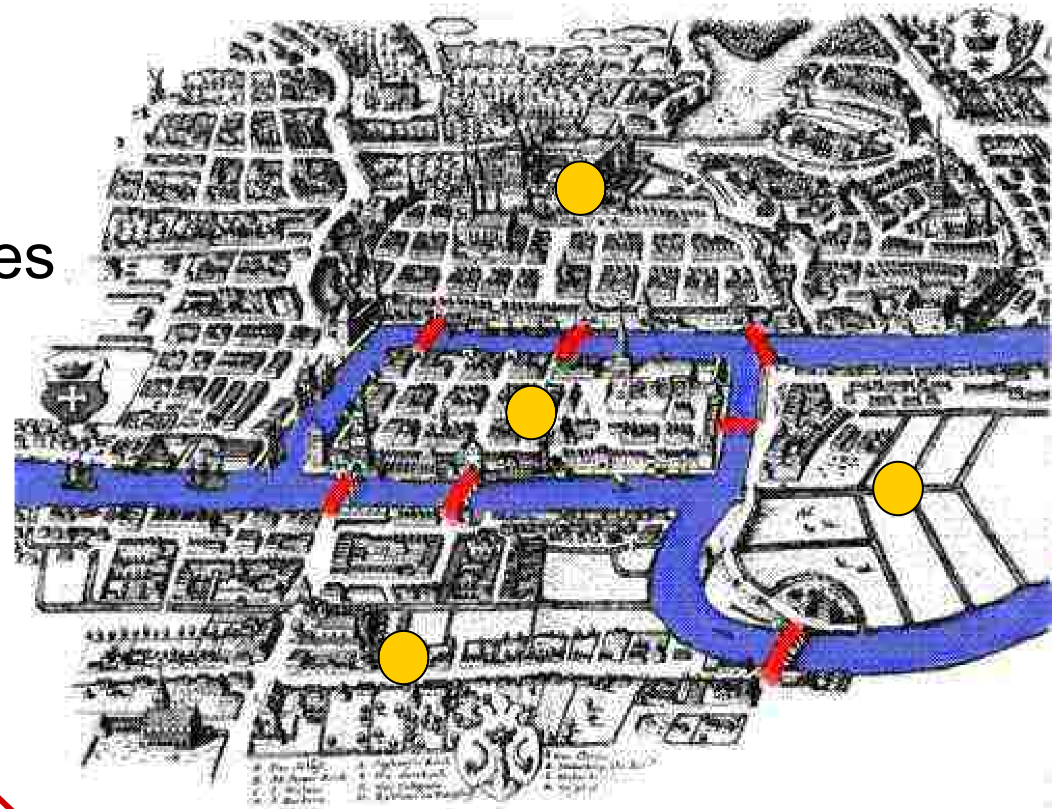
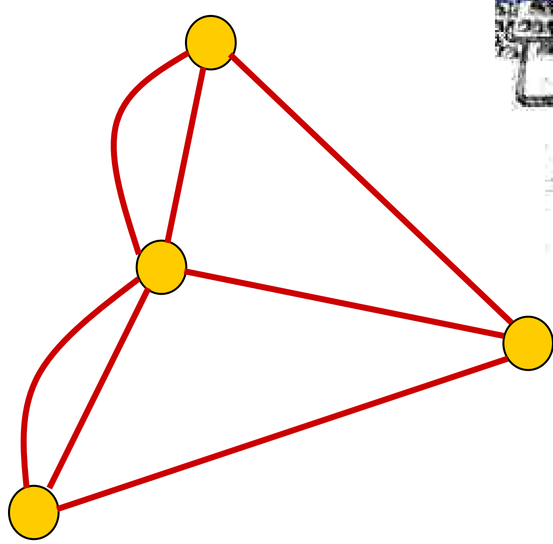
GRAPH THEORY and APPLICATIONS

Basic Concepts

A bit of History…

- Father of graph theory, Euler
  - Konigsberg bridges problem (1736)

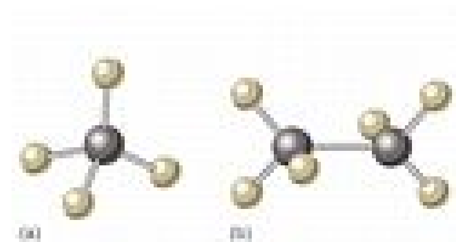


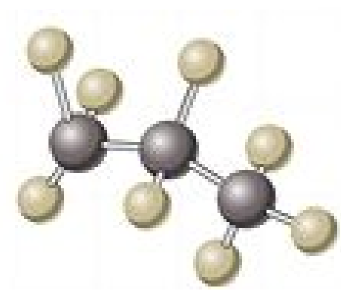


Kirchhoff and Cayley

- Kirchhoff developped the theory of trees in 1847 to solve the linear equations in branches and circuits of an electric network.

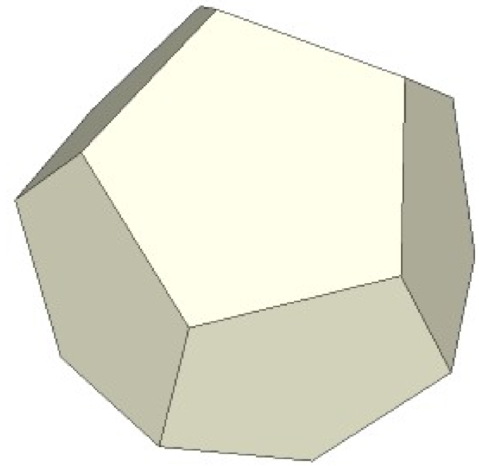
- In 1857, Cayley discovered the trees. Later he engaged in enumerating the isomers of saturated hyrocarbons with a given number of carbon atoms.

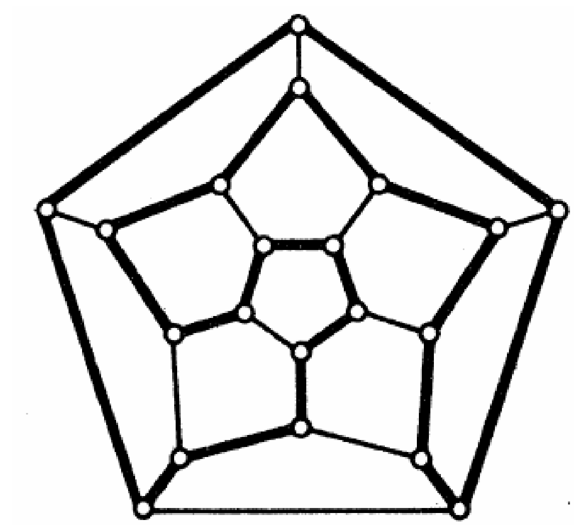




A game

- In 1859, Hamilton used a regular solid dodecahedron whose 20 corners are labeled with famous cities.
- The player is challenged to travel *"around the world"* by finding a closed circuit along the edges, passing through each city exactly once.





Applications

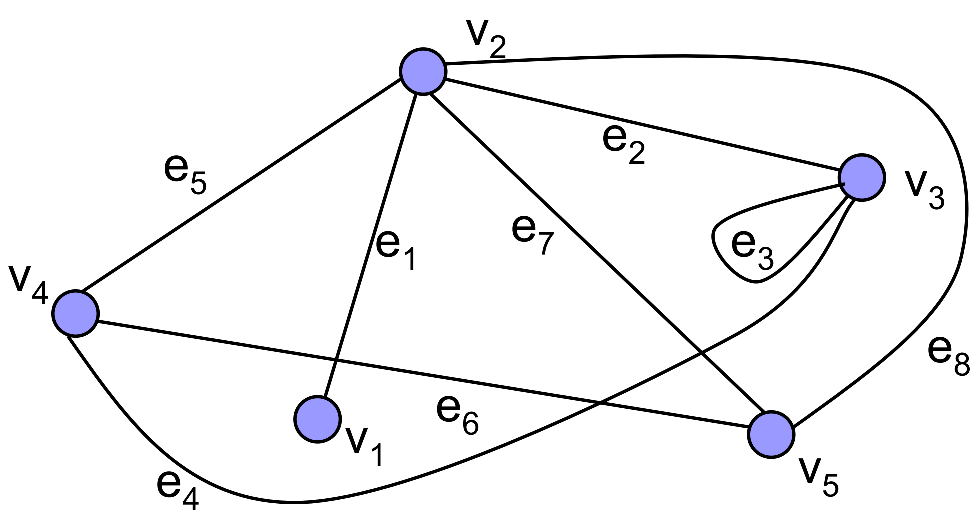
- Psychology, Lewin 1936, life-space of an individual
- Theoretical physics
- Probability, Markov chains
- Study of network flows
- Gant charts

Graphs

- A diagram consisting of:
  - A set of points
  - Lines joining certain pairs of these points
- Example:
  - Points: people; lines: joining pairs of friends
  - Points: communication centers;
    lines: communication on links
- Graph: G is an ordered triple (V, E, ψ_G)
  V: nonempty set of vertices
  E: set of edges
  ψ_G: incidence function

Example of a Graph

- $V = \{v_1, v_2, v_3, v_4, v_5\}$
- $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$
- $\psi_G(e_1) = v_1v_2$, $\psi_G(e_2) = v_2v_3$, $\psi_G(e_3) = v_3v_3$
  $\psi_G(e_4) = v_3v_4$, $\psi_G(e_5) = v_2v_4$, $\psi_G(e_6) = v_4v_5$
  $\psi_G(e_7) = v_2v_5$, $\psi_G(e_8) = v_2v_5$



There is no unique way of drawing a graph.

Terminology

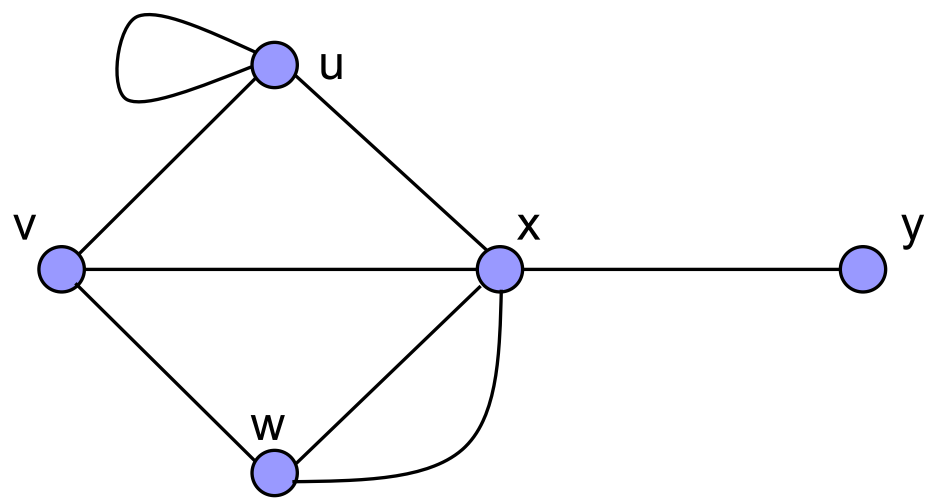
- Two vertices which are incident with a common edge are adjacent.
- An edge with identical ends: a loop.
- An edge with distinct ends: a link.
- Finite graph: both the vertex set and edge set are finite.
- Simple graph: it has no loops and no two of its links join the same pair of vertices.

Isomorphism

- Two graphs G and H are isomorphic if there are bijections:
  - Θ: V(G) → V(H)
  - Φ: E(G) → E(H)

  such that:

  $\psi_G(e) = uv$ if and only if $\psi_H(\Phi(e)) = \Theta(u)\,\Theta(v)$

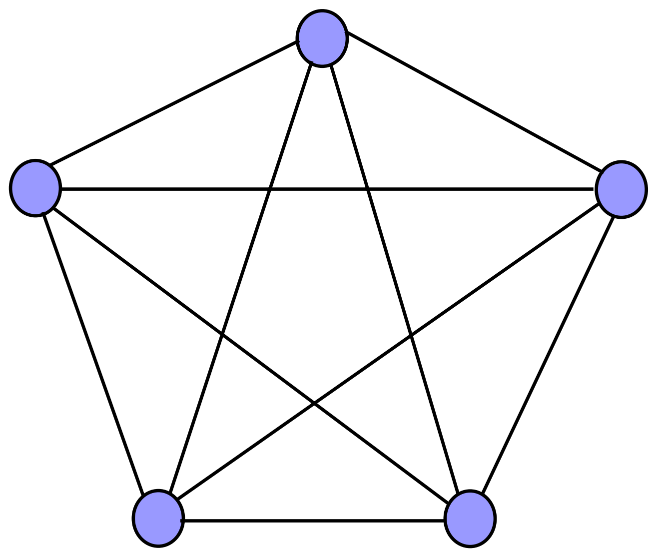


This graph is isomorphic to (has the *same structure* with) the graph in slide 7.

Complete Graph

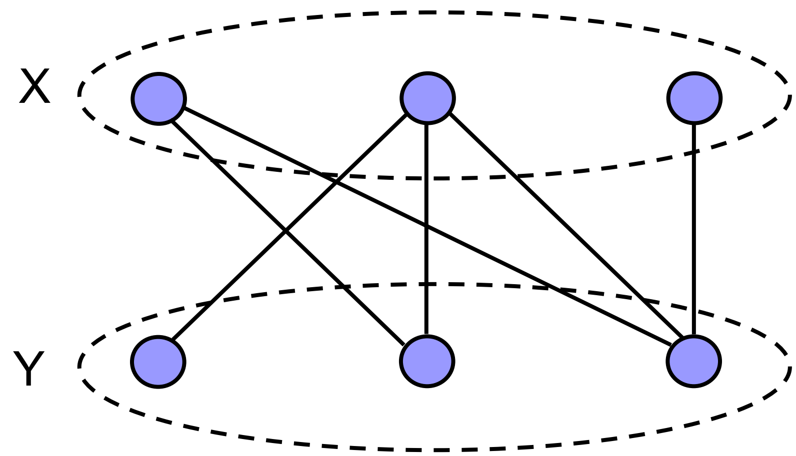
- Simple graph
- Each pair of vertices is joined by an edge
- Complete graph of n vertices: K_n

K_5



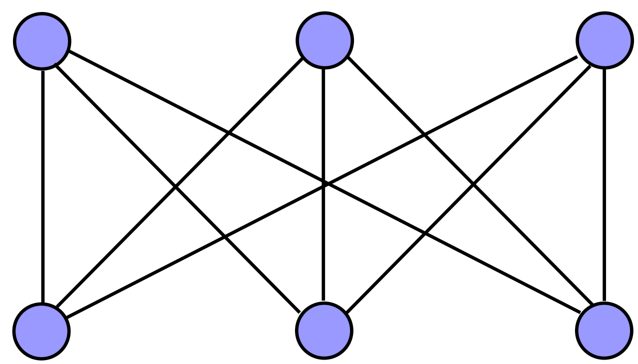
Bipartite Graph

- Empty graph: a graph with no edges.
- Bipartite graph:
  - Vertex set can be partitioned into two sets X and Y.
  - Each edge has one end in X and one end in Y.



Complete Bipartite Graph

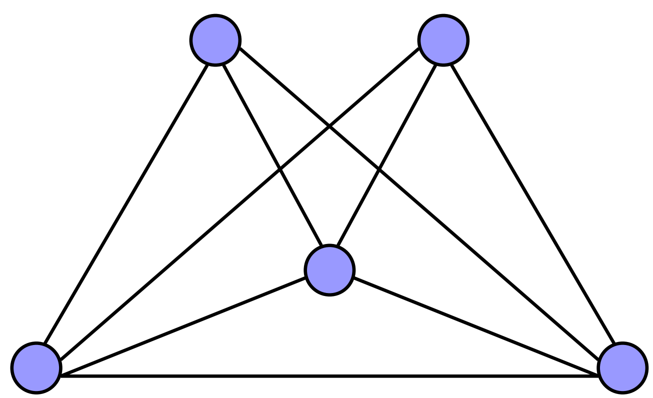
- Complete bipartite graph: each vertex of X is joined to each vertex of Y.
  - Denoted by $K_{m,n}$



$K_{3,3}$

Planar Graph

- Two edges in a diagram of a graph may intersect at a point that is not a vertex
- Graphs that have a diagram whose edges intersect only at their ends are called planar.



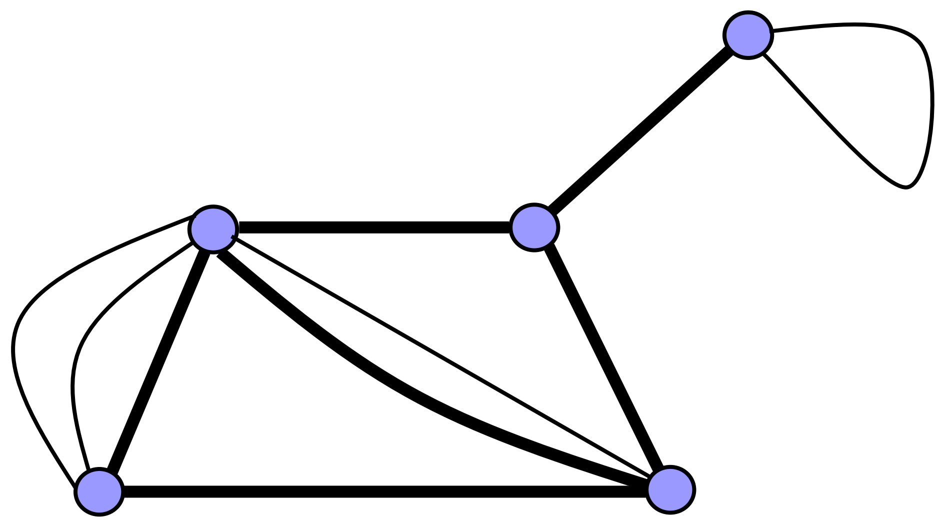
A planar graph

Subgraphs

- H is a subgraph of G if:
  - $V(H) \subseteq V(G)$,
  - $E(H) \subseteq E(G)$,
  - ψ_H is the restriction of ψ_G to $E(H)$.
- When $H \neq G$, H is a proper subgraph of G.
- If H is a subgraph of G, then G is a supergraph of H.
- Spanning subgraph of G is a subgraph H with $V(H) = V(G)$.

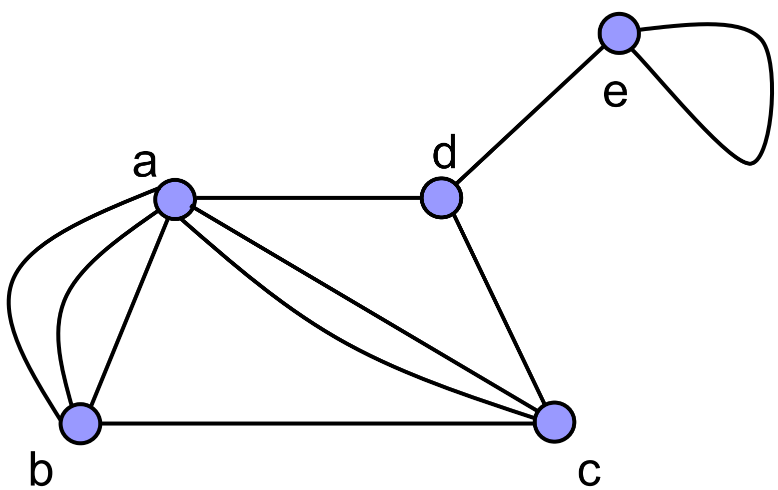
Subgraphs

- Underlying simple graph is obtained by deleting all loops and all parallel edges between node pairs except one.

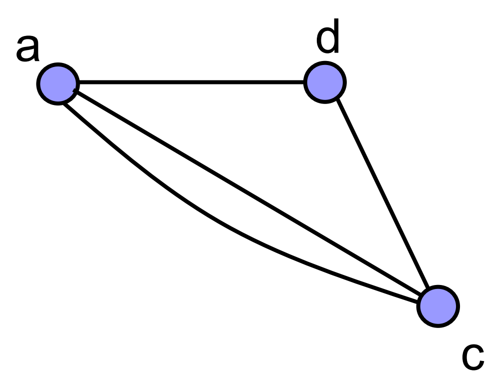


Induced Subgraph

- The induced subgraph, denoted by G-V’, is obtained from G by deleting the vertices in V’ together with their incident edges.



G



G – {b,e}

Edge/Vertex Disjoint

- Let G_1 and G_2 be subgraphs of G.
- G_1 and G_2 are disjoint if they have no vertex in common.
- They are edge-disjoint if they have no edge in common.

Vertex Degree

- Degree: number of edges incident with a vertex
  - each loop counts as two.

Theorem:

$$\sum_v d(v) = 2e \quad e\text{: number of edges}$$

Theorem: In any graph, the number of vertices of odd degree is even.

- A graph is k-regular if d(v)=k for all v∈V.
  - regular graphs, regular bipartite graphs $K_{n,n}$

Paths

- Walk: A finite non-null sequence:

  $$W = v_0 e_1 v_1 e_2 \ldots e_k v_k$$

  - terms are alternately vertices and edges
    for $1 \le i \le k$ the ends of e_i are v_{i-1} and v_i.
  - The vertices v_0 and v_k are called the origin and terminus of W.
- A walk in a simple graph can be specified simply by its *vertex sequence*.

Paths

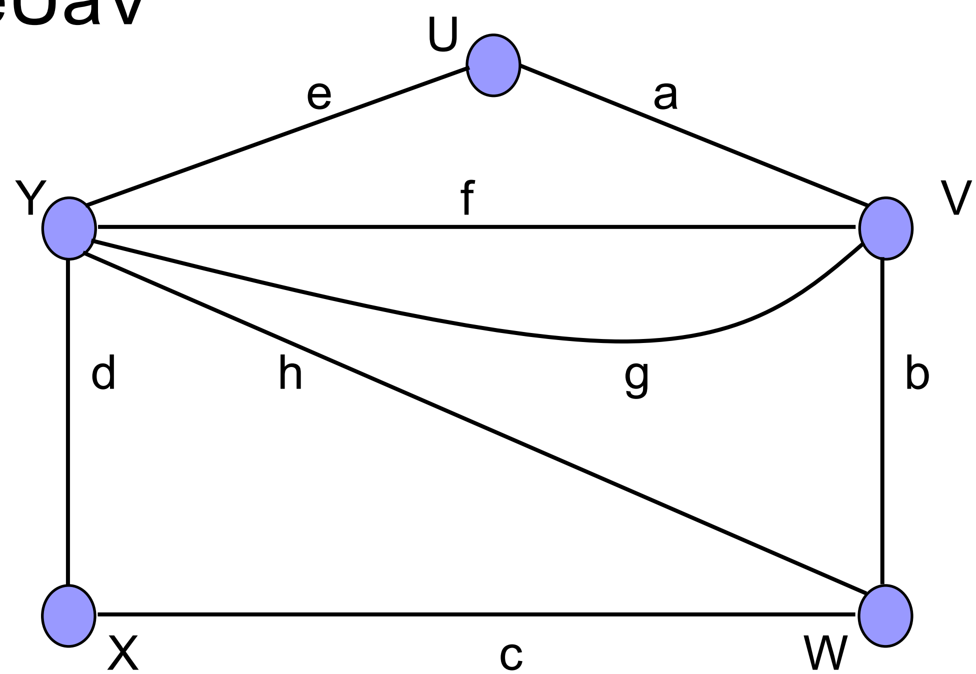
- Trail: W is a trail, if the edges e_1, e_2, …, e_k of the walk are distinct.
- Path: If the vertices of a trail are distinct, it is called a path.
- Two vertices u and v of a graph are connected if there is a path (u,v).
- If all pairs are connected, then graph is also connected.

Example

walk: UaVfYfVgYhWbV

trail: WcXdYhWbVgY

path: XcWhYeUaV



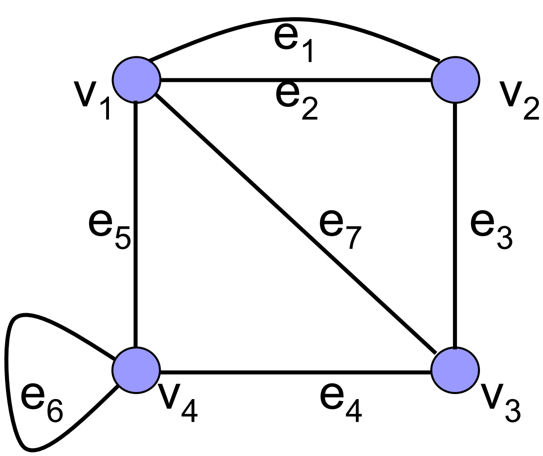
Distance, Diameter, Cycle

- The distance between u and v, $d_G(u,v)$ is the length of a shortest (u,v) path.
- The diameter of G is the *maximum distance* between two vertices of G.
- A walk is closed if its origin and terminus are the same.
- A closed path is called a cycle.
  - k-cycle: A cycle of length k.

Theorem: A graph is bipartite if and only if it contains no odd cycle.

Incidence and Adjacency Matrices

- Vertices: $v_1, v_2, \ldots, v_\nu$
- Edges: $e_1, e_2, \ldots, e_\varepsilon$
- Incidence matrix: $M_G = [m_{ij}]$ where m_{ij} is the number of times that v_i and e_j are incident.
- Adjacency matrix: $A_G = [a_{ij}]$ where a_{ij} is the number of edges joining v_i and v_j.



Incidence matrix

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
v_1	1	1	0	0	1	0	1
v_2	1	1	1	0	0	0	0
v_3	0	0	1	1	0	0	1
v_4	0	0	0	1	1	2	0

Adjacency matrix

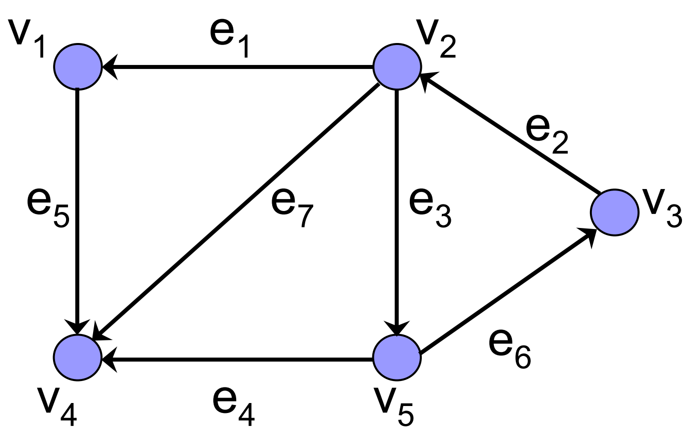
	v_1	v_2	v_3	v_4
v_1	0	2	1	1
v_2	2	0	1	0
v_3	1	1	0	1
v_4	1	0	1	1

Directed graphs

- If each edge has a direction, the graph is called a digraph.
  - The edge (u,v) is different from edge (v,u).
  - The degree of a vertex v:
    - in-degree $d^-(v)$: number of edges incident to v
    - out-degree $d^+(v)$: number of edges incident from v
  - The digraph is balanced if for every vertex v, $d^-(v) = d^+(v)$
- Each digraph has an *underlying undirected graph*, obtained by deleting the direction of its edges.

Directed Path

- In a digraph, a directed path is an alternating sequence of vertices and edges:
  $$S = v_1 e_1 v_2 e_2 \ldots v_{k-1} e_{k-1} v_k$$
  where for all i, $1 \leq i < k$, e_i is incident
  - from v_i
  - to v_{i+1}
- Otherwise, S is an undirected path.



Undirected path:

$v_1 v_4 v_5 v_2 v_3$

Directed path:

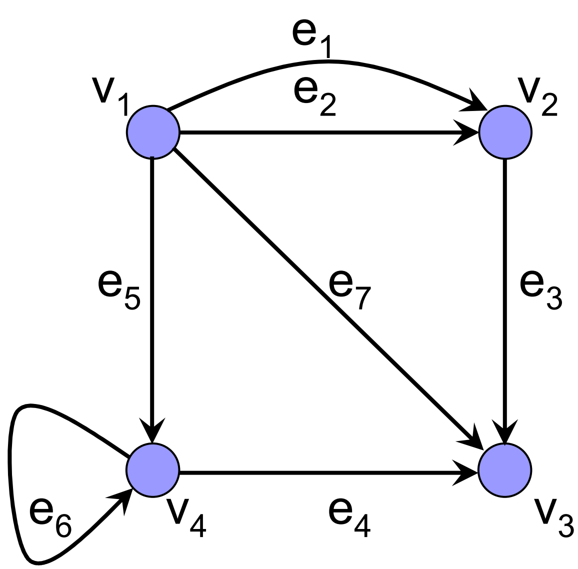
$v_5 v_3 v_2 v_4$

Connectivity in Digraphs

- Two types of connectivity:
  - Strongly connected
    u and v are strongly connected if there is:
    - a directed (u,v) path, and
    - a directed (v,u) path
  - Weakly connected
    u and v are weakly connected if there is:
    - an undirected (u,v) path

Adjacency Matrix of a Digraph

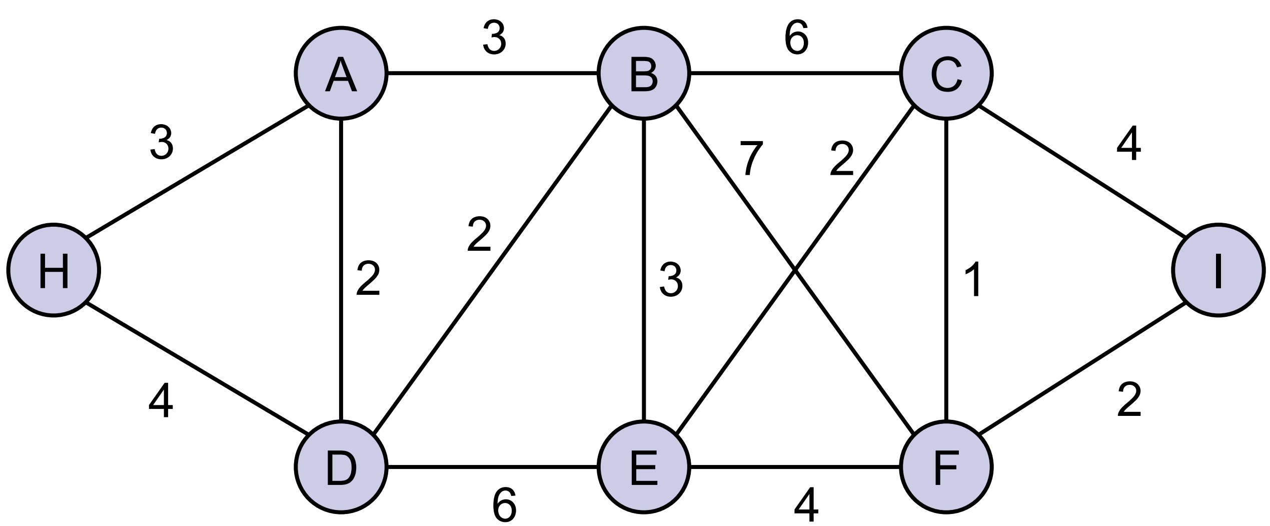
- Vertices: $v_1, v_2, \ldots, v_\nu$
- Edges: $e_1, e_2, \ldots, e_\varepsilon$
- Adjacency matrix: $A_G = [a_{jk}]$ where a_{jk} is the number of edges incident from v_j to v_k.



	v_1	v_2	v_3	v_4
v_1	0	2	1	1
v_2	0	0	1	0
v_3	0	0	0	0
v_4	0	0	1	1

Weighted Graphs

- Each edge is assigned a number.
  - cost, weight, length
- Weight of a subgraph: Sum of all edges of the subgraph
  - Example: weight of a path



Algorithmic Complexity

- Complexity: Number of computational steps that it takes to transform the input data to the result of a computation.
  - This is a function of the problem size.
- For graph algorithms, the problem size is determined by one or both
  - number of nodes
  - number of edges.

Algorithmic Complexity

- For a problem size s, the complexity of an algorithm A is $C_A(s)$.
  - The complexity may vary significantly if A is applied to structurally different graphs of the same size.
  - We use worst-case complexity:
    The maximum number of computational steps, over all inputs of size s.

Asymptotic Growth

- Let A_1 and A_2 be two algorithms for the same problem.
  - $C_{A1}(n) = n^2/2$
  - $C_{A2}(n) = 5n$
  - A_2 is faster than A_1 for all $n>10$.
- Asymptotic growth: As the problem size tends to infinity, growth of n^2 is greater than n.
- The complexity of A_2 is of lower order than that of A_1.

Order

- Given two functions F and G whose domain is the natural numbers,
  - The order of F is lower than or equal to the order of G if:

$$F(n) \leq K \,.\, G(n) \quad \text{for } n > n_0$$

K and n_0 are positive constants.

We write: F = O(G)

- Low order terms of a function can be ignored in determining the overall order.
  - Example: $3n^3 + 6n^2 + n + 6$ is $O(n^3)$

Comparing Two Functions

- Let:

$$\lim_{n \to \infty} \frac{F(n)}{G(n)} = L$$

  - If L = a finite positive constant, then F = Θ(G)
  - If L = 0, then F is of lower order than G.
  - If L = ∞, then G is of lower order than F.

Examples

- Compare $F(n)=3n^2 - 4n + 2$ and $G(n)=n^2/2$

$$\lim_{n\to\infty} \frac{3n^2 - 4n + 2}{n^2/2} = 6$$

then $F=\Theta(G)$.

- Compare $F(n)=\log_2 n$ and $G(n)=n$

$$\lim_{n\to\infty} \frac{\ln n}{n} \cdot \log_2 e = \lim_{n\to\infty} \frac{1/n}{1} \log_2 e = \lim_{n\to\infty} \frac{\log_2 e}{n} = 0$$

$\log_2 n$ is of lower order than n.

Comparison of Complexities

- It can be shown that:
  - An exponential in n is of greater order than any polynomial in n.
  - Factorial n is of greater order than exponential in n.

	2	8	128	1024
n	2	8	128	1024
n.logn	2	24	896	10240
n^2	4	64	16384	1048576
n^3	8	512	2097152	2^{30}
2^n	4	256	2^{128}	2^{1024}
n!	2	40320	$\sim 5x2^{714}$	$\sim 7x2^{8766}$

Efficiency vs. Intractability

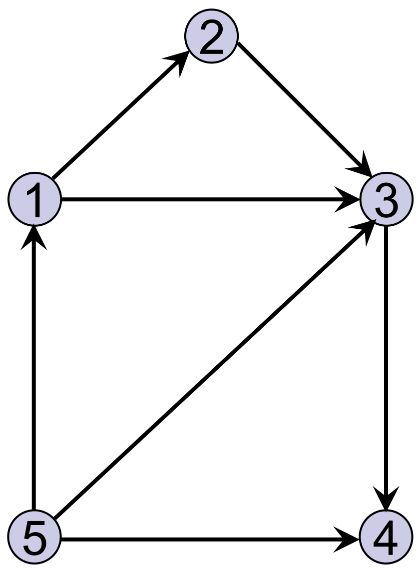
- Any O(P)-algorithm, where P is a polynomial in the problem size, is an efficient algorithm.
- Any problem for which
  - no polynomial-time algorithm is known,
  - it is conjectured that no such algorithm exists,

  is an intractable problem.

Graph Representation

- Adjacency matrices
  - 2-D Arrays
- Adjacency lists
  - Each vertex has a list of its adjacent vertices.
  - Tables or linked lists (doubly linked lists)

Example – Digraph representation



Adjacency matrix

A =

0	1	1	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	0
1	0	1	1	0

Adjacency lists

1 [2 |] → [3 | 0]

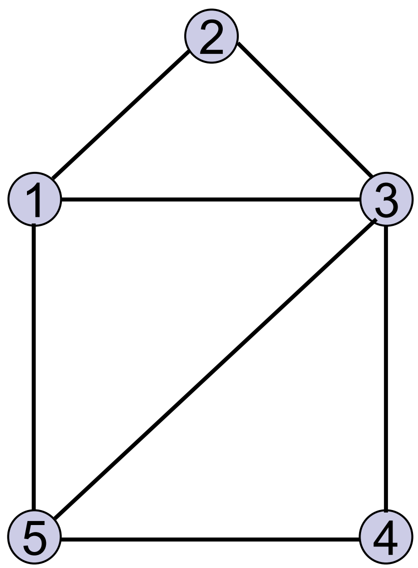
2 [3 | 0]

3 [4 | 0]

4 empty list

5 [1 |] → [3 |] → [4 | 0]

Example-Undirected graph representation

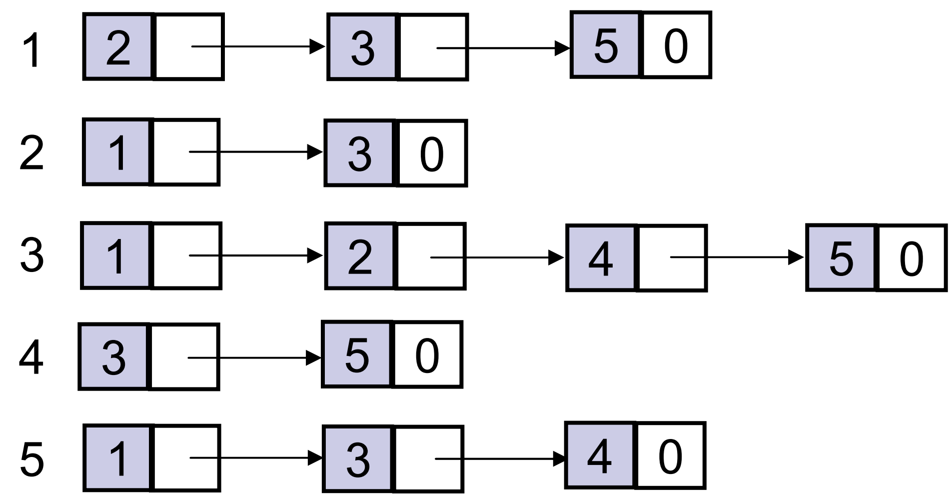


Adjacency matrix

A =

0	1	1	0	1
1	0	1	0	0
1	1	0	1	1
0	0	1	0	1
1	0	1	1	0

Adjacency lists



Products of Adjacency Matrix

- A^k: k-th matrical product of the adjacency matrix

$$A^k = A^{k-1} \times A$$

where

$$A^1 = A$$

Theorem: $A^k(i,j)$ is the number of walks from i to j, containing k edges.

Connection Matrix

- If graph G has n vertices, then the number of walks of *length < n* can be found as follows:

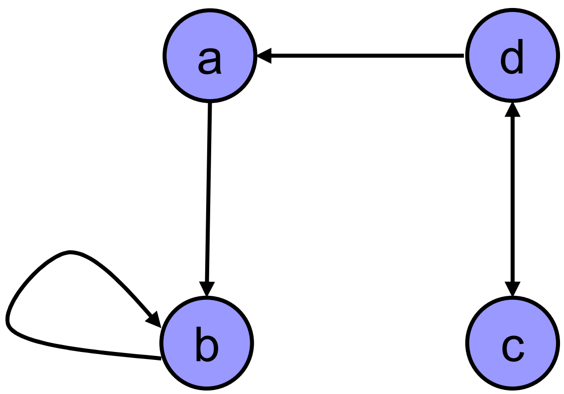
$$A^0 + A^1 + A^2 + A^3 + \ldots + A^{n-1}$$

- The connection matrix C of a graph of n vertices:
  - an nxn matrix
  - element (i,k) is 1 if there is a path from v_i to v_k
- C can be calculated using the above formula.

Warshall’s Algorithm

- Finding the connection matrix
  - Will not give the number of walks, only the connectivity
- For each vertex v:
  - There is a walk:
    - from each vertex that can reach v
    - to each vertex that can be reached from v.
  - Check the corresponding column of the matrix for 1’s
  - Match them to 1’s in the corresponding raw.

Example



	a	b	c	d
a	0	1	0	0
b	0	1	0	0
c	0	0	0	1
d	1	0	1	0

	a	b	c	d
a	0	1	0	0
b	0	1	0	0
c	0	0	0	1
d	1	1	1	0

	a	b	c	d
a	0	1	0	0
b	0	1	0	0
c	0	0	0	1
d	1	1	1	0

	a	b	c	d
a	0	1	0	0
b	0	1	0	0
c	0	0	0	1
d	1	1	1	1

	a	b	c	d
a	0	1	0	0
b	0	1	0	0
c	1	1	1	1
d	1	1	1	1

Graph Traversals

- Depth first search
  - Systematic method of visiting the vertices of a graph
  - Finds all reacheable nodes starting from a node.
  - Backtracking
  - Recursive programming or stack required

```
DFS(u):
Mark u explored
for each edge (u,v) incident to u do
   if v is not marked explored then
        Recursively invoke DFS(v)
   endif
endfor
```

Home study:

- Read
  - Gibbons, Section 1.3.2
- Research
  - Breadth-first search