Graphs - Maximum Flow

Fundamental Algorithms

Rodica Potolea, Camelia Lemnaru and Ciprian Oprișa

Technical University of Cluj-Napoca
Computer Science Department

Agenda

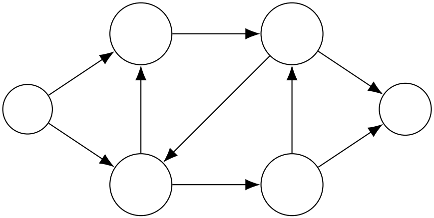
1. Maximum Flow concepts
2. The Ford-Fulkerson method
3. Maximum bipartite matching
4. Graphs recap
5. Exam info

Agenda

1. Maximum Flow concepts
2. The Ford-Fulkerson method
3. Maximum bipartite matching
4. Graphs recap
5. Exam info

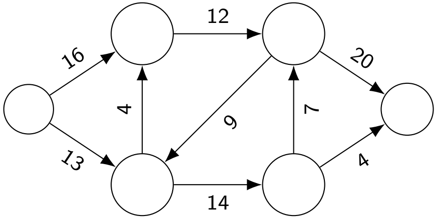
Flow networks

- a directed graph $G = (V, E)$



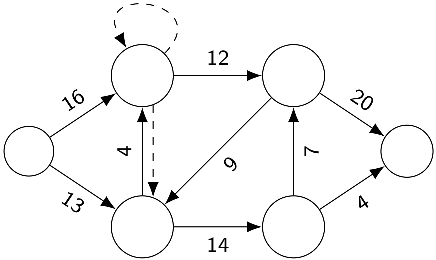
Flow networks

- a directed graph $G = (V, E)$
- a **capacity** function $c : E \to [0, \infty)$
  - $c(u, v) \geq 0$
  - if $(u, v) \notin E$, then $c(u, v) = 0$



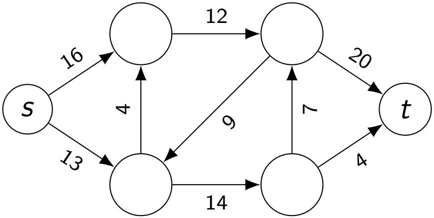
Flow networks

- a directed graph $G = (V, E)$
- a **capacity** function $c : E \to [0, \infty)$
  - $c(u, v) \geq 0$
  - if $(u, v) \notin E$, then $c(u, v) = 0$
- no antiparallel edges (if $(u, v) \in E$ then $(v, u) \notin E$)
- no self-loops



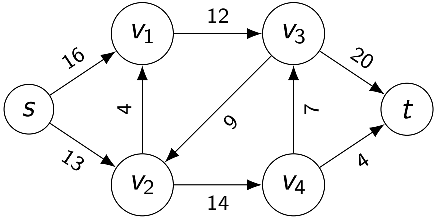
Flow networks

- a directed graph $G = (V, E)$
- a **capacity** function $c : E \to [0, \infty)$
  - $c(u, v) \geq 0$
  - if $(u, v) \notin E$, then $c(u, v) = 0$
- no antiparallel edges (if $(u, v) \in E$ then $(v, u) \notin E$)
- no self-loops
- two special vertices:
  - a **source** s
  - a **target**/**sink** t



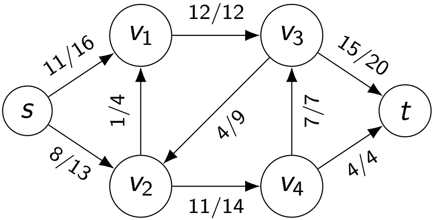
Flow networks

- a directed graph $G = (V, E)$
- a **capacity** function $c : E \to [0, \infty)$
  - $c(u, v) \geq 0$
  - if $(u, v) \notin E$, then $c(u, v) = 0$
- no antiparallel edges (if $(u, v) \in E$ then $(v, u) \notin E$)
- no self-loops
- two special vertices:
  - a **source** s
  - a **target**/**sink** t
- all other nodes $v \in V$ are on a path from s to t ($s \rightsquigarrow v \rightsquigarrow t$)



Flow formalism

$f : V \times V \to \mathbb{R}$

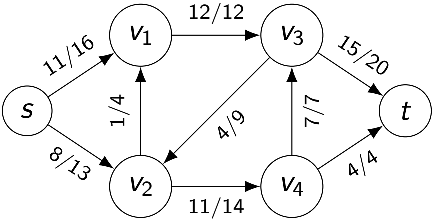


Flow formalism

$f : V \times V \to \mathbb{R}$

Capacity constraint:

$\forall u, v \in V,\ 0 \leq f(u, v) \leq c(u, v)$



Flow formalism

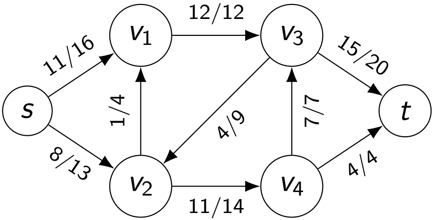
$f : V \times V \to \mathbb{R}$

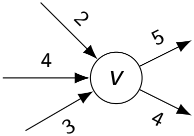
Capacity constraint:

$\forall u, v \in V,\ 0 \leq f(u, v) \leq c(u, v)$

Flow conservation: $\forall u \in V \setminus \{s, t\}$,

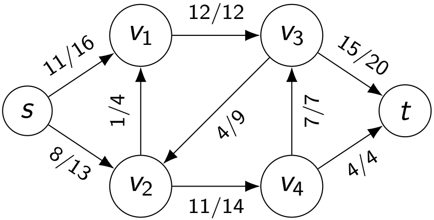
$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$



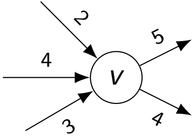


Flow formalism

$f : V \times V \to \mathbb{R}$
Capacity constraint:
$\forall u, v \in V,\ 0 \le f(u,v) \le c(u,v)$
Flow conservation: $\forall u \in V \setminus \{s,t\}$,
$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$



Flow value:
$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$

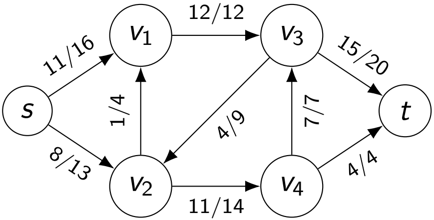


Flow formalism

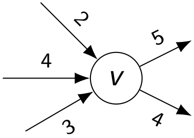
$f : V \times V \to \mathbb{R}$

Capacity constraint:
$\forall u, v \in V,\ 0 \le f(u,v) \le c(u,v)$

Flow conservation: $\forall u \in V \setminus \{s,t\}$,
$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$



Flow value:
$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$

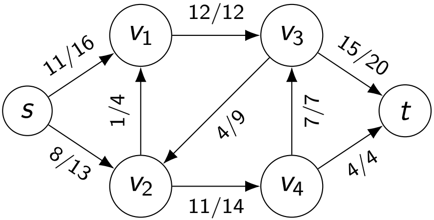


Maximum-flow problem

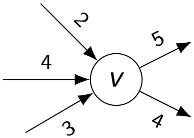
Given a directed graph $G = (V, E)$, a source s, a sink t and a capacity function $c : E \to [0, \infty)$, find the flow f with the maximum value.

Flow formalism

$f : V \times V \to \mathbb{R}$
Capacity constraint:
$\forall u, v \in V,\ 0 \leq f(u,v) \leq c(u,v)$
Flow conservation: $\forall u \in V \setminus \{s,t\}$,
$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$



Flow value:
$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$



Maximum-flow problem

Given a directed graph $G = (V, E)$, a source s, a sink t and a capacity function $c : E \to [0, \infty)$, find the flow f with the maximum value.

Real world applications

- water pipes
- electrical networks

Agenda

1. Maximum Flow concepts
2. The Ford-Fulkerson method
3. Maximum bipartite matching
4. Graphs recap
5. Exam info

Residual networks

- define the remaining capacity after some flow f passes
  - subtract the flow from each edge capacity
  - add reversed edges (so we can decrease the flow later)

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

- use all the edges with a positive remaining flow

$$G_f = (V, E_f), \quad E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

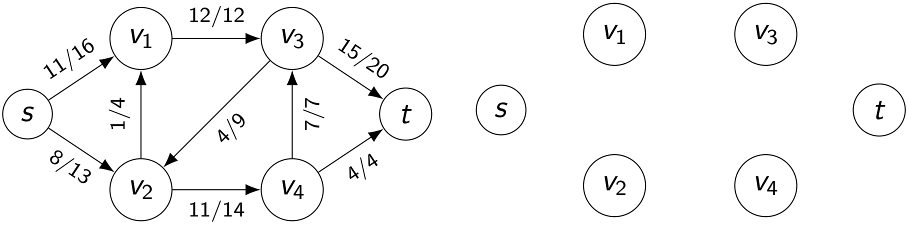
Residual networks

- define the remaining capacity after some flow f passes
  - subtract the flow from each edge capacity
  - add reversed edges (so we can decrease the flow later)

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

- use all the edges with a positive remaining flow

$$G_f = (V, E_f), \quad E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$



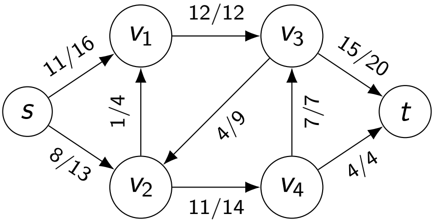
Residual networks

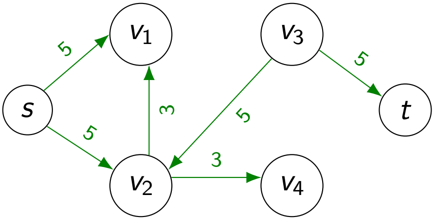
- define the remaining capacity after some flow f passes
  - subtract the flow from each edge capacity
  - add reversed edges (so we can decrease the flow later)

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

- use all the edges with a positive remaining flow

$$G_f = (V, E_f), \quad E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$





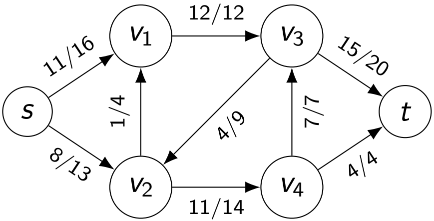
Residual networks

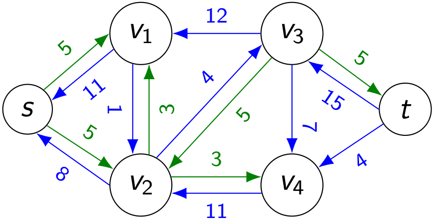
- define the remaining capacity after some flow f passes
  - subtract the flow from each edge capacity
  - add reversed edges (so we can decrease the flow later)

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

- use all the edges with a positive remaining flow

$$G_f = (V, E_f), \quad E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$





Flow augmentation

- let f be a flow in the network G
- let f' be a flow in the residual network G_f

Flow augmentation

- let f be a flow in the network G
- let f' be a flow in the residual network G_f
- we define $f \uparrow f'$ the **augmentation** of flow f by f'

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

Flow augmentation

- let f be a flow in the network G
- let f' be a flow in the residual network G_f
- we define $f \uparrow f'$ the **augmentation** of flow f by f'

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

Lemma 26.1

The function $f \uparrow f'$ is a flow in G with the value $|f \uparrow f'| = |f| + |f'|$.

Augmenting paths

- let f be a flow in the network G
- let p be a simple path $s \rightsquigarrow t$ in the residual network G_f
- the **residual capacity** of p is $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
- f_p is a flow in G_f with the value $|f_p| = c_f(p) > 0$

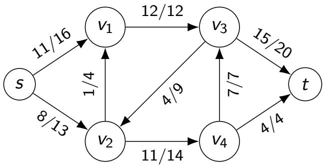
$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{otherwise} \end{cases}$$

Augmenting paths

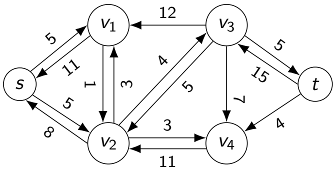
- let f be a flow in the network G
- let p be a simple path $s \rightsquigarrow t$ in the residual network G_f
- the **residual capacity** of p is $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
- f_p is a flow in G_f with the value $|f_p| = c_f(p) > 0$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{otherwise} \end{cases}$$

G :



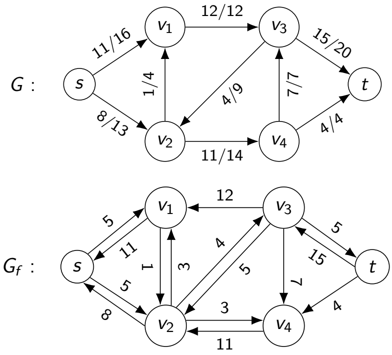
G_f :



Augmenting paths

- let f be a flow in the network G
- let p be a simple path $s \rightsquigarrow t$ in the residual network G_f
- the **residual capacity** of p is $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
- f_p is a flow in G_f with the value $|f_p| = c_f(p) > 0$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{otherwise} \end{cases}$$



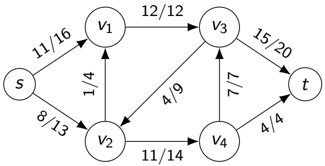
- $|f| = 11 + 8 = 19$

Augmenting paths

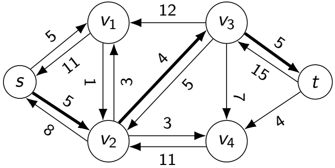
- let f be a flow in the network G
- let p be a simple path $s \rightsquigarrow t$ in the residual network G_f
- the **residual capacity** of p is $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
- f_p is a flow in G_f with the value $|f_p| = c_f(p) > 0$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{otherwise} \end{cases}$$

G :



G_f :



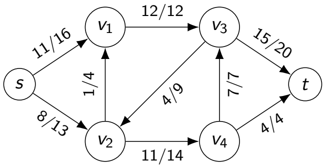
- $|f| = 11 + 8 = 19$
- $p = < s, v_2, v_3, t >$

Augmenting paths

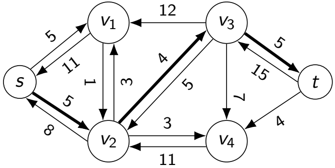
- let f be a flow in the network G
- let p be a simple path $s \rightsquigarrow t$ in the residual network G_f
- the **residual capacity** of p is $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
- f_p is a flow in G_f with the value $|f_p| = c_f(p) > 0$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{otherwise} \end{cases}$$

G :



G_f :



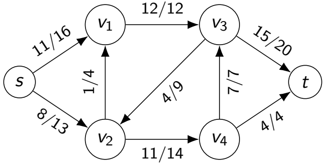
- $|f| = 11 + 8 = 19$
- $p = < s, v_2, v_3, t >,\ c_f(p) = 4$

Augmenting paths

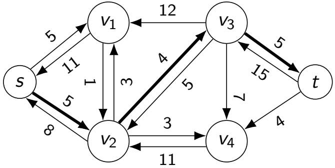
- let f be a flow in the network G
- let p be a simple path $s \rightsquigarrow t$ in the residual network G_f
- the **residual capacity** of p is $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
- f_p is a flow in G_f with the value $|f_p| = c_f(p) > 0$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{otherwise} \end{cases}$$

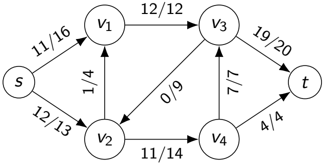
G :



G_f :



- $|f| = 11 + 8 = 19$
- $p =< s, v_2, v_3, t >$, $c_f(p) = 4$
- G with augmented flow $f \uparrow f_p$:

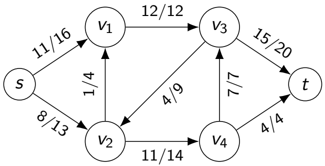


Augmenting paths

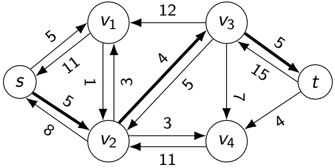
- let f be a flow in the network G
- let p be a simple path $s \rightsquigarrow t$ in the residual network G_f
- the **residual capacity** of p is $c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}$
- f_p is a flow in G_f with the value $|f_p| = c_f(p) > 0$

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ 0 & \text{otherwise} \end{cases}$$

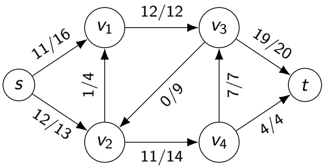
G :



G_f :



- $|f| = 11 + 8 = 19$
- $p =< s, v_2, v_3, t >$, $c_f(p) = 4$
- G with augmented flow $f \uparrow f_p$:



- $|f \uparrow f_p| = 11 + 12 = 19 + 4 = 23$

The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

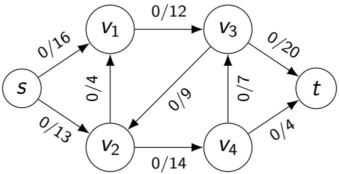
```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
    augment flow f along p
return f
```

The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
      augment flow f along p
return f
```

G :



$|f| = 0$

The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

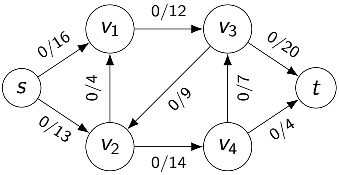
initialize flow f to 0

while $\exists$ augmenting path p in the residual network G_f

  augment flow f along p

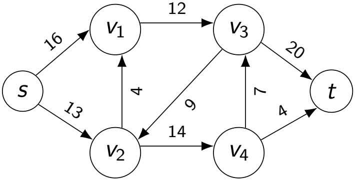
return f

G :



$|f| = 0$

G_f :

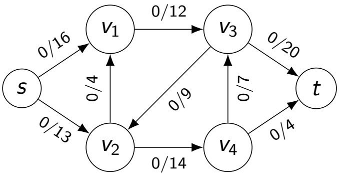


The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

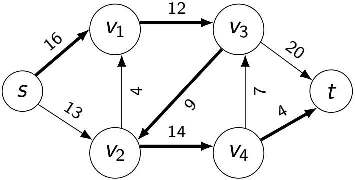
initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f

G :



$|f| = 0$

G_f :



$p =< s, v_1, v_3, v_2, v_4, t >$

The Ford-Fulkerson method

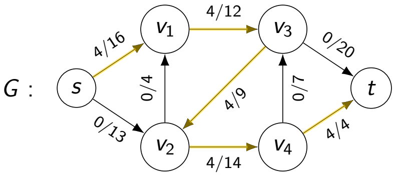
FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0

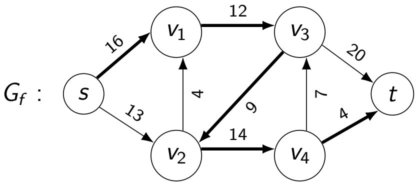
2 **while** $\exists$ augmenting path p in the residual network G_f

3 augment flow f along p

4 **return** f



$|f| = 4$

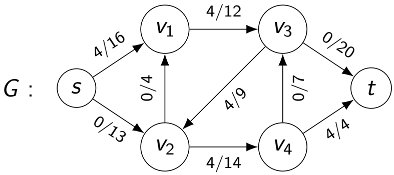


$p =< s, v_1, v_3, v_2, v_4, t >$

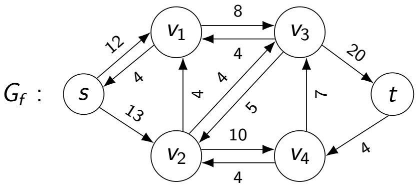
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
    augment flow f along p
return f
```



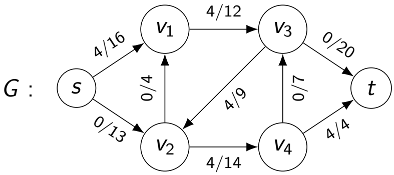
$|f| = 4$



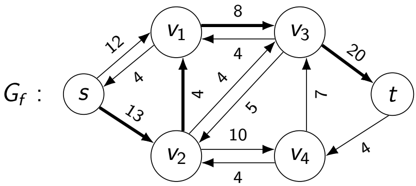
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
      augment flow f along p
return f
```



$|f| = 4$

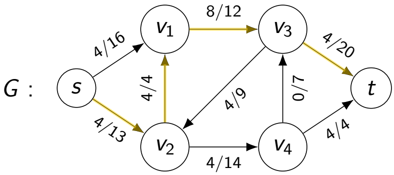


$p =< s, v_2, v_1, v_3, t >$

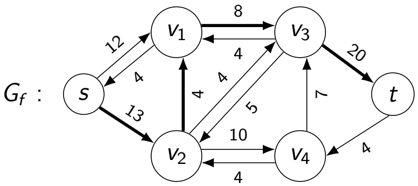
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f



$|f| = 8$

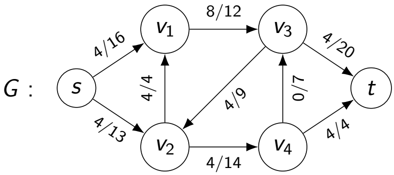


$p =< s, v_2, v_1, v_3, t >$

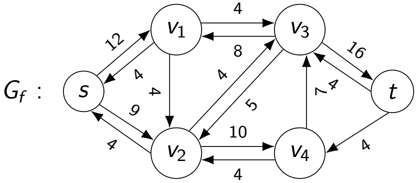
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
    augment flow f along p
return f
```



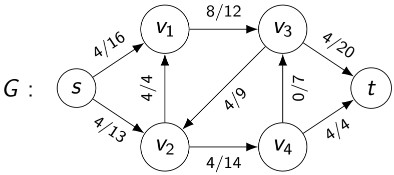
$|f| = 8$



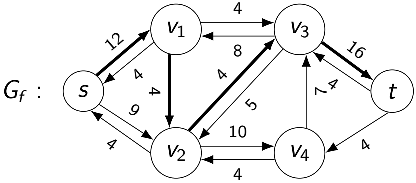
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f



$|f| = 8$



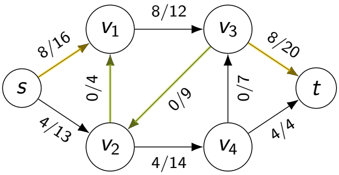
$p =< s, v_1, v_2, v_3, t >$

The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

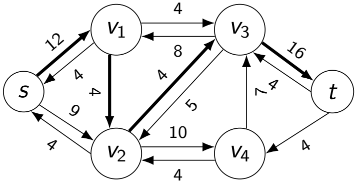
initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f

G :



$|f| = 12$

G_f :

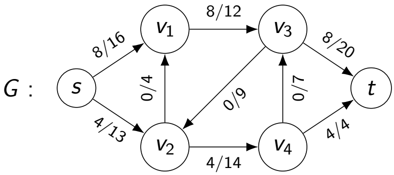


$p =< s, v_1, v_2, v_3, t >$

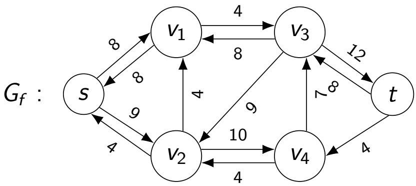
The Ford-Fulkerson method

Ford-Fulkerson-Method(G, s, t)

```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
    augment flow f along p
return f
```



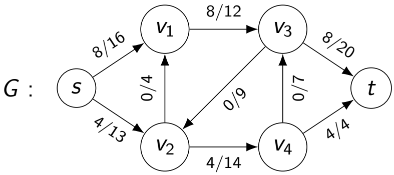
$|f| = 12$



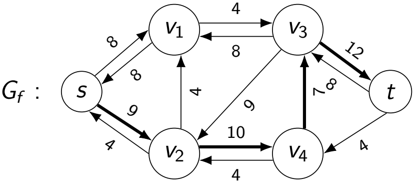
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f



$|f| = 12$



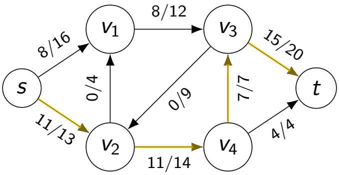
$p =< s, v_2, v_4, v_3, t >$

The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

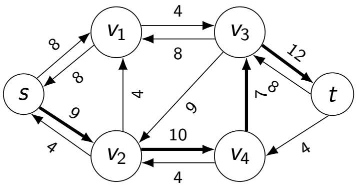
```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
    augment flow f along p
return f
```

G :



$|f| = 19$

G_f :

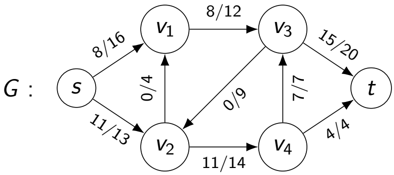


$p = < s, v_2, v_4, v_3, t >$

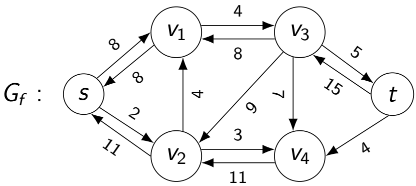
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f



$|f| = 19$



The Ford-Fulkerson method

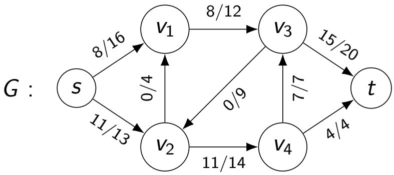
FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0

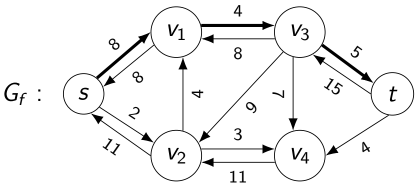
while $\exists$ augmenting path p in the residual network G_f

  augment flow f along p

return f



$|f| = 19$

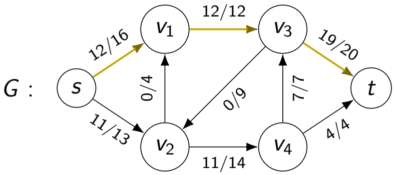


$p =< s, v_1, v_3, t >$

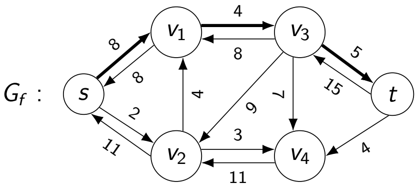
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f



$|f| = 23$



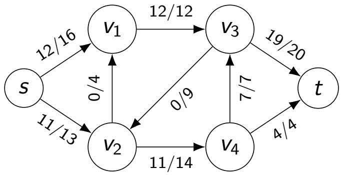
$p =< s, v_1, v_3, t >$

The Ford-Fulkerson method

$\textsc{Ford-Fulkerson-Method}(G, s, t)$

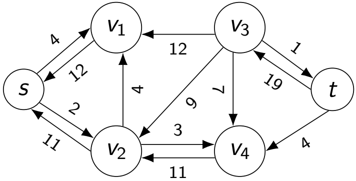
```
initialize flow f to 0
while ∃ augmenting path p in the residual network G_f
    augment flow f along p
return f
```

G :



$|f| = 23$

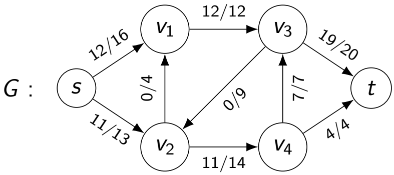
G_f :



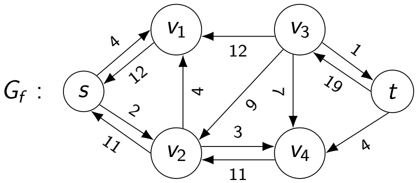
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0
2 **while** $\exists$ augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f



$|f| = 23$

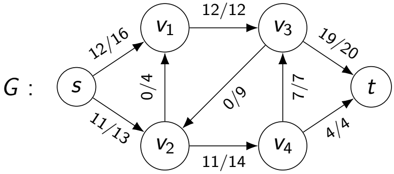


$p = \mathrm{NIL}$

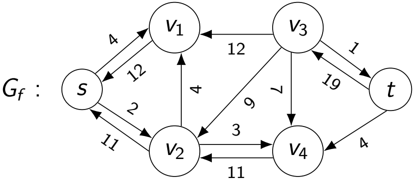
The Ford-Fulkerson method

FORD-FULKERSON-METHOD(G, s, t)

initialize flow f to 0
while $\exists$ augmenting path p in the residual network G_f
    augment flow f along p
return f



$|f| = 23$



$p = \text{NIL}$

The Ford-Fulkerson method - analysis (1)

- Ford-Fulkerson is a method, not an algorithm
  - it does not specify how the augmenting path p is selected

The Ford-Fulkerson method - analysis (1)

- Ford-Fulkerson is a method, not an algorithm
    - it does not specify how the augmenting path p is selected
- termination

The Ford-Fulkerson method - analysis (1)

- Ford-Fulkerson is a method, not an algorithm
  - it does not specify how the augmenting path p is selected
- termination
  - if all capacities are integers, each iteration increases the flow value by at least 1
  - the maximum flow is finite $\Rightarrow$ the flow cannot increase forever

The Ford-Fulkerson method - analysis (1)

- Ford-Fulkerson is a method, not an algorithm
  - it does not specify how the augmenting path p is selected
- termination
  - if all capacities are integers, each iteration increases the flow value by at least 1
  - the maximum flow is finite $\Rightarrow$ the flow cannot increase forever
  - if all capacities are rational, we can scale with the least common multiple of all denominators and work with integer capacities

The Ford-Fulkerson method - analysis (1)

- Ford-Fulkerson is a method, not an algorithm
    - it does not specify how the augmenting path p is selected
- termination
    - if all capacities are integers, each iteration increases the flow value by at least 1
    - the maximum flow is finite $\Rightarrow$ the flow cannot increase forever
    - if all capacities are rational, we can scale with the least common multiple of all denominators and work with integer capacities
    - with irrational capacities and a poor choice of augmenting paths, the algorithm might not terminate (the flow value increases with smaller and smaller values)
    - see the link below for a pathological example where the algorithm doesn't terminate:

      `https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/07DemoFordFulkersonPathological.pdf`

The Ford-Fulkerson method - analysis (2)

- correctness
  - no augmenting paths in $G_f \rightarrow f$ is a maximum flow (from the *max-flow min-cut theorem*, that will follow)

The Ford-Fulkerson method - analysis (2)

- correctness
  - no augmenting paths in $G_f \rightarrow f$ is a maximum flow (from the *max-flow min-cut theorem*, that will follow)
- complexity
  - finding an augmenting path and augmenting the flow: $O(V + E) = O(E)$
  - for integer capacities, if the maximum flow is $f^\star$, the number of iterations is at most $|f^\star|$
  - total running time: $O(E \cdot |f^\star|)$

Cuts of flow networks (1)

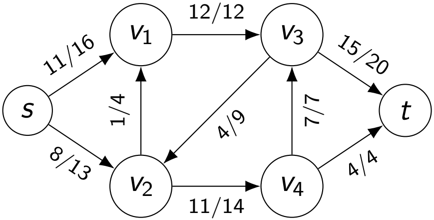
(S, T) is a **cut** of the flow network $G = (V, E)$ if

- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$, $t \in T$

Cuts of flow networks (1)

(S, T) is a **cut** of the flow network $G = (V, E)$ if

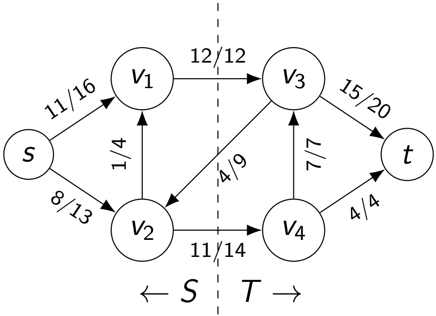
- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$, $t \in T$



Cuts of flow networks (1)

(S, T) is a **cut** of the flow network $G = (V, E)$ if

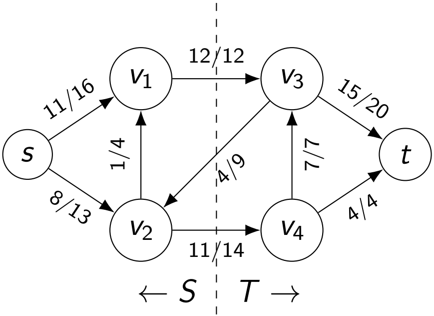
- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$, $t \in T$



Cuts of flow networks (1)

(S, T) is a **cut** of the flow network $G = (V, E)$ if

- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$, $t \in T$

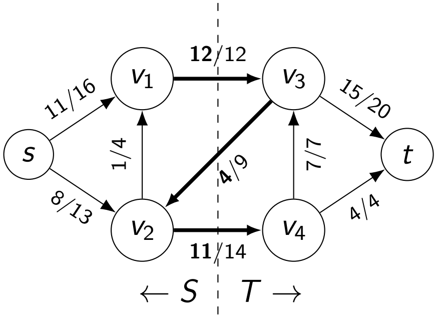


The **net flow** $f(S, T)$ across the cut is
$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$.

Cuts of flow networks (1)

(S, T) is a **cut** of the flow network $G = (V, E)$ if

- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$, $t \in T$



$f(S, T) = 12 + 11 - 4 = 19$

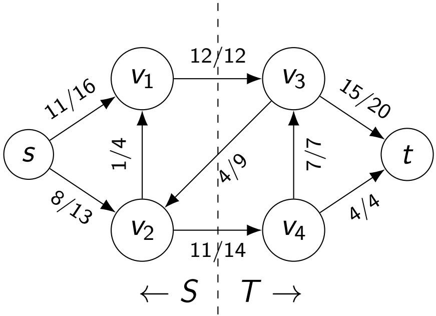
The **net flow** $f(S, T)$ across the cut is

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u).$$

Cuts of flow networks (1)

(S, T) is a **cut** of the flow network $G = (V, E)$ if

- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$, $t \in T$



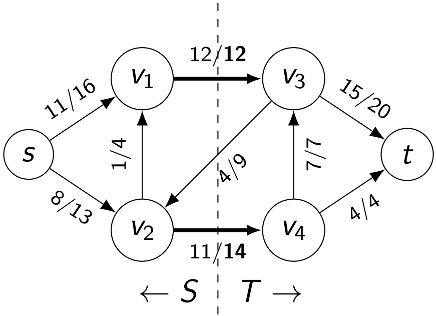
$$f(S, T) = 12 + 11 - 4 = 19$$

The **net flow** $f(S, T)$ across the cut is
$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$.
The **capacity** of the cut (S, T) is $c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$.

Cuts of flow networks (1)

(S, T) is a **cut** of the flow network $G = (V, E)$ if

- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$, $t \in T$



$f(S, T) = 12 + 11 - 4 = 19$
$c(S, T) = 12 + 14 = 26$

The **net flow** $f(S, T)$ across the cut is
$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$.
The **capacity** of the cut (S, T) is $c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$.

Cuts of flow networks (2)

Lemma 26.4

Let f be a flow in the network G with source s and sink t.
$\forall(S, T)$ a cut of G, the flow across (S, T) is $f(S, T) = |f|$.

Cuts of flow networks (2)

Lemma 26.4

Let f be a flow in the network G with source s and sink t.
$\forall (S, T)$ a cut of G, the flow across (S, T) is $f(S, T) = |f|$.

Corollary 26.5

The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.

Max-flow min-cut theorem

Theorem 26.6

If f is a flow in a network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G
2. the residual network G_f contains no augmenting paths
3. $|f| = c(S, T)$ for some cut (S, T) of G

Max-flow min-cut theorem

Theorem 26.6

If f is a flow in a network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G
2. the residual network G_f contains no augmenting paths
3. $|f| = c(S, T)$ for some cut (S, T) of G

Proof:

- $(1) \Rightarrow (2)$: contradiction $|f \uparrow f_p| = |f| + |f_p| > |f|$

Max-flow min-cut theorem

Theorem 26.6

If f is a flow in a network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G
2. the residual network G_f contains no augmenting paths
3. $|f| = c(S, T)$ for some cut (S, T) of G

Proof:

- $(1) \Rightarrow (2)$: contradiction $|f \uparrow f_p| = |f| + |f_p| > |f|$
- $(2) \Rightarrow (3)$: let $S = \{v \in V : \exists s \rightsquigarrow v \text{ in } G_f\}$, $T = V \setminus S$
  $\Rightarrow |f| \overset{\text{Lemma 26.4}}{=} f(S, T) = c(S, T)$

Max-flow min-cut theorem

Theorem 26.6

If f is a flow in a network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G
2. the residual network G_f contains no augmenting paths
3. $|f| = c(S, T)$ for some cut (S, T) of G

Proof:

- $(1) \Rightarrow (2)$: contradiction $|f \uparrow f_p| = |f| + |f_p| > |f|$
- $(2) \Rightarrow (3)$: let $S = \{v \in V : \exists s \rightsquigarrow v \text{ in } G_f\}$, $T = V \setminus S$
  $\Rightarrow |f| \overset{\text{Lemma 26.4}}{=} f(S, T) = c(S, T)$
- $(3) \Rightarrow (1)$: from Corollary 26.5

The Edmonds-Karp algorithm - approach

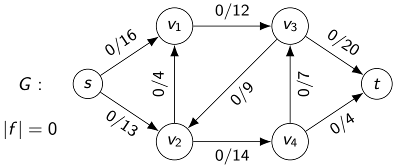
- based on the Ford-Fulkerson method
- finds the augmenting path in G_f using the BFS algorithm

The Edmonds-Karp algorithm

```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

The Edmonds-Karp algorithm

```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```



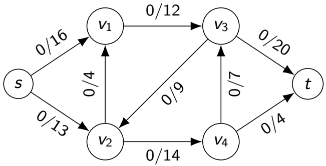
The Edmonds-Karp algorithm

EDMONDS-KARP(G, s, t)

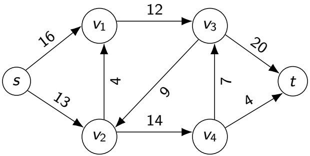
for each edge $(u, v) \in G.E$
    $(u, v).f = 0$
repeat
    $G_f = $ COMPUTE-RESIDUAL-NETWORK(G, s, t)
    $p = $ BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if $p \neq$ NIL
        $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
        for each edge $(u, v) \in p$
            if $(u, v) \in E$
                $(u, v).f = (u, v).f + c_f(p)$
            else $(v, u).f = (v, u).f - c_f(p)$
until $p ==$ NIL

G :

$|f| = 0$



G_f :



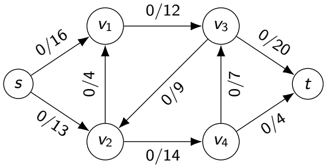
The Edmonds-Karp algorithm

EDMONDS-KARP(G, s, t)

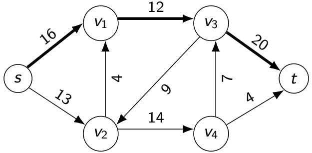
for each edge $(u, v) \in G.E$
    $(u, v).f = 0$
repeat
    $G_f = $ COMPUTE-RESIDUAL-NETWORK(G, s, t)
    $p = $ BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if $p \neq$ NIL
        $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
        for each edge $(u, v) \in p$
            if $(u, v) \in E$
                $(u, v).f = (u, v).f + c_f(p)$
            else $(v, u).f = (v, u).f - c_f(p)$
until $p ==$ NIL

G :

$|f| = 0$



G_f :



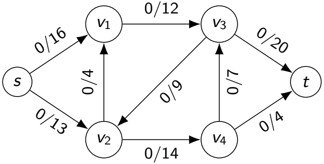
$p =< s, v_1, v_3, t >$

The Edmonds-Karp algorithm

EDMONDS-KARP(G, s, t)

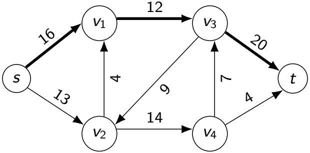
for each edge $(u, v) \in G.E$
    $(u, v).f = 0$
repeat
    $G_f = $ COMPUTE-RESIDUAL-NETWORK(G, s, t)
    $p = $ BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if $p \neq$ NIL
        $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
        for each edge $(u, v) \in p$
            if $(u, v) \in E$
                $(u, v).f = (u, v).f + c_f(p)$
            else $(v, u).f = (v, u).f - c_f(p)$
until $p ==$ NIL

G :



$|f| = 0$

G_f :



$p =< s, v_1, v_3, t >$ $c_f(p) = 12$

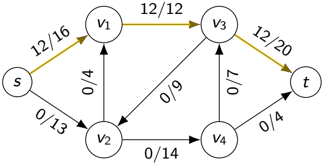
The Edmonds-Karp algorithm

$\text{EDMONDS-KARP}(G, s, t)$

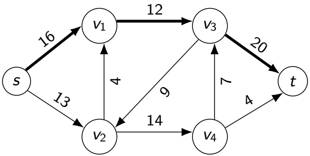
for each edge $(u, v) \in G.E$
    $(u, v).f = 0$
repeat
    $G_f = \text{COMPUTE-RESIDUAL-NETWORK}(G, s, t)$
    $p = \text{BFS-PATH}(G_f, s, t)$ // call $\text{BFS}(G_f, s)$ and find path to t
    if $p \neq \text{NIL}$
        $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
        for each edge $(u, v) \in p$
            if $(u, v) \in E$
                $(u, v).f = (u, v).f + c_f(p)$
            else $(v, u).f = (v, u).f - c_f(p)$
until $p == \text{NIL}$

G :

$|f| = 12$



G_f :



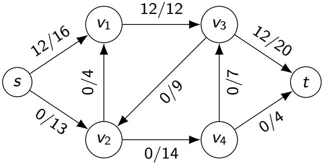
$p = < s, v_1, v_3, t >$ $c_f(p) = 12$

The Edmonds-Karp algorithm

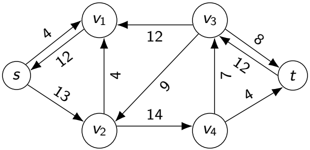
```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :

$|f| = 12$



G_f :

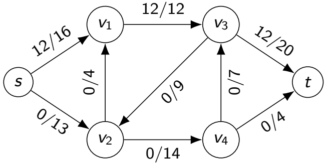


The Edmonds-Karp algorithm

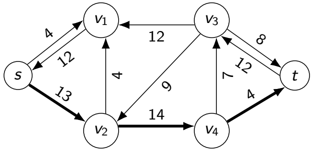
```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :

$|f| = 12$



G_f :

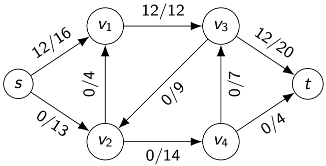


$p =< s, v_2, v_4, t >$

The Edmonds-Karp algorithm

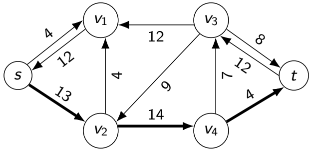
```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :



$|f| = 12$

G_f :



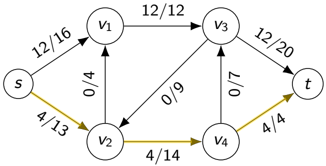
$p =< s, v_2, v_4, t >$ $\quad c_f(p) = 4$

The Edmonds-Karp algorithm

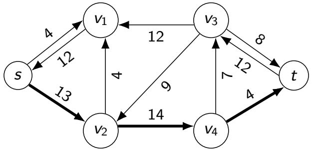
```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
      (u, v).f = 0
repeat
      G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
      p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
      if p ≠ NIL
            c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
            for each edge (u, v) ∈ p
                  if (u, v) ∈ E
                        (u, v).f = (u, v).f + c_f(p)
                  else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :

$|f| = 16$



G_f :



$p =< s, v_2, v_4, t >$ $c_f(p) = 4$

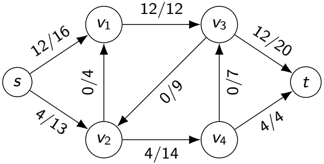
The Edmonds-Karp algorithm

EDMONDS-KARP(G, s, t)

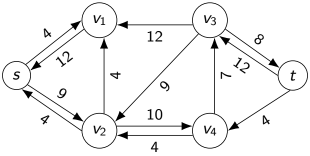
for each edge $(u, v) \in G.E$
    $(u, v).f = 0$
repeat
    $G_f = $ COMPUTE-RESIDUAL-NETWORK(G, s, t)
    $p = $ BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if $p \neq$ NIL
        $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
        for each edge $(u, v) \in p$
            if $(u, v) \in E$
                $(u, v).f = (u, v).f + c_f(p)$
            else $(v, u).f = (v, u).f - c_f(p)$
until $p ==$ NIL

G :

$|f| = 16$



G_f :



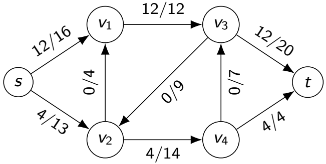
The Edmonds-Karp algorithm

EDMONDS-KARP(G, s, t)

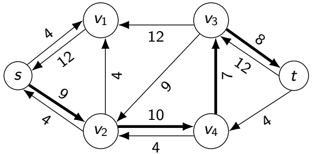
```
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :

$|f| = 16$



G_f :

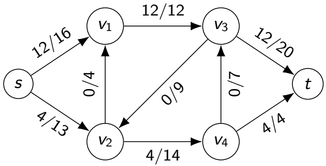


$p = < s, v_2, v_4, v_3, t >$

The Edmonds-Karp algorithm

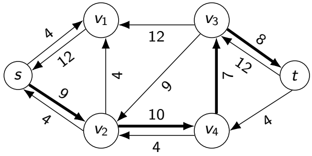
```
EDMONDS-KARP(G, s, t)
    for each edge (u, v) ∈ G.E
        (u, v).f = 0
    repeat
        G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
        p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
        if p ≠ NIL
            c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
            for each edge (u, v) ∈ p
                if (u, v) ∈ E
                    (u, v).f = (u, v).f + c_f(p)
                else (v, u).f = (v, u).f − c_f(p)
    until p == NIL
```

G :



$|f| = 16$

G_f :



$p =< s, v_2, v_4, v_3, t >$ $c_f(p) = 7$

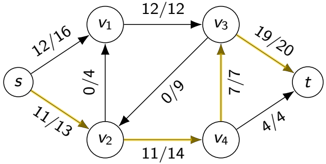
The Edmonds-Karp algorithm

EDMONDS-KARP(G, s, t)

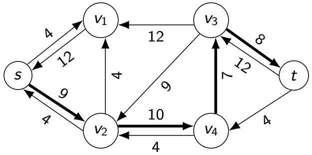
```
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :

$|f| = 23$



G_f :



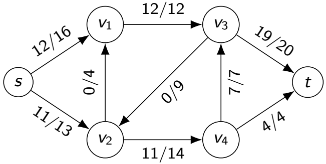
$p = < s, v_2, v_4, v_3, t >$ $c_f(p) = 7$

The Edmonds-Karp algorithm

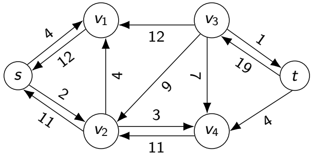
```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :

$|f| = 23$



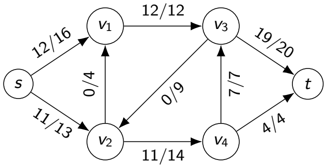
G_f :



The Edmonds-Karp algorithm

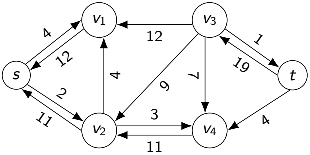
```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :



$|f| = 23$

G_f :



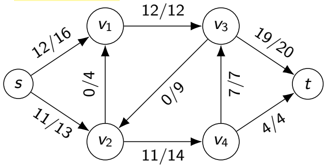
$p = \text{NIL}$

The Edmonds-Karp algorithm

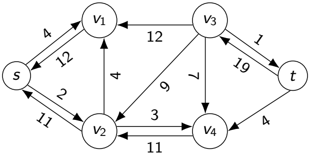
```
EDMONDS-KARP(G, s, t)
for each edge (u, v) ∈ G.E
    (u, v).f = 0
repeat
    G_f = COMPUTE-RESIDUAL-NETWORK(G, s, t)
    p = BFS-PATH(G_f, s, t) // call BFS(G_f, s) and find path to t
    if p ≠ NIL
        c_f(p) = min{c_f(u, v) : (u, v) ∈ p}
        for each edge (u, v) ∈ p
            if (u, v) ∈ E
                (u, v).f = (u, v).f + c_f(p)
            else (v, u).f = (v, u).f − c_f(p)
until p == NIL
```

G :

$|f| = 23$



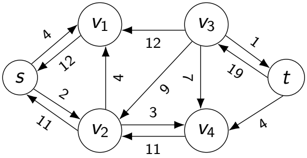
G_f :



$p = \text{NIL}$

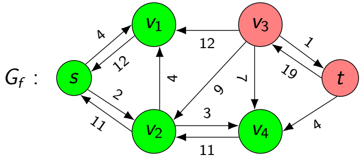
Max-flow min-cut exemplification

G_f :



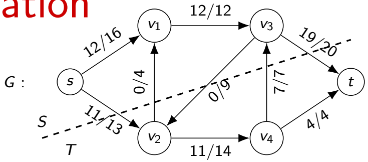
No path from s to t in the residual network

Max-flow min-cut exemplification

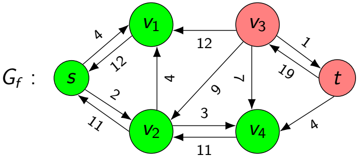


No path from s to t in the residual network

Max-flow min-cut exemplification

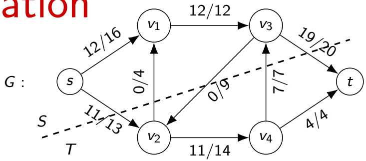


- $f(S, T) = 11 + 0 + 19 - 0 - 7 = 23$
- $c(S, T) = 13 + 9 + 20 = 42$
- any cut has the same net flow

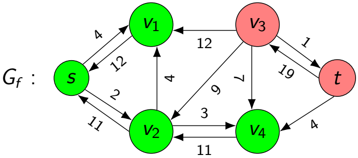


No path from s to t in the residual network

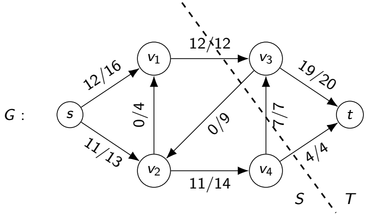
Max-flow min-cut exemplification



- $f(S, T) = 11 + 0 + 19 - 0 - 7 = 23$
- $c(S, T) = 13 + 9 + 20 = 42$
- any cut has the same net flow



No path from s to t in the residual network



- $f(S, T) = 12 + 7 + 4 - 0 = 23$
- $c(S, T) = 12 + 7 + 4 = 23$
- $f(S, T) = c(S, T) \Rightarrow$ **min-cut**

The Edmonds-Karp algorithm - analysis

Theorem 26.8

If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then the total number of flow augmentations performed by the algorithm is $O(V \cdot E)$.

The Edmonds-Karp algorithm - analysis

Theorem 26.8

If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then the total number of flow augmentations performed by the algorithm is $O(V \cdot E)$.

- since each BFS takes $O(V + E) = O(E)$, the total running time is $O(V \cdot E^2)$

The Edmonds-Karp algorithm - analysis

Theorem 26.8

If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source s and sink t, then the total number of flow augmentations performed by the algorithm is $O(V \cdot E)$.

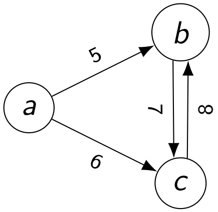
- since each BFS takes $O(V + E) = O(E)$, the total running time is $O(V \cdot E^2)$
- there exist $O(V^3)$ algorithms for computing the maximum flow (see textbook) and even faster ones

Networks with anti-parallel edges

- we didn't allow anti-parallel edges so we can add them in the residual network
- in real-world problems we can avoid anti-parallel edges by adding an extra vertex

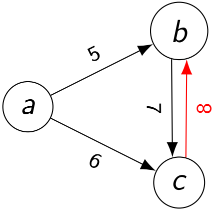
Networks with anti-parallel edges

- we didn't allow anti-parallel edges so we can add them in the residual network
- in real-world problems we can avoid anti-parallel edges by adding an extra vertex



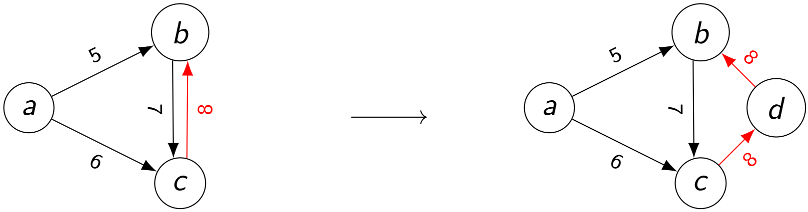
Networks with anti-parallel edges

- we didn't allow anti-parallel edges so we can add them in the residual network
- in real-world problems we can avoid anti-parallel edges by adding an extra vertex



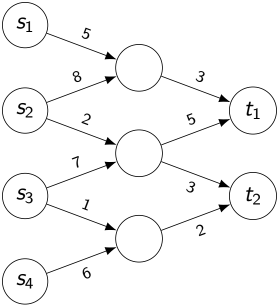
Networks with anti-parallel edges

- we didn't allow anti-parallel edges so we can add them in the residual network
- in real-world problems we can avoid anti-parallel edges by adding an extra vertex



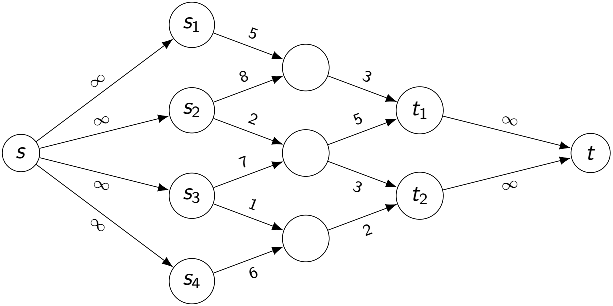
Networks with multiple sources and sinks

- the problem is more generic



Networks with multiple sources and sinks

- the problem is more generic
- we can reduce it to the single source single sink problem by adding two extra vertices
    - source s with infinite-capacity edges to previous sources
    - sink t with infinite-capacity edges from previous sinks



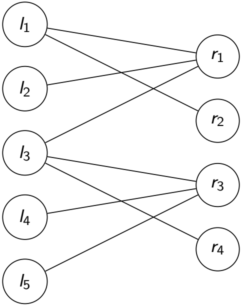
Agenda

Intuition

- we are given a team of people L and a set of jobs R
- each person can perform a specific set of jobs
- assign at most one job to each person in order to perform as many jobs as possible

Intuition

- we are given a team of people L and a set of jobs R
- each person can perform a specific set of jobs
- assign at most one job to each person in order to perform as many jobs as possible
- we can model the problem as a graph $G = (V, E)$ where $V = L \cup R$ and $E = \{(l_i, r_j) : \text{person } l_i \text{ can perform job } r_j\}$

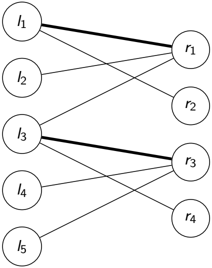


Formalism

- Given a graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq$ such that $\forall v \in V$, at most one edge of M is incident on v.
- A **maximum matching** is a matching of maximum cardinality.

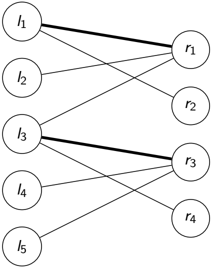
Formalism

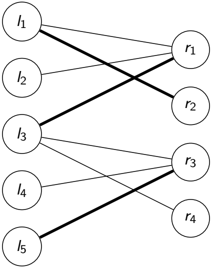
- Given a graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq$ such that $\forall v \in V$, at most one edge of M is incident on v.
- A **maximum matching** is a matching of maximum cardinality.



Formalism

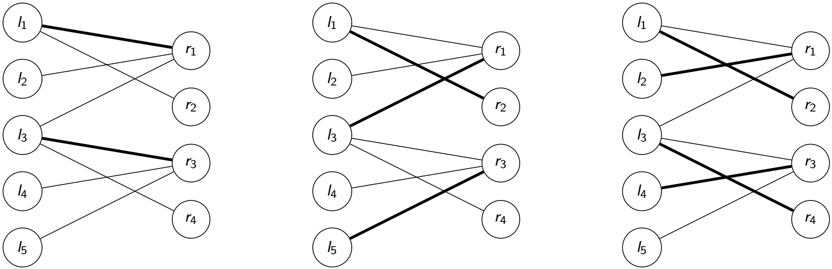
- Given a graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq$ such that $\forall v \in V$, at most one edge of M is incident on v.
- A **maximum matching** is a matching of maximum cardinality.





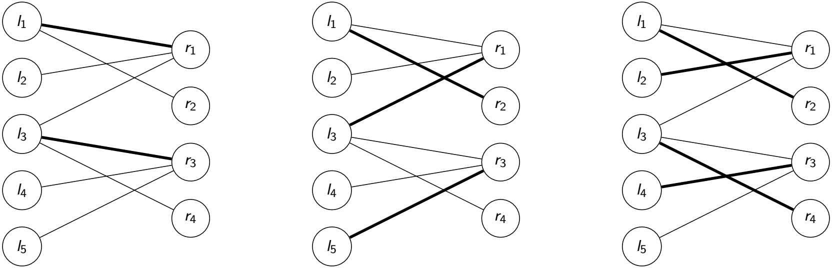
Formalism

- Given a graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq$ such that $\forall v \in V$, at most one edge of M is incident on v.
- A **maximum matching** is a matching of maximum cardinality.



Formalism

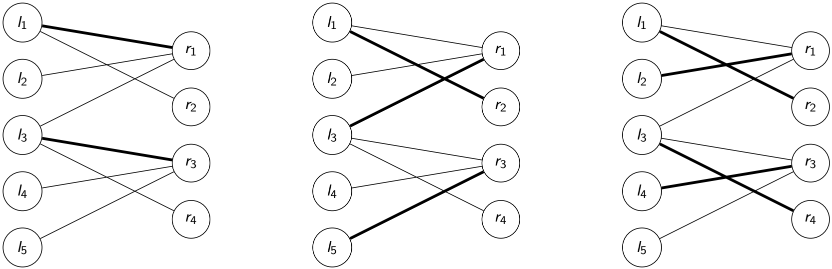
- Given a graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq$ such that $\forall v \in V$, at most one edge of M is incident on v.
- A **maximum matching** is a matching of maximum cardinality.



- We are interested in finding the **maximum matching** in **bipartite graphs**.

Formalism

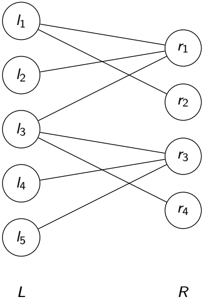
- Given a graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq$ such that $\forall v \in V$, at most one edge of M is incident on v.
- A **maximum matching** is a matching of maximum cardinality.



- We are interested in finding the **maximum matching** in **bipartite graphs**.
- Greedy approach doesn't work (see the figure on the left).

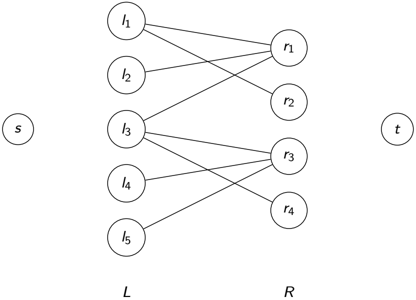
Maximum bipartite matching as a flow problem

- define the corresponding flow network $G' = (V', E')$



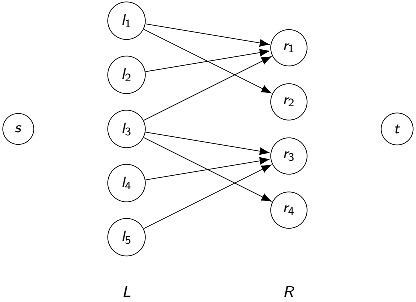
Maximum bipartite matching as a flow problem

- define the corresponding flow network $G' = (V', E')$
    - $V' = V \cup \{s, t\}$



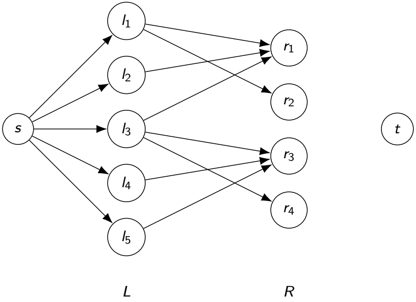
Maximum bipartite matching as a flow problem

- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\}$



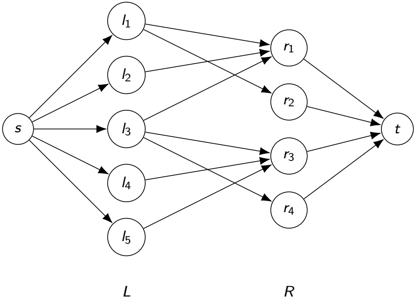
Maximum bipartite matching as a flow problem

- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\}$



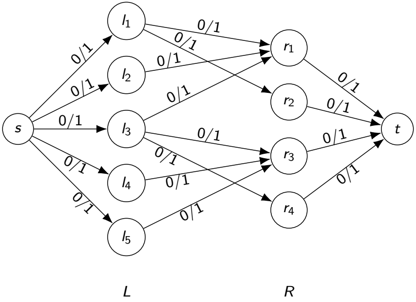
Maximum bipartite matching as a flow problem

- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$



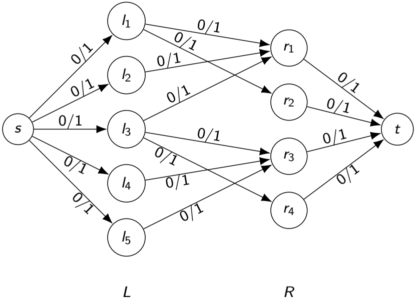
Maximum bipartite matching as a flow problem

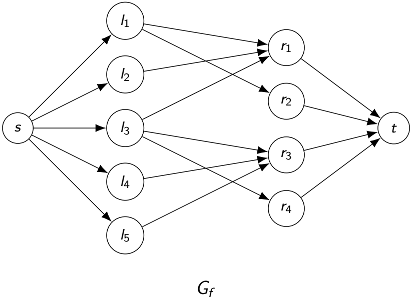
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$



Maximum bipartite matching as a flow problem

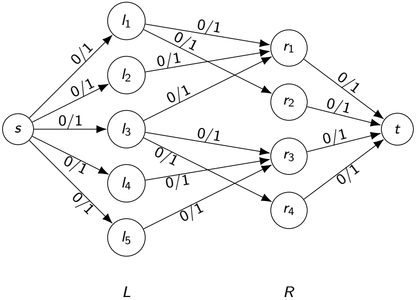
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

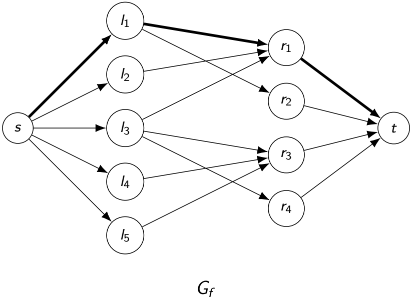




Maximum bipartite matching as a flow problem

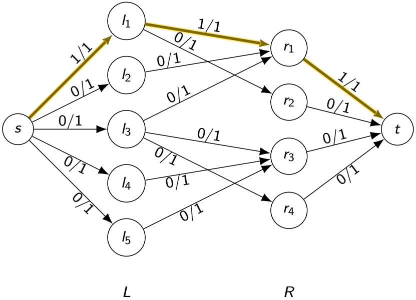
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

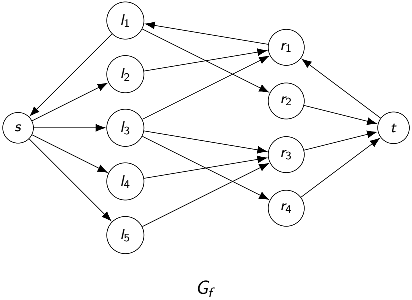




Maximum bipartite matching as a flow problem

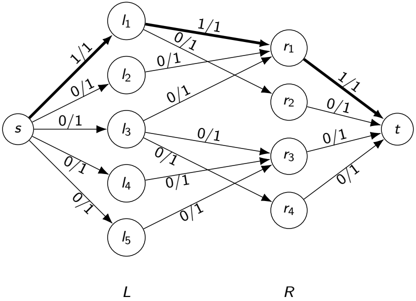
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

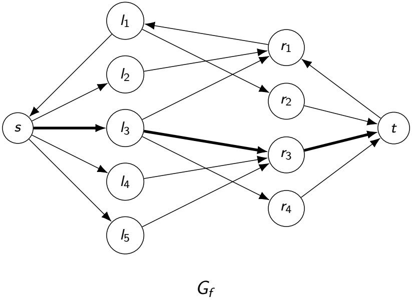




Maximum bipartite matching as a flow problem

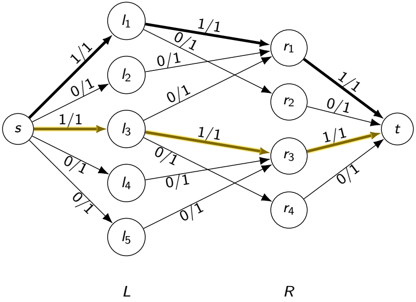
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

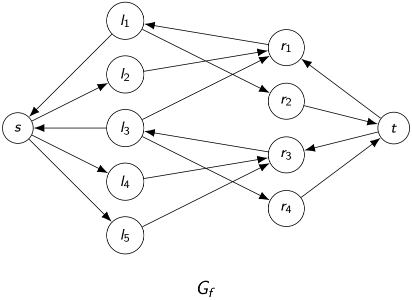




Maximum bipartite matching as a flow problem

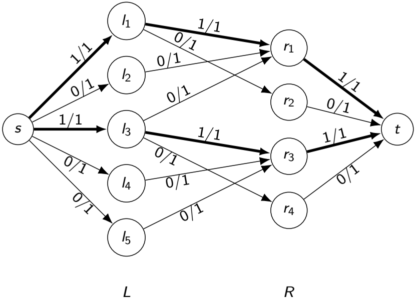
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

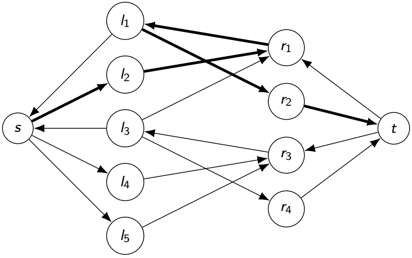




Maximum bipartite matching as a flow problem

- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

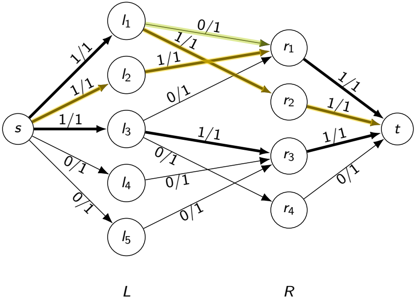


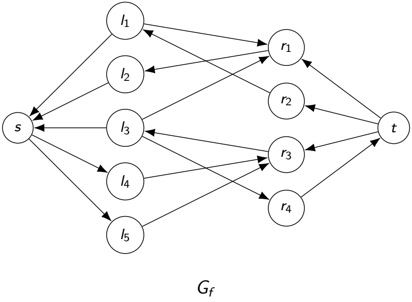


G_f

Maximum bipartite matching as a flow problem

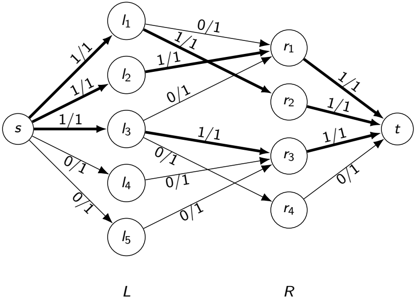
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

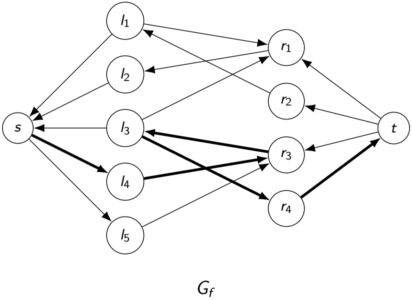




Maximum bipartite matching as a flow problem

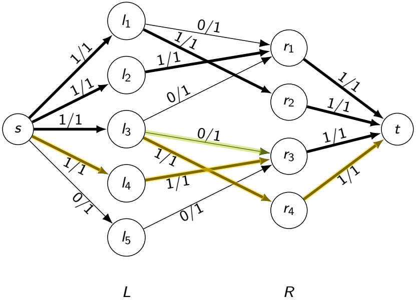
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

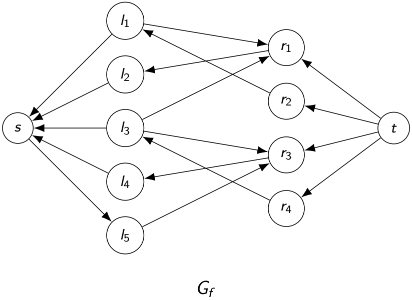




Maximum bipartite matching as a flow problem

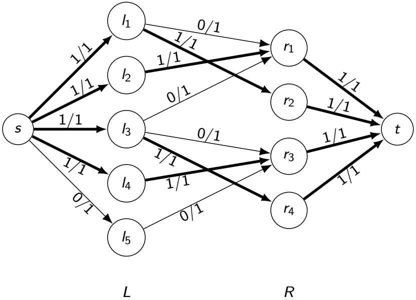
- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$

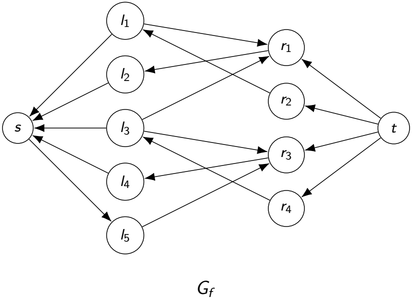




Maximum bipartite matching as a flow problem

- define the corresponding flow network $G' = (V', E')$
  - $V' = V \cup \{s, t\}$
  - $E' = \{(u, v) : (u, v) \in E\} \cup \{(s, u) : u \in L\} \cup \{(v, t) : t \in R\}$
  - $c(u, v) = 1, \forall (u, v) \in E'$





Maximum bipartite matching - analysis

- maximum bipartite matching $\leftrightarrow$ maximum flow

Maximum bipartite matching - analysis

- maximum bipartite matching $\leftrightarrow$ maximum flow
- Ford-Fulkerson algorithm complexity: $O(E \cdot |f^{\star}|)$

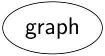
Maximum bipartite matching - analysis

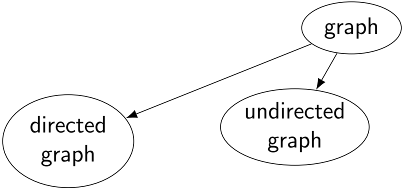
- maximum bipartite matching $\leftrightarrow$ maximum flow
- Ford-Fulkerson algorithm complexity: $O(E \cdot |f^\star|)$
- $|f^\star| \leq \min(|L|, |R|) \leq \dfrac{|V|}{2}$ (at best, we find a match for all people or for all jobs)

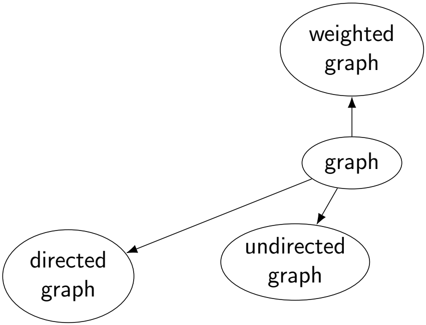
Maximum bipartite matching - analysis

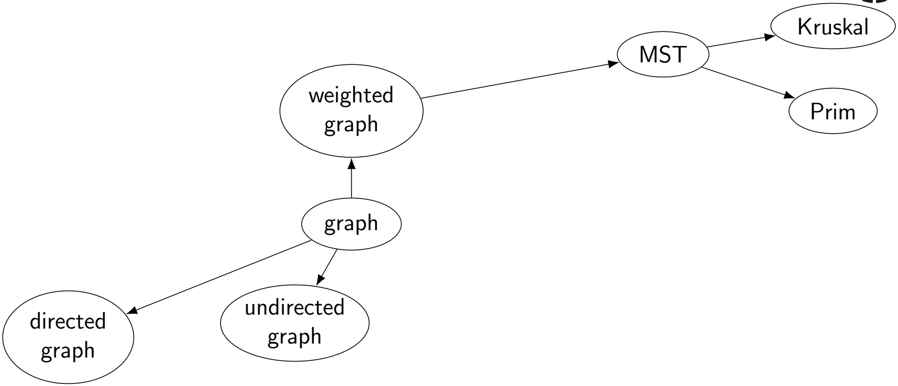
- maximum bipartite matching $\leftrightarrow$ maximum flow
- Ford-Fulkerson algorithm complexity: $O(E \cdot |f^\star|)$
- $|f^\star| \leq \min(|L|, |R|) \leq \dfrac{|V|}{2}$ (at best, we find a match for all people or for all jobs)
- complexity for maximum bipartite matching: $O(V \cdot E)$

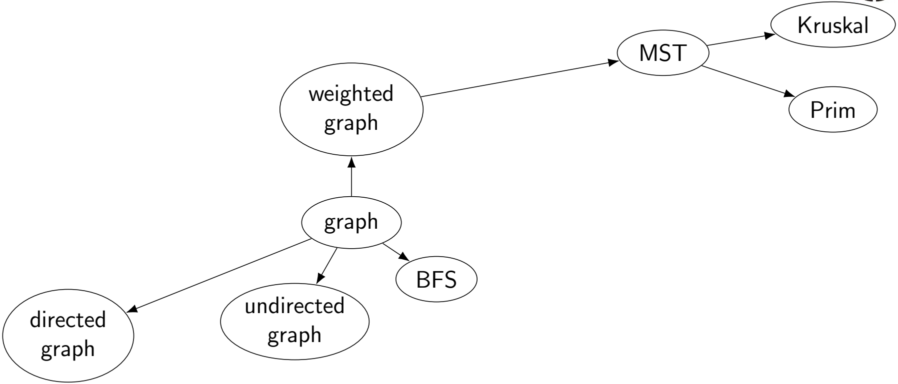
Agenda

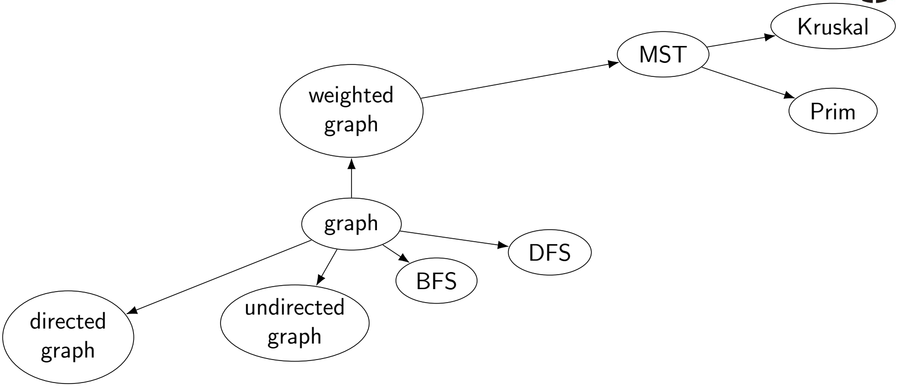

graph

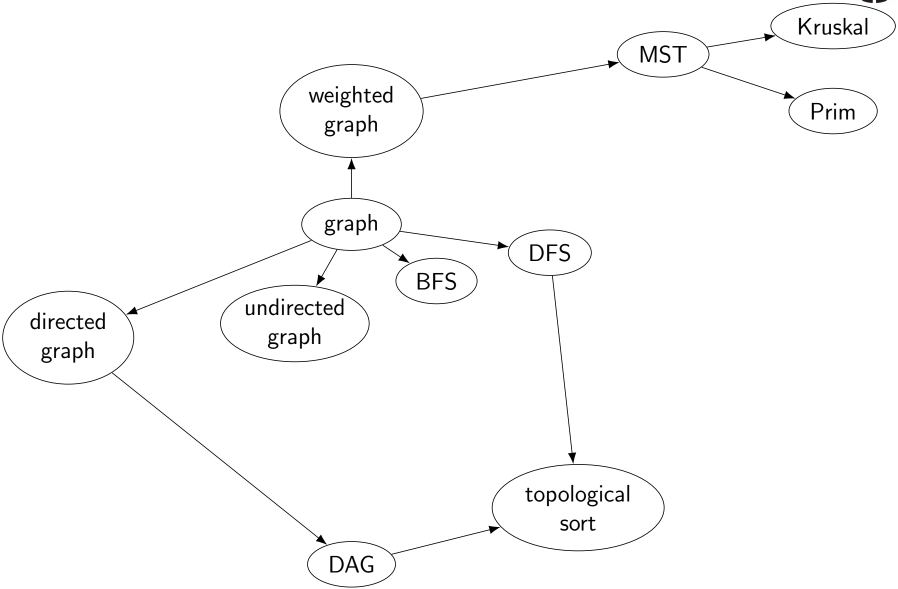

graph
directed graph
undirected graph

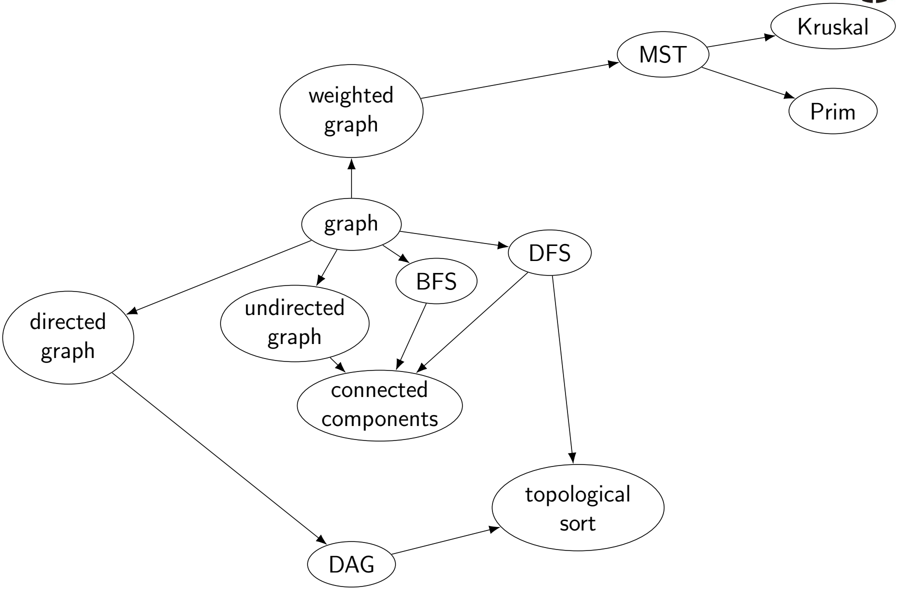

weighted graph
graph
directed graph
undirected graph

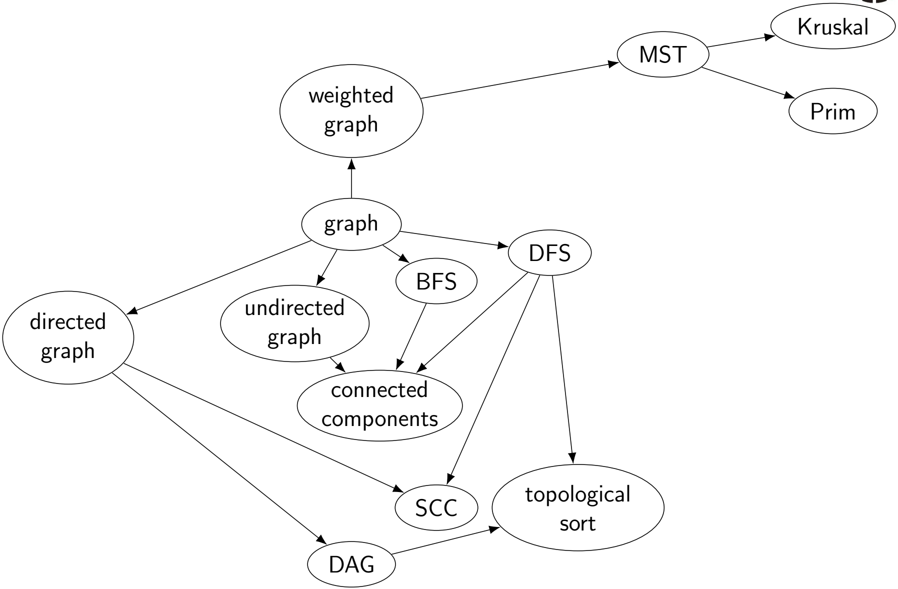

Kruskal
MST
weighted graph
Prim
graph
directed graph
undirected graph

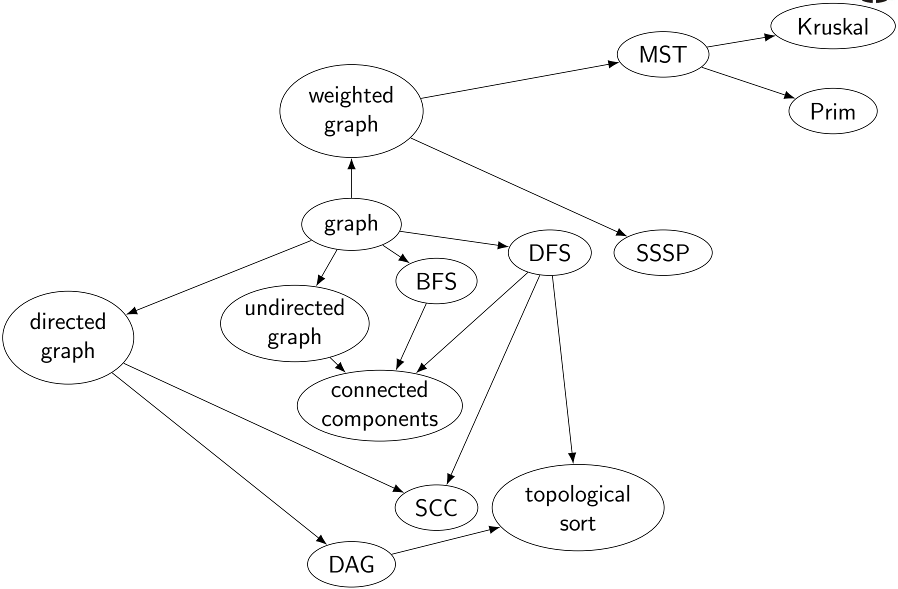

Kruskal
MST
weighted graph
Prim
graph
BFS
directed graph
undirected graph

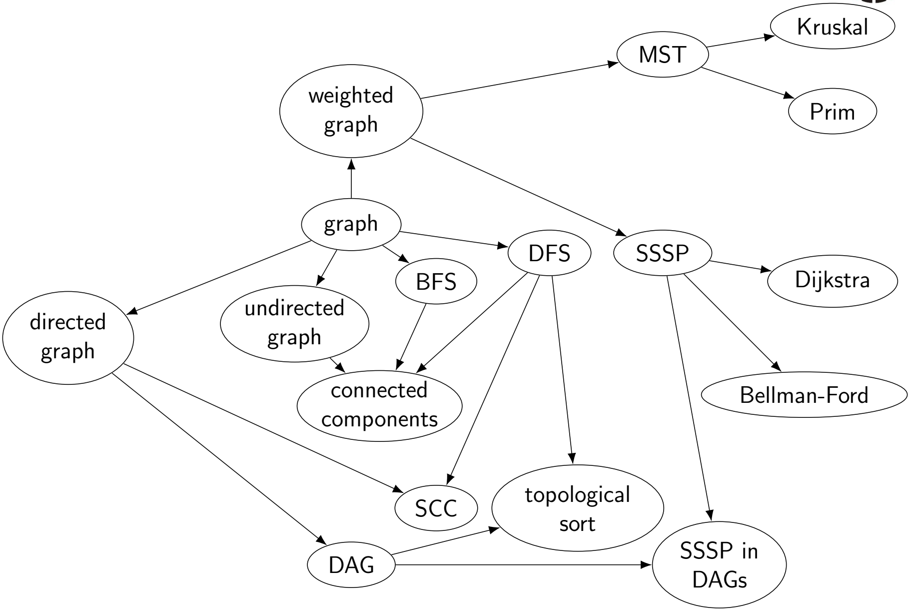

Kruskal
MST
weighted
graph
Prim
graph
DFS
BFS
directed
graph
undirected
graph

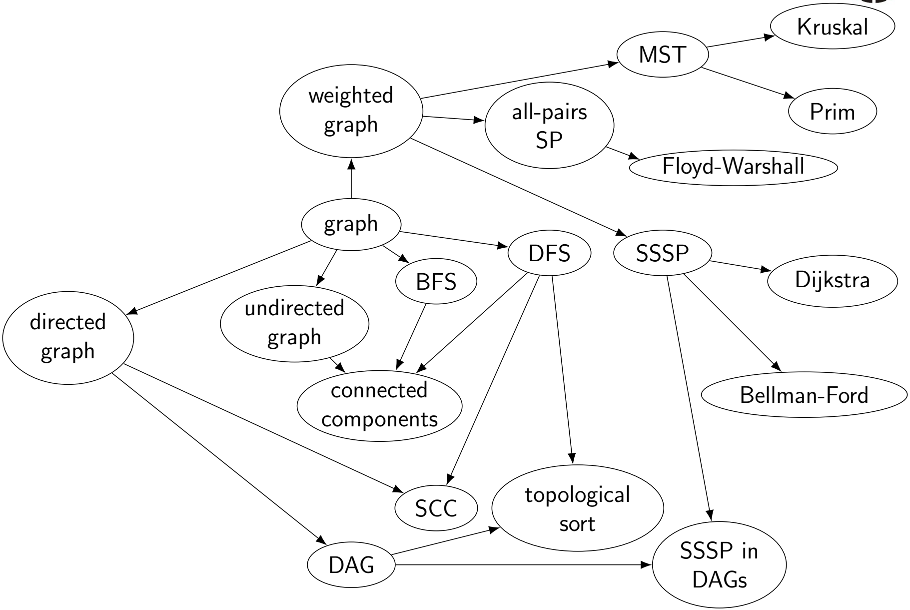

Kruskal
MST
weighted graph
Prim
graph
DFS
BFS
undirected graph
directed graph
topological sort
DAG

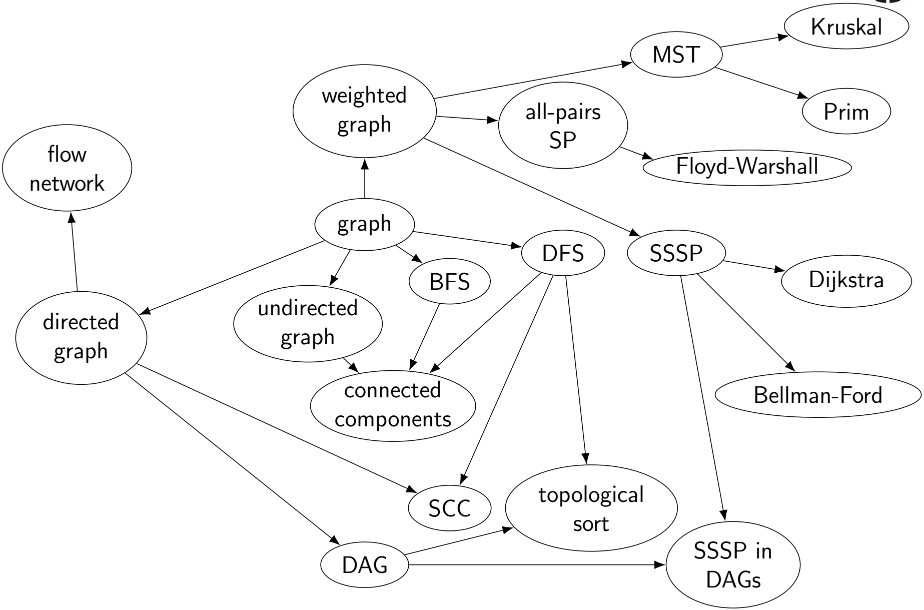

Kruskal
MST
weighted graph
Prim
graph
DFS
BFS
undirected graph
directed graph
connected components
topological sort
DAG

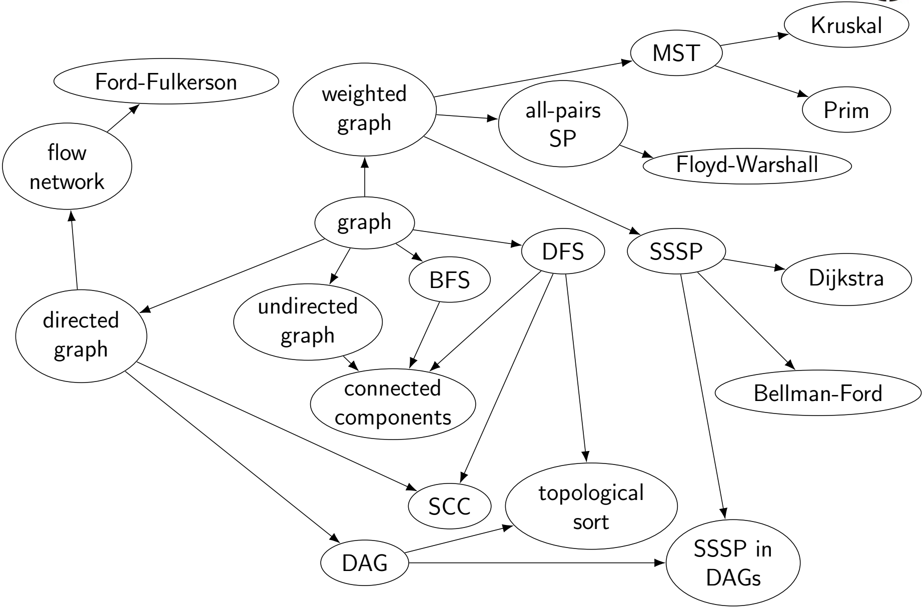

Kruskal
MST
weighted graph
Prim
graph
DFS
BFS
undirected graph
directed graph
connected components
SCC
topological sort
DAG

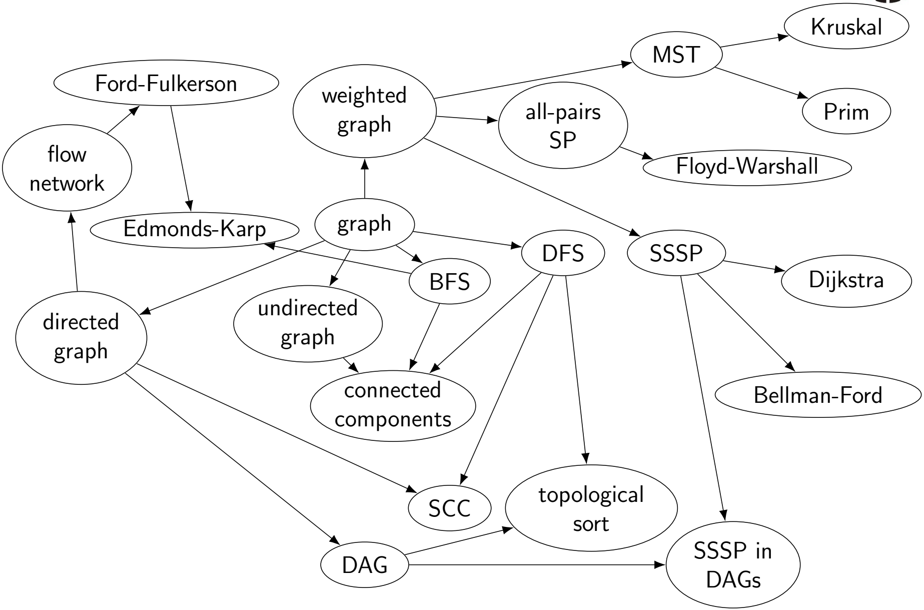

Kruskal
MST
weighted graph
Prim
graph
DFS
SSSP
BFS
directed graph
undirected graph
connected components
SCC
topological sort
DAG

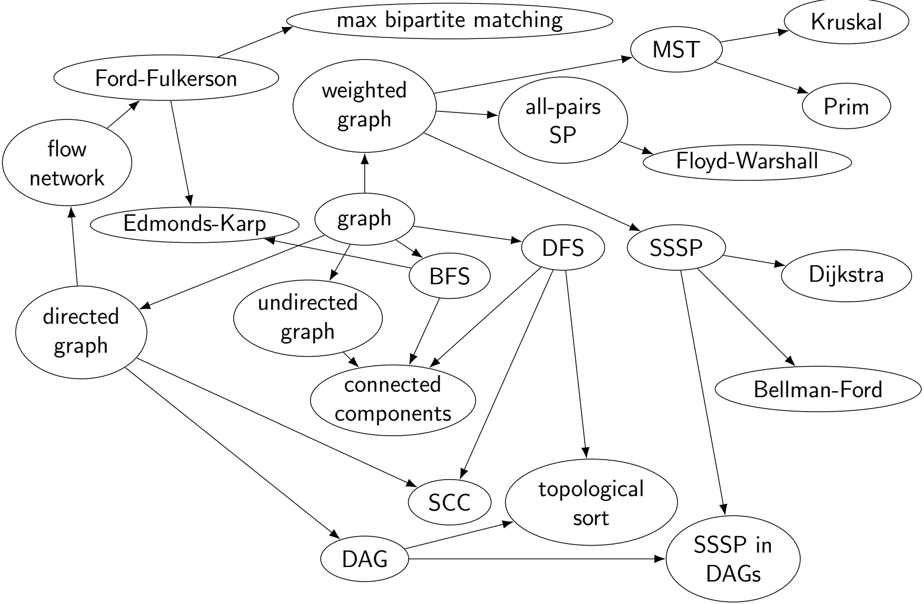

Kruskal
MST
weighted graph
Prim
graph
DFS
SSSP
BFS
Dijkstra
directed graph
undirected graph
connected components
Bellman-Ford
SCC
topological sort
DAG
SSSP in DAGs


weighted graph
MST
Kruskal
Prim
all-pairs SP
Floyd-Warshall
graph
DFS
SSSP
Dijkstra
BFS
undirected graph
directed graph
connected components
Bellman-Ford
SCC
topological sort
SSSP in DAGs
DAG


Kruskal
MST
weighted graph
all-pairs SP
Prim
Floyd-Warshall
flow network
graph
DFS
SSSP
BFS
Dijkstra
directed graph
undirected graph
connected components
Bellman-Ford
SCC
topological sort
SSSP in DAGs
DAG


Kruskal
MST
Ford-Fulkerson
weighted graph
Prim
all-pairs SP
flow network
Floyd-Warshall
graph
DFS
SSSP
BFS
Dijkstra
directed graph
undirected graph
connected components
Bellman-Ford
topological sort
SCC
SSSP in DAGs
DAG


Kruskal
MST
Ford-Fulkerson
weighted graph
Prim
all-pairs SP
flow network
Floyd-Warshall
Edmonds-Karp
graph
DFS
SSSP
BFS
Dijkstra
directed graph
undirected graph
connected components
Bellman-Ford
SCC
topological sort
DAG
SSSP in DAGs


max bipartite matching
Kruskal
MST
Ford-Fulkerson
weighted graph
Prim
all-pairs SP
flow network
Floyd-Warshall
Edmonds-Karp
graph
DFS
SSSP
BFS
Dijkstra
directed graph
undirected graph
connected components
Bellman-Ford
SCC
topological sort
DAG
SSSP in DAGs

Agenda

1. Maximum Flow concepts
2. The Ford-Fulkerson method
3. Maximum bipartite matching
4. Graphs recap
5. Exam info

Location

- Romanian series A:
  `https://moodle1.cs.utcluj.ro/course/view.php?id=292`
- Romanian series B:
  `https://moodle2.cs.utcluj.ro/course/view.php?id=292`
- English series:
  `https://moodle3.cs.utcluj.ro/course/view.php?id=292`

- the three server will be clones of the main moodle server
  - make sure your login works on them 48h before the exam

Format

- several Moodle quizes
  - multiple choice - automatically graded
  - short answer - automatically graded
  - fill in the gaps - automatically graded
  - essay (text/images) - manually graded
- for each question/part you will have a fixed time interval
- sequential access (once you answer or skip a question, you won't be able to return to it)

Structure and grading

- 30% lab grade
- 20% course quizzes
- 50% final exam
  - 30% part 1 - questions resembling the course quizzes
  - 40% part 2 - questions focused on tracing the studied algorithms
  - 40% part 3 - questions focused on designing and analyzing algorithms
    - explain the solution and (informally) justify the correctness
    - write the pseudocode (without defining data structures)
    - analyze the algorithm complexity

Bibliography

- Cormen, Thomas H., et al., *"Introduction to algorithms."*, MIT press, 2009, cap. 26