Review of last lecture

- Bellman-Ford algorithm: a dynamic programming that
  1. Reports a negative cycle when one exists;
  2. Finds min-cost paths when no negative cycle exists.
- Running time: $O(mn)$: m operations per iteration, $O(n)$ iterations.
- Proof ideas:
  1. first i iterations find out min-cost paths with at most i edges
  2. Min-cost paths must be simple when there is no negative cycle

Learning Goals

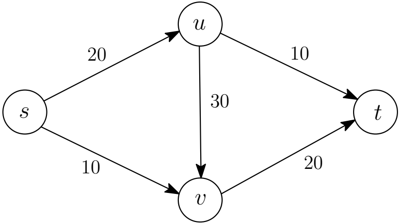
- Definition of flow network and flows
- Motivation and definition of Residual graphs
- Ford-Fulkerson algorithm description
- Running time of Ford-Fulkerson

Flow Networks

A *flow network* is a directed graph $G = (V, E)$, which includes two special nodes: s, called the *source*, and t, called the *sink*. Each edge e is associated with a capacity $c_e > 0$.
We assume there is no incoming edge to s and no outgoing edge from t.

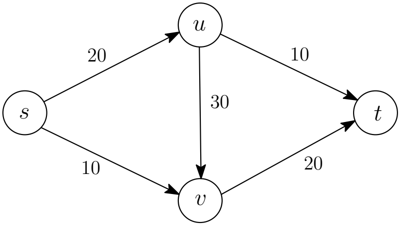
Flow Networks

A *flow network* is a directed graph $G = (V, E)$, which includes two special nodes: s, called the *source*, and t, called the *sink*. Each edge e is associated with a capacity $c_e > 0$.
We assume there is no incoming edge to s and no outgoing edge from t.



Flow Networks

A *flow network* is a directed graph $G = (V, E)$, which includes two special nodes: s, called the *source*, and t, called the *sink*. Each edge e is associated with a capacity $c_e > 0$.
We assume there is no incoming edge to s and no outgoing edge from t.



Notation: for $v \in V$, $\delta_{\mathrm{in}}(v)$ is the set of edges going into v, and $\delta_{\mathrm{out}}(v)$ the set of edges going out of v.

Flows

Definition

A *flow* is a function $f : E \to \mathbb{R}^+$ satisfying:

1. *Capacity conditions*: $\forall e \in E,\ 0 \leq f(e) \leq c_e$.
2. *Conservation conditions*: $\forall v \in V \setminus \{s, t\}$,

$$\sum_{e \in \delta_{\mathrm{in}}(v)} f(e) = \sum_{e \in \delta_{\mathrm{out}}(v)} f(e).$$

Flows

Definition

A *flow* is a function $f : E \to \mathbb{R}^+$ satisfying:

1. *Capacity conditions*: $\forall e \in E$, $0 \leq f(e) \leq c_e$.
2. *Conservation conditions*: $\forall v \in V \setminus \{s, t\}$,

$$\sum_{e \in \delta_{\mathrm{in}}(v)} f(e) = \sum_{e \in \delta_{\mathrm{out}}(v)} f(e).$$

The *value* of a flow f is $\sum_{e \in \delta_{\mathrm{out}}(s)} f(e)$, denoted as $|f|$.

Flows

Definition

A *flow* is a function $f : E \to \mathbb{R}^+$ satisfying:

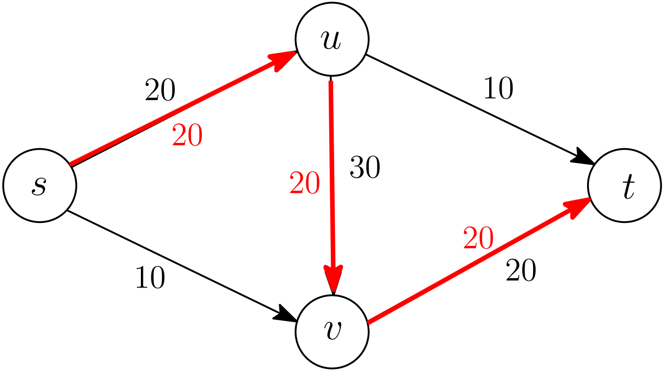
1. *Capacity conditions*: $\forall e \in E$, $0 \leq f(e) \leq c_e$.
2. *Conservation conditions*: $\forall v \in V \setminus \{s, t\}$,

$$\sum_{e \in \delta_{\mathrm{in}}(v)} f(e) = \sum_{e \in \delta_{\mathrm{out}}(v)} f(e).$$

The *value* of a flow f is $\sum_{e \in \delta_{\mathrm{out}}(s)} f(e)$, denoted as $|f|$.

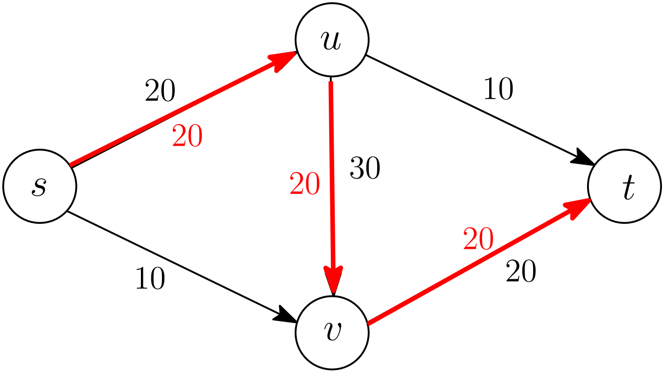
The maximum flow problem: given a flow network, compute a flow with maximum value.

Flow Example



Example of a flow (in red).

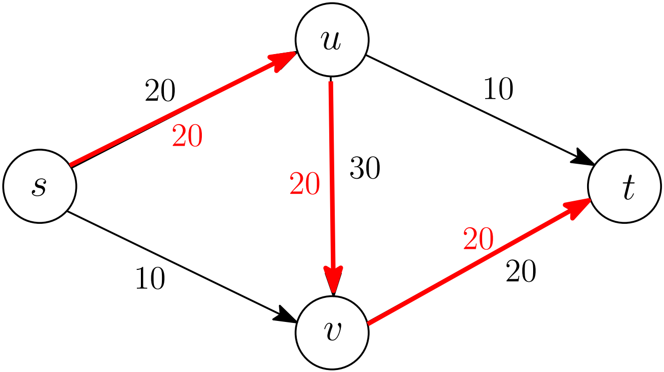
Flow Example



Example of a flow (in red).

The value of the flow is 20.

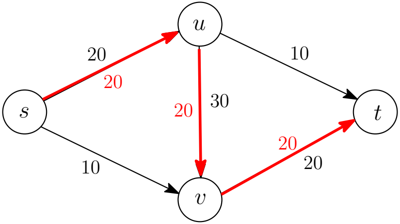
Flow Example



Example of a flow (in red).

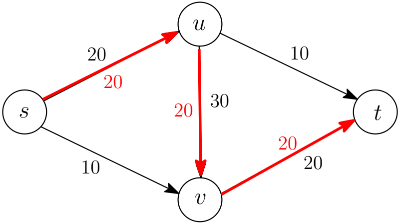
The value of the flow is 20.
Is this a maximum flow?

Flow Example Cont.

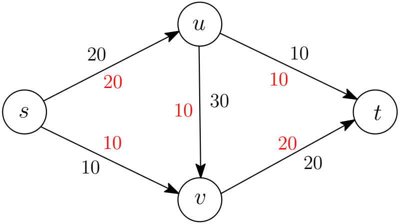


Flow from previous page.

Flow Example Cont.

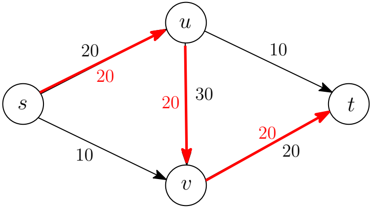


Flow from previous page.



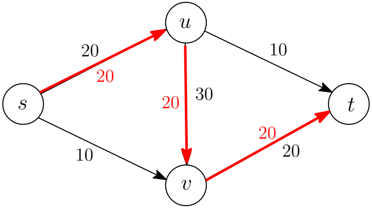
Maximum flow.

Residual Graph

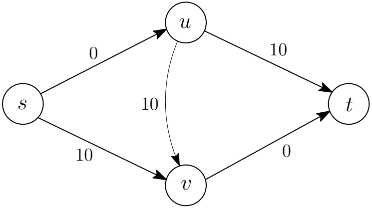


Flow from previous page.

Residual Graph

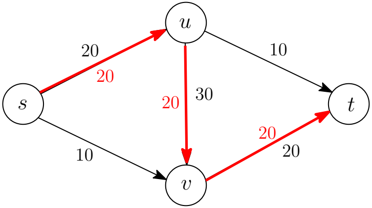


Flow from previous page.

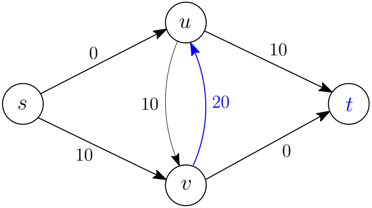


How much capacity is left on each edge?

Residual Graph

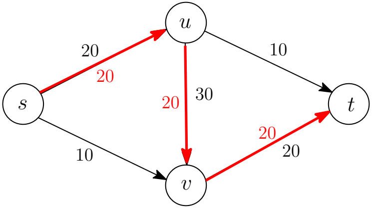


Flow from previous page.

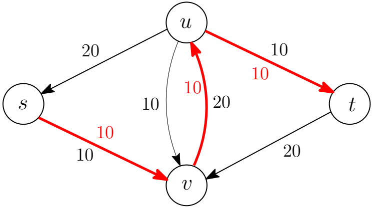


How much capacity is left on each edge?

Residual Graph



Flow from previous page.



A flow pushed in the residual graph along a path.

The Residual Graph

Definition

Given a flow f in a flow network G, the *residual graph* G_f is defined as:

- G_f has the same set of nodes as G (including s and t);
- for each $e = (u, v)$ of G on which $f(e) < c_e$, e is in G_f and is called a *forward edge*; its capacity in G_f is $c_e - f(e)$;
- for each $e = (u, v)$ of G on which $f(e) > 0$, (v, u) is in G_f and is called a *backward edge*; its capacity in G_f is $f(e)$.

The Residual Graph

Definition

Given a flow f in a flow network G, the *residual graph* G_f is defined as:

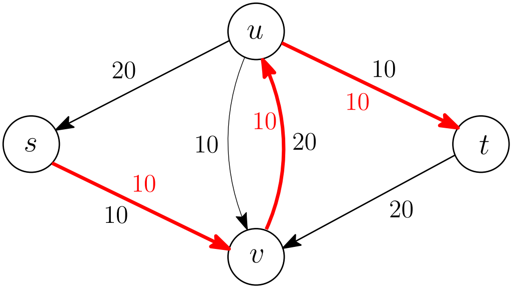
- G_f has the same set of nodes as G (including s and t);
- for each $e = (u, v)$ of G on which $f(e) < c_e$, e is in G_f and is called a *forward edge*; its capacity in G_f is $c_e - f(e)$;
- for each $e = (u, v)$ of G on which $f(e) > 0$, (v, u) is in G_f and is called a *backward edge*; its capacity in G_f is $f(e)$.

The capacities in G_f are called the *residual capacities*.
Note that the residual capacities are all strictly positive.

Augmenting paths

Definition

Any (simple) s-t path in the residual graph G_f is called an *augmenting path*. In an augmenting path P in the residual graph G_f, the minimum residual capacity is called the *bottleneck*, denoted as bottleneck(P, f).



(s, u, v, t) is an augmenting path; its bottleneck is 10.

Augmenting along a path

Given a flow f in G, if there is an s-t path P in its residual graph G_f, the following operations on f is called an *augmentation along P*

- let b be bottleneck$(P, f) > 0$;

Augmenting along a path

Given a flow f in G, if there is an s-t path P in its residual graph G_f, the following operations on f is called an *augmentation along P*

- let b be bottleneck$(P, f) > 0$;
- for each e in P that is forward, increase $f(e)$ in G by b;

Augmenting along a path

Given a flow f in G, if there is an s-t path P in its residual graph G_f, the following operations on f is called an *augmentation along* P

- let b be bottleneck$(P, f) > 0$;
- for each e in P that is forward, increase $f(e)$ in G by b;
- for each $e = (u, v)$ in P that is backward, decrease $f((v, u))$ in G by b.

Augmenting along a path

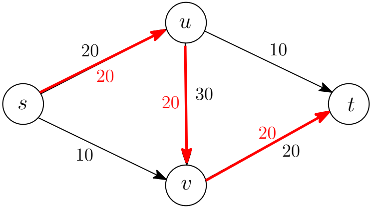
Given a flow f in G, if there is an s-t path P in its residual graph G_f, the following operations on f is called an *augmentation along P*

- let b be bottleneck$(P, f) > 0$;
- for each e in P that is forward, increase $f(e)$ in G by b;
- for each $e = (u, v)$ in P that is backward, decrease $f((v, u))$ in G by b.

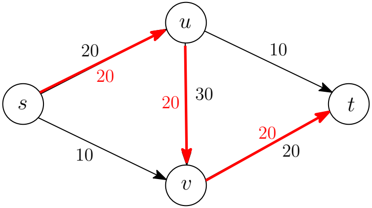
Proposition

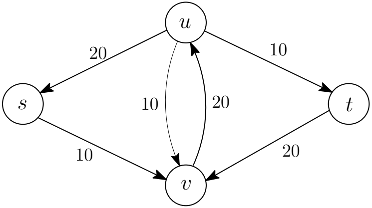
The result of an augmentation, f', is a flow in G, with value $|f| + b$.

Augmentation example

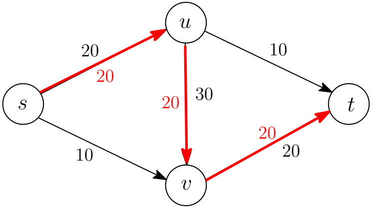


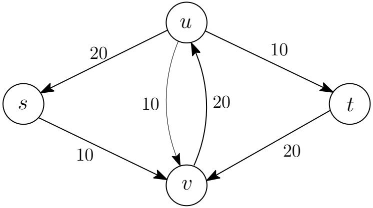
Augmentation example

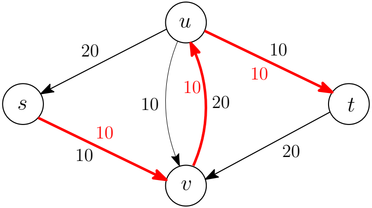




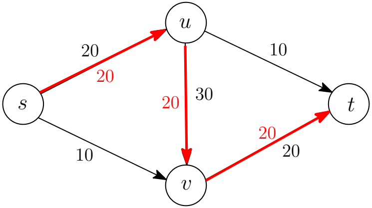
Augmentation example

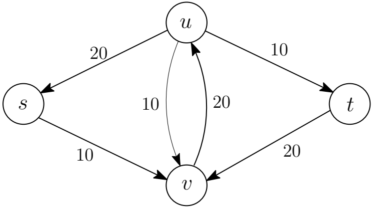


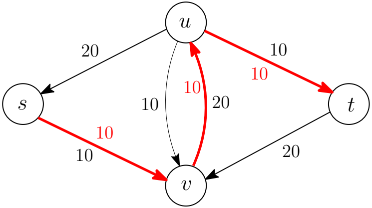


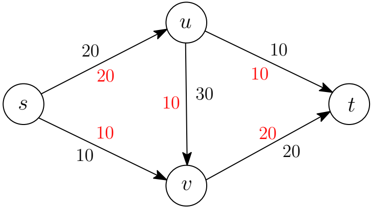


Augmentation example









Proof of the proposition

Proposition

The result of an augmentation, f', is a flow in G, with value $|f| + b$.

Proof of the proposition

Proposition

The result of an augmentation, f', is a flow in G, with value $|f| + b$.

Proof.

To check f' is a flow, we check:

1. Capacity conditions:

Proof of the proposition

Proposition

The result of an augmentation, f', is a flow in G, with value $|f| + b$.

Proof.

To check f' is a flow, we check:

1. Capacity conditions:
   - If e is forward, its residual capacity is $c_e - f(e)$, so $f'(e) = f(e) + b \leq f(e) + (c_e - f(e)) = c_e$;

Proof of the proposition

Proposition

The result of an augmentation, f', is a flow in G, with value $|f| + b$.

Proof.

To check f' is a flow, we check:

1. Capacity conditions:
   - If e is forward, its residual capacity is $c_e - f(e)$, so $f'(e) = f(e) + b \leq f(e) + (c_e - f(e)) = c_e$;
   - If $e = (u, v)$ is backward, its residual capacity is $f((v, u))$, so $f'((v, u)) = f((v, u)) - b \geq f((v, u)) - f((v, u)) = 0$.

Proof of the proposition

Proposition

The result of an augmentation, f', is a flow in G, with value $|f| + b$.

Proof.

To check f' is a flow, we check:

1. Capacity conditions:
   - If e is forward, its residual capacity is $c_e - f(e)$, so $f'(e) = f(e) + b \leq f(e) + (c_e - f(e)) = c_e$;
   - If $e = (u, v)$ is backward, its residual capacity is $f((v, u))$, so $f'((v, u)) = f((v, u)) - b \geq f((v, u)) - f((v, u)) = 0$.
2. Conservation conditions: case study for each node on the augmenting path.

Proof of the proposition

Proposition

The result of an augmentation, f', is a flow in G, with value $|f| + b$.

Proof.

To check f' is a flow, we check:

1. Capacity conditions:
   - If e is forward, its residual capacity is $c_e - f(e)$, so $f'(e) = f(e) + b \leq f(e) + (c_e - f(e)) = c_e$;
   - If $e = (u, v)$ is backward, its residual capacity is $f((v, u))$, so $f'((v, u)) = f((v, u)) - b \geq f((v, u)) - f((v, u)) = 0$.
2. Conservation conditions: case study for each node on the augmenting path.

An augmenting path starts from s, and $\delta_{\text{in}}(s) = \emptyset$, so the first edge in the path is forward.

$\square$

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return f. Otherwise augment along the path found and repeat.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return f. Otherwise augment along the path found and repeat.

Running time: each round takes $O(m)$ time, but how many rounds?

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return f. Otherwise augment along the path found and repeat.

Running time: each round takes $O(m)$ time, but how many rounds?

Proposition

Suppose all capacities are integers. Let C be $\sum_{e \in \delta_{\text{out}}(s)} c_e$. The Ford-Fulkerson algorithm terminates in at most C rounds.

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm:

- Initialize: $f(e) \leftarrow 0$ for all $e \in E$.
- Iterate: construct G_f and search for an augmenting path. If no augmenting path can be found, terminate and return f. Otherwise augment along the path found and repeat.

Running time: each round takes $O(m)$ time, but how many rounds?

Proposition

Suppose all capacities are integers. Let C be $\sum_{e \in \delta_{\text{out}}(s)} c_e$. The Ford-Fulkerson algorithm terminates in at most C rounds.

Running time $O(Cm)$.