



BRACU BYTE

Department of Computer Science and Engineering

Chairperson: Dr. Mumit Khan

Note Book Compiled by

S.Mahbub – Uz – Zaman (Ananda)

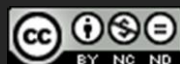
ID: 09301004

Tanjina Islam (Micky)

ID: 09301018

This is a small effort to develop our Programming Skills for the ACM

GRAPH THEORY



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License](https://creativecommons.org/licenses/by-nc-nd/3.0/)

INDEX

✓ An Introduction to Graph

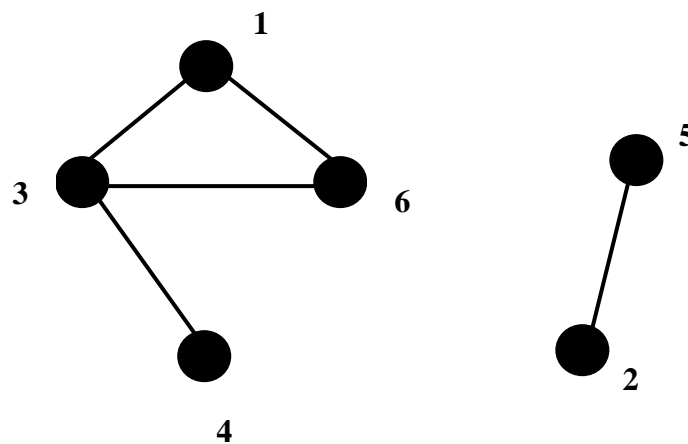
1. Graph
2. Types of graphs
3. Graph Representation's
4. Data Struture Tradeoffs
5. Special Thanks
5. References

An Introduction to GRAPH

Graph

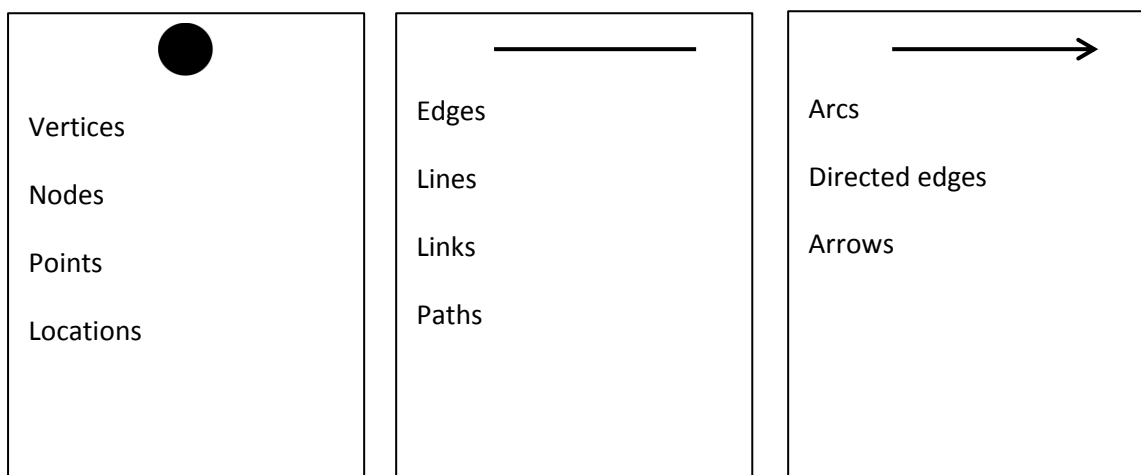
A graph is a collection of vertices of V and a collection of edge E consisting of pairs of vertices. A graph represented $G = (V, E)$

Think of vertices as locations. The set of vertices is the set of all the possible locations. In this analogy, edges represent paths between pairs of those locations. The set E contains all the paths between the locations.



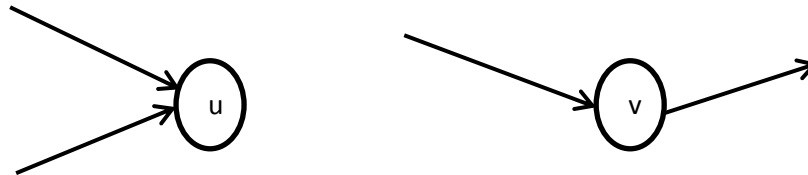
$$V = \{1, 2, 3, 4, 5, 6\}$$

$$E = \{(1, 3), (1, 6), (2, 5), (3, 1), (3, 4), (4, 3), (5, 2), (6, 1), (6, 3)\}$$

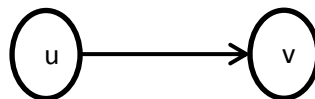


Adjacent:

Two edges of a graph are called **adjacent** (sometimes **coincident**) if they share a common vertex. Two arrows of a directed graph are called **consecutive** if the head of the first one is at the notch (notch end) of the second one.

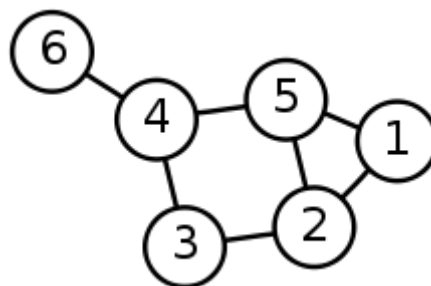


Two vertices are called **adjacent** if they share a common edge (**consecutive** if they are at the notch and at the head of an arrow), in that case the common edge is said to **join** the two vertices.



Pendent:

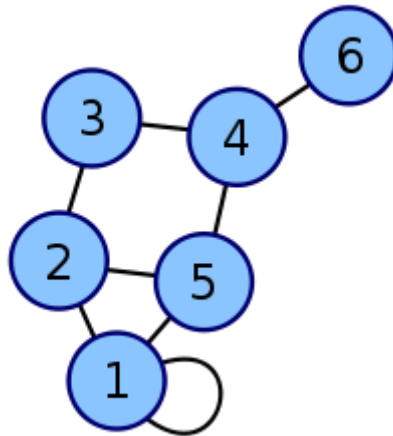
A vertex of degree one (with only one edge connected) is a pendant edge.



Here the vertex number 6 is a leaf vertex or a pendent vertex.

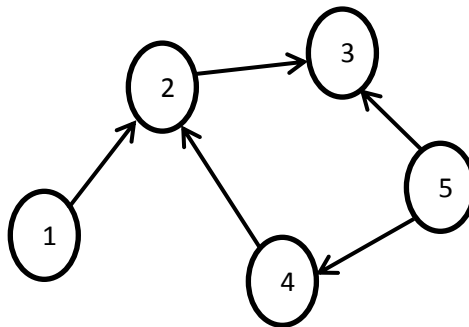
Self – loop:

A loop (also called a self-loop or a "buckle") is an edge that connects a vertex to itself. A simple graph contains no loops. Here Vertex 1 has a loop.



Degree:

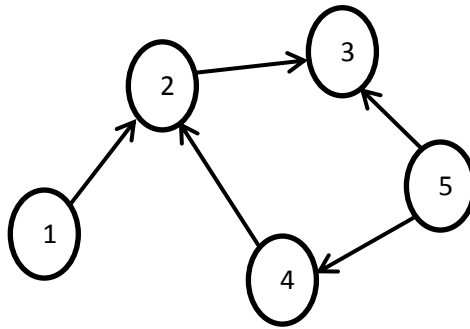
The **degree** (or **valency**) of a vertex of a graph is the number of edges incident to the vertex, with loops counted twice. The degree of a vertex v is denoted $\deg(v)$. The maximum degrees of a graph G , denoted by $\Delta(G)$, and the minimum degree of a graph, denoted by $\delta(G)$.



Vertex 4 has degree 2 and vertex 1 has degree 1.

In – degree:

For a node, the number of head endpoints adjacent to a node is called the **indegree** of the node. The indegree is denoted $\deg^-(v)$. A vertex with $\deg^-(v) = 0$ is called a **source**, as it is the origin of each of its incident edges (a **source vertex** is a vertex with indegree zero).



$$\deg^-(1) = 0$$

$$\deg^-(2) = 2$$

$$\deg^-(3) = 2$$

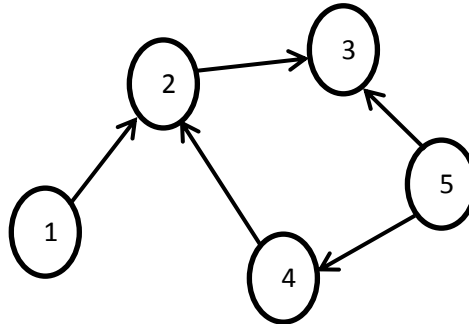
$$\deg^-(4) = 1$$

$$\deg^-(5) = 0$$

$$\sum_{v \in V} \deg^-(v) = 5$$

Out – degree:

For a node, the number of tail endpoints is its **outdegree**. The outdegree denoted as $\deg^+(v)$. A vertex with $\deg^+(v) = 0$ is called a **sink** (**sink vertex** is a vertex with outdegree zero).



$$\deg^+(1) = 1$$

$$\deg^+(2) = 1$$

$$\deg^+(3) = 0$$

$$\deg^+(4) = 1$$

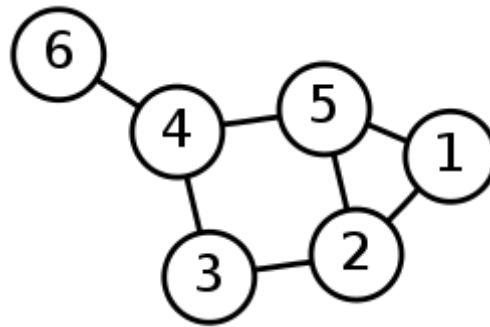
$$\deg^+(5) = 2$$

$$\sum_{v \in V} \deg^+(v) = 5$$

Hand Shaking lemma (Theorem):

In any graph, the sum of all the vertex-degree is equal to twice the number of edges

$$\sum_{v \in V} \deg(v) = 2 |E|$$



In this graph, an even number of vertices (the four vertices numbered 2, 4, 5, and 6) have odd degrees. The sum of the degrees of the vertices is $2 + 3 + 2 + 3 + 3 + 1 = 14$, twice the number of edges.

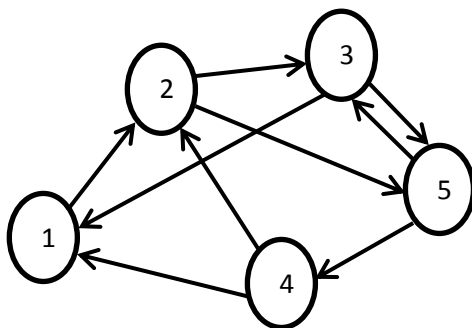
The name “Hand Shaking Theorem” comes from the following party analogy: Consider a collection of guests at a party. Suppose some guest shook hands with some other guest. If we asked everyone at the party how many guests they shook hands with and added those numbers all up, this sum would be equal to twice the number of total hand shakes. In graph theory terms, each vertex represents a guest, and an edge between two guests represents a hand shake between them.

Degree sum Theorem:

The **degree sum formula** states that, for a directed graph,

$$\sum_{v \in V} \deg^{-}(v) = \sum_{v \in V} \deg^{+}(v) = |A|$$

If for every node $v \in V$, $\deg^{+}(v) = \deg^{-}(v)$, the graph is called a **balanced digraph**.



$$\deg^{-}(1) = 1$$

$$\deg^{-}(2) = 2$$

$$\deg^{-}(3) = 2$$

$$\deg^{-}(4) = 1$$

$$\deg^{-}(5) = 2$$

$$\sum_{v \in V} \deg^{-}(v) = 8$$

$$\deg^{+}(1) = 1$$

$$\deg^{+}(2) = 2$$

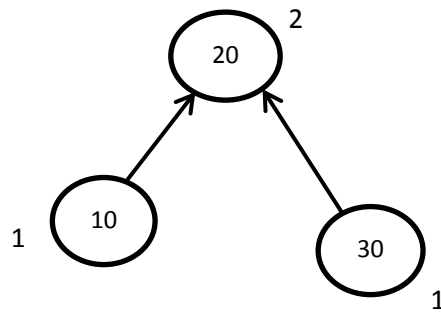
$$\deg^{+}(3) = 2$$

$$\deg^{+}(4) = 1$$

$$\deg^{+}(5) = 2$$

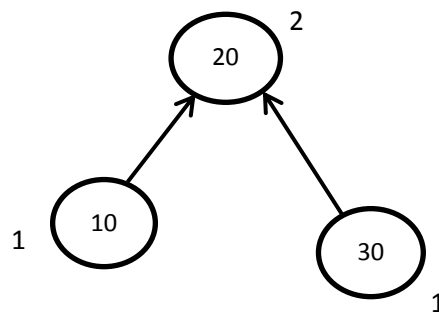
$$\sum_{v \in V} \deg^{+}(v) = 8$$

- In any graph, the sum of all the vertex-degree is an even number.



$$1 (10) + 2 (20) + 1 (30) = 4 \text{ [even]}$$

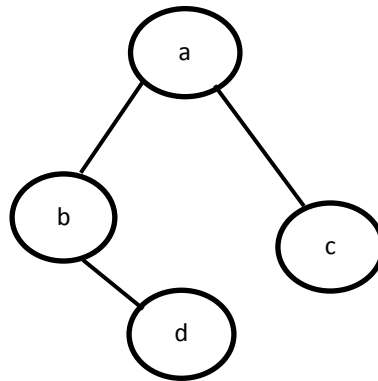
- In any graph, the number of vertices of odd degree is even.



Here, vertices 10 and 30 have odd degree. So number is even (2)

Sparse:

A graph is said to be **sparse** if the total number of edges is small compared to the total number possible $\frac{n(n-1)}{2}$.

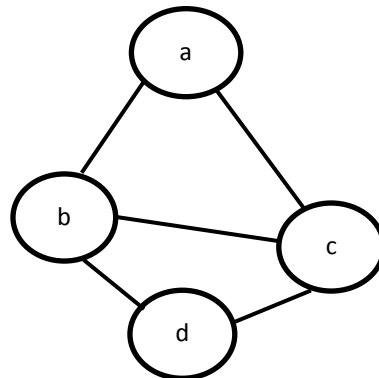


$$4(3)/2 = 6$$

Missing 3 edges

Dense:

A graph is said to be **dense** if the total number of edges is almost equal compared to the total number possible $\frac{n(n-1)}{2}$.

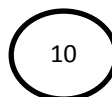


$$4(3)/2 = 6$$

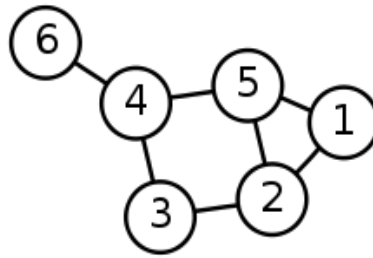
Missing 1 edge

Isolated:

A vertex of degree zero (with no edges connected) is isolated.

**Chromatic Number:**

The smallest number of colors needed to color a graph G is called its **chromatic number**, and is often denoted $\chi(G)$. Sometimes $\gamma(G)$ is used,



Walks:

A **walk** is an alternating sequence of vertices and edges, beginning and ending with a vertex, where each vertex is incident to both the edge that precedes it and the edge that follows it in the sequence, and where the vertices that precede and follow an edge are the end vertices of that edge. A walk is **closed** if its first and last vertices are the same, and **open** if they are different. In the example graph, $(1, 2, 5, 1, 2, 3)$ is an open walk with length 5, and $(4, 5, 2, 1, 5, 4)$ is a closed walk of length 5.

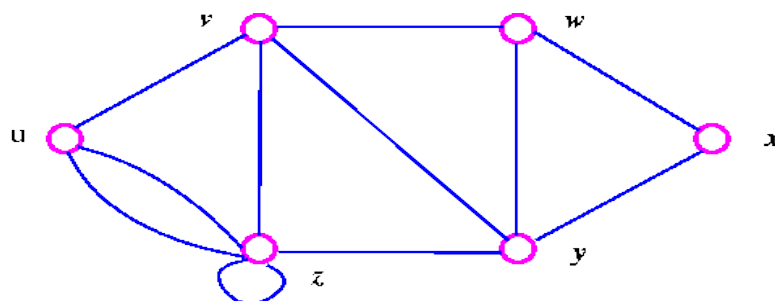
Paths (usually known as an open walk):

A path is a sequence of consecutive edges in a graph and the length of the path is the number of edges traversed.

Path is usually understood to be **simple**, meaning that no vertices (and thus no edges) are repeated. (The term **chain** has also been used to refer to a walk in which all vertices and edges are distinct.) In the example graph, $(5, 2, 1)$ is a path of length 2

Trail:

If all the edges (but not necessarily all the vertices) of a walk are different, then the walk is called a trail

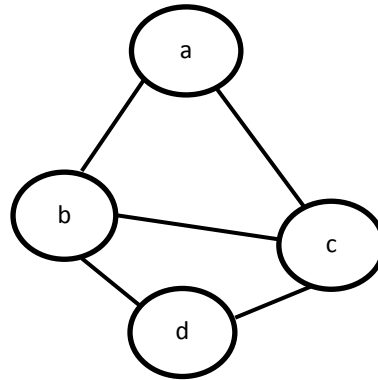


The walk $vzz ywx y$ is a trail since the vertices y and z both occur twice.

The walk $vwxyz$ is a path since the walk has no repeated vertices.

Biconnected Graph:

In graph theory, a **biconnected graph** is a connected and "nonseparable" graph, meaning that if any vertex were to be removed, the graph will remain connected. Therefore a biconnected graph has no articulation vertices. The use of **biconnected** graphs is very important in the field of networking (see Network flow), because of this property of redundancy.



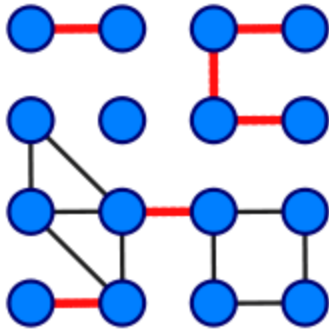
Articulation Point:

a node u is an articulation point, if for every child v of u , there is no back edge from v to a node higher in the DFS tree than u .

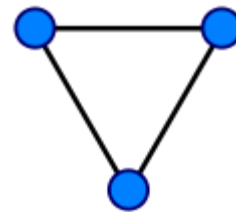
A connected graph is *biconnected* if the removal of any single vertex (and all edges incident on that vertex) can not disconnect the graph. More generally, the biconnected components of a graph are the maximal subsets of vertices such that the removal of a vertex from a particular component will not disconnect the component. Unlike connected components, vertices may belong to multiple biconnected components: those vertices that belong to more than one biconnected component are called *articulation points* or, equivalently, *cut vertices*. Articulation points are vertices whose removal would increase the number of connected components in the graph. Thus, a graph without articulation points is biconnected. The following figure illustrates the articulation points and biconnected components of a small graph:

Bridge:

In graph theory, a **bridge** (also known as a **cut-edge** or **cut arc** or an **isthmus**) is an edge whose deletion increases the number of connected components.^[1] Equivalently, an edge is a bridge if and only if it is not contained in any cycle. A graph is said to be **bridgeless** if it contains no bridges.



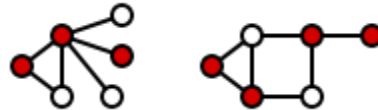
A graph with 6 bridges
(highlighted in red)



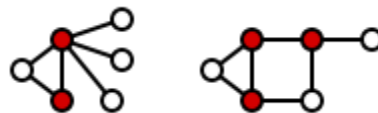
An undirected connected
graph with no cut edges

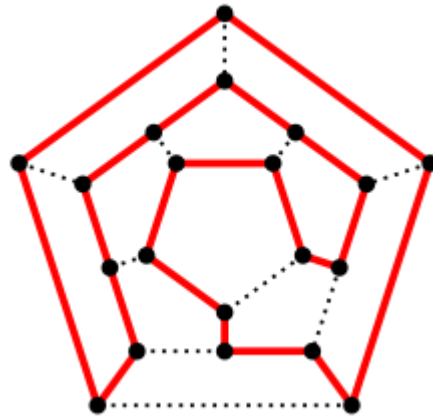
Minimum Vertex Cover:

Formally, a vertex cover of a graph G is a set C of vertices such that each edge of G is incident to at least one vertex in C . The set C is said to *cover* the edges of G . The following figure shows examples of vertex covers in two graphs (and the set C is marked with red).



A **minimum vertex cover** is a vertex cover of smallest possible size. The **vertex cover number** τ is the size of a minimum vertex cover. The following figure shows examples of minimum vertex covers in two graphs.

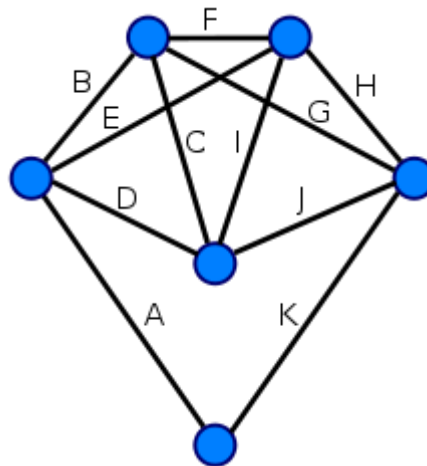




Hamilton path & Hamilton circuit:

A **Hamiltonian path** (or **traceable path**) is a path in an undirected graph that visits each vertex exactly once. A **Hamiltonian cycle** (or **Hamiltonian circuit**) is a cycle in an undirected graph which visits each vertex exactly once and also returns to the starting vertex.

Eulerian Graph:



Every vertex of this graph has an even degree; therefore this is a Eulerian graph.
Following the edges in alphabetical order gives an Eulerian circuit/cycle

Euler path:

An **Euler path** is defined as a path in a graph which visits every edge exactly once, although it may visit a vertex more than once and has different starting and ending vertices.

Euler cycle:

An **Euler cycle (Euler circuit/ tour)** is that which starts and ends on the same vertex.

A graph which has either an Euler path or an Euler cycle is called an Eulerian graph.

Eulers Theorem:

Euler Path Theorem

If a connected graph has exactly two vertices of odd degree, then it has a non-circuit Euler path, which must start and end at the odd-degree vertices.

Euler Circuit Theorem

A (connected) graph has an Euler circuit if and only if all its vertices have even degree (indegree (v) = outdegree (v)).

N.B A graph with more than 2 odd vertices does not contain any Euler path or circuit.

Cycle:

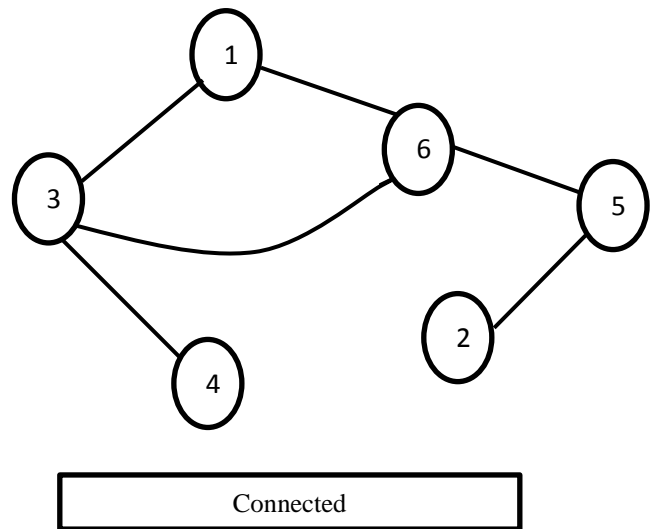
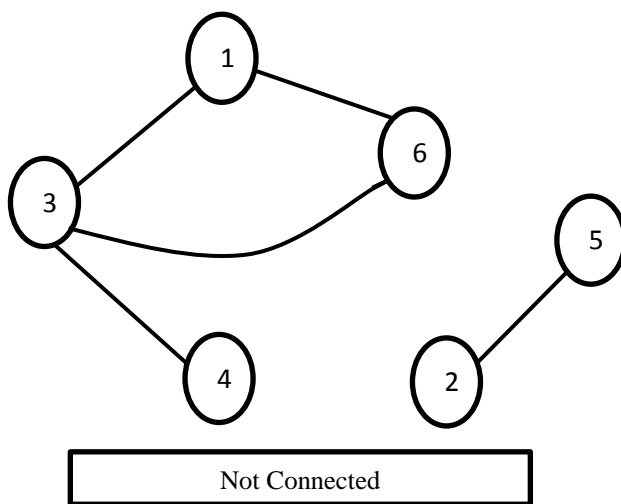
A walk that starts and ends at the same vertex but otherwise has no repeated vertices or edges, is called a **cycle**. Like *path*, this term traditionally referred to any closed walk. In the example graph, (1, 5, 2, 1) is a cycle of length 3. (A cycle, unlike a path, is not allowed to have length 0.)

Acyclic:

A graph is **acyclic** if it contains no cycles.

Unicyclic:

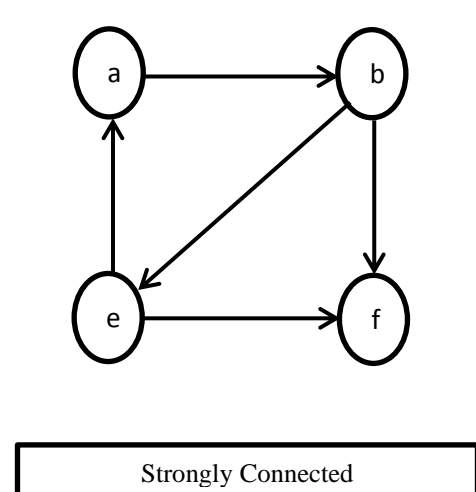
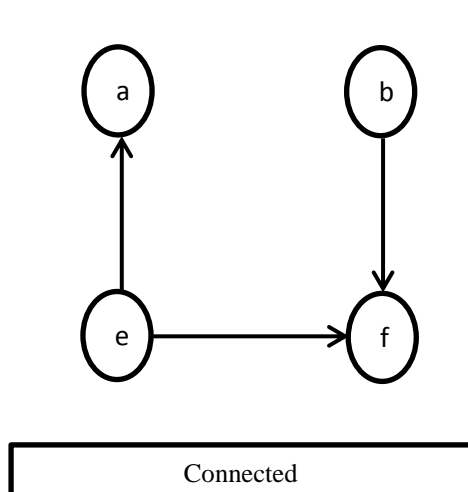
A graph contains exactly one cycle.

Connected graph:

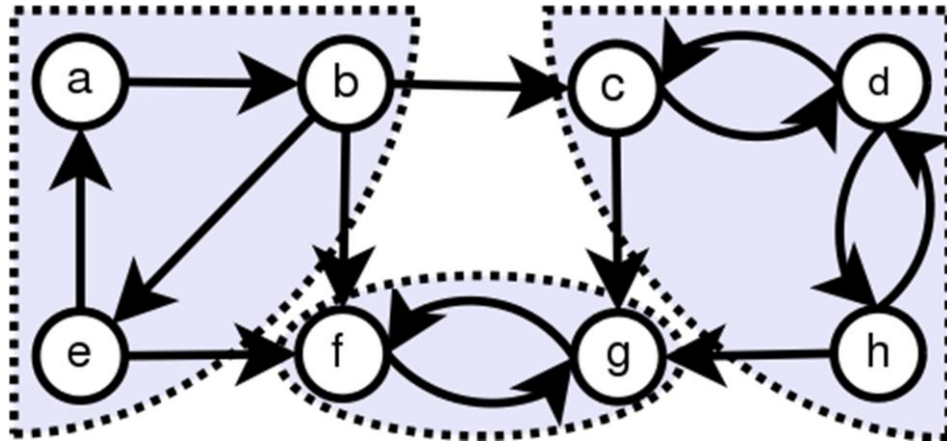
An undirected graph is said to be *connected* if there is a path from every vertex to every other vertex. The example graph is not connected, as there is no path from vertex 2 to vertex 4. However, if you add an edge between vertex 5 and vertex 6, then the graph becomes connected.

A **connected component** of an undirected graph of a graph is a maximal subset of the vertices such that every vertex is reachable from each other vertex in the component. The original example graph has two components: $\{1, 3, 4, 6\}$ and $\{2, 5\}$. Note that $\{1, 3, 4\}$ is not a component, as it is not maximal.

A directed graph is called *strongly connected* if there is a path from each vertex in the graph to every other vertex. In particular, this means paths in each direction; a path from a to b and also a path from b to a .



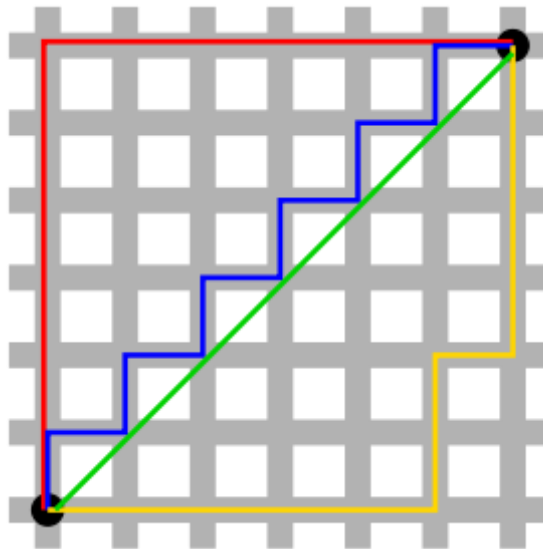
Strongly connected component is (SCC) A subset, S , of the nodes of a directed graph such that any node in S is reachable from any other node in S and S is not a subset of any larger such set.



Graph with strongly connected components marked

Manhattan Distance:

The distance between two points measured along axes at right angles. In a plane with p_1 at (x_1, y_1) and p_2 at (x_2, y_2) , it is $|x_1 - x_2| + |y_1 - y_2|$.



Red: **Manhattan distance**.

Green: diagonal, straight-line distance.

Blue, yellow: equivalent Manhattan distances.

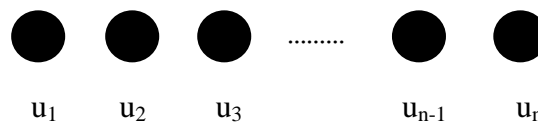
Types of graphs

1. Null Graph or Empty Graph

The graph with no vertices and (hence) no edges, or any graph with no edges.

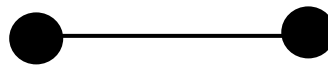
$$V(N_n) = \{u_1, u_2, \dots, u_n\}$$

$$E(N_n) = \emptyset$$



2. Undirected Graph

In this graph the edges go both ways. If one can travel from vertex 1 to 2 one also travel vertex 2 to 1.

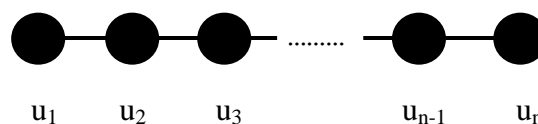


The edges are undirected (symmetric). For example, if the vertices represent people at a party, and there is an edge between two people if they shake hands, then this is an undirected graph, because if person A shook hands with person B, then person B also shook hands with person A.

$$n \geq 2$$

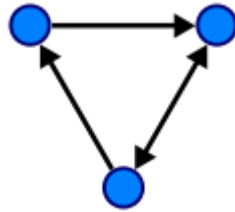
$$V(N_n) = \{u_1, u_2, \dots, u_n\}$$

$$E(N_n) = \{\{u_1, u_2\}, \{u_2, u_3\}, \dots, \{u_{n-1}, u_n\}\}$$



3. Directed Graph or digraph

A graph is directed in which case the edges have a direction. The edges are called arcs.

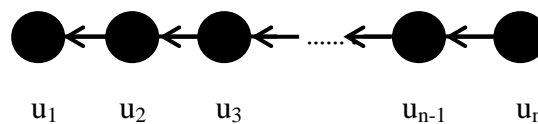


The edges are directed (asymmetric) For example, if the vertices represent people at a party, and there is an edge from person A to person B when person A knows of person B, then this graph is directed, because knowing of someone is not necessarily a symmetric relation (that is, one person knowing of another person does not necessarily imply the reverse; for example, many fans may know of a celebrity, but the celebrity is unlikely to know of all their fans).

$$n \geq 2$$

$$V(N_n) = \{u_1, u_2, \dots, u_n\}$$

$$E(N_n) = \{\{u_1, u_2\}, \{u_2, u_3\}, \dots, \{u_{n-1}, u_n\}\}$$



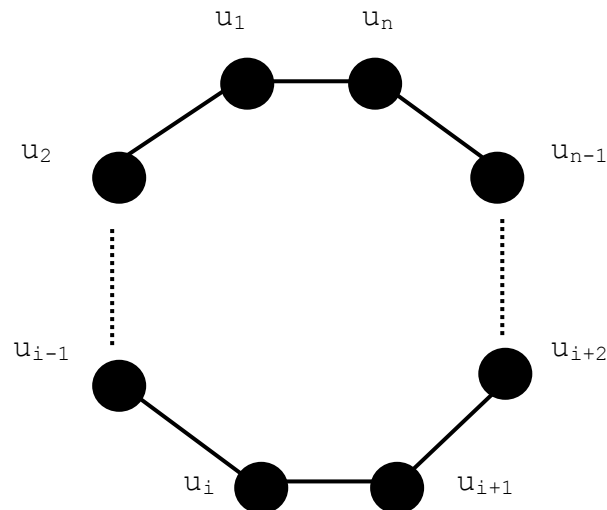
4. Cycle Graph

A cycle graph (circular graph, simple cycle graph, cyclic graph) is a graph that consists of a single cycle, or in other words, some number of vertices connected in a closed chain. The cycle graph with n vertices is called C_n . The number of vertices in a C_n equals the number of edges, and every vertex has degree 2.

$$n \geq 3$$

$$V(N_n) = \{u_1, u_2, \dots, u_n\}$$

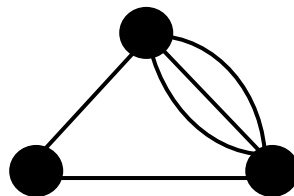
$$E(N_n) = \{\{u_1, u_2\}, \{u_2, u_3\}, \dots, \{u_{n-1}, u_n\}, \{u_n, u_1\}\}$$



A cycle with an even number of vertices is called an **even cycle**; a cycle with an odd number of vertices is called an **odd cycle**.

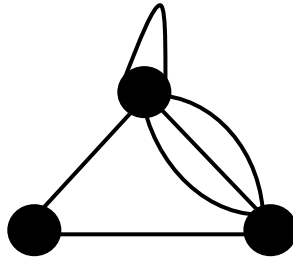
5. Multi Graph

Multigraph is a graph which is permitted to have multiple edges, (also called "parallel edges"), that is, edges that have the same end nodes. Thus two vertices may be connected by more than one edge.



6. Pseudo Graph

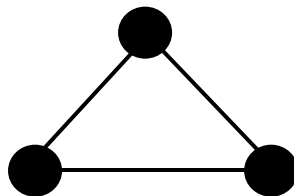
In a pseudograph $G = (V, E)$ two or more edges may connect the same pair of vertices, and in addition, an edge may connect a vertex to itself.



N.B Some authors also allow multigraphs to have loops, that is, an edge that connects a vertex to itself, while others call these **pseudographs**, reserving the term multigraph for the case with no loops.

7. Simple Graph



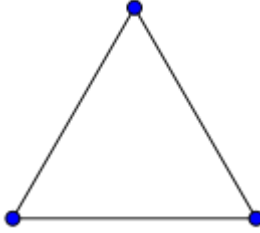
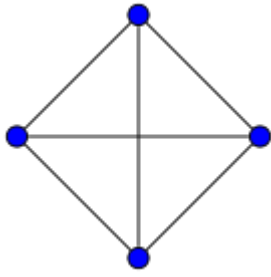
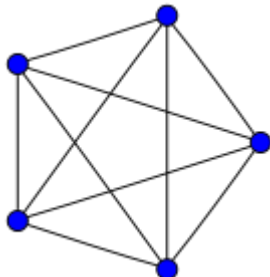
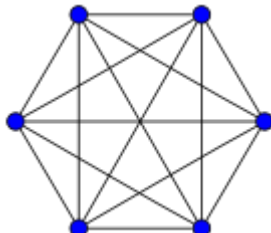
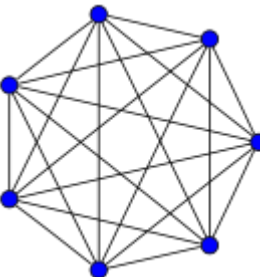
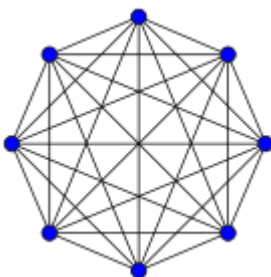
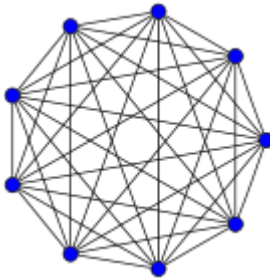
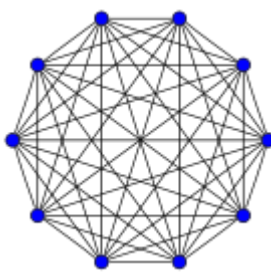
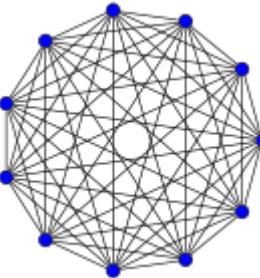
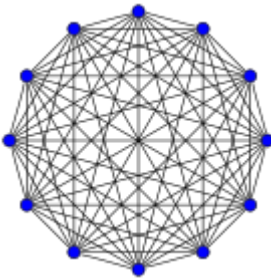
A simple graph is an undirected graph that has no loops and no more than one edge between any two different vertices.



8. Complete Graph

For any $n \in \mathbb{N}$, a complete graph on n vertices, K_n , is a simple graph with n nodes in which every node is adjacent to every other node:

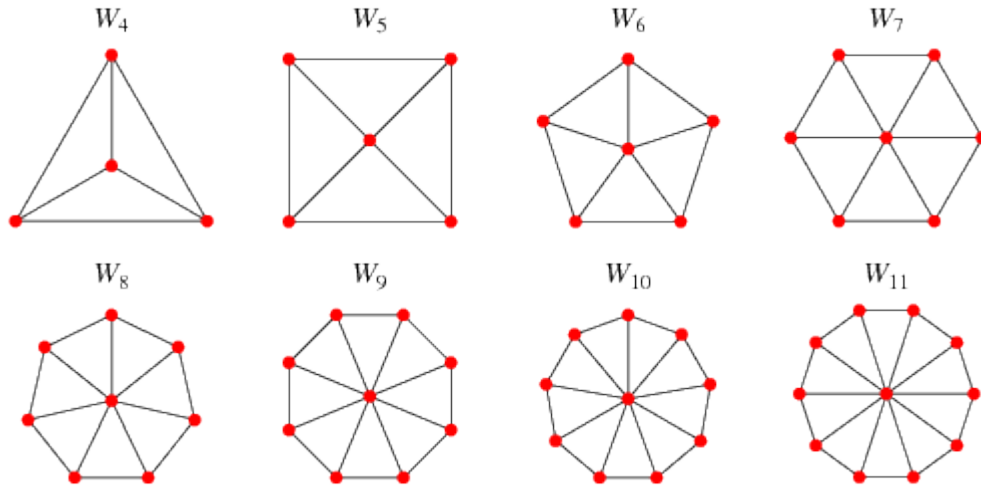
Complete graphs on n vertices, for n between 1 and 12, are shown below along with the numbers of edges:

$K_1:0$	$K_2:1$	$K_3:3$	$K_4:6$
			
$K_5:10$	$K_6:15$	$K_7:21$	$K_8:28$
			
$K_9:36$	$K_{10}:45$	$K_{11}:55$	$K_{12}:66$
			

The complete graph on n vertices has $\frac{n(n-1)}{2}$ edges, and is denoted by K_n

9. Wheel Graph

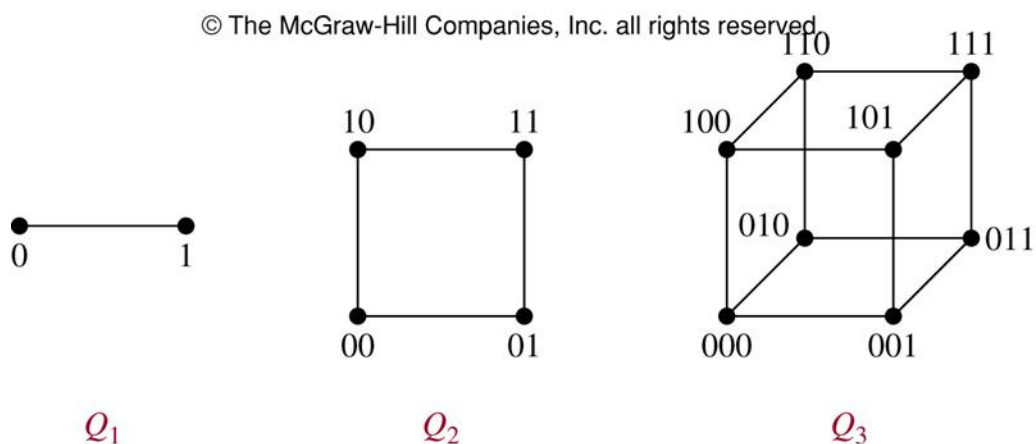
For any $n \geq 3$, a wheel W_n , is a simple graph obtained by taking the cycle C_n and adding one extra vertex v_{hub} and n extra edges $\{\{v_{\text{hub}}, v_1\}, \{v_{\text{hub}}, v_2\}, \dots, \{v_{\text{hub}}, v_n\}\}$.



The Wheel graph on n vertices has $2n$ edges, and is denoted by W_n .

10. Hypercube Graph

For any $n \in \mathbb{N}$, the hypercube Q_n is a simple graph consisting of two copies of Q_{n-1} connected together at corresponding nodes. Q_0 has 1 node.

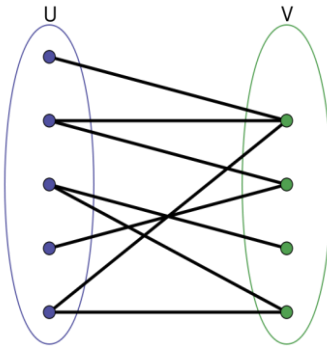


The Hypercube Graph on 2^n vertices has $2^{n-1}n$ edges, and is denoted by Q_n .

11. Bipartite Graph

A bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V .

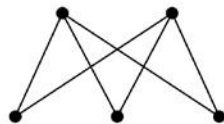
A graph is said to be bipartite if the vertices can be split into two sets V_1 and V_2 such that there are no edges between two vertices of V_1 or two vertices of V_2 .



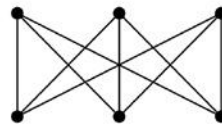
12. Complete Bipartite Graph

A bipartite graph is a **complete bipartite** graph if every vertex in U is connected to every vertex in V .

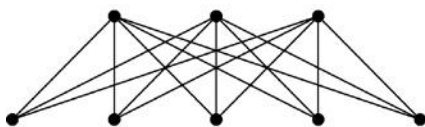
© The McGraw-Hill Companies, Inc. all rights reserved.



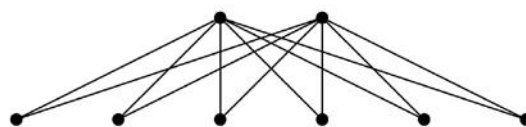
$K_{2,3}$



$K_{3,3}$



$K_{3,5}$

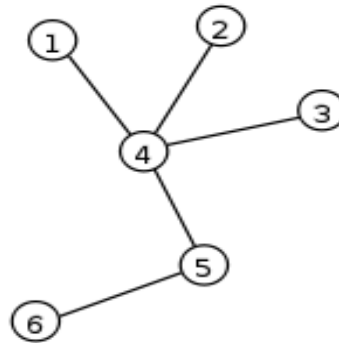


$K_{2,6}$

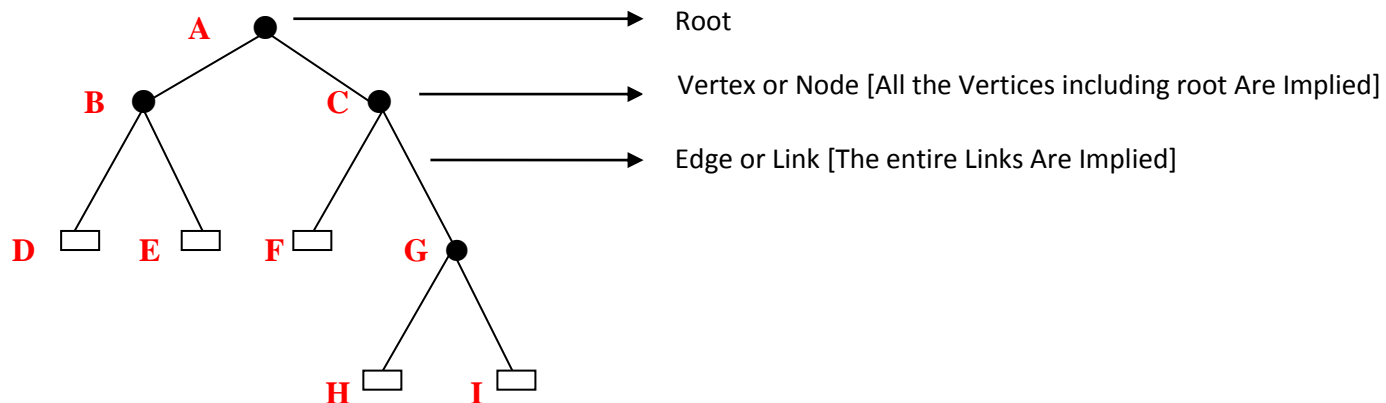
If U has n elements and V has m , then we denote the resulting complete bipartite graph by $K_{n,m}$.

13. Tree

A tree is an undirected graph in which any two vertices are connected by exactly one edge. In other words, any connected graph without cycles is a tree.



The Tree Graph on n vertices has $n - 1$ edge.



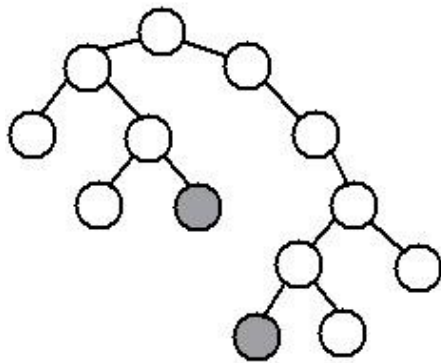
Binary Tree

1. **A** is the root of the tree [Node Without Parent] [Every Tree has exactly one root]
2. **B** is the parent of **D** and **E** [Each Node Except The Root Has Exactly One Node Above It Is Called Its Parent]
3. **D** and **E** is the children of **B** [The Nodes Directly Below The Parent Called Its Children]
4. **C** is the sibling of **B** [Nodes Which Share Same Parent]
5. **D, E, F, H** and **I** are terminals or external nodes [Nodes With No Children]
6. **A, B, C** and **G** are the non terminals or internal nodes
7. Node : stores a data element [Here We Have 9 Nodes; 4 internal and 5 external]
8. Ancestors of a node: itself, its parent, grandparent, grand-grandparent.
Descendant of a node: itself, its child, grandchild, grand-grandchild.

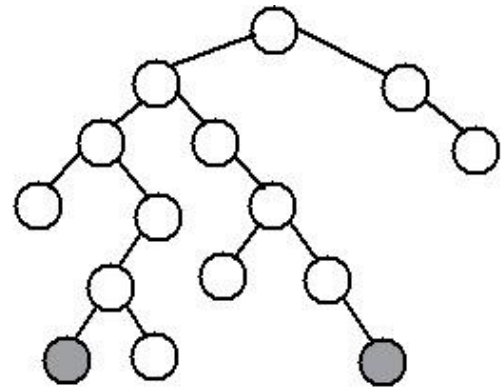
Diameter of Tree

The **diameter** of a tree (sometimes called the width) is the number of nodes on the longest path between two leaves in the tree.

The diagram below shows two trees each with diameter nine, the leaves that form the ends of a longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).



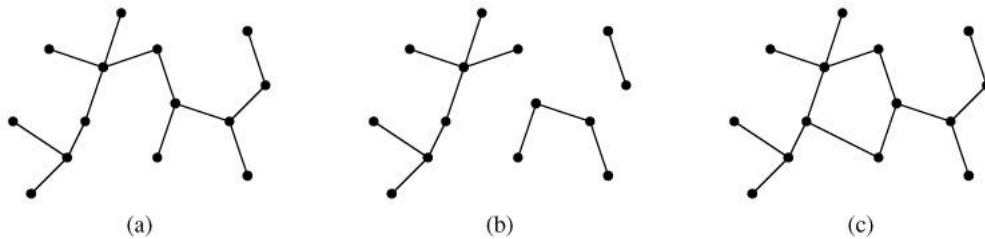
diameter, 9 nodes, through root



diameter, 9 nodes, NOT through root

14. Forest

A forest is a disjoint union of trees. An undirected graph which contains no cycles is called a forest. A directed acyclic graph is often referred to as a dag. The collection of trees is called forest



(a) A tree

(b) A forest

(c) A graph that contains a cycle and is therefore neither a tree nor a forest

15. Subgraph

A **subgraph** of a graph G is a graph whose vertex set is a subset of that of G , and whose adjacency relation is a subset of that of G restricted to this subset. In the other direction, a **supergraph** of a graph G is a graph of which G is a subgraph. We say a graph G **contains** another graph H if some subgraph of G is H or is isomorphic to H .

A subgraph H is a **spanning subgraph**, or **factor**, of a graph G if it has the same vertex set as G . We say H spans G .

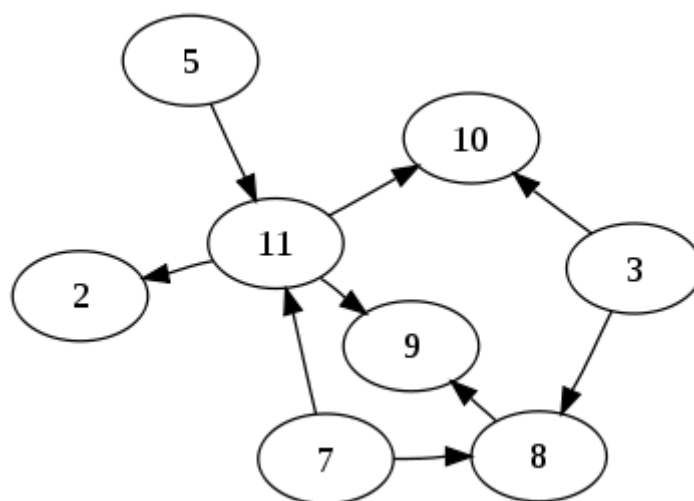
A subgraph H of a graph G is said to be **induced** if, for any pair of vertices x and y of H , xy is an edge of H if and only if xy is an edge of G . In other words, H is an induced subgraph of G if it has exactly the edges that appear in G over the same vertex set. If the vertex set of H is the subset S of $V(G)$, then H can be written as $G[S]$ and is said to be **induced by S** .

A graph that does *not* contain H as an induced subgraph is said to be **H -free**.

A **universal graph** in a class K of graphs is a simple graph in which every element in K can be embedded as a subgraph.

16. DAG

An acyclic digraph (occasionally called a dag or DAG for "directed acyclic graph", although it is not the same as an orientation of an acyclic graph) is a directed graph with no directed cycles.



DAG is a directed graph with no directed cycles. That is, it is formed by a collection of vertices and directed edges, each edge connecting one vertex to another, such that there is no way to start at some vertex v and follow a sequence of edges that eventually loops back to v again.

Topological Sorting:

In computer science, if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG) a **topological sort** (sometimes abbreviated **topsort** or **toposort**) or **topological ordering** of a directed graph is a linear ordering of its vertices such that, for every edge uv , u comes before v in the ordering. For instance, the vertices of the graph may represent tasks to be performed, and the edges may represent constraints that one task must be performed before another; in this application, a topological ordering is just a valid sequence for the tasks.

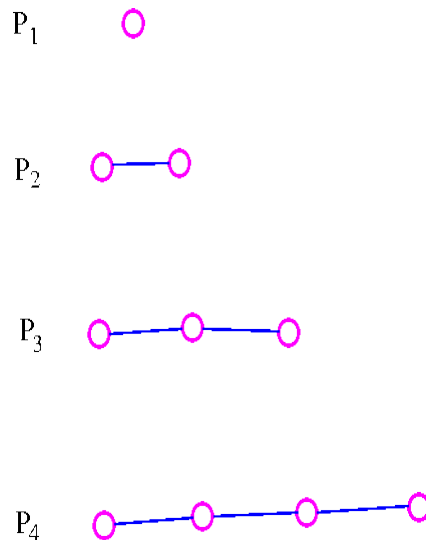
Any DAG has at least one topological ordering, and algorithms are known for constructing a topological ordering of any DAG in linear time

Topological orderings are also closely related to the concept of a linear extension of a partial order in mathematics. A partially ordered set is just a set of objects together with a definition of the " \leq " inequality relation, satisfying the axioms of reflexivity ($x = x$), antisymmetry (if $x \leq y$ and $y \leq x$ then $x = y$) and transitivity (if $x \leq y$ and $y \leq z$, then $x \leq z$).

One can define a partial ordering from any DAG by letting the set of objects be the vertices of the DAG, and defining $x \leq y$ to be true, for any two vertices x and y , whenever there exists a directed path from x to y ; that is, whenever y is reachable from x . With these definitions, a topological ordering of the DAG is the same thing as a linear extension of this partial order.

17. Path Graph

A path graph is a graph consisting of a single path. The path graph with n vertices is denoted by P_n .

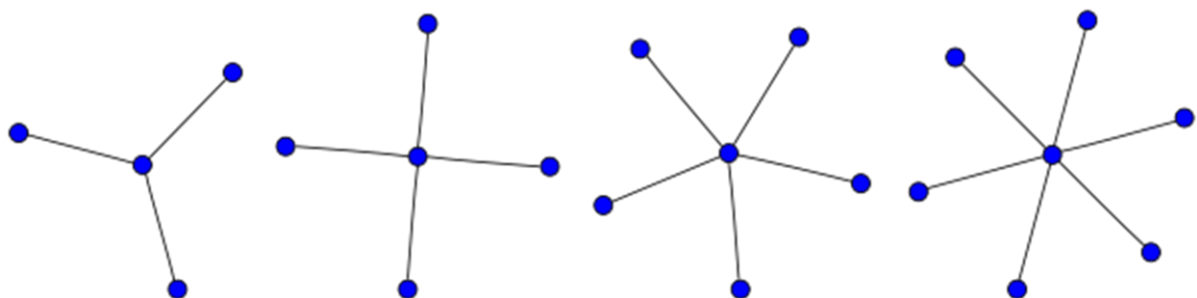


Path graph, P_n , has $n-1$ edges, and can be obtained from cycle graph, C_n , by removing any edge

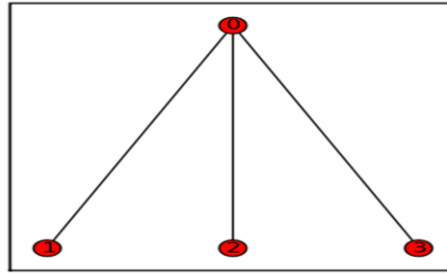
18. Star Graph

In graph theory, a **star** S_k is the complete bipartite graph $K_{1,k}$: a tree with one internal node and k leaves (but, no internal nodes and $k + 1$ leaves when $k \leq 1$). Alternatively, some authors define S_k to be the tree of order k with maximum diameter 2; in which case a star of $k > 2$ has $k - 1$ leaves.

A star with 3 edges is called a **claw**.



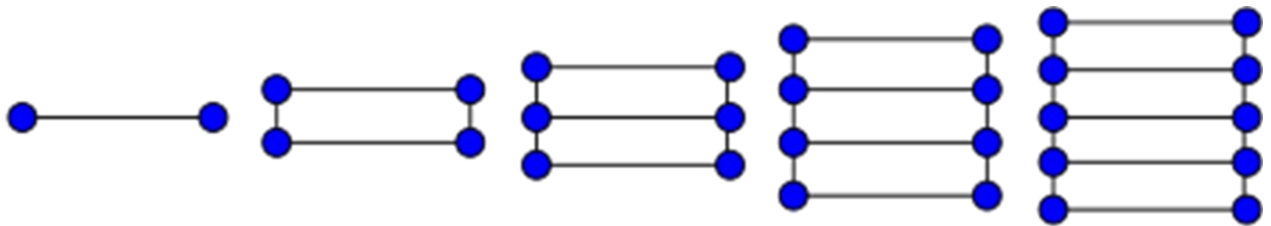
The Star graphs S_3 , S_4 , S_5 and S_6 .



Claw Graph

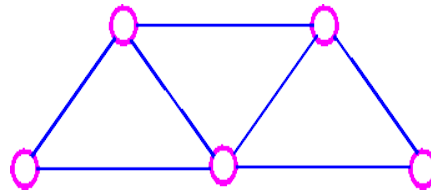
18. Ladder graph

In the mathematical field of graph theory, the **ladder graph** L_n is a planar undirected graph with $2n$ vertices and $n+2(n-1)$ edges.

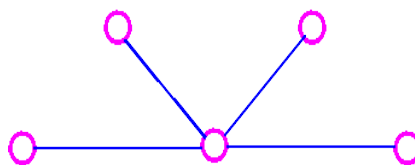
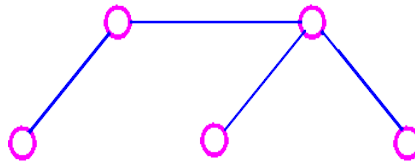
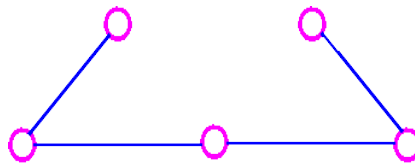
The ladder graphs L_1 , L_2 , L_3 , L_4 and L_5 .

19. Spanning Tree

Spanning tree T of a connected, undirected graph G is a tree and a subgraph composed of all the vertices and some (or perhaps all) of the edges of G .



The following are the three of its spanning trees:



Minimum Spanning Tree:

A **minimum spanning tree (MST)** or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree.

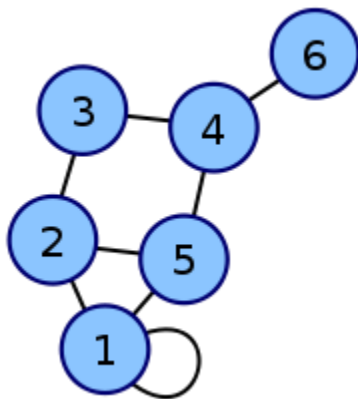
Maximum Spanning Tree:

A **maximum spanning tree (MST)** or **maximum weight spanning tree** is then a spanning tree with weight greater than or equal to the weight of every other spanning tree.

Graph Representations

1. Adjacency list - An adjacency list is implemented as an array of lists.

In the Graph Representation we use a matrix to keep track of all the edges incident to a given vertex. This can be done by using an array of length N , where N is the number of vertices. The i -th entry in this array is a list of the edges incident to i -th vertex (edges are represented by the index of the other vertex incident to that edge). This representation is much more difficult to code, especially if the number of edges incident to each vertex is not bounded, so the lists must be linked lists (or dynamically allocated). Debugging this is difficult, as following linked lists is more difficult. However, this representation uses about as much memory as the edge list. Finding the vertices adjacent to each node is very cheap in this structure, but checking if two vertices are adjacent requires checking all the edges adjacent to one of the vertices. Adding an edge is easy, but deleting an edge is difficult, if the locations of the edge in the appropriate lists are not known.

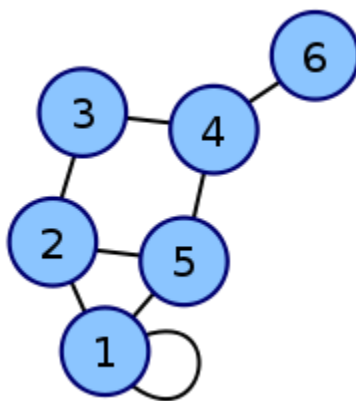


Vertex	Adjacent Vertices
1	1, 2, 5
2	1, 5
3	2, 4
4	3, 5
5	1, 2, 4
6	4

2. Adjacency matrix – This is an N by N array (N is the number of vertices). The i, j entry contains a 1 if the edge (i, j) is in the graph; otherwise it contains a 0. For an undirected graph, this matrix is symmetric.

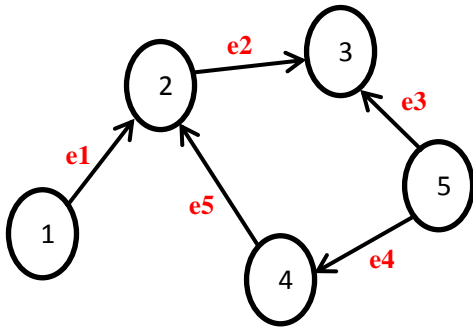
This representation is easy to code. It's much less space efficient, especially for large, sparse graphs. Debugging is harder, as the matrix is large. Finding all the edges incident to a given vertex is fairly expensive (linear in the number of vertices), but checking if two vertices are adjacent is very quick. Adding and removing edges are also very inexpensive operations.

For weighted graphs, the value of the (i, j) entry is used to store the weight of the edge.



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

3. Edge List – The most obvious way to keep track of the edges is to keep a list of the pairs of vertices representing the edges in the graph. This representation is easy to code, fairly easy to debug, and fairly space efficient. However, determining the edges incident to a given vertex is expensive, as is determining if two vertices are adjacent. Adding an edge is quick, but deleting one is difficult if its location in the list is not known. For weighted graphs, this representation also keeps one more number for each edge, the edge weight.



e	V1	V2
e1	1	2
e2	2	3
e3	3	5
e4	5	4
e5	4	2

Data Structure Tradeoffs

	Adjacency list	Incidence list	Adjacency matrix	Incidence matrix
Storage	$O(V + E)$	$O(V + E)$	$O(V ^2)$	$O(V \cdot E)$
Add vertex	$O(1)$	$O(1)$	$O(V ^2)$	$O(V \cdot E)$
Add edge	$O(1)$	$O(1)$	$O(1)$	$O(V \cdot E)$
Remove vertex	$O(E)$	$O(E)$	$O(V ^2)$	$O(V \cdot E)$
Remove edge	$O(E)$	$O(E)$	$O(1)$	$O(V \cdot E)$
Query: are vertices u, v adjacent? (Assuming that the storage positions for u, v are known)	$O(E)$	$O(E)$	$O(1)$	$O(E)$
Remarks	When removing edges or vertices, need to find all vertices or edges		Slow to add or remove vertices, because matrix must be resized/copied	Slow to add or remove vertices and edges, because matrix must be resized/copied

Speciale Thanks

- Dr. Mumit Khan (Professor and Chairperson, CSE, BRACU)
- Annajiat Alim Rasel (Lecturer-II, BRACU)

- Raduanul Islam
- Sadaf Noor

References

1. CSE 221, Algorithm by Dr. Mumit Khan, BRAC University
2. CSE 230, Discrete Mathematics Afroza Sultana, BRAC University
3. Graph Theory by Geir Agnarsson, Raymond Greenlaw
4. Art of Programming Contest by Ahmed Shamsul Arefin
5. Competitive Programming 2 by Steven Halim and Felix Halim
6. <http://en.wikipedia.org>
7. <http://tech-queries.blogspot.com/2010/09/diameter-of-tree-in-on.html>
8. <http://www.cs.duke.edu/courses/spring00/cps100/assign/trees/diameter.html>
9. <http://www.personal.kent.edu/~rmuhamma/GraphTheory/MyGraphTheory/defEx.htm>
10. http://wiki.sagemath.org/graph_generators
11. <http://dictionary.reference.com/browse/strongly+connected+component>
12. <http://xlinux.nist.gov/dads/HTML/manhattanDistance.html>