Analysis of Algorithms, I

CSOR W4231.002

Eleni Drinea
Computer Science Department

Columbia University

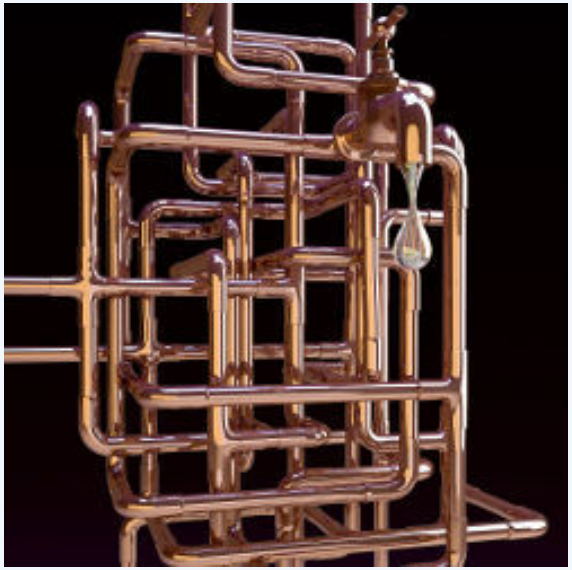
Network flows

Outline

Today

1. Flow networks
   - Applications
2. The residual graph and augmenting paths
3. The Ford-Fulkerson algorithm for max flow
4. Correctness of the Ford-Fulkerson algorithm
5. Application: max bipartite matching

Modeling transportation networks

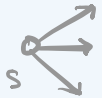


Source: Communications of the ACM, Vol. 57, No. 8

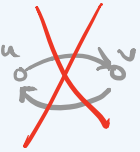
Can model a fluid network or a highway system by a **graph**:
edges carry *traffic*, nodes are *switches* where traffic gets diverted.

Flow networks

A flow network $G = (V, E)$ is a directed graph such that

1. Every edge has a capacity $c_e \geq 0$. *A1: integer capacities*
2. There is a single source $s \in V$.  *A2: no edge enters s*
3. There is a single sink $t \in V$.  *A3: no edge leaves t*

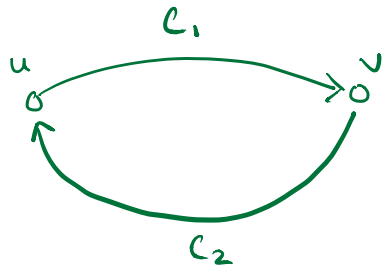
Two more assumptions for the purposes of the analysis



- *A4: If $(u, v) \in E$ then $(v, u) \notin E$.* → There are no anti-parallel edges in a flow network.
- *A5: Every $v \in V - \{s, t\}$ is on some s-t path.*
  Hence G has $m \geq n - 1$ edges.

If there are any nodes not on a s-t path, these nodes may be ignored since they are not useful.

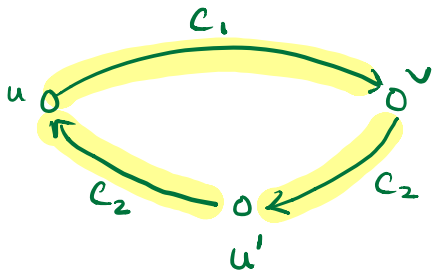
CANNOT USE THIS:



G

(u,v) and (v,u) are antiparallel, but Anti-parallel edges are not allowed, so construct an equivalent network G'.

USE THIS INSTEAD:



G'

Equivalent network with no anti-parallel edges.

$G = (V, E)$

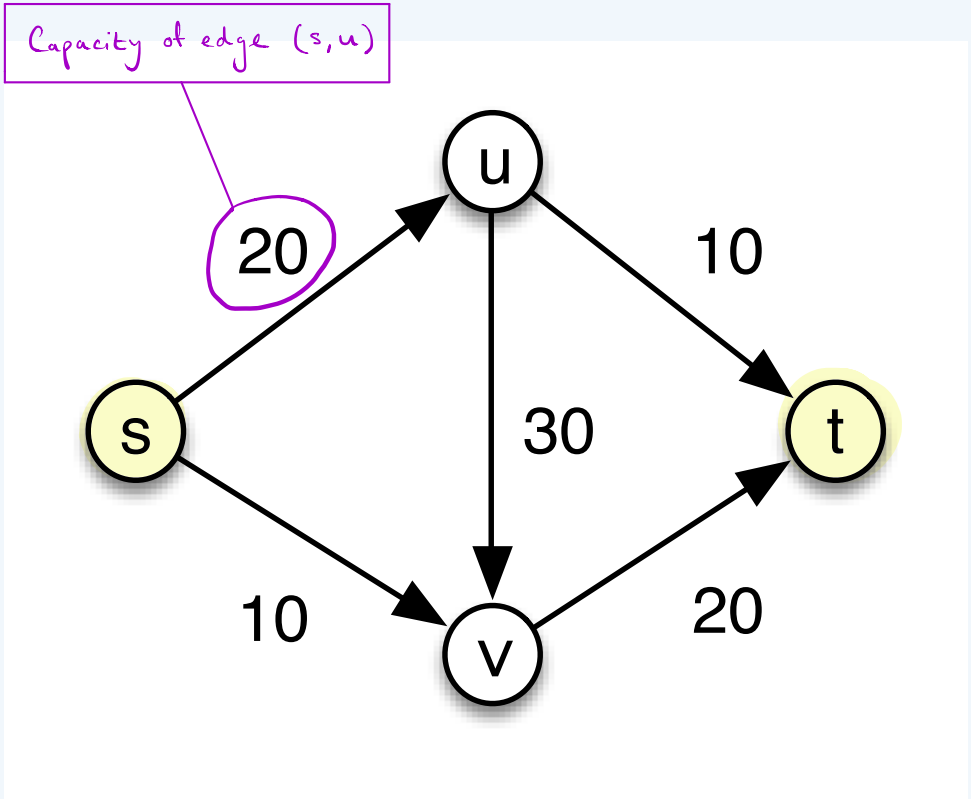
n $m = |E|$

Time to transform G into G'?

In the worst-case, it has another $\frac{m}{2}$ additional nodes for a total of $n + \frac{m}{2}$ nodes, and $\frac{m}{2}$ additional edges for a total of $m + \frac{m}{2}$ edges.

To construct G' from G it takes linear time, since in the worst case you only add $\frac{m}{2}$ nodes and $\frac{m}{2}$ edges. Therefore, this is an efficient transformation.

An example flow network



Flows

Given a flow network G, an *s-t flow* f in G is a function

$$f : E \to R^+.$$

Intuitively, the flow $f(e)$ on edge e is the amount of *traffic* that edge e carries.

It's best to think of the flow as an m-dim vector:

$$\vec{f} = \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} \begin{matrix} 1 \\ 2 \\ \vdots \\ \\ m \end{matrix}$$

Flows must satisfy two constraints:

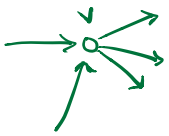
(1) Capacity constraints

$$\forall e \in E, \quad 0 \le f(e) \le c_e$$

(2) Flow conservation constraints

$$\forall v \in V - \{s, t\},$$

$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,x) \in E} f(v,x)$$



the amount of flow that comes into a vertex must equal the amount of flow that leaves the vertex.

} must hold for all $V - \{s, t\}$. Not s and t since they generate traffic.

Define:

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

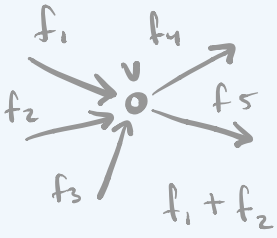
$$f^{out}(v) = \sum_{e \text{ out of } v} f(e)$$

$$\Rightarrow f^{in}(v) = f^{out}(v)$$

Two kinds of constraints that every flow must satisfy

1. Capacity constraints: for all $e \in E$, $0 \leq f(e) \leq c_e$.

2. Flow conservation: for all $v \in V - \{s, t\}$,



$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \tag{1}$$

In words, the flow into node v equals the flow out of v, or

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

A cleaner equation for flow conservation constraints

Define

1. $f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$
2. $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$

So we can rewrite equation (1) as: for all $v \in V - \{s, t\}$

$$f^{\text{in}}(v) = f^{\text{out}}(v) \tag{2}$$

The value of a flow

Definition 1.

The value of a flow f, denoted by $|f|$, is

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

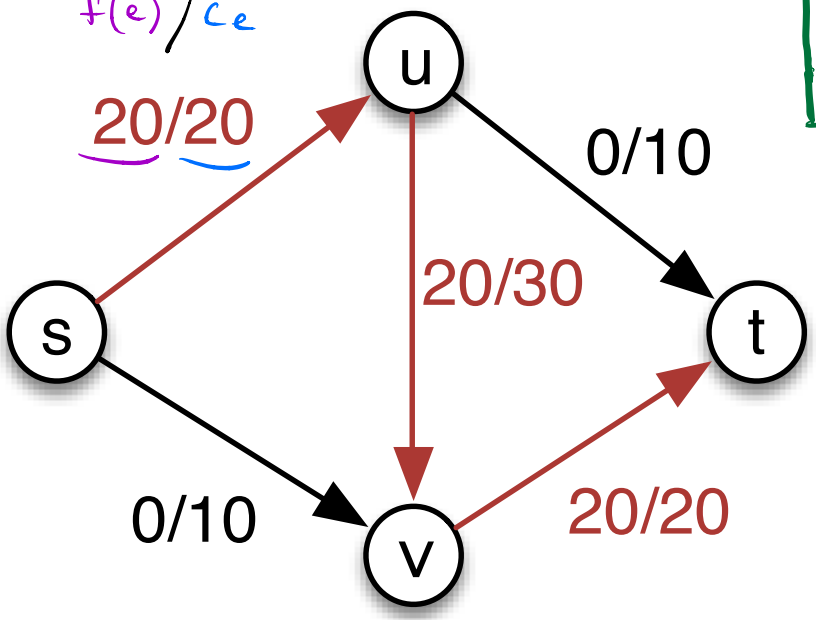
Exercise: show that $|f| = f^{\text{in}}(t)$.

An example flow of value 20

$|f| = f^{out}(s) = 20$

$\vec{f} = \begin{bmatrix} 20 \\ 0 \\ 20 \\ 0 \\ 20 \end{bmatrix} \begin{matrix} \leftarrow su \\ \leftarrow sv \\ \leftarrow uv \\ \leftarrow ut \\ \leftarrow vt \end{matrix}$

This is a valid flow since it follows the capacity constraints and the flow conserv. constraints.



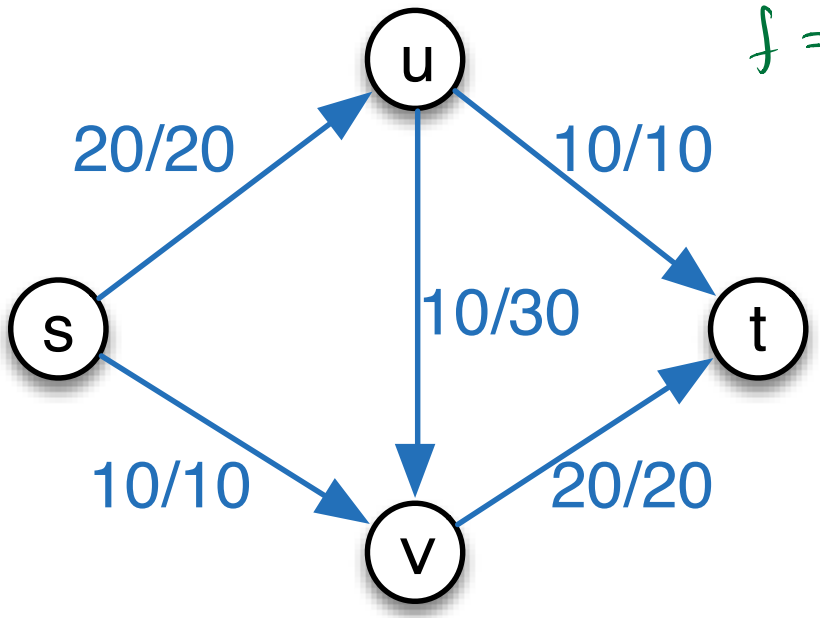
A flow f of value 20.

Can we get a larger flow? Yes. $\Rightarrow$ $\vec{f}' = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 10 \\ 20 \end{bmatrix} \Rightarrow |f| = 30$

A flow of value 30

$|f| = f^{out}(s) = 30$

This a max flow since all edges out of s are saturated and conservation constraint is satisfied.



$\vec{f} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 10 \\ 20 \end{bmatrix}$ su, sv, uv, ut, vt

A (max) flow of value 30.

$|f| = f^{out}(s) = 20 + 10 = 30$

Max flow problem

flow network

Input: (G, s, t, c) such that

- $G = (V, E)$ is a flow network;
- $s, t \in V$ are the source and sink respectively;
- c is the (integer-valued) capacity function.

Output: a flow of maximum possible value

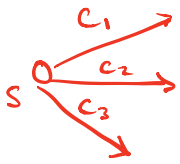
$\vec{f}$

$|f| = f^{out}(s)$

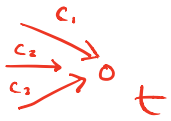
Constraints on max flow problem :

$|f| = f^{out}(s)$

① $|f| \leq \sum_{e \text{ out of } s} c_e$



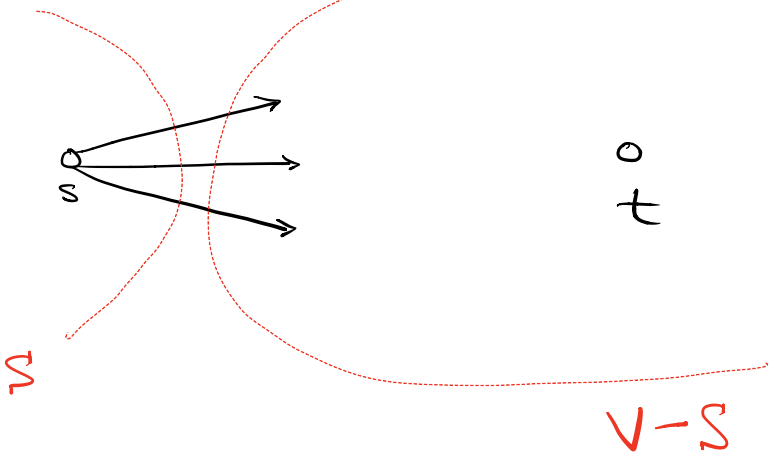
② $|f| \leq \sum_{e \text{ into } t} c_e$



←

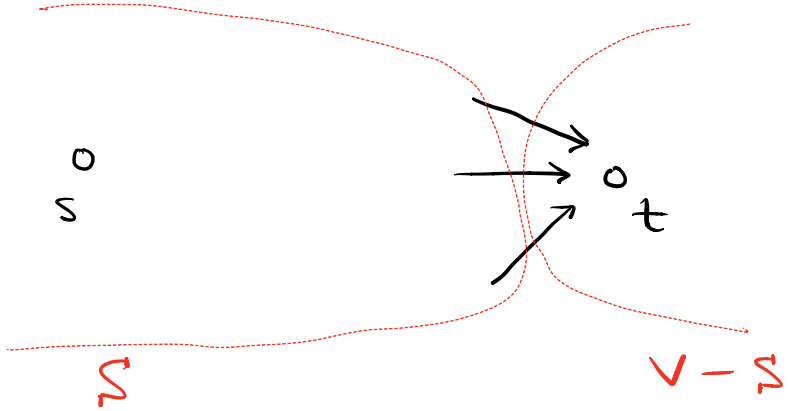
① Scenario 1

Put s in a set S by itself and every other node in V into the set $V-S$. Then look at the sum of the capacity of edges going out of S

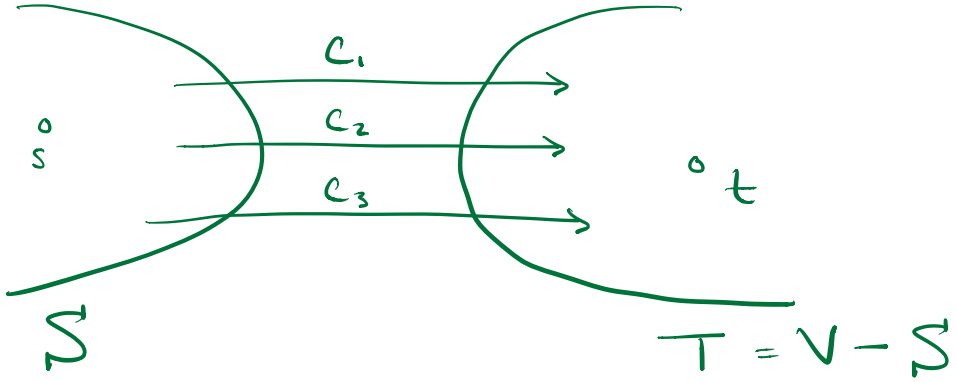


② Scenario 2:

$\forall v \in V \setminus \{t\}$ place $v \in S$, and put $t \in V-S$ by itself. Then look @ sum of capacity of edges leaving S.



Def'n: An s-t cut (S, T) $\longrightarrow$ $T = V - S$ is a bipartition of V s.t. $s \in S$ and $t \in T$.



$$|f| \leq \sum_{\substack{e \text{ crosses} \\ \text{from } S \text{ to } T}} c_e \triangleq c(S, T)$$

$c(S, T)$ is denoted as the capacity of the cut.

This holds for every cut. No matter what cut you choose, the flow will have to move from one side to the other.

Therefore, we get the following tighter bound:

$$|f| \leq \min_{(S,T) \text{ cut in } G} c(S, T)$$

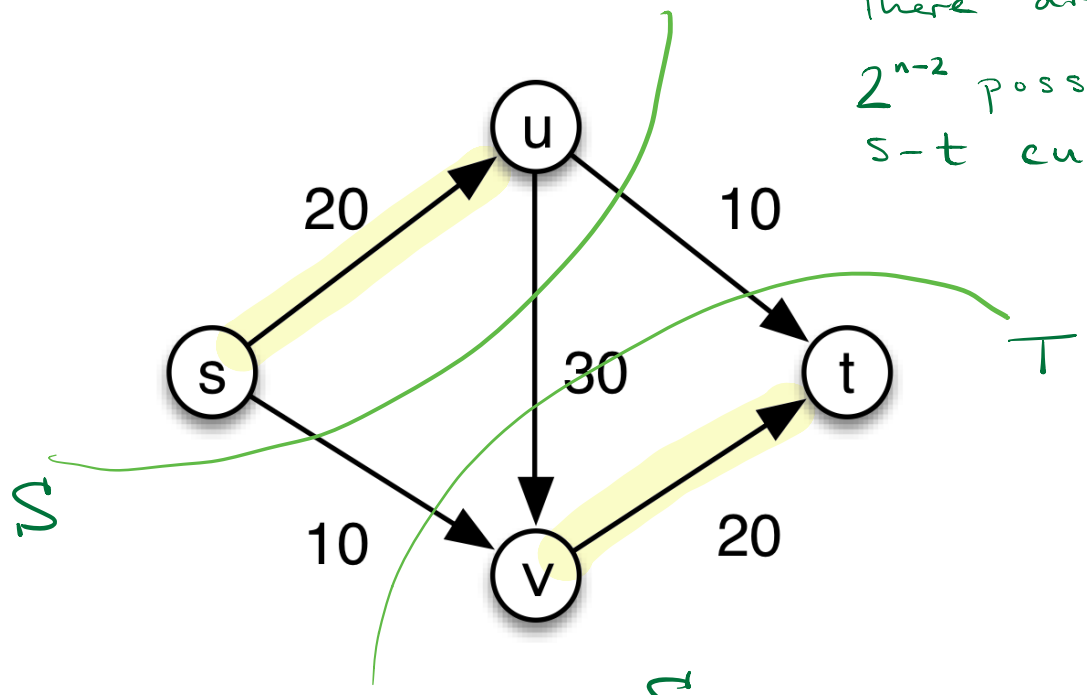
← since $|f|$ cannot exceed the capacity of ANY cut.

We will show next that

$$\max |f| = \min_{(S,T) \text{ cut in } G} c(S, T)$$

Examples of s-t cuts (S, T):

There are 2^{n-2} possible s-t cuts.



S

$(S, T) \longrightarrow c(S, T) = \sum_{\text{edge crosses from } S \text{ to } T} c_e$

$(\{s\}, \{u, v, t\}) \longrightarrow 30$

$(\{s, u, v\}, \{t\}) \longrightarrow 30$

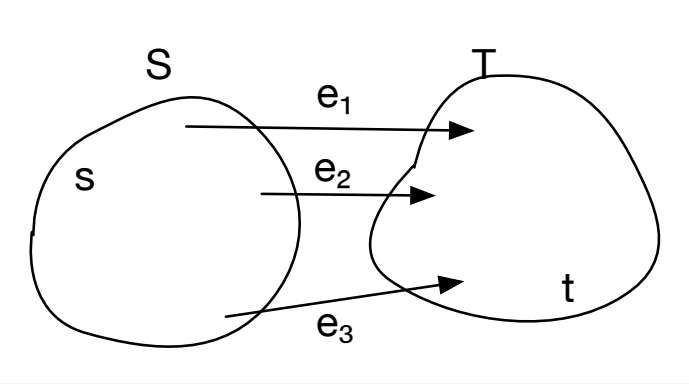
$(\{s, u\}, \{v, t\}) \longrightarrow 50$

$(\{s, v\}, \{u, t\}) \longrightarrow 40$

s-t cuts in flow networks

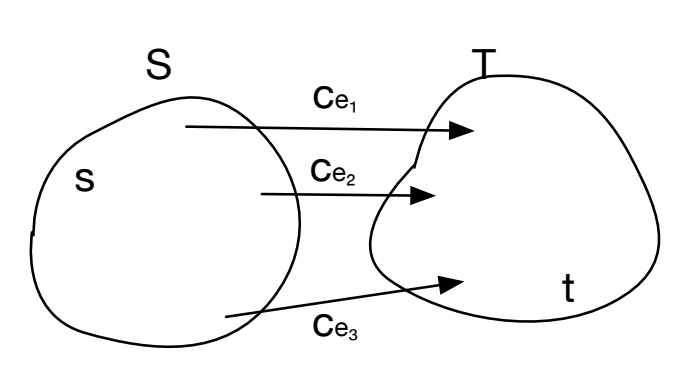
Definition 2.

An s-t cut (S, T) in G is a bipartition of the vertices into two sets S and T, such that $s \in S$ and $t \in T$.



A natural upper bound for the max value of a flow

- Flow f must cross (S, T) to go from source s to sink t.
- So it uses some (at most all) of the capacity of the edges crossing from S to T.



- So, intuitively, the value of the flow cannot exceed

$$\sum_{\text{e out of } S} c_e$$

Max flow and min cut

Definition 3.

The capacity $c(S,T)$ of an s-t cut (S,T) is defined as

$$c(S,T) = \sum_{\text{e out of } S} c_e.$$

△ Note asymmetry in the definition of $c(S,T)$!

So, *intuitively,* the value of the max flow is upper bounded by the capacity of *every* cut in the flow network, that is,

$$\max_f |f| \leq \min_{(S,T) \text{ cut in } G} c(S,T) \tag{3}$$

Since the flow cannot exceed the capacity of any S,T-cut, then clearly the max flow cannot exceed the minimum S-T cut.

Applications of max-flow and min-cut

These can be solved by constructing an appropriate flow network for the problem and then solving a max flow problem. This technique is called reduction.

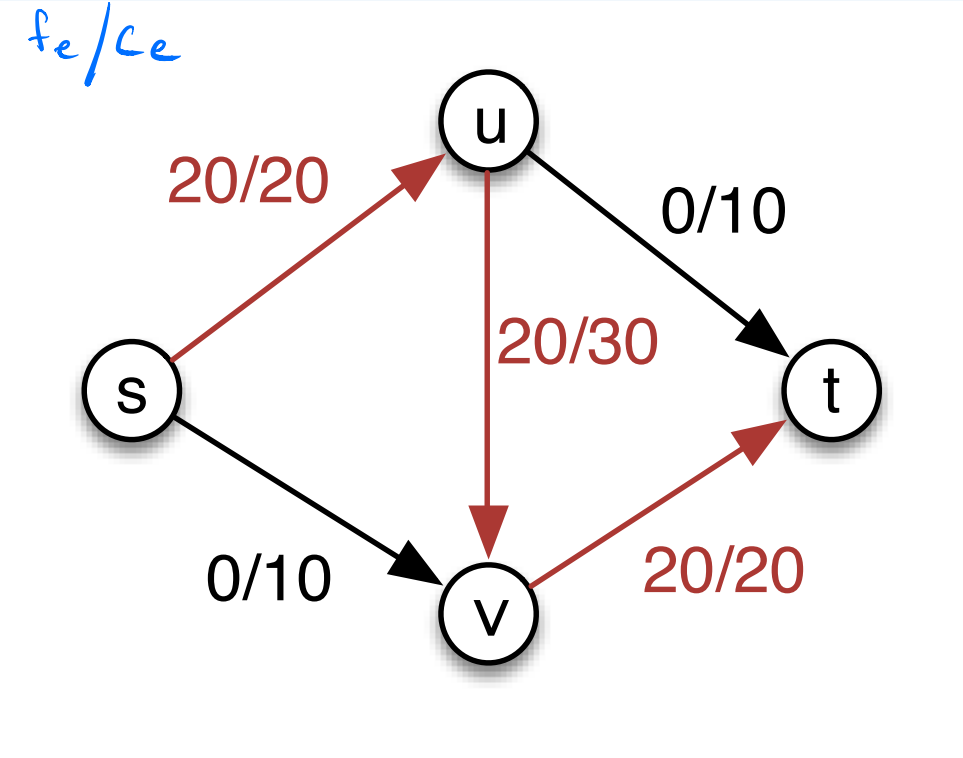
- Find a set of edges of smallest capacity whose deletion disconnects the network (min cut)
- Bipartite matching (max flow) —*coming up*
- Airline scheduling (max flow)
- Baseball elimination (max flow)
- Distribution of goods to cities (max flow)
- Image segmentation (min cut)
- Survey design (max flow) — output whether or not it's possible to design a survey on all products w/o asking too many questions to customers.
- ...

These are both optimization and yes/no problems.
Yes/no problems are often referred to as feasibility problems.
Yes/no problems can be answered/solved with dynamic programming.

Today

Now, we want to design an efficient algorithm to compute the maximum flow in a flow network because if we can we will be able to solve a great number of problems (above).

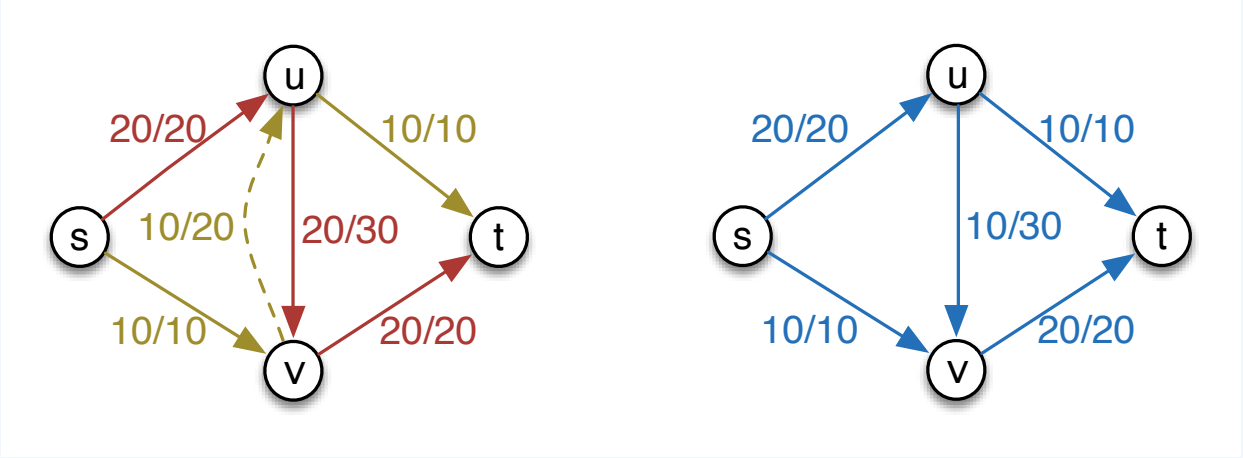
"Undoing" flow



A flow f of value 20.

Goal: *undo* 10 units of flow along (u, v), divert it along (u, t).

Pushing flow back



- "Push back" 10 units of flow along (v, u).
- Send 10 more units from s to t along the green path edges $(s, v), (v, u), (u, t)$.
- New flow f' (on the right) with value 30.

Pushing flow forward and backward

By pushing flow back on (v, u), we created an s-t path on which we are pushing flow

- forward, on edges with leftover capacity (e.g, (s, v));
- backward, on edges that are already carrying flow so as to divert it to a different direction (e.g., (u, v)).

* Identify an s-t path in G_f s.t.

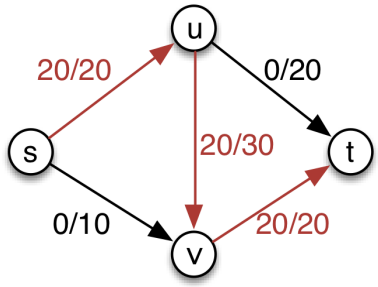
- push flow forward on edges with leftover capacity.

- (undo flow) push flow backward on edges that already carry some flow.

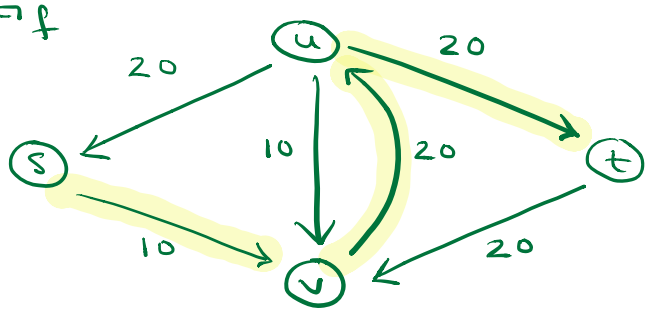
Ex:

Values on edges in G_f are the residual capacities.

(G, f)



G_f



Is there an s-t path in the residual graph along which we can push more flow?

Yes: $s \to v \to u \to t$ where we can push at most 10 more units of flow (the minimum capacity on this path, i.e., the bottleneck capacity).

The residual graph G_f of a flow network G w.r.t. a flow f has

- the same vertices as G
- $\forall e \in G$ with $f(e) < c_e$, (edges with leftover capacity.)

  introduce e in G_f with residual capacity $c_f(e) = c_e - f(e) > 0$ (leftover capacity)

  FORWARD edges
- $\forall e = (u,v) \in G$ with $f(e) > 0$,

  introduce $e^r = (v,u)$ in G_f with residual capacity $c_f(e^r) = f(e) > 0$

  BACKWARD edges

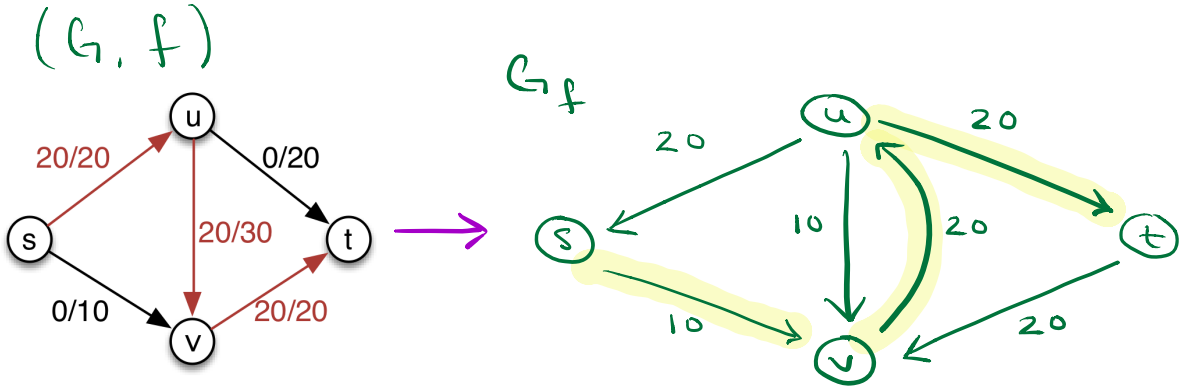
FORWARD edges in same direction in G_f as in G.
BACKWARD edges in opposite direction in G_f as in G.

$\Rightarrow$ • G_f has $\leq 2m$ edges

(since for every edge $e \in G$, you can introduce 2 residual edges in G_f.)

Ex:

Values on edges in G_f are the residual capacities.



Is there an s-t path in the residual graph along which we can push more flow?

Yes: $s \to v \to u \to t$ where we can push at most 10 more units of flow (the minimum capacity on this path, i.e., the bottleneck capacity).

Let P be a simple s-t path in G_f.

Let $\underbrace{c(P)}_{\text{bottleneck capacity of } P} \triangleq \min_{e \in P} c_f(e)$

($\Rightarrow$ $c(P)$ is the maximum amount of flow that we can push along the path.)

Let P be a simple s-t path in G_f.

Let $c(P) \triangleq \min_{e \in P} c_f(e)$

Augment (f, P):

for $\forall e \in P$:

- if e is FORWARD, then

$$f'(e) = f(e) + c(P)$$

the bottleneck capacity of the path

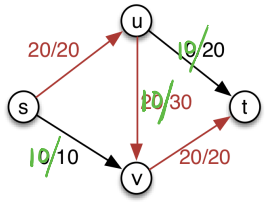
- if $e = (u,v)$ is BACKWARD, then

$$f'(v,u) = f(v,u) - c(P)$$

Return f'. // We only change flow for edges on the path P.

Ex:

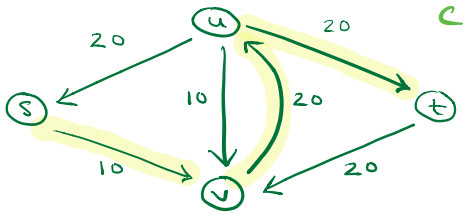
(G, f)



G_f

$P = \{s, v, u, t\}$

$c(P) = 10$



Values on edges in G_f are the residual capacities.

(FORWARD) $f'(s,v) = f(s,v) + c(P) = 0 + 10 = 10$

(BACKWARD) $f'(u,v) = f(u,v) - c(P) = 20 - 10 = 10$

(FORWARD) $f'(u,t) = f(u,t) + c(P) = 0 + 10 = 10$

We only update edges on the s-t path in the original flow network / graph G.

Augment (f, P):

$\forall e \in P$

- If e is FORWARD, ← in original graph

$$f'(e) = f(e) + c(P)$$

- If e is BACKWARD, ← in original graph

$$f'(e^r) = f(e^r) - c(P)$$

$\forall e \notin P$

- $f'(e) = f(e)$

We only update edges on the s-t path in the original flow network / graph G.

FORWARD edges appear in the same direction in G_f as in the original graph G.

BACKWARD edges appear in the opposite direction in G_f compared to the direction in G.

The residual graph G_f

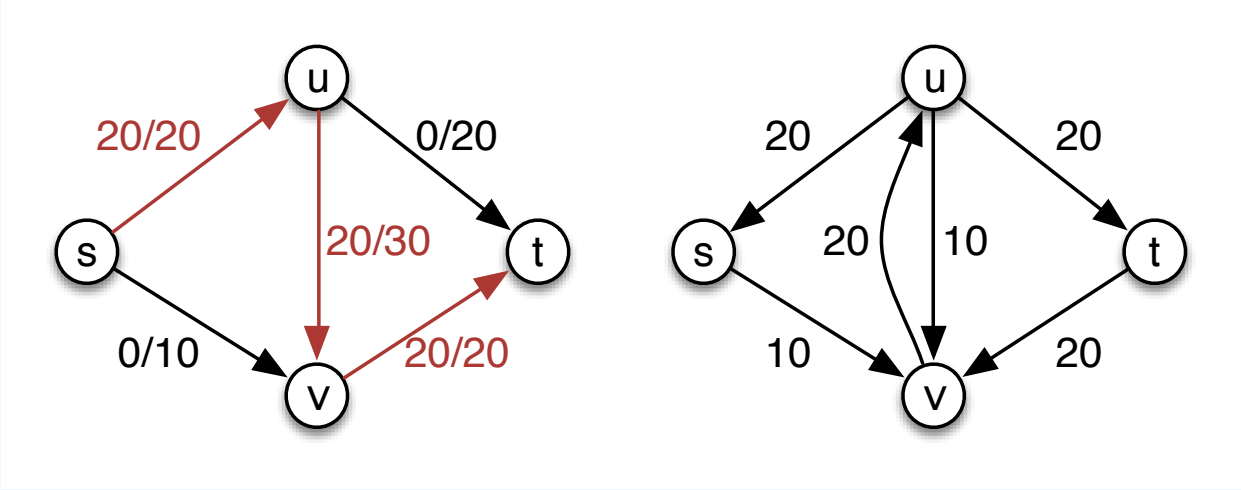
Definition 4.

Given flow network G and flow f, the residual graph G_f has

- the same vertices as G;
- for every edge $e = (u, v) \in E$ with $f(e) < c_e$, an edge $e = (u, v)$ with residual capacity $c_f(e) = c_e - f(e)$ (forward edge);
- for every edge $e = (u, v) \in E$ such that $f(e) > 0$, an edge $e^r = (v, u)$ in G_f with residual capacity $c_f(e^r) = f(e)$ (backward edge).

So G_f has $\leq 2m$ edges and every $e \in G_f$ has $c_f(e) > 0$.

Example residual graph



Left: a graph G and a flow f of value 20.
Right: the residual graph G_f for flow network G and flow f.

The residual graph G_f as a roadmap for augmenting f

1. Let P be a **simple** s-t path in G_f.
2. Augment f by pushing extra flow on P.

Question: How much flow can we push on P without violating capacity constraints in G_f?

The residual graph G_f as a roadmap for augmenting f

1. Let P be a **simple** s-t path in G_f.
2. Augment f by pushing extra flow on P.

Question: How much flow can we push on P without violating capacity constraints in G_f?

Definition 5.

The **bottleneck** capacity $c(P)$ of a simple path P is the minimum residual capacity of **any** edge of P. In symbols

$$c(P) = \min_{e \in P} c_f(e).$$

The residual graph G_f as a roadmap for augmenting f

1. Let P be a **simple** s-t path in G_f.
2. Augment f by pushing extra flow on P.

Question: How much flow can we push on P without violating capacity constraints in G_f?

Definition 5.

The **bottleneck** capacity $c(P)$ of a simple path P is the minimum residual capacity of **any** edge of P. In symbols

$$c(P) = \min_{e \in P} c_f(e).$$

Answer: The max amount of flow we can safely push on **every** edge of P is $c(P)$.

The augmented flow f'

Let P be an augmenting path in the residual graph G_f.

We obtain an **augmented flow** f' as follows:

1. For a **forward** edge $e \in P$, set $f'(e) = f(e) + c(P)$
2. For a **backward** edge $e^r = (u, v) \in P$, let $e = (v, u) \in G$; set $f'(e) = f(e) - c(P)$
3. For $e \in E$ but not in P, $f'(e) = f(e)$.

Claim 1.

f' is a flow.

- Prove by showing that flow conservation and capacity constraints hold.

Pseudocode for subroutine `Augment`

Input: a flow f, and an augmenting path P in G_f
Output: the augmented flow f'

```
Augment(f, P)
    for each edge (u, v) ∈ P do
        if e = (u, v) is a forward edge then
            f(e) = f(e) + c(P)
        else
            f(v, u) = f(v, u) − c(P)
        end if
    end for
    return f
```

Today

1. Flow networks
   - Applications
2. The residual graph and augmenting paths
3. The Ford-Fulkerson algorithm for max flow
4. Correctness of the Ford-Fulkerson algorithm
5. Application: max bipartite matching

The Ford-Fulkerson algorithm — Algorithm that computes maximum flow

INTEGERS

- Restrict capacities to be integers to guarantee the algorithm terminates after a finite # of steps.

Ford-Fulkerson$(G = (V, E, c), s, t)$

  for all $e \in E$ **do** $f(e) = 0$
  end for
  while there is an s-t path in G_f **do**
    Let P be a simple s-t path in G_f
    $f' =$ **Augment**(f, P)
    Set $f = f'$
    Set $G_f = G_{f'}$
  end while
  Return f

// Augment procedure given above.

Returns the maximum flow

(Analysis of run-time below).

The Ford-Fulkerson algorithm — Algorithm that computes maximum flow

INTEGERS

- Restrict capacities to be integers to guarantee the algorithm terminates after a finite # of steps.

Ford-Fulkerson$(G = (V, E, c), s, t)$

for all $e \in E$ **do** $f(e) = 0$

end for

while there is an s-t path in G_f **do**

terminates if flow increase by ≥ 1 in every iteration.

- Let P be a simple s-t path in G_f → $O(n+m) = O(m)$
- $f' = \textbf{Augment}(f, P)$ → $O(n) = O(m)$ // Augment procedure given above.
- Set $f = f'$ → $O(m)$
- Set $G_f = G_{f'}$ → $O(n+m) = O(m)$

since path has $\geq n-1$ edges. (simple path)

end while

Return f

Let $U = \max_{e \in E} c_e$

$$|f| = f^{out}(s) \leq \sum_{e \text{ out of } s} c_e \leq nU$$

(since source can have at most $n-1$ out neighbors)

$\Rightarrow$ Max # iterations in while-loop is nU.

The Ford-Fulkerson algorithm

- We enter the while-loop at most nU times. and increase the value of the flow by ≥ 1 each time.

Ford-Fulkerson$(G = (V, E, c), s, t)$

for all $e \in E$ **do** $f(e) = 0$
end for
while there is an s-t path in G_f **do**
  Let P be a simple s-t path in G_f
  $f' =$ **Augment**(f, P)
  Set $f = f'$
  Set $G_f = G_{f'}$
end while
Return f

Find s-t path using either BFS or DFS.

→ $O(n+m) = O(m)$

→ $O(n) = O(m)$

→ $O(m)$

→ $O(n+m) = O(m)$

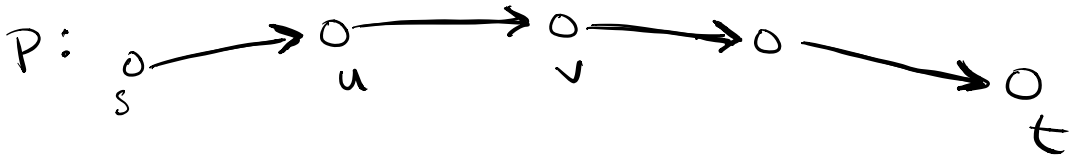
$O(n) = O(m)$ since every node lies on a s-t path and we proved that $m \geq n-1$

TOTAL RUNNING-TIME: $nU \cdot O(m) = O(nmU)$

Proving that flow increases by at least 1 in every iteration of the while-loop:

Pf:

Consider some s-t path P in G_f:



$|f| = f^{out}(s)$ $\quad$ (s,u) is FORWARD

$f'(s,u) = f(s,u) + c(P)$ $\quad$ (Since (s,u) is necessarily a FORWARD edge)

$$|f'| = |f| - f(s,u) + f'(s,u)$$

$$= |f| - f(s,u) + f(s,u) + c(P)$$

$$= |f| + c(P)$$

$c(P) > 0$ since all residual capacities in G_f are > 0

$\Rightarrow c(P) \geq 1$ $\quad$ Since $c(P) > 0$ and $c(P) \in \mathbb{Z}$

$\Rightarrow |f'| = |f| + c(P) \geq |f| + 1$

$\Rightarrow |f'| \geq |f| + 1$

$\Rightarrow$ Every time you enter the while-loop, you increase the flow value by at least 1. //

□

Running time analysis

The algorithm terminates if the following facts are both true.

Fact 6.

Every iteration of the while loop returns a flow increased by an integer amount.

Fact 7.

There is a finite upper bound to the flow.

To obtain the Min-Cut :

We can obtain the min cut efficiently. Upon termination of the FF algorithm, run BFS or DFS in the final residual graph G_f from the source node.

All nodes reachable from s is the cut for S^*; all other nodes are in T^*. Indeed,

S^* = set of nodes reachable from s in G_f upon termination of FT.

Running time analysis

The algorithm terminates if the following facts are both true.

Fact 6.

Every iteration of the while loop returns a flow increased by an integer amount.

Fact 7.

There is a finite upper bound to the flow.

Proof of Fact 7.

Let U be the largest edge capacity, that is, $U = \max_e c_e$. Then

$$|f| \leq \sum_{e \text{ out of } s} c_e \leq nU.$$

f increases by an integer amount after Augment(f, P)

Proof of Fact 6.

It follows from the following claims.

Claim 2.

During execution of the Ford-Fulkerson algorithm, the flow values $\{f(e)\}$ and the residual capacities in G_f are all integers.

Since $\mathbb{Z}$ is closed under addition.

Claim 3.

Let f be a flow in G and P a simple s-t path in G_f with residual capacity $c(P) > 0$. Then after Augment(f, P)

$$|f'| = |f| + c(P) \geq |f| + 1.$$

□

f increases by an integer amount after Augment(f, P)

Proof of Claim 3.

Recall that $|f| = f^{\text{out}}(s)$.

1. Since P is an s-t path, it contains some edge out of s, say (s, u).
2. Since P is simple, it does not contain any edge entering s (P is in G_f, where there could be edges entering s!).
3. Since no edge enters s in G, (s, u) is a forward edge in G_f, thus its augmented flow is $f(s, u) + c(P) \geq f(s, u) + 1$.
4. Since no other edge going out of s is updated, it follows that the value of f' is

$$|f'| = |f| + c(P) \geq |f| + 1.$$

Running time of Ford-Fulkerson

$U \rightarrow \underbrace{11\cdots1}_{U}$

1. Claim 3 guarantees at most nU iterations. $U \rightarrow \log_b U$ bits (But not U bits)
2. The running time of each iteration is bounded as follows:
   - $O(m+n)$ to create G_f using adjacency list representation.
   - $O(m+n)$ to run BFS or DFS to find the augmenting path.
   - $O(n)$ for Augment(f, P) since P has at most $n-1$ edges.

   $\Rightarrow$ Hence one iteration requires $O(m)$ time.

The running time of Ford-Fulkerson is $O(mnU)$. size: $\log_b U$

Definition 8 (Pseudo-polynomial algorithms).

An algorithm is pseudo-polynomial if it is polynomial in the size of the input when the **numeric** part of the input is encoded in **unary**.

Remark 1.

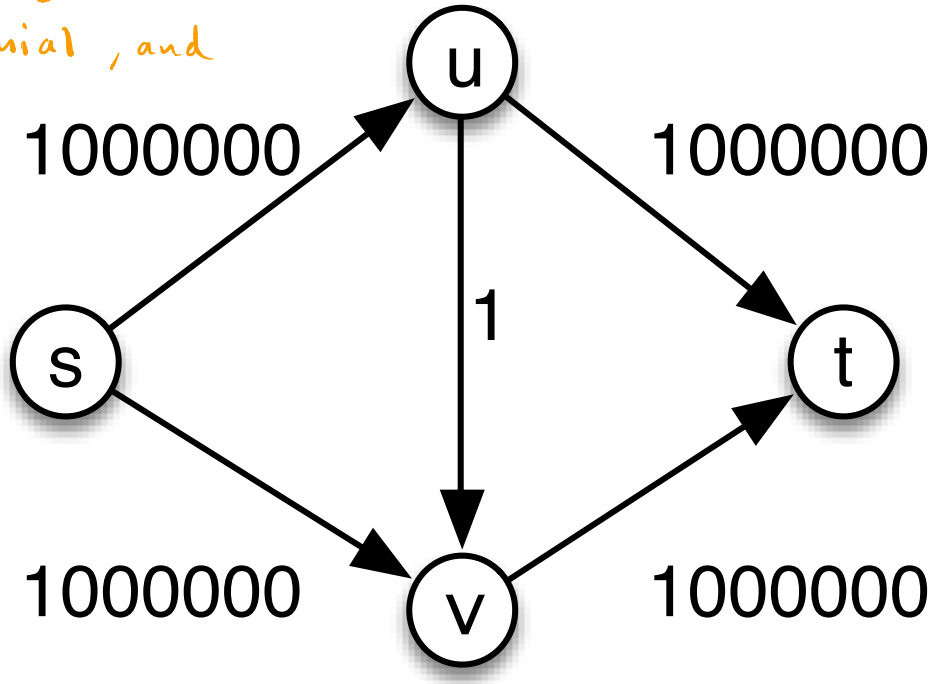
Ford-Fulkerson is a pseudo-polynomial time algorithm.

since it depends on actual #'s and not on the size of these #'s.

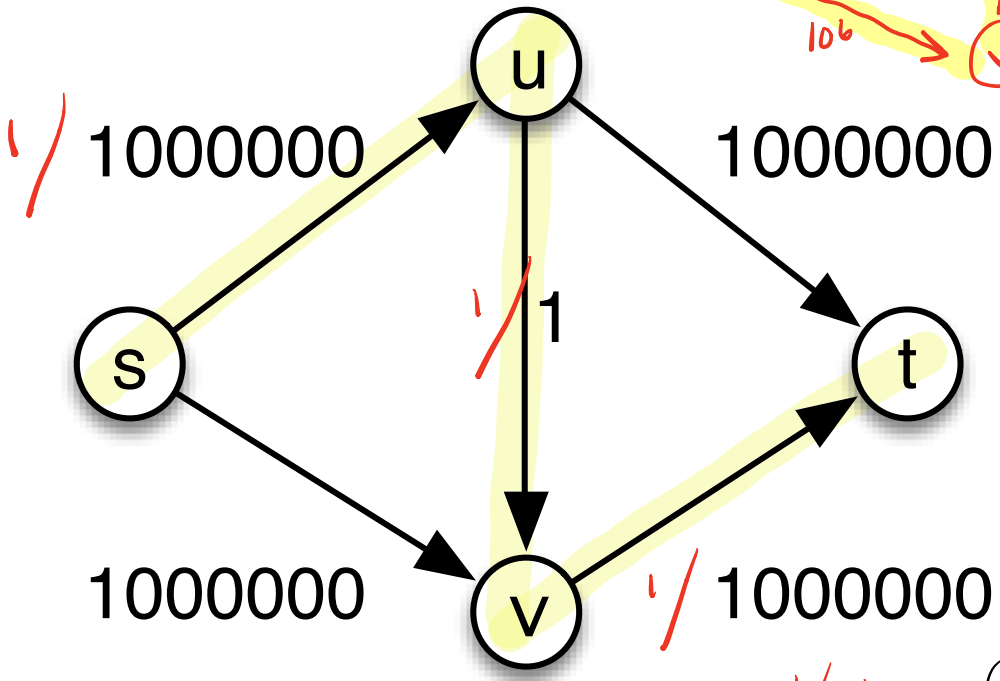
Running time of FF could be an issue for large values of U

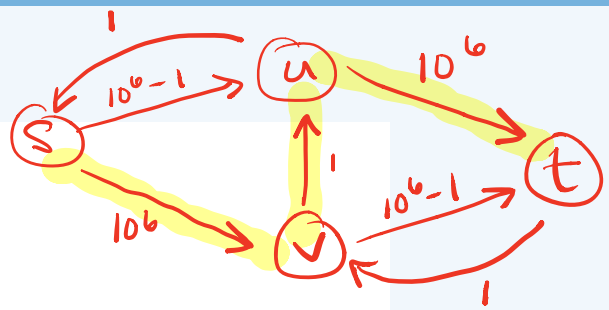
Problems with pseudo-polynomial running times

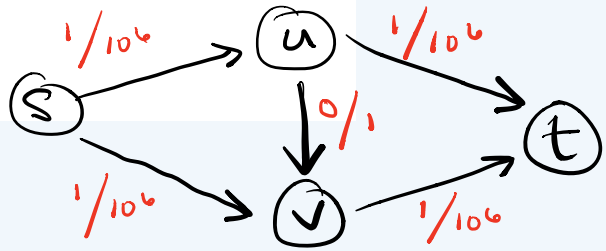
Algorithms that depend on actual numbers (like U) and not on the size of these numbers (i.e., not the # of bits we need to encode these numbers) are called pseudo-polynomial, and this is not a desired behavior because they are actually exponential in the input size. That is, the numeric part of the input (U in this case) truly has size $\log_b U$ bits (not U bits). You're not using U once to store U.



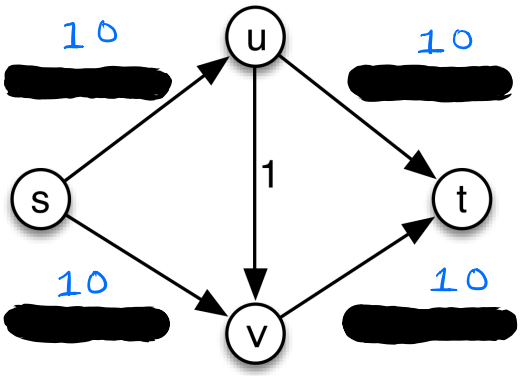
Problems with pseudo-polynomial running times







If you continue to get so unlucky, it will take 2×10^6 iterations to get max flow.



U = max capacity

U	size of U (# of digits)	# iterations (worst-case)
1	1	2
10	2	$2 \cdot 10$
100	3	$2 \cdot 10^2$
1000	4	$2 \cdot 10^3$
10000	5	$2 \cdot 10^4$

This is why pseudo polynomial algorithms are undesirable. → The running time (represented by the # of iterations here) grows exponentially in the size of the numeric part of the input.

- Although your input has small size since only using small # of bits to represent the numbers, the actual running time grows exponentially in the input size.

Improved algorithms

Go back to 3/30 lecture @ 1:00:00 to review this slide.

Takeaway from this slide: Max Flow can be solved in polynomial time. It's an easy problem.

- FF can be made polynomial: use BFS instead of DFS
  - Edmonds-Karp: $O(nm^2)$, Dinitz: $O(n^2m)$, other improvements: $O(nm \log n)$, $O(n^3)$
- **Unit** capacities: $O(\min\{m^{3/2}, mn^{2/3}\})$ [EvenTarjan1975]
  - Improved for sparse graphs: $\tilde{O}(m^{10/7})$ [Madry2013]
- **Integral** capacities: $O(\min\{m^{3/2}, mn^{2/3}\} \log(n^2/m) \log U)$ [GoldbergRao1998]
  - Improved: $\tilde{O}(m\sqrt{n} \log^2 U)$ [LeeSidfort2014]; *also improves for dense graphs with unit capacities*
- **Real** capacities: $O(nm \log(n^2/m))$
  - Improved: $O(nm)$ [Orlin2013]

$O(m^{3/2} \cdot \log U)$ for dense graphs

For dense graphs: $\tilde{O}((m + n^{1.5}) \cdot \log U)$

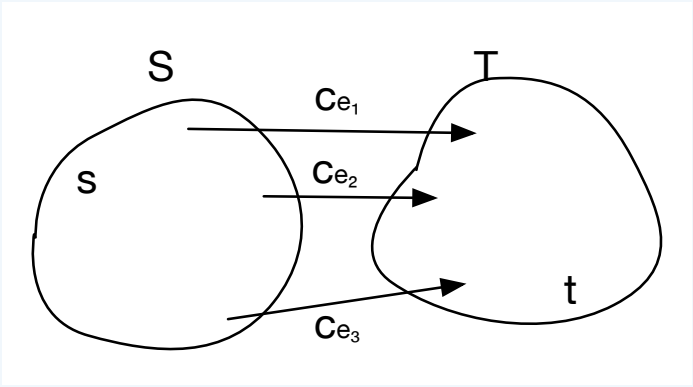
Today

$m = |E|$, $n = |V|$, for this flow network $m \geq n-1 \Rightarrow O(n) = O(m)$.
so if procedure takes $O(n)$ time $\Rightarrow$ it is also $O(m)$

A natural upper bound for the max value of a flow

- An s-t cut (S, T) in G is a bipartition of the vertices into two sets S and T, such that $s \in S$ and $t \in T$.

$= V - S$



- The capacity $c(S, T)$ of s-t cut (S, T) is $\sum_{\text{e out of } S} c_e$.
- Then intuitively

$$\max_f |f| \leq \min_{(S,T) \text{ cut in } G} c(S, T) \tag{4}$$

Even if $\max|f| = \min c(S,T)$ (which is the case), it still does not provide and easy way to find max flow. Indeed, there are an exponential # of cuts.

Roadmap for proving optimality of Ford-Fulkerson

Let f be the flow *upon termination* of the Ford-Fulkerson algorithm. Recall that $|f| = f^{\text{out}}(s)$.

1. Exhibit a specific s-t cut (S^*, T^*) in G such that

   ① $|f| = c(S^*, T^*)$

   $|f|$ = capacity of the cut (S^*, T^*)

   $c(S,T) = \sum_{e \text{ crosses from } S \text{ to } T} c_e$

2. Show that $|f|$ (value of any flow) cannot exceed the capacity of **any** cut in G.

   ② $\forall (S,T)$ cut, $|f| \leq c(S,T)$

3. Conclude that f is a maximum flow.
   - And (S^*, T^*) is a cut of minimum capacity.

   ① & ② $\Rightarrow$ ③

$|f'| \overset{?}{\geq} |f| = c(S^*, T^*)$

$c(S',T') < c(S^*,T^*) = |f|$, a contradiction since the value of any flow cannot exceed the value of any cut.

Roadmap for proving optimality of Ford-Fulkerson

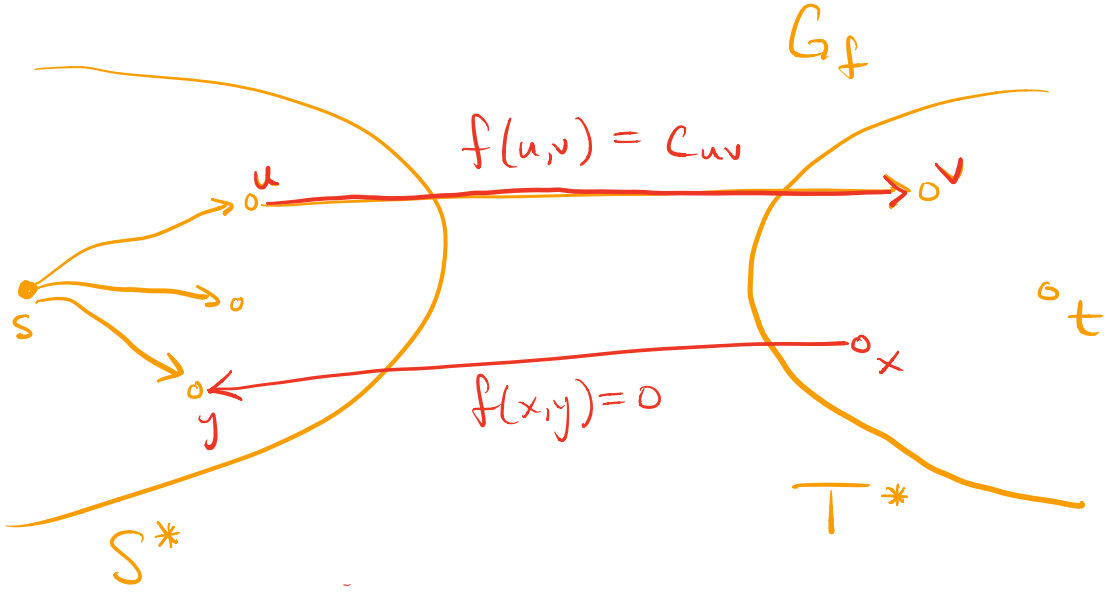
Recall that $|f| = f^{\text{out}}(s)$. We will prove the following.

1. For any flow f, the value of the flow $|f|$ cannot exceed the capacity of **any** cut in G.
2. Let f be the flow *upon termination* of the Ford-Fulkerson algorithm. We will exhibit a specific cut (S^*, T^*) such that the value of f equals the capacity of (S^*, T^*). In symbols,

$$|f| = c(S^*, T^*)$$

From 1., 2., we can conclude that f is a maximum flow.

- And (S^*, T^*) is a cut of minimum capacity.



S^* = set of nodes reachable from s in G_f upon termination of FT.

(S^*, T^*) is an s-t cut in G_f.

(S^*, T^*) is an s-t cut in G.

- $\nexists\ (u,v) \in G_f$ with $u \in S^*$, $v \in T^*$
- If $\exists\ (u,v) \in G$ with $u \in S^*$ and $v \in T^*$, then $f(u,v) = c_{uv}$; (otherwise, the edge (u,v) would appear in the residual graph G_f.)
- If $(x,y) \in G$ with $x \in T^*$, $y \in S^*$, then $f(x,y) = 0$

$$\text{net flow}(S,T) \triangleq f^{out}(S) - f^{in}(S)$$

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

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

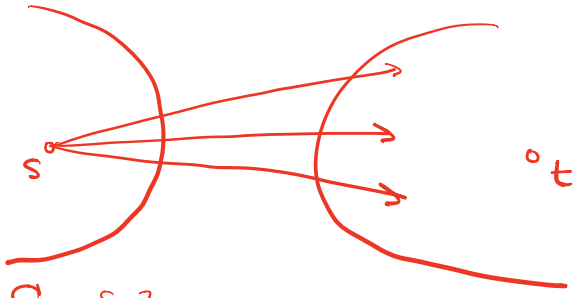
net flow (S,T) is the net amount of flow that makes it across the cut.

$$\text{net flow}(S^*,T^*) = f^{out}(S^*) - f^{in}(S^*)$$

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

$$= \sum_{e \text{ out of } S^*} c_e - \sum_{e \text{ into } S^*} 0$$

$$= \sum_{e \text{ out of } S^*} c_e = c(S^*,T^*)$$

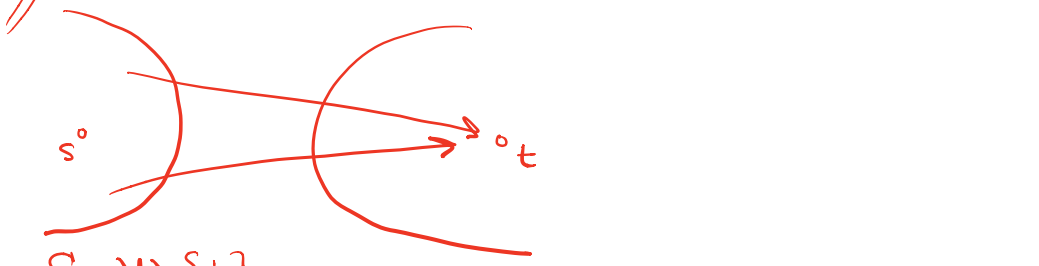


$S = \{s\}$ $\quad T = V \setminus \{s\}$

$$\text{net flow}(S,T) = f^{out}(S) - f^{in}(S)$$

$$= \sum_{e \text{ out of } S} f(e) - \cancel{\sum_{e \text{ into } S} f(e)}^{\,0}$$

$$= f^{out}(s) = |f|$$



$S = V \setminus \{t\}$ $\quad T = \{t\}$

$$\text{netflow}(S,T) = f^{out}(S) - f^{in}(S)$$

$$= \sum_{e \text{ out of } S} f(e) - \cancel{\sum_{e \text{ into } S} f(e)}^{\,0}$$

$$= f^{in}(t) = |f|$$

Lemma: $\forall$ s-t cuts (S,T),

$$\text{net flow}(S,T) = |f|$$

Proof:

$$|f| = f^{out}(s) = f^{out}(s) - \overbrace{f^{in}(s)}^{=0}$$

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

($=0$ for $\forall v \neq s$ because of Flow Conserv.)

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

$$= \sum_{v \in S} \left(\sum_{(v,u) \in E} f(v,u) - \sum_{(u,v) \in E} f(u,v) \right)$$

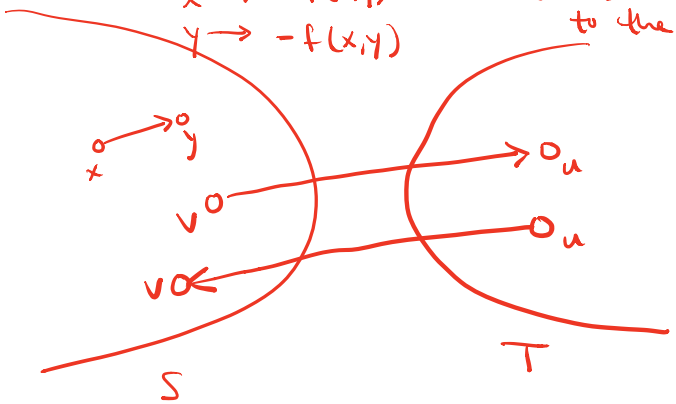
$$= \sum_{v \in S} \sum_{\substack{u \in T \\ (v,u) \in E}} f(v,u) - \sum_{v \in S} \sum_{\substack{u \in T \\ (u,v) \in E}} f(u,v)$$

edges in S appear twice in the sum since both nodes are in S.

$x \longrightarrow +f(x,y)$

$y \longrightarrow -f(x,y)$

$\Rightarrow$ $\forall e$ that fully lie in S, there net contribution to the sum above is 0.



- edges that cross from S to T appear only once in the sum, un-negated (added).
- edges crossig from T to S appear once in sum and are subtracted.

(cont'd...)

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

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

$$= \text{net flow}(S, T)$$

$$\Rightarrow |f| = \text{net flow}(S, T) \quad \square$$

$$\Rightarrow |f| = c(S^*, T^*)$$

We proved the following Lemma above (for the Roadmap.)

Lemma : $\forall$ s-t cuts (S,T), net flow $(S,T) = |f|$

Now we want to show that $|f| \leq c(S,T)$, for every s-t cut.

Consider :

$$|f| = \text{net flow}(S,T) = f^{out}(S) - f^{in}(S)$$

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

minimum amount of flow back into S is 0.

$$\leq \sum_{e \text{ out of } S} c_e - 0$$

$$= c(S,T)$$

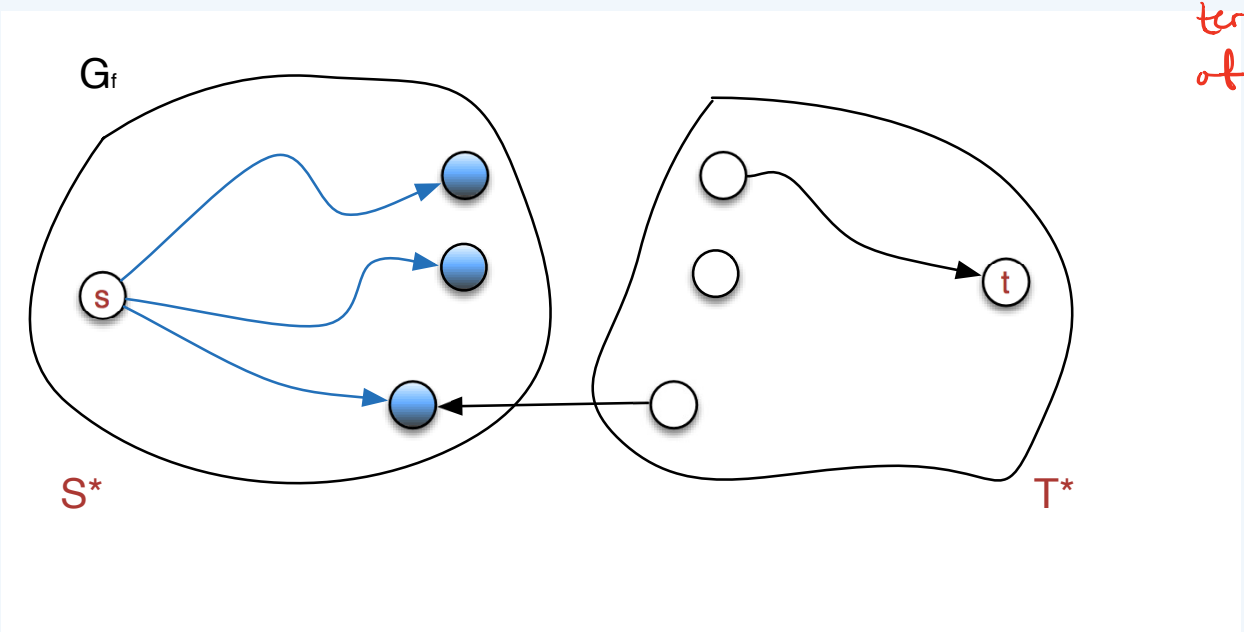
$$\Rightarrow |f| \leq c(S,T)$$

Ford-Fulkerson terminates when $\nexists$ s-t path in G_f

Consider the residual graph G_f *upon termination* of the algorithm. Let (S^*, T^*) be the cut in G_f where

- S^* is the set of nodes reachable from the source s;
- T^* contains every other node.

Use BFS/DFS to determine these upon termination of FF.



Is (S^*, T^*) also a cut in G?

On the cut (S^*, T^*)

1. (S^*, T^*) is an s-t cut: that is, $s \in S^*$, $t \in T^*$. *Why?*

2. In G_f, no edge crosses from S^* to T^*. *Why?*

3. Hence, if $e = (x, y) \in E$ with $x \in S^*$ and $y \in T^*$, then $f(e) = c_e$ (thus $e \notin E_f$).

4. Similarly, if $e' = (u, v) \in E$ with $u \in T^*$ and $v \in S^*$, then $f(e') = 0$. *Why?*

On the cut (S^*, T^*)

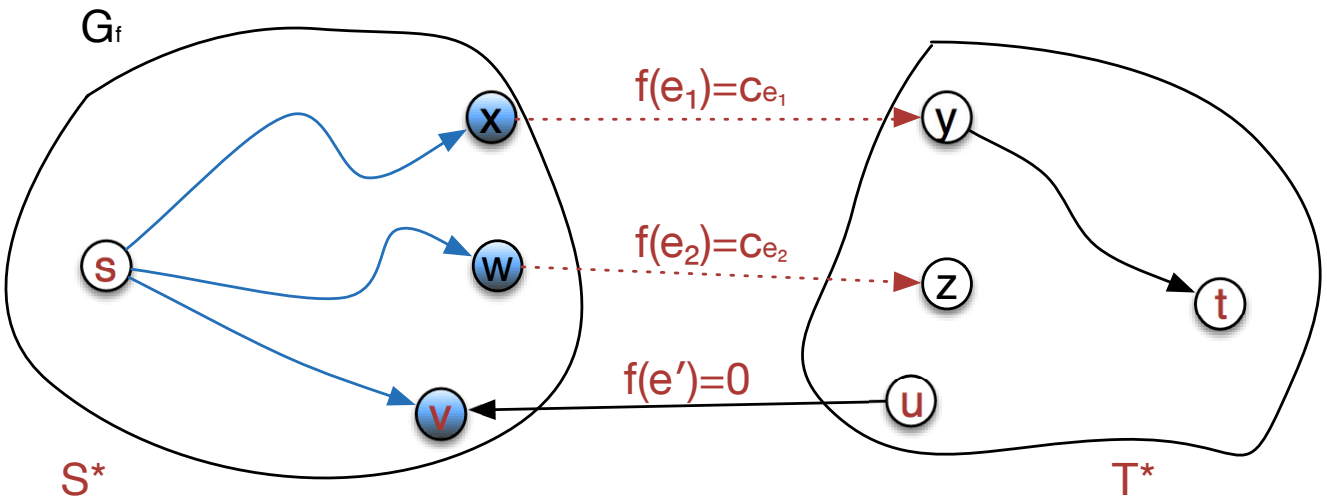
1. (S^*, T^*) is an s-t cut: that is, $s \in S^*$, $t \in T^*$. *Why?*
   Because there is no s-t path in G_f.

2. In G_f, no edge crosses from S^* to T^*. *Why?*
   If (u, v) crosses from S^* to T^*, thus $u \in S^*, v \in T^*$, then $\exists$ s-v path in G_f. Hence $v \in S^*$; contradiction.

3. Hence, if $e = (x, y) \in E$ with $x \in S^*$ and $y \in T^*$, then $f(e) = c_e$ (thus $e \notin E_f$).

4. Similarly, if $e' = (u, v) \in E$ with $u \in T^*$ and $v \in S^*$, then $f(e') = 0$. *Why?*
   If $f(e') > 0$, then $(v, u) \in E_f$, with $c_f(v, u) = f(e') > 0$. Contradicts our second observation.

Our observations on (S^*, T^*) in a diagram



In G, every edge e crossing from S* to T* satisfies $f(e)=c_e$ (of course, such e does not appear in G_f).
Every edge e′ in G crossing from T* to S* satisfies $f(e')=0$.

Net flow across a cut

Definition 9.

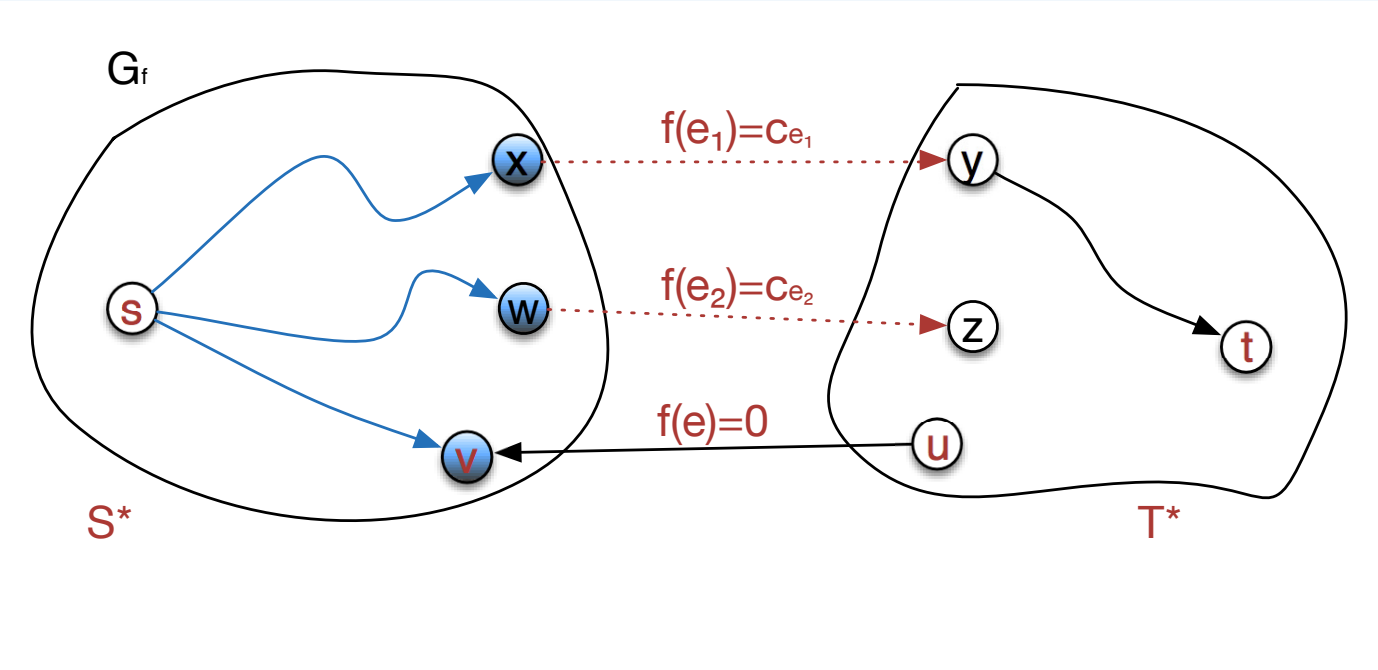
The net flow across an s-t cut (S, T) is the amount of flow leaving the cut minus the amount of flow entering the cut

$$f^{\text{out}}(S) - f^{\text{in}}(S), \tag{5}$$

where

1. $f^{\text{out}}(S) = \sum_{e \text{ out of } S} f(e)$
2. $f^{\text{in}}(S) = \sum_{e \text{ into } S} f(e)$

Net flow across (S^*, T^*) equals capacity of (S^*, T^*)



$$
\begin{aligned}
f^{\text{out}}(S^*) - f^{\text{in}}(S^*) &= \sum_{e \text{ out of } S^*} f(e) - \sum_{e \text{ into } S^*} f(e) \\
&= \sum_{e \text{ out of } S^*} c_e - 0 \\
&= c(S^*, T^*) \qquad (6)
\end{aligned}
$$

Roadmap revisited

Let f be the flow upon termination of the Ford-Fulkerson algorithm.

1. Exhibit a specific s-t cut (S^*, T^*) in G such that the

$$|f| = c(S^*, T^*).$$

   Not quite there yet!

   - *We exhibited (S^*, T^*) with net flow equal to its capacity.*
   - *We need to relate the net flow across (S^*, T^*) to $|f|$ (that is, the flow out of s).*
   - *In particular,* **if we showed them equal**, *then we'd have $|f| = c(S^*, T^*)$.*

2. Show that $|f|$ cannot exceed the capacity of **any** cut in G.

3. Conclude that f is a maximum flow.

net flow across any s-t cut $= |f|$

Recall that

- $f^{\text{out}}(S) = \sum_{e \text{ out of } S} f(e)$
- $f^{\text{in}}(S) = \sum_{e \text{ into } S} f(e)$
- net flow across $(S,T) \triangleq f^{\text{out}}(S) - f^{\text{in}}(S)$

Lemma 10.

Let f be any s-t flow, and (S,T) any s-t cut. Then

$$|f| = f^{out}(S) - f^{in}(S).$$

net flow → the net amount of flow that makes it across the cut.

Proof of Lemma 10

First, rewrite the flow out of s in terms of the flow on the vertices on S:

$$|f| \quad = \quad f^{\text{out}}(s) = \sum_{v \in S} \left(f^{\text{out}}(v) - f^{\text{in}}(v) \right) \tag{7}$$

since

- $f^{\text{in}}(s) = 0$;
- for every $v \in S - \{s\}$, the terms in the right-hand side of (7) cancel out because of flow conservation constraints.

Next, rewrite the right-hand side of equation 7 in terms of the *edges* that participate in these sums.
There are three types of edges.

Proof of Lemma 10 (cont'd)

1. Edges with both endpoints in S: such edges appear once in the first sum in equation 7 and once in the second, hence their flows cancel out.
2. Edges with the tail in S and head in T (out of S): such edges contribute to the first sum, $\sum_{v \in S} f^{\text{out}}(v)$, in equation 7 so they appear with a $+$.
3. Edges with the tail in T and head in S (into S): such edges contribute to the second sum, $\sum_{v \in S} f^{\text{in}}(v)$, in equation 7 so they appear with a $-$.

In effect, the right-hand side of equation 7 becomes

$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ into } S} f(e).$$

The lemma follows.

The value of a flow cannot exceed capacity of any cut

Corollary 11.

Let f be any s-t flow and (S, T) any s-t cut. Then

$$|f| \leq c(S, T).$$

Proof.

$$|f| = f^{\text{out}}(S) - f^{\text{in}}(S) \leq f^{\text{out}}(S) \leq c(S, T).$$

Putting everything together

- By Corollary 11, the value of a flow cannot exceed the capacity of any cut; in particular,

$$|f| \leq c(S^*, T^*).$$

- By Lemma 10, $|f|$ equals the net flow across any s-t cut; in particular,

$$|f| = f^{\text{out}}(S^*) - f^{\text{in}}(S^*).$$

- From (6), the net flow across (S^*, T^*) equals $c(S^*, T^*)$. Hence the above becomes

$$|f| = f^{\text{out}}(S^*) - f^{\text{in}}(S^*) = c(S^*, T^*).$$

$\Rightarrow$ Thus the flow computed by Ford-Fulkerson is a maximum flow because it cannot be increased anymore.

The max-flow min-cut theorem

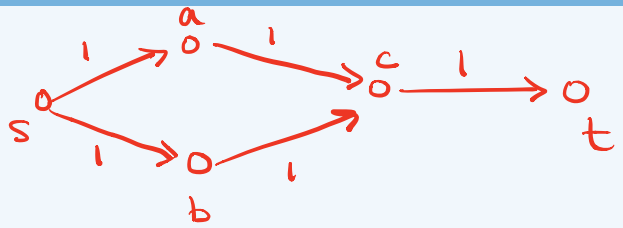
Theorem 12.

If f is an s-t flow such that there is no s-t path in G_f, then there is an s-t cut (S^*, T^*) in G such that $|f| = c(S^*, T^*)$. Therefore, f is a max flow and (S^*, T^*) is a cut of min capacity.

Theorem 13 (Max-flow Min-cut).

In every flow network, the maximum value of an s-t flow equals the minimum capacity of an s-t cut.

Integrality theorem



Recall the following claim.

Claim 4.

During execution of the Ford-Fulkerson algorithm, the flow values $\{f(e)\}$ and the residual capacities in G_f are all integers.

Combine with Theorem 12 to conclude:

Theorem 14 (Integrality theorem).

If all capacities in a flow network are integers, then there is a maximum flow for which every flow value $f(e)$ is an integer.

FF gives an integral max flow vector.

We can obtain the min cut efficiently. Upon termination of the FF algorithm, run BFS or DFS in the final residual graph G_f from the source node.

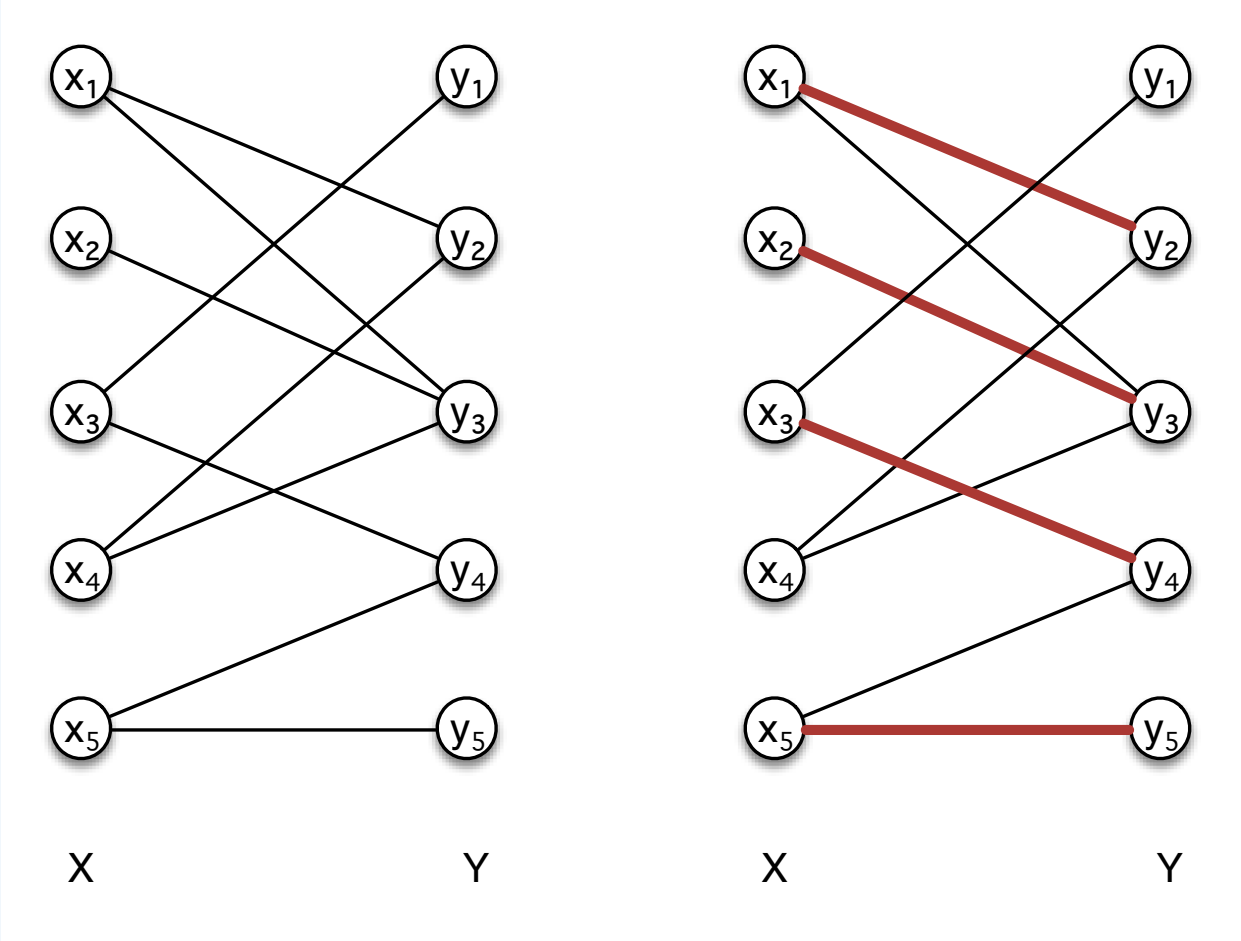
All nodes reachable from s is the cut for S^*; all other nodes are in T^*. Indeed,

S^* = set of nodes reachable from s in G_f upon termination of FT.

Today

1. Flow networks
   - Applications
2. The residual graph and augmenting paths
3. The Ford-Fulkerson algorithm for max flow
4. Correctness of the Ford-Fulkerson algorithm
5. Application: max bipartite matching

Bipartite Matching



Matchings

Definition 15.

A matching M is a subset of edges where every vertex in $X \cup Y$ appears at most once.

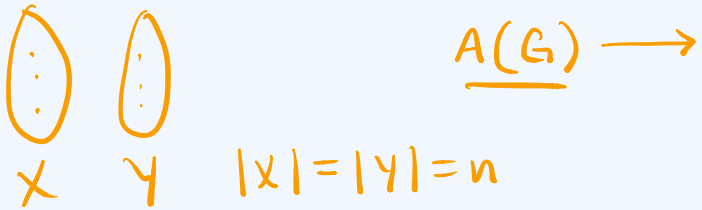
Example: $\{(x_1, y_2), (x_2, y_3), (x_3, y_4), (x_5, y_5)\}$ is a matching.

Perfect matching: every vertex in $X \cup Y$ appears exactly once.

- Not always possible: e.g., $|X| \neq |Y|$.

Maximum matching still desirable in applications.

- If we had an algorithm A to find maximum matching then we could also find a perfect matching, if one exists (*why?*).



Finding maximum matchings in bipartite graphs

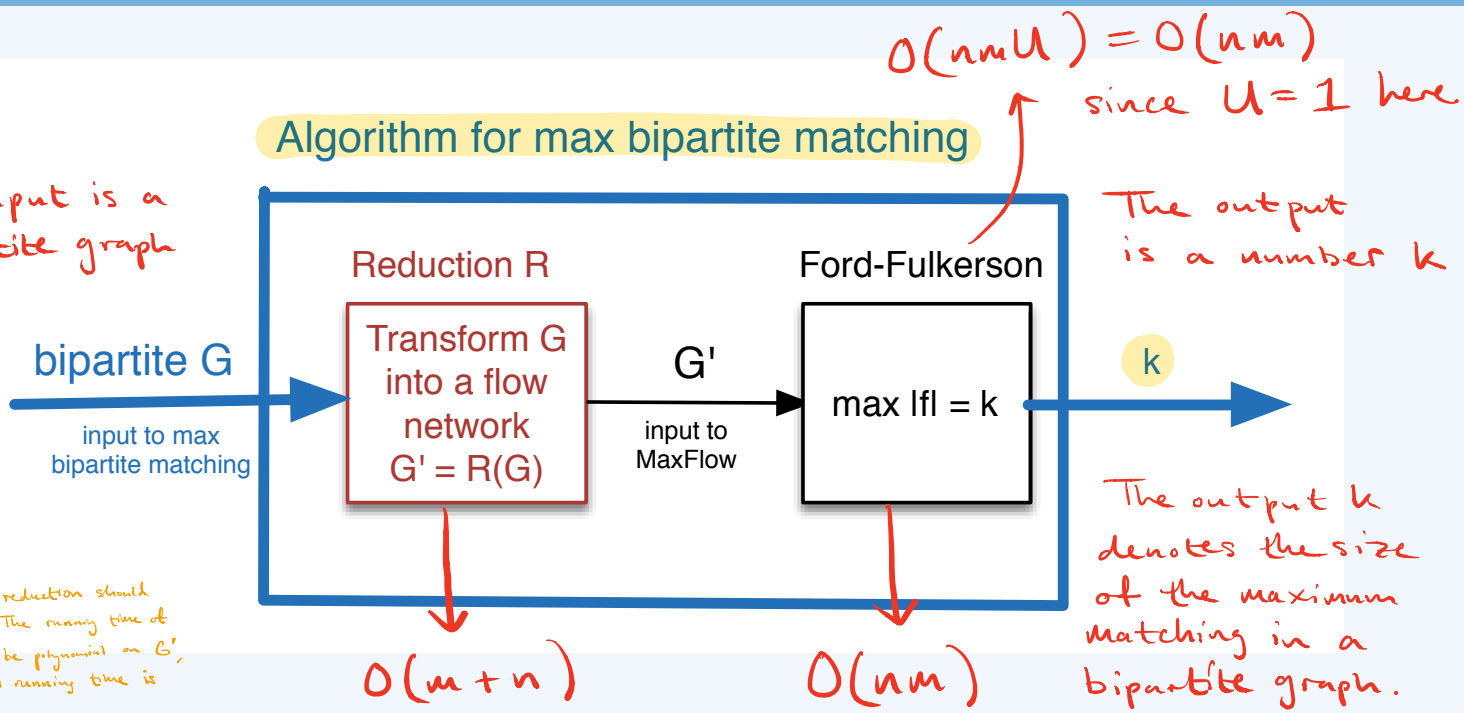
Note: The FF algorithm requires a flow network as input, so we must transform the bipartite graph into a flow network.

Idea: Use the Ford-Fulkerson algorithm to find maximum (or perfect) matchings in bipartite graphs.

Strategy: reformulate the problem as a max flow problem which we know how to solve (reduction).

To this end, we need to transform our input bipartite graph into a flow network.

A diagram of the algorithm for max bipartite matching



Remark 2.

1. *The reduction R must be efficient (polynomial in the size of G).*
2. *G and G' should be equivalent, in the sense that G has a max matching of size k if and only if the max flow in G' has value k.*

FF must also run in polynomial time (depends on capacity sizes)

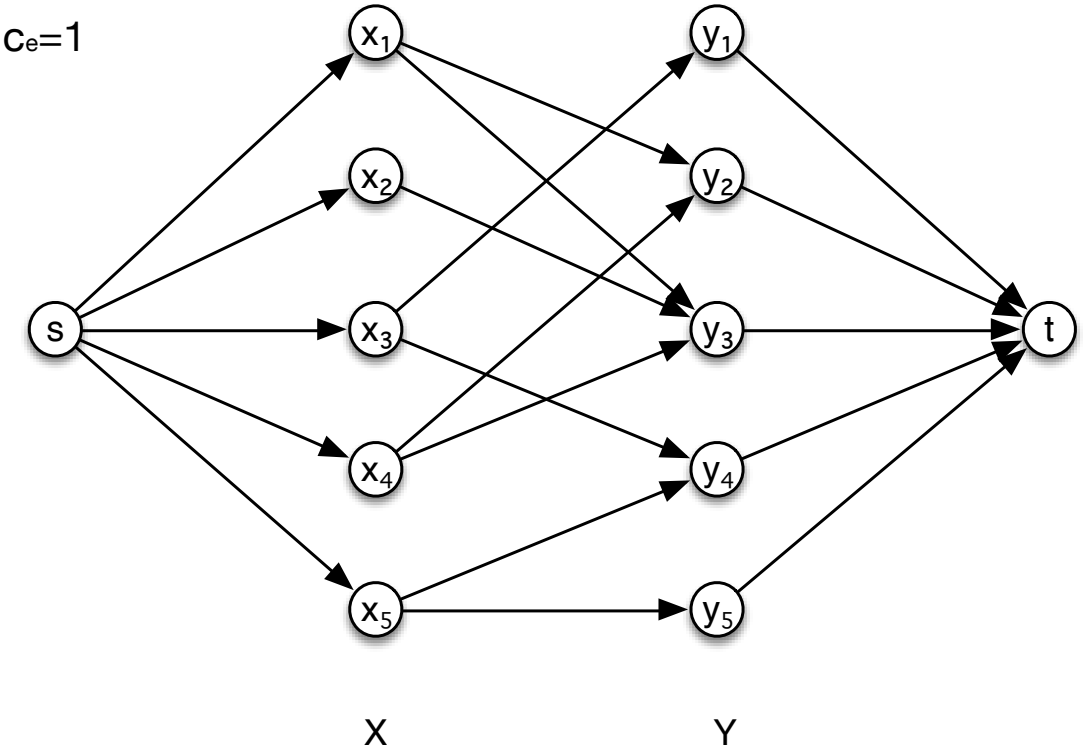
Deriving a flow network given a bipartite graph

Given a bipartite graph $G = (X \cup Y, E)$, we construct a flow network G' as follows.

- Add a source s.
- Add a sink t.
- Add (s, x) edges for all $x \in X$.
- Add (y, t) edges for all $y \in Y$.
- Direct all $e \in E$ from X to Y.
- Assign to every edge capacity of 1.

The flow network for the example bipartite graph

$|X| = |Y| = n$, $|E| = m$



The reduction transformation requires linear time since only need to add $2n$ edges, s and t, and the capacities. $\therefore O(n+m)$

Computing matchings in G from flows in G'

- $G = (X \cup Y, E)$ is the bipartite graph
- G' is the derived flow network

Claim 5.

The size of the maximum matching in G equals the value of the maximum flow in G'. The edges of the matching are the edges that carry flow from X to Y in G'.

Proof of Claim 5

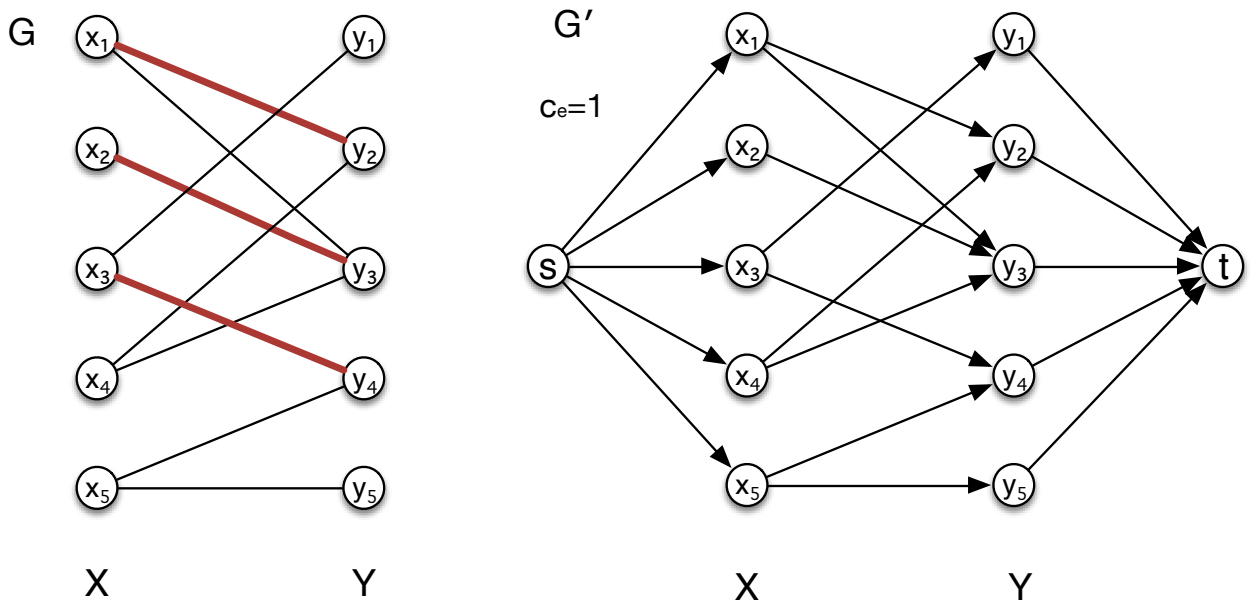
The claim follows if we show the following two statements (*why?*).

1. ($\Rightarrow$ Forward direction) For any matching M in G, we can construct a flow f in G' with value equal to the size of M, that is, $|f| = |M|$.

2. ($\Leftarrow$ Reverse direction) Given a max flow f' in G', we can construct a matching M' in G, with size equal to the value of the max flow, that is, $|M'| = |f'|$.

(1. $\Rightarrow$) From a matching M to a flow f with $|f| = |M|$

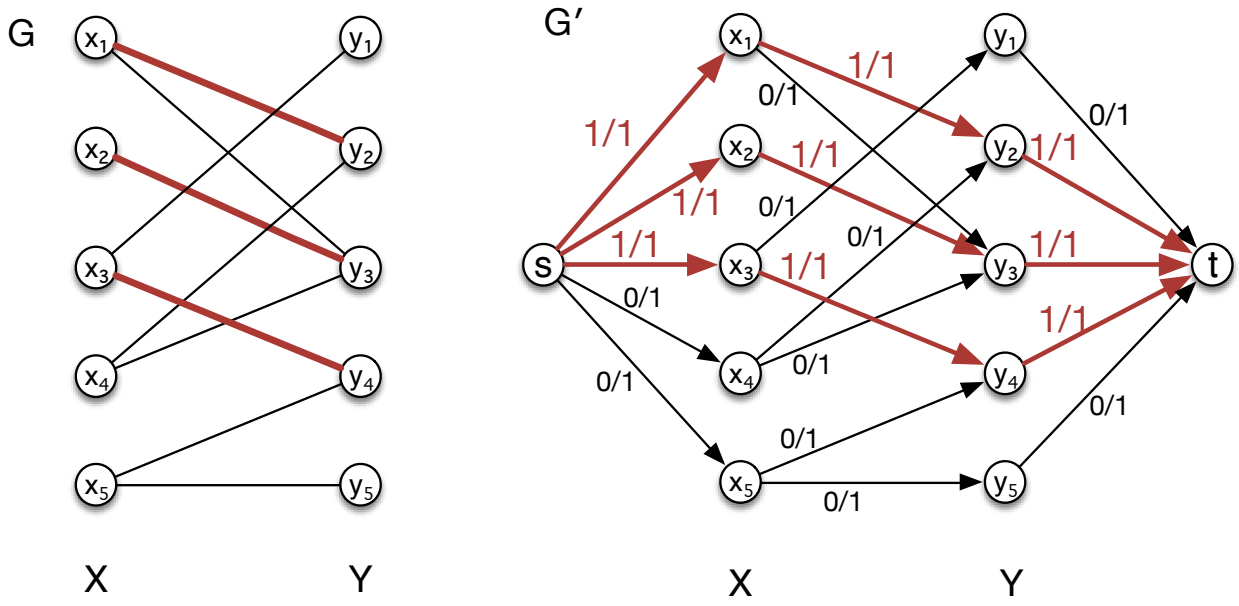
Let $|M| = k$.



Given matching M (the red edges in G), construct an integral flow f in G′, such that the value of f equals the number of edges in M.

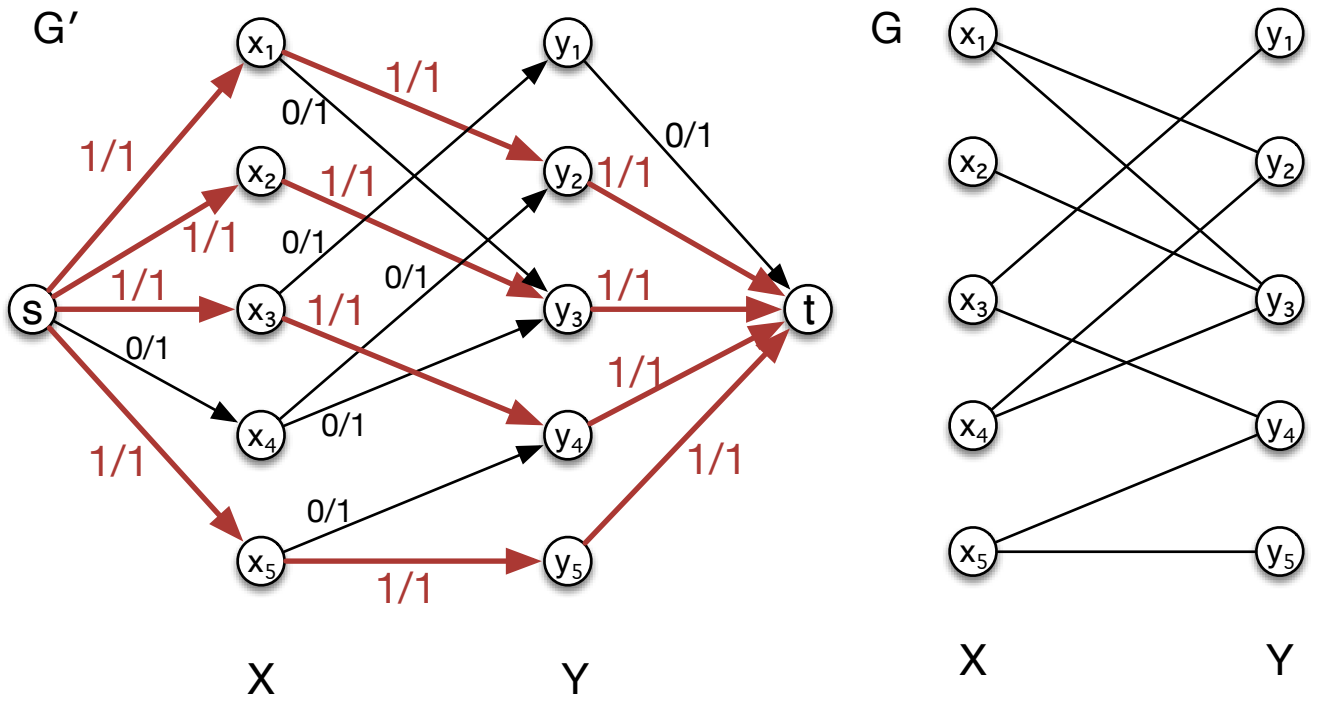
(1. $\Rightarrow$) From a matching M to a flow f with $|f| = |M|$

Let $|M| = k$. Send one unit of flow along each of the k edge-disjoint s-t paths that use the edges in M; then $|f| = k$.



Given matching M (the red edges in G), construct the integral flow f in G′. Then the value of f equals the number of edges in M.

(2. $\Leftarrow$) From max flow f' to M' with $|M'| = |f'|$



Given integral max flow f′ in G′, construct a matching M′ in G so that the number of edges in M′ equals the value of f′.

We can safely assume the max flow is integral since running Ford-Fulkerson returns an integral max flow vector.

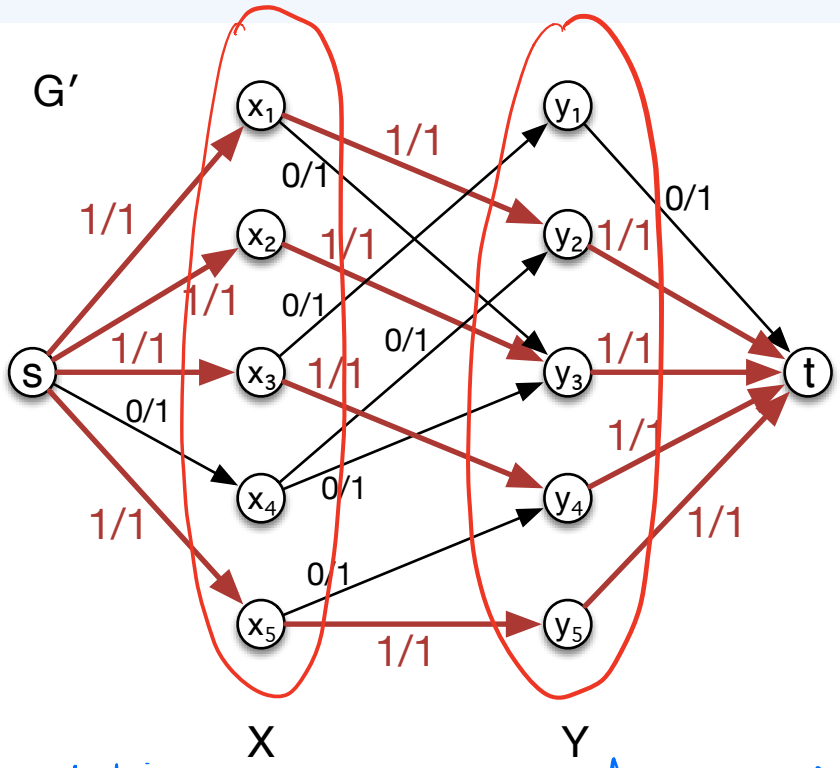
From max flow f' to matching M' with $|M'| = |f'|$

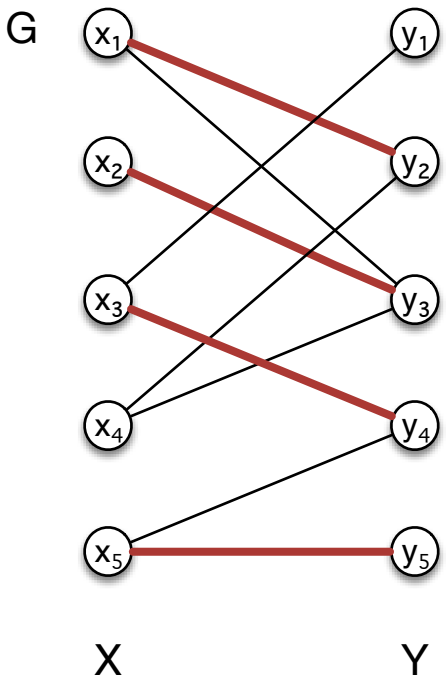
Given a max flow f' in G' with $|f'| = k$, we want to select a set of edges M' in G so that M' is a matching of size k.

- By the integrality theorem, there is an integer-valued flow f of value k.
- Then for every edge e, $f(e) = 0$ or $f(e) = 1$ (*why?*).
- Define the following matching M':

$$M' = \Big\{ e = (x, y) : x \in X, y \in Y \text{ and } f(e) = 1 \Big\}.$$

Obtaining a matching M' from an integral flow f

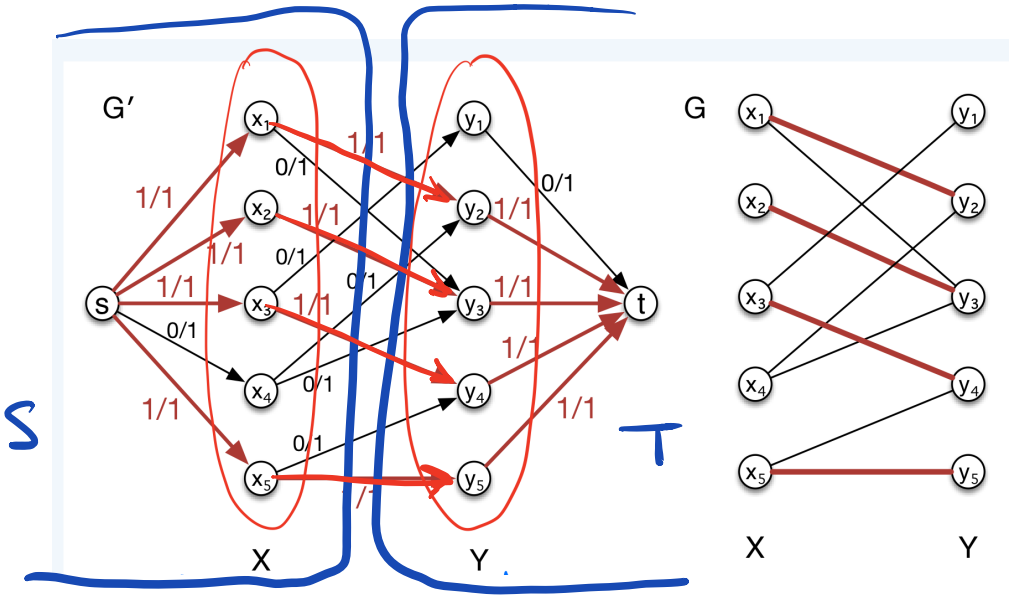




Each matching will only consist of edges from X to Y w/ flow value = 1

Given integral flow f in G′, construct matching M′ (the red edges in G), so that the number of edges in M′ equals the value of f.

Select $M' = \{(x,y) \in G' \text{ & } G \text{ s.t. } x \in X \text{ and } y \in Y, \text{ and } f(x,y) = 1\}$



Very useful for equiv. proofs.

$$|f| = \text{net flow}(S,T) = f^{out}(S) - f^{in}(S)$$

$$|f| = f^{out}(S) - \cancel{f^{in}(S)}^{\,0}$$

← no nodes going into S.

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

$$= |M'|$$

This sum only counts the edges that have been included in M' since only those edges have flow value = 1.

$$\Rightarrow |f| = |M'|$$

M' is a matching of size k

We need to show the following two facts.

1. **Fact 1**: M' is a matching.
2. **Fact 2**: M' has size k.

Proof of Fact 1.

Must show that every node in G' appears at most once in M'.

- Each node in X is the tail of at most one edge in M' (*flow conservation constraints*).
- Each node in Y is the head of at most one edge in M' (*flow conservation constraints*).

M' has size k

Proof of Fact 2.

- Consider the cut (S, T) where $S = \{s\} \cup X$, $T = Y \cup \{t\}$.
- We will compute its net flow.
  1. By definition, the net flow of (S, T) is

$$f^{\text{out}}(S) - f^{\text{in}}(S) = |M'|$$

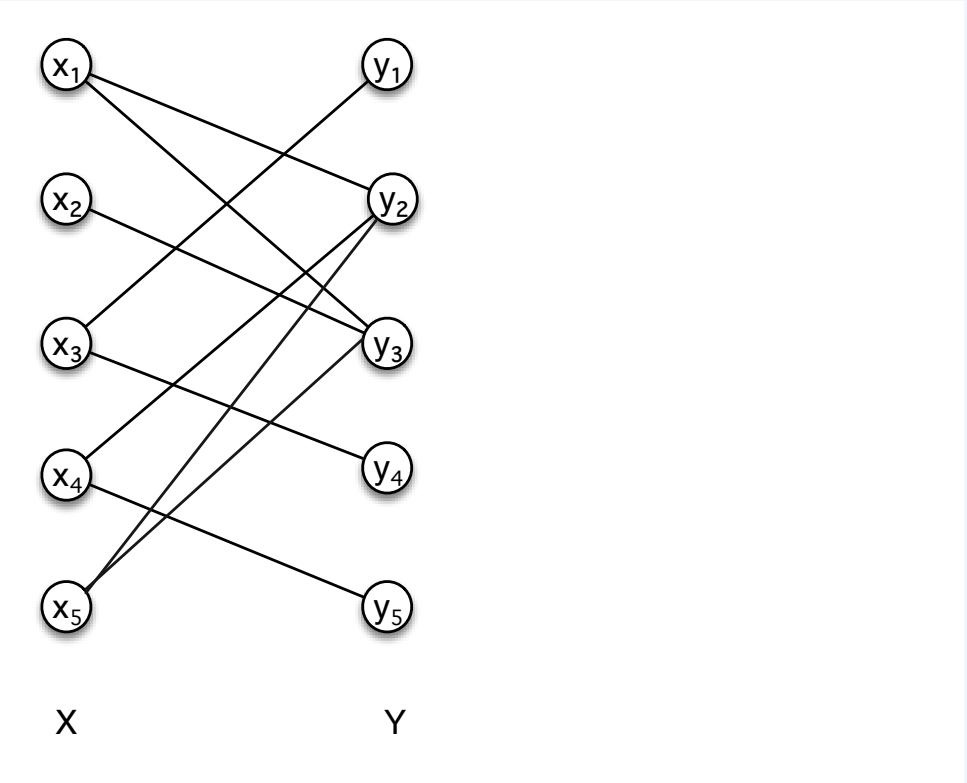
     since
     - the only edges that carry flow out of S are the edges in M';
     - the flow into S is 0 (no edges enter S).
  2. By Lemma 10, the net flow across (S, T) equals $|f|$; hence net flow across $(S, T) = k$.

$\Rightarrow$ Thus $|M'| = k$.

Time for finding max matching in bipartite graphs

1. Ford-Fulkerson: $O(mnU) = O(mn)$
2. Improved: $O(m\sqrt{n})$ [HopcroftKarp, Karzanov 1973]
3. Improved further for sparse ($m = O(n)$) graphs: $\tilde{O}(m^{10/7})$ [Madry2013]

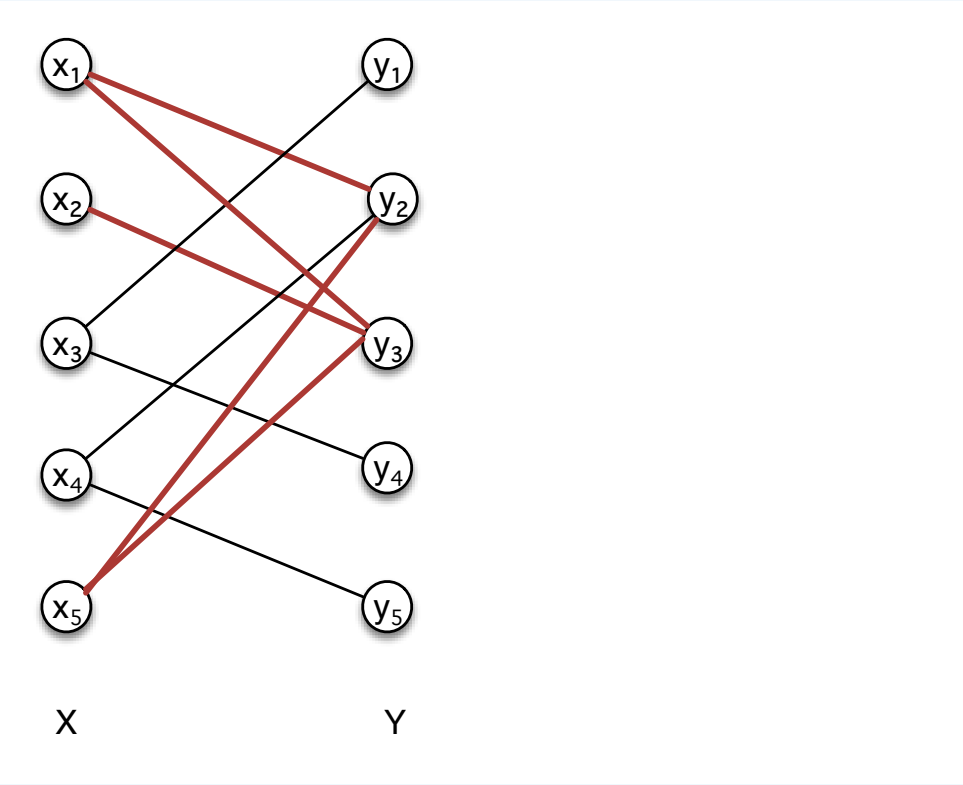
Perfect matchings in bipartite graphs with $|X| = |Y| = n$ (Hall's theorem)



Is there a matching of size n, that is, a perfect matching in G?

A necessary condition for a perfect matching to exist

For every subset A of nodes in X, there are at least as many neighbors of A in Y. In symbols, $\forall A \subseteq X, |N(A)| \geq |A|$.

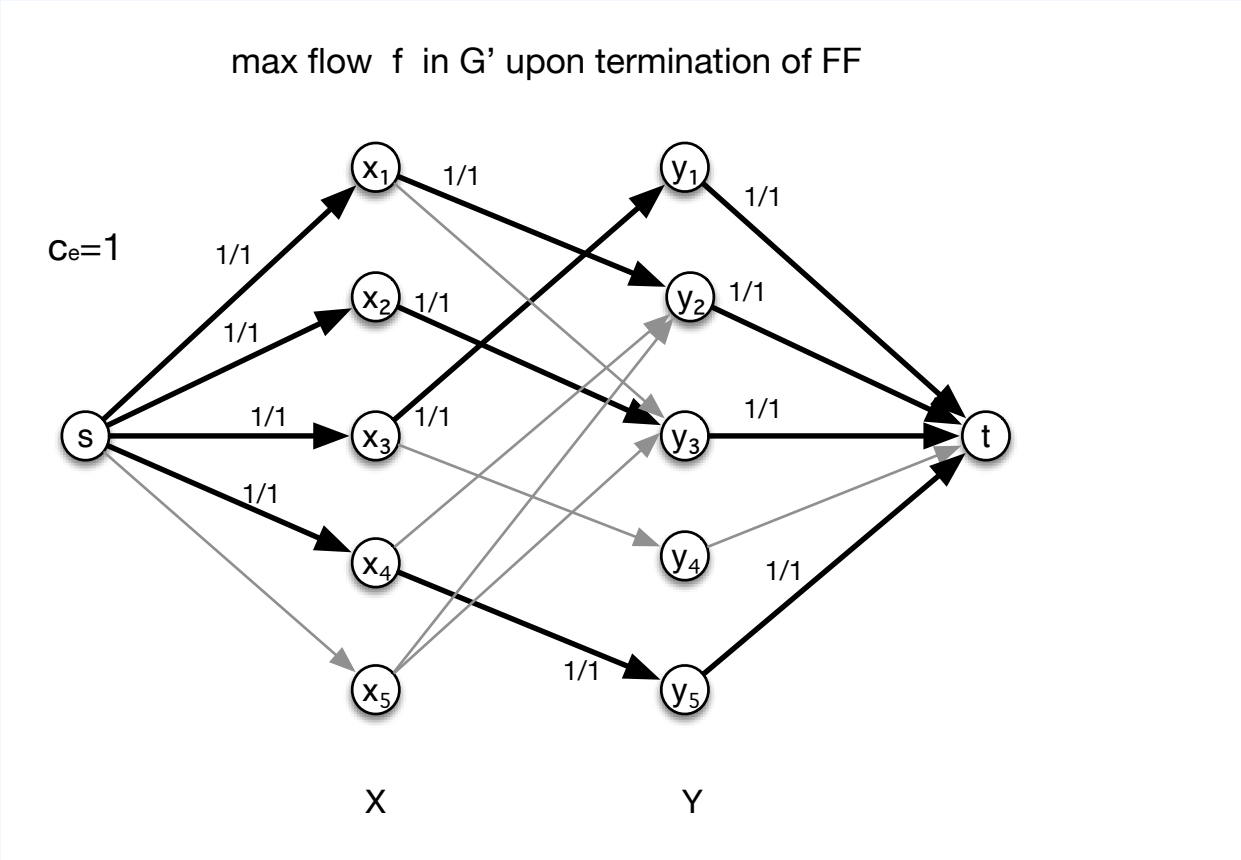


Is this a **sufficient** *condition as well?*

That is, if G does not have a perfect matching, is there always a subset $A \subseteq X$ such that $|N(A)| < |A|$?

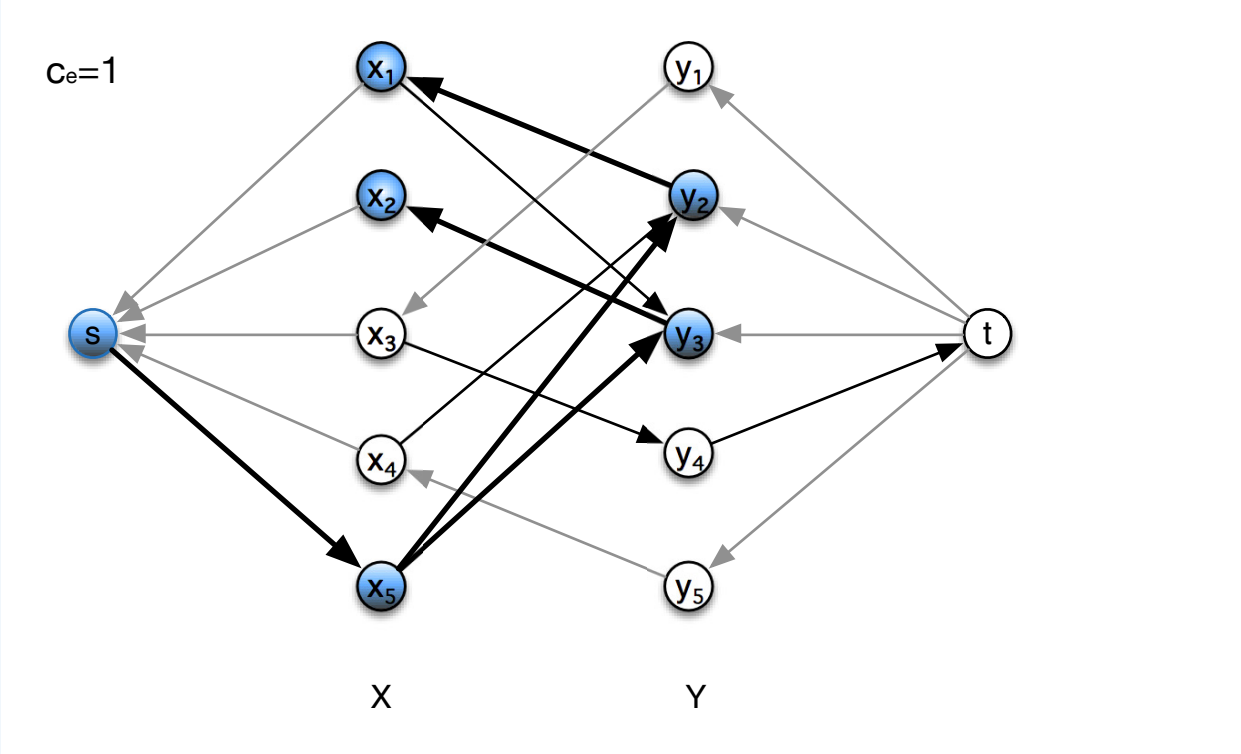
- Given bipartite G, transform it into a flow network G'.
- Run Ford-Fulkerson on G'.
- Assume $\max |f| < n$. We want to exhibit a set A as above.
- Since $\max |f| < n$, we know that $\min_{(S,T)} c(S,T) = \max |f| < n$.
- Consider the residual graph G_f upon termination of FF.
- Define the cut (S^*, T^*) as before, that is, S^* consists of all vertices reachable from s and T^* of everything else.
- We claim that the set $A = S^* \cap X$ satisfies $|A| > |N(A)|$.

A max flow in G'



The residual graph upon termination of FF

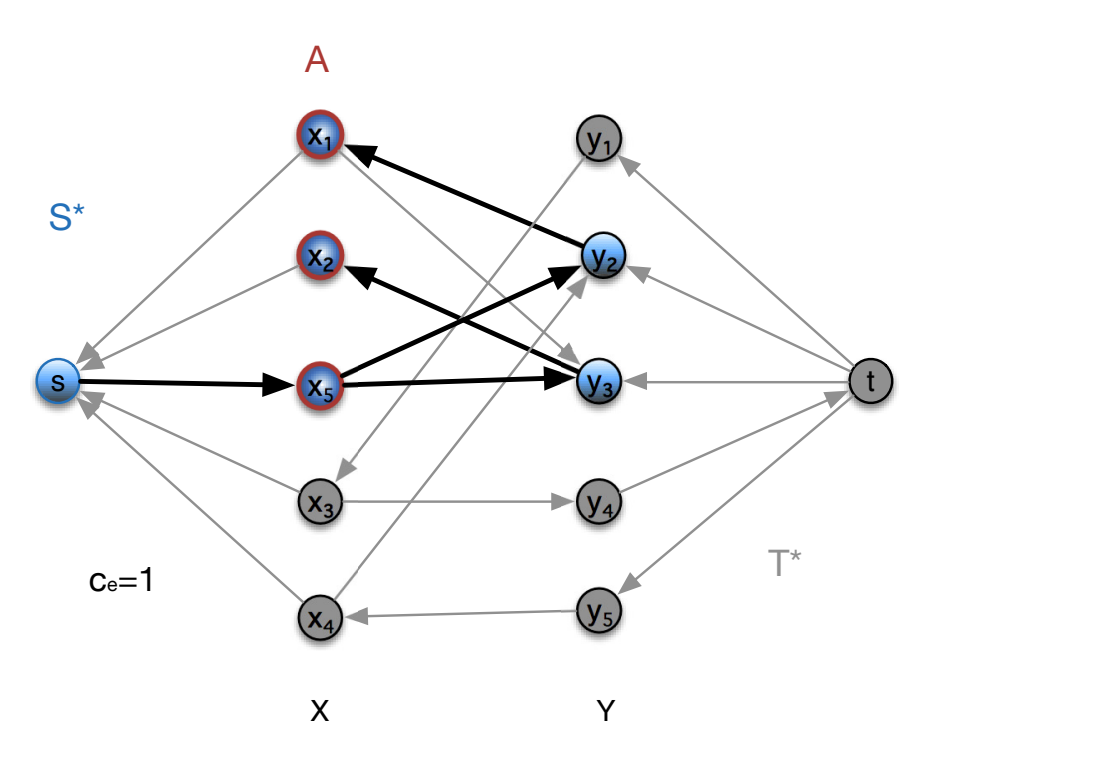
Define the cut (S^*, T^*) where S^* consists of all vertices reachable from s in G_f and T^* of everything else.



Here S^* consists of the blue vertices.

The set A that satisfies $|A| > |N(A)|$

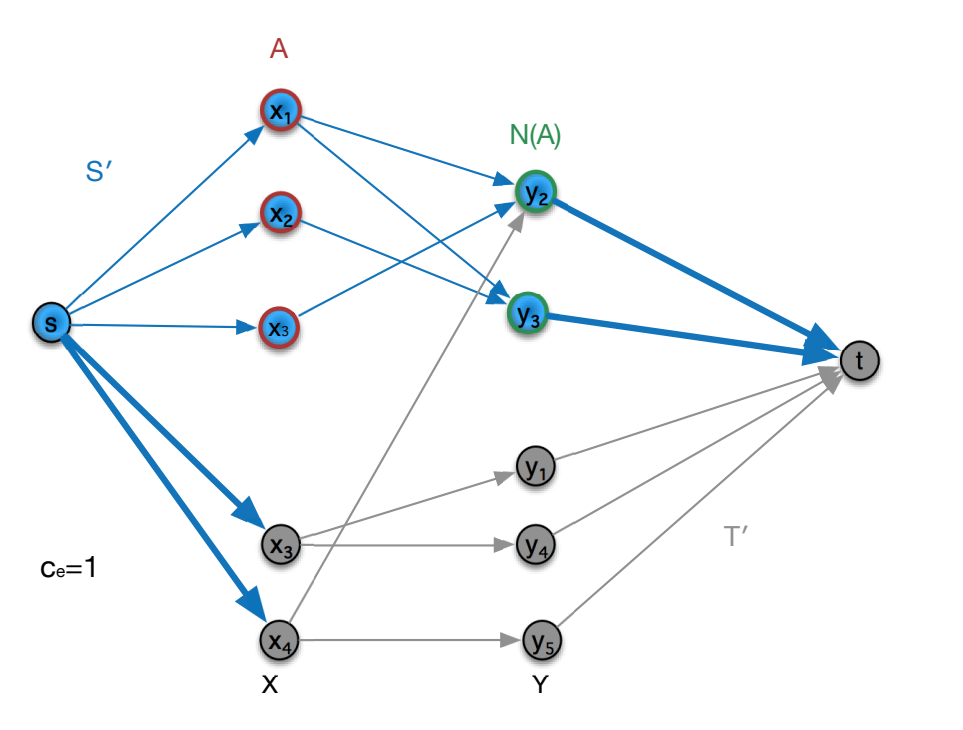
Clearly S^* consists of s, some vertices in X *(why?)* and possibly some vertices in Y. We set $A = S^* \cap X$.



The residual graph slightly rearranged to group the vertices in A together.

Augmenting S^* to include all neighbors of A

Note that S^* might contain some neighbors of A.
Augment S^* by moving *all* neighbors of A into S^*.
Let S' be the resulting set, $T' = V - S'$.



Blue nodes belong to S': among them, nodes with red contour are in A and nodes with green contour in $N(A)$. Silver nodes belong to T'.

How does $c(S', T')$ compare to $c(S^*, T^*)$?

For every node u we move from T^* to S'

- we add 1 to $c(S^*, T^*)$ because of the edge (u, t) that now crosses from S' to T';
- we subtract at least 1 from $c(S^*, T^*)$ because u is a neighbor of at least one node $v \in A$, hence the edge (v, u) no longer crosses from S^* to T^*.

Hence $c(S', T') \leq c(S^*, T^*)$.

Since (S^*, T^*) is a min cut, and $\max |f| < n$, we have

$$c(S', T') < n.$$

The capacity of (S', T')

What exactly is the capacity of (S', T')*?*

- $n - |A|$ edges cross from $s \in S'$ to T'
- $|N(A)|$ edges cross from S' to $t \in T'$

Hence $c(S', T') = n - |A| + |N(A)|$.

Since $c(S', T') < n$, we have

$$n - |A| + |N(A)| < n.$$

Hence

$$|A| > |N(A)|.$$