Chapter 5

Network Flow Problems

5.1 Graph Theory: Basics and Definitions

Directed Graph (Digraph)

A weighted directed graph (digraph) $G = (V, A, c)$ is defined by

- a set of nodes V,
- a set of arcs A where each arc (i,j) is defined by a start node i and a terminal node j and
- for each arc (i,j) a weight (cost) c_{ij}.

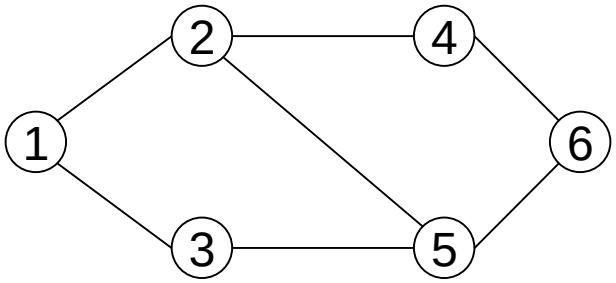
Example:

Undirected Graph

An undirected graph $G = [V, A, c]$ is defined

- by a set of nodes V,
- a set of edges A where each edge $[i, j]$ is defined by node i and node j and
- a weight (cost) c_{ij} fore each edge $[i, j]$.

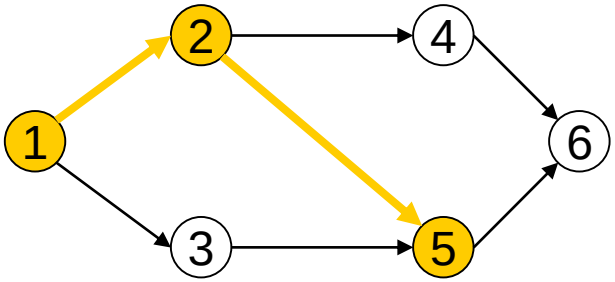
Example:



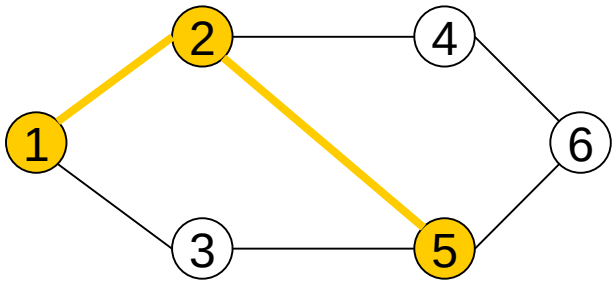
Path

A path in a directed or undirected graph is a sequence of $k \geq 2$ nodes $(n_1, n_2, \ldots, n_k)$ and associated $k - 1$ arcs $((n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, nk))$.

Example:



Directed Path



Undirected Path

5.2 Minimum Cost Flow Problems

Minimum Cost Flow (MCF): Introduction

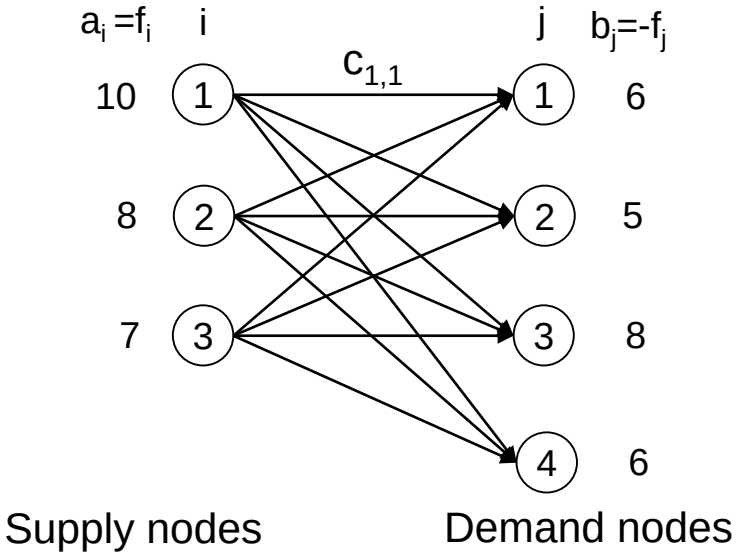
- Given digraph G=(V,A,c)
- For each node j $\in$ V we have net flow (outflow – inflow) $f_j \in \mathbb{R}$
  - $f_j > 0$ equals supply f_j of node j (supply node)
  - $f_j < 0$ equals demand $|f_j|$ of node j (demand node)
  - $f_j = 0$ implies that inflow = outflow (transshipment node)

- For each arc (i,j) $\in$ A we have
  - variable cost $c_{i,j}$ per transported unit
  - maximum flow $\kappa_{i,j}$
  - minimum flow $\lambda_{i,j}$
- We want to determine the flow $x_{i,j}$ for each arc (i,j) $\in$ A such that supply nodes serve the demand nodes at minimal cost.

Linear Program of the MCF Problem

5.2.1 The Transportation Problem

Transportation Problem (TPP)

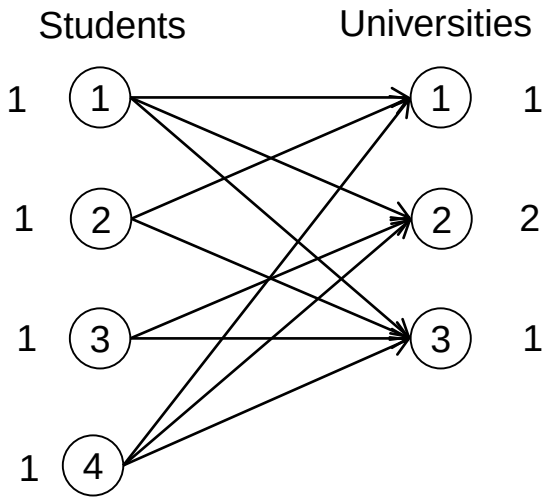


$$(c) = \begin{pmatrix} 7 & 2 & 4 & 7 \\ 9 & 5 & 3 & 3 \\ 7 & 7 & 6 & 4 \end{pmatrix}$$

Application of TPP: Assignment of Students to Exchange Places

- a_i=1, if student i (i=1,…,m) requests an exchange place
- $b_j \in Z^+$ number of exchange places of university j (j=1,…,n)
- $-c_{i,j}$ is the preference of student i for university j
- → Objective: Maximize the sum of the preferences

Example: Assignment of Students to Exchange Places

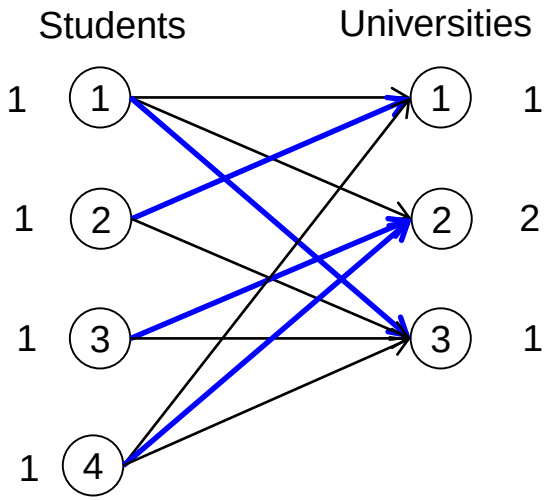


$$(c) = \begin{pmatrix} -8 & -2 & 0 \\ -10 & 0 & 0 \\ 0 & -10 & 0 \\ -5 & -5 & 0 \end{pmatrix}$$

LINDO-Model

```
Min -8x11 - 2x12 - 10x21 - 10x32 - 5x41 - 5x42
st
student1) x11 + x12 + x13 = 1
student2) x21 + x22 + x23 = 1
student3) x31 + x32 + x33 = 1
student4) x41 + x42 + x43 = 1
uni1) x11 + x21 + x31 + x41 = 1
uni2) x12 + x22 + x32 + x42 = 2
uni3) x13 + x23 + x33 + x43 = 1
end
```

Optimal Solution of the Exchange Place Assignment Problem

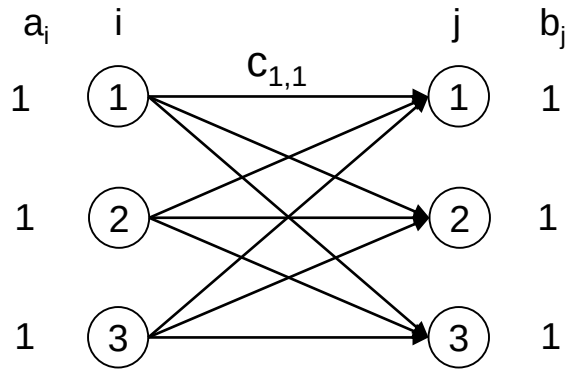


$$(c) = \begin{pmatrix} -8 & -2 & 0 \\ -10 & 0 & 0 \\ 0 & -10 & 0 \\ -5 & -5 & 0 \end{pmatrix}$$

Optimal objective function value: $z^*=-25$

5.2.2 The Assignment Problem

Assignment Problem



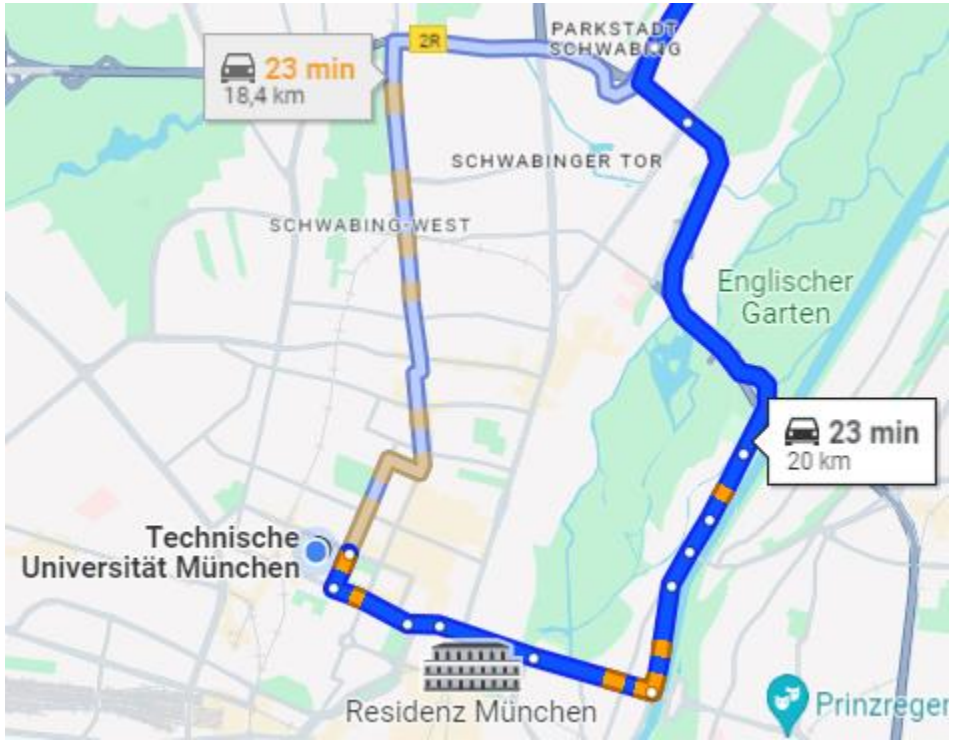
Supply nodes Demand nodes

Assignment Problem: Applications

- Assignment of jobs to workers
  - in case of external workers $c_{i,j}$ equals the costs
  - in case of internal workers $c_{i,j}$ equals the duration
  - → Objective: Minimize cost / Minimize the total duration spent for performing the jobs

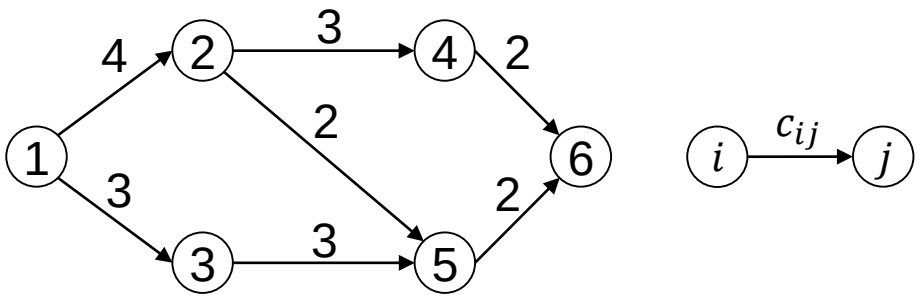
- Assignment of seminar topics to students
  - $-c_{i,j}$ equals the preference of student i for topic j
  - → Objective: Maximize the sum of preferences

5.2.3 Shortest Path Problems



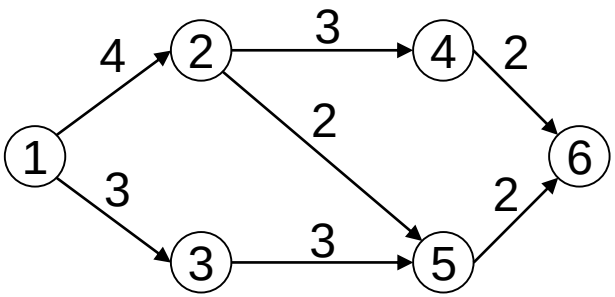
Length of a Path and The Shortest Path Problem

The length of a path is the sum of the cost weights of the arcs $((n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k))$.



Example: Find the shortest (i.e., cost minimal) path from node 1 to node 6.

LP-Formulation of the Shortest Path Problem



Solving Minimum Cost Flow Problems

- Minimum cost flow problems can be modelled as LPs and solved with the Simplex algorithm.

- However, due to the special structure of minimum cost flow problems there are more efficient algorithms than the Simplex.

The Dijkstra Algorithm for Solving Shortest Path Problems

Dijkstra's Algorithm

Dijkstra´s Algorithm

Calculates the shortest paths from an initial node s to all nodes of a digraph with cost weights $G = (V, A, c)$.

Prerequisites:

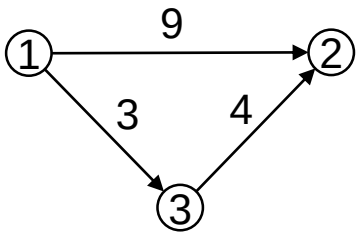
A digraph $G = (V, A, c)$ with non-negative cost weights, i.e., $c_{ij} \geq 0$.

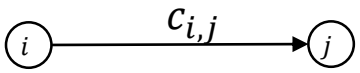
Edsger Dijkstra (1930-2002)
Professor at TU Eindhoven and the University of Texas at Austin

Dijkstra's Algorithm: Data Structure

$d[1,\ldots,n]$	d[i] provides the shortest distance from node s to node i.
$p[1,\ldots,n]$	p[i] provides the immediate predecessor node on the shortest path from node s to node i.
M	Set of marked nodes.
	A node i enters M when a first path from s to i has been found. When i leaves M, the shortest distance from s to i has been established.

Example:

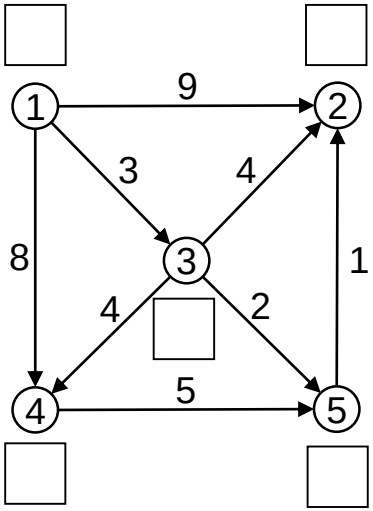


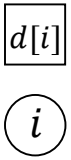


Dijkstra's Algorithm: Initialization

Initialize $M = \{s\}$
$d[s] = 0$
$d[s] = \infty$ for all $i \in V \backslash \{s\}$
$p[s] = s$; no entry for all $i \in V \backslash \{s\}$

Example:



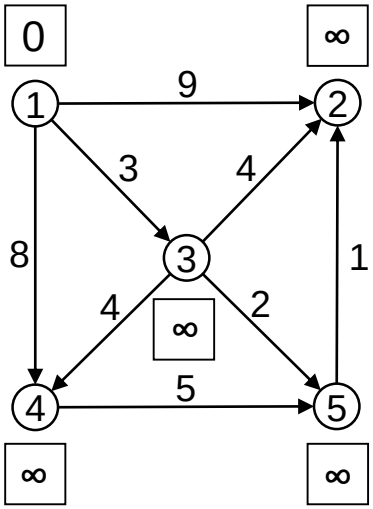


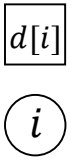
node i	1	2	3	4	5
$d[i]$					
$p[i]$					
$M = \{$					$\}$

Dijkstra's Algorithm: First Iteration

If set M is not empty **then**
  Select node $h \in M$ with smallest $d[h]$
  Delete h from M
  for all nodes $j \in V$ with arc $(h, j) \in A$ **do**
    if $d[j] > d[h] + c_{hj}$ **then**
      $d[j] = d[h] + c_{hj}$
      $p[j] = h$
      Mark and add j to M
    end if
  end for
end if

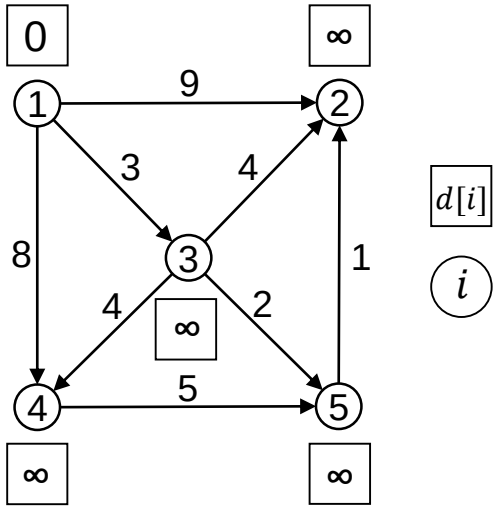
Example:





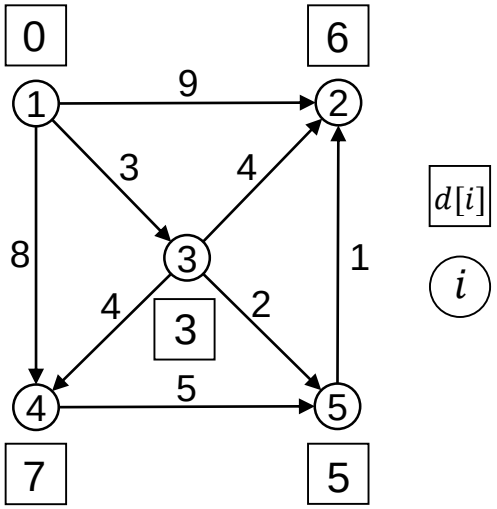
node i	1	2	3	4	5
$d[i]$	0	∞	∞	∞	∞
$p[i]$					
$M = \{1\}$					

Dijkstra's Algorithm: Iterations



node i	1	2	3	4	5
$d[i]$	0				
$p[i]$					
$M = \{$					$\}$

Example: Dijkstra's Algorithm - Results



node i	1	2	3	4	5
d[i]	0	6	3	7	5
p[i]	1	5	1	3	3
M = {}					

$d[i]$ contains for each $i \in V$ the shortest distance from $s = 1$ to i.

Example: $d[5]$=

The shortest path from $a = 1$ to i is derived recursively with the $p[i]$-entries.

Example:

The Floyd-Warshall Algorithm for Solving Shortest Path Problems

Floyd-Warshall Algorithm (FWA)

- Calculates shortest paths between every pair of nodes of a digraph.
- No prerequisites
- Applications: Scheduling, Vehicle Routing, ….

Robert Floyd
(1936-2001)
Stanford University

Stephen Warshall
(1935-2006)
Founder of Massachusetts Computer Associaton

FWA: Data Structures

Two $n \times n$ -matrices (n= number of nodes):

$\boldsymbol{d}$ $d[i,j]$ = length of the shortest path from node i to j.

$\boldsymbol{p}$ $p[i,j]$ = immediate predecessor node on the shortest path from i to j.

Example:

	$j=$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i=1$	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Length of the shortest path from node 1 to 2:

Shortest path from node 1 to 2:

Floyd-Warshall Algorithm

Initialization:

For all $i \in V$:

Set $d_{i,i} = 0,\ p_{i,i} = i$

For all $j \in V \backslash \{i\}$:

If $(i,j) \in A$

set $d_{i,j} = c_{i,j}$ and $p_{i,j} = i$

else

set $d_{i,j} = \infty$ and $p_{i,j} =$ ' $-$ '

For all arcs (i,j) we set $d_{i,j} = c_{i,j}$ and save i as immediate predecessor of j on the shortest path from i to j.

Iteration

For all $v \in V$

For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

For each triple i, v, j we check if the length of the path $i - v - j$ is shorter than the path $i - j$.

Initialization of the FWA

Initialization:

For all $i \in V$:

Set $d_{i,i} = 0,\ p_{i,i} = i$

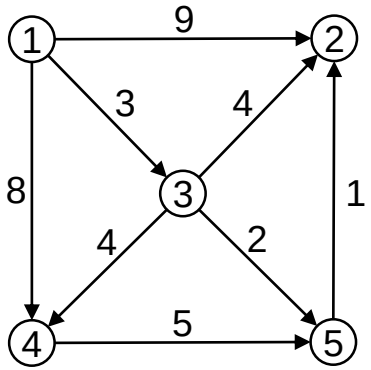
For all $j \in V \backslash \{i\}$:

If $(i,j) \in A$

set $d_{i,j} = c_{i,j}$ and $p_{i,j} = i$

else

set $d_{i,j} = \infty$ and $p_{i,j} =$ '–'



$j =$	1	2	3	4	5
$i =$	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
1					
2					
3					
4					
5					

Initialization of the FWA: Result

Initialization:

For all $i \in V$:

Set $d_{i,i} = 0,\ p_{i,i} = i$

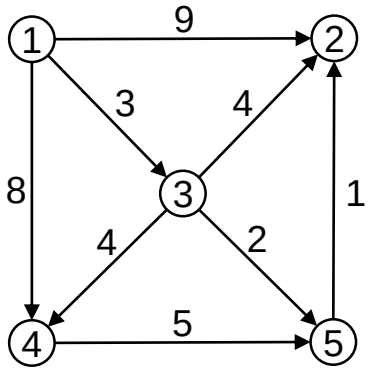
For all $j \in V \backslash \{i\}$:

If $(i,j) \in A$

set $d_{i,j} = c_{i,j}$ and $p_{i,j} = i$

else

set $d_{i,j} = \infty$ and $p_{i,j} =$ '–'



$j =$	1	2	3	4	5
$i =$	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
1	0; 1	9; 1	3; 1	8; 1	∞; –
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	4; 3	0; 3	4; 3	2; 3
4	∞; –	∞; –	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration of the FWA

Iteration

For all $v \in V$

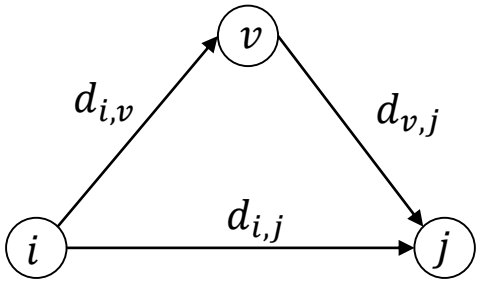
For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

For each triple i, v, j we check if the length of the path $i - v - j$ is shorter than the path $i - j$.



Iteration $v = 5$ of the FWA

Note: Nodes can be selected in any order; here we select nodes v by decreasing number.

Iteration
- For all $v \in V$
  - For all $i \in V$
    - For all $j \in V$
      - If $d_{i,j} > d_{i,v} + d_{v,j}$ then
        - Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i = 1$	0; 1	9; 1	3; 1	8; 1	∞; –
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	4; 3	0; 3	4; 3	2; 3
4	∞; –	∞; –	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration $v = 5$ of the FWA: Result

Iteration

For all $v \in V$

For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i = 1$	0; 1	9; 1	3; 1	8; 1	∞; –
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration $v = 4$ of the FWA

Iteration

For all $v \in V$

For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$	1	2	3	4	5
		$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i =$	1	0; 1	9; 1	3; 1	8; 1	∞; –
	2	∞; –	0; 2	∞; –	∞; –	∞; –
	3	∞; –	3; 5	0; 3	4; 3	2; 3
	4	∞; –	6; 5	∞; –	0; 4	5; 4
	5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration $v = 4$ of the FWA: Result

Iteration

For all $v \in V$

For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i = 1$	0; 1	9; 1	3; 1	8; 1	13; 4
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration $v = 3$ of the FWA

Iteration

For all $v \in V$

For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i = 1$	0; 1	9; 1	3; 1	8; 1	13; 4
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration $v = 3$ of the FWA: Result

Iteration

For all $v \in V$

For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i = 1$	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration and Result for $v = 2$ of the FWA

Iteration

For all $v \in V$

For all $i \in V$

For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i = 1$	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Iteration and Result for $v = 1$ of the FWA

Iteration

For all $v \in V$

For all $i \in V$

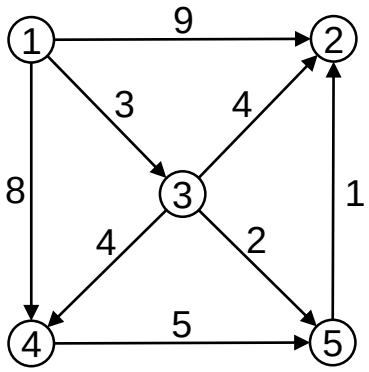
For all $j \in V$

If $d_{i,j} > d_{i,v} + d_{v,j}$ then

Set $d_{i,j} = d_{i,v} + d_{v,j}$ and $p_{i,j} = p_{v,j}$

	$j =$ 1	2	3	4	5
	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
$i = 1$	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

Result of the FWA



$j =$	1	2	3	4	5
$i =$	$d_{i,1}, p_{i,1}$	$d_{i,2}, p_{i,2}$	$d_{i,3}, p_{i,3}$	$d_{i,4}, p_{i,4}$	$d_{i,5}, p_{i,5}$
1	0; 1	6; 5	3; 1	7; 3	5; 3
2	∞; –	0; 2	∞; –	∞; –	∞; –
3	∞; –	3; 5	0; 3	4; 3	2; 3
4	∞; –	6; 5	∞; –	0; 4	5; 4
5	∞; –	1; 5	∞; –	∞; –	0; 5

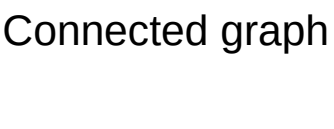
5.3 The Minimum Spanning Tree Problem

Connected Graph

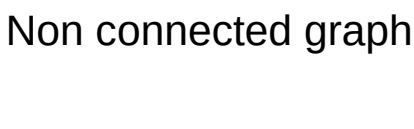
A graph is connected if it contains at least one path between each pair of nodes.

Example:

Connected graph

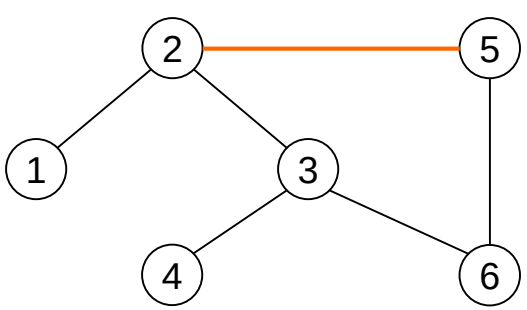


Non connected graph

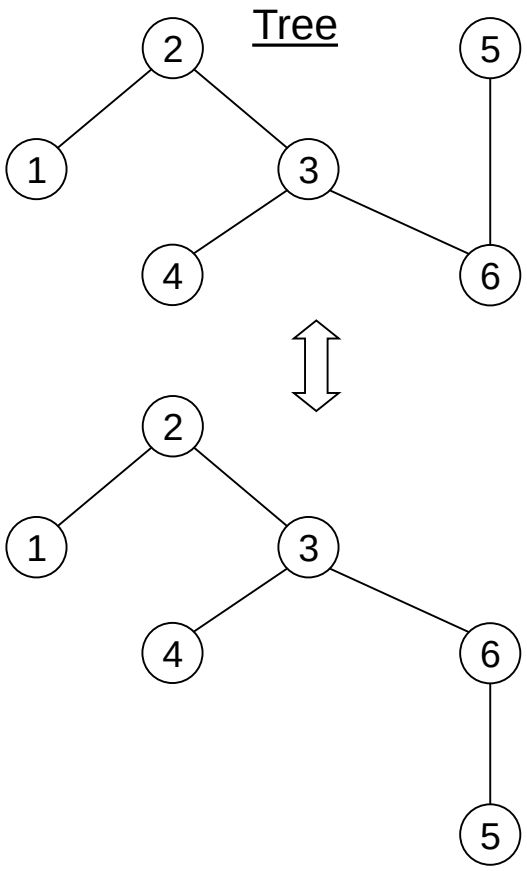


Tree

A connected graph without cycles is a tree.



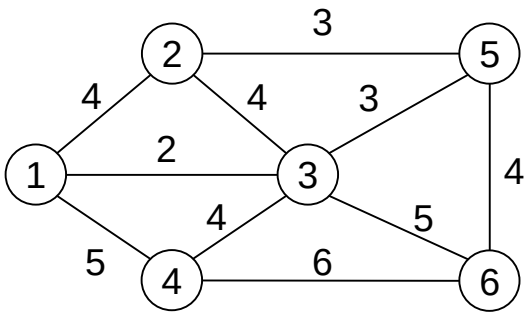
Connected graph with cycle



Note: A tree with n nodes has $n - 1$ edges.

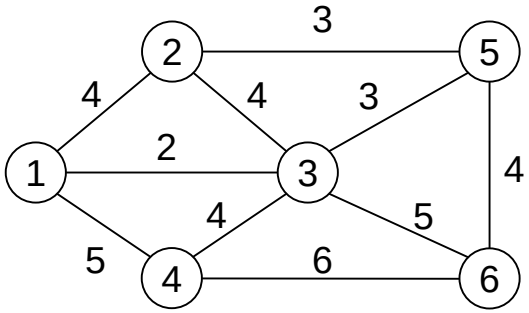
Minimum Spanning Tree

Given graph $G = [V, A, c]$ we want to determine a subset of edges $A' \subseteq A$ such that graph $G' = [V, A', c]$ is connected and the sum of the cost of the selected edges is minimal.

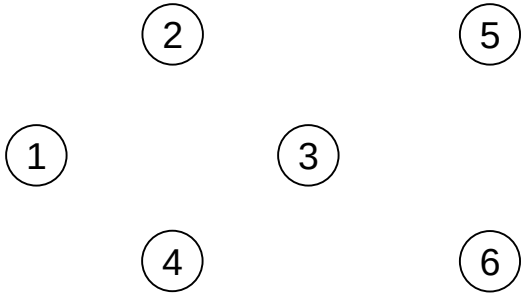


Applications: Electricity networks, water networks, communication networks.

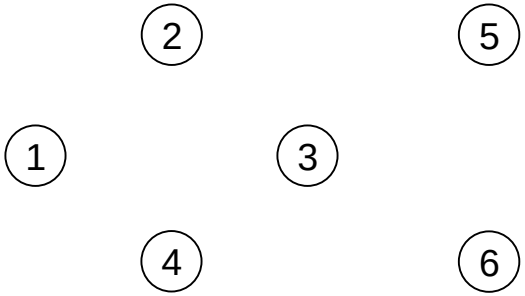
Observation: A Minimum Spanning Tree is Cycle Free



Connected graph with $\Sigma\, c = 40$

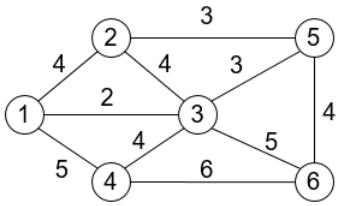


Connected graph with one cycle

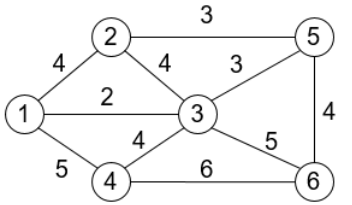


Connected graph without cycle

MIP for the Minimum Spanning Tree Problem



Cycle Elimination Constraints

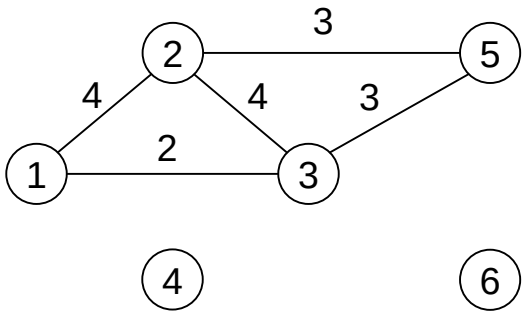


Min $4x_{1,2} + 2x_{1,3} + 5x_{1,4} + 4x_{2,3} + 3x_{2,5} + 4x_{3,4} + 3x_{3,5} + 5x_{3,6} + 6x_{4,6} + 4x_{5,6}$

s.t. $1x_{1,2} + 1x_{1,3} + 1x_{1,4} + 1x_{2,3} + 1x_{2,5} + 1x_{3,4} + 1x_{3,5} + 1x_{3,6} + 1x_{4,6} + 1x_{5,6} = 5$

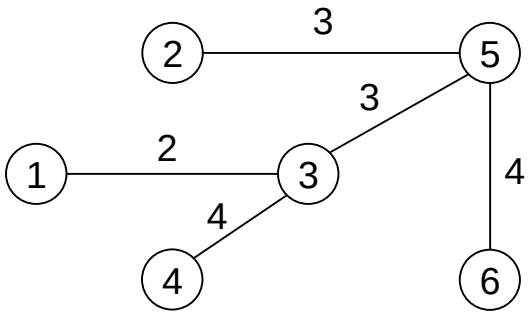
$x_{1,2}, x_{1,3}, x_{1,4}, x_{2,3}, x_{2,5}, x_{3,4}, x_{3,5}, x_{3,6}, x_{4,6}, x_{5,6} \in \{0,1\}$

Solution without subtour elimination constraint:



z=16

Solution with subtour elimination constraint:



z*=16

Kruskal's Algorithm

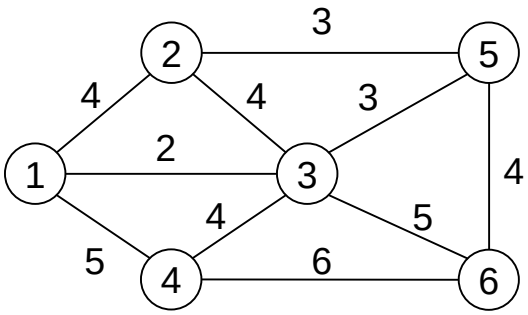
Initialization

Sort edges $[i,j] \in A$ in list L according to non-decreasing c_{ij} (in case of ties sort according to increasing i and j)

Set $A' = \varnothing$

Example:

$L =<$]

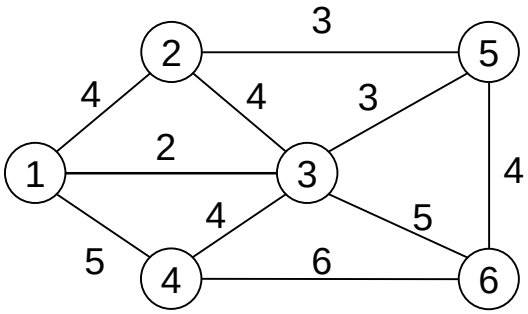


Kruskal's Algorithm

Iteration

While $L \neq \emptyset$

1. Select the next edge $[i,j]$ from the list L
2. If graph $G = [V, A' \cup [i,j]]$ has no cycles set $A' = A' \cup [i,j]$
3. Delete $[i,j]$ from L

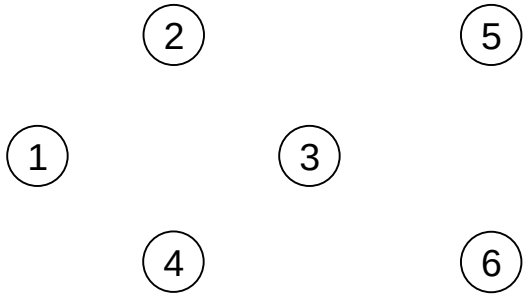


Example:

Iteration 1

$L =< [1,3], [2,5], [3,5], \ldots]$

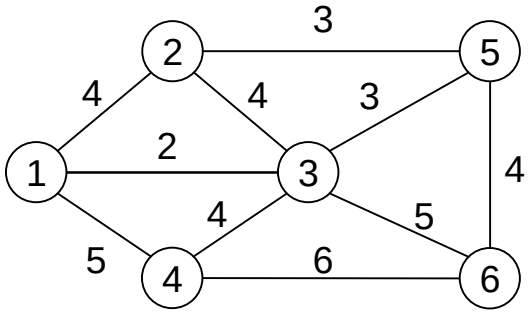
1. Select edge
2. $A' = \{$ $\}$
3. $L =<$]



Kruskal's Algorithm in Table Format

$L =< [1,3], [2,5], [3,5], [1,2], [2,3], [3,4], [5,6], [1,4], [3,6], [4,6]]$

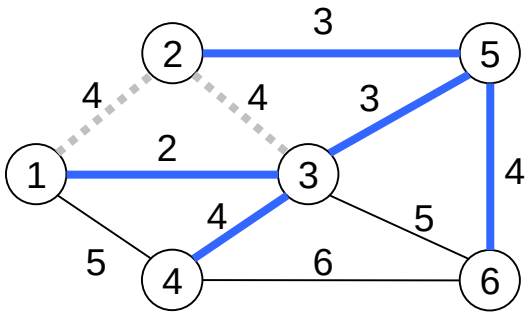
Iteration	$[i,j]$	A'	$\sum_{[i,j]\in A'} c_{ij}$
Initialization			



Kruskal's Algorithm: Result

$L =< [1,3], [2,5], [3,5], [1,2], [2,3], [3,4], [5,6], [1,4], [3,6], [4,6]]$

Iteration	$[i,j]$	A'	$\sum_{[i,j]\in A'} c_{ij}$
Initialization		$\varnothing$	0
1	[1,3]	{[1,3]}	2
2	[2,5]	{[1,3],[2,5]}	5
3	[3,5]	{[1,3],[2,5],[3,5]}	8
4	[1,2]	{[1,3],[2,5],[3,5]}	8
5	[2,3]	{[1,3],[2,5],[3,5]}	8
6	[3,4]	{[1,3],[2,5],[3,5],[3,4]}	12
7	[5,6]	{[1,3],[2,5],[3,5],[3,4],[5,6]}	16

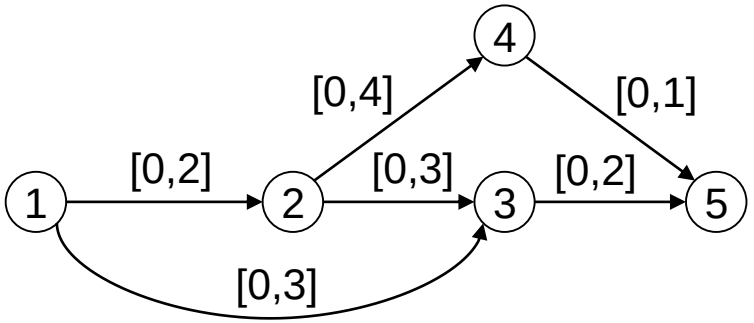


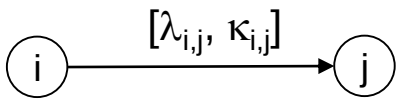
5.4 The Maximum Flow Problem

Maximum Flow Problem

- Given digraph G=(V,A,λ,κ).
- We want to determine the maximum flow we can send from node q to node s, q,s ∈ V, q≠s.
- Applications: Determine the capacity of a network (infrastructure such as internet, streets, railroads, pipeline, …).

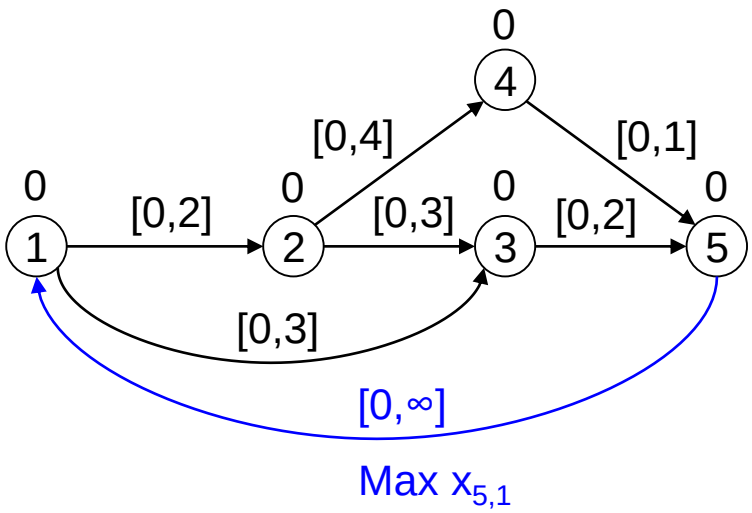
- Example: Maximum flow from q=1 to s=5.

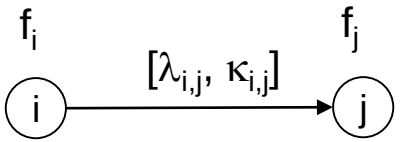




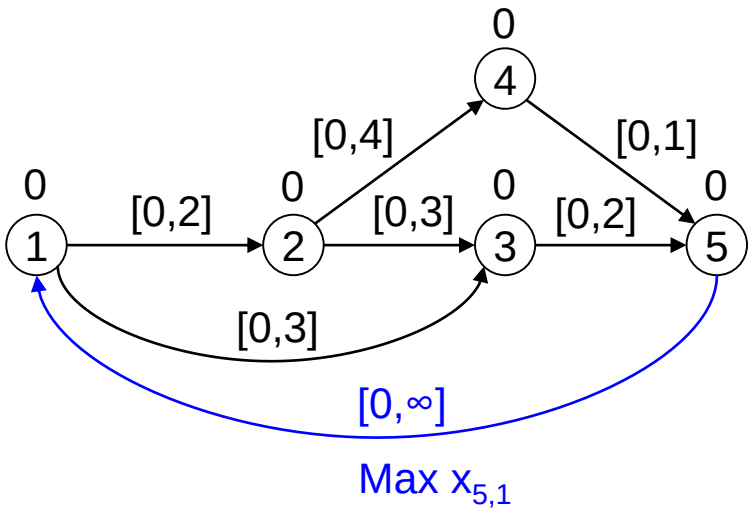
Modeling the Maximum Flow Problem as LP

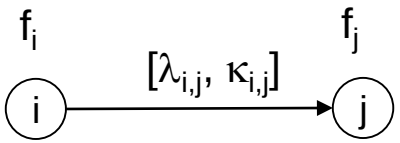
- Adding artificial arc (5,1) with $\lambda_{5,1}=0$ und $\kappa_{5,1}=\infty$.
- Max $x_{5,1}$ taking into account real arcs and nodes.





Modeling the Maximum Flow Problem as LP



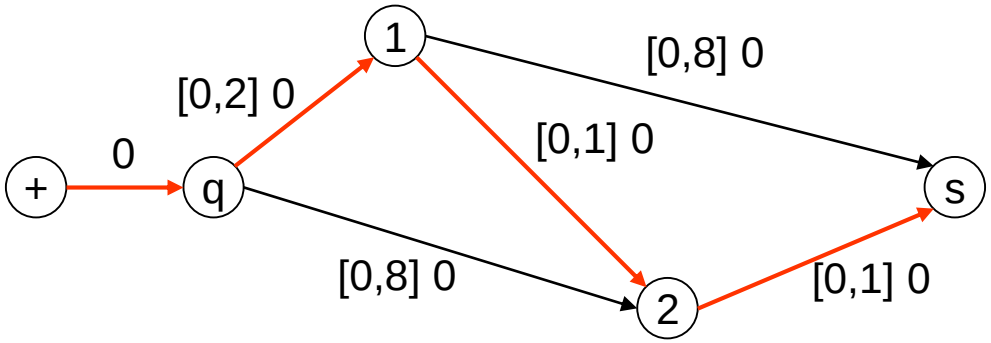


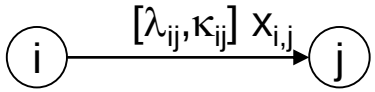
The Ford-Fulkerson Algorithm for Solving the Maximum Flow Problem

Ford-Fulkerson Algorithm: Outline

- Start with a feasible (not necessarily optimal) solution
- If $\lambda_{i,j} = 0$ holds for all $(i,j) \in A$, a first feasible solution is $x_{i,j}=0$ for all $(i,j) \in A$.
- In each iteration we try to improve the solution until the optimality criterion is fulfilled.

Ford-Fulkerson Algorithm: Start Solution

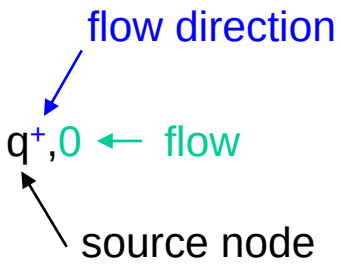




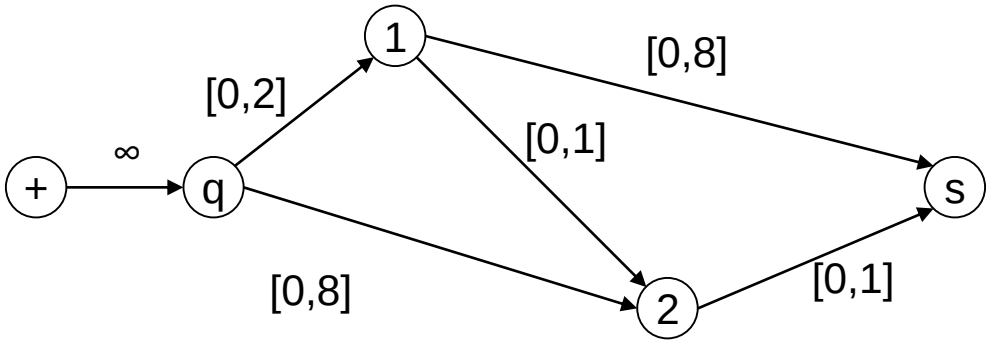
We use the following table format in order to depict an increment of the existing flow:

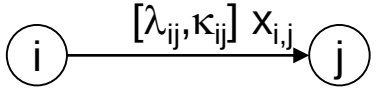
Node	Source node, flow direction, flow
q	
1	
2	
s	
Incremental flow Δ	
Total flow Σ	

Legend:



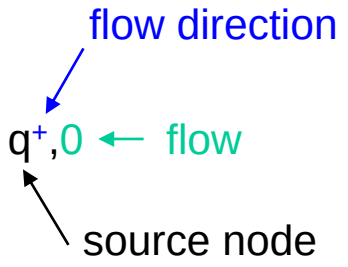
Ford-Fulkerson-Algorithm: Example



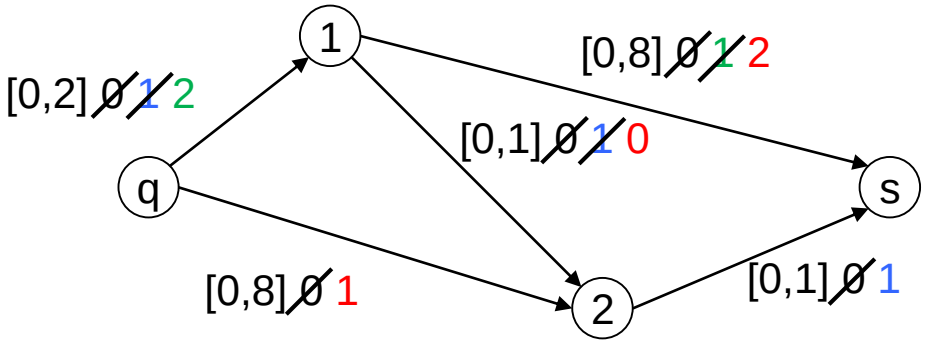


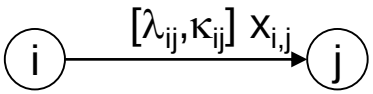
Node	Start solution	Iteration 1	Iteration 2	Iteration 3	Iteration 4
q					
1					
2					
s					
Incremental flow Δ					
Total flow Σ					

Legend:



Ford-Fulkerson Algorithm: Protocol and Optimal Solution



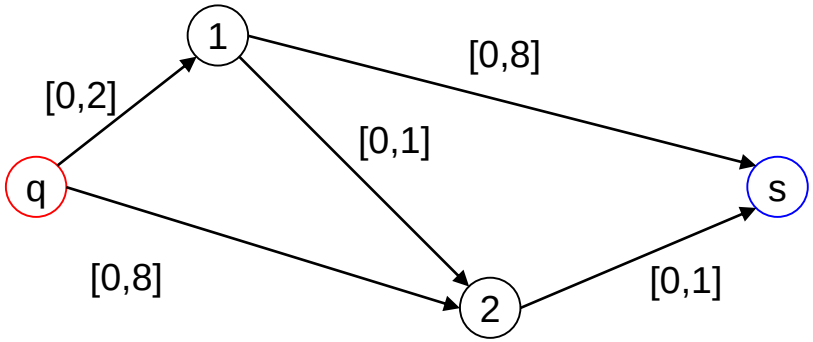


Node	Start solution	Iteration 1	Iteration 2	Iteration 3	Iteration 4
q	(+, 0)	(+, ∞)	(+, ∞)	(+, ∞)	(+,∞)
1	(q^+,0)	(q^+,2)	(q^+,1)	(2^-,1)	
2	(1^+,0)	(1^+,1)		(q^+,8)	(q^+,7)
s	(2^+,0)	(2^+,1)	(1^+,1)	(1^+,1)	
Incremental flow Δ	0	1	1	1	0
Total flow Σ	0	1	2	3	3

Cut

A cut divides node set V into the following two subsets V^q and V^s. V^q contains source node q and V^s contains sink node s. The intersection of V^q and V^s is empty, that is $V^q \cap V^s = \emptyset$.

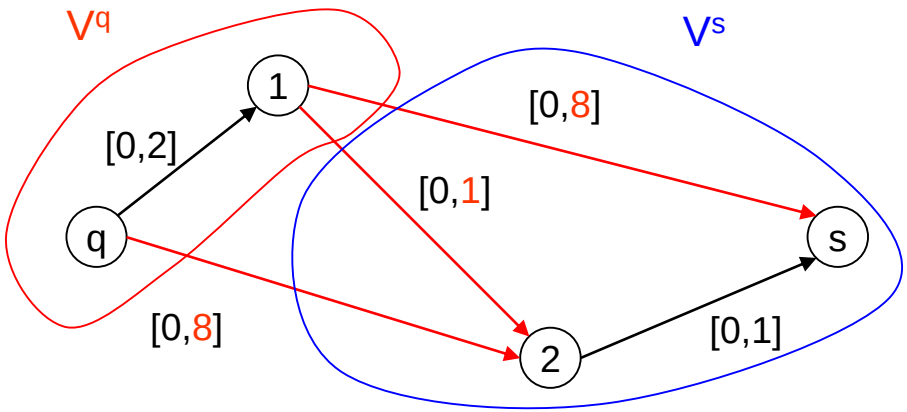
Example: Cut



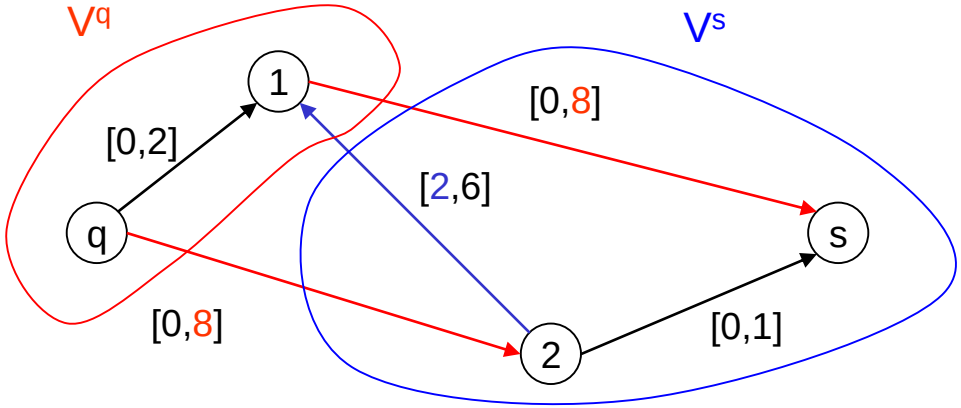
Capacity of a Cut

The capacity of a cut is the sum of $\kappa_{i,j}$ for all arcs (i,j) leading from V^q to V^s minus the sum of $\lambda_{i,j}$ for all arcs leading from V^s to V^q.

Example:



Capacity =

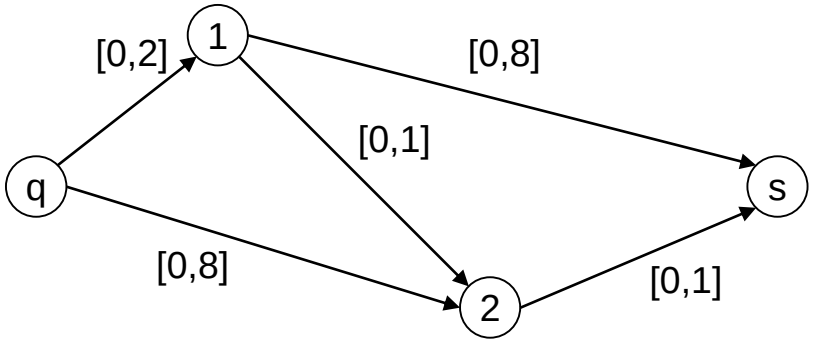


Capacity =

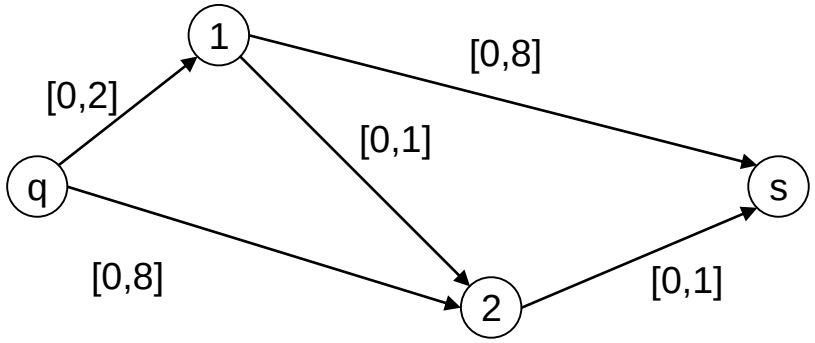
Minimum Cut Problem

- For a digraph $G=(V,A,\lambda,\kappa)$ divide the nodes into disjoint subsets V^q and V^s such that the capacity of the cut is minimal.

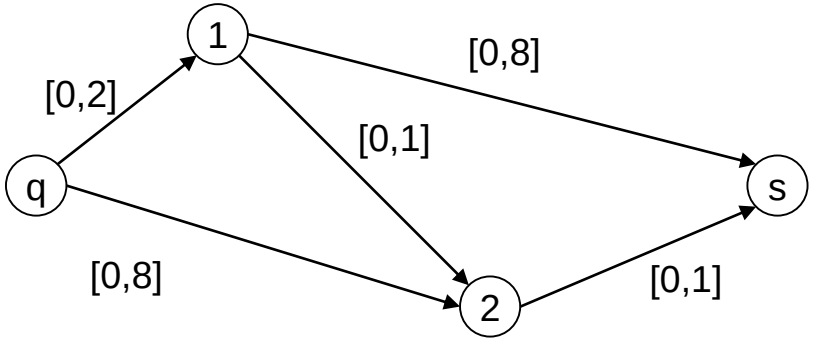
All Solutions of the Minimum Cut Problem



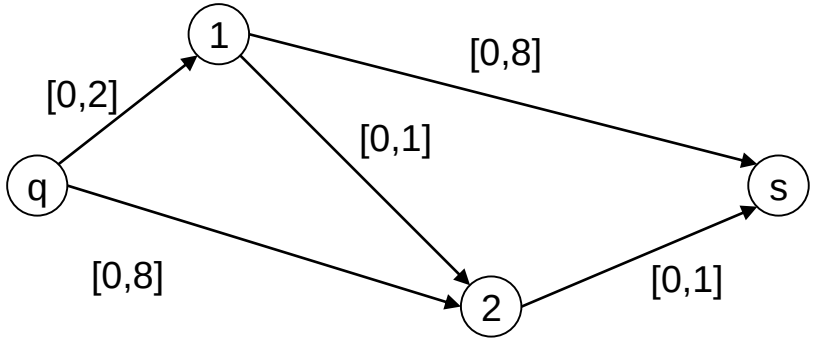
Capacity (cut)=



Capacity (cut)=



Capacity (cut)=



Capacity (cut)=

Relation between Max Flow and Min Cut

■ The min cut problem is the dual of the max flow problem.

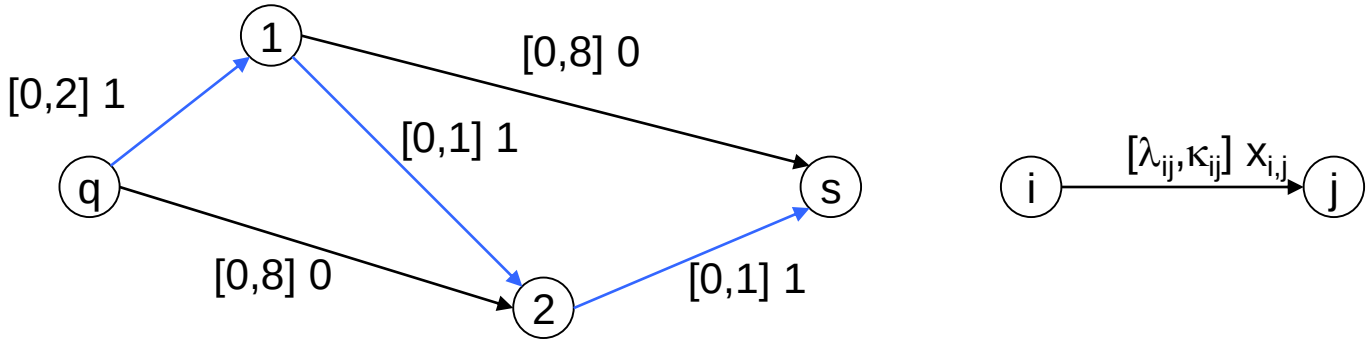
This implies:

■ Weak Duality Property: The objective function value of a feasible but not optimal solution of the max flow problem is strictly smaller than the objective function value of a feasible but not optimal min cut problem.

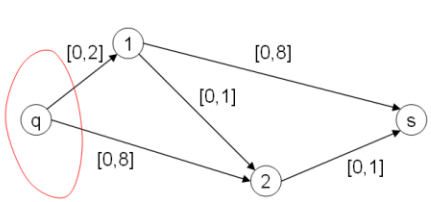
■ Strong Duality Property: The objective function value of an optimal solution of the max flow problem equals the objective function of an optimal solution of the min cut problem.

Example: Feasible but not Optimal Solutions

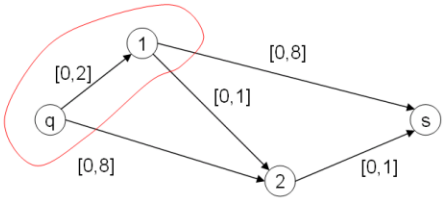
Feasible but not optimal solution of the max flow problem with z=1



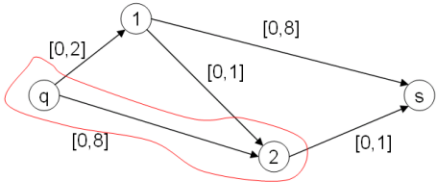
Each solution of the min cut problem has an objective function value of z > 1.



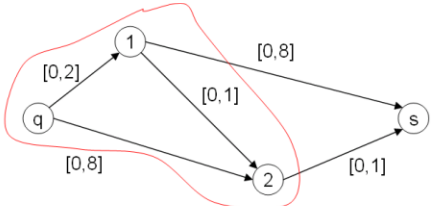
Capacity= 2 + 8 =10



Capacity= 8 + 1 + 8 =17



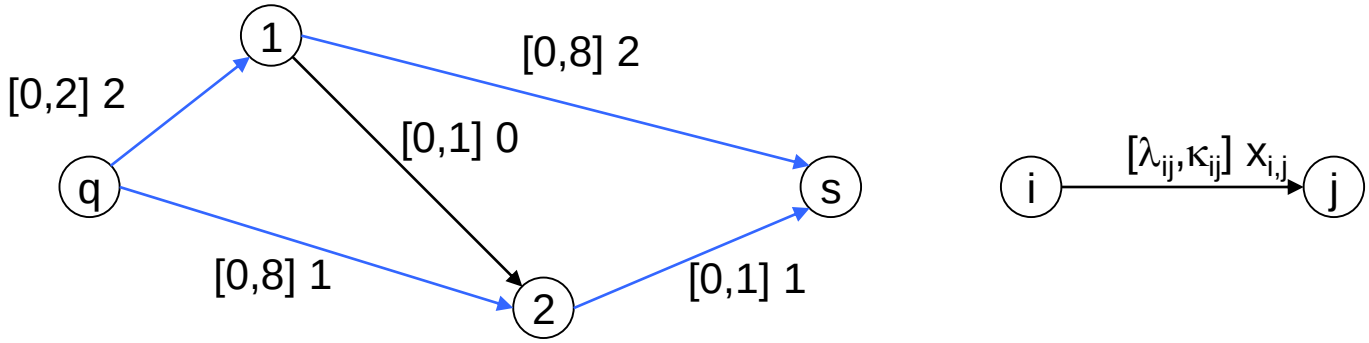
Capacity= 2 + 1 - 0 = 3 (=Min!)



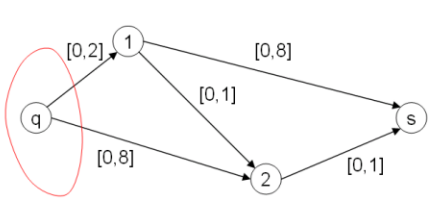
Capacity= 8 + 1 = 9

Example: Optimum Solution

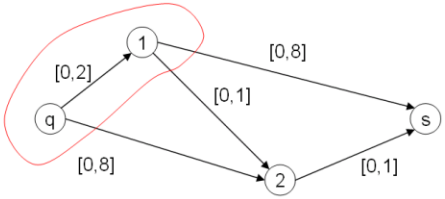
Optimal solution of the max flow problem with z=3.



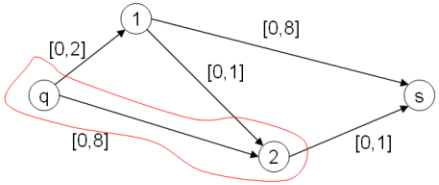
The optimal solution of the min cut problem has an objective function value of z = 3.



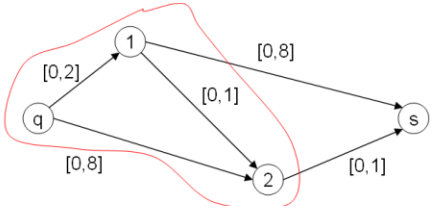
Capacity= 2 + 8 =10



Capacity= 8 + 1 + 8 =17



Capacity= 2 + 1 - 0 = 3 (=Min!)

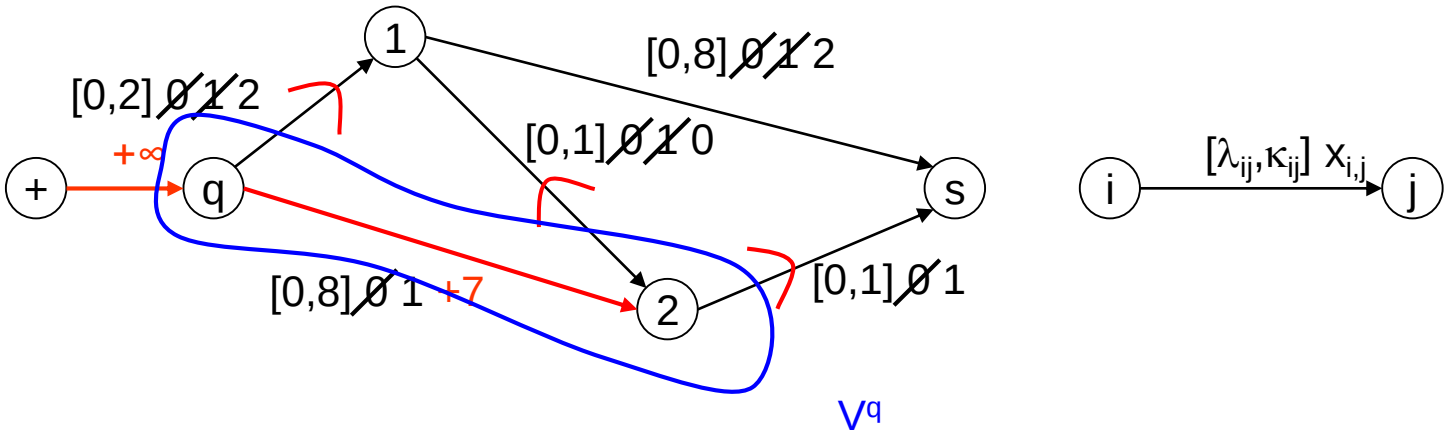


Capacity= 8 + 1 = 9

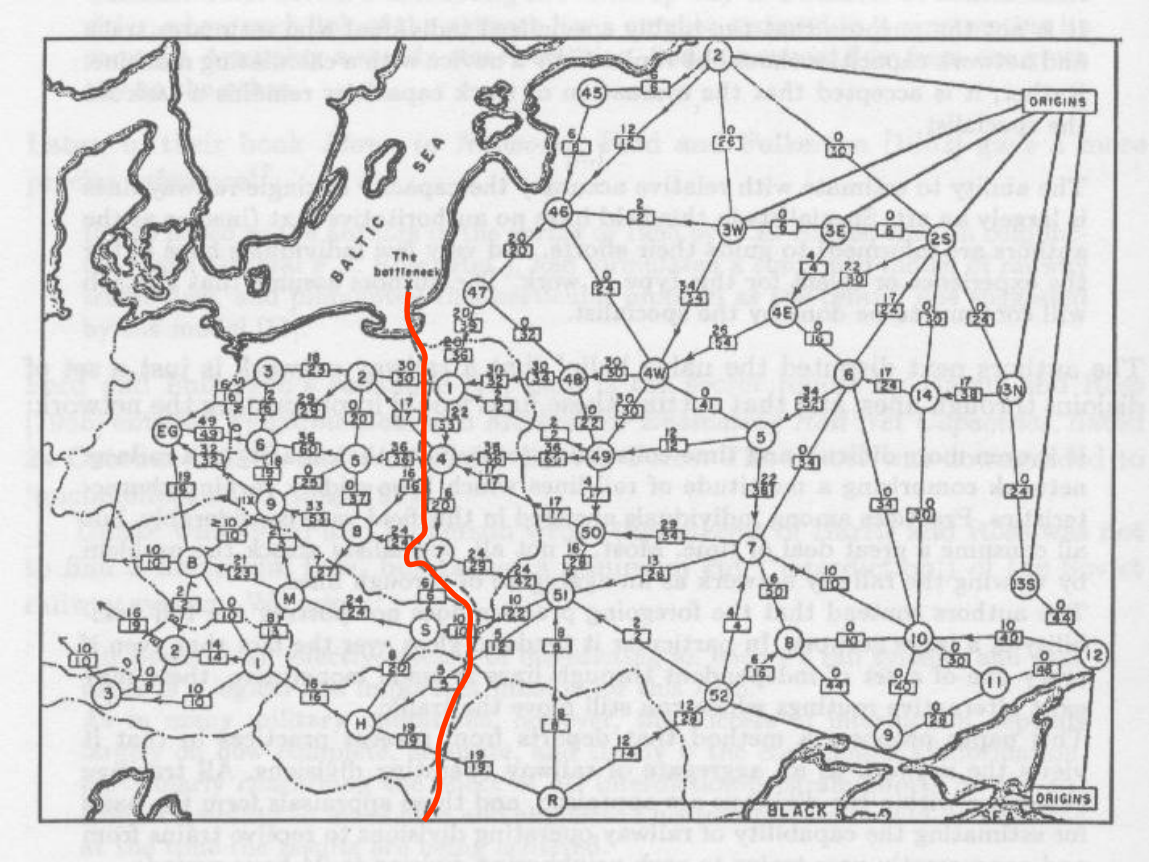
The Ford-Fulkerson Algorithm terminates with Min Cut and thus Max Flow Solution

- Let all nodes visited in order to increment the flow be in V^q.
- Let all nodes not visited when incrementing the flow be in V^s.
- If we stop incrementing the flow not reaching sink node s, the following holds
  - $x_{i,j}=\kappa_{i,j}$ for all arcs (i,j) with $i \in V^q$ and $j \in V^s$
  - $x_{i,j}=\lambda_{i,j}$ for all arcs (i,j) with $i \in V^s$ and $j \in V^q$.
- The associated cut has the same optimal objective function value as the current solution of the max flow problem.

Termination of the FF Algorithm and Associated Cut



Application of Max Flow and Min Cut



Source: Schrivjer (2003) p. 168