Network flow problems

Network flow problems

- Introduction of:

  network, max-flow problem

  capacity, flow

- Ford-Fulkerson method

  pseudo code, residual networks, augmenting paths

  cuts of networks

  max-flow min-cut theorem

  example of an execution

  analysis, running time,

  variations of the max-flow problem

Introduction – network

Practical examples of a network

- liquids flowing through pipes

- parts through assembly lines

- current through electrical network

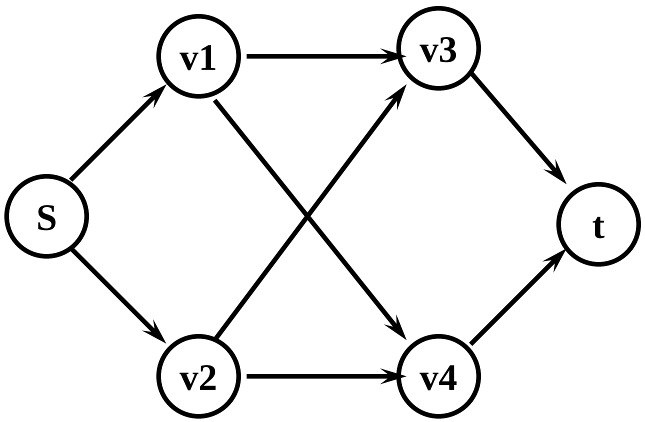
- information through communication network

- goods transported on the road…

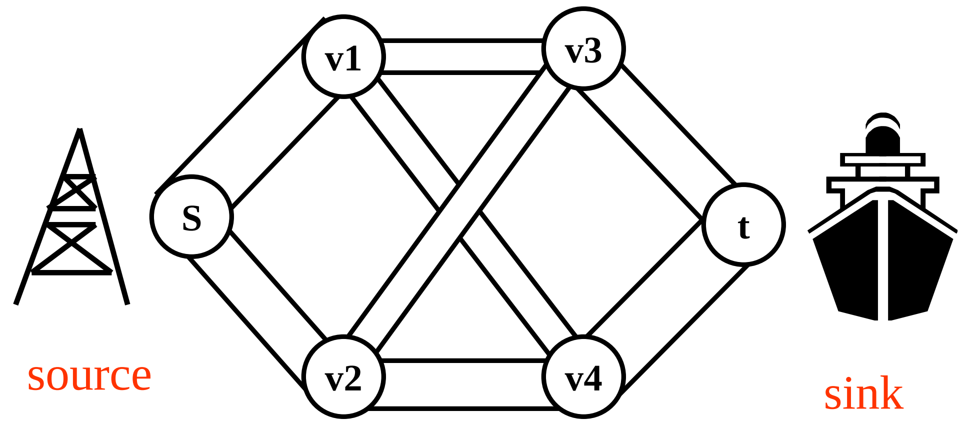
Introduction - network

Representation

Flow network: directed graph G=(V,E)



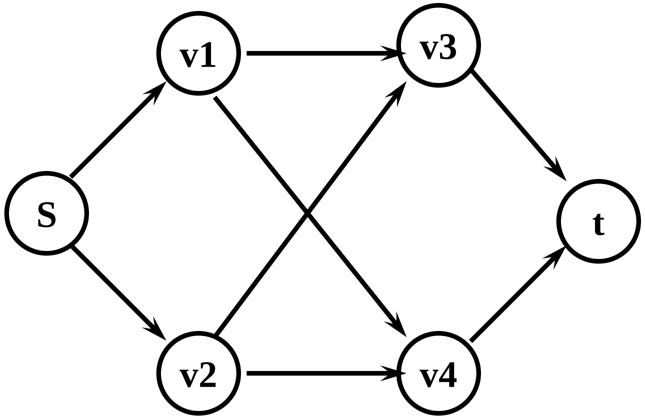
Example: oil pipeline



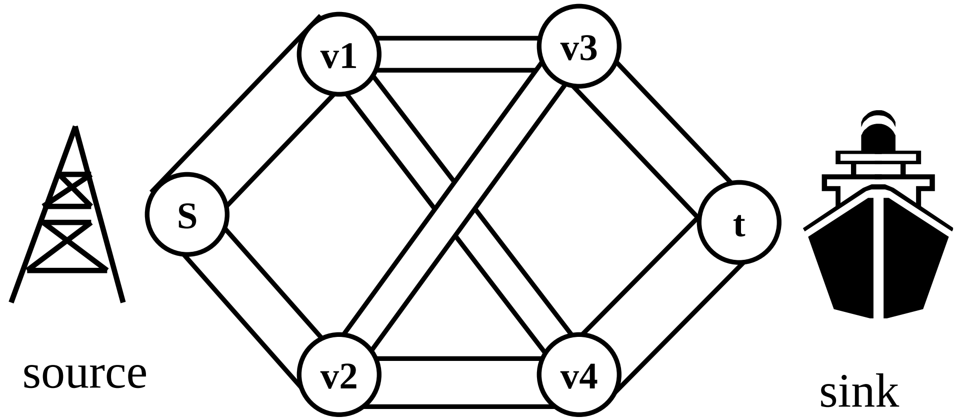
Introduction – max-flow problem

Representation

Flow network: directed graph G=(V,E)



Example: oil pipeline



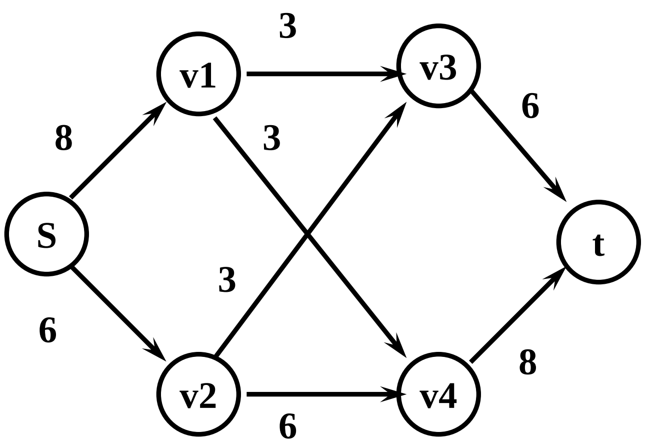
Informal definition of the max-flow problem:

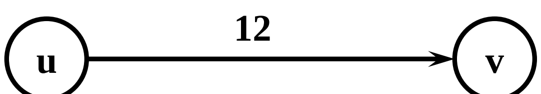
What is the greatest rate at which material can be shipped from the source to the sink without violating any capacity contraints?

Introduction - capacity

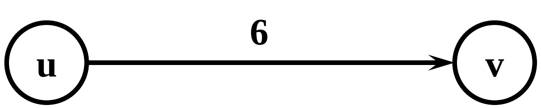
Representation

Flow network: directed graph G=(V,E)



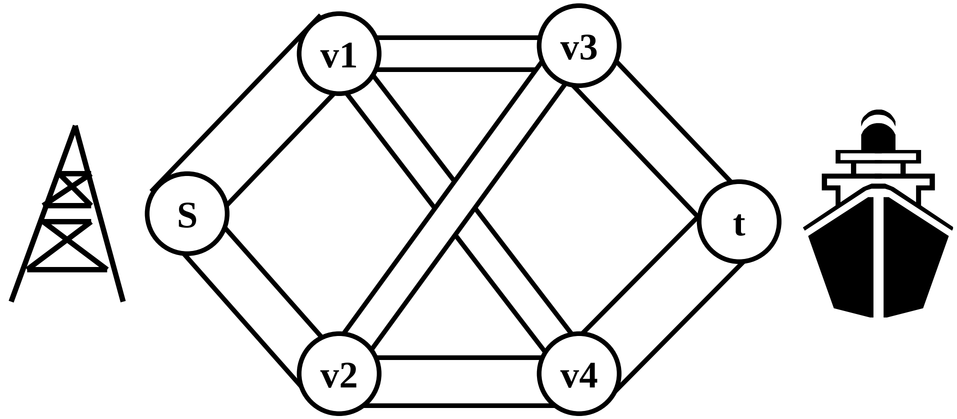


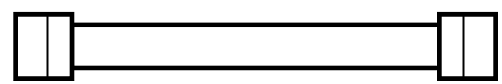
c(u,v)=12



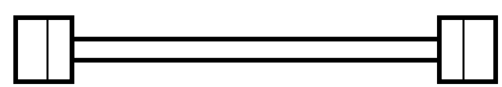
c(u,v)=6

Example: oil pipeline





Big pipe

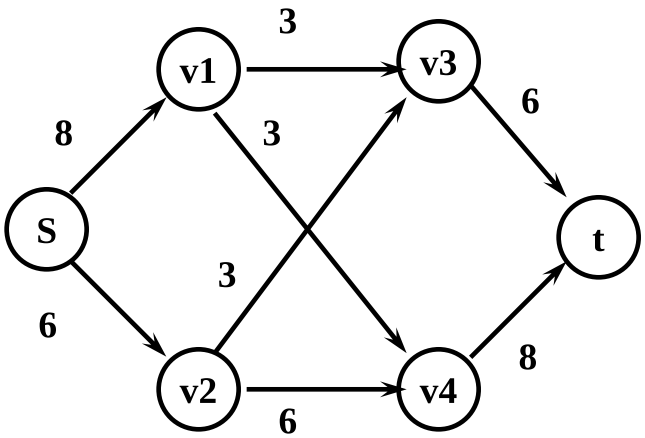


Small pipe

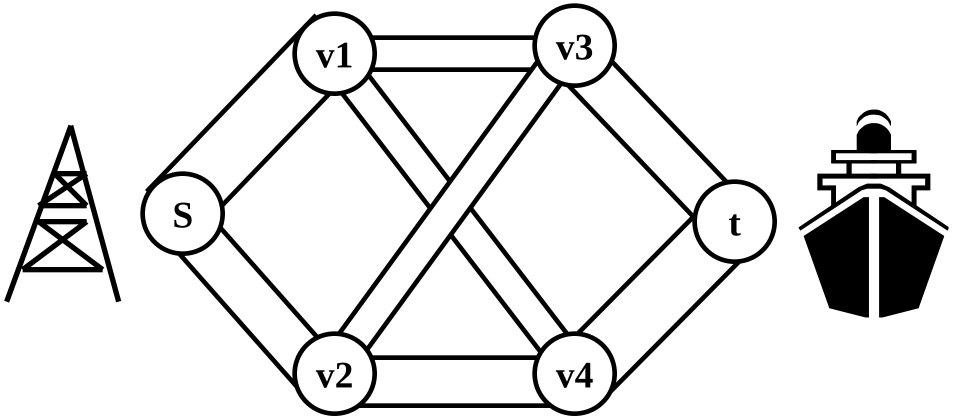
Introduction - capacity

Representation

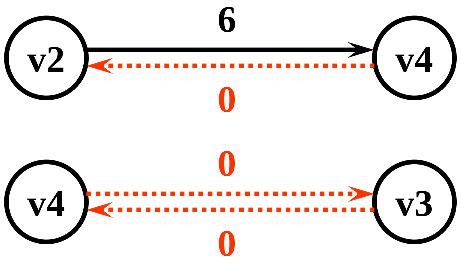
Flow network: directed graph G=(V,E)



Example: oil pipeline



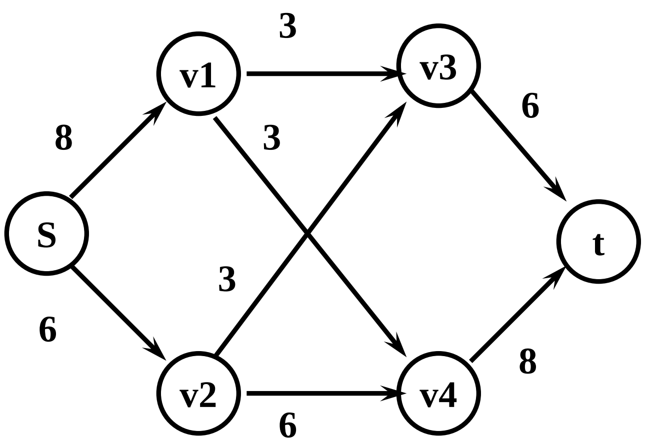
If $(u,v) \notin E \Rightarrow c(u,v) = 0$

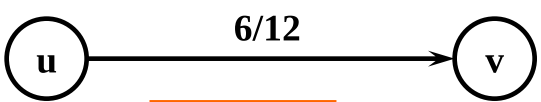


Introduction – flow

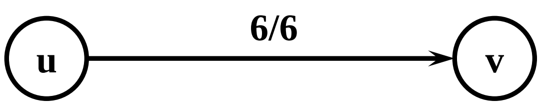
Representation

Flow network: directed graph G=(V,E)



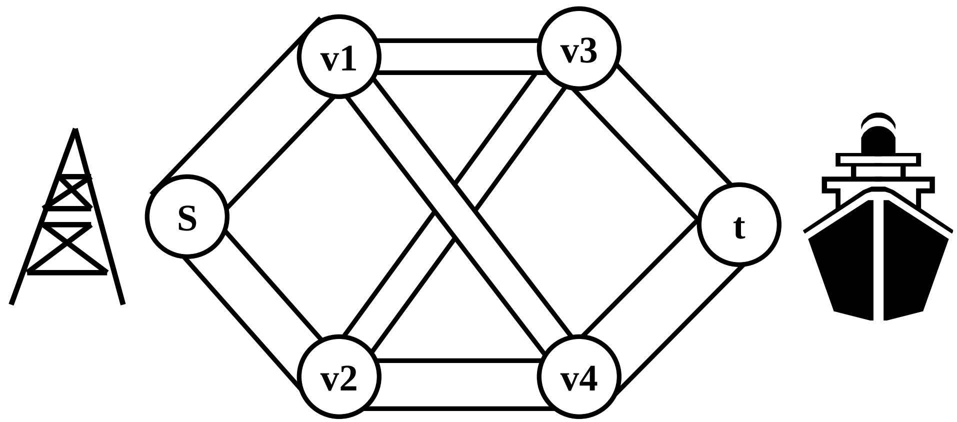


f(u,v)=6



f(u,v)=6

Example: oil pipeline



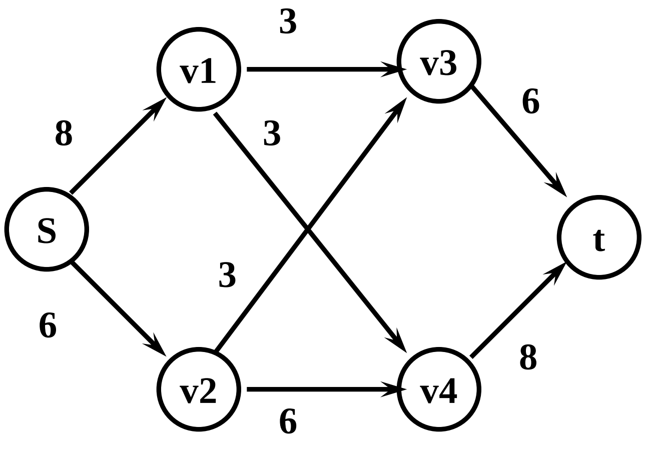
Flow below capacity

Maximum flow

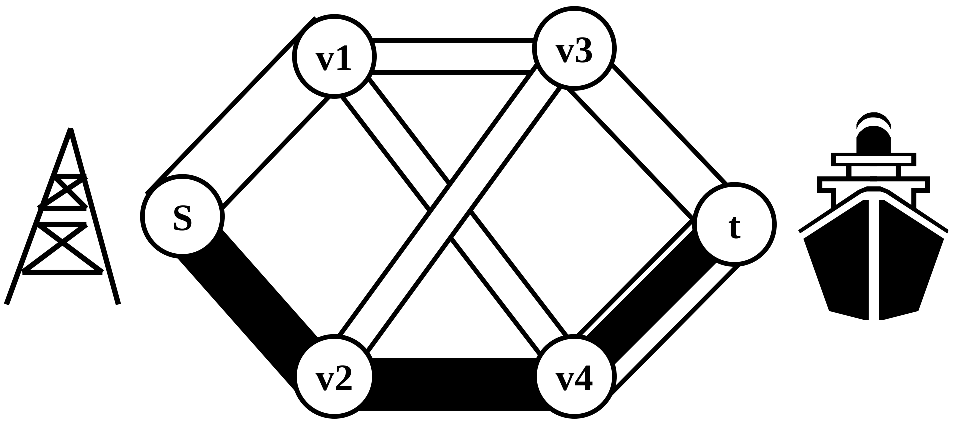
Introduction – flow

Representation

Flow network: directed graph G=(V,E)



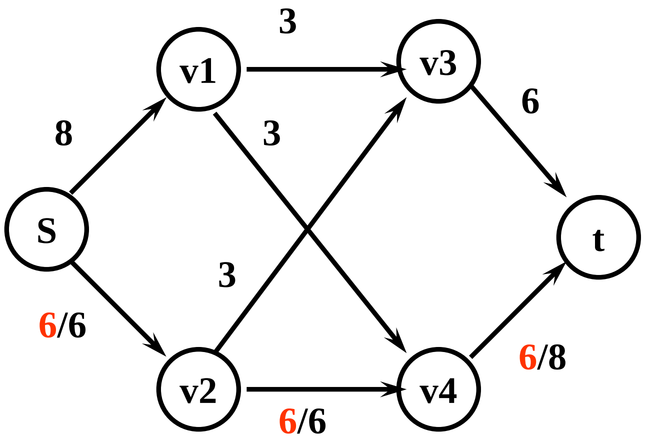
Example: oil pipeline



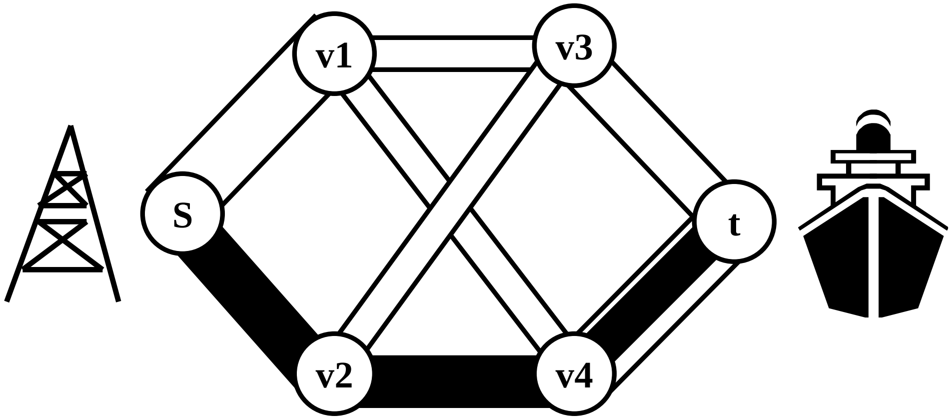
Introduction – flow

Representation

Flow network: directed graph G=(V,E)



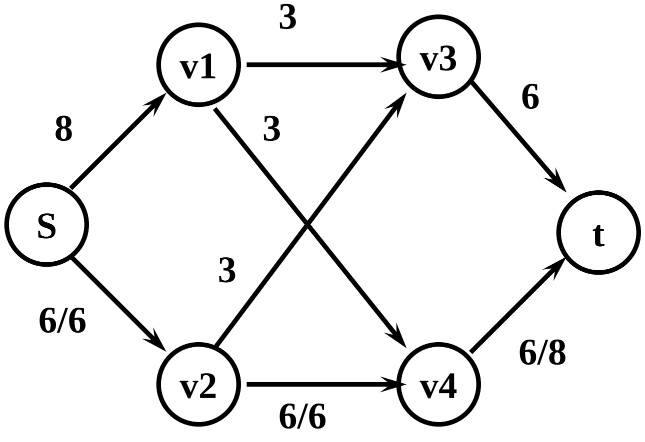
Example: oil pipeline



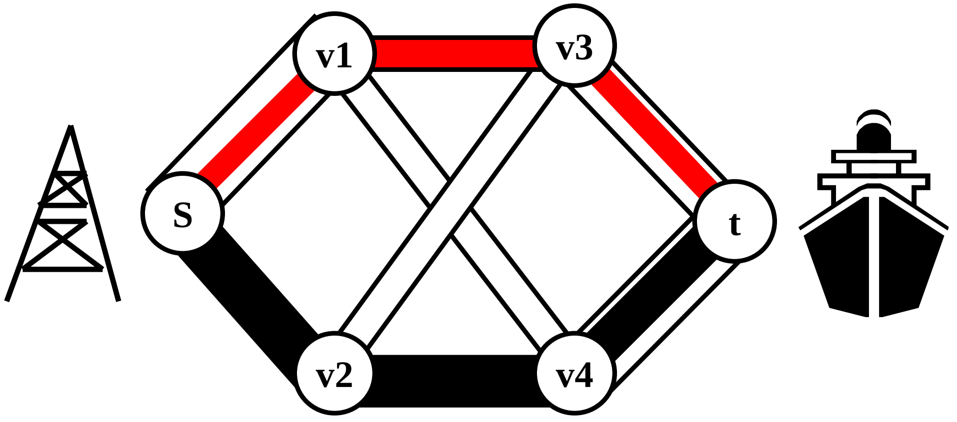
Introduction – flow

Representation

Flow network: directed graph G=(V,E)



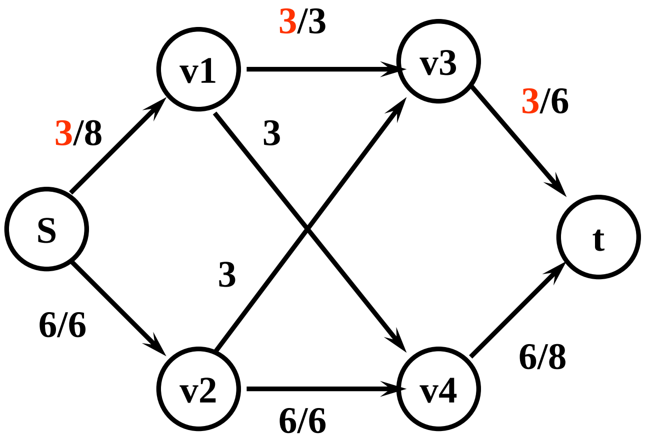
Example: oil pipeline



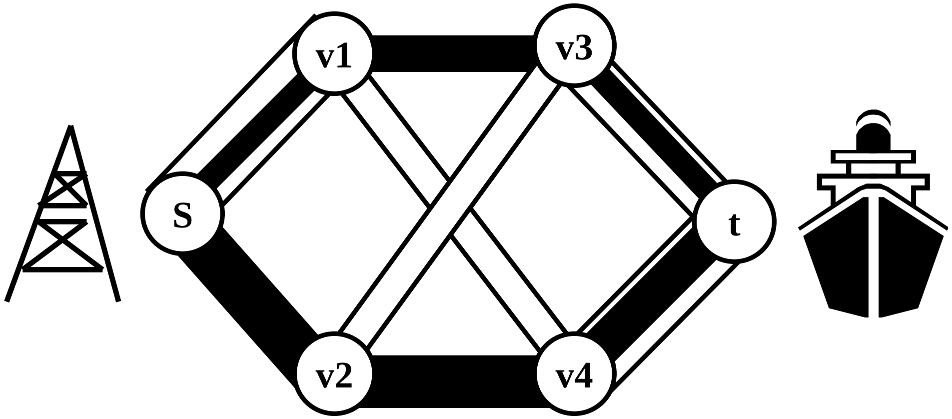
Introduction – flow

Representation

Flow network: directed graph G=(V,E)



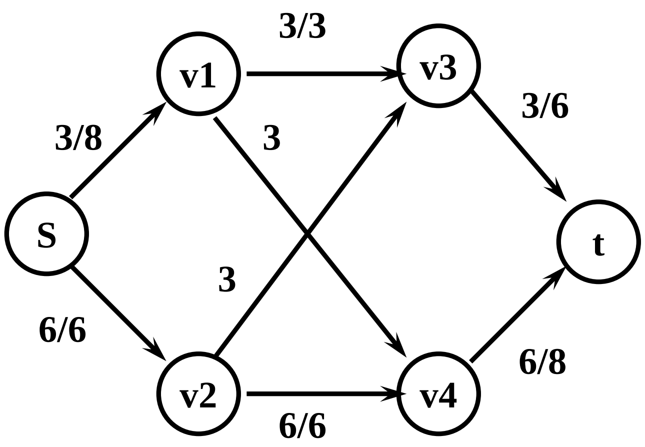
Example: oil pipeline



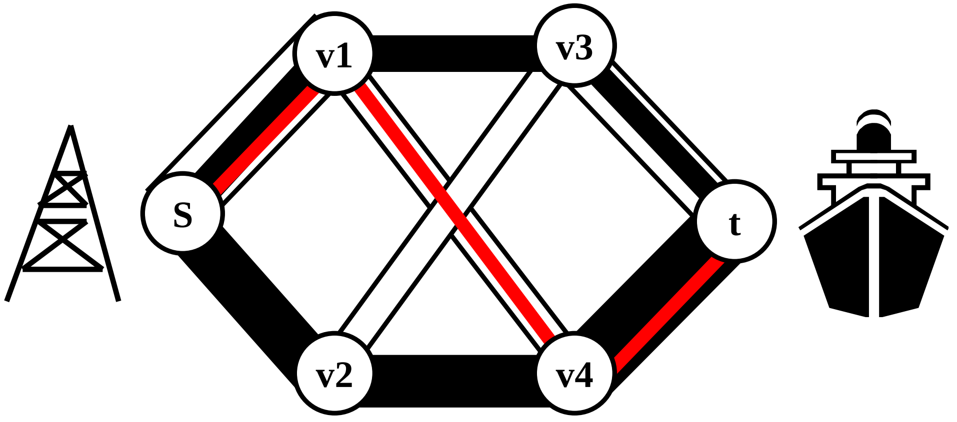
Introduction – flow

Representation

Flow network: directed graph G=(V,E)



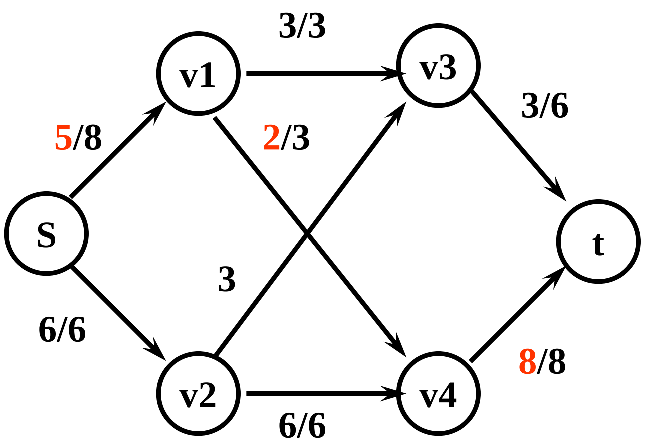
Example: oil pipeline



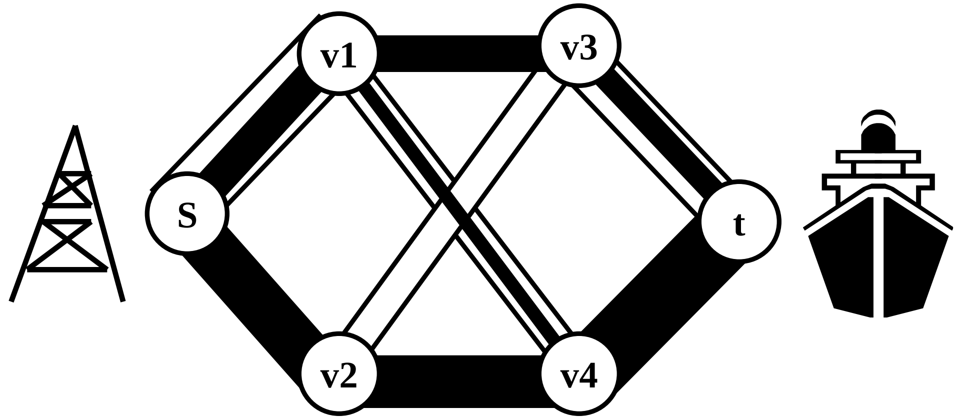
Introduction – flow

Representation

Flow network: directed graph G=(V,E)



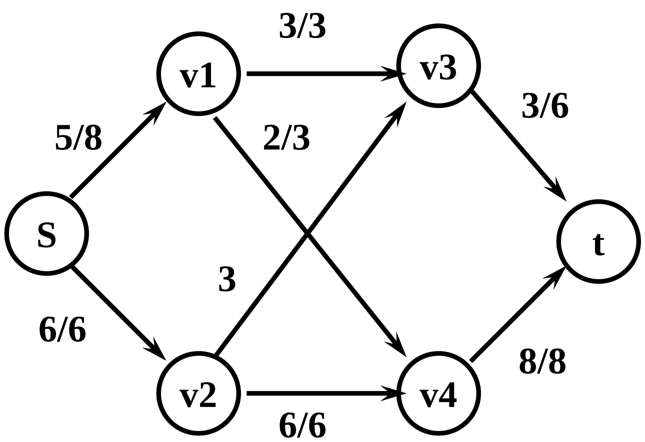
Example: oil pipeline



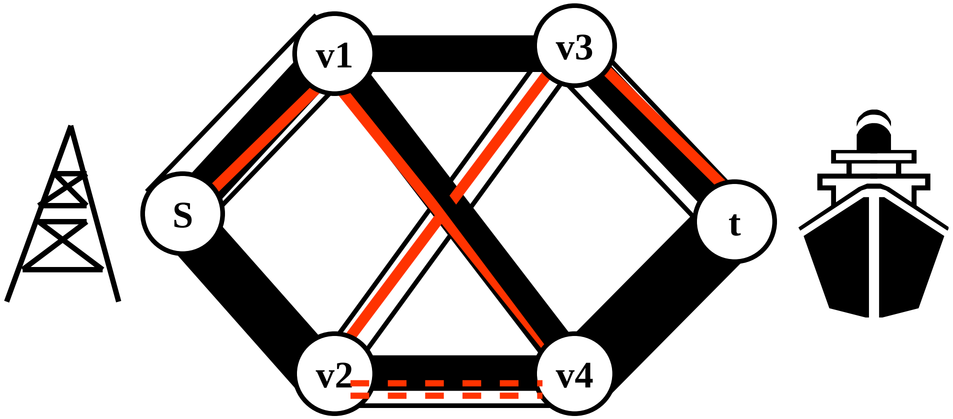
Introduction – cancellation

Representation

Flow network: directed graph G=(V,E)



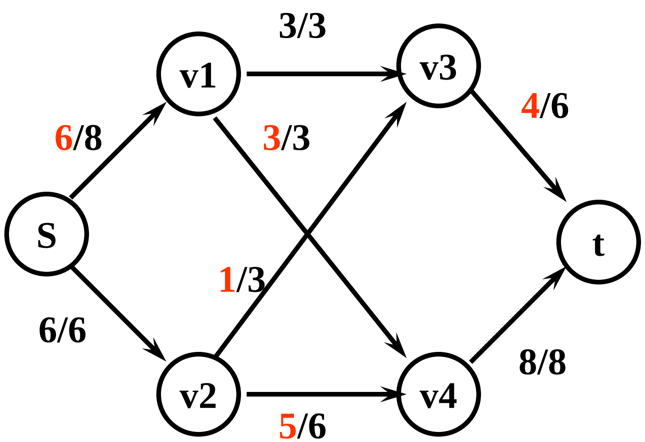
Example: oil pipeline



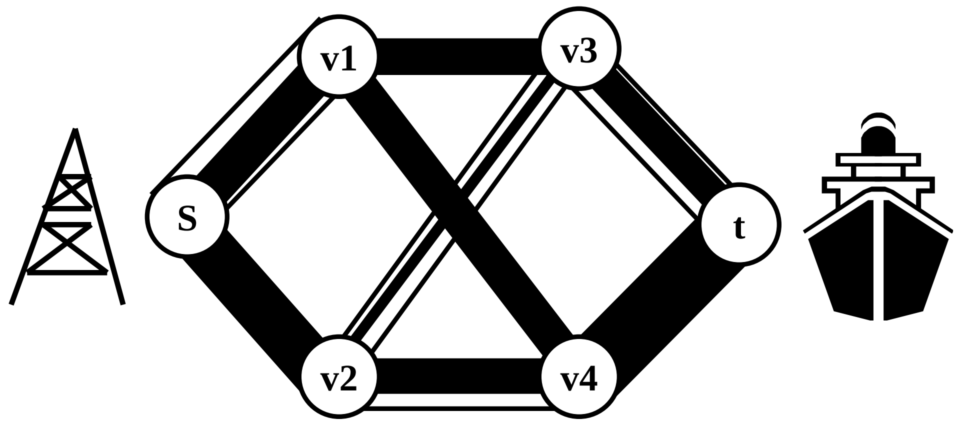
Introduction – cancellation

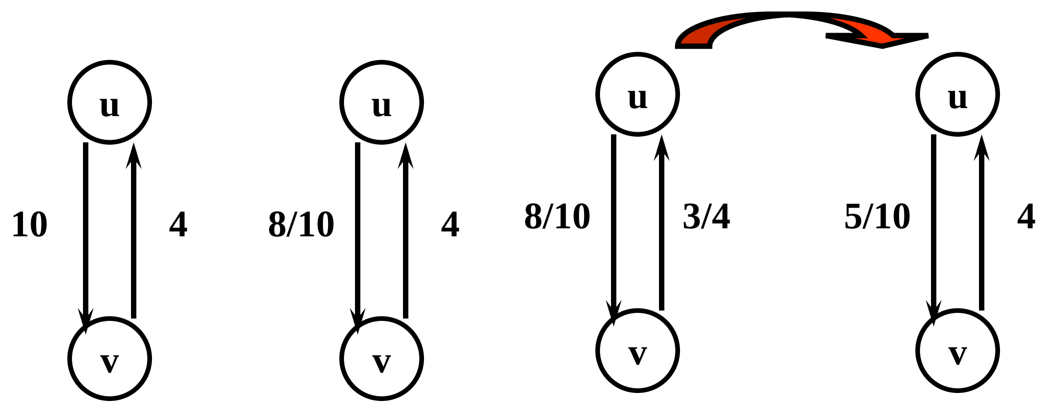
Representation

Flow network: directed graph G=(V,E)



Example: oil pipeline





Flow properties

Flow in $G = (V,E)$: $f: V \times V \rightarrow R$ with 3 properties:

1) Capacity constraint: For all $u,v \in V$: $f(u,v) \leq c(u,v)$

2) Skew symmetry: For all $u,v \in V$: $f(u,v) = - f(v,u)$

3) Flow conservation: For all $u \in V \setminus \{s,t\}$: $\sum_{v \in V} f(u,v) = 0$

Flow properties

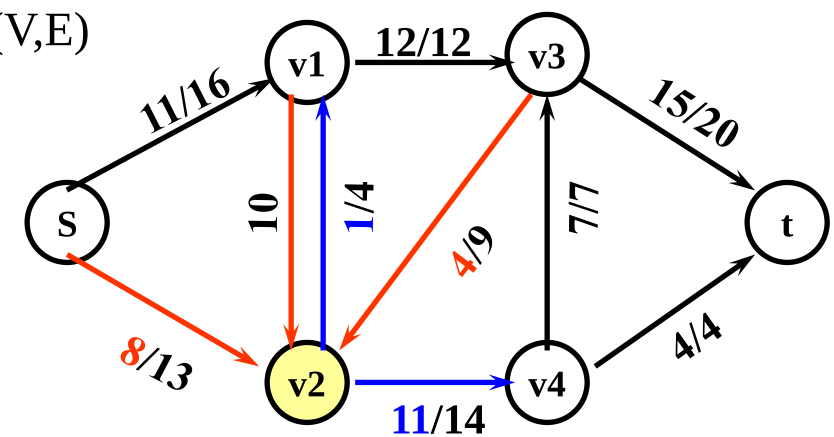
Flow in $G = (V,E)$: $f: V \times V \rightarrow R$ with 3 properties:

1) Capacity constraint: For all $u,v \in V$: $f(u,v) \leq c(u,v)$

2) Skew symmetry: For all $u,v \in V$: $f(u,v) = - f(v,u)$

3) Flow conservation: For all $u \in V \setminus \{s,t\}$: $\sum_{v \in V} f(u,v) = 0$

Flow network G = (V,E)



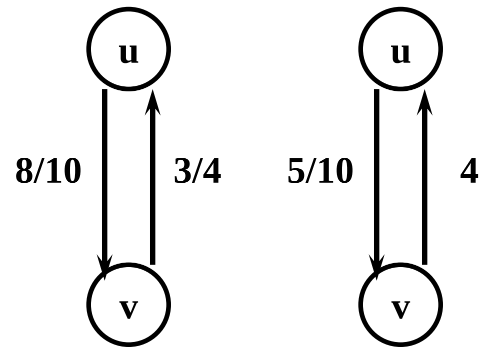
Note:

by skew symmetry

$f(v3,v1) = - 12$

Net flow and value of a flow

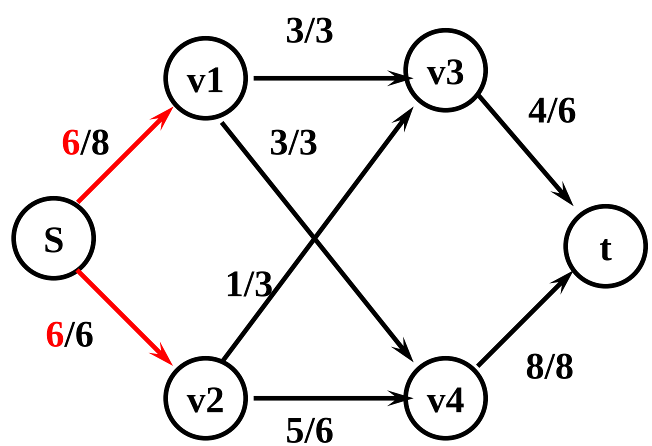
Net Flow:
positive or negative value of f(u,v)



$f(u,v) = 5$

$f(v,u) = -5$

Value of a Flow f:
Def: $|f| = \sum_{v \in V} f(s,v)$



The max-flow problem

Informal definition of the max-flow problem:

What is the greatest rate at which material can be shipped from the source to the sink without violating any capacity contraints?

Formal definition of the max-flow problem:

The max-flow problem is to find a valid flow for a given weighted directed graph G, that has the maximum value over all valid flows.

The Ford-Fulkerson method

a way how to find the max-flow

This method contains 3 important ideas:

1) residual networks
2) augmenting paths
3) cuts of flow networks

Ford-Fulkerson – pseudo code

initialize flow f to 0

while there exits an augmenting path p

    do augment flow f along p

return f

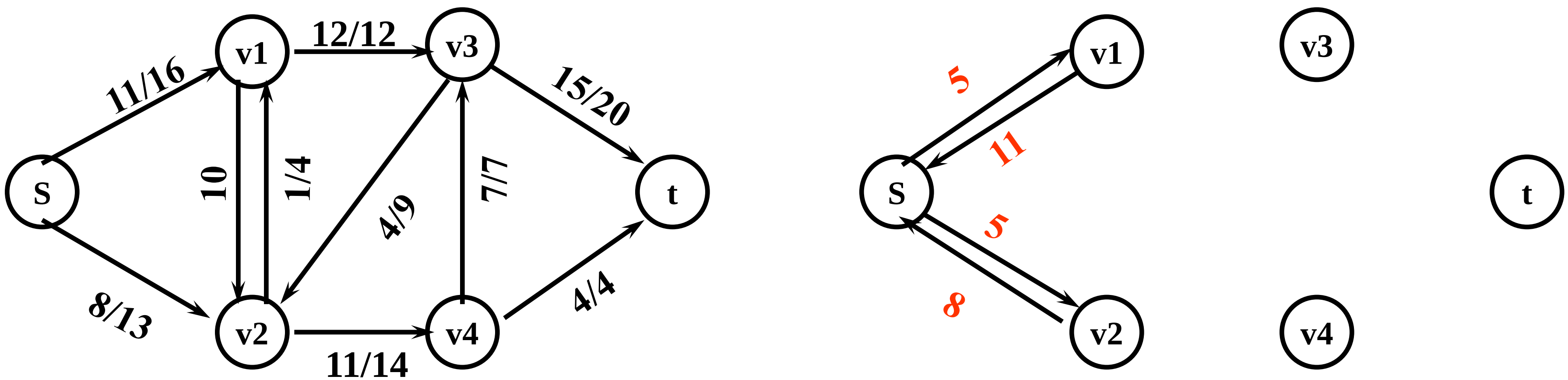
Ford Fulkerson – residual networks

The residual network G_f of a given flow network G with a valid flow f consists of the same vertices $v \in V$ as in G which are linked with residual edges $(u,v) \in E_f$ that can admit more strictly positive net flow.

The residual capacity c_f represents the weight of each edge E_f and is the amount of additional net flow f(u,v) before exceeding the capacity c(u,v)

Flow network $G = (V,E)$

residual network $G_f = (V,E_f)$



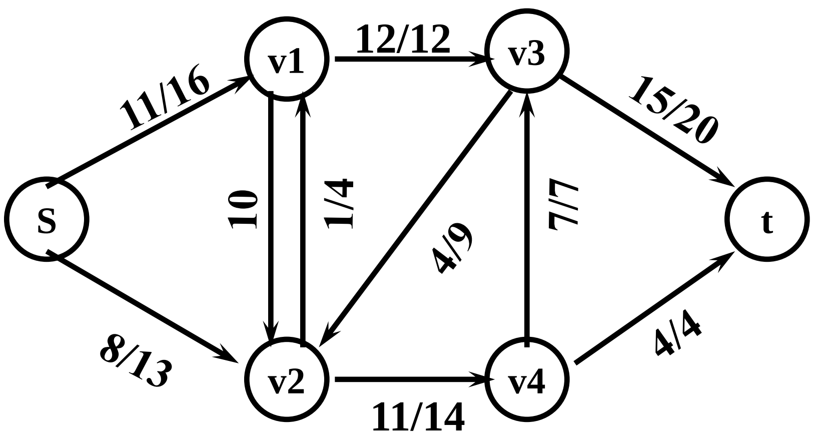
$$c_f(u,v) = c(u,v) - f(u,v)$$

Ford Fulkerson – residual networks

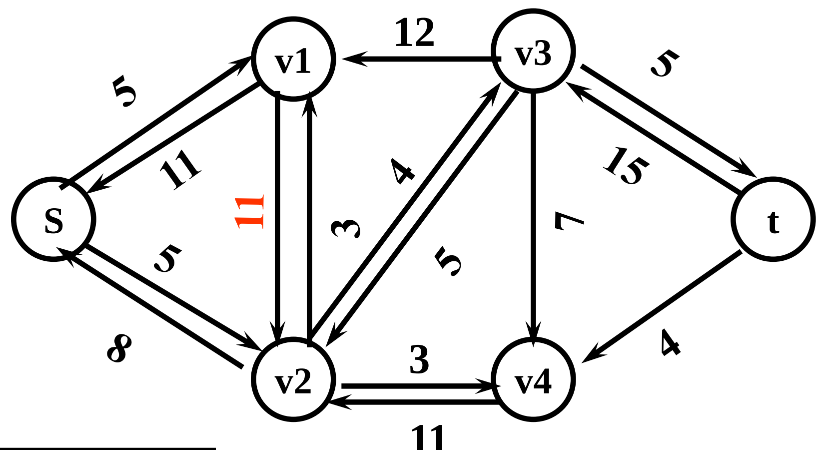
The residual network G_f of a given flow network G with a valid flow f consists of the same vertices $v \in V$ as in G which are linked with residual edges $(u,v) \in E_f$ that can admit more strictly positive net flow.

The residual capacity c_f represents the weight of each edge E_f and is the amount of additional net flow f(u,v) before exceeding the capacity c(u,v)

Flow network $G = (V,E)$



residual network $G_f = (V,E_f)$



$$c_f(u,v) = c(u,v) - f(u,v)$$

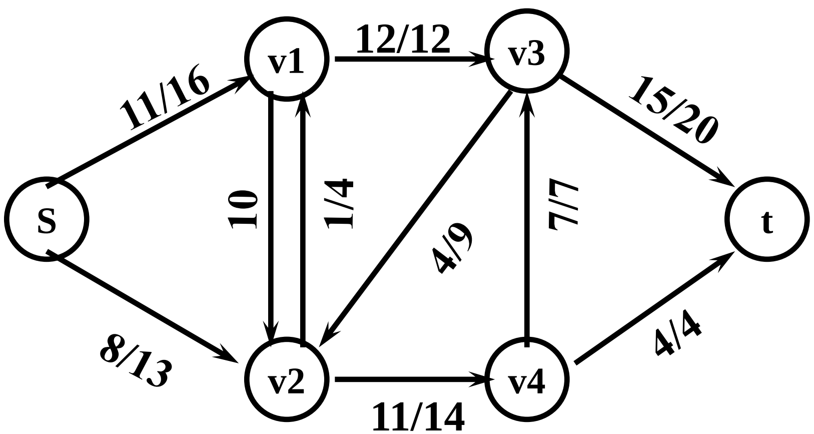
Ford Fulkerson – augmenting paths

Definition: An augmenting path p is a simple (free of any cycle) path from s to t in the residual network G_f

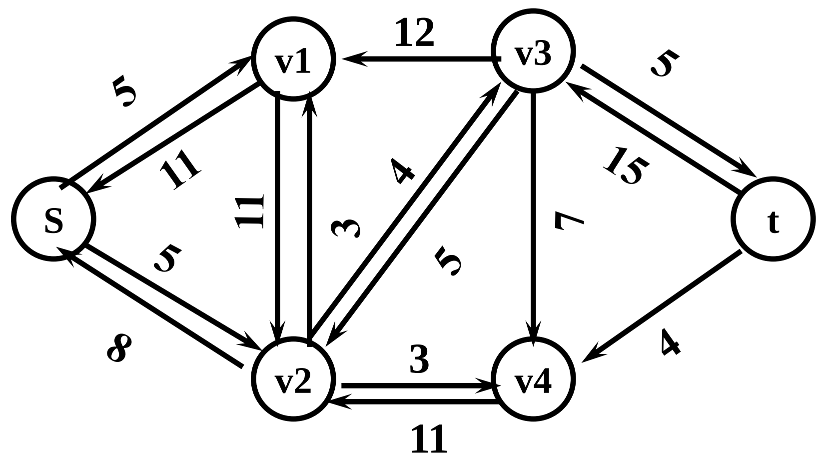
Residual capacity of p

$c_f(p) = \min\{c_f(u,v): (u,v) \text{ is on } p\}$

Flow network G = (V,E)



residual network $G_f = (V,E_f)$



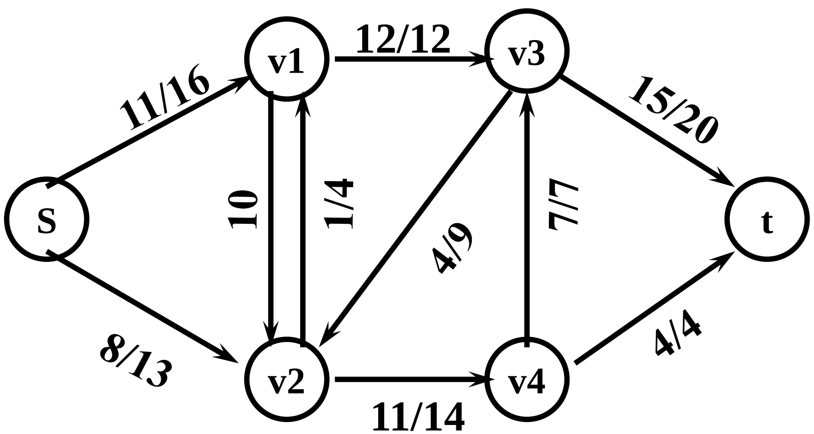
Ford Fulkerson – augmenting paths

Definition: An augmenting path p is a simple (free of any cycle) path from s to t in the residual network G_f

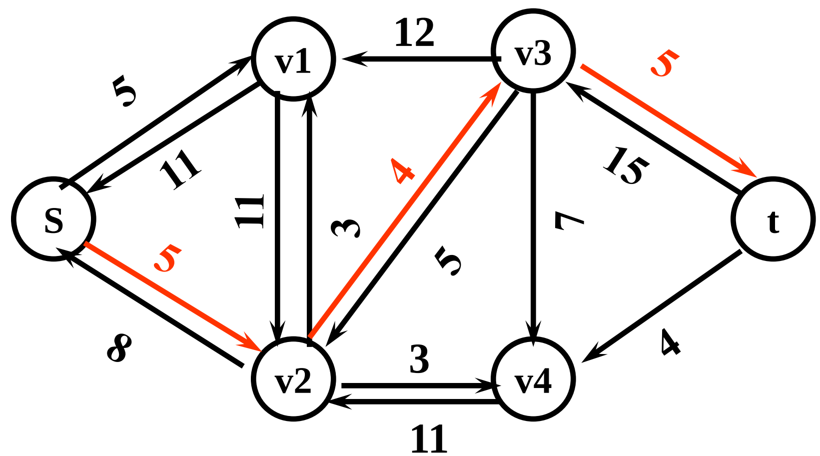
Residual capacity of p

$c_f(p) = \min\{c_f\ (u,v) : (u,v) \text{ is on } p\}$

Flow network G = (V,E)



residual network $G_f = (V, E_f)$



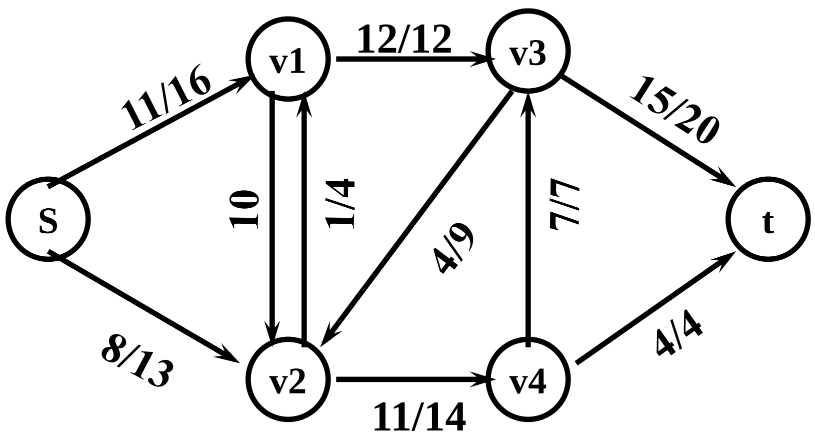
Augmenting path

Ford Fulkerson – augmenting paths

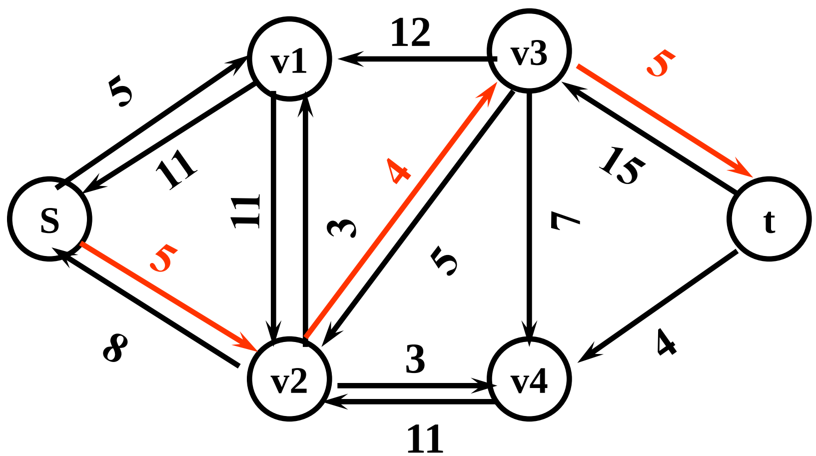
We define a flow: f_p: V x V $\rightarrow$ R such as:

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

Flow network G = (V,E)



residual network G_f = (V,E_f)

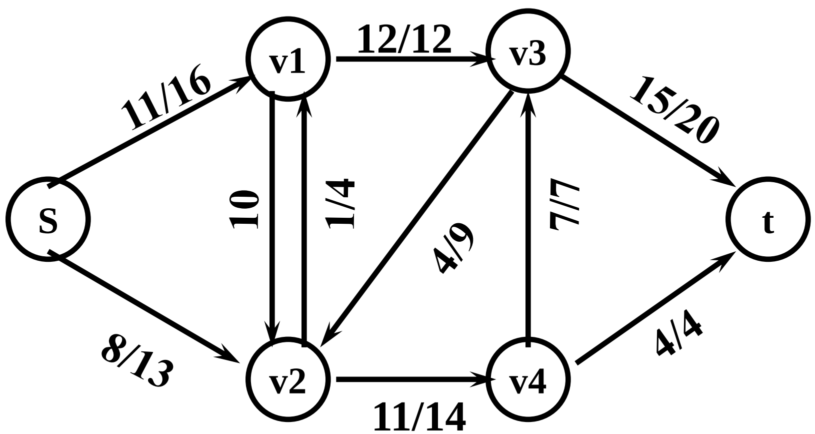


Ford Fulkerson – augmenting paths

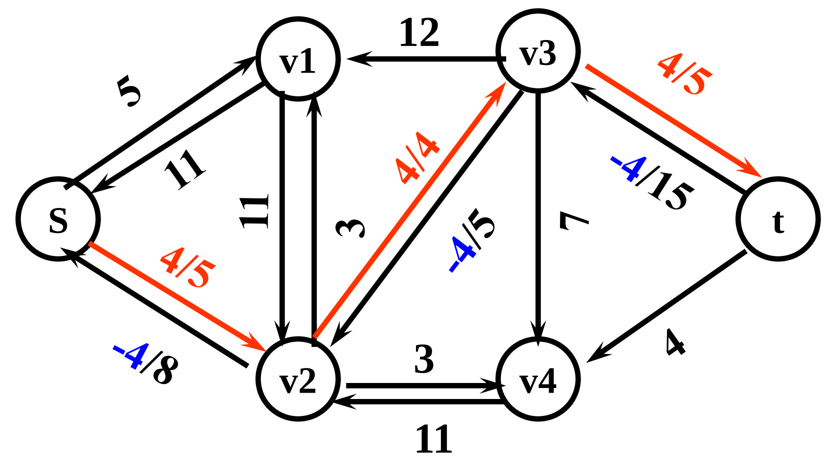
We define a flow: f_p: V x V $\rightarrow$ R such as:

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

Flow network G = (V,E)



residual network G_f = (V,E_f)



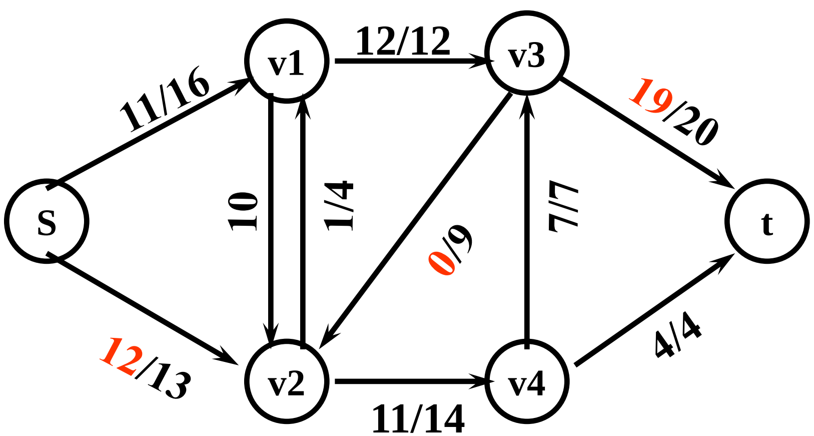
Our virtual flow f_p along the augmenting path p in G_f

Ford Fulkerson – augmenting the flow

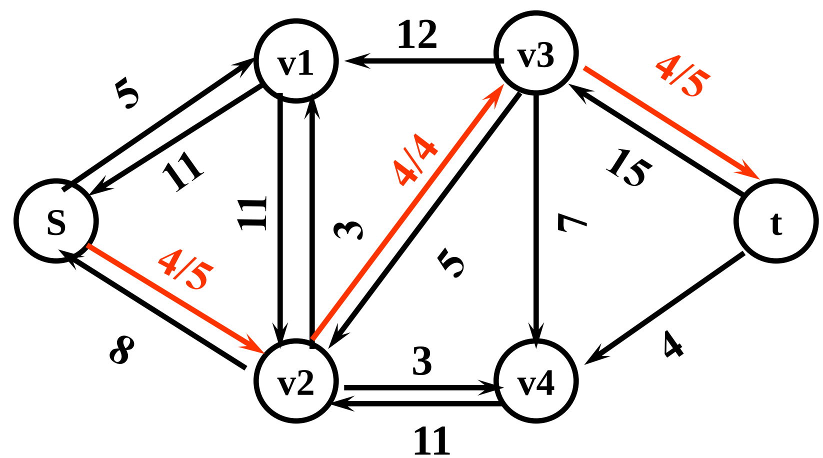
We define a flow: f_p: V x V $\rightarrow$ R such as:

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

Flow network G = (V,E)



residual network G_f = (V,E_f)



New flow: $f': V \times V \rightarrow R : \ f' = f + f_p$

Our virtual flow f_p along the augmenting path p in G_f

Ford Fulkerson – new valid flow

proof of capacity constraint

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

$c_f(p) = \min\{c_f\,(u,v): (u,v) \text{ is on } p\}$

$c_f(u,v) = c(u,v) – f(u,v)$

Lemma:

$f' : V \times V \rightarrow R: \quad f' = f + f_p \quad$ in G

Capacity constraint:

For all $u,v \in V$, we require $f(u,v) \leq c(u,v)$

Proof:

$$f_p\,(u\,,v) \leq c_f\,(u\,,v) = c\,(u\,,v) – f\,(u\,,v)$$

$$\Rightarrow (f + f_p)\,(u\,,v) = f\,(u\,,v) + f_p\,(u\,,v) \leq c\,(u\,,v)$$

Ford Fulkerson – new valid flow

proof of Skew symmetry

Lemma:

$$\boxed{f' : V \times V \rightarrow R : \quad f' = f + f_p \qquad \text{in } G}$$

Skew symmetry:

For all $u,v \in V$, we require $f(u,v) = - f(v,u)$

Proof:

$$(f + f_p)(u,v) = f(u,v) + f_p(u,v) = - f(v,u) - f_p(v,u)$$

$$= - (f(v,u) + f_p(v,u)) = - (f + f_p)(v,u)$$

Ford Fulkerson – new valid flow

proof of flow conservation

Lemma:

$$\boxed{f' : V \times V \rightarrow R : \quad f' = f + f_p \qquad \text{in } G}$$

Flow conservation:

For all $u \in V \setminus \{s,t\}$: $\sum_{v \in V} f(u,v) = 0$

Proof:

$$u \in V - \{s,t\} \Rightarrow \sum_{v \in V}(f + f_p)(u,v) = \sum_{v \in V}(f(u,v) + f_p(u,v))$$

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

Ford Fulkerson – new valid flow

Lemma:

$$\boxed{|(f + f_p)| = |f| + |f_p|}$$

Value of a Flow f:

Def:

$$\boxed{|f| = \sum_{v \in V} f(s,v)}$$

Proof:

$$|(f + f_p)| = \sum_{v \in V}(f + f_p)(s,v) = \sum_{v \in V}(f(s,v) + f_p(s,v))$$

$$= \sum_{v \in V} f(s,v) + \sum_{v \in V} f_p(s,v) = |f| + |f_p|$$

Ford Fulkerson – new valid flow

Lemma:

$f' : V \times V \rightarrow R : \quad f' = f + f_p \quad \text{in } G$

$|(f + f_p)| = |f| + |f_p| > |f|$

Lemma shows:

if an augmenting path can be found then the above flow augmentation will result in a flow improvement.

Question: If we cannot find any more an augmenting path is our flow then maximum?

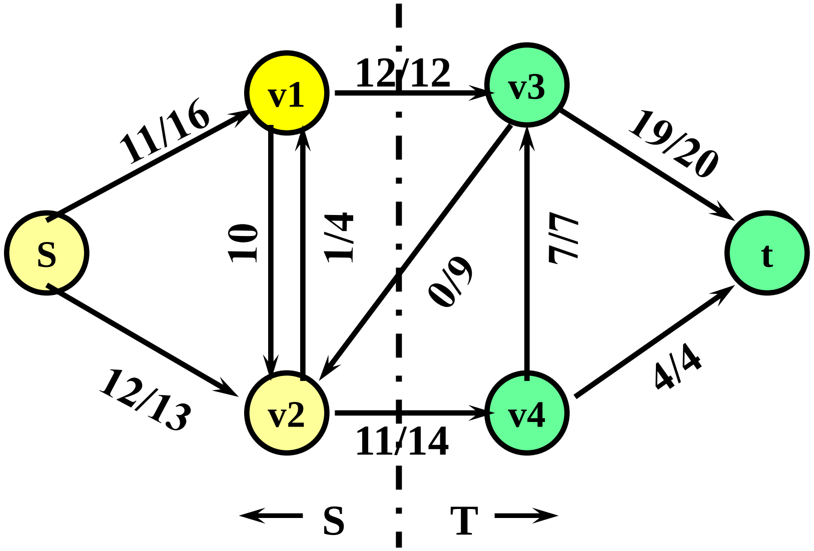
Idea: The flow in G is maximum $\Leftrightarrow$ the residual G_f contains no augmenting path.

Ford Fulkerson – cuts of flow networks

New notion: cut (S,T) of a flow network

A cut (S,T) of a flow network G=(V,E) is a partiton of V into S and T = V \ S such that s ∈ S and t ∈ T.

Practical example



In the example:

S = {s,v1,v2) , T = {v3,v4,t}

Net flow f(S ,T) = f(v1,v3) + f(v2,v4) + f(v2,v3)

= 12 + 11 + (-0) = 23

Capacity c(S,T) = c(v1,v3) + c(v2,v4)

= 12 + 14 = 26

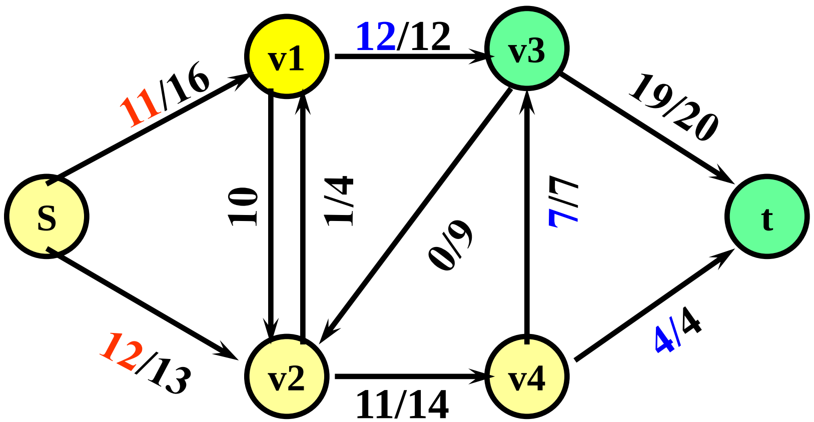
Implicit summation notation: $f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v)$

Ford Fulkerson – cuts of flow networks

Lemma:

the value of a flow in a network is the net flow across any cut of the network

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



Ford Fulkerson – cuts of flow networks

the value of a flow in a network is the net flow across any cut of the network

Lemma: $\boxed{f\ (S\ ,T) = |\ f\ |}$

Proof:

$f\ (S, T) = f\ (S, V \backslash S)$

$= f\ (S, V) - f\ (S, S)$

$= f\ (S, V) = f\ (s \cup [S \backslash s], V)$

$= f\ (s, V) + f\ (S \backslash s, V)$

$= f\ (s, V) = |\ f\ |$

Working with flows:

$X,Y,Z \subseteq V$ and $X \cap Y = \emptyset$

$f\ (X, Y) = \sum_{x \in X} \sum_{y \in Y} f\ (x, y)$

$f\ (X, X) = 0$

$f\ (X, Y) = -\ f\ (Y, X)$

$f\ (u, V) = 0$ for all $u \in V \setminus \{s, t\}$

$f\ (X \cup Y, Z) = f\ (X, Z) + f\ (Y, Z)$

$f\ (Z, X \cup Y) = f\ (Z, X) + f\ (Z, Y)$

Ford Fulkerson – cuts of flow networks

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

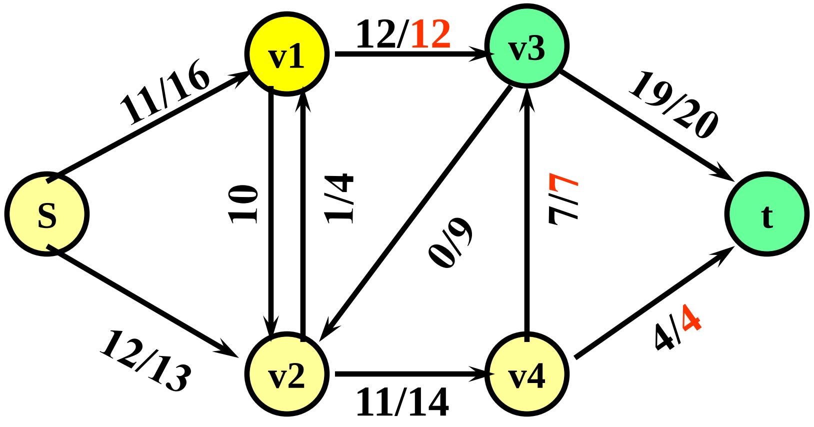
Lemma: $\boxed{|f| \leq c(S,T)}$

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

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

$$\leq \sum_{u \in S} \sum_{v \in T} c(u,v)$$

$$= c(S,T)$$



F. Fulkerson: Max-flow min-cut theorem

If f is a flow in a flow 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):

We assume for the sake of contradiction that f is a maximum flow in G but that there still exists an augmenting path p in G_f.

Then as we know from above, we can augment the flow in G according to the formula: $f' = f + f_p$. That would create a flow f' that is strictly greater than the former flow f which is in contradiction to our assumption that f is a maximum flow.

F. Fulkerson: Max-flow min-cut theorem

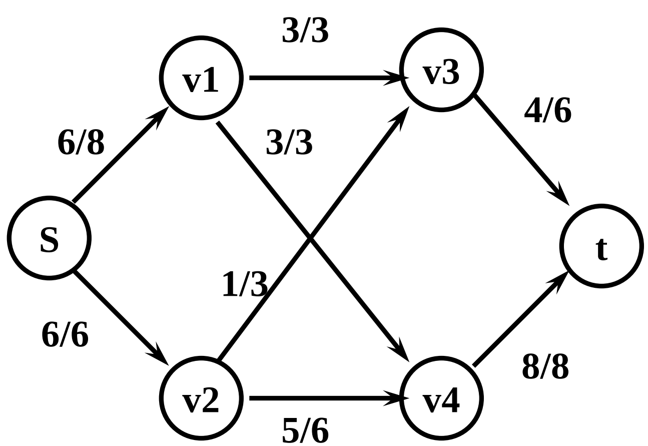
If f is a flow in a flow network G = (V,E) with source s and sink t, then the following conditions are equivalent:

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.

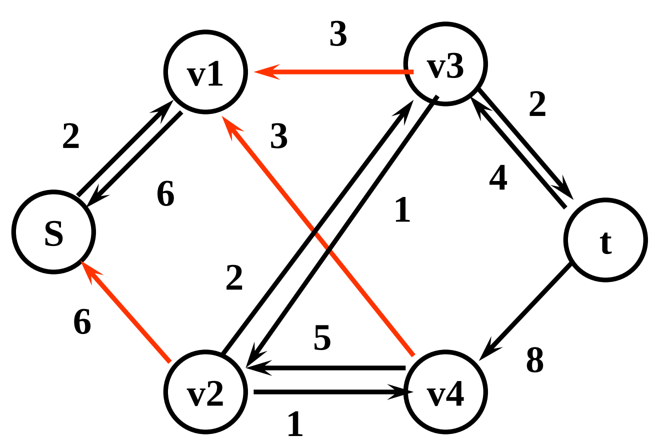
proof:

(2) $\Rightarrow$ (3):

original flow network G



residual network G_f



F. Fulkerson: Max-flow min-cut theorem

If f is a flow in a flow 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.

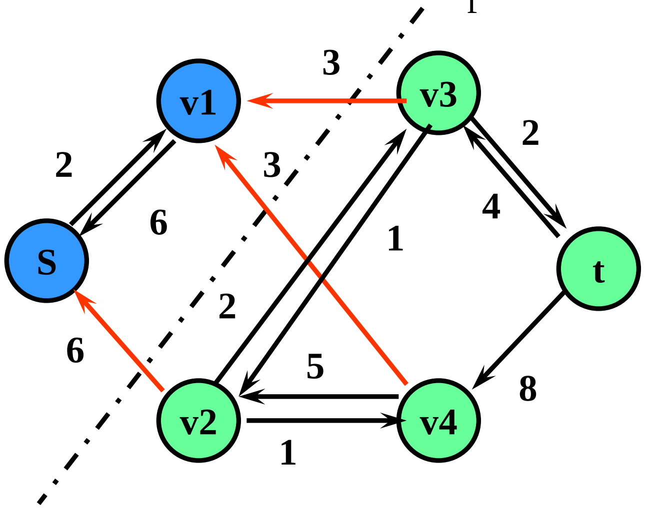
proof:

(2) $\Rightarrow$ (3): Define

$S = \{v \in V \mid \exists \text{ path } p \text{ from } s \text{ to } v \text{ in } G_f\}$

$T = V \setminus S$ (note $t \notin S$ according to (2))

residual network G_f



F. Fulkerson: Max-flow min-cut theorem

If f is a flow in a flow 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:

(2) $\Rightarrow$ (3): Define

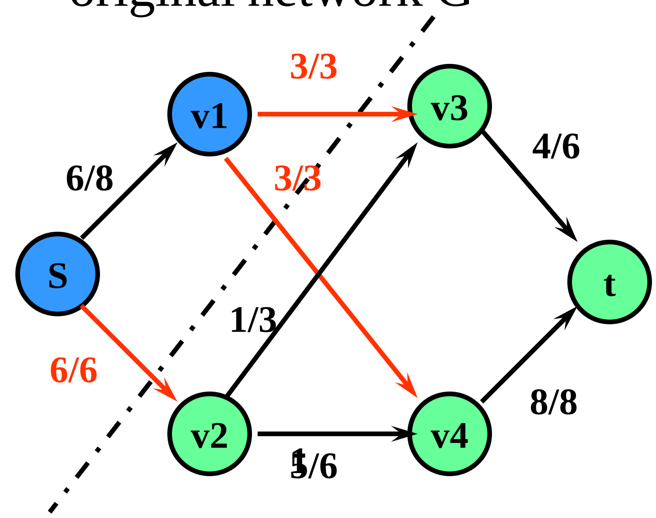
$S = \{v \in V \mid \exists \text{ path } p \text{ from } s \text{ to } v \text{ in } G_f\}$

$T = V \setminus S$ (note $t \notin S$ according to (2))

$\Rightarrow$ for $\forall\, u \in S, v \in T$: $f(u, v) = c(u, v)$ (otherwise $(u, v) \in E_f$ and $v \in S$)

$\Rightarrow$ $|f| = f(S, T) = c(S, T)$

original network G



F. Fulkerson: Max-flow min-cut theorem

If f is a fow in a flow 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:

(3) $\Rightarrow$ (1): as proofed before $| f | = f (S, T) \leq c (S, T)$

the statement of (3) : $| f | = c (S, T)$ implies that f is a maximum flow

Ford-Fulkerson – pseudo code

initialize flow f to 0
while there exits an augmenting path p
    do augment flow f along p
return f

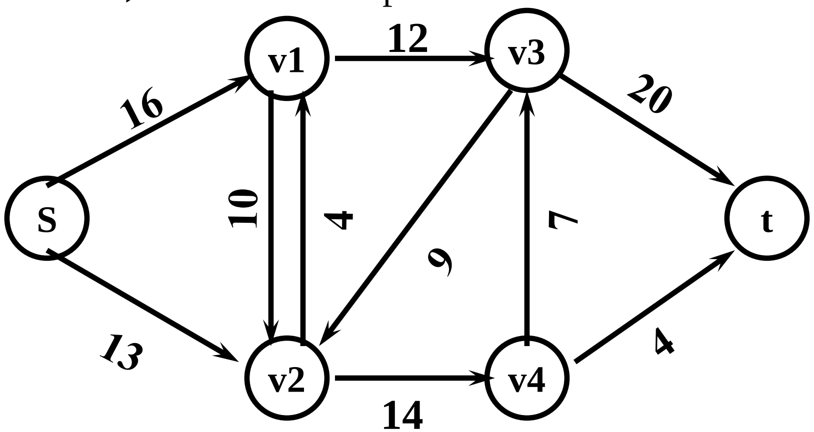
The basic Ford Fulkerson algorithm

for *each edge* $(u, v) \in E[G]$
    do $f[u, v] = 0$
        $f[v, u] = 0$
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \text{ is in } p\}$
        for *each edge (u, v) in p*
            do $f[u, v] = f[u, v] + c_f(p)$
                $f[v, u] = -f[u, v]$

The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f

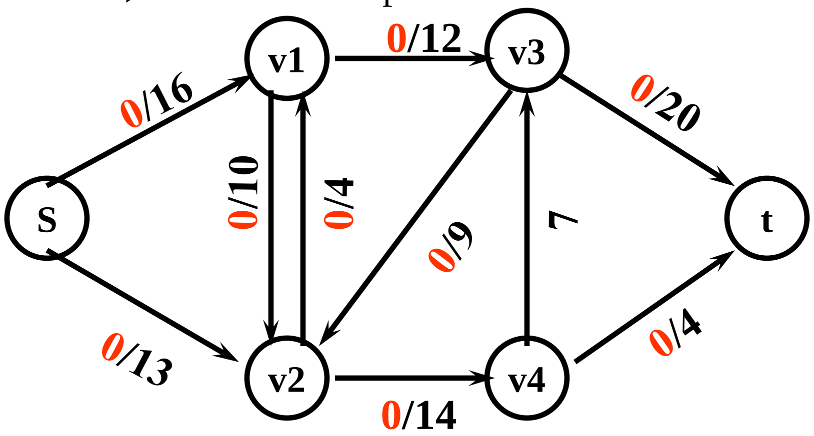



```
for each edge (u, v) ∈ E [G]
    do f [u, v] = 0
       f [v, u] = 0
while there exists a path p from s to t
in the residual network G_f
    do c_f(p) = min{c_f(u, v) | (u, v) ∈ p}
       for each edge (u, v) in p
           do f [u, v] = f [u, v] + c_f(p)
              f [v, u] = - f [u, v]
```

The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f

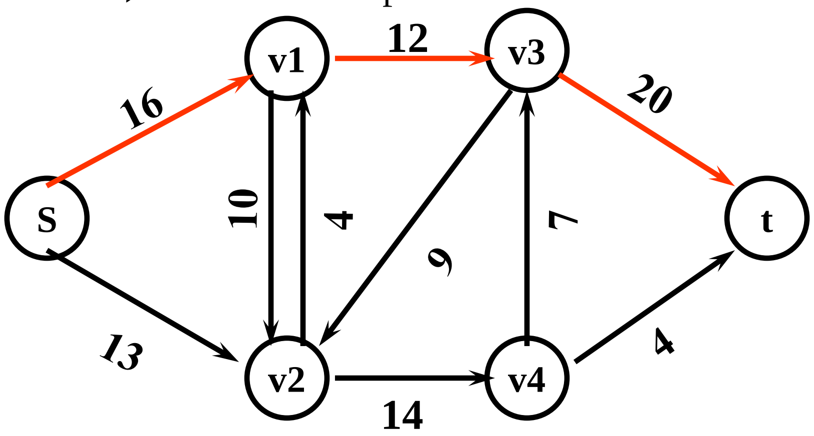


for *each edge* $(u, v) \in E[G]$
  do $f[u, v] = 0$
    $f[v, u] = 0$
while *there exists a path p from s to t in the residual network* G_f
  do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
    for *each edge* (u, v) *in* p
      do $f[u, v] = f[u, v] + c_f(p)$
        $f[v, u] = - f[u, v]$

The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f

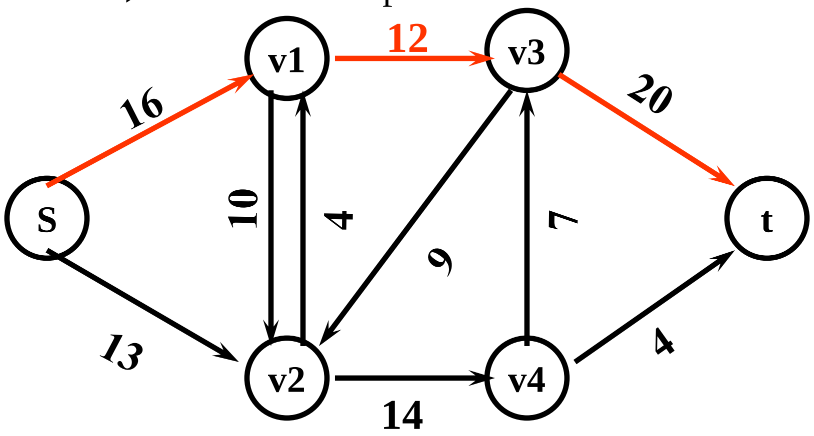


for *each edge* $(u, v) \in E[G]$
  do f [u, v] = 0
    f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
  do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
    for *each edge (u, v) in p*
      do f [u, v] = f [u, v] + $c_f(p)$
        f [v, u] = - f [u, v]

The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



for *each edge* $(u, v) \in E[G]$
    do $f[u, v] = 0$
        $f[v, u] = 0$
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
        for *each edge* (u, v) *in p*
            do $f[u, v] = f[u, v] + c_f(p)$
                $f[v, u] = - f[u, v]$

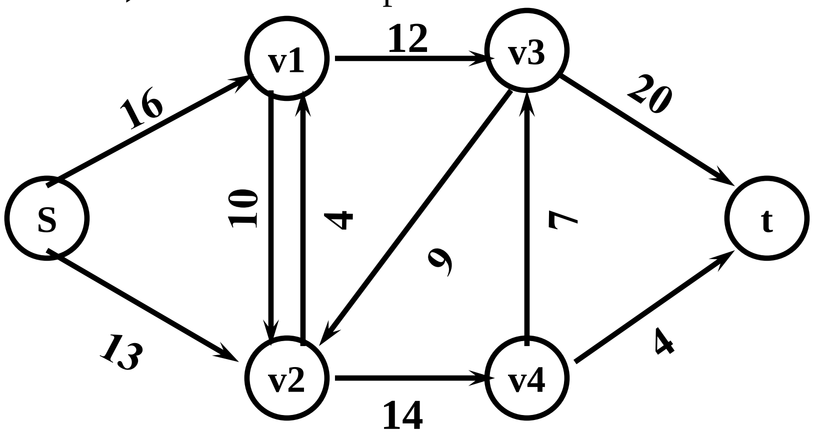
temporary variable:

$c_f(p) = 12$

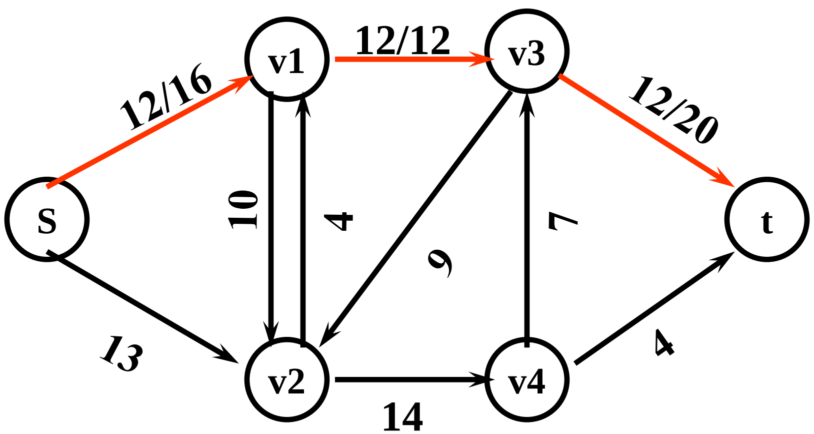
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G



for *each edge* $(u, v) \in E\ [G]$
  do f [u, v] = 0
    f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
  do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
    for *each edge (u, v) in p*
      do $f[u, v] = f[u, v] + c_f(p)$
        $f[v, u] = - f[u, v]$

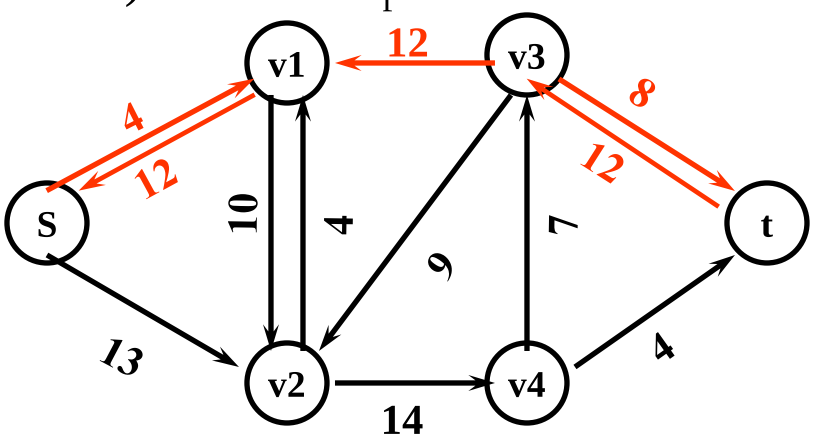
temporary variable:

$c_f(p) = 12$

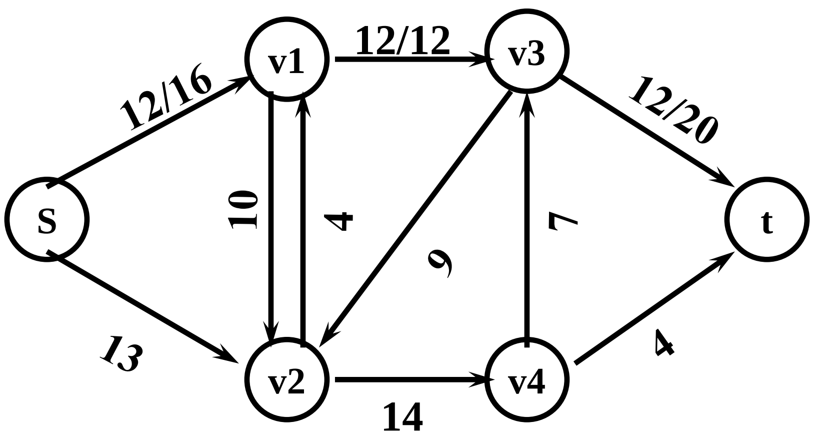
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G

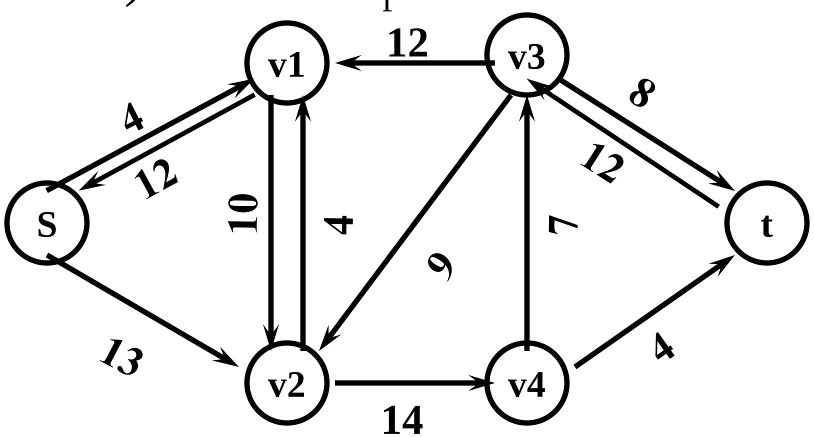


for *each edge (u, v)* $\in$ *E [G]*
    do f [u, v] = 0
        f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
        for *each edge (u, v) in p*
            do $f[u, v] = f[u, v] + c_f(p)$
                $f[v, u] = - f[u, v]$

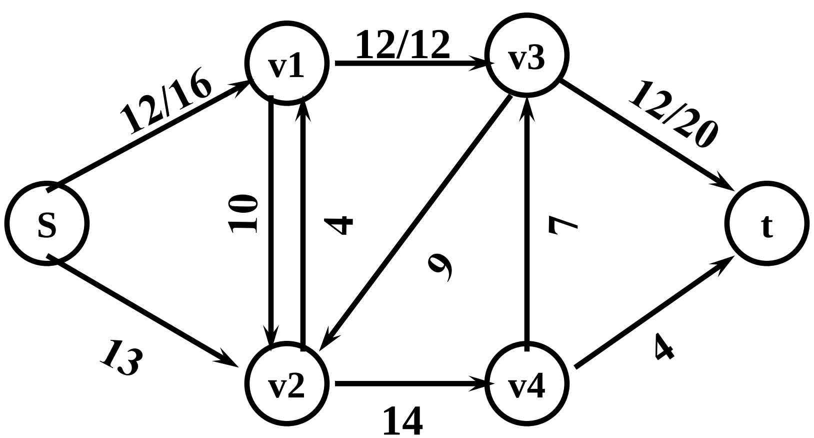
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G

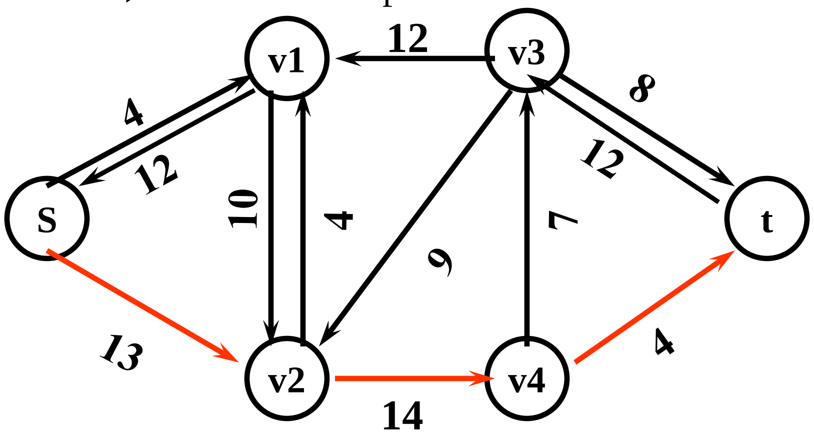


for *each edge (u, v)* $\in$ *E [G]*
  do f [u, v] = 0
    f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
  do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
    for *each edge (u, v) in p*
      do $f[u, v] = f[u, v] + c_f(p)$
        $f[v, u] = - f[u, v]$

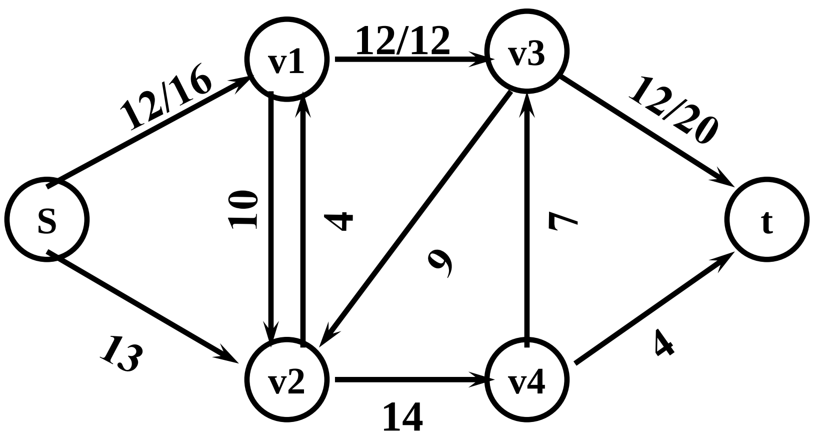
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G

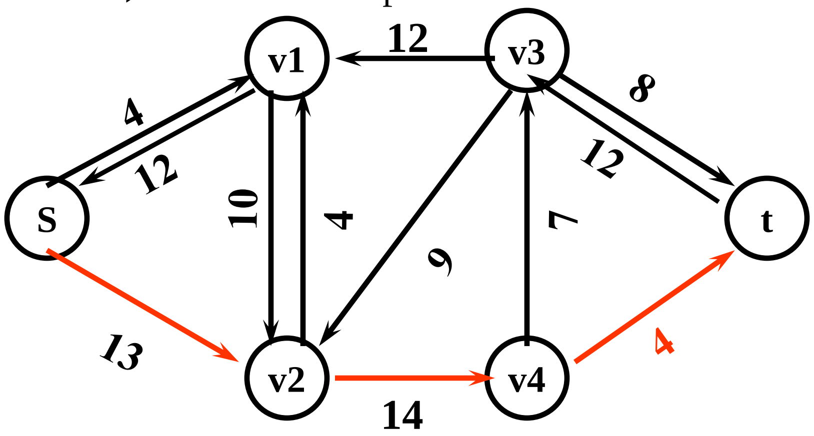


for *each edge (u, v)* $\in$ *E [G]*
    do f [u, v] = 0
        f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
        for *each edge (u, v) in p*
            do $f[u, v] = f[u, v] + c_f(p)$
                $f[v, u] = - f[u, v]$

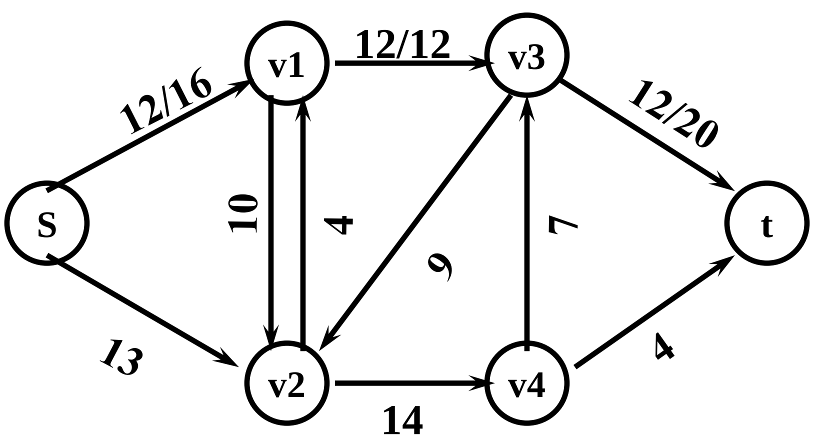
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G




```
for each edge (u, v) ∈ E [G]
    do f [u, v] = 0
        f [v, u] = 0
while there exists a path p from s to t
in the residual network G_f
    do c_f(p) = min{c_f(u, v) | (u, v) ∈ p}
        for each edge (u, v) in p
            do f [u, v] = f [u, v] + c_f(p)
                f [v, u] = - f [u, v]
```

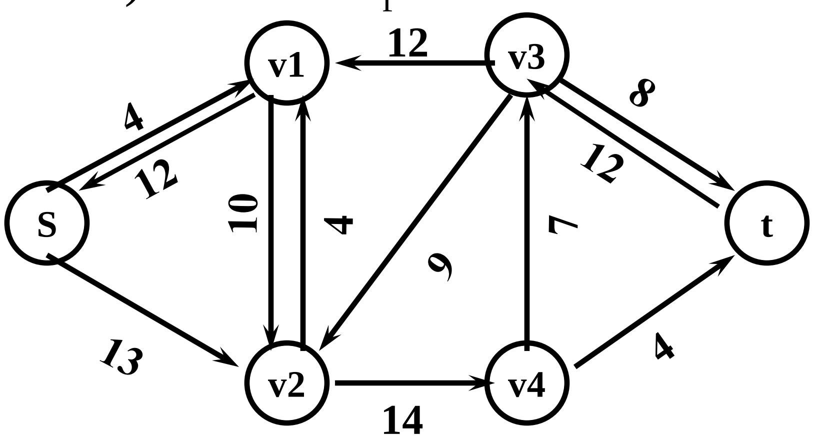
temporary variable:

$c_f\,(p) = 4$

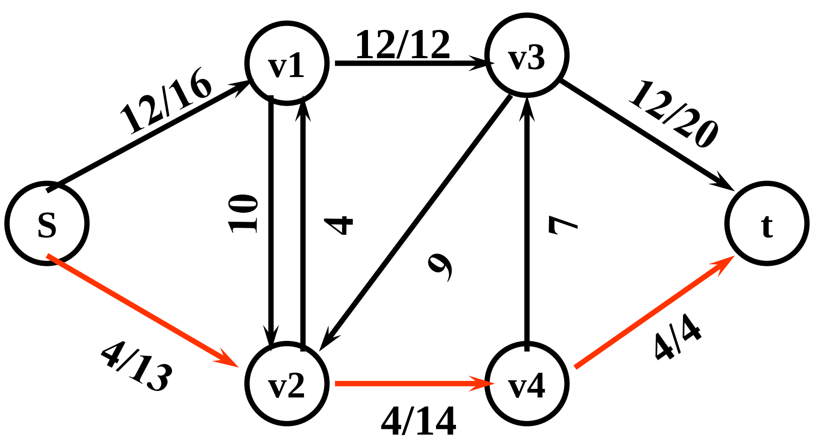
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G



for *each edge (u, v)* $\in$ *E [G]*
    do f [u, v] = 0
        f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
        for *each edge (u, v) in p*
            do $f[u, v] = f[u, v] + c_f(p)$
                $f[v, u] = - f[u, v]$

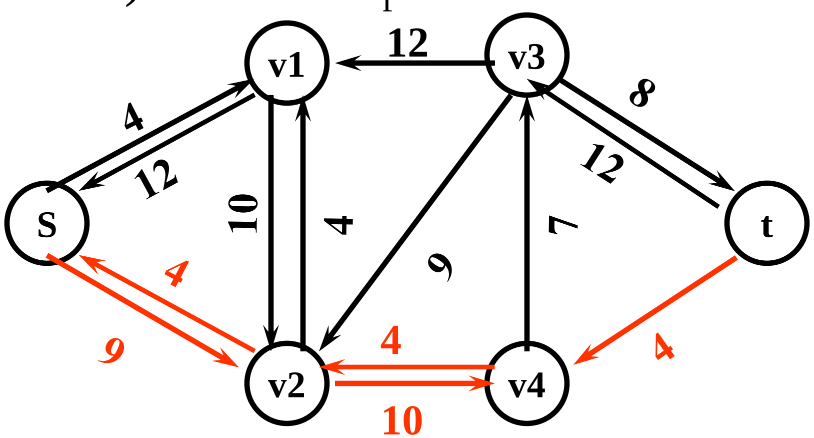
temporary variable:

$c_f(p) = 4$

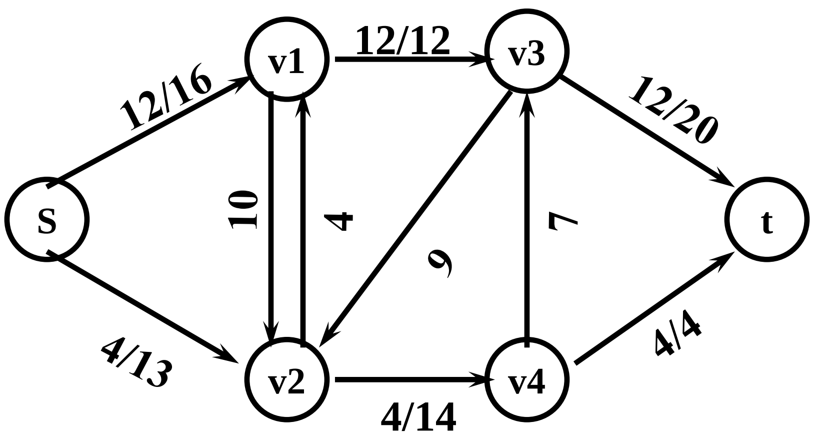
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G

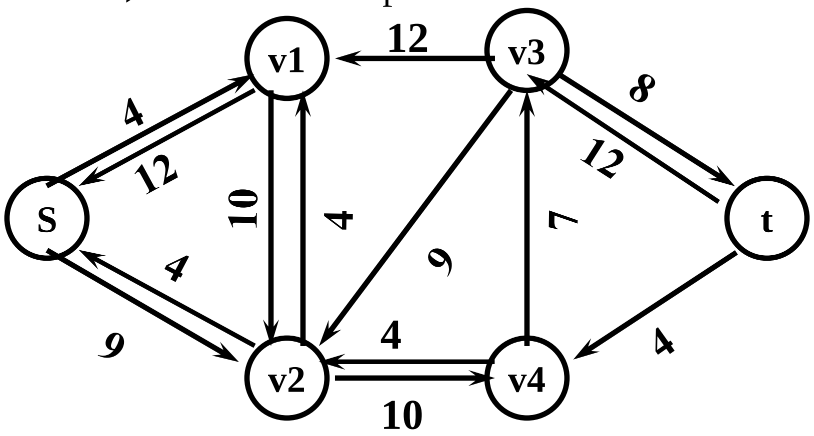


for *each edge* $(u, v) \in E[G]$
  do $f[u, v] = 0$
    $f[v, u] = 0$
while *there exists a path p from s to t in the residual network* G_f
  do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
    for *each edge* (u, v) *in p*
      do $f[u, v] = f[u, v] + c_f(p)$
        $f[v, u] = - f[u, v]$

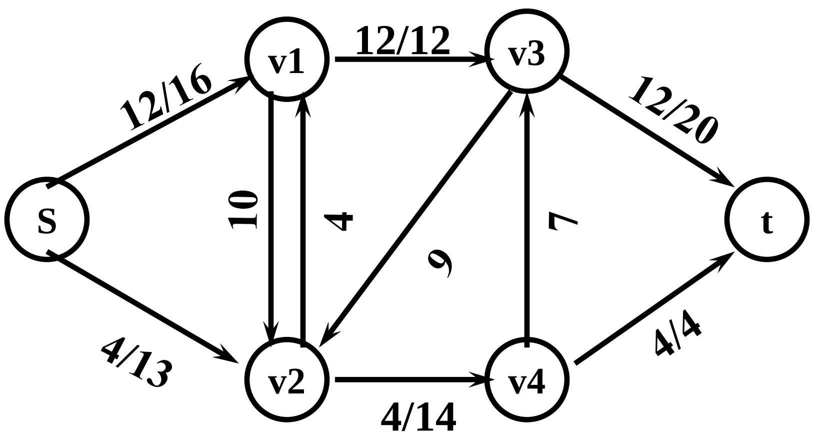
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G

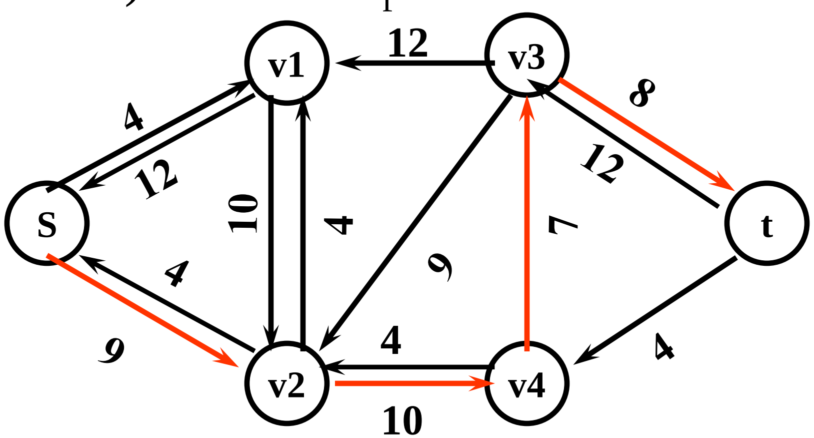


for *each edge* $(u, v) \in E[G]$
  do $f[u, v] = 0$
    $f[v, u] = 0$
while *there exists a path p from s to t in the residual network* G_f
  do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
    for *each edge* (u, v) *in p*
      do $f[u, v] = f[u, v] + c_f(p)$
        $f[v, u] = - f[u, v]$

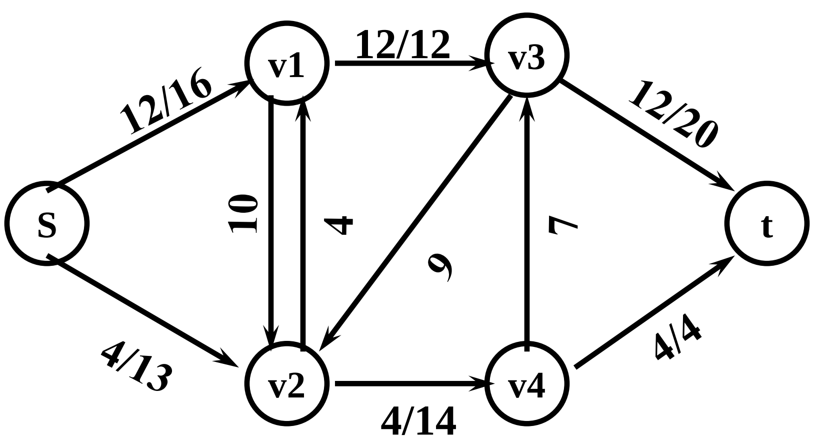
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G

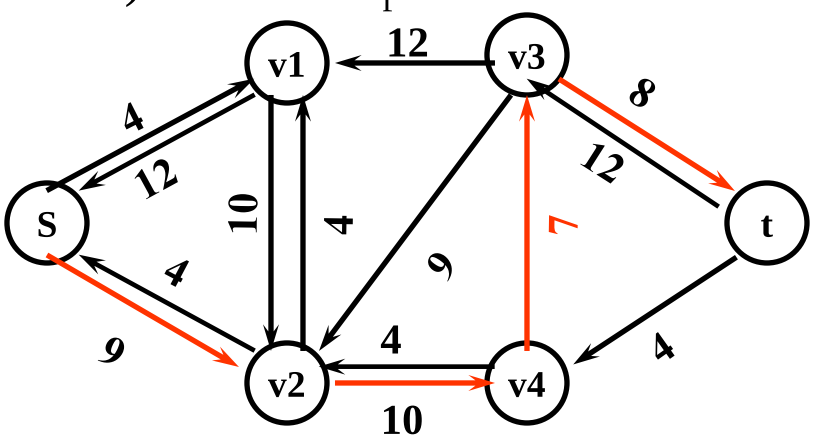


for *each edge (u, v)* $\in$ *E [G]*
    do f [u, v] = 0
        f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
        for *each edge (u, v) in p*
            do f [u, v] = f [u, v] + $c_f(p)$
                f [v, u] = - f [u, v]

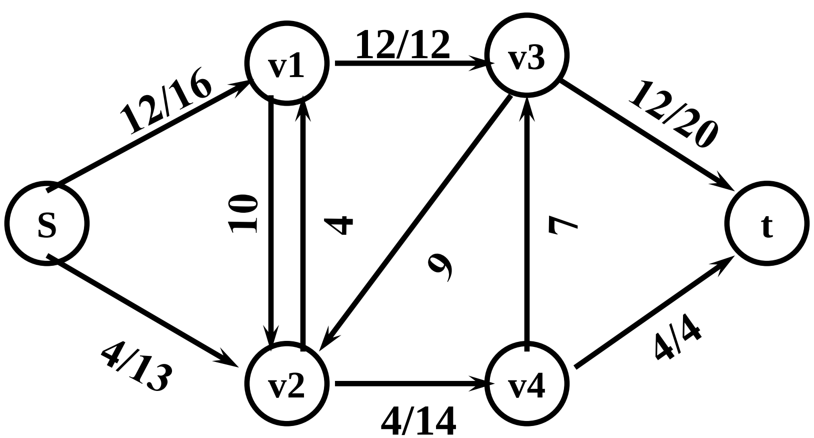
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G



for *each edge* $(u, v) \in E[G]$
    do f [u, v] = 0
        f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
        for *each edge (u, v) in p*
            do $f[u, v] = f[u, v] + c_f(p)$
                $f[v, u] = - f[u, v]$

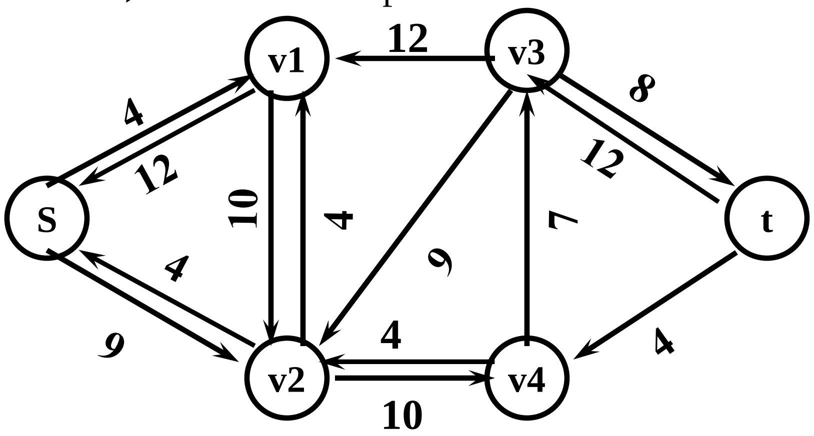
temporary variable:

$c_f(p) = 7$

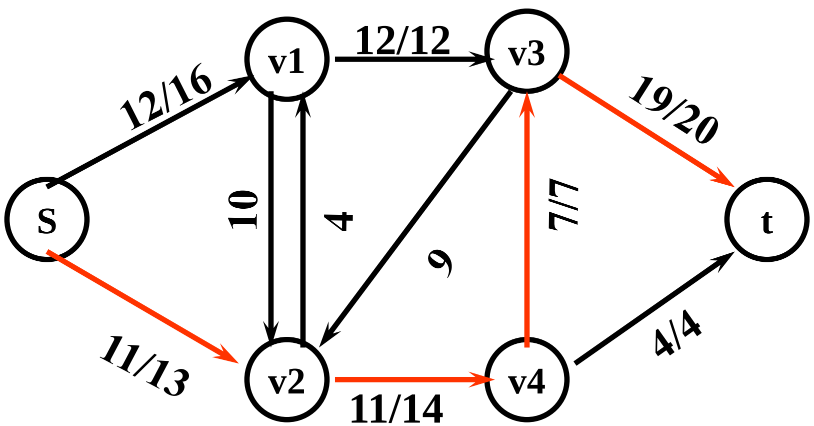
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G



for *each edge* $(u, v) \in E[G]$
  do $f[u, v] = 0$
    $f[v, u] = 0$
while *there exists a path p from s to t in the residual network* G_f
  do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
    for *each edge* (u, v) *in p*
      do $f[u, v] = f[u, v] + c_f(p)$
        $f[v, u] = - f[u, v]$

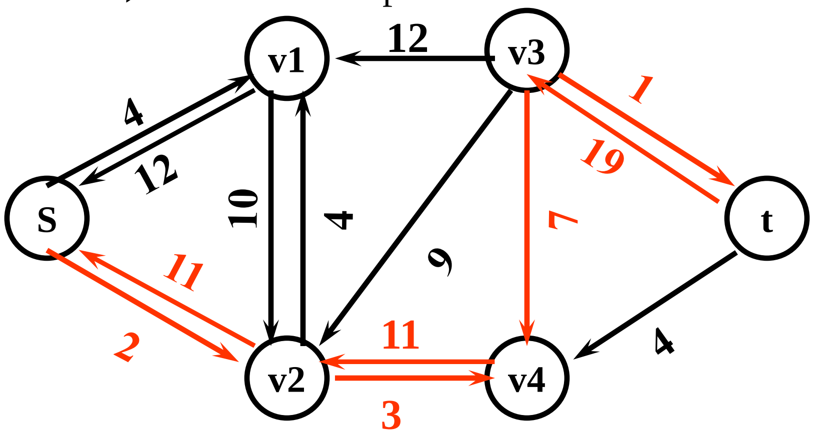
temporary variable:

$c_f(p) = 7$

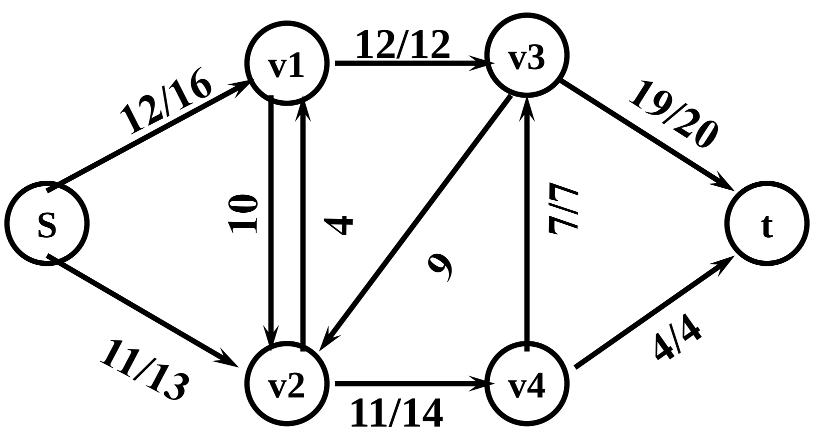
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G

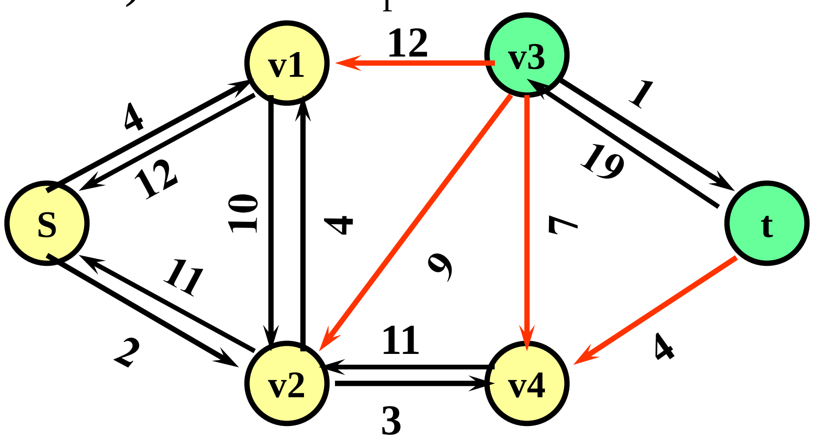


for *each edge (u, v)* $\in$ *E [G]*
    do f [u, v] = 0
        f [v, u] = 0
while *there exists a path p from s to t in the residual network* G_f
    do $c_f(p) = \min\{c_f(u, v) \mid (u, v) \in p\}$
        for *each edge (u, v) in p*
            do $f[u, v] = f[u, v] + c_f(p)$
                $f[v, u] = - f[u, v]$

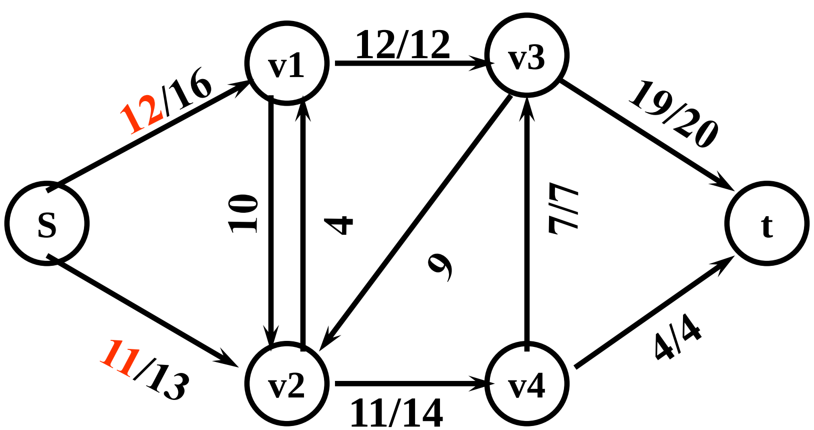
The basic Ford Fulkerson algorithm

example of an execution

(residual) network G_f



new flow network G




```
for each edge (u, v) ∈ E [G]
    do f [u, v] = 0
       f [v, u] = 0
while there exists a path p from s to t
in the residual network G_f
    do c_f(p) = min{c_f(u, v) | (u, v) ∈ p}
       for each edge (u, v) in p
           do f [u, v] = f [u, v] + c_f(p)
              f [v, u] = - f [u, v]
```

Finally we have:

$| f | = f(s, V) = 23$

Analysis of the Ford Fulkerson algorithm

Running time

```
1   for each edge (u, v) ∈ E [G]
2       do f [u, v] = 0
3          f [v, u] = 0
4   while there exists a path p from s to t
    in the residual network G_f
5       do c_f(p) = min{c_f(u, v) | (u, v) ∈ p}
6          for each edge (u, v) in p
7              do f [u, v] = f [u, v] + c_f(p)
8                 f [v, u] = - f [u, v]
```

The running time depends on how the augmenting path p in line 4 is determined.

Analysis of the Ford Fulkerson algorithm

Running time (arbitrary choice of p)

```
for each edge (u, v) ∈ E [G]
    do f [u, v] = 0                                        O(|E|)
        f [v, u] = 0
while there exists a path p from s to t                    O(|E|)
in the residual network G_f
    do c_f(p) = min{c_f(u, v) | (u, v) ∈ p}                O(|E| |f_max|)
        for each edge (u, v) in p
            do f [u, v] = f [u, v] + c_f(p)                O(|E|)
                f [v, u] = - f [u, v]
```

running time: $O(\ |E|\ |f_{max}|\ )$

with f_{max} as maximum flow

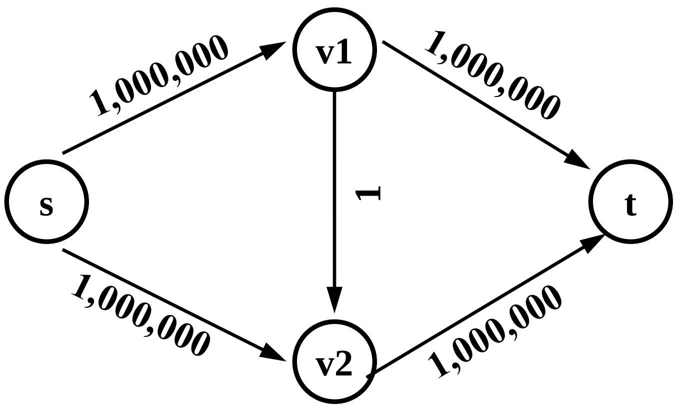
(1) The augmenting path is chosen arbitrarily and all capacities are integers

Analysis of the Ford Fulkerson algorithm

Running time (arbitrary choice of p)

Consequencies of an arbitrarily choice:

Example if |f*| is large:



running time: O (|E| $|f_{max}|$)

with f_{max} as maximum flow

(1) The augmenting path is chosen arbitrarily and all capacities are integers

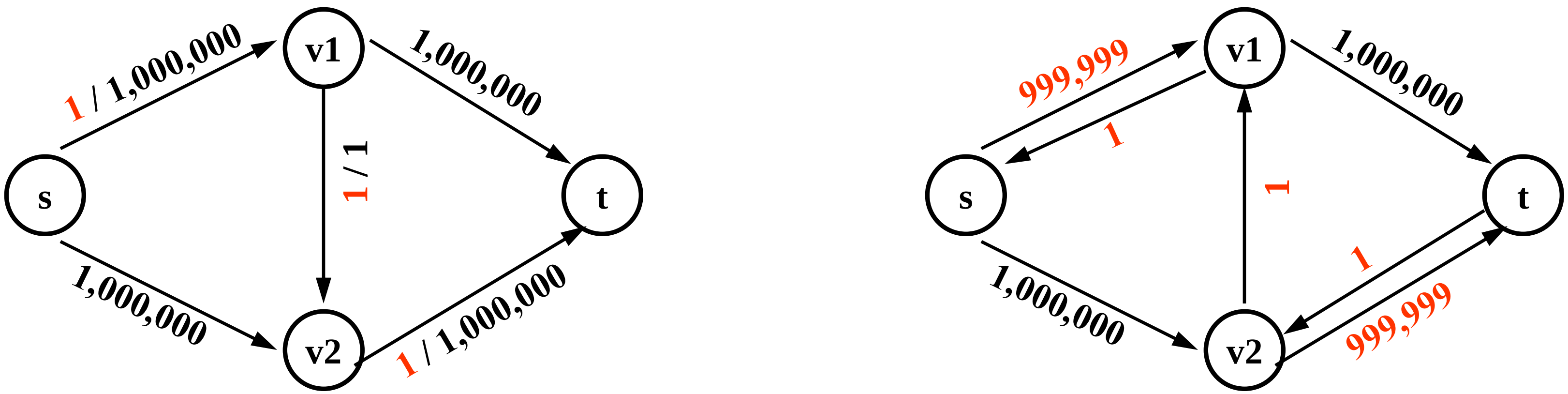
Analysis of the Ford Fulkerson algorithm

Running time (arbitrary choice of p)

Consequencies of an arbitrarily choice:

Example if $|f^*|$ is large:

residual network G_f



running time: $O(|E|\,|f_{max}|)$

with f_{max} as maximum flow

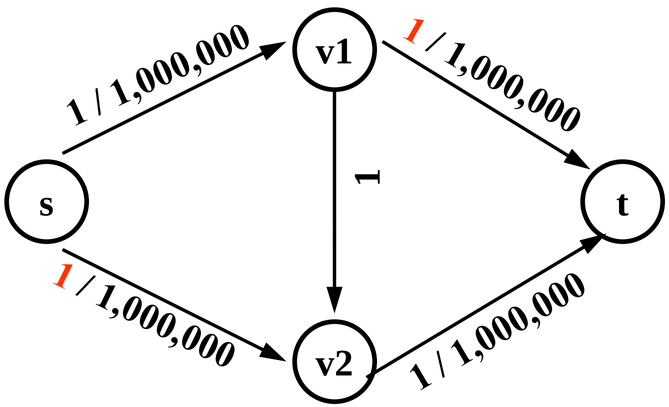
(1) The augmenting path is chosen arbitrarily and all capacities are integers

Analysis of the Ford Fulkerson algorithm

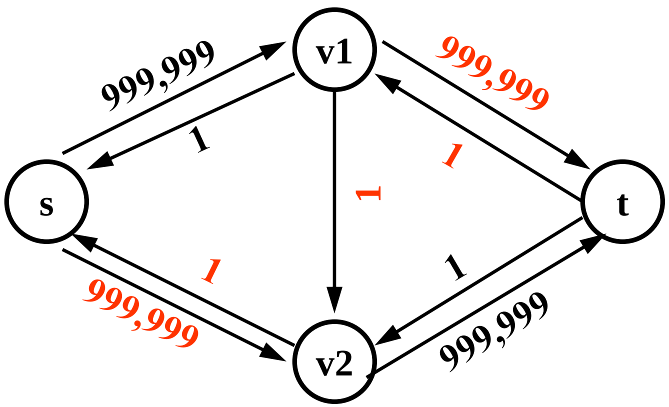
Running time (arbitrary choice of p)

Consequencies of an arbitrarily choice:

Example if $|f^*|$ is large:



residual network G_f



running time: $O(|E|\,|f_{max}|)$

with f_{max} as maximum flow

(1) The augmenting path is chosen arbitrarily and all capacities are integers

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

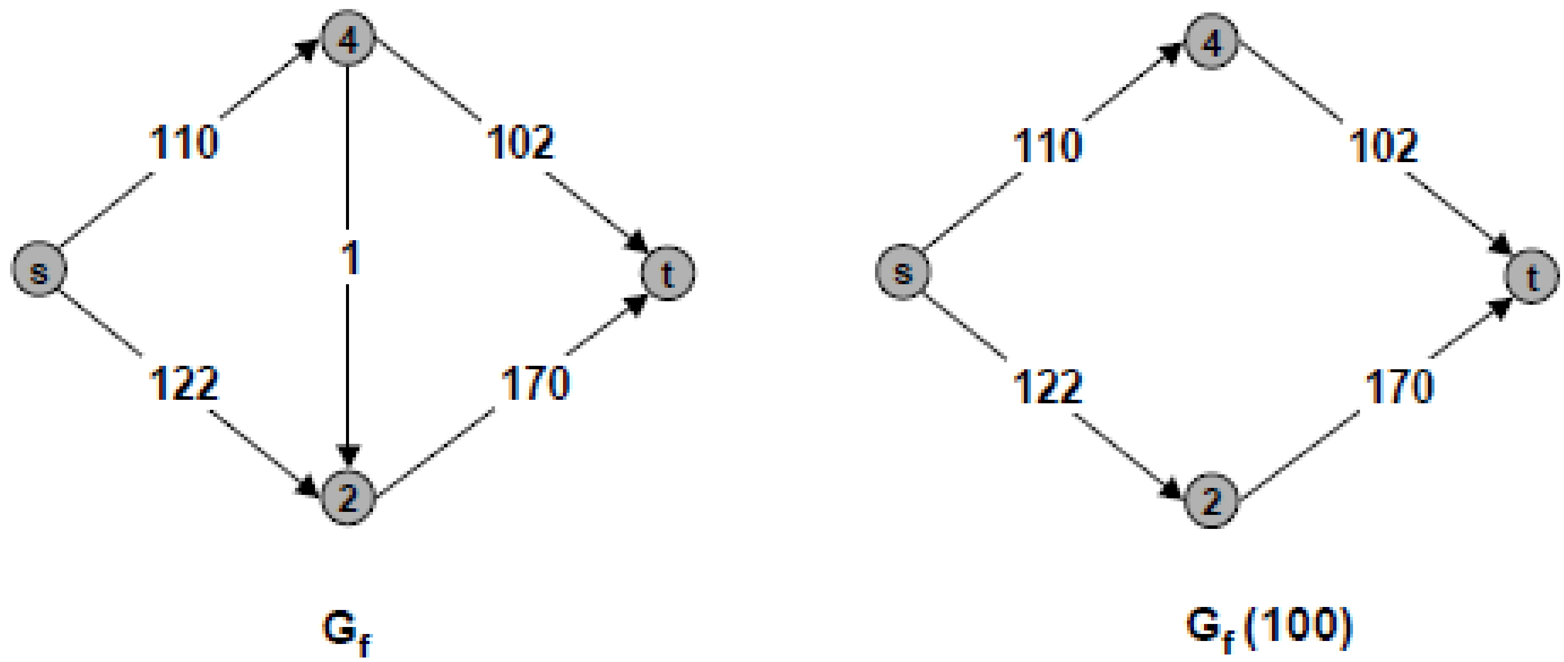
Edmonds-Karp (1972): choose augmenting path with

- Max bottleneck capacity. (fat path)
- Sufficiently large capacity. (capacity-scaling)
- Fewest number of arcs. (shortest path)

Capacity Scaling

Intuition: choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.



Capacity Scaling

Intuition: choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.

ScalingMaxFlow(V, E, s, t)

```
FOREACH e ∈ E, f(e) ← 0
Δ ← smallest power of 2 greater than or equal to U

WHILE (Δ ≥ 1)
   G_f(Δ) ← Δ-residual graph
   WHILE (there exists augmenting path P in G_f(Δ))
      f ← augment(f, P)
      update G_f(Δ)
   Δ ← Δ / 2
RETURN f
```

Capacity Scaling: Analysis

L1. If all arc capacities are integers, then throughout the algorithm, all flow and residual capacity values remain integers.

- Thus, $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$, so upon termination f is a max flow.

L2. The outer while loop repeats $1 + \lfloor \log_2 U \rfloor$ times.

- Initially $U \leq \Delta < 2U$, and Δ decreases by a factor of 2 each iteration.

L3. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $| f | + m \Delta$.

L4. There are at most 2m augmentations per scaling phase.

- Let f be the flow at the end of the previous scaling phase.
- L3 $\Rightarrow |f^*| \leq | f | + m (2\Delta)$.
- Each augmentation in a Δ-phase increases $| f |$ by at least Δ.

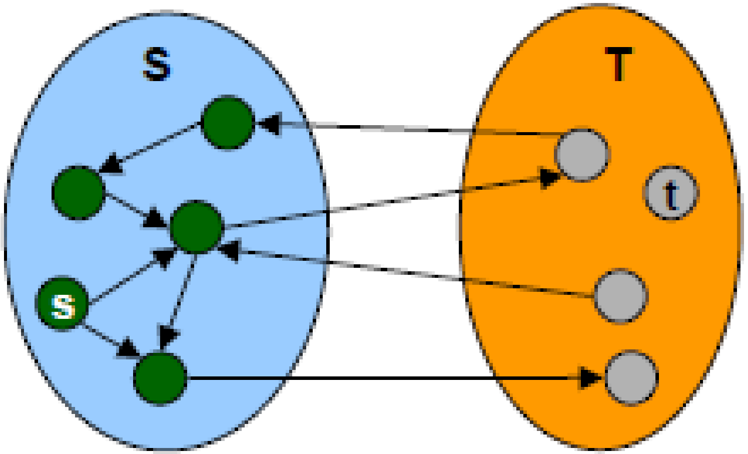
Theorem. The algorithm runs in $O(m^2 \log (2U))$ time.

Capacity Scaling: Analysis

L3. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $|f| + m\,\Delta$.

- We show that at the end of a Δ-phase, there exists a cut (S, T) such that $\text{cap}(S, T) \le |f| + m\,\Delta$.
- Choose S to be the set of nodes reachable from s in $G_f(\Delta)$.
  - clearly $s \in S$, and $t \notin S$ by definition of S

$$
\begin{aligned}
|f| &= \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) \\
&\ge \sum_{e \text{ out of } S} (u(e) - \Delta) - \sum_{e \text{ in to } S} \Delta \\
&= \sum_{e \text{ out of } S} u(e) - \sum_{e \text{ out of } S} \Delta - \sum_{e \text{ in to } S} \Delta \\
&= cap(S, T) - m\Delta
\end{aligned}
$$



Original Network

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Edmonds-Karp (1972): choose augmenting path with

- Max bottleneck capacity. (fat path)
- Sufficiently large capacity. (capacity-scaling)
- Fewest number of arcs. (shortest path)

Shortest Augmenting Path

Intuition: choosing path via breadth first search.

- Easy to implement.
  - may implement by coincidence!
- Finds augmenting path with fewest number of arcs.

ShortestAugmentingPath(V, E, s, t)

```
FOREACH e ∈ E
   f(e) ← 0
G_f ← residual graph

WHILE (there exists augmenting path)
   find such a path P by BFS
   f ← augment(f, P)
   update G_f
RETURN f
```

Shortest Augmenting Path: Overview of Analysis

L1. Throughout the algorithm, the length of the shortest path never decreases.

- Proof ahead.

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

- Proof ahead.

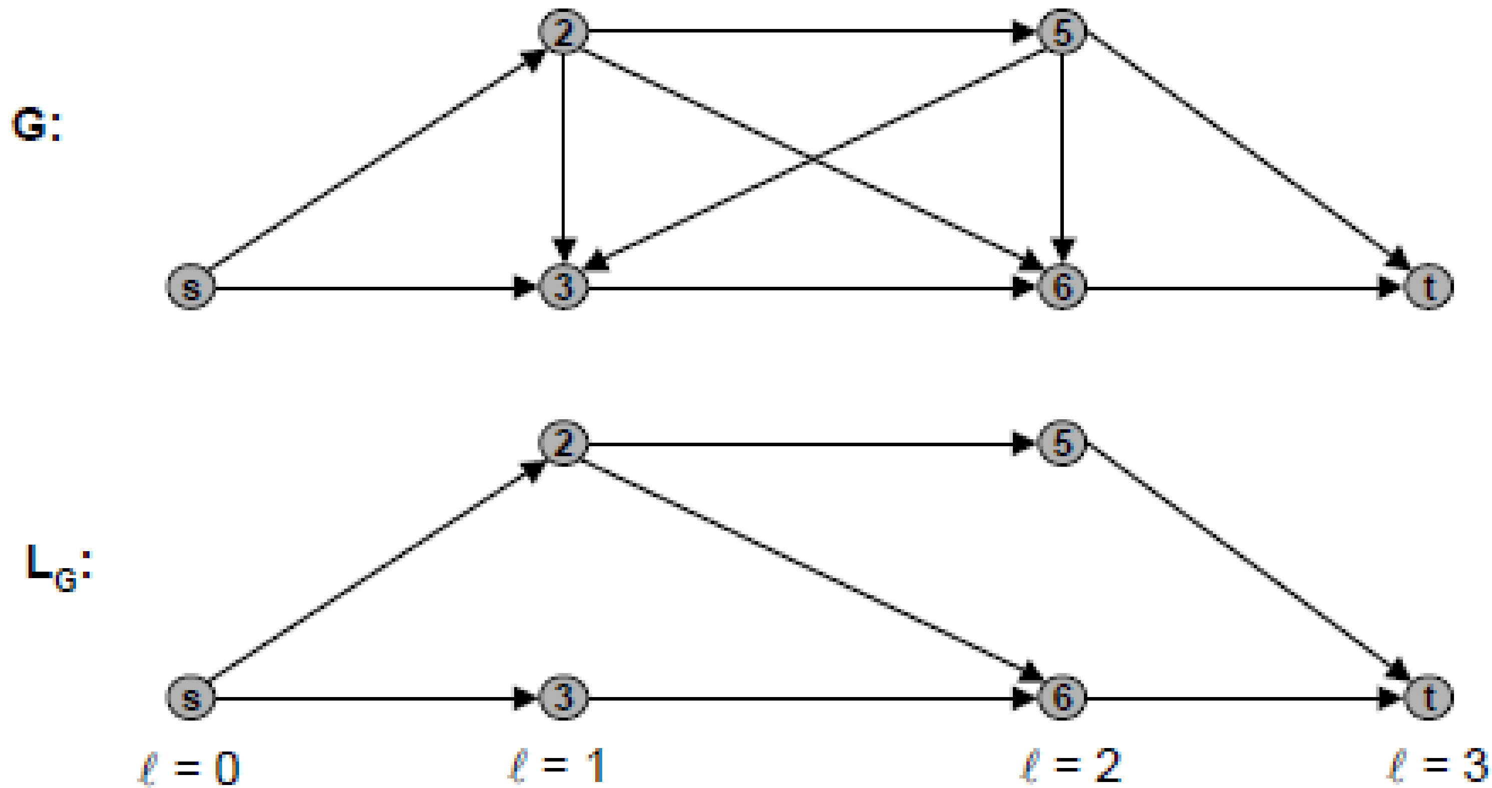
Theorem. The shortest augmenting path algorithm runs in $O(m^2n)$ time.

- $O(m+n)$ time to find shortest augmenting path via BFS.
- $O(m)$ augmentations for paths of exactly k arcs.
- If there is an augmenting path, there is a simple one.
  - $\Rightarrow$ $1 \leq k < n$
  - $\Rightarrow$ $O(mn)$ augmentations.

Shortest Augmenting Path: Analysis

Level graph of (V, E, s).

- For each vertex v, define $\ell(v)$ to be the length (number of arcs) of shortest path from s to v.
- $L_G = (V, E_G)$ is subgraph of G that contains only those arcs $(v,w) \in E$ with $\ell(w) = \ell(v) + 1$.

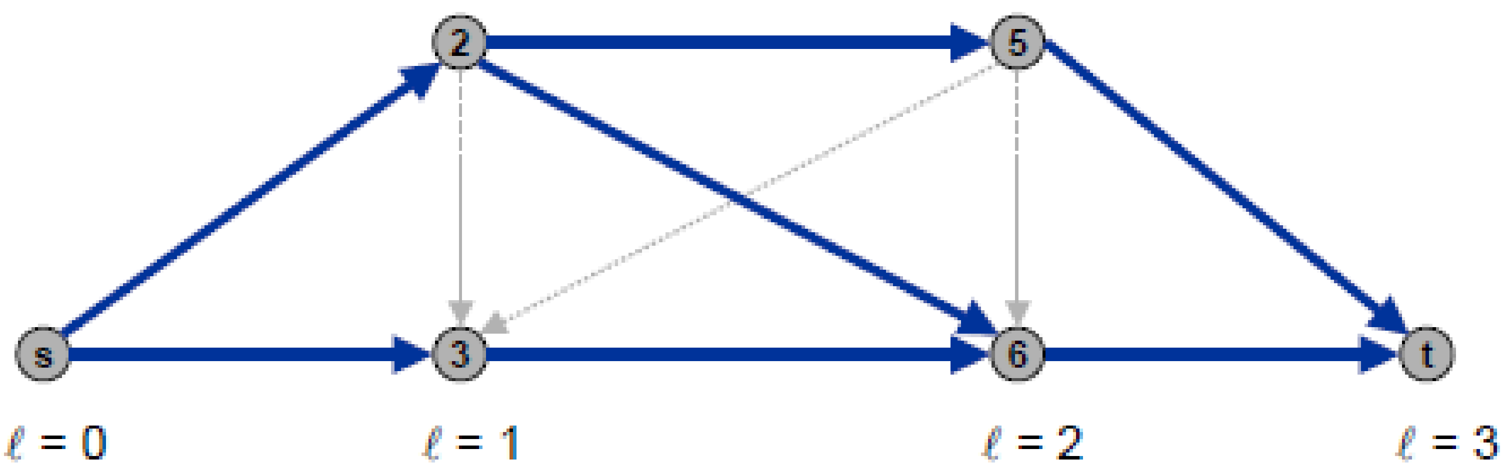


Shortest Augmenting Path: Analysis

Level graph of (V, E, s).

- For each vertex v, define $\ell(v)$ to be the length (number of arcs) of shortest path from s to v.
- L = (V, F) is subgraph of G that contains only those arcs $(v,w) \in E$ with $\ell(w) = \ell(v) + 1$.
- Compute in O(m+n) time using BFS, deleting back and side arcs.
- P is a shortest s-v path in G if and only if it is an s-v path L.

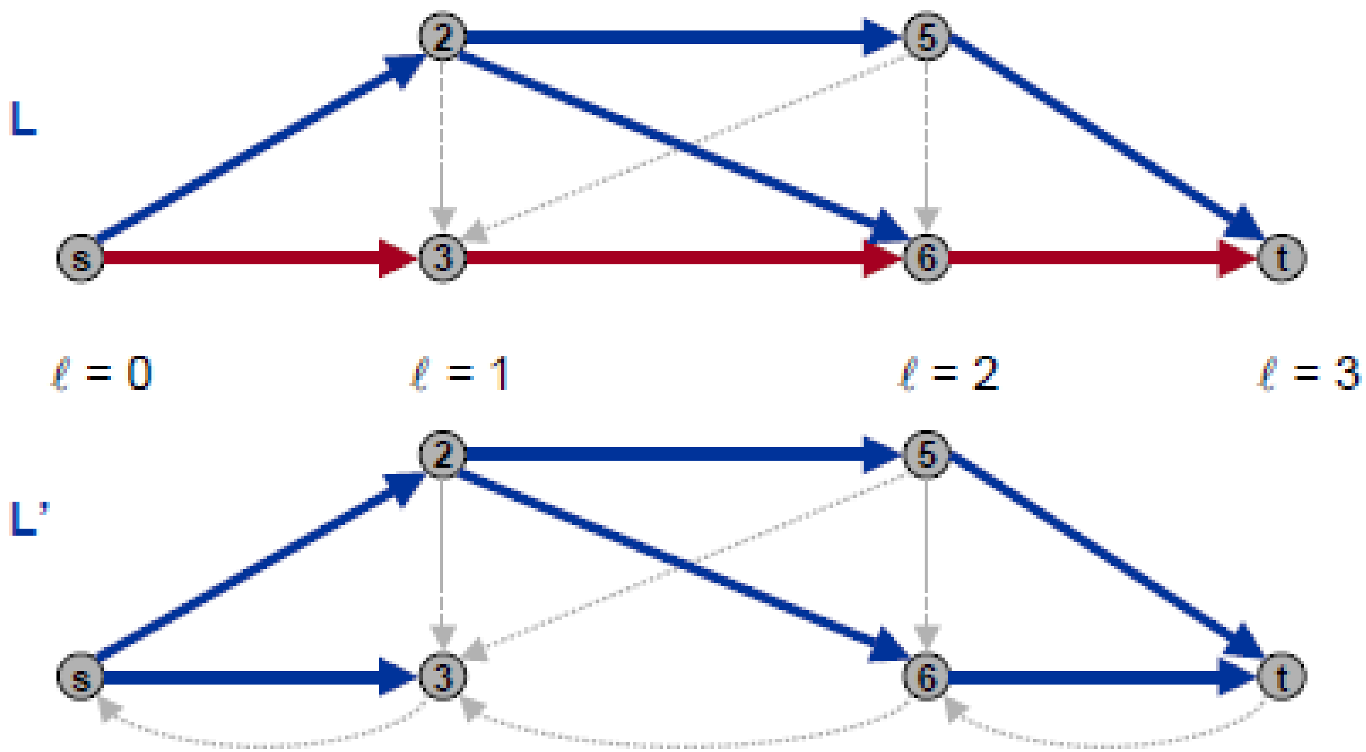
L:



Shortest Augmenting Path: Analysis

L1. Throughout the algorithm, the length of the shortest path never decreases.

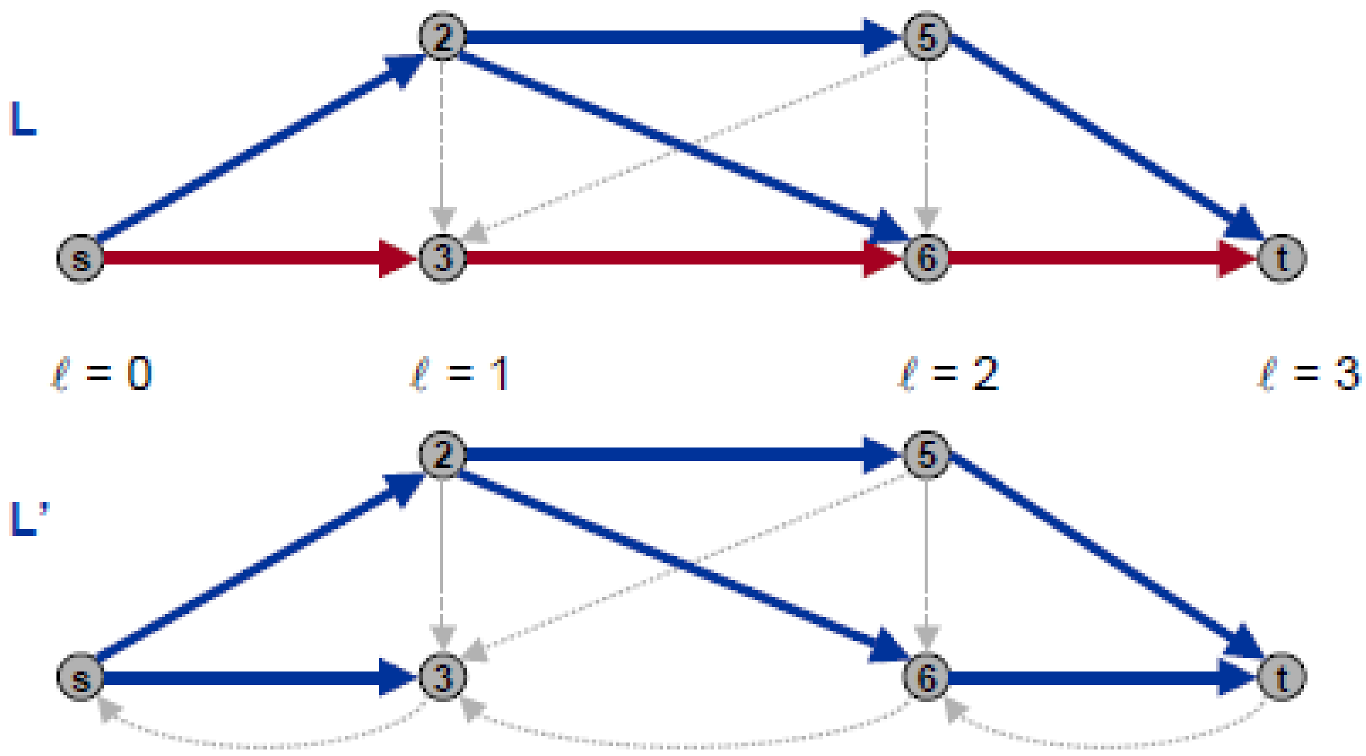
- Let f and f' be flow before and after a shortest path augmentation.
- Let L and L' be level graphs of G_f and $G_{f'}$.
- Only back arcs added to $G_{f'}$.
  - path with back arc has length greater than previous length



Shortest Augmenting Path: Analysis

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

- At least one arc (the bottleneck arc) is deleted from L after each augmentation.
- No new arcs added to L until length of shortest path strictly increases.



Shortest Augmenting Path: Review of Analysis

L1. Throughout the algorithm, the length of the shortest path never decreases.

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

Theorem. The shortest augmenting path algorithm runs in $O(m^2n)$ time.

- $O(m+n)$ time to find shortest augmenting path via BFS.
- $O(m)$ augmentations for paths of exactly k arcs.
- $O(mn)$ augmentations.

Note: $\Theta(mn)$ augmentations necessary on some networks.

- Try to decrease time per augmentation instead.
- Dynamic trees $\Rightarrow$ $O(mn \log n)$ Sleator-Tarjan, 1983
- Simple idea $\Rightarrow$ $O(mn^2)$

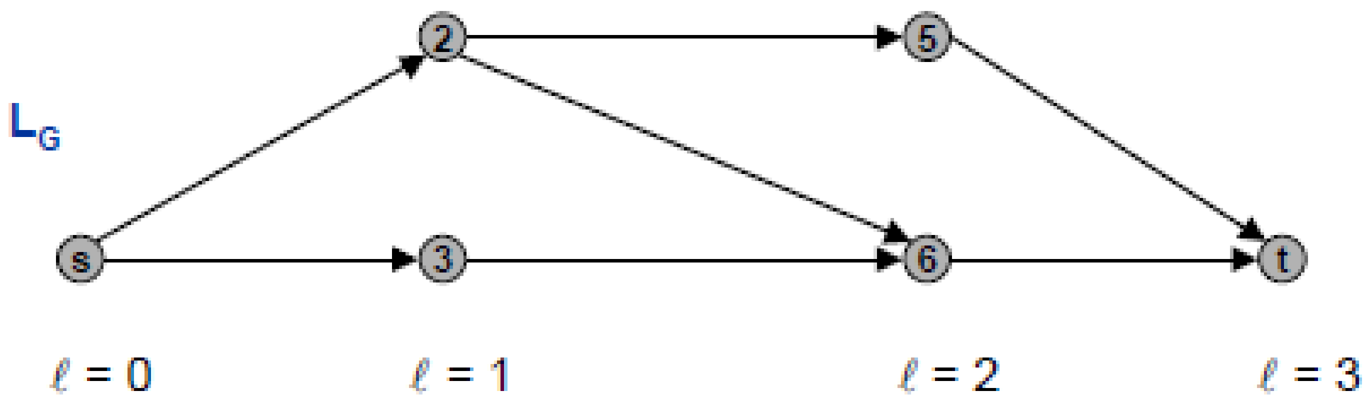
Shortest Augmenting Path: Improved Version

Two types of augmentations.

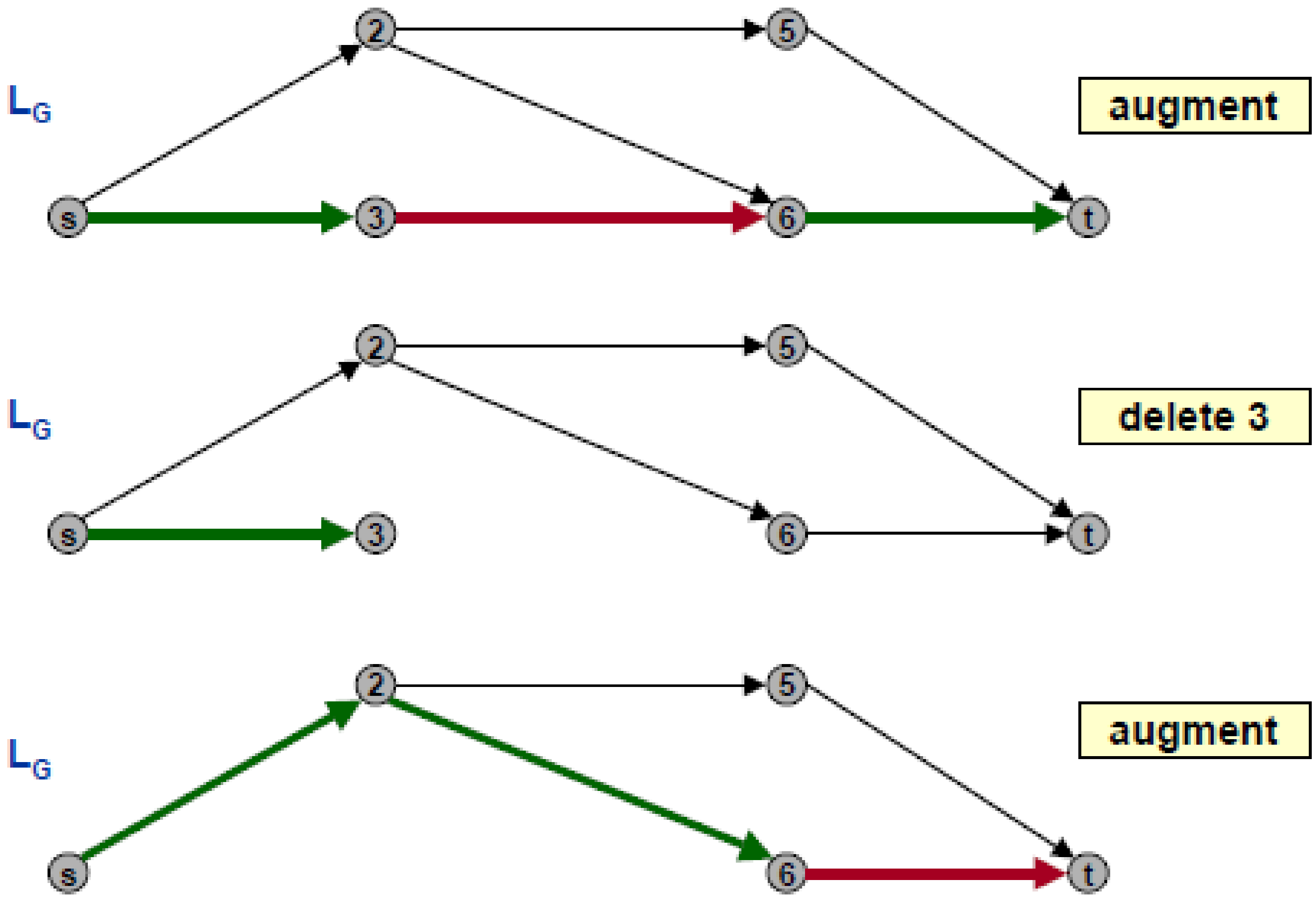
- Normal augmentation: length of shortest path doesn't change.
- Special augmentation: length of shortest path strictly increases.

L3. Group of normal augmentations takes O(mn) time.

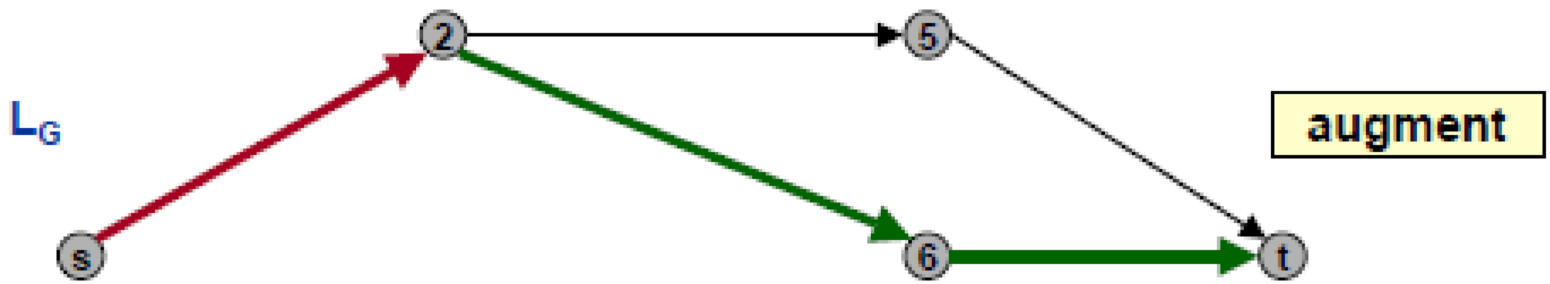
- Explicitly maintain level graph - it changes by at most 2n arcs after each normal augmentation.
- Start at s, advance along an arc in L_G until reach t or get stuck.
  - if reach t, augment and delete at least one arc
  - if get stuck, delete node

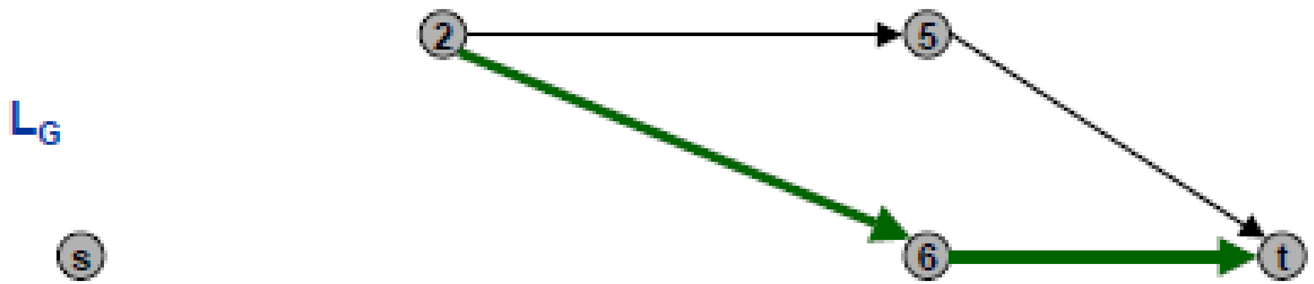


Shortest Augmenting Path: Improved Version



Shortest Augmenting Path: Improved Version





STOP: length of shortest path must have strictly increased

Shortest Augmenting Path: Improved Version

AdvanceRetreat(V, E, f, s, t)

```
ARRAY pred[v ∈ V]
L_G ← level graph of G_f
v ← s, pred[v] ← nil

REPEAT
   WHILE (there exists (v,w) ∈ L_G)
      pred[w] ← v, v ← w                    advance
      IF (v = t)
         P ← path defined by pred[]         augment
         f ← augment(f, P)
         update L_G
         v ← s, pred[v] ← nil
   delete v from L_G                        retreat
UNTIL (v = s)

RETURN f
```

Shortest Augmenting Path: Improved Version

Two types of augmentations.

- Normal augmentation: length of shortest path doesn't change.
- Special augmentation: length of shortest path strictly increases.

L3. Group of normal augmentations takes O(mn) time.

- Explicitly maintain level graph - it changes by at most 2n arcs after each normal augmentation.
- Start at s, advance along an arc in L_G until reach t or get stuck.
  - if reach t, augment and delete at least one arc
  - if get stuck, delete node
  - at most n advance steps before one of above events

Theorem. Algorithm runs in $O(mn^2)$ time.

- O(mn) time between special augmentations.
- At most n special augmentations.

Choosing Good Augmenting Paths: Summary

Method	Augmentations	Running time
Augmenting path	nU	mnU
Max capacity	m log U	m log U (m + n log n)
Capacity scaling	m log U	m^2 log U
Improved capacity scaling	m log U	mn log U
Shortest path	mn	m^2n
Improved shortest path	mn	mn^2

First 4 rules assume arc capacities are between 0 and U.

History

Year	Discoverer	Method	Big-Oh
1951	Dantzig	Simplex	mn^2U
1955	Ford, Fulkerson	Augmenting path	mnU
1970	Edmonds-Karp	Shortest path	m^2n
1970	Dinitz	Shortest path	mn^2
1972	Edmonds-Karp, Dinitz	Capacity scaling	$m^2 \log U$
1973	Dinitz-Gabow	Capacity scaling	$mn \log U$
1974	Karzanov	Preflow-push	n^3
1983	Sleator-Tarjan	Dynamic trees	$mn \log n$
1986	Goldberg-Tarjan	FIFO preflow-push	$mn \log (n^2 / m)$
. . .	. . .	. . .	. . .
1997	Goldberg-Rao	Length function	$m^{3/2} \log (n^2 / m) \log U$ $mn^{2/3} \log (n^2 / m) \log U$

MAXIMUM FLOW: THE PREFLOW/PUSH METHOD

Goldberg-Tarjan

Some definitions

- Saturated edge (u,v) $\Leftrightarrow$ c(u,v) = f(u,v)
- r(u, v) = c(u,v) – f(u,v)
- Residual graph R(V, E`) where E` is all the edges (u,v) where r(u,v) $\geq$0

Overview

- Use the residual graph
- Don’t look for augmenting paths
- Instead saturate all outgoing edges of the source and strive to make this “preflow” reach the sink
- Otherwise have to flow it back.

Preflow

- **Flow constrains:**

$\forall (u,v) \in V \qquad f(u,v) \le c(u,v)$

$\forall (u,v) \in V \qquad f(v,u) = -f(v,u)$

$\forall v \in V-\{s\} \qquad \sum f(u,v) \ge 0$

- **Every vertex *v* may keep some “excess” flow *e(v)* inside the vertex**

Excess handeling

- Strive to push this excess toward the sink
- If the sink is not reachable on the residual graph the algorithm pushes the excess toward the source
- When no vertices with e(v) >0 are left the algorithm halts, and the resulting flow (!) is the max –flow

Valid distance labeling

- A mapping function $d: V \rightarrow N + \{ \infty \}$
- $d(s) = n$, $d(t) = 0$
- $r(u,v) > 0 \rightarrow d(u) \leq d(v)+1$
- $d(v) < n \rightarrow d(v)$ is the lower bound on the distance from v to the sink (residual graph)
  - Let $p= v, v_1 ,v_2 , v_{3....} v_k ,t$ be the s.p. $v \rightarrow t$
  - $d(v) \leq d(v_1) + 1 \leq d(v_2) + 2 ..... \leq d(t) + k = k$
- Same way $d(v) \geq n \rightarrow d(v) -n$ is the lower bound on the distance from v to the source

Active vertex

- Active vertex:
  - $v \in V-\{s,t\}$ is active if
    - $d(v) < \infty$
    - $e(v) > 0$
- Eventually, I'll show that $d(v)$ is always finite and therefore only the $e(v) > 0$ part is relevant

Basic operations

- Applied on active vertices only

- Push (u,v)
  - Requires: $r(u,v) > 0$, $d(u)=d(v)+1$
  - Action:
    - $\delta = \min(e(v) , r(u,v))$
    - $f(u,v) \mathrel{+}= \delta$, $f(v,u) \mathrel{-}= \delta$
    - $e(u) \mathrel{-}= \delta$, $e(v) \mathrel{+}= \delta$

Basic operations , contd.

- Relabel (u)
  - Requires: $\forall\ v \in V\ r(u,v) > 0 \rightarrow d(u) \leq d(v)$
  - Action:
    - $d(u) = \min \{ d(v) + 1 \mid r(u,v) > 0 \}$

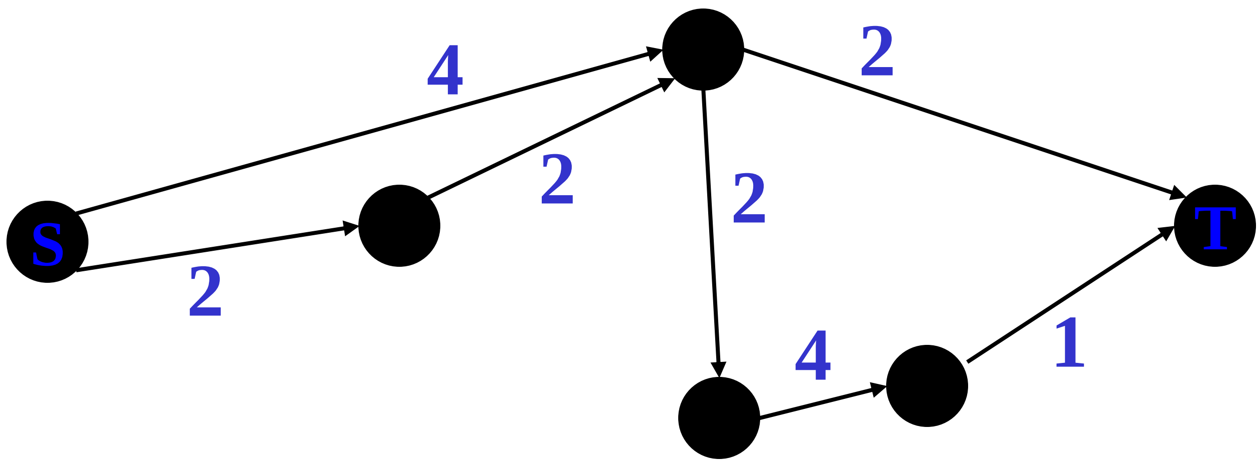
- Lemma. One of the basic operations is applicable on an active vertex:
  - PUSH: Any residual edge (u,v) with $d(u) = d(v) + 1$
  - Otherwise: $d(u) \leq d(v)$ for all residual edges, allows relabel

The algorithm

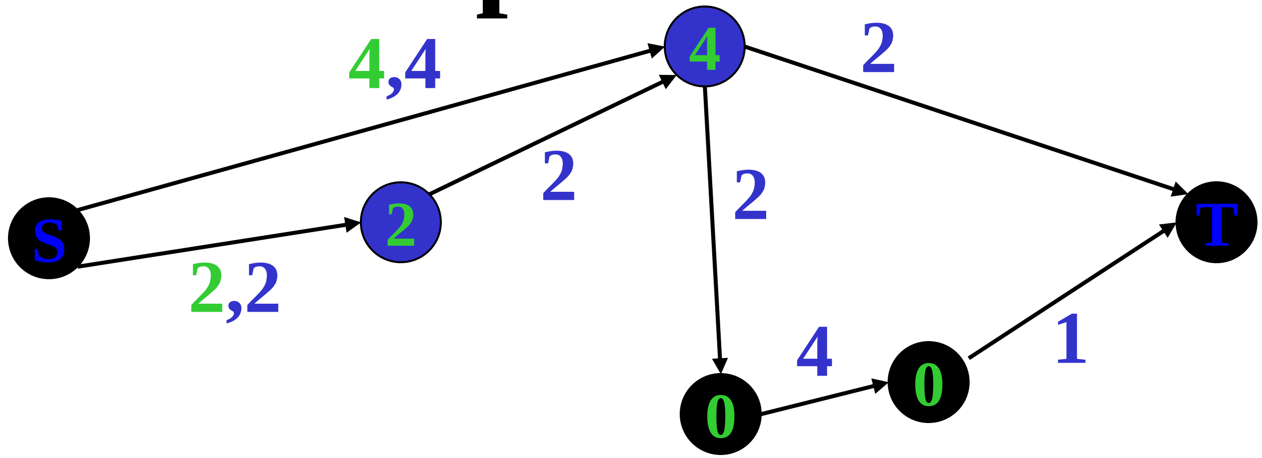
- Initialize: d(s) = n, $v \in V-\{s\}$ d(v) =0
- Saturate the outgoing edges of s
- While there are active vertices apply one of the basic actions on the vertex

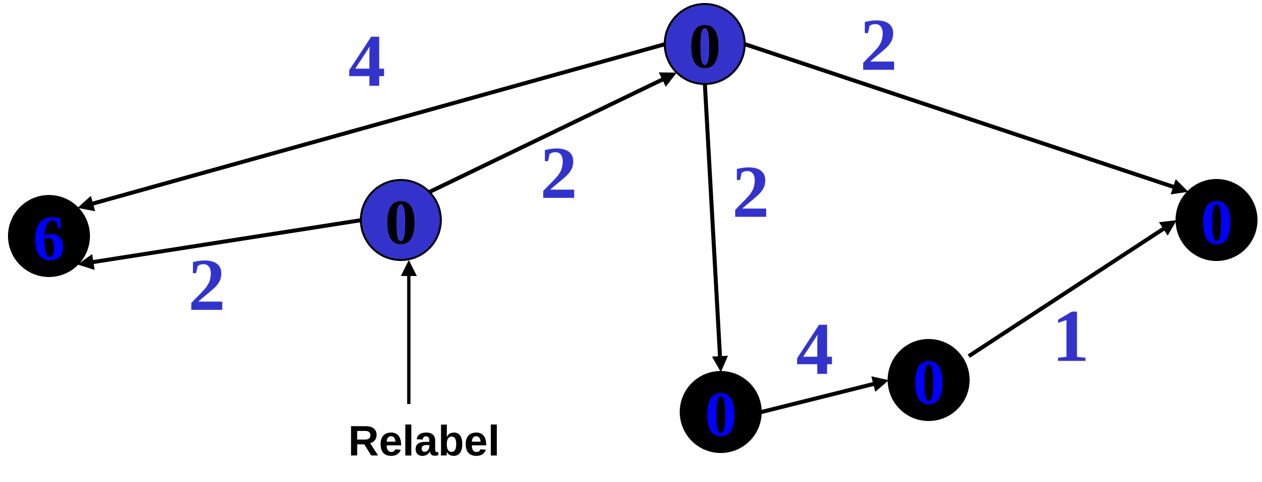
- Simple!

Example - Saturate all source edges

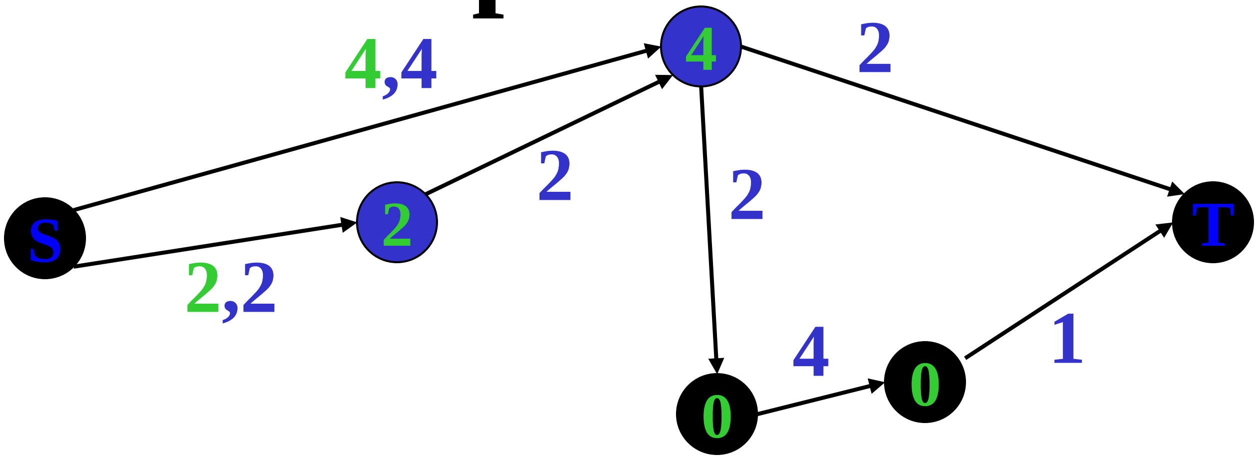


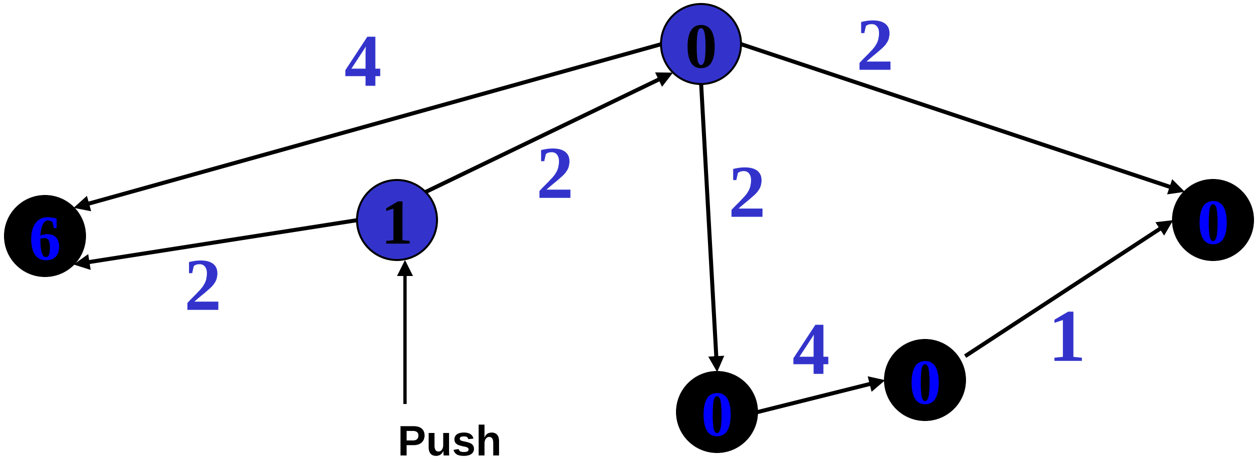
Example – contd.



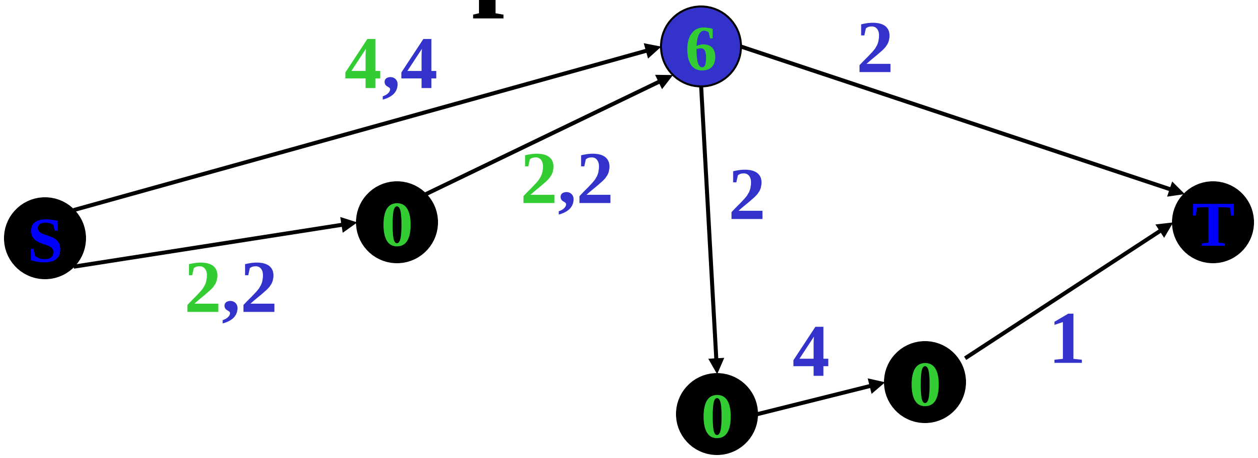


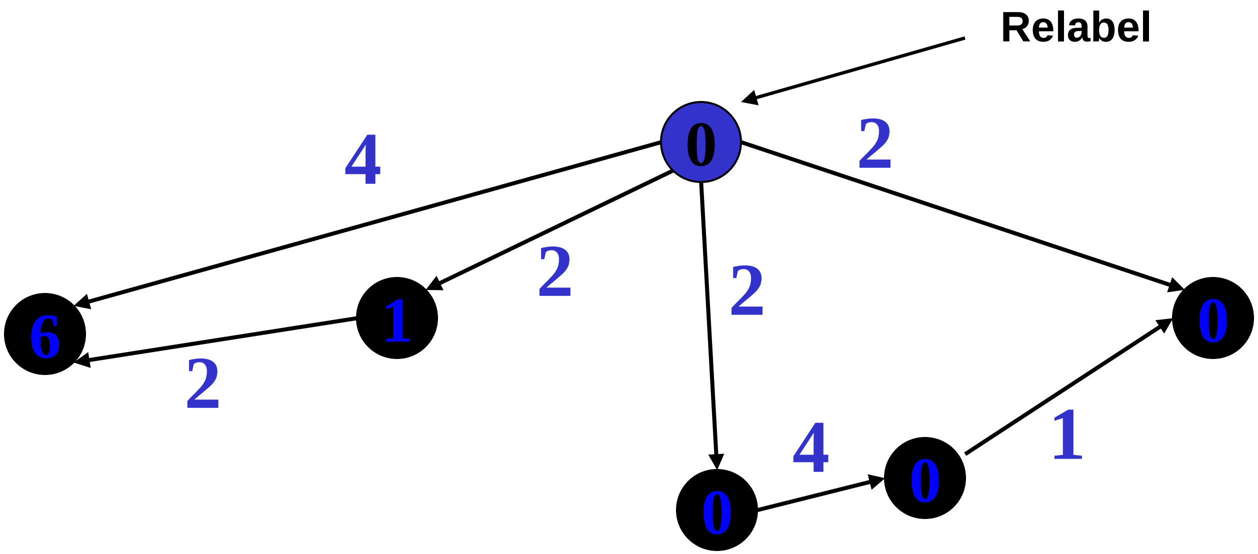
Example – contd.



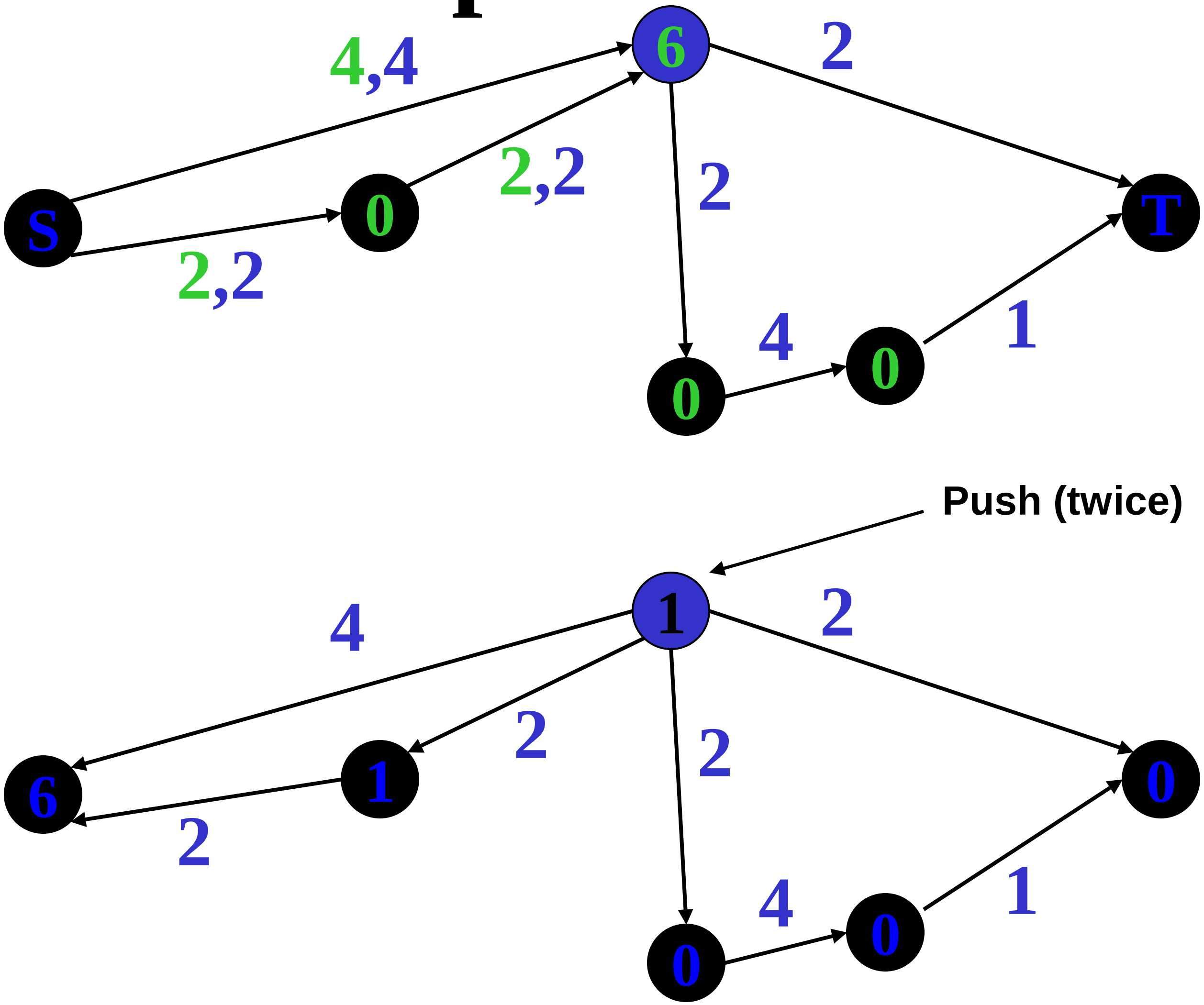


Example – contd.

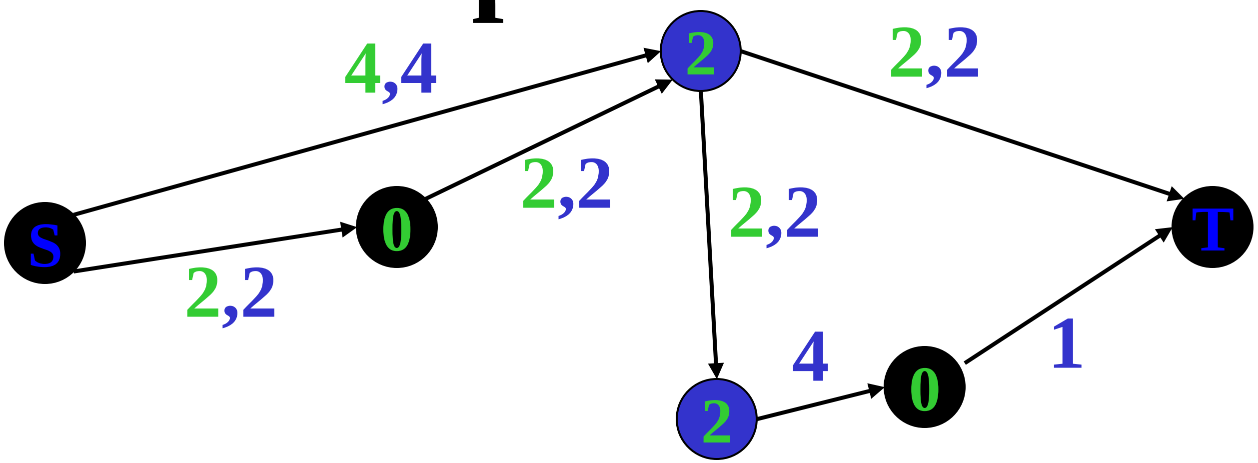


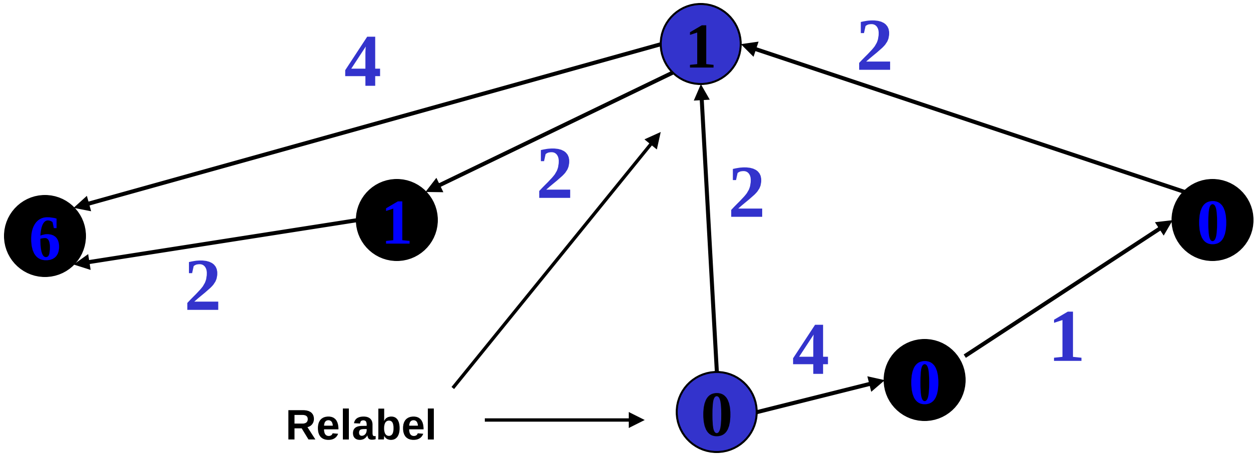


Example – contd.

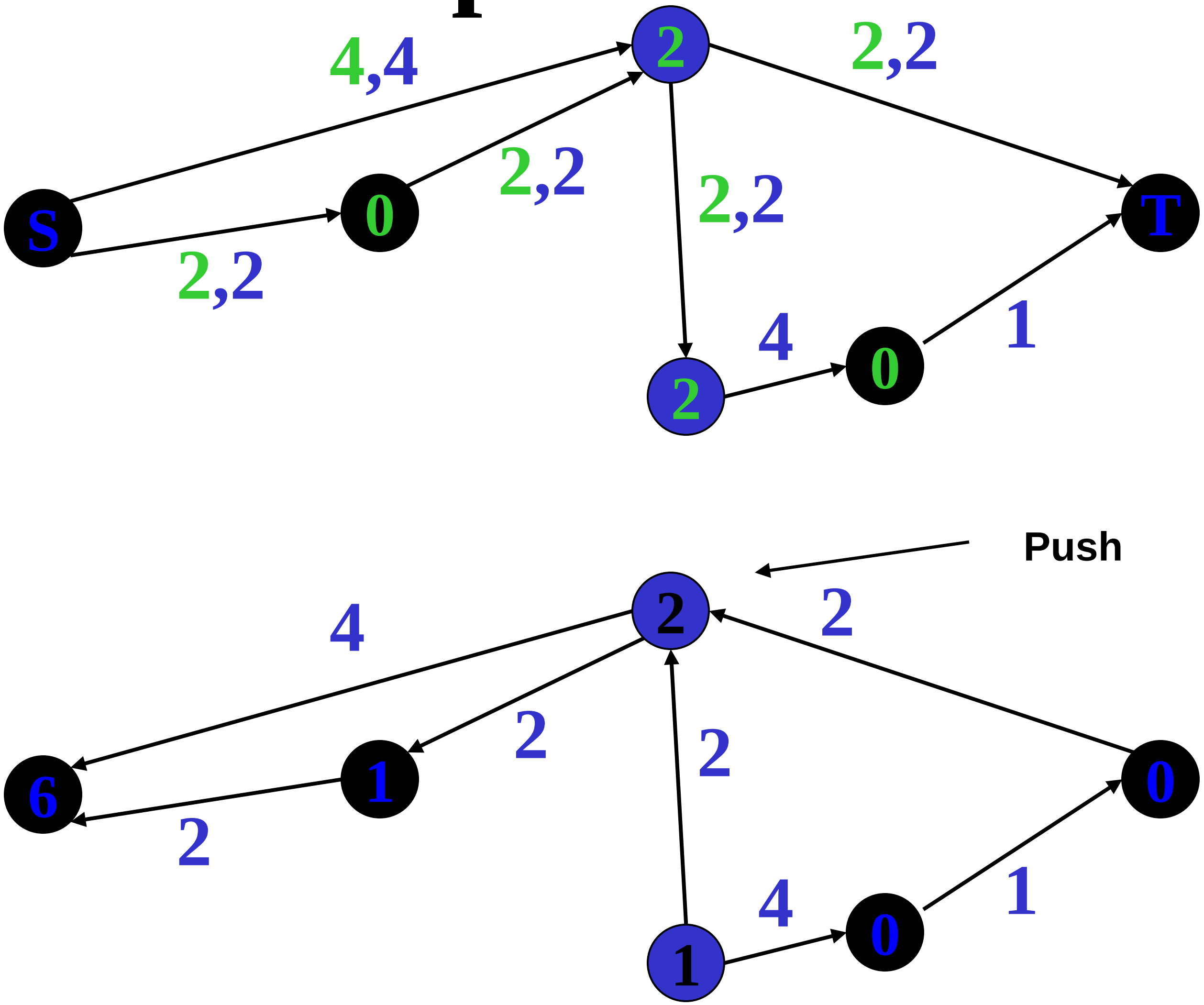


Example – contd.

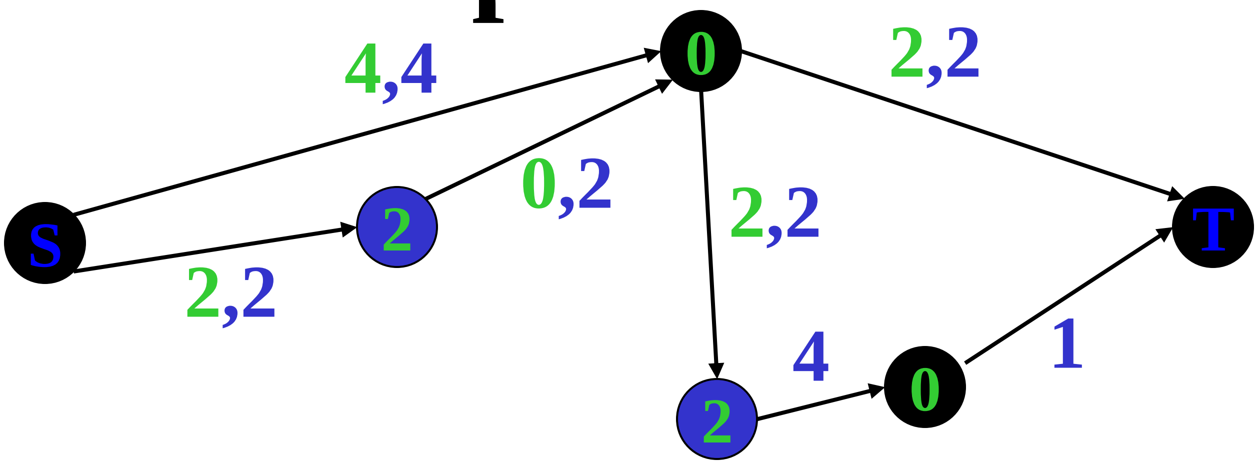


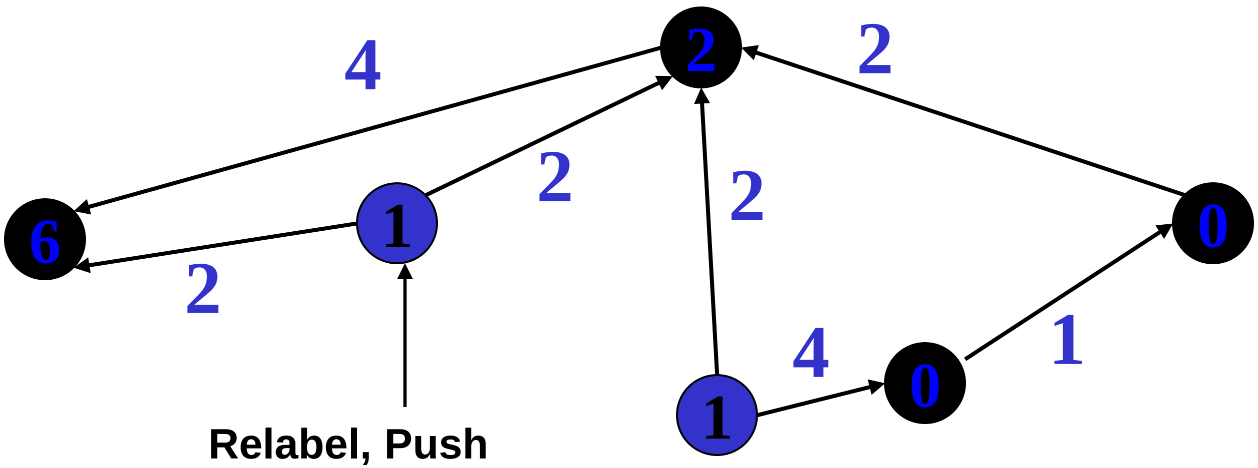


Example – contd.

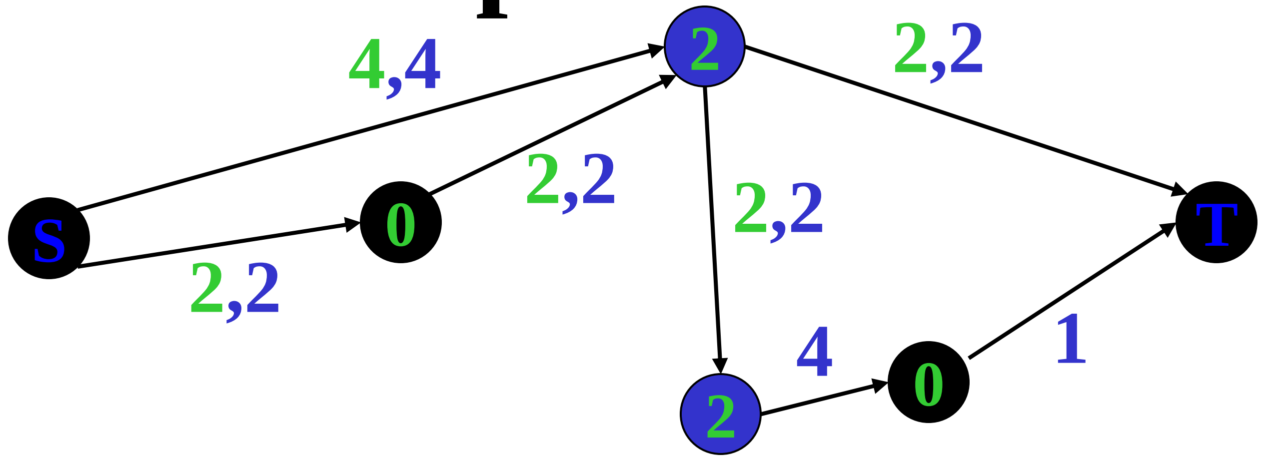


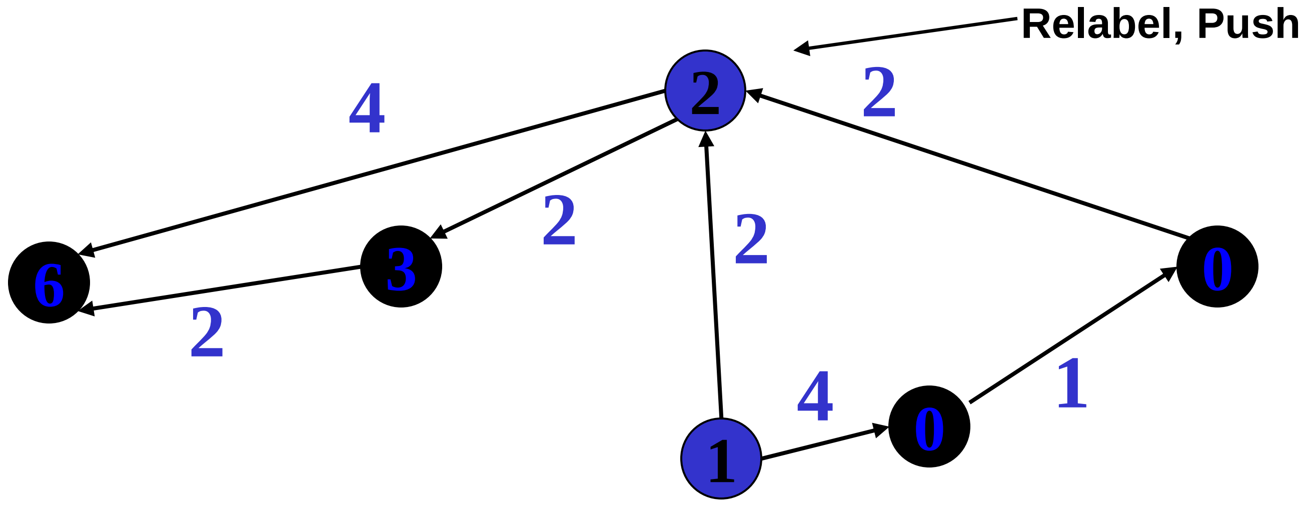
Example – contd.



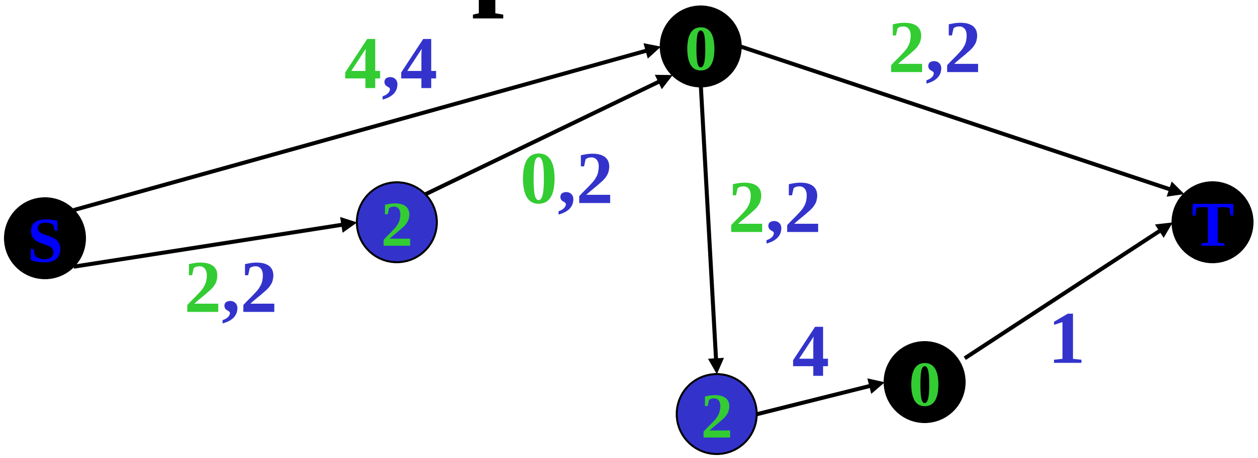


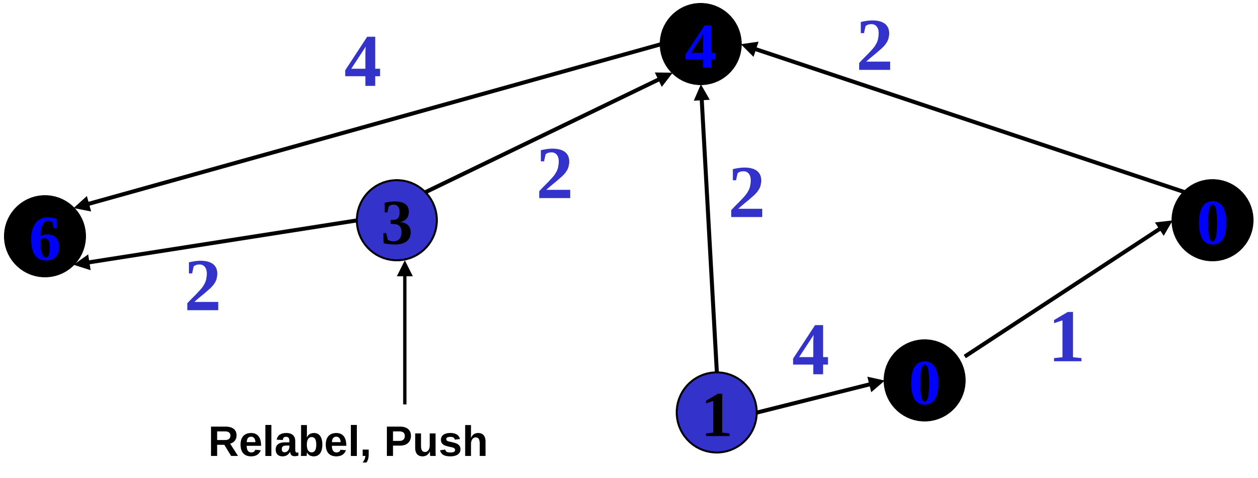
Example – contd.



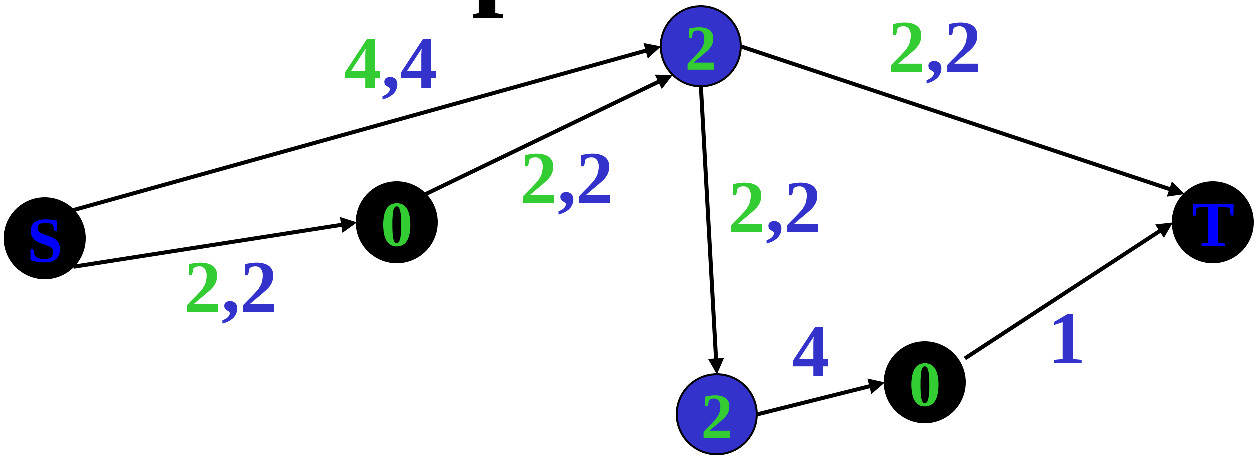


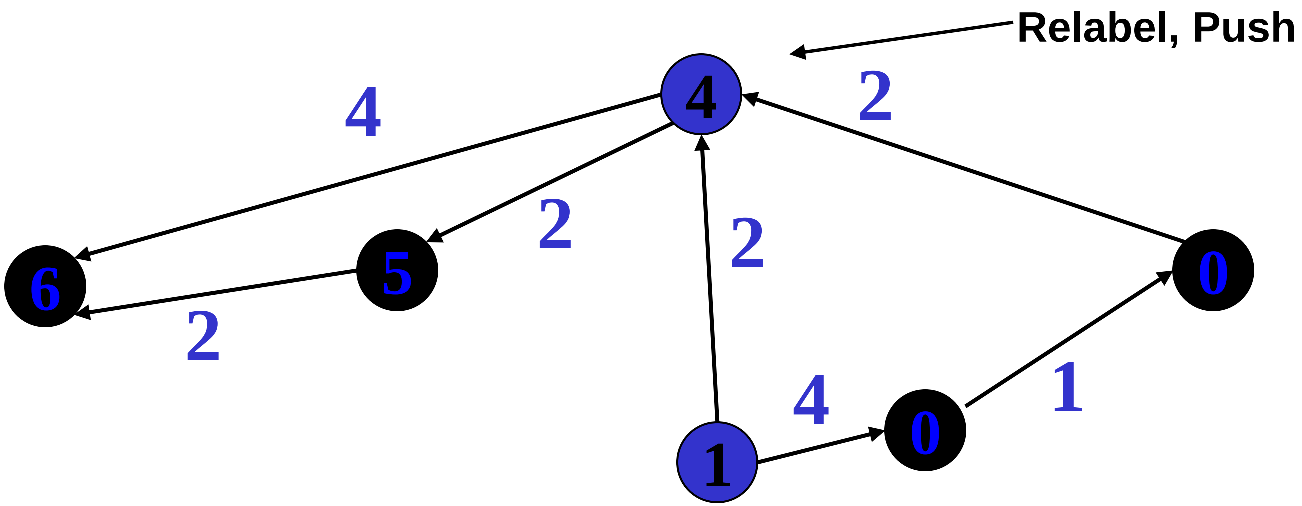
Example – contd.



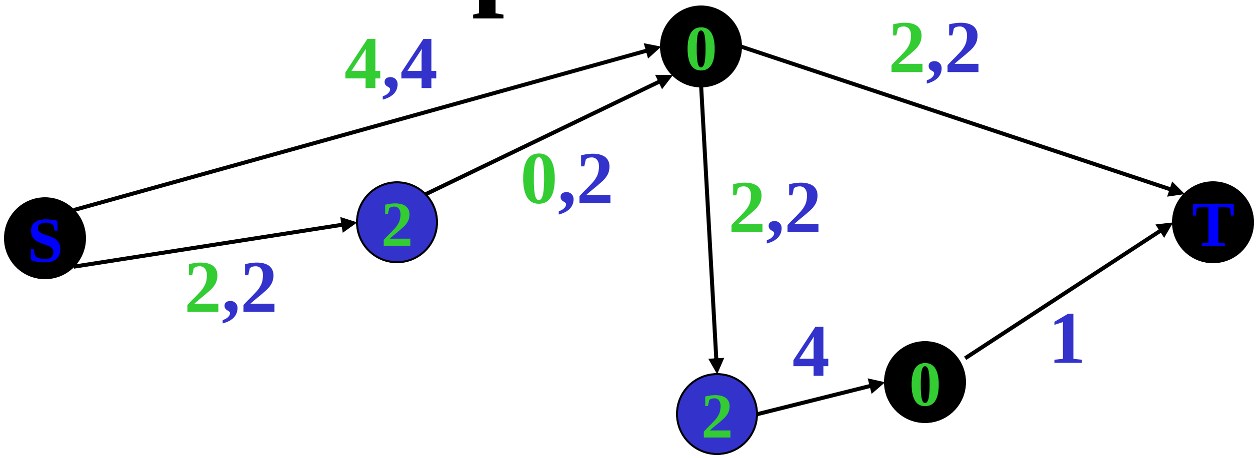


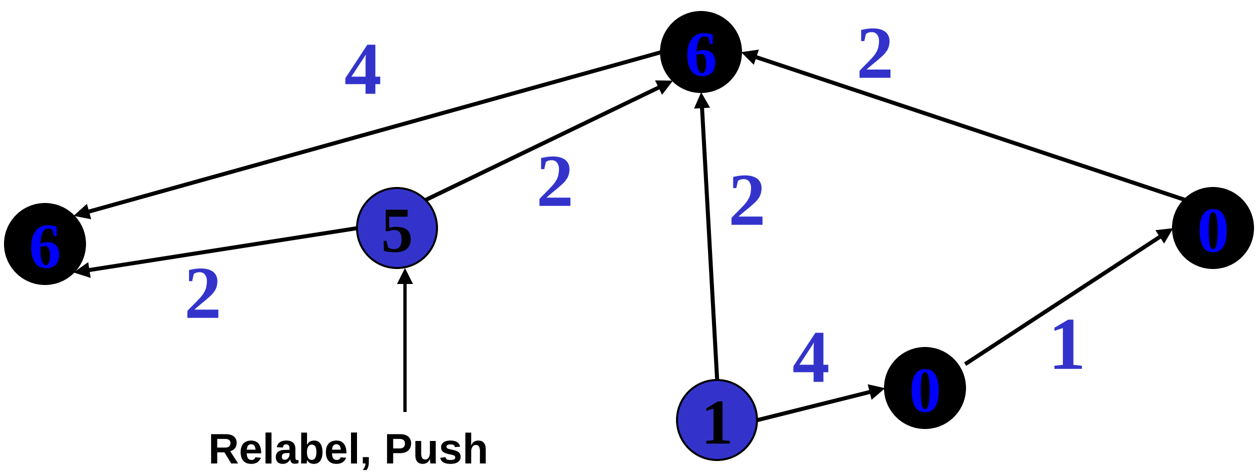
Example – contd.



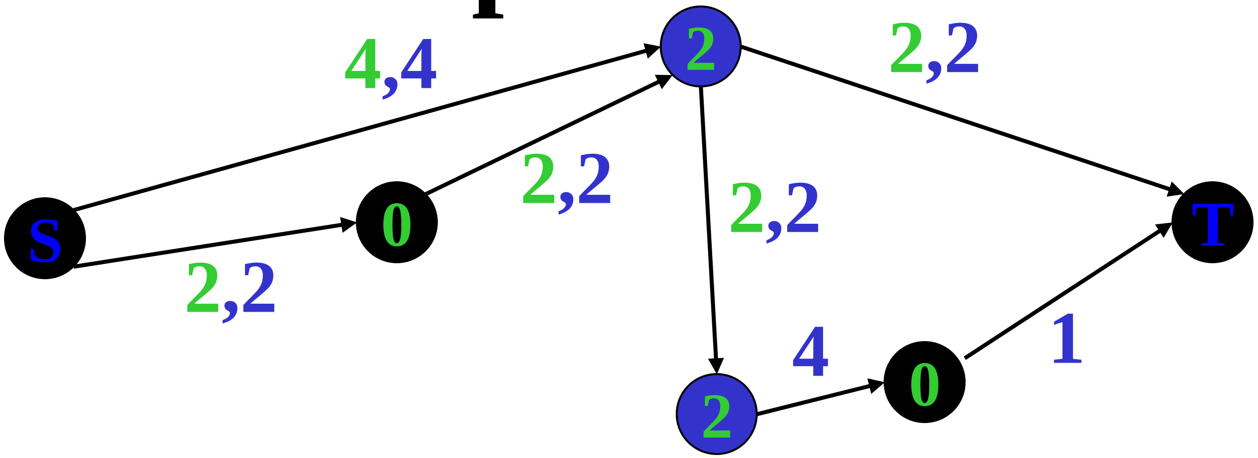


Example – contd.

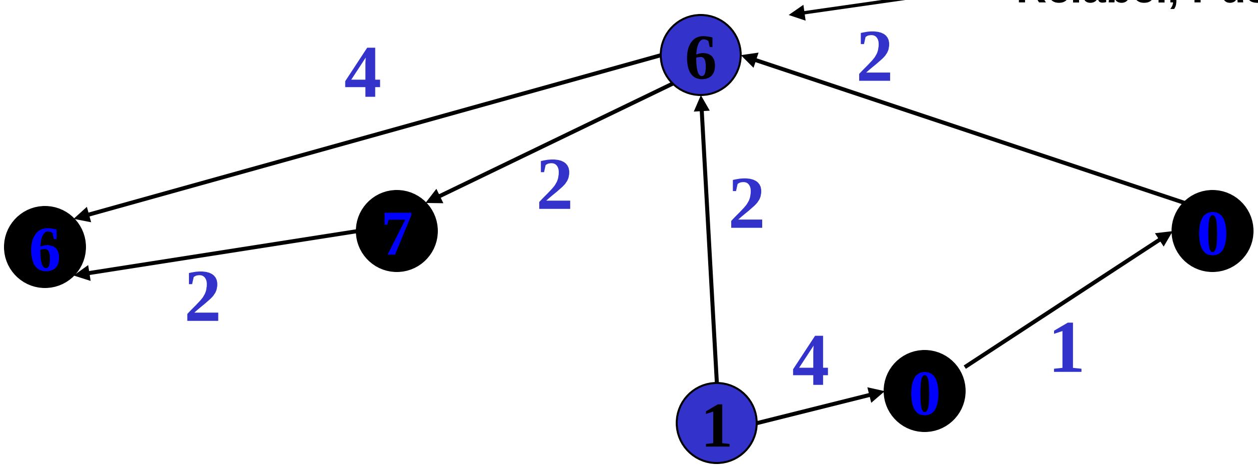




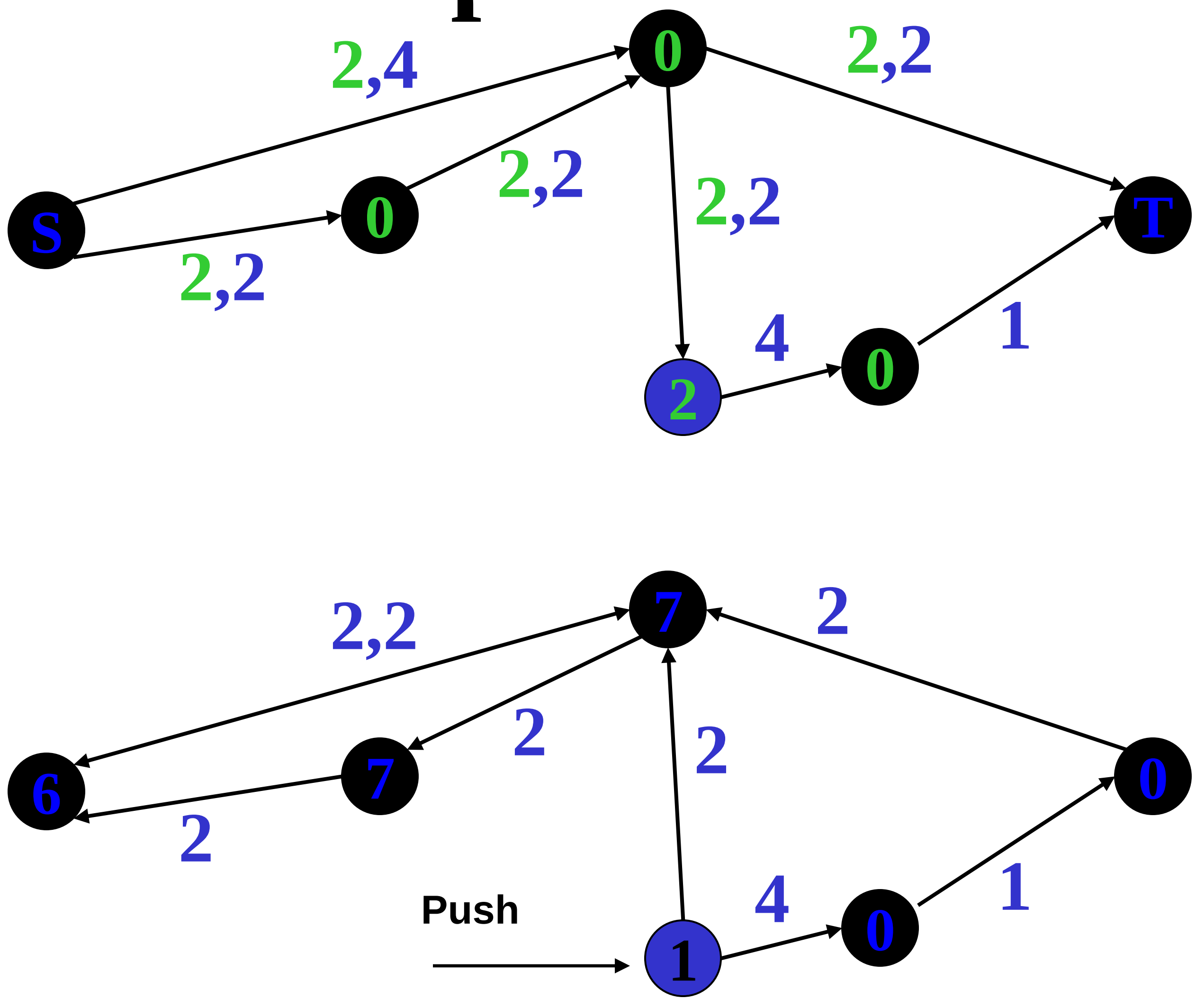
Example – contd.



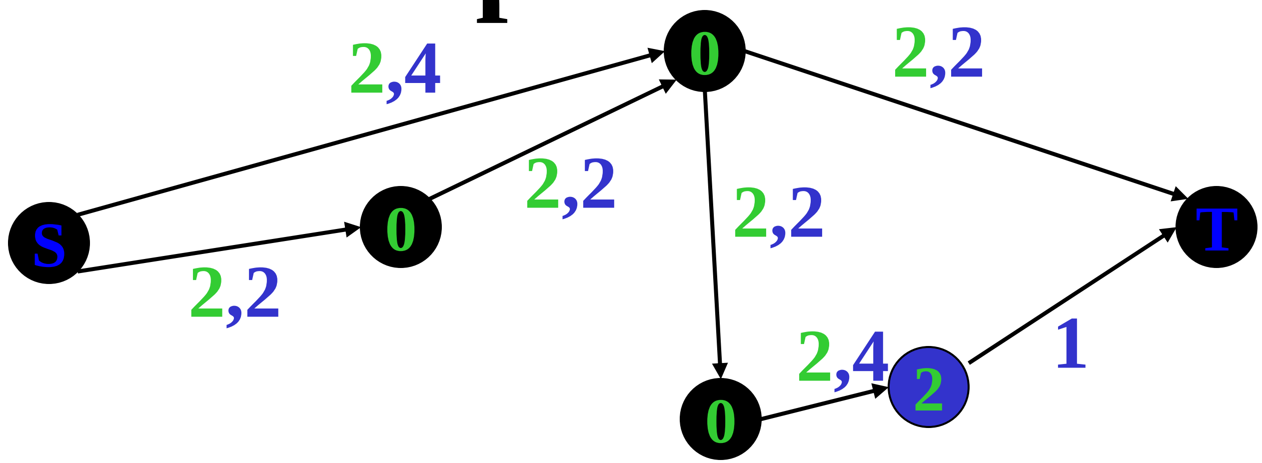
Relabel, Push

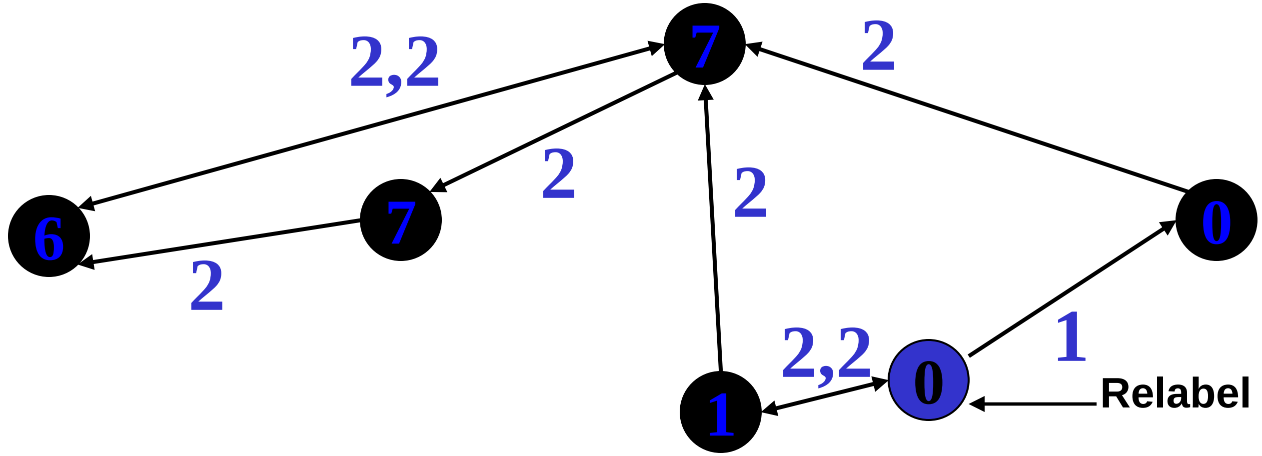


Example – contd.

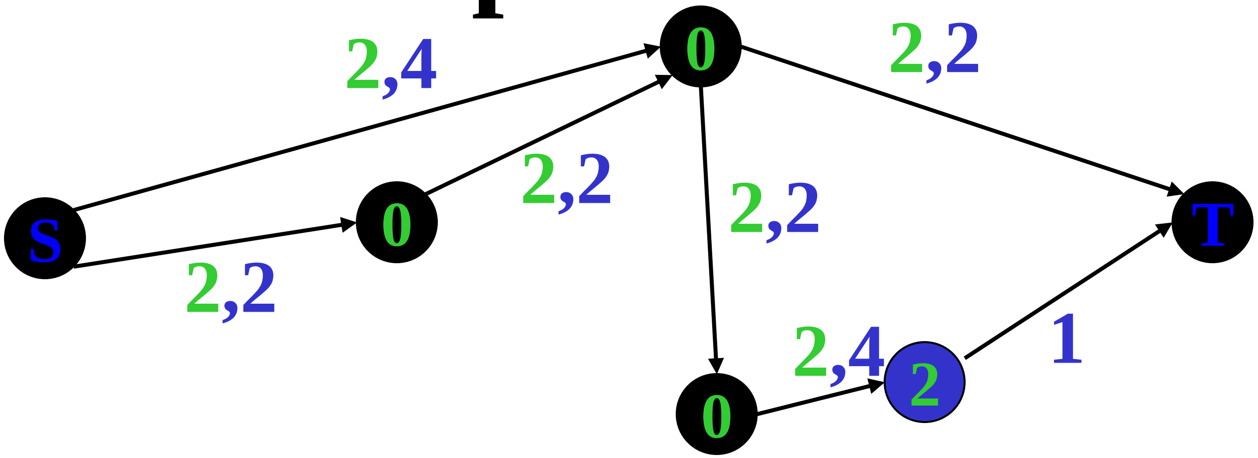


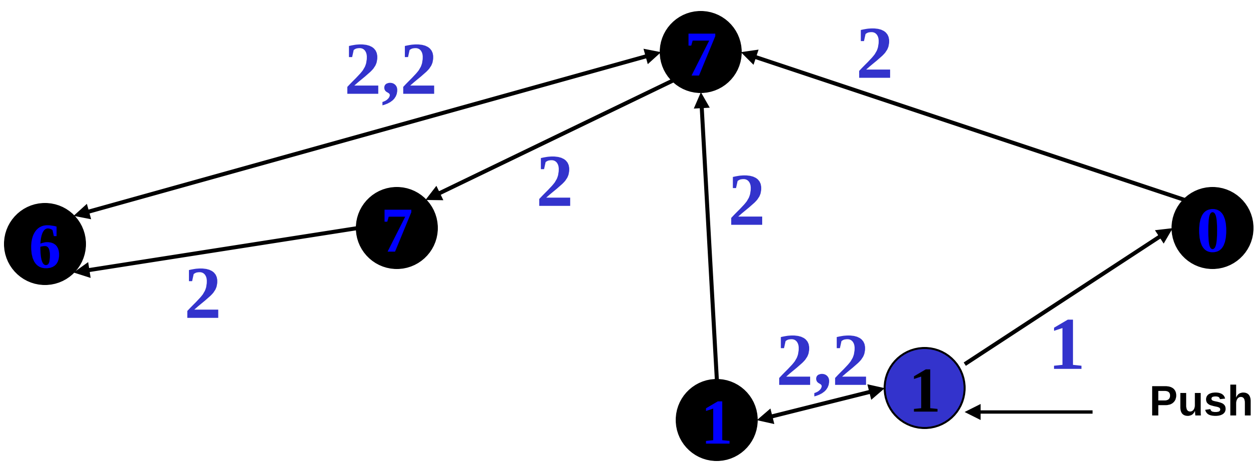
Example – contd.



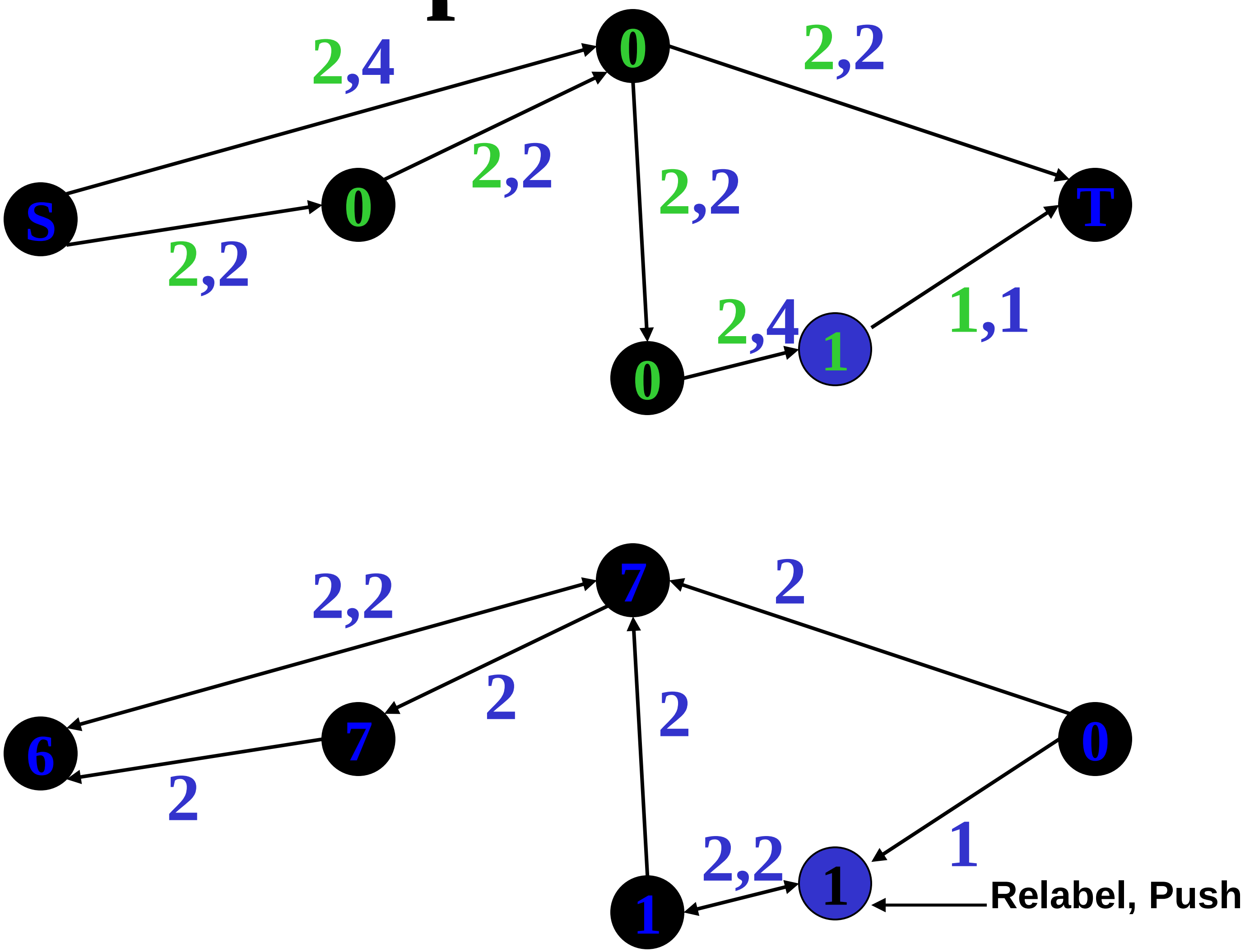


Example – contd.

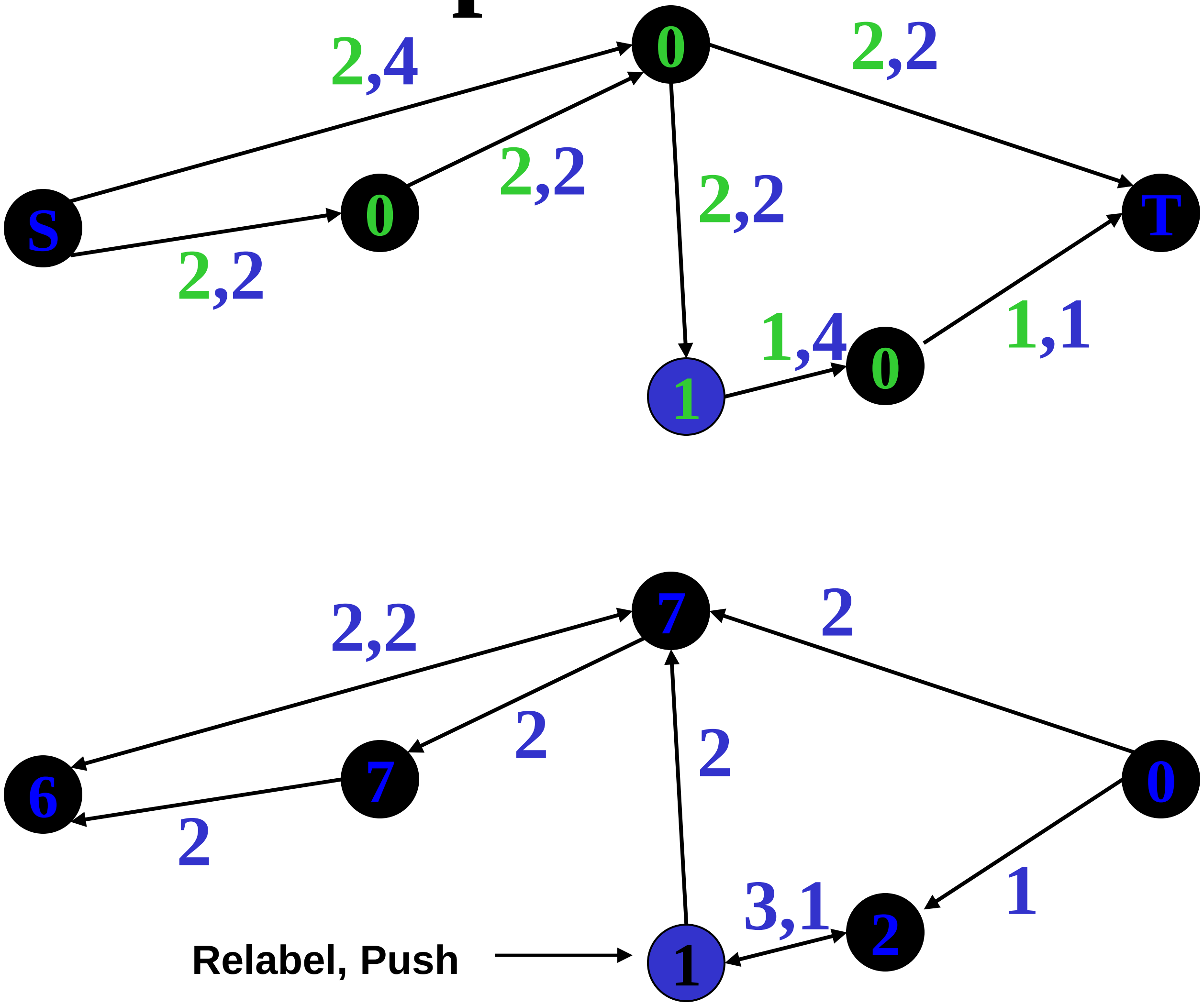




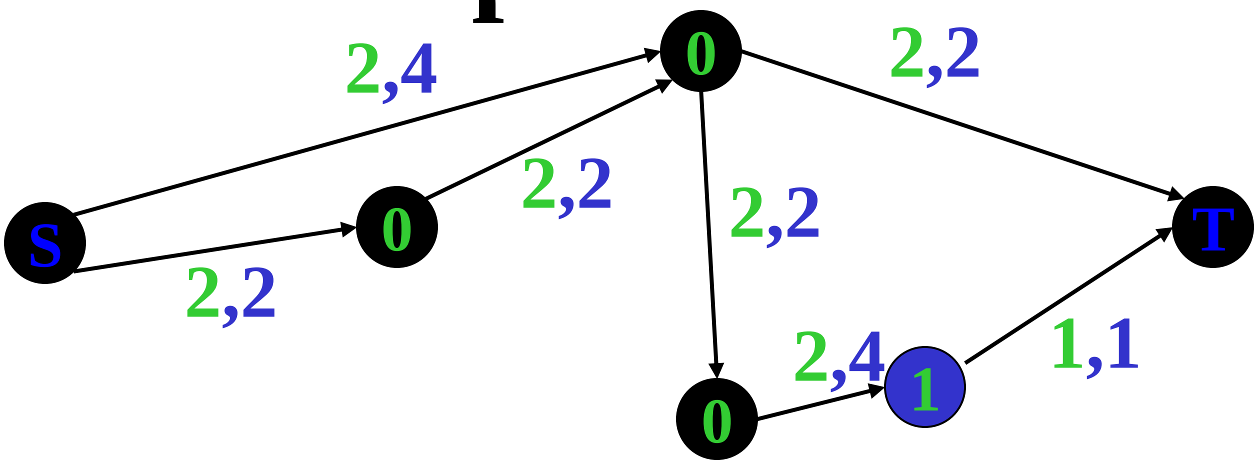
Example – contd.

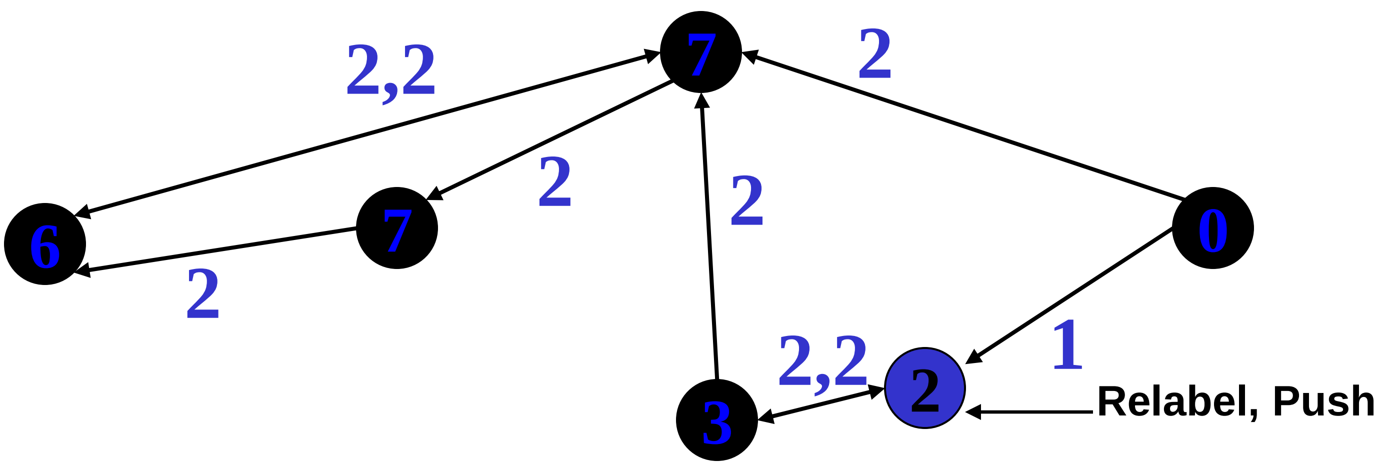


Example – contd.

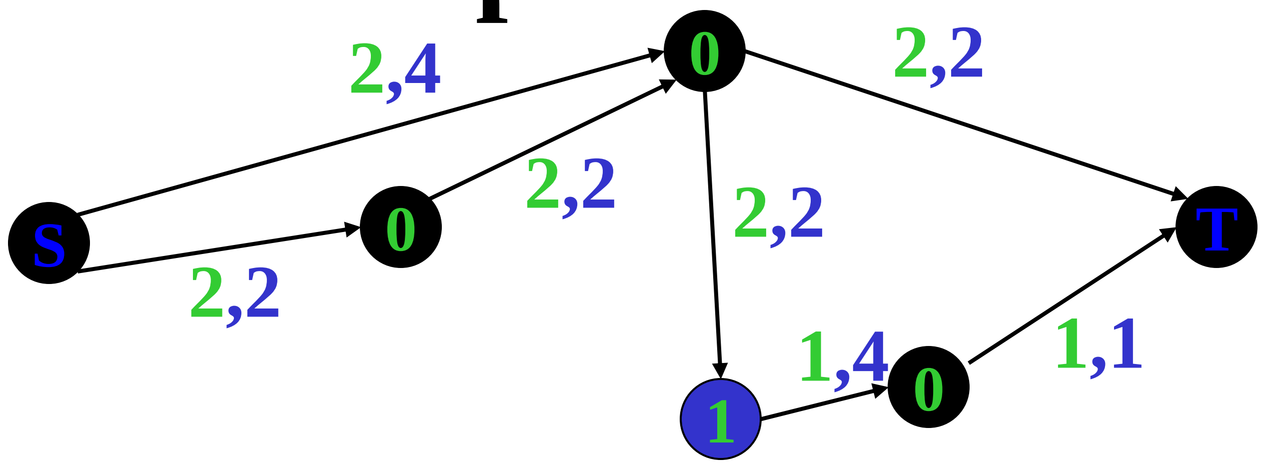


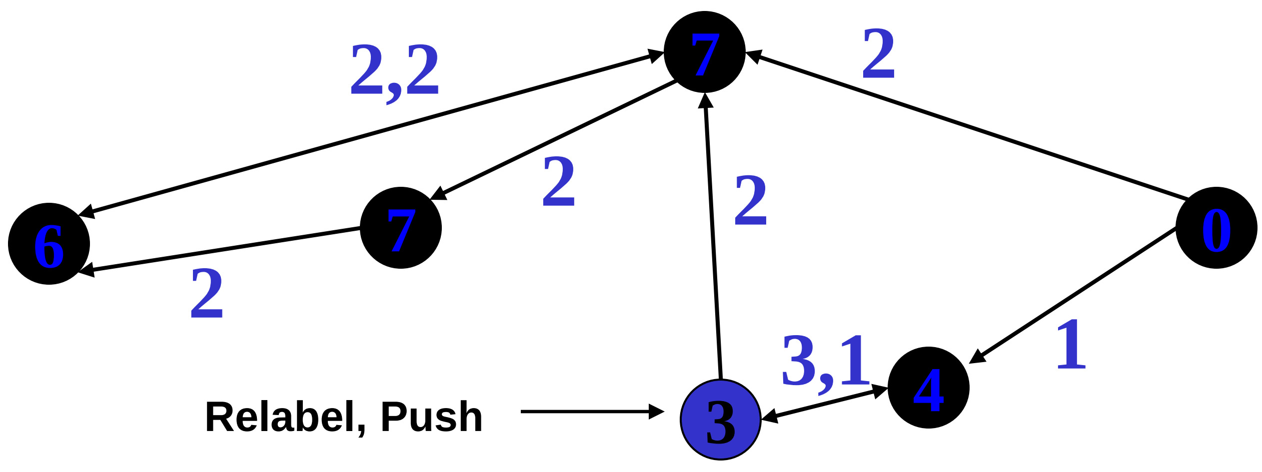
Example – contd.





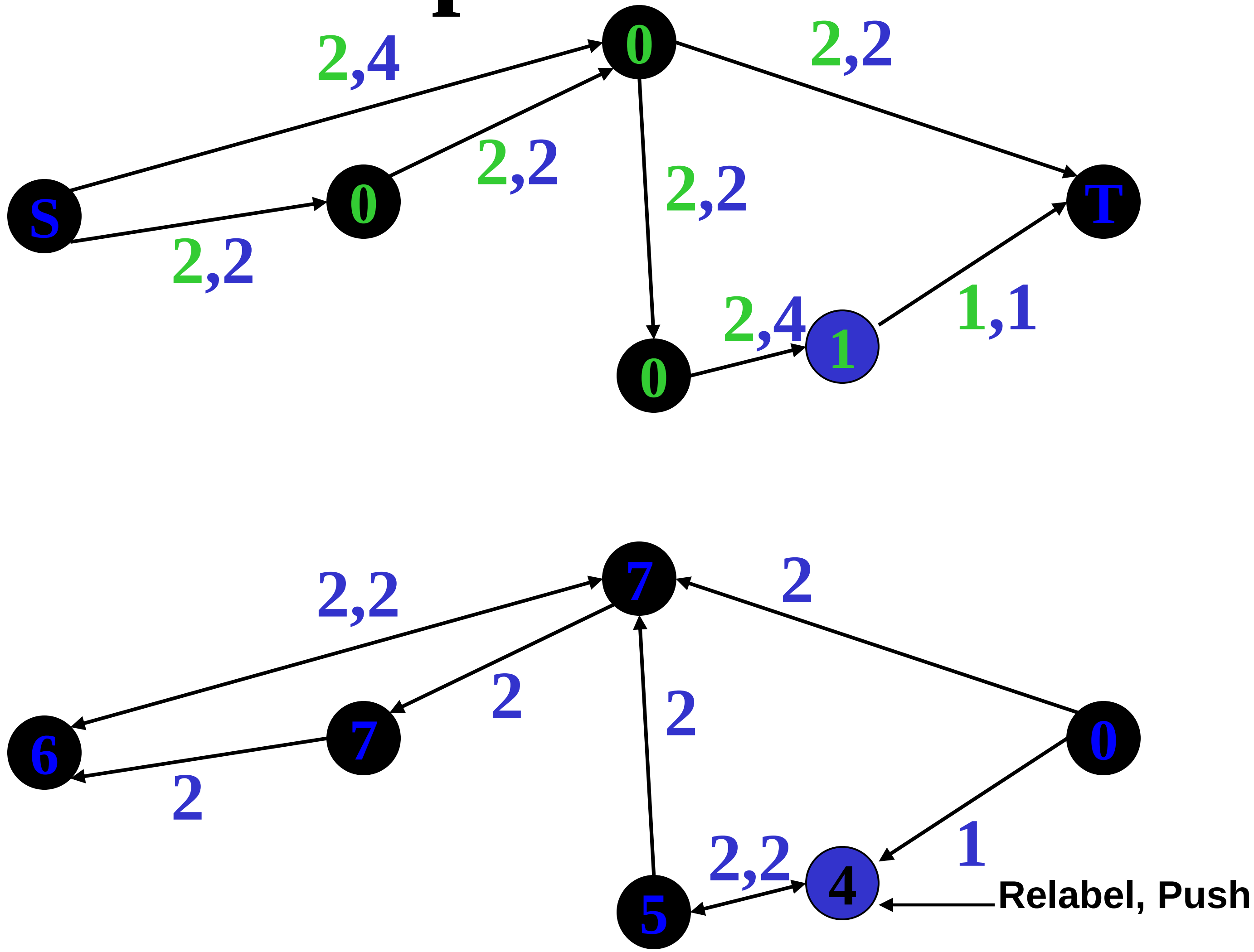
Example – contd.



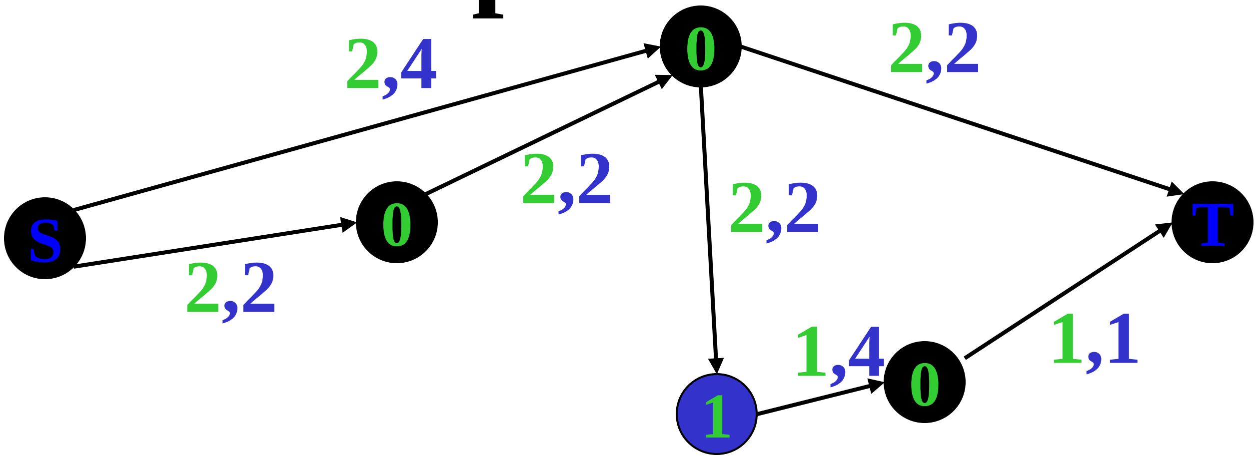


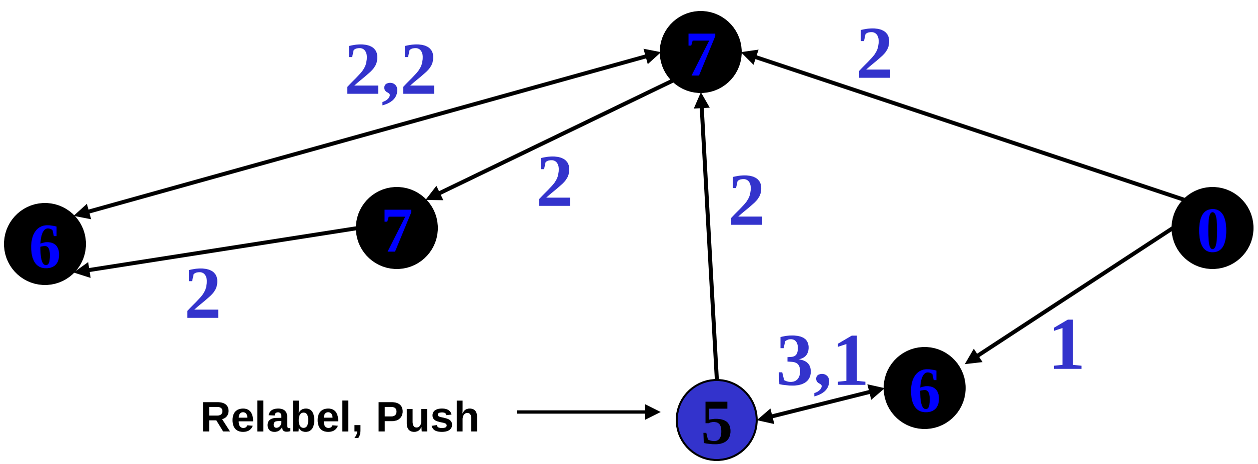
Relabel, Push

Example – contd.

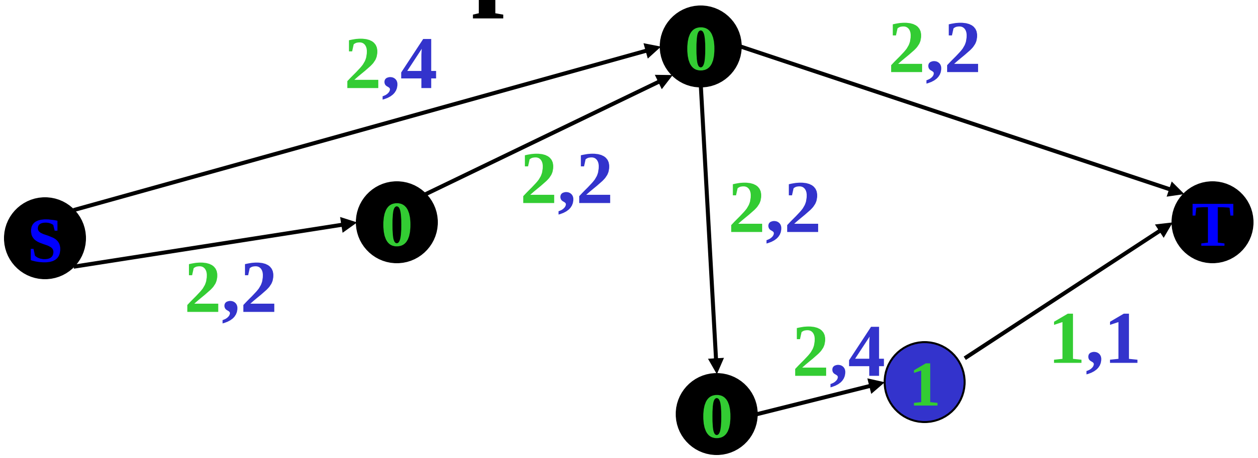


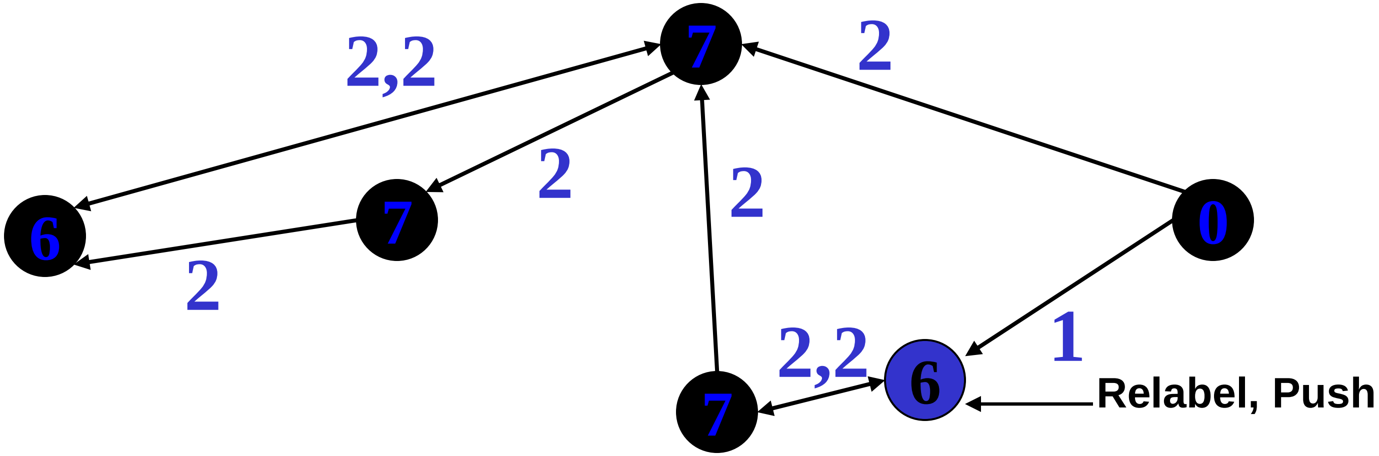
Example – contd.



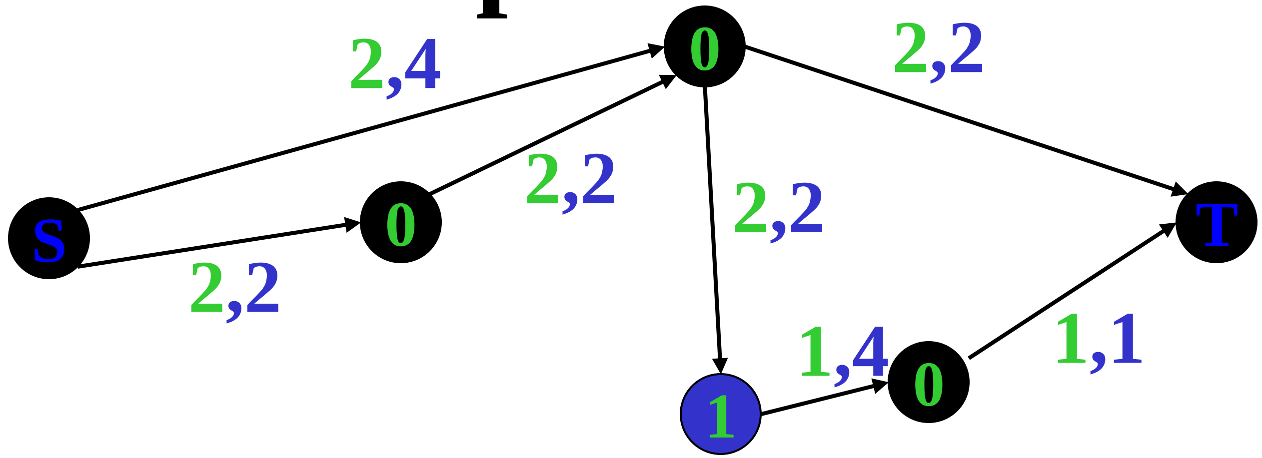


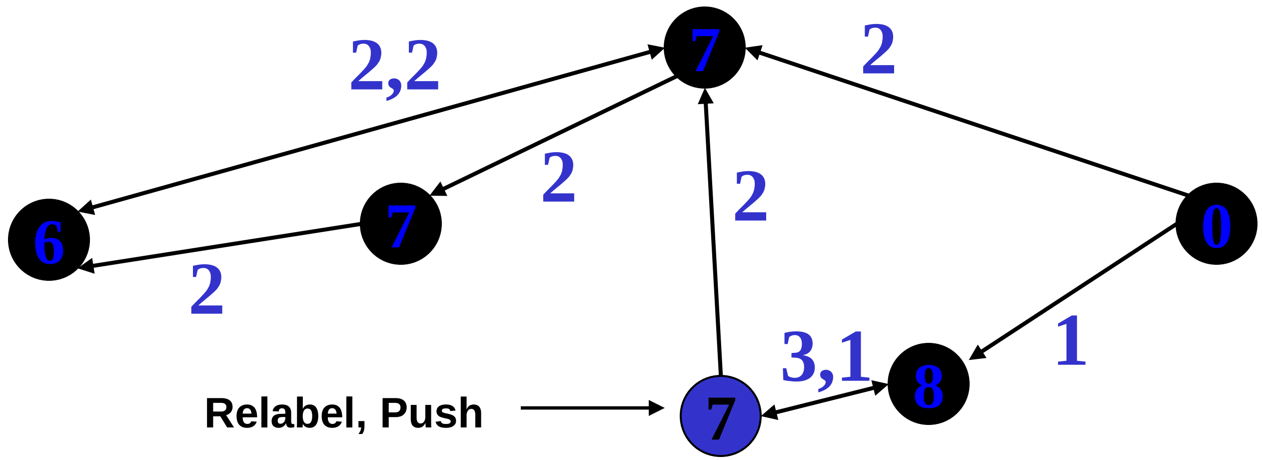
Example – contd.



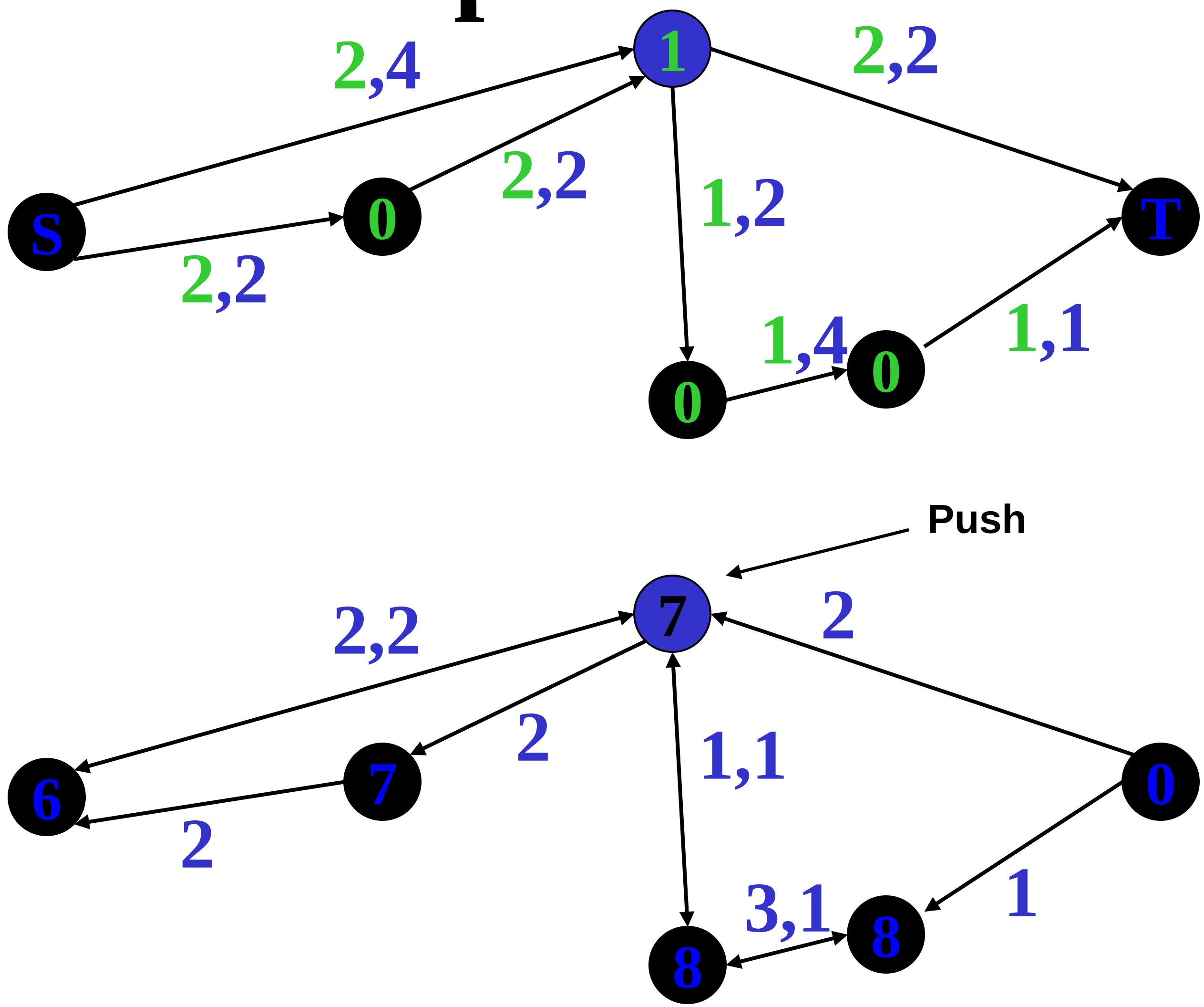


Example – contd.

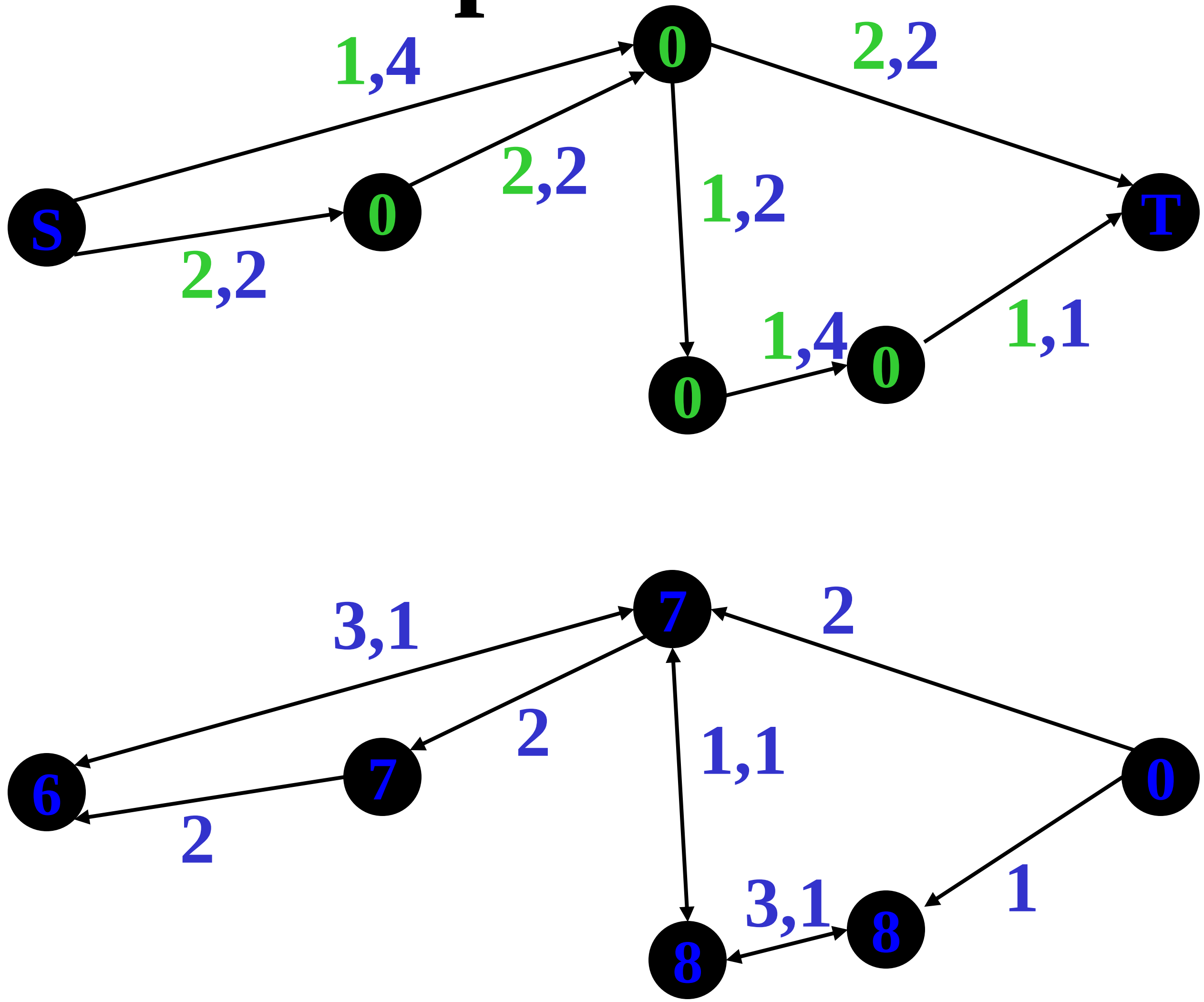




Example – contd.



Example – contd.



Correctness

- For an active vertex v, there must be a residual path v→…→s
  - Otherwise, no flow enters v, and it is clearly not active
- So, every active vertex v has an outgoing edge
  - And this means, that if the distance labels are valid, v can be either relabled or pushed

Correctness of d(v)

- $r(u,v) > 0 \rightarrow d(u) \leq d(v)+1$
- By induction on the basic operations
- We begin with a valid labeling
- Relabel keeps the invariant
  - By definition for the outgoing edges
  - Only grows, so holds for all the incoming ones
- Push
  - Can only introduce (v,u) – back edge, but since $d(u) = d(v)+1$ the correctness is kept

Correctness of d(v) – contd.

- d(v) is finite for any v during the run of the algorithm
- For any active vertex v, $d(v) < 2n$
  - Let $p = v, v_1, v_2, v_3 \ldots v_k, s$ be a path $v \rightarrow s$
  - $d(v) \le d(v_1) + 1 \le d(v_2) + 2 \ldots\ldots \le d(s) + k = n+k$
  - The length of the path is $\le n-1$, so $k \le n-1$
  - $\Rightarrow d(v) \le 2n-1$
- For a non active, it is kept when the vertex is active, or it is 0.

Correctness contd.

- At the end, for all the vertices besides {s,t} no excess is left in the vertices
  - ➔ Our preflow is a flow
- The sink is not reachable from the source in the residual graph
  - Let p= s, v_1 ,v_2 , $v_{3\dots}$ v_k ,t be a path s→t
  - Notice $k \leq n-2$
  - $n = d(s) \leq d(v_1)+1 \leq d(v_2)+2 \ldots \leq d(t) + k+1 = k+1$
  - Implies that $n \leq k+1$ in contradiction to above
  - ➔ Our preflow is maximum

Complexity analysis

- $d(v) \leq 2n-1$, and can only grow during the execution, and only by relabel operation
- n-2 vertices are relabeled
  - → At most $(n-2)(2n-1) < 2n^2 = O(n^2)$ relabels.

Complexity analysis –
Saturating push

- First saturating push $1 \leq d(u) + d(v)$
- Last saturating push $d(u) + d(v) \leq 4n - 3$
- Must grow by 2 between 2 adjutant pushes
- → 2n-1 saturating pushes on (u,v) [or (v,u)].
- → $m(2n-1) = O(nm)$ saturating pushes at all

Complexity analysis –

Non Saturating push

- $\Phi = \sum d(v) \mid v$ is active
  - Φ is 0 in the beginning and in the end
- A saturating push increases Φ by $\leq$ 2n-1
  - All saturating pushes worth $O(mn^2)$
- All relabelings increase Φ by $\leq (2n-1)(n-2)$
- Each non saturating push decreases Φ by at least 1
- There are up to $O(mn^2)$ non saturating pushes

Complexity analysis

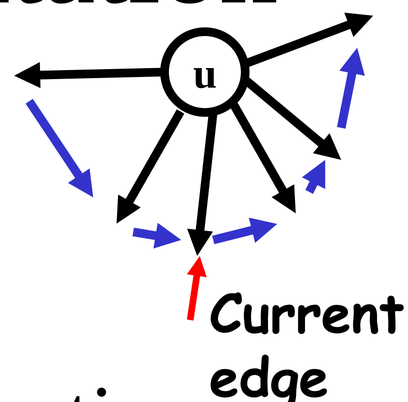
- Any reasonable sequential implementation will provide us a polynomial algorithm
  - How much a relabel operation cost?
  - How much a push operation cost?
  - How much cost to hold the active vertices?
- How will we improve this?

Implementation

- For an edge in $\{e = (u,v) \mid (u,v) \in E \text{ or } (v,u) \in E\}$ hold a struct of 3 values:
  - $c(u,v)$ & $c(v,u)$
  - $f(u,v)$
- For a vertex $v \in V$ we hold a list of all incident edges in some fixed order
  - Each edge appears in two lists.
- We also hold a “current edge” pointer for each vertex

Implementation – contd.

Admissible arc in the residual graph



d(u) = d(v) + 1

- Push/relabel operation:
  - If the current edge is admissible perform push on the current edge and return
  - If the current edge is the last one, relabel the node and set the current edge to the first one in the list
  - Otherwise, just advance the current edge to the next one in line

Is this correct?

- When we relabel a node we'll have no admissible edges:
  - Any of the other edges (u,v) wasn't admissible before and $d(u)$ can only grow
  - If it had $r(u,v) = 0$ before and now it is positive we had $d(v) = d(u) + 1$, and so $d(u) < d(v)$
- Hold a list of all active nodes – $O(1)$ extra cost per push/relabel operation

And it costs

- Number of relabelings – 2n-1 per vertex
- Each relabeling causes a pass over all the edges of the vertex – m for all the vertices
- Besides that we have o(1) per push performed (recall $O(mn^2)$ non saturating pushes).

- Total – $O(mn + mn^2) = O(n^2m)$

Use FIFO ordering

- discharge(v) = perform push/relabel(v) until e(v) = 0 or the vertex is relabled
- Hold two queues – one is the active, the other is for the next iteration
- Iteration:
  - While the active queue is not empty
    - Discharge the vertex in the front
  - Any vertex that becomes active is inserted to the other queue

Use FIFO ordering - complexity

- There are up to $2n^2$ relabels during the run - $2n^2$ iteration of the first kind.
- Each iteration will have up to 1 non saturating push per vertex
- $O(n^3)$ non saturating pushes at all
- ➔$O(n^3)$ total run time

Dynamic tree operations

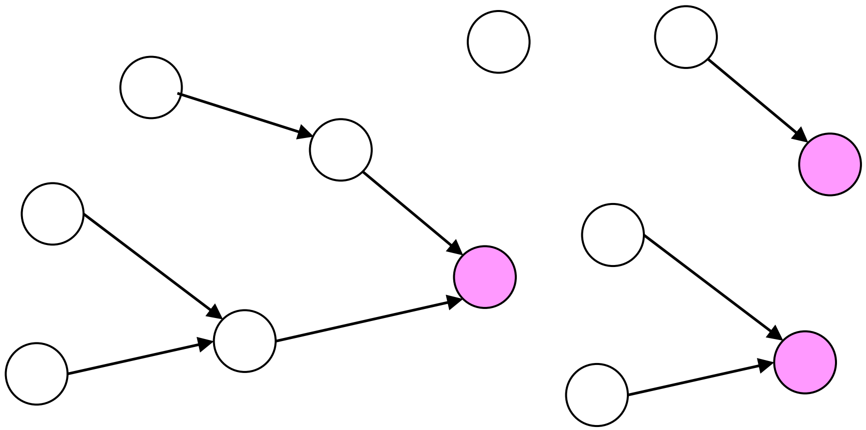
- FindRoot(v)
- FindSize(v)
- FindValue(v)
- FindMin(v)
- ChangeValue(v, delta)
- Link(v,w) – v becomes the child of w, must be a root before that.
- Cut(v) – cuts the link between v and its’ parent

The algorithm using dynamic trees

- All that said before holds, but we also add dynamic trees
- Initially every vertex is a one node dynamic tree.
- The edges (u,v) that are eligible to be in the trees are those that hold
  - d(u) = d(v)+1 (admissible)
  - r(u,v) > 0
  - (u,v) is the "current edge" of the vertex u

The algorithm using dynamic trees

- Yet, not all eligible edges are tree edges
- If an edge (u,v) is in the tree v= p(u) and value(v) = r(u,v)
- For the roots of the trees value(v) = ∞



And the algorithm is…

- As before – two queues etc.
- Discharge the vertex in the front
  - Use tree-Push/ Relabel instead of Push/ Relabel

- We'll set some constant – k – to be the upper limit of the size of a tree during the algorithm execution

Tree-Push/Relabel operation

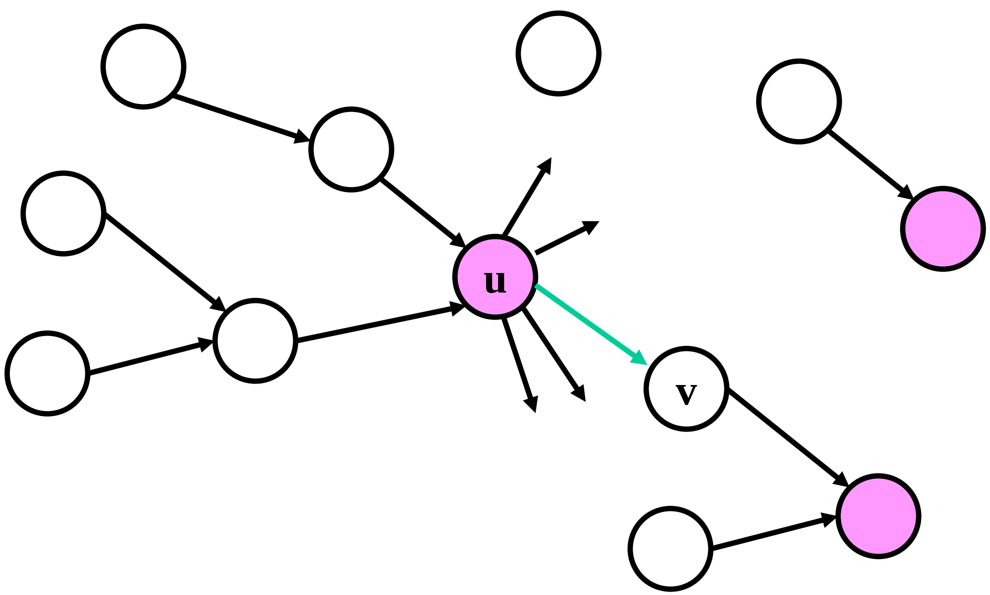
- Applied on an active vertex u
- If the current edge (u,v) is addmissible
  - If (FindSize(u) + FindSize(v) $\leq$ k)
    - Link (u,v), Send (u)
  - Else
    - Push (u,v), Send (v)
- Else
  - Advance the current edge
  - If (u,v) was the last one cut all the children of u & Relabel(u)

The Send operation

- The send operation will either cause e(v) become 0, or it will make it the root
- This implies v will not be active unless it is a root of a tree

Tree-Push/Relabel operation – contd.

- The operation insures that all vertices with positive excess are the roots of some tree



The Send operation

- Requires: u is active
- Action:
  - while FindRoot(u) != u && e(u) > 0
    - δ = min(e(u) , FindValue(FindMin(u)))
    - ChangeValue(u, -δ)
  - while FindValue(FindMin(u)) == 0
    - v = FindMin(u)
    - Cut(v)
    - ChangeValue(v, ∞)

Add the maximal possible flow on the path to the root

Remove all the edges saturated by the addition

Complexity Tree-Push/Relabel

- Each dynamic tree operation is O(log(k))
- Each Tree-Push/Relabel operation takes
  - O(1) operations
  - O(1) tree operations
  - Relabeling time
  - O(1) tree operations per cut performed

Complexity – contd.

- The total relabeling time is O(mn)
- Total number of cut operations O(mn):
  - Due to relabeling – O(mn)
  - Due to saturating push – O(mn)
- Total number of link operations < Number of cut operations + n → O(mn)

- So we reach O(mn) tree operations + O(1) tree operations per vertex entering Q.

How many times will a vertex become active?

- Due to increase of d(v) – $O(n^2)$
- Due to Send operation, e(v) grows from 0
  - Any cut performed – total (mn)
  - One more per send operation
    - Link case – O(mn)
    - Push case - Need to split to saturating and not
    - There can be up to O(mn) such saturating pushes

Non saturating push analysis

- For a non saturating push (u,v) either Tu or Tv must be large - contain more than k/2 vertices
- For a single iteration, only 1 such push is possible per vertex
- Charge it to the link or cut creating the large tree if it did not exist at the beginning of the phase – O(mn)
- Otherwise charge it to the tree itself

Non saturating push analysis – contd.

- There are up to 2n/k large trees at the beginning of the iteration

- Total of $O(n^3/k)$ for all $O(n^2)$ iterations

- ➔ A vertex enters the Queue $O(mn + n^3/k)$ times due to a non saturating push

Total complexity

- Total of $O(mn + n^3/k)$ tree operations with tree size of k.
- We reach total of $O(\log(k)\ (mn + n^3/k))$ runtime complexity
- Choose $k = n^2/m$

- We reach $O(\log(n^2/m)\ (mn))$ runtime complexity