CSEP 521
Applied Algorithms
Spring 2005

Maximum Flow

Reading

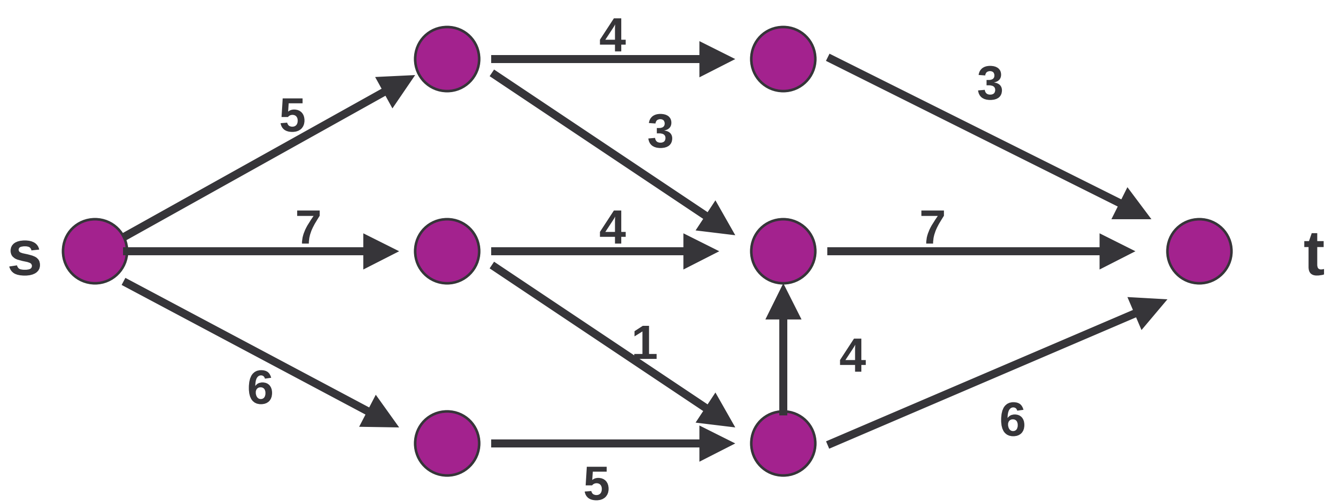
- Chapter 26

Outline:

- Properties of flow
- Augmenting paths
- Max-flow min-cut theorem
- Ford-Fulkerson method
- Edmonds-Karp method
- Applications, bipartite matching and more.
- Variants: min cost max flow

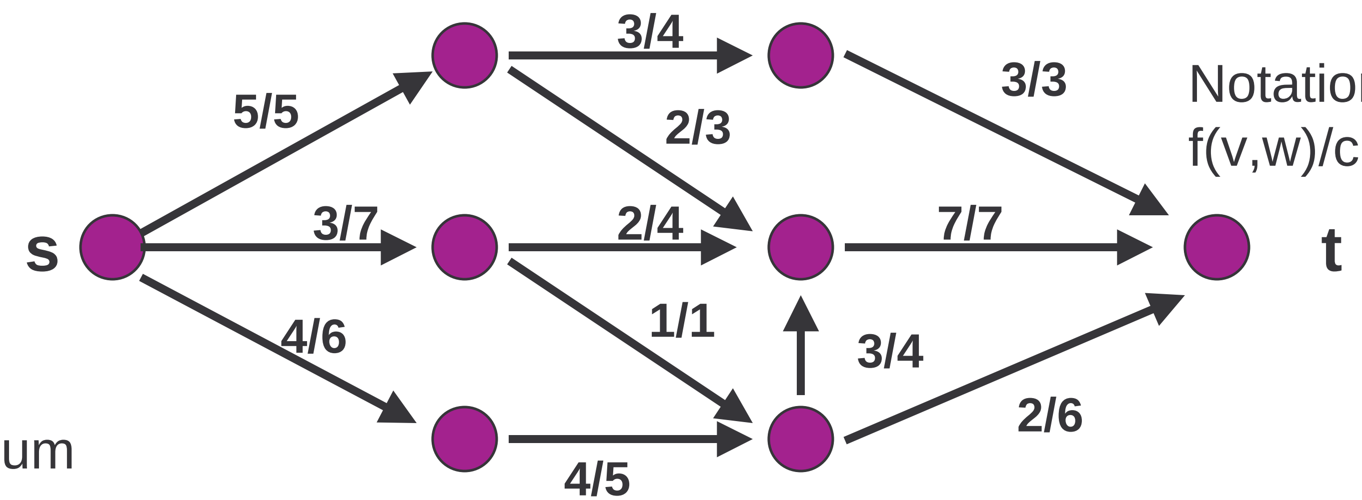
Maximum Flow

- Input: a directed graph (network) G
  - each edge (v,w) has associated capacity c(v,w)
  - a specified source node s and target node t
- Optimization Problem: What is the maximum flow you can route from s to t while respecting the capacity constraint of each edge?



Properties of Flow:
f(v,w) - flow on edge (v,w)

- **Edge condition:** $0 \leq f(v,w) \leq c(v,w)$: the flow through an edge cannot exceed the capacity of an edge.
- **Vertex condition:** for all v except s,t : $\Sigma_u f(u,\mathbf{v}) = \Sigma_w f(\mathbf{v},w)$ the total flow entering a vertex is equal to total flow exiting this vertex.
- total flow leaving s = total flow entering t.

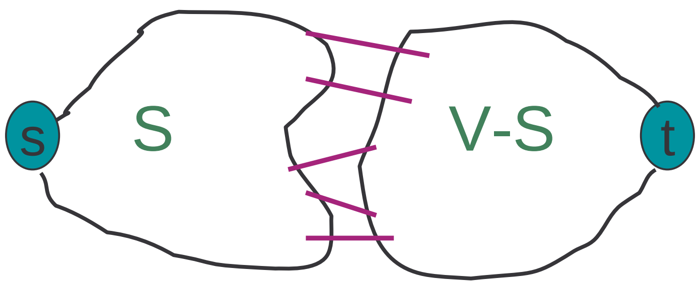


Notation on edges f(v,w)/c(v,w)

Not a maximum flow!

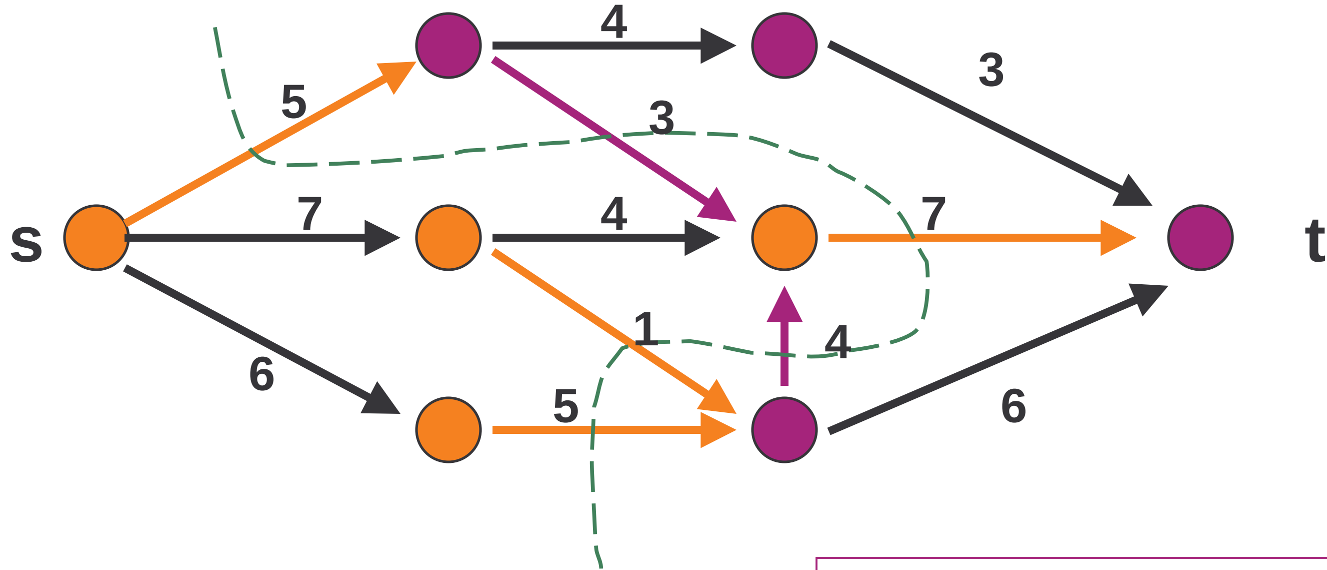
Cut

- Cut - a set of edges that separates s from t.
- A cut is defined by a set of vertices, S. This set includes s and maybe additional vertices reachable from s. The sink t is not in S.
- The cut is the set of edges (u,v) such that $u \in S$ and $v \notin S$, or $v \in S$ and $u \notin S$.



- out(S) – edges in the cut directed from S to V-S
- in(S) – edges in the cut directed from V-S to S

Cut - example



The capacity of this cut = 18

S –

out(S) –

in(S) –

Value of a Flow:

- A flow function f is an assignment of a real number f(e) to each edge e such that the edge and vertex conditions hold for all the vertices/edges.

- Definition: The value of the flow is the flow net flow from s

$$F = \sum_{e \in out(s)} f(e) - \sum_{e \in in(s)} f(e).$$

Flow

- Theorem: The net flow into t equals the net flow out of s.

$$F = \sum_{e \in out(s)} f(e) - \sum_{e \in in(s)} f(e) = \sum_{e \in in(t)} f(e) - \sum_{e \in out(t)} f(e)$$

Capacity of a cut

For a cut S, the *capacity of S* is $c(S) = \sum_{e \in out(S)} c(e).$

Claim: For every flow function f with total flow F, and every cut S, $F \leq c(S)$.

Proof: We know that $F = \sum_{e \in out(S)} f(e) - \sum_{e \in in(S)} f(e).$

By the edge condition, $0 \leq f(e) \leq c(e)$, for all $e \in E$. Thus,

$$F \leq \sum_{e \in out(S)} c(e) - 0 = c(S).$$

Max-flow Min-Cut Theorem

The value of a maximum flow in a network is equal to the minimum capacity of a cut.

Proof:

max flow $\leq$ min cut: follows from the previous lemma.

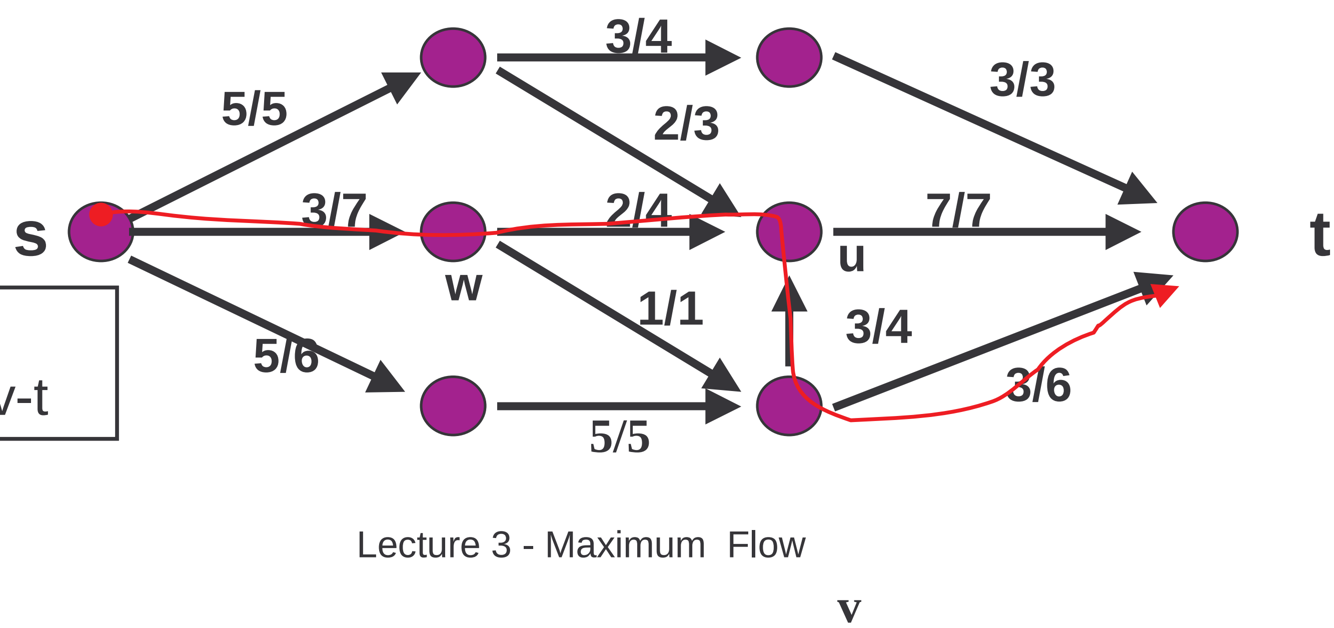
max flow $\geq$ min cut: we will see an algorithm that produces a flow in which some cut is saturated.

An augmenting path with respect to a given flow f:

A directed path from s to t which consists of edges from G, but **not necessarily in the original direction**.

forward edge: (w,u) in same direction as G and f(w,u) < c(w,u). (c(w,u)-f(w,u) is called *slack*) à has room for more flow.

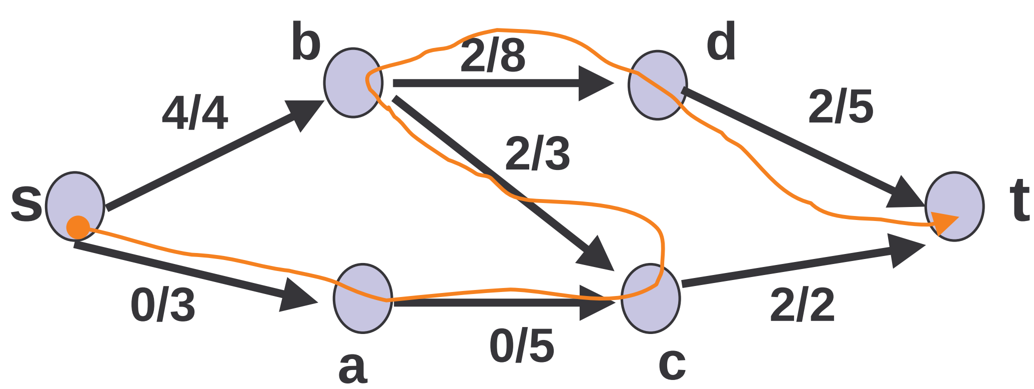
backward edge: (u,v) in opposite direction in G (i.e., (v,u) in E) and f(v,u) > 0 à can 'take back' flow from such an edge.



augmenting path s-w-u-v-t

Using an augmenting path to increase flow

- Push flow forward on forward edges, deduct flow from backward edges.



p = s-a-c-b-d-t

•The amount of flow we can push:

$$\text{minimum}\begin{cases}\text{slacks along the forward edges on the path}\\\text{flow along the backward edges on the path}\end{cases}$$

The Ford-Fulkerson Method

- Initialize flow on all edges to 0.
- While there is an augmenting path, improve the flow along this path.

To implement F&F, we need a way to detect augmenting paths.

We build a residual graph with respect to the current flow.

Residual Graph w.r.t. flow f

- Given f, we build the residual graph: a network flow R=(V,E')
- An edge (v,w)$\in$ E' if either
  - (v,w) is a forward edge, and then its capacity in R is c(v,w)-f(v,w)
  - or (v,w) is a backward edge (that is, (w,v) is an edge with positive flow in G) , and then its capacity in R is f(w,v).
- An augmenting path is a regular directed path from s to t in R.

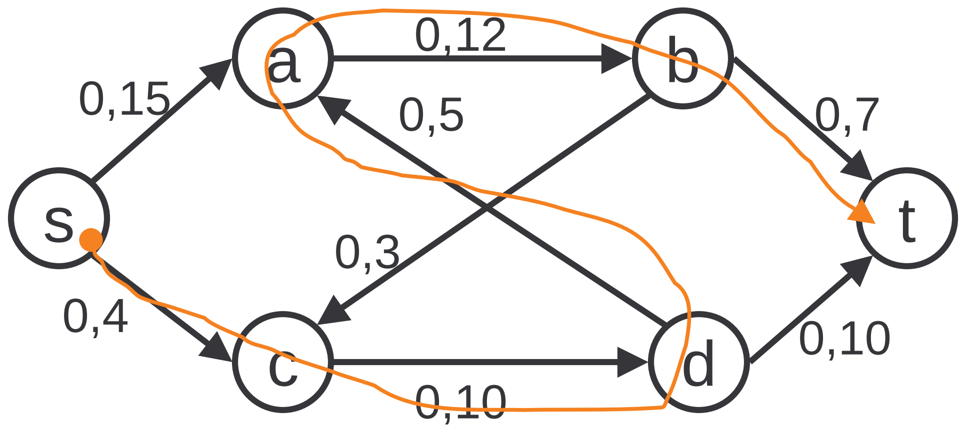
Ford-Fulkerson Method (G,s,t)

- Initialize flow on all edges to 0.
- While there is a path p from s to t in residual network R
  - δ = minimum capacity along p in R
  - augment δ units of flow along p and update R.

Ford-Fulkerson Method. Example (1)

Example taken from the book Graph Algorithms by Shimon Even

The given network, with initial all-0 flow.



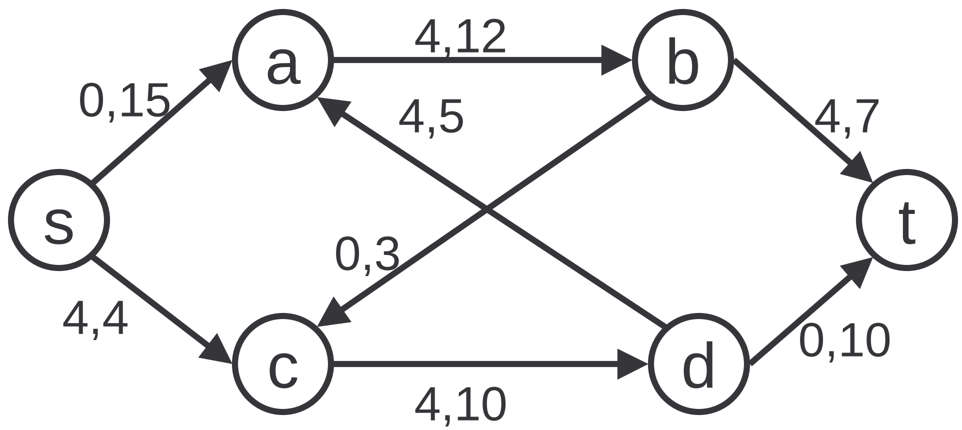
First augmenting path: $s \to c \to d \to a \to b \to t$

$\delta = 4$

Remark: in the first iteration R=G.

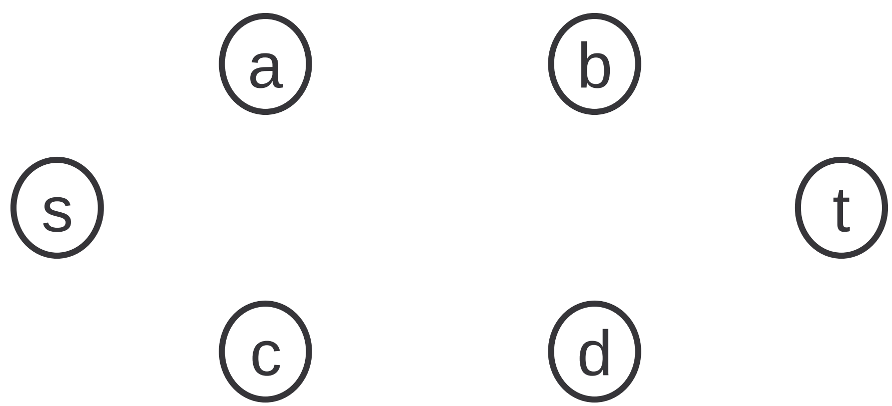
Ford-Fulkerson Method. Example (2)

The network after applying the first augmenting path:

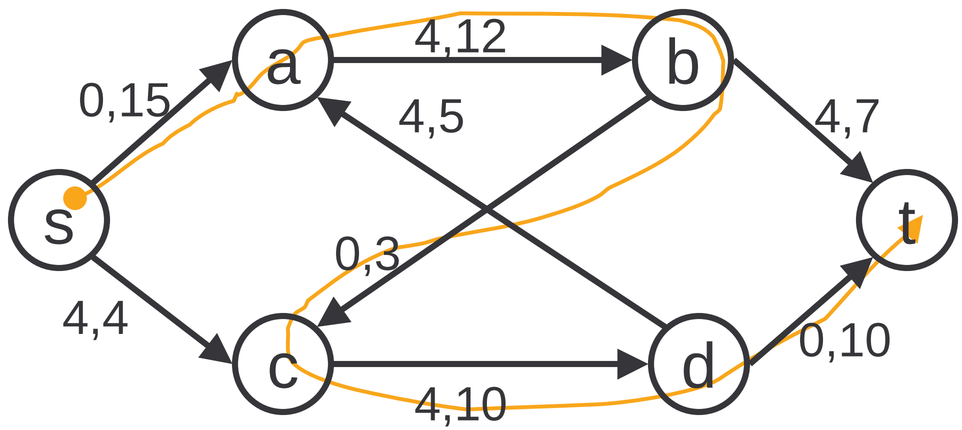


The residual network:

(complete in class)



Ford-Fulkerson Method. Example (3)

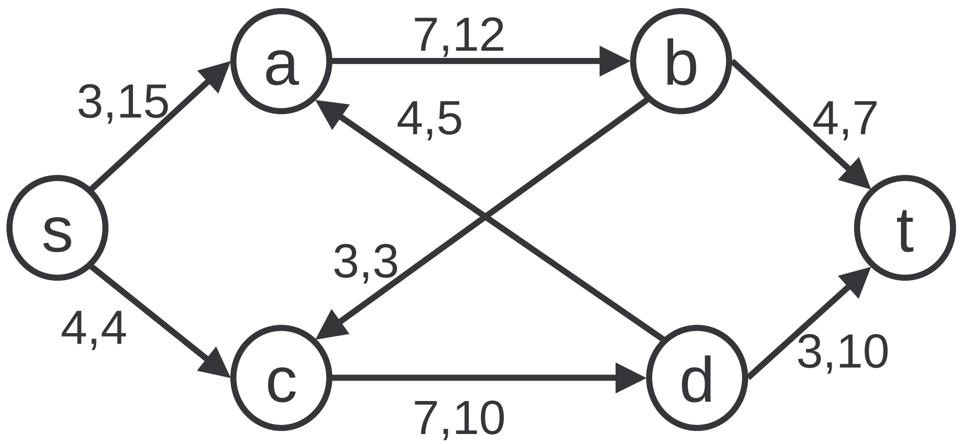


Second augmenting path: s ⟶ a ⟶ b ⟶ c ⟶ d ⟶ t

$\delta = 3$

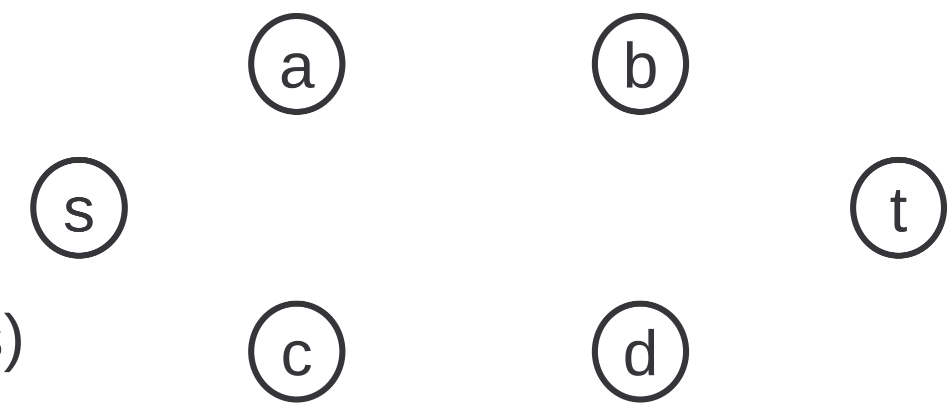
Ford-Fulkerson Method. Example (4)

The flow after applying 2nd augmenting path:

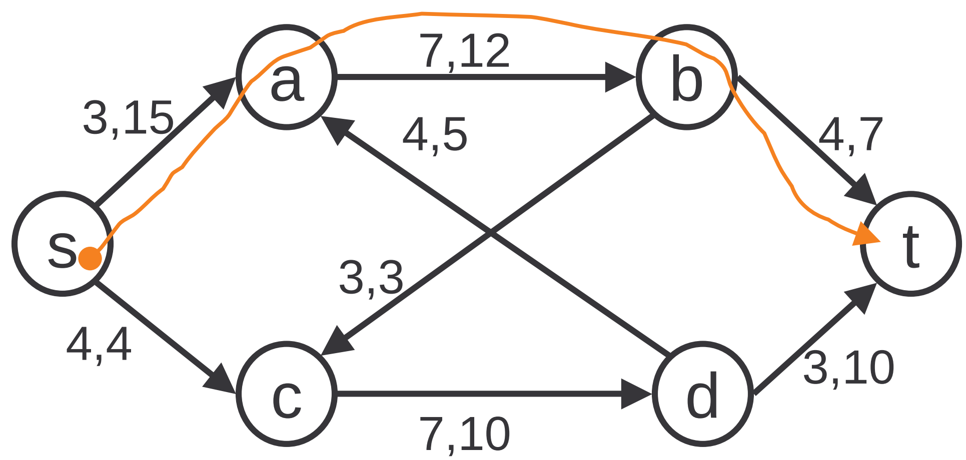


The residual network:

(complete in class)



Ford-Fulkerson Method. Example (5)

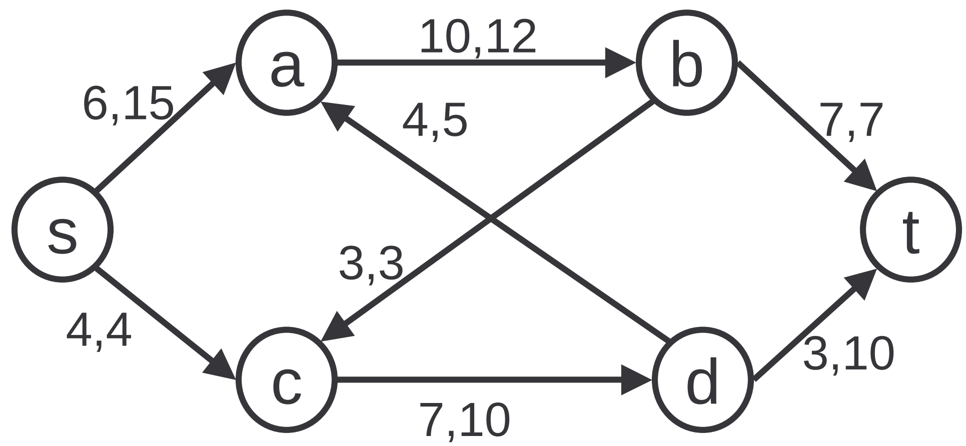


Third augmenting path: s ⟶ a ⟶ b ⟶ t

$\delta = 3$

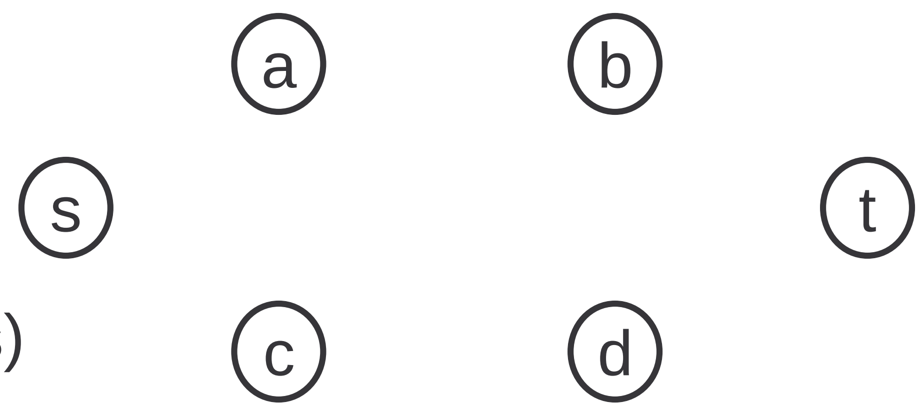
Ford-Fulkerson Method. Example (6)

The flow after applying 3rd augmenting path:

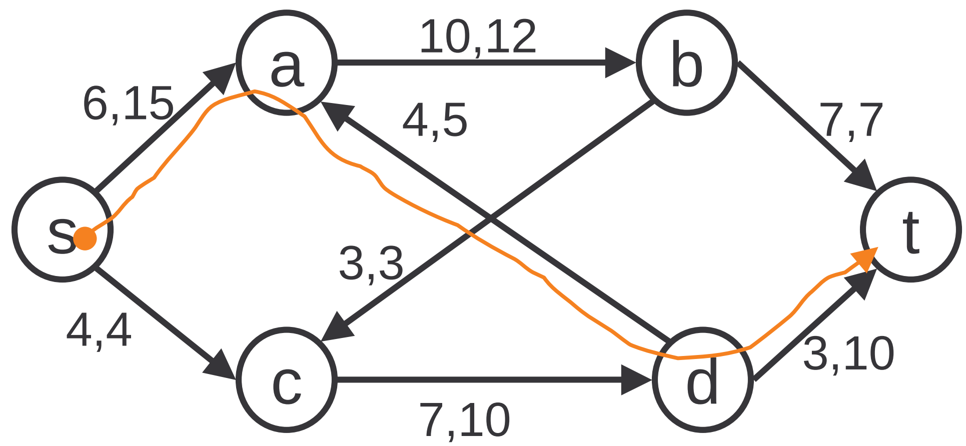


The residual network:

(complete in class)



Ford-Fulkerson Method. Example (7)



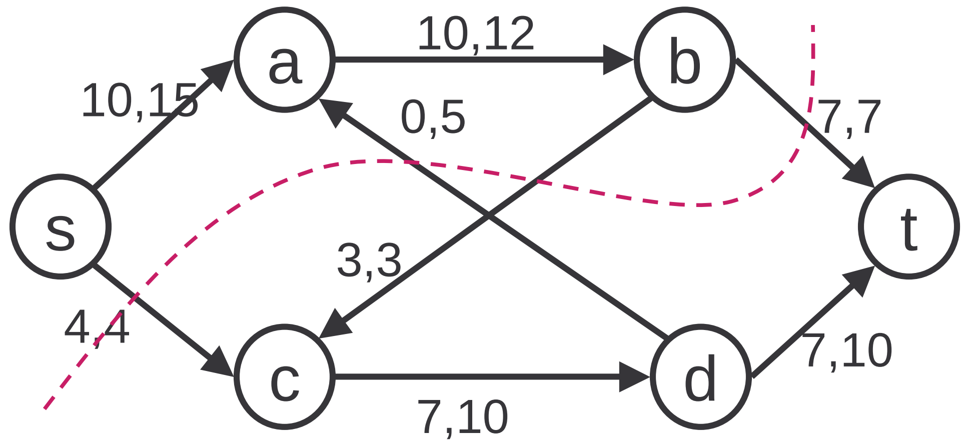
Forth augmenting path: $s \rightarrow a \leftarrow d \rightarrow t$

$\delta = 4$

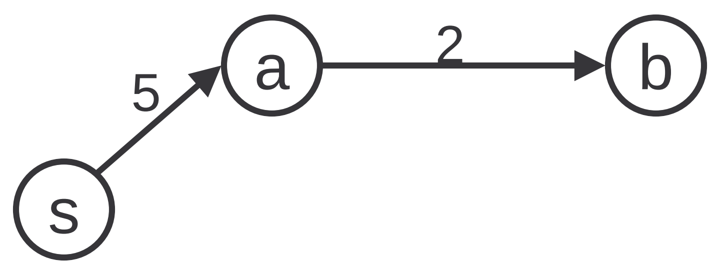
Ford-Fulkerson Method. Example (8)

Final flow:

{s,a,b} is a saturated min-cut



There are no paths connecting s and t in the residual network



Proof of Ford-Fulkerson Method.

Claim: The flow after each iteration is legal

Proof: The initial assignment (of f(e)=0 for all e) is clearly legal.

Let p be an augmenting path. Let d be the minimum capacity along p in R.

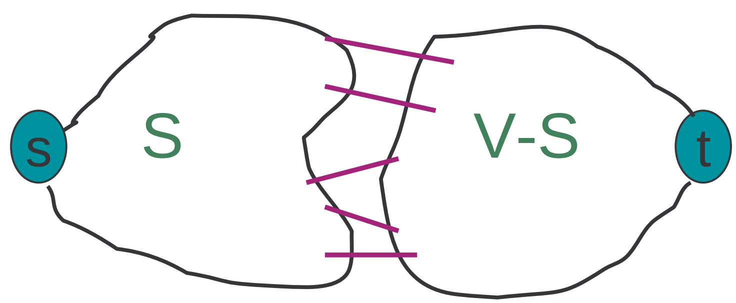
Vertex condition: For each $v \notin p$, the flow that passes v does not change. For each $v \in p$ ($v \neq s,t$), exactly one edge of p enters v and exactly one edge of p goes out of v. In each of these edges the flow increase by δ**.** The value of the flow in and out of v remains 0.

Egde condition: preserved by the selection of δ

Proof of Ford-Fulkerson Method.

Theorem: A flow f is maximum if and only if it admits no augmenting path

- Already saw that if an augmenting path exists, then the flow is not maximum (can be improved).
- Suppose f admits no augmenting path. We need to show that f is maximum.
- We use the min-cut max-flow theorem: we will see that when no augmenting path exists, some cut is saturated.



Proof of Ford-Fulkerson Method.

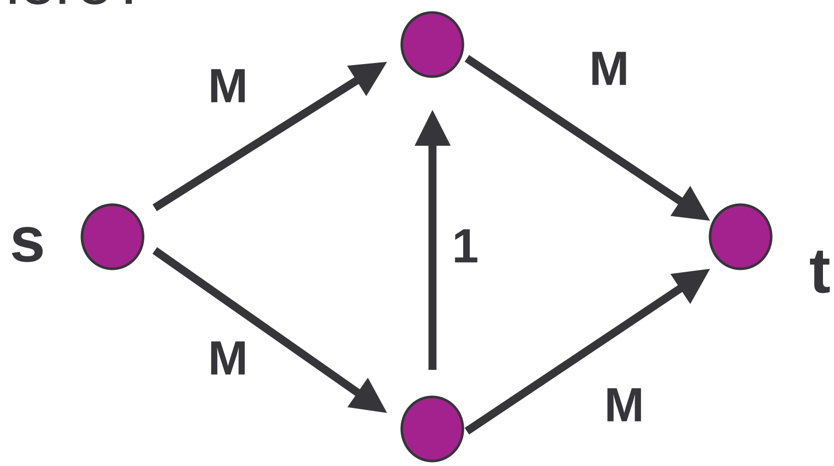
- Let A be the vertices such that for each v∈ A, there is an augmenting path from s to v.
- The set A defines a cut.
- Claim: for all edges in cut, f(v,w)=c(v,w).
- Proof: if f(v,w)< c(v,w) then w should join A.
- Therefore: The value of the flow is the capacity of the cut defined by A à (min cut theorem) f is maximum.

Running time of Ford-Fulkerson

Each iteration (building R and detecting an augmenting path) takes O(|E|) (how?).

How many iterations are there?

Could be f^* when f* is the value of the maximum flow.



The time complexity of F&F is O(|E|f*), when f* is the value of the maximum flow.

Edmonds-Karp Algorithm:

- Use F&F method. Search for augmenting path using breadth-first search, i.e., the augmenting path is always a shortest path from s to t in the residual network.
- Theorem: This way, the number of augmentations is $O(|V||E|)$.
- The resulting complexity: $O(|V||E|^2)$
  - each iteration takes $O(|E|)$

Greedy augmenting path Selection:

- Use F&F method. In each iteration select an augmenting path with the maximal δ value.

- The time complexity of this algorithm is
  $O(|E|\log_2 f^*)$.

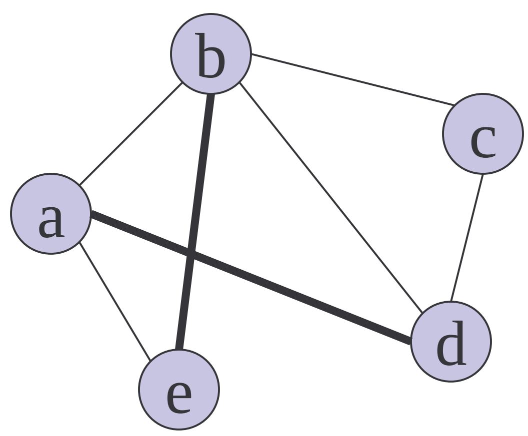
Some applications of max-flow and max-flow min-cut theorem

- Bipartite matching
- Network connectivity
- Video on demand
- Many many more…

Matching

- Definition: a matching in a graph G is a subset M of E such that the degree of each vertex in G'=(V',M) is 0 or 1.
- Example: M={(a,d),(b,e)} is a matching.

  S={(a,d), (c,d)} is not a matching.



Bipartite Matching

- Example 1: In a party there are n_1 boys and n_2 girls. Each boy tells the DJ the girls with whom he is ready to dance with. Each girl tells the DJ the boys with whom she is ready to dance with.

- DJ's goal: As many dancing pairs as possible.

- Note: This has nothing to do with the stable pairing problem. No preferences. Some participants can remain lonely (even if $n_1=n_2$).

- Example 2 (production planning) : n_2 identical servers need to serve n_1 clients. Each client specifies the subset of servers that can serve him.

- Goal: Serve as many clients as possible.

Bipartite Matching

Graph representation: $G=(V,E)$.

$V= V_1 \cup V_2$.

In 1st problem $(u,v) \in E$, if u is ready to dance with v and vice versa.

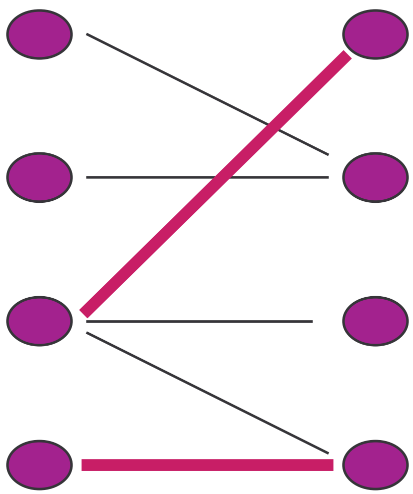
In 2nd problem $(u,v) \in E$, if u can be served by v.

This is a bipartite!

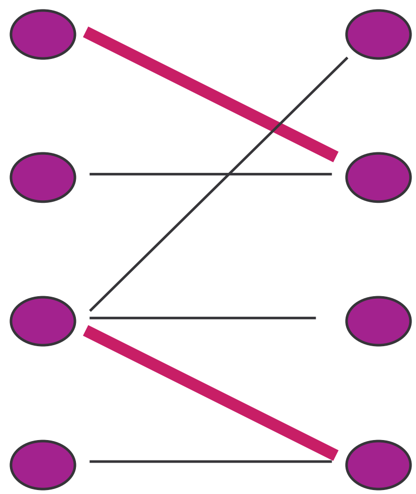
We are looking for the largest possible matching.

Bipartite Matching

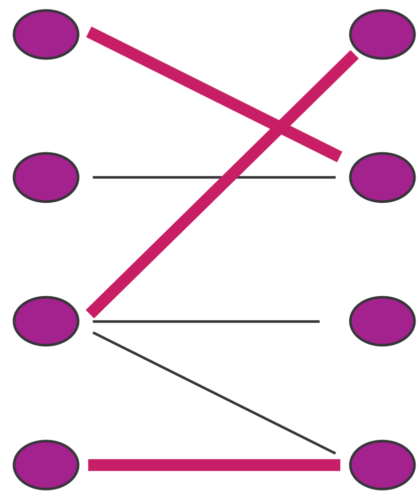
- Input: a bipartite graph $G=(V_1 \cup V_2, E)$
- Goal: A matching of maximal size.



A matching



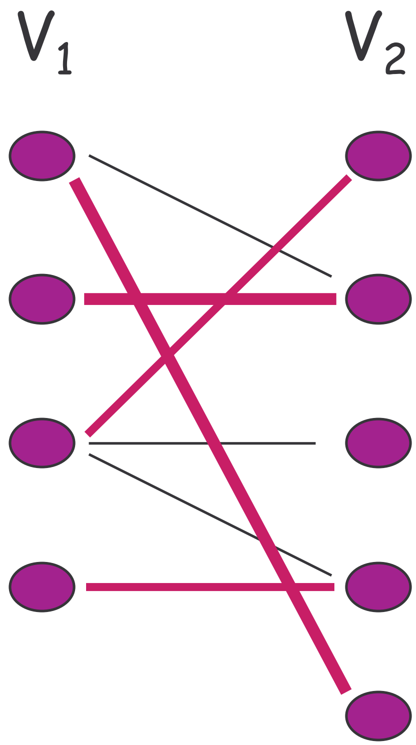
A maxim**al** matching – can not be extended.



A maxim**um** matching – largest maximal.

Our goal !

Bipartite Matching



Special cases:

- A perfect matching: $|M|=|V_1|=|V_2|$

  (An ideal instance and solution for problem 1)

- A full matching for V_1: $|M|=|V_1|\leq|V_2|$

  (what we need in problem 2)

Maximum matching in a bipartite can be found using flow algorithms.

Using Flow for Bipartite Matching

Input: A bipartite $G=(V_1 \cup V_2, E)$

Output : Maximum matching $M \subseteq E$.

Algorithm:

1. Build a network flow $N=(V', E')$

   $V' = V_1 \cup V_2 \cup \{s,t\}$

   $E' = E \cup \{(s \to u) | \forall u \in V_1\} \cup \{(v \to t) | \forall v \in V_2\}$

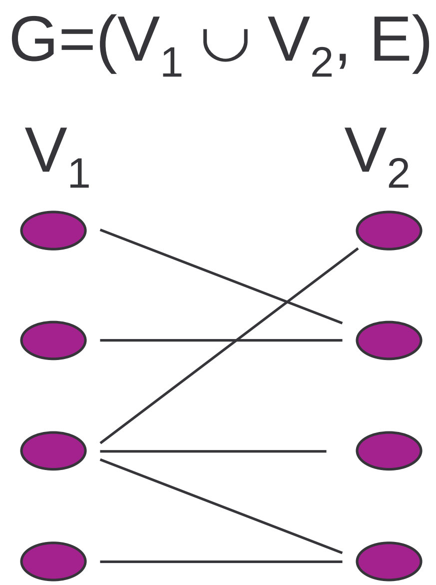
   All $e \in E'$ have the capacity $c(e)=1$.

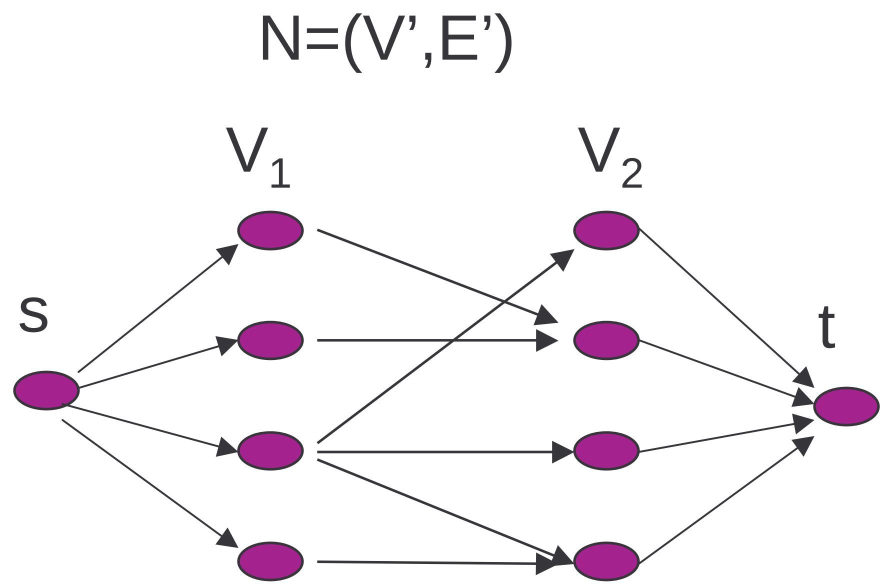
2. Find a maximum flow in N.

3. M = saturated edges in the cut defined by $\{s, V_1\}$.

Vertices of E are directed from V_1 to V_2

Using Flow for Bipartite Matching (Example)





$V' = V_1 \cup V_2 \cup \{s,t\}$

$E' = E \cup \{(s \rightarrow u) | \forall u \in V_1\} \cup \{(v \rightarrow t) | \forall v \in V_2\}$

For all $e \in E'$, $c(e)=1$.

Using Flow for Bipartite Matching (proof)

Theorem: G includes a matching of size k $\Leftrightarrow$ N has flow with value k.

Proof:

1. ($\Rightarrow$) Given a matching of size k, define the flow f(u,v)= 1 for all (u,v) in M, all all (s,u) and (v,t) such that u or v are matched. For all the other edges f=0.

- F is legal (proof in class)
- The value of f is k (consider the cut $\{s\} \cup V_1$).

2. ($\Leftarrow$) Similar. Based on the capacities of the edges (s,u), (v,t), and the fact that f is legal.

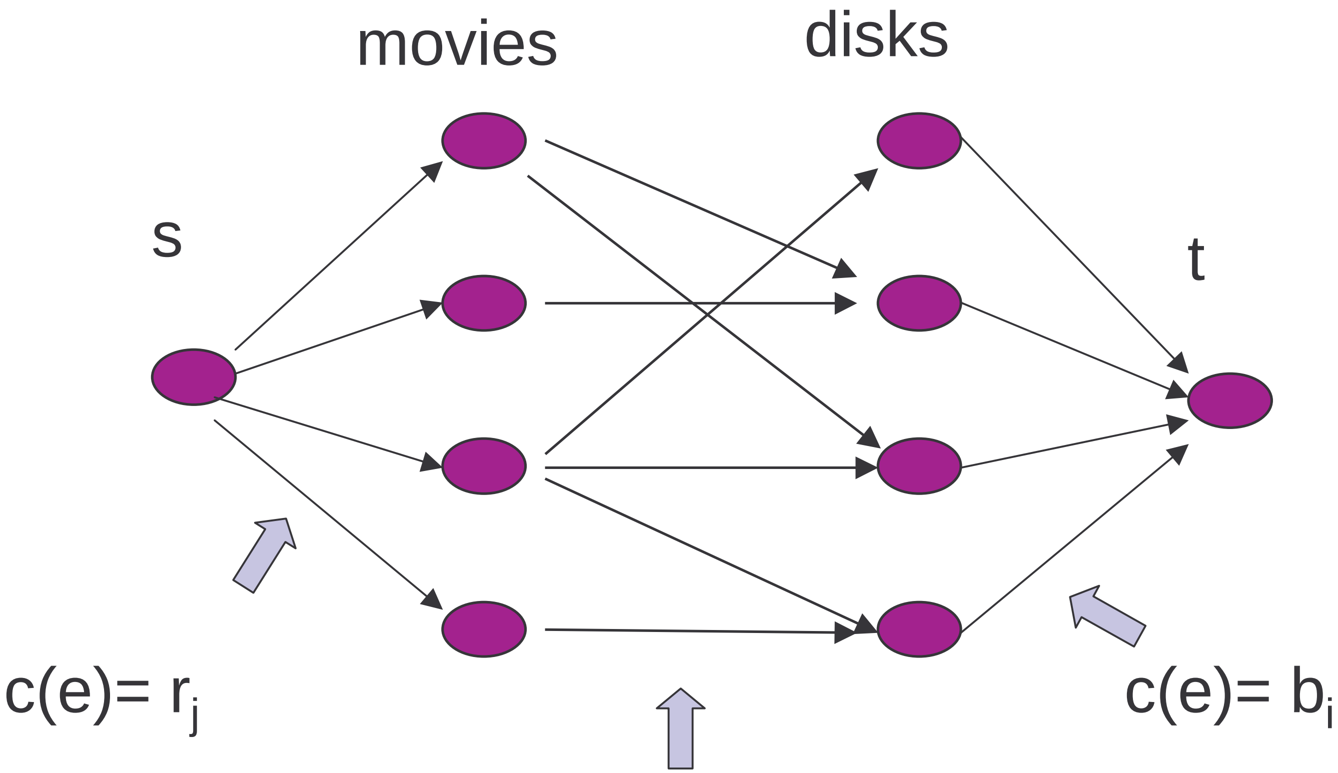
Network Connectivity

- What is the minimum number of links in the network such that if that many links go down, it is possible for nodes s and t to become disconnected?
- Solution using flow:

Video on Demand

- m storage devices (e.g., disks), The i-th disk is capable of supporting b_i simultaneous streams.
- k movies, one copy of each on some of the disks (this assignment is given as input).
- Given set of R movie requests, (r_j requests to movie j) how would you assign the requests to disks so that no disk is assigned more than bi requests and the maximum number of requests is served?

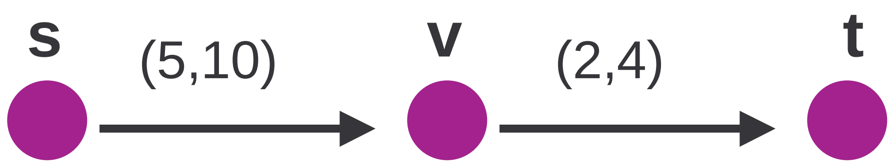
Video on Demand



A copy of movie j on disk i. $c(e)= \infty$

Other network flow problems:

1. Lower bounds on flow.
   – For each (v,w): $0 \leq lb(v,w) \leq f(v,w) \leq c(v,w)$
   – Not always possible:



2. Minimum flow
   - Want to send minimum amount of flow from source to sink, while satisfying certain lower and upper bounds on flow on each edge.

Other network flow problems:

3. Min-cost max-flow

Input: a graph (network) G where each edge (v,w) has associated capacity c(v,w), and ***a cost* cost(v,w)**.

Goal: Find a maximum flow of minimum cost.

The cost of a flow :

$$\Sigma_{f(v,w)>0} \text{cost}(v,w)f(v,w)$$

Out of all the maximum flows, which has minimal cost?

Weighted Assignment - Min-cost max-flow example

Production planning : n_2 servers need to serve n_1 clients. Each client specifies for each server how much he is ready to pay in order to be served by this server (this is given by revenue(client, server)).

- Goal: Maximize the profit.