CS 473: Algorithms

Ruta Mehta

University of Illinois, Urbana-Champaign

Spring 2021

Midterm 2 Review Session

Lecture 99
April 10, 2021

~~Some of the slides are courtesy Prof. Chekuri~~

Topics

Randomozed Algorithms

- Univesal Hashing, Fingerprinting

Max-Flow, Min-Cut

- Flow: edge and path based definitions, flow decomposition, acyclicity, integral flow
- Max-Flow $=$ Min-Cut
- Residual graphs, Ford-Fulkerson Algorithm and it's polynomial-time variants.
- Variants of max-flow and their applications

Part I

Max-flow, Min-cut

The Maximum-Flow Problem

Problem

Input A directed network $G = (V, E)$ with integer capacity $c(e)$ for each $e \in E$, and source s and sink t.

Goal Find flow of **maximum** value.

The Maximum-Flow Problem

Problem

Input A directed network $G = (V, E)$ with integer capacity $c(e)$ for each $e \in E$, and source s and sink t.

Goal Find flow of **maximum** value.

Question: Given a flow network, what is an *upper bound* on the maximum flow between source and sink?

Definition of Flow

Two ways to define flows:

1. edge based, or
2. path based.

Essentially equivalent but have different uses.

Edge based definition is more compact.

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \rightarrow \mathbb{R}^{\geq 0}$ s.t.

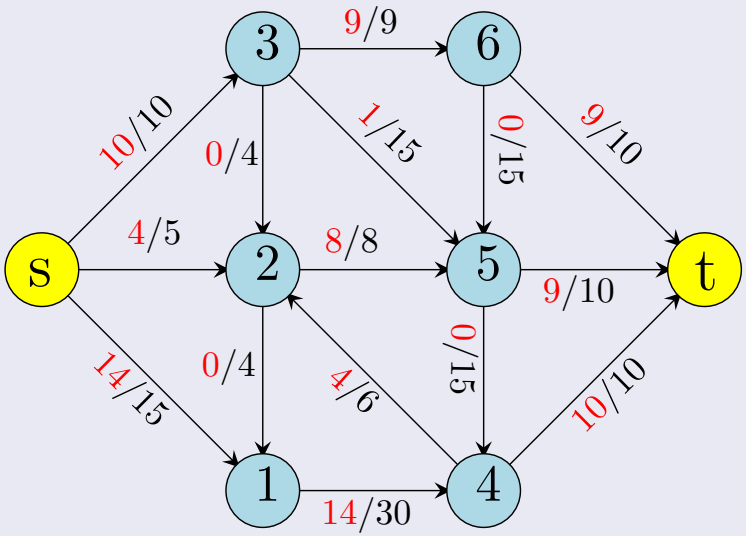


Figure: Flow with value.

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.

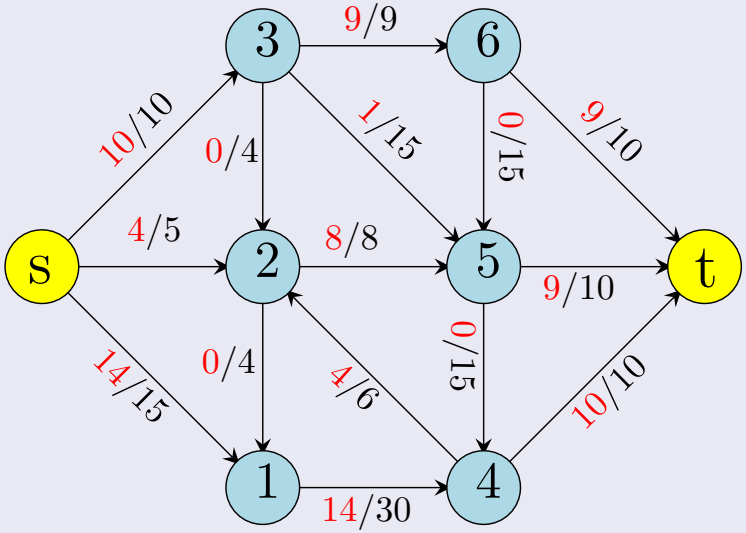


Figure: Flow with value.

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.
2. Conservation Constraint: For each vertex $v \neq s, t$.

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

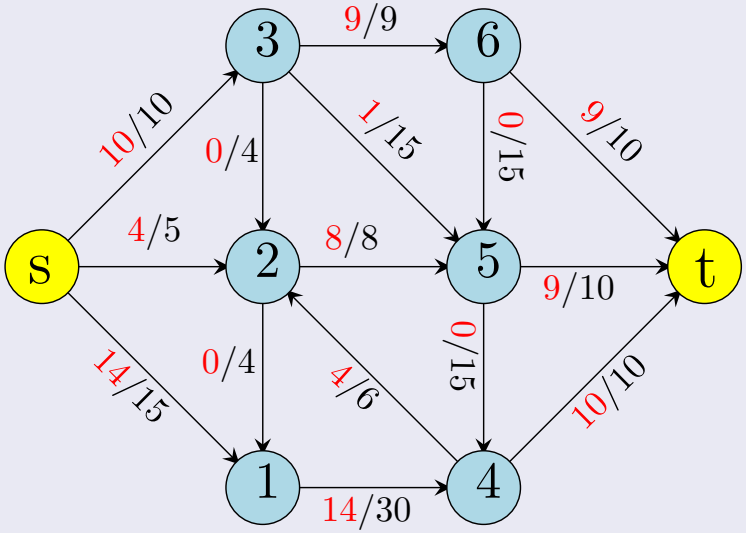


Figure: Flow with value.

Edge Based Definition of Flow

Definition

Flow in network $G = (V, E)$, is function $f : E \to \mathbb{R}^{\geq 0}$ s.t.

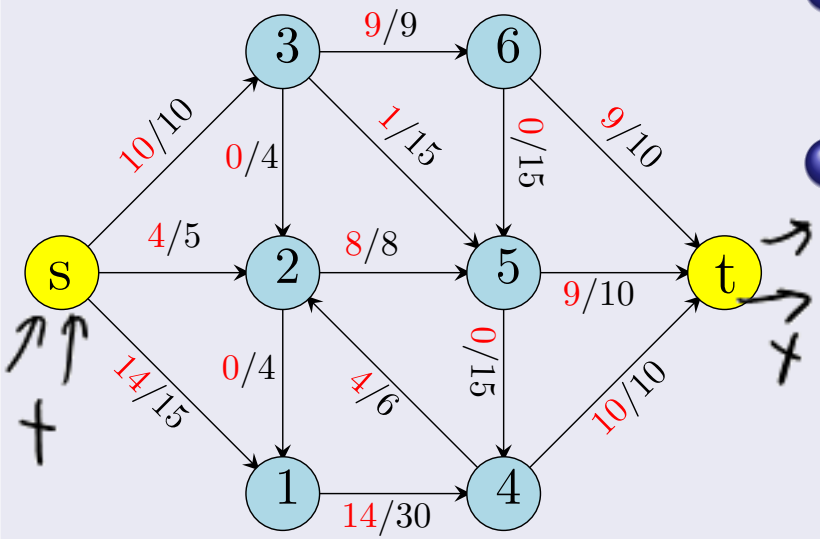


Figure: Flow with value.

1. Capacity Constraint: For each edge e, $f(e) \leq c(e)$.
2. Conservation Constraint: For each vertex $v \neq s, t$.

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

3. Value of flow= (total flow out of source) − (total flow in to source).

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all *simple* paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all *simple* paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all *simple* paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all *simple* paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P}: e \in p} f(p) \leq c(e)$$

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all *simple* paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P}: e \in p} f(p) \leq c(e)$$

2. Conservation Constraint:

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all *simple* paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P}: e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

A Path Based Definition of Flow

Intuition: Flow goes from source s to sink t along a path.

$\mathcal{P}$: set of all *simple* paths from s to t. $|\mathcal{P}|$ can be **exponential** in n.

Definition (Flow by paths.)

A **flow** in network $G = (V, E)$, is function $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ s.t.

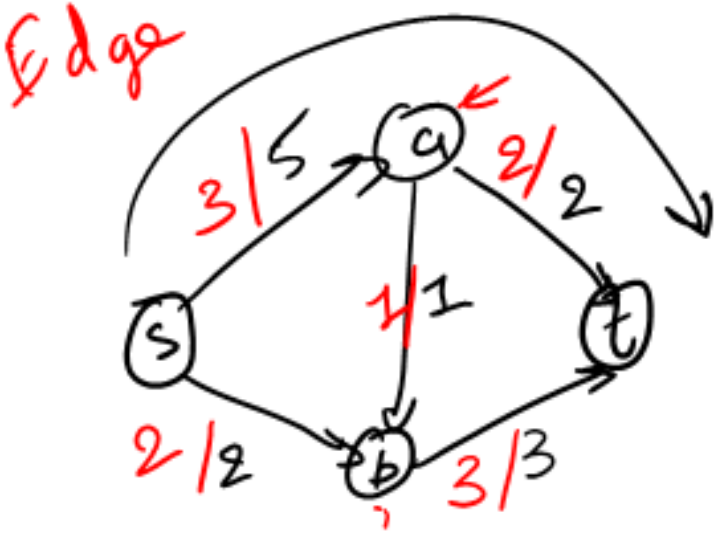
1. Capacity Constraint: For each edge e, total flow on e is $\leq c(e)$.

$$\sum_{p \in \mathcal{P}: e \in p} f(p) \leq c(e)$$

2. Conservation Constraint: No need! Automatic.

Value of flow: $\sum_{p \in \mathcal{P}} f(p)$.

Examples



Path : flow.

s-a-t : 2

s-a-b-t : 1

s-b-t : 2

Flow Decomposition

Edge based flow to Path based Flow

Lemma

Given an edge based flow $f' : E \to \mathbb{R}^{\geq 0}$, *there is a path based flow* $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ *of same value.*

Flow Decomposition

Edge based flow to Path based Flow

Lemma

Given an edge based flow $f' : E \to \mathbb{R}^{\geq 0}$, *there is a path based flow* $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ *of same value. Moreover,* f *assigns non-negative flow to at most* m *paths where* $|E| = m$ *and* $|V| = n$.

Flow Decomposition

Edge based flow to Path based Flow

Lemma

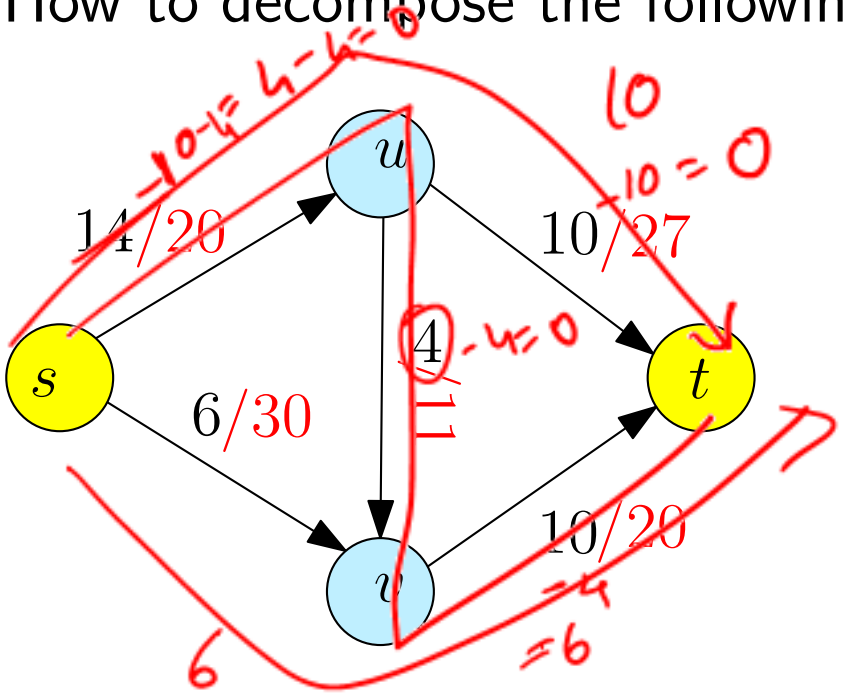
Given an edge based flow $f' : E \to \mathbb{R}^{\geq 0}$, there is a path based flow $f : \mathcal{P} \to \mathbb{R}^{\geq 0}$ of same value. Moreover, f assigns non-negative flow to at most m paths where $|E| = m$ and $|V| = n$.

Given f', the path based flow can be computed in $O(mn)$ time.

Flow Decomposition

Example

How to decompose the following flow:



s - u - t : 10

s - u - v - t : 4

s - v - t : 6

Flow Decomposition

Edge based flow to Path based Flow

Algorithm

1. Remove all edges with $f'(e) = 0$.
2. Find a path p from s to t.

Flow Decomposition

Edge based flow to Path based Flow

Algorithm

1. Remove all edges with $f'(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f'(e)$.

Flow Decomposition

Edge based flow to Path based Flow

Algorithm

1. Remove all edges with $f'(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f'(e)$.
4. Reduce $f'(e)$ for all $e \in p$ by $f(p)$.

Flow Decomposition

Edge based flow to Path based Flow

Algorithm

1. Remove all edges with $f'(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f'(e)$.
4. Reduce $f'(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.

Flow Decomposition

Edge based flow to Path based Flow

Algorithm

1. Remove all edges with $f'(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f'(e)$.
4. Reduce $f'(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.

Proof Idea.

- In each iteration at least one edge has flow reduced to zero.

Flow Decomposition

Edge based flow to Path based Flow

Algorithm

1. Remove all edges with $f'(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f'(e)$.
4. Reduce $f'(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.

Proof Idea.

- In each iteration at least one edge has flow reduced to zero.
- Hence, at most m iterations. Can be implemented in

Flow Decomposition

Edge based flow to Path based Flow

Algorithm

1. Remove all edges with $f'(e) = 0$.
2. Find a path p from s to t.
3. Assign $f(p)$ to be $\min_{e \in p} f'(e)$.
4. Reduce $f'(e)$ for all $e \in p$ by $f(p)$.
5. Repeat until no path from s to t.

Proof Idea.

- In each iteration at least one edge has flow reduced to zero.
- Hence, at most m iterations. Can be implemented in $O(m(m+n))$ time. $O(mn)$ time requires care. □

Part II

Ford-Fulkerson Algorithm

Residual Graph (The "leftover" graph)

Edge flow

Definition. For a network $G = (V, E)$ and flow f, the **residual graph** $G_f = (V', E')$ of G with respect to f is where $V' = V$ and

Residual graph

Residual Graph (The "leftover" graph)

Definition. For a network $G = (V, E)$ and flow f, the **residual graph** $G_f = (V', E')$ of G with respect to f is where $V' = V$ and

1. **Forward Edges**: For each edge $e \in E$ with $f(e) < c(e)$, we add $e \in E'$ with capacity $c(e) - f(e)$.

Residual Graph (The "leftover" graph)

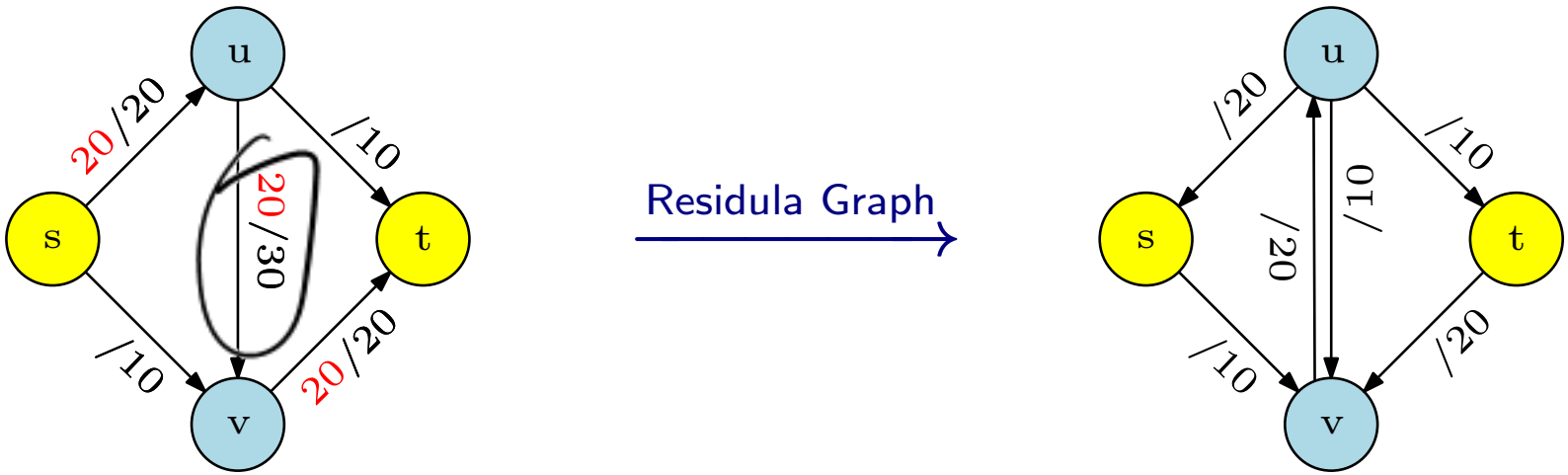
Definition. For a network $G = (V, E)$ and flow f, the **residual graph** $G_f = (V', E')$ of G with respect to f is where $V' = V$ and

1. **Forward Edges**: For each edge $e \in E$ with $f(e) < c(e)$, we add $e \in E'$ with capacity $c(e) - f(e)$.
2. **Backward Edges**: For each edge $e = (u, v) \in E$ with $f(e) > 0$, we add $(v, u) \in E'$ with capacity $f(e)$.

Residual Graph (The "leftover" graph)

Definition. For a network $G = (V, E)$ and flow f, the **residual graph** $G_f = (V', E')$ of G with respect to f is where $V' = V$ and

1. **Forward Edges**: For each edge $e \in E$ with $f(e) < c(e)$, we add $e \in E'$ with capacity $c(e) - f(e)$.
2. **Backward Edges**: For each edge $e = (u, v) \in E$ with $f(e) > 0$, we add $(v, u) \in E'$ with capacity $f(e)$.



Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let f be a flow in G and G_f be the residual graph. If f' is a flow in G_f then $f + f'$ is a flow in G of value $v(f) + v(f')$.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let f be a flow in G and G_f be the residual graph. If f' is a flow in G_f then $f + f'$ is a flow in G of value $v(f) + v(f')$.

Lemma

Let f and f' be two flows in G with $v(f') \geq v(f)$. Then there is a flow f'' of value $v(f') - v(f)$ in G_f.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let f be a flow in G and G_f be the residual graph. If f' is a flow in G_f then $f + f'$ is a flow in G of value $v(f) + v(f')$.

Lemma

Let f and f' be two flows in G with $v(f') \geq v(f)$. Then there is a flow f'' of value $v(f') - v(f)$ in G_f.

No s to t flow in G_f then f is a maximum flow.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let f be a flow in G and G_f be the residual graph. If f' is a flow in G_f then $f + f'$ is a flow in G of value $v(f) + v(f')$.

Lemma

Let f and f' be two flows in G with $v(f') \geq v(f)$. Then there is a flow f'' of value $v(f') - v(f)$ in G_f.

No s to t flow in G_f then f is a maximum flow.

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

Ford-Fulkerson Algorithm

algFordFulkerson

for every edge e, $f(e) = 0$
G_f is residual graph of G with respect to f
while G_f has a simple s-t path **do**
  let P be simple s-t path in G_f
  $f = \text{augment}(f, P)$
  Construct new residual graph G_f.

$O(C)$

largest bottleneck
$O(m \log n)$

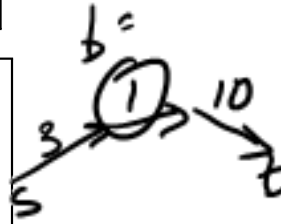
Ford-Fulkerson Algorithm

algFordFulkerson

for every edge e, $f(e) = 0$
G_f is residual graph of G with respect to f
while G_f has a simple s-t path **do**
  let P be simple s-t path in G_f
  $f = \text{augment}(f, P)$
  Construct new residual graph G_f.

augment(f, P)

let b be bottleneck capacity,
  i.e., min capacity of edges in P (in G_f)
for each edge (u, v) in P **do**
  if $e = (u, v)$ is a forward edge **then**
    $f(e) = f(e) + b$
  else (* (u, v) is a backward edge *)
    let $e = (v, u)$ (* (v, u) is in G *)
    $f(e) = f(e) - b$
return f



Properties of Augmentation

Lemma

At every stage of the Ford-Fulkerson algorithm, the flow values on the edges (i.e., $f(e)$, for all edges e) and the residual capacities in G_f are integers.

Proposition

Let f be a flow and f' be flow after one augmentation. Then $v(f) < v(f')$.

Properties of Augmentation

Lemma

At every stage of the Ford-Fulkerson algorithm, the flow values on the edges (i.e., $f(e)$, for all edges e) and the residual capacities in G_f are integers.

Proposition

Let f be a flow and f' be flow after one augmentation. Then $v(f) < v(f')$.

Since edges in G_f have integer capacities, $v(f') \geq v(f) + 1$.

Properties of Augmentation

Lemma

At every stage of the Ford-Fulkerson algorithm, the flow values on the edges (i.e., $f(e)$, for all edges e) and the residual capacities in G_f are integers.

Proposition

Let f be a flow and f' be flow after one augmentation. Then $v(f) < v(f')$.

Since edges in G_f have integer capacities, $v(f') \geq v(f) + 1$.
Runtime: If max-flow value is C, then terminates in $O(mC)$ time.
This is tight.

Cuts

Definition

Given a flow network an **s-t cut** is a set of edges $E' \subset E$ such that removing E' *disconnects* s from t: in other words there is no directed $s \rightarrow t$ path in $E - E'$. **Capacity** of cut E' is $\sum_{e \in E'} c(e)$.

Let $A \subset V$ such that

1. $s \in A$, $t \notin A$, and
2. $B = V \setminus A$ and hence $t \in B$.

$B = \overline{A}$

Define $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$

$c(A,B) = \sum_{e \in (A,B)} c(e)$

Claim

(A, B) *is an* s-t *cut.*

Recall: Every *minimal* s-t cut E' is a cut of the form (A, B).

Cuts

Definition

Given a flow network an **s-t cut** is a set of edges $E' \subset E$ such that removing E' *disconnects* s from t: in other words there is no directed $s \rightarrow t$ path in $E - E'$. **Capacity** of cut E' is $\sum_{e \in E'} c(e)$.

Let $A \subset V$ such that

1. $s \in A$, $t \notin A$, and
2. $B = V \setminus -A$ and hence $t \in B$.

Define $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$

Claim

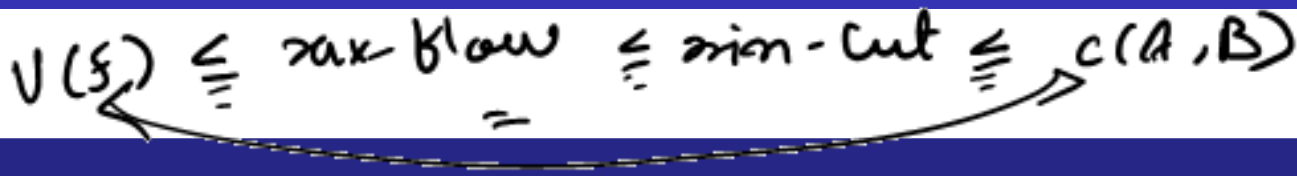
(A, B) *is an* s-t *cut.*

Recall: Every *minimal* s-t cut E' is a cut of the form (A, B).

Max-flow $\leq$ Min-cut

Ford-Fulkerson Correctness

Max-flow = Min-cut Theorem



Lemma

If there is no s-t path in G_f then there is some cut (A, B) such that $v(f) = c(A, B)$

Ford-Fulkerson Correctness

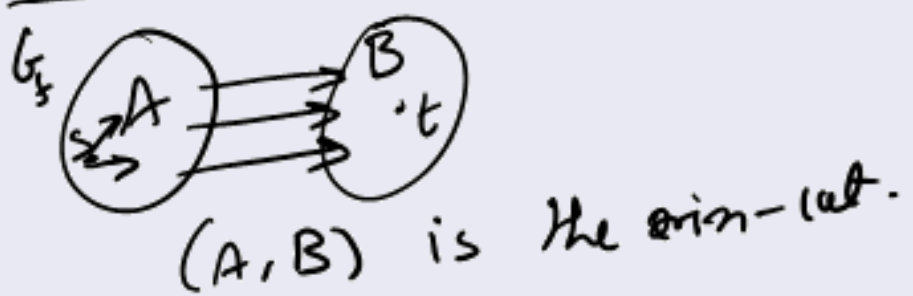
Max-flow = Min-cut Theorem

Lemma

If there is no s-t path in G_f then there is some cut (A, B) such that $v(f) = c(A, B)$

Proof.

Let A be all vertices reachable from s in G_f; $B = V \setminus A$.



Ford-Fulkerson Correctness

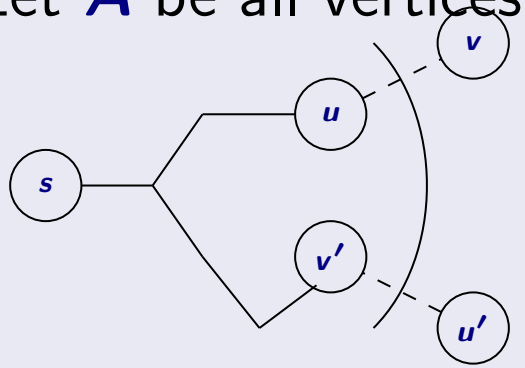
Max-flow = Min-cut Theorem

Lemma

If there is no s-t path in G_f then there is some cut (A, B) such that $v(f) = c(A, B)$

Proof.

Let A be all vertices reachable from s in G_f; $B = V \setminus A$.

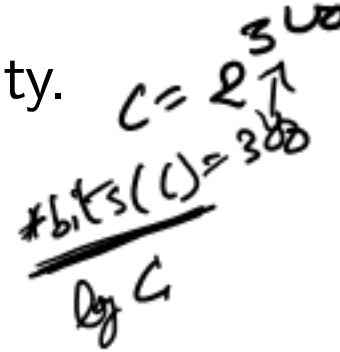


We showed that $c(A, B) = v(f)$

Polynomial-time variants

Choose the augmenting path with largest bottleneck capacity.

- Finding such a path takes $O(m \log m)$ time.
- Algorithm terminates in $O(m \log mC)$ iterations.
- Overall run-time: $(m^2 \log m \log mC)$



Polynomial-time variants

Choose the augmenting path with largest bottleneck capacity.

- Finding such a path takes $O(m \log m)$ time.
- Algorithm terminates in $O(m \log mC)$ iterations.
- Overall run-time:

Edmond-Karp: Augment along shortest path

- Terminates in $O(mn)$ iterations, hence overall run-time: $O(mn^2)$

Polynomial-time variants

Choose the augmenting path with largest bottleneck capacity.

- Finding such a path takes $O(m \log m)$ time.
- Algorithm terminates in $O(m \log mC)$ iterations.
- Overall run-time:

Edmond-Karp: Augment along shortest path

- Terminates in $O(mn)$ iterations, hence overall run-time:

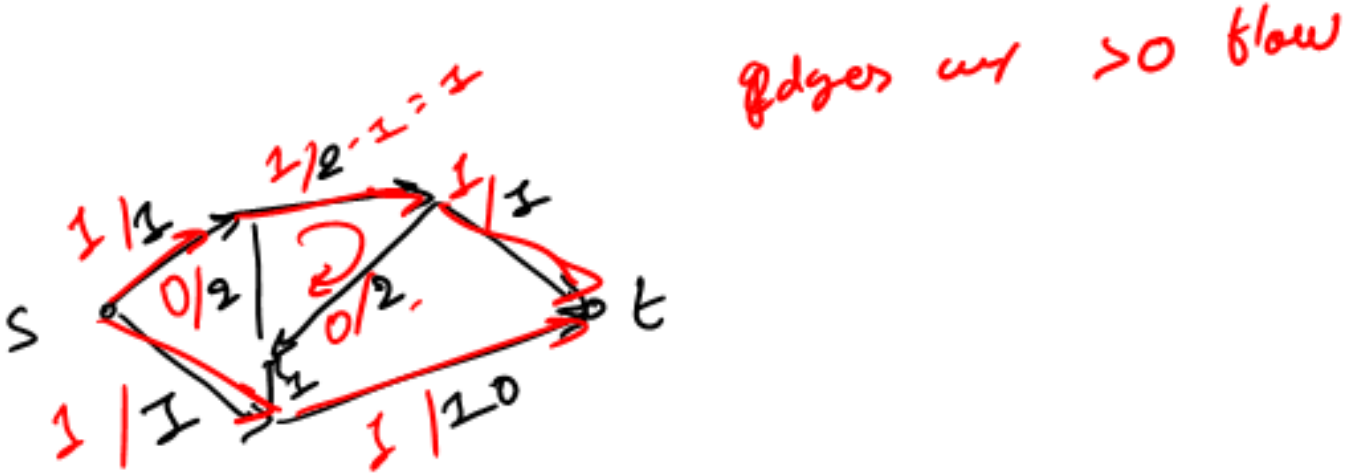
Orlin gave an $O(mn)$ time algorithm.

Acyclicity of Flows

Proposition

In any flow network, if f *is a flow then there is another flow* f' *such that the support of* f' *is an* acyclic *graph and* $v(f') = v(f)$. *Further if* f *is an integral flow then so is* f'.

Proof: Repeatedly cancel flow along a cycle to get an acyclic flow.



Circulation problem

Problem

Input A network G with capacity c and lower bound ℓ

Goal Find a feasible circulation

Simply a feasibility problem.

Observation: Equivalent to s-t maxflow!

Important properties of circulations

- Reduction shows that one can find in $O(mn)$ time a feasible circulation in a network with capacities and lower bounds
- If edge capacities and lower bounds are integer valued then there is always a feasible integer-valued circulation
- Hoffman's circulation theorem is the equivalent of maxflow-mincut theorem.
- Circulation can be decomposed into at most m cycles in $O(mn)$ time.

Minimum Cost Flows

1. **Input:** Given a flow network G and also edge costs, $w(e)$ for edge e, and a flow requirement F.
2. **Goal;** Find a *minimum cost* flow of value F from s to t

Given flow $f : E \rightarrow R^+$, cost of flow $= \sum_{e \in E} w(e) f(e)$.

Minimum Cost Flows

1. **Input:** Given a flow network G and also edge costs, $w(e)$ for edge e, and a flow requirement F.
2. **Goal;** Find a *minimum cost* flow of value F from s to t

Given flow $f : E \rightarrow R^+$, cost of flow $= \sum_{e \in E} w(e)f(e)$.

Much more general than the shortest path problem.

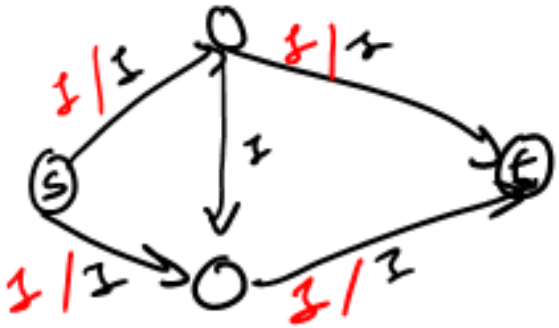
Minimum Cost Flow: Facts

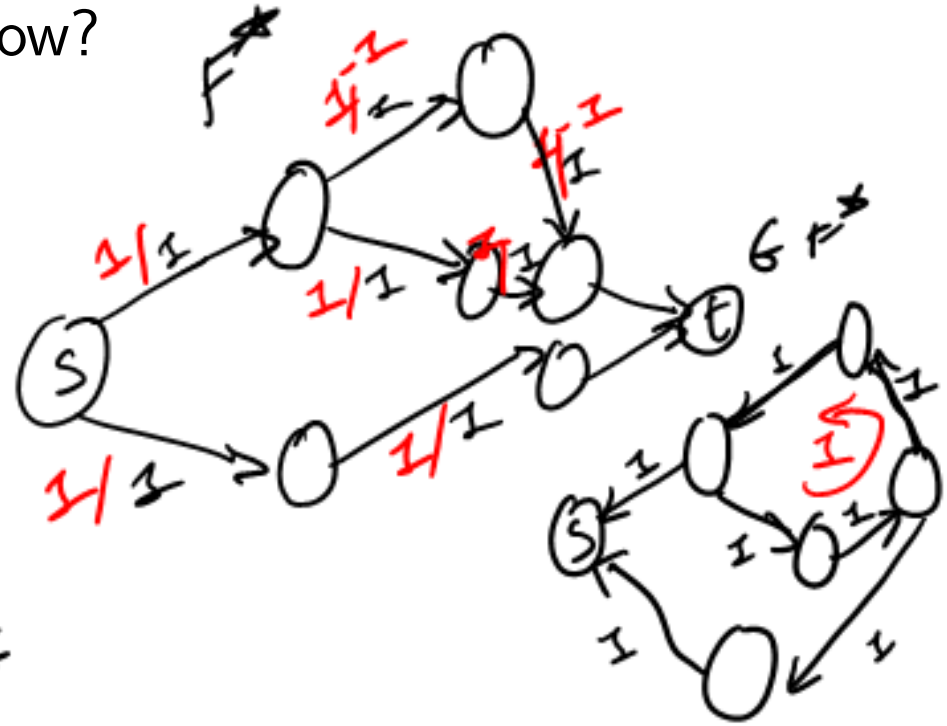
1. problem can be solved efficiently in polynomial time
   1. $O(nm \log C \log(nW))$ time algorithm where C is maximum edge capacity and W is maximum edge cost
   2. $O(m \log n(m + n \log n))$ time strongly polynomial time algorithm
2. for integer capacities there is always an optimum solution in which flow is integral

Max-Flows/Min-Cut: Example Problem

Max-Flows/Min-Cut: Example Problem

How to tell if G has a unique max-flow?





→ Compute max-flow F^*

→ check if G_{F^*} is DAG

Claim: F^* is unique $\Leftrightarrow$ G_{F^*} is a DAG.

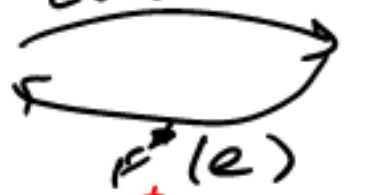
Pf: ① G_{F^*} has a cycle ⇒ push flow along the cycle ↓ f' $val(f') = Val(F^*)$

(2) f' & F^* both are max flow $\Rightarrow$ G_{F^*} has a cycle.

$f' - F^* \Rightarrow$ gives a cycle in G_{F^*}

$0 \le F^*(e),\ f'(e) \le c(e)$

G_{F^*}

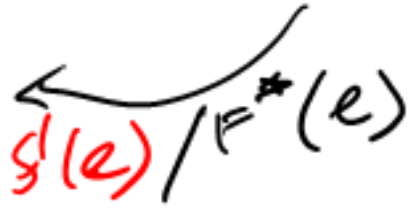


if $f'(e) > F^*(e)$ $\longrightarrow$ $f'(e) - F^*(e)$ / $c(e) - F^*(e)$

$\wedge$

$c(e)$

if $F^*(e) > f'(e)$ $\longrightarrow$



Claim: $f'' = f' - F^*$ is a circulation.

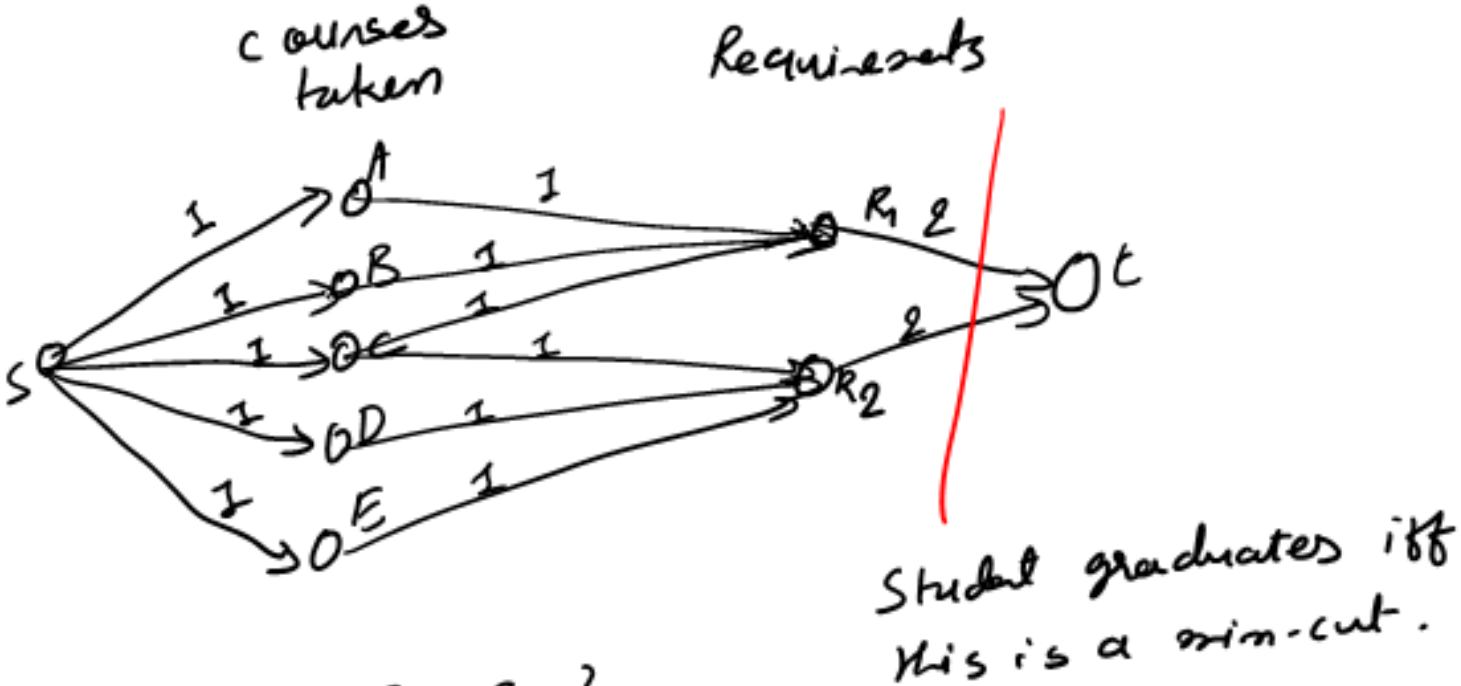
Pf: Since $\text{Val}(f') = \text{Val}(F^*)$, net flow out of s & into t are 0 as per f''. At every other node f'' conserves flow.

Applications: Example Problem

Menger's Theorem: Max # edge disjoint s-t paths = Min # edges needed to disconnect s from t.

Applications: Example Problem

Menger's Theorem: Max # edge disjoint s-t paths = Min # edges needed to disconnect s from t. Spring 2017, Problem 4.



Spring 2017, Problem 3 (min vertex-cut)

Given undirected graph $G = (V, E)$

$u \in V$

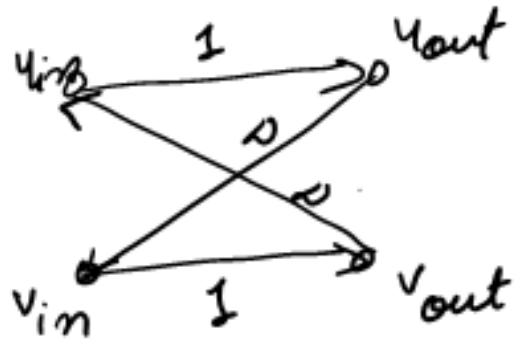
$\Rightarrow$

$\Rightarrow \{u, v\} \in V$

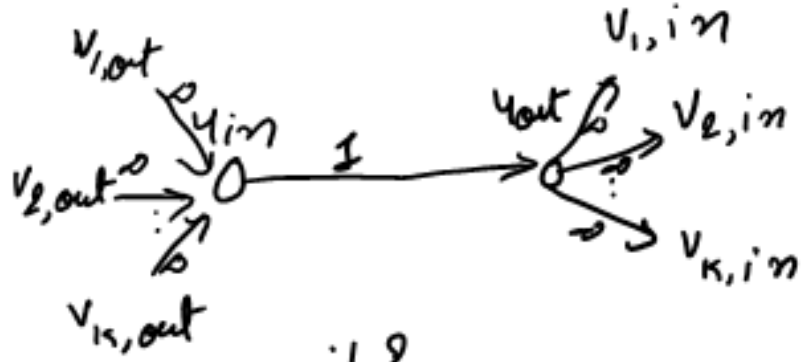
$\longrightarrow$

$G' = (V', E')$

$u_{in}, u_{out} \in V'$



$\infty \sim (n+1)$



Obs 1: $\exists$ a cut in G' of cap n. What is it?

Algo:

-> construct G'. set $S = \emptyset$

-> compute min-cut in G'

(every edge in the min-cut is of type (u_{in}, u_{out}))

-> For each edge e in the min-cut

if $e = (u_{in}, u_{out})$

$S = S \cup \{u\}$.

Proof of correctness:

-> Consider the corresponding max-flow in G. And it's flow decomposition into paths $P_1, \ldots, P_k$.

-> $f(P_i) = 1$. And P_i is of type $u_{in} \to u_{out} \to v_{in} \to v_{out} \to \ldots$ (in & out vertices alternate) ... $\leftarrow u_{out} \leftarrow u_{in}$

$\Downarrow$

P_i's are edge disjoint. $\Rightarrow$ corresponding paths in G are vertex-disjoint

□

Finger Printing: Example

Describe and analyze an algorithm to determine, given two strings $A[1 \ldots m]$ and $B[1 \ldots n]$ with $m \leq n$, whether A is a substring of some left cyclic shift of B.

Universal Hashing: Example