Introduction to Algorithms

Lecture 12

Last Time

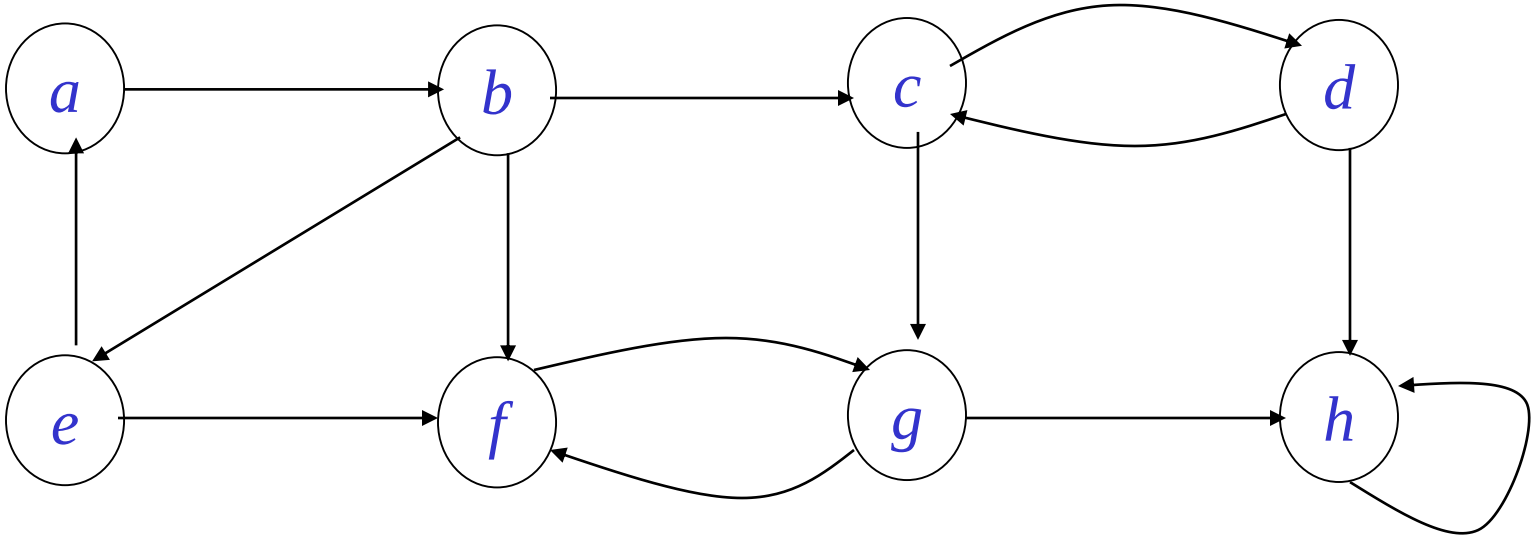
- Shortest paths
- Dijkstra's algorithm
- Breadth-First-Search
- Depth-First-Search

Today's Topics

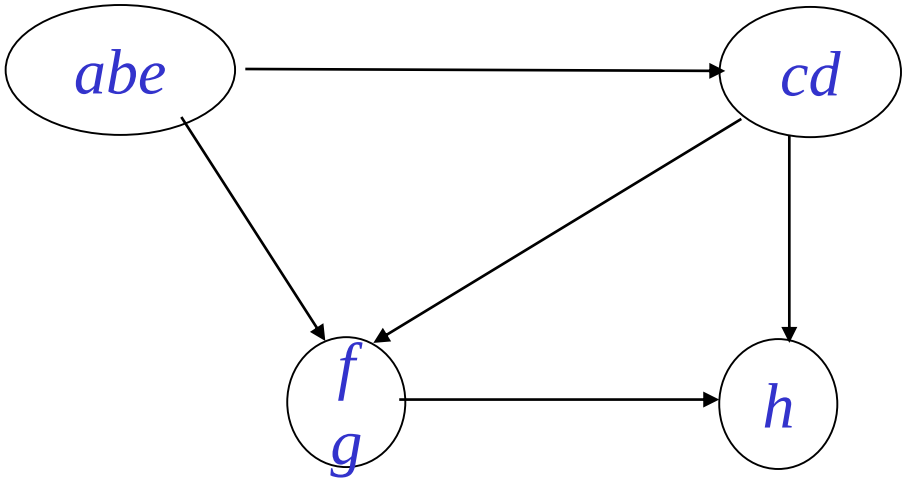
- Connected Components
- Flow networks
- Residual networks
- Augmenting paths
- Max-flow Min-cut theorem
- Ford-Fulkerson algorithm
- Edmond-Karp algorithm

Strongly Connected Component

Strongly Connected Component of digraph $G = (V, E)$ is maximal set of vertices $C \subseteq V$ such that all pairs of vertices of C are reachable from each other.



Strongly Connected Component



In analysis of certain types of engineering systems, want to examine state space and find whether appropriate states of system are reachable from one another and others appropriately unreachable.

Strongly Connected Component

Note: G^T is transpose of $G = (V, E)$. $G^T = (V, E^T)$, so all edges have direction reversed. Can compute in $\Theta(V + E)$ from adjacency list, G and G^T have some strongly connected components.

Strongly Connected Component

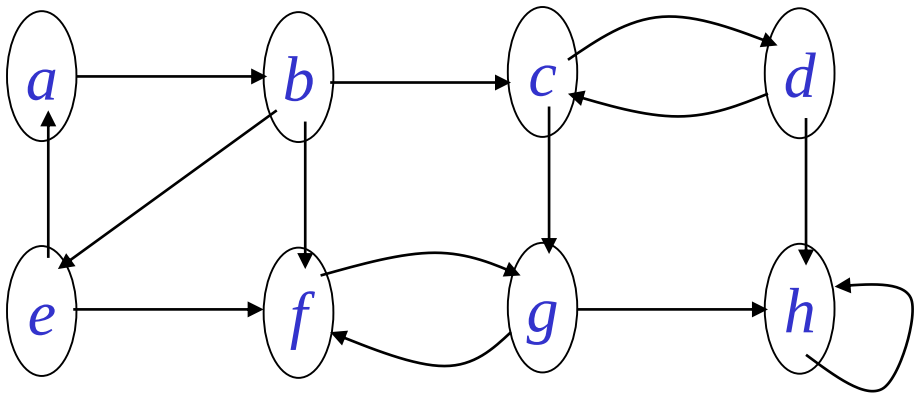
Strongly-Connected-Components(G)

1. Call DFS(G) $\Rightarrow$ $f[u]$ — $\Theta(V + E)$
2. Compute G^T — $\Theta(V + E)$
3. Call DFS(G^T), but in main loop consider vertices in decreasing $f[u]$ order. — $\Theta(V + E)$
4. Output vertices of each tree in depth-first forest as s.c.c. — $\Theta(V)$

Total: $\Theta(V + E)$

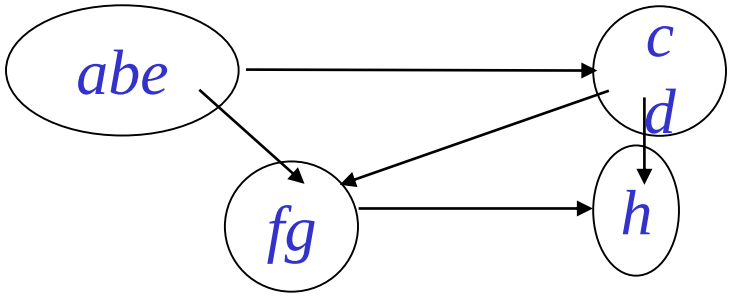
Key Idea

Consider G^{SCC}: Strongly connected components of G represented as single vertex; edges in G^{SCC} correspond to edges in G.



G

$\Rightarrow$



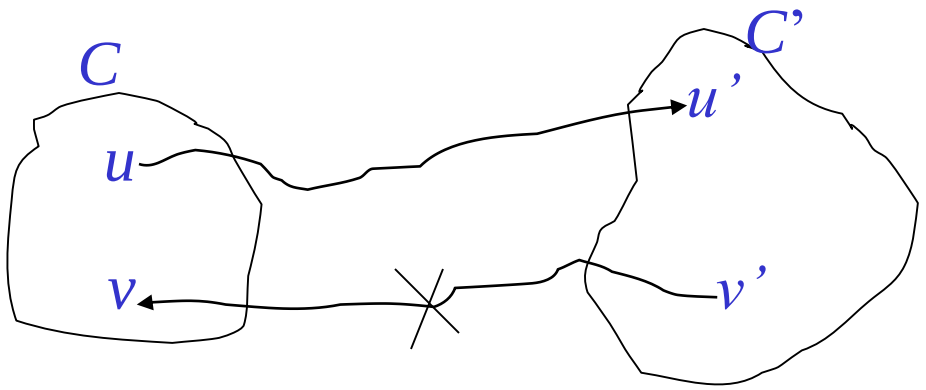
G^{SCC}

Key Idea

Lemma: C, C' distinct S.C.C.'s with $u, v \in C$, $u', v' \in C'$, and path $u \sim\sim\sim> u'$ exists in G. Then $v' \sim\sim\sim> v$ does not exist in G.

Proof: $v' \sim\sim\sim> v$ existing $\Rightarrow$ $u \sim\sim\sim> u' \sim\sim\sim> v'$ and $v' \sim\sim\sim> v \sim\sim\sim> u$.

Thus, u and v reachable from each other. Contradiction.



Analysis

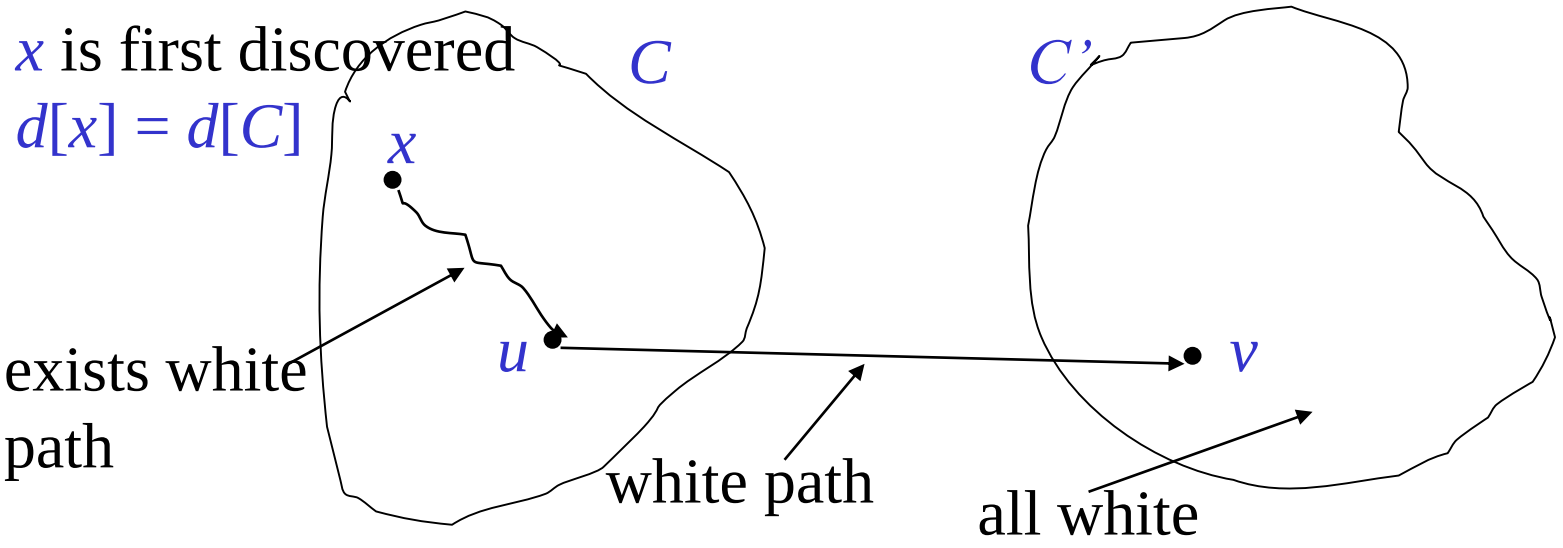
Notation: $d(U) = \min_{u \in U}\{d[u]\}$

$f(U) = \max_{u \in U}\{f[u]\}$

Lemma: Let C & C' be distinct S.C.C's of digraph $G = (V, E)$. Let edge $(u, v) \in E$ with $u \in C$ and $v \in C$'. Then $f(C) > f(C')$ in G.

Analysis

Case 1: $d(C) < d(C')$

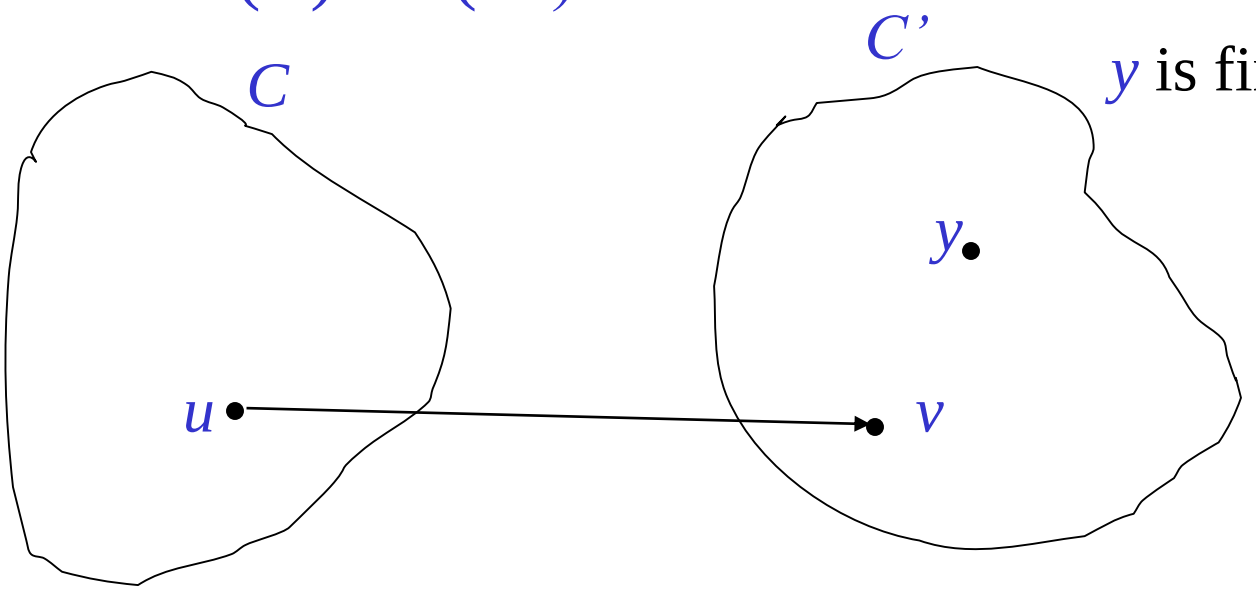


By white path theorem, all vertices in C' are descendant of x and thus have earlier finishing time

$$f(C) = f[x] > f(C')$$

Analysis

Case 2: $d(C) > d(C')$



No vertex in C reachable from C' (previous lemma), so at $f[y]$, C is all white. Thus $f(C) > f(C')$

So, whether we begin C or C' first, C' will always finish first!

Analysis

Corollary: Let C & C' be distinct S.C.C's of digraph $G = (V, E)$. Let edge $(u, v) \in E^T$ with $u \in C$ and $v \in C$'. Then $f(C) < f(C')$ in G.

Strongly-Connected-Component

So, here's how Strongly-Connected-Component works:

- DFS of G^T starts with $f(C)$ with maximum finishing time and explores this SCC completely.
- By previous corollary, there are no edges leading out of this SCC in G^T.
- When we finish, we start new tree with an element of next S.C.C (C), whose only edges out (if any) return to C', which has already been finished.

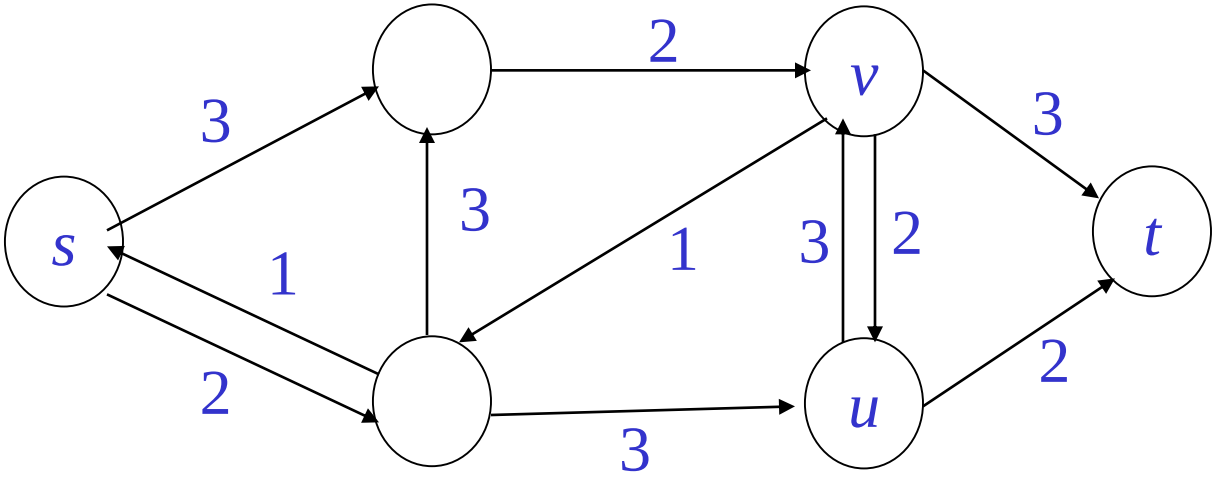
Flow networks

Model fundamental notion of material being transported across finite-capacity channels

Examples: water, current, data, etc.

Single source/sink, multi-source/sink, “multi-commodity”

Flow networks



Model as network (digraph $G = (V, E)$) with:

"source" node s and "sink" node t

Non-negative capacities $c(u, v)$

If $(u, v) \notin E$, then $c(u, v) = 0$.

Flow

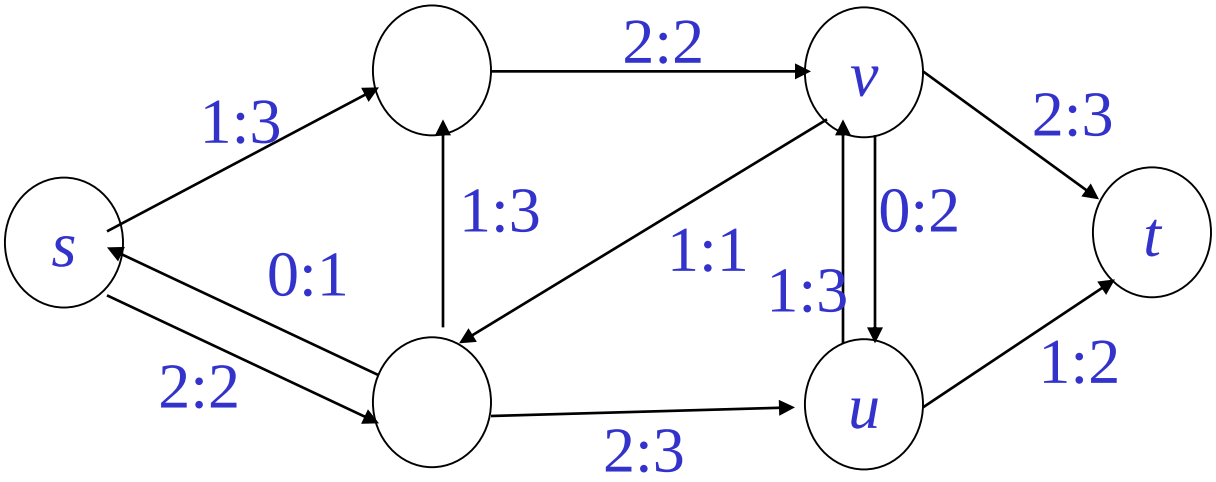
Flow on G is a function f: $V \times V \rightarrow R$ satisfying

- Capacity constraint: $f(u,v) \leq c(u,v)\ \forall u,v \in V$.
- Flow conservation: $\sum_{v \in V} f(u,v) = 0 \quad \forall u \in V - \{s,t\}$
- Skew symmetry: $f(u,v) = -f(v,u)$.

Flow value, $|f| = \sum_{v \in V} f(s,v) = f(s,V)$ is total flow out of source.

Goal: Finding Maximum Flow Value.

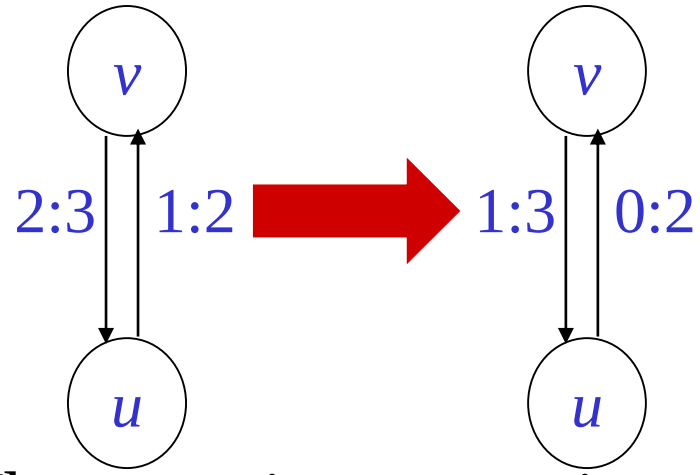
Example



$|f| = 1 + 2 = 3$

Flow Cancellation

- Without loss of generality, positive flow goes either from u to v, or from v to u, but not both.



Net flow from u to v in both case is 1.

- The capacity constraint and flow conservation are preserved by this transformation.
- **INTUITION:** View flow as a *rate*, not a *quantity*.

Residual Network

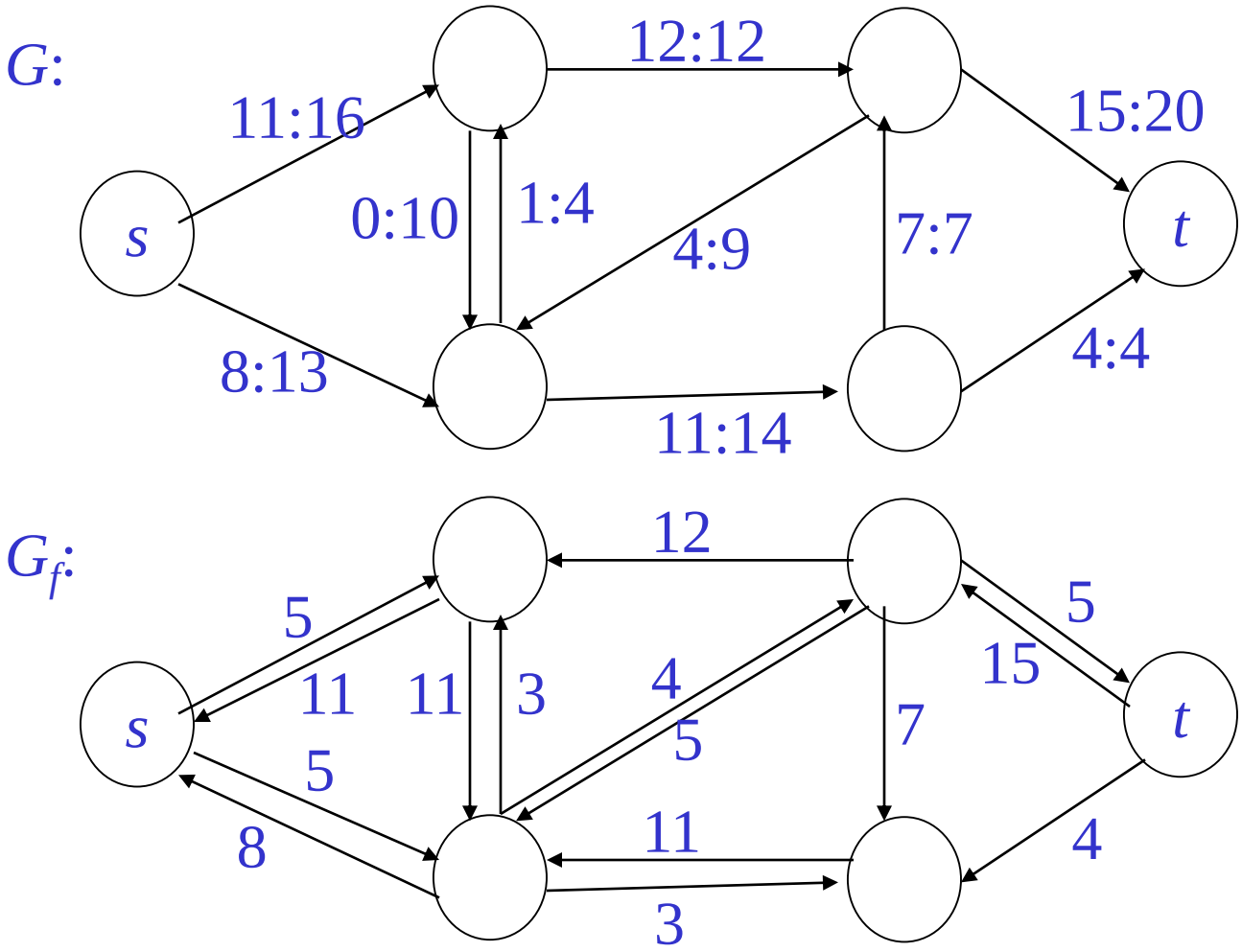
Residual Capacity of (u, v):

$$c_f(u,v) = c(u,v) - f(u,v) \geq 0$$

Residual Network G_f is graph of edges with strictly positive residual capacity (that can support more flow)

Its edges come from $G \cup G^T$, then $|E_f| \leq 2|E|$.

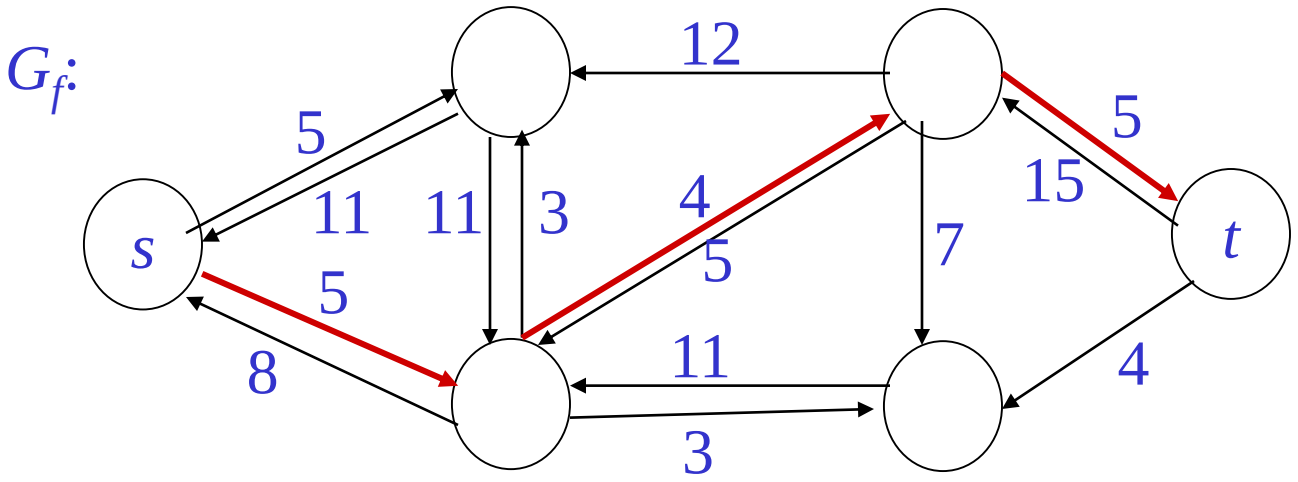
Residual Network



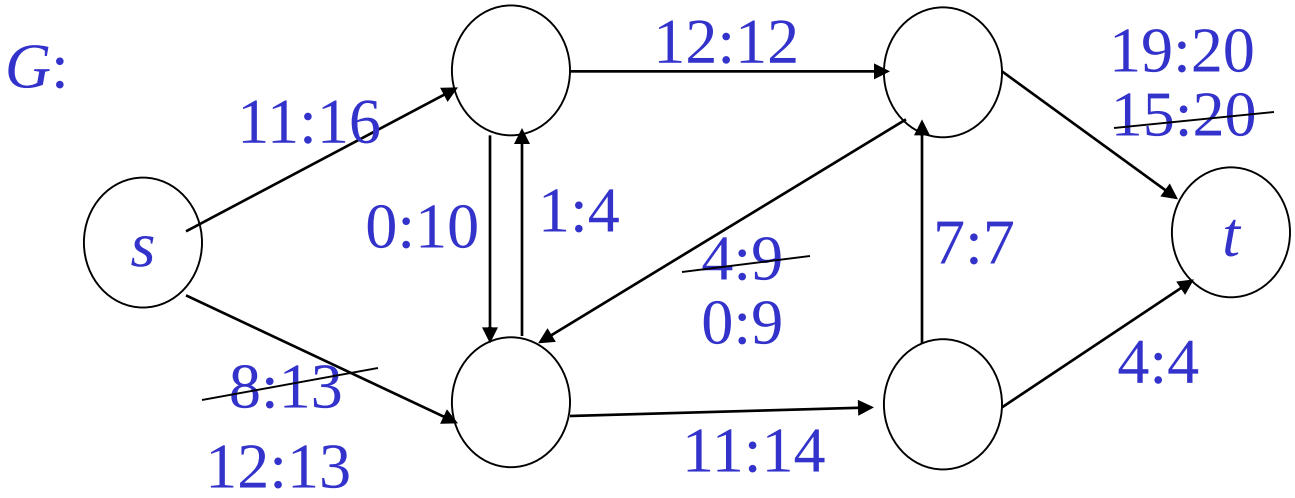
Augmenting Path

An augmenting path is a path p from s to t in G_f (or "in G with respect to f")

Increase flow in G by $c_f(p) = \min_{(u,v)\in p} c_f(u,v)$



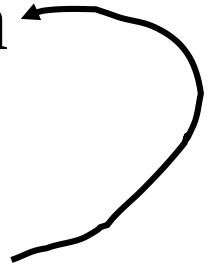
Augmenting Path



Augmenting Path

Suggests an algorithm (method):

1. Start with Zero Flow.
2. Find an augmenting path p in corresponding G_f.
3. Augment f by $c_f(p)$ and loop.



(Ford-Fulkerson Algorithm)

Potential Problems

- Convergence? Can it enter infinite loop? No, because we only accept augmenting paths.
- Can we be guaranteed of finding an augmenting path if one exists? BFS finds all shortest paths from a source. If none are found to sink, then none exist.
  - Do we want a shortest path?
  - Do we want a longest path?
  - We really want a path of maximum capacity.
- Global optimum? When no more augmenting paths can be found, how do we know we haven't found a local optimum rather than the global optimum?

Properties of Flows

Lemma

For $G=(V, E)$ a flow network and f a flow in G.

1) $f(X, X) = 0$ for all $X \subseteq V$.
2) $f(X, Y) = -f(Y, X)$ for all $X, Y \subseteq V$.
3) $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ for all $X, Y, Z \subseteq V$ and $X \cap Y = \Phi$.

where $f(X, Y) = \Sigma_{x \in X, y \in Y} f(x, y)$.

Illustrate Use

Flow is defined as net flow leaving source; show equal to that entering sink.

$$|f| \equiv f(s, V) = f(V, t).$$

Proof.

$$\begin{aligned} |f| &\equiv f(s, V) \\ &= f(V, V) - f(V - \{s\}, V) \\ &= 0 + f(V, V - \{s\}) \\ &= f(V, t) + f(V, V - \{s,t\}) = f(V, t) \end{aligned}$$

Illustrate Use

Slightly more intuitive proof:

$$
\begin{aligned}
|f| &\equiv f(s, V) + f(V, t) - f(V, t) \\
&= f(V, t) - (f(V, s) + f(V, t)) \\
&= f(V, t) - f(V, \{s,t\}) + f(V, V) \\
&= f(V, t) + f(V, V - \{s,t\}) \\
&= f(V, t)
\end{aligned}
$$

Cut

Cut is a partition of V into S and $T = V - S$ with $s \in S$ and $t \in T$.

Capacity of cut $c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$

Forward direction only

- Small capacity cuts serve as "barrier" to large flows.
- A minimum cut of a network is that whose capacity is a minimum over all cuts in network.

Cut

Lemma: Given any flow f, $|f| = f(S, T)$ for and any (S, T).

Proof: $f(S, T) = f(S, V) - f(S, S)$

$= f(s, V) + f(S - s, V)$

$= f(s, V) = |f|$

- This is like another conservation of flow idea
  - by definition for flow out of source ($S=\{s\}$)
  - already proven for flow into sink ($T=\{t\}$)
  - have proven for any partition (cut) dividing s, t

Cut

Corollary: The value of any flow is bounded above by the capacity of any cut.

$f(S, T) \leq c(S, T)$ for all cuts (S, T)

Proof:

$$
\begin{aligned}
|f| &= f(S,T) \\
&= \sum_{u \in S} \sum_{v \in T} f(u,v) \\
&\leq \sum_{u \in S} \sum_{v \in T} c(u,v) \\
&= c(S,T)
\end{aligned}
$$

The Max-Flow Min-Cut Theorems

The following are equivalent:

1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T) of G.

Proof sketch

1⇒2 Augmenting path would allow flow in G to be increased above f, contradicting its being a maximum flow .

The Max-Flow Min-Cut Theorems

$3 \Rightarrow 1$ By corollary, $|f|$ is bounded from above by the capacity of any cut, so $|f| = c(S,T)$ means it can't be increased.

$2 \Rightarrow 3$ G_f has no augmenting path. Let $S = \{u \in V$: $\exists$path from s to u in $G_f\}$ and $T = V - S$.

$s \in S$ and $t \in T$ (otherwise would be augmenting path).

(S, T) is a cut.

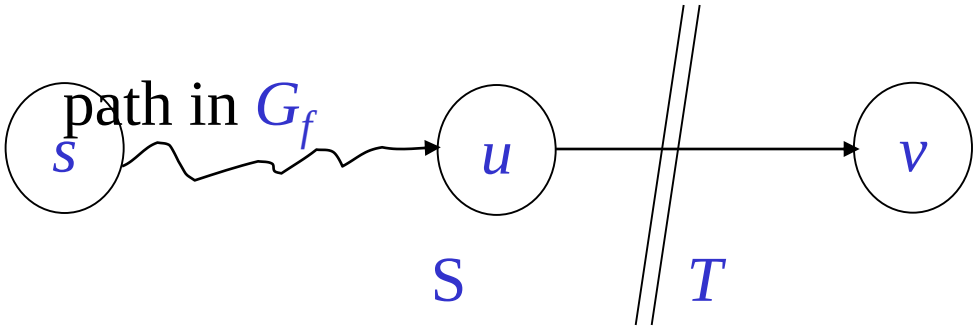
$c_f(u, v)=0$ for $u \in S$, $v \in T$ (otherwise v would be in S).

The Max-Flow Min-Cut Theorems

$\therefore$ $f(u,v)=c(u,v)$, since $c_f(u, v) = c(u,v) - f(u,v)$.

summation

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



Ford-Fulkerson Algorithm

$f(u, v) \leftarrow 0 \ \forall u, v$

while $\exists$ augmenting path p in G_f

do augment f by $c_f(p)$

- Thus, the Ford-Fulkerson algorithm is guaranteed to result in successive improvements. When no more improvements are possible, the maximum flow has been found.

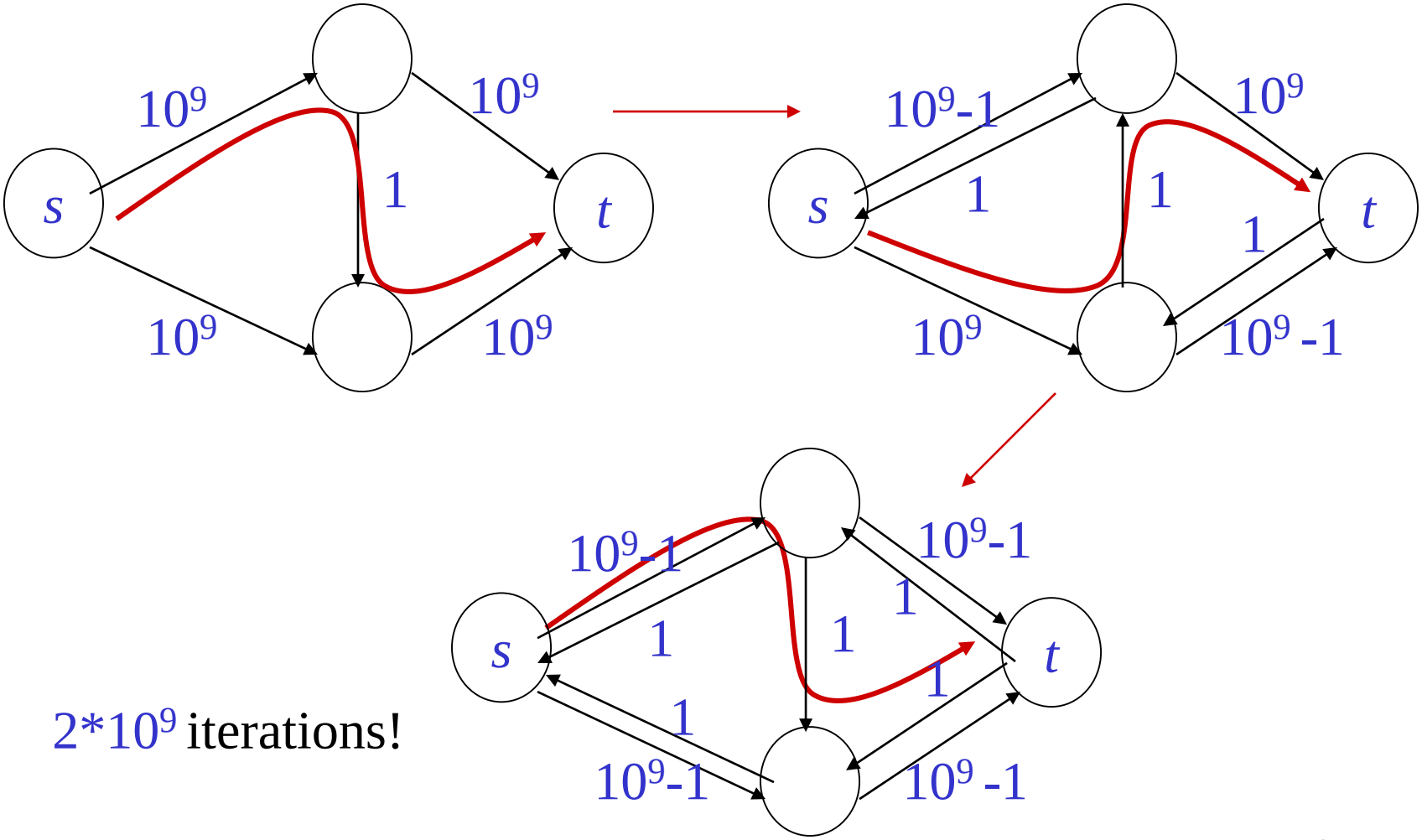
Ford-Fulkerson Algorithm

Notice: F-F doesn't specify *which path*!

Analysis (with Integer Capacities):

- Cost per path: $O(E)$ using DFS or BFS
- Thus run time is $O(E\,|f^*|)$, where f^* is max flow
- This is not polynomial in $|V|$, $|E|$!

Possible Efficiency Problem



$2*10^9$ iterations!

Edmonds-Karp Algorithm

- Use BFS to find breadth-first augmenting path
- Time to find an augmenting path $\Theta(V + E)$ $\Rightarrow \Theta(E)$ if all vertices reachable from source.
- Claim: number of augmentations limited to $\Theta(VE)$

  $\Rightarrow$ Running time $\Theta(VE^2)$.

Edmonds-Karp Analysis

Monotonicity Lemma

Let $\delta(v) = \delta_f(s, v)$

= B-F distance from s to v in G_f.

Then $\delta(v)$ increases monotonically during E-K.

Suppose algorithm augments f to produce f'.

Prove by $\delta'(v) \geq \delta(v)$ induction on $\delta(v)$.

Monotonicity Lemma

Base: $\delta'(s) = \delta(s) = 0$

Inductive Step: Consider BF-path $s \to \ldots u \to v$ in G_f

$\delta'(v) = \delta'(u) + 1$

$(u, v) \in E_f$ [What about E_f?]

Case 1: $(u, v) \in E_f$

Then $\delta(v) \leq \delta(u) + 1$ [triangle ineq.]

$\leq \delta'(u) + 1$ [induction]

$= \delta'(v)$ [BF path]

Monotonicity Lemma

Case 2: $(u, v) \notin E_f$

Augmenting path must have included (v,u)

$$s \rightarrow \ldots v \rightarrow u \rightarrow \ldots \rightarrow t$$

Then $\delta(v) = \delta(u) - 1$ [BF path]

$\leq \delta'(u) - 1$ [induction]

$= \delta'(v) - 2$ [BF path]

$< \delta'(v)$

Edmonds-Karp Analysis

Theorem: #Augmenting Steps in E-K algorithm is $\Theta(VE)$.

Proof: Let p be an augmenting path.

Residual capacity of some edge $(u,v) \in p$ is $c_f(p) = c_f(u, v) \Rightarrow (u,v)$ called critical

Augmentation removes (u,v) from critical path (for now)

Must wait until (v,u) augmented before re-appears.

Edmonds-Karp Analysis

Before p augmented (u,v): $\delta(v)=\delta(u)+1$ (p: BF path)

Before augmenting (v,u) later:

$$\delta'(u) = \delta'(v) + 1$$
$$\geq \delta(v) + 1$$
$$\geq \delta(u) + 2$$

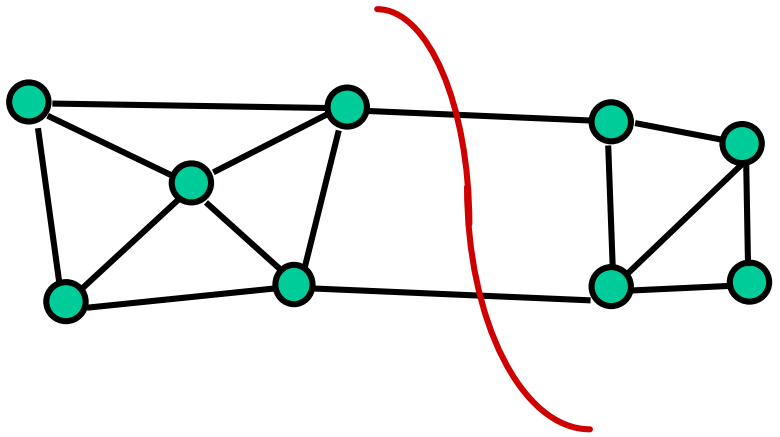
$\Rightarrow$ $\delta(u)$ increases ≥ 2 between twice when (u,v) is critical. Each $\delta(u)$ starts ≥ 0, never decreases, and remains $\leq |V| - 1$ until unreachable.

$\Rightarrow$$(u, v)$ is critical at most $\Theta(V)$ times

$\Theta(E)$ edges in $G_f \Rightarrow \Theta(VE)$ augmentation

Cuts in Graphs

- Focus on undirected graphs
- A cut is a vertex partition
- Value is number (or total weight) of crossing edges

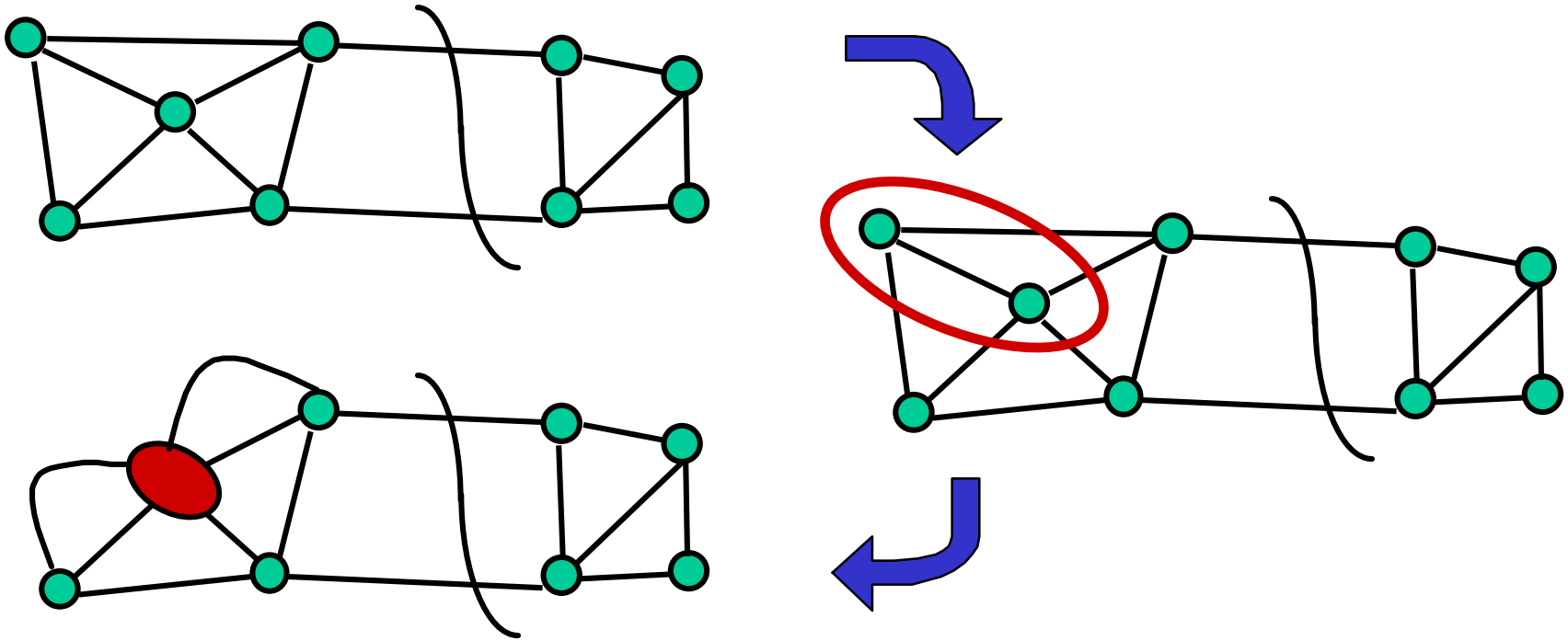


Minimum Cut

- Smallest cut of graph
- Cheapest way to separate into 2 parts
- Various applications:
  - network reliability (small cuts are weakest)
  - subtour elimination constraints for TSP
  - separation oracle for network design
- Not *s*-*t* min-cut

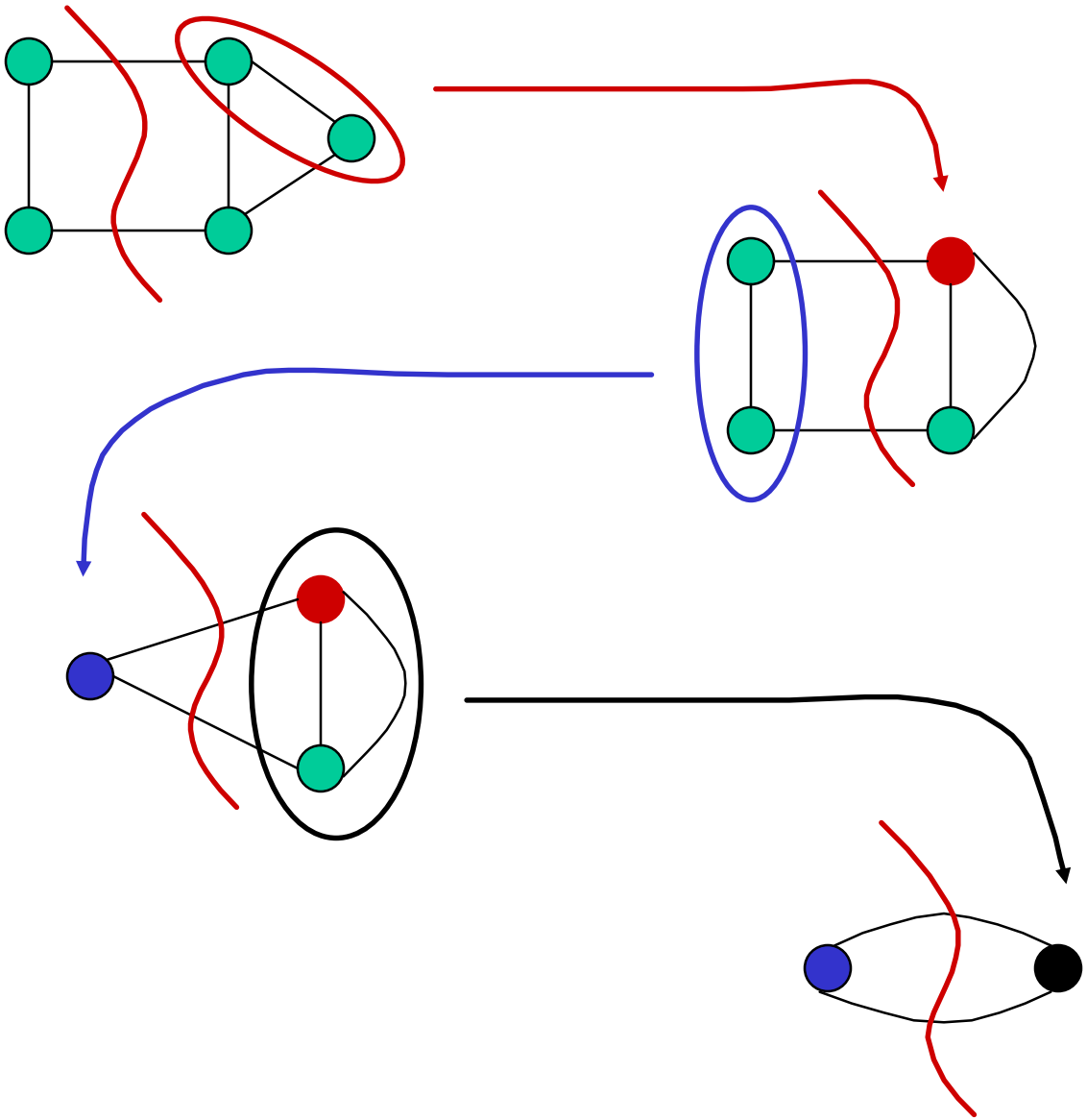
Contraction

- Find edge that doesn't cross min-cut
- Contract (merge) endpoints to 1 vertex



Contraction Algorithm

- Repeat $n - 2$ times:
  - find non-min-cut edge
  - contract it (keep parallel edges)
- Each contraction decrements #vertices
- At end, 2 vertices left
  - unique cut
  - corresponds to min-cut of starting graph



Picking an Edge

- Must contract non-min-cut edges
- [NI]: $O(m)$ time algorithm to pick edge
  - n contractions: $O(mn)$ time for min-cut
  - slightly faster than flows

If only could find edge faster….

Idea: min-cut edges are few

Randomize

Repeat until 2 vertices remain

pick a random edge

contract it

Analysis I

- Min-cut is small---few edges
  - Suppose graph has min-cut c
  - Then minimum degree at least c
  - Thus at least $nc/2$ edges
- Random edge is probably safe

$$\Pr[\text{min-cut edge}] \leq c/(nc/2)$$
$$= 2/n$$

(easy generalization to capacitated case)

Analysis II

- Algorithm succeeds if never accidentally contracts min-cut edge
- Contracts #vertices from n down to 2
- When k vertices, chance of error is $2/k$
  - thus, chance of being right is $1-2/k$
- Pr[always right] is product of probabilities of being right each time

Analysis III

$$\Pr[\text{success}] = \prod_{k=2}^{n} \Pr[k^{th} \text{ contraction safe}]$$

$$= (1-\tfrac{2}{n})(1-\tfrac{2}{n-1})\cdots(1-\tfrac{2}{3})$$

$$= (\tfrac{n-2}{n})(\tfrac{n-3}{n-1})\cdots(\tfrac{1}{3})$$

$$= \frac{2}{n(n-1)}$$

$$\approx \frac{2}{n^2}$$

…not too good!

Repetition

- Repetition amplifies success probability
  - basic failure probability $1 - 2/n^2$
  - so repeat $7n^2$ times
  - output the minimum size cut found in all the iterations

$$\Pr[\text{complete failure}] = \Pr[\text{fail } 7n^2 \text{ times}]$$

$$= (\Pr[\text{fail once}])^{7n^2}$$

$$= \left(1 - \frac{2}{n^2}\right)^{7n^2}$$

$$\leq 10^{-6}$$

How fast?

- Easy to perform 1 trial in $O(m)$ time
  - just use array of edges, no data structures
- But need n^2 trials: $O(mn^2)$ time
- Simpler than flows, but slower

An improvement [KS]

- When k vertices, error probability $2/k$
  - big when k small
- Idea: once k small, change algorithm
  - algorithm needs to be safer
  - but can afford to be slower
- Amplify by repetition!
  - Repeat base algorithm many times

Recursive Algorithm

Algorithm RCA (G, n)

{G has n vertices}

repeat twice

randomly contract G to $n/2^{½}$ vertices

RCA(G,$n/2^{1/2}$)

(50-50 chance of avoiding min-cut)

Main Theorem

- On any capacitated, undirected graph, Algorithm RCA
  - runs in $O^*(n^2)$ time with simple structures
  - finds min-cut with probability $\geq 1/\log n$
- Thus, $O(\log n)$ repetitions suffice to find the minimum cut (failure probability 10^{-6}) in $O(n^2 \log^2 n)$ time.

Proof Outline

- Graph has $O(n^2)$ (capacitated) edges
- So $O(n^2)$ work to contract, then two subproblems of size $n/2^{½}$
  - $T(n) = 2\,T(n/2^{½}) + O(n^2) = O(n^2 \log n)$
- Algorithm fails if both iterations fail
  - Iteration succeeds if contractions and recursion succeed
  - $P(n)=1 - [1 - ½\,P(n/2^{½})]^2 = \Omega(1 / \log n)$

Failure Modes

- Monte Carlo algorithms always run fast and probably give you the right answer
- Las Vegas algorithms probably run fast and always give you the right answer
- To make a Monte Carlo algorithm Las Vegas, need a way to check answer
  - repeat till answer is right
- No fast min-cut check known (flow slow!)

Exercise

- 26.1-1, 1-5, 1-6
- 26.2-3, 2-6, 2-10, 2-13