity.
  - Find the max flow from s to t.

- Circulation:
- Input:
  - Directed graph $G=(V,E)$
  - Edge capacities $c: E \to N$
  - Node demands $d: V \to Z$

- Output:
  - Some circulation $f: E \to N$ satisfying:
    1. For each $e \in E$: $0 \leq f(e) \leq c(e)$
    2. For each $v \in V$: $\sum_{e \text{ entering } v} f(v) - \sum_{e \text{ leaving } v} f(v) = d(v)$

    $d(v)$ equals to the sum of the incoming flows - the sum of the outgoing flows.

    Note: We need $\sum_{v: d(v)>0} d(v) = -\sum_{v: d(v)<0} d(v)$

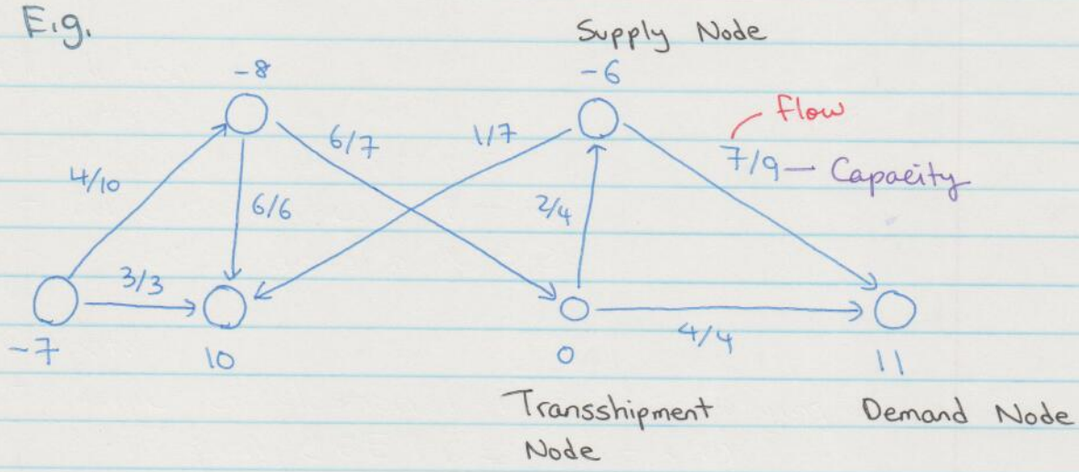
- Demand at v, $d(v)$, is the amount of flow you need to take out at node v.

- If $d(v) > 0$:
  - There should be more incoming flow than outgoing flow.
  - This is a demand node.

- If $d(v) < 0$:
  - There should be more outgoing flow than incoming flow.
  - This is a supply node.

- If $d(v) = 0$:
  - Equal outgoing and incoming flows.
  - Transshipment node

- E.g.



Note:

1. Each $d(v)$ equals to the sum of the incoming flow(s) minus the sum of the outgoing flow(s).

2. The sum of $d(v)$'s > 0 equal the sum of the $d(v)$'s < 0 times -1.

$$\sum_{d(v)>0} d(v) = 10 + 11 = 21 \qquad -\sum_{d(v)<0} d(v) = -(-7-8-6) = -(-21) = 21$$

- Soln:
  - Create a start node s and a sink node T.
  - Connect s with each supply node v, such that the edge (s,v) has capacity $-d(v)$.
  - Connect T with each demand node v, s.t. the edge (v, T) has capacity $d(v)$.
  - Claim: G has a circulation iff G' has a max flow of value

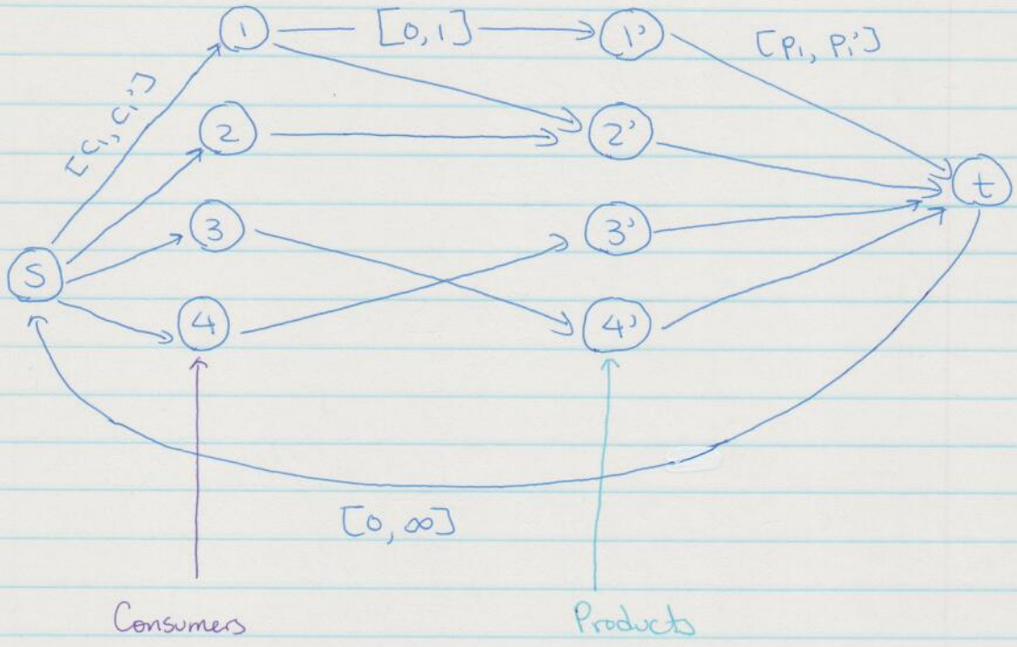
$$\sum_{v: d(v)>0} d(v) = -\sum_{v: d(v)<0} d(v)$$

- Survey Design:
  - Problem: We want to design a survey about m products. We have 1 question per product and we need to ask product j's question to btwn p_j and p'_j consumers.

    There are a ~~total~~ of n consumers and each consumer i owns a subset of products O_i. We can only ask consumer i questions about these products and we want to ask each consumer i between c_i and c'_i questions.

  - Soln:
    - Create a network with special nodes s and t.
    - Create an edge from s to each consumer i with flow $\in [c_i, c'_i]$.
    - Create an edge from each consumer i to each product $j \in O_i$ with flow $\in [0,1]$.
    - Create an edge from each product j to t with flow $\in [p_j, p'_j]$.
    - Create an edge from t to s with flow $\in [0, \infty]$. This is to ensure that the flow coming into t equals the flow leaving s.

- E.g.



- Profit Maximization:
- Problem: Given n tasks, each of which generates some profit (and the profit could be 0 or negative) and a set E of precedence relations, we want to find a subset of tasks, S, that is subject to the precedence constraints s.t. profit(S) is maximized.

Note: If $(i,j) \in E$, that means we have to perform job/task i before we can perform task j. This is the precedence relation/predence constraint.

- Soln:
- We can represent this problem as a network flow problem.

We will represent the input as a graph:

1. Tasks are nodes
2. Profits are node weights
3. Precedence constraints are edges.
   Note: The capacity of all edges s.t. the edges are not connected to s or t will be infinite. This is because when we take the min-cut, we don't want node i to be on one side and node j to be on the other side if we have the precedence constraint (i, j).

Furthermore, we will have a start node, s, and a sink node, t, and connect s and t with the other nodes like this:

1. If P_i, the profit of node i, is positive, we connect node i to s and the capacity of $(s, i) = P_i$.

2. If P_i is negative, then we connect node i to t and the capacity of edge (i, t) will be $-P_i$.

- If $cap(A, B)$ is finite, then $A \setminus \{s\}$ is a valid soln.
- Min $cap(A, B)$ max $profit(A \setminus \{s\})$
- $cap(A, B) = \sum_{i \in \text{positive nodes}} P_i - profit(A \setminus \{s\})$.