CSE 421: Introduction to Algorithms

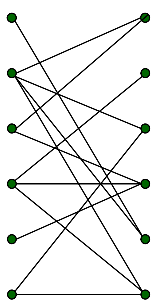
Network Flow

Winter 2003
Paul Beame

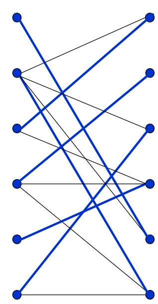
Bipartite Matching

- Given: A bipartite graph G=(V,E)
  - **M Í E** is a matching in **G** iff no two edges in **M** share a vertex
- Goal: Find a matching **M** in **G** of maximum possible size

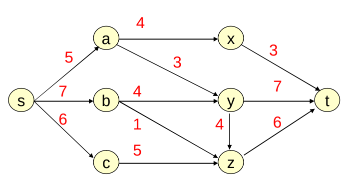
Bipartite Matching



Bipartite Matching

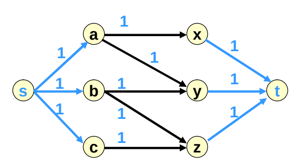


The Network Flow Problem



- How much stuff can flow from s to t?

Bipartite matching as a special case of flow



Net Flow: Formal Definition

Given:

A digraph **G = (V,E)**

Two vertices **s,t** in **V** (source & sink)

A *capacity* **c(u,v) ≥ 0** for each **(u,v) ∈ E** (and **c(u,v) = 0** for all non-edges **(u,v)**)

Find:

A *flow function* **f: E → R** s.t., for all **u,v**:

- **0 £ f(u,v) ≤ c(u,v)** [Capacity Constraint]
- if **u ≠ s,t**, i.e. $f^{out}(u)=f^{in}(u)$ [Flow Conservation]

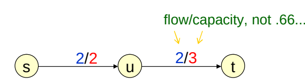
Maximizing total flow $v(f) = f^{out}(s)$

Notation:

$$f^{in}(v) = \sum_{e=(u,v)\in E} f(u,v) \qquad f^{out}(v) = \sum_{e=(v,w)\in E} f(v,w)$$

7

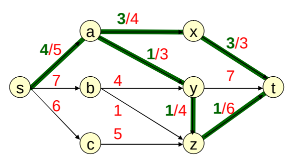
Example: A Flow Function



$f^{in}(u)=f(s,u)=2=f(u,t)=f^{out}(u)$

8

Example: A Flow Function



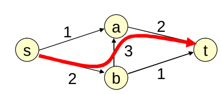
- Not shown: f(u,v) if = 0
- Note: max flow ≥ 4 since f is a flow function, with v(f) = 4

9

Max Flow via a Greedy Alg?

While there is an s → t path in G
  Pick such a path, **p**
  Find **c**, the min capacity of any edge in **p**
  Subtract **c** from all capacities on **p**
  Delete edges of capacity **0**

- This does NOT always find a max flow:



If pick s →b →a →t first, flow stuck at 2.
But flow 3 possible.

10

A Brief History of Flow

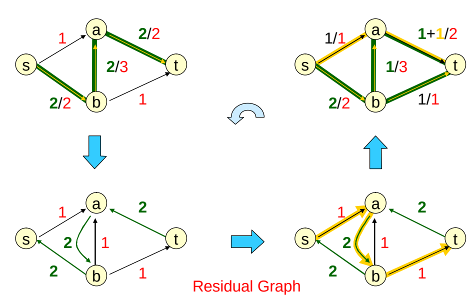
#	year	discoverer(s)	bounds bound exact
1	1951	Dantzig	$O(n^2mU)$
2	1955	Ford & Fulkerson	$O(nmU)$
3	1970	Dinitz Edmonds & Karp	$O(nm^2)$
4	1970	Dinitz	$O(n^2m)$
5	1972	Edmonds & Karp Dinitz	$O(m^2 \log U)$
6	1973	Dinitz Gabow	$O(nm \log U)$
7	1974	Karzanov	$O(n^3)$
8	1977	Cherkassky	$O(n^2\sqrt{m})$
9	1980	Galil & Naamad	$O(nm \log^2 n)$
10	1983	Sleator & Tarjan	$O(nm \log n)$
11	1986	Goldberg & Tarjan	$O(nm \log(n^2/m))$
12	1987	Ahuja & Orlin	$O(nm + n^2 \log U)$
13	1987	Ahuja et al.	$O(nm \log(n\sqrt{\log U}/(m+2))$
14	1989	Cheriyan & Hagerup	$E(nm + n^2 \log^2 n)$
15	1990	Cheriyan et al.	$O(n^3/\log n)$
16	1990	Alon	$O(nm + n^{8/3} \log n)$
17	1992	King et al.	$O(nm + n^{2+\epsilon})$
18	1993	Phillips & Westbrook	$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$
19	1994	King et al.	$O(nm \log_{m/(n \log n)} n)$
20	1997	Goldberg & Rao	$O(m^{3/2} \log(n^2/m) \log U)$ $O(n^{2/3} m \log(n^2/m) \log U)$

n = # of vertices
m= # of edges
U = Max capacity

Source: Goldberg & Rao, FOCS '97

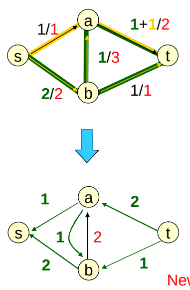
11

Greed Revisited: Residual Graph & Augmenting Path



12

Greed Revisited: An Augmenting Path

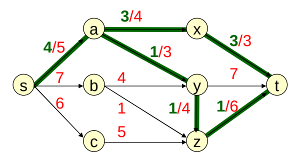


New Residual Graph

13

Residual Capacity

- The *residual capacity* (w.r.t. **f**) of **(u,v)** is $c_f(u,v) = c(u,v) - f(u,v)$ if $f(u,v) \leq c(u,v)$ and $c_f(u,v) = f(v,u)$ if $f(v,u) > 0$



- e.g. $c_f(s,b) = 7$; $c_f(a,x) = 1$; $c_f(x,a) = 3$

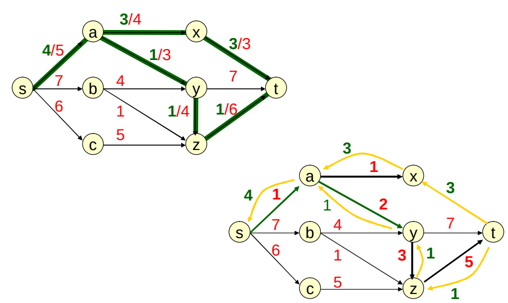
14

Residual Graph & Augmenting Paths

- The *residual graph* (w.r.t. **f**) is the graph $G_f = (V, E_f)$, where
  $E_f = \{ (u,v) \mid c_f(u,v) > 0 \}$
  - Two kinds of edges
    - Forward edges
      - $f(u,v) < c(u,v)$ so $c_f(u,v) = c(u,v) - f(u,v) > 0$
    - Backward edges
      - $f(u,v) > 0$ so $c_f(v,u) \geq -f(v,u) = f(u,v) > 0$
- An *augmenting path* (w.r.t. **f**) is a simple **s ® t** path in G_f.

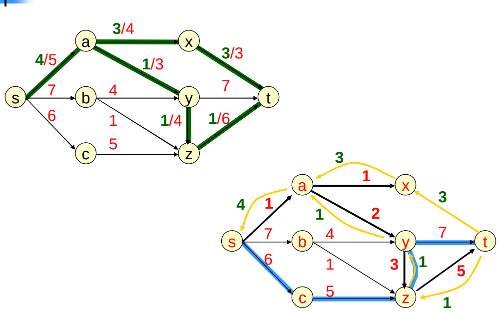
15

A Residual Network



16

An Augmenting Path



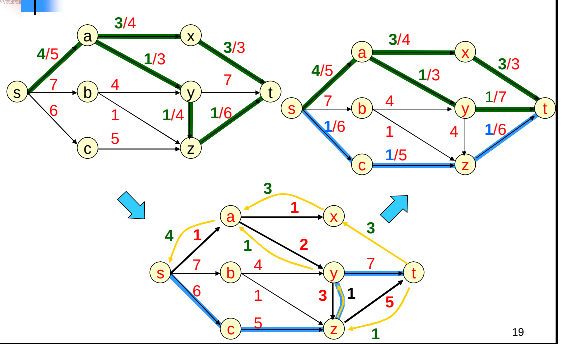
17

Augmenting A Flow

augment(**f**,**P**)
  $c_P \leftarrow \min_{(u,v)\ \text{î P}} c_f(u,v)$ "bottleneck(P)"
  for each **e** Î **P**
    if **e** is a forward edge then
      increase **f**(**e**) by c_P
    else (**e** is a backward edge)
      decrease **f**(**e**) by c_P
    endif
  endfor
  return(**f**)

18

Augmenting A Flow



Claim 6.1

If G_f has an augmenting path **P**, then the function **f'=**augment(**f**,**P**) is a legal flow.

Proof:

- **f'** and **f** differ only on the edges of **P** so only need to consider such edges (**u**,**v**)

Proof of Claim 6.1

- If (**u**,**v**) is a forward edge then
  f'(**u**,**v**)=**f**(**u**,**v**)+c_P £ **f**(**u**,**v**)+c_f(**u**,**v**)
  = **f**(**u**,**v**)+**c**(**u**,**v**)-**f**(**u**,**v**)
  =**c**(**u**,**v**)
- If (**u**,**v**) is a backward edge then **f** and **f'** differ on flow along (**v**,**u**) instead of (**u**,**v**)
  f'(**v**,**u**)=**f**(**v**,**u**)-c_P ³ **f**(**v**,**u**)-c_f(**u**,**v**)
  = **f**(**v**,**u**)-**f**(**v**,**u**)=**0**
- Other conditions like flow conservation still met

Ford-Fulkerson Method

Start with **f=0** for every edge

While G_f has an augmenting path, augment

- Questions:
  - Does it halt?
  - Does it find a maximum flow?
  - How fast?

Observations about Ford-Fulkerson Algorithm

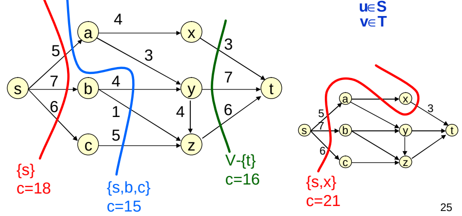
- At every stage the capacities and flow values are always integers (if they start that way)
- The flow value n(**f'**)=n(**f**)+c_P>n(**f**) for **f'**=augment(**f**,**P**)
  - Since edges of residual capacity **0** do not appear in the residual graph
- Let **C**=S$_{(s,u)\,î\,E}$ **c**(**s**,**u**)
  - n(**f**)≤**C**
  - **F-F** does at most **C** rounds of augmentation since flows are integers and increase by at least **1** per step

Running Time of Ford-Fulkerson

- For **f=0**, G_f**=G**
- Finding an augmenting path in G_f is graph search O(**n**+**m**)=O(**m**) time
- Augmenting and updating G_f is O(**n**) time
- Total O(**mC**) time
- Does is find a maximum flow?
  - Need to show that for every flow **f** that isn't maximum G_f contains an **s**-**t**-path

Cuts

- A partition **(S,T)** of **V** is an *s-t-cut* if
  - **s Î S, t Î T**
- *Capacity* of cut (S,T) is $\mathbf{c(S,T)} = \sum_{\substack{u\in S\\ v\in T}} \mathbf{c(u,v)}$



25

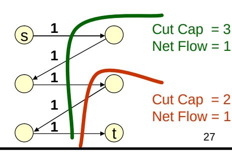
Convenient Definition

- **f^out(A)=S_{v Î A, w Ï A} f (v,w)**
- **f^in(A)=S_{v Î A, u Ï A} f (u,v)**

26

Claims 6.6 and 6.8

- For any flow **f** and any cut **(S,T)**,
  - the net flow across the cut equals the total flow, i.e., ν**(f)** = **f^out(S)-f^in(S)**, and
  - the net flow across the cut cannot exceed the capacity of the cut, i.e. **f^out(S)-f^in(S)** ≤ **c(S,T)**
- Corollary:
  Max flow ≤ Min cut



27

Proof of Claim 6.6

- Consider a set **S** with **s Î S, t Ï S**
- **f^out(S)-f^in(S) =S_{vÎS, wÏS} f (v,w)-S_{vÎS, uÏS} f (u,v)**
- We can add flow values for edges with both endpoints in **S** to **both** sums and they would cancel out so
- **f^out(S)-f^in(S)= S_{vÎS, wÎV} f (v,w)-S_{vÎS, uÎV} f (u,v)**
  = S_{vÎS} (S_{wÎV} f (v,w) - S_{uÎV} f (u,v))
  =S_{vÎS} f^out (v) - f^in(v)
  =f^out(S)-f^in(S)

  since all other vertices have **f^out(v)=f^in(v)**
- ν**(f)** = **f^out(s)** and **f^in(s)=0**

28

Proof of Claim 6.8

- ν**(f)=f^out(S)-f^in(S)**
  ≤ f^out(S)
  = S_{vÎS, wÏS} f (v,w)
  ≤ S_{vÎS, wÏS} c(v,w)
  =c(S,T)

29

Max Flow / Min Cut Theorem

Claim 6.10 For any flow **f**, if $\mathbf{G_f}$ has no augmenting path then there is some **s-t**-cut **(S,T)** such that ν**(f)=c(S,T)** (proof next slide)

- We know by Claims 6.6 & 6.8 that any flow **f'** satisfies ν**(f')** ≤ **c(S,T)** and we know that F-F runs for finite time until it finds a flow **f** satisfying conditions of Claim 6.10
  - Therefore by 6.10 for any flow **f'**, ν**(f')** ≤ν**(f)**
- Corollary (1) F-F computes a maximum flow in **G**

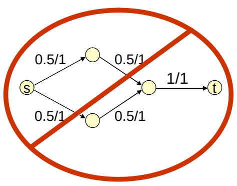
  (2) For any graph **G**, the value ν**(f)** of a maximum flow = minimum capacity **c(S,T)** of any **s-t**-cut in **G**

30

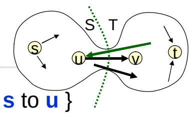
Flow Integrality Theorem

If all capacities are integers

- The max flow has an integer value
- Ford-Fulkerson method finds a max flow in which f(u,v) is an integer for all edges (u,v)



Claim 6.10



Let **S** = { **u** | ∃ an path in G_f from **s** to **u** }

T = V - S; **s** Î **S**, **t** Î **T**

For any **(u,v)** in **S** × **T**, ∃ an path in G_f from **s** to **u**, but not to **v**.

∴ **(u,v)** has **0** residual capacity:

(u,v) ∈ **E** ⇒ saturated   **f(u,v) = c(u,v)**

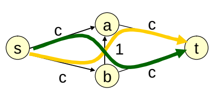
(v,u) ∈ **E** ⇒ no flow   **f(v,u) = 0**

This is true for **every** edge crossing the cut, i.e.

$f^{out}(S) = \sum_{u\in S, v\in T} f(u,v) = \sum_{u\in S, v\in T} c(u,v) = c(S,T)$ and $f^{in}(S)=0$ so $n(f)=f^{out}(S)-f^{in}(S)=c(S,T)$

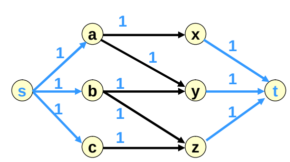
Corollaries & Facts

- If Ford-Fulkerson terminates, then it's found a max flow.
- It will terminate if **c(e)** integer or rational (but may not if they're irrational).
- However, may take exponential time, even with integer capacities:



$c = 10^9$, say

Bipartite matching as a special case of flow



Integer flows implies each flow is just a subset of the edges

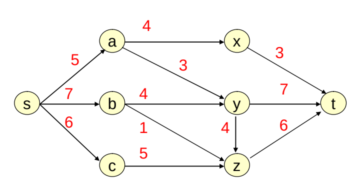
Therefore flow corresponds to a matching

O(mC)=O(nm) running time

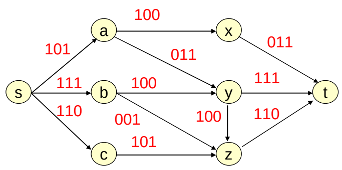
Capacity-scaling algorithm

- General idea:
  - Choose augmenting paths **P** with 'large' capacity c_P
  - Can augment flows along a path **P** by any amount **b£**c_P
    - Ford-Fulkerson still works
  - Get a flow that is maximum for the high-order bits first and then add more bits later

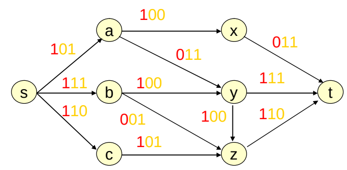
Capacity Scaling



Capacity Scaling

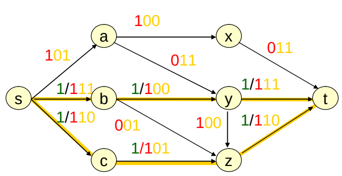


Capacity Scaling Bit 1



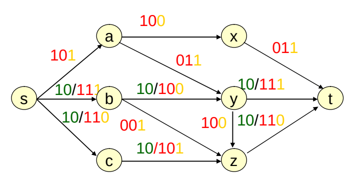
Capacity on each edge is at most **1**

Capacity Scaling Bit 1



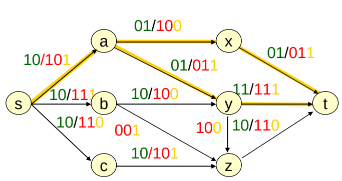
O(nm) time

Capacity Scaling Bit 2



Residual capacity across min cut is at most **m**

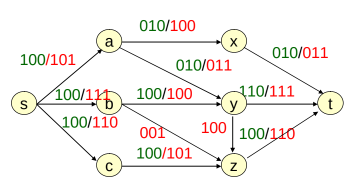
Capacity Scaling Bit 2



Residual capacity across min cut is at most **m**

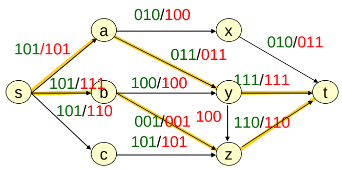
Þ **O(m)** augmentations

Capacity Scaling Bit 3



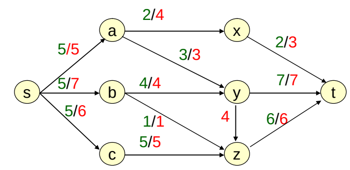
Residual capacity across min cut is at most **m**

Capacity Scaling Bit 3

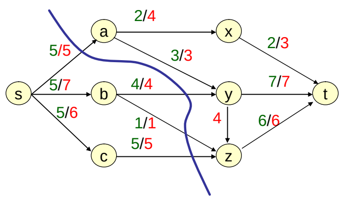


After $O(\mathbf{m})$ augmentations

Capacity Scaling Final



Capacity Scaling Min Cut



Total time for capacity scaling

- $\log_2 \mathbf{U}$ rounds where $\mathbf{U}$ is largest capacity
- At most $\mathbf{m}$ augmentations per round
  - Let $\mathbf{c_i}$ be the capacities used in the $\mathbf{i}^{\text{th}}$ round and $\mathbf{f_i}$ be the maxflow found in the $\mathbf{i}^{\text{th}}$ round
    - For any edge $(\mathbf{u},\mathbf{v})$, $\mathbf{c_{i+1}}(\mathbf{u},\mathbf{v}) \le 2\mathbf{c_i}(\mathbf{u},\mathbf{v})+1$
  - $\mathbf{i+1}^{\text{st}}$ round starts with flow $\mathbf{f} = 2\,\mathbf{f_i}$
  - Let $(\mathbf{S},\mathbf{T})$ be a min cut from the $\mathbf{i}^{\text{th}}$ round
    - $n(\mathbf{f_i})=\mathbf{c_i}(\mathbf{S},\mathbf{T})$ so $n(\mathbf{f})=2\mathbf{c_i}(\mathbf{S},\mathbf{T})$
  - $n(\mathbf{f_{i+1}}) \le \mathbf{c_{i+1}}(\mathbf{S},\mathbf{T}) \le 2\mathbf{c_i}(\mathbf{S},\mathbf{T})+\mathbf{m} = n(\mathbf{f})+\mathbf{m}$
- $O(\mathbf{m})$ time per augmentation
- Total time $O(\mathbf{m}^2 \log \mathbf{U})$

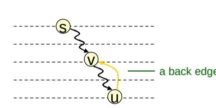
Edmonds-Karp Algorithm

- Use a shortest augmenting path
  (via Breadth First Search in residual graph)
- Time: $O(\mathbf{n}\ \mathbf{m}^2)$

BFS/Shortest Path Lemmas

Distance from $\mathbf{s}$ in $\mathbf{G_f}$ is never reduced by:

- Deleting an edge
  Proof: no new (hence no shorter) path created
- Adding an edge $(\mathbf{u},\mathbf{v})$, provided $\mathbf{v}$ is nearer than $\mathbf{u}$
  Proof: BFS is unchanged, since $\mathbf{v}$ visited before $(\mathbf{u},\mathbf{v})$ examined

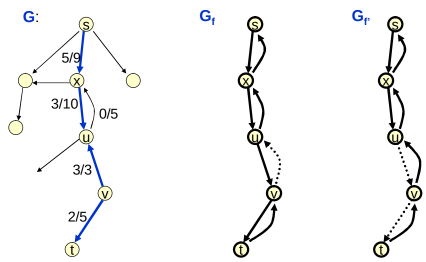


Key Lemma

Let f be a flow, G_f the residual graph, and P a shortest augmenting path. Then no vertex is closer to **s** after augmentation along **P**.

Proof: Augmentation along **P** only deletes forward edges, or adds back edges that go to previous vertices along **P**

Augmentation vs BFS



Theorem

The Edmonds-Karp Algorithm performs O(**mn**) flow augmentations

Proof:

Call (**u**,**v**) critical for augmenting path **P** if it's closest to **s** having min residual capacity

It will disappear from G_f after augmenting along **P**

In order for **(u,v)** to be critical again the **(u,v)** edge must re-appear in G_f but that will only happen when the distance to **u** has increased by **1**

It won't be critical again until farther from **s** so each edge critical at most **n** times

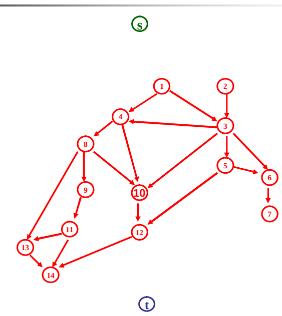
Corollary

- Edmonds-Karp runs in O(**nm²**) time

Project Selection a.k.a. The Strip Mining Problem

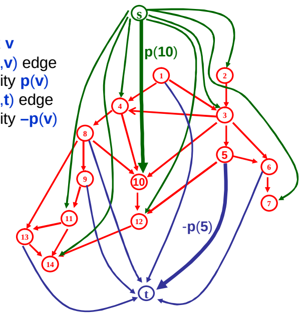
- Given
  - a directed acyclic graph **G**=(**V**,**E**) representing precedence constraints on tasks (a task points to its **predecessors**)
  - a profit value **p**(**v**) associated with each task **v** Î **V** (may be positive or negative)
- Find
  - a set **A** Í **V** of tasks that is closed under predecessors, i.e. if (**u**,**v**)∈**E** and **u** Î **A** then **v** Î **A**, that maximizes Profit(**A**)=$S_{v\,Î\,A}$ **p**(**v**)

Extended Graph



Extended Graph G'

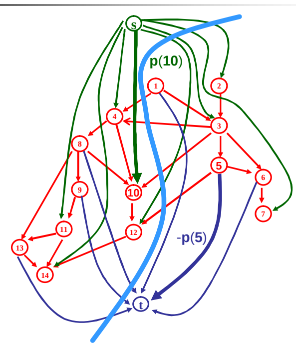
For each vertex **v**
If **p(v)≥0** add **(s,v)** edge
with capacity **p(v)**
If **p(v)<0** add **(v,t)** edge
with capacity **–p(v)**



Extended Graph G'

- Want to arrange capacities on edges of **G** so that for minimum **s-t**-cut **(S,T)** in **G'**, the set **A=S-{s}**
  - satisfies precedence constraints
  - has maximum possible profit in **G**
- Cut capacity with **S={s}** is just **C=**$\sum_{v:\, p(v)\geq 0}$ **p(v)**
  - **Profit(A)** ≤ **C** for any set **A**
- To satisfy precedence constraints don't want any original edges of **G** going forward across the minimum cut
  - That would correspond to a task in **A=S-{s}** that had a predecessor not in **A=S-{s}**
- Set capacity of each of these edges to **C+1**
  - The minimum cut has size at most **C**

Extended Graph G'



Project Selection

- **Claim** Any **s-t**-cut **(S,T)** in **G'** such that **A=S-{s}** satisfies precedence constraints has capacity

  c(S,T)=C - $\sum_{v\in A}$ **p(v)** = **C** - **Profit(A)**

- **Corollary** A minimum cut **(S,T)** in **G'** yields an optimal solution **A=S-{s}** to the profit selection problem
- **Algorithm** Compute maximum flow **f** in **G'**, find the set **S** of nodes reachable from **s** in G'_f and return **S-{s}**

Proof of Claim

- **A=S-{s}** satisfies precedence constraints
  - No edge of **G** crosses forward out of **A** by our choice of capacities
  - Only forward edges cut are of the form **(v,t)** for $v \in A$ or **(s,v)** for $v \notin A$
  - The **(v,t)** edges for $v \in A$ contribute

    $\sum_{v\in A:p(v)<0} -p(v) = -\sum_{v\in A:p(v)<0} p(v)$
  - The **(s,v)** edges for $v \notin A$ contribute

    $\sum_{v\notin A:\, p(v)\geq 0} p(v) = C - \sum_{v\in A:\, p(v)\geq 0} p(v)$
  - Therefore the total capacity of the cut is

    $c(S,T) = C - \sum_{v\in A} p(v) = C - \text{Profit}(A)$