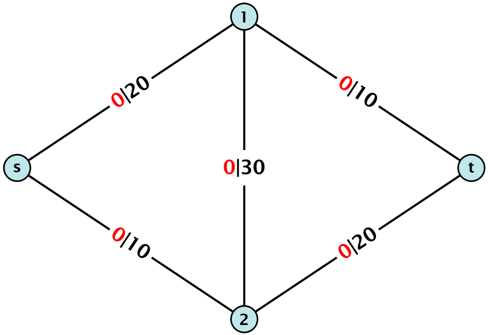
12 Augmenting Path Algorithms

Greedy-algorithm:

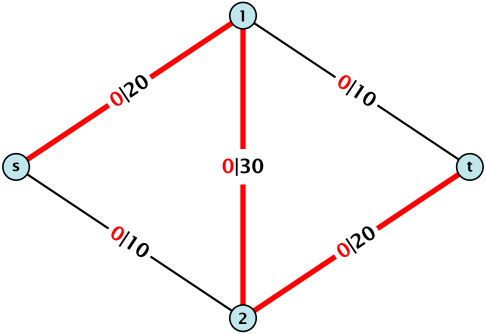
- start with $f(e) = 0$ everywhere
- find an s-t path with $f(e) < c(e)$ on every edge
- augment flow along the path
- repeat as long as possible



12 Augmenting Path Algorithms

Greedy-algorithm:

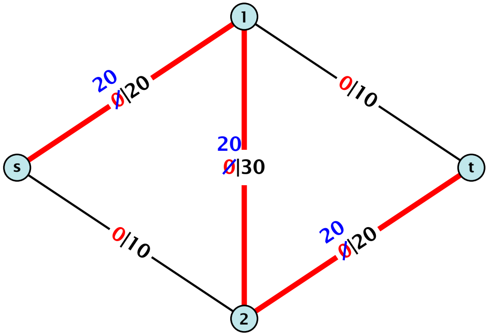
- start with $f(e) = 0$ everywhere
- find an s-t path with $f(e) < c(e)$ on every edge
- augment flow along the path
- repeat as long as possible



12 Augmenting Path Algorithms

Greedy-algorithm:

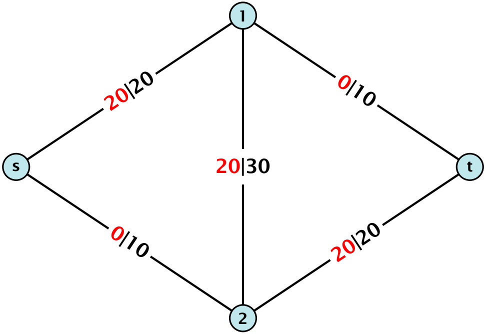
- start with $f(e) = 0$ everywhere
- find an s-t path with $f(e) < c(e)$ on every edge
- augment flow along the path
- repeat as long as possible



12 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



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

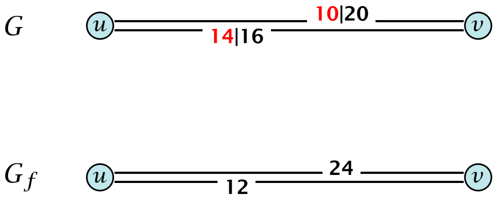
From the graph $G = (V, E, c)$ and the current flow f we construct an auxiliary graph $G_f = (V, E_f, c_f)$ (the residual graph):

- 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 50

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

Algorithm 45 FordFulkerson($G = (V, E, c)$)

```
Initialize f(e) ← 0 for all edges.
while ∃ augmenting path p in G_f do
    augment as much flow along p as possible.
```

Augmenting Path Algorithm

Definition 50

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

Algorithm 45 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 Path Algorithm

Theorem 51

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

Theorem 52

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

$\square$

Augmenting Path Algorithm

Theorem 51

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

Theorem 52

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

∎

Augmenting Path Algorithm

Theorem 51

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

Theorem 52

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

Augmenting Path Algorithm

Theorem 51

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

Theorem 52

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

□

Augmenting Path Algorithm

Theorem 51

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

Theorem 52

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

□

Augmenting Path Algorithm

Theorem 51

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

Theorem 52

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, B such that $\mathrm{val}(f) = \mathrm{cap}(A, B)$.
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\. $\Rightarrow$ 2.

This we already showed.

2\. $\Rightarrow$ 3.

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

3\. $\Rightarrow$ 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. $\Rightarrow$ 2.

This we already showed.

2. $\Rightarrow$ 3.

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

3. $\Rightarrow$ 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

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

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 53

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 54

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

Lemma 53

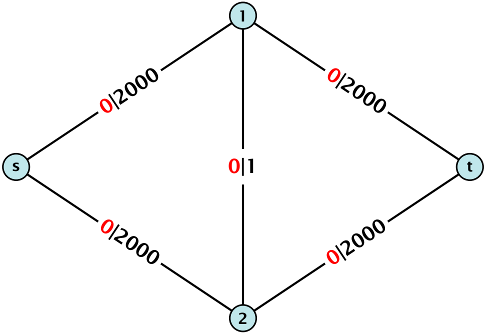
The algorithm terminates in at most $\mathrm{val}(f^*) \le nC$ *iterations, where* f^* *denotes the maximum flow. Each iteration can be implemented in time* $\mathcal{O}(m)$. *This gives a total running time of* $\mathcal{O}(nmC)$.

Theorem 54

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

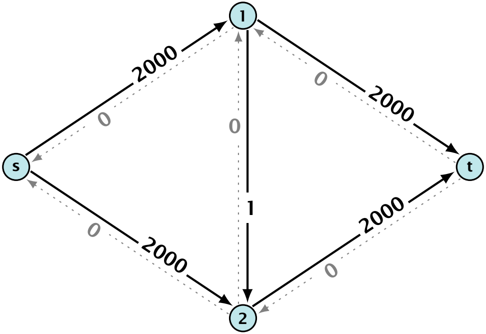
A bad input

Problem: The running time may not be polynomial.



A bad input

Problem: The running time may not be polynomial.

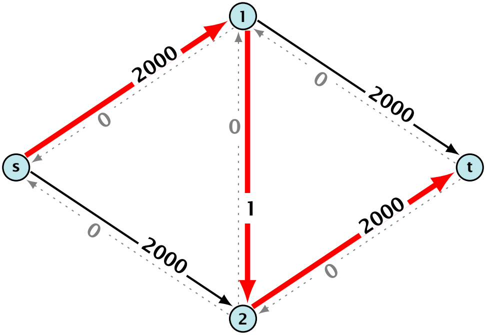


Question:

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.

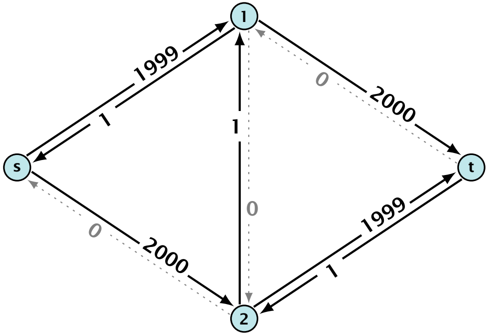


Question:

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.

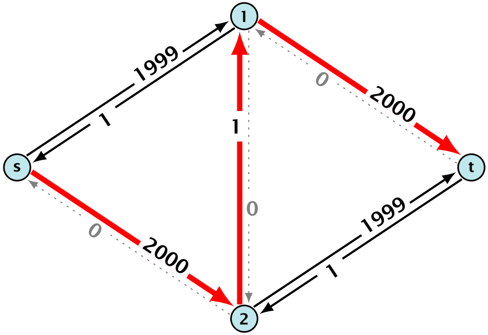


Question:

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.

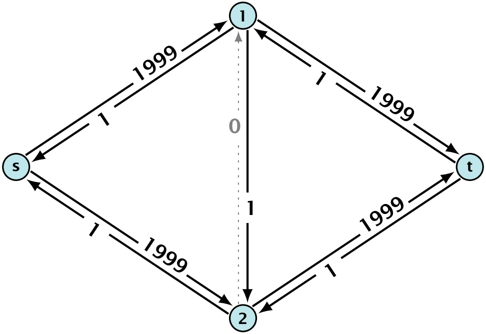


Question:

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.

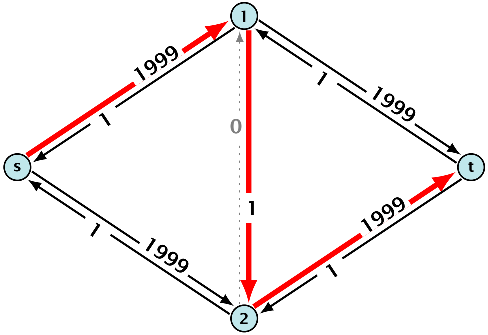


Question:

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.

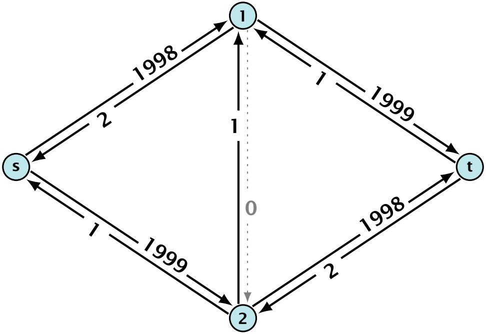


Question:

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.

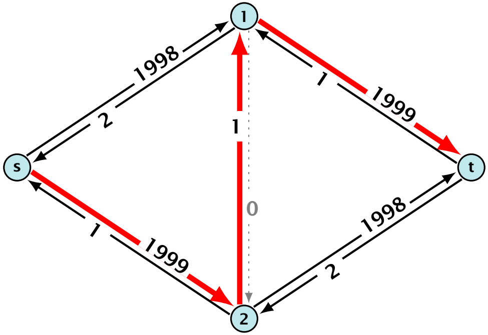


Question:

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.

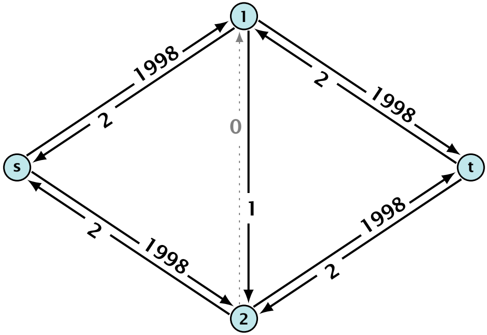


Question:

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.

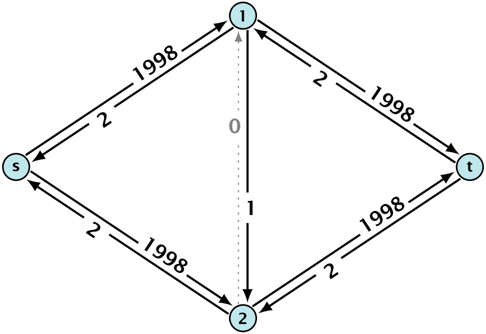


Question:

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.

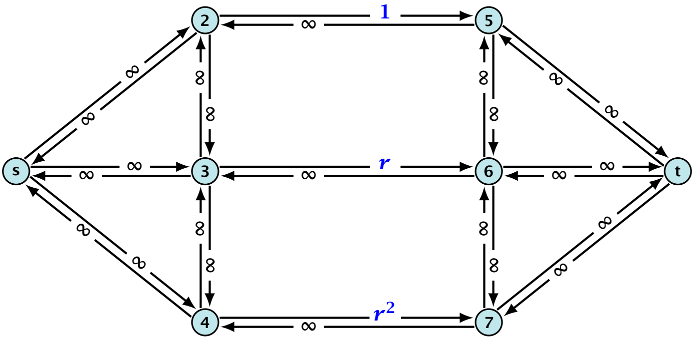


Question:

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

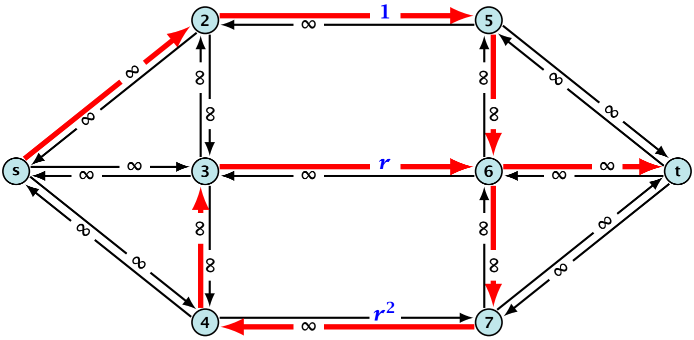
A Pathological Input

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



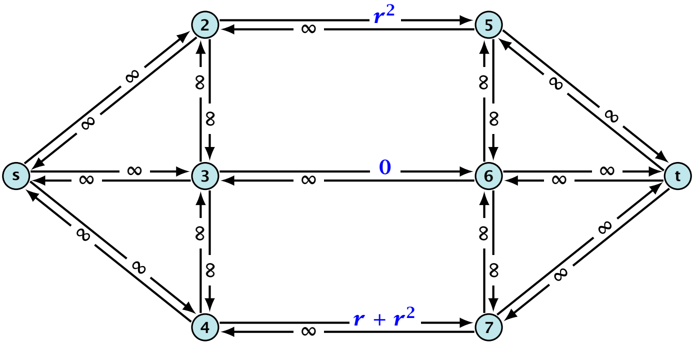
A Pathological Input

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



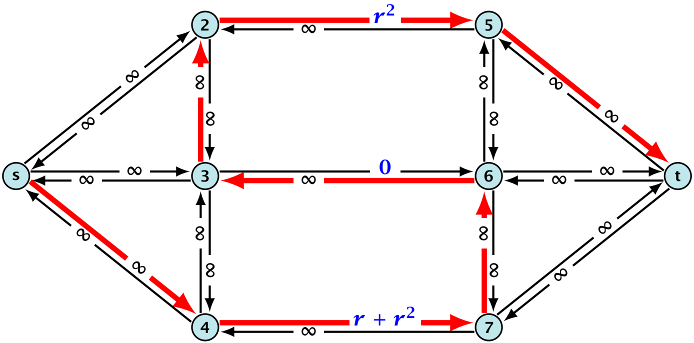
A Pathological Input

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



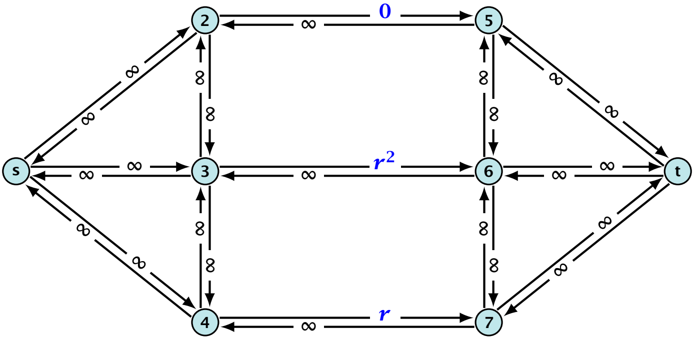
A Pathological Input

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



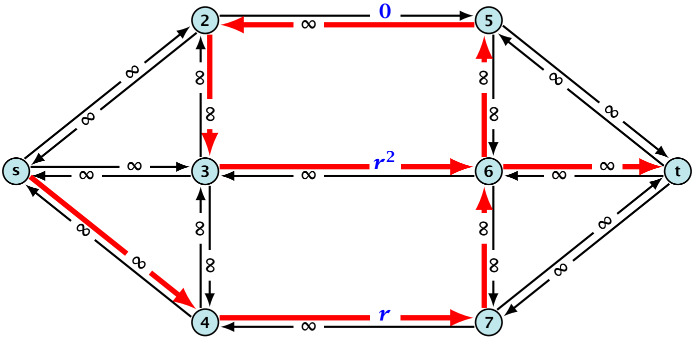
A Pathological Input

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



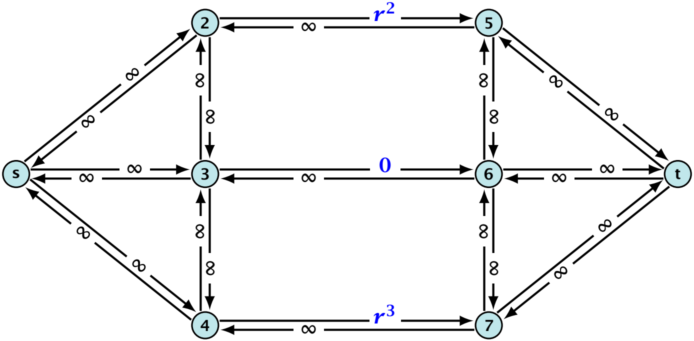
A Pathological Input

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



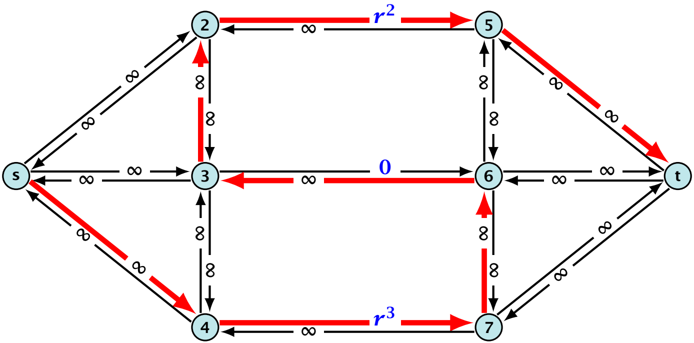
A Pathological Input

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



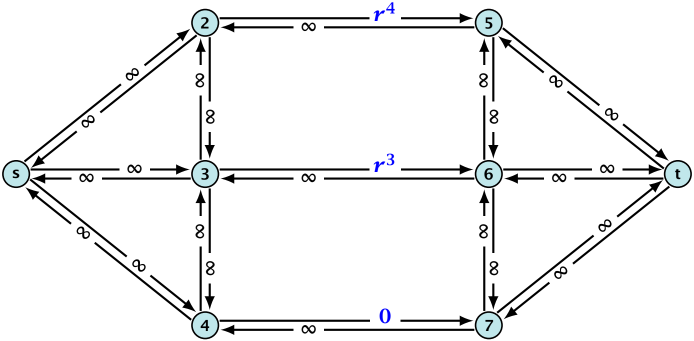
A Pathological Input

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



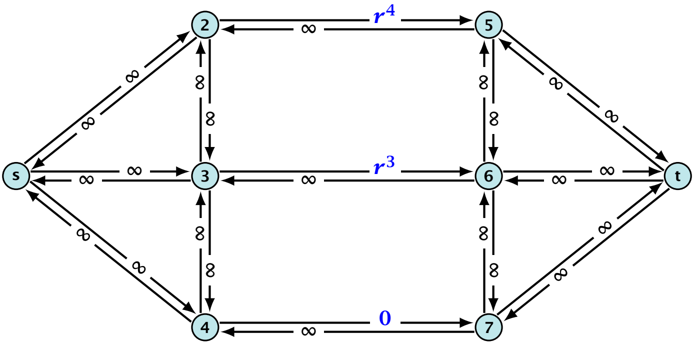
A Pathological Input

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



A Pathological Input

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



Running time may be infinite!!!

How to choose augmenting paths?

How to choose augmenting paths?

- We need to find paths efficiently.

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

- Choose path with maximum bottleneck capacity.

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

- Choose path with maximum bottleneck capacity.
- Choose path with sufficiently large bottleneck capacity.

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

- Choose path with maximum bottleneck capacity.
- Choose path with sufficiently large bottleneck capacity.
- Choose the shortest augmenting path.

=[fill=DarkGreen,draw=DarkGreen]

Overview: Shortest Augmenting Paths

Lemma 55

The length of the shortest augmenting path never decreases.

Lemma 56

After at most $\mathcal{O}(m)$ augmentations, the length of the shortest augmenting path strictly increases.

Overview: Shortest Augmenting Paths

Lemma 55

The length of the shortest augmenting path never decreases.

Lemma 56

After at most $\mathcal{O}(m)$ augmentations, the length of the shortest augmenting path strictly increases.

Overview: Shortest Augmenting Paths

Lemma 55
The length of the shortest augmenting path never decreases.

Lemma 56
After at most $\mathcal{O}(m)$ augmentations, the length of the shortest augmenting path strictly increases.

Overview: Shortest Augmenting Paths

These two lemmas give the following theorem:

Theorem 57

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. This gives a running time of $\mathcal{O}(m^2n)$.

Proof.

- We can find the shortest augmenting paths in time $\mathcal{O}(m)$ via BFS.
- $\mathcal{O}(m)$ augmentations for paths of exactly $k < n$ edges.

□

Overview: Shortest Augmenting Paths

These two lemmas give the following theorem:

Theorem 57

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. This gives a running time of $\mathcal{O}(m^2n)$.

Proof.

- We can find the shortest augmenting paths in time $\mathcal{O}(m)$ via BFS.
- $\mathcal{O}(m)$ augmentations for paths of exactly $k < n$ edges.

□

Overview: Shortest Augmenting Paths

These two lemmas give the following theorem:

Theorem 57

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. This gives a running time of $\mathcal{O}(m^2n)$.

Proof.

- We can find the shortest augmenting paths in time $\mathcal{O}(m)$ via BFS.
- $\mathcal{O}(m)$ augmentations for paths of exactly $k < n$ edges.

□

Overview: Shortest Augmenting Paths

These two lemmas give the following theorem:

Theorem 57

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. This gives a running time of $\mathcal{O}(m^2 n)$.

Proof.

- We can find the shortest augmenting paths in time $\mathcal{O}(m)$ via BFS.
- $\mathcal{O}(m)$ augmentations for paths of exactly $k < n$ edges.

□

Shortest Augmenting Paths

Define the level $\ell(v)$ of a node as the length of the shortest s-v path in G_f.

Shortest Augmenting Paths

Define the level $\ell(v)$ of a node as the length of the shortest s-v path in G_f.

Let L_G denote the subgraph of the residual graph G_f that contains only those edges (u,v) with $\ell(v) = \ell(u) + 1$.

Shortest Augmenting Paths

Define the level $\ell(v)$ of a node as the length of the shortest s-v path in G_f.

Let L_G denote the subgraph of the residual graph G_f that contains only those edges (u,v) with $\ell(v) = \ell(u) + 1$.

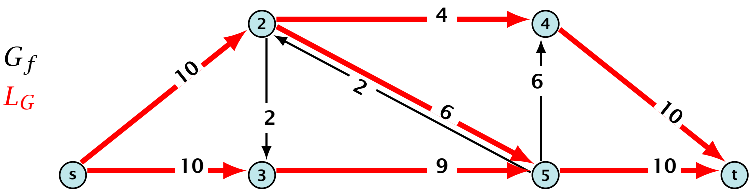
A path P is a shortest s-u path in G_f if it is a an s-u path in L_G.

Shortest Augmenting Paths

Define the level $\ell(v)$ of a node as the length of the shortest s-v path in G_f.

Let L_G denote the subgraph of the residual graph G_f that contains only those edges (u, v) with $\ell(v) = \ell(u) + 1$.

A path P is a shortest s-u path in G_f if it is a an s-u path in L_G.



Shortest Augmenting Path

First Lemma: The length of the shortest augmenting path never decreases.

Shortest Augmenting Path

First Lemma: The length of the shortest augmenting path never decreases.

- After an augmentation the following changes are done in G_f.

Shortest Augmenting Path

First Lemma: The length of the shortest augmenting path never decreases.

- After an augmentation the following changes are done in G_f.
- Some edges of the chosen path may be deleted (bottleneck edges).

Shortest Augmenting Path

First Lemma: The length of the shortest augmenting path never decreases.

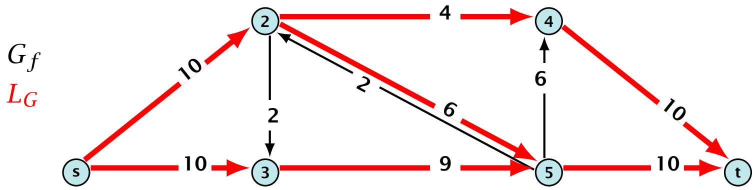
- After an augmentation the following changes are done in G_f.
- Some edges of the chosen path may be deleted (bottleneck edges).
- Back edges are added to all edges that don't have back edges so far.

Shortest Augmenting Path

First Lemma: The length of the shortest augmenting path never decreases.

- After an augmentation the following changes are done in G_f.
- Some edges of the chosen path may be deleted (bottleneck edges).
- Back edges are added to all edges that don't have back edges so far.

These changes cannot decrease the distance between s and t.



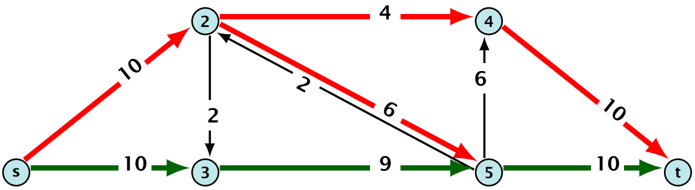
Shortest Augmenting Path

First Lemma: The length of the shortest augmenting path never decreases.

- After an augmentation the following changes are done in G_f.
- Some edges of the chosen path may be deleted (bottleneck edges).
- Back edges are added to all edges that don't have back edges so far.

These changes cannot decrease the distance between s and t.

G_f
L_G

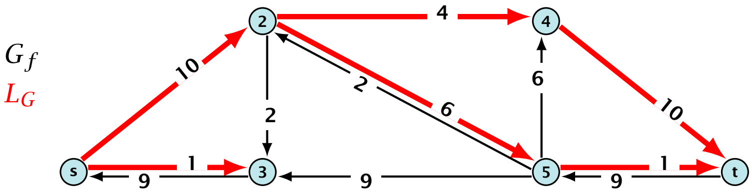


Shortest Augmenting Path

First Lemma: The length of the shortest augmenting path never decreases.

- After an augmentation the following changes are done in G_f.
- Some edges of the chosen path may be deleted (bottleneck edges).
- Back edges are added to all edges that don't have back edges so far.

These changes cannot decrease the distance between s and t.



Shortest Augmenting Path

Second Lemma: After at most m augmentations the length of the shortest augmenting path strictly increases.

Shortest Augmenting Path

Second Lemma: After at most m augmentations the length of the shortest augmenting path strictly increases.

Let E_L denote the set of edges in graph L_G at the beginning of a round when the distance between s and t is k.

Shortest Augmenting Path

Second Lemma: After at most m augmentations the length of the shortest augmenting path strictly increases.

Let E_L denote the set of edges in graph L_G at the beginning of a round when the distance between s and t is k.

An s-t path in G_f that does use edges not in E_L has length larger than k, even when considering edges added to G_f during the round.

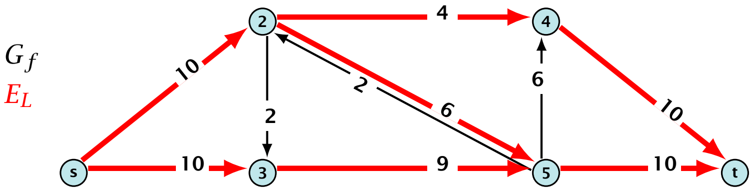
Shortest Augmenting Path

Second Lemma: After at most m augmentations the length of the shortest augmenting path strictly increases.

Let E_L denote the set of edges in graph L_G at the beginning of a round when the distance between s and t is k.

An s-t path in G_f that does use edges not in E_L has length larger than k, even when considering edges added to G_f during the round.

In each augmentation one edge is deleted from E_L.



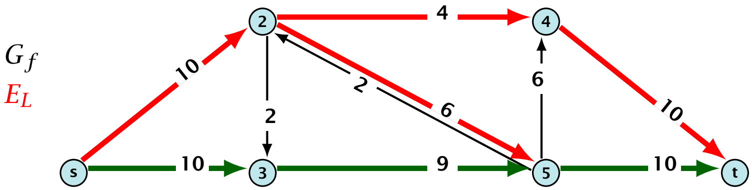
Shortest Augmenting Path

Second Lemma: After at most m augmentations the length of the shortest augmenting path strictly increases.

Let E_L denote the set of edges in graph L_G at the beginning of a round when the distance between s and t is k.

An s-t path in G_f that does use edges not in E_L has length larger than k, even when considering edges added to G_f during the round.

In each augmentation one edge is deleted from E_L.



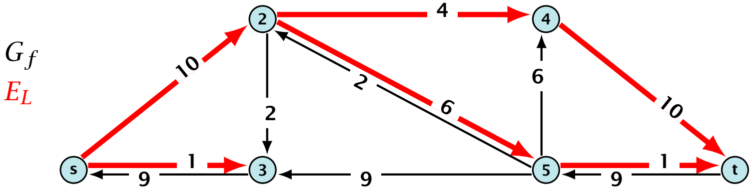
Shortest Augmenting Path

Second Lemma: After at most m augmentations the length of the shortest augmenting path strictly increases.

Let E_L denote the set of edges in graph L_G at the beginning of a round when the distance between s and t is k.

An s-t path in G_f that does use edges not in E_L has length larger than k, even when considering edges added to G_f during the round.

In each augmentation one edge is deleted from E_L.



Shortest Augmenting Paths

Theorem 58

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. Each augmentation can be performed in time $\mathcal{O}(m)$.

Theorem 59 (without proof)

There exist networks with $m = \Theta(n^2)$ that require $\mathcal{O}(mn)$ augmentations, when we restrict ourselves to only augment along shortest augmenting paths.

Note:

There always exists a set of m augmentations that gives a maximum flow.

Shortest Augmenting Paths

Theorem 58

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. Each augmentation can be performed in time $\mathcal{O}(m)$.

Theorem 59 (without proof)

There exist networks with $m = \Theta(n^2)$ that require $\mathcal{O}(mn)$ augmentations, when we restrict ourselves to only augment along shortest augmenting paths.

Note:

There always exists a set of m augmentations that gives a maximum flow.

Shortest Augmenting Paths

Theorem 58

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. Each augmentation can be performed in time $\mathcal{O}(m)$.

Theorem 59 (without proof)

There exist networks with $m = \Theta(n^2)$ that require $\mathcal{O}(mn)$ augmentations, when we restrict ourselves to only augment along shortest augmenting paths.

Note:

There always exists a set of m augmentations that gives a maximum flow.

Shortest Augmenting Paths

Theorem 58

The shortest augmenting path algorithm performs at most $\mathcal{O}(mn)$ augmentations. Each augmentation can be performed in time $\mathcal{O}(m)$.

Theorem 59 (without proof)

There exist networks with $m = \Theta(n^2)$ that require $\mathcal{O}(mn)$ augmentations, when we restrict ourselves to only augment along shortest augmenting paths.

Note:

There always exists a set of m augmentations that gives a maximum flow.

Shortest Augmenting Paths

When sticking to shortest augmenting paths we cannot improve (asymptotically) on the number of augmentations.

However, we can improve the running time to $\mathcal{O}(mn^2)$ by improving the running time for finding an augmenting path (currently we assume $\mathcal{O}(m)$ per augmentation for this).

Shortest Augmenting Paths

When sticking to shortest augmenting paths we cannot improve (asymptotically) on the number of augmentations.

However, we can improve the running time to $\mathcal{O}(mn^2)$ by improving the running time for finding an augmenting path (currently we assume $\mathcal{O}(m)$ per augmentation for this).

Shortest Augmenting Paths

We maintain a subset E_L of the edges of G_f with the guarantee that a shortest s-t path using only edges from E_L is a shortest augmenting path.

With each augmentation some edges are deleted from E_L.

When E_L does not contain an s-t path anymore the distance between s and t strictly increases.

Note that E_L is not the set of edges of the level graph but a subset of level-graph edges.

Shortest Augmenting Paths

We maintain a subset E_L of the edges of G_f with the guarantee that a shortest s-t path using only edges from E_L is a shortest augmenting path.

With each augmentation some edges are deleted from E_L.

When E_L does not contain an s-t path anymore the distance between s and t strictly increases.

Note that E_L is not the set of edges of the level graph but a subset of level-graph edges.

Shortest Augmenting Paths

We maintain a subset E_L of the edges of G_f with the guarantee that a shortest s-t path using only edges from E_L is a shortest augmenting path.

With each augmentation some edges are deleted from E_L.

When E_L does not contain an s-t path anymore the distance between s and t strictly increases.

Note that E_L is not the set of edges of the level graph but a subset of level-graph edges.

Shortest Augmenting Paths

We maintain a subset E_L of the edges of G_f with the guarantee that a shortest s-t path using only edges from E_L is a shortest augmenting path.

With each augmentation some edges are deleted from E_L.

When E_L does not contain an s-t path anymore the distance between s and t strictly increases.

Note that E_L is not the set of edges of the level graph but a subset of level-graph edges.

Suppose that the initial distance between s and t in G_f is k.

E_L is initialized as the level graph L_G.

Perform a DFS search to find a path from s to t using edges from E_L.

Either you find t after at most n steps, or you end at a node v that does not have any outgoing edges.

You can delete incoming edges of v from E_L.

Suppose that the initial distance between s and t in G_f is k.

E_L is initialized as the level graph L_G.

Perform a DFS search to find a path from s to t using edges from E_L.

Either you find t after at most n steps, or you end at a node v that does not have any outgoing edges.

You can delete incoming edges of v from E_L.

Suppose that the initial distance between s and t in G_f is k.

E_L is initialized as the level graph L_G.

Perform a DFS search to find a path from s to t using edges from E_L.

Either you find t after at most n steps, or you end at a node v that does not have any outgoing edges.

You can delete incoming edges of v from E_L.

Suppose that the initial distance between s and t in G_f is k.

E_L is initialized as the level graph L_G.

Perform a DFS search to find a path from s to t using edges from E_L.

Either you find t after at most n steps, or you end at a node v that does not have any outgoing edges.

You can delete incoming edges of v from E_L.

Suppose that the initial distance between s and t in G_f is k.

E_L is initialized as the level graph L_G.

Perform a DFS search to find a path from s to t using edges from E_L.

Either you find t after at most n steps, or you end at a node v that does not have any outgoing edges.

You can delete incoming edges of v from E_L.

Let a phase of the algorithm be defined by the time between two augmentations during which the distance between s and t strictly increases.

Initializing E_L for the phase takes time $\mathcal{O}(m)$.

The total cost for searching for augmenting paths during a phase is at most $\mathcal{O}(mn)$, since every search (successful (i.e., reaching t) or unsuccessful) decreases the number of edges in E_L and takes time $\mathcal{O}(n)$.

The total cost for performing an augmentation during a phase is only $\mathcal{O}(n)$. For every edge in the augmenting path one has to update the residual graph G_f and has to check whether the edge is still in E_L for the next search.

There are at most n phases. Hence, total cost is $\mathcal{O}(mn^2)$.

Let a phase of the algorithm be defined by the time between two augmentations during which the distance between s and t strictly increases.

Initializing E_L for the phase takes time $\mathcal{O}(m)$.

The total cost for searching for augmenting paths during a phase is at most $\mathcal{O}(mn)$, since every search (successful (i.e., reaching t) or unsuccessful) decreases the number of edges in E_L and takes time $\mathcal{O}(n)$.

The total cost for performing an augmentation during a phase is only $\mathcal{O}(n)$. For every edge in the augmenting path one has to update the residual graph G_f and has to check whether the edge is still in E_L for the next search.

There are at most n phases. Hence, total cost is $\mathcal{O}(mn^2)$.

Let a phase of the algorithm be defined by the time between two augmentations during which the distance between s and t strictly increases.

Initializing E_L for the phase takes time $\mathcal{O}(m)$.

The total cost for searching for augmenting paths during a phase is at most $\mathcal{O}(mn)$, since every search (successful (i.e., reaching t) or unsuccessful) decreases the number of edges in E_L and takes time $\mathcal{O}(n)$.

The total cost for performing an augmentation during a phase is only $\mathcal{O}(n)$. For every edge in the augmenting path one has to update the residual graph G_f and has to check whether the edge is still in E_L for the next search.

There are at most n phases. Hence, total cost is $\mathcal{O}(mn^2)$.

Let a phase of the algorithm be defined by the time between two augmentations during which the distance between s and t strictly increases.

Initializing E_L for the phase takes time $\mathcal{O}(m)$.

The total cost for searching for augmenting paths during a phase is at most $\mathcal{O}(mn)$, since every search (successful (i.e., reaching t) or unsuccessful) decreases the number of edges in E_L and takes time $\mathcal{O}(n)$.

The total cost for performing an augmentation during a phase is only $\mathcal{O}(n)$. For every edge in the augmenting path one has to update the residual graph G_f and has to check whether the edge is still in E_L for the next search.

There are at most n phases. Hence, total cost is $\mathcal{O}(mn^2)$.

Let a phase of the algorithm be defined by the time between two augmentations during which the distance between s and t strictly increases.

Initializing E_L for the phase takes time $\mathcal{O}(m)$.

The total cost for searching for augmenting paths during a phase is at most $\mathcal{O}(mn)$, since every search (successful (i.e., reaching t) or unsuccessful) decreases the number of edges in E_L and takes time $\mathcal{O}(n)$.

The total cost for performing an augmentation during a phase is only $\mathcal{O}(n)$. For every edge in the augmenting path one has to update the residual graph G_f and has to check whether the edge is still in E_L for the next search.

There are at most n phases. Hence, total cost is $\mathcal{O}(mn^2)$.

Let a phase of the algorithm be defined by the time between two augmentations during which the distance between s and t strictly increases.

Initializing E_L for the phase takes time $\mathcal{O}(m)$.

The total cost for searching for augmenting paths during a phase is at most $\mathcal{O}(mn)$, since every search (successful (i.e., reaching t) or unsuccessful) decreases the number of edges in E_L and takes time $\mathcal{O}(n)$.

The total cost for performing an augmentation during a phase is only $\mathcal{O}(n)$. For every edge in the augmenting path one has to update the residual graph G_f and has to check whether the edge is still in E_L for the next search.

There are at most n phases. Hence, total cost is $\mathcal{O}(mn^2)$.

How to choose augmenting paths?

- We need to find paths efficiently.

Several possibilities:

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

- Choose path with maximum bottleneck capacity.
- Choose path with sufficiently large bottleneck capacity.
- Choose the shortest augmenting path.

Capacity Scaling

Capacity Scaling

Intuition:

- Choosing a path with the highest bottleneck increases the flow as much as possible in a single step.

Capacity Scaling

Intuition:

- Choosing a path with the highest bottleneck increases the flow as much as possible in a single step.
- Don't worry about finding the exact bottleneck.

Capacity Scaling

Intuition:

- Choosing a path with the highest bottleneck increases the flow as much as possible in a single step.
- Don't worry about finding the exact bottleneck.
- Maintain scaling parameter Δ.

Capacity Scaling

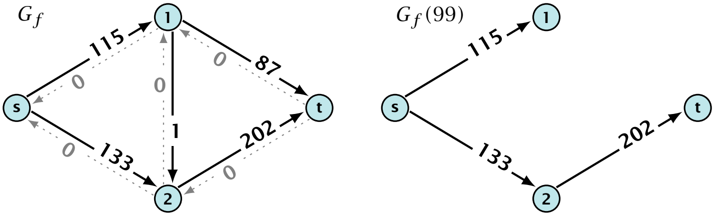
Intuition:

- Choosing a path with the highest bottleneck increases the flow as much as possible in a single step.
- Don't worry about finding the exact bottleneck.
- Maintain scaling parameter Δ.
- $G_f(\Delta)$ is a sub-graph of the residual graph G_f that contains only edges with capacity at least Δ.

Capacity Scaling

Intuition:

- Choosing a path with the highest bottleneck increases the flow as much as possible in a single step.
- Don't worry about finding the exact bottleneck.
- Maintain scaling parameter Δ.
- $G_f(\Delta)$ is a sub-graph of the residual graph G_f that contains only edges with capacity at least Δ.



Capacity Scaling

Algorithm 46 maxflow(G, s, t, c)

```
foreach e ∈ E do f_e ← 0;
Δ ← 2^⌈log₂ C⌉
while Δ ≥ 1 do
    G_f(Δ) ← Δ-residual graph
    while there is augmenting path P in G_f(Δ) do
        f ← augment(f, c, P)
        update(G_f(Δ))
    Δ ← Δ/2
return f
```

Capacity Scaling

Capacity Scaling

Assumption:

All capacities are integers between 1 and C.

Capacity Scaling

Assumption:
All capacities are integers between 1 and C.

Invariant:
All flows and capacities are/remain integral throughout the algorithm.

Capacity Scaling

Assumption:
All capacities are integers between 1 and C.

Invariant:
All flows and capacities are/remain integral throughout the algorithm.

Correctness:
The algorithm computes a maxflow:

- because of integrality we have $G_f(1) = G_f$

Capacity Scaling

Assumption:
All capacities are integers between 1 and C.

Invariant:
All flows and capacities are/remain integral throughout the algorithm.

Correctness:
The algorithm computes a maxflow:

- because of integrality we have $G_f(1) = G_f$
- therefore after the last phase there are no augmenting paths anymore

Capacity Scaling

Assumption:
All capacities are integers between 1 and C.

Invariant:
All flows and capacities are/remain integral throughout the algorithm.

Correctness:
The algorithm computes a maxflow:

- because of integrality we have $G_f(1) = G_f$
- therefore after the last phase there are no augmenting paths anymore
- this means we have a maximum flow.

Capacity Scaling

Capacity Scaling

Lemma 60

There are $\lceil \log C \rceil$ *iterations over* Δ*.*

Proof: obvious.

Capacity Scaling

Lemma 60

There are $\lceil \log C \rceil$ *iterations over* Δ.

Proof: obvious.

Lemma 61

Let f *be the flow at the end of a* Δ*-phase. Then the maximum flow is smaller than* $\mathrm{val}(f) + 2m\Delta$.

Proof: less obvious, but simple:

Capacity Scaling

Lemma 60

There are $\lceil \log C \rceil$ *iterations over* Δ.

Proof: obvious.

Lemma 61

Let f *be the flow at the end of a* Δ*-phase. Then the maximum flow is smaller than* $\mathrm{val}(f) + 2m\Delta$.

Proof: less obvious, but simple:

- An s-t cut in $G_f(\Delta)$ gives me an upper bound on the amount of flow that my algorithm can still add to f.

Capacity Scaling

Lemma 60

There are $\lceil \log C \rceil$ iterations over Δ.

Proof: obvious.

Lemma 61

Let f be the flow at the end of a Δ-phase. Then the maximum flow is smaller than $\mathrm{val}(f) + 2m\Delta$.

Proof: less obvious, but simple:

- An s-t cut in $G_f(\Delta)$ gives me an upper bound on the amount of flow that my algorithm can still add to f.
- The edges that currently have capacity at most Δ in G_f form an s-t cut with capacity at most $2m\Delta$.

Capacity Scaling

Capacity Scaling

Lemma 62

There are at most $2m$ augmentations per scaling-phase.

Capacity Scaling

Lemma 62

There are at most $2m$ augmentations per scaling-phase.

Proof:

- Let f be the flow at the end of the previous phase.

Capacity Scaling

Lemma 62

There are at most $2m$ augmentations per scaling-phase.

Proof:

- Let f be the flow at the end of the previous phase.
- $\text{val}(f^*) \leq \text{val}(f) + 2m\Delta$

Capacity Scaling

Lemma 62

There are at most $2m$ augmentations per scaling-phase.

Proof:

- Let f be the flow at the end of the previous phase.
- $\mathrm{val}(f^*) \le \mathrm{val}(f) + 2m\Delta$
- each augmentation increases flow by Δ.

Capacity Scaling

Lemma 62

There are at most $2m$ augmentations per scaling-phase.

Proof:

- Let f be the flow at the end of the previous phase.
- $\text{val}(f^*) \le \text{val}(f) + 2m\Delta$
- each augmentation increases flow by Δ.

Theorem 63

We need $\mathcal{O}(m \log C)$ augmentations. The algorithm can be implemented in time $\mathcal{O}(m^2 \log C)$.