Network Flow

How does one <u>design an algorithm</u>?

- Identify a known paradigm, or
- Take a fresh approach.
  - → by considering small examples
  - → learning by mistakes
  - → building a theory/notation.

– What is a <u>network</u>?

Eg. network of pipes with some fluid,
network of roads or rails,
network of wires.

– What is a <u>flow</u>?

Flow along an edge is a number limited by the <u>capacity</u>.

Flow at every vertex should be <u>conserved</u>, i.e. incoming & outgoing flows are the same.

Defn: • Network is modelled as a graph $G = (V, E, c)$ with source s & sink t.

• Flow is modelled as an edge-weight assignment f s.t. $\forall (x,y) \in E$, $f(x,y) \leq c(x,y)$ and $\forall v \in V \setminus \{s,t\}$:

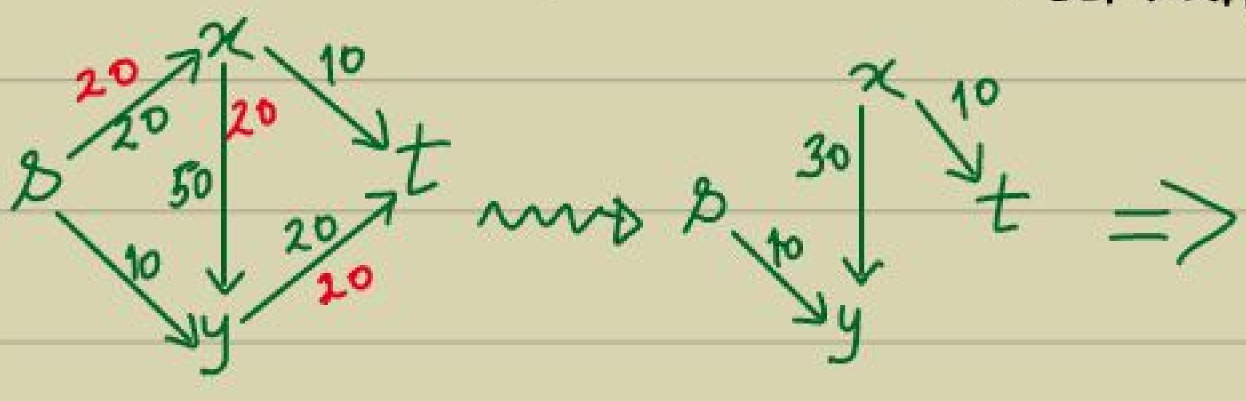
$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w).$$

• $\text{value}(f) :=$ flow out of $s = \sum_{(s,v)\in E} f(s,v)$.

• Max-flow problem is to compute the maximum value(f).

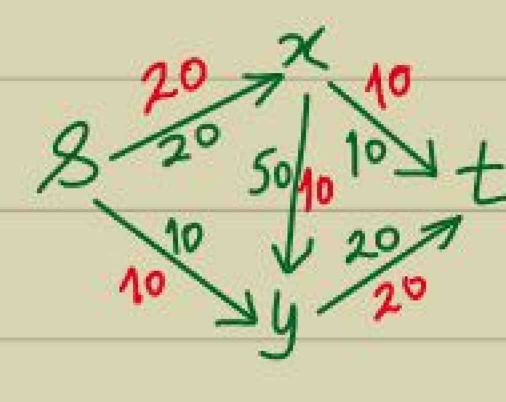
It's a linear program.

– Idea 0: Find an s-t path. Send minimum capacity along it. Repeat the above on the remaining capacity.

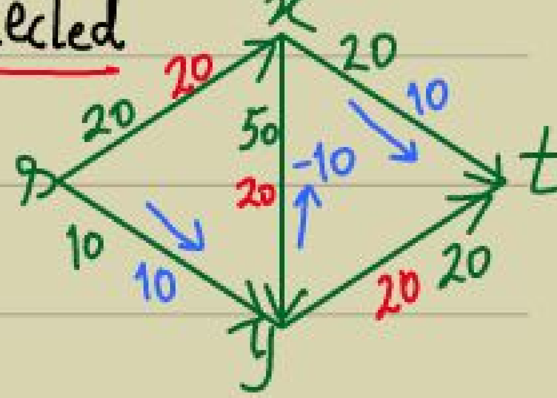


gives flow=20

But max. flow = 30:



- *Idea 1*: Redistribution of flow might be required.
  - increase flow along an edge, &
  - decrease flow " " ".
  - do this along an undirected s-t path.



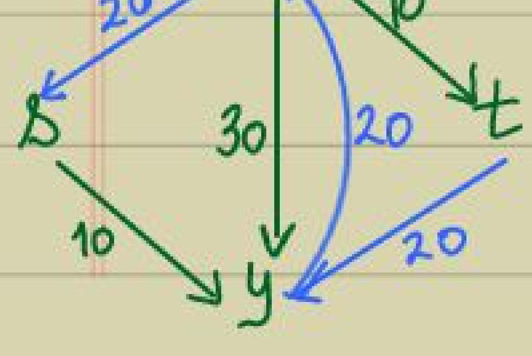
- The new structure is –

  Residual Network

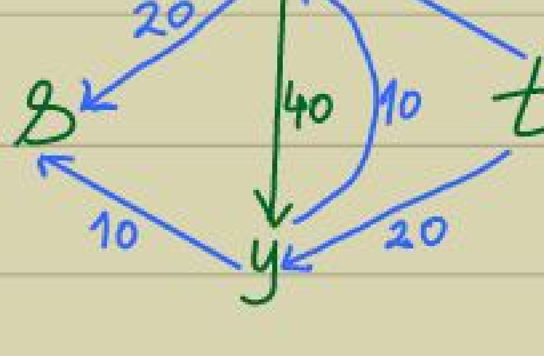
- *Defn*: • For a network $G=(V,E,c)$ with an s-t flow f, the residual network $G_f=(V,E_f,c_f)$ has edge (x,y) with capacity
  $c_f(x,y) := c(x,y) - f(x,y)$ & if $f(x,y)>0$
  then the edge (y,x) with capacity $c_f(y,x) := f(x,y) + c(y,x)$.

  • The former is called a forward edge & the latter is a backward edge.

Egs:



use flow=10



- This construction suggests an interesting iterative algorithm:

Find an s-t path in the current G_f. Update the flow & G_f. Repeat!

```
Ford-Fulkerson(G=(V,E,c)) {
    f ← 0;
    while (∃ s-t path in G_f) {        compute the residual graph wrt flow f
        P be an s-t path in G_f;       augmenting-path
        c' ← min. capacity in P;
        For (x,y) ∈ P
            if ((x,y) is forward)
                f(x,y) ← f(x,y) + c';
            else f(x,y) ← f(x,y) - c';
    }
    return f;
}
```

- Qn: • Does it ever stop?
  • Does it output max. flow?

- To analyze, we need a new concept –
  - For $A \subset V$ s.t. $s \in A$, $t \notin A$, consider the edges that go from A to $\bar{A}$,

    $\underline{\mathrm{cut}(A)} := E \cap A \times \bar{A}$.

  - Capacity of a cut, $\underline{c(A)} := \sum_{\substack{(u,v) \in \\ \mathrm{cut}(A)}} c(u,v)$.

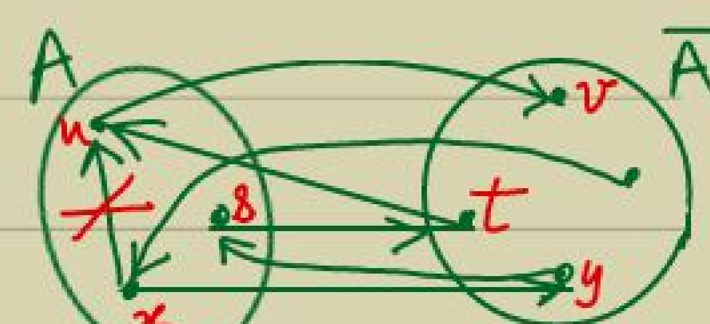
- Note that, in some sense: $s \in A$ means that A is a "source" & $t \in \bar{A}$ means that $\bar{A}$ is a "sink".

- For a flow f we can define the flow amounts leaving & entering A:

$$f_{out}(A) := \sum_{(x,y) \in \mathrm{cut}(A)} f(x,y) .$$

$$f_{in}(A) := \sum_{(x,y) \in \mathrm{cut}(\bar{A})} f(x,y) = \sum_{(x,y) \in \mathrm{cut}(A)} f(y,x) .$$

G:



Lemma 1: $f_{out}(A) - f_{in}(A) = \text{value}(f)$.

Pf:

- $\text{value}(f) = f_{out}(s) - f_{in}(s)$

$$= \sum_{a \in A} \left(f_{out}(a) - f_{in}(a) \right) \quad [\because \text{conservation}]$$

$$= \sum_{a \in A} \left(\sum_{(a,y) \in E} f(a,y) - \sum_{(x,a) \in E} f(x,a) \right)$$

($\because$ flow within A cancels)

$$= \sum_{a \in A} \sum_{(a,y) \in cut(A)} f(a,y) - \sum_{a \in A} \sum_{(x,a) \in cut(\bar{A})} f(x,a)$$

$$= f_{out}(A) - f_{in}(A). \qquad \square$$

Lemma 2: $f_{out}(A) - f_{in}(A) \leq c(A)$.

Pf:

- $\text{LHS} \leq f_{out}(A) = \sum_{(u,v) \in cut(A)} f(u,v)$

$$\leq \sum c(u,v) = c(A). \qquad \square$$

- What about the min s-t cut vs. max flow?

Max-flow Min-cut Theorem

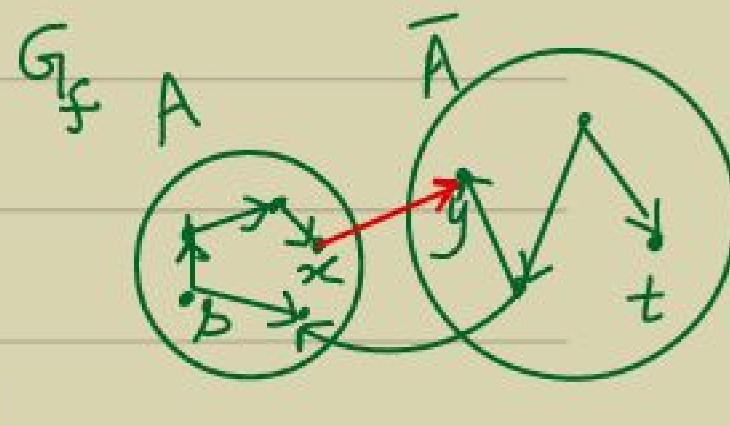
- Upon termination of Ford-Fulkerson s & t get disconnected in the current graph G_f, where f is the final flow.

- Let A be the set of vertices reachable from s. Let $\bar{A}$ be the rest (includes t).

Theorem: In G, value$(f) = c(A)$. Thus, max. flow equals min. cut capacity in any graph.

Proof:

- We know that value$(f) \leq c(A)$.
- Suppose at the end of Ford-Fulkerson value$(f) < c(A)$.



- $\Rightarrow \exists (x,y) \in \text{cut}(A)$ in G_f s.t. $c_f(x,y) > 0$.
- This contradicts $y \notin A$.

$\Rightarrow$ value$(f) = c(A)$.

- For any s-t cut B, $c(A) = \text{value}(f) \leq c(B)$.

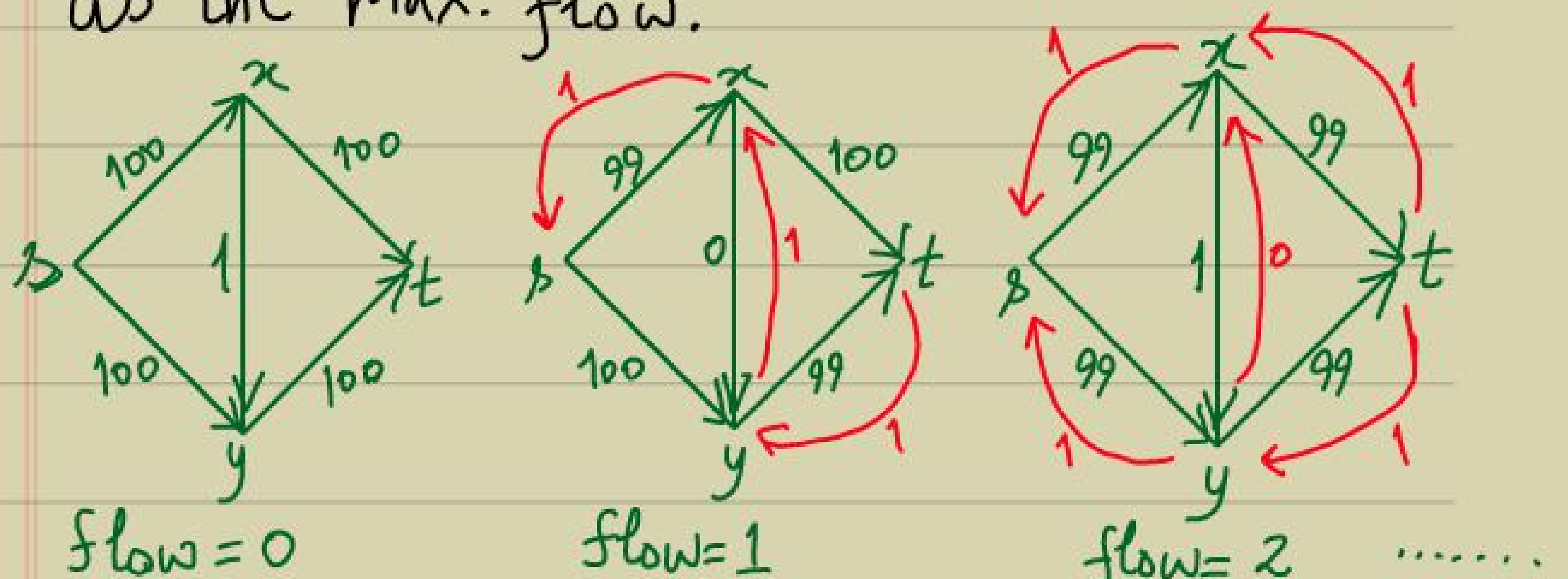
$\Rightarrow$ A is the min. s-t-cut in G.

$\Rightarrow$ f is the max. flow & it equals the capacity of min. s-t-cut. $\square$

$\Rightarrow$ ▷ Ford-Fulkerson algorithm is correct.

▷ For integral weights, the max. flow is integral!

- The number of steps might be as many as the max. flow.



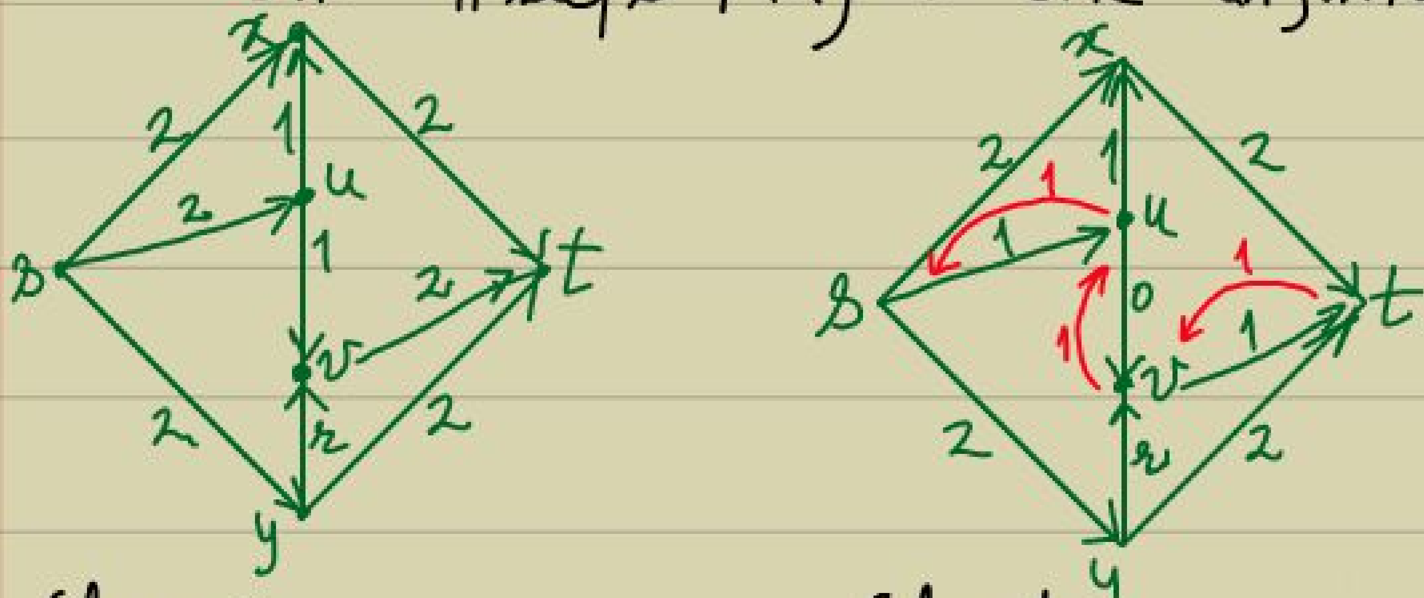
$\Rightarrow$ The number of steps above is 200!

Theorem (Ford, Fulkerson, 1956): If G with integral c has max. flow f then the algorithm takes time $O(|E| \cdot f)$.

- Technically, it is an exponential-time algorithm as capacities are given in binary.

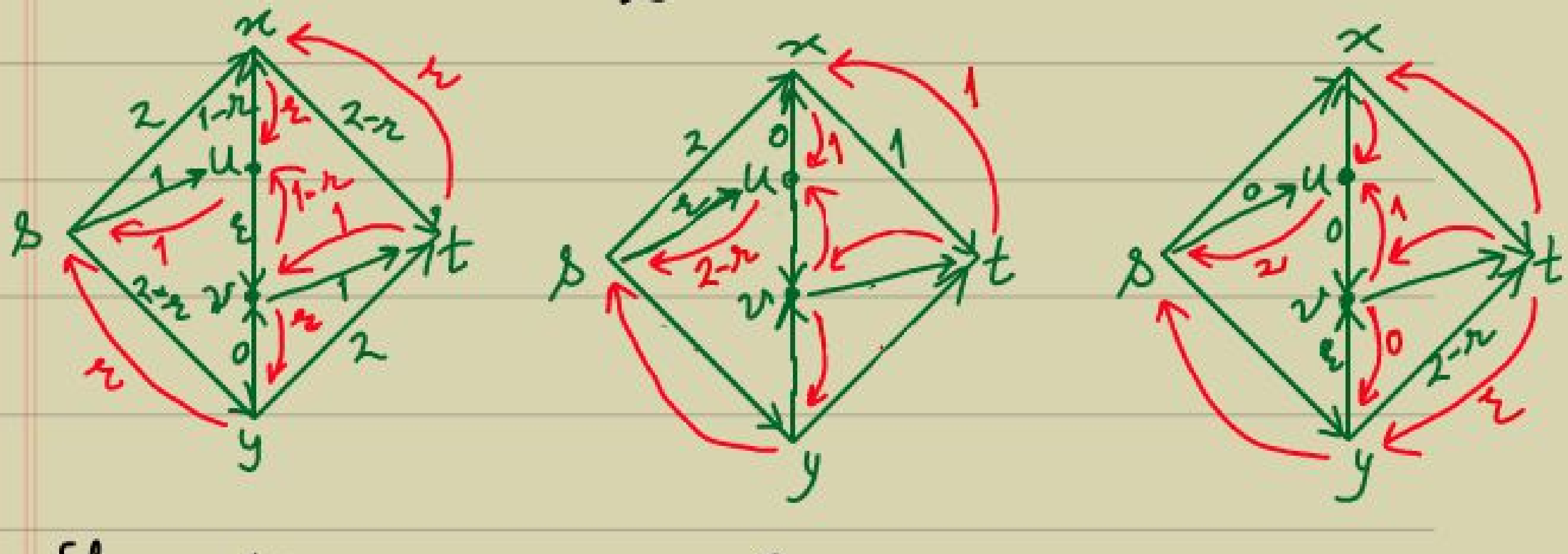
- What happens if the capacities were <u>not</u> integers?

The #steps may become infinite!



flow = 0 flow = 1

- where $r = \frac{\sqrt{5}-1}{2} \approx 0.618$



flow = $1+r$ 2 $2+r$ ......

$\Rightarrow$ <u>Claim</u>: Ford-Fulkerson may not terminate on even 6 vertices if the weights are not positive rationals.

- For positive integer weights can we do better than exp. time?

    Select the paths cleverly?

Idea 1: Select a path with max. capacity. This raises the hope of faster convergence to max. flow.

- The carefully chosen path (to augment the flow) is called an augmenting path.

```
Algo1 (G=(V,E,c), s,t) {
    f←0; k← max.capacity(E);
    while (k≥1) {
      while (∃ s-t path P in G_f with cap.≥k) {
         c'← capacity of P;
         for (x,y) ∈ P
            if (x,y) is a forward-edge
               f(x,y) ← f(x,y) + c';
            else f(y,x) ← f(y,x) − c'; }
    k←k/2; } }
```

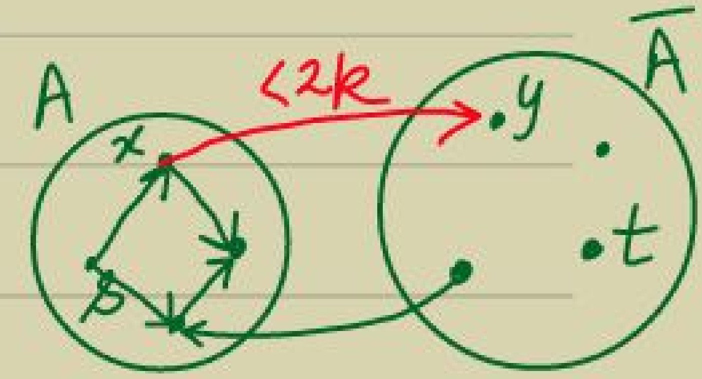
P can be found by a greedy approach in $O(m)$ time.

▷ The first while-loop runs $O(\lg c_{max})$ times, where c_{max} is the max. capacity on E.

Lemma: The second while-loop runs for $O(m)$ times.

Pf:

- Note that there is a path P of cap. $\geq k$, but not $\geq 2k$, in G_f.
- Let A be the set of vertices in G_f that are reachable from s by a path of cap. $\geq 2k$.



$\Rightarrow$ each cut-edge (A to $\bar{A}$) has cap. $< 2k$.

$\Rightarrow$ value$(f) > f_{max} - m \times 2k$.

- Note that each iteration of the inner while-loop increases the flow by $> k$.

$\Rightarrow$ # iterations $< 2m$. □

Theorem: For an integral network G, Algo1 finds max. flow in $m \times \lg c_{max} \times O(m)$ time.

- We can give a different analysis/algo., that does not use the capacities.

Idea 2: Let $\delta_f(s,v)$ be the shortest-distance in G_f considering only the # hops & not the edge-capacities from s to v.
Pick a shortest-path P as an augmenting-path (to augment f to f').

- The intuition is that since in the new $G_{f'}$ an edge in P disappears (say $x \to y$), $\delta_{f'}(s,y) \geq \delta_f(s,y)$.
In a later iteration (in $G_{f''}$) if the edge (x,y) reappears then what can we deduce?

$\Rightarrow$ Backedge (y,x) is there in the picked path P.

$\triangleright$ $\delta_{f''}(s,x) = \delta_{f''}(s,y) + 1 \geq \delta_{f'}(s,y) + 1 \geq \delta_f(s,y) + 1$
$= \delta_f(s,x) + 2.$

Lemma: Whenever (x,y) reappears in a residual graph, $\delta(s,x)$ increases by ≥ 2.

Thus, (x,y) can disappear/reappear $\leq \frac{n-1}{2}$ times.

$\Rightarrow$ $\triangleright$ #augmentations $\leq \frac{m(n-1)}{2}$.

Theorem (Edmonds, Karp 1972): Max. flow in a real weighted graph is computable in $O(m^2 n)$ time.

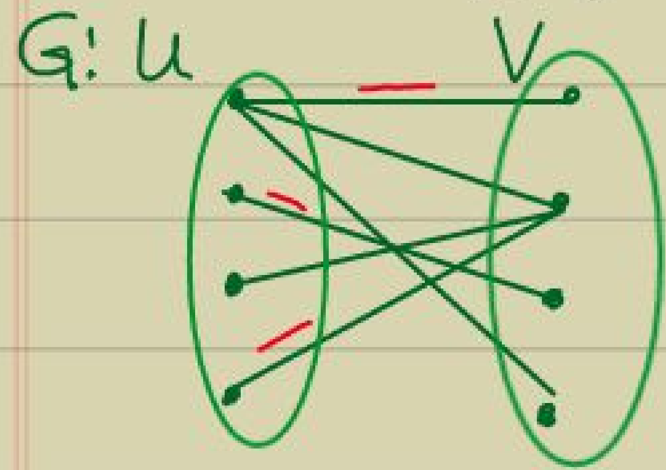
- Later improvements:
  [Malhotra, Kumar, Maheshwari '78] $O(n^3)$.
  [Orlin 2013] $O(mn)$.

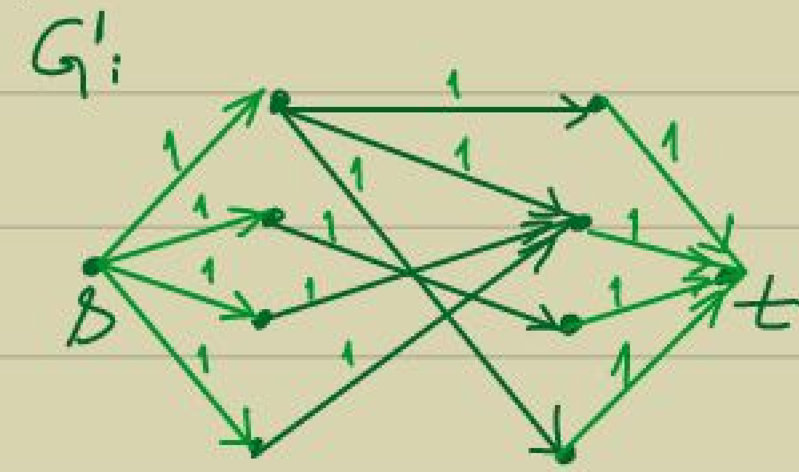
Exercise: Show that after every augmentation, $\forall v \in V$, $\delta_f(s,v) \leq \delta_{f'}(s,v)$.

Variants: 1) Multiple sources & sinks, or
2) Nodes have capacities, or
3) Flow with lower bound.

Application 1: Bipartite Matching

- Problem: Given a bipartite graph $G = (U, V, E)$ find a largest set of non-overlapping edges $M \subseteq E$.





- G has a max. matching of size = 3.
  G' has max. flow = 3.

Theorem: G has a size-k matching iff the related network G' has flow = k.

Proof:
- [$\Rightarrow$] is the easy direction.
- [$\Leftarrow$] Exercise (or see the next appln.).

$\square$

Exercise: It can be computed in $O((m+n)n)$-time.

Application 2: Edge disjoint s-t paths

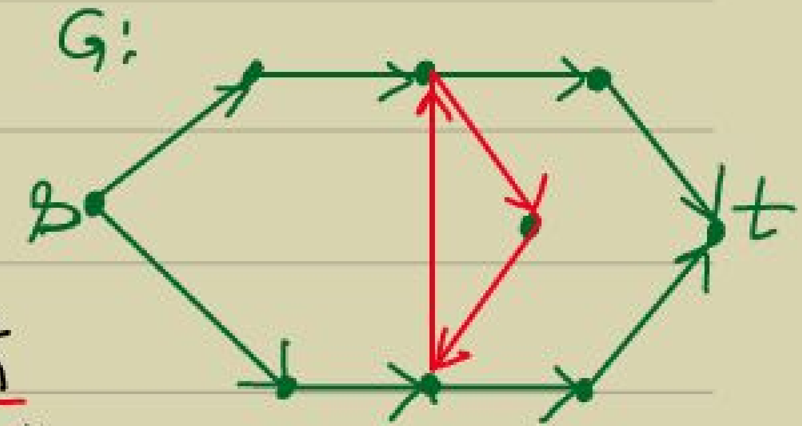
- Let $G=(V,E)$ be the graph in which we are interested in edge disjoint st paths.

  Let G' be the related flow network with edge-weights 0 or 1.

Theorem: G' has an st flow $=k$ iff G has k edge-disjoint s-t paths.

Proof:

- [$\Leftarrow$] This is clear.
- [$\Rightarrow$]



Note that in Ford–Fulkerson any augmenting-path P has capacity $=1$ (in G'_f).

- No edge of P can survive in $G'_{f'}$. Thus, the next augmenting-path is disjoint.

$\Rightarrow$ We get k edge-disjoint s-t paths. $\square$