**Graph Theory and Algorithms**

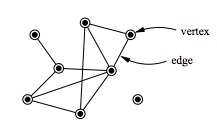
**Dynamic Programming**

*Lecture 14:*

**Graph Theory**

Standard Terminologies:

Graph: **G = (V, E)** where **V**={v1,…,vi,…vn} is the set of **vertices**/**nodes** and **E**={e1,…,ek,…,em} is the set of **edges**.



d(vi): Degree of a vertex – the **number of edges** adjacent to vertex vi.

Walk: a sequence of alternating vertices and edges of a graph (vertex can be repeated, edge can be repeated).

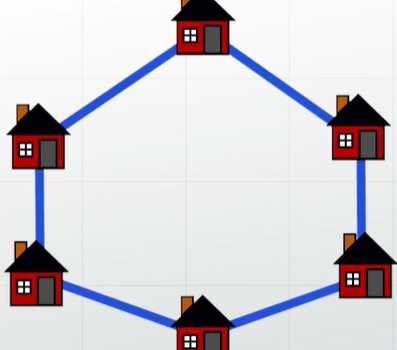
Trail: if the **edges** of a walk are **distinct**, then the walk is a trail.

Path: if, in **addition**, the **vertices** of the trail are **distinct**, then it is called a path.

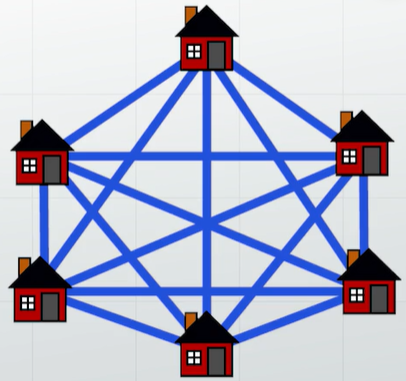
Circuit: a **closed** **trail**. A closed trail is when the starting vertex and the end vertex are the same.

Cycle: a **closed** **path**. A closed path is when the starting vertex and the end vertex are the same.

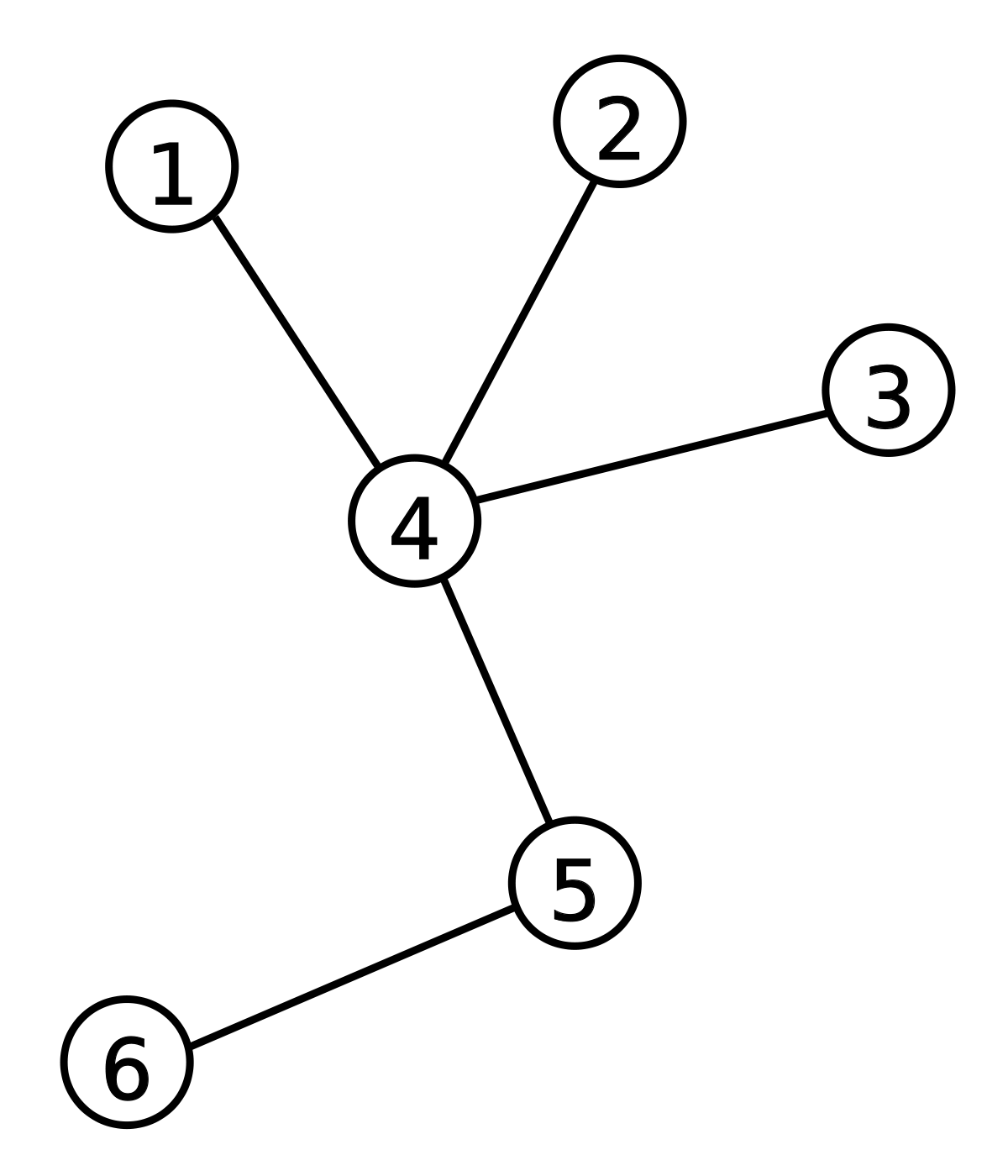
Connected graph: a **graph** in which there is a **route** **from each vertex** **to any other vertex.**



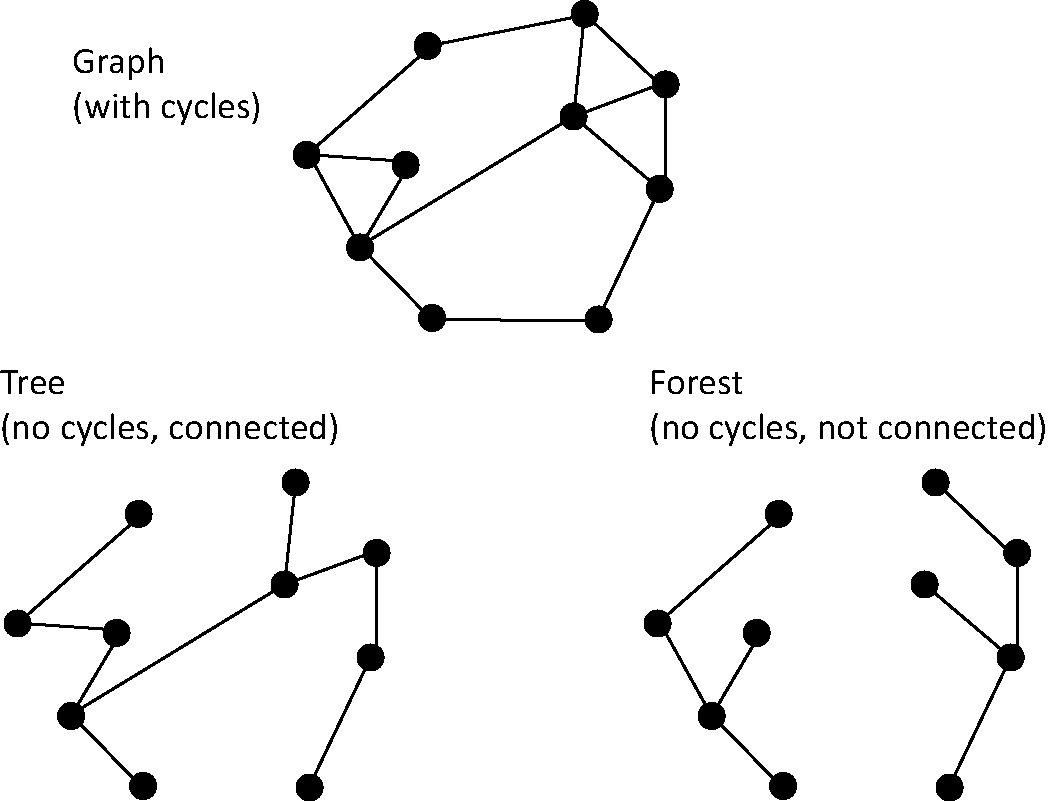
Complete graph: there is an edge between every single pair of vertices.



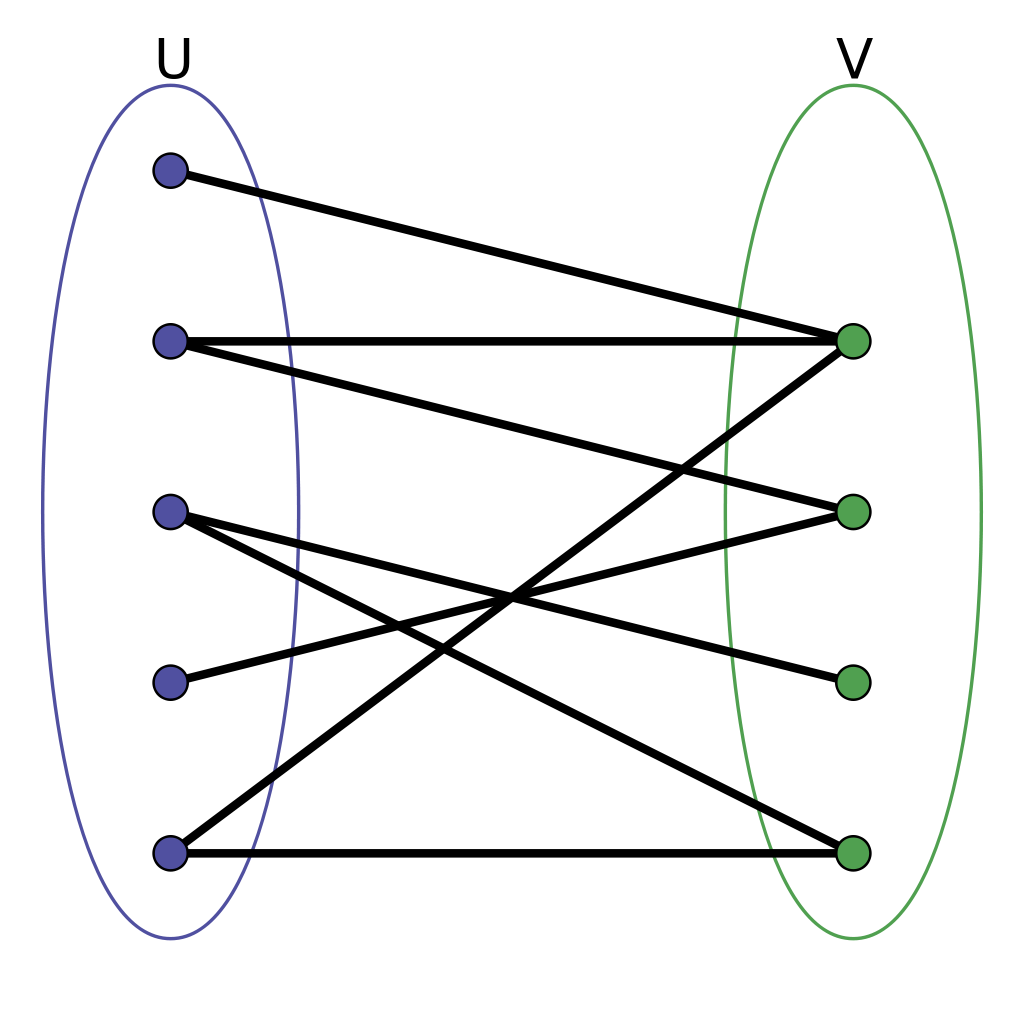
Tree: a connected graph with no cycles. Its degree-1 vertices are called **leaves.**



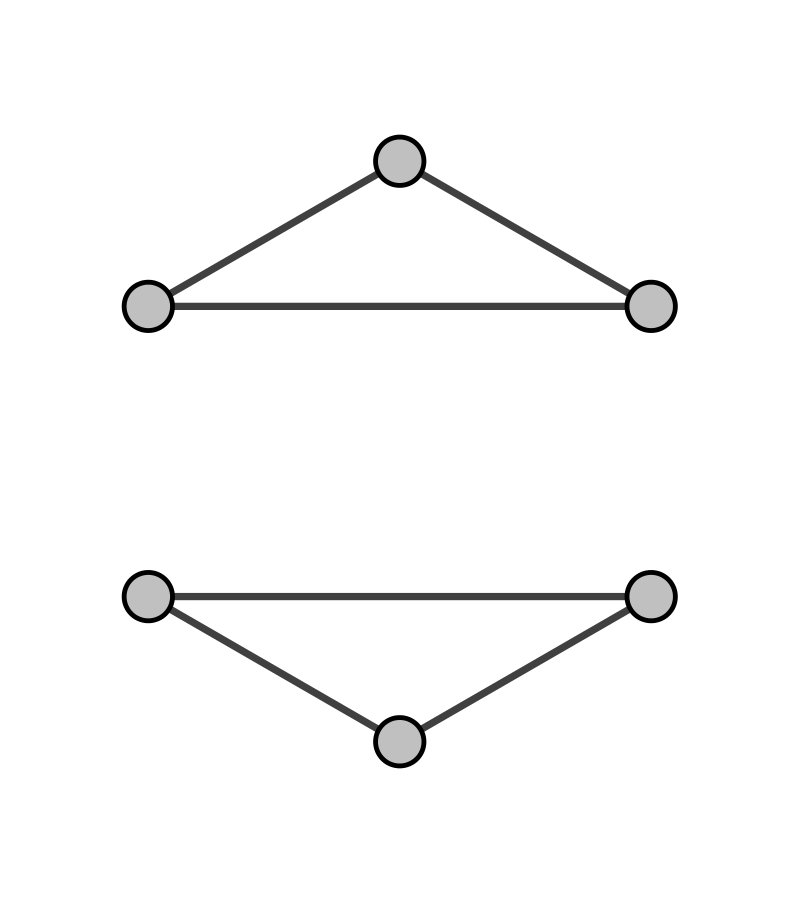
Forest: every connected component is a tree. Alternatively speaking, a graph with no cycles but not necessarily connected.



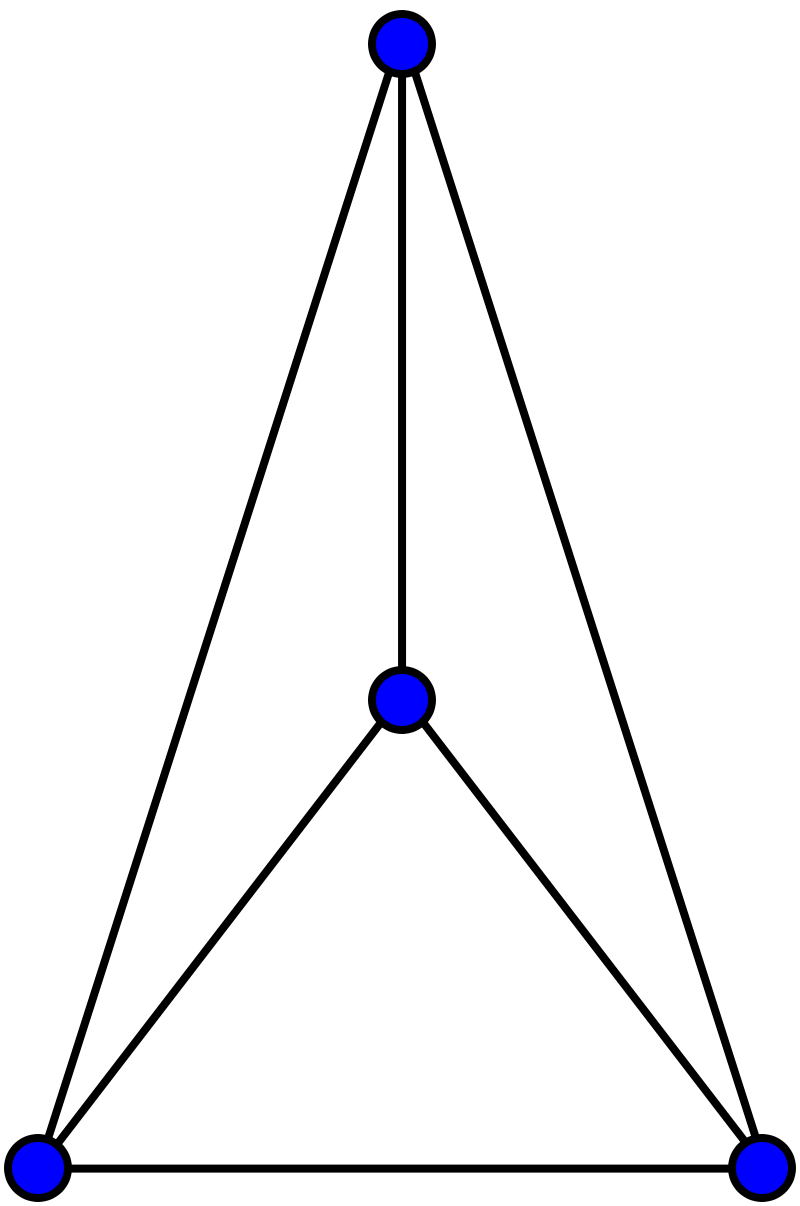
Bipartite graph: the vertices can be partitioned into two classes such that every edge has its ends in a different class.



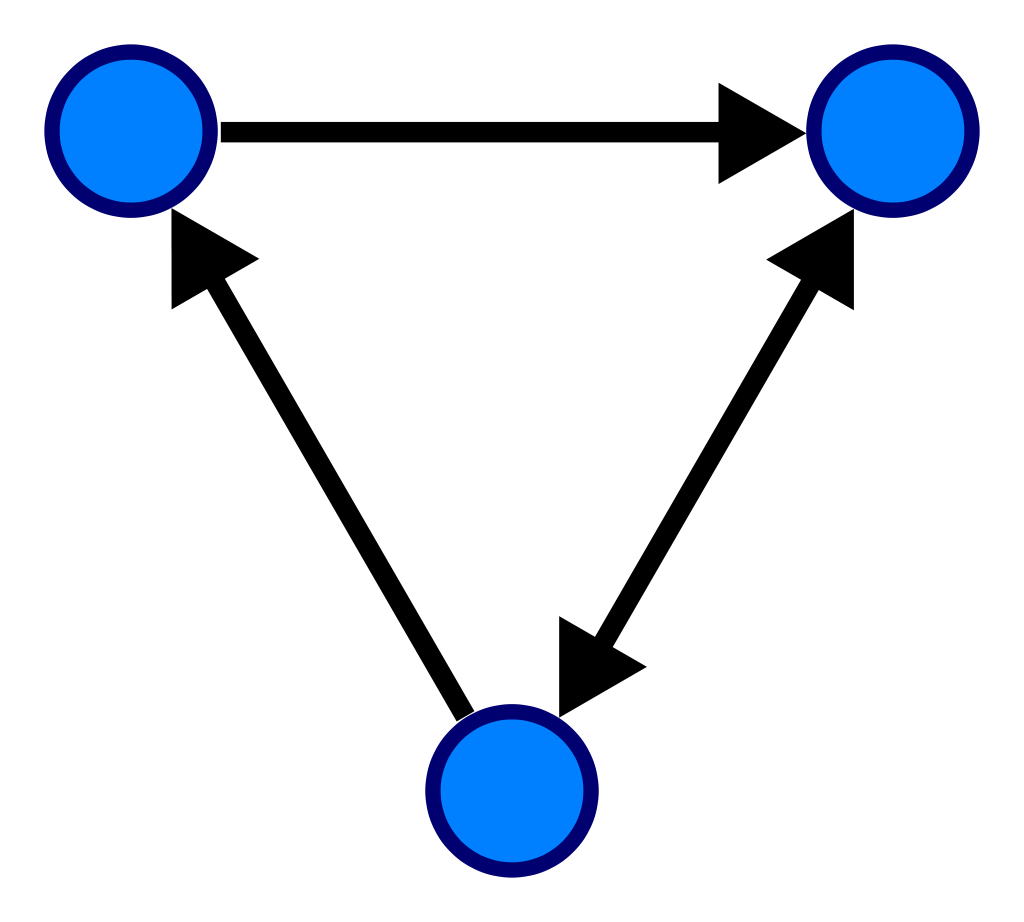
k-regular graph: every vertex of the graph has the same degree k. (e.g. 2-regular graph, 2 neighbours)



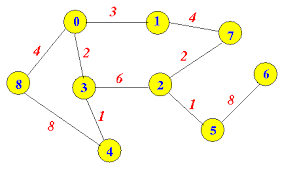
Planar graph: a graph that can be drawn with no edges crossing.



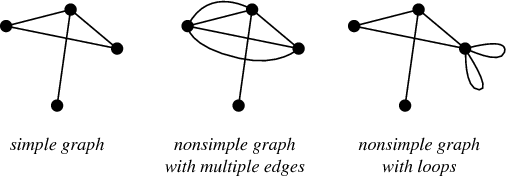
Directed graph: a graph in which the edges have directions.



Weighted graph: a graph in which the edges have weights.



Simple graph: finite, unweighted, undirected, no loops, no multiple edges (multiple edges: 2 or more).



**Vertex degree:**

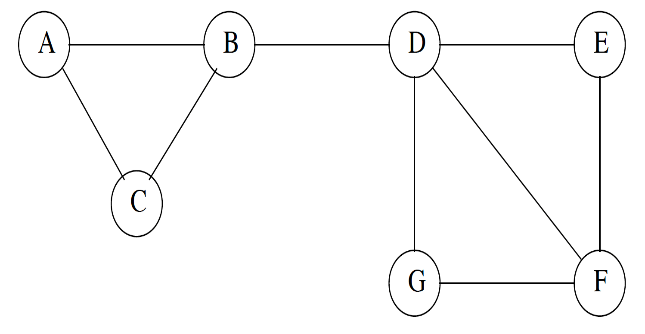
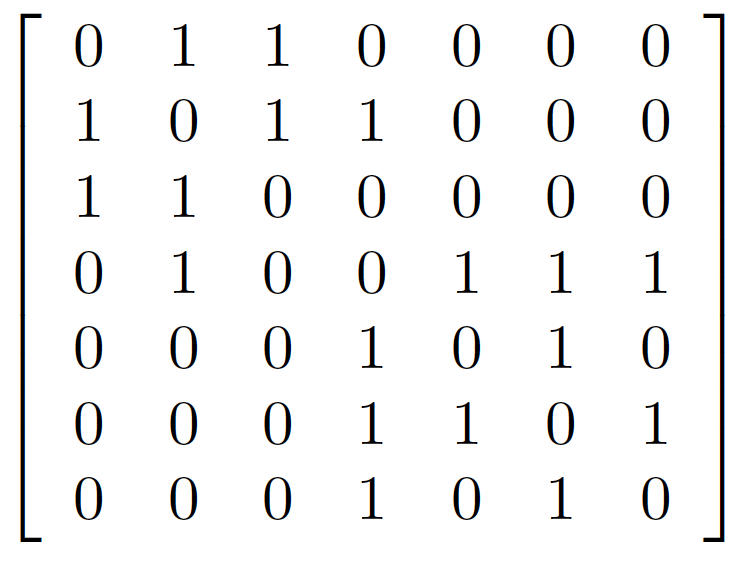
Handshaking Lemma: The number of degrees of the vertices is always twice the number of edges!

A Little more about trees: (easy)

The following statements are equivalent:

* T is a tree (def: no cycles, connected).
* Any two vertices of T are linked by a unique path.
* T is minimally connected, i.e., T is connected but T – e is disconnected for every edge e ∈ T. (removing an edge will leave us with a disconnected graph)
* T is maximally acyclic, i.e., T contains no cycle but T + v1v2 does, for any two non-adjacent vertices v1, v2 ∈ T. (adding an edge will create a cycle).
* T has n-1 edges.

Adjacency matrix

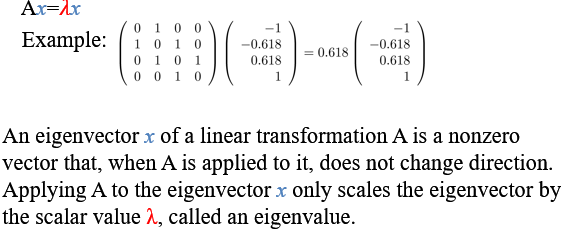
{A = (aij) where aij = 1 if there is an edge between vertices vi and vj; otherwise aij = 0}

It is a symmetric and square matrix.

The sum of the row/col is the degree of the vertex.

Eigenvalues & eigenvectors

**Graph spectrum: The set of graph eigenvalues of the adjacency matrix is called the spectrum of the graph. (**The eigenvalues and the eigenvectors of the adjacency matrix.)



Graph representation: Adjacency list

An adjacent list representation for a graph associates each vertex in the graph with the collection of its neighbouring vertices or edges.

A close up of a white background

Description automatically generatedGraph traversal:

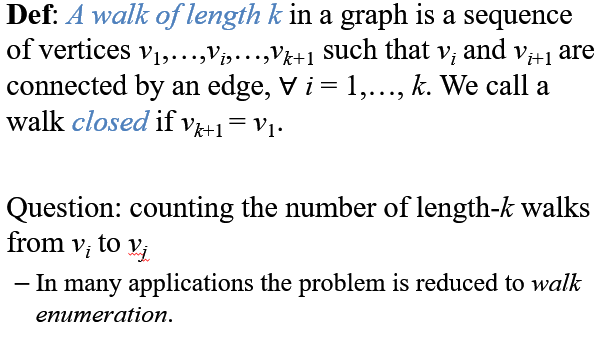
This is the process of visiting each vertex in a graph.

There are two ways: **Breadth first search (BFS)** or **Depth first search (DFS)**

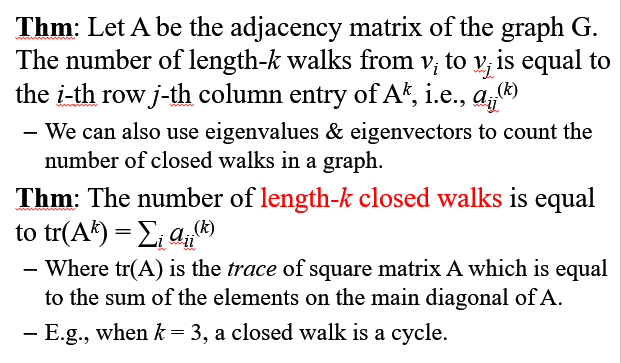
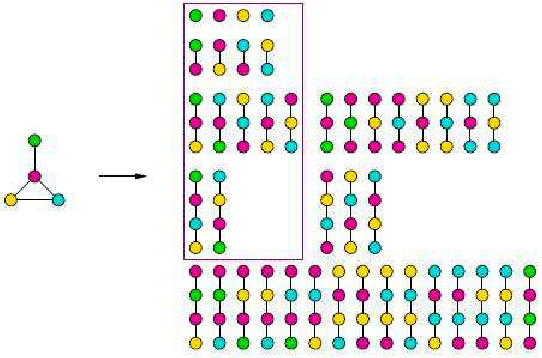
* BFS: start at a vertex, explores all of its neighbour nodes at the present depth prior to moving on to the nodes at next depth level. (e.g. 1, 2, 5, 3, 4) (<https://en.wikipedia.org/wiki/Breadth-first_search#/media/File:Animated_BFS.gif>)
* DFS: traverses the depth of any particular path before exploring its depth (e.g. 1, 2, 5, 4, 3) (<https://en.wikipedia.org/wiki/Depth-first_search#/media/File:Depth-first-tree.svg>)

<https://www.youtube.com/watch?v=iaBEKo5sM7w>

Walk enumeration:



So what that means is that we traverse through vertices, which are connected by edges, and the walk is closed if we finish at the same node that we stared on.

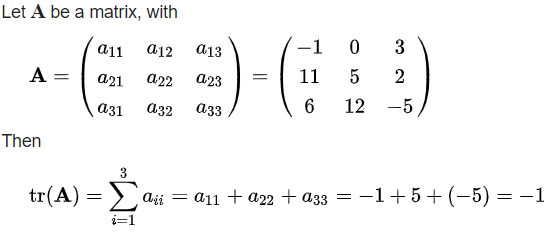


First theorem is easy, just read it.

Second theorem:

(Trace of matrix:

)



So if you want to find how many 2-length paths there are from vertex 2 to vertex 1 then you should:

Get the matrix: (A\*A)

Then go to 1st row and 2nd column (or 2nd row and 1st column, should be the same number)

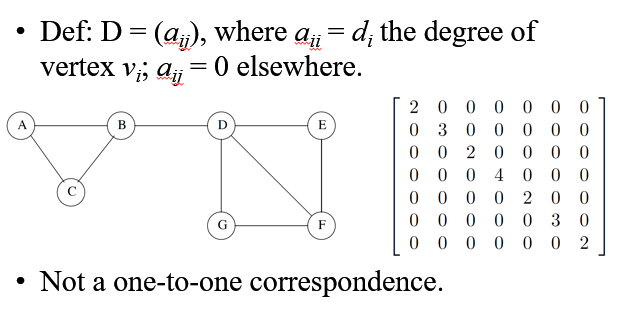
That number is the number of how many 2 length paths there are.

(<https://math.stackexchange.com/questions/1890620/finding-path-lengths-by-the-power-of-adjacency-matrix-of-an-undirected-graph>)

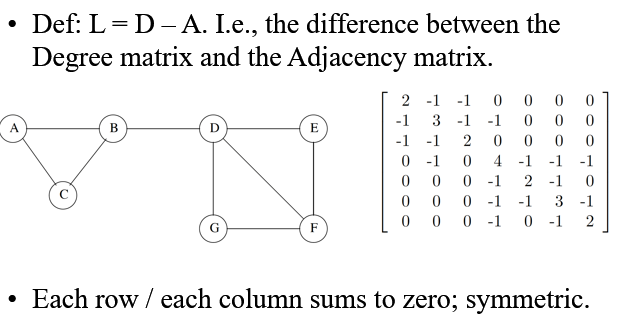
A close up of text on a whiteboard

Description automatically generated

Degree matrix:

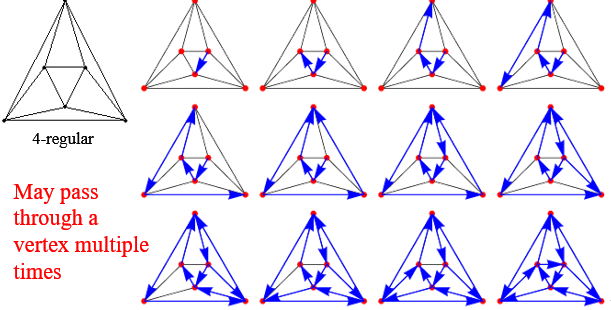


Laplacian matrix:



Eulerian graphs:

An **Euler** **trail** travels **every** **edge** exactly **once**; if it ends at the **initial** **vertex**, then it is an **Euler** **circuit**.



*How to determine if a graph is Eulerian?*

*We observe that in any Eulerian graph, all, or almost all of the vertices, have an even degree.*

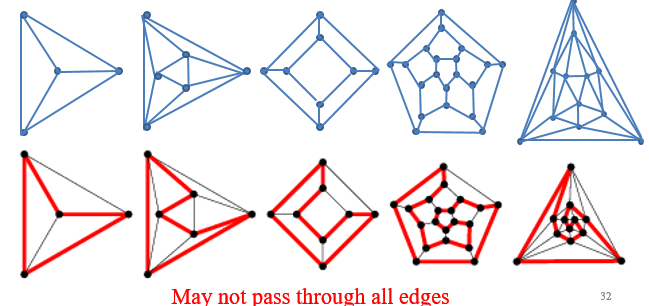
*If a graph has no odd-degree vertices, then the graph admits an Euler circuit.*

*If a graph has only two odd-degree vertices, then the graph admits an Euler trail.*

Hamiltonian graphs:

A **Hamiltonian** **path** is a path that visits **every vertex exactly once**; if it ends at the **initial** **vertex** then it is a **Hamiltonian** **cycle**.

(Need to visit all vertices, I,e, you can ignore some edges if you want).



Hamiltonian graphs:

**Hamiltonian graph**: a graph that admits a **Hamiltonian cycle.**

There is no simple way to determine if a graph is Hamiltonian.

*Lecture 15:*

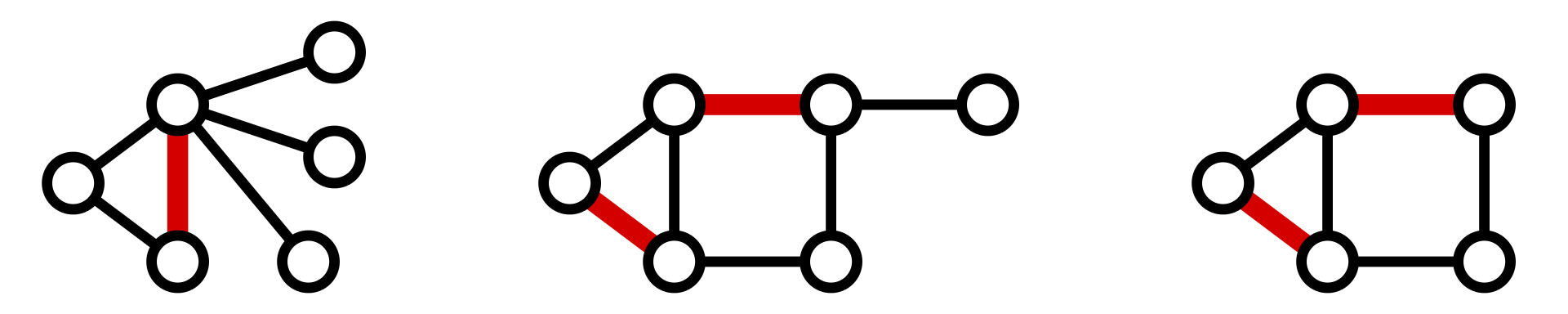
**Matching**

A **matching** in a graph is a set of edges without common vertices. I.e., it is an **independent** **edge** **set**.

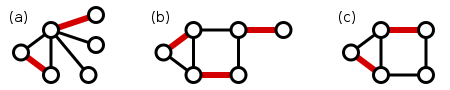


(<https://www.youtube.com/watch?v=bOJC93XxoFc>) maximal vs maximum matching.

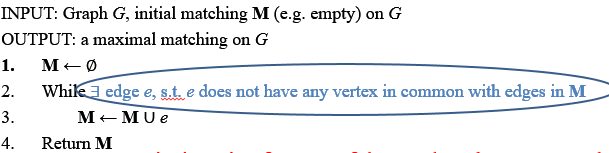
**Maximal matching –** the least amount of matches, so that you cannot add any more edges. Maximal matchings are easier to compute.



**Maximum matching –** the biggest amount of matches, so that you cannot add any more edges.

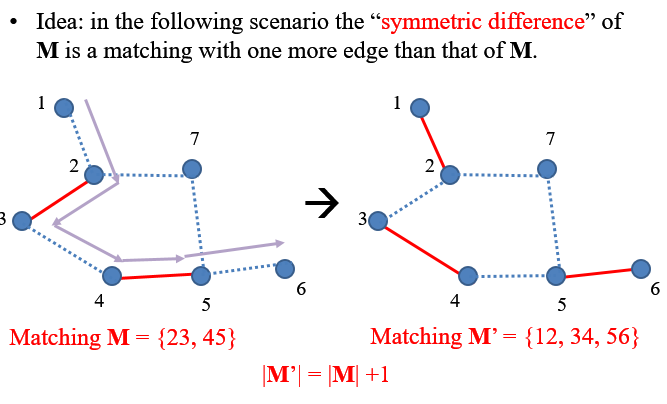


Maximal Matching Algorithm



Size of maximal matching >= half of the size of a maximum matching.

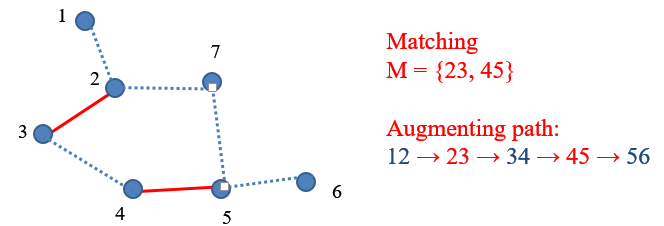
Another way of matching – symmetric difference



Augmenting path

Given a graph *G = (V, E)* and a matching *M*. A path *P* is called an **augmenting path** for *M* if:

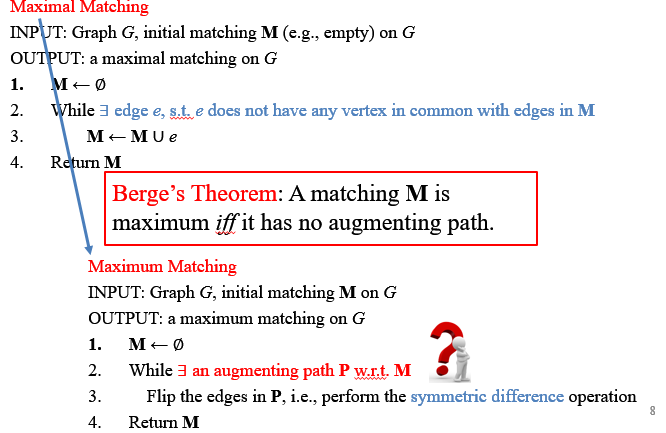
* The two end points of *P* are unmatched by *M*.
* The edges of *P* alternate between edges *elements of* *M* and edges *not elements of* *M*.



From Maximal to Maximum

A matching M is maximum *iff* it has no augmenting path.

To get maximum matching, we just perform the symmetric difference operation from maximal.

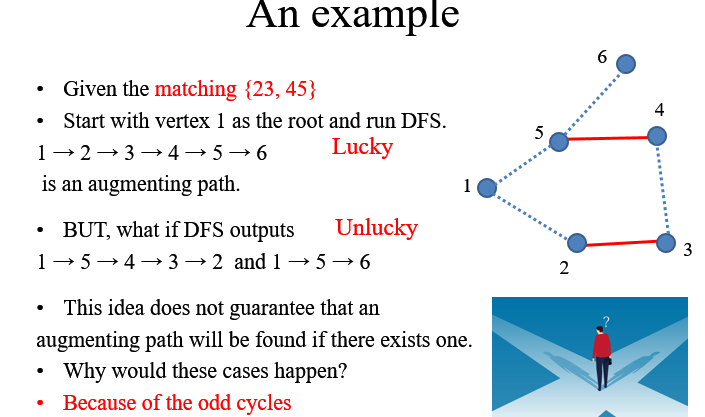


Getting an augmented path

Given a matching M, in searching for an augmented path, we start with an unmatched vertex v0 as the root and run a DFS.

A path *v*0*v*1*v*2…*v*2*kv*2*k+*1is an augmenting path if

* *v*0*v*1, *v*2*v*3, … , *v*2*kv*2*k+*1 ∉ **M**, and
* *v*1*v*2, *v*3*v*4, … , *v*2*k-*1*v*2*k* ∈ **M**.



So TLDR DFS doesn’t work all the time if the graph is an odd cycle (the cycle has an odd number of edges). Based on luck I guess.

Bipartite graphs