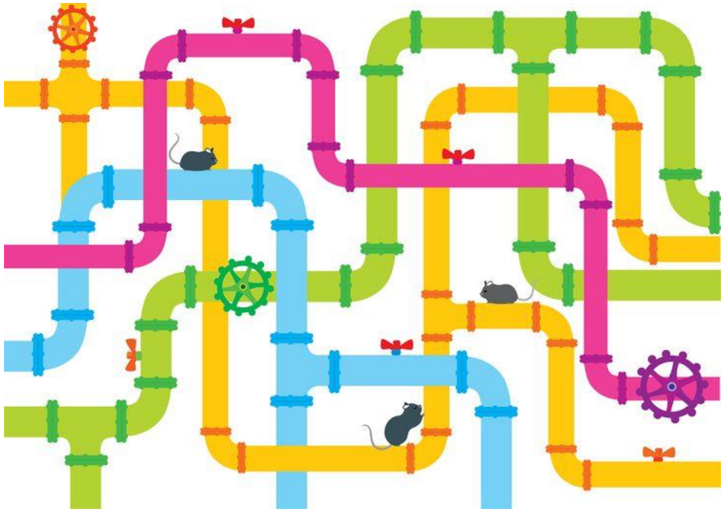
Maximum Flows in Capacitated Networks

Reading from Special Topics Textbook:

Chapter 4, Section 4.2, pp. 107-126.



Definitions

Digraph $D = (V,E)$

Source. A vertex s with all incident edges directed out of s.

Sink. A vertex t with all incident edges directed into t.

Capacity of an edge. Nonnegative real weight $c(e), e \in E$

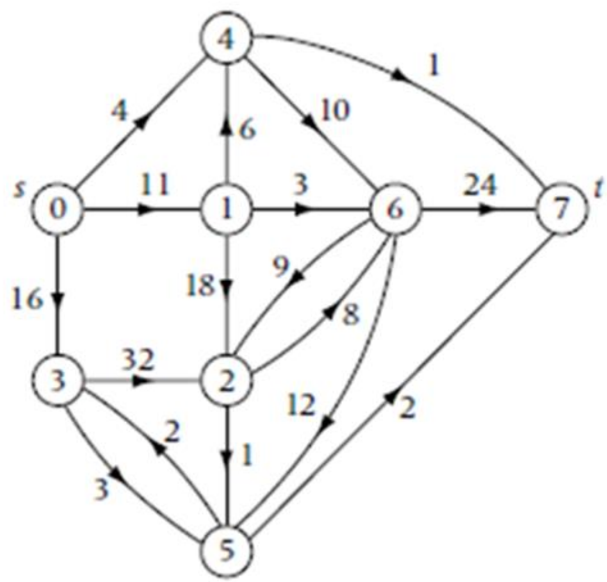
Flow. Weighting f so that the sum of the f-weights of the edges directed into any vertex v different from s or t equals that sum of the edges weights directed out v.

Value of the flow. Sum of the f-weights out of source s, which equals the sum of f-weights into sink t.

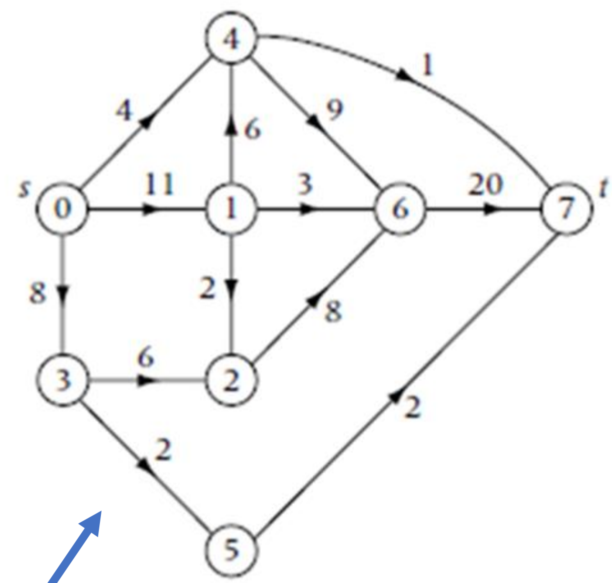
Constraint. The weight of the flow on each edge must not exceed the capacity of the edge, i.e.,

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

Example



A capacitated network N

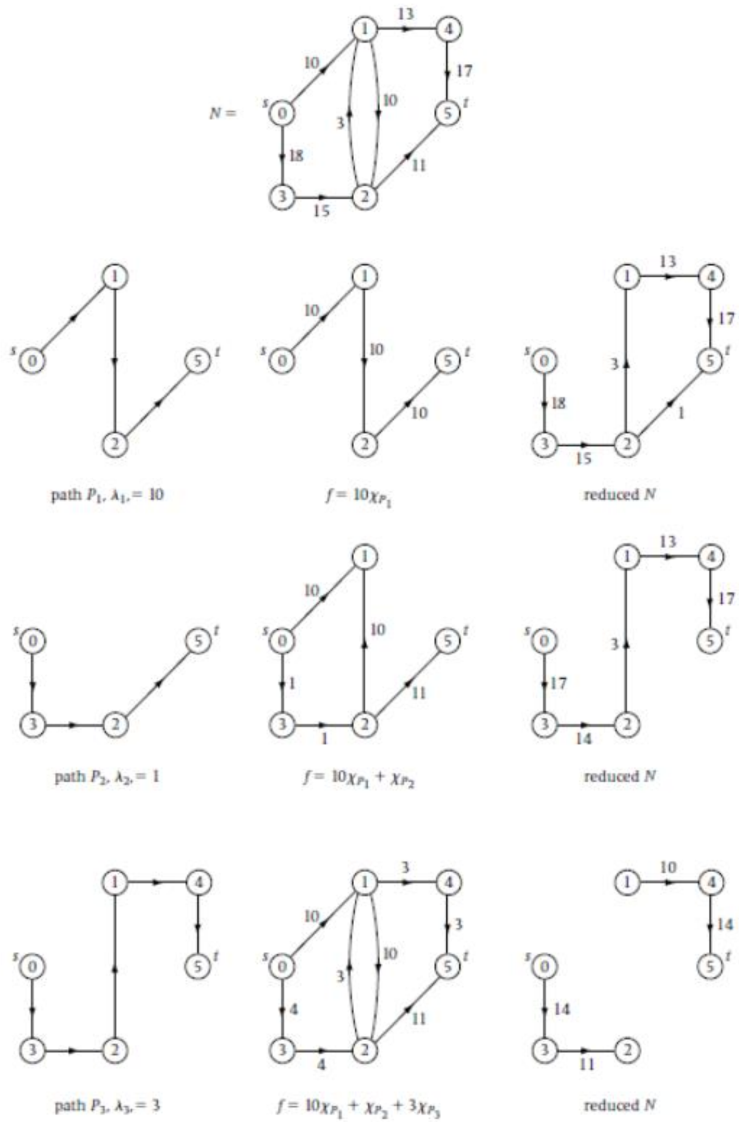


A flow f in N

Flow values equal to 0 are omitted

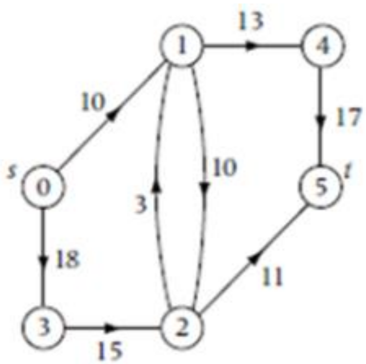
Naïve Algorithm keeps pumping more flow along paths from s to t without exceeding capacity and reducing capacity accordingly

χ_{P_i} is characteristic weighting for path P_i that is $\chi_{P_i}(e) = 1$ if e in P_i and 0, otherwise

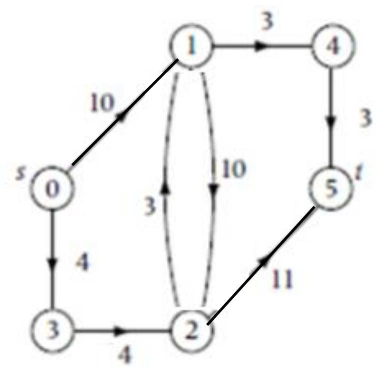


Can not increase flow more than 14.

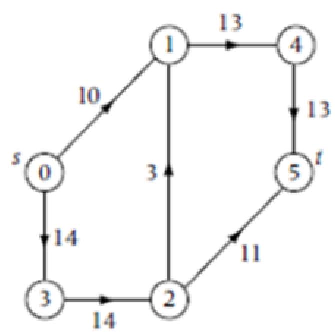
Flow we computed has value 14, which is not maximum. There exists a flow having value 24.



capacitated network



Flow of value 14 we computed



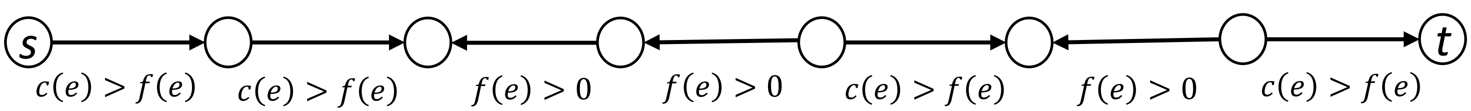
Maximum flow has value 24

How to continue increasing the flow?

Augmenting semipaths

Augmenting Semipath

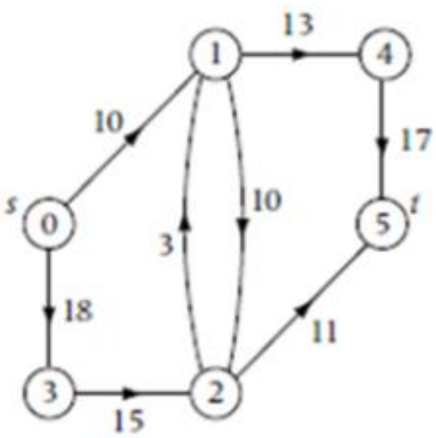
- **Semipath** – a path from s to t where we all edges in reverse direction, i.e., pointing towards s.
- **Augmenting semipath** – a semipath such that the **residual capacity**, i.e., $c(e) - f(e)$ on every forward edge e (directed towards t) and the flow $f(e)$ on every backwards edge (directed towards s) is strictly positive.



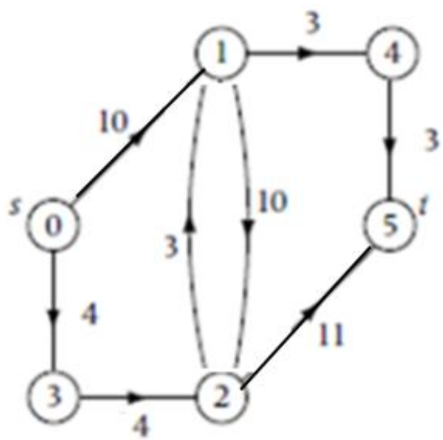
Action of augmenting semipath in increasing the flow

- Let c_f be the **minimum** of all such values on the augmenting semipath.
- Without exceeding the capacity, we can increase the value of the flow, by **increasing** the flow on the forward edges of the augmenting path by c_f and **decreasing** the flow on backwards edges by

PSN. Continue increasing the flow in our example using augmenting semipaths. We ended up with the following flow using directed paths.



capacitated network



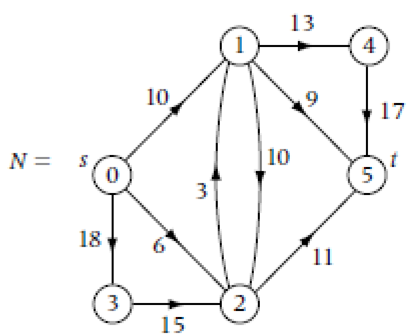
Flow of value 14 we computed

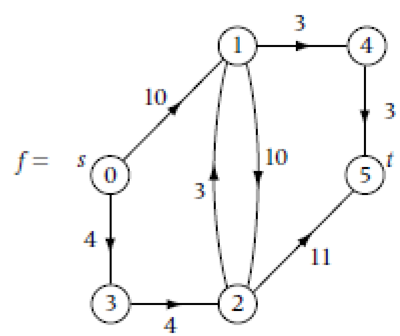
f-derived network N_f

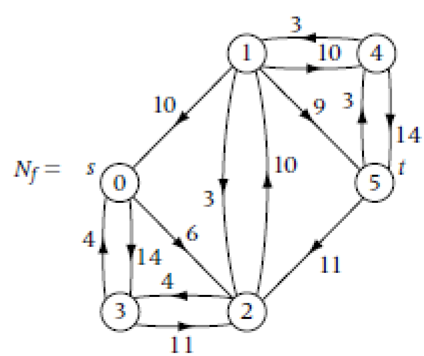
The vertices of N_f are the same as N. For each edge e of N_f:

(1) if $c(e) - f(e)$ is nonzero edge e is included in N_f and given the weight $c(e) - f(e)$;

(2) if the flow $f(e)$ is nonzero the orientation of edge e is reversed and included in N_f and given the weight $f(e)$.

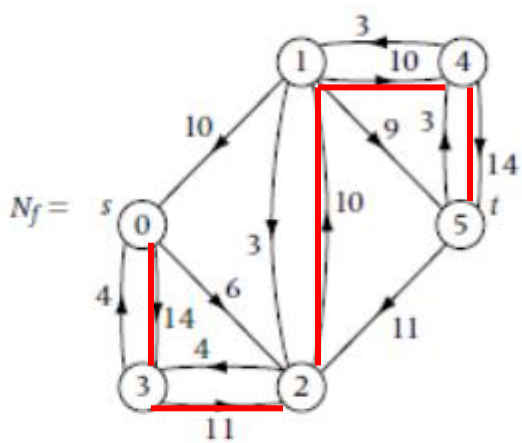






Computing an augmenting path

An algorithm like BFS can be used to find a directed path in the *f*-derived network N_f. In the original network *N* this path must be an augmenting semipath.



How do we know that we have a maximum flow when there are no more augmenting paths?

Cuts

Cut separating *s* and *t*

Bipartition of the vertex set *V* into two disjoint sets *X* and *Y* denoted by (*X*,*Y*).

Cut associated with the bipartition (*X*,*Y*), denoted *cut*(*X*,*Y*), is defined by

$$cut(X,Y) = \{(x,y)\ | x \in X, y \in Y\}.$$

$\Gamma = cut(X,Y)$ **separates *s* and *t*** if $s \in X$ and $t \in Y$.

Capacity of $\boldsymbol{\Gamma}$ denote by $cap(\Gamma)$ is the sum of the capacities on the edges of the cut, i.e.,

$$cap(\Gamma) = \sum_{e \in \Gamma} c(e).$$

Beautiful Idea

1. The value of any flow f from s to t is no greater then the capacity of any cut Γ separating s and t.

2. If $val(f^*) = cap(\Gamma^*)$ then
   f^* is a maximum flow and Γ^* is a minimum cut.

PSN. Prove 2. assuming 1.

f is a flow for which there are **no** f-augmenting paths

X is the set of vertices that are reachable from *s in* N_f

$Y = V \ - X$

$\Gamma = cut(X, Y)$

Proposition. $val(f) = cap(\Gamma)$.

Consider vertices $x \in X$ and $y \in Y$. Since x is reachable from s but y is not, it follows that $f(x) = c(xy)$ for $xy \in E$, i.e., $xy \in \Gamma$ and $f(x) = 0$, for $yx \in E$. Since there is no back flow

$$val(f) = \sum_{e \in \Gamma} f(e) = \sum_{e \in \Gamma} c(e) = cap(\Gamma).$$

Max-Flow Min-Cut

In the PSN exercise, we showed that if $val(f) = cap(\Gamma)$, then

f is a maximum flow

Γ is a minimum cut

Max-Flow Min-Cut Theorem (Ford-Fulkerson). The value of a maximum flow from s to t equals the capacity of a minimum cut separating s and t .

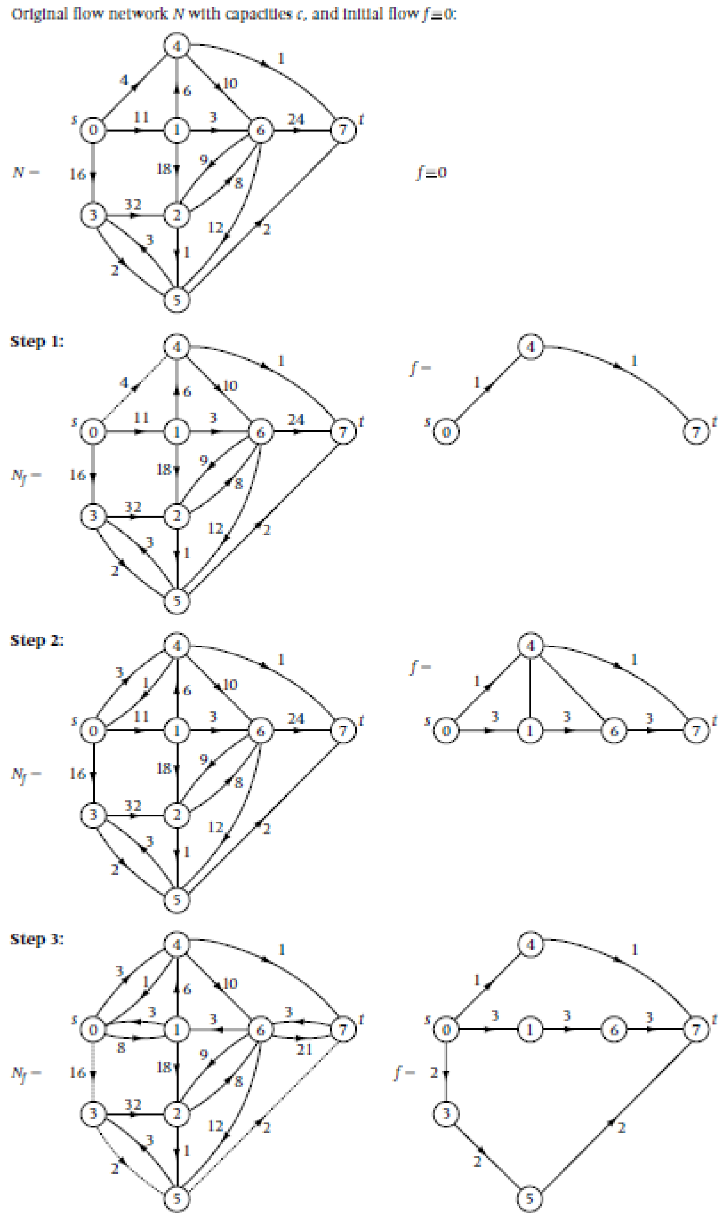
Ford-Fulkerson Algorithm

procedure *FordFulkerson*(N, f, Γ)
Input: $N = (D,s,t,c)$ (a capacitated network)
Output: f (maximum flow)
Γ(minimum cut)
$f \leftarrow 0$
$N_f \leftarrow N$
while there is a directed path from s to t in the f-derived network N_f **do**
$P \leftarrow$ a path from s to t in N_f
$S \leftarrow$ the f-augmenting semipath in N corresponding to path P in N_f
$c_f(S) \leftarrow \mu(P)$ //$\mu(P)$ is the minimum weight over all the edges of P
$f \leftarrow f + c_f(S)\chi_S$ //augment f
update N_f
endwhile
$X \leftarrow$ set of vertices that are accessible from s in N_f //$s \in X$
$Y \leftarrow$ set of vertices that are not accessible from s in N_f //$t \in Y$
$\Gamma \leftarrow cut(X,Y)$
end *FordFulkerson*

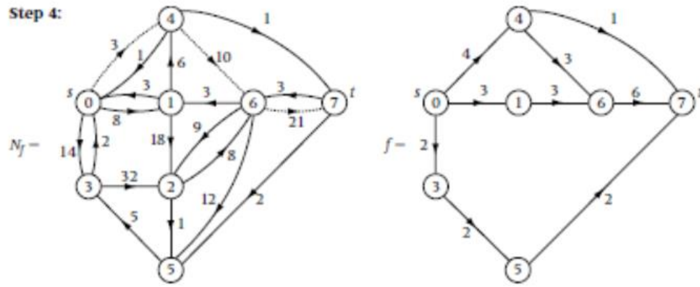
Edmonds-Karp algorithm

- Edmonds and Karp showed that a good choice for the augmenting semipath S at each stage of procedure *FordFulkerson* is one which is shortest over all such semipaths.
- At each stage, a shortest augmenting semipath S can be found by performing a breadth-first search of the f-derived network N_f to find a shortest path P from s to t.
- The Edmonds-Karp algorithm has worst-case complexity $W(n,m) \in O(nm^2)$.

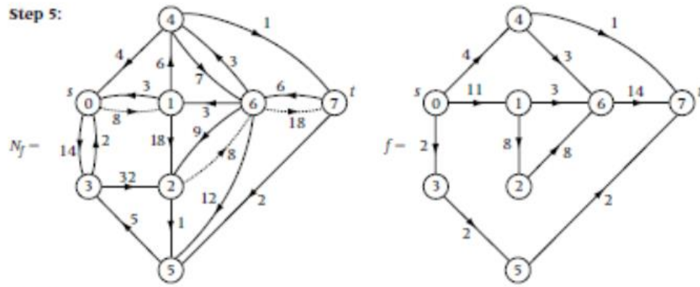
Action of Edmonds-Karp algorithm for sample network



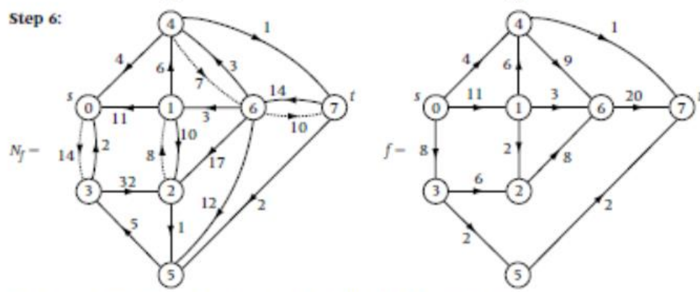
Step 4:



Step 5:

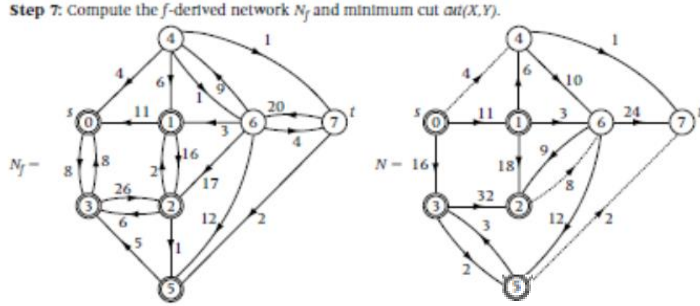


Step 6:



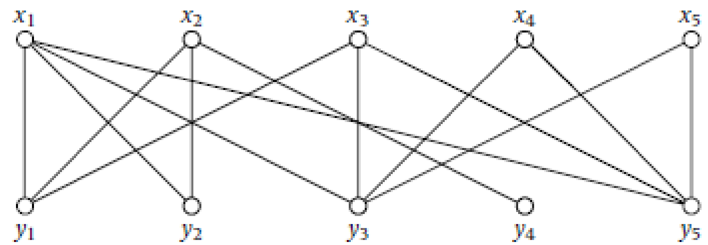
There are no more augmenting semipaths. The final flow f has value 23.

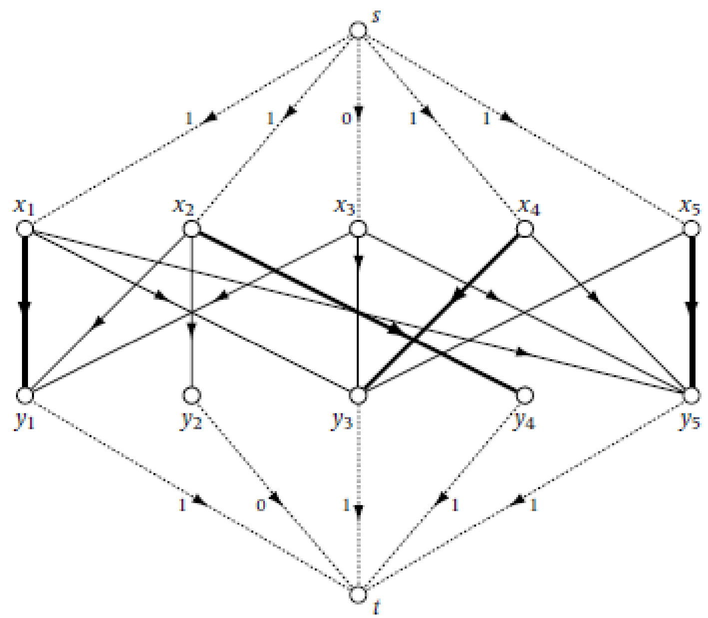
Step 7: Compute the f-derived network N_f and minimum cut $cut(X,Y)$.



The set $X = \{1,2,3,4,6\}$ of vertices that are accessible in N_f from the source s (marked with ◎) and the set $Y = \{5,7,8\}$ of vertices that are not accessible from s determine a cut $\Gamma = cut(X,Y)$ of capacity $c(X,Y) = 4 + 6 + 3 + 8 + 2 = 23$. Hence, we have $val(f) = 23 = cap(\Gamma)$, so that f is a maximum flow Γ and is a minimum cut.

Flows can be used to obtain a maximum cardinality matching in a bipartite graph





Why don’t A.I. engineers need a resume?

They just let their projects speak for themselves.