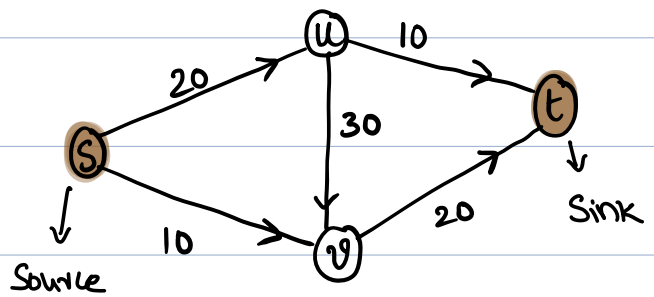
[Reference: Chapter 7 of Kleinberg & Tardos]
+
Coreman ...

Flow Networks:

A flow network is a directed graph $G = (V, E)$ with the following features.

- Associated with each edge e is a capacity which is a non-negative number c_e
- There is a single Source node $s \in V$
- There is a single Sink node $t \in V$

Example of a flow network:



@ The numbers next to the edges are the capacities

Assumptions

- No edge enters the Source s and no edge leaves the Sink t.
- All capacities are integers.
- Each vertex lies on some path from the source to the Sink.

  ie, for each vertex v, $s \rightsquigarrow v \rightsquigarrow t$

  $\therefore$ # of edges $\geq n-1$

- If $uv \in E(G)$ then $vu \notin E(G)$

  (we shall see how to work around this restriction) (Simply add artificial vertices)

- If $e = uv \notin E$ then $c_e = 0$

Defining flow:

We say that an s-t flow is a function f that maps each edge e to a non-negative real number, $f : E \longrightarrow \mathbb{R}^+$. The value $f(e)$ intuitively represents the amount of flow carried by edge e.

A flow f must satisfy the following two properties

(i) (Capacity conditions/constraint)

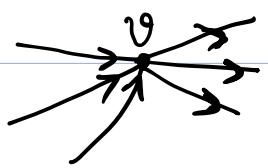
For each $e \in E$, we have $0 \leq f(e) \leq c_e$

(ii) (flow conservation condition)

For each node v other than s and t, we have

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Flow entering v → flow leaving v.



(or)

$$\sum_{u \in V} f(uv) = \sum_{u \in V} f(vu)$$

By our assumption, the source s has no incoming edges, but it is allowed to have flow going out. Similary the sink has no outgoing edges, but it is allowed to have flow coming in.

Value of a flow:

The value of a flow f, denoted $v(f)$ is defined to be the amount of flow generated at the source.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

Notation: $f^{out}(v) = \sum_{e \text{ out of } v} f(e)$

$$f^{in}(v) = \sum_{e \text{ into } v} f(e)$$

$S \subseteq V$

$$f^{out}(S) = \sum_{e \text{ out of } S} f(e)$$

$$f^{in}(S) = \sum_{e \text{ into } S} f(e)$$

For all nodes $v \neq s, t$, $f^{in}(v) = f^{out}(v)$

$$v(f) = f^{out}(s).$$

The Maximum-flow Problem

Given a flow network G with source s and sink t

Find a flow of maximum value.

Example

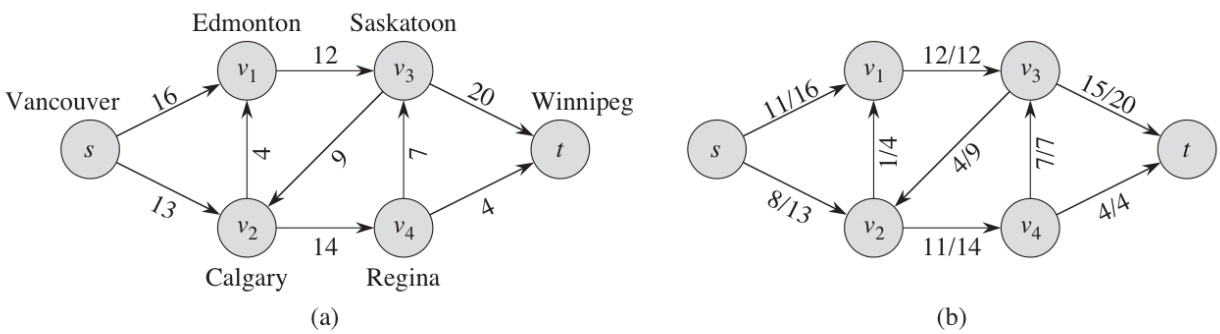
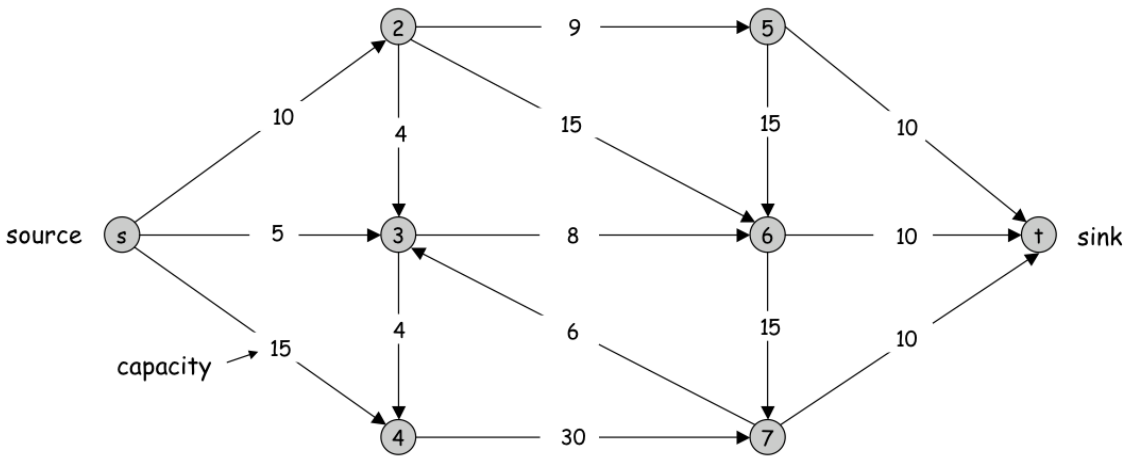


Figure 26.1 **(a)** A flow network $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s, and the Winnipeg warehouse is the sink t. The company ships pucks through intermediate cities, but only $c(u, v)$ crates per day can go from city u to city v. Each edge is labeled with its capacity. **(b)** A flow f in G with value $|f| = 19$. Each edge (u, v) is labeled by $f(u, v)/c(u, v)$. The slash notation merely separates the flow and capacity; it does not indicate division.

Value of the flow $= \sum_{e \text{ out of } s} f(e)$

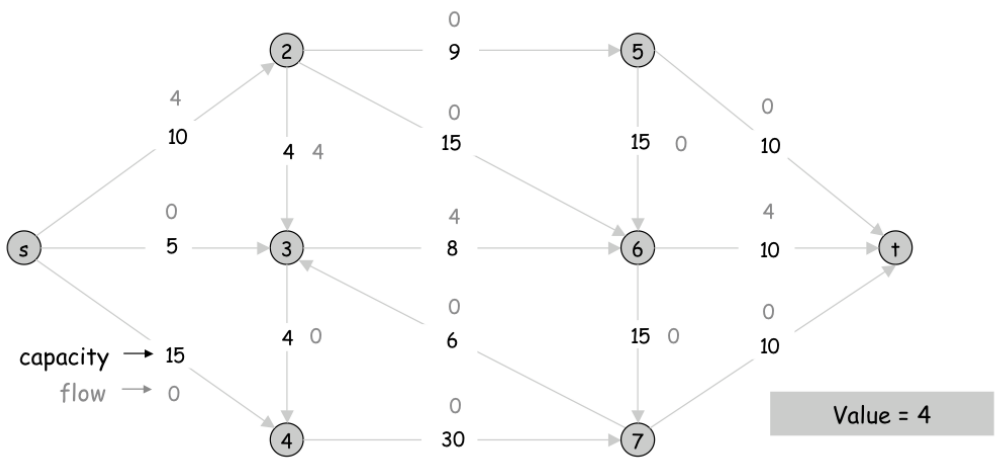
$= 11 + 8 = 19$

Warmup



4

Q1 What is the value of the flow?



8

Anti Parallel edges

We call two edges uv and vu antiparallel

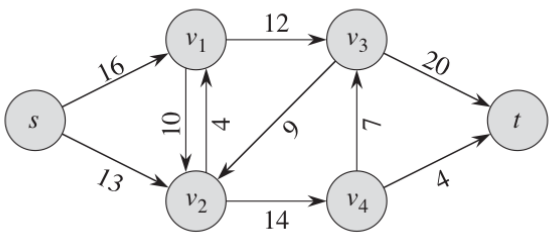
Our earlier assumption

Network has no anti Parallel edges.

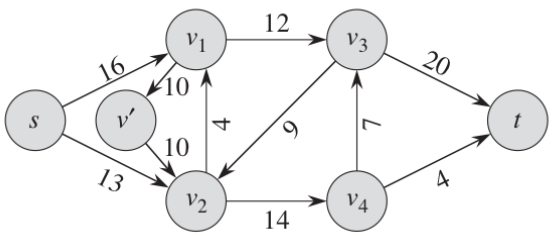
Q Given a flow network with antiparallel edges. How do we transform the network into an equivalent one containing no anti Parallel edges?

Ans Next Page (example)

Example



(a)



(b)

v_1v_2, v_2v_1

antiParallel

Networks with multiple Sources and Sinks :

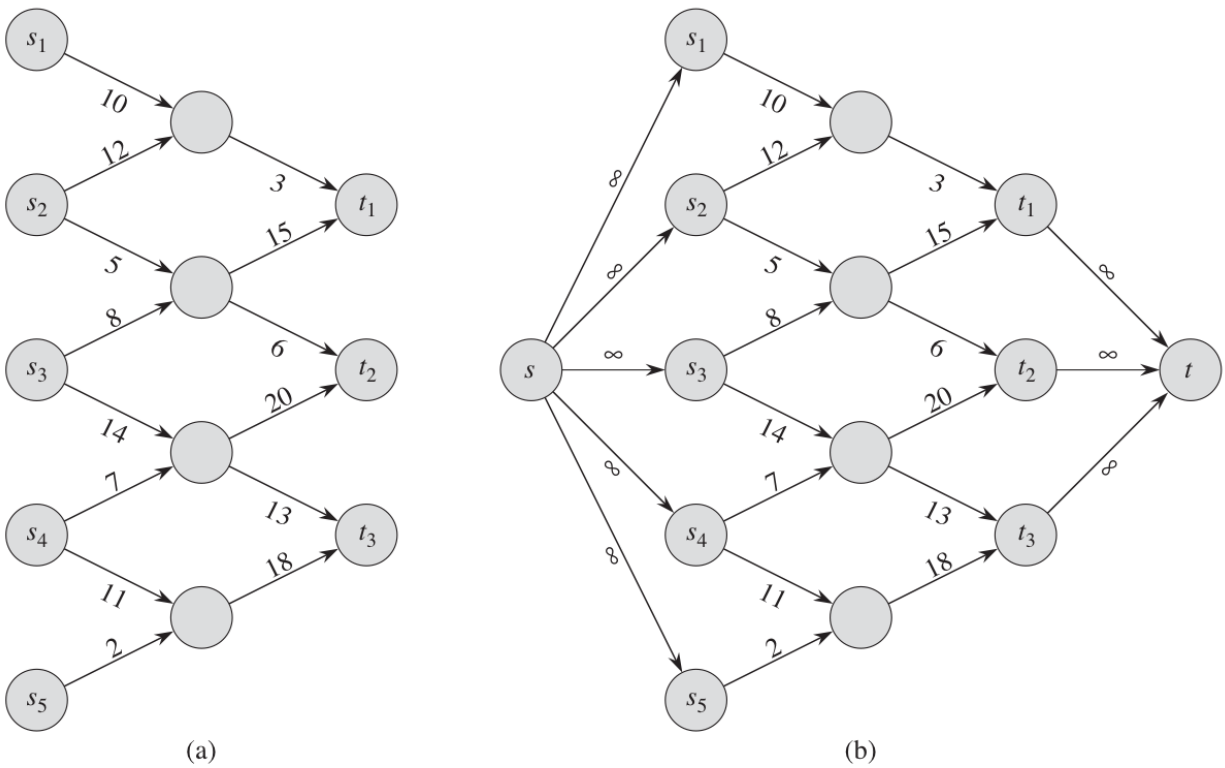


Figure 26.3 Converting a multiple-source, multiple-sink maximum-flow problem into a problem with a single source and a single sink. **(a)** A flow network with five sources $S = \{s_1, s_2, s_3, s_4, s_5\}$ and three sinks $T = \{t_1, t_2, t_3\}$. **(b)** An equivalent single-source, single-sink flow network. We add a supersource s and an edge with infinite capacity from s to each of the multiple sources. We also add a supersink t and an edge with infinite capacity from each of the multiple sinks to t.

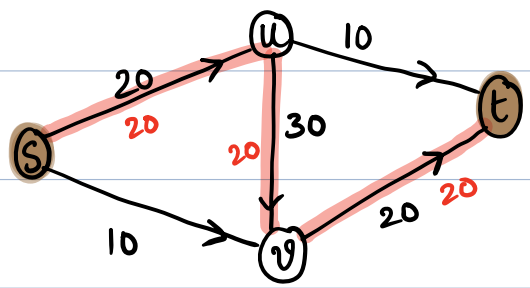
Overview of the Algorithm (Ford-Fulkerson)

Start with zero flow, i.e, $f(e)=0$ for all e.

Clearly this respects the capacity and conservation conditions.

We now try to increase the value of f by Pushing flow along a Path from s to t upto the limits imposed by the edge capacities.

Ex:



we select a s-t Path with edges $\{su, uv, vt\}$ and increase the flow on each edges to 20 and leave $f(e)=0$ for the other two edges.

Clearly this also respects the capacity and conservation conditions.

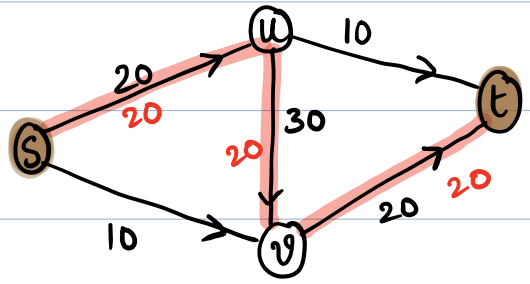
Now the value of the flow is 20.

Q Is this the maximum possible flow value for the above graph?

Ans: NO, maximum flow = 30

We are now stuck, as there no s-t path on which we can directly push flow without exceeding some capacity, also we do not have a maximum flow.

We have to find a way to increase the value of the current flow.



Ideally we want to Push 10 units of flow on the edge sv and reduce 10 units of flow on the edge uv and Push 10 units of flow on the edge ut.

The Residual Graph:

Given a flow network G, and a flow f on G, we define the residual graph G_f of G with respect to f as follows.

- The vertex set of G_f is same as that of G.
- For each edge $e = uv$ of G on which $f(e) < c_e$, there are $c_e - f(e)$ "left over" units of capacity on which we could try pushing flow forward. So we include the edge $e = uv$ in G_f with capacity $c_e - f(e)$. [forward edges]
- For each edge $e = uv$ of G on which $f(e) > 0$, we can undo it needed to, by pushing flow backward.
  So, we include the edge $e = vu$ in G_f with capacity f_e. [Backward edges]

ie, for all $u, v \in V$

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

ie, $G_f = (V_f, E_f)$

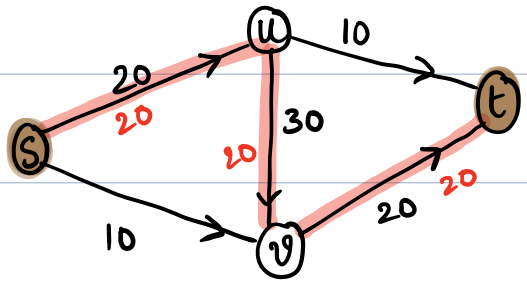
where $V_f = V(G)$

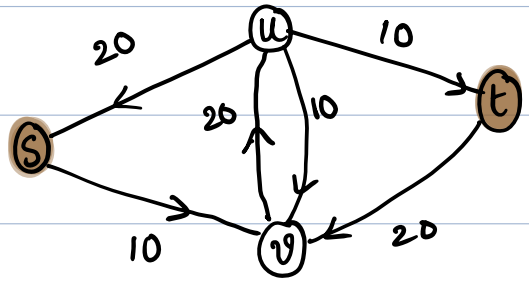
$$E_f = \{ (u,v) \in V \times V : c_f(u,v) > 0 \}$$

* Notice that each edge e in G can give rise to one or two edges in G_f: If $0 < f(e) < c_e$ it results in both a forward edge & a backward edge being included in G_f.

* We call the capacity of an edge in the residual graph as a residual capacity.

Example:





G_f

Terminology: Any s-t Path in the residual graph is refered as an augmenting Path.

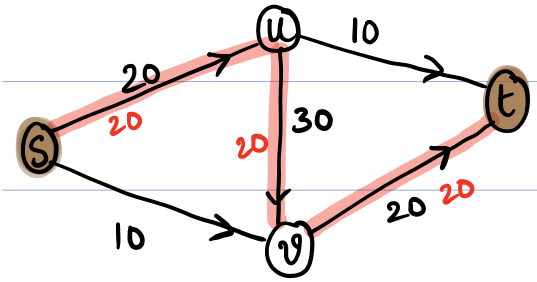
In the graph G_f, $s - v - u - t$ is an augmenting Path.

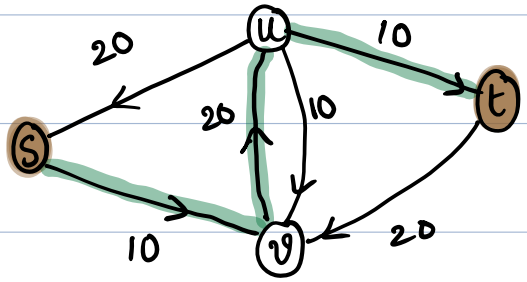
Augmenting Paths in a Residual graph:

let P be a simple s-t Path in G_f.

define bottleneck (P, f) = minimum residual capacity of any edge on P, with respect to the flow f.

```
augment(f, P)
  Let b = bottleneck(P, f)
  For each edge (u, v) ∈ P
    If e = (u, v) is a forward edge then
      increase f(e) in G by b
    else (u,v) is a backward edge and let e = (v,u)
      decrease f(e) in G by b
    Endif
  Endfor
  Return(f)
```

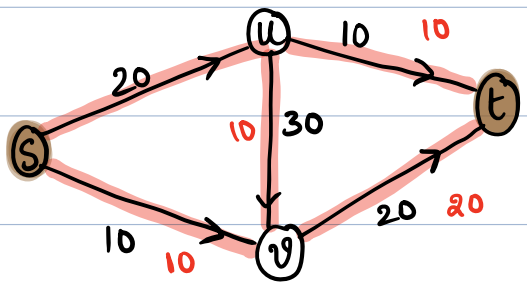




G_f

P: s, v, u, t

bottleneck(f, P) = 10



augment(P, f)

The result of augment (f, P) is a new flow f' in G_f, obtained by increasing & decreasing the flow values on edges of P.

Lemma: f' is a flow in G_f.

Proof:- We need to show that f' satisfies capacity and conservation conditions.

(i) Capacity conditions

As f' differs from f on edges of P, we need to check the capacity conditions only on these edges.

if $e = uv$ be an edge of P.

if e is a forward edge then its residual capacity $c_e - f_e$, hence we have

$$0 \leq f(e) \leq f'(e) \leq f(e) + \text{bottleneck}(P, f)$$

$$\leq f(e) + c_e - f(e) = c_e$$

if $e = uv$ is a backward edge, arising from edge $e = vu$, then its residual capacity is $f(e)$, hence we have

$$c_e \geq f(e) \geq f'(e) = f(e) - bottleneck(P, f) \geq f(e) - f(e) = 0$$

(ii) Conservation Condition

We have to check this condition at each internal node that lies on the Path P.

Let v be such a node, we can verify that the change in the amount of flow entering v is the same as the change in the amount of flow exiting v.

Since f is satisfied the Conservation Condition at v, so must f'.

(There are four cases to check, depending on whether the edge of P that enters (leaves) is a forward or Backward edge.

Final Algorithm

Ford-Fulkerson Algorithm

```
Max-Flow
  Initially f(e) = 0 for all e in G
  While there is an s-t path in the residual graph G_f
    Let P be a simple s-t path in G_f
    f' = augment(f, P)
    Update f to be f'
    Update the residual graph G_f to be G_f'
  Endwhile
  Return f
```

Correctness of the algorithm follows from the following theorem

Theorem:

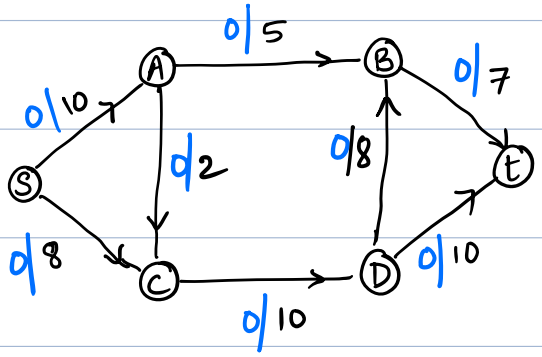
If f is a maximum flow in G iff

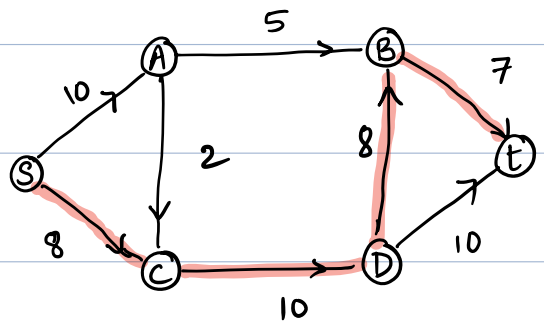
The residual network G_f has no augmenting Paths

[Proof is given at the end]

Example:

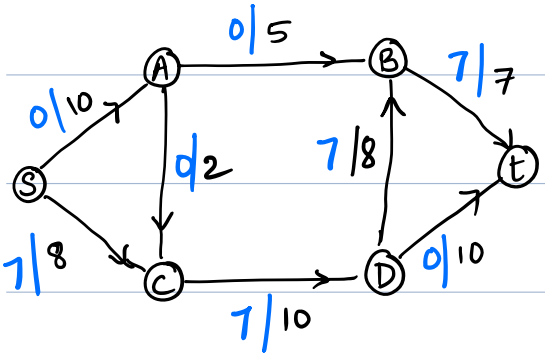
flow = 0

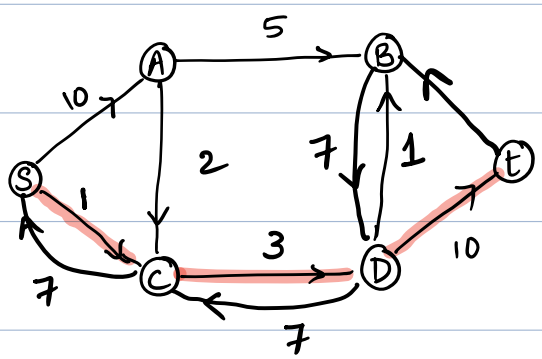




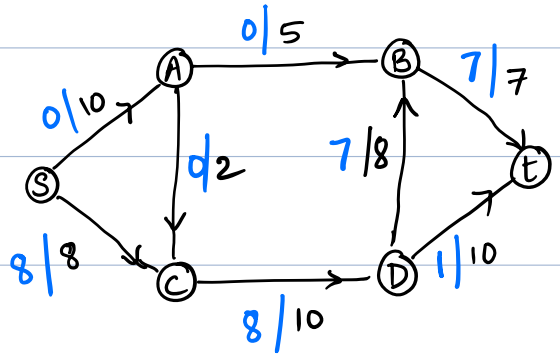
Residual graph

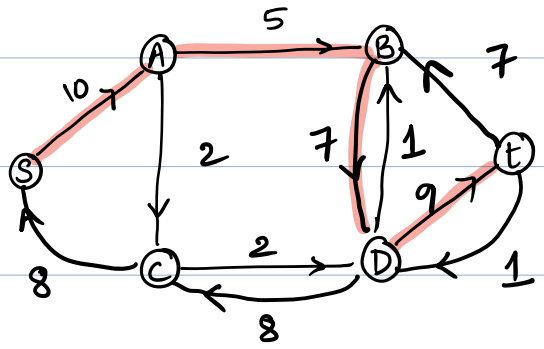
Aug Path: S - C - D - t → flow = flow + 7





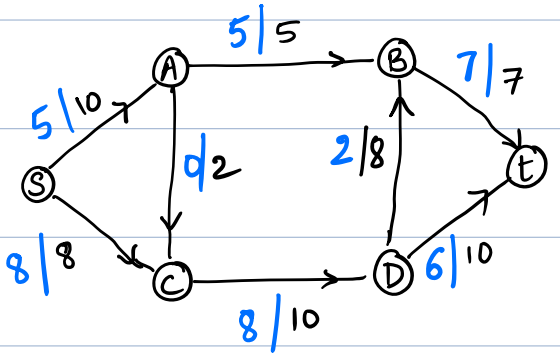
Aug Path: S - C - D - t → flow = flow + 1 = 8

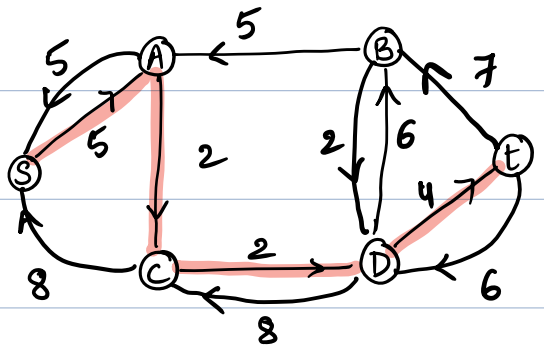




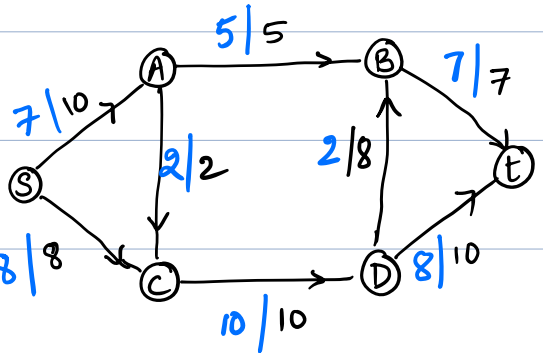
Aug Path : S-A-B-D-T → flow = flow + 5

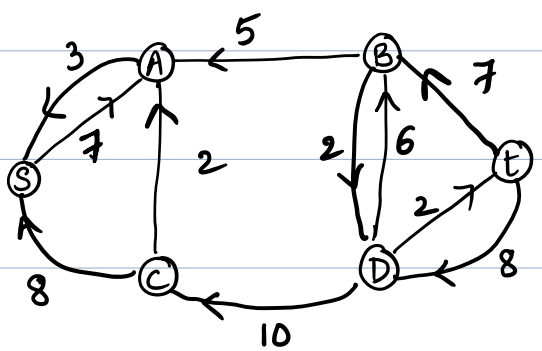
= 8 + 5 = 13





Aug Path : S-A-C-D-T → flow = flow + 2 = 15





No augmenting paths left.

Q How to find an augmenting path? BFS

Q If FF terminates, does it always compute a maxflow? YES

Q Does FF always terminate? If so, after how many augmentations?

↓

?

↓

Yes, Provided edge capacities are integers

Suppose edge capacities are integers b/w 1 and U

Invariant: The flow is integer valued throughout

Ford-Fulkerson. (bottleneck capacity is an integer)

Proposition: Number of augmentations ≤ the value of the max-flow

Pf:- Each augmentation increases the value by ≥ 1

Integrality theorem: There exists an integer-valued max-flow.

FF finds one.

flow value strictly increases when we apply an augmentation.

Lemma:

Let f be a flow in G, and let P be a simple s-t path in G_f. Then $v(f') = v(f) + \texttt{bottleneck}(P, f)$; and since $\texttt{bottleneck}(P, f) > 0$, we have $v(f') > v(f)$.

Lemma:

Suppose, as above, that all capacities in the flow network G are integers. Then the Ford-Fulkerson Algorithm terminates in at most C iterations of the `While` *loop.*

Where $C = \sum_{e \text{ out of } s} c_e$

Lemma: Ford-Fulkerson Algorithm runs in $O(mC)$ time, where $C = \sum_{e \text{ out of } s} c_e$

Proof: From the previous lemma, we know that the algorithm terminates in at most C iterations.

Therefore, we find the time taken for one iteration.

residual graph G_f has at most $2m$ edges.

We will maintain G_f using an adjacency list representation.

To find an s-t Path in G_f, we can use BFS/DFS which run in $O(m+n)$ time,

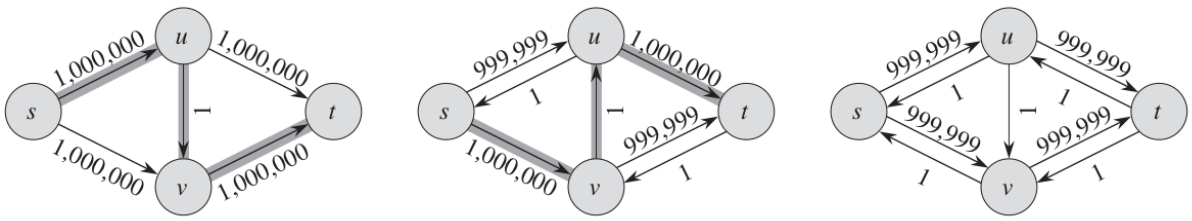
By our assumption $m \geq n/2$,

So BFS/DFS take $O(m)$ time.

The Procedure augment (f, P) takes time $O(n)$, as Path P has at most $n-1$ edges.

Given the new flow f', we can build the new residual graph in $O(m)$ time: for each edge e of G we construct the correct forward and backward edges in $G_{f'}$.

Bad case for FORD-FULKERSON:



(a) (b) (c)

(a) ↓

S-u-v-t is an augmenting path with residual capacity 1

(b) ↓

Residual Network with another augmenting path

S-v-u-t

(c) ↓

Residual network obtained from (b) (Next iteration)

For the above flow network FF can take $\Theta(mf^*)$ time, where f^* is a maximum flow, with $f^* = 2000000$.

Q Can we avoid this?

Ans: YES, Edmonds-Karp algorithm, $O(nm^2)$

We choose the augmenting path as a shortest path from s to t in the residual network, where each edge has unit weight.

Edmonds-Karp

1. Initialize $f = 0$
2. While $\exists$ a s-t path in G_f → $O(mn)$
3. Find a **shortest** s-t path P
4. let $b = \min_{uv \in P} c_f(uv)$

   Augment f by b over P

   } $O(m)$
5. Return f

Running time: $O(m^2 n)$

Running time is independent of capacities.

and capacities don't need to be integers.

Q: How do we know that when Ford-Fulkerson terminates, we have actually found a maximum flow?

We shall Prove the following Shortly

" A flow is maximum if and only if its residual network Contains no augmenting Path'

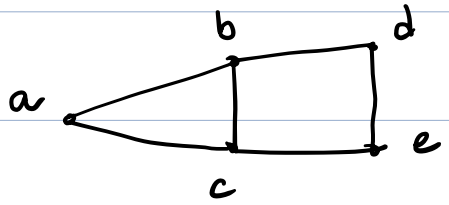
We study about Cuts in graphs to Prove the above result.

Cut in graph theory

Def:- A cut $C = (A, B)$ is a Partition of $V(G)$ of a Undirected graph G into the subsets A and B.

The Size or weight of a cut is the number of edges Crossing the cut. In a weighted graph the value or weight is defined by the sum of the weights of the edges crossing the cut.

Def: A cut is minimum if the Size or weight of the cut is not larger than the Size of any other cut.



$(\{a, b, c\}, \{d, e\})$ is a minimum cut

directed graph

Def: A cut (A, B) of flow network $G = (V, E)$ is a partition of V into A and $B = V - A$ such that $s \in A$ and $t \in B$.

In some textbooks cut (A, B) is also called as st-cut.

Note: In this chapter cut (A, B) means an st-cut.

The capacity of the cut (A, B) is the sum of the capacities of the edges from A to B.

$$c(A, B) = \sum_{u \in A} \sum_{v \in B} c(u, v)$$

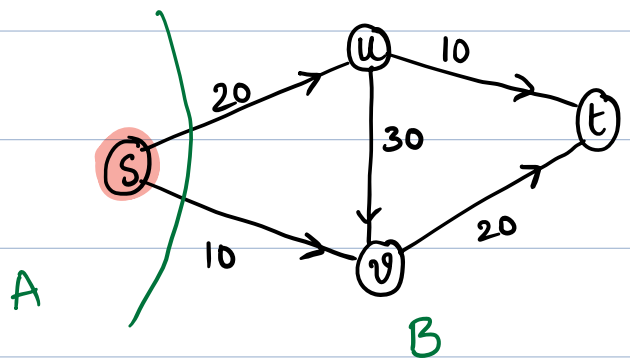
A minimum cut of a network is a cut whose capacity is minimum over all cuts of the network.

The Mincut | minimum cut Problem

Given an edge-weighted digraph G, with source vertex s and sink vertex t.

Find an st-cut of minimum capacity.

Example:



A = {s}, B = {u, v, t}, (A, B) is an s-t cut

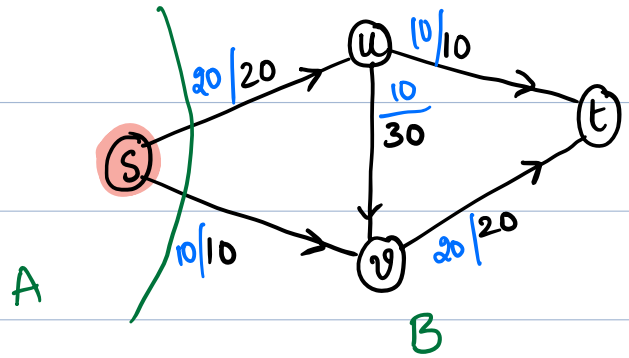
Capacity of (A, B) = C(A, B) = 10 + 20 = 30

Net flow

Def: The **net flow** across a **cut (A, B)** is the sum of the flows on its edges from **A to B** **minus** the sum of the flows on its edges from **B to A.** Denoted as **$f(A, B)$**

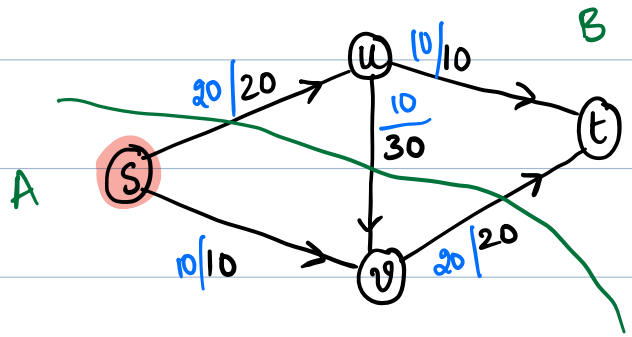
$$f(A,B) = \sum_{u \in A} \sum_{v \in B} f(u,v) - \sum_{u \in A} \sum_{v \in B} f(v,u)$$

Example



(i) $A = \{s\}$, $B = \{u, v, t\}$

net flow across the cut $(A, B) = f(A,B) = 20 + 10 = 30$



(ii) $A = \{s, v\}$ $B = \{u, t\}$

net flow across the cut $(A, B) = f(A, B)$ = $20 + 20 - 10 = 30$

Flow-value lemma:

Let f be any flow in a flow network G (with source s & sink t) and let (A, B) be any cut of G. Then, the netflow across (A, B) equals the value of f.

ie, $$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = v(f) \quad \text{—(1)}$$

Proof 1 Proof by induction on the size of A.

Base Case: $A = \{s\}$, as s has no incoming edges

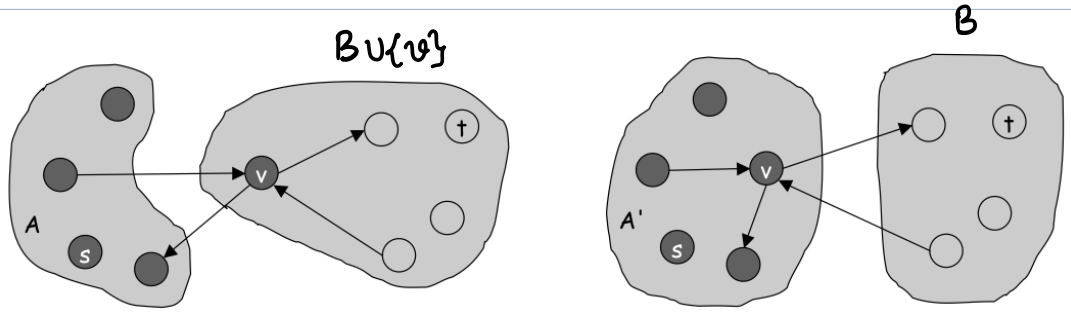
(1) holds trivially.

Inductive hypothesis: Assume true for all cut (A, B) with $|A| < k$

Consider a cut (A', B) with $|A'| = k$

where $A' = A \cup \{v\}$, for some A and $v \notin \{s, t\}$

By induction hypothesis, netflow across (A, B) $= v(f)$



Adding v to A increase the cut capacity by

$$\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) = 0$$

[By flow conservation property]

Proof 2:

We know $v(f) = f^{out}(s)$, $f^{in}(s) = 0$ as no edge enters s.

$\therefore$ $v(f) = f^{out}(s) - f^{in}(s)$.

For every node v in A other than s, we have

$$f^{out}(v) - f^{in}(v) = 0.$$

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

The only term in this sum that is non-zero is the one in which v is set to s.

We will rewrite the sum on the right side as follows.

- If an edge e has both ends in A, then f(e) appears once in the sum with a '+' and once with a '−' and hence these two terms cancel out.

- If e has only its tail in A then f(e) appears just once in the sum with a '+'.

- If e has only its head in A then f(e) appears just once in the sum with a '−'.

- If e has neither end in A then f(e) doesn't appear in the sum at all.

$$\therefore v(f) = \sum_{v \in A} f^{out}(v) - f^{in}(v) = \sum_{e \text{ out } A} f(e) - \sum_{e \text{ into } A} f(e)$$

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

$$= \text{Net-flow}(A, B)$$

Corollary: Outflow from S = inflow to t = Value of flow

Relationship between flows and cuts

Weak duality:

Lemma: Let f be any flow and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut. ie, $v(f) \leq cap(A, B)$

Proof:

$v(f)$ = net flow across cut $(A, B) \leq cap(A, B)$

(or)

$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$

(from flow value lemma)

$$\leq \sum_{e \text{ out of } A} f(e)$$

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

$$= cap(A, B)$$

The above lemma says that "the value of every flow is upper bounded by the capacity of every cut".

In otherwords, if we exhibit any s-t cut in G of some value C^*, then there can't be an s-t flow in G of value greater than C^*.

Conversly if we exhibit any s-t flow in G of some value V^*, then there cannot be an s-t cut in G of value less than V^*

Maxflow - mincut theorem:

Theorem: If f is a flow in a network $G=(V,E)$ with source s and sink t, then the following conditions are equivalent.

(i) There exists a cut (A,B) such that $V(f) = cap(A,B)$

(ii) Flow f is a max flow in G.

(iii) There is no augmenting path in the residual network G_f.

(i) $\Rightarrow$ (ii)

Suppose that (A, B) is a cut with capacity equal to the value of f.

then the value of any flow $f' \leq$ Cap(A, B) = Value of f

$\therefore$ f is a max-flow

(ii) $\Rightarrow$ (iii) $\equiv$ $\sim$iii $\Rightarrow$ $\sim$ii (Contrapositive)

Let f be a flow.

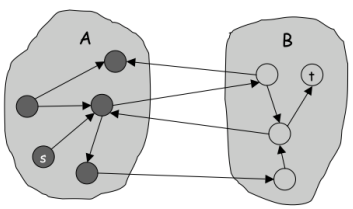
If there exists an augmenting Path, then we can improve f by sending flow along the Path.

(iii) $\Rightarrow$ (i) let f be a flow with no augmenting paths in G_f

let A be a set of vertices reachable from s in residual graph G_f.

ie, $A = \{ v \in V : \text{there exists a path from } s \text{ to } v \text{ in } G_f \}$

$B = V - A$



original network

The partition (A, B) is a cut.

By defn of A, $s \in A$

By defn of f, $t \notin A$

Consider a pair of vertices $u \in A$, $v \in B$.

- If $uv \in E(G)$, we must have $f(u,v) = c(u,v)$

otherwise $uv \in E(G_f)$, which implies $v \in A$.

- If $vu \in E(G)$, we must have $f(v,u) = 0$

otherwise $c_f(u,v) = f(v,u)$ would be positive and we would have $(u,v) \in E(G_f)$, which would place v in S.

- If neither uv or vu is in $E(G)$ then

$$f(u,v) = f(v,u) = 0$$

$$V(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

$$= \sum_{u \in A} \sum_{v \in B} f(u,v) - \sum_{v \in B} \sum_{u \in A} f(v,u)$$

$$= \sum_{u \in A} \sum_{v \in B} c(u,v) - \sum_{v \in B} \sum_{u \in A} 0$$

$$= \text{Cap}(A,B)$$

Max-Flow-Min-cut theorem :

In every flow network, the maximum value of an S-t flow is equal to the minimum capacity of an S-t cut.

Edmonds - Karp algorithm

If we find the augmenting path in FORD-FULKERSON using BFS, then the algorithm is called Edmonds-Karp algorithm.

Lemma

If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then for all vertices $v \in V - \{s, t\}$, the shortest-path distance $\delta_f(s, v)$ in the residual network G_f increases monotonically with each flow augmentation.

Proof: Proof by Contradiction

Suppose that for some vertex $v \in V - \{s, t\}$, there is a flow augmentation that causes the shortest path distance from s to v to decrease.

Let f be the flow just before the first augmentation that decreases some shortest path distance.

Let f' be the flow afterward.

Let v be the vertex with the minimum $\delta_{f'}(s, v)$ whose distance was decreased by the augmentation.

ie, $\delta_{f'}(s, v) < \delta_f(s, v)$

Let $p = s \rightsquigarrow u \to v$ be a shortest path from s to v in $G_{f'}$ and $(u, v) \in E_{f'}$ and

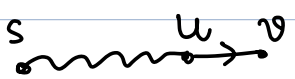
$$\delta_{f'}(s,u) = \delta_{f'}(s,v) - 1 \quad \text{— (A)}$$

By our choice of v, we have

$$\delta_{f'}(s,u) \geq \delta_f(s,v) \quad \text{— (B)}$$

Claim: $(u,v) \notin E_f$

if $(u,v) \in E_f$ then



$$\delta_f(s,v) \leq \delta_f(s,u) + 1$$

$$\leq \delta_{f'}(s,u) + 1$$

$$\leq \delta_{f'}(s,v)$$

Which contradicts our assumption that $\delta_{f'}(s,v) < \delta_f(s,v)$.

∴ we have $(u,v) \notin E_f$ & $(u,v) \in E_{f'}$.

Q: When this will happen?

If the augmentation must have increased the flow from v to u.

* The Edmonds-Karp algorithm always augments flow along shortest paths, and therefore the shortest path from s to u in G_f has (v,u) has the last edge

$\therefore \quad \delta_f(s,v) = \delta_f(s,u) - 1$

$\leq \delta_{f'}(s,u) - 1 \quad$ (from (B))

$= \delta_{f'}(s,v) - 2 \quad$ (from (A))

Which contradicts our assumption that $\delta_{f'}(s,v) < \delta_f(s,v)$.

$\therefore$ our assumption that such a vertex v exists is incorrect.

Theorem

If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then the total number of flow augmentations performed by the algorithm is $O(VE)$.

Proof: def: we say an edge (u,v) in a residual network G_f is critical on an augmenting path P if the residual capacity of P is the residual capacity of (u,v), ie., $c_f(P) = c_f(u,v)$.

After we have augmented flow along an augmenting path, any critical edge on the path disappears from the residual network. Note that at least one edge on any augmenting path must be critical.

Claim: Each edge can become critical at most $\frac{|V|}{2}$ times.

Proof: let $(u,v) \in E(G)$.

As augmenting paths are shortest paths, when (u,v) is critical for the first time we have

$$\delta_f(s,v) = \delta_f(s,u) + 1$$

Once the flow is augmented, the edge (u,v) disappears from the residual network.

It can only appear on another augmenting path if the flow from u to v is decreased, which occurs only if (v,u) appear on an augmenting path.

If f' is the flow in G_1 when this event occurs, then we have

$$\delta_{f'}(s,u) = \delta_{f'}(s,v) + 1 \quad -(A)$$

we know (Previous lemma) $\delta_f(s,v) \leq \delta_{f'}(s,v)$

(A) becomes

$$\delta_{f'}(s,u) \geq \delta_f(s,v) + 1$$

$$\geq \delta_f(s,u) + 2$$

ie, the time (u,v) becomes critical to the time when it next becomes critical, the distance of u from the source increases by at least 2.

The distance of u from the source is initially at least zero.

The intermediate vertices on a shortest S-U path cannot contain s, u and t.

until u becomes unreachable from the source its distance is at most $|V|-2$.

Thus, after the first time (u,v) becomes critical, it can become critical at most $\frac{(|V|-2)}{2} = \frac{|V|}{2} - 1$ times.

∴ Total # of critical edges during the entire execution of the Edmonds-Karp algorithm is $O(VE)$, as each aug-path has at least one critical edge.

Running time of Edmonds-Karp algorithm is $O(VE^2)$

Max-flow applications

- Bipartite matching ✓ → We shall see how to use FF algo to solve the maximum bipartite matching
- Image segmentation
- Network Connectivity
- " reliability
- etc

Bipartite matching Problem

N Students & N Jobs

1 Alice
- Adobe
- Amazon
- Google

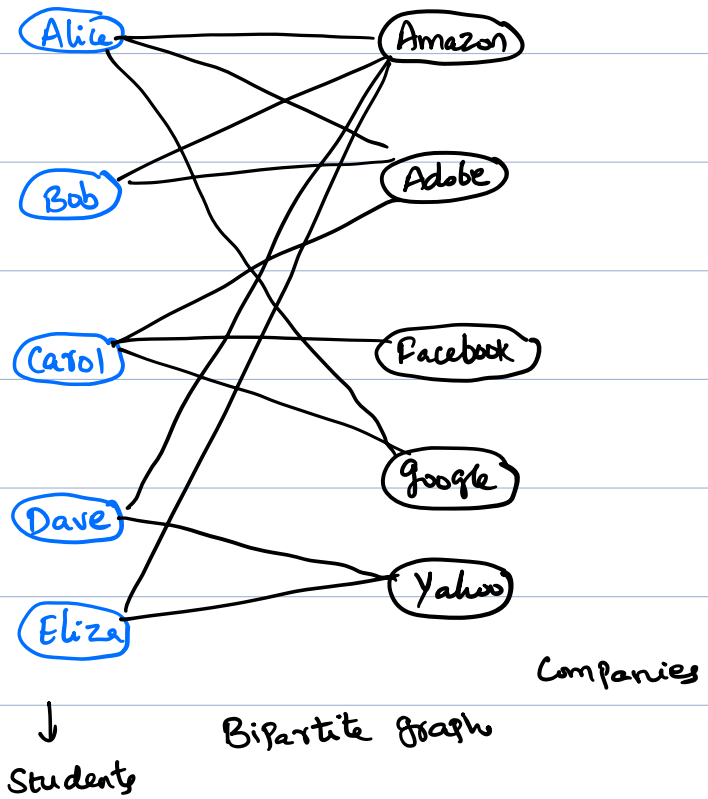
2 Bob
- Adobe
- Amazon

3. Carl
- Adobe
- Facebook
- Google

4. Dave
- Amazon
- Yahoo

5. Eliza
- Amazon
- Yahoo

Is there a way to match all students to jobs?



Solution - (Perfect matching)

Alice — Google

Bob - Adobe

Carol - Facebook

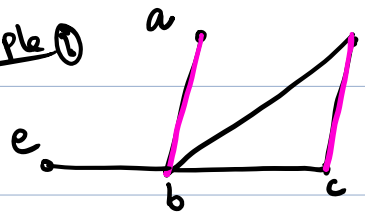
Dave - Yahoo

Eliza - Amazon

Matchings in Graphs

Def: Matching: A matching in a graph is a set of edges with no shared end points.
[independent edges]

Example ①



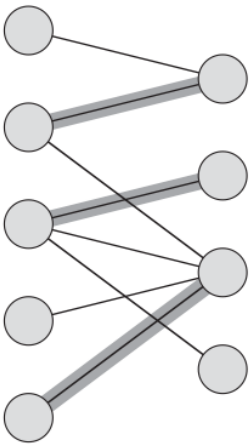
$M = \{ab, cd\}$ is a matching.

We say that a vertex $v \in V$ is matched by the matching M if some edge in M is incident on v. Otherwise v is unmatched.

A MAXIMUM matching is a matching of maximum size among all matchings in the graph.

In this topic, we restrict our attention to finding maximum matchings in Bipartite graphs.

Example.

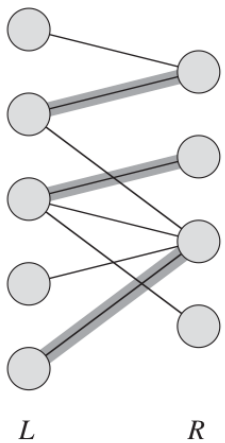


A maximum matching of size 3.

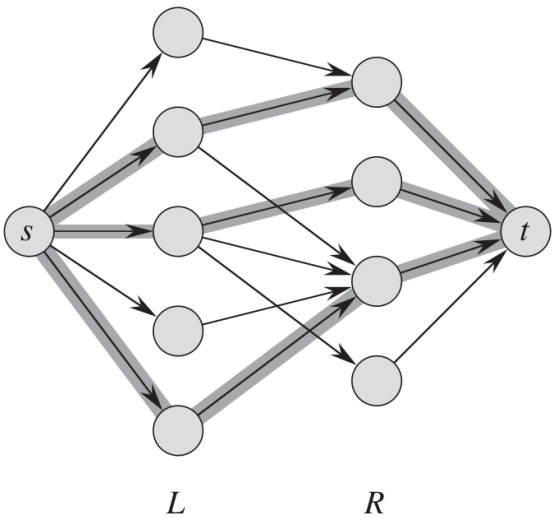
↓

(shaded edges)

Network flow formulation of bipartite matching



Bipartite graph $G = (V, E)$



$G' = (V', E')$

$V' = V \cup \{s, t\}$

$E' = \{su \mid u \in L\} \cup \{vt \mid v \in R\}$

$\cup \{uv : uv \in E\}$

↓

directed from u to v

Each edge has a unit capacity

Since each vertex in V has **at least one incident edge.**

ie, $|E| \geq \frac{|V|}{2}$. Thus $|E| \leq |E'| = |E| + |V| \leq 3|E|$

So $|E'| = O(|E|)$

Def: We say that a flow f on a flow network $G = (V, E)$ is integer-valued if $f(u,v)$ is an integer for all $(u,v) \in V \times V$

Lemma:

Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$, and let $G' = (V', E')$ be its corresponding flow network. If M is a matching in G, then there is an integer-valued flow f in G' with value $|f| = |M|$. Conversely, if f is an integer-valued flow in G', then there is a matching M in G with cardinality $|M| = |f|$.

Val(f)

Proof idea: M is a matching in G

Define f as follows

if $uv \in M$, then $f(su) = f(uv) = f(vt) = 1$

for all $uv \in E'$, $f(uv) = 0$.

Easy to see that f satisfies the Capacity & flow Conservation constraints.

Netflow-across cut $(L \cup \{s\}, R \cup \{t\})$ is $|M|$

$\therefore$ $V(f) = |M|$

Converse: let f be an integer valued flow

$M = \{ uv \mid u \in L, v \in R \text{ and } f(uv) > 0 \}$

Each vertex $u \in L$ has only one entering edge i.e., (s, u) and it capacity is 1. Thus, each $u \in L$ has at most one unit of flow entering it. If one unit of flow does enter, one unit of flow must leave.

As f is an integer valued,

For each $u \in L$, the one unit of flow enter on at most one edge and can leave on at most one edge.

Thus, one unit of flow enters u iff there is exactly one vertex $v \in R$ such that $f(u, v) = 1$ and at most one edge leaving each $u \in L$ carries positive flow. A symmetric argument applies to each $v \in R$.

$\therefore$ the set M is a matching.

Theorem: The size of a maximum matching M in a bipartite graph G equals the value of a maximum flow f in its corresponding flow network G'.

Running time: $O(VE)$

because size of any matching is $O(V)$