IE 604: Network Flow Optimization[1]

Course Notes, Part I: Introduction

Dr. Jim Ostrowski

University of Tennessee - Knoxville

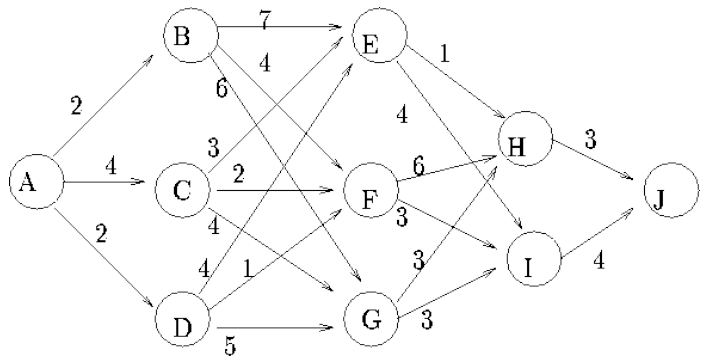
[1]Last updated: January 23, 2022

Section 1

A First Taste of Network Flows

A Shortest Path Problem

In the following network the numbers on the edges denote the edge's travel time.



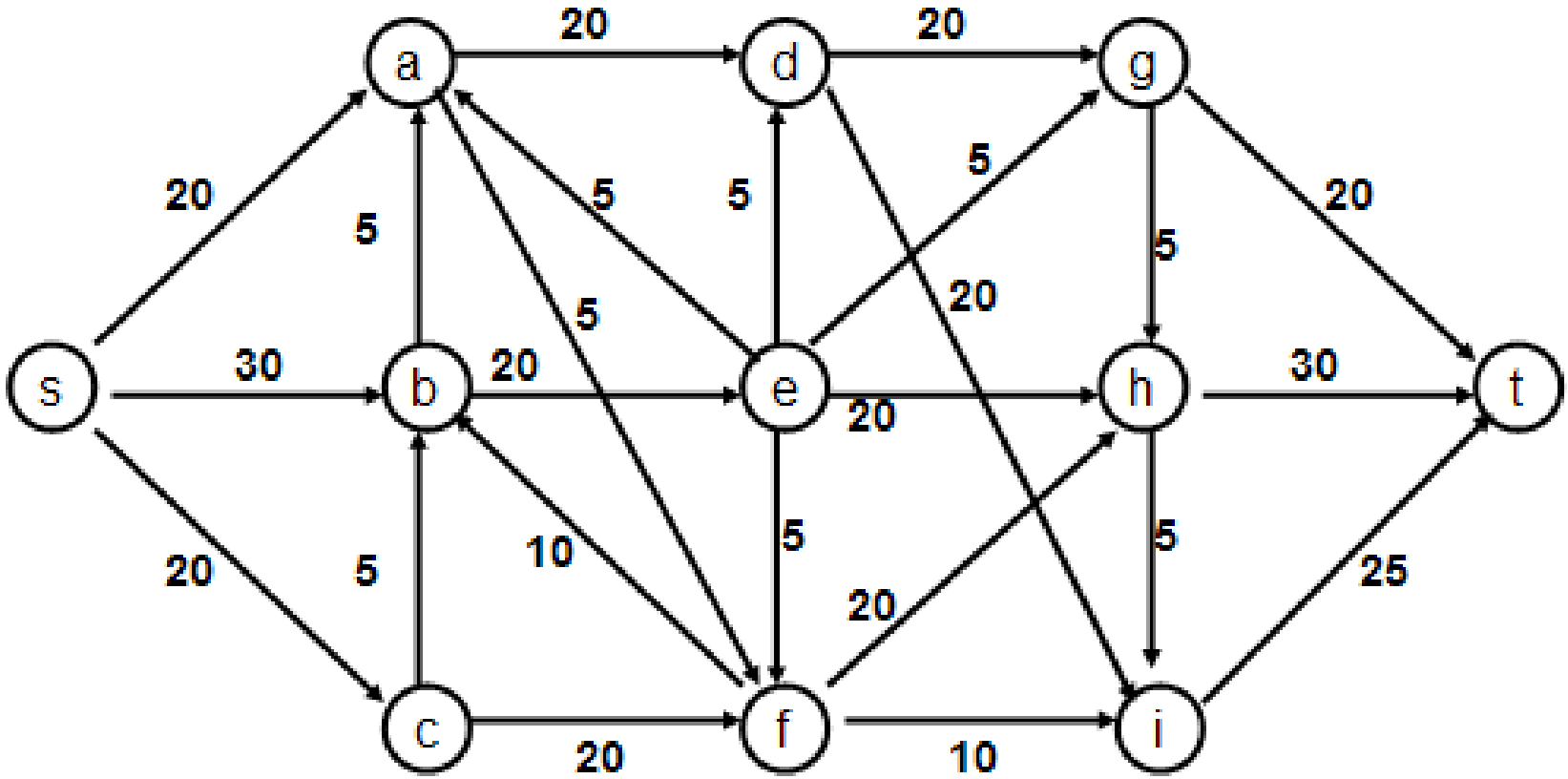
Find the shortest path from the node A to node J.

Your Work Goes Here

Unit 1

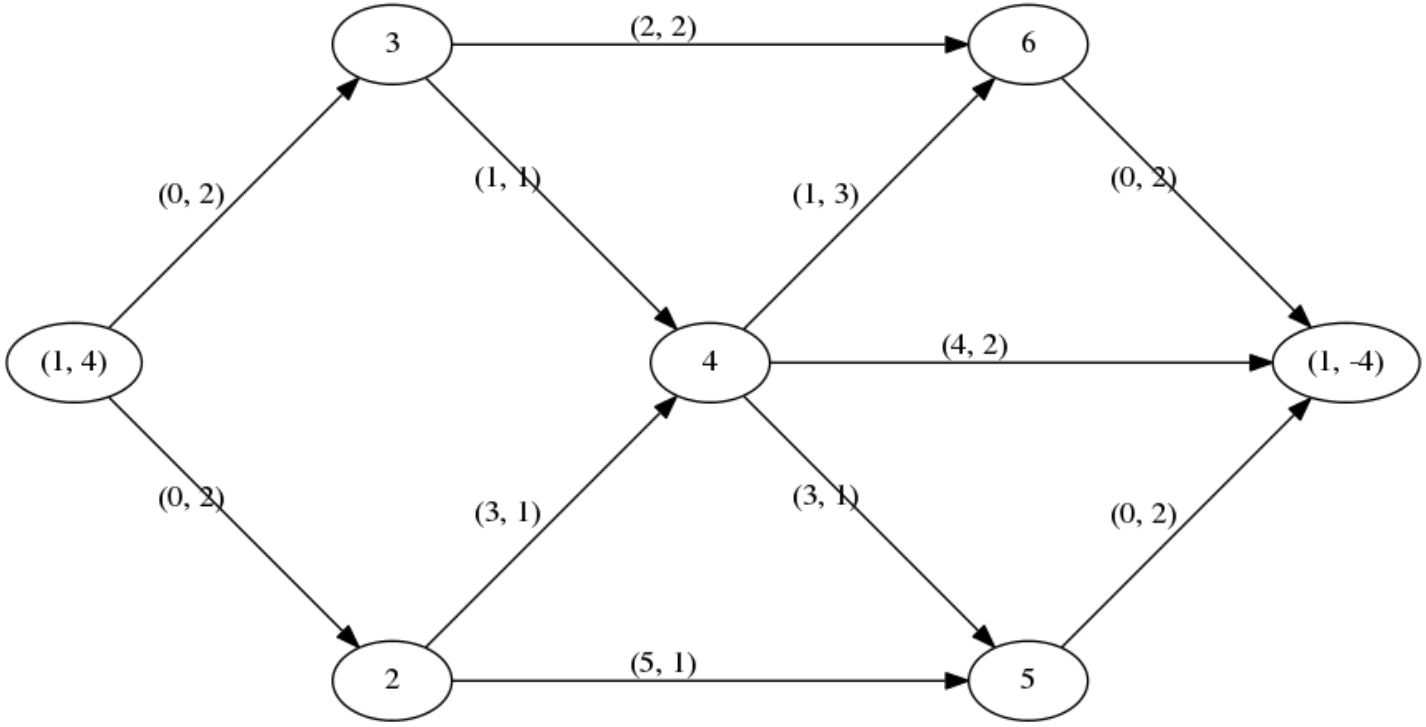
Other Network Flow Problems

Maximum Flow Problem



Minimum Cost Flow Problem

In the graph below the first label of each edge denotes the cost per unit flow to ship goods from one facility to the next. The second denotes the capacity. The first label of each node indicates its number and the second its demand. Find a minimum cost set of flows that obeys the capacity constraints. This is an example of the minimum cost flow problem.



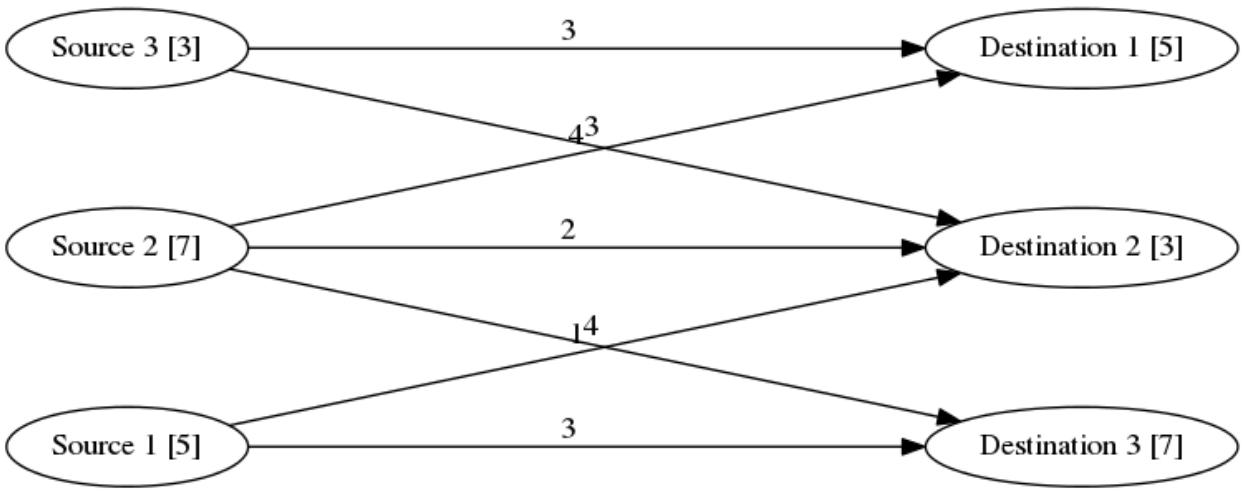
Assignment Problem (2)

Transportation Problem

In the following matrix the rows represent sources and the columns represent destinations. The cells of the matrix contain the per-unit cost for shipping from a source to a destination. Each source is constrained by a limited supply, and each destination has a demand. The transportation problem is to choose a set of flows that minimizes the total cost of meeting the demand of each of the destinations while not exceeding the supply of any of the suppliers.

	D1	D2	D3	Supply
S1	3	1	-	5
S2	4	2	4	7
S3	-	3	3	3
Demand	7	3	5	

Transportation Problem (2)

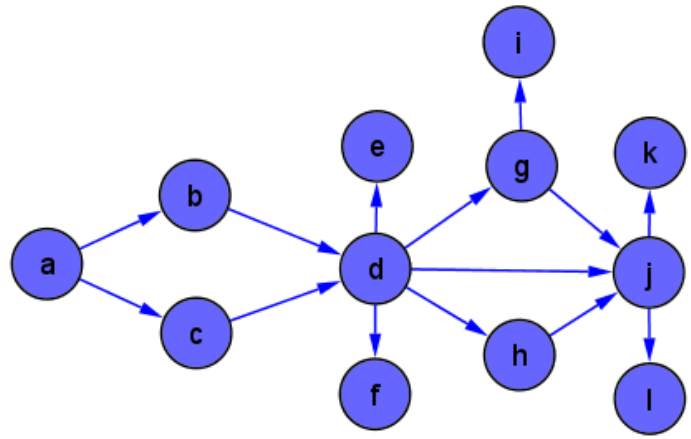


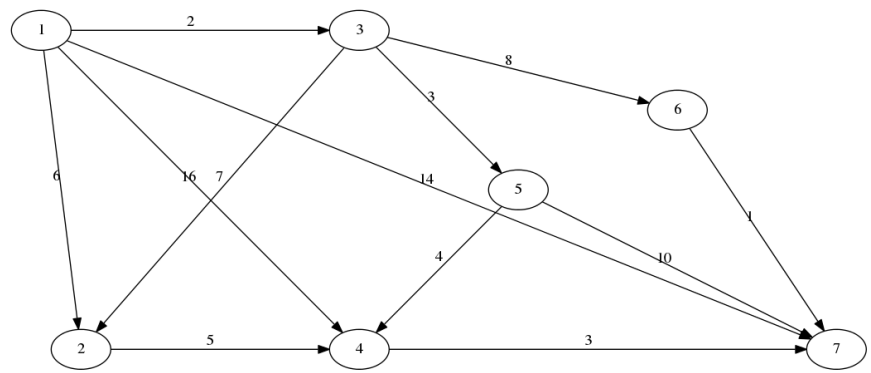
Section 2

Notation and Concepts

Directed Graphs and Networks

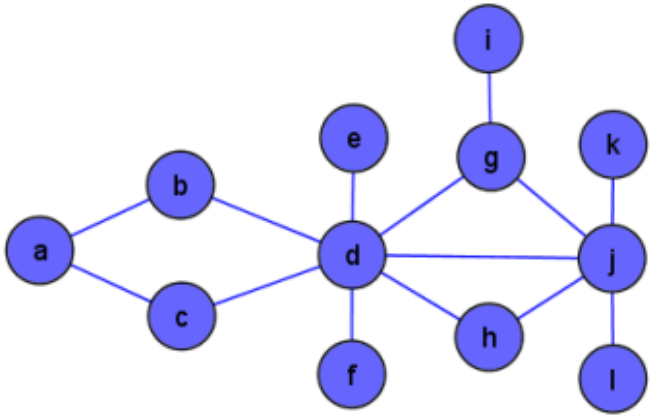
- Directed graph, $G = (N, A)$
- consists of nodes and arcs (ordered pairs of nodes, (i, j))
- Directed network
  - arcs and/or nodes have numerical values (data) associated with them such as costs, capacities, etc.
- Notation
  - $n = |N|$, number of nodes
  - $m = |A|$, number of arcs

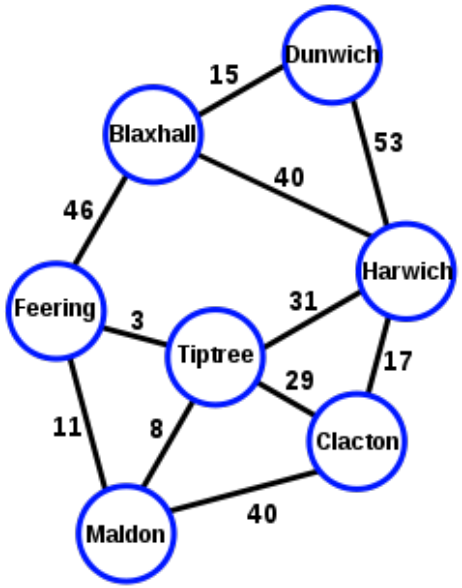




Undirected Graphs and Networks

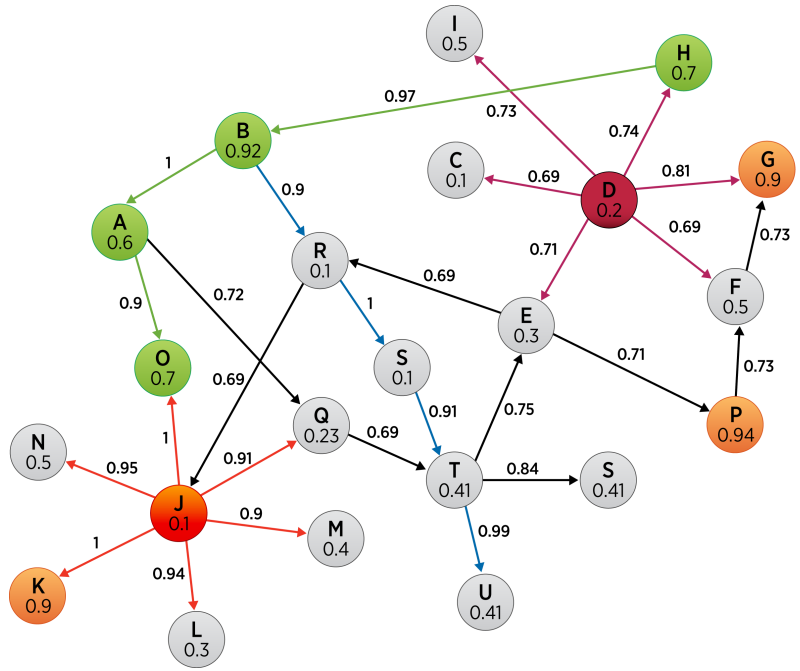
- Undirected graph or network, $G = (N, A)$
  - arcs are unordered pairs of distinct nodes, (i, j) or (j, i)





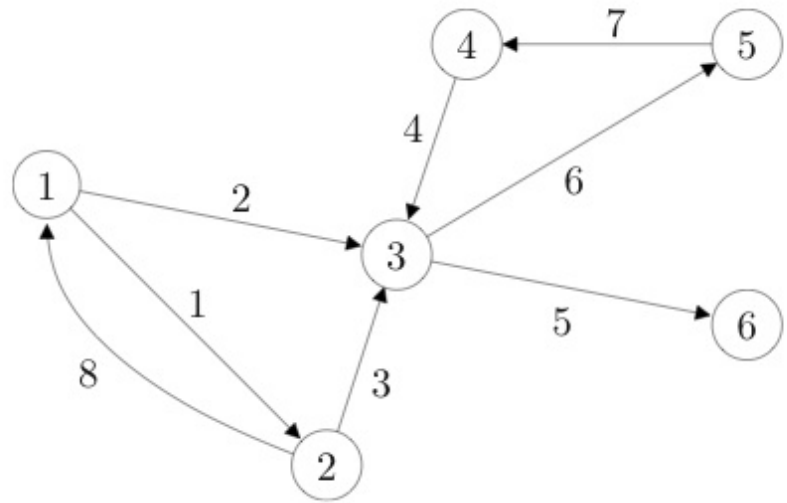
Nodes

- Also known as
  - vertex
- Data
  - capacity
  - supply/demand
- Topological properties
  - indegree: number of incoming arcs
  - outdegree: number of outgoing arcs
  - degree = indegree + outdegree



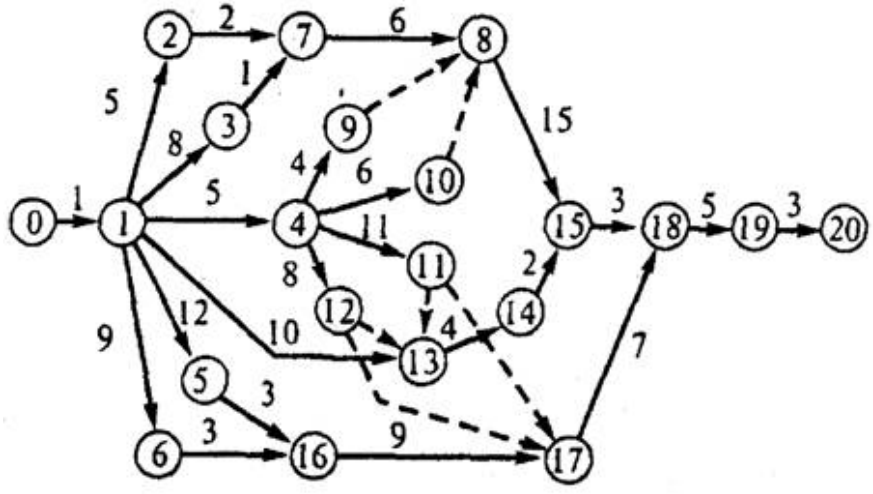
Arcs

- Also known as
  - edge
  - link
- Data
  - capacity
  - cost/weight
- Topological properties
  - a directed arc has two endpoints: j and i
  - head, j (the endpoint with the arrowhead)
  - tail, i



Special Nodes

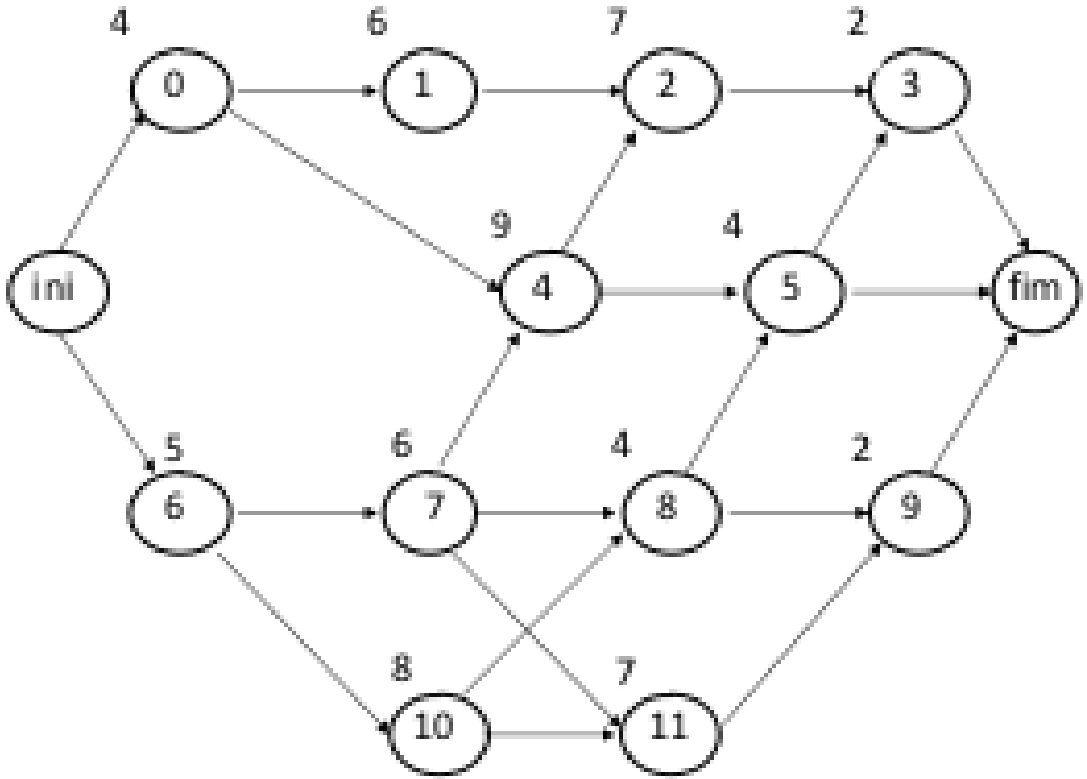
- Source / origin
  - the node from which flow originates
- Sink / destination
  - the node from which flow terminates
- Example
  - find the shortest path from node 0 to node 20



Adjacency List

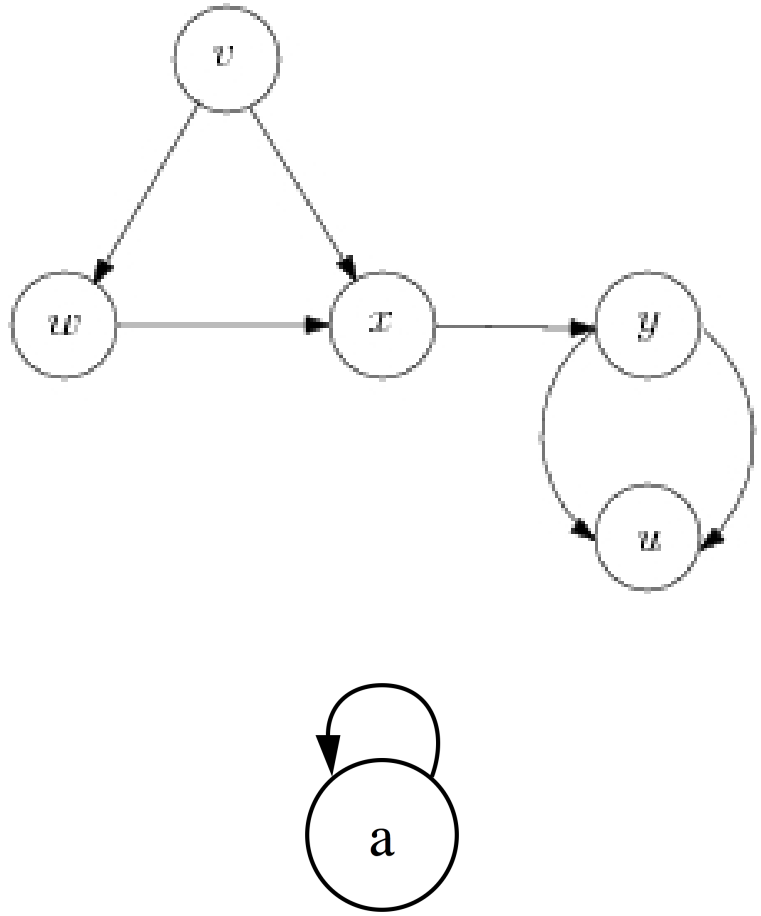
- Arc adjacency list A_i of a node i
  - set of all arcs emanating from that node
  - $A_i = \{(i,j) \in A : j \in N\}$
- Node adjacency list N_i of a node i
  - set of nodes adjacent to that node
  - $N_i = \{j \in N : (i,j) \in A\}$
- The forward star A_i^+ of a node i is the set of outgoing arcs from a node
- The reverse star A_i^- of a node i is the set of incoming arcs to a node

Adjacency List (2)



Multiarcs and Loops

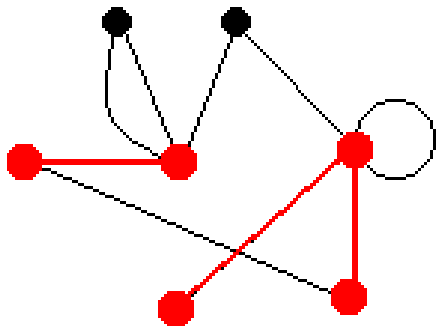
- Multiarc
  - two or more arcs with the same tail and head nodes
- (Self) loop
  - an arc whose tail node is the same as its head node



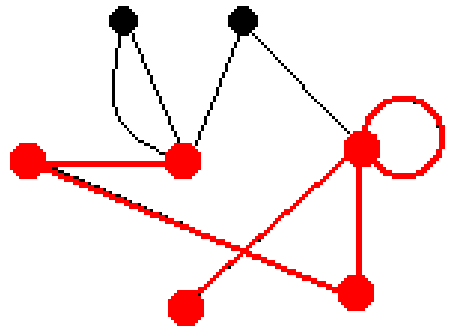
Subgraph

- A graph $G' = (N', A')$ is a subgraph of a graph $G = (N, A)$ if:
  - $N' \subseteq N$
  - $A' \subseteq A$
- $G' = (N', A')$ is the subgraph induced by N' if
  - A' contains each arc in A with both endpoints in N'
- $G' = (N', A')$ is a spanning subgraph of $G = (N, A)$ if
  - $N' = N$
  - $A' \subseteq A$

Subgraph



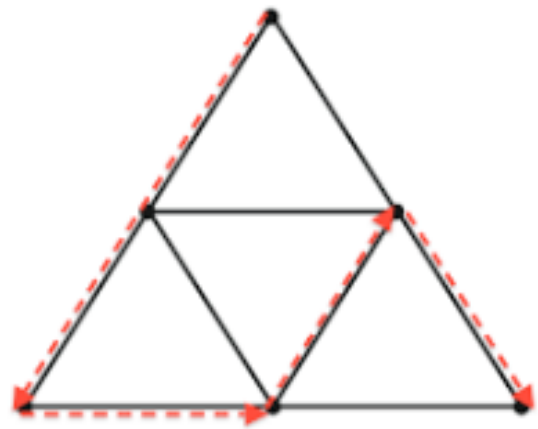
Subgraph (in red)



Induced Subgraph

Walks

- A walk in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes and arcs
  $i_1 - a_1 - i_2 - a_2 - \cdots - i_{r-1} - a_{r-1} - i_r$
  such that
  - for all $1 \leq k \leq r-1$, either
    $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$
- Directed walk
  - for any two consequtive nodes, i_k and i_{k+1}, $(i_k, i_{k+1}) \in A$
  - it is possible to walk along the sequence of arcs and nodes defined by a directed walk

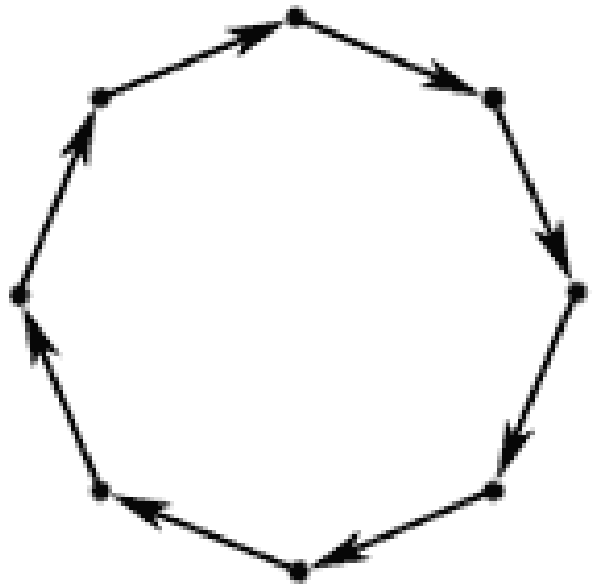


Paths

- A path is a walk that does not visit any node more than once
- A directed path is a directed walk that does not visit any node more than once
  - no backward arcs
- Example
  - the **shortest path problem** is to find the minimum weight directed path from the origin to the destination

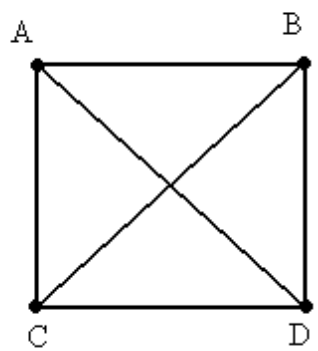
Cycles

- A cycle is a path in which the start node is the same as the finish node
- A directed cycle is a directed path in which the start node is the same as the finish node
- An acyclic graph contains no directed cycle

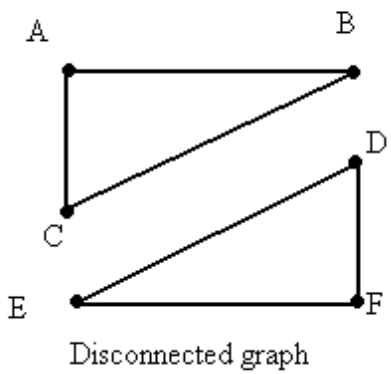


Connectivity

- Nodes i and j are connected if the graph contains at least one path from i to j
  - they are strongly connected if the graph contains at least one directed path from i to j
- A graph is a connected graph if every pair of nodes is connected; otherwise the graph is disconnected



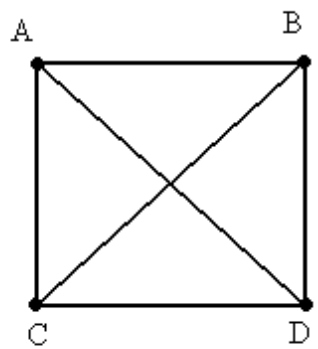
Connected graph



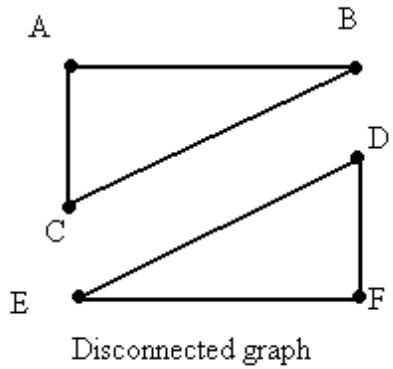
Disconnected graph

Connectivity (2)

- A graph is a strongly connected graph if every pair of nodes is strongly connected;
- The components of a disconnected graph are its maximally connected subgraphs



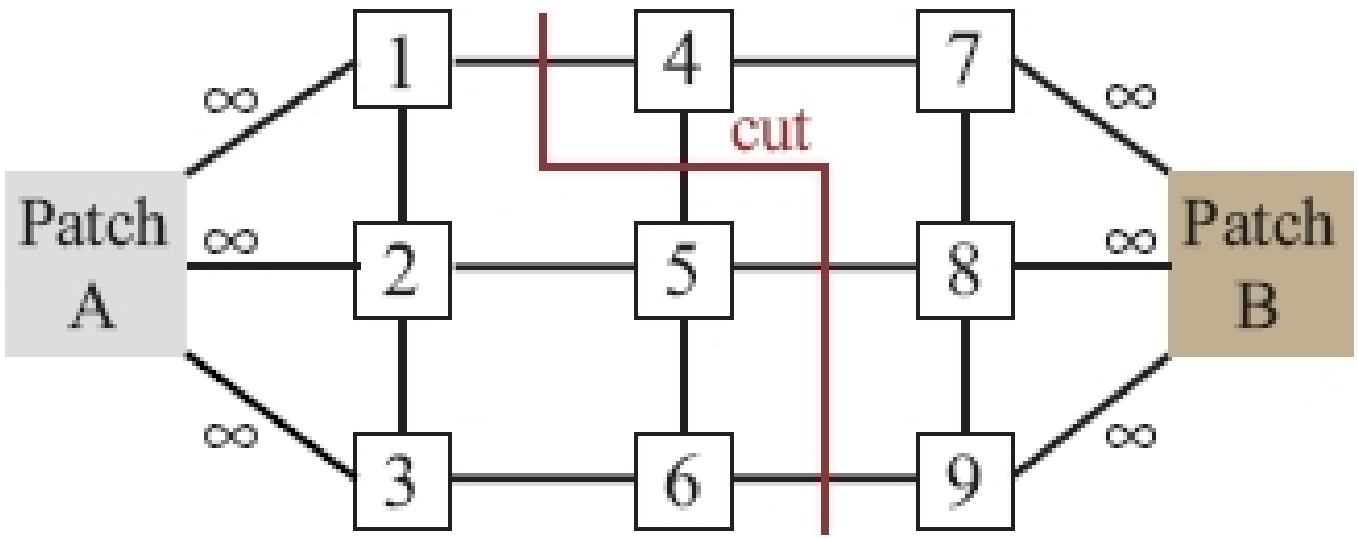
Connected graph



Disconnected graph

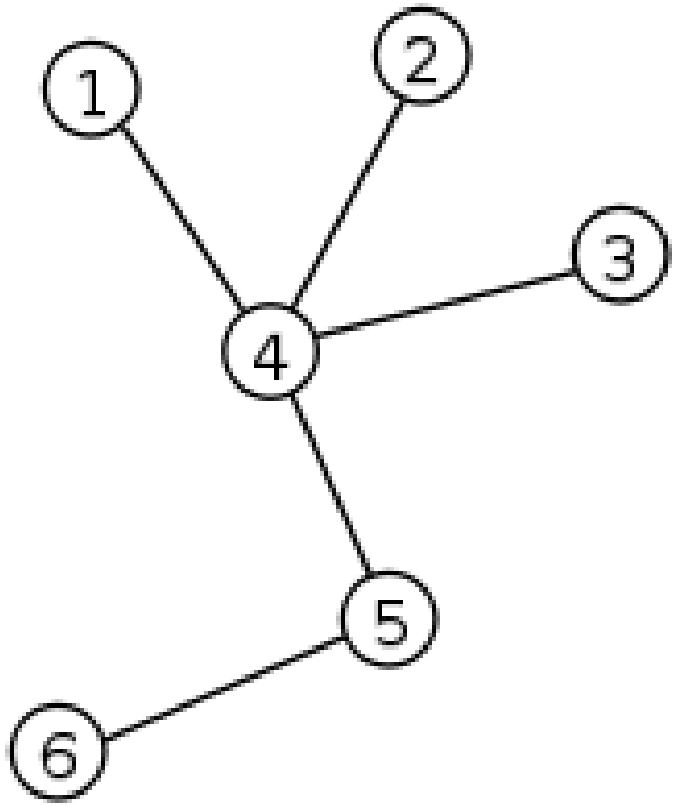
Cuts

- A cut is a partition of the node set N into two parts, S and $\bar{S} = N - S$
- The cut defines a cutset $[S, \bar{S}]$ which denotes the set of arcs which have one endpoint in S and the other in $\bar{S}$; thus $[S, \bar{S}]$ can refer to the cut or the cutset
- An $s - t$ cut is a cut $[S, \bar{S}]$ such that $s \in S$ and $t \in \bar{S}$



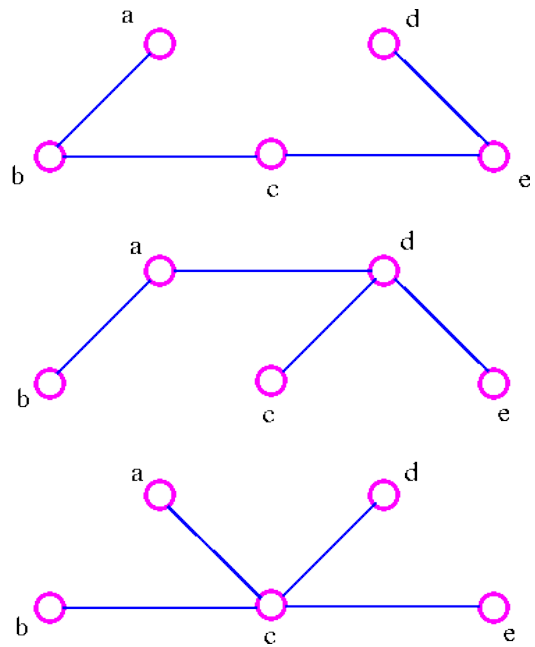
Trees

- A tree T is a connected graph that contains no cycles (directed or undirected)
- A leaf node of a tree is a node that has degree 1
- A forest graph is a graph that contains no cycles
- A subtree is a connected subgraph of a tree



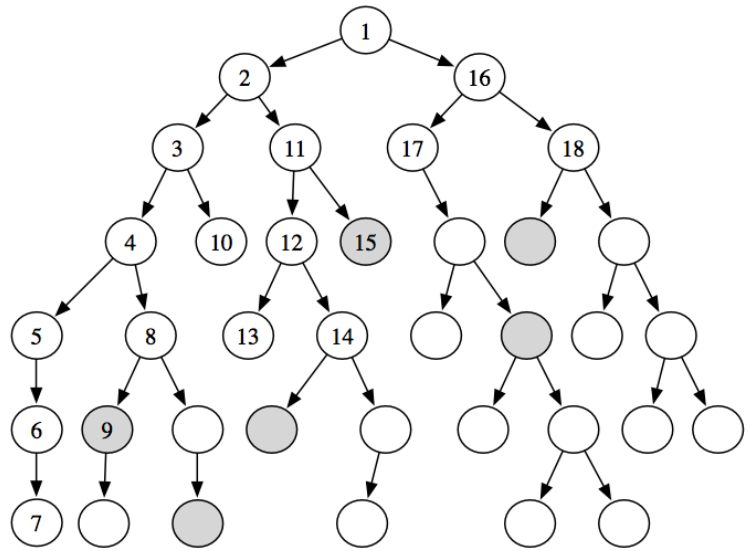
Trees (2)

- Tree properties
  - a tree on n nodes contains exactly $n - 1$ arcs
  - A tree has at least two leaf nodes
  - Every two nodes of a tree are connected by a unique path



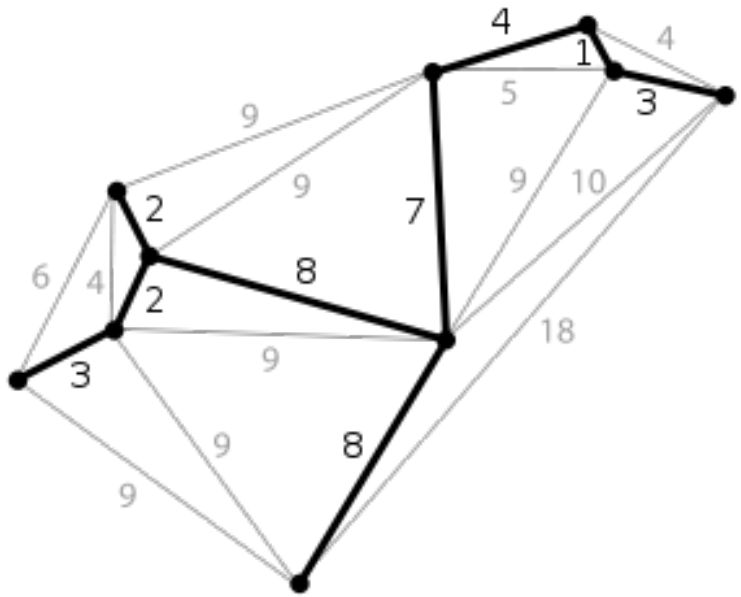
Trees (3)

- A rooted tree is a tree with a designated node, called the root
  - have parent-child relationship
  - each node has a set of descendents (including itself)
  - each node has a set of ancestors (including itself)
- A tree is a directed-out tree rooted at s if
  - every path from s to another node is a directed path
- A tree is a directed-in tree rooted at s if
  - every path from a node to node s is a directed path



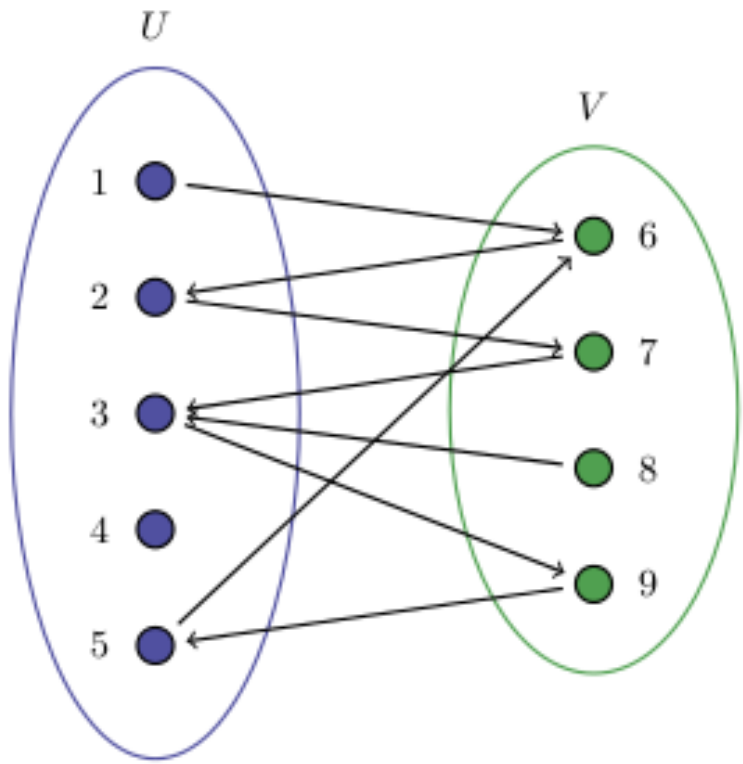
Spanning Tree

- A tree T is a spanning tree of G if T is a spanning subgraph of G
  - all of the arcs in a spanning tree T are tree arcs and the other arcs are non-tree arcs
- If we add a non-tree arc to T, then we create a cycle, namely a fundamental cycle
- A cut on a spanning tree is called a fundamental cut



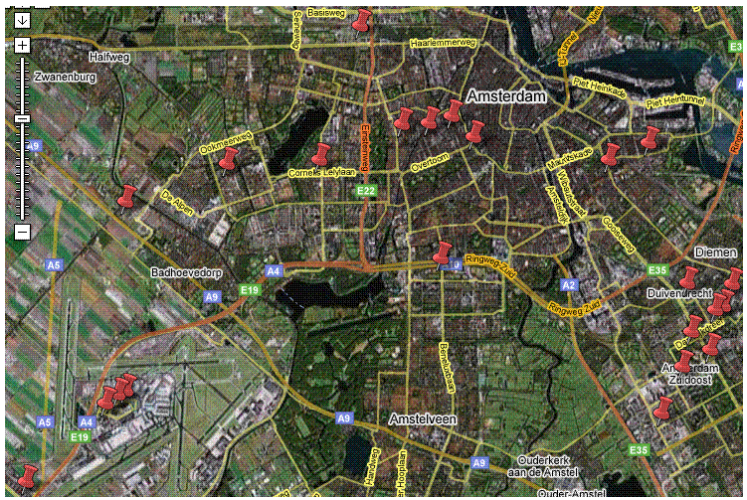
Bipartite Graph

- A graph $G = (N, A)$ is a bipartite graph if we can partition its node set into two subsets N_1 and N_2 so that for each arc (i, j) in A
  - $i \in N_1$ and $j \in N_2$ or
  - $i \in N_2$ and $j \in N_1$
- Key property
  - every cycle in G contains an even number of arcs



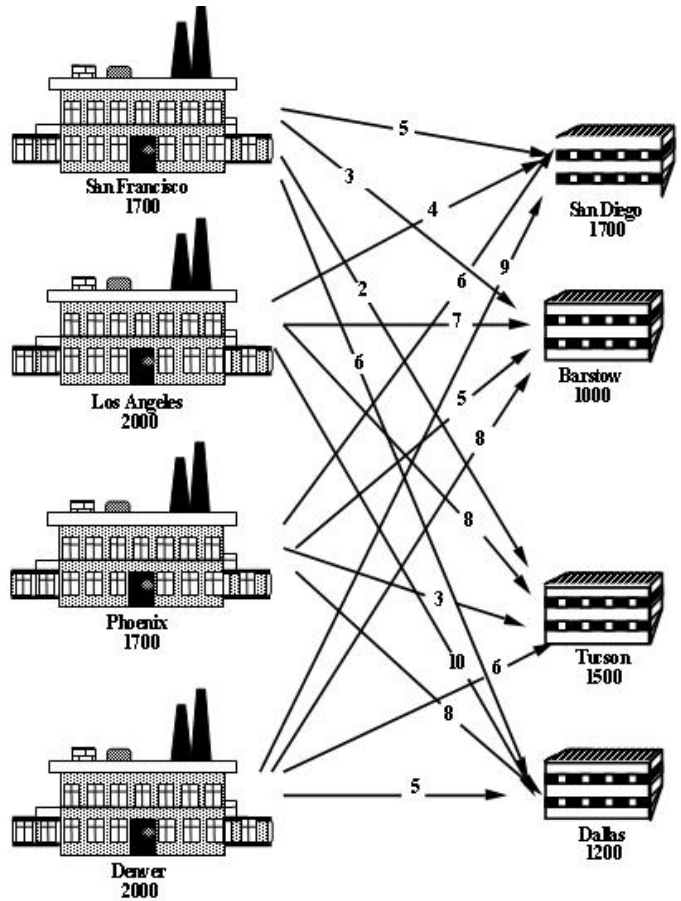
Physical Network

- Features
  - nodes have a spatial location
  - weight on arc typically represents travel time or cost
- Examples
  - street grid



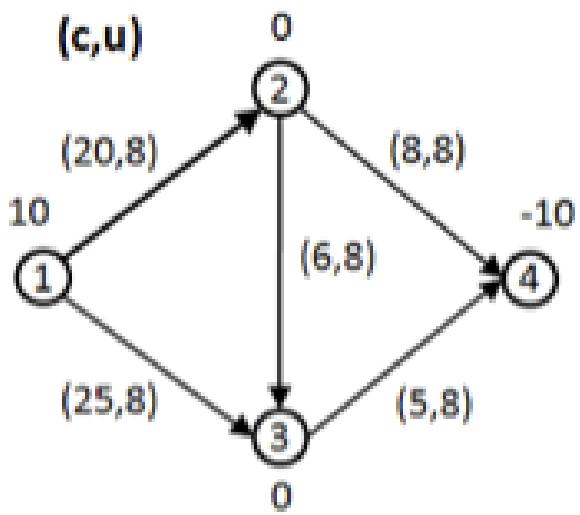
Route Networks

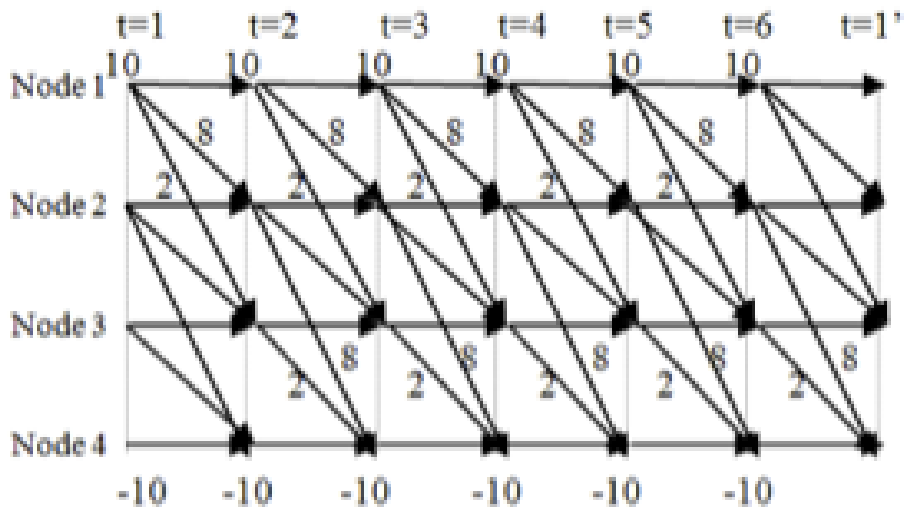
- Features
  - preprocessing is done on a physical network to convert it to a bipartite network
  - paths in physical are converted to arcs in bipartite network
  - bipartite network only contains sources (suppliers) and sinks (customers)
- Examples
  - supply chain logistics



Space-Time Networks

- Features
  - several nodes represent a facility but at different points in time
- Examples
  - economic lot sizing problem
  - airline scheduling problem

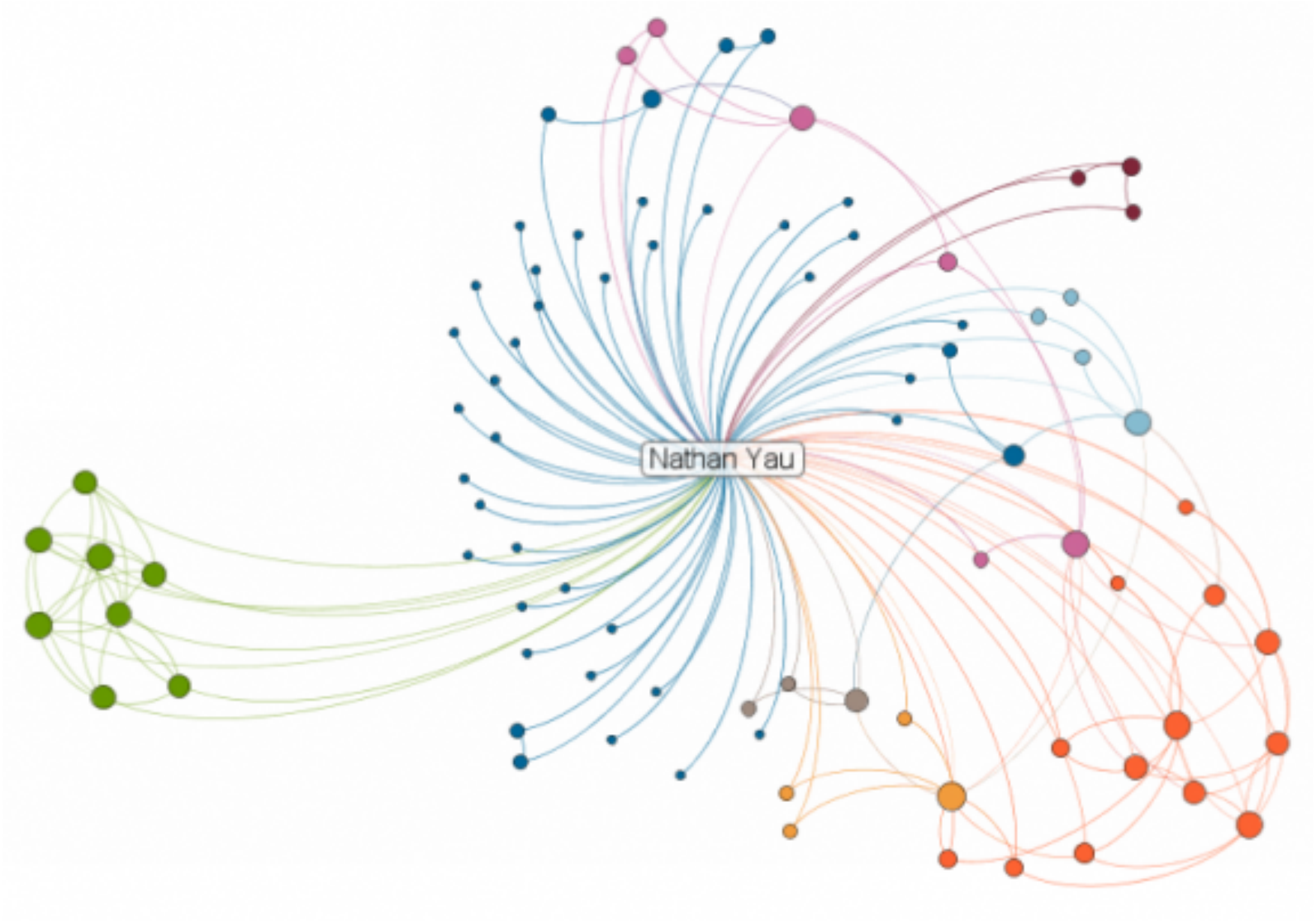




Social Networks

- Features
  - nodes represent persons or groups
  - arcs represent relationships
- Examples
  - actors who act together in a movie

Social Networks (2)



Summary

In summary, here are some examples of features that can be modeled using a network:

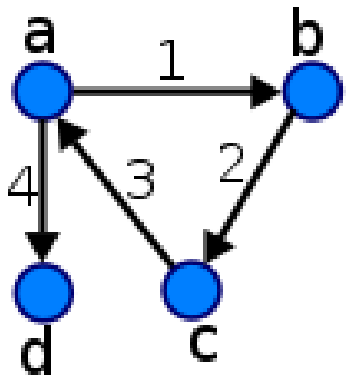
- Physical movement (transportation, water flow, electricity flow, etc.)
- Time relationships
- Precedence relationships
- Social relationships

Section 3

Network Representations

Node-Arc Incidence Matrix

- Rows represent nodes
- Columns represent arcs
- Values in cells
  - 1: arc flows out of node
  - -1: arc flows into node
  - 0: arc is not adjacent to node
- Advantages
  - represents the constraint matrix of a linear program
- Disadvantages
  - not efficient for sparse graphs



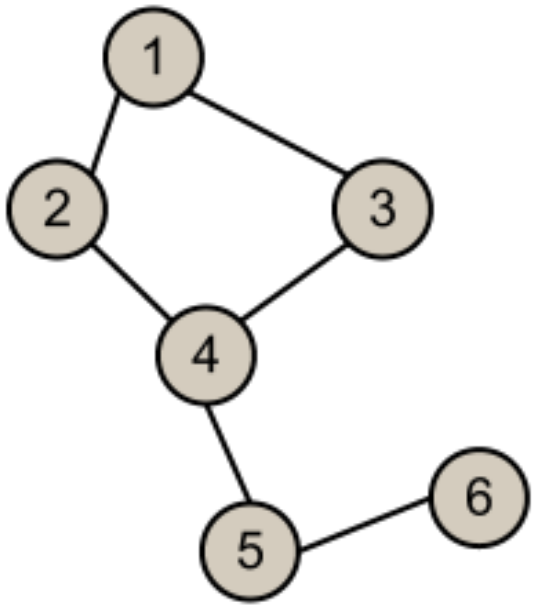
	1	2	3	4
a	1	0	-1	1
b	-1	1	0	0
c	0	-1	1	0
d	0	0	0	-1

Node-Node Adjacency Matrix

- Row-column pair represent a pair of nodes
  - 1: column and row nodes are adjacent
  - 0: nodes are not adjacent
- Advantages
  - simplicity
- Disadvantages
  - not efficient for sparse graphs

Node-Node Adjacency Matrix

Undirected Graph & Adjacency Matrix



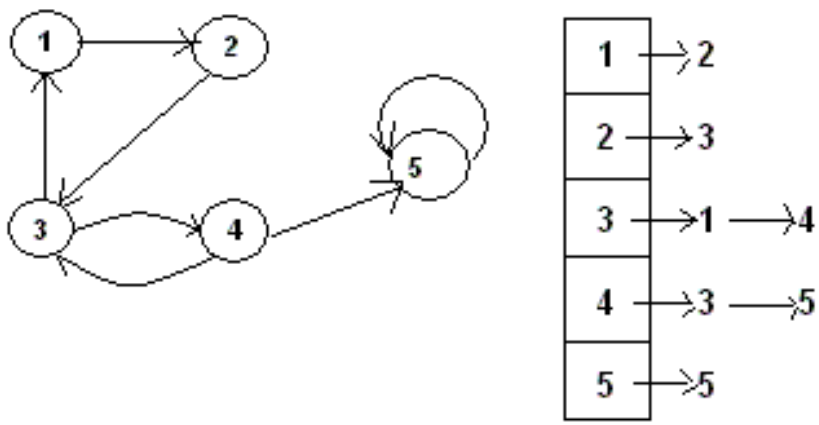
Undirected Graph

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	0	0
4	0	1	1	0	1	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

Adjacency Matrix

Adjacency Lists

- Stores the node adjacency for each node as a singly linked list
  - Advantages
    - more efficient use of storage than matrix data structures



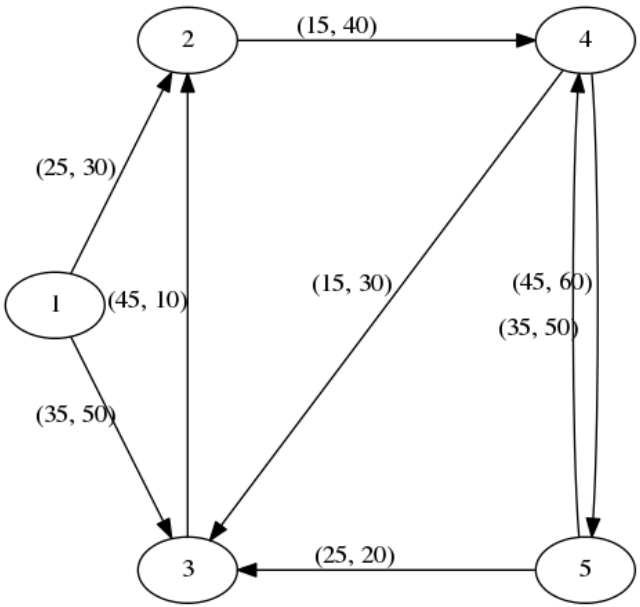
Forward and Reverse Star Representations

How It Works

- Order arcs in a specific order (first those emanating from node 1, then those emanating from node 2, etc.)
- Store information about each arc in an arc list
  - *tail*, *head*, *cost*, and *capacity*
- Maintain a pointer for each node i, denoted by $point(i)$ that indicates the smallest numbered arc in the arc list that emanates from node i
- Forward star (set of outgoing arcs) for node i is stored at positions $point(i)$ to $point(i+1)-1$ in the arc list

Forward and Reverse Star Representations

Example



	point
1	1
2	3
3	4
4	5
5	7
6	9

	tail	head	cost	capacity
1	1	2	25	30
2	1	3	35	50
3	2	4	15	40
4	3	2	45	10
5	4	3	15	30
6	4	5	45	60
7	5	3	25	20
8	5	4	35	50

Forward and Reverse Star Representations

More

- Reverse star is just the reverse of the forward star
- Advantages
  - more efficient use of storage than adjacency list
- Disadvantages
  - more difficult to implement

Section 4

Analyzing Algorithm Performance

Unit 1

Introduction to Algorithms

What You Need to Solve a Network Problem

- A problem instance, which is composed of
  - a Model of the problem (e.g., shortest path problem, etc.)
  - a network
- An algorithm

What is An Algorithm?

- A step-by-step procedure for solving a problem
- Operations
  - assignment (e.g., assigning a value to a variable)
  - arithmetic (e.g., addition)
  - logical (e.g., comparison)

Goals for Designing an Algorithm

- Time
- Storage space
- How do we measure an algorithm's time performance?
  - empirically
    - implement the an algorithm
    - run the algorithm on a set of problem instances
    - collect statistics on run time (mean, maximum, etc.)
  - theoretical
    - derive statistics on run time (mean, maximum, etc.)

Unit 2

Measuring Algorithm Performance

Empirical Analysis

- Advantages
  - easy to perform
- Disadvantages
  - depends on:
    - programming language
    - compiler
    - computer used
    - skill of programmer
    - problem instances chosen
  - time consuming

Average-case analysis

- Advantages
  - depends only on the probability distribution chosen to represent problem instances
- Disadvantages
  - depends on probability distribution chosen to represent problem instances
  - it is difficult to choose a meaningful probability distribution
  - math is difficult

Worst-case analysis

- Advantages
  - does not depend on the computing environment or any probability distribution
  - gives a performance guarantee
  - not as difficult to perform
- Disadvantages
  - is the bound tight?
  - how likely are we to see run times as bad as the bound?

Unit 3

Worst-Case Complexity

Big-O Notation

Definition

An algorithm is said to run in $O(f(n))$ time if for some numbers c and n_0, the time taken by the algorithm is at most $cf(n)$ for all $n \geq n_0$.

- Summary
  - provides an asymptotic bound
  - ignores constants
- Implications
  - need sufficiently large values of the problem size (e.g., n)
  - measures the asymptotic growth rate of the algorithm
  - only the dominant term is significant
  - we can assume that each elementary mathematical operation requires an equal amount of time
- Similarity assumption
  - we assume that the data are polynomially bounded

Polynomial Algorithms

Definition

Worst-case complexity is bounded by a polynomial function of the problem's parameters

- Variations
  - Strongly polynomial: depends only on n and m and does not involve $\log C$ or $\log U$
    - e.g., $\mathcal{O}(nm)$ and $\mathcal{O}(n \log n)$
  - Weakly polynomial: otherwise
    - e.g., $\mathcal{O}(n \log C)$
- Example
  - Dijkstra's algorithm for the shortest path problem has $O(n^2)$ worst-case complexity
- Implications
  - run time grows rather slowly as a function of the problem size
- A polynomial-time algorithm is generally considered a "good" algorithm

Exponential-Time Algorithms

Definition

worst-case running time grows as a function that cannot be polynomially bounded by the input length

- Examples
  - $\mathcal{O}(2^n)$, $\mathcal{O}(n!)$
- Implications
  - run time grows rapidly as a function of the problem size
- "Good" exponential-time algorithms
  - simplex algorithm for linear programming
- Variations
  - pseudopolynomial-time: running time is polynomially bounded in n, m, C, and U (e.g., $\mathcal{O}(m + nC)$)

Complexity Theory

- Does a polynomial-time algorithm exist for every problem?
- NP-complete problems
  - There is a huge collection of problems for which researchers have not been able to find polynomial-time algorithms
  - researchers have been able to show that most of these problems belong to a class of problems called NP-complete
  - the implication of this findings is that if you find a polynomial-time algorithm for one problem, there exists a polynomial time algorithm for all of them, proving wrong many smart people