



Design and analysis of algorithms

Lecture 28 & 29 & 30

Edyta Szymańska

edyta@cc.gatech.edu

Network flows - overview

- ⑥ use a graph to model material that flows through conduits

Network flows - overview

- ⑥ use a graph to model material that flows through conduits
- ⑥ each edge represents one conduit, and has a **capacity**, which is an upper bound on the **flow rate**=units/time

Network flows - overview

- ⑥ use a graph to model material that flows through conduits
- ⑥ each edge represents one conduit, and has a **capacity**, which is an upper bound on the **flow rate**=units/time
- ⑥ can think of edges as pipes, but flows don't need to be of liquids (communication network for moving data, vertices - routers)

Network flows - overview

- ⑥ use a graph to model material that flows through conduits
- ⑥ each edge represents one conduit, and has a **capacity**, which is an upper bound on the **flow rate**=units/time
- ⑥ can think of edges as pipes, but flows don't need to be of liquids (communication network for moving data, vertices - routers)
- ⑥ Goal: to compute max rate of material flow from a designated **source** to a designated **sink**

Formalization

FLOW NETWORK: $G = (V, E)$ directed

each edge (u, v) has a **capacity** $c(u, v) \geq 0$

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

source vertex s , **sink vertex** t ,

there is a path $s \rightsquigarrow v \rightsquigarrow t$ for all $v \in V$.

Formalization

FLOW NETWORK: $G = (V, E)$ directed

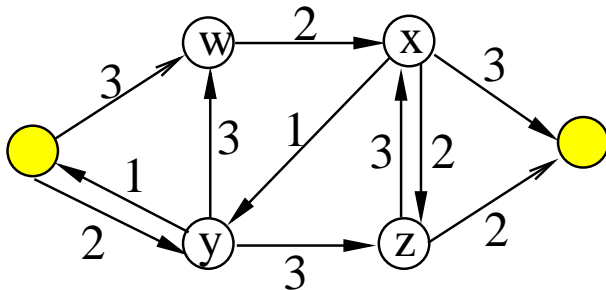
each edge (u, v) has a **capacity** $c(u, v) \geq 0$

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

source vertex s , **sink** vertex t ,

there is a path $s \rightsquigarrow v \rightsquigarrow t$ for all $v \in V$.

Flow network example:



Defining flow

Flow: a function $f : V \times V \rightarrow \mathbb{R}$ satisfying

⑥ **capacity constraint:** $\forall u, v \in V, f(u, v) \leq c(u, v)$

⑥ **skew symmetry:** $\forall u, v \in V : f(u, v) = -f(v, u)$

⑥ **flow conservation:** $\forall u \in V - \{s, t\}$ we require

$$\sum_{v \in V} f(u, v) = 0$$

Equivalently, "flow in = flow out"

Defining flow

Flow: a function $f : V \times V \rightarrow \mathbb{R}$ satisfying

⑥ **capacity constraint:** $\forall u, v \in V, f(u, v) \leq c(u, v)$

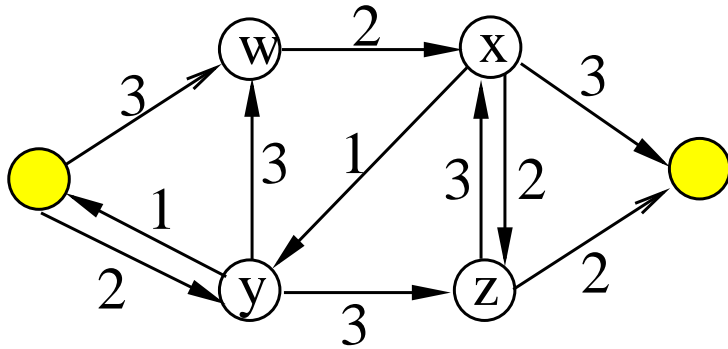
⑥ **skew symmetry:** $\forall u, v \in V : f(u, v) = -f(v, u)$

⑥ **flow conservation:** $\forall u \in V - \{s, t\}$ we require

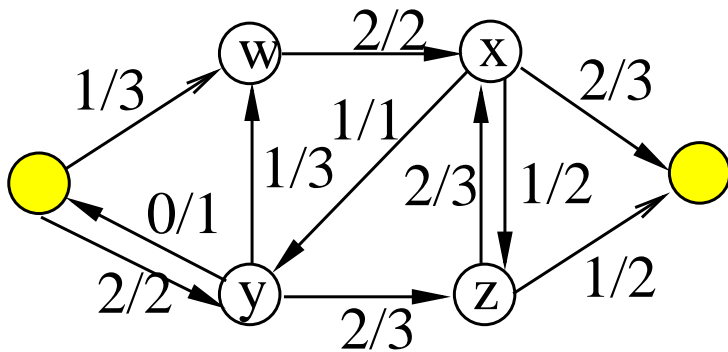
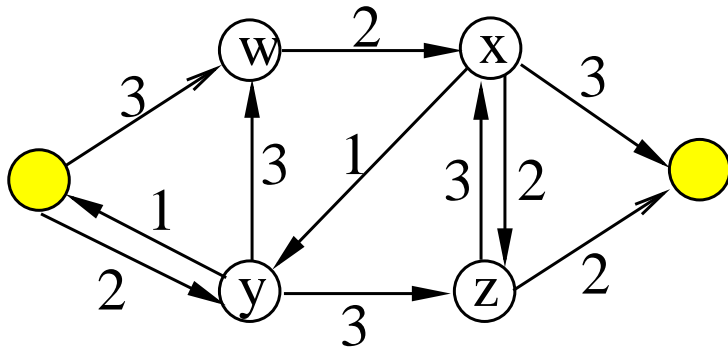
$$\sum_{v \in V} f(u, v) = 0$$

Equivalently, "flow in = flow out"

Example




Example



edges labelled flow/capacity

Cancellation of flows



Do we want to have positive flows going in both directions between two vertices ?

Cancellation of flows

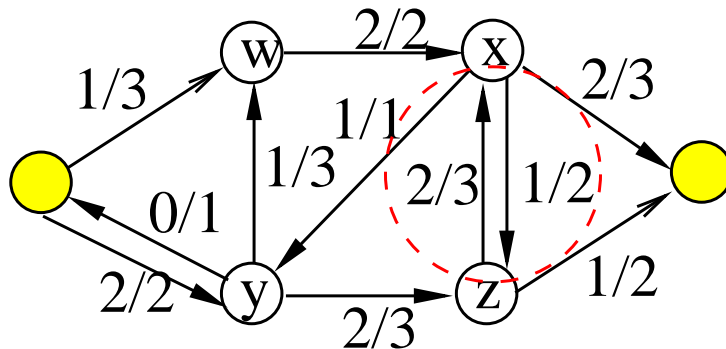
Do we want to have positive flows going in both directions between two vertices ?

No! such flows cancel (maybe partially) each other.

Cancellation of flows

Do we want to have positive flows going in both directions between two vertices ?

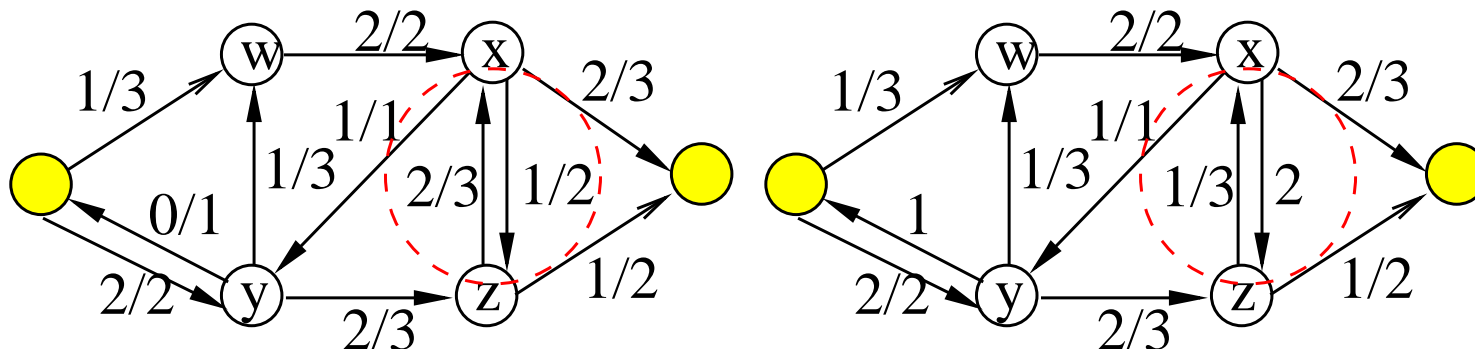
No! such flows cancel (maybe partially) each other.



Cancellation of flows

Do we want to have positive flows going in both directions between two vertices ?

No! such flows cancel (maybe partially) each other.



Maximum-flow problem

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

Maximum-flow problem

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

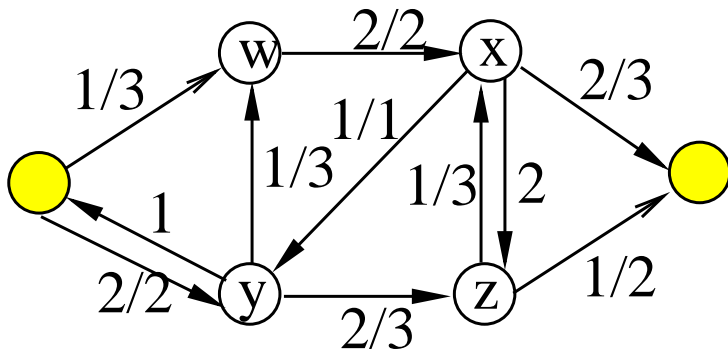
Maximum-flow problem: given a flow network G, s, t and c , find a flow of maximum value.

Maximum-flow problem

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

Maximum-flow problem: given a flow network G, s, t and c , find a flow of maximum value.

Example:

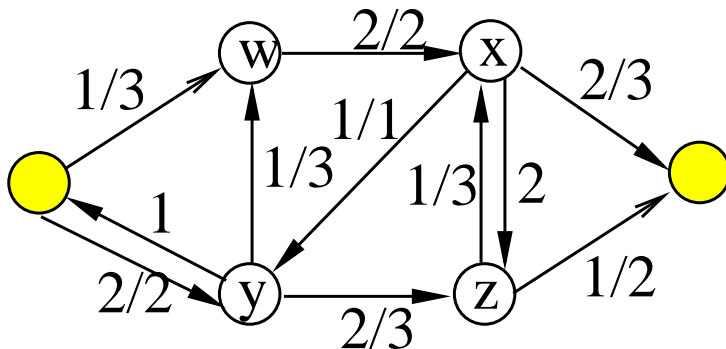


Maximum-flow problem

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

Maximum-flow problem: given a flow network G, s, t and c , find a flow of maximum value.

Example:



$$f = |f| = 3.$$

Designing the Algorithm

Dynamic programming? - not known, greedy?-doesn't work

Designing the Algorithm

Idea:



Designing the Algorithm

Idea:

If we have some flow ...

Designing the Algorithm

Idea:

If we have some flow ...

... and can find a path $p : s \rightsquigarrow t$ (**augmenting path**) s.t.

there is an $a > 0$, and for each edge $(u, v) \in p$ we can add a units of flow : $f(u, v) + a \leq c(u, v)$

Designing the Algorithm

Idea:

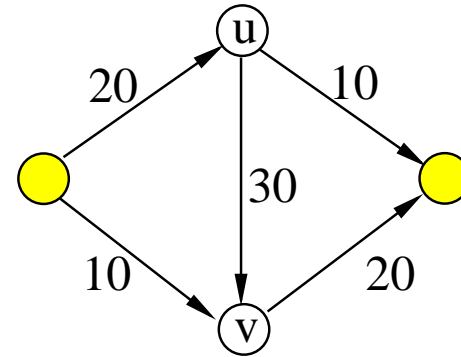
If we have some flow ...

... and can find a path $p : s \rightsquigarrow t$ (**augmenting path**) s.t.

there is an $a > 0$, and for each edge $(u, v) \in p$ we can add a units of flow : $f(u, v) + a \leq c(u, v)$

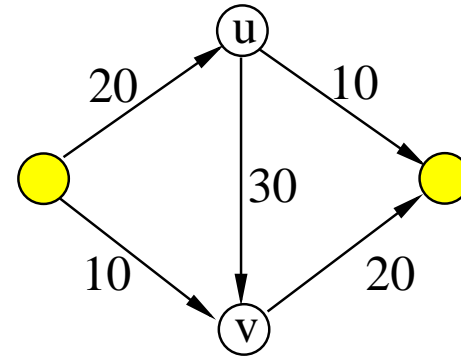
Then just do it, to get a better flow!

Augmenting path



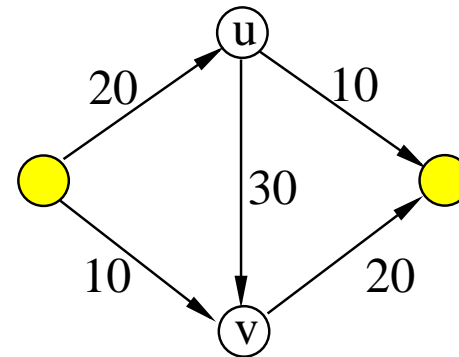
Consider the following network

Augmenting path



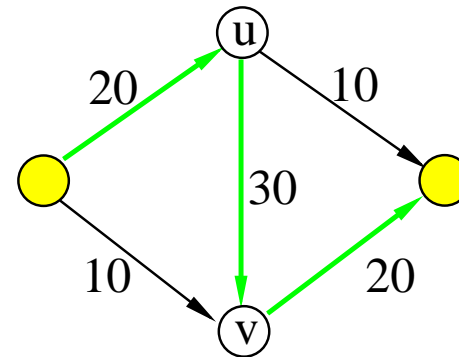
Consider the following network

Augmenting path



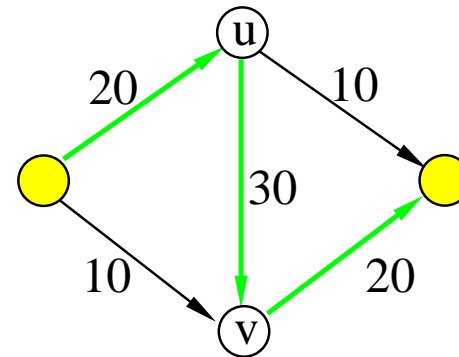
Consider the following network
Push 20 units of flow along the path
 s, u, v, t

Augmenting path



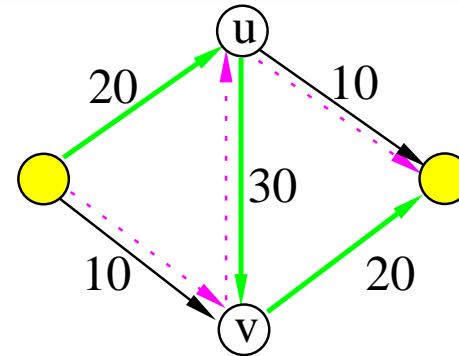
Consider the following network
Push 20 units of flow along the path
 s, u, v, t

Augmenting path



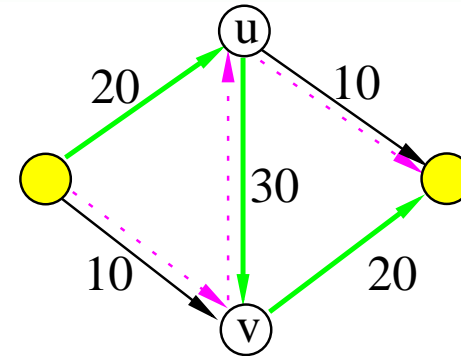
Consider the following network
Push 20 units of flow along the path
 s, u, v, t
Augmenting path ?

Augmenting path



Consider the following network
Push 20 units of flow along the path
 s, u, v, t
Augmenting path ?

Augmenting path



Consider the following network

Push 20 units of flow along the path

s, u, v, t

Augmenting path ?

YES, $s \rightarrow v \rightarrow u \rightarrow t$, e.i. we are pushing 10 units backward on (u, v)

Ford-Fulkerson method

```
FORD_FULKERSON( $G, s, t$ )  
initialize flow  $f$  to 0 everywhere  
while there is an augmenting path  $p$  do  
    augment flow  $f$  along path  $p$   
return  $f$ 
```

Ford-Fulkerson method

```
FORD_FULKERSON( $G, s, t$ )  
initialize flow  $f$  to 0 everywhere  
while there is an augmenting path  $p$  do  
    augment flow  $f$  along path  $p$   
return  $f$ 
```

- ⑥ How do we find augmenting paths p ?
- ⑥ How much additional flow can we send through p ?
- ⑥ Does the algorithm always find the maximum flow ?

Residual network

How do we find augmenting path?

Residual network

How do we find augmenting path?
It is any path in **residual network**

Residual network

How do we find augmenting path?
It is any path in **residual network**

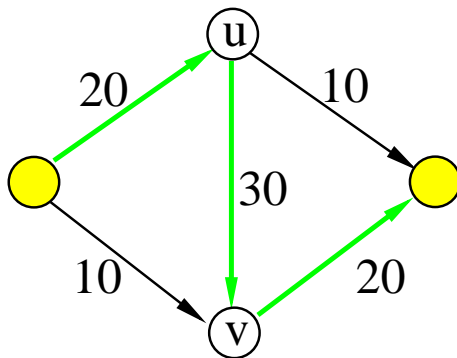
- ⑥ residual capacities: $c_f(u, v) = c(u, v) - f(u, v)$
- ⑥ residual network: $G_f = (V, E_f)$, where
 $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$

Residual network

How do we find augmenting path?
It is any path in **residual network**

- ⑥ residual capacities: $c_f(u, v) = c(u, v) - f(u, v)$
- ⑥ residual network: $G_f = (V, E_f)$, where
 $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$

Example:

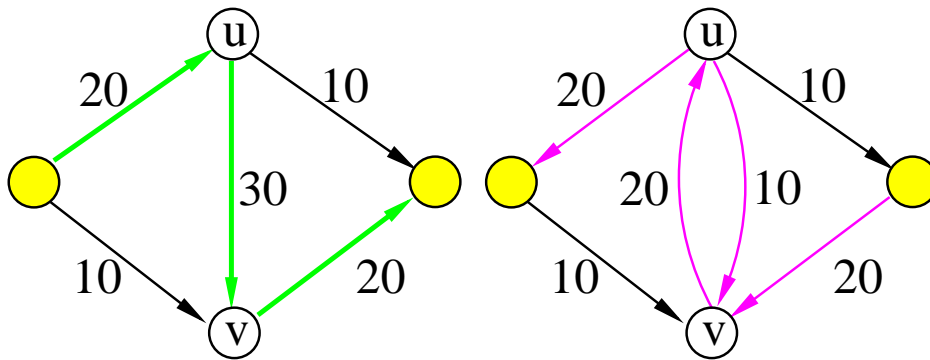


Residual network

How do we find augmenting path?
It is any path in **residual network**

- ⑥ residual capacities: $c_f(u, v) = c(u, v) - f(u, v)$
- ⑥ residual network: $G_f = (V, E_f)$, where
 $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$

Example:



Residual capacity of a path



How much additional flow can we send through an augmenting path?

Residual capacity of a path

How much additional flow can we send through an augmenting path?

Residual capacity of a path p in G_f :

$$c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$$

Residual capacity of a path

How much additional flow can we send through an augmenting path?

Residual capacity of a path p in G_f :

$$c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$$

Doing augmentation:

- ⑥ $\forall (u, v) \in p$ add $c_f(p)$ to $f(u, v)$ (and subtract from $f(v, u)$)
- ⑥ resulting flow is a valid flow with a larger value

Residual capacity of a path

How much additional flow can we send through an augmenting path?

Residual capacity of a path p in G_f :

$$c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$$

Doing augmentation:

- ⑥ $\forall (u, v) \in p$ add $c_f(p)$ to $f(u, v)$ (and subtract from $f(v, u)$)
- ⑥ resulting flow is a valid flow with a larger value

Ford-Fulkerson method formalized

```
FORD_FULKERSON( $G, s, t, c$ )
for each edge  $(u, v) \in E$  do  $f(u, v) := f(v, u) := 0$ 
while  $\exists$  path  $p$   $s \rightsquigarrow t$  in residual network  $G_f$  do
     $c_f := \min\{c_f(u, v) : (u, v) \in p\}$ 
    for each edge  $(u, v) \in p$  do
         $f(u, v) := f(u, v) + c_f$ 
         $f(v, u) := -f(u, v)$ 
return  $f$ 
```

Ford-Fulkerson method formalized

```
FORD_FULKERSON( $G, s, t, c$ )
for each edge  $(u, v) \in E$  do  $f(u, v) := f(v, u) := 0$ 
while  $\exists$  path  $p$   $s \rightsquigarrow t$  in residual network  $G_f$  do
     $c_f := \min\{c_f(u, v) : (u, v) \in p\}$ 
    for each edge  $(u, v) \in p$  do
         $f(u, v) := f(u, v) + c_f$ 
         $f(v, u) := -f(u, v)$ 
return  $f$ 
```

The algorithms based on this method differ in choosing the path p .

Ford-Fulkerson method formalized

```
FORD_FULKERSON( $G, s, t, c$ )  
for each edge  $(u, v) \in E$  do  $f(u, v) := f(v, u) := 0$   
while  $\exists$  path  $p$   $s \rightsquigarrow t$  in residual network  $G_f$  do  
     $c_f := \min\{c_f(u, v) : (u, v) \in p\}$   
    for each edge  $(u, v) \in p$  do  
         $f(u, v) := f(u, v) + c_f$   
         $f(v, u) := -f(u, v)$   
return  $f$ 
```

The algorithms based on this method differ in choosing the path p .

Example:

Correctness of Ford-Fulkerson

Does it always find the maximum flow ?

Correctness of Ford-Fulkerson

Does it always find the maximum flow ?

Cut: a partition of V into S and $T = V - S$ such that
 $s \in S, t \in T$

Correctness of Ford-Fulkerson

Does it always find the maximum flow ?

Cut: a partition of V into S and $T = V - S$ such that

$s \in S, t \in T$

Net flow $f(S, T) := \sum_{u \in S, v \in T} f(u, v)$

Correctness of Ford-Fulkerson

Does it always find the maximum flow ?

Cut: a partition of V into S and $T = V - S$ such that

$s \in S, t \in T$

Net flow $f(S, T) := \sum_{u \in S, v \in T} f(u, v)$

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

Correctness of Ford-Fulkerson

Does it always find the maximum flow ?

Cut: a partition of V into S and $T = V - S$ such that $s \in S, t \in T$

Net flow $f(S, T) := \sum_{u \in S, v \in T} f(u, v)$

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

Minimum cut - a cut with the smallest capacity of all cuts

Correctness of Ford-Fulkerson

Does it always find the maximum flow ?

Cut: a partition of V into S and $T = V - S$ such that

$s \in S, t \in T$

Net flow $f(S, T) := \sum_{u \in S, v \in T} f(u, v)$

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

Minimum cut - a cut with the smallest capacity of all cuts

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

Correctness of Ford-Fulkerson

Max-flow min-cut theorem

Correctness of Ford-Fulkerson

Max-flow min-cut theorem

If f is a flow in (G, s, t, c) then the following conditions are equivalent:

- ⑥ f is a maximum flow in G
- ⑥ the residual network G_f contains no augmenting path
- ⑥ $|f| = c(S, T)$ for some cut (S, T) of G

Correctness of Ford-Fulkerson

Max-flow min-cut theorem

If f is a flow in (G, s, t, c) then the following conditions are equivalent:

- ⑥ f is a maximum flow in G
- ⑥ the residual network G_f contains no augmenting path
- ⑥ $|f| = c(S, T)$ for some cut (S, T) of G

Proof:

Ford-Fulkerson method analysis

What is the worst-case running time of this method?

Ford-Fulkerson method analysis

What is the worst-case running time of this method?
Augmentation: $O(E)$

Ford-Fulkerson method analysis

What is the worst-case running time of this method?

Augmentation: $O(E)$

How many augmentations?

Ford-Fulkerson method analysis

What is the worst-case running time of this method?

Augmentation: $O(E)$

How many augmentations?

- ⑥ assume integer flows
- ⑥ every augmentation increases the value of the flow by some integer
- ⑥ $O(|f^*|)$, where f^* is the maximum flow

Ford-Fulkerson method analysis

What is the worst-case running time of this method?

Augmentation: $O(E)$

How many augmentations?

- ⑥ assume integer flows
- ⑥ every augmentation increases the value of the flow by some integer
- ⑥ $O(|f^*|)$, where f^* is the maximum flow

Total worst case: $O(|E||f^*|)$

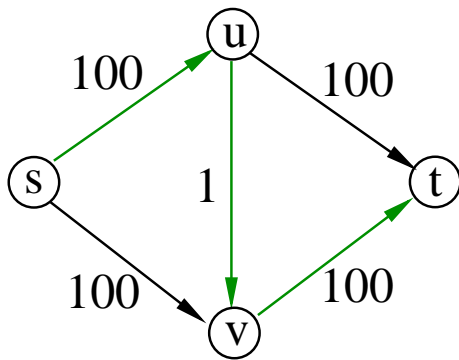
Choosing good augmenting paths



Pathological example:

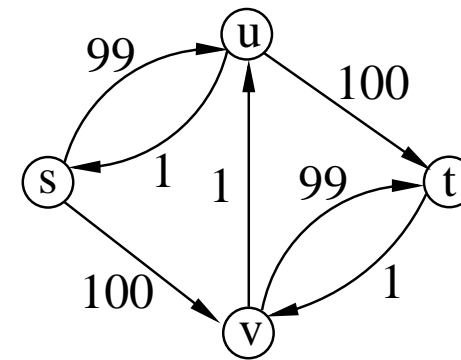
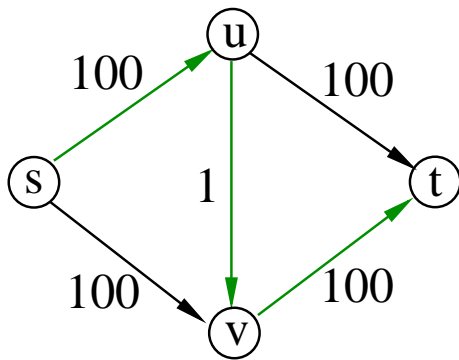
Choosing good augmenting paths

Pathological example:



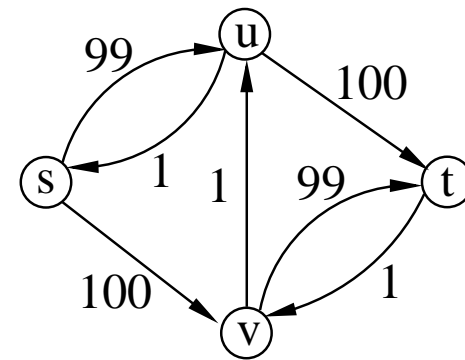
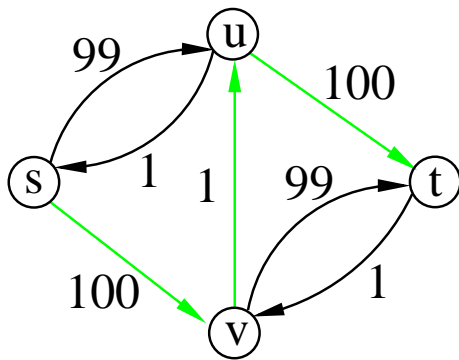
Choosing good augmenting paths

Pathological example:



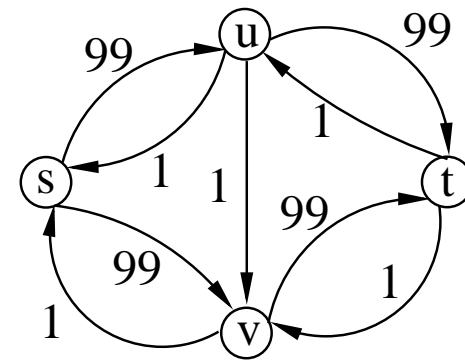
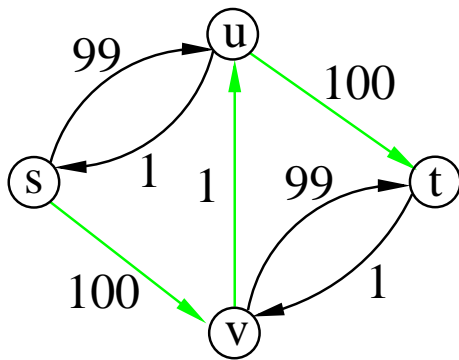
Choosing good augmenting paths

Pathological example:



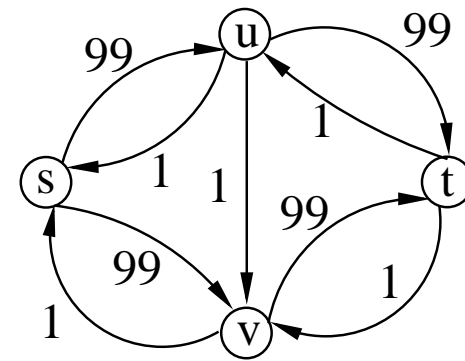
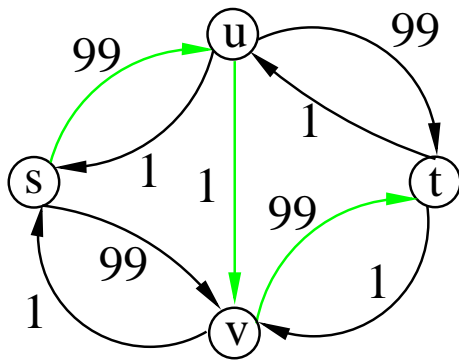
Choosing good augmenting paths

Pathological example:



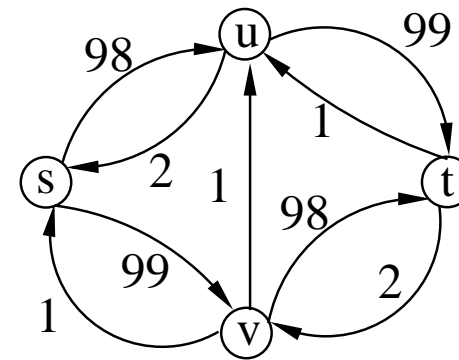
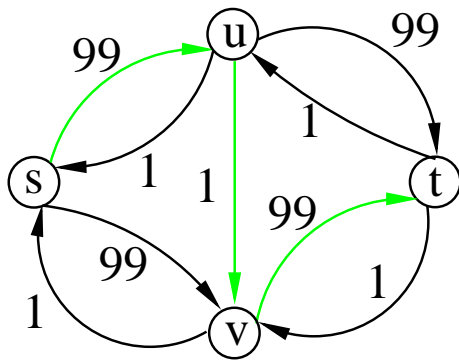
Choosing good augmenting paths

Pathological example:



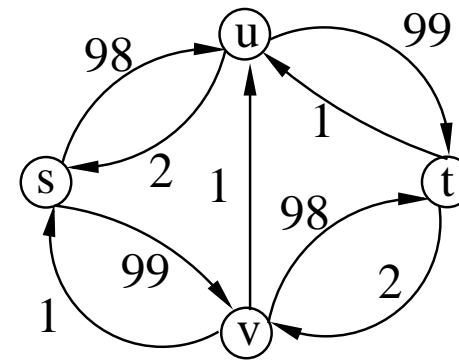
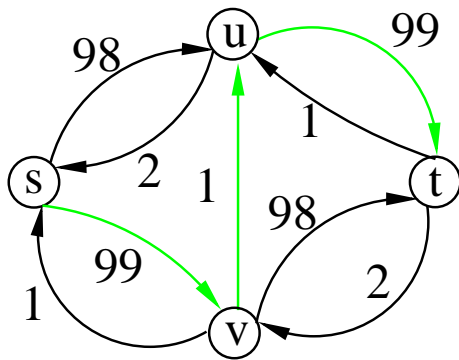
Choosing good augmenting paths

Pathological example:



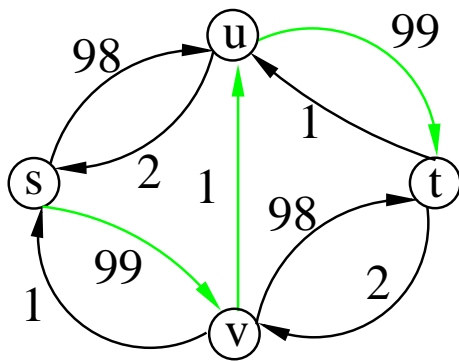
Choosing good augmenting paths

Pathological example:



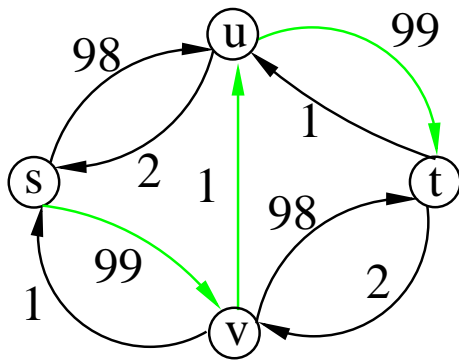
Choosing good augmenting paths

Pathological example:



Choosing good augmenting paths

Pathological example:



total number of augmentations: 200

Edmonds-Karp method algorithm

Take shortest augmenting path (in the number of edges)

Edmonds-Karp method algorithm

Take shortest augmenting path (in the number of edges)

Breadth-first search: $O(m + n) = O(m)$

Edmonds-Karp method algorithm

Take shortest augmenting path (in the number of edges)

Breadth-first search: $O(m + n) = O(m)$

To prove: number of augmentations

$O(nm)$, $n = |V|$, $m = |E|$

Edmonds-Karp method algorithm

Take shortest augmenting path (in the number of edges)

Breadth-first search: $O(m + n) = O(m)$

To prove: number of augmentations

$O(nm)$, $n = |V|$, $m = |E|$

Fact 1: *The length of the shortest augmenting path does not decrease.*

Edmonds-Karp method algorithm

Take shortest augmenting path (in the number of edges)

Breadth-first search: $O(m + n) = O(m)$

To prove: number of augmentations

$O(nm)$, $n = |V|$, $m = |E|$

Fact 1: *The length of the shortest augmenting path does not decrease.*

Fact 2: *Each edge can become **critical** (has the minimum residual capacity in the path) at most $\sim \frac{n}{2}$ times*

Edmonds-Karp method algorithm

Take shortest augmenting path (in the number of edges)

Breadth-first search: $O(m + n) = O(m)$

To prove: number of augmentations

$O(nm)$, $n = |V|$, $m = |E|$

Fact 1: *The length of the shortest augmenting path does not decrease.*

Fact 2: *Each edge can become **critical** (has the minimum residual capacity in the path) at most $\sim \frac{n}{2}$ times*

Total = $O(nm^2)$, polynomial !

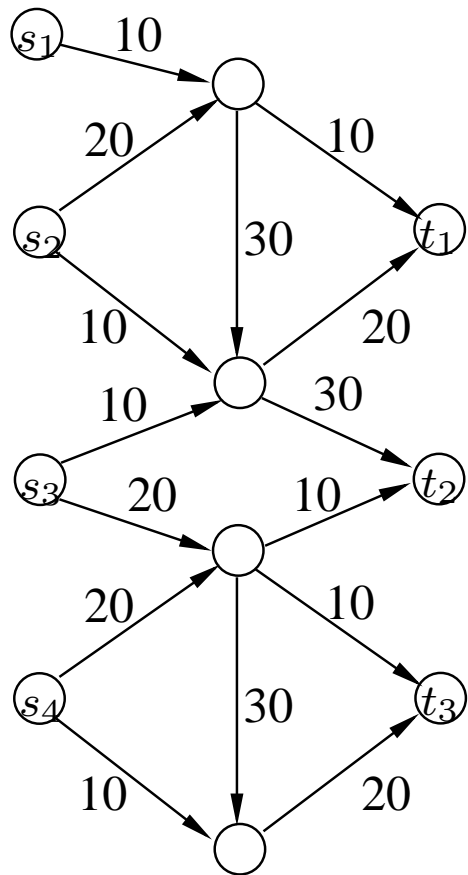
Multiple sources or sinks



Add two extra vertices to make one **supersource** and one **supersink**.

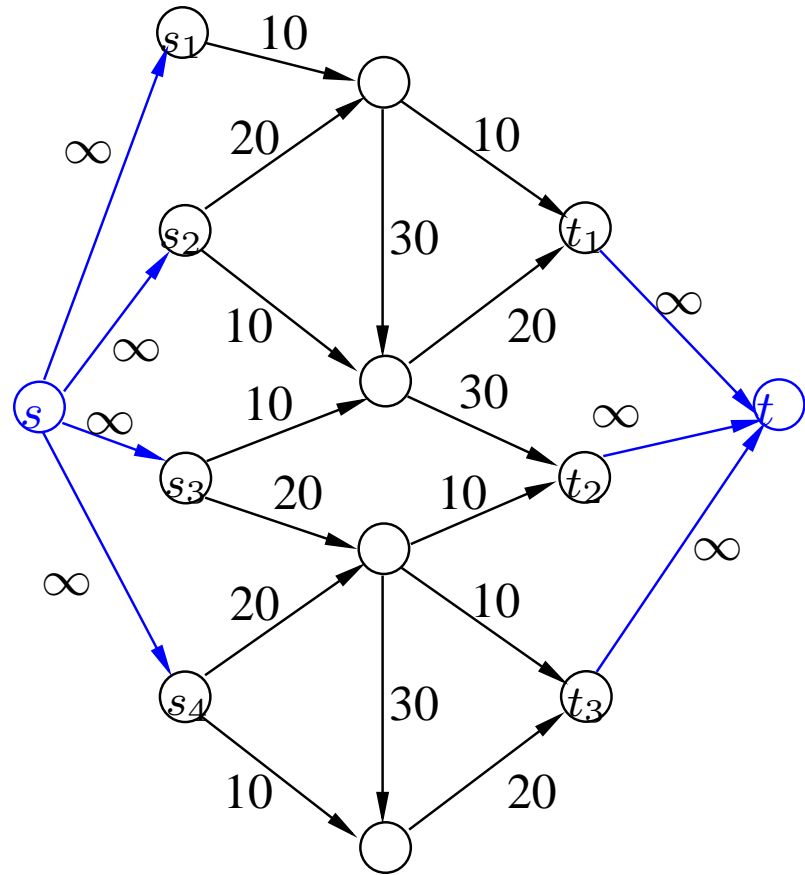
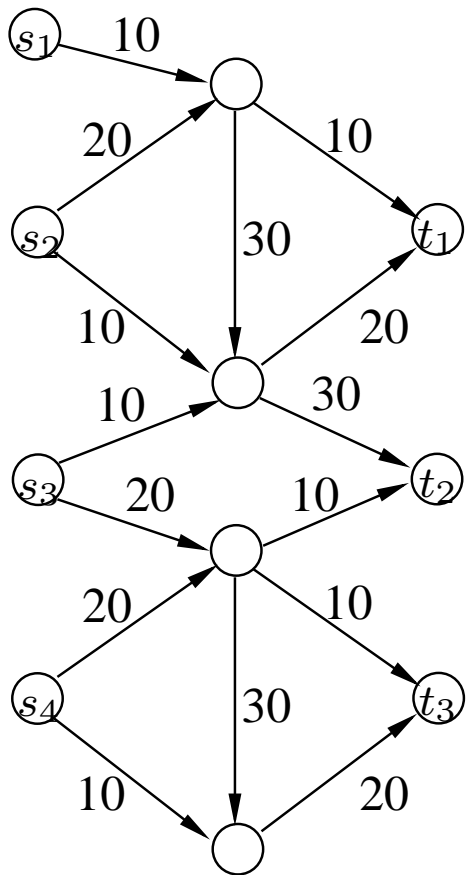
Multiple sources or sinks

Add two extra vertices to make one **supersource** and one **supersink**.



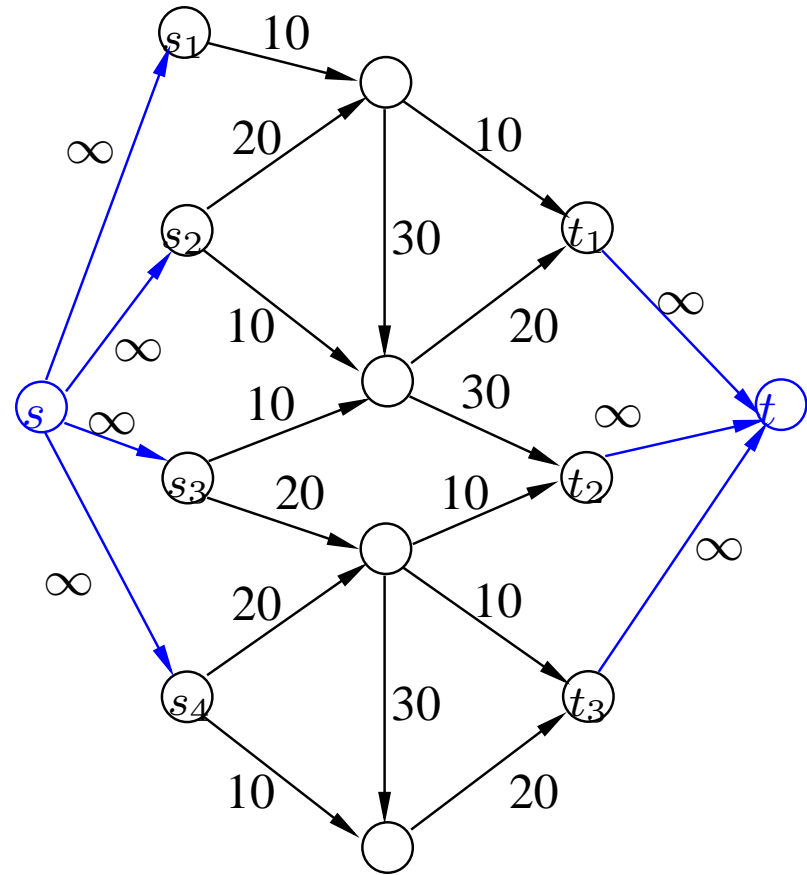
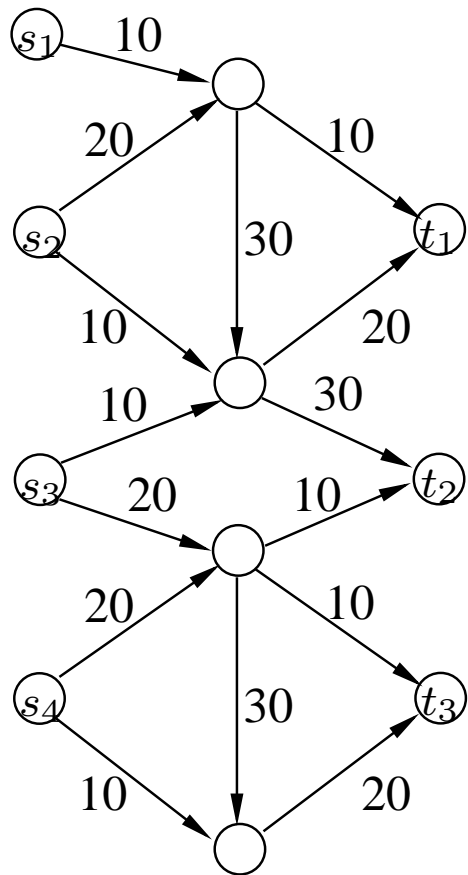
Multiple sources or sinks

Add two extra vertices to make one **supersource** and one **supersink**.



Multiple sources or sinks

Add two extra vertices to make one **supersource** and one **supersink**.



Other approaches to Max-Flow

Several other methods are known:

Other approaches to Max-Flow

Several other methods are known:

- ⑥ push-relabel method (Goldberg & Tarjan)- $O(nm \log(n^2/m + 2))$
- ⑥ many further improvements

An application of Max-Flow

Maximum bipartite matching problem

An application of Max-Flow

Maximum bipartite matching problem

Matching M in a graph is a subset of edges $M \subseteq E$ such that each vertex appears in at most one edge in M .

An application of Max-Flow

Maximum bipartite matching problem

Matching M in a graph is a subset of edges $M \subseteq E$ such that each vertex appears in at most one edge in M .

Bipartite graph: $G = (V, E)$, such that V can be partitioned as $V = X \cup Y$, and every edge in E has one end in X and one end in Y .

An application of Max-Flow

Maximum bipartite matching problem

Matching M in a graph is a subset of edges $M \subseteq E$ such that each vertex appears in at most one edge in M .

Bipartite graph: $G = (V, E)$, such that V can be partitioned as $V = X \cup Y$, and every edge in E has one end in X and one end in Y .

Goal: *Find a matching in bipartite G of the largest possible number of edges.*

Maximum bipartite matching problem

How can we formulate this problem to become a max-flow problem ?

Maximum bipartite matching problem

How can we formulate this problem to become a max-flow problem ?

Build a flow network $G' = (V', E)$ as follows:

$$V' := V \cup \{s, t\}, E' := \{(s, v) : v \in X\} \cup \{(v, t) : v \in Y\} \cup \{(u, v) : \{u, v\} \in E, \& u \in X, v \in Y\}$$
$$\forall e \in E' c(e) := 1$$

Maximum bipartite matching problem

How can we formulate this problem to become a max-flow problem ?

Build a flow network $G' = (V', E)$ as follows:

$$V' := V \cup \{s, t\}, E' := \{(s, v) : v \in X\} \cup \{(v, t) : v \in Y\} \cup \{(u, v) : \{u, v\} \in E, \& u \in X, v \in Y\}$$
$$\forall e \in E' c(e) := 1$$

Maximum bipartite matching problem

How can we formulate this problem to become a max-flow problem ?

Build a flow network $G' = (V', E')$ as follows:

$V' := V \cup \{s, t\}$, $E' := \{(s, v) : v \in X\} \cup \{(v, t) : v \in Y\} \cup \{(u, v) : \{u, v\} \in E, \& u \in X, v \in Y\}$

$\forall e \in E' c(e) := 1$

Theorem: *The cardinality of a maximum matching M in a bipartite graph G equals the value of a maximum flow f in its corresponding flow network G' .*