$\leq \frac{n}{2^s}$ nodes of rank s

What is the total charge made to nodes?

- The total charge is at most

$$\sum_g n(g) \cdot \mathrm{tow}(g) \ ,$$

↳ # vertices in group g

where $n(g)$ is the number of nodes in group g.

Amortized Analysis

For $g \geq 1$ we have

$$n(g)$$

Amortized Analysis

For $g \geq 1$ we have

$$n(g) \leq \sum_{s=\text{tow}(g-1)+1}^{\text{tow}(g)} \frac{n}{2^s}$$

Amortized Analysis

For $g \geq 1$ we have

$$n(g) \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\mathrm{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\infty} \frac{n}{2^s}$$

Amortized Analysis

For $g \geq 1$ we have

$$n(g) \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\mathrm{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\infty} \frac{n}{2^s}$$

$$= \frac{n}{2^{\mathrm{tow}(g-1)+1}} \sum_{s=0}^{\infty} \frac{1}{2^s}$$

Amortized Analysis

For $g \geq 1$ we have

$$n(g) \leq \sum_{s=\text{tow}(g-1)+1}^{\text{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\text{tow}(g-1)+1}^{\infty} \frac{n}{2^s}$$

$$= \frac{n}{2^{\text{tow}(g-1)+1}} \sum_{s=0}^{\infty} \frac{1}{2^s} = \frac{n}{2^{\text{tow}(g-1)+1}} \cdot 2$$

Amortized Analysis

For $g \geq 1$ we have

$$
\begin{aligned}
n(g) &\leq \sum_{s=\mathrm{tow}(g-1)+1}^{\mathrm{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\infty} \frac{n}{2^s} \\
&= \frac{n}{2^{\mathrm{tow}(g-1)+1}} \sum_{s=0}^{\infty} \frac{1}{2^s} = \frac{n}{2^{\mathrm{tow}(g-1)+1}} \cdot 2 \\
&= \frac{n}{2^{\mathrm{tow}(g-1)}}
\end{aligned}
$$

Amortized Analysis

For $g \geq 1$ we have

$$
\begin{aligned}
n(g) &\leq \sum_{s=\mathrm{tow}(g-1)+1}^{\mathrm{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\infty} \frac{n}{2^s} \\
&= \frac{n}{2^{\mathrm{tow}(g-1)+1}} \sum_{s=0}^{\infty} \frac{1}{2^s} = \frac{n}{2^{\mathrm{tow}(g-1)+1}} \cdot 2 \\
&= \frac{n}{2^{\mathrm{tow}(g-1)}} = \frac{n}{\mathrm{tow}(g)} \enspace .
\end{aligned}
$$

Amortized Analysis

For $g \geq 1$ we have

$$n(g) \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\mathrm{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\infty} \frac{n}{2^s}$$

$$= \frac{n}{2^{\mathrm{tow}(g-1)+1}} \sum_{s=0}^{\infty} \frac{1}{2^s} = \frac{n}{2^{\mathrm{tow}(g-1)+1}} \cdot 2$$

$$= \frac{n}{2^{\mathrm{tow}(g-1)}} = \frac{n}{\mathrm{tow}(g)} \ .$$

Hence,

$$\sum_g n(g)\,\mathrm{tow}(g)$$

Amortized Analysis

For $g \geq 1$ we have

$$n(g) \leq \sum_{s=\text{tow}(g-1)+1}^{\text{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\text{tow}(g-1)+1}^{\infty} \frac{n}{2^s}$$

$$= \frac{n}{2^{\text{tow}(g-1)+1}} \sum_{s=0}^{\infty} \frac{1}{2^s} = \frac{n}{2^{\text{tow}(g-1)+1}} \cdot 2$$

$$= \frac{n}{2^{\text{tow}(g-1)}} = \frac{n}{\text{tow}(g)} .$$

Hence,

$$\sum_g n(g)\,\text{tow}(g) \leq \underbrace{n(0)\,\text{tow}(0)}_{\leq n} + \sum_{g\geq 1} \underbrace{n(g)\,\text{tow}(g)}_{\leq n}$$

Amortized Analysis

For $g \geq 1$ we have

$$
\begin{aligned}
n(g) &\leq \sum_{s=\mathrm{tow}(g-1)+1}^{\mathrm{tow}(g)} \frac{n}{2^s} \leq \sum_{s=\mathrm{tow}(g-1)+1}^{\infty} \frac{n}{2^s} \\
&= \frac{n}{2^{\mathrm{tow}(g-1)+1}} \sum_{s=0}^{\infty} \frac{1}{2^s} = \frac{n}{2^{\mathrm{tow}(g-1)+1}} \cdot 2 \\
&= \frac{n}{2^{\mathrm{tow}(g-1)}} = \frac{n}{\mathrm{tow}(g)} \enspace .
\end{aligned}
$$

Hence,

$$
\sum_g n(g)\,\mathrm{tow}(g) \leq n(0)\,\mathrm{tow}(0) + \sum_{g \geq 1} n(g)\,\mathrm{tow}(g) \leq n \log^*(n)
$$

Amortized Analysis

Without loss of generality we can assume that all makeset-operations occur at the start.

This means if we inflate the cost of makeset to $\log^* n$ and add this to the node account of v then the balances of all node accounts will sum up to a positive value (this is sufficient to obtain an amortized bound).

Amortized Analysis

Without loss of generality we can assume that all makeset-operations occur at the start.

This means if we inflate the cost of makeset to $\log^* n$ and add this to the node account of v then the balances of all node accounts will sum up to a positive value (this is sufficient to obtain an amortized bound).

Amortized Analysis

The analysis is not tight. In fact it has been shown that the amortized time for the union-find data structure with path compression is $\mathcal{O}(\alpha(m,n))$, where $\alpha(m,n)$ is the inverse Ackermann function which grows a lot lot slower than $\log^* n$. (Here, we consider the average running time of m operations on at most n elements).

There is also a lower bound of $\Omega(\alpha(m,n))$.

Amortized Analysis

The analysis is not tight. In fact it has been shown that the amortized time for the union-find data structure with path compression is $\mathcal{O}(\alpha(m,n))$, where $\alpha(m,n)$ is the inverse Ackermann function which grows a lot lot slower than $\log^* n$. (Here, we consider the average running time of m operations on at most n elements).

There is also a lower bound of $\Omega(\alpha(m,n))$.

Amortized Analysis

The analysis is not tight. In fact it has been shown that the amortized time for the union-find data structure with path compression is $\mathcal{O}(\alpha(m,n))$, where $\alpha(m,n)$ is the inverse Ackermann function which grows a lot lot slower than $\log^* n$. (Here, we consider the average running time of m operations on at most n elements).

There is also a lower bound of $\Omega(\alpha(m,n))$.

Amortized Analysis

$$A(x,y) = \begin{cases} y+1 & \text{if } x = 0 \\ A(x-1,1) & \text{if } y = 0 \\ A(x-1, A(x,y-1)) & \text{otw.} \end{cases}$$

$$\alpha(m,n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log n\}$$

- $A(0,y) = y+1$
- $A(1,y) = y+2$
- $A(2,y) = 2y+3$
- $A(3,y) = 2^{y+3} - 3$
- $A(4,y) = \underbrace{2^{2^{2^{2}}}}_{y+3 \text{ times}} - 3$

Amortized Analysis

$$A(x,y) = \begin{cases} y+1 & \text{if } x = 0 \\ A(x-1,1) & \text{if } y = 0 \\ A(x-1, A(x,y-1)) & \text{otw.} \end{cases}$$

$$\alpha(m,n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log n\}$$

- $A(0,y) = y + 1$
- $A(1,y) = y + 2$
- $A(2,y) = 2y + 3$
- $A(3,y) = 2^{y+3} - 3$
- $A(4,y) = \underbrace{2^{2^{2^{2}}}}_{y+3 \text{ times}} - 3$

Part IV

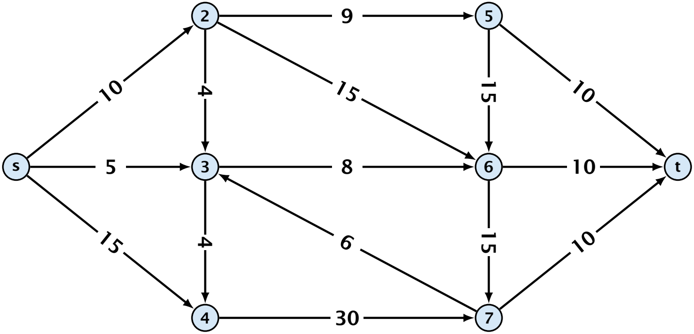
Flows and Cuts

The following slides are partially based on slides by Kevin Wayne.

10 Introduction

Flow Network

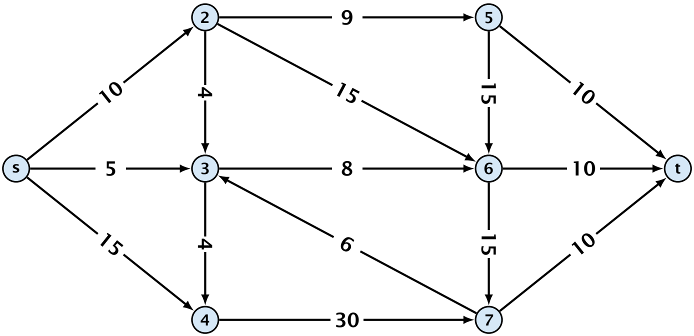
- directed graph $G = (V, E)$; edge capacities $c(e)$
- two special nodes: source s; target t;
- no edges entering s or leaving t;
- at least for now: no parallel edges;



10 Introduction

Flow Network

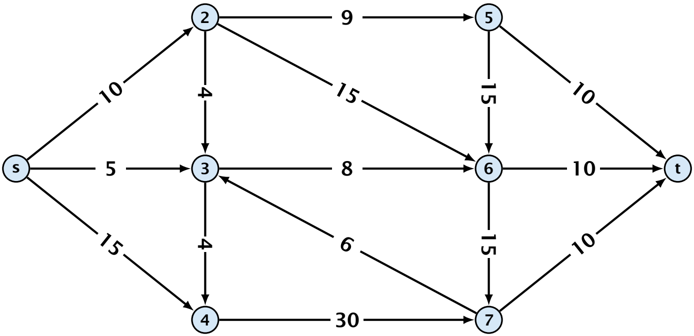
- directed graph $G = (V, E)$; edge capacities $c(e)$
- two special nodes: source s; target t;
- no edges entering s or leaving t;
- at least for now: no parallel edges;



10 Introduction

Flow Network

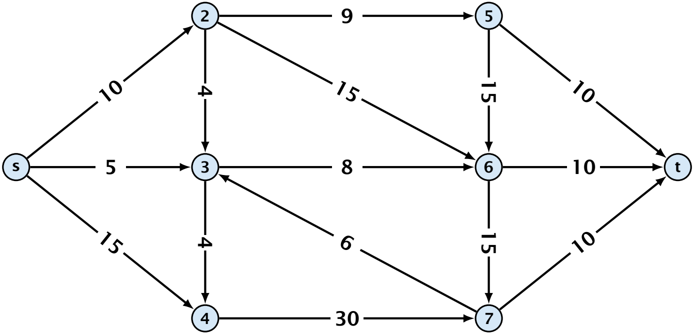
- directed graph $G = (V, E)$; edge capacities $c(e)$
- two special nodes: source s; target t;
- no edges entering s or leaving t;
- at least for now: no parallel edges;



10 Introduction

Flow Network

- directed graph $G = (V, E)$; edge capacities $c(e)$
- two special nodes: source s; target t;
- no edges entering s or leaving t;
- at least for now: no parallel edges;



Cuts

Definition 40

An (s, t)-cut in the graph G is given by a set $A \subset V$ with $s \in A$ and $t \in V \setminus A$.

Cuts

Definition 40

An (s,t)-cut in the graph G is given by a set $A \subset V$ with $s \in A$ and $t \in V \setminus A$.

Definition 41

The capacity of a cut A is defined as

$$\operatorname{cap}(A, V \setminus A) := \sum_{e \in \operatorname{out}(A)} c(e) \ ,$$

where $\operatorname{out}(A)$ denotes the set of edges of the form $A \times V \setminus A$ (i.e. edges leaving A).

Cuts

Definition 40
An (s,t)-cut in the graph G is given by a set $A \subset V$ with $s \in A$ and $t \in V \setminus A$.

Definition 41
The capacity of a cut A is defined as

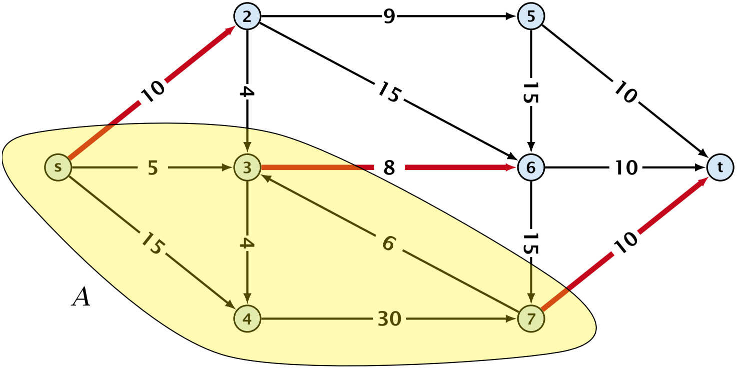
$$\text{cap}(A, V \setminus A) := \sum_{e \in \text{out}(A)} c(e) \ ,$$

where $\text{out}(A)$ denotes the set of edges of the form $A \times V \setminus A$ (i.e. edges leaving A).

Minimum Cut Problem: Find an (s,t)-cut with minimum capacity.

Cuts

Example 42



The capacity of the cut is $\text{cap}(A, V \setminus A) = 28$.

Flows

Definition 43

An (s,t)-flow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e

$$0 \le f(e) \le c(e) \enspace .$$

(capacity constraints)

2. For each $v \in V \setminus \{s,t\}$

$$\sum_{e \in \text{out}(v)} f(e) = \sum_{e \in \text{into}(v)} f(e) \enspace .$$

(flow conservation constraints)

Flows

Definition 43

An (s,t)-flow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e
$$0 \le f(e) \le c(e) \ .$$
(capacity constraints)
2. For each $v \in V \setminus \{s,t\}$
$$\sum_{e \in \text{out}(v)} f(e) = \sum_{e \in \text{into}(v)} f(e) \ .$$
(flow conservation constraints)

Flows

Definition 44
The value of an (s,t)-flow f is defined as

$$\text{val}(f) = \sum_{e \in \text{out}(s)} f(e) \ .$$

Maximum Flow Problem: Find an (s,t)-flow with maximum value.

Definition 44

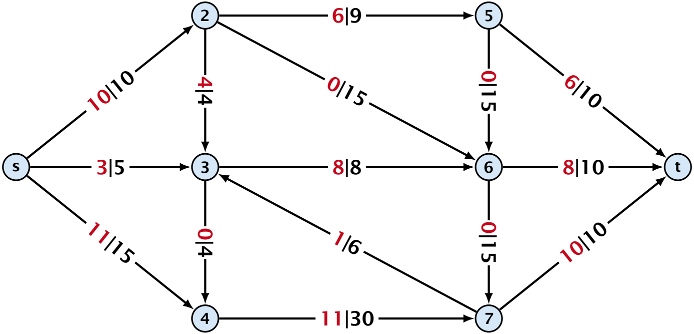
The value of an (s,t)-flow f is defined as

$$\mathrm{val}(f) = \sum_{e \in \mathrm{out}(s)} f(e) \enspace .$$

Maximum Flow Problem: Find an (s,t)-flow with maximum value.

Flows

Example 45



The value of the flow is $\text{val}(f) = 24$.

Flows

Lemma 46 (Flow value lemma)

Let f be a flow, and let $A \subseteq V$ be an (s,t)-cut. Then the net-flow across the cut is equal to the amount of flow leaving s, i.e.,

$$\operatorname{val}(f) = \sum_{e \in \operatorname{out}(A)} f(e) - \sum_{e \in \operatorname{into}(A)} f(e) \ .$$

Proof.

$\mathrm{val}(f)$

Proof.

$$\text{val}(f) = \sum_{e \in \text{out}(s)} f(e)$$

Proof.

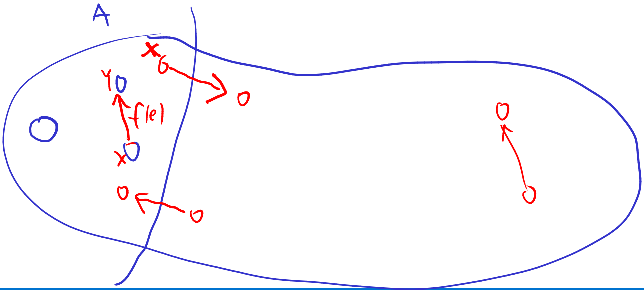
$$\begin{aligned}\mathrm{val}(f) &= \sum_{e\in\mathrm{out}(s)} f(e)\\ &= \sum_{e\in\mathrm{out}(s)} f(e) + \sum_{v\in A\setminus\{s\}}\left(\sum_{e\in\mathrm{out}(v)} f(e) - \sum_{e\in\mathrm{in}(v)} f(e)\right)\end{aligned}$$

Proof.

$$\begin{aligned}\mathrm{val}(f) &= \sum_{e\in\mathrm{out}(s)} f(e)\\ &= \sum_{e\in\mathrm{out}(s)} f(e) + \sum_{v\in A\setminus\{s\}} \underbrace{\left(\sum_{e\in\mathrm{out}(v)} f(e) - \sum_{e\in\mathrm{in}(v)} f(e)\right)}_{=\,0}\end{aligned}$$

Proof.

$$
\begin{aligned}
\mathrm{val}(f) &= \sum_{e \in \mathrm{out}(s)} f(e) \\
&= \sum_{e \in \mathrm{out}(s)} f(e) + \sum_{v \in A \setminus \{s\}} \Bigg(\sum_{e \in \mathrm{out}(v)} f(e) - \sum_{e \in \mathrm{in}(v)} f(e) \Bigg) \\
&= \sum_{e \in \mathrm{out}(A)} f(e) - \sum_{e \in \mathrm{into}(A)} f(e)
\end{aligned}
$$

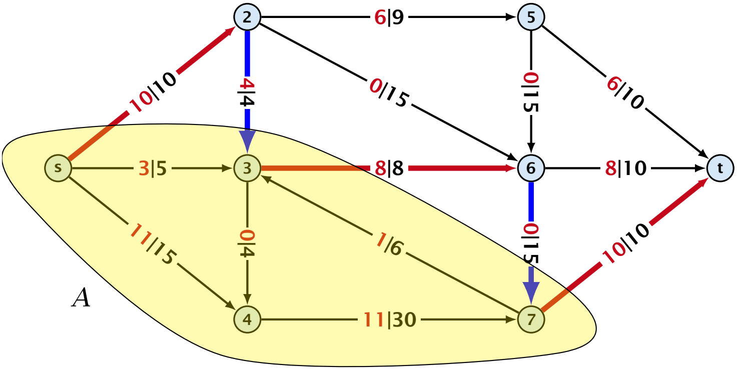


Proof.

$$\begin{aligned}
\operatorname{val}(f) &= \sum_{e\in\operatorname{out}(s)} f(e) \\
&= \sum_{e\in\operatorname{out}(s)} f(e) + \sum_{v\in A\setminus\{s\}} \left(\sum_{e\in\operatorname{out}(v)} f(e) - \sum_{e\in\operatorname{in}(v)} f(e) \right) \\
&= \sum_{e\in\operatorname{out}(A)} f(e) - \sum_{e\in\operatorname{into}(A)} f(e)
\end{aligned}$$

The last equality holds since every edge with both end-points in A contributes negatively as well as positively to the sum in Line 2. The only edges whose contribution doesn't cancel out are edges leaving or entering A. □

Example 47



2
5
s
3
6
t
4
7
A
6|9
10|10
4|4
0|15
0|15
6|10
3|5
8|8
8|10
11|15
0|4
1|6
0|15
10|10
11|30

Corollary 48

Let f be an (s,t)-flow and let A be an (s,t)-cut, such that

$$\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A).$$

Then f is a maximum flow.

Corollary 48

Let f be an (s,t)-flow and let A be an (s,t)-cut, such that

$$\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A).$$

Then f is a maximum flow.

Proof.

Corollary 48

Let f be an (s,t)-flow and let A be an (s,t)-cut, such that

$$\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A).$$

Then f is a maximum flow.

Proof.

Suppose that there is a flow f' with larger value. Then

□

Corollary 48

Let f be an (s,t)-flow and let A be an (s,t)-cut, such that

$$\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A).$$

Then f is a maximum flow.

Proof.

Suppose that there is a flow f' with larger value. Then

$$\mathrm{cap}(A, V \setminus A) < \mathrm{val}(f')$$

□

Corollary 48

Let f be an (s,t)-flow and let A be an (s,t)-cut, such that

$$\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A).$$

Then f is a maximum flow.

Proof.

Suppose that there is a flow f' with larger value. Then

$$\begin{aligned}\mathrm{cap}(A, V \setminus A) &< \mathrm{val}(f') \\ &= \underbrace{\sum_{e \in \mathrm{out}(A)} f'(e)}_{\leq\, \mathrm{cap}(A,\, V-A)} - \sum_{e \in \mathrm{into}(A)} f'(e)\end{aligned}$$

□

Corollary 48

Let f be an (s,t)-flow and let A be an (s,t)-cut, such that

$$\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A).$$

Then f is a maximum flow.

Proof.

Suppose that there is a flow f' with larger value. Then

$$\begin{aligned}
\mathrm{cap}(A, V \setminus A) &< \mathrm{val}(f') \\
&= \sum_{e \in \mathrm{out}(A)} f'(e) - \sum_{e \in \mathrm{into}(A)} f'(e) \\
&\leq \sum_{e \in \mathrm{out}(A)} f'(e)
\end{aligned}$$

□

Corollary 48

Let f be an (s,t)-flow and let A be an (s,t)-cut, such that

$$\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A).$$

Then f is a maximum flow.

Proof.

Suppose that there is a flow f' with larger value. Then

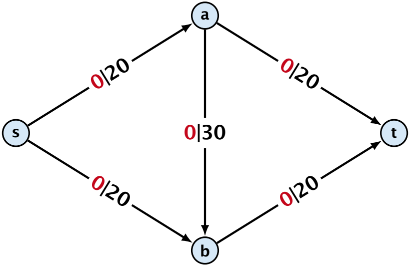
$$\begin{aligned}
\mathrm{cap}(A, V \setminus A) &< \mathrm{val}(f') \\
&= \sum_{e \in \mathrm{out}(A)} f'(e) - \sum_{e \in \mathrm{into}(A)} f'(e) \\
&\le \sum_{e \in \mathrm{out}(A)} f'(e) \\
&\le \mathrm{cap}(A, V \setminus A)
\end{aligned}$$

□

11 Augmenting Path Algorithms

Greedy-algorithm:

- start with $f(e) = 0$ everywhere
- find an s-t path with $f(e) < c(e)$ on every edge
- augment flow along the path
- repeat as long as possible

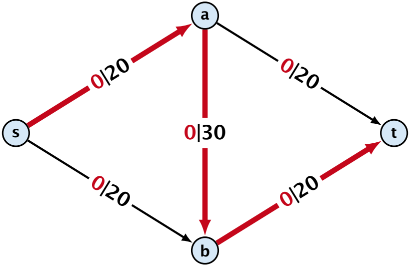


flow value: 0

11 Augmenting Path Algorithms

Greedy-algorithm:

- start with $f(e) = 0$ everywhere
- find an s-t path with $f(e) < c(e)$ on every edge
- augment flow along the path
- repeat as long as possible

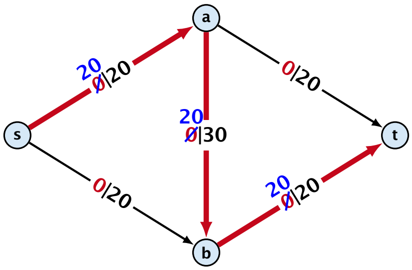


flow value: 0

11 Augmenting Path Algorithms

Greedy-algorithm:

- start with $f(e) = 0$ everywhere
- find an s-t path with $f(e) < c(e)$ on every edge
- augment flow along the path
- repeat as long as possible

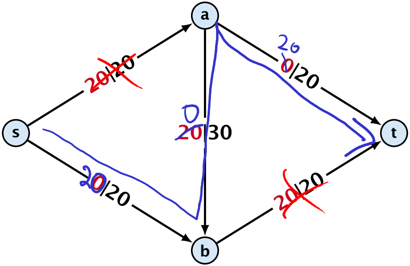


flow value: 0

11 Augmenting Path Algorithms

Greedy-algorithm:

- start with $f(e) = 0$ everywhere
- find an s-t path with $f(e) < c(e)$ on every edge
- augment flow along the path
- repeat as long as possible



flow value: 20

The Residual Graph

From the graph $G = (V, E, c)$ and the current flow f we construct an auxiliary graph $G_f = (V, E_f, c_f)$ (the residual graph):

The Residual Graph

From the graph $G = (V, E, c)$ and the current flow f we construct an auxiliary graph $G_f = (V, E_f, c_f)$ (the residual graph):

- Suppose the original graph has edges $e_1 = (u, v)$, and $e_2 = (v, u)$ between u and v.

The Residual Graph

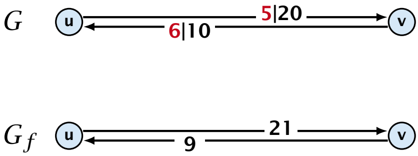
From the graph $G = (V, E, c)$ and the current flow f we construct an auxiliary graph $G_f = (V, E_f, c_f)$ (the residual graph):

- Suppose the original graph has edges $e_1 = (u, v)$, and $e_2 = (v, u)$ between u and v.
- G_f has edge e_1' with capacity $\max\{0, c(e_1) - f(e_1) + f(e_2)\}$ and e_2' with with capacity $\max\{0, c(e_2) - f(e_2) + f(e_1)\}$.

The Residual Graph

From the graph $G = (V, E, c)$ and the current flow f we construct an auxiliary graph $G_f = (V, E_f, c_f)$ (the residual graph):

- Suppose the original graph has edges $e_1 = (u, v)$, and $e_2 = (v, u)$ between u and v.
- G_f has edge e_1' with capacity $\max\{0, c(e_1) - f(e_1) + f(e_2)\}$ and e_2' with with capacity $\max\{0, c(e_2) - f(e_2) + f(e_1)\}$.



Augmenting Path Algorithm

Definition 49
An augmenting path with respect to flow f, is a path from s to t in the auxiliary graph G_f that contains only edges with non-zero capacity.

Algorithm 17 FordFulkerson$(G = (V, E, c))$

Initialize $f(e) \leftarrow 0$ for all edges.
while $\exists$ augmenting path p in G_f **do**
    augment as much flow along p as possible.

Augmenting Path Algorithm

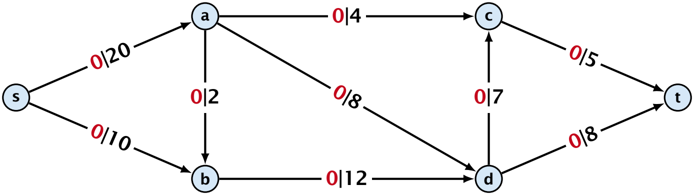
Definition 49

An augmenting path with respect to flow f, is a path from s to t in the auxiliary graph G_f that contains only edges with non-zero capacity.

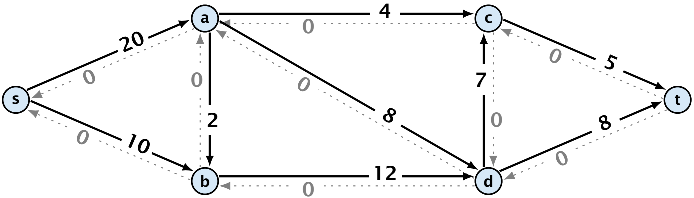
Algorithm 17 FordFulkerson$(G = (V, E, c))$

1: Initialize $f(e) \leftarrow 0$ for all edges.
2: **while** $\exists$ augmenting path p in G_f **do**
3:     augment as much flow along p as possible.

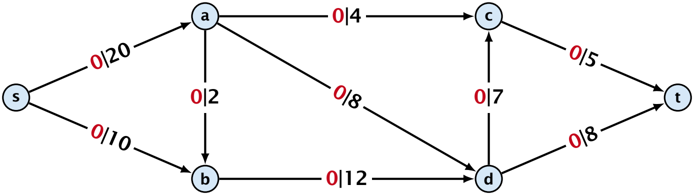
Augmenting Paths



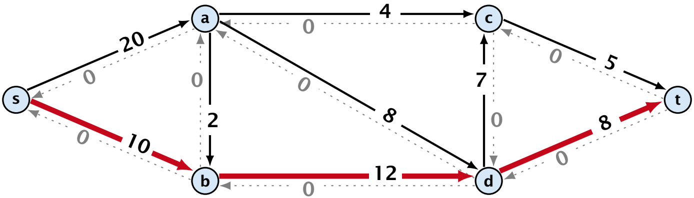
flow value: 0



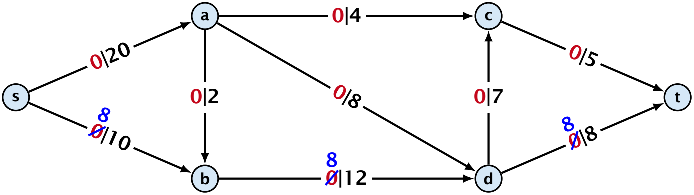
Augmenting Paths



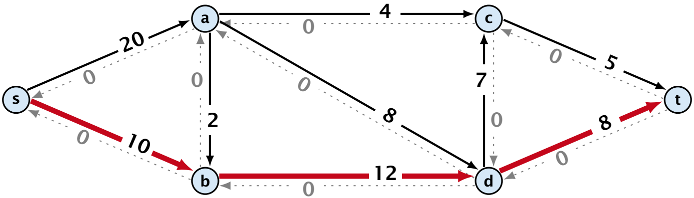
flow value: 0



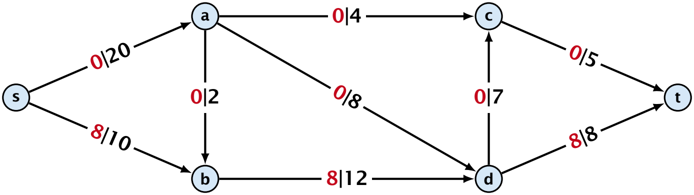
Augmenting Paths



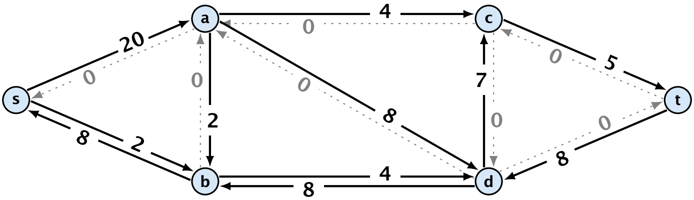
flow value: 0



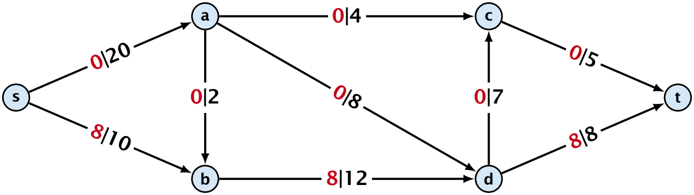
Augmenting Paths



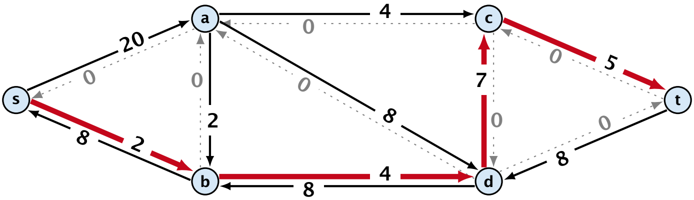
flow value: 8



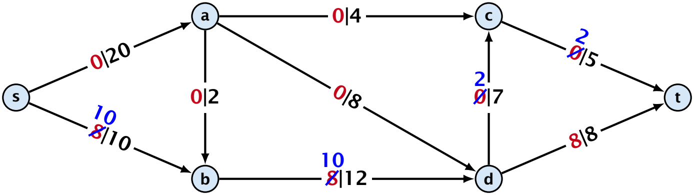
Augmenting Paths



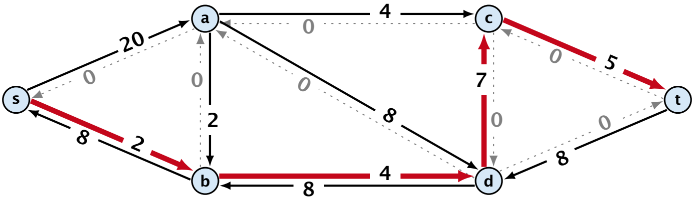
flow value: 8



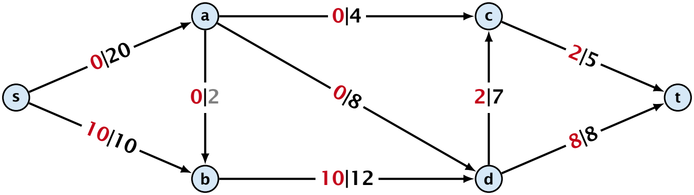
Augmenting Paths



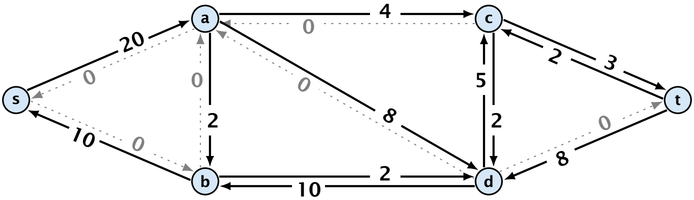
flow value: 8



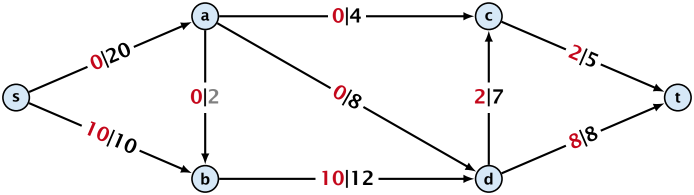
Augmenting Paths



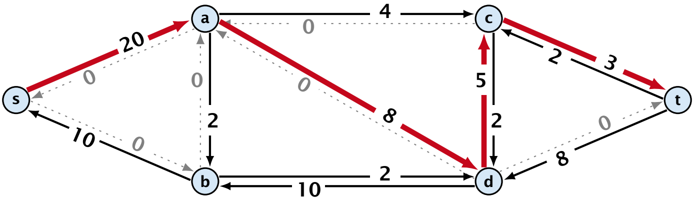
flow value: 10



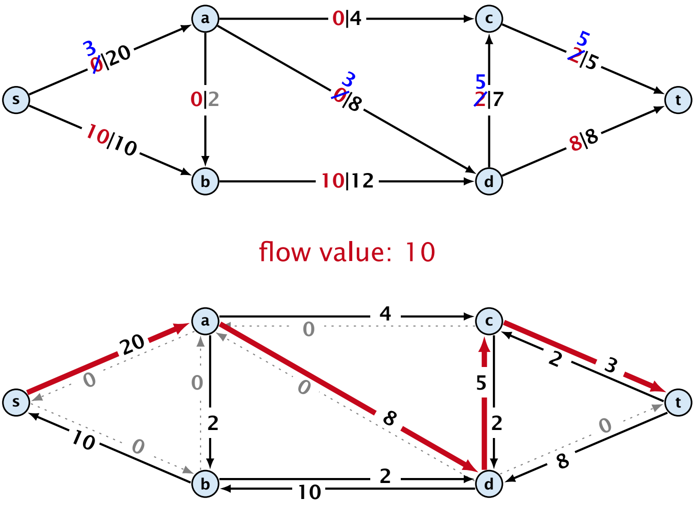
Augmenting Paths



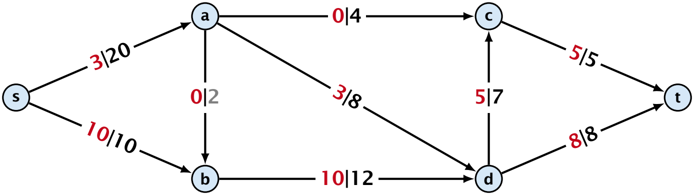
flow value: 10



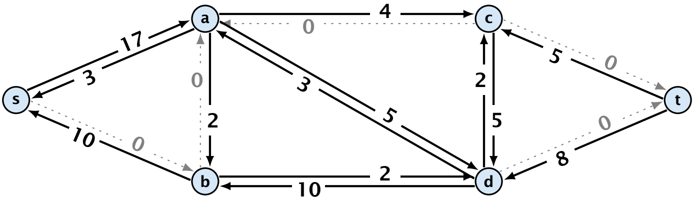
Augmenting Paths



Augmenting Paths



flow value: 13



Augmenting Path Algorithm

Theorem 50

*A flow f is a maximum flow **iff** there are no augmenting paths.*

Theorem 51

The value of a maximum flow is equal to the value of a minimum cut.

Proof.

Let f be a flow. The following are equivalent:

1. There exists a cut A such that $\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A)$.
2. Flow f is a maximum flow.
3. There is no augmenting path w.r.t. f.

□

Augmenting Path Algorithm

Theorem 50

A flow f is a maximum flow ***iff*** *there are no augmenting paths.*

Theorem 51

The value of a maximum flow is equal to the value of a minimum cut.

Proof.

Let f be a flow. The following are equivalent:

1. There exists a cut A such that $\text{val}(f) = \text{cap}(A, V \setminus A)$.
2. Flow f is a maximum flow.
3. There is no augmenting path w.r.t. f.

□

Augmenting Path Algorithm

Theorem 50

*A flow f is a maximum flow **iff** there are no augmenting paths.*

Theorem 51

The value of a maximum flow is equal to the value of a minimum cut.

Proof.

Let f be a flow. The following are equivalent:

1. There exists a cut A such that $\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A)$.
2. Flow f is a maximum flow.
3. There is no augmenting path w.r.t. f.

□

Augmenting Path Algorithm

Theorem 50

*A flow f is a maximum flow **iff** there are no augmenting paths.*

Theorem 51

The value of a maximum flow is equal to the value of a minimum cut.

Proof.

Let f be a flow. The following are equivalent:

1. There exists a cut A such that $\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A)$.
2. Flow f is a maximum flow.
3. There is no augmenting path w.r.t. f.

□

Augmenting Path Algorithm

Theorem 50

*A flow f is a maximum flow **iff** there are no augmenting paths.*

Theorem 51

The value of a maximum flow is equal to the value of a minimum cut.

Proof.

Let f be a flow. The following are equivalent:

1. There exists a cut A such that $\mathrm{val}(f) = \mathrm{cap}(A, V \setminus A)$.
2. Flow f is a maximum flow.
3. There is no augmenting path w.r.t. f.

□

Augmenting Path Algorithm

Theorem 50

*A flow f is a maximum flow **iff** there are no augmenting paths.*

Theorem 51

The value of a maximum flow is equal to the value of a minimum cut.

Proof.

Let f be a flow. The following are equivalent:

1. There exists a cut A such that $\text{val}(f) = \text{cap}(A, V \setminus A)$.
2. Flow f is a maximum flow.
3. There is no augmenting path w.r.t. f.

□

Augmenting Path Algorithm

1\. $\Longrightarrow$ 2.

This we already showed.

2\. $\Longrightarrow$ 3.

If there were an augmenting path, we could improve the flow. Contradiction.

3\. $\Longrightarrow$ 1.

- Let f be a flow with no augmenting paths.
- Let A be the set of vertices reachable from s in the residual graph along non-zero capacity edges.
- Since there is no augmenting path we have $s \in A$ and $t \notin A$.

Augmenting Path Algorithm

1\. $\Longrightarrow$ 2.
This we already showed.

2\. $\Longrightarrow$ 3.
If there were an augmenting path, we could improve the flow. Contradiction.

3\. $\Longrightarrow$ 1.

- Let f be a flow with no augmenting paths.
- Let A be the set of vertices reachable from s in the residual graph along non-zero capacity edges.
- Since there is no augmenting path we have $s \in A$ and $t \notin A$.

Augmenting Path Algorithm

1. $\Longrightarrow$ 2.

This we already showed.

2. $\Longrightarrow$ 3.

If there were an augmenting path, we could improve the flow. Contradiction.

3. $\Longrightarrow$ 1.

- Let f be a flow with no augmenting paths.
- Let A be the set of vertices reachable from s in the residual graph along non-zero capacity edges.
- Since there is no augmenting path we have $s \in A$ and $t \notin A$.

Augmenting Path Algorithm

1\. $\Longrightarrow$ 2.

This we already showed.

2\. $\Longrightarrow$ 3.

If there were an augmenting path, we could improve the flow. Contradiction.

3\. $\Longrightarrow$ 1.

- Let f be a flow with no augmenting paths.
- Let A be the set of vertices reachable from s in the residual graph along non-zero capacity edges.
- Since there is no augmenting path we have $s \in A$ and $t \notin A$.

Augmenting Path Algorithm

1\. $\Longrightarrow$ 2.

This we already showed.

2\. $\Longrightarrow$ 3.

If there were an augmenting path, we could improve the flow. Contradiction.

3\. $\Longrightarrow$ 1.

- Let f be a flow with no augmenting paths.
- Let A be the set of vertices reachable from s in the residual graph along non-zero capacity edges.
- Since there is no augmenting path we have $s \in A$ and $t \notin A$.

Augmenting Path Algorithm

1. $\Longrightarrow$ 2.

This we already showed.

2. $\Longrightarrow$ 3.

If there were an augmenting path, we could improve the flow. Contradiction.

3. $\Longrightarrow$ 1.

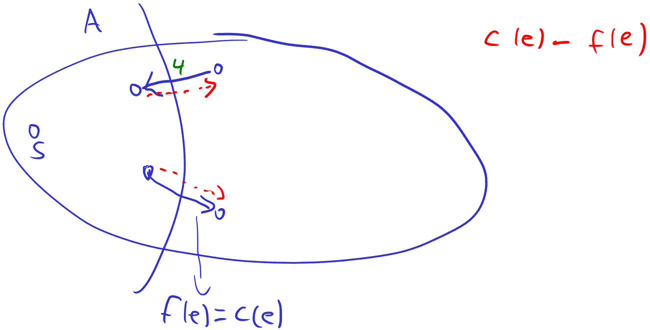
- Let f be a flow with no augmenting paths.
- Let A be the set of vertices reachable from s in the residual graph along non-zero capacity edges.
- Since there is no augmenting path we have $s \in A$ and $t \notin A$.

Augmenting Path Algorithm

$\mathrm{val}(f)$

Augmenting Path Algorithm

$$\mathrm{val}(f) = \sum_{e \in \mathrm{out}(A)} f(e) - \sum_{e \in \mathrm{into}(A)} f(e)$$



Augmenting Path Algorithm

$$\begin{aligned}\mathrm{val}(f) &= \sum_{e\in\mathrm{out}(A)} f(e) - \sum_{e\in\mathrm{into}(A)} f(e)\\ &= \sum_{e\in\mathrm{out}(A)} c(e)\end{aligned}$$

Augmenting Path Algorithm

$$\begin{aligned}\mathrm{val}(f) &= \sum_{e\in\mathrm{out}(A)} f(e) - \sum_{e\in\mathrm{into}(A)} f(e)\\ &= \sum_{e\in\mathrm{out}(A)} c(e)\\ &= \mathrm{cap}(A, V\setminus A)\end{aligned}$$

Augmenting Path Algorithm

$$\begin{aligned}\mathrm{val}(f) &= \sum_{e \in \mathrm{out}(A)} f(e) - \sum_{e \in \mathrm{into}(A)} f(e) \\ &= \sum_{e \in \mathrm{out}(A)} c(e) \\ &= \mathrm{cap}(A, V \setminus A)\end{aligned}$$

This finishes the proof.

Here the first equality uses the flow value lemma, and the second exploits the fact that the flow along incoming edges must be 0 as the residual graph does not have edges leaving A.

Analysis

Assumption:
All capacities are integers between 1 and C.

Invariant:
Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains integral troughout the algorithm.

Analysis

Assumption:
All capacities are integers between 1 and C.

Invariant:
Every flow value $f(e)$ and every residual capacity $c_f(e)$ remains integral troughout the algorithm.

Lemma 52

The algorithm terminates in at most $\text{val}(f^*) \le nC$ *iterations, where* f^* *denotes the maximum flow. Each iteration can be implemented in time* $\mathcal{O}(m)$. *This gives a total running time of* $\mathcal{O}(nmC)$.

Theorem 53

If all capacities are integers, then there exists a maximum flow for which every flow value $f(e)$ *is integral.*

Lemma 52

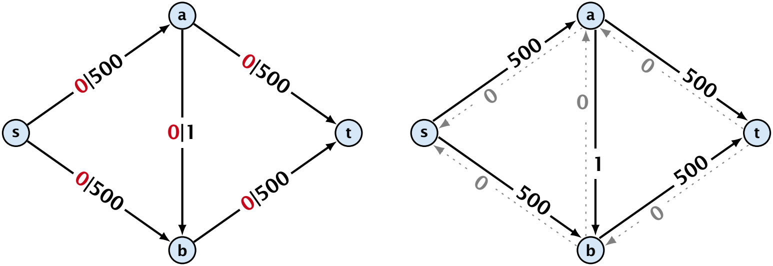
The algorithm terminates in at most $\mathrm{val}(f^*) \le nC$ *iterations, where* f^* *denotes the maximum flow. Each iteration can be implemented in time* $\mathcal{O}(m)$*. This gives a total running time of* $\mathcal{O}(nmC)$*.*

Theorem 53

If all capacities are integers, then there exists a maximum flow for which every flow value $f(e)$ *is integral.*

A Bad Input

Problem: The running time may not be polynomial



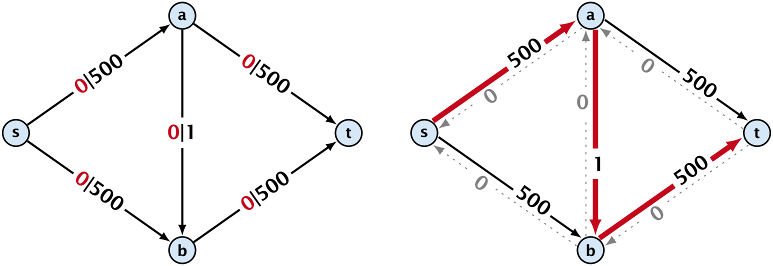
flow value: 0

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



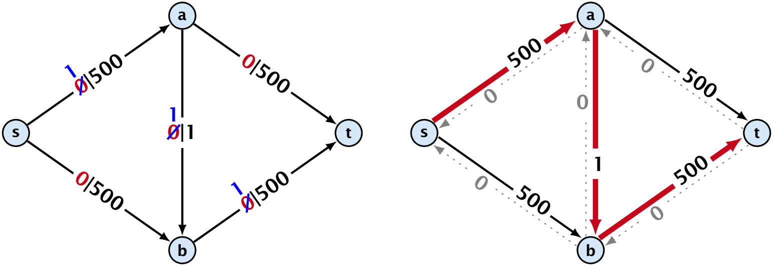
flow value: 0

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



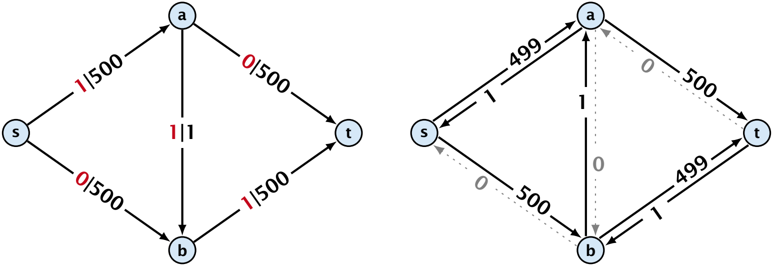
flow value: 0

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



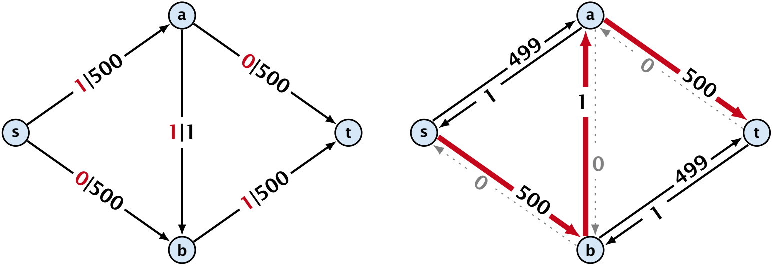
flow value: 1

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



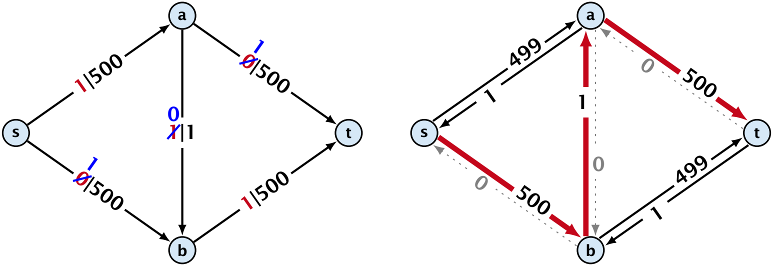
flow value: 1

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



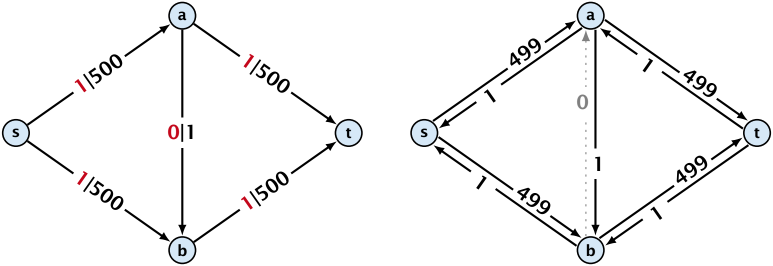
flow value: 1

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial

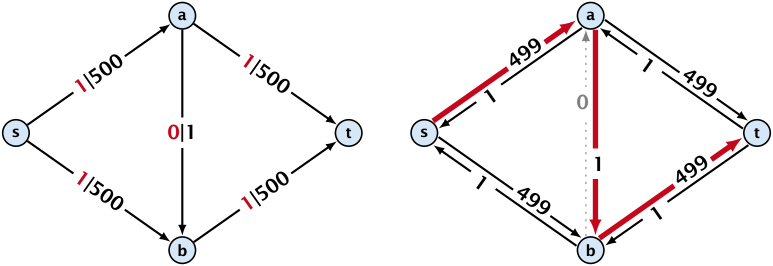


flow value: 2

Question:
Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



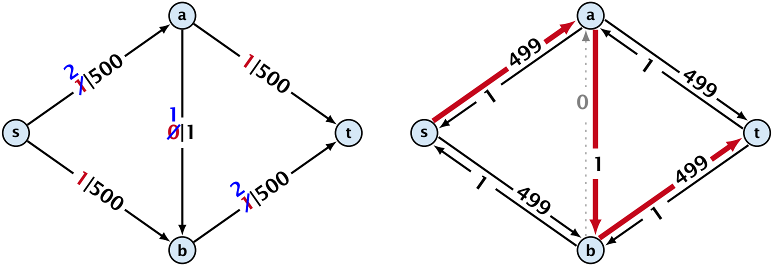
flow value: 2

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



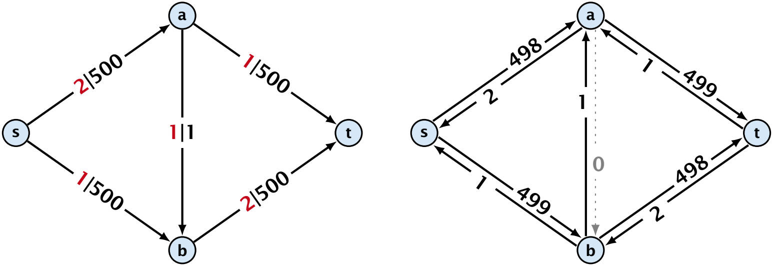
flow value: 2

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



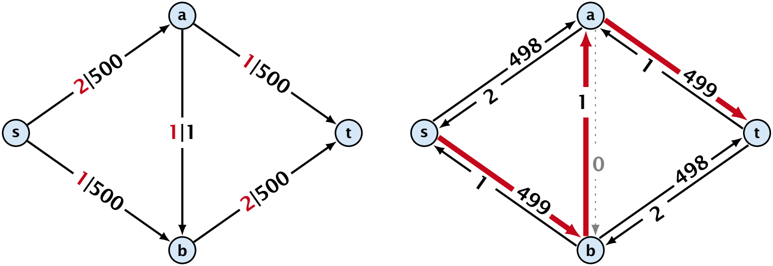
flow value: 3

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



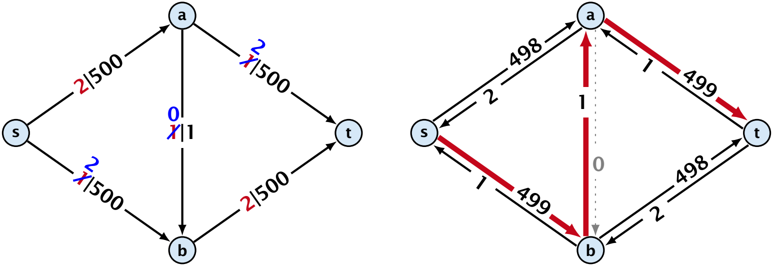
flow value: 3

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



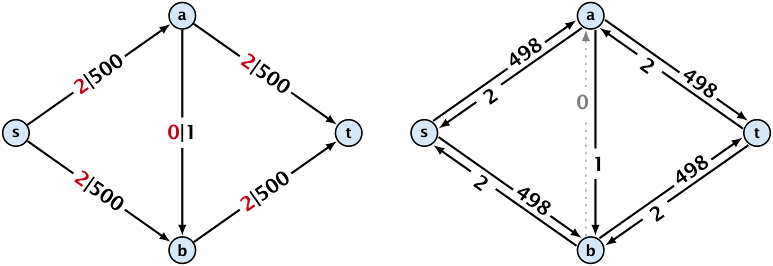
flow value: 3

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial

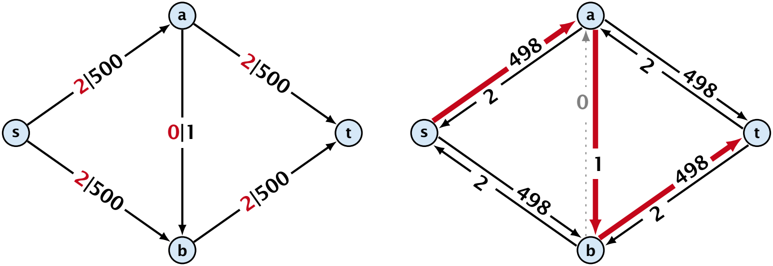


flow value: 4

Question:
Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



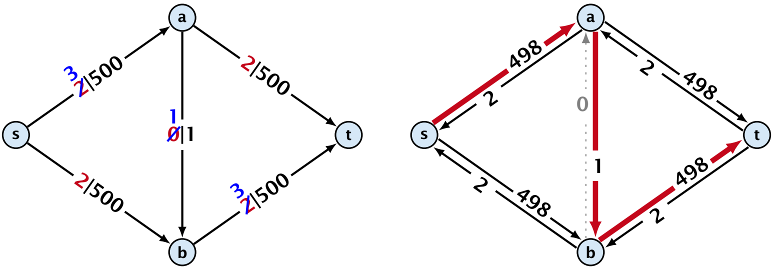
flow value: 4

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



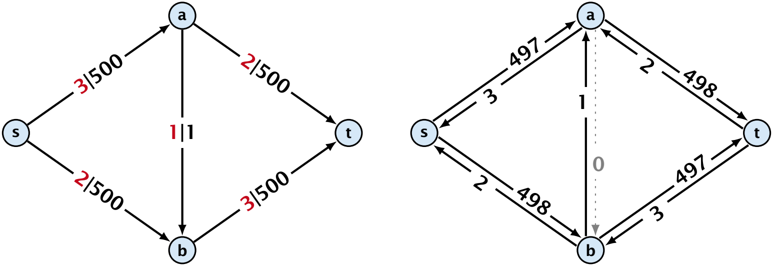
flow value: 4

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



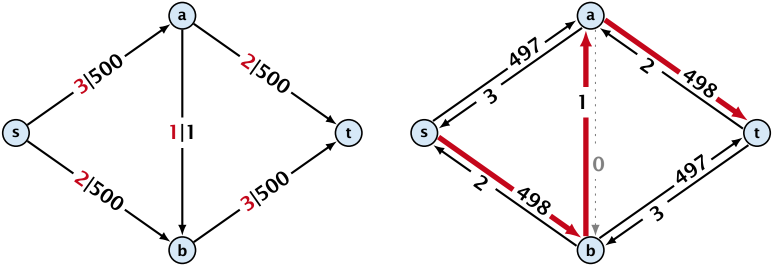
flow value: 5

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



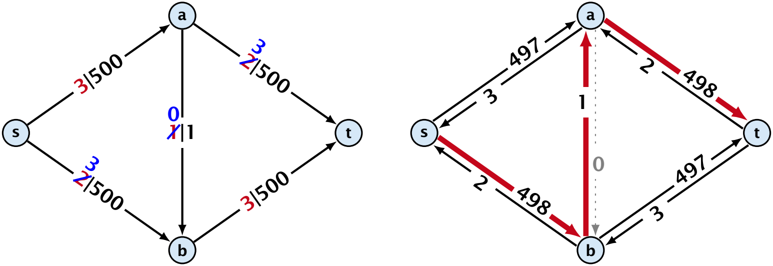
flow value: 5

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



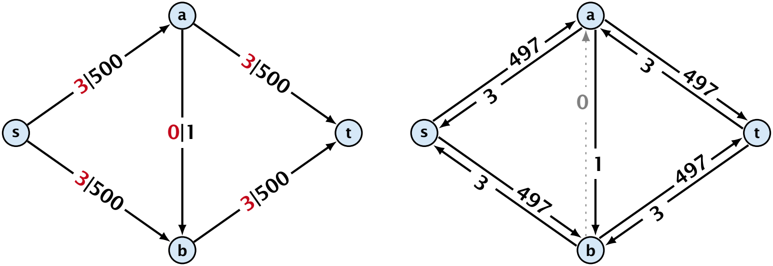
flow value: 5

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial



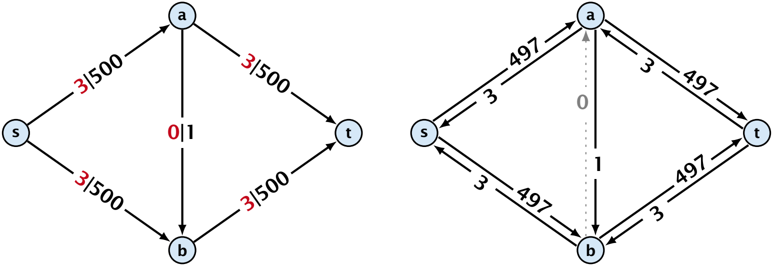
flow value: 6

Question:

Can we tweak the algorithm so that the running time is polynomial in the input length?

A Bad Input

Problem: The running time may not be polynomial

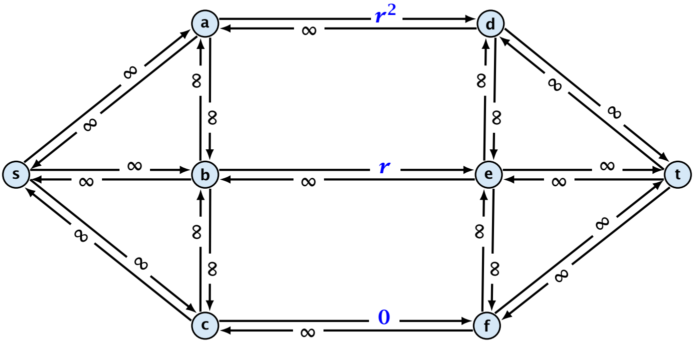


flow value: 6

Question:
Can we tweak the algorithm so that the running time is polynomial in the input length?

A Pathological Input

Let $r = \frac{1}{2}(\sqrt{5} - 1)$. Then $r^{n+2} = r^n - r^{n+1}$.



flow value: 0

Running time may be infinite!!!