DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

MASTER THEOREM – GENERAL FORM

GRAPH ALGORITHMS

Course Instructor: Dr. Shreya Ghosh

MASTER THEOREM (MORE GENERALIZED VERSION)

Master Theorem

The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$.
   Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

MASTER THEOREM (MORE GENERALIZED VERSION)

Master Theorem

The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ w $f(n))$.
   Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

This case applies when the work to split the problem and combine its solutions (represented by $f(n)$) is asymptotically less than the work to solve the subproblems (represented by $n^{\log_b a}$). The cost of the algorithm is dominated by the cost of the recursive calls.

MASTER THEOREM (MORE GENERALIZED VERSION)

Master Theorem

The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b$ $= \Theta(f(n))$.
   Regularity condit n.

This case applies when the work to split the problem and combine its solutions is asymptotically the same as the work to solve the subproblems. The overall cost includes a logarithmic factor due to the depth of the recursion.

MASTER THEOREM (MORE GENERALIZED VERSION)

Master Theorem

The Master Theorem applies to recurrences of the following form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n$

2. If $f(n) = \Theta(n$

This case applies when the work to split the problem and combine its solutions is asymptotically greater than the work to solve the subproblems. The cost of the algorithm is dominated by the work done outside the recursive calls.

3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$.
   Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

The regularity condition ($af\left(\frac{n}{b}\right) \leq kf(n)$ for some $k < 1$ and sufficiently large n) in Case 3 ensures that the work done at each level of the recursion tree decreases by a constant factor. This condition is necessary to guarantee that the work done outside of the recursive calls dominates the total work.

**guarantees that as you move from level to level in the recursion tree, the total cost of the subproblems at each level decreases by a constant factor. This decrease is what allows us to conclude that the cost f(n) is the dominant term in the total cost of the algorithm.*

REGULARITY CONDITION – WHY?

WHEN THE REGULARITY CONDITION IS NOT SATISFIED:

1. **Potential Misestimation**: Without the regularity condition being met, the total cost of the recursive calls might not decrease fast enough compared to the cost of the operations outside the recursive calls. **This means the actual total cost could be higher than what Case 3 of the Master Theorem would suggest.**
2. **Different Asymptotic Behavior**: The actual time complexity of the algorithm could follow a different pattern than the one predicted by applying Case 3 of the Master Theorem. It may be that another case applies, or the recurrence does not fit neatly into the Master Theorem's framework at all, requiring a different method of analysis.
3. **Alternative Methods Required**: You may need to use alternative methods for solving the recurrence relation, such as the substitution method, the recursion tree method, or generating functions. These methods can be more labor-intensive but are necessary when the Master Theorem cannot be applied directly due to the failure of specific conditions like the regularity condition.

MASTER THEOREM

- When analyzing algorithms, recall that we only care about the **asymptotic behavior**.
- Recursive algorithms are no different. Rather than solve exactly the recurrence relation associated with the cost of an algorithm, it is enough to give an asymptotic characterization.
- The main tool for doing this is the master theorem

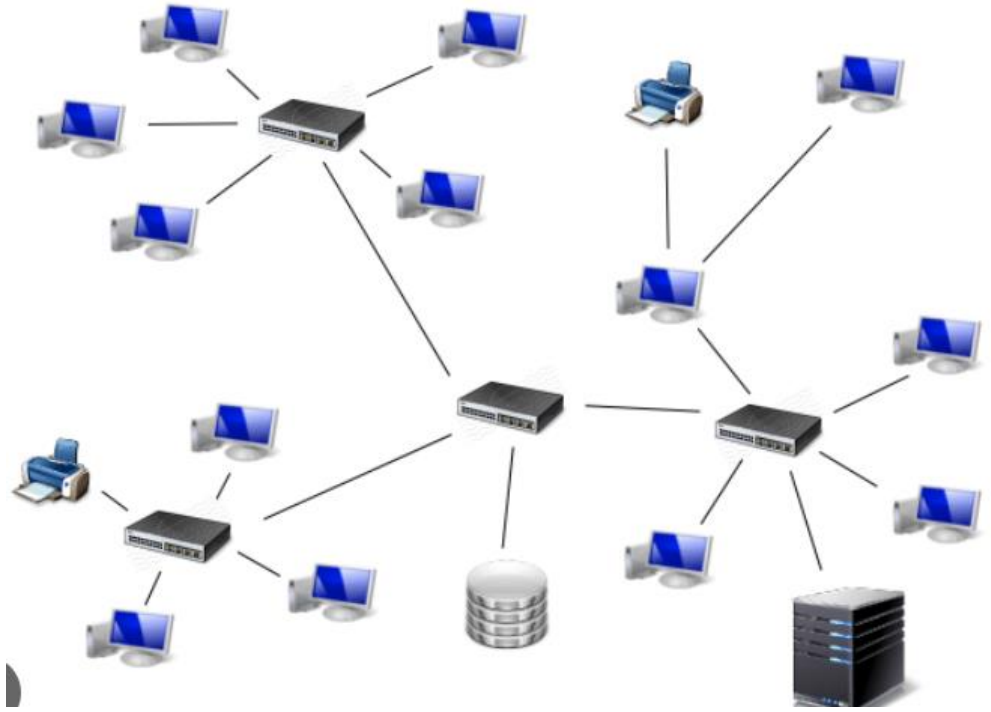
"Asymptotic" refers to the behavior of a function as its argument or input heads towards a particular value or infinity.

GRAPH ALGORITHMS

FUNDAMENTALS AND PATHFINDING ALGORITHMS

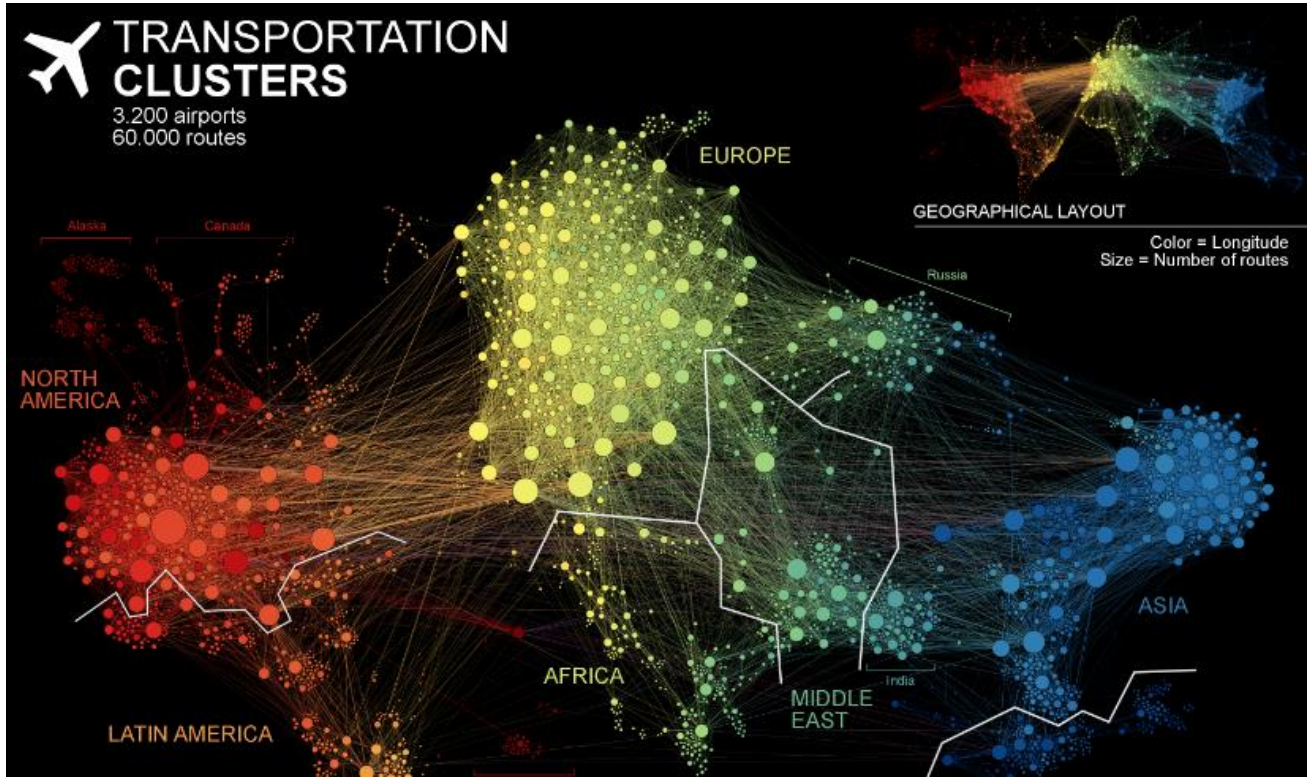
VARYING APPLICATIONS (EXAMPLES)

- Computer networks

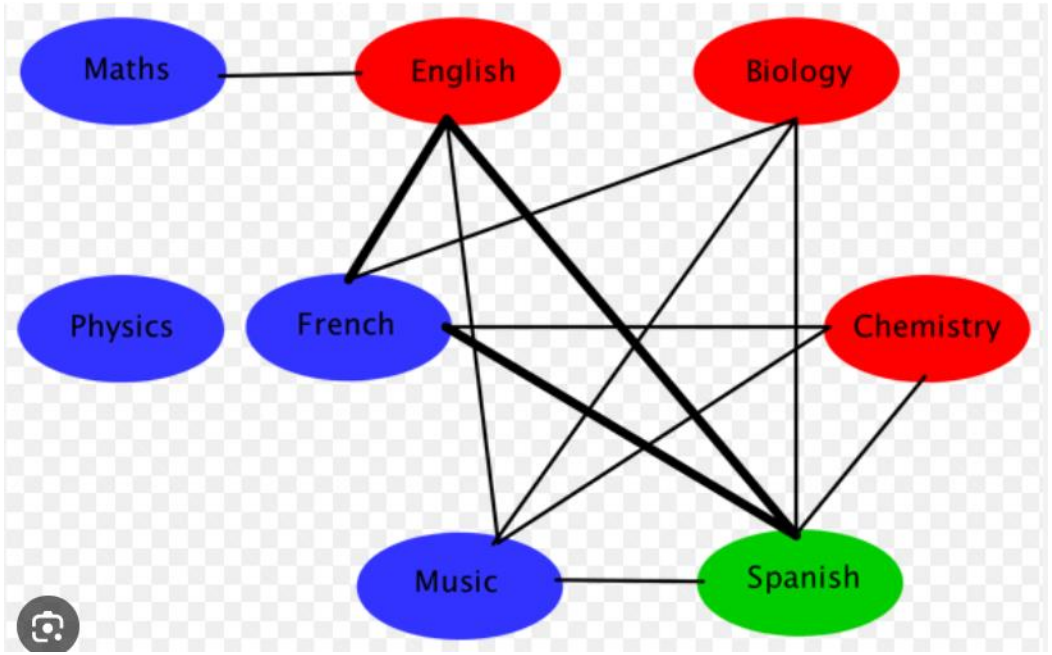


VARYING APPLICATIONS (EXAMPLES)

- Computer networks
- Solve shortest path problems between cities



VARYING APPLICATIONS (EXAMPLES)



- Computer networks
- Solve shortest path problems between cities
- **Scheduling exams**
  - Each vertex represents an exam
  - An edge between two vertices indicates that some students need to take both exams
  - Goal: To color each vertex (schedule exams) in such a way that no two adjacent vertices (exams taken by the same student) share the same color (are scheduled at the same time)
  - Minimizing the overall number of time slots (colors).
  - This ensures no student has overlapping exams.

HOW DO SEARCH ENGINES MANAGE TO ORGANIZE AND RETRIEVE ALL THIS INFORMATION?

USE OF **GRAPH THEORY** AND **GRAPH DATABASES** TO MODEL **RELATIONSHIPS BETWEEN ENTITIES, INDEX WEB PAGES, AND PROVIDE RELEVANT SEARCH RESULTS**

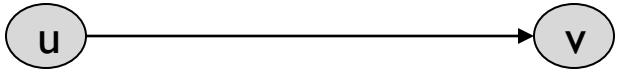
DEFINITIONS - GRAPH

A generalization of the simple concept of a set of dots, links, edges or arcs.

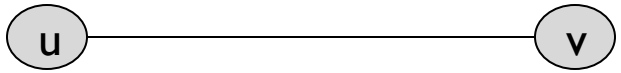
Representation: Graph G =(V, E) consists set of vertices denoted by V, or by V(G) and set of edges E, or E(G)

DEFINITIONS – EDGE TYPE

Directed: Ordered pair of vertices. Represented as (u, v) directed from vertex u to v.



Undirected: Unordered pair of vertices. Represented as {u, v}. Disregards any sense of direction and treats both end vertices interchangeably.

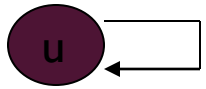


Twitter Follow network vs Facebook friendship network

DEFINITIONS – EDGE TYPE

- **Loop:** A loop is an edge whose endpoints are equal i.e., an edge joining a vertex to it self is called a loop. Represented as {u, u} = {u}

  Self transition in State Machine, Feedback loops in System



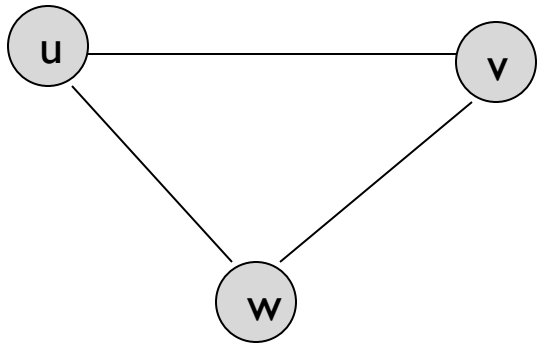
- **Multiple Edges:** Two or more edges joining the same pair of vertices.

  Transportation Networks: *For instance, there might be several flights, trains, or bus routes connecting the same two cities. Each of these routes can be represented as parallel edges in a graph that models the transportation network.*

DEFINITIONS – GRAPH TYPE

Simple (Undirected) Graph: consists of V, a nonempty set of vertices, and E, a set of unordered pairs of distinct elements of V called edges (undirected)

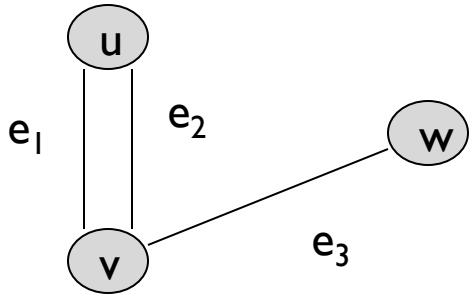
Representation Example: G(V, E), V = {u, v, w}, E = {{u, v}, {v, w}, {u, w}}



DEFINITIONS – GRAPH TYPE

Multigraph: G(V,E), consists of set of vertices V, set of Edges E and a function f from E to {{u, v}| u, v in V, u ≠ v}. The edges e1 and e2 are called multiple or parallel edges if f (e1) = f (e2).

Representation Example: $V = \{u, v, w\}, E = \{e_1, e_2, e_3\}$

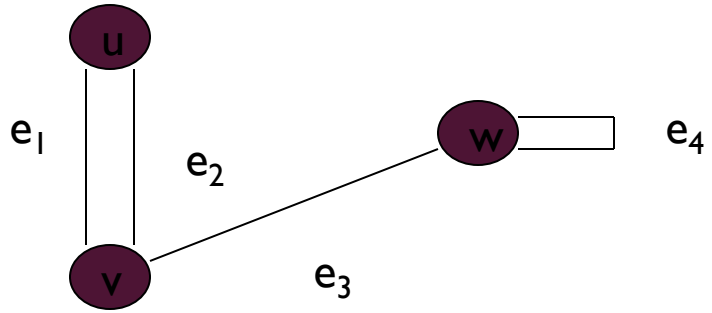


DEFINITIONS – GRAPH TYPE

Pseudograph:

- A multigraph that is permitted to have loops.

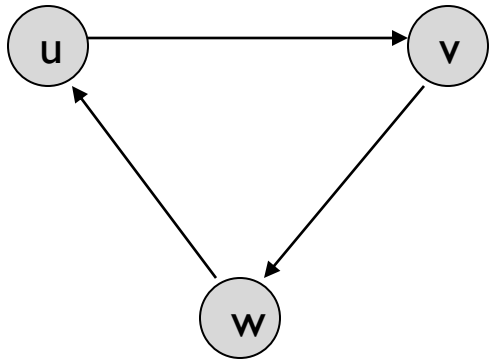
Representation Example: $V = \{u, v, w\}$, $E = \{e_1, e_2, e_3, e_4\}$



DEFINITIONS – GRAPH TYPE

Directed Graph: G(V, E), set of vertices V, and set of Edges E, that are ordered pair of elements of V (directed edges)

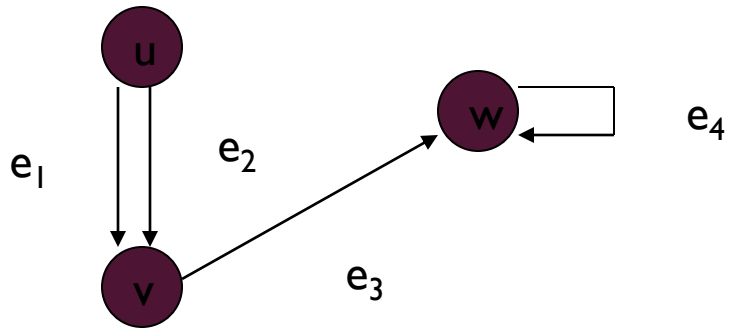
Representation Example: G(V, E), V = {u, v, w}, E = {(u, v), (v, w), (w, u)}



DEFINITIONS – GRAPH TYPE

Directed Multigraph: G(V,E), consists of set of vertices V, set of Edges E and a function f from E to {{u, v}| u, v in V}. The edges e1 and e2 are multiple edges if f(e1) = f(e2)

Representation Example: $V = \{u, v, w\}$, $E = \{e_1, e_2, e_3, e_4\}$

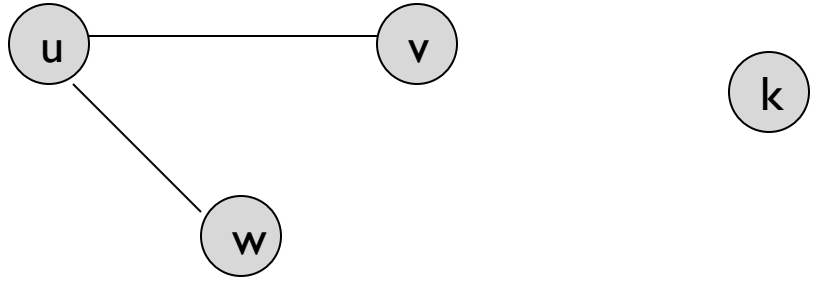


Type	Edges	Multiple Edges Allowed ?	Loops Allowed ?
Simple Graph	undirected	No	No
Multigraph	undirected	Yes	No
Pseudograph	undirected	Yes	Yes
Directed Graph	directed	No	Yes
Directed Multigraph	directed	Yes	Yes

TERMINOLOGY – UNDIRECTED GRAPHS

- u and v are **adjacent** if {u, v} is an edge, e is called **incident** with u and v. u and v are called **endpoints** of {u, v}
- **Degree of Vertex (deg (v)):** the number of edges incident on a vertex. A loop contributes twice to the degree (why?).
- **Pendant Vertex:** deg (v) = 1
- **Isolated Vertex:** deg (k) = 0

Representation Example: *For V = {u, v, w} , E = { {u, w}, {u, v} }, deg (u) = 2, deg (v) = 1, deg (w) = 1, deg (k) = 0, w and v are pendant , k is isolated*

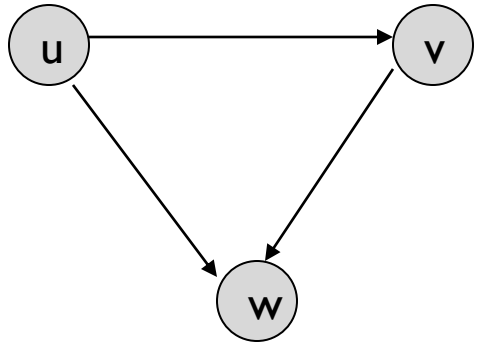


TERMINOLOGY – DIRECTED GRAPHS

- For the edge (u, v), u is **adjacent to** v OR v is **adjacent from** u, u – **Initial vertex**, v – **Terminal vertex**
- **In-degree (deg^- (u)):** number of edges for which u is terminal vertex
- **Out-degree (deg^+ (u)):** number of edges for which u is initial vertex

Note: A loop contributes 1 to both in-degree and out-degree (why?)

Representation Example: *For $V = \{u, v, w\}$, $E = \{ (u, w), (v, w), (u, v) \}$, $deg^-(u) = 0$, $deg^+(u) = 2$, $deg^-(v) = 1$, $deg^+(v) = 1$, and $deg^-(w) = 2$, $deg^+(u) = 0$*



Theorem 1

The Handshaking theorem: The Handshaking Theorem states that in any undirected graph, the sum of the degrees of all vertices is twice the number of edges

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

(why?) Every edge connects 2 vertices.

Each edge contributes twice to the total degree count of all vertices. Thus, both sides of the equation eq ual to twice the number of edges.

**It is called the "handshaking lemma" because it can be visualised as a gathering of people where each handshake involves two individuals, analogous to edges connecting vertices in a graph.

FEW EXAMPLES…

- In a social network, if the sum of degrees of all users is 90 , how many connections (edges) are there in the network?

Let 'E' be the number of edges.

According to the Handshaking Theorem, the sum of degrees equals $2 \times E$.

So, we have $90 = 2 \times E$, which implies $E = 45$.

There are 45 connections in the social network.

FEW EXAMPLES…

- A sports tournament has 12 teams, and each team plays against every other team exactly once. How many total matches were played?

- In a classroom, there are 25 students. If each student shakes hands with exactly 3 other students, how many total handshakes occur?

Theorem 2:

An undirected graph has even number of vertices with odd degree

***Proof*:** Let V_1 be the vertices of even degree and V_2 be the vertices of odd degree in an undirected graph $G = (V, E)$ with m edges. Then

even → $2m = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v).$

must be even since $\deg(v)$ is even for each $v \in V_1$

This sum must be even because $2m$ is even and the sum of the degrees of the vertices of even degrees is also even. Because this is the sum of the degrees of all vertices of odd degree in the graph, there must be an even number of such vertices.

- **Theorem 3:** Let G = (V, E) be a graph with directed edges

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

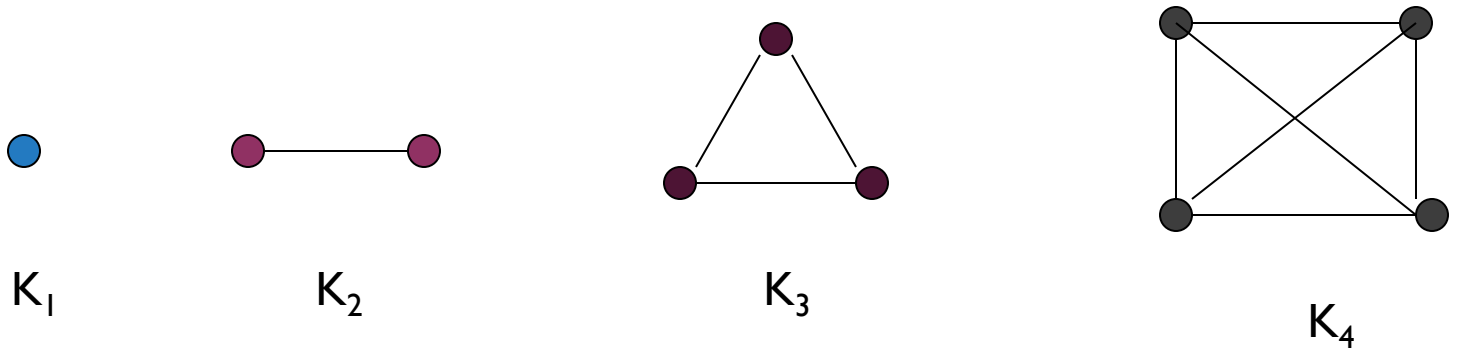
Proof: The first sum counts the number of outgoing edges over all vertices and the second sum counts the number of incoming edges over all vertices. It follows that both sums equal the number of edges in the graph. ◀

In a directed graph, every edge has a direction, going from one vertex to another. This means that every edge contributes exactly once to the out-degree of one vertex (the source) and once to the in-degree of another vertex (the target). Therefore, if we sum up the out-degrees of all vertices, we count each edge exactly once as it leaves its source. Similarly, if we sum up the in-degrees of all vertices, we again count each edge exactly once as it enters its target.

SIMPLE GRAPHS – SPECIAL CASES

- **Complete graph:** K_n, is the simple graph that contains exactly one edge between each pair of distinct vertices.

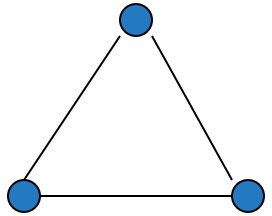
  Representation Example: K_1, K_2, K_3, K_4



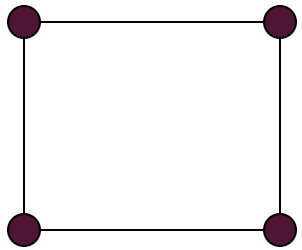
SIMPLE GRAPHS – SPECIAL CASES

- **Cycle:** C_n, $n \geq 3$ consists of n vertices $v_1, v_2, v_3 \ldots v_n$ and edges $\{v_1, v_2\}$, $\{v_2, v_3\}, \{v_3, v_4\} \ldots \{v_{n-1}, v_n\}, \{v_n, v_1\}$

  Representation Example: C_3, C_4



C_3

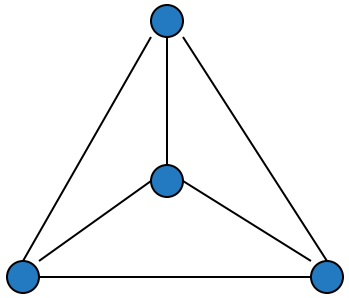


C_4

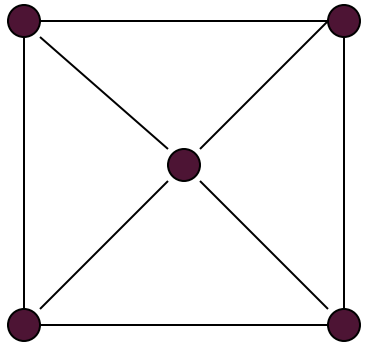
SIMPLE GRAPHS – SPECIAL CASES

- **Wheels:** W_n, obtained by adding additional vertex to Cn and connecting all vertices to this new vertex by new edges.

  Representation Example: W_3, W_4



W_3



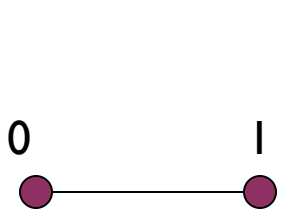
W_4

SIMPLE GRAPHS – SPECIAL CASES

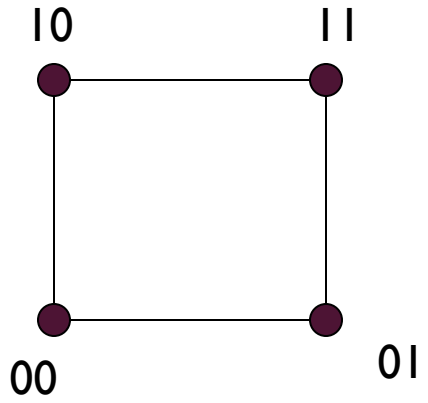
- **N-cubes:**

An *n-dimensional hypercube*, or *n-cube*, Q_n, is a graph with 2^n vertices representing all bit strings of length n, where there is an edge between two vertices that differ in exactly one bit position.

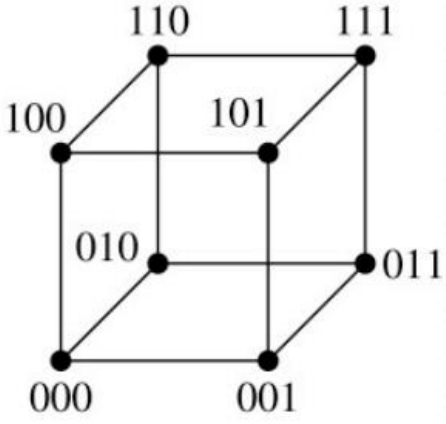
Representation Example: Q_1, Q_2



Q_1



Q_2

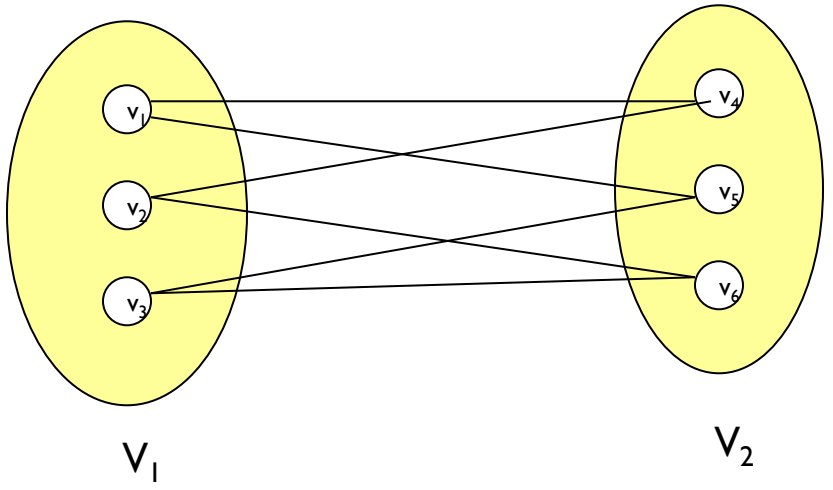


BIPARTITE GRAPHS

- In a simple graph G, if V can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 and a vertex V_2 (so that no edge in G connects either two vertices in V_1 or two vertices in V_2)

  Application example: Representing Relations

  Representation example: $V_1 = \{v_1, v_2, v_3\}$ and $V_2 = \{v_4, v_5, v_6\}$,

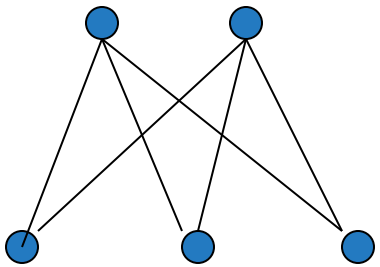


In the context of employment or job matching, a bipartite graph can effectively model the relationship between job applicants and available positions.

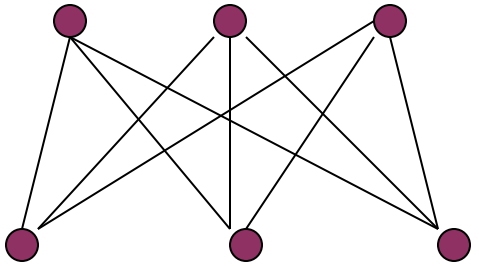
COMPLETE BIPARTITE GRAPHS

- $K_{m,n}$ is the graph that has its vertex set portioned into two subsets of m and n vertices, respectively. There is an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset.

  Representation example: $K_{2,3}$, $K_{3,3}$



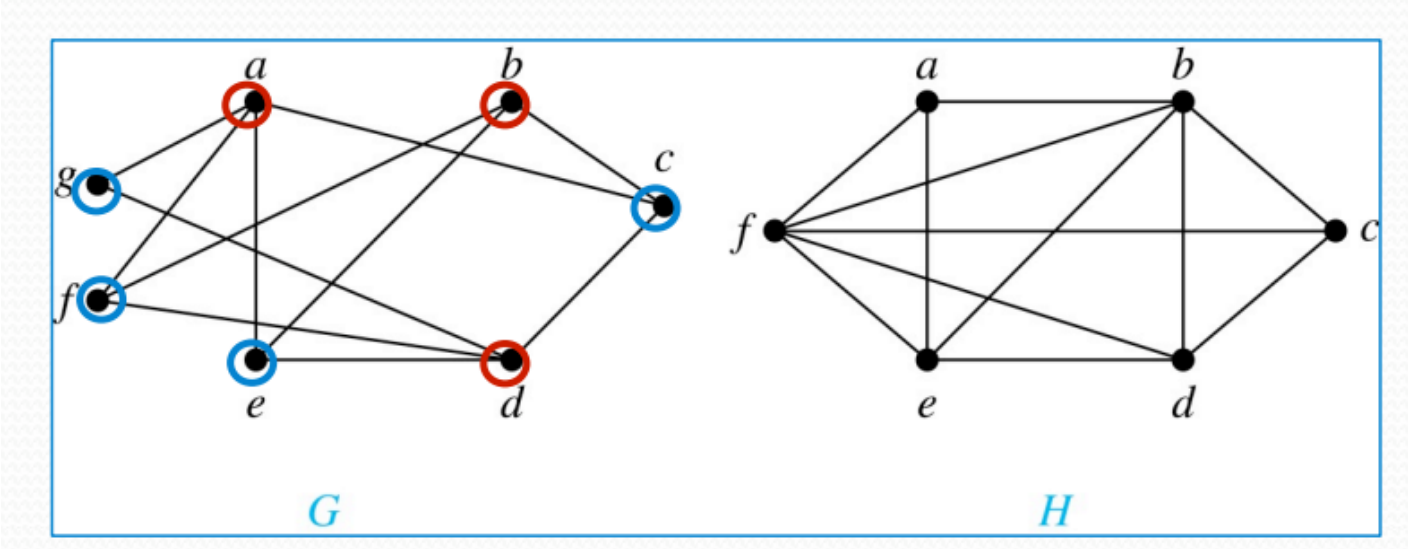
$K_{2,3}$



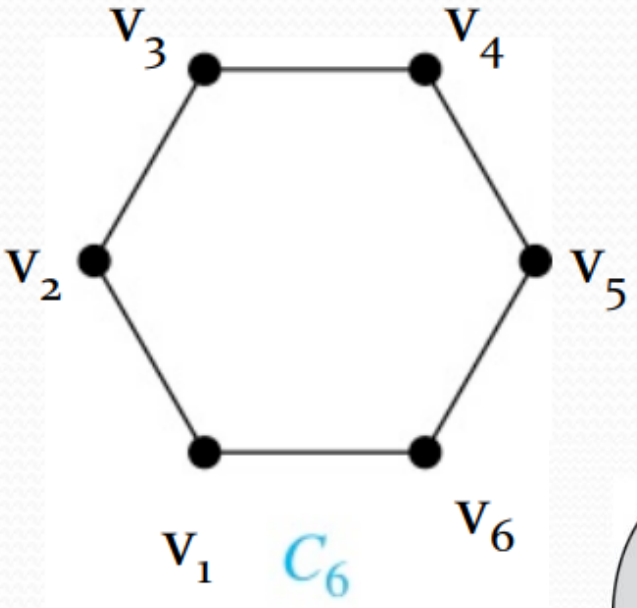
$K_{3,3}$

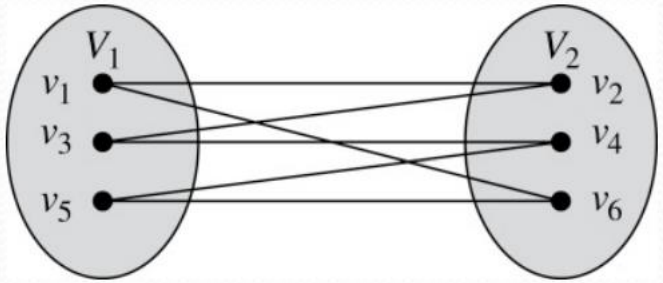
HOW TO VALIDATE IF IT IS A BIPARTITE GRAPH OR NOT!

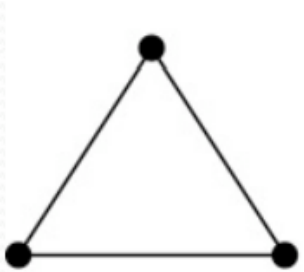
- It is not hard to show that an equivalent definition of a bipartite graph is a graph where it is possible to color the vertices red or blue so that no two adjacent vertices are the same color



SHOW C_6 AND C_3 IS BIPARTITE OR NOT?







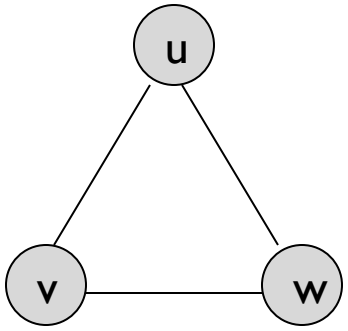
If we divide the vertex set of C3 into tw o nonempty sets, one of the two must c ontain two vertices. But in C3 every ve rtex is connected to every other vertex. Therefore, the two vertices in the same partition are connected. Hence, C3 is no t bipartite

SUBGRAPHS

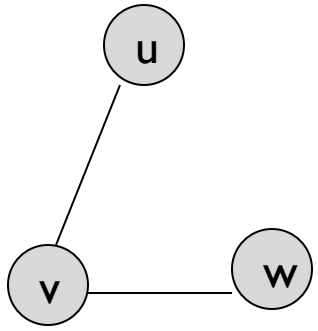
- A subgraph of a graph G = (V, E) is a graph H =(V', E') where V' is a subset of V and E' is a subset of E

  Application example: solving sub-problems within a graph

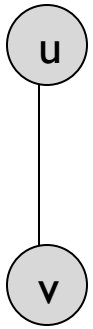
  Representation example: $V = \{u, v, w\}$, $E = (\{u, v\}, \{v, w\}, \{w, u\})$, H_1 , H_2



G



H_1

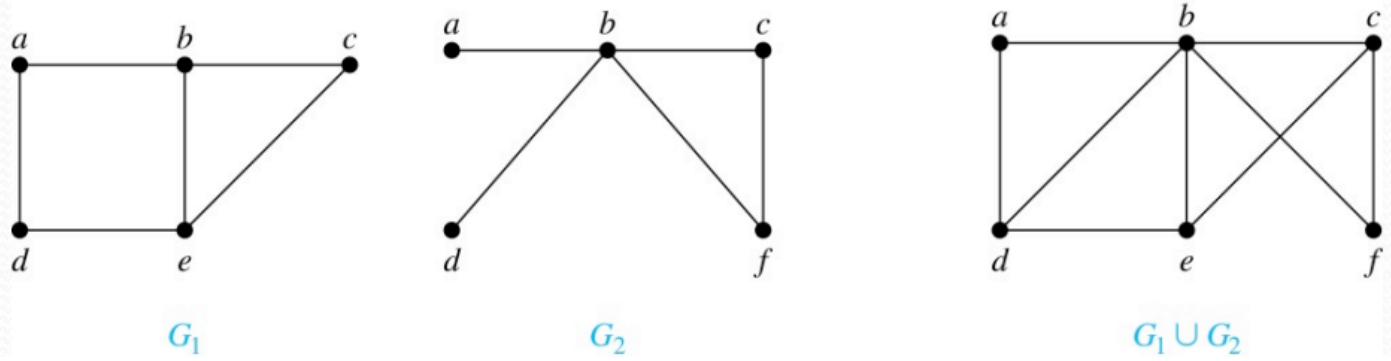


H_2

SUBGRAPHS

Definition: The *union* of two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the simple graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$. The union of G_1 and G_2 is denoted by $G_1 \cup G_2$.

Example:



REPRESENTATION

- **Incidence (Matrix):** Most useful when information about edges is more desirable than information about vertices.

- **Adjacency (Matrix/List):** Most useful when information about the vertices is more desirable than information about the edges. These two representations are also most popular since information about the vertices is often more desirable than edges in most applications

- G = (V, E) be an undirected graph. Suppose that $v_1, v_2, v_3, \ldots, v_n$ are the vertices and $e_1, e_2, \ldots, e_m$ are the edges of G. Then the incidence matrix with respect to this ordering of V and E is the nx m matrix M = $[m_{ij}]$, where

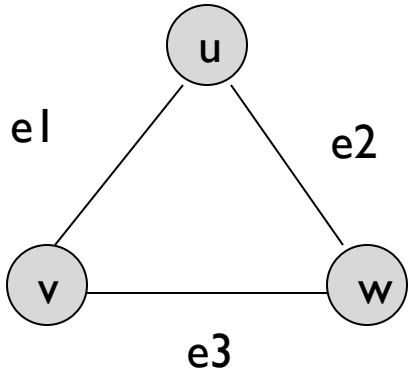
$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise} \end{cases}$$

Can also be used to represent :

Multiple edges: by using columns with identical entries, since these edges are incident with the same pair of vertices

Loops: by using a column with exactly one entry equal to 1, corresponding to the vertex that is incident with the loop

- Representation Example: G = (V, E)



	e_1	e_2	e_3
v	1	0	1
u	1	1	0
w	0	1	1

- There is an N x N matrix, where |V| = N , the Adjacency Matrix (NxN) A = $[a_{ij}]$

For undirected graph

$$a_{ij} = \begin{cases} 1 \text{ if } \{v_i, v_j\} \text{ is an edge of } G \\ 0 \text{ otherwise} \end{cases}$$

- **For directed graph**

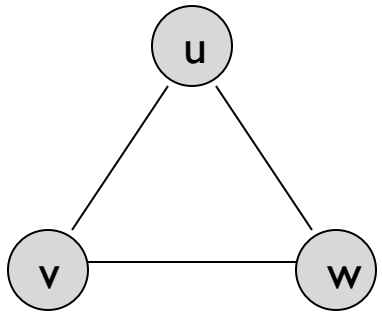
$$a_{ij} = \begin{cases} 1 \text{ if } (v_i, v_j) \text{ is an edge of } G \\ 0 \text{ otherwise} \end{cases}$$

- This makes it easier to find subgraphs, and to reverse graphs if needed.

REPRESENTATION- ADJACENCY MATRIX

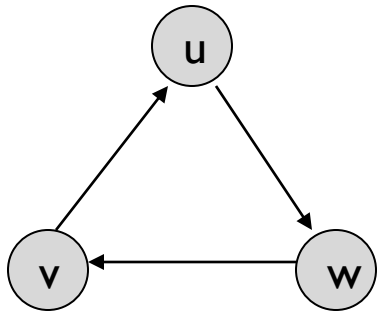
- Adjacency is chosen on the ordering of vertices. Hence, there as are as many as n! such matrices.
- The adjacency matrix of simple graphs are symmetric ($a_{ij} = a_{ji}$) (why?)
- When there are relatively few edges in the graph the adjacency matrix is a **sparse matrix**
- Directed Multigraphs can be represented by using aij = number of edges from v_i to v_j

- Example: Undirected Graph G (V, E)



	v	u	w
v	0	1	1
u	1	0	1
w	1	1	0

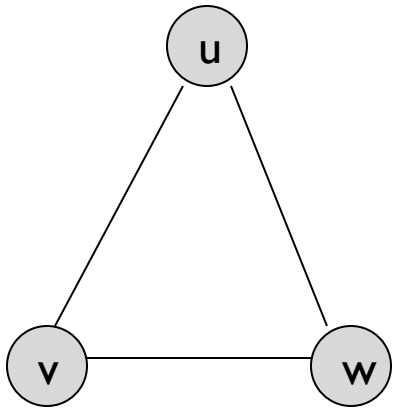
- Example: directed Graph G (V, E)



	v	u	w
v	0	1	0
u	0	0	1
w	1	0	0

Each node (vertex) has a list of which nodes (vertex) it is adjacent

Example: undirectd graph G (V, E)



node	Adjacency List
u	v , w
v	w, u
w	u , v

GRAPH - ISOMORPHISM

- G1 = (V1, E2) and G2 = (V2, E2) are isomorphic if:
- There is a one-to-one and onto function f from V1 to V2 with the property that
  - a and b are adjacent in G1 if and only if f (a) and f (b) are adjacent in G2, for all a and b in V1.
- Function f is called isomorphism

Application Example:

In chemistry, to find if two compounds have the same structure

GRAPH - ISOMORPHISM

Representation example: G1 = (V1, E1) , G2 = (V2, E2)

$f(u_1) = v_1, f(u_2) = v_4, f(u_3) = v_3, f(u_4) = v_2,$

CONNECTIVITY

- Basic Idea: In a Graph Reachability among vertices by traversing the edges

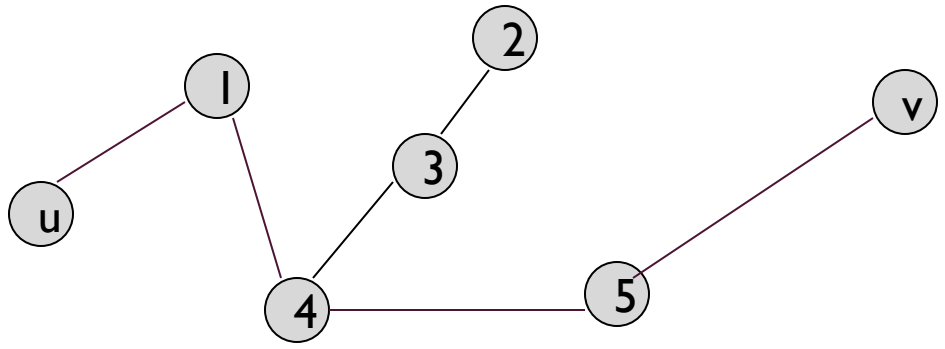
  Application Example:

  - In a city to city road-network, if one city can be reached from another city.
  - Problems if determining whether a message can be sent between two computer using intermediate links
  - Efficiently planning routes for data delivery in the Internet

CONNECTIVITY – PATH

A **Path** is a sequence of edges that begins at a vertex of a graph and travels along edges of the graph, always connecting pairs of adjacent vertices.

Representation example: G = (V, E), Path P represented, from u to v is {{u, 1}, {1, 4}, {4, 5}, {5, v}}



CONNECTIVITY – PATH

Definition for Directed Graphs

A **Path** of length n (> 0) from u to v in G is a sequence of n edges $e_1, e_2, e_3, \ldots, e_n$ of G such that f $(e_1) = (x_o, x_1)$, f $(e_2) = (x_1, x_2)$, …, f $(e_n) = (x_{n-1}, x_{n)}$, where $x_0 = u$ and $x_n = v$. A path is said to pass through $x_0, x_1, \ldots, x_n$ or traverse $e_1, e_2, e_3, \ldots, e_n$

For Simple Graphs, sequence is $x_0, x_1, \ldots, x_n$

In directed multigraphs when it is not necessary to distinguish between their edges, we can use sequence of vertices to represent the path

Circuit/Cycle: u = v, length of path > 0

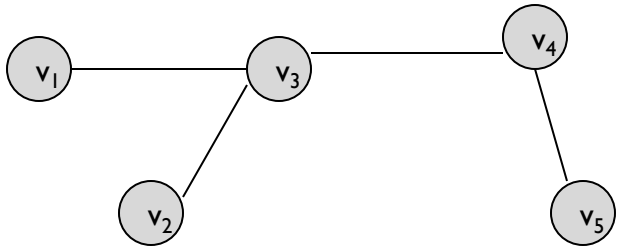
Simple Path: does not contain an edge more than once

CONNECTIVITY – CONNECTEDNESS

Undirected Graph

An undirected graph is connected if there exists is a simple path between every pair of vertices

Representation Example: G (V, E) is connected since for V = $\{v_1, v_2, v_3, v_4, v_5\}$, there exists a path between $\{v_i, v_j\}$, $1 \leq i, j \leq 5$



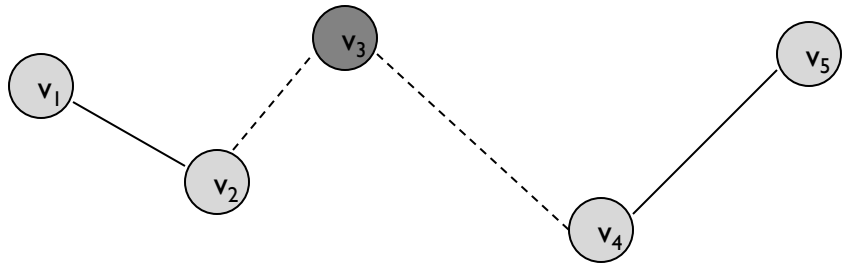
CONNECTIVITY – CONNECTEDNESS

Undirected Graph

- **Articulation Point (Cut vertex):** removal of a vertex produces a subgraph with more connected components than in the original graph. The removal of a cut vertex from a connected graph produces a graph that is not connected

- **Cut Edge:** An edge whose removal produces a subgraph with more connected components than in the original graph.

  Representation example: G (V, E), v_3 is the articulation point or edge $\{v_2, v_3\}$, the number of connected components is 2 (> 1)

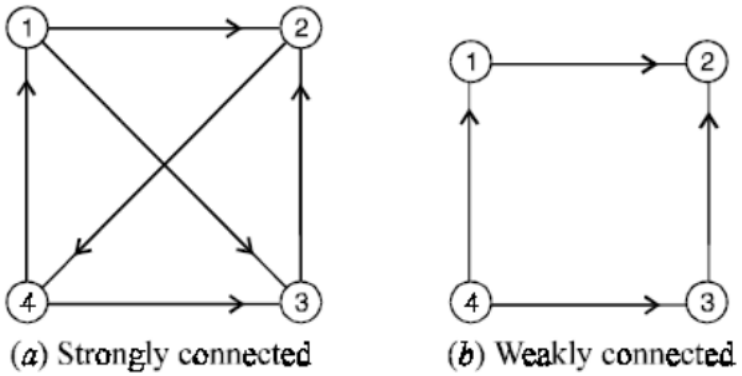


CONNECTIVITY – CONNECTEDNESS

Directed Graph

- A directed graph is **strongly connected** if there is a path from a to b and from b to a whenever a and b are vertices in the graph
- A directed graph is **weakly connected** if there is a (undirected) path between every two vertices in the underlying undirected path

A strongly connected Graph can be weakly connected but the vice-versa is not true

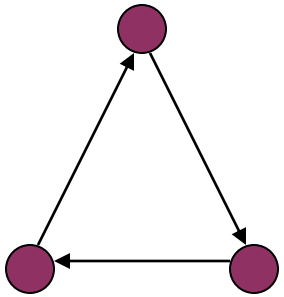


(*a*) Strongly connected (*b*) Weakly connected

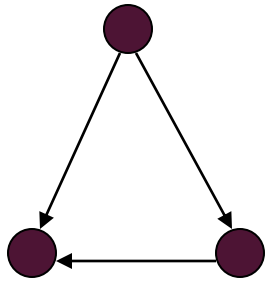
CONNECTIVITY – CONNECTEDNESS

Directed Graph

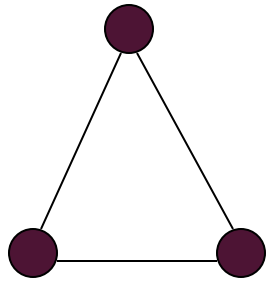
Representation example: G1 (Strong component), G2 (Weak Component), G3 is undirected graph representation of G2 or G1



G1



G2



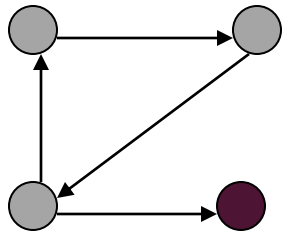
G3

CONNECTIVITY – CONNECTEDNESS

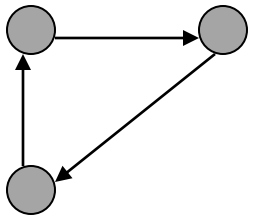
- **Directed Graph**

  Strongly connected Components: subgraphs of a Graph G that are strongly connected

  Representation example: G1 is the strongly connected component in G



G

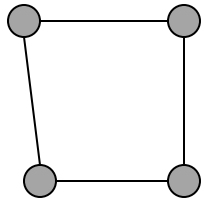


G1

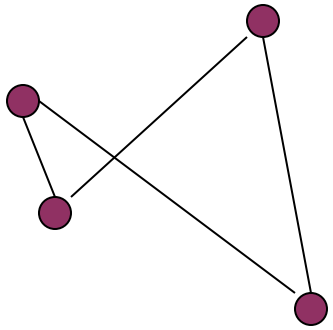
ISOMORPHISM - REVISITED

A isomorphic invariant for simple graphs is the existence of a simple circuit of length k , k is an integer > 2 (why ?)

Representation example: G1 and G2 are isomorphic since we have the invariants, similarity in degree of nodes, number of edges, length of circuits



G1



G2

COUNTING PATHS

- **Theorem:** Let G be a graph with adjacency matrix A with respect to the ordering $v_1, v_2, \ldots, V_n$ (with directed on undirected edges, with multiple edges and loops allowed). The number of different paths of length r from Vi to Vj, where r is a positive integer, equals the $(i, j)^{th}$ entry of (adjacency matrix) $A^{r.}$

Proof: By Mathematical Induction.

Base Case: For the case N = 1, a_{ij} =1 implies that there is a path of length 1. This is true since this corresponds to an edge between two vertices.

We assume that theorem is true for N = r and prove the same for N = r +1. Assume that the $(i, j)^{th}$ entry of A^r is the number of different paths of length r from v_i to v_j. By induction hypothesis, b_{ik} is the number of paths of length r from v_i to v_k.

COUNTING PATHS

Case r +1: In $A^{r+1} = A^r . A$,

The $(i, j)^{th}$ entry in A^{r+1} , $b_{i1}a_{1j} + b_{i2}\, a_{2j} + \ldots + b_{in}\, a_{nj}$

where b_{ik} is the $(i, j)^{th}$ entry of A^r.

By induction hypothesis, b_{ik} is the number of paths of length r from v_i to v_k.

The $(i, j)^{th}$ entry in A^{r+1} corresponds to the length between i and j and the length is r+1. This path is made up of length r from v_i to v_k and of length from v_k to vj. By product rule for counting, the number of such paths is $b_{ik*}\, a_{kj}$ The result is $b_{i1}a_{1j} + b_{i2}\, a_{2j} + \ldots + b_{in}\, a_{nj}$, the desired result.