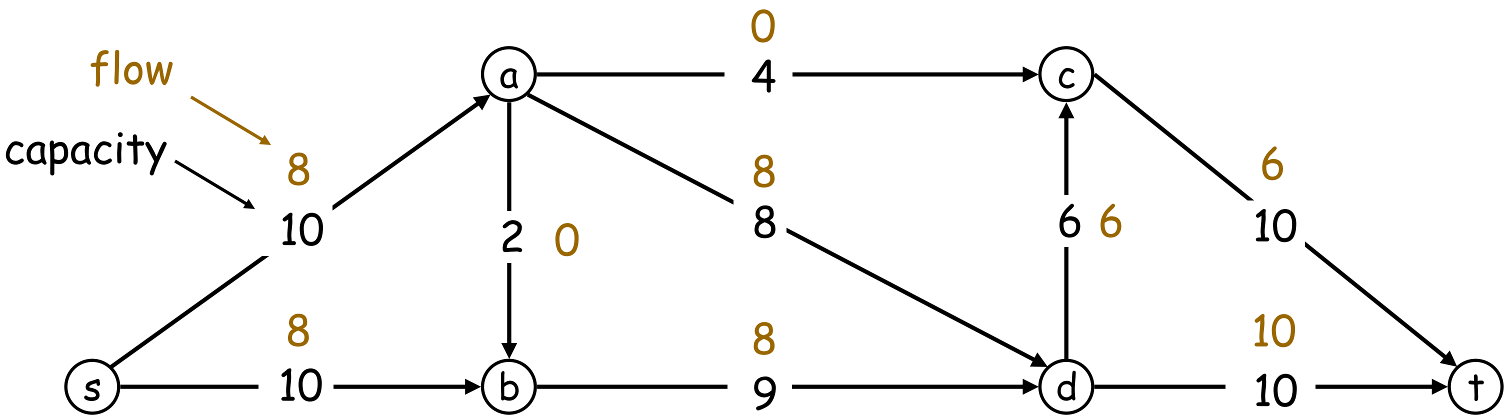
CSC3160: Design and Analysis of Algorithms

Week 8: Maximum Network Flow

Instructor: Shengyu Zhang

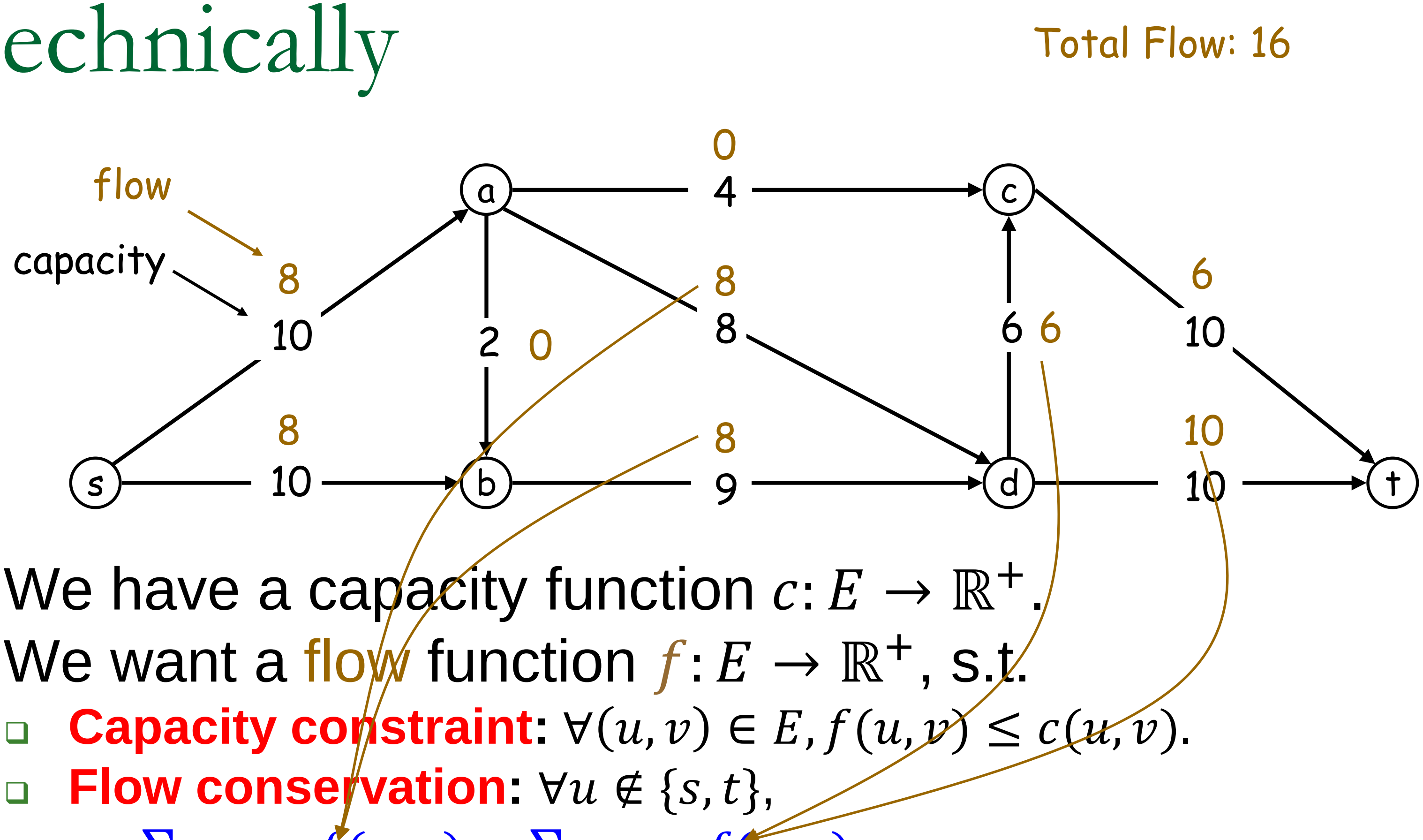
Transportation

Total Flow: 16



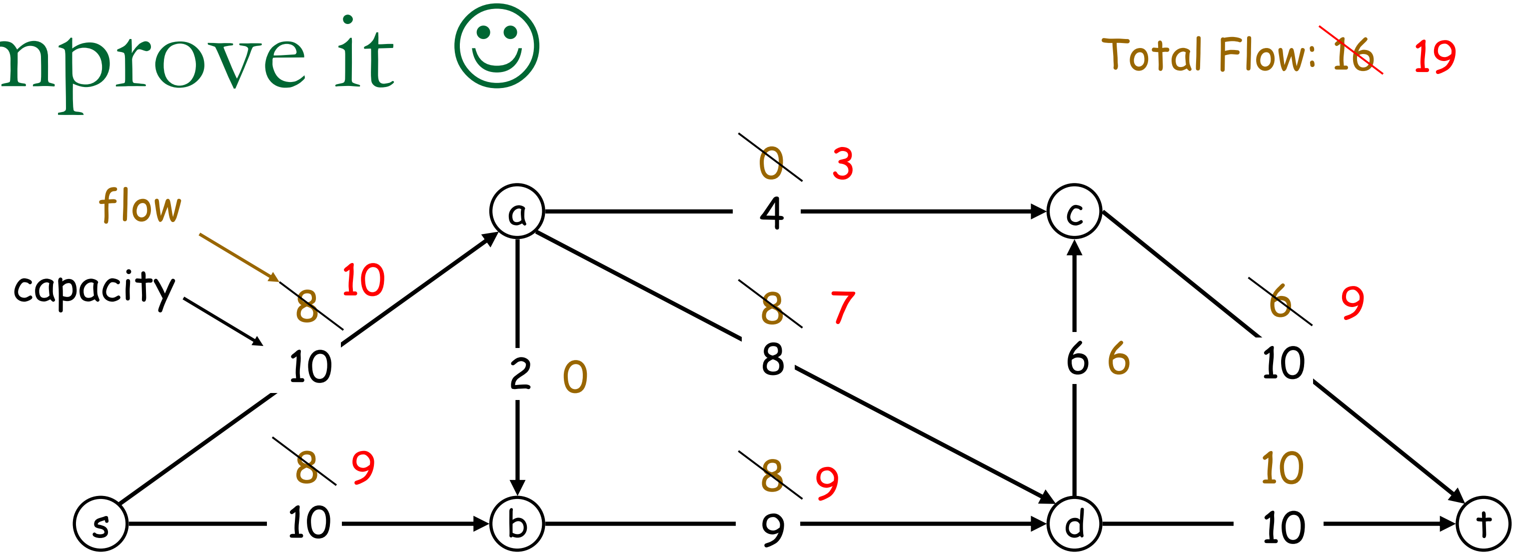
- Suppose we want to transport commodity from s to t in a directed graph $G = (V, E)$.
- Each directed edge $(u, v) \in E$ has a capacity
  - Max amount of commodity allowed
- *Question: How much can we transport?*

Technically



- We have a capacity function $c: E \rightarrow \mathbb{R}^+$.
- We want a flow function $f: E \rightarrow \mathbb{R}^+$, s.t.
  - **Capacity constraint:** $\forall (u,v) \in E, f(u,v) \leq c(u,v)$.
  - **Flow conservation:** $\forall u \notin \{s,t\}$,
    $$\sum_{(v,u)\in E} f(v,u) = \sum_{(u,v)\in E} f(u,v)$$
    - Incoming flow = outgoing flow
  - **Goal:** $\max_f \left(\sum_{(s,v)\in E} f(s,v) - \sum_{(u,s)\in E} f(u,s) \right)$
    - Net flow = flow going out of source – flow coming back into source

Improve it ☺



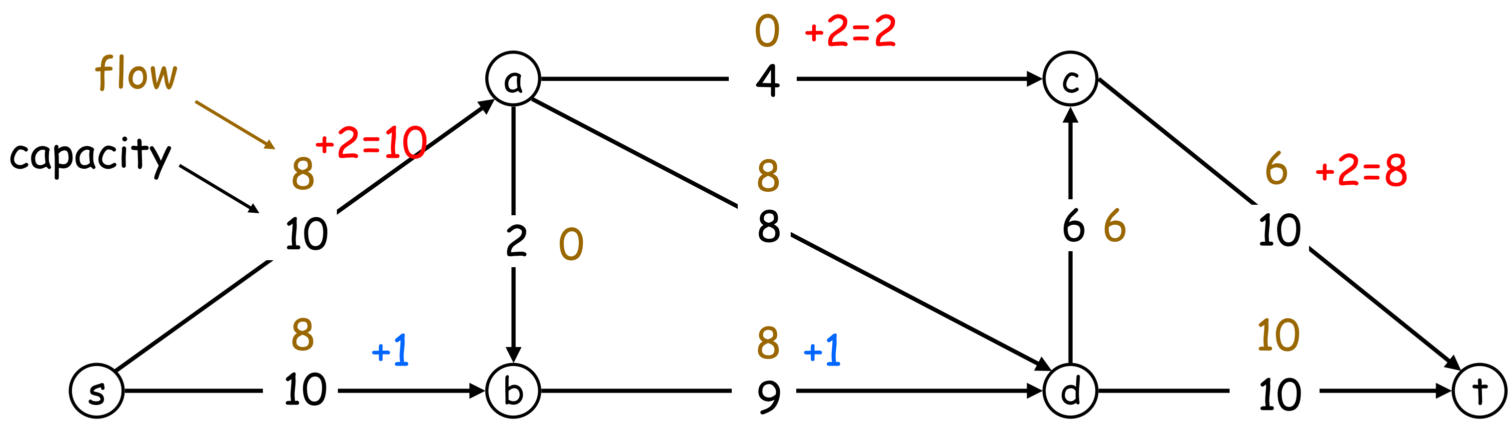
- Can we improve this?
  - For some edges: we gave too much.
  - For some other edges: we didn't give enough.
- Can we further improve this?

Network Flow

- Can you give a good algorithm?
- Methodology 5: Approach to the optimum by a sequence of improvements.

Improve it little by little

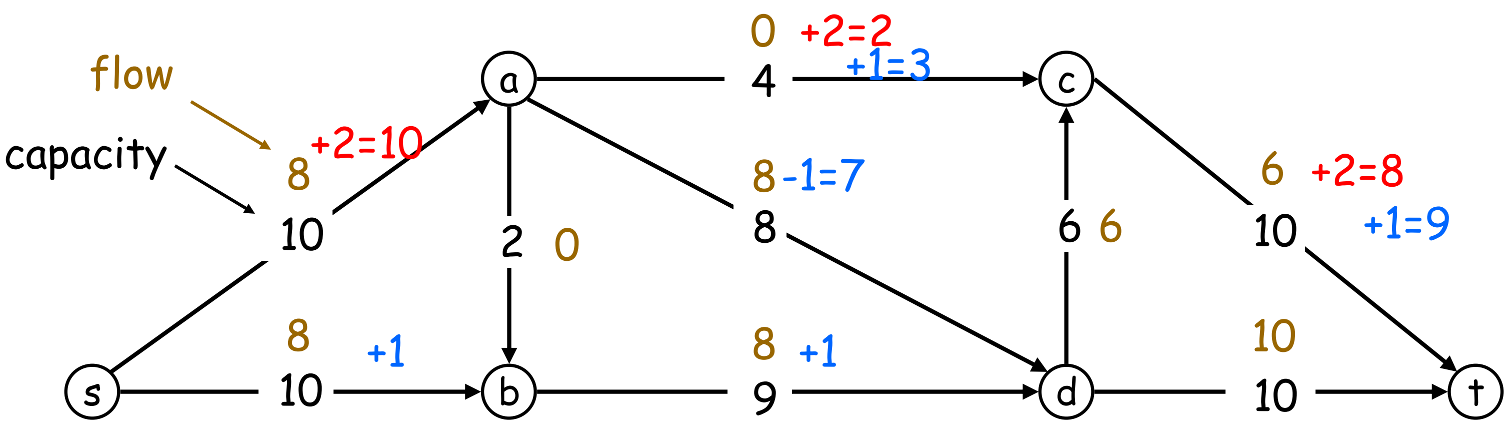
Total Flow: ~~16~~ 18



- We can find a path $s \to a \to c \to t$ on which we can add 2 (on every edge)
- Total flow becomes 18.
- We also like to add 1 via $s \to b \to d \to t$, …
- but the edge $d \to t$ is a bottleneck.
  - Actually the edge $d \to c$ is as well.

Improve it little by little

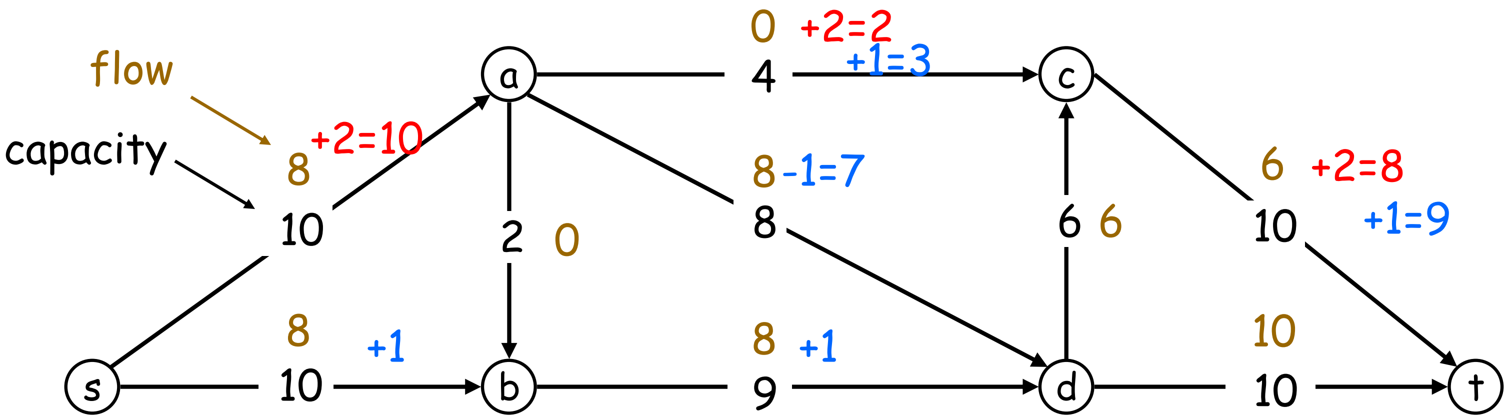
Total Flow: ~~16~~ ~~18~~ 19



- We can still squeeze some juice along the path $a \to c \to t$.
- But vertex a already allocates all its incoming 10 flow.
- Let's withdraw 1 unit on the edge $a \to d$ and assign it along $a \to c \to t$!
- Total flow becomes 19.
- In some sense, it looks like we injected a unit of flow along $s \to b \to d \to a \to c \to t$.

Improve it little by little

Total Flow: ~~16~~ ~~18~~ 19

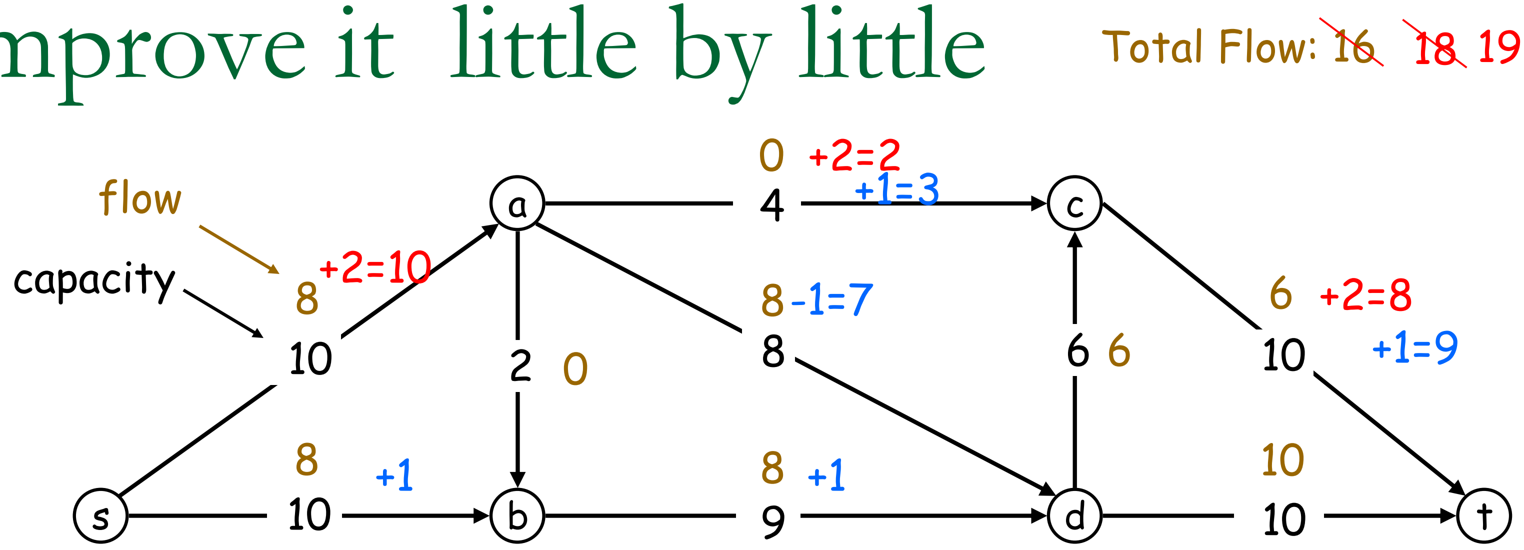


Ford-Fulkerson Algorithm:

- initialize flow f to 0
- **while** there exists an *augmenting path* p
  - Inject more flow along p (as much as possible)
- **return** f

- Question 1: What is an augmenting path?

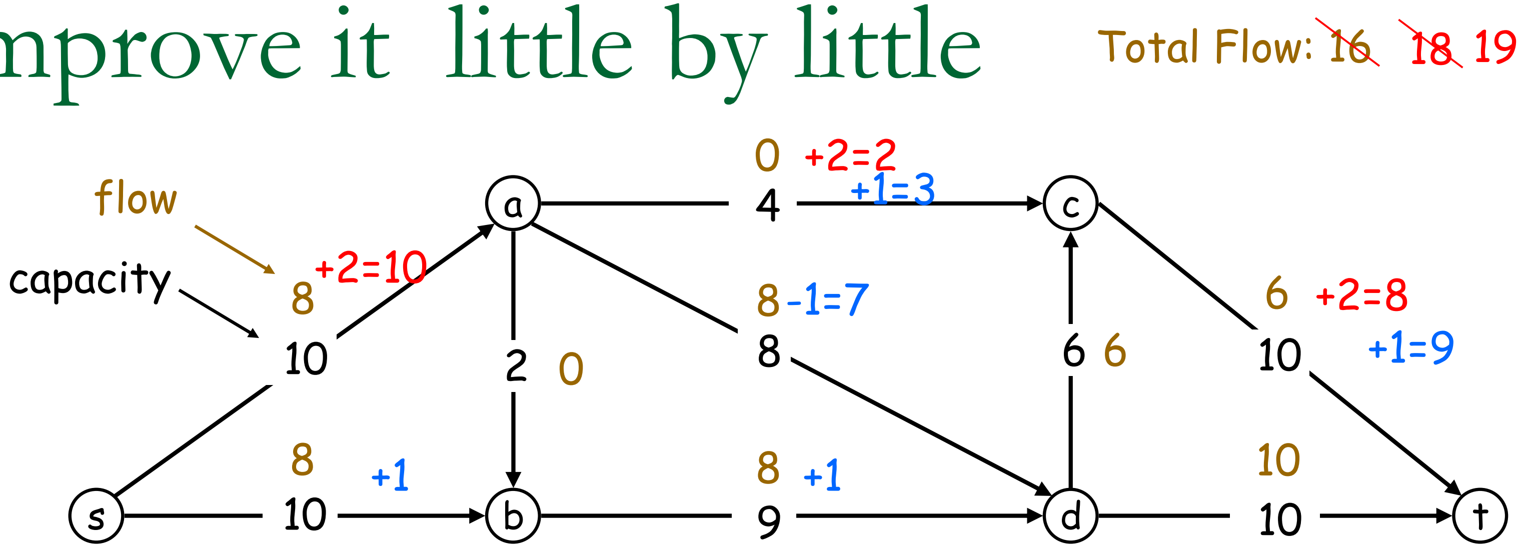
Improve it little by little



- Case 1: if the capacity hasn't been used up for each edge on the path, then it's an augmenting path.

- Case 2: if some edge $u \rightarrow v$ already has a flow, then it amounts to a capacity in direction $v \rightarrow u$
  - By withdrawing the previously assigned flow.

Improve it little by little

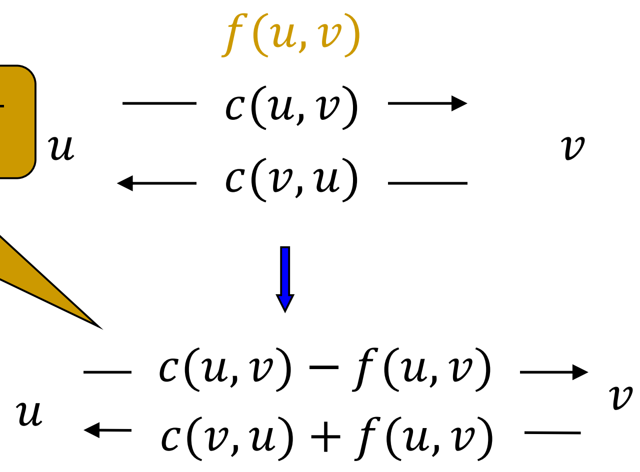


- Question 2: How to find an augmenting path?
- By *residual networks.*

Residual networks

- For a flow f, the residual capacity c_f is:
  - $c_f(u,v) = c(u,v) - f(u,v)$
  - $c_f(v,u) = c(v,u) + f(u,v)$
- Residual network:
  - $G_f = (V, E_f)$, where
  - $E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$.
- Now an augmenting path is just a path from s to t in the residual network.

We can still inject up to $c(u,v) - f(u,v)$ flow from u to v



We can withdraw f flow from u to v first, and then inject up to $c(v,u)$ flow from v to u

Residual networks

- The residual network gives the info of how we can get more flow from s to t in the graph.
  - And how much.
- So we define an augmenting path to be a path from s to t in the residual network.
- Now finding an augmenting path amounts to finding a path from s to t in the residual network,
- which we know how to do
  - e.g. BFS algorithm.

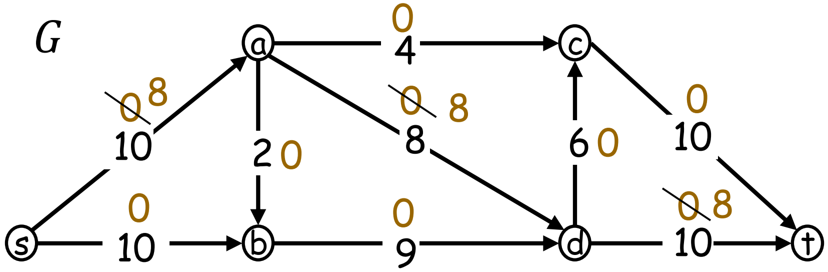
FORD-FULKERSON(G, s, t)

- **for** each edge $(u, v) \in E$ // **Initialization**
  - $f(u, v) \leftarrow 0, f(v, u) \leftarrow 0$
- $G_f = G$
- **while** there exists a path p from s to t in the residual network G_f
  - $c_f(p) \leftarrow \min\{c_f(u, v)\colon (u, v)\ is\ in\ p\}$ // max to inject on p
  - **for** each edge $(u, v)\ in\ p$ // update flow
    - **if** $f(v, u) = 0$, // no backward flow on this edge
      - $f(u, v) \leftarrow f(u, v) + c_f(p)$
    - **else if** $c_f(p) \leq f(v, u)$ // has backward flow, withdraw part
      - $f(v, u) \leftarrow f(v, u) - c_f(p)$
    - **else** // has backward flow, withdraw all, add forward flow
      - $f(u, v) \leftarrow c_f(p) - f(v, u)$
      - $f(v, u) \leftarrow 0$
  - Update the residual network G_f.

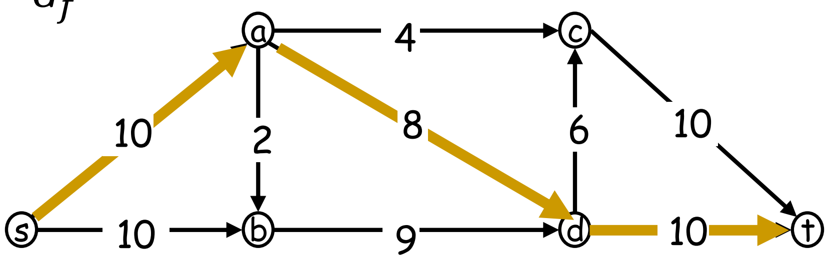
FORD-FULKERSON(G, s, t)

- **for** each edge $(u, v) \in E$
  - $f(u, v) \leftarrow 0, f(v, u) \leftarrow 0$
- $G_f = G$
- **while** there exists path p from s to t in residual network G_f
  - $c_f(p) \leftarrow \min\{c_f(u, v):\ (u, v) \text{ is in } p\}$
  - **for** each edge (u, v) in p
    - **if** $f(v, u) = 0,$
      - $f(u, v) \leftarrow f(u, v) + c_f(p)$
    - **else if** $c_f(p) \le f(v, u)$
      - $f(v, u) \leftarrow f(v, u) - c_f(p)$
    - **else**
      - $f(u, v) \leftarrow c_f(p) - f(v, u)$
      - $f(v, u) \leftarrow 0$
  - Update the residual network G_f.

G



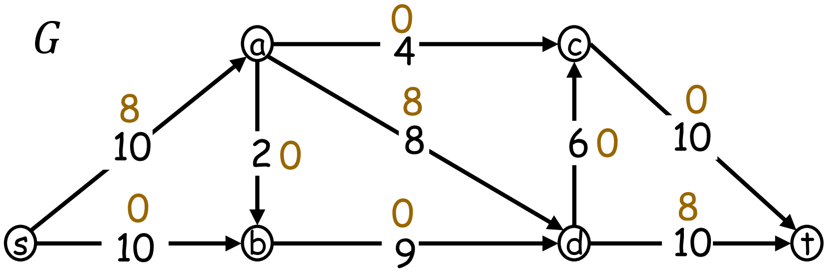
G_f

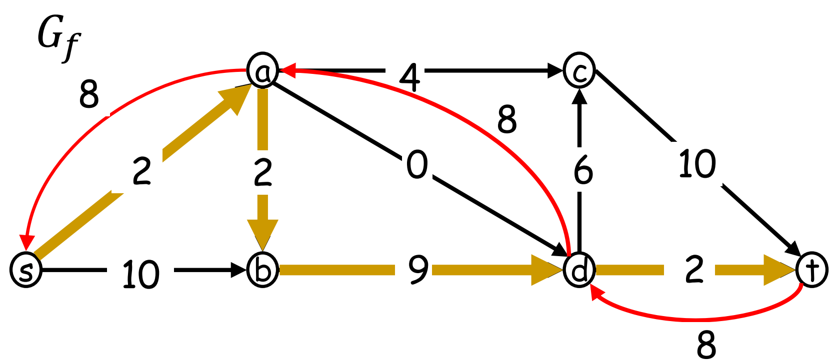


$c_f(p) = 8, \text{flow} = 8$

FORD-FULKERSON(G, s, t)

- **for** each edge $(u, v) \in E$
  - $f(u, v) \leftarrow 0, f(v, u) \leftarrow 0$
- $G_f = G$
- **while** there exists path p from s to t in residual network G_f
  - $c_f(p) \leftarrow \min\{c_f(u, v)\text{: } (u, v) \text{ is in } p\}$
  - **for** each edge (u, v) in p
    - **if** $f(v, u) = 0,$
      - $f(u, v) \leftarrow f(u, v) + c_f(p)$
    - **else if** $c_f(p) \leq f(v, u)$
      - $f(v, u) \leftarrow f(v, u) - c_f(p)$
    - **else**
      - $f(u, v) \leftarrow c_f(p) - f(v, u)$
      - $f(v, u) \leftarrow 0$
  - Update the residual network G_f.





$c_f(p) = 8, \text{flow} = 8$

New $c_f(p)$? New flow? New G_f?

Questions left

- How to find an augmenting path?
- What if, at some step, there is no augmenting path in the residual network?
  - Can we conclude that we've found the maximum flow?

Cut

- Cut: a partition of vertices into two parts S and T.
- capacity of cut (S, T): $\sum_{u \in S, v \in T} c(u, v)$.
- Fact. Flow $\leq$ capacity of any cut (S, T).
- Proof.

flow value of f = net flow from S to T

$=$ flow S to T $-$ flow T to S *// conservation*

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

$\leq \sum_{u \in S, v \in T} c(u, v)$

- How good is this upper bound of flow?

Max-flow min-cut Theorem.

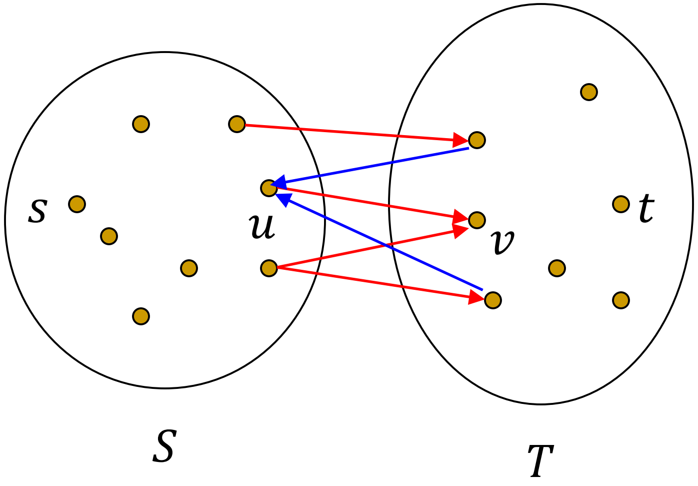
- An important fact relating max flow and min cut: the previous upper bound is perfect
  - as long as we find a correct cut.
- [Theorem] The following are equivalent:

1. f is a maximum flow in G
2. G_f contains no augmenting paths.

- [Proof] 1 ⇒ 2: trivial since otherwise f can be further increased.
- Next: 2 ⇒ 1.
  - In the proof you'll see a cut with capacity achieving the max flow.

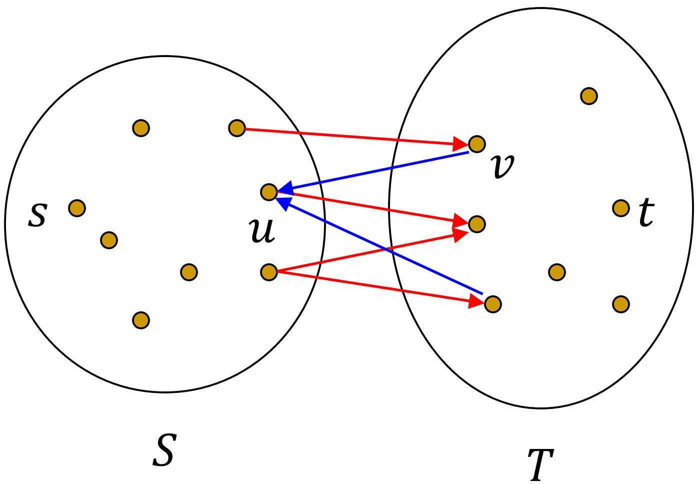
G_f contains no augmenting paths $\Rightarrow f$ is a maximum flow in G

- Consider all vertices in G_f reachable from s.
  - Call the set S.
  - The rest is T.
  - $s \in S, t \in T$.
- Consider this cut (S, T):
  - Two types of crossing edges in G
    - Type 1: $S \to T$. (u, v): $u \in S, v \in T$.
    - Type 2: $T \to S$. (v, u): $u \in S, v \in T$.
  - For type 1: $f(u, v) = c(u, v)$
    - Otherwise v is reachable from s in G_f!



G_f contains no augmenting paths $\Rightarrow f$ is a maximum flow in G

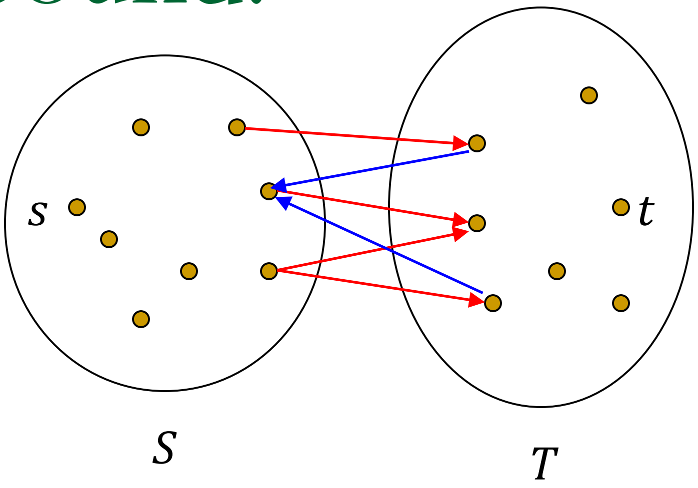
- Consider all vertices in G_f reachable from s.
  - Call the set S.
  - The rest is T.
  - $s \in S, t \in T$.
- Consider this cut (S, T):
  - Two types of crossing edges in G
    - Type 1: $S \to T$. $(u, v): u \in S, v \in T$
    - Type 2: $T \to S$. $(v, u): u \in S, v \in T$
  - For type 1: $f(u, v) = c(u, v)$
    - Otherwise v is reachable from s in G_f!
  - For type 2: $f(v, u) = 0$.
    - Otherwise v is also reachable from s in G_f!



Done!

Why?

Any cut gives an upper bound!



- flow value of f
  $= $ flow "$S \to T$" $-$ flow "$T \to S$"
  $= \sum_{(u,v):\text{type 1}} f(u,v)$
  $\quad - \sum_{(u,v):\text{type 2}} f(v,u)$
  $\leq \sum_{(u,v):\text{type 1}} c(u,v)$
- i.e. the best we can hope for f is to
  - use up full capacity of all type 1 edges
  - not to use any capacity of any type 2 edge.
- This essentially repeats the proof of flow $\leq$ cut capacity.
- Last slide: If G_f has no augmenting path, then f already satisfies these two properties.
  - Thus f achieves $\sum_{(u,v):\text{type 1}} c(u,v)$---It is maximum.

Next

- How to find an augmenting path?
  - It matters. If we don't pick a good path, it may take forever and may not even converge to the optimum.
- [Edmonds-Karp] Use a shortest path will do.
  - Unweighted, thus BFS suffices.
- [Fact] At most $O(|V| \cdot |E|)$ augmenting path findings.
- Using BFS costs $O(|E|)$ for each path, thus $O(|V| \cdot |E|^2)$ for the total cost.

Edmonds-Karp Algorithm

- **for** each edge $(u, v) \in E$
  - $f(u, v) \leftarrow 0, f(v, u) \leftarrow 0$
- $G_f = G$
- **while** we can use BFS to find a shortest path p from s to t in the residual network G_f
  - $c_f(p) \leftarrow \min\{c_f(u, v): (u, v) \text{ is in } p\}$
  - Update the flow f
  - Update the residual network G_f.

- Complexity? Depends on how many iterations are executed in the **while** loop.
- [Thm] $O(|V| \cdot |E|)$ iterations.

Edmonds-Karp Algorithm

- **for** each edge $(u, v) \in E$
  - $f(u, v) \leftarrow 0, f(v, u) \leftarrow 0$
- $G_f = G$
- **while** we can use BFS to find a shortest path p from s to t in the residual network G_f
  - $c_f(p) \leftarrow \min\{c_f(u, v)\colon (u, v) \text{ is in } p\}$
  - Update the flow f
  - Update the residual network G_f.

- An edge (u, v) is critical on an augmenting path p if the "min" in $c_f(p) \leftarrow \min\{c_f(u, v)\colon (u, v) \text{ is in } p\}$ is achieved by (u, v)

Analysis

- [Lemma] Any (u, v) can be critical at most $|V|/2$ times.

- Once we prove this, we are done proving the theorem of "$O(|V||E|)$ iterations",
  - because there are $|E|$ edges, so at most $|V||E|/2$ iterations in total.

Proof of the lemma

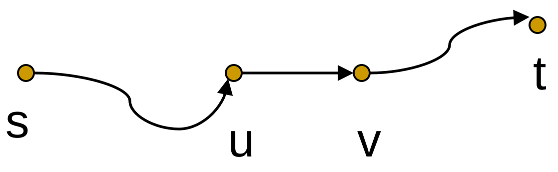
- We'll prove that each time (u, v) becomes critical, the distance $d(s, u)$ increases by at least 2.
  - $d(s, u)$: least number of edges on a path from s to u in graph G_f
- Since $0 < d(s, u) < |V|$, (u, v) is critical at most $|V|/2$ times.

Proof (continued)

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

$$d_f(s, v) = d_f(s, u) + 1.$$

  - d_f: distance on G_f.

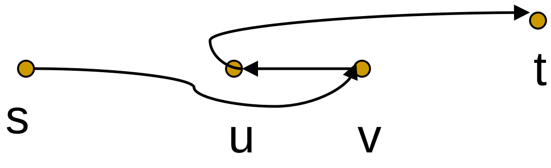


- Once the flow is augmented, the edge (u, v) disappears from the residual network.
  - Critical: $f(u, v) = c(u, v)$,
  - So $c_f(u, v) = 0$, i.e. (u, v) disappears from the residual network.

Proof (continued)

- It cannot reappear later on another augmenting path until after the flow from u to v is decreased,
  - which occurs only if (v, u) appears on an augmenting path.
- If f' is the flow in G when this event occurs, then we have

$$d_{f'}(s,u) = d_{f'}(s,v) + 1.$$



Proof (continued)

- We've shown
  - $d_f(s,v) = d_f(s,u) + 1$
  - $d_{f'}(s,u) = d_{f'}(s,v) + 1$
- Now if $d_{f'}(s,v) \geq d_f(s,v)$ …

Exercise!

- Then $d_{f'}(s,u) = d_{f'}(s,v) + 1$

$$\geq d_f(s,v) + 1$$

$$= d_f(s,u) + 2.$$

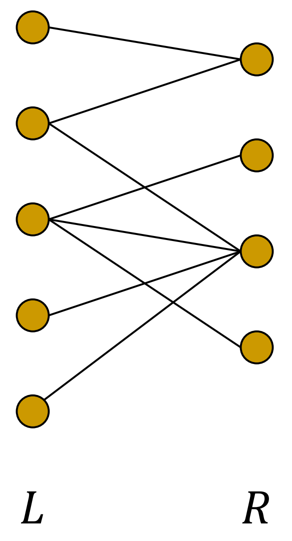
- Mission accomplished!

Application: Max bipartite matching

- We've learned maximum flow problem and algorithms.
- Next we apply it to solve the maximum bipartite matching problem.

Maximum bipartite matching

- Bipartite graph: $G = (V, E)$ that can be partitioned into two parts with all edges crossing
  - $V = L \cup R$ with $L \cap R = \emptyset$,
  - All edges $(i, j) \in E$ have $i \in L$ and $j \in R$.
- Matching: a collection of edges (i_k, j_k) that are vertex disjoint
  - All i_k's are distinct. So are all j_k's.

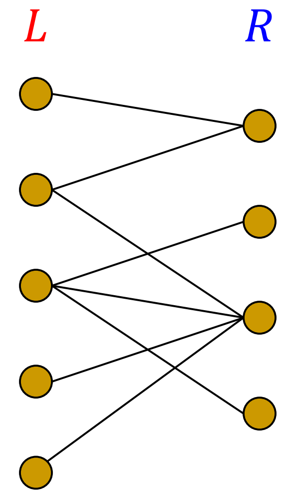


- Question: Find a max matching in a bipartite graph.
  - Max matching: matching with maximum number of edges.

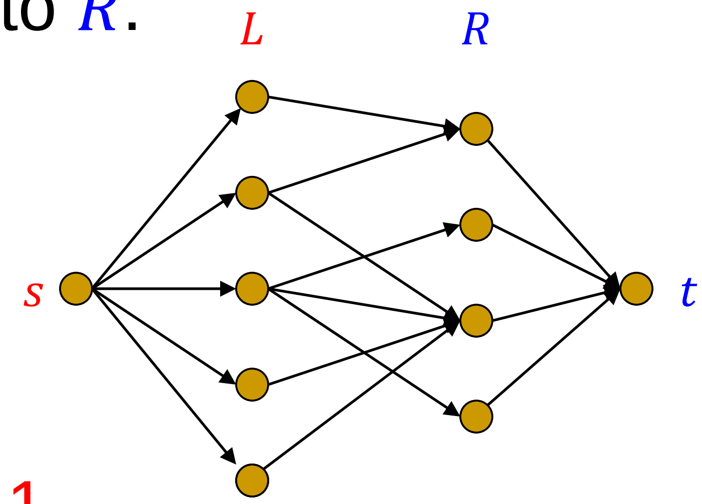
- Methodology 0: See whether the problem can be reduced to another one whose answer is known.

- Very often, the problem that you are facing appeared to other people before.
  - Solutions are known.
- Also very often, the problem is probably new, but it’s very similar to, or essentially the same as an old one.
- Then a simple transformation or reduction works.

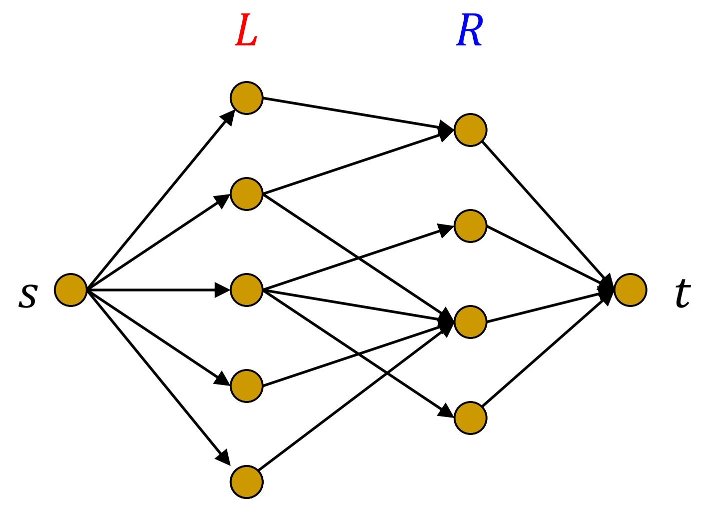
- Orient existing edges from L to R.



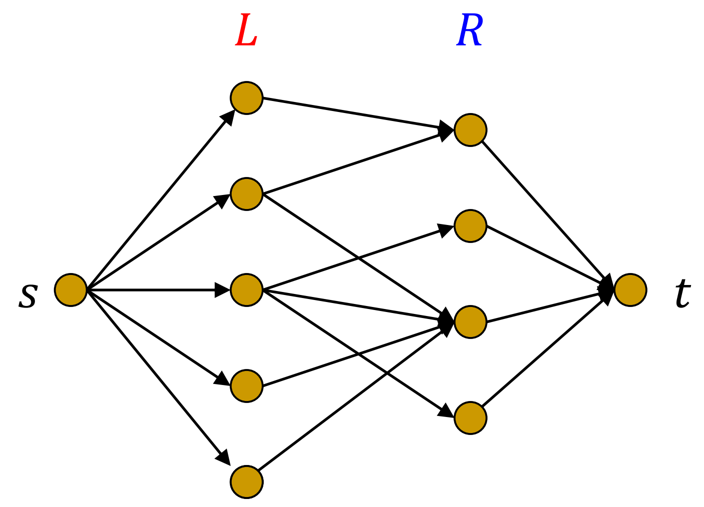
- Orient existing edges from L to R.
- Add one more node s.
- Link s to all vertices in L.
- Add one more node t.
- Link all vertices in R to t.
- All capacities (on edges) are 1.



Equivalence



- [Fact] ∃ matching of size m in original graph ⇔ ∃ integral flow of value m in the new graph
  - Integral: flow is integer on each edge
- ⇒: For matching $\{(i_k, j_k): k = 1, \dots, m\}$, give a unit flow to each edge (s, i_k), (i_k, j_k), and (j_k, t).
- ⇐: Since all capacities are 1 and flow is integral, flow on each edge is either 0 or 1.
  - So there are m "middle" edges (i_k, j_k) with flow 1.
  - And these edges are all vertex-disjoint because of the flow conservation.

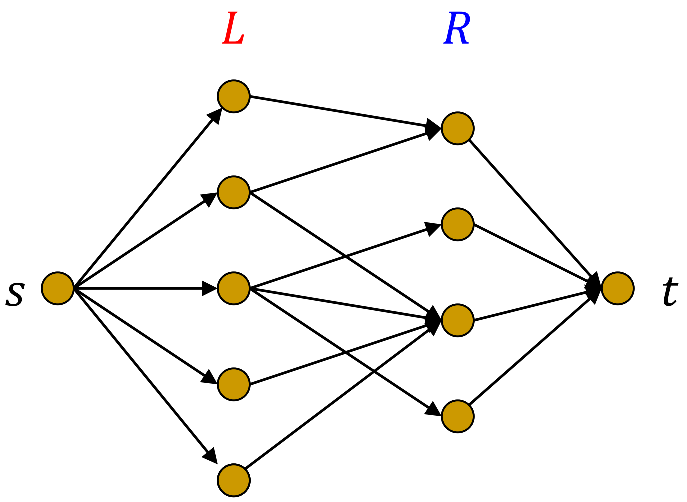


- [Fact] ∃ matching of size s in original graph ⇔ ∃ integral flow of value s in the new graph
  - Integral: flow is integer on each edge
- [Fact] maximum matching in the original graph ⇔ maximum integral flow in the new graph.
- So it's sufficient to find a maximum integral flow in the new graph.
- We've learned how to find a maximum flow. But how to handle the integral constraint?
- Answer: We don't handle it.

Integral constraint: automatic

- [Fact] In a graph with integral capacities, max flow is achieved by integral flows.
- Why? By our algorithm!
- Each time we follow an augmenting path to increase the flow …
  by how much?
  - $c_f(p) \leftarrow \min\{c_f(u,v): (u,v) \text{ is in } p\}$
  - It's an integer!
  - So the total flow is always an integer during the algorithm.
  - In particular, the final answer, i.e. a max flow, is an integral flow.

algorithm



- Overall, the algorithm is as follows.
  - Create the new graph.
    - Orienting edges, adding s and t, giving unit capacity.
  - Find a max flow of the new graph.
  - Output middle edges with flow 1.

Summary

- Network flow problem.
- Augmenting path algorithm.
- Why correct? Max-flow Min-cut Theorem.
- How to find? One way: Shortest one by BFS.
- Complexity? $O(|V||E|^2)$ by analysis.
- One application: max bipartite matching