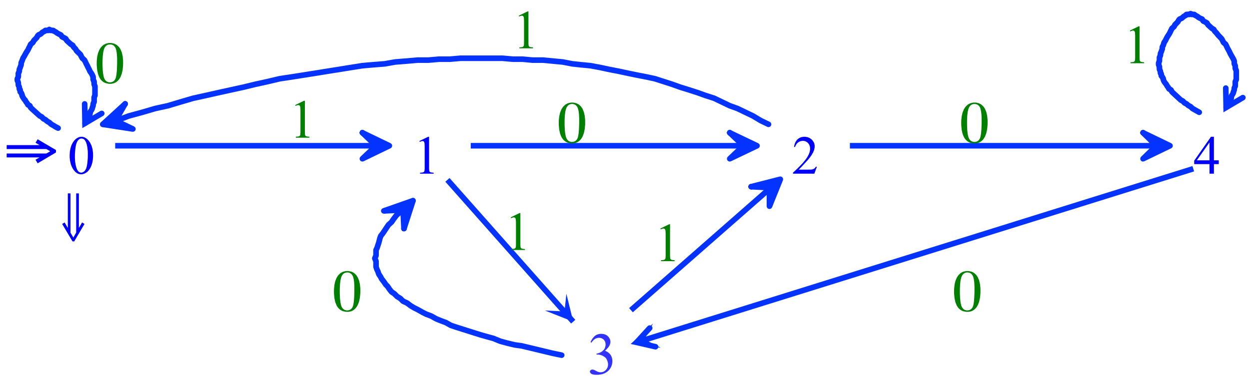
Application of FIND/UNION :

testing the equivalence of deterministic finite automata

Deterministic finite automata

$A = (Q, \Sigma, i, F, \delta)$

Q	states	finite set
Σ	alphabet	finite set
i	initial state	$i \in Q$
F	final (accepting) states	$F \subseteq Q$
δ	transition function	$\delta : Q \times \Sigma \to Q$



$L(A) = \{ x \in \Sigma^* \mid x$ label of the path from i to $t \in T\}$

Algorithmes sur automates

- pruning
- automaton → regular expression
- regular expression → automaton (e.g. Thompson)
- determinisation
- minimisation
- testing « $L(a) = \emptyset$? » or « $L(a) = \mathcal{A}^*$? »
- Constructing an automaton A such that
  - $L(A) = \Sigma^* - L(B)$
  - $L(A) = L(B) \cup L(C)$
  - $L(A) = L(B) \cap L(C)$
  - $L(A) = L(B)^*$
  - **...**
- testing $L(A) = B$
- testing equivalence : $L(A) = L(B)$?
- **...**

Equivalence of automata

Test $L(A_1) = L(A_2)$ $\qquad$ $n = |Q_1| + |Q_2|$

- if A_1 and A_2 are minimal
  $L(A_1) = L(A_2)$ iff $A_1 = A_2$

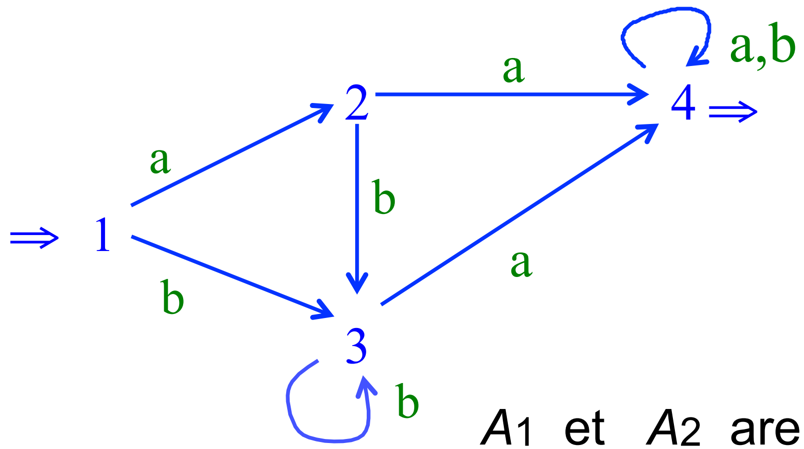
- par minimisation
  -- minimise A_1 et A_2 (Hopcroft's or Moore's algorithm)
  -- then test $A_1 = A_2$

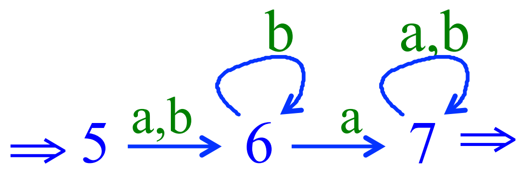
Time (with Hopcroft's) : $O(|\Sigma| \cdot n \cdot \log(n))$

- direct test
  use UNION / FIND

Time : $O(|\Sigma| \cdot n \cdot \alpha(n))$

Example





A_1 et A_2 are equivalent $\leftrightarrow$ i_1 and i_2 are equivalent (in disjoint union)

Construction of the equivalence :

1 | 2 | 3 | 4 | 5 | 6 | 7

1 5 | 2 | 3 | 4 | 6 | 7

1 5 | 2 6 | 3 | 4 | 7

1 5 | 2 3 6 | 4 | 7

1 5 | 2 3 6 | 4 7

A_1 and A_2 are equivalent

Algorithm

```
procedure Equiv( q1, q2 ) ;
begin
   if   one of q1, q2 is final but not the other   then
      « A1 and A2 are not equivalent »
   else {
      if FIND(q1) ≠ FIND(q2)   then {
         UNION(q1, q2) ;
         for all a ∈ Σ  do
               Equiv( δ1(q1, a), δ2(q2, a)) ;
      }
end
```

Initial call : Equiv(i_1, i_2)

Time $O(|A| \cdot n \cdot \alpha(n))$ ($n = |Q_1| + |Q_2|$)

Maximum flow in networks

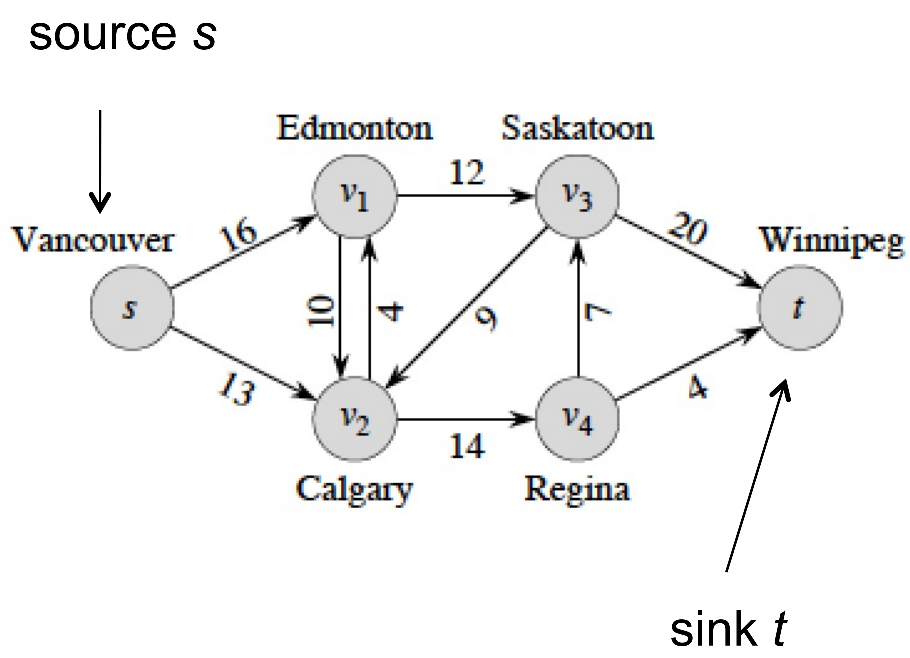
Flow network

Directed weighted graph $G = (S, A, c)$

$c(p, q)$: capacity of edge (p, q)

Examples

- Water systems
- Production lines
- Traffic roads
- Transportation of goods
- Electricity
- etc.



Conditions

Capacity $c : S \times S \rightarrow \mathbf{R}$ with $c(p, q) \geq 0$

if $(p, q) \neq A$, then assume $c(p, q) = 0$

Flow $f : S \times S \rightarrow \mathbf{R}$

source $s \in S$, sink $t \in S$

Accessiblity

all nodes appear on a path from s to t

Capacity constraint

for all $p, q \in S$, $f(p, q) \leq c(p, q)$

Conditions (cont)

Anti-symmetry

for all p, $q \in S$, $f\,(q, p)$ **=** $-\,f\,(p, q)$

$$p \xrightarrow{17} q$$

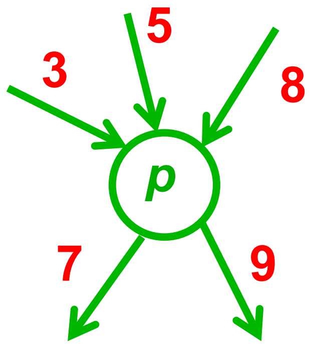
positive flow

$$p \xleftarrow{-17} q$$

negative flow

Flow conservation

for all $p \in S \setminus \{s,t\}$, $\boldsymbol{\Sigma}\,\mathbf{(}\,f\,(p, q) \mid q \in S\,\mathbf{)} = 0$



Flow

Flow value:

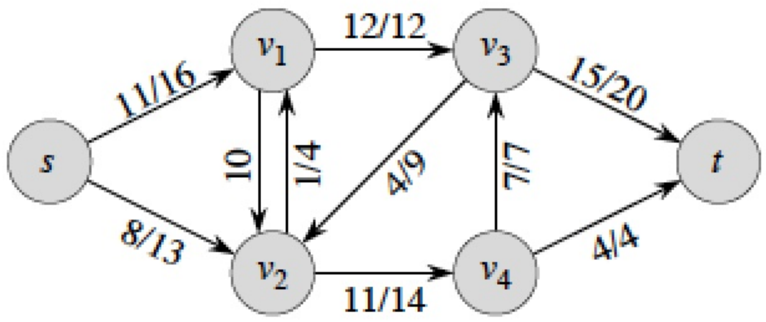
$$| f | = \sum (f (s, q) \mid q \in S)$$

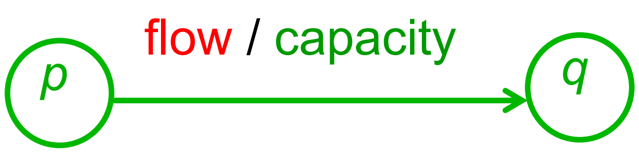
what flows out of the source

Property:

$$| f | = \sum (f (p, t) \mid p \in S)$$

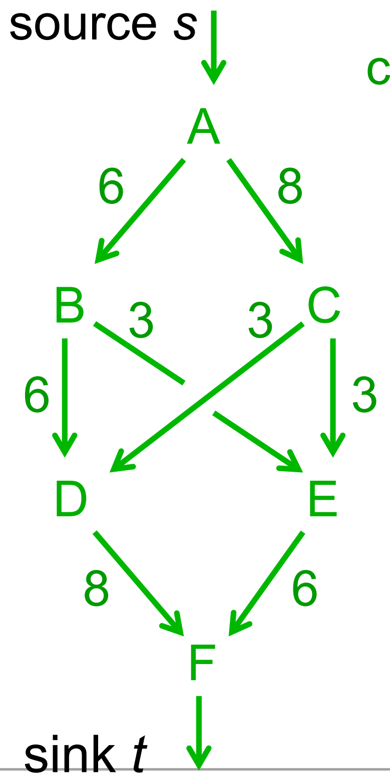
what arrives to the sink





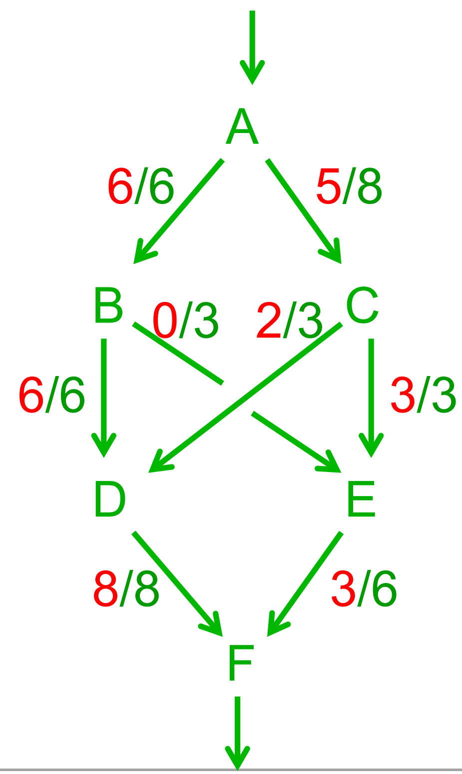
Problem

Flow network $G = (S, A, c)$
Compute the **maximum flow**,
i.e. the flow f of maximal value



capacities

maximum flow ?
$|\, f \,| = 11$

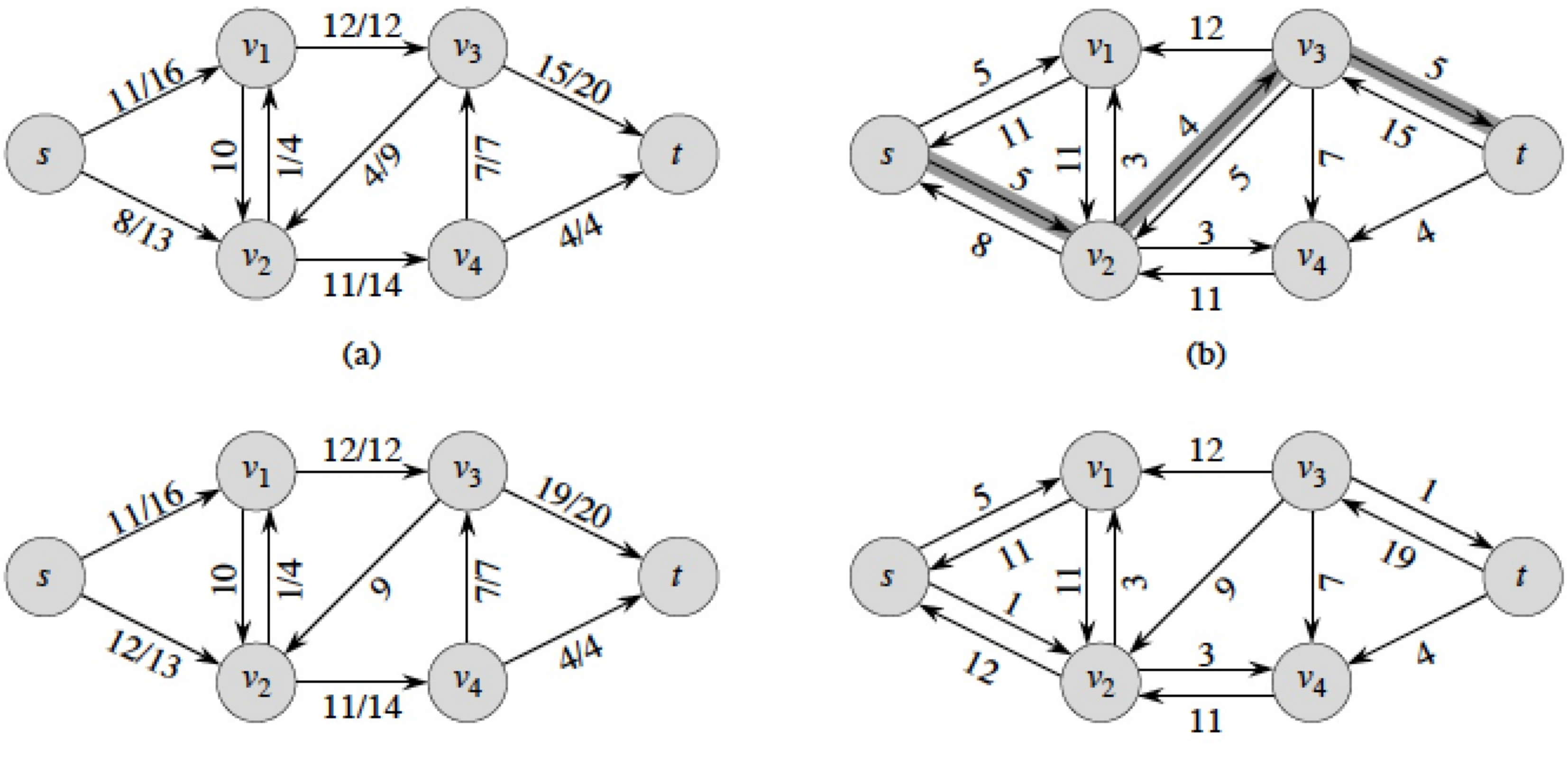


The Ford-Fulkerson method

initialize flow f to 0 ;
while there exists an augmenting path from s to t **do**
    augment flow f along this path ;
return f

Augmenting path

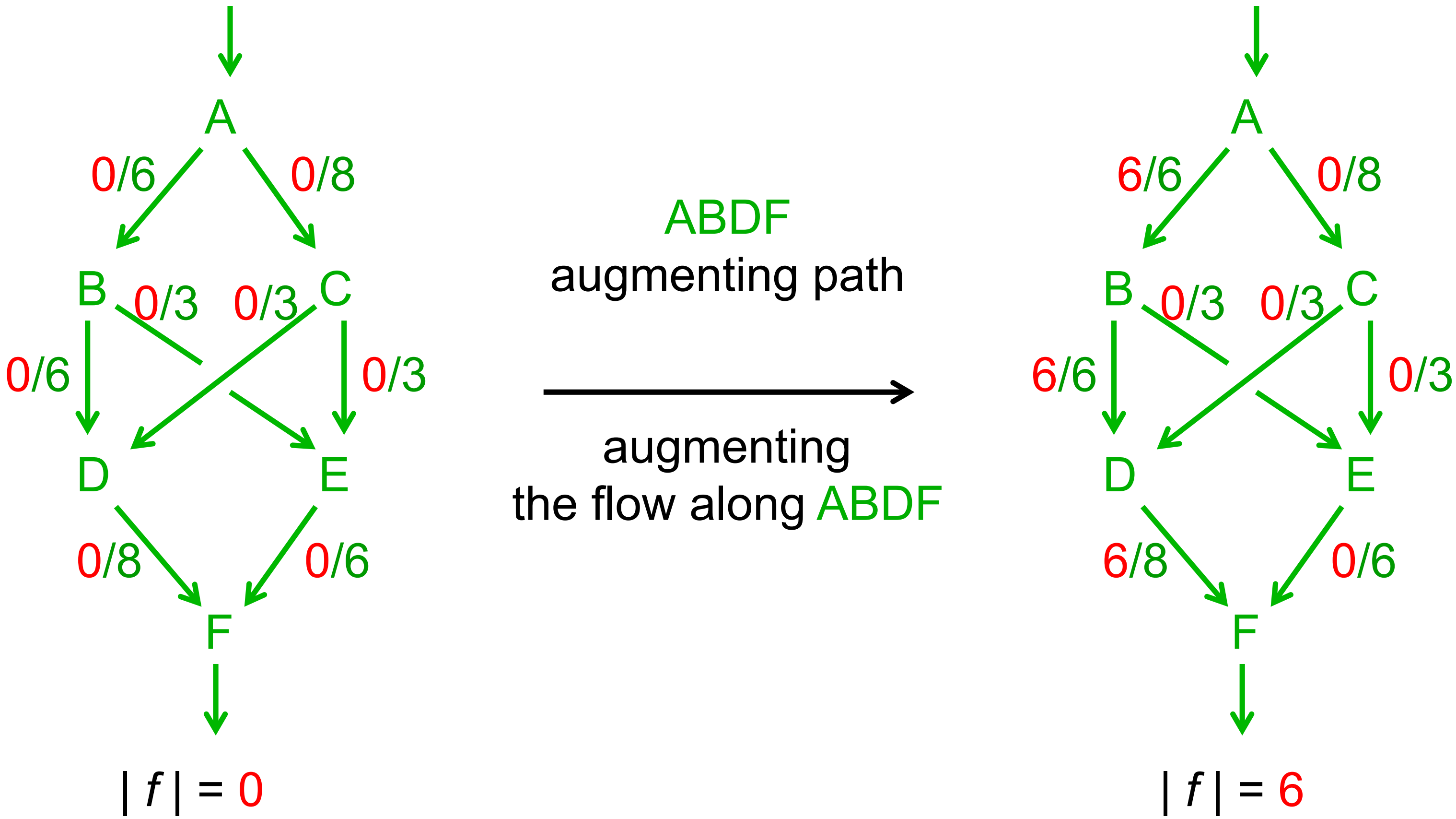
(b), (d): residual networks



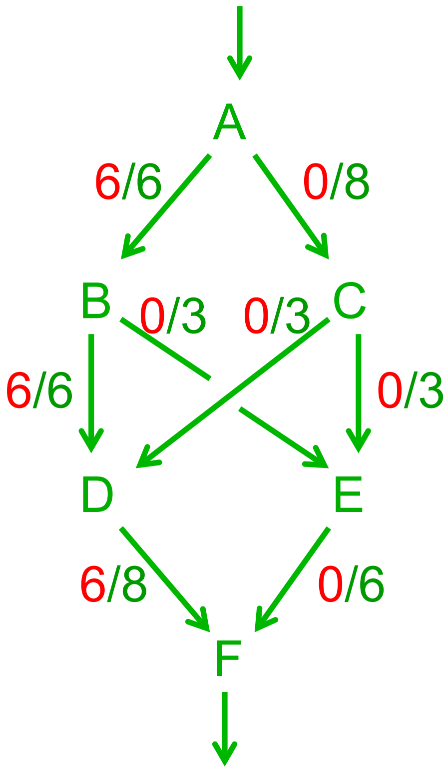
(a) (b) (c) (d)

An augmenting path is a simple path in the residual network

Example


A
0/6
0/8
B
0/3
0/3
C
0/6
0/3
D
E
0/8
0/6
F
| f | = 0
ABDF
augmenting path
augmenting
the flow along ABDF
A
6/6
0/8
B
0/3
0/3
C
6/6
0/3
D
E
6/8
0/6
F
| f | = 6

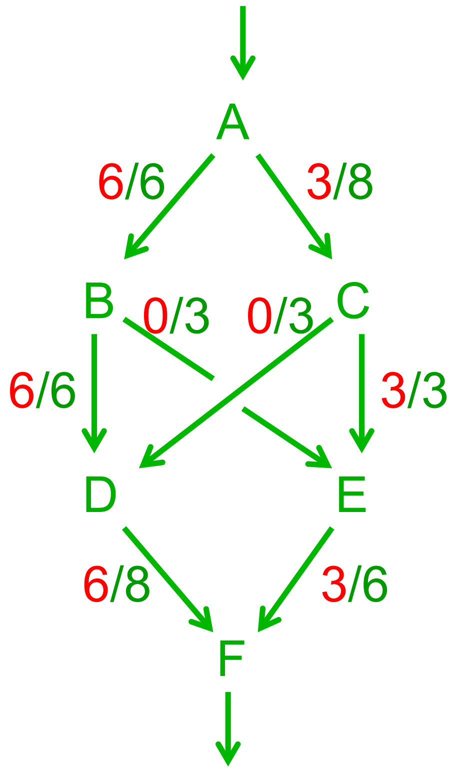
Example (cont)



| f | = 6

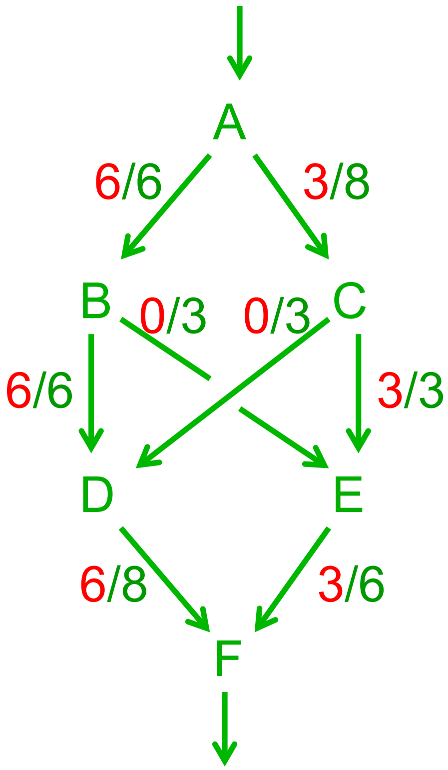
ACEF
augmenting path

augmenting
the flow along ACEF



| f | = 9

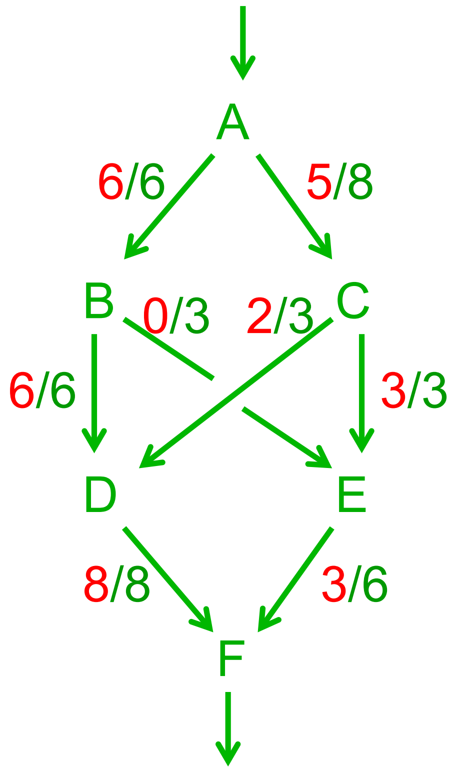
Example (cont)



$|f| = 9$

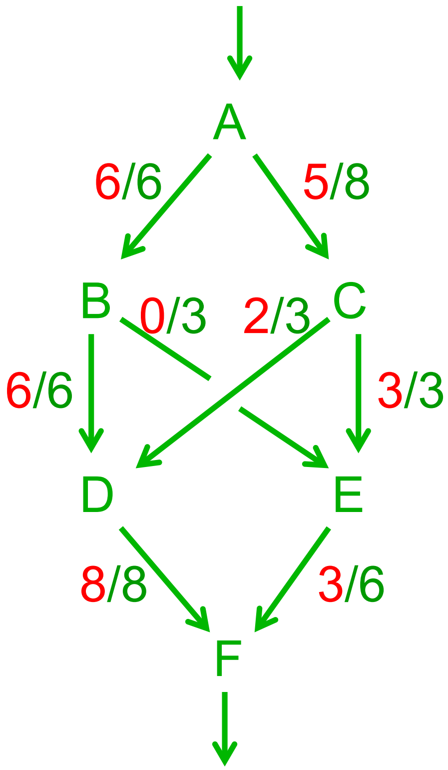
ACDF
augmenting path

augmenting
the flow along ACDF



$|f| = 11$

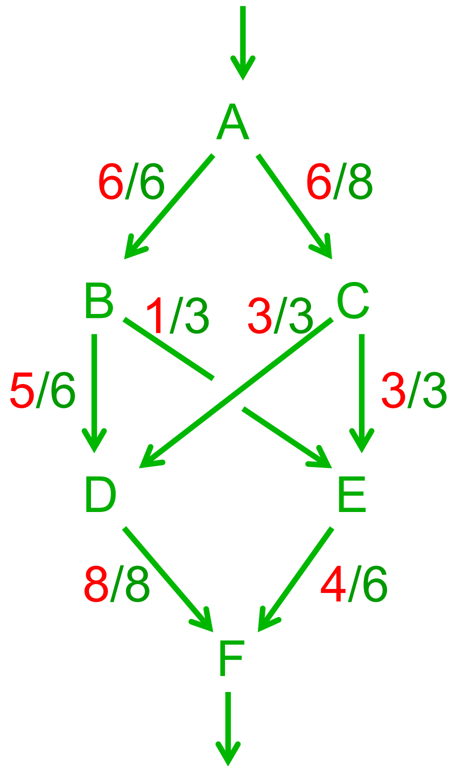
Example (cont)



| f | = 11

ACDBEF
augmenting path

augmenting
the flow along ACDBEF



| f | = 12
maximum

Cut

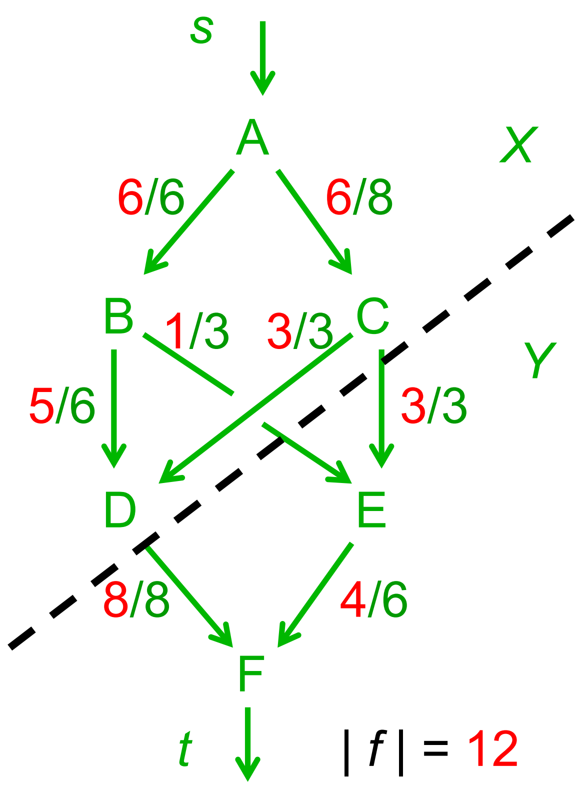
(X, Y) cut of $G = (S, A, c)$:
(X, Y) partition of S such that $s \in X$, $t \in Y$

capacity of the cut

$$c(X, Y) = \Sigma\, (\, c(x, y) \mid x \in X, y \in Y\,)$$

flow

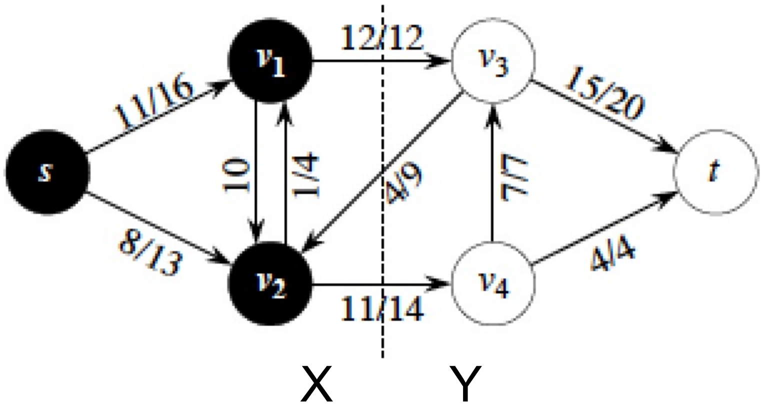
$$f(X, Y) = \Sigma\, (\, f(x, y) \mid x \in X, y \in Y\,)$$



X = {A,B,C,D} Y = {E,F} $c(X,Y)$ = 14 $f(X,Y)$ = 12

Cut

Note that the flow from Y to X ***is*** counted negatively, but the capacity does ***not*** take into account edges from Y to X



$c(X, Y) = 26 \quad f(X, Y) = 19$

Properties

Properties Let (X, Y) be a cut. Then

1 $f(X, Y) = | f |$

2 $f(X, Y) \leq c(X, Y)$

The maximum flow is bounded by the minimal capacity of a cut

Theorem (max-flow min-cut theorem)
The following conditions are equivalent:

1 f is a maximal flow

2 there is no augmenting paths

3 $| f | = c(X', Y')$ for some cut (X', Y')

Optimal cut

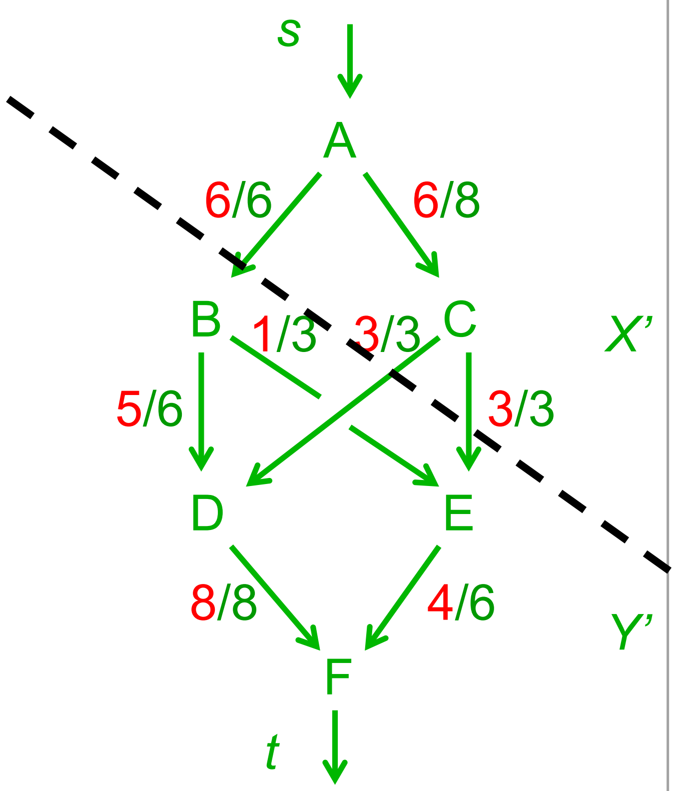
$X' = \{A,C\}$

$Y' = \{B,D,E,F\}$

$c(X',Y') = 12$

(X',Y') of minimal capacity

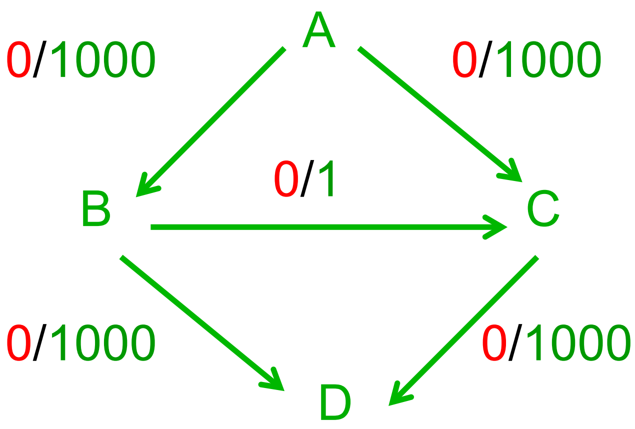
$f\,(X',Y') = 12$ is the maximum flow



$|\,f\,| = 12$

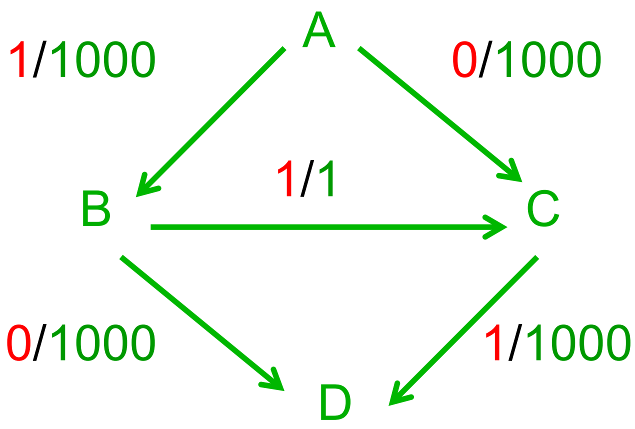
Example

The number of iterations depends on the choice of the paths



Example

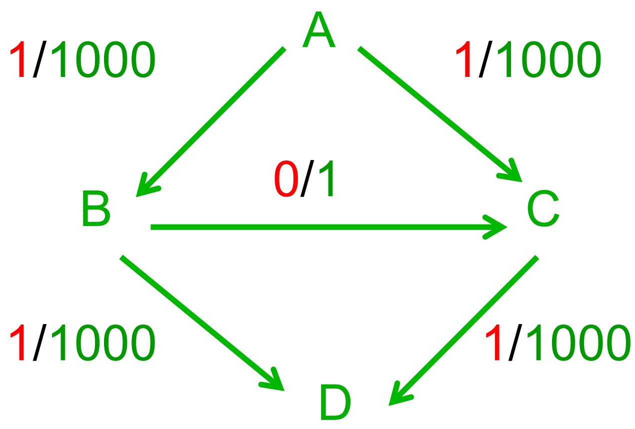
The number of iteration depends on the choice of the paths



augmentation	path
1	ABCD

Example

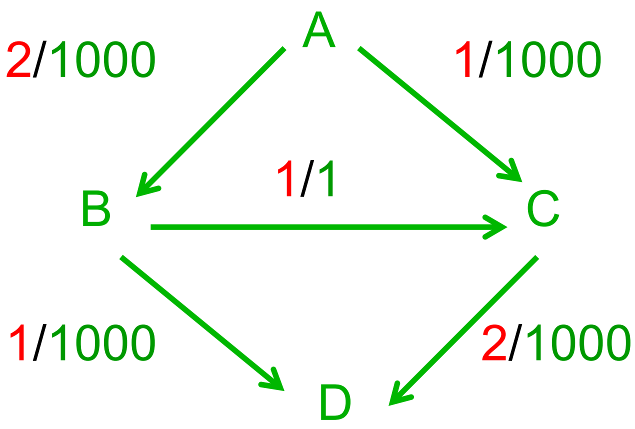
The number of iteration depends on the choice of the paths



augmentation	path
1	ABCD
1	ACBD

Example

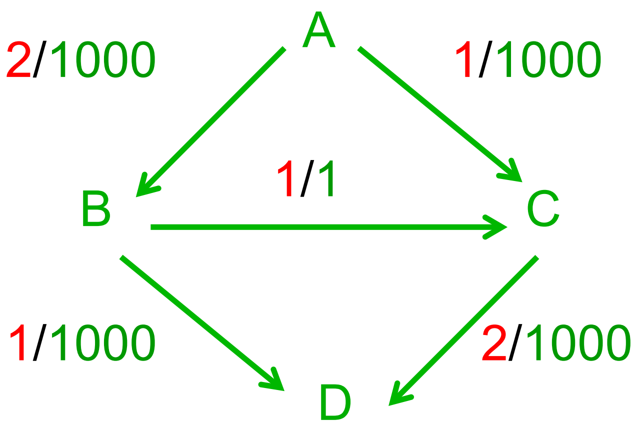
The number of iteration depends on the choice of the paths

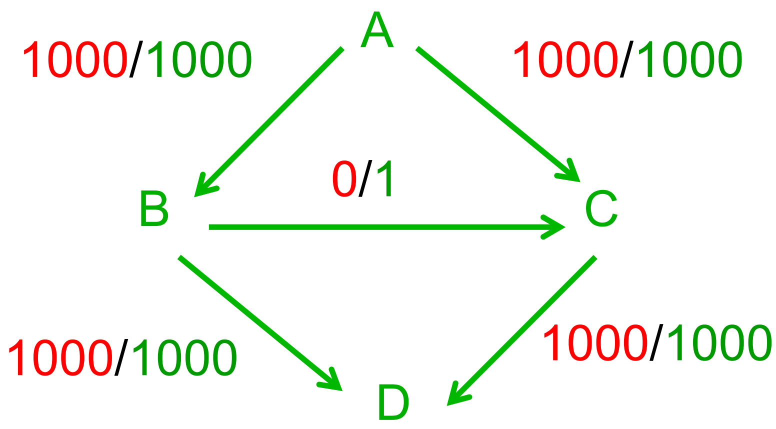


augmentation	path
1	ABCD
1	ACBD
1	ABCD
etc.	

Example

The number of iteration depends on the choice of the paths





augmentation	path
1	ABCD
1	ACBD
1	ABCD
etc.	

augmentation	path
1000	ABD
1000	ACD
maximum flow	

Strategy **1**

To augment the flow:
choose the shortest augmenting path (using BFT)

Theorem

Computing the maximum flow using this strategy requires at most $|S| \cdot |A|$ augmentations.
The running time is $O(|S| \cdot |A|^2)$

This strategy is known as the *Edmonds-Karp algorithm*

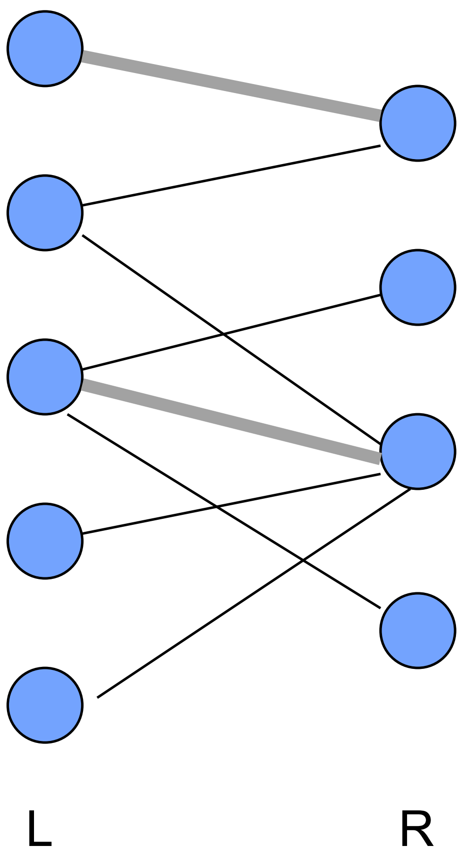
Other strategies

Push-relabel algorithm : $O(|S|^2 \cdot |A|)$

Relabel-to-front algorithm : $O(|S|^3)$

Maximal Bipartite Matching

Maximal matching

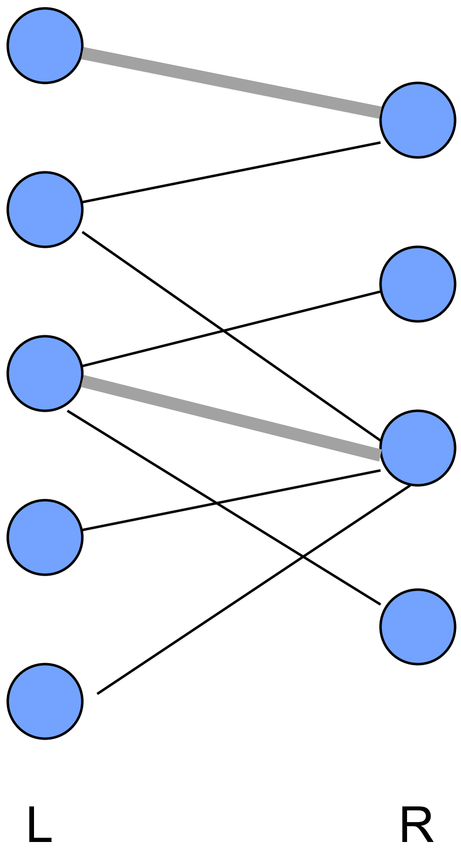


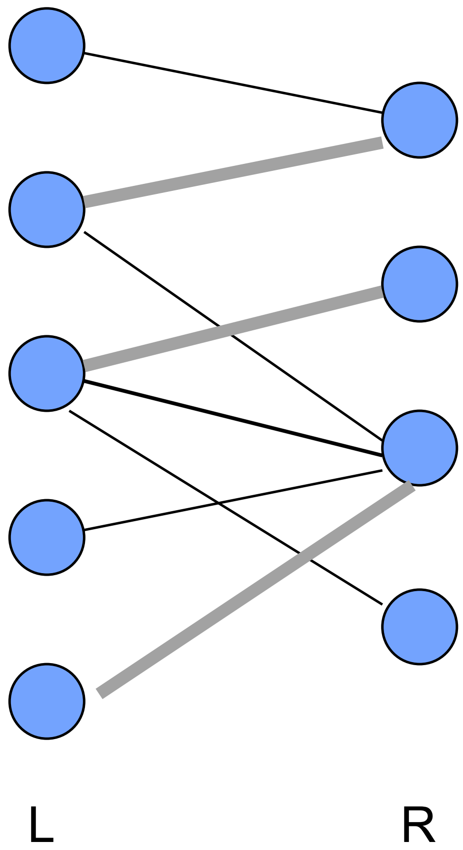
Bipartite graph $G = (S, A)$, $S = L \uplus R$, and $\forall (p,q) \in A$, $p \in L$ et $q \in R$

Matching: $C \subseteq A$ such that for all $p \in S$ $\exists$ at most one edge in A incident to p (i.e. having p as one of the endpoints)

Maximal matching: matching with the maximal number of edges (**NB**: maximality here does not mean maximality by inclusion!)

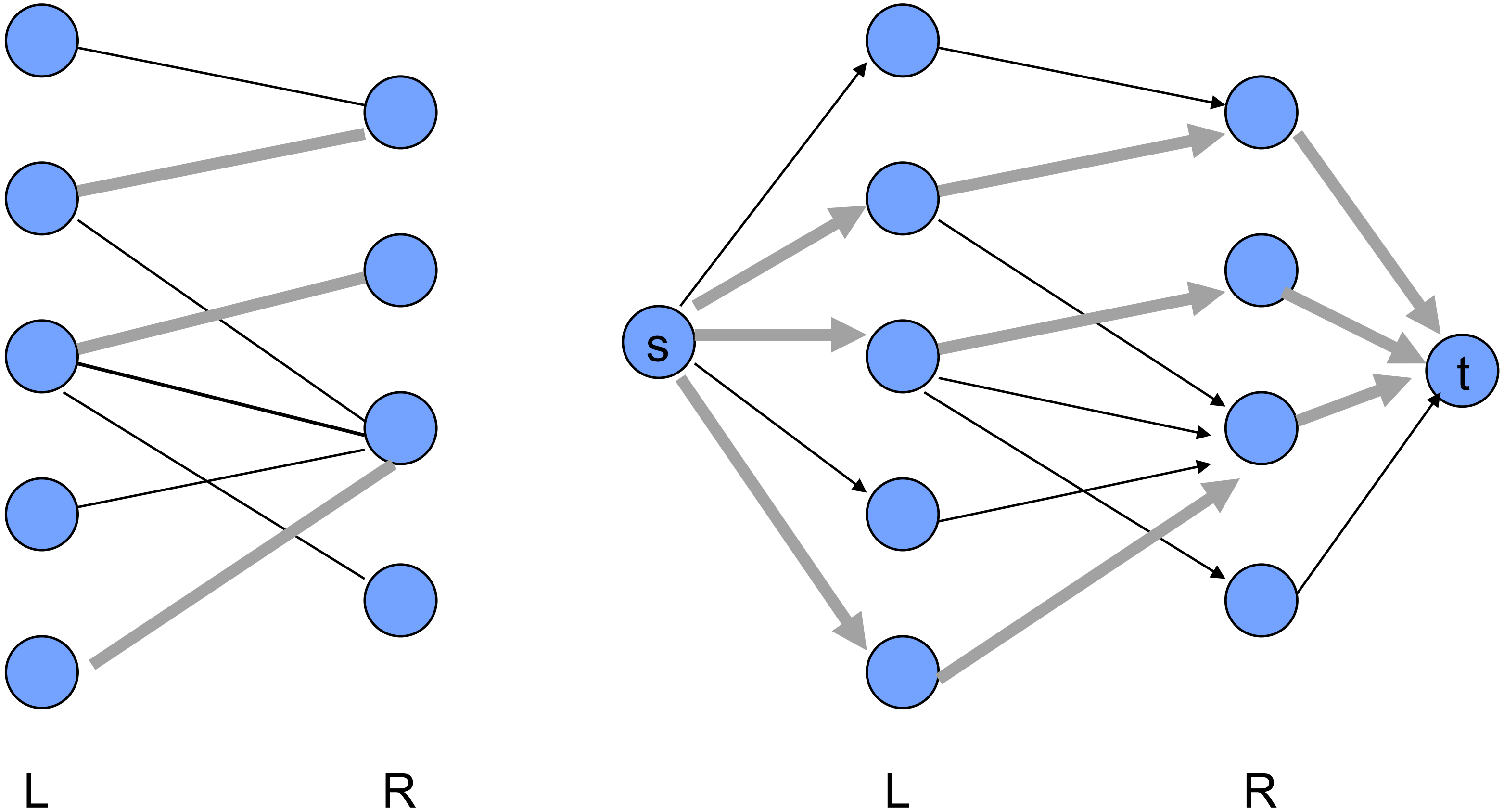
Maximal matching





maximal matching

Encoding by maximum flow



Encoding of a bipartite graph by a directed graph. Maximal matching and corresponding maximum flow. Each edge has capacity 1.

Lemma: Let $G = (S = L \uplus R, A)$ be a bipartite graph and G' be the corresponding directed graph. If C is a matching of G, then there exists a flow in G' of value $|C|$. Conversely, if f is a flow in G' (of an integer value), then there exists a matching in G of cardinality $|f|$.

The Ford-Fulkerson method computes a maximum flow maximal of integer value for each edge. It determines a maximal matching.

The complexity can be shown to be $O(|S| \cdot |A|)$.

Improvements have been proposed: for example, the Hopcroft-Karp algorithm works in time $O(|S|^{1/2} \cdot |A|)$