

Network Optimization

Optimization Problems on Networks

Network Models

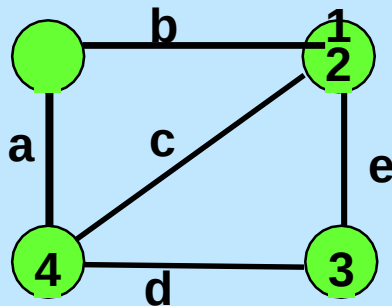
- Optimization models that exhibit a very special structure.
- For special cases, this structure to dramatically reduce computational complexity (running time).
- First widespread application of LP to problems of industrial logistics
- Addresses huge number of diverse applications

Network Flows and Routing Models

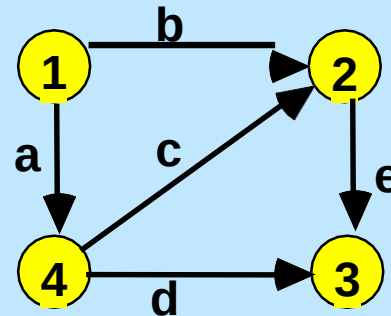
- **Phone network:** route calls, messages, data
- **Power network:** transmit power from power plants to consumers
- **Highway, street, rail, airline, shipping networks:** transport people, vehicles, goods
- **Computer networks:** transmit info, data, messages
- **Pipeline networks:** transport crude oil, gasoline
- **Satellite networks:** worldwide communication system
- **Abstract networks:** using networks to model relations

Notation and Terminology

Network terminology as used in AMO.



An Undirected Graph or
Undirected Network



A Directed Graph or
Directed Network

Network $G = (N, A)$

Node set $N = \{1, 2, 3, 4\}$

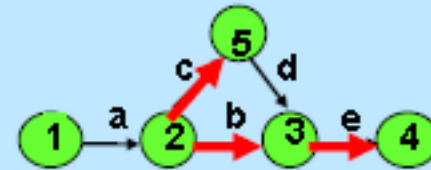
Arc Set $A = \{(1,2), (1,4), (3,2), (3,4), (2,4)\}$

In an undirected graph, $(i,j) = (j,i)$

Path: Example: 5, 2, 3, 4.

(or 5, c, 2, b, 3, e, 4)

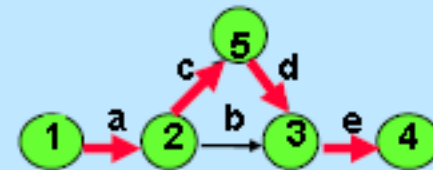
- No node is repeated.
- Directions are ignored.



Directed Path . Example: 1, 2, 5, 3, 4

(or 1, a, 2, c, 5, d, 3, e, 4)

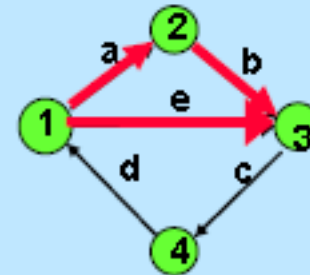
- No node is repeated.
- Directions are important.



Cycle (or circuit or loop)

1, 2, 3, 1. (or 1, a, 2, b, 3, e)

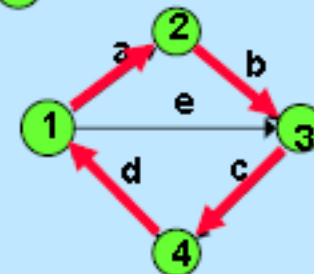
- A path with 2 or more nodes, except that the first node is the last node.
- Directions are ignored.



Directed Cycle: (1, 2, 3, 4, 1) or

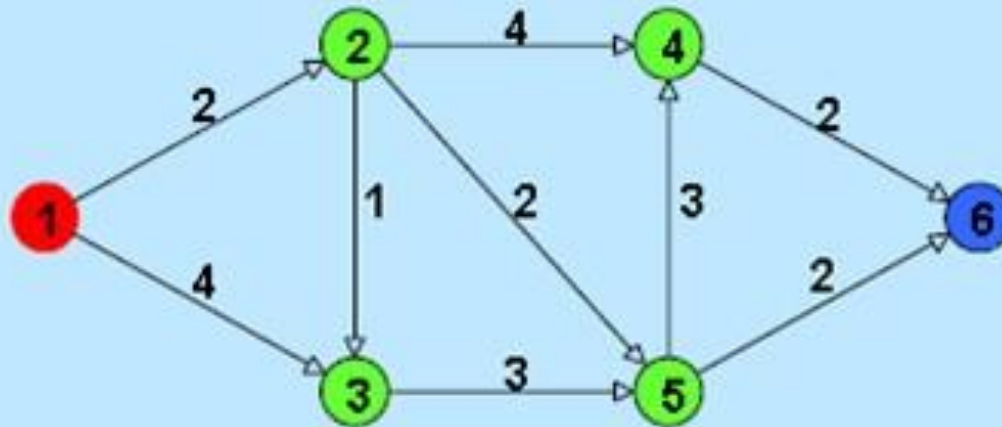
1, a, 2, b, 3, c, 4, d, 1

- No node is repeated, except that the first node is the last node.
- Directions are important.



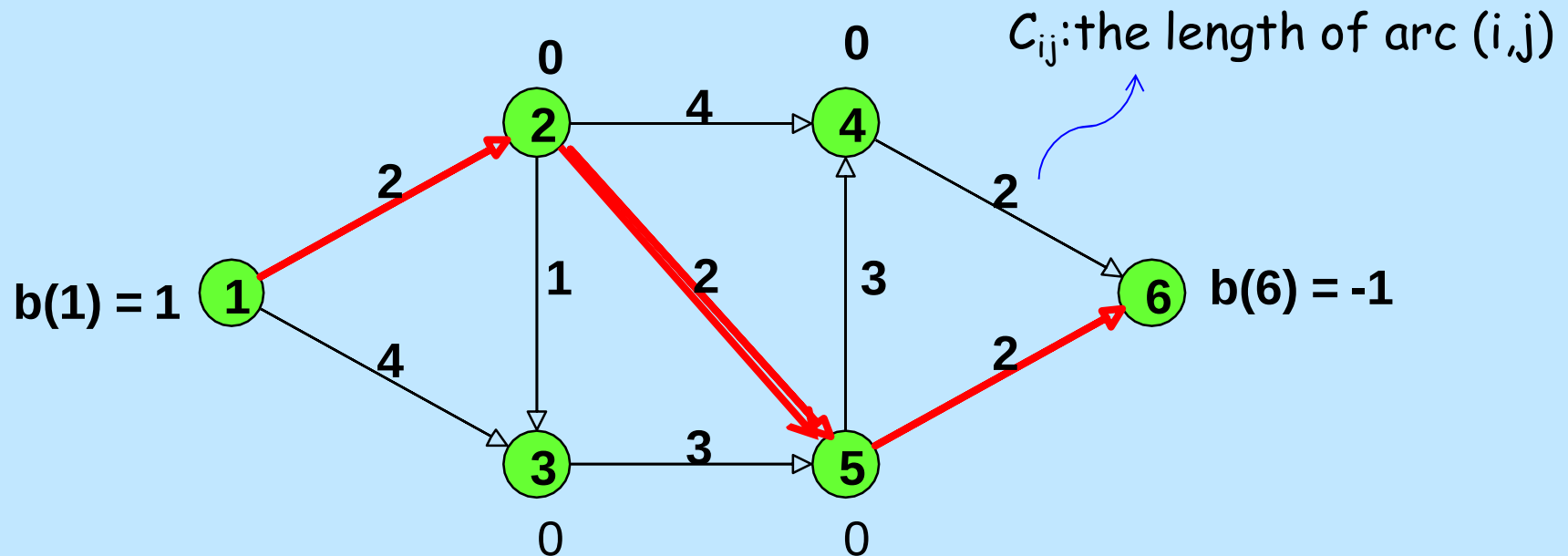
Shortest Paths

- ✚ Single Pair Shortest Path Problem
- ✚ Single Source Shortest Path Problem
- ✚ All-Pairs Shortest Path Problem



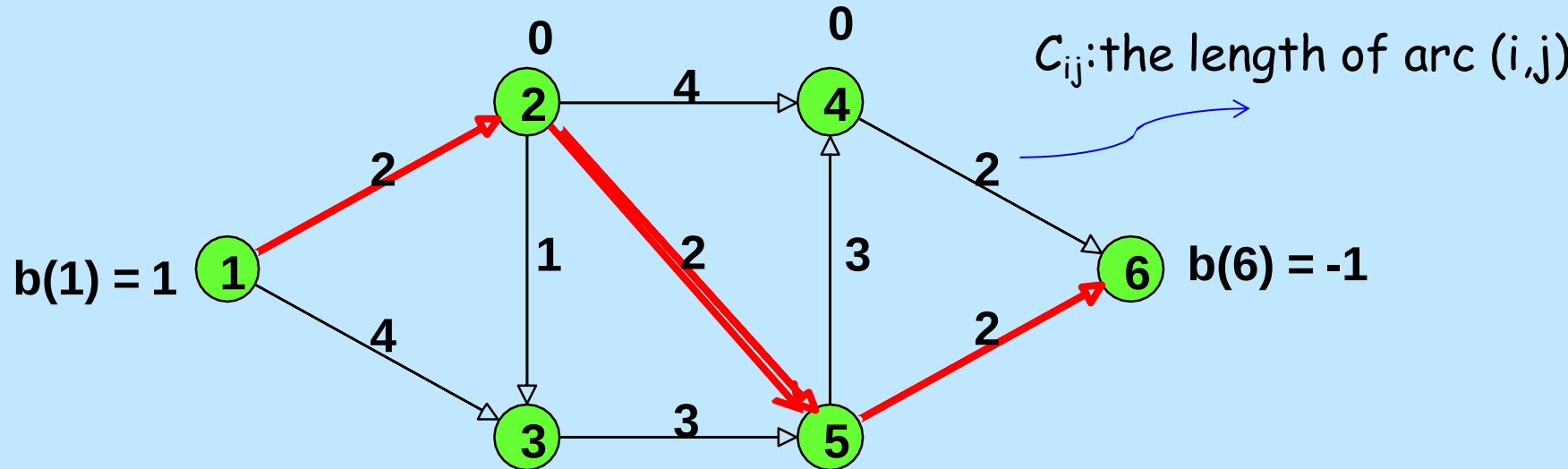
Length of (directed) path = total cost of arcs on this path

Find the shortest path from source node 1 to sink node 6
(single-pair shortest path problem)



- ✚ Think of sending one unit of flow from 1 to 6
- ✚ This transformation works so long as there are no negative cost cycles in G .
- ✚ What if there are **negative cost cycles**?

LP Formulation of the Shortest Path Problem



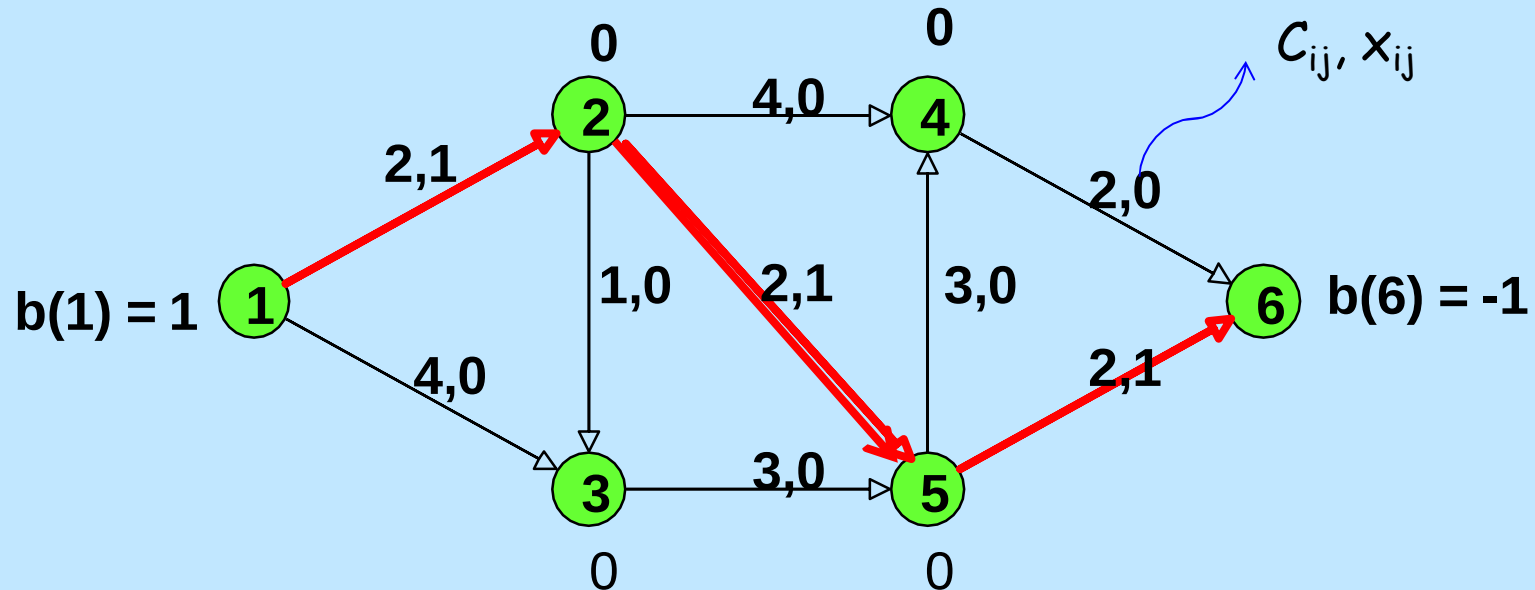
Find the shortest
path from node 1
to node 6

$$\text{Min} \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = \begin{cases} 1 & i = s \\ 0 & i \neq s, t \\ -1 & i = t \end{cases}$$

$$x_{ij} \geq 0, \text{ integer} \quad (i, j) \in A$$

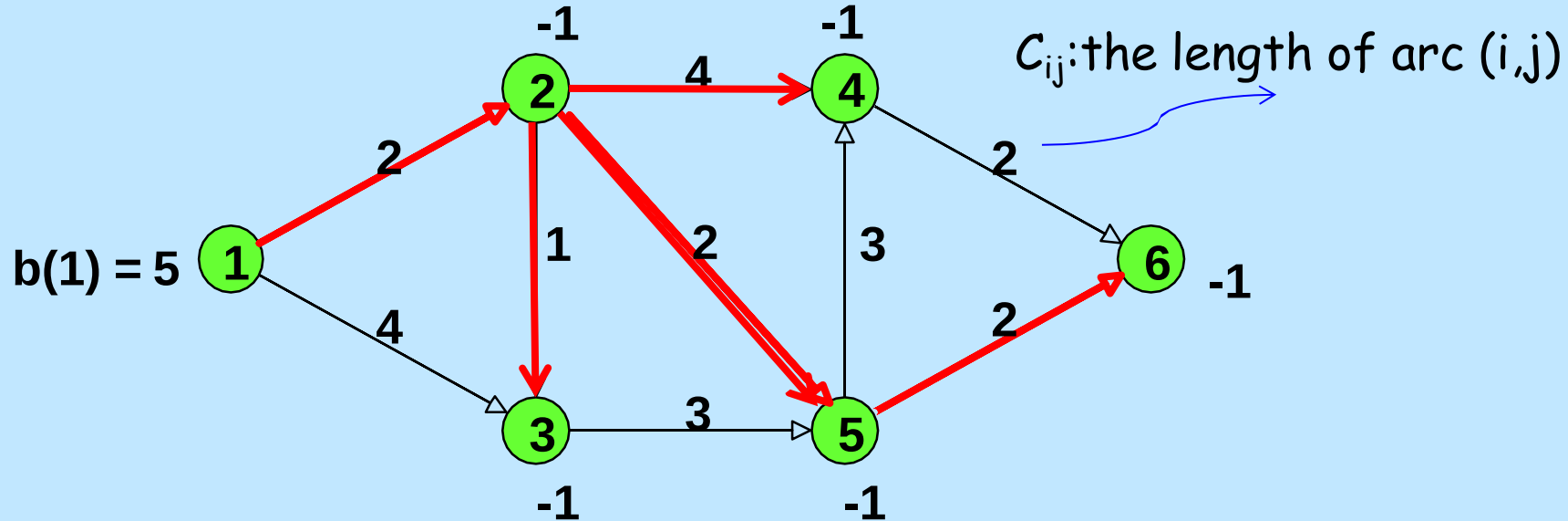
Find the shortest path from source node 1 to sink node 6
(single-pair shortest path problem)



- ✚ Optimal flow is to send one unit of flow along 1-2-5-6.
- ✚ $X_{12}=1; X_{25}=1, X_{56}=1, X_{13}=X_{23}=X_{24}=X_{35}=X_{45}=X_{46}=0$
- ✚ Obj.fcn. Value= $2.1+2.1+2.1=6$

✚ C_{ij} : the length of arc (i,j)
 x_{ij} : the amount of flow on
 arc (i,j)

Find the shortest path from source node 1 to all other nodes
(single-source shortest path problem)



- Each node except the source node is treated as a sink node
- Assume that a source of $n-1$ units is at node 1 and a demand of 1 unit at all other nodes

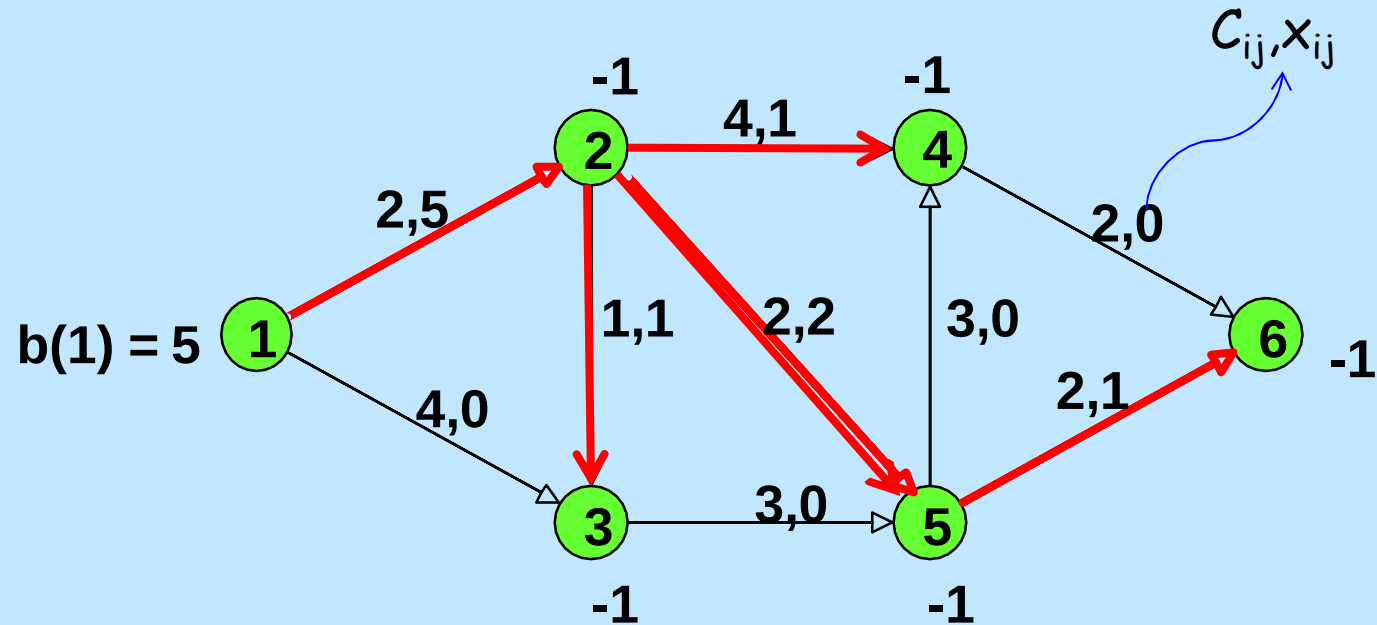
LP Formulation
of the single source
Shortest Path
Problem

$$\text{Min} \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$s.t. \quad \sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = \begin{cases} n-1 & i = s \\ -1 & i \neq s \end{cases}$$

$$x_{ij} \geq 0, \text{ integer} \quad (i,j) \in A$$

Find the shortest path from source node 1 to all other nodes
(single-source shortest path problem)



Obj.fcn value = $2.5 + 2.2 + 1.1 + 4.1 + 2.1 = 21$

Integrality Property

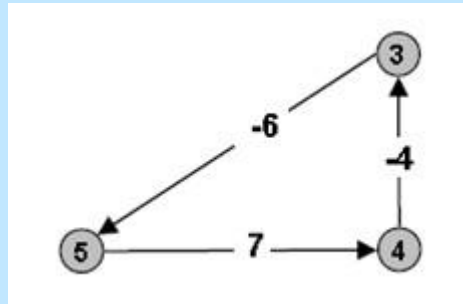
- ✚ If we relax the integrality requirement and solve as an LP, we get an integral solution and the arcs with positive x_{ij} values give a shortest path tree rooted at s .
- ✚ The constraint matrix has a special property called "total unimodularity". Any extreme point is an integer vector and hence the optimal solution is integral.
- ✚ In general, network flow problems have this special "integrality property".

Total Unimodularity

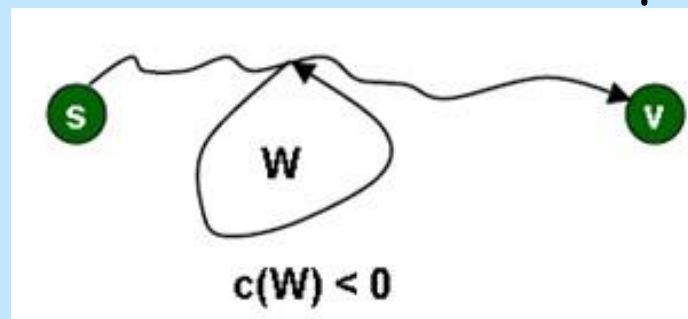
- ✚ A matrix A is called *totally unimodular* if each of its subdeterminants equals 0, 1, or -1. (In particular, each entry of A is 0, 1, or -1.)
- ✚ If A is totally unimodular and b is integer valued, then the polyhedron $P = \{x \mid Ax \leq b\}$ is integral.

Shortest Paths with Negative Directed Cost Cycles

+ A **directed cycle** is a path that begins and ends at the same node and a **negative directed cycle** is a directed cycle of negative total length

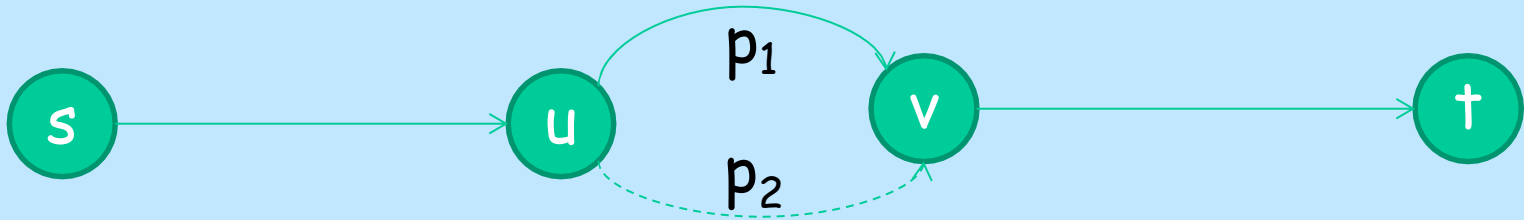


+ If some path from s to v contains a negative directed cost cycle, there does not exist a shortest s - v path, otherwise, there exists one that is simple



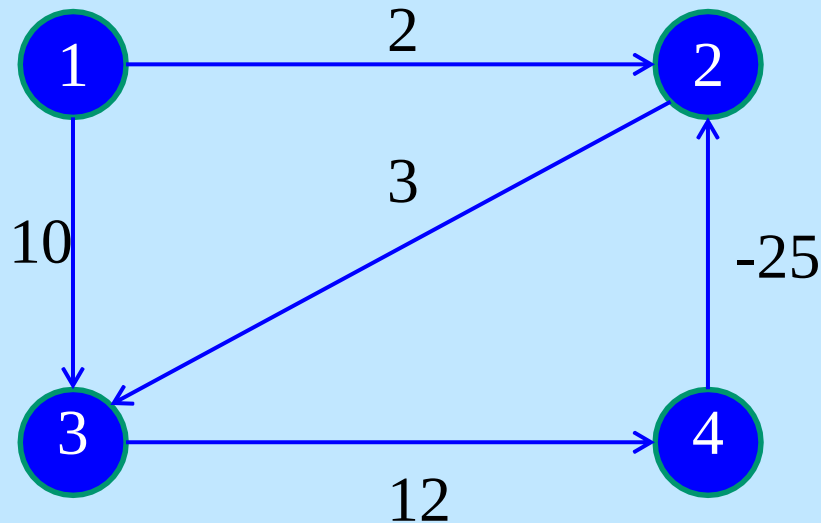
Principle of Optimality

In a network with no negative directed cycles, if P is a shortest path from node s to node t and if it goes through nodes u and v , then the subpath of P from u to v is a shortest path from u to v .



If the subpath p_2 is shorter than the subpath p_1 from u to v , then the shortest path from s to v cannot use the subpath p_1 from u to v .

Shortest Paths with Negative Cost Cycles



- + Shortest path from 1 to 3: 1-2-3 with length 5
- + Shortest path from 1 to 2: 1-3-4-2 with length -3
- + The shortest path from 1 to 3 does not use the shortest subpath from node 1 to node 2 (it should have been 1-3-4-2-3, which is not a path)

Shortest path problems in the presence of negative directed cycles are much less tractable!

Dynamic Programming Approach to Shortest Path Problems

- ✚ Divide and conquer method
- ✚ Dynamic programming exploits the fact that it is sometimes easier to solve one optimization problem by taking on an entire family
- ✚ If strong relationships can be found among optimal solutions to the various members of the family, attacking all together can be the most efficient way of solving the one(s) we really care about

Dynamic Programming Approach to Shortest Path Problems

- ✚ Shortest path algorithms exploit relationships among optimal paths (and path lengths) for different pairs of nodes

Functional Notation

For Single Source Shortest Path Problem

- + $v[k]$: the length of a shortest path from the source node to node k
 - + If no path exists, $v[k]=\infty$

Functional Equations

- ✚ If optimal paths must have optimal subpaths, it follows that a shortest path can be constructed by extending known shortest paths
- ✚ Functional equations of a dynamic program encode the recursive connections among optimal solution values that are implied by a principle of optimality
 - ✚ In a graph with no negative directed cycles, optimal paths must have optimal subpaths

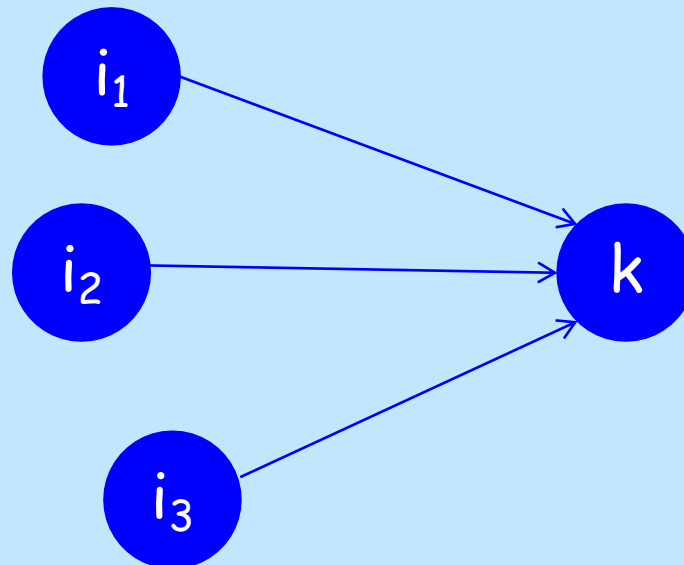
Functional Equations

$$v[s]=0$$

$$v[k]=\min_{i:(i,k)\in A} \{v[i]+c_{ik}\} \quad "k \in N \setminus \{s\}$$

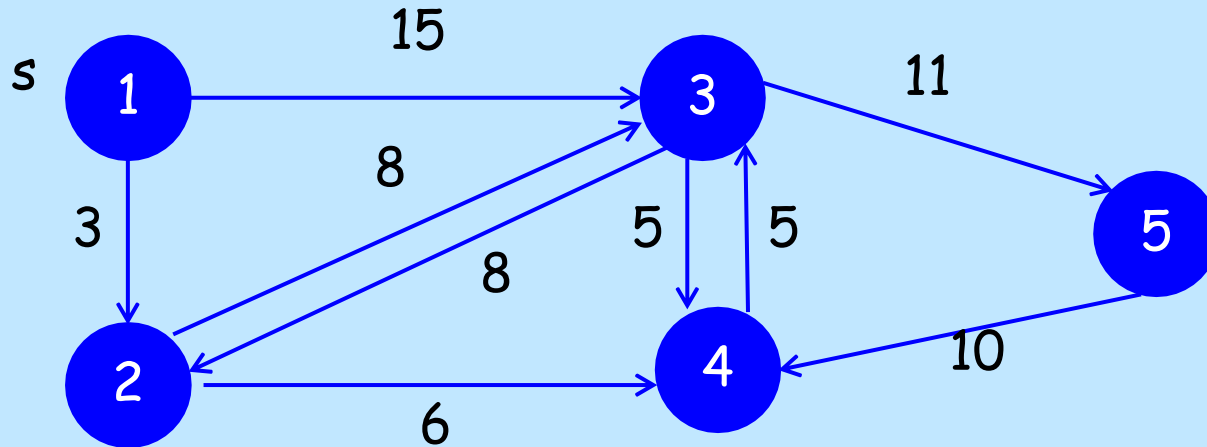
If there is no neighbor i leading to k , then the minimum is ∞

$d[k]$ =preceeding node on the shortest path from s to k



Functional Equations

Example



$$v[1]=0$$

$$v[2]=\min \{v[1]+3, v[3]+8\}=3$$

$$v[3]=\min \{v[1]+15, v[2]+8, v[4]+5, v[4]+5, v[5]+11\}=11$$

$$v[4]=\min \{v[2]+6, v[3]+5, v[5]+10\}=9$$

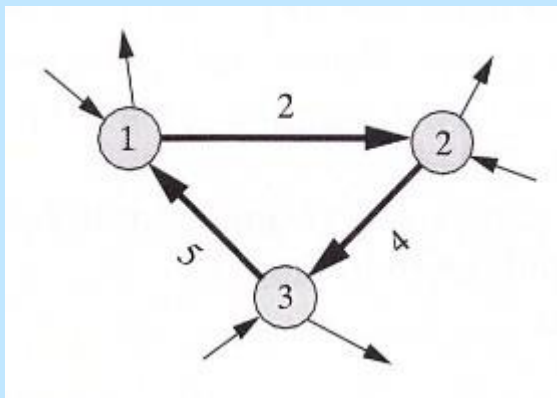
$$V[5]=\infty$$

Optimal paths must extend optimal subpaths

Functional Equations

How to solve functional equations

- ✚ Nonlinear due to min operator
- ✚ Because each equation shows an expression for a single value $v[k]$, possible just to evaluate those expressions (one-pass evaluation).
- ✚ However, **directed cycles** introduce circular dependencies in functional equations that preclude their solution by one-pass evaluations



$$v[1] = \min\{v[3] + 5, \dots\}$$

$$v[2] = \min\{v[1] + 2, \dots\}$$

$$v[3] = \min\{v[2] + 4, \dots\}$$

Evaluating the expression for $v[2]$ requires $v[1]$; evaluating the expression for $v[3]$ requires $v[2]$; and evaluating the expression for $v[1]$ requires $v[3]$.

Bellman-Ford Algorithm

Repeated Evaluation Algorithm for Single Source Shortest Path Problem

✚ Each major iteration of the search evaluates the functional equations for each $v[k]$ using results from the preceding iteration. The algorithm stops, when the results do not change.

✚ $v^{(t)}[k]$: value of $v[k]$ obtained on the t^{th} iteration

✚ To be able to know the shortest path at the termination of the algorithm, labels $d[k]$ keep notes to allow us to recover the optimal paths

✚ $d[k]$: node preceding k in the best known path from s to k

Bellman-Ford Algorithm

Step 0: Initialization. With s the source node, initialize optimal path lengths

$$v^{(0)}[k] \leftarrow \begin{cases} 0 & \text{if } k = s \\ +\infty & \text{otherwise} \end{cases}$$

and set iteration counter $t \leftarrow 1$.

Step 1: Evaluation. For each k evaluate

$$v^{(t)}[k] \leftarrow \min\{v^{(t-1)}[i] + c_{i,k} : (i, k) \text{ exists}\}$$

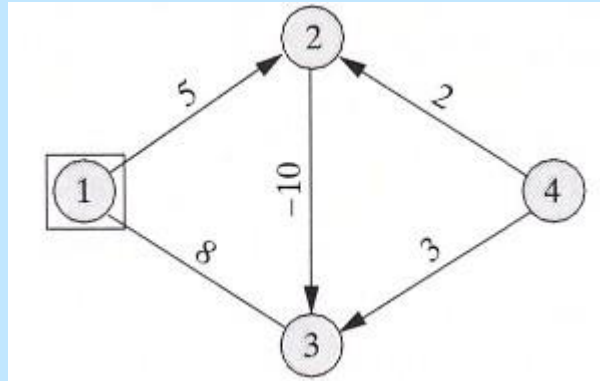
If $v^{(t)}[k] < v^{(t-1)}[k]$, also set $d[k] \leftarrow$ the number of a neighboring node i achieving the minimum $v^{(t)}[k]$.

Step 2: Stopping. Terminate if $v^{(t)}[k] = v^{(t-1)}[k]$ for every k , or if $t =$ the number of nodes in the graph. Values $v^{(t)}[k]$ then equal the required shortest path lengths unless some $v^{(t)}[k]$ changed at the last t , in which case the graph contains a negative dicycle.

Step 3: Advance. If some $v[k]$ changed and $t <$ the number of nodes, increment $t \leftarrow t + 1$ and return to Step 1.

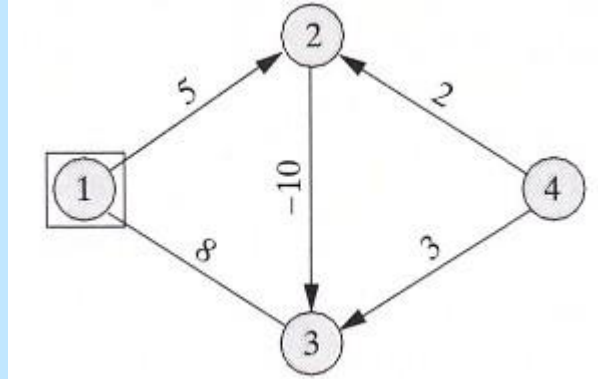
The algorithm works on any graph with no negative directed cycles

Bellman-Ford Algorithm



t	$v^t[1]$	$v^t[2]$	$v^t[3]$	$v^t[4]$		d(1)	d(2)	d(3)	d(4)
0									
1									
2									
3									

Bellman-Ford Algorithm



t	$v^{(t)}[1]$	$v^{(t)}[2]$	$v^{(t)}[3]$	$v^{(t)}[4]$
0	0	$+\infty$	$+\infty$	$+\infty$
1		5	8	
2			-5	
3				

t	$d[1]$	$d[2]$	$d[3]$	$d[4]$
1		1	1	
2			2	
3				

✚ $v[1]=0; v[2]=0; v[3]=-5; v[4]=\infty$

✚ A shortest path from source s to any other node k can be recovered by starting at k , backtracking to neighboring node $d[k]$, and continuing with an optimal path from s to the neighbor until source is encountered

✚ Paths: 1-2; 1-2-3

Bellman-Ford Algorithm

Why does the algorithm work?

- ✚ Remark that in a graph with n nodes, a path can contain at most $n-1$ arcs
- ✚ At the t^{th} iteration, the length of optimal paths of at most t arcs should be correct
- ✚ Moreover, if we have a whole iteration without changing any node's value, no more changes can occur

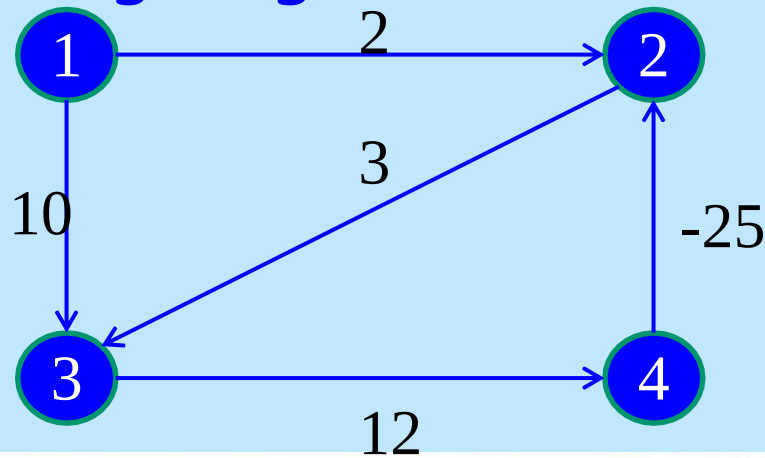
Bellman-Ford Algorithm

Encountering Negative Directed Cycles

- ✚ If the algorithm encounters negative directed cycles, it demonstrates their presence by continuing to change $v^{(t)}[k]$ on iteration $t = \text{the number of nodes}$. Any node k with such a changing $v^{(t)}[k]$ belongs to a negative directed cycle

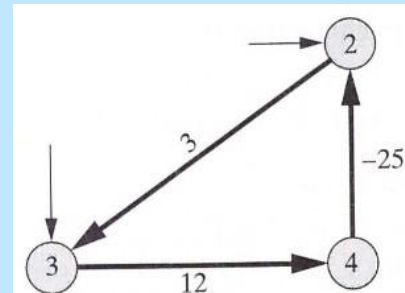
Bellman-Ford Algorithm

Encountering Negative Directed Cycles



t	$v^{(t)}[1]$	$v^{(t)}[2]$	$v^{(t)}[3]$	$v^{(t)}[4]$	$d[1]$	$d[2]$	$d[3]$	$d[4]$
0	0	$+\infty$	$+\infty$	$+\infty$				
1		2	10			1	1	
2			5	22			2	3
3		-3		17		4		
4		-8	0					

$$\begin{aligned}
 v^{(t)}[2] &\leftarrow v^{(t-1)}[4] - 25 \\
 v^{(t)}[3] &\leftarrow v^{(t-1)}[2] + 3 \\
 v^{(t)}[4] &\leftarrow v^{(t-1)}[3] + 12
 \end{aligned}$$



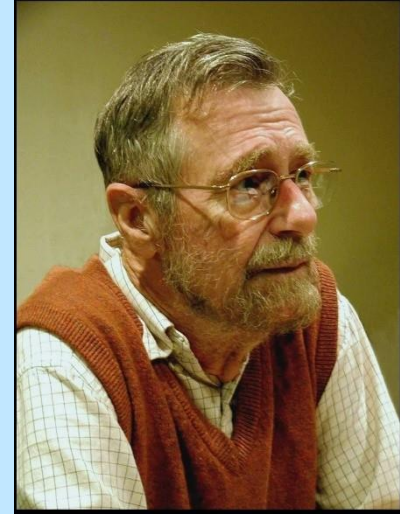
Single-Source Shortest Path Problem

When All Costs are Nonnegative

Dijkstra's Algorithm

- + More efficient than Bellman-Ford Algorithm
- + Directed negative cycles are not possible, so functional equations are valid
- + Suggests a different way of solving these equations
- + At any step, the nodes are divided into two sets
 - + Permanently labeled nodes correspond to shortest path lengths
 - + Temporarily labeled nodes correspond to path lengths using only permanently labeled nodes (so upper bounds on shortest path lengths)

Edsger W. Dijkstra: select quotes



Turing award 1972

“ The question of whether computers can think is like the question of whether submarines can swim. ”

“ Do only what only you can do. ”

“ In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind. ”

“ The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence. ”

“ APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums. ”

- $x[\Delta x \leftarrow 6?40]$ (lottery: 6 different no's, 1 through 40)
- Hated “GoTo” Statement!

Dijkstra's Algorithm

Step 0: Initialization. With s the source node, initialize optimal path lengths

$$v[i] \leftarrow \begin{cases} 0 & \text{if } i = s \\ +\infty & \text{otherwise} \end{cases}$$

Then mark all nodes temporary, and choose $p \leftarrow s$ as the next permanently labeled node.

Step 1: Processing. Mark node p permanent, and for every arc/edge (p, i) leading from p to a temporary node, update

$$v[i] \leftarrow \min\{v[i], v[p] + c_{p,i}\}$$

If $v[i]$ changed in value, also set $d[i] \leftarrow p$.

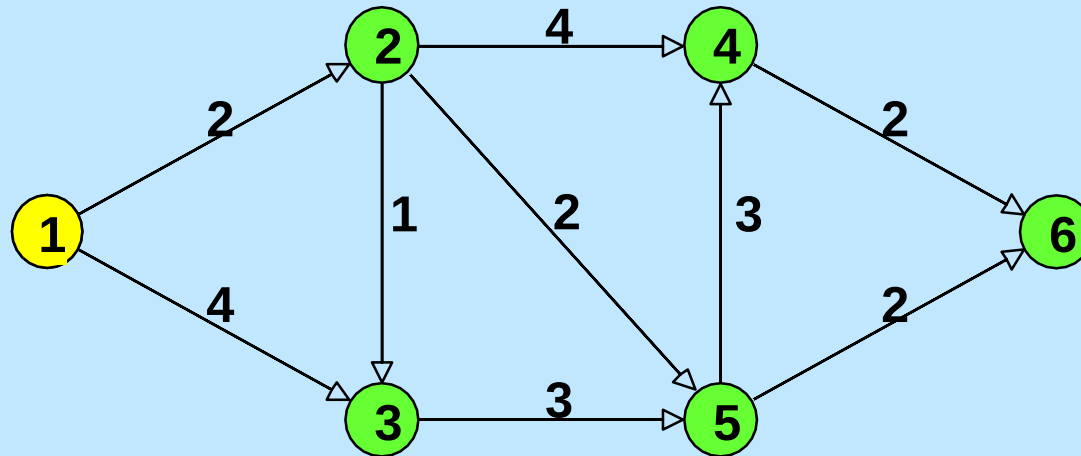
Step 2: Stopping. If no temporary nodes remain, stop; values $v[i]$ now reflect the required shortest path lengths.

Step 3: Next Permanent. Choose as next permanently labeled node p a temporary node with least current value $v[i]$, that is,

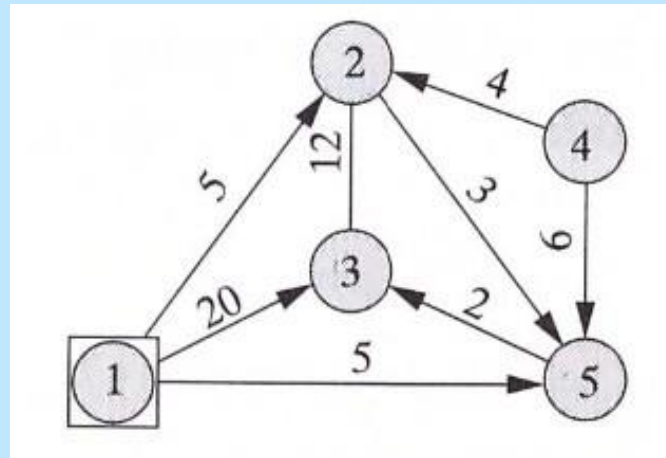
$$v[p] = \min\{v[i] : i \text{ temporary}\}$$

Then return to Step 1.

An Example



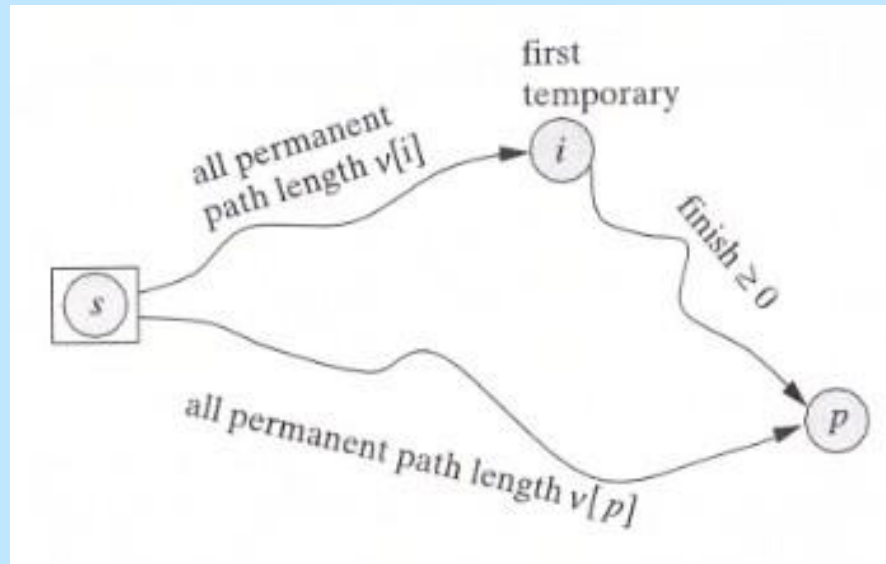
Dijkstra's Algorithm



p	$v[1]$	$v[2]$	$v[3]$	$v[4]$	$v[5]$
(init)	0	∞	∞	∞	∞
1	(perm)	5	20		5
2		(perm)	17		
5			7		(perm)
3			(perm)		
4				(perm)	
(final)	0	5	7	∞	5

p	$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$
1		1	1		1
2			2		
5			5		
3					
4					
(final)	—	1	5	—	1

Dijkstra's Algorithm



$$(s\text{-to-}i \text{ subpath}) + (i\text{-to-}p \text{ subpath}) \geq v[i] + 0$$

$$v[i] + 0 = v[i] \geq v[p]$$

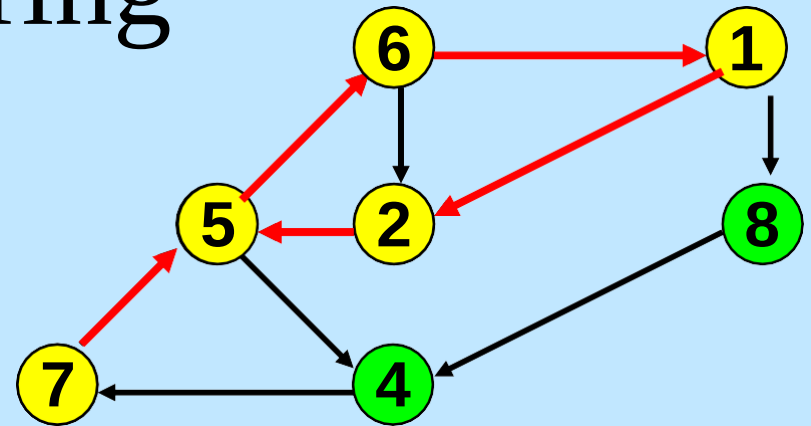
No path can be shorter than the one for $v[p]$

Single-Source Shortest Path Problem When Network is Acyclic

- ✚ Most efficient shortest path algorithm
- ✚ Works even if there are negative arc costs
- ✚ Label the nodes topologically (or lexicographically) such that if (i,j) is an arc $\text{label}(i) < \text{label}(j)$
- ✚ Solve the functional equations with respect to this ordering

Topological (Lexicographic) Ordering

LEMMA. If each node has at least one arc going out, then the network has a directed cycle.



COROLLARY 1. If G has no directed cycle, then there is a node in G with no arcs going. And there is at least one node in G with no arcs coming in.

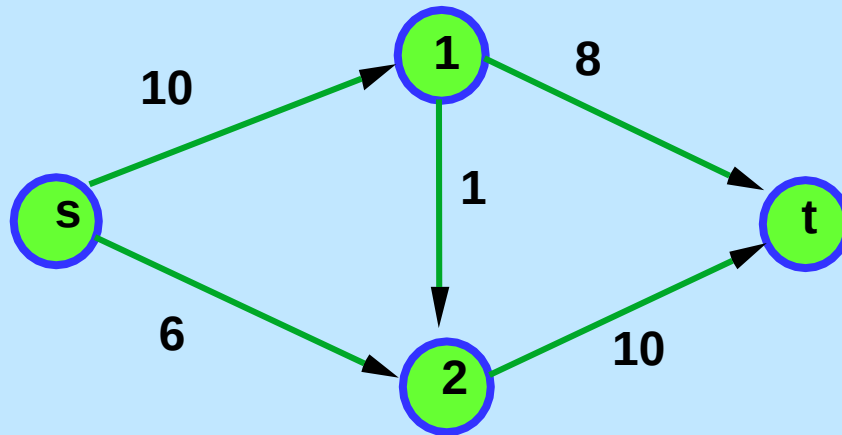
- **COROLLARY 2.** If G has no directed cycle, then one can relabel the nodes so that for each arc (i,j) , $i < j$.

Topological (lexicographic) ordering

- begin
 - next:=0;
 - while G contains a node with indegree=0 do
 - begin
 - pick any node i from G with indegree=0
 - next := next + 1;
 - order(i) := next;
 - remove node i and all its incident arcs
 - end
 - if next < n then the network contains a directed cycle
 - else the network is acyclic and the order is topological
- end

Maximum Flows

Given:



$G = (N, A)$

u_{ij} = capacity of flow in arc (i, j)

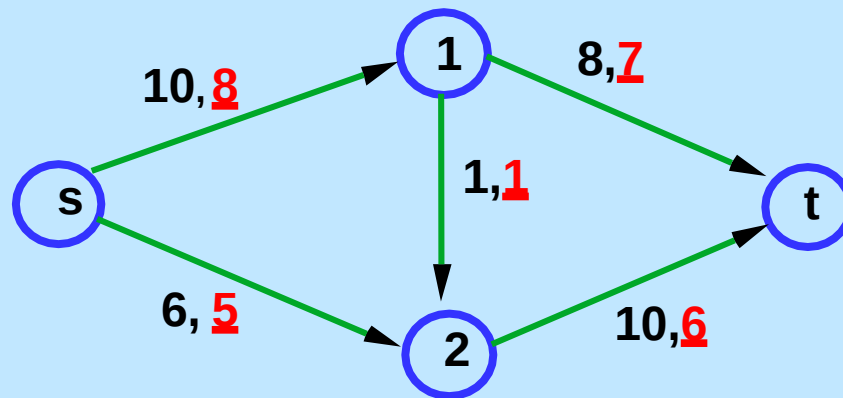
s = source node

t = sink node

Find the maximum amount of net flow out of node s into node t obeying the capacity limitations

Intermediate nodes should satisfy flow balance

We refer to a flow x as *maximum* if it is feasible and maximizes the net flow out of s . Our objective in the max flow problem is to find a maximum flow.



A max flow problem.

Capacities and a **non-optimum flow**.

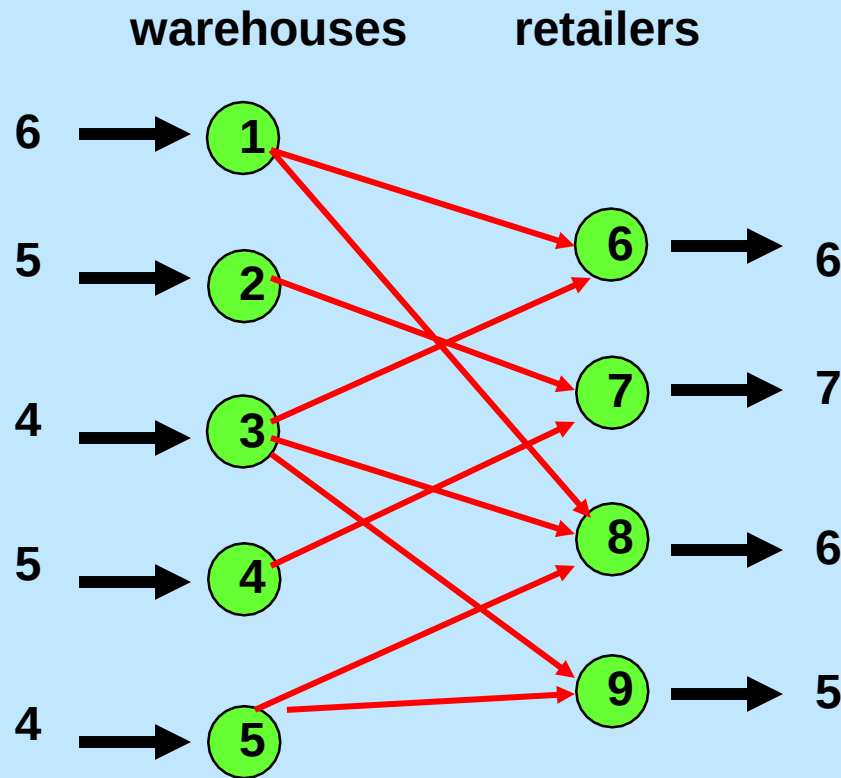
Decision Variables:

x_{ij} = flow on arc (i,j)

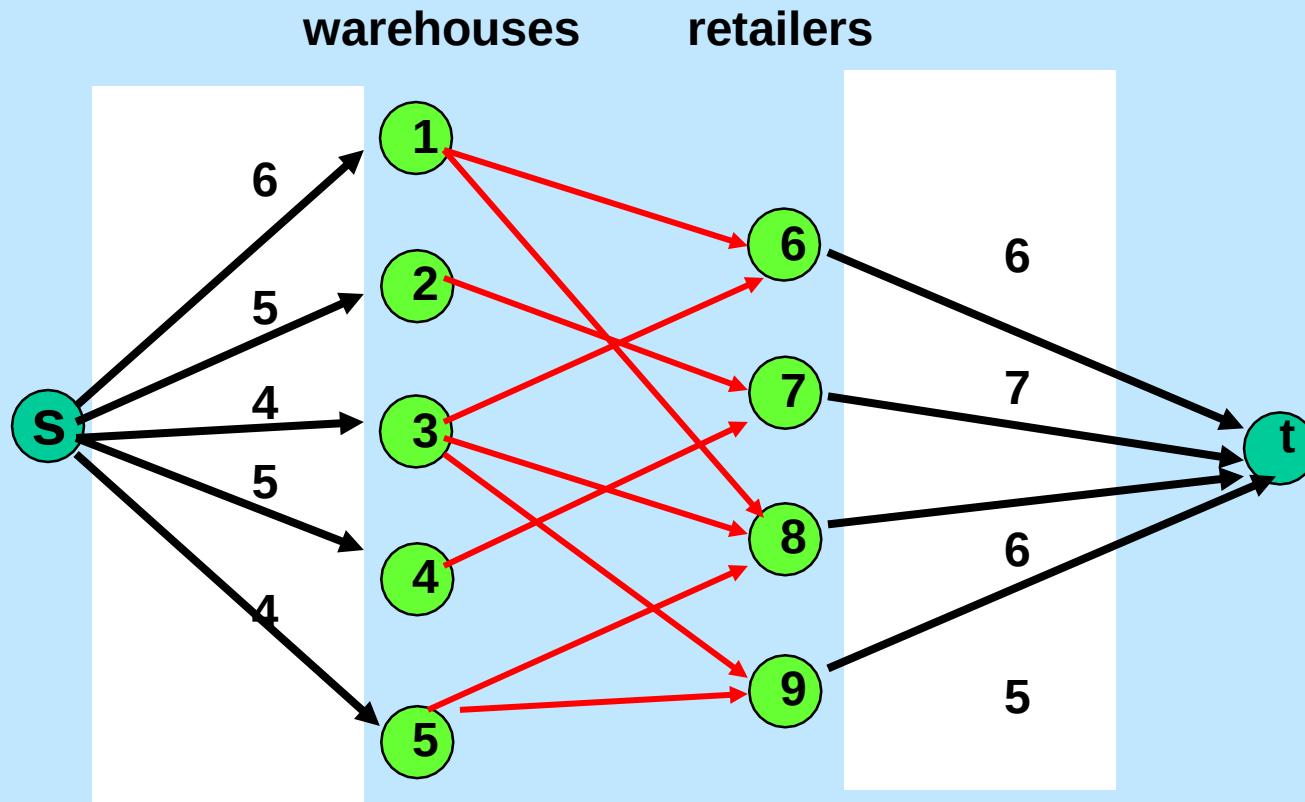
v = net flow out of s (into t)

Model:

The feasibility problem: find a feasible flow



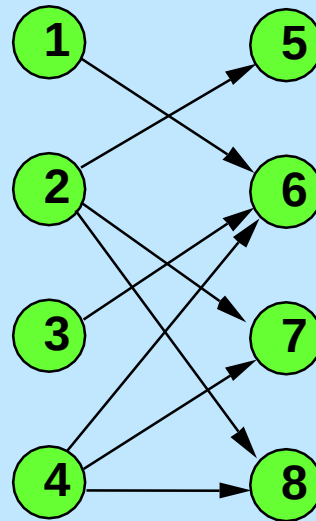
Transformation to a max flow problem



There is a 1-1 correspondence with flows from s to t with 24 units (why 24?) and feasible flows for the transportation problem.

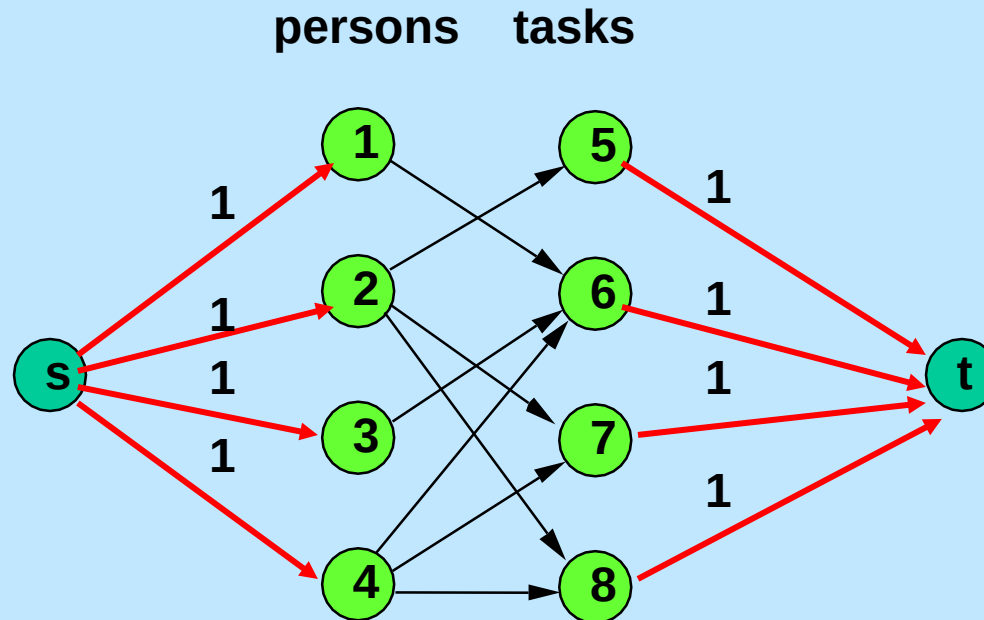
The feasibility problem: find a matching

persons tasks



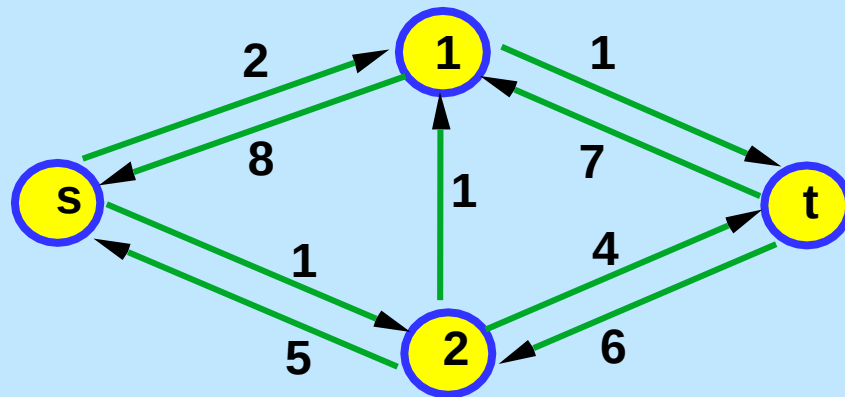
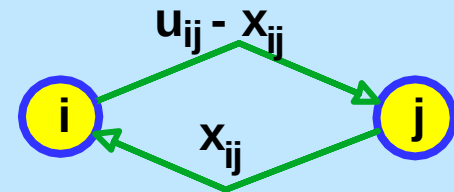
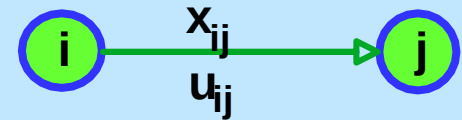
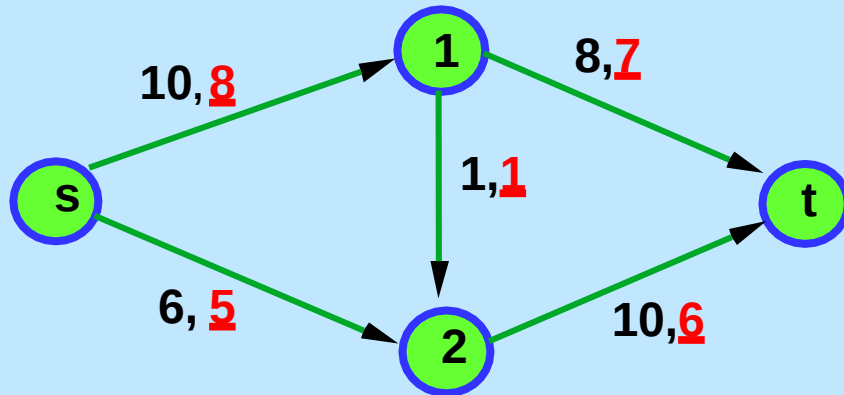
Is there a way of assigning persons to tasks so that each person is assigned a task, and each task has a person assigned to it?

Transformation to a maximum flow problem



Does the maximum flow from s to t have 4 units?

The Residual Network



The **Residual Network** $G(x)$

We let r_{ij} denote the **residual capacity** of arc (i,j)

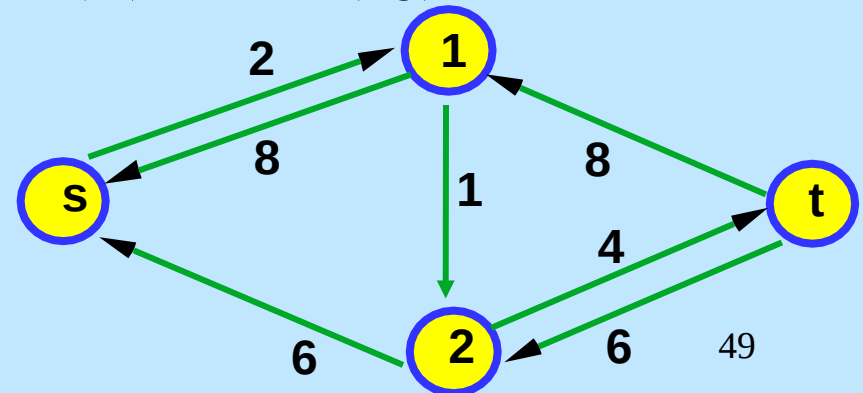
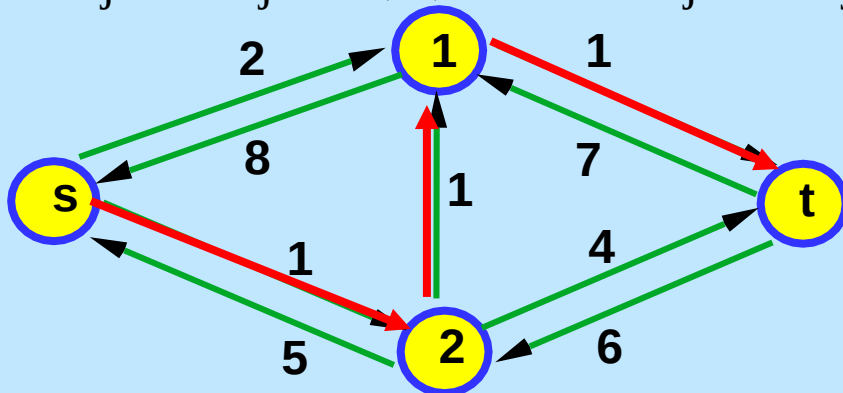
A Useful Idea: Augmenting Paths

An *augmenting path* is a path from s to t in the residual network.

The *residual capacity* of the augmenting path P is
$$\delta(P) = \min\{r_{ij} : (i,j) \in P\}.$$

To *augment along P* is to send $\delta(P)$ units of flow along each arc of the path. We modify x and the residual capacities appropriately.

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



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 $G(x)$;

$\Delta := \delta(P)$;

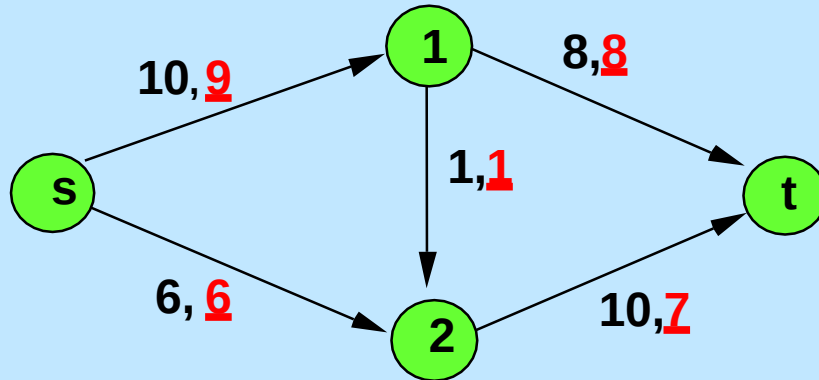
send Δ units of flow along P ;

update the r 's;

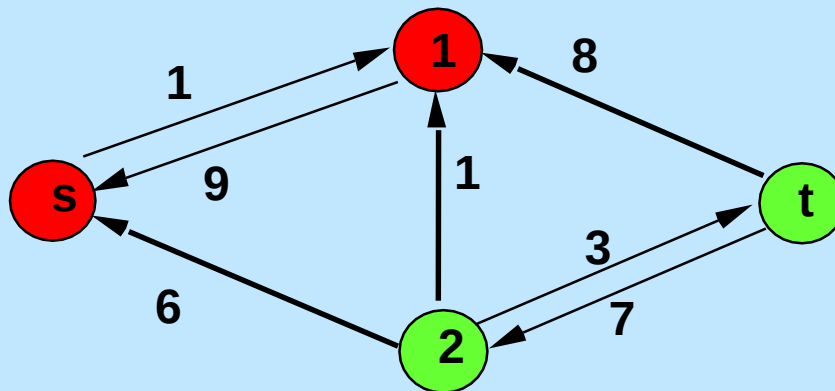
end

end {the flow x is now maximum}.

How do we know when a flow is optimal?

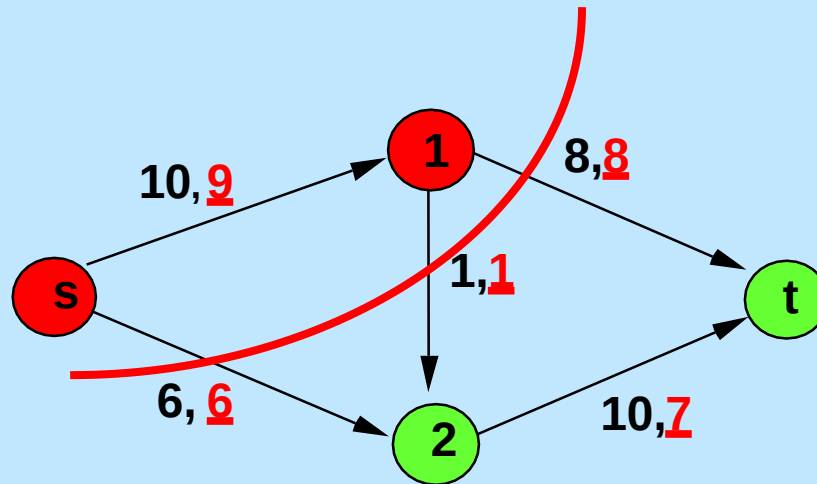


There is no augmenting path in the residual network.



- reachable from s in $G(x)$
- not reachable from s in $G(x)$

(s,t)-cuts



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

The **capacity** of a cut (S,T) is

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

Observation: If x is any feasible flow and if (S,T) is an (s,t) -cut, then the flow $v(x)$ from source to sink in x is at most $CAP(S,T)$.

In other words, capacity of any cut is an upper bound on the value of any feasible flow

Max Flow Min Cut Theorem

Theorem. (Optimality conditions for max flows)

A flow x is maximum

if and only if

there is no augmenting path in $G(x)$.

Proof

Suppose that there is an augmenting path in $G(x)$. Then x is not maximum.

Suppose there is no augmenting path in $G(x)$.

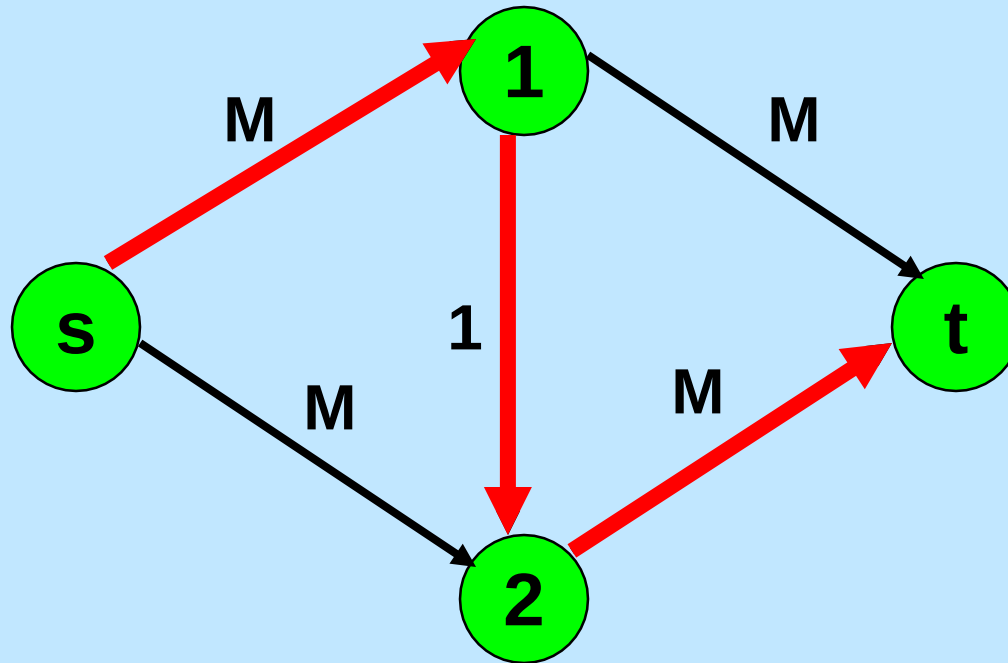
Claim: Let S be the set of nodes reachable from s in $G(x)$. Let $T = N \setminus S$. Then there is no arc in $G(x)$ from S to T .

Thus,

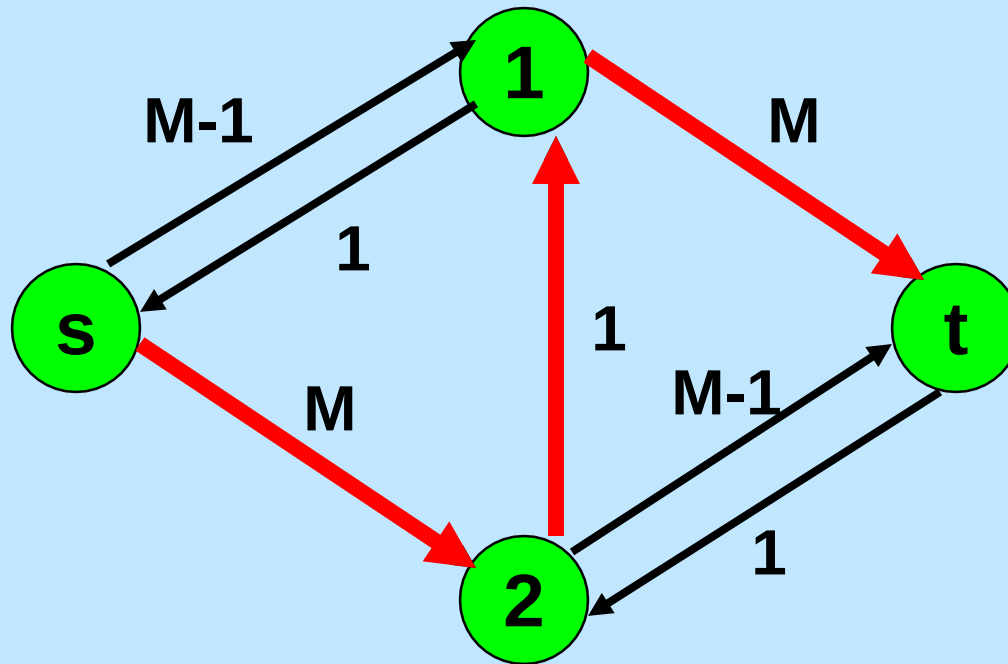
$$\begin{array}{llll} i \in S & \text{and } j \in T & \Rightarrow & x_{ij} = u_{ij} \\ i \in T & \text{and } j \in S & \Rightarrow & x_{ij} = 0. \end{array}$$

Thus flow across this cut = $\text{cap}(S, T)$ and thus is optimal!

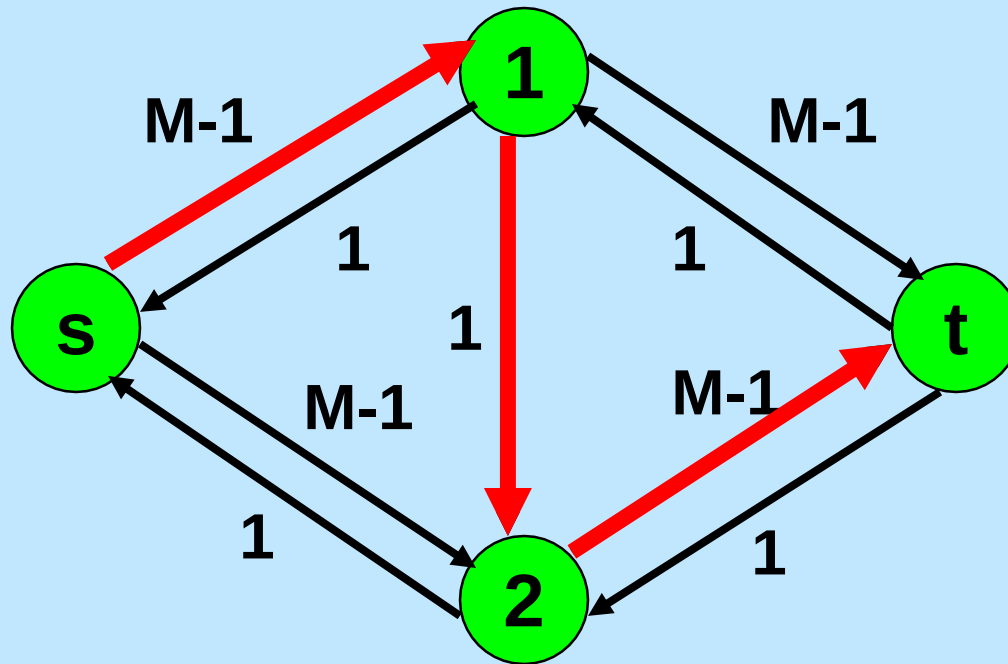
A simple and very bad example



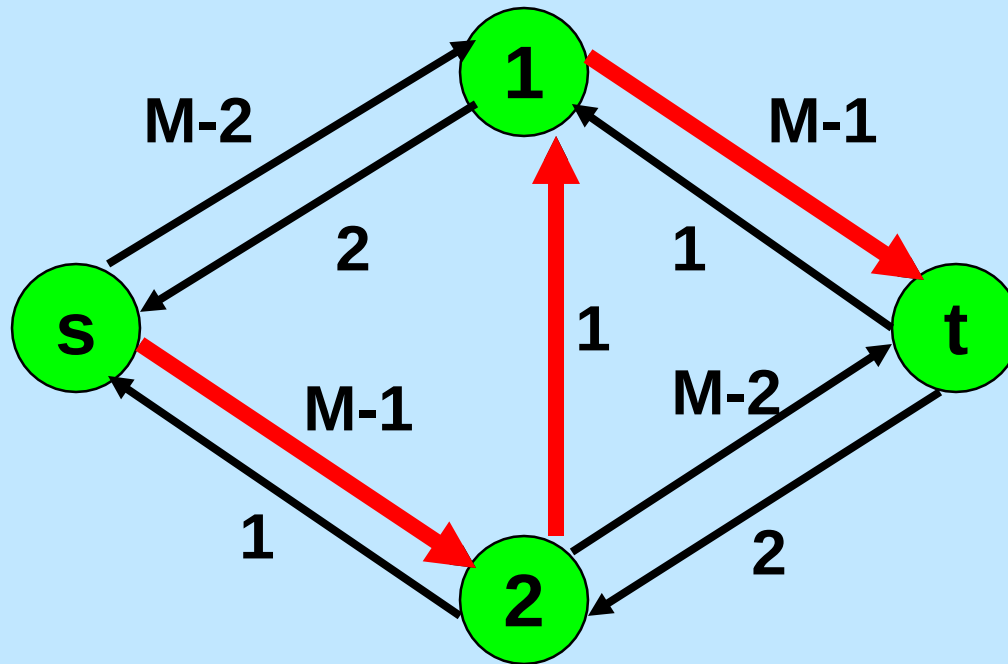
After 1 augmentation



After two augmentations



After 3 augmentations



And so on



After $2M$ augmentations

