第二章 网络中的若干优化问题

第二节 最大流问题

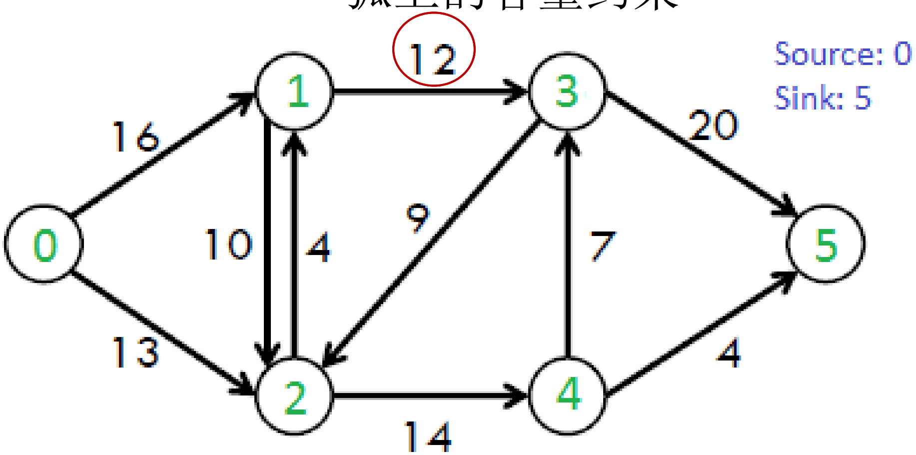
内容

- 最大流问题的定义
- 最大流问题的算法构造思路
  - 剩余网络
  - 增广路径
  - 最优性条件
- 三种最大流算法：
  - Ford-Fulkerson Algorithm
  - Capacity Scaling Algorithm
  - Preflow Push Algorithms
- 最大流问题的应用

最大流问题的定义

- 最大流问题(maximum flow problem)解决的是当网络中的弧有容量限制时，如何充分利用这些容量，使得从起点到终点的运输量最大的问题。

弧上的容量约束

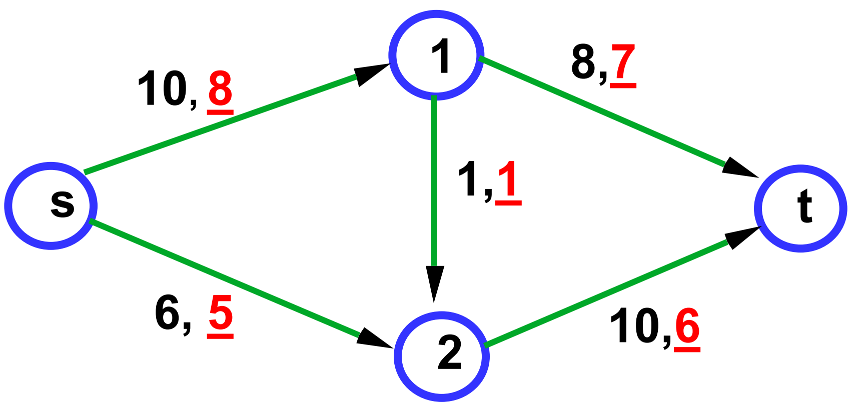


最大流问题

In a capacitated , send flows as many as possible between two special nodes, without exceeding the capacity of any arc.

- G = (N,A)
- x_{ij} = flow on arc (i,j)
- U_{ij} = capacity of flow in arc (i,j)
- s = source node
- t = destination node

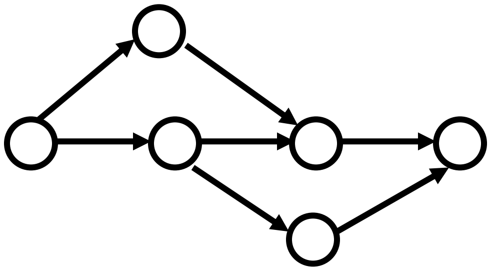
The flow value v(x) is the sum of flow sent from source node s to destination node t. We want to find the maximum flow v(x).



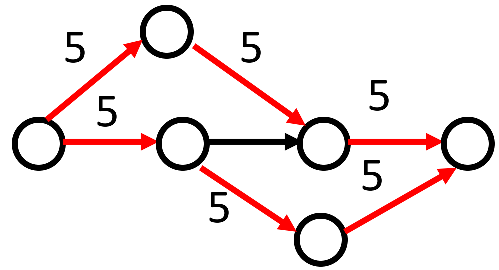
An example: Capacities and a non-optimal flow. The flow value is v(x)=8+5=13

最大流问题

每条边的容量假设相同，都为5。

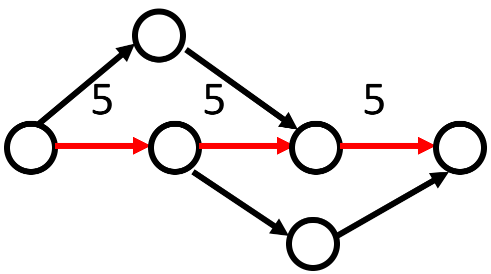


原网络

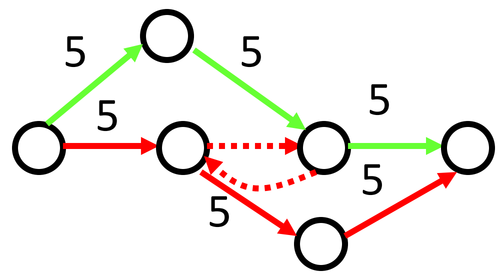


最优解

如何找到最优解？

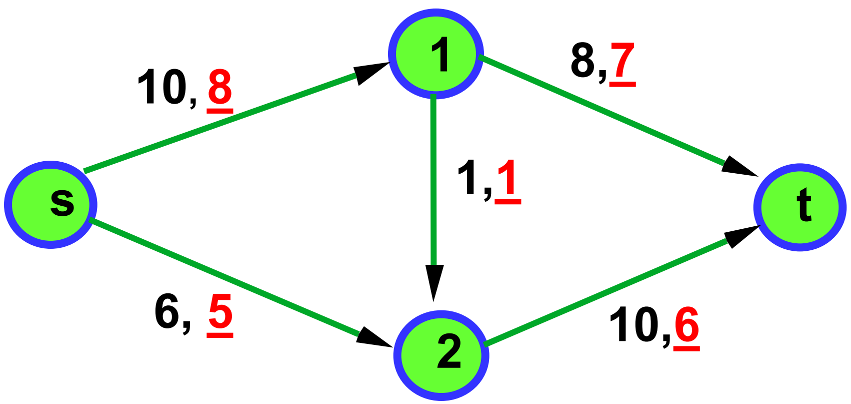


首先，找到一条路径，
输送最大流量5

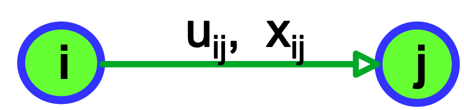


前面的路径用掉了一些关键弧上的容量，导致我们无法再趋向于最优解，因此我们需要一种流量擦除的机制。

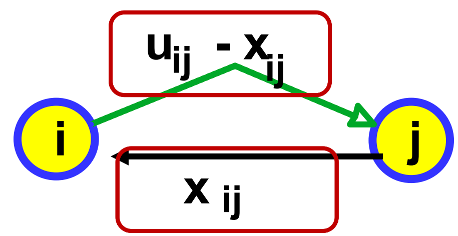
剩余网络（Residual network）



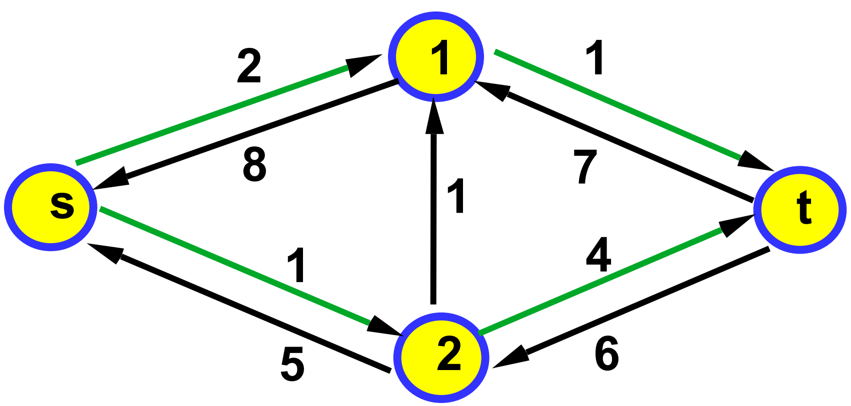
给定原网络及一个可行流x（红色数字）



原网络中边上的剩余容量



原网络中边上能够撤回的流量



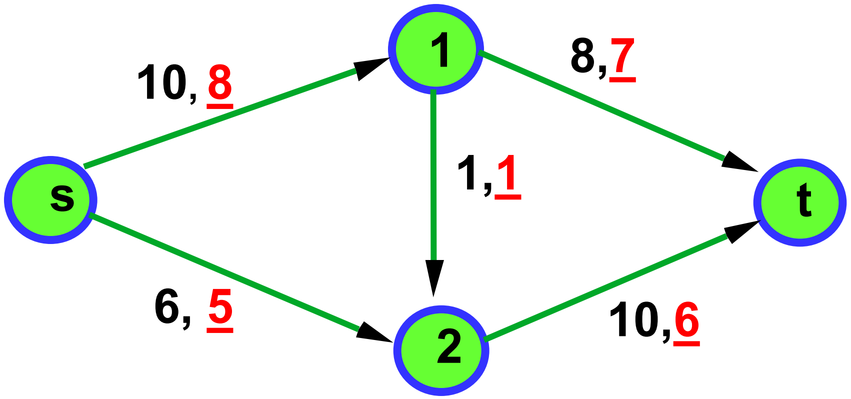
给定可行流x下原网络的剩余网络

在剩余网络中，我们为原网络中的每一条有流量的边加入一条反向的边。新生成的网络中，每条边的容量限制 r_{ij} 更新为：

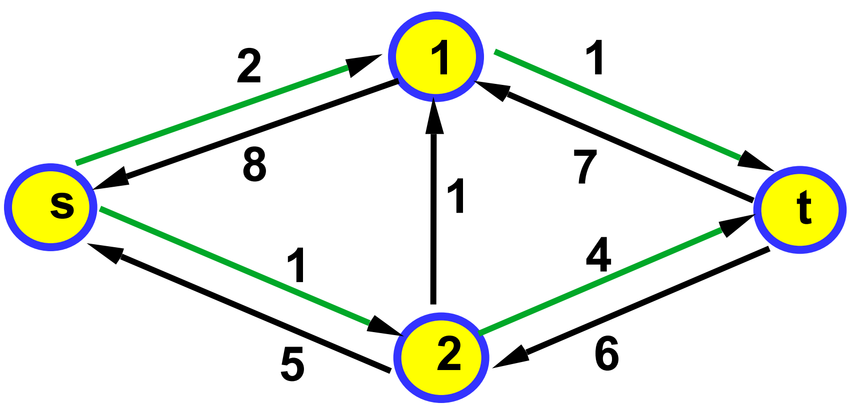
$r_{ij}=u_{ij}-x_{ij}$，(i,j)为原网络中的边

$r_{ji}=x_{ij}$，(i,j)为新加入的边

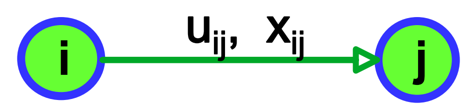
剩余网络（Residual network）



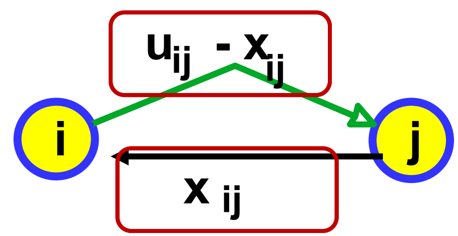
给定原网络及一个可行流x（红色数字）



给定可行流x下原网络的剩余网络



原网络中边上的剩余容量



原网络中边上能够撤回的流量

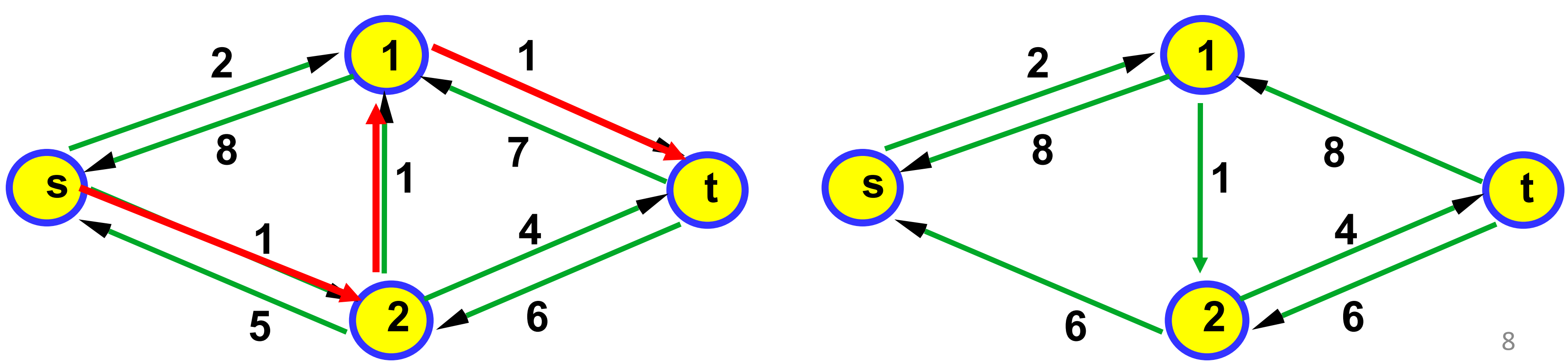
剩余网络是与当前的可行流x相关的，随着可行流的变化而变化；

剩余网络记录了在当前可行流下每条边上的剩余流量和可撤回流量。

增广路径（ Augmenting path ）

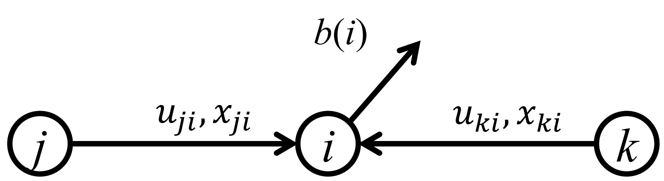
- 在剩余网络中从起点s至终点t的路径成为增广路径.
- 对于增广路径P，定义它的剩余容量：
  $\delta(P) = \min\{r_{ij} : (i,j) \in P\}$.
- 每当我们沿着增广路径运送流量δ (P) 之后，就要对剩余网络进行修改：

$$r_{ij} := r_{ij} - \delta(P) \text{ and } r_{ji} := r_{ji} + \delta(P) \quad \text{for } (i,j) \in P.$$

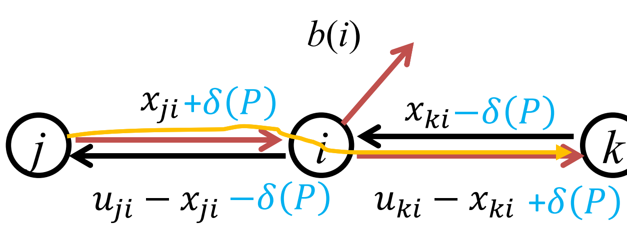


增广路径（ Augmenting path ）

- 当我们沿着增广路径运送流量时，我们实际上在对原网络上的 流量做怎样的修改？
  - 对于增广路径所通过的与原网络中方向相同的边，沿着这条边运输流量是在消耗原网络中这条边的剩余容量；



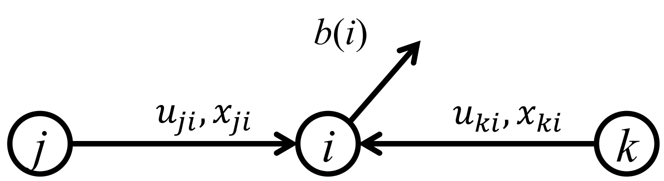
原网络



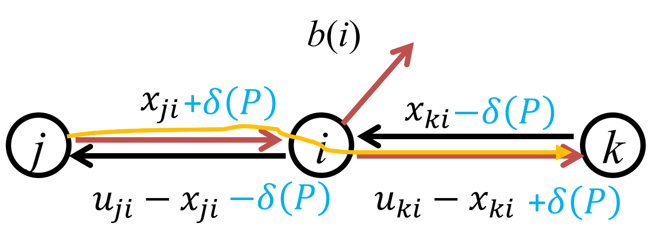
剩余网络

增广路径（ Augmenting path ）

- 当我们沿着增广路径运送流量时，我们实际上在对原网络上的 流量做怎样的修改？
  - 对于增广路径所通过的与原网络中方向相反的边，沿着这条边运输流量是在撤回原网络中对应边的已运送流量，释放容量；



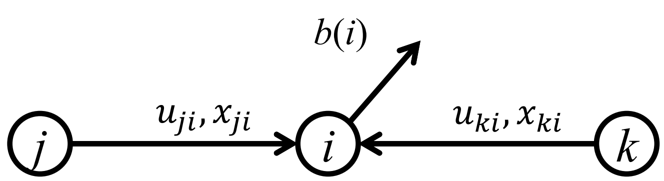
原网络



剩余网络

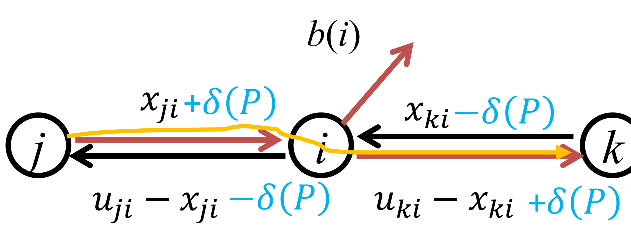
增广路径（ Augmenting path ）

- 当我们沿着增广路径运送流量时，**原网络中的流量始终保持是可行流**
  - 除了s和t点外，其余节点始终保持流入等于流出；
  - 对于s点，流出的流量增加δ(P)；
  - 对于t点，流入的流量增加δ(P)。



$x_{ji}+x_{ki}=b(i)$

原网络

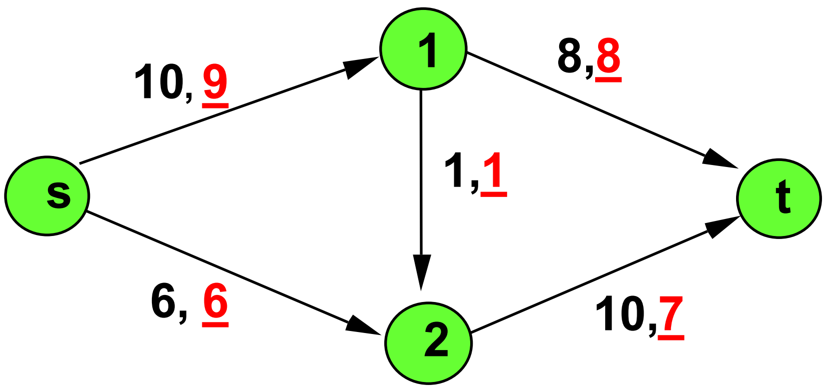


$[x_{ji}+\delta(P)]+[x_{ki}-\delta(P)]=x_{ji}+x_{ki}=b(i)$

剩余网络

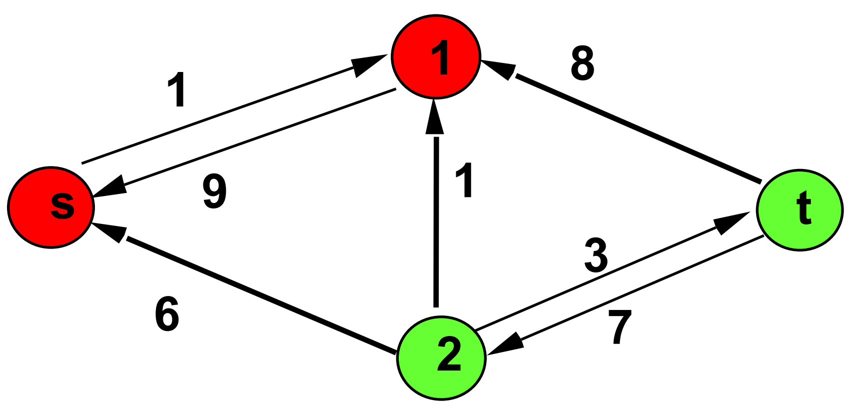
最大流问题的最优化条件

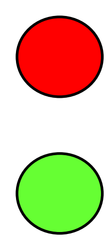
对于一个可行流 x，当且仅当其对应的剩余网络$G(x)$中不存在增广路径时，x为最大流问题的解.



←若可行流 x是最大流，则$G(x)$中不存在增广路径；
→若$G(x)$中不存在增广路径，则可行流x是最大流。

其对应的剩余网络中不存在增广路径：





从起点s可达的点

从起点s不可达的点

Ford-Fulkerson 算法

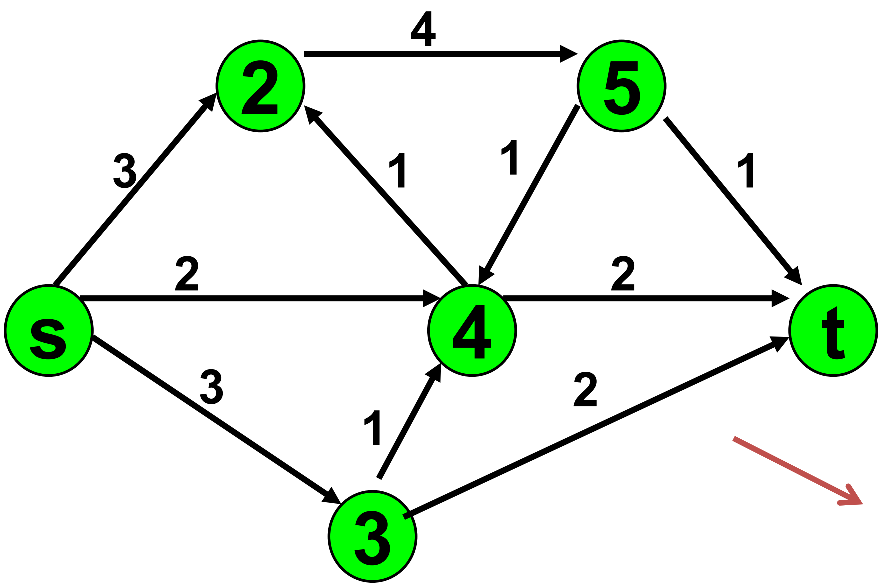
- Ford-Fulkerson算法的基本思想：
  - 沿着增广路径运送流量，在每一次流量运送后更新剩余网络
  - 循环上述过程直到更新后的剩余网络不存在增广路径

The Generic Algorithm: The Ford Fulkerson Maximum Flow Algorithm

- Begin
  - x := 0;
  - create the residual network G(x);
  - while there is some directed path from s to t in G(x) do
  - begin
    - let P be a path from s to t in G(x);
    - Δ := δ(P);
    - send Δ units of flow along P;
    - update the r's;
  - end
- end {the flow x is now maximum}.

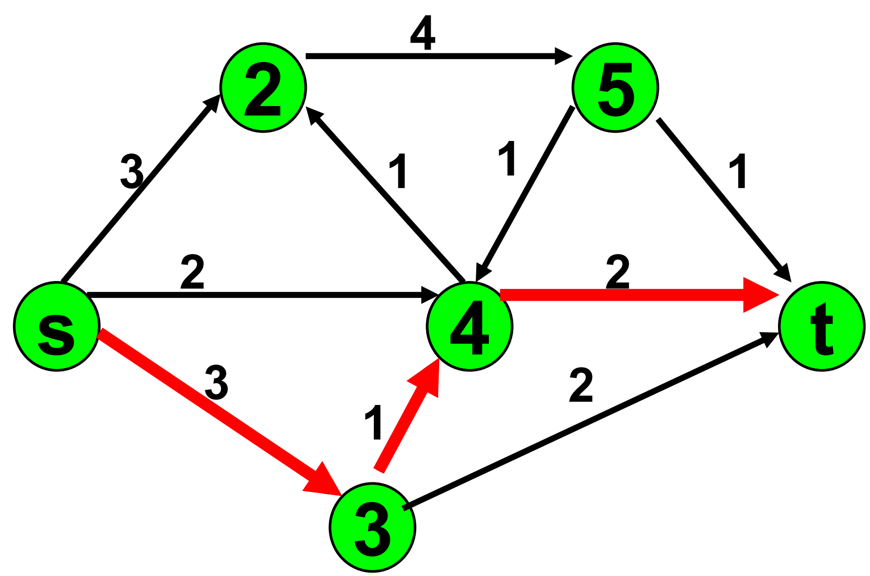
How to find this?

Ford-Fulkerson Algorithm

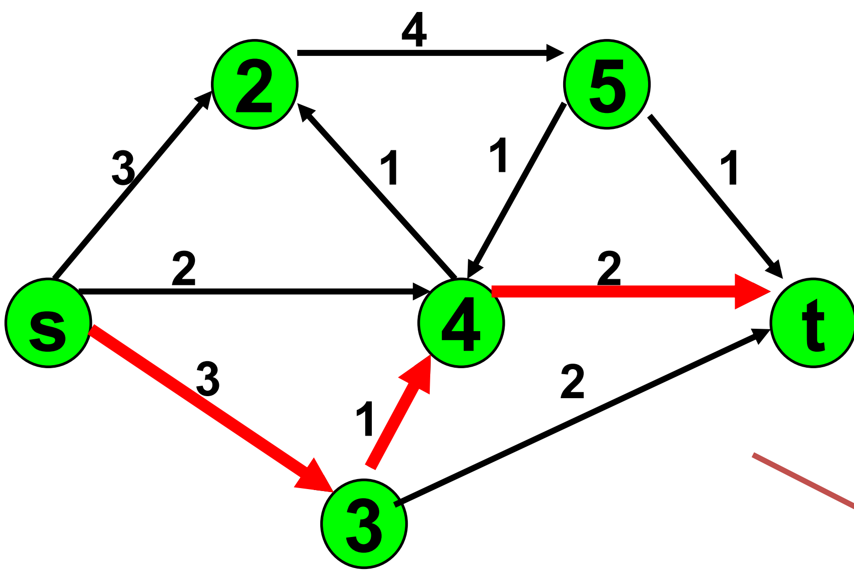


Initial network

Find any s-t path in G(x)

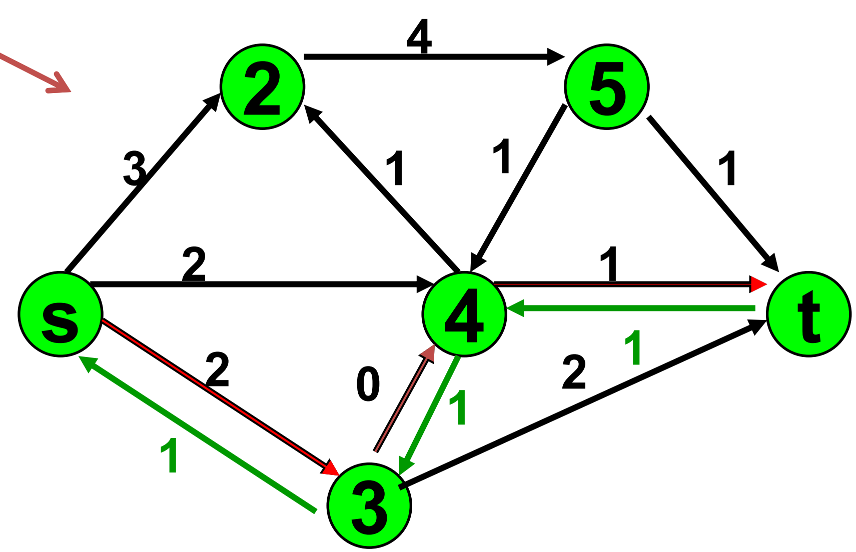


Ford-Fulkerson Algorithm

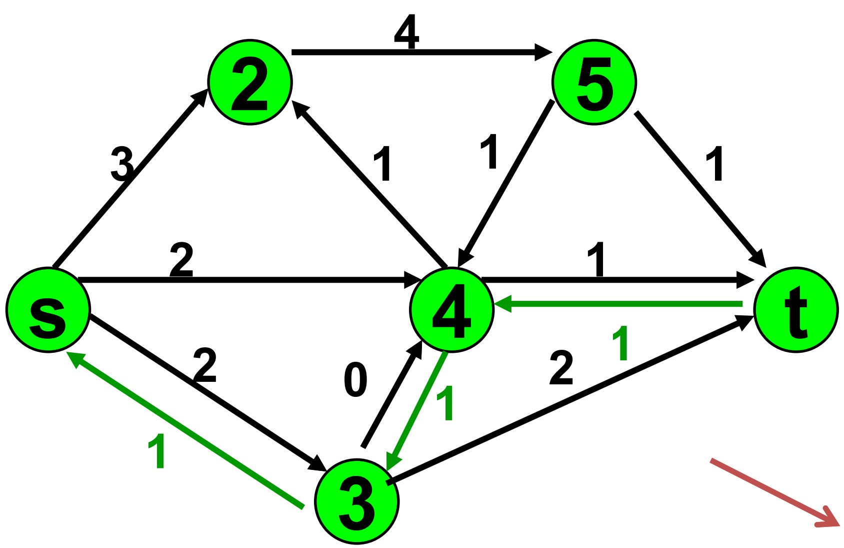


$\delta(P)=1$

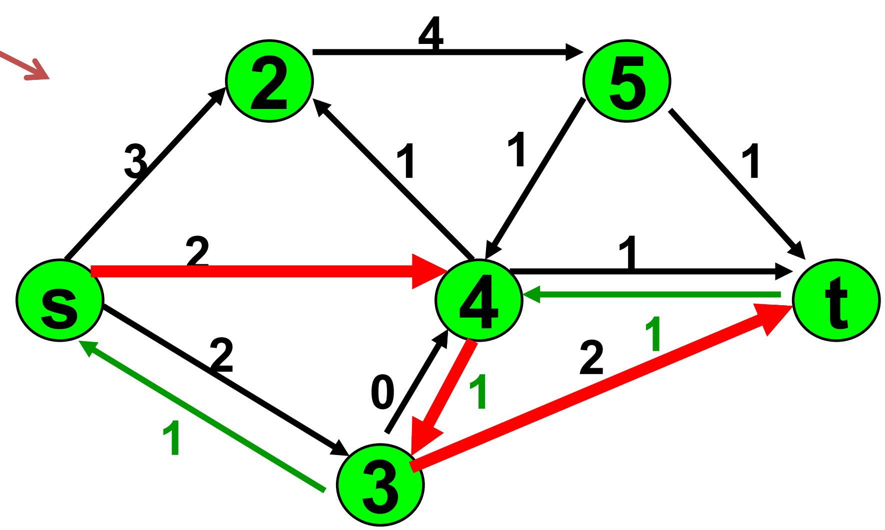
Send $\Delta=1$ unit of flow in the path.
Update residual network



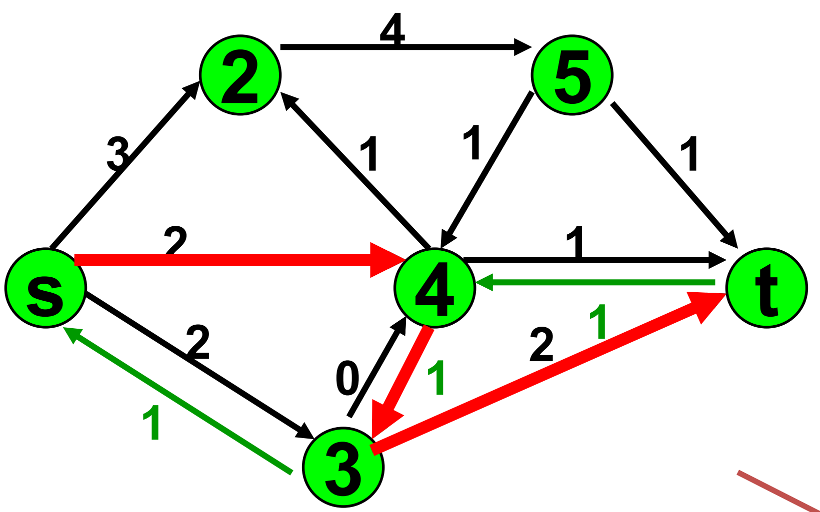
Ford-Fulkerson Max Flow



Find any s-t path

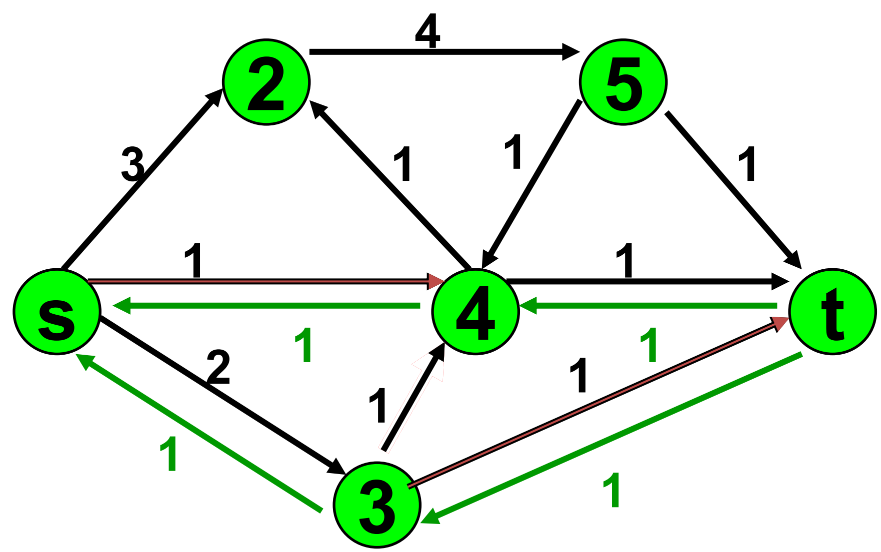


Ford-Fulkerson Max Flow

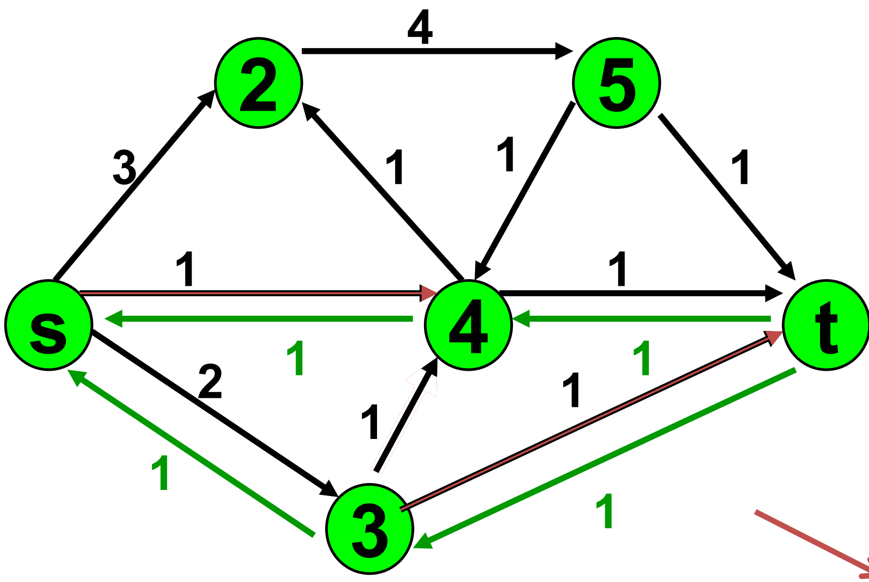


Send $\Delta=1$ unit of flow in the path.
Update residual network

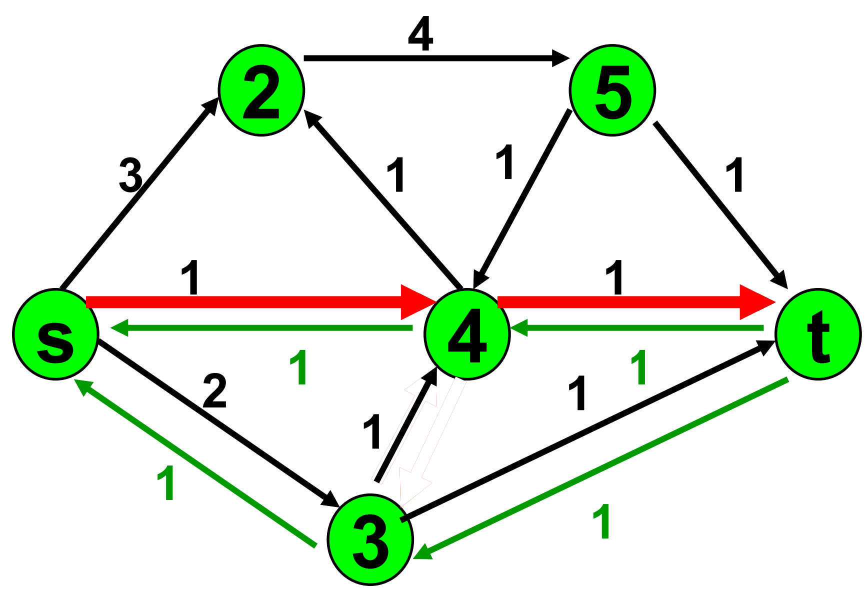
δ(P)=1



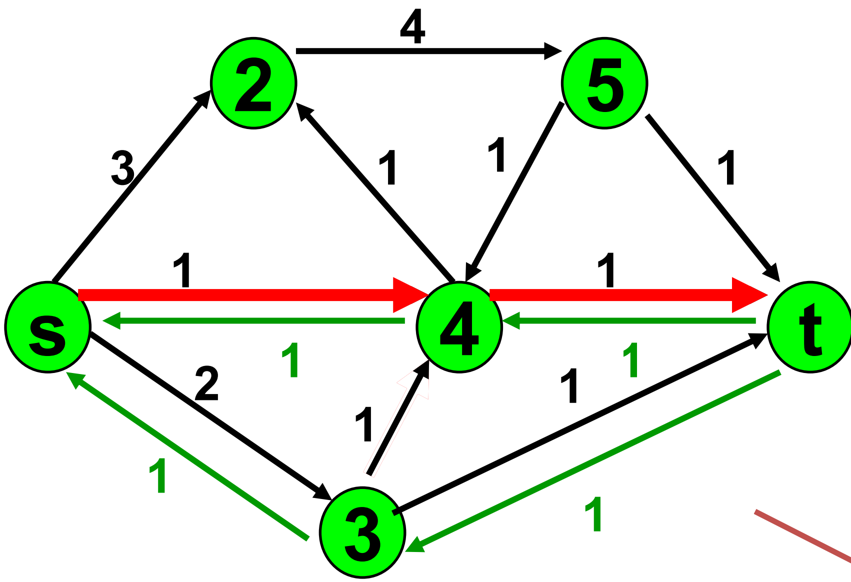
Ford-Fulkerson Max Flow



Find any s-t path

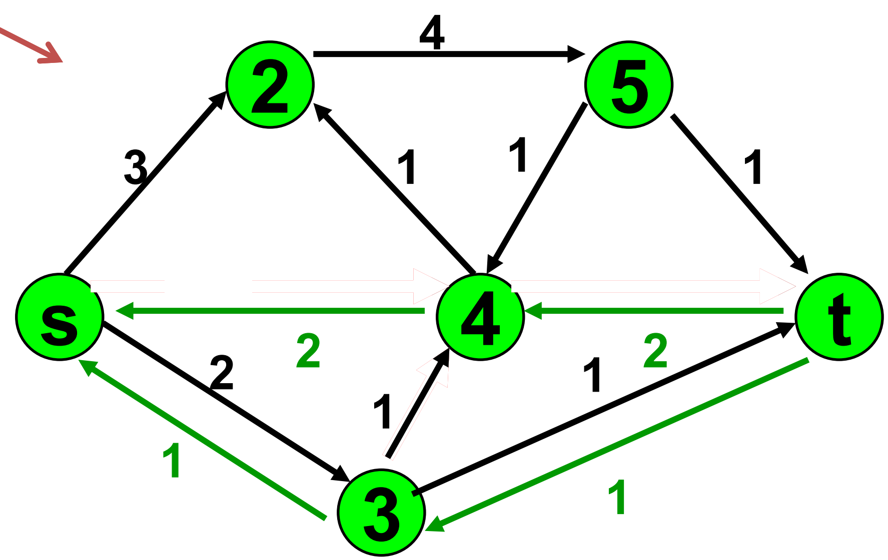


Ford-Fulkerson Max Flow

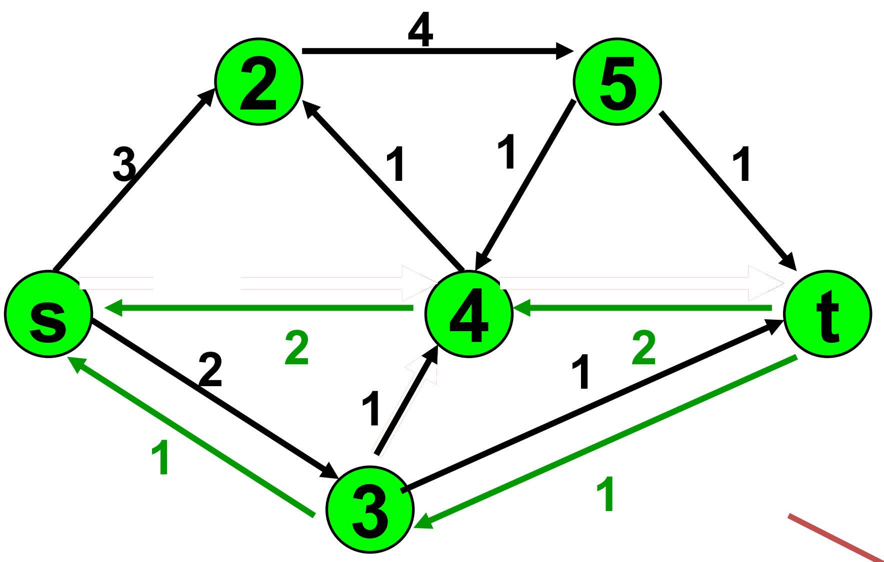


δ(P)=1

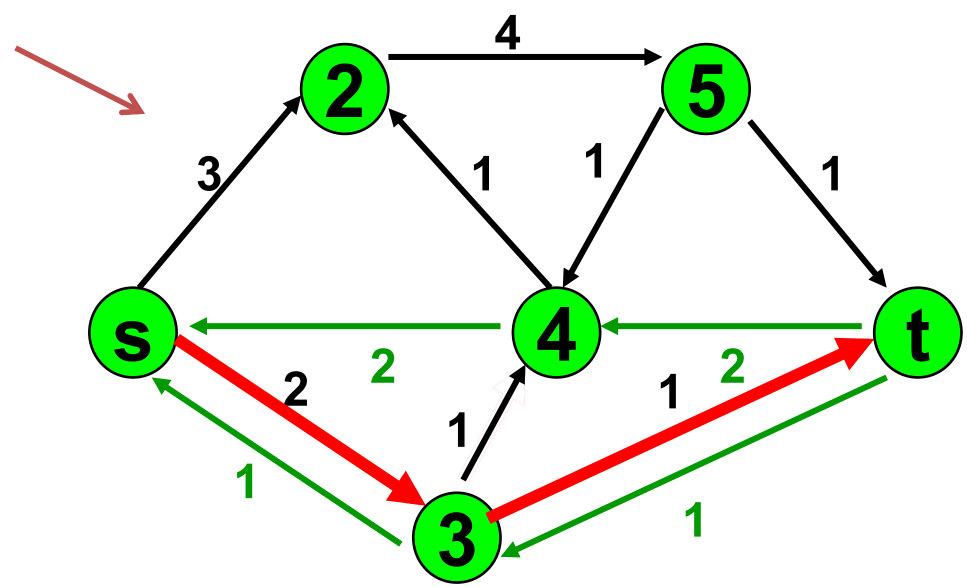
Send $\Delta=1$ unit of flow in the path.
Update residual network



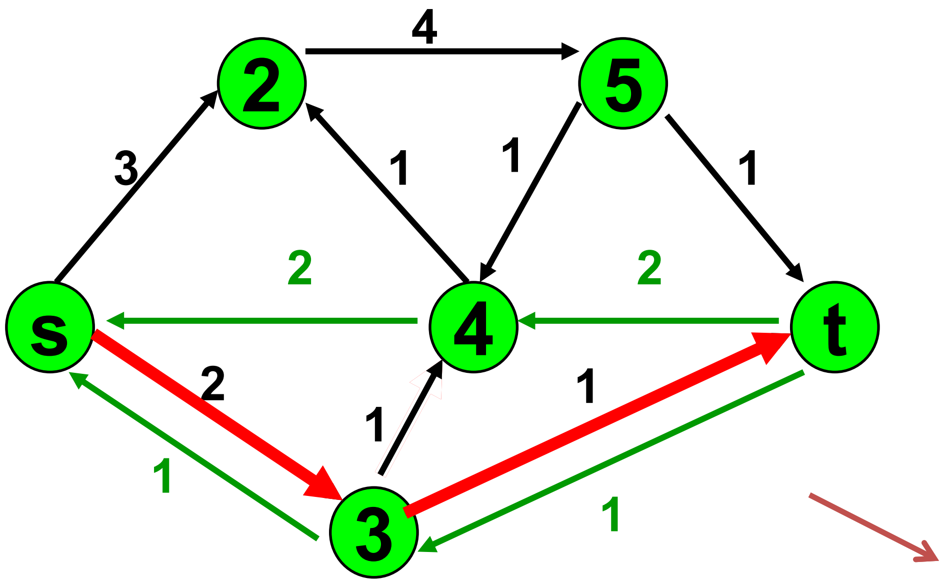
Ford-Fulkerson Max Flow

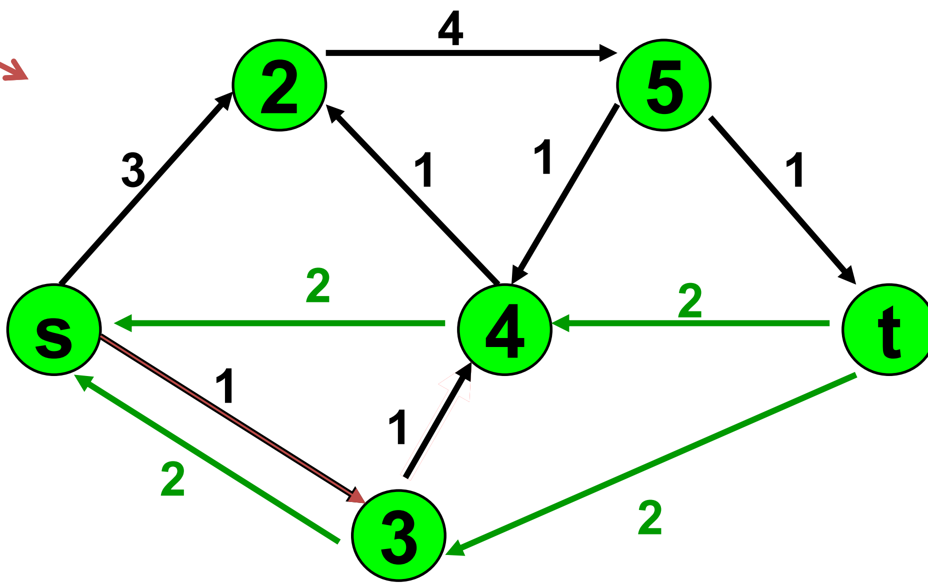


Find any s-t path



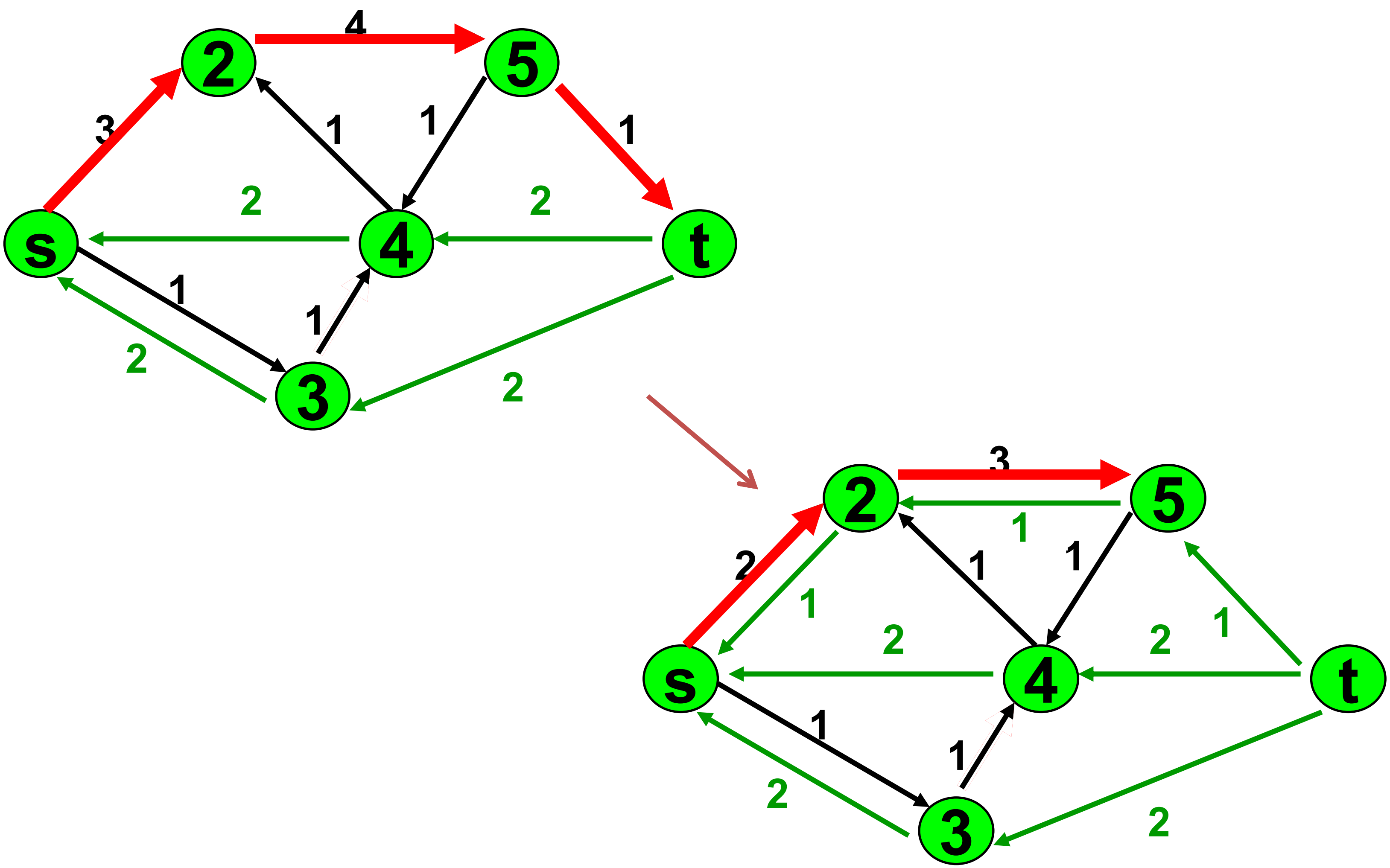
Ford-Fulkerson Max Flow



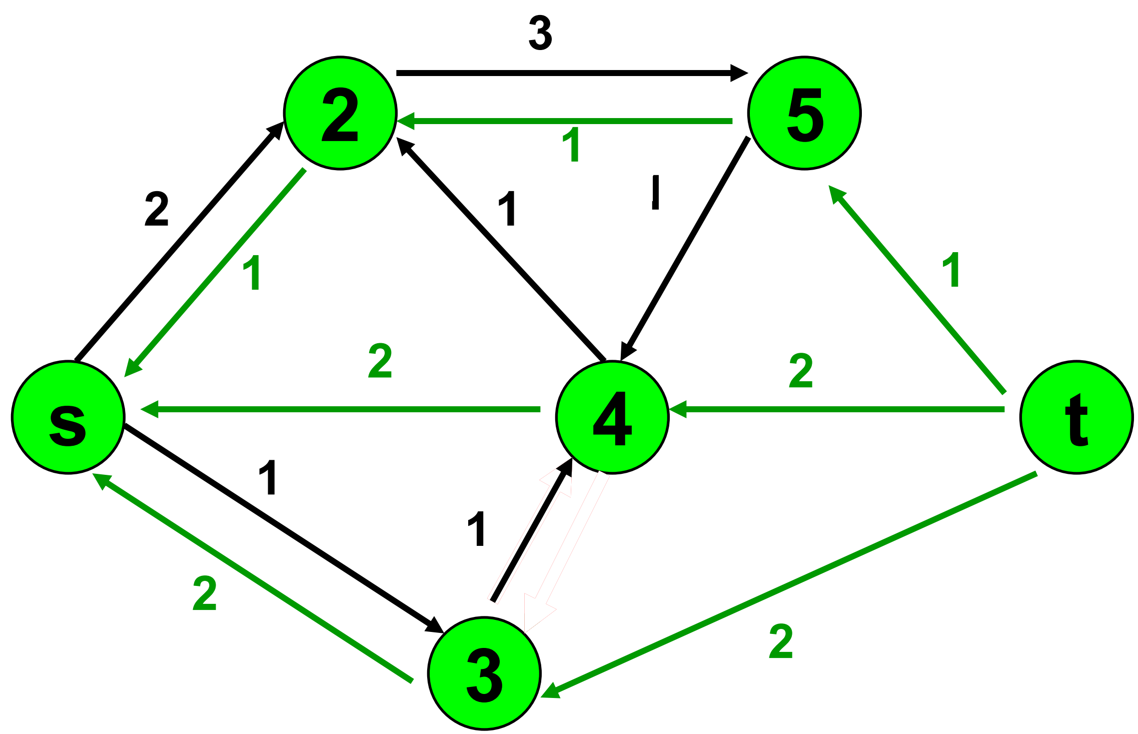


Send $\Delta=1$ unit of flow in the path.
Update residual network

Ford-Fulkerson Max Flow

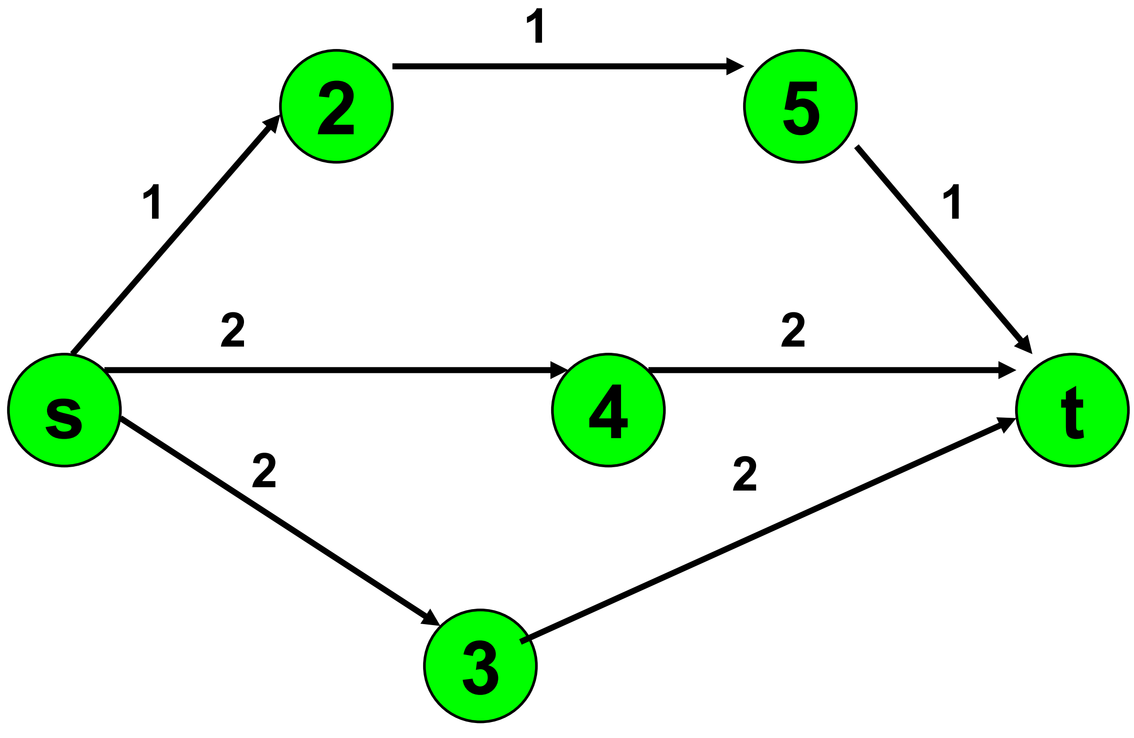


Ford-Fulkerson Max Flow



There is no s-t path in the residual network. This flow is optimal

Ford-Fulkerson Max Flow



The maximum flow solution

The Ford-Fulkerson Algorithm: Complexity

- When all capacities are integer, the Ford-Fulkerson algorithm will terminate in finite iterations
- Reason: The capacity of each augmenting path must be integer.
- Consider all arcs starting from the source node s:
- Each augmentation reduces the residual capacity of some arc (s, j) and does not increase the residual capacity of (s, i) for any i.
- So, the sum of the residual capacities of arcs out of s keeps decreasing, and is bounded below by 0.

- Number of augmentations is O(nU), where U is the largest capacity in the network, $U=\max\{u_{ij}$, for all (i,j) in A$\}$
- Each augmentation takes O(m) time by a breadth-first or depth-first search
- Overall complexity: O(nmU)
- pseudo polynomial time algorithm

Correctness of Ford Fulkerson Algorithm

The algorithm is correct when the following two statements are equivalent:

1. *A flow x is maximum.*
2. *There is no augmenting path in G(x).*

1→2 is easy: Suppose that x is maximum, but there is still an augmenting path in G(x). Then x is not maximum because we can send more flows.

Maximum Flow Problem: Capacity Scaling and Preflow Push Algorithms

Review of Ford-Fulkerson Algorithm

For Ford-Fulkerson algorithm, we know

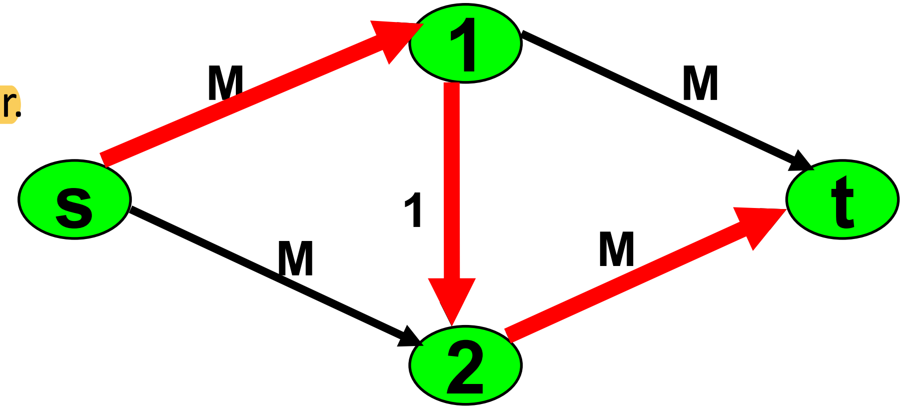
Correctness: A maximum flow is found when there is no augment path on the residual network

Finiteness: When all arc capacities are integers, Ford-Fulkerson algorithm will stop in finite iterations

Remaining problems:

- It may take a very long time for some instances

Example: If M is a very big number.



Irrational Number Capacities

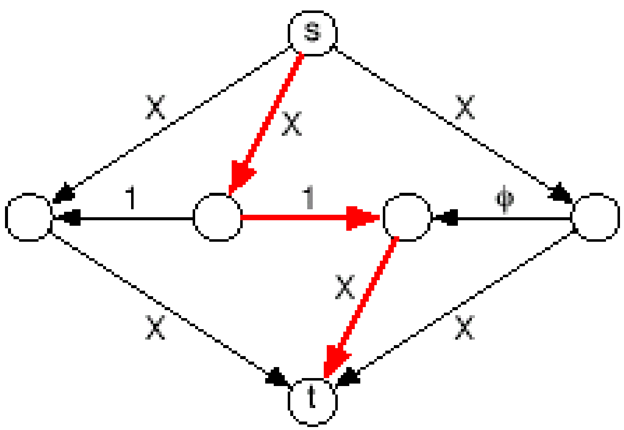
- If some capacities are irrational numbers, the Ford Fulkerson algorithm may never stop

- It may find infinite number of augmenting paths, each with a very small flow to augment

- Sometimes, the total flow converges to the optimal maximum flow

- Sometimes, the total flow converges to a number that is smaller than the maximum flow

Example when Ford-Fulkerson Algorithm Fails

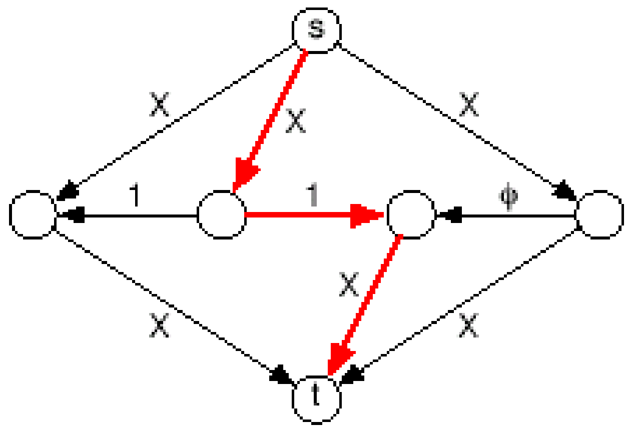


$$X > 3$$

$$\phi = \frac{\sqrt{5}-1}{2}, \text{ or } \phi^2 = 1 - \phi$$

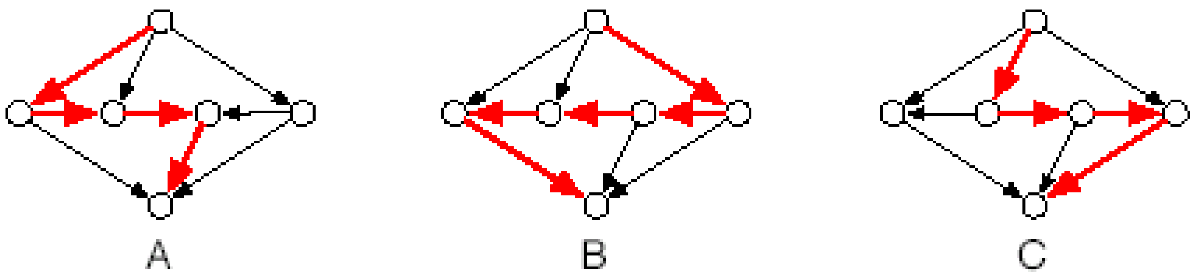
The problem has a maximum flow of 2X+1. Such a maximum flow can be found by 3 augmentations if we choose the augmenting paths correctly.

Example when Ford-Fulkerson Algorithm Fails



$$X > 3$$

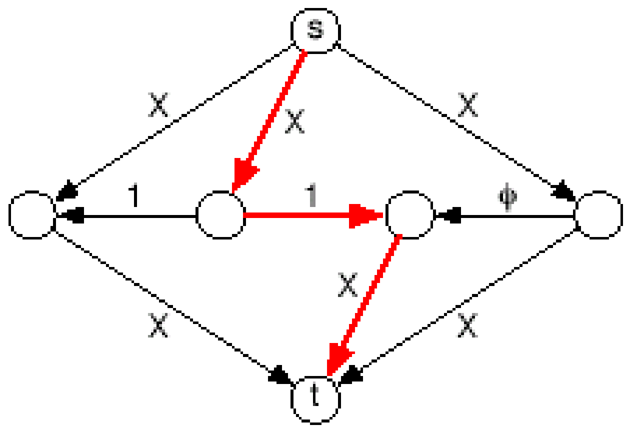
$$\phi = \frac{\sqrt{5}-1}{2}, \text{ or } \phi^2 = 1 - \phi$$



A bad choice of augmenting path

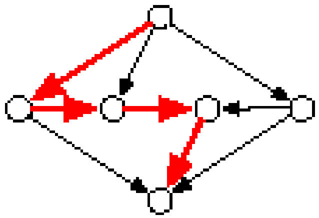
The algorithm starts with the red path in the larger graph, sending 1 unit flow, then chooses paths in B, C, B, A. We will find that we can repeat the above sequence endlessly

Example when Ford-Fulkerson Algorithm Fails

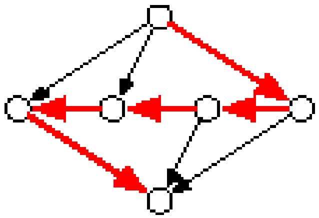


$$X > 3$$

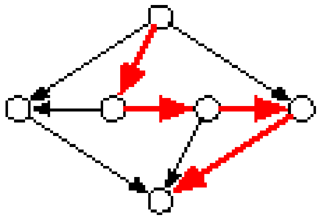
$$\phi = \frac{\sqrt{5}-1}{2}, \text{ or } \phi^2 = 1-\phi$$



A



B



C

Flows that can be sent for each augmenting path (where k =1,3,5…)

Path B	augment φ^k
Path C	augment φ^k
Path B	augment φ^{k+1}
Path A	augment φ^{k+1}

Example when Ford-Fulkerson Algorithm Fails

$$\phi = \frac{\sqrt{5}-1}{2}, \text{or } \phi^2 = 1-\phi$$

Path B	**augment φ^k**
Path C	**augment φ^k**
Path B	**augment φ^{k+1}**
Path A	**augment φ^{k+1}**

In the limit, the total flow to be sent is

$$1 + \sum_{k=1}^{n}(2\phi^k) \rightarrow 1 + \frac{2}{1-\phi} = 1 + \frac{2}{1 - \frac{\sqrt{5}-1}{2}} = 4 + \sqrt{5} < 7$$

But the real maximum flow is 2x+1, larger than 7 when x>3

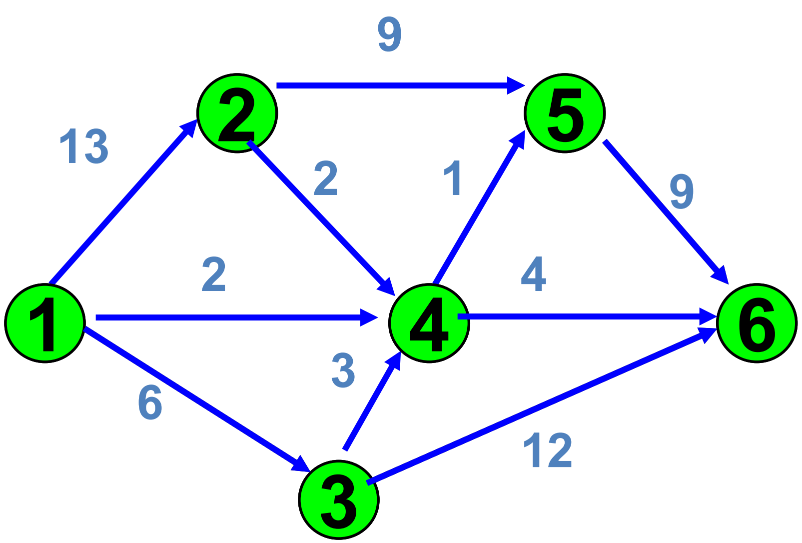
To Improve the Algorithm

- Why the Ford-Fulkerson's algorithm may be inefficient?
  - It may choose a wrong sequence of augmenting paths, which causes only a small amount of flow to be sent for each augmentation
- Improvement by a systematic way to find augmenting paths
  - Find an augmenting path with a sufficiently large capacity
    - The capacity scaling algorithm

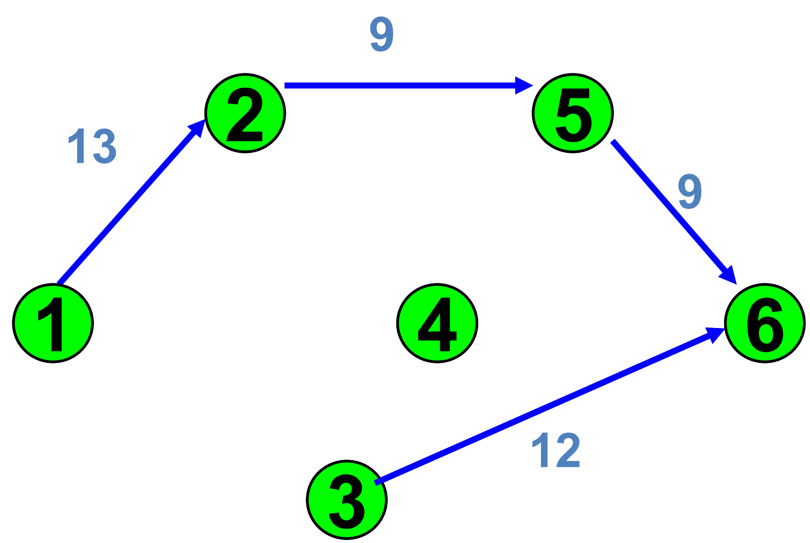
Capacity scaling algorithm

Capacity Scaling Algorithm

- Introduce a parameter Δ, and define the Δ-residual network G(x, Δ) by including only arcs with residual capacity being at least Δ
- Observations
  - Each augmentation on G(x, Δ) sends at least Δ units of flow
  - G(x,1) is the same as G(x)



G(x)



G(x,8), an augmenting path with 9 units can be found

Capacity Scaling Algorithm

- Algorithm ideas: working on $G(x, \Delta)$, with a large Δ at the beginning, and reducing Δ gradually

- Algorithm steps:

  Let $x=0$, $\Delta=2^{\lfloor \log_2 U \rfloor}$

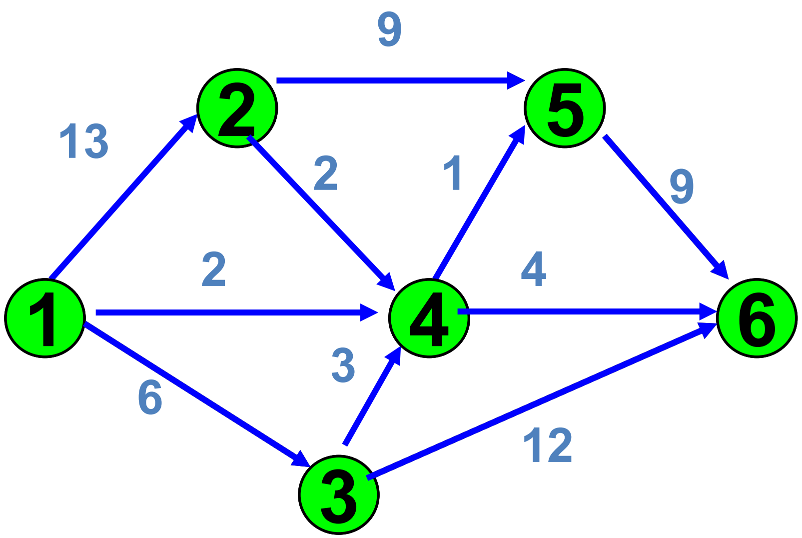
  While $\Delta \geq 1$ do Begin

  - While $G(x, \Delta)$ has an augmenting path do Begin
    - Augment flow along the augmenting path
    - Update $G(x, \Delta)$
  - End
  - $\Delta = \Delta/2$

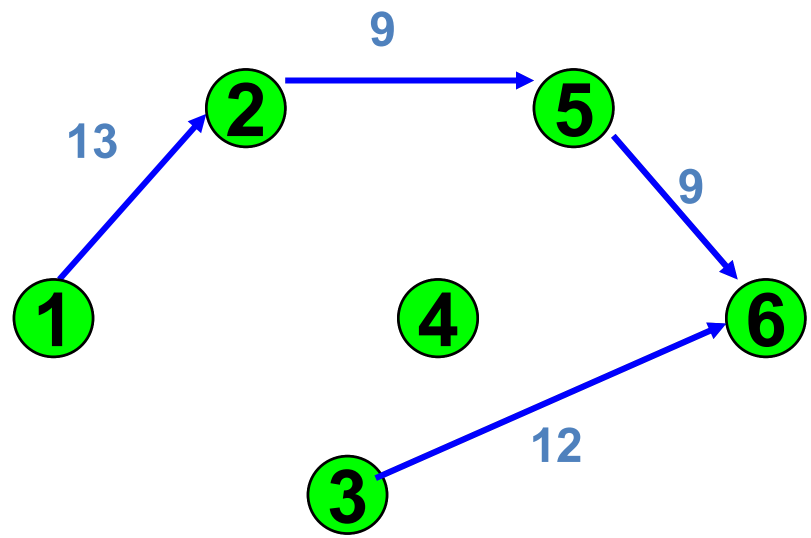
  End

The algorithm is correct because it ends at $\Delta=1$, and $G(x,1)$ is the same as $G(x)$

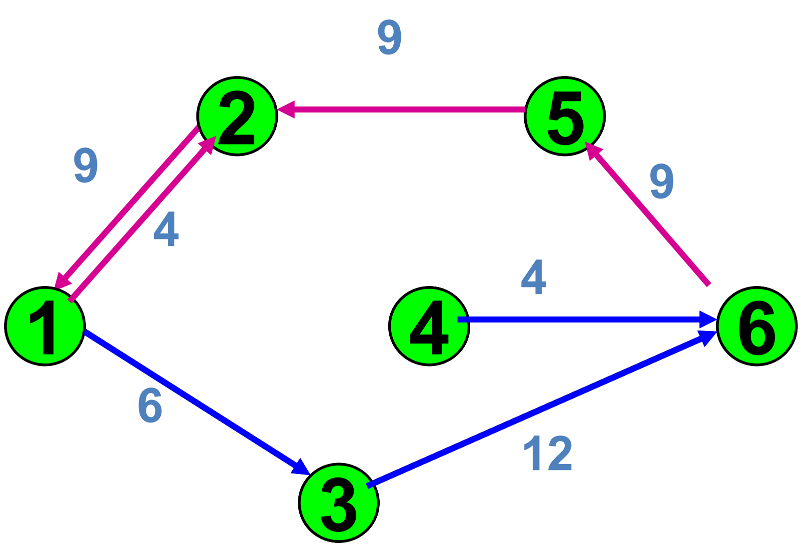
Example of G(x,Δ)



G(x)

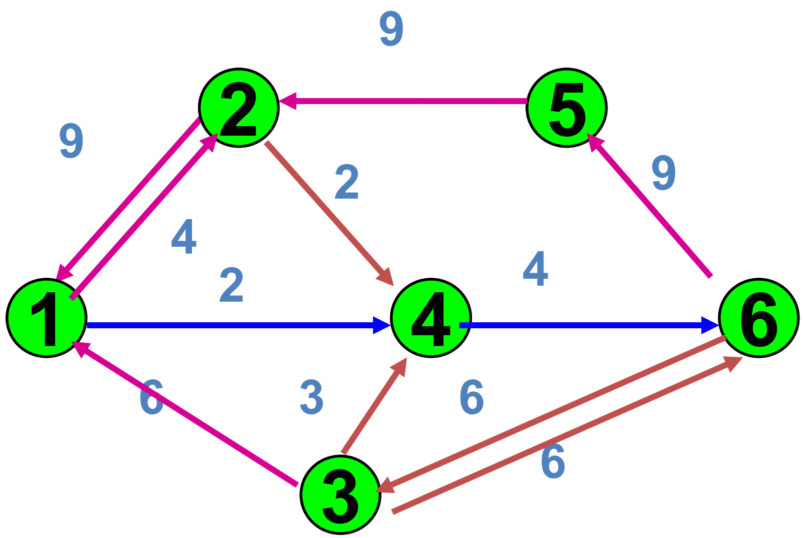


G(x,8), an augmenting path with 9 units can be found

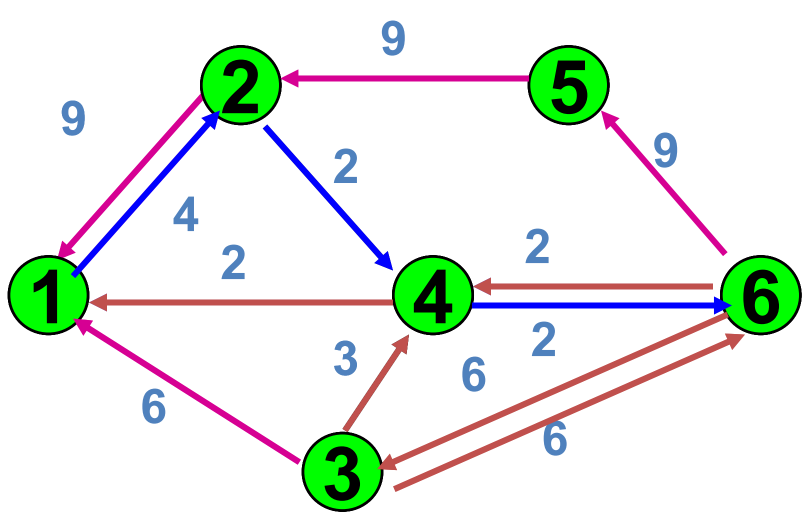


G(x,4) after phase 1: an augmenting path with 6 units of flow can be found

Example of G(x,Δ)

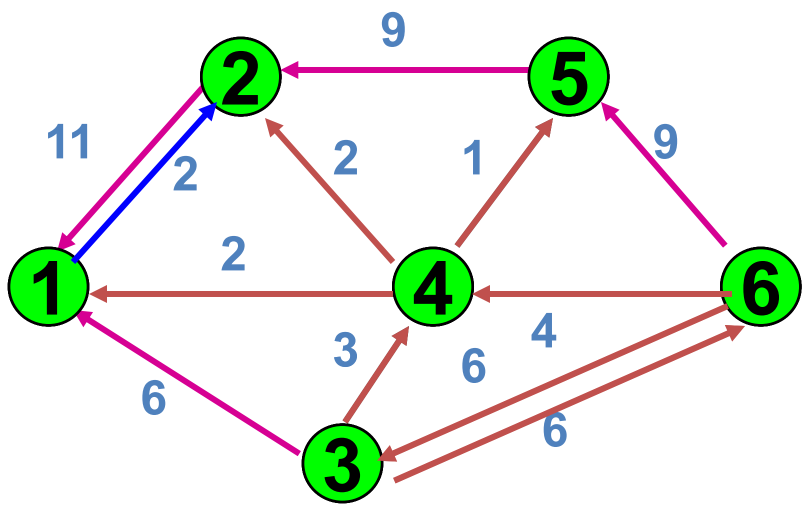


G(x,2): an augmenting path with 2 units of flow can be found



G(x,2): another augmenting path with 2 units of flow

Example of G(x,Δ)



G(x,1)=G(x), no augmenting path can be found

Maximum flow = 9+6+2+2 =19

Total No. of iterations:4

Preflow Push Algorithm

Preflow Algorithms

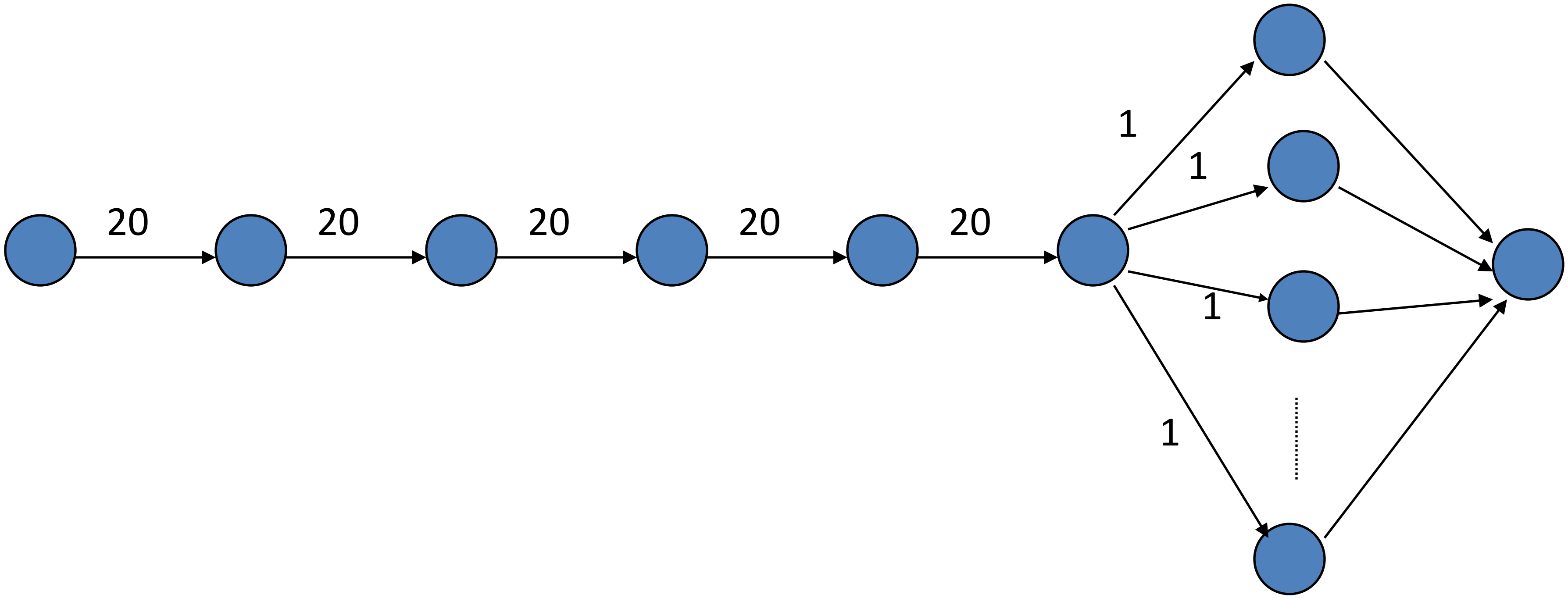
For algorithms depending on augmenting paths:

- What is good?
  - It always maintains a feasible solution
- What might be inefficient?
  - It takes time to find an augmenting path
  - It takes time to send flow along an augmenting path (i.e. to update residual network)

New ideas:

- Allow infeasibility before the final solution
  - called "preflow"
- push flows from source to destination arc by arc

A Bad Example for Augmenting Path Algorithm



- It requires 20 times of flow augmenting along the long augmenting path, regardless of the path choice
- Can we be more efficient?

Active Nodes

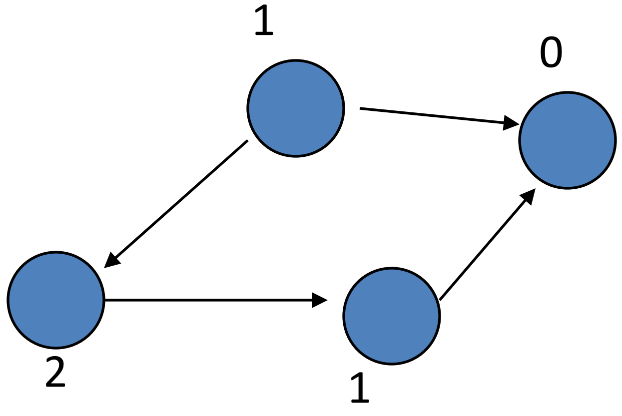
- Given a flow x (which may be infeasible),
  - The excess of a node, denoted by e(i), is
  - total inflow to node i – total outflow from node i

- A node i is active if e(i)>0
  - existence of active node → flow x is infeasible

- Ideas of Preflow Push Algorithm
  - choose an active node and try to push its excess flow to other nodes closer to destination t (done by pushing along an admissible arc)

Distance Labels

- For a network G with source node s, destination node t, and a flow x (which may violate flow conservation at some nodes)
- Assume unit arc cost
- d(t)=0
- d(i) is a lower bound on the distance from i to t in the residual network.
- Smaller d(i) implies closer to destination t
- Given valid distance labels,
  an arc (i,j) is called admissible if d(i)=d(j)+1
- An admissible path from node i is a shortest path from node i to node t



Preflow Push Algorithm

- Step 1 (Preprocess)
- compute exact distance labels d(i) (the shortest path from node i to destination node t)
- Push flow from source node s along all arcs (s,i)
- using full capacity of arc (s,i), i.e., let $x_{s,i}=r_{s,i}$
- Construct G(x), and let d(s)=n

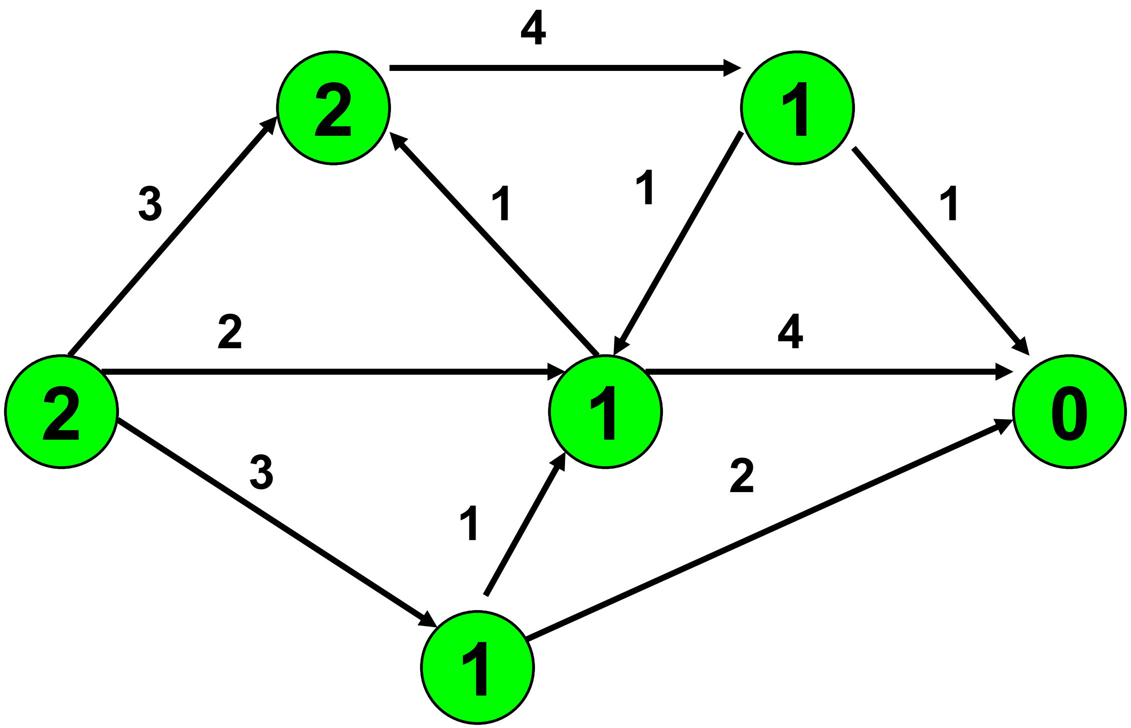
- Step 2 (main loop)
- While G(x) has an active node i Do
- if G(x) has an admissible arc (i,j), then
- push min{e(i),r_{ij}} units of flow to node j
- update G(x)
- else // no admissible arcs from node i
- update d(i)=min{d(j)+1 | for all (i,j) in G(x) }

Push

Relabel

- // the algorithm terminates when only node s has positive excess
- // some nodes end up with distance labels larger than n when pushing flow to node s (sending flow back to source node)

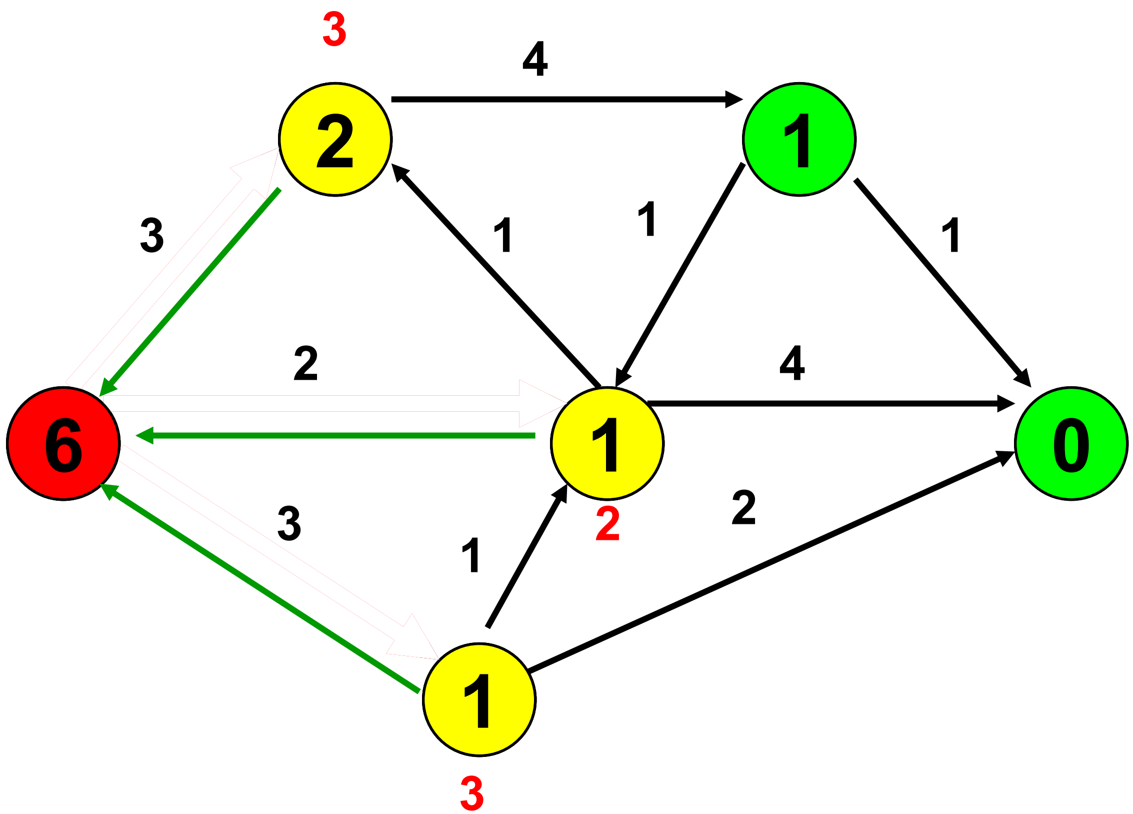
Initialize Distances



Change the node label to distance label.

d(j) is at most the distance of j to t in G(x)

Saturate Arcs out of node s

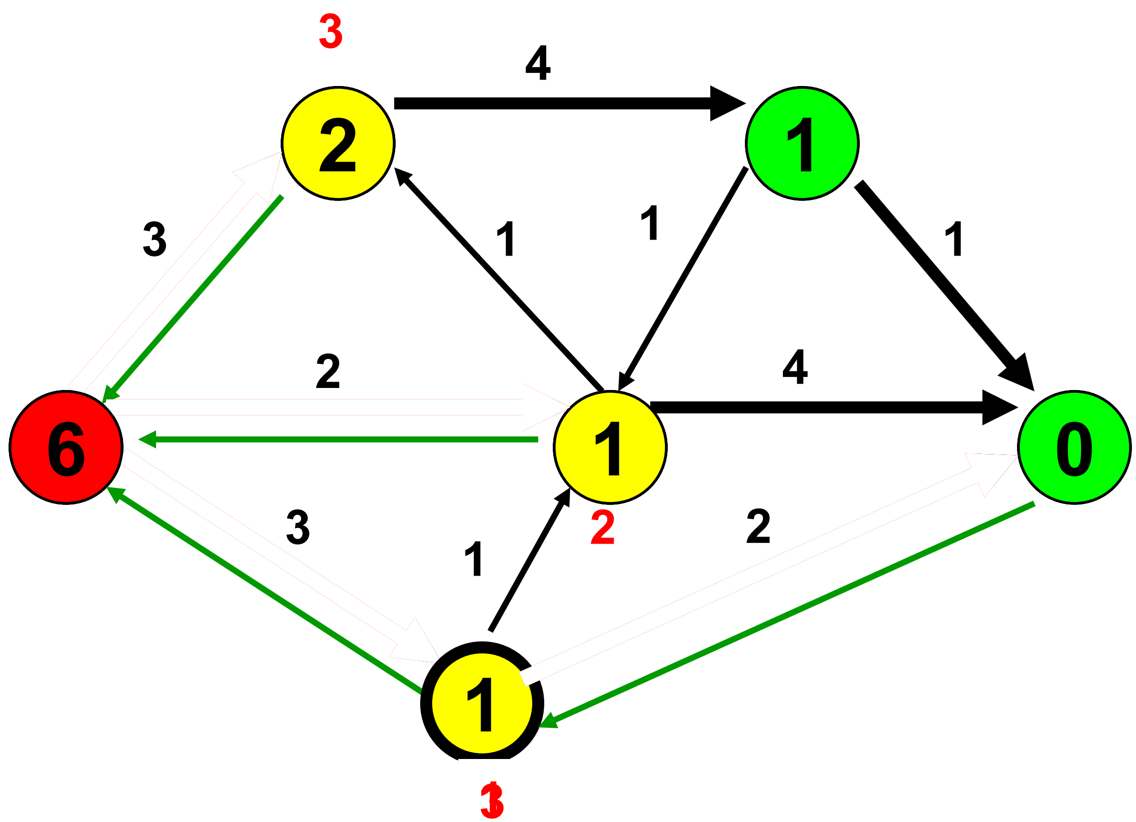


Saturate arcs out of node s.

Move excess to the adjacent arcs

Relabel node s after all incident arcs have been saturated.

Select, then relabel/push

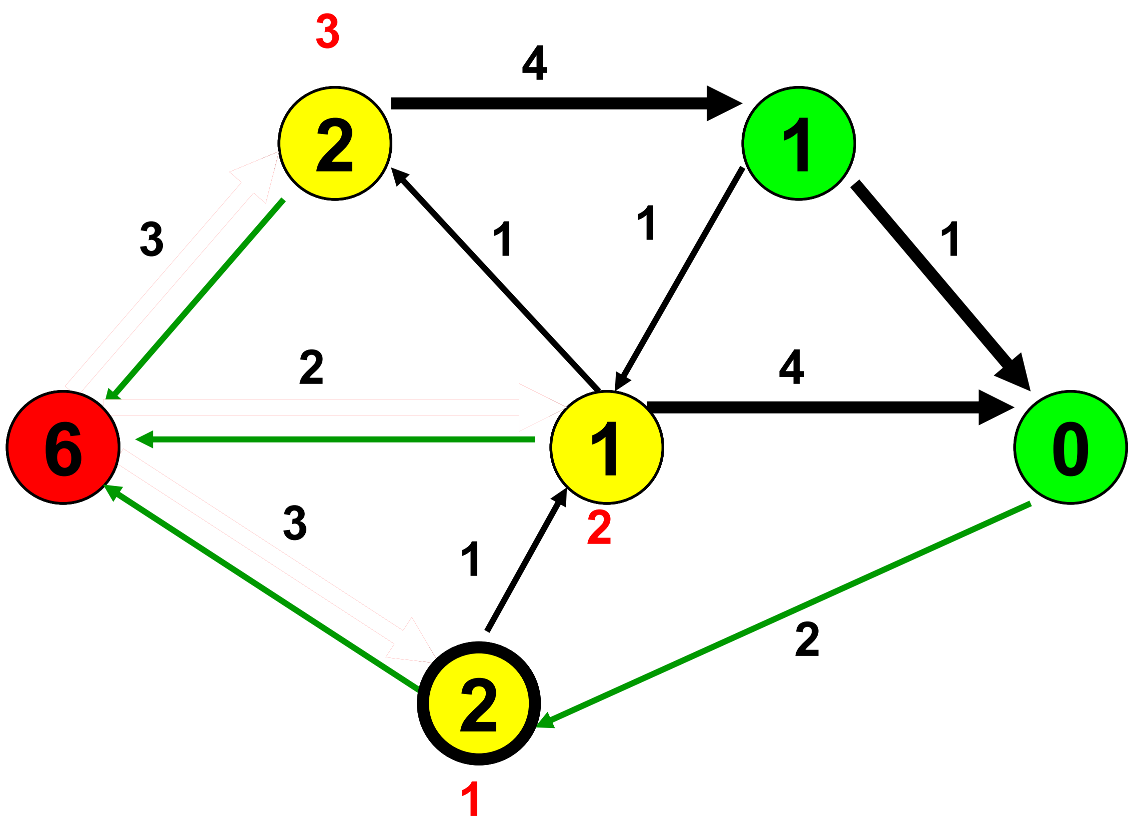


Select an active node, that is, one with excess

Push/Relabel

Update excess after a push

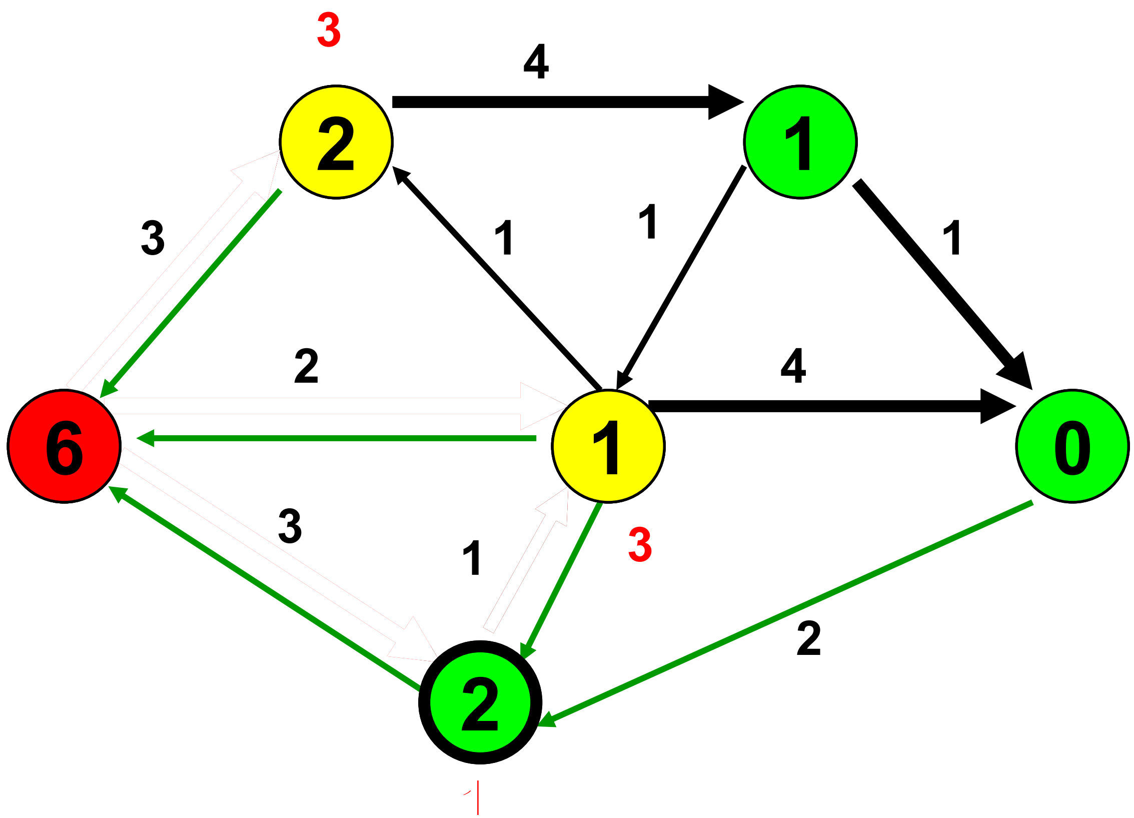
Select, then relabel/push



Select an active node, that is, one with excess

No arc incident to the selected node is admissible. So relabel.

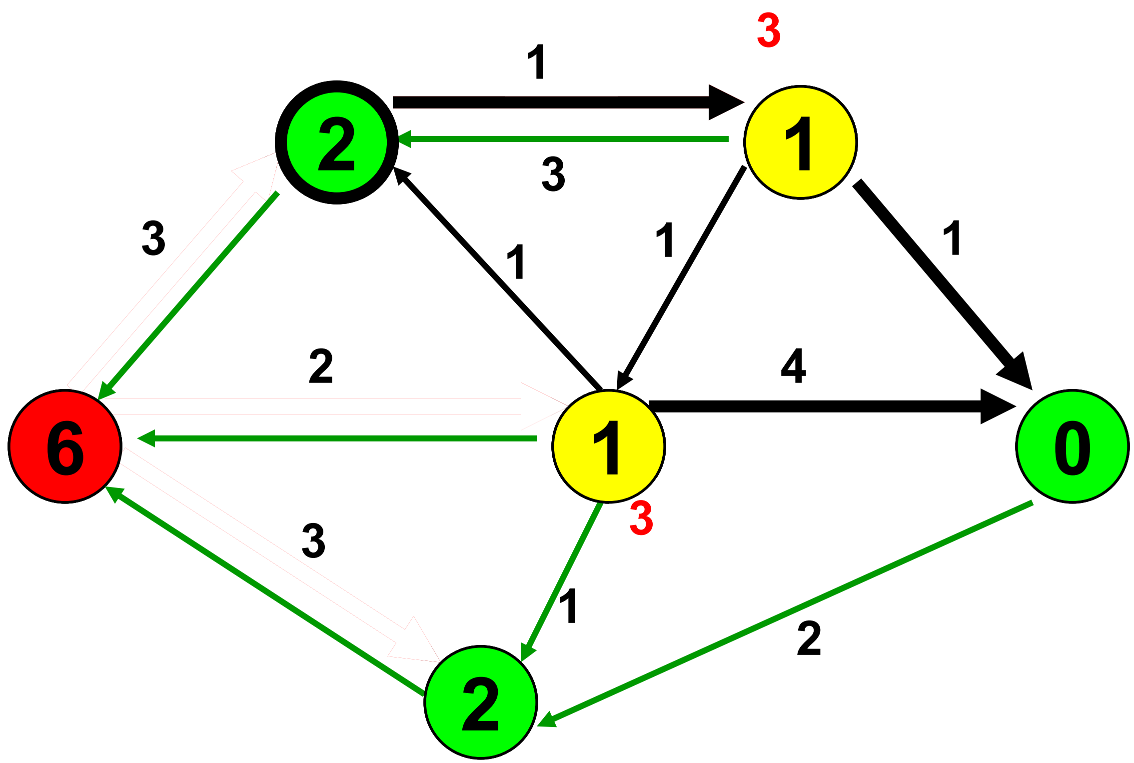
Select, then relabel/push



Select an active node, that is, one with excess

Push/Relabel

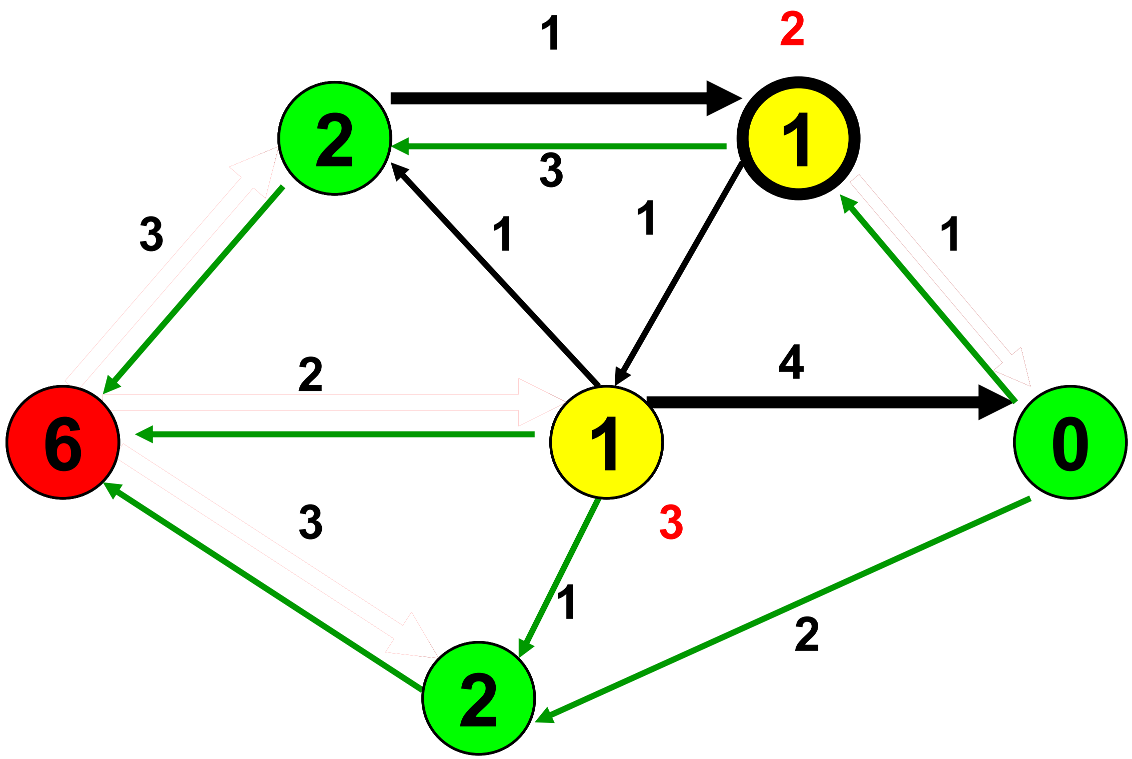
Select, then relabel/push



Select an active node.

Push/Relabel

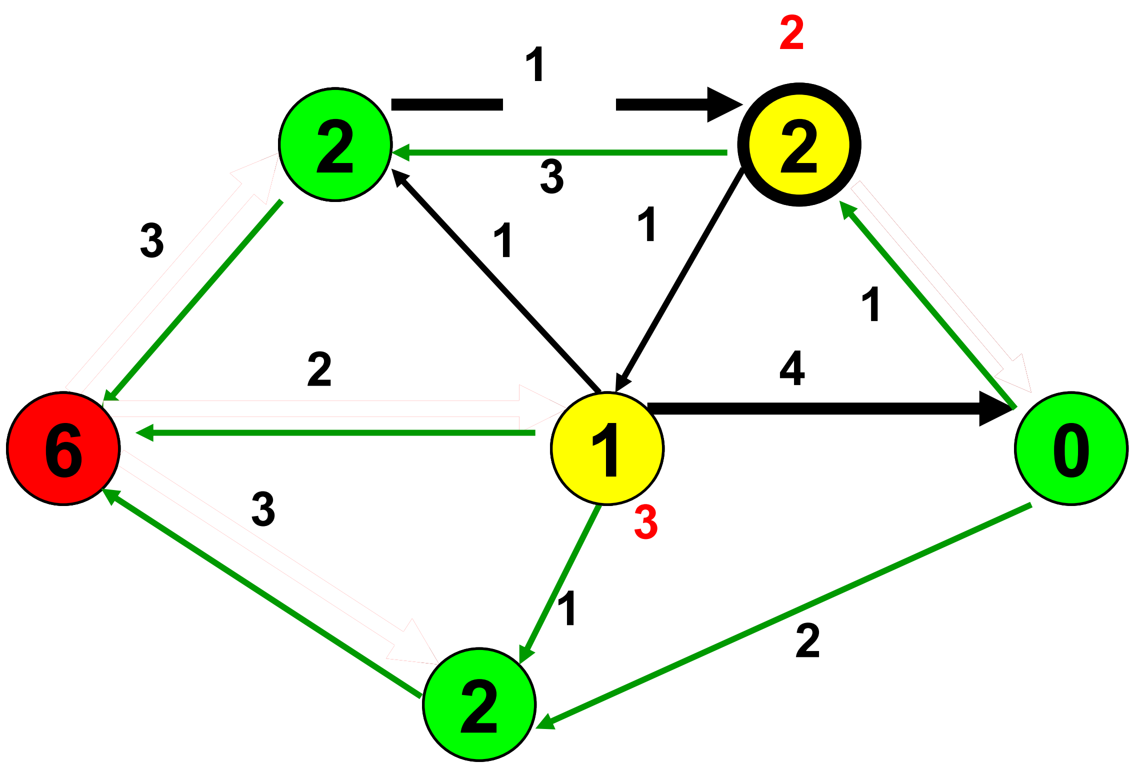
Select, then relabel/push



Select an active node.

Push/Relabel

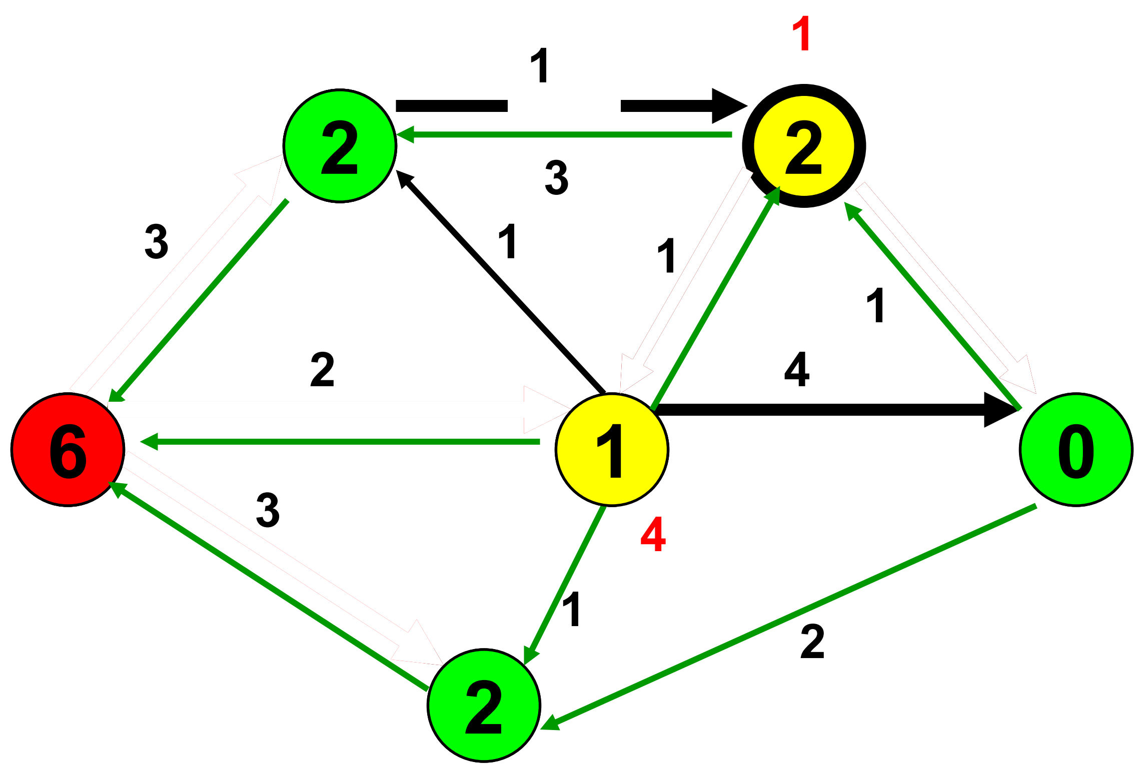
Select, then relabel/push



Select an active node.

There is no incident admissible arc. So Relabel.

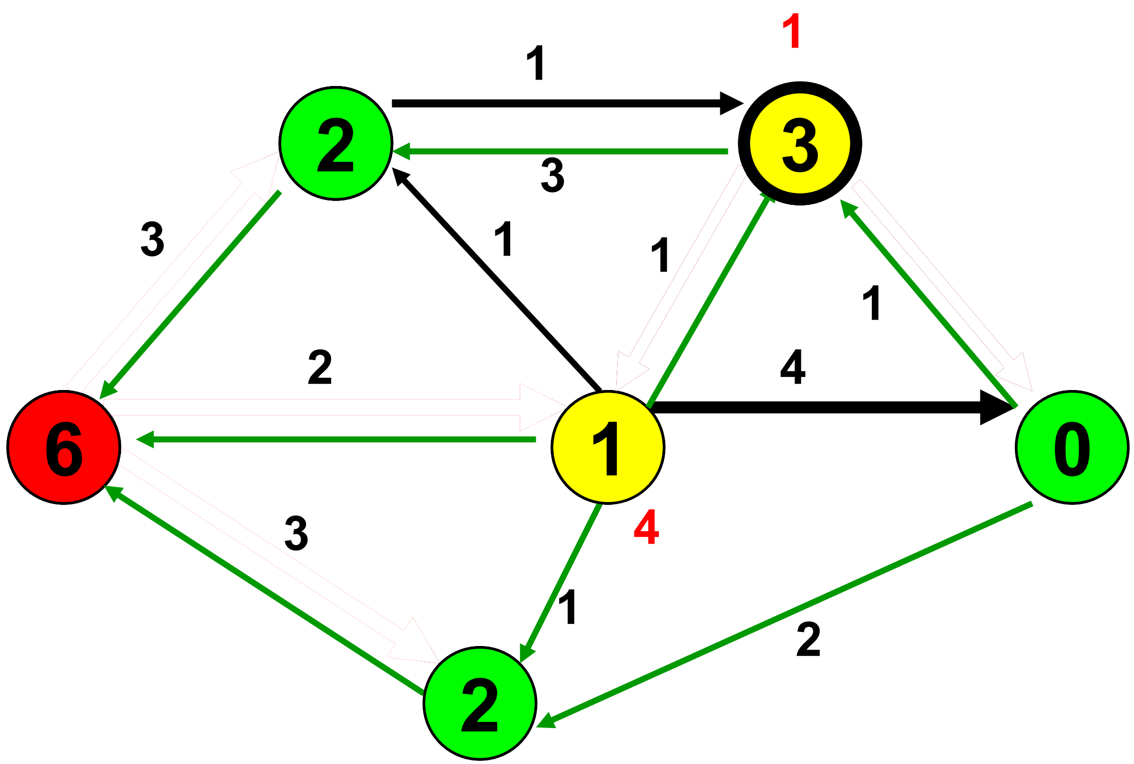
Select, then relabel/push



Select an active node.

Push/Relabel

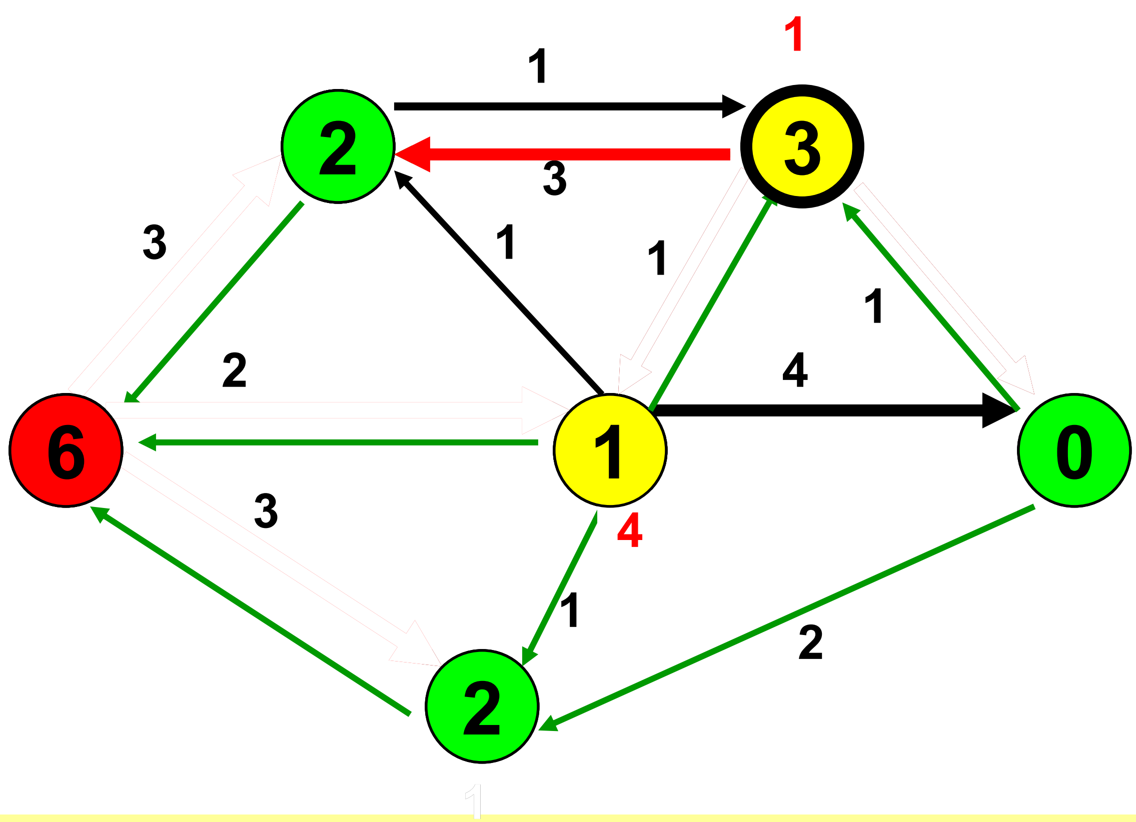
Select, then relabel/push



Select an active node.

There is no incident admissible arc. So relabel.

Select, then relabel/push



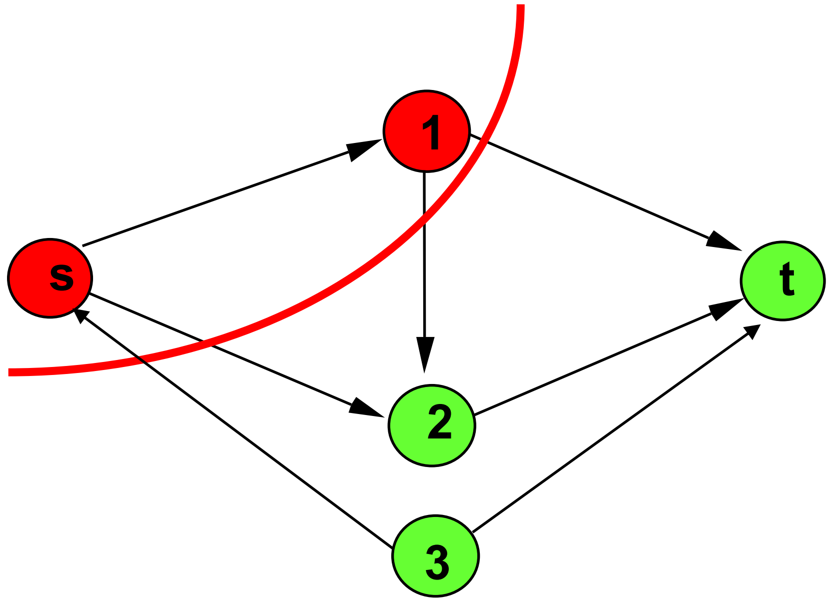
Select an active node.

Push/Relabel

Maximum flow and minimum cut

A duality theory, and a proof of the correctness of the maximum flow algorithm

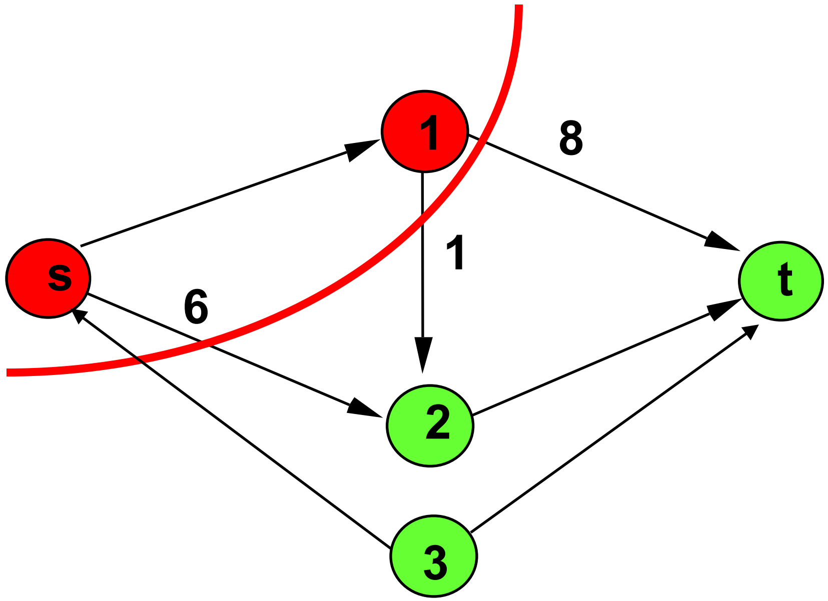
A Cut

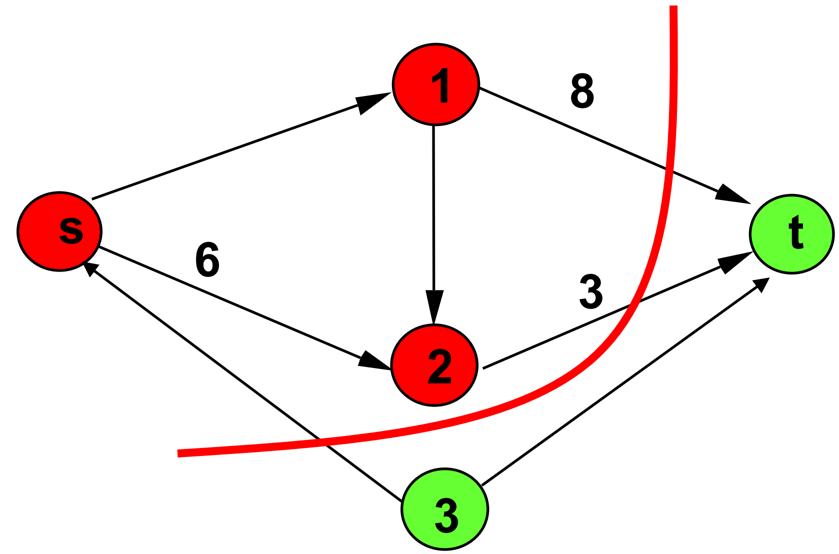


An *(s,t)-cut* in a network G = (N,A) is a partition of N into two disjoint subsets S and T such that s ∈ S and t ∈ T, e.g., S = { s, 1 } and T = { 2, 3, t }.

In the given cut (S,T),
an arc (i,j) is a forward arc if i ∈S and j ∈T, e.g., (1,t), (1,2), (s,2)
an arc (i,j) is a backward arc if j ∈S and i ∈T, e.g., (3,s)

Cut Capacity





The ***capacity*** of a cut (S,T) is the sum of capacities of all forward arcs

CAP(S,T) = $\sum_{i \in S} \sum_{j \in T} u_{ij}$

In the example, the cut capacity of S=(s,1) is 6+1+8=15

the cut capacity of S=(s,1,2) is 8+3=11

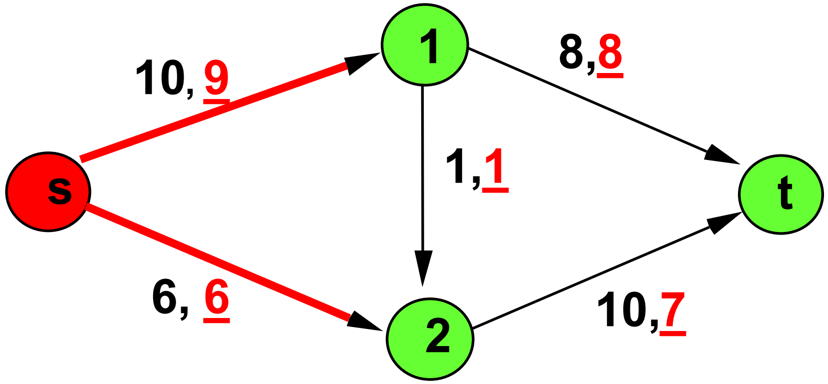
Min-Cut problem is to find a cut with the smallest capacity

Weak Duality Theorem for the Max Flow Problem

- *Theorem. If x is any feasible flow and if* (*S,T*) *is an* (*s,t*)*-cut, then the flow value v*(*x*) *from source to destination in x is at most CAP*(*S,T*).

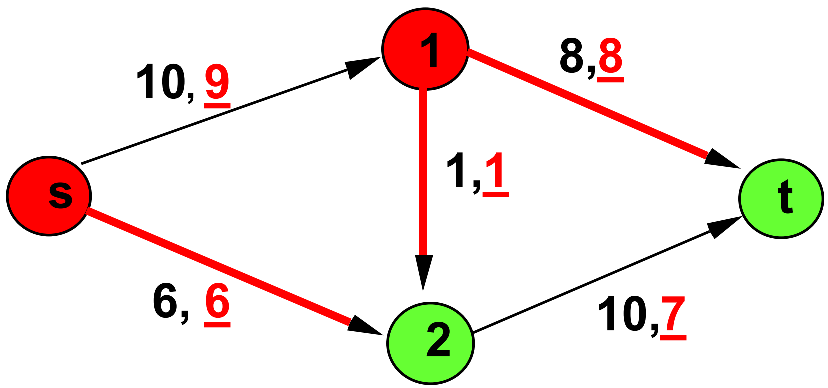
- Ideas for proof: We define the *flow across the cut* (S,T) to be all flow on forward arcs minus all flow on backward arcs

- $F_x(S,T) \quad = \quad \sum_{i\in S}\sum_{j\in T} x_{ij} \quad - \quad \sum_{i\in S}\sum_{j\in T} x_{ji}$

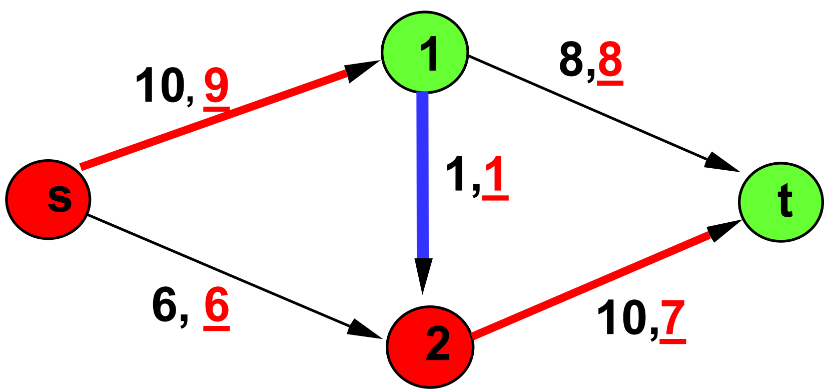


If S = {s}, then the flow across (S, T) is 9 + 6 = 15.

Flows Across Cuts



If S = {s,1}, then the flow across (S, T) is 8+1+6 = 15.



If S = {s,2}, then the flow across (S, T) is 9 + 7 - 1 = 15.

More on Flows Across Cuts

- Claim 1: Let (S,T) be any s-t cut. Then $F_x(S,T) = v =$ flow into t.
- In simple words, the flow sent from s to t across any cut (S,T) is $F_x(S,T)$, forward flow minus backward flow
- Claim 2: The flow across (S,T) is at most the capacity of a cut.
- This happens only when all backward flow is 0

Strong Duality: Max Flow Min Cut Theorem

- *Theorem. (Optimality conditions for max flows).* *The following are equivalent.*

  - *1. A flow x is maximum.*
  - *2. There is no augmenting path in G(x).*
  - *3. There is an s-t cut (S, T) whose capacity is the flow value of x.*

- *Corollary. (Max-flow Min-Cut). The maximum flow value is the minimum value of a cut.*

- Proof of Theorem. $1 \Rightarrow 2.$
- Suppose that there is an augmenting path in G(x). Then x is not maximum.

Continuation of the proof.

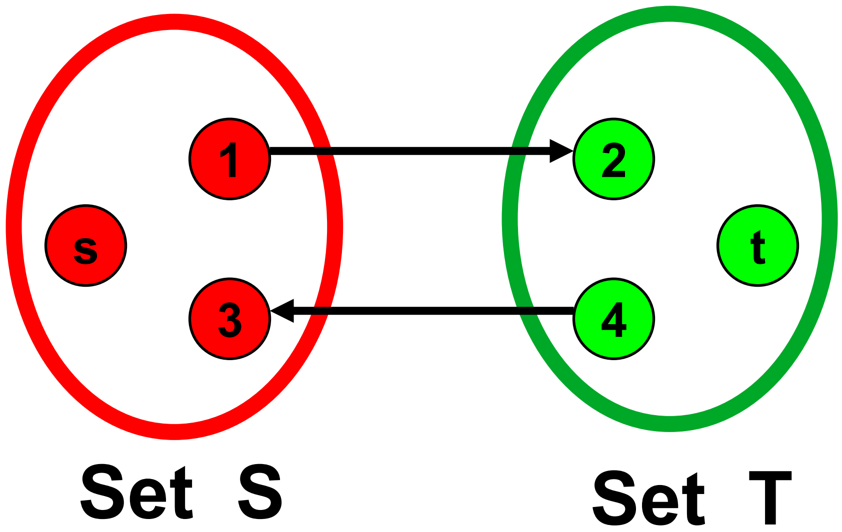
- $3 \Rightarrow 1$. Let $v = F_x(S, T)$ be the flow from s to t. By assumption, $v = CAP(S, T)$. By weak duality, the maximum flow is at most $CAP(S, T)$. Thus the flow is maximum.

- $2 \Rightarrow 3$. Suppose there is no augmenting path in G(x).
- Claim: Let S be the set of nodes reachable from s in G(x). Let T = N\S. Then there is no arc in G(x) from S to T.

- Thus $i \in S$ and $j \in T \Rightarrow$ (In the original graph) $x_{ij} = u_{ij}$
- $i \in T$ and $j \in S \Rightarrow$ (In the original graph)$x_{ij} = 0$.

Final steps of the proof

- Thus $i \in S$ and $j \in T \Rightarrow x_{ij} = u_{ij}$
- $i \in T$ and $j \in S \Rightarrow x_{ij} = 0.$



There is no arc from S to T in G(x)

$x_{12} = u_{12}$

$x_{43} = 0$

If follows that

$F_x(S,T) = \sum_{i \in S}\sum_{j \in T} x_{ij} - \sum_{i \in S}\sum_{j \in T} x_{ji}$

$= \sum_{i \in S}\sum_{j \in T} u_{ij} - \sum_{i \in S}\sum_{j \in T} 0 = \text{CAP}(S,T)$

An Alternative Viewpoint

LP formulation for maximum flow problem

$$\max v$$

s.t.

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v, & \text{for } i = s \\ 0, & \text{for } i \in N - \{s,t\} \\ -v, & \text{for } i = t \end{cases}$$

$$0 \le x_{ij} \le u_{ij}, \text{ for } \forall (i,j) \in A$$

The dual of the LP is

$$\min \quad \sum_{(i,j)} u_{i,j} v_{i,j}$$

$$s.t. \quad y_t - y_s = 1$$

$$y_i - y_j + v_{i,j} \ge 0, \forall (i,j)$$

$$v_{i,j} \ge 0, \forall (i,j)$$

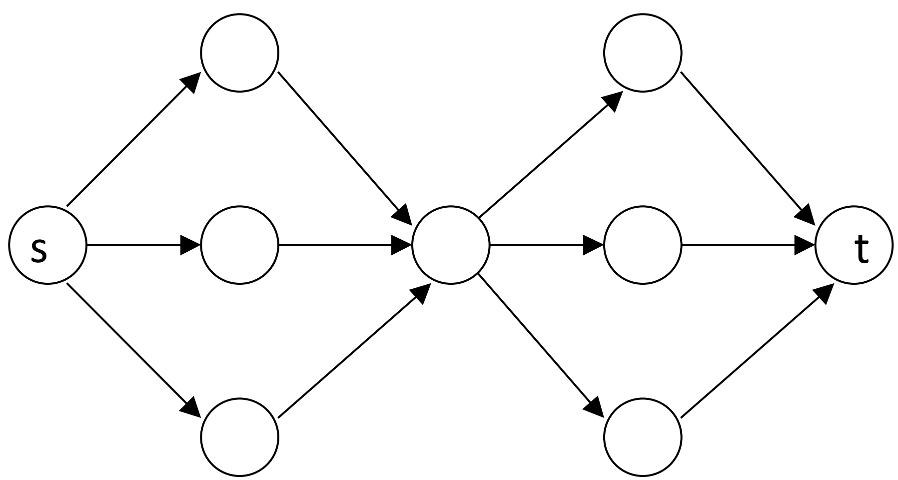
Interpretation of dual

y_i=0 → node *i* in S

y_i=1 → node *i* in T

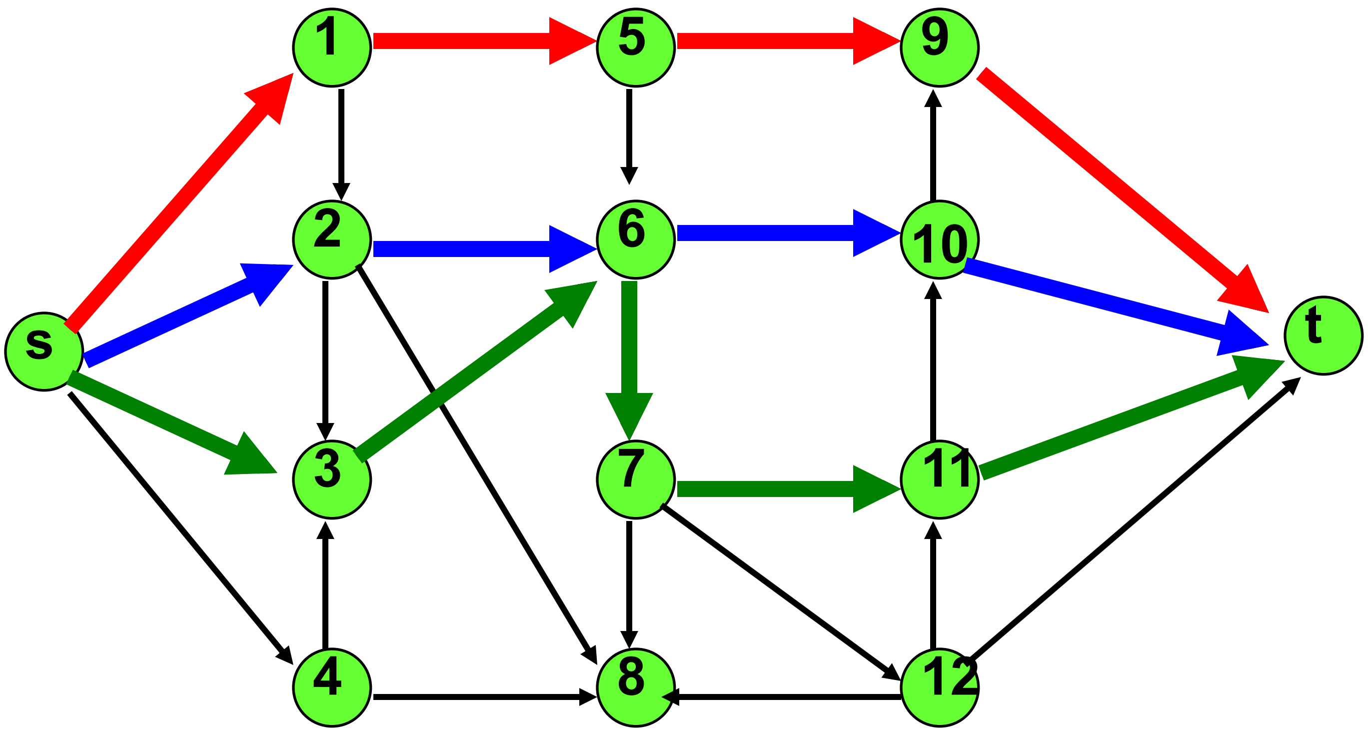
More on Duality: Network Reliability

- Communication Network
- What is the maximum number of arc-disjoint paths from s to t?
  - Two paths are arc-disjoint if they do not have any common arcs.
  - How can we determine this number?
  - Theorem. *Let G = (N,A) be a directed graph. Then the maximum number of arc-disjoint paths from s to t is equal to the minimum number of arcs upon whose deletion there is no directed s-t path.*



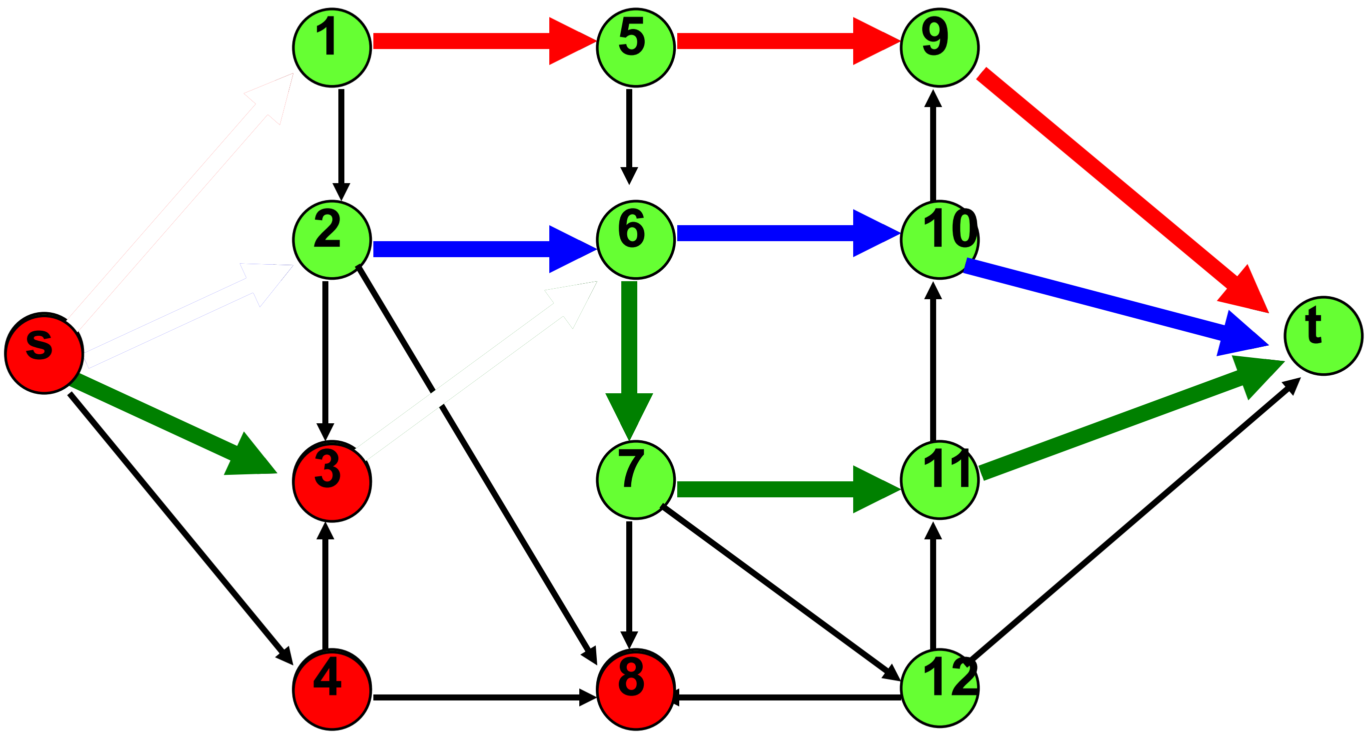
There are 3 arc-disjoint s-t paths

Can be found by a maximum flow problem by assuming unit arc capacity



Deleting 3 arcs disconnects s and t

A minimum cut is found

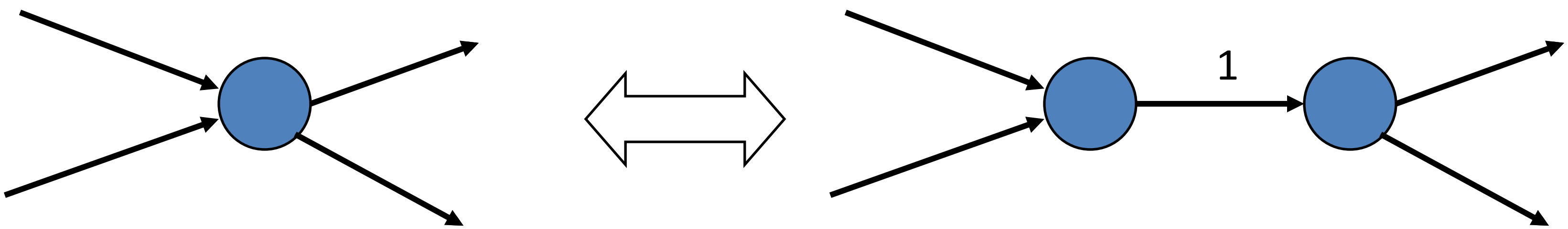


Let S = {s, 3, 4, 8}. The only arcs from S to T = N\S are the 3 deleted arcs.

More on Duality: Node disjoint paths

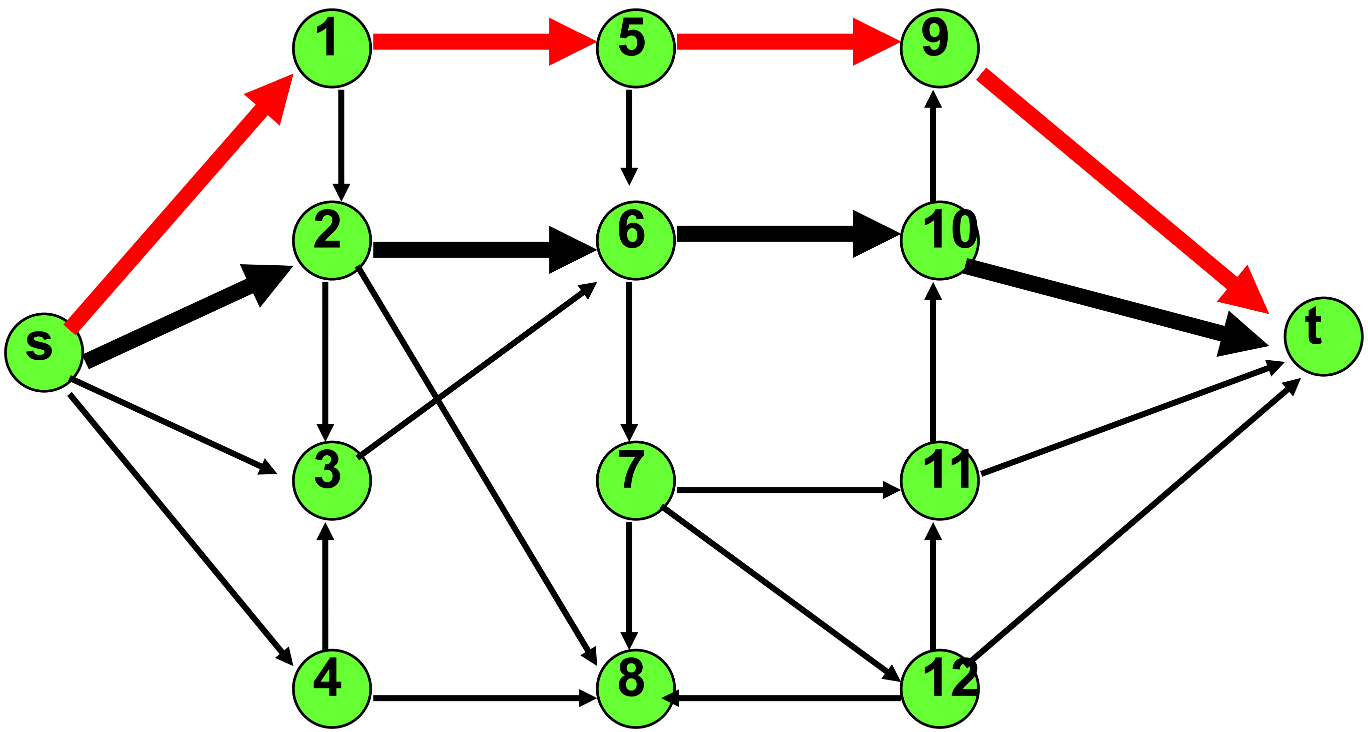
- Two s-t paths P and P' f are said to be *node-disjoint* if the only nodes in common to P and P' are s and t.

- How can one determine the maximum number of node disjoint s-t paths?
- Answer: maximum flow after node splitting

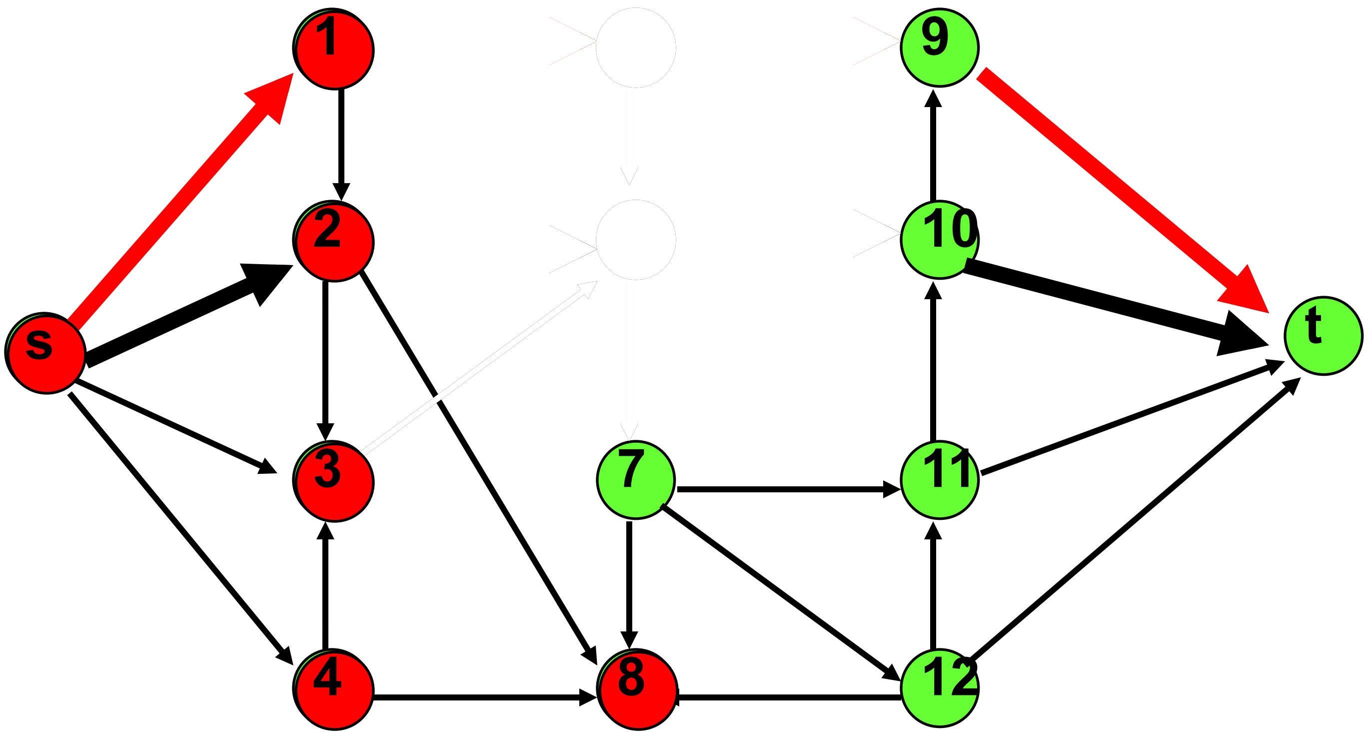


There are 2 node disjoint s-t paths.

Theorem. ***Let G = (N,A) be a network with no arc from s to t. The maximum number of node-disjoint paths from s to t equals the minimum number of nodes whose removal from G disconnects all paths from nodes s to node t.***



Deleting 5 and 6 disconnects t from s?



Let S = {s, 1, 2, 3, 4, 8}

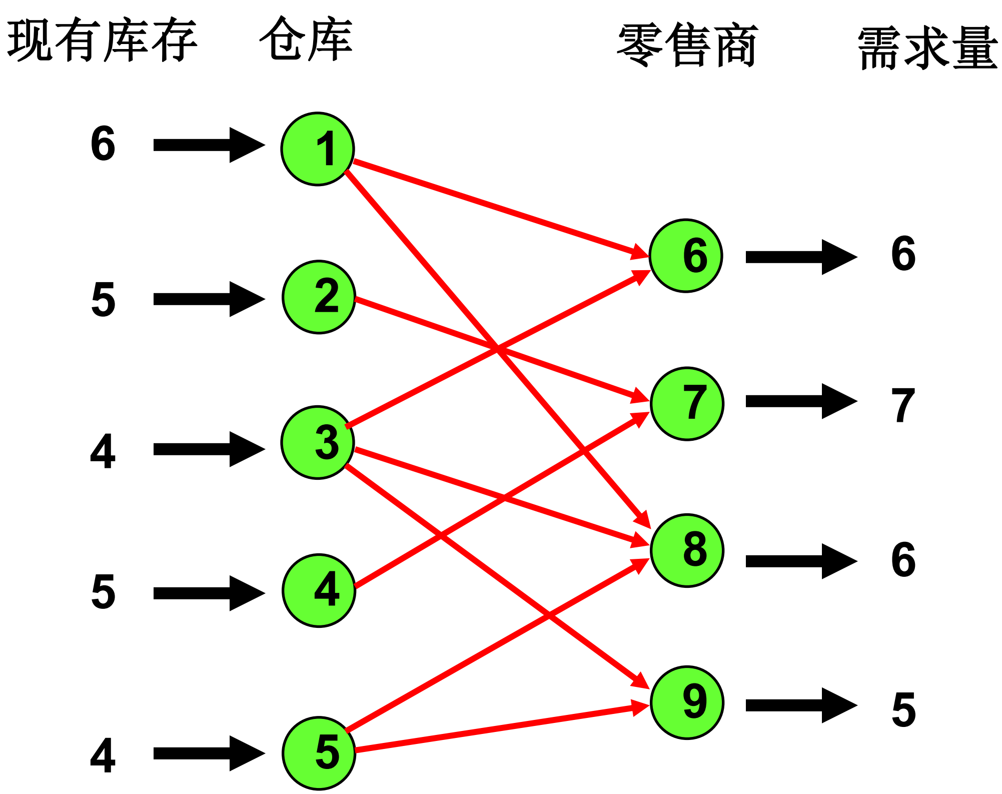
Let T = {7, 9, 10, 11, 12, t}

There is no arc directed from S to T.

最大流问题的应用

1. 可行流问题

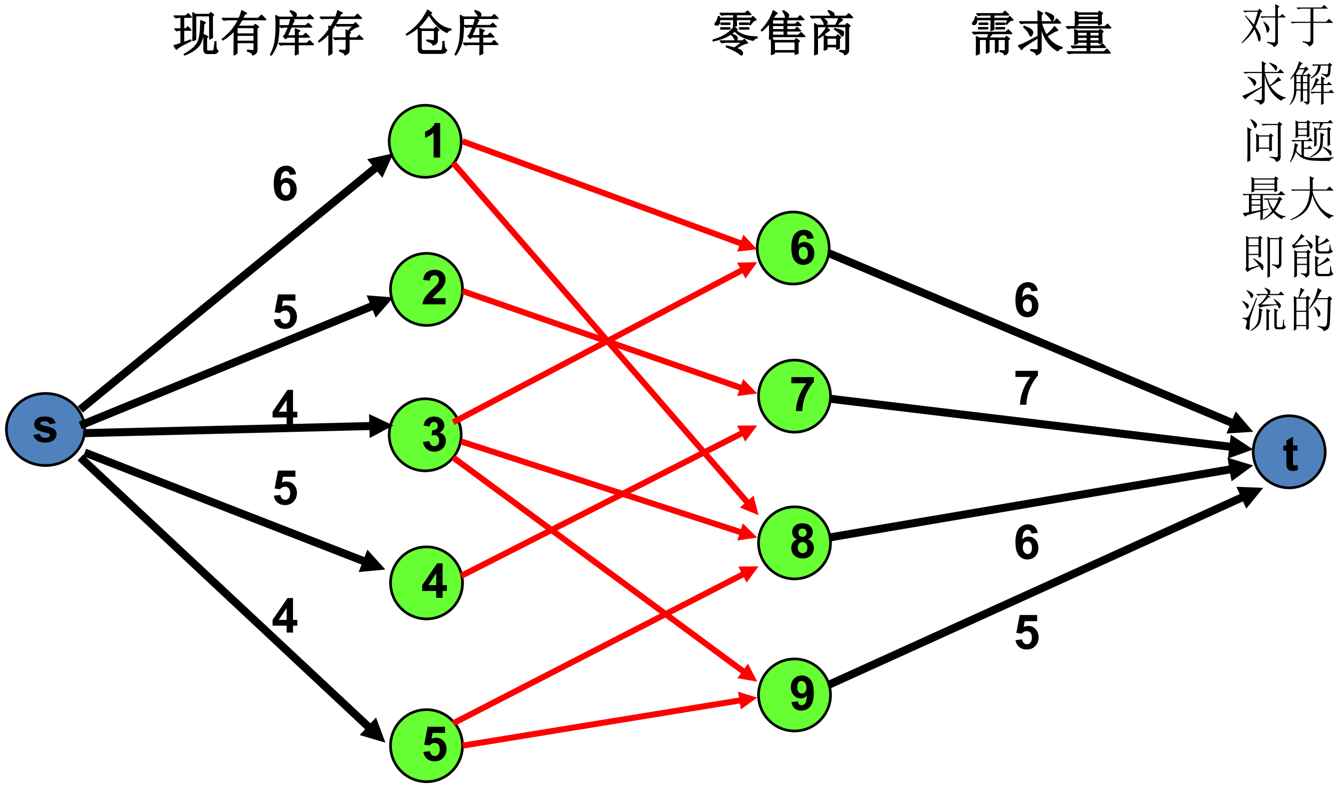
是否存在从各个仓库至零售商的分配方法，使得在每个仓库的现有库存约束下，所有零售商的需求都能得到满足？



最大流问题的应用

1. 可行流问题

该问题可以转化为如下的最大流问题：



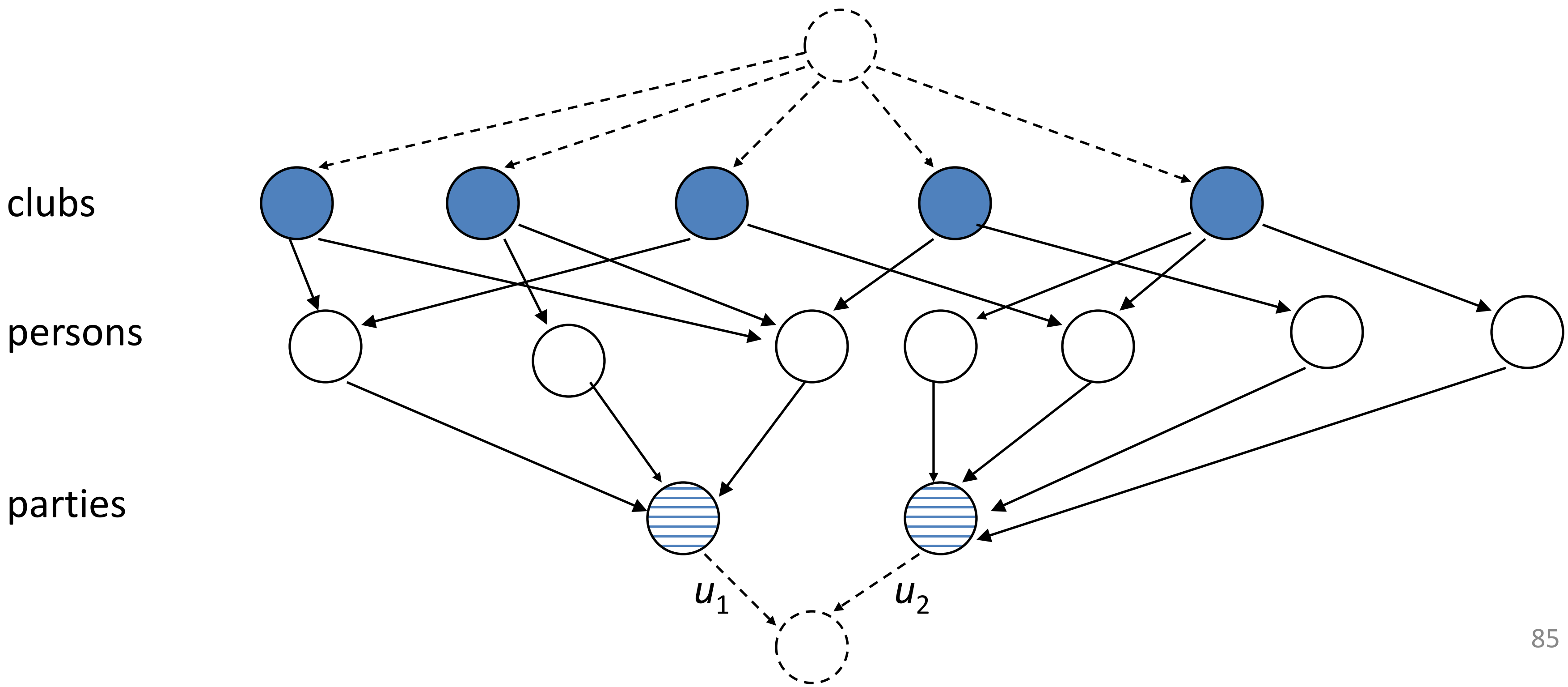
对于新构造的网络，求解从s至t的最大流问题，观察该问题的最大流解是不是24，即能回答原问题可行流的存在性。

最大流问题的应用

2. 选代表问题

There are *r* persons, *q* clubs, and *p* political parties, where each person belongs to at least one club and exactly one party.

We want to find q representatives, each from a club, such that the number of representatives in party *k* is no more than u_k.



Application 3: Matrix Rounding

Given a matrix of real number D=[d_{ij}]

with a_i=sum of row i, b_j=sum of column j

Decision: to round d_{ij}, a_i, b_j to integers, either rounding up or down

Constraint: in the rounded matrix

each row sum is the round a_i, each column sum is the round b_j

Model: each element as an arc with lower and upper bound

			a_i
3.1	6.8	7.3	17.2
9.6	2.4	0.7	12.7
3.6	1.2	6.5	11.3
16.3	10.4	14.5	

(bottom row: b_j)