DSA through C++

Graph

Saurabh Shukla (MySirG)

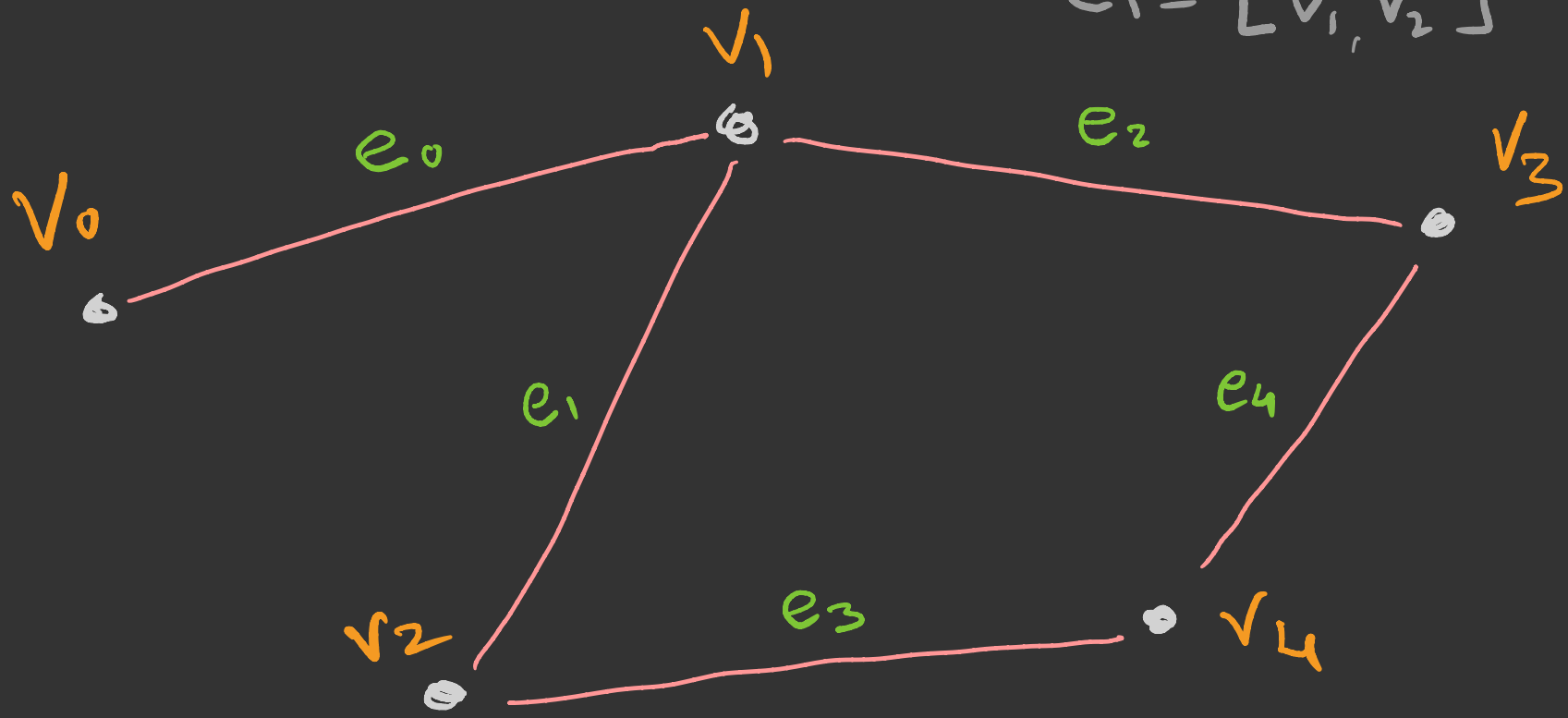
Agenda

1. Graph
2. Adjacent nodes
3. Degree of a node
4. Path
5. Connected Graph
6. Labelled & Weighted Graph
7. Multi Graph & Directed Graph
8. Complete Graph
9. Representation of Graph

Graph

- Graph is a non linear data structure

$e_0 = [v_0, v_1]$
$e_1 = [v_1, v_2]$



$V = \{ v_0, v_1, v_2, v_3, v_4 \}$

$E = \{ e_0, e_1, e_2, e_3, e_4 \}$

Graph

- A Graph Consists of two things
  - A set V of elements called nodes.
  - A set E of edges such that each edge e in E is identified with a unique (unordered) pair [u, v] of nodes in V, denoted by e = [u, v]
  - We indicate the parts of the graph by writing G = (V, E)

Adjacent nodes

If $e = [u, v]$, then u and v are called **adjacent nodes** or **neighbors**.

Degree of node

The degree of node u, written $\deg(u)$, is the number of edges containing u.

If $\deg(u) = 0$, then u is called isolated node.

Path

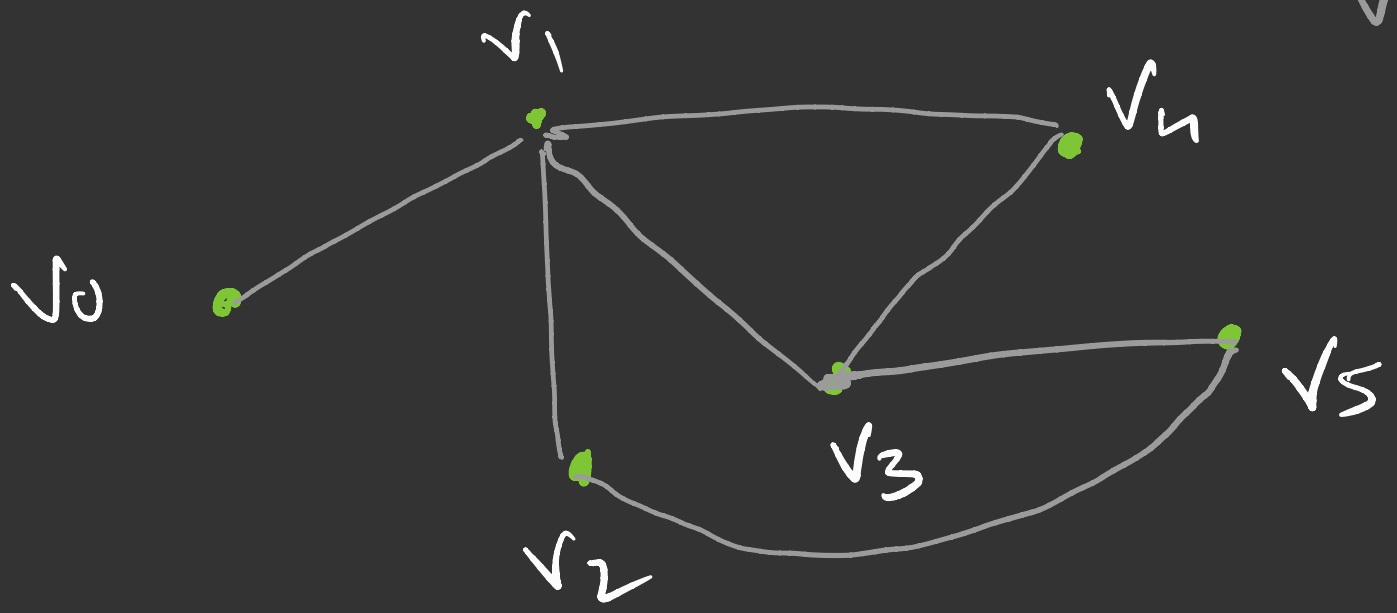
A path of length n from a node u to a node v is defined as a sequence of n+1 nodes.

$$P = (v_0, v_1, v_2, \cdots v_n)$$

The path is said to be closed if $v_0 = v_n$

Simple & Complex Path

The path is said to be simple if all the nodes are distinct, with the exception that v_0 may equal to v_n otherwise it is complex path.

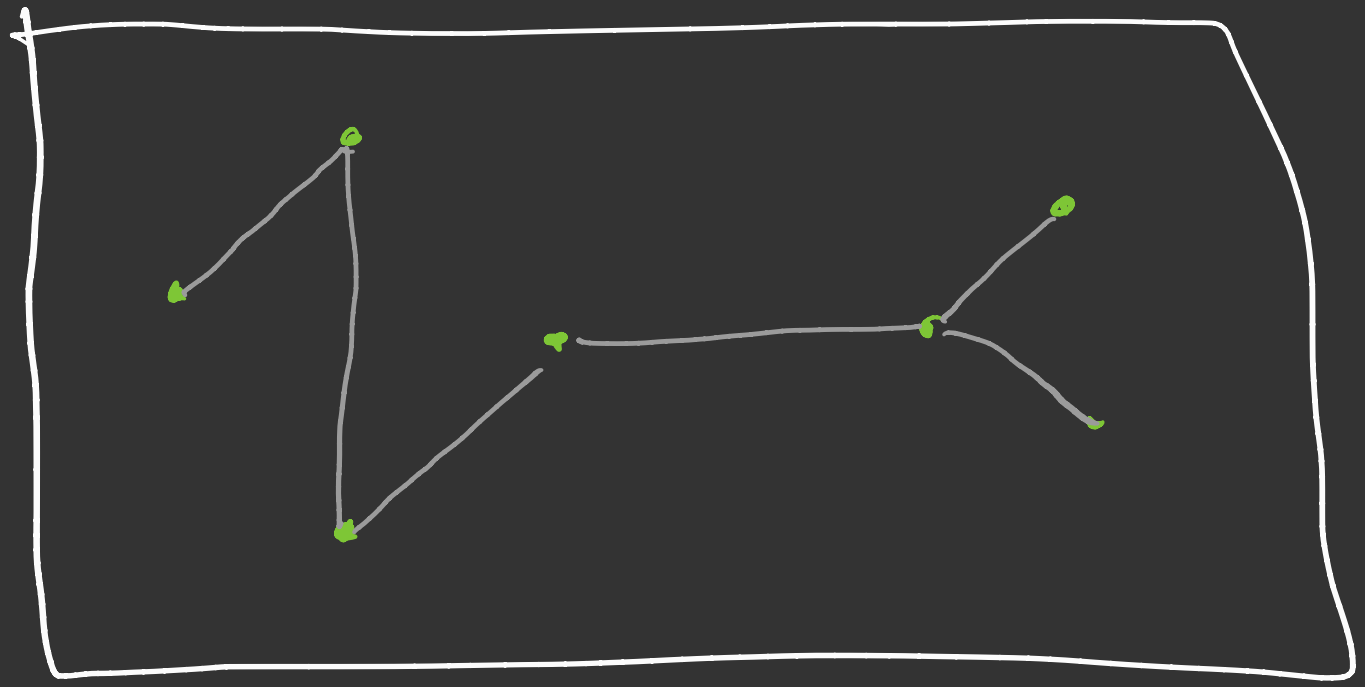


$v_0\ v_1\ v_4\ v_3$

$v_1\ v_2\ v_5\ v_3$

Connected Graph

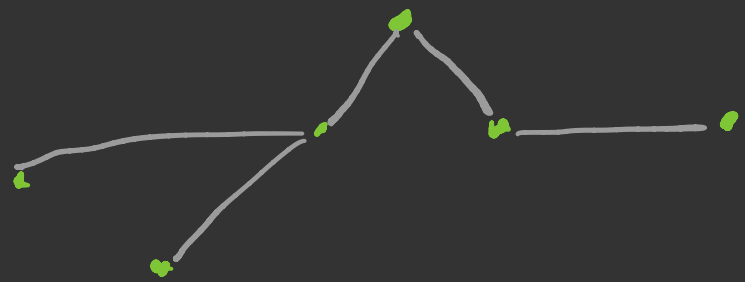
A graph is said to be connected if there is a path between any two of its nodes.



Tree Graph

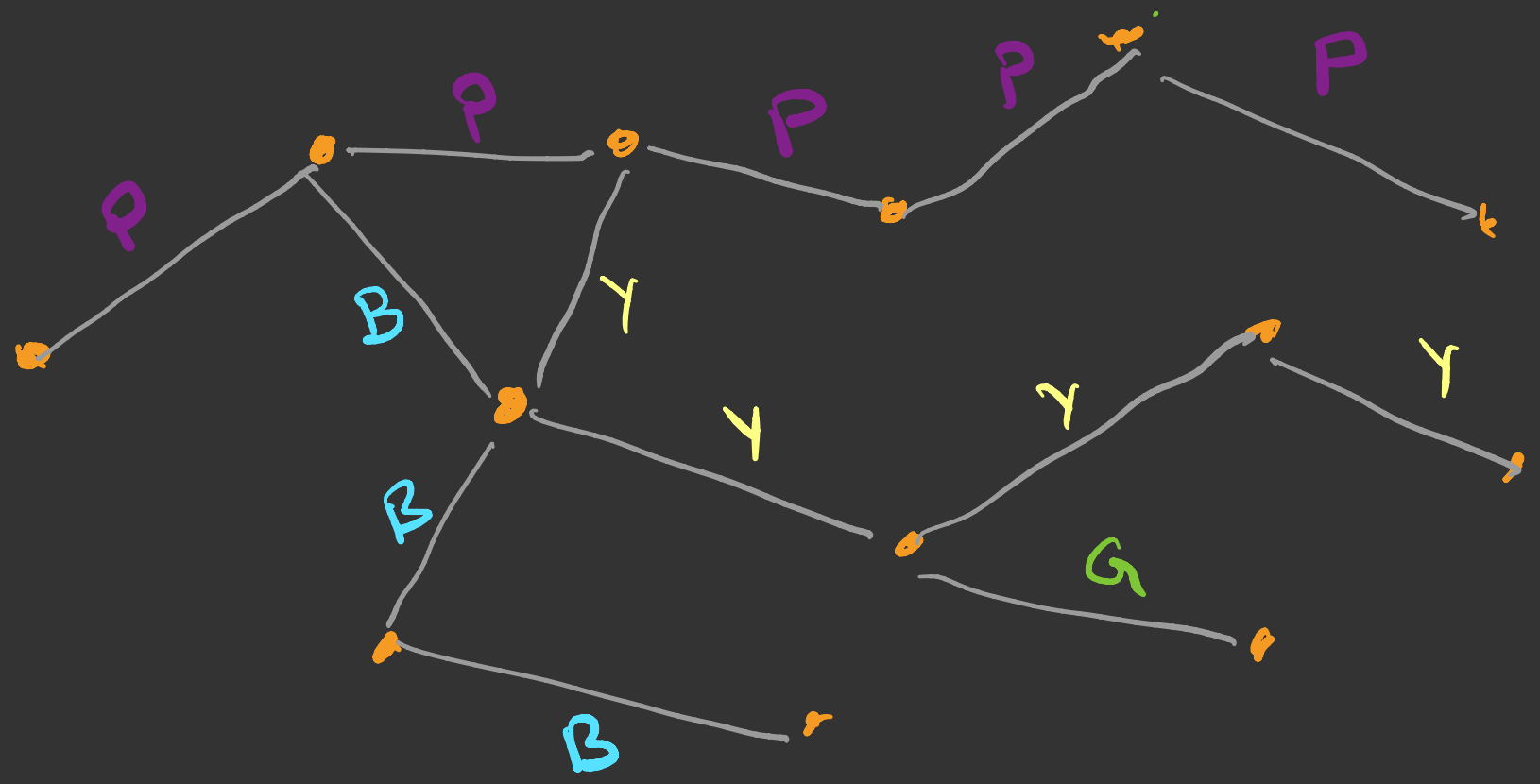
A connected graph T without any cycles is called a **tree graph** or **free tree**, or simply a **tree**.

This means in particular, that there is a unique simple path P between any two nodes u and v.



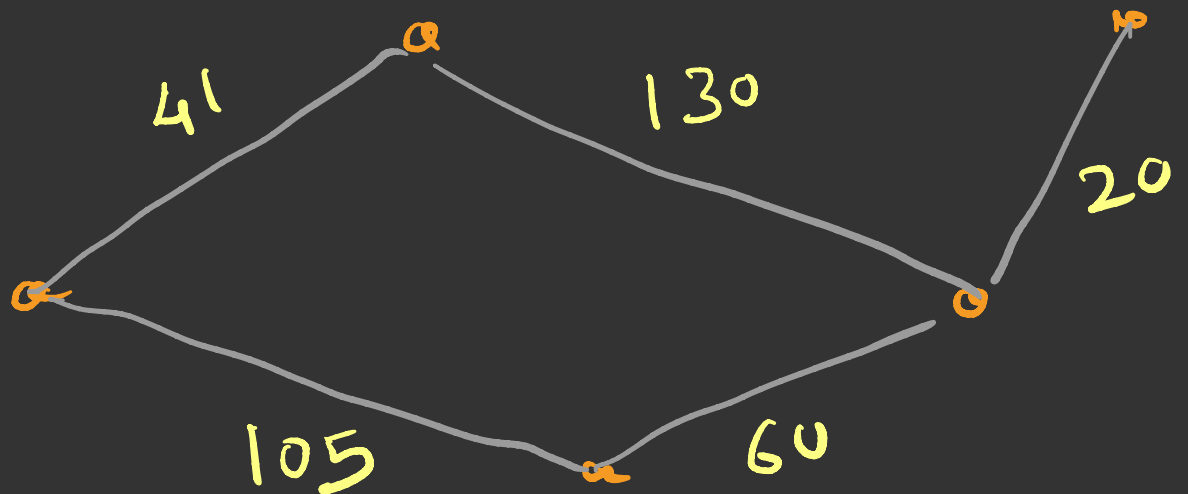
Labelled Graph

A graph is to be labelled if its edges are assigned data.



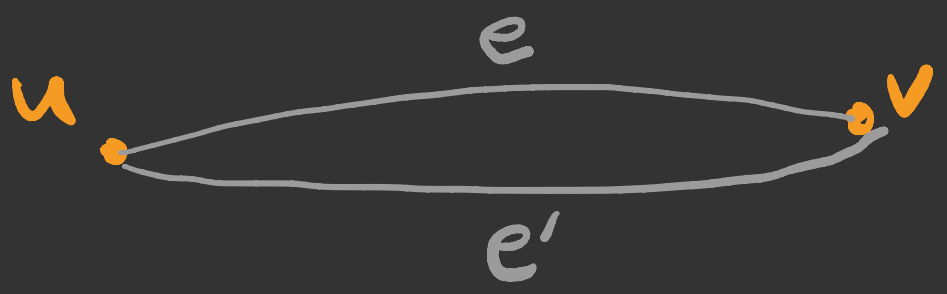
Weighted Graph

A graph G is said to be weighted if each edge e in G is assigned a non-negative numerical value w(e) called the weight or length of e.



Multiple edges

Distinct edges e and e' are called multiple edges if they connect the same end points, that is, if $e=[u,v]$ and $e'=[u,v]$



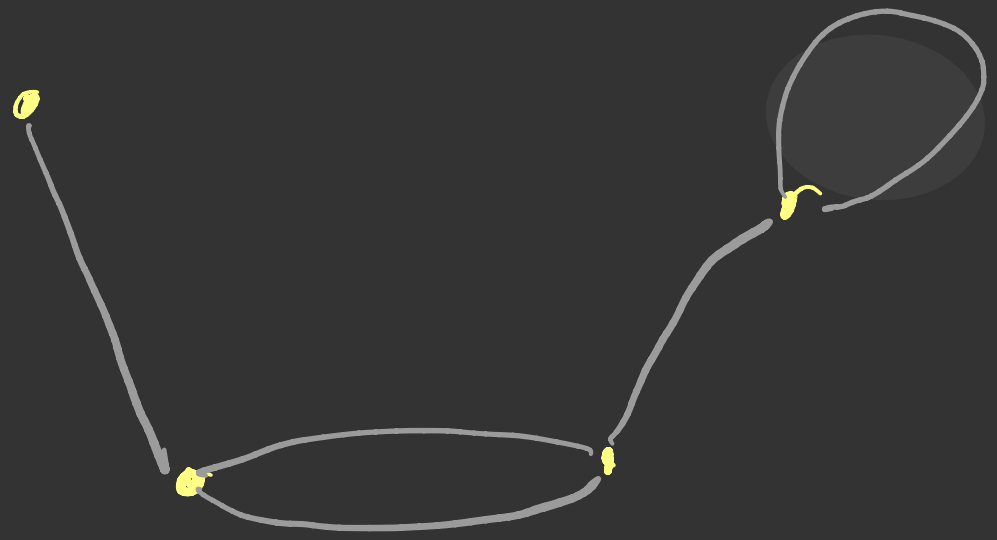
Loop

An edge is called loop if it has identical end points.

$e = [u, u]$

Multi Graph

Multi Graph is a graph consisting of multiple edges and loops.



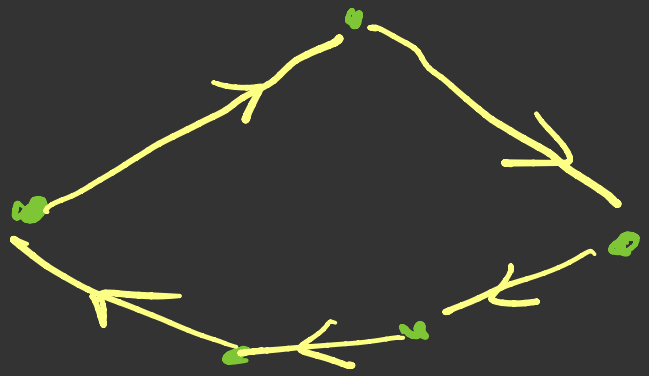
multigraph

Directed Graph

A directed graph G also called digraph is same as multigraph except that each edge e is assigned a direction.

$$e = (u, v)$$

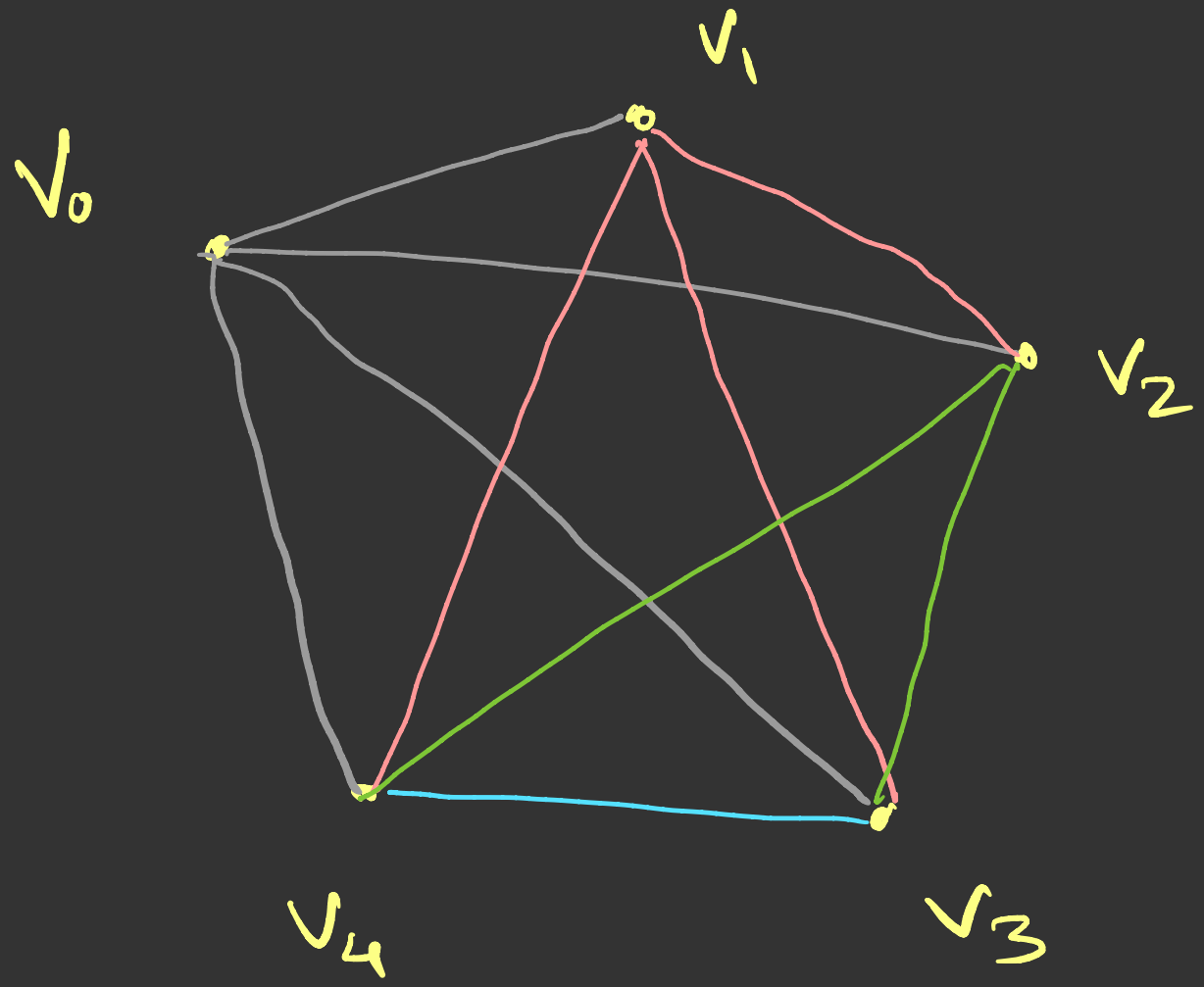
ordered pair



Complete Graph

A simple graph in which there exists an edge between every pair of vertices is called a complete graph.

It is also known as a universal graph or clique.


v_0
v_1
v_2
v_3
v_4

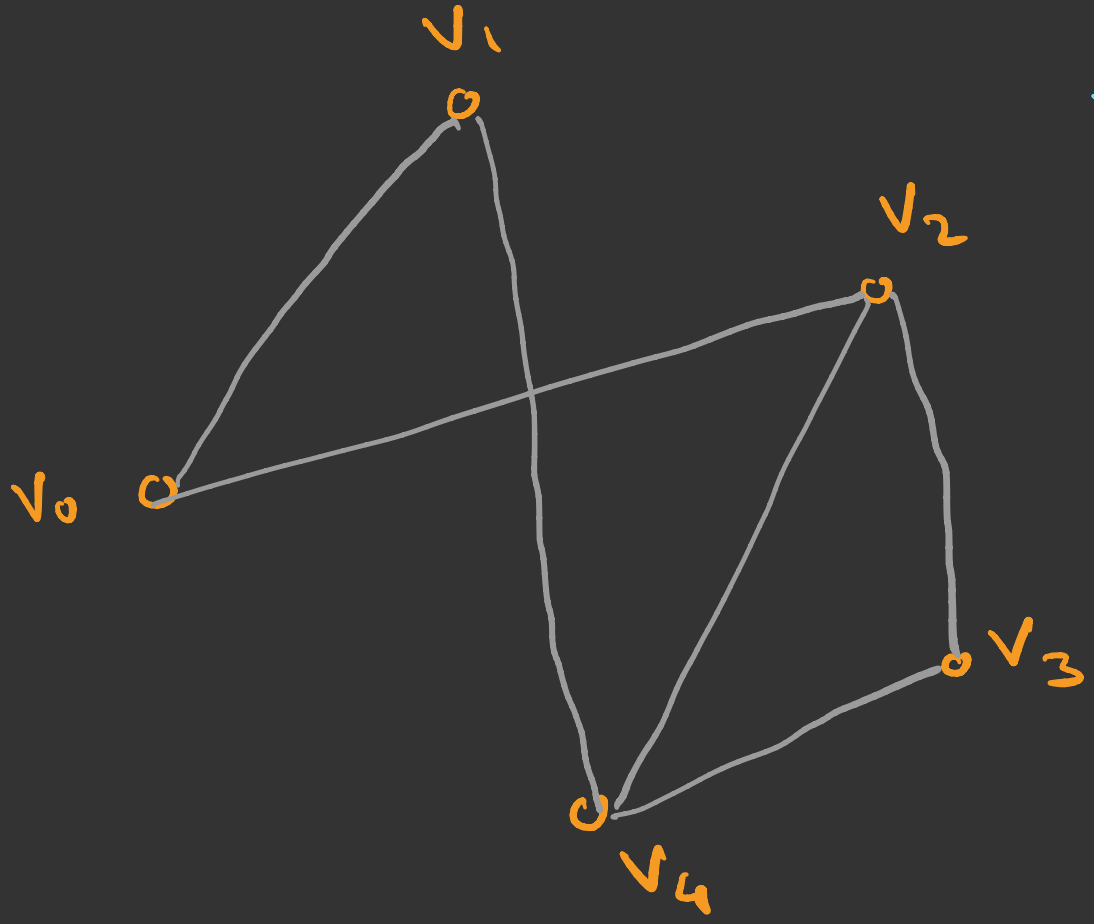
Representation of Graph

1. Adjacency Matrix Representation
2. List Representation

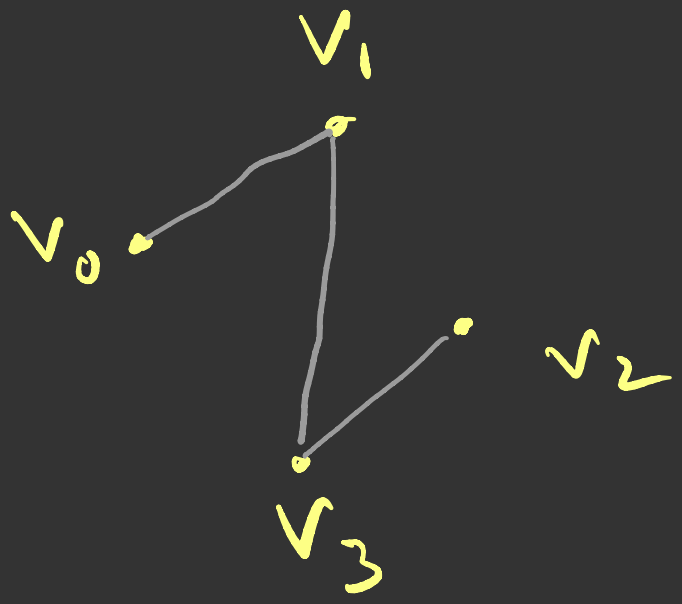
Adjacency Matrix Representation

Suppose G is a simple graph with m nodes, and suppose the nodes of G have been ordered and are called $v_1\ v_2\ v_3\ \cdots\ v_m$. Then the adjacency matrix $A = (a_{ij})$ of the graph G is the $m \times m$ matrix defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

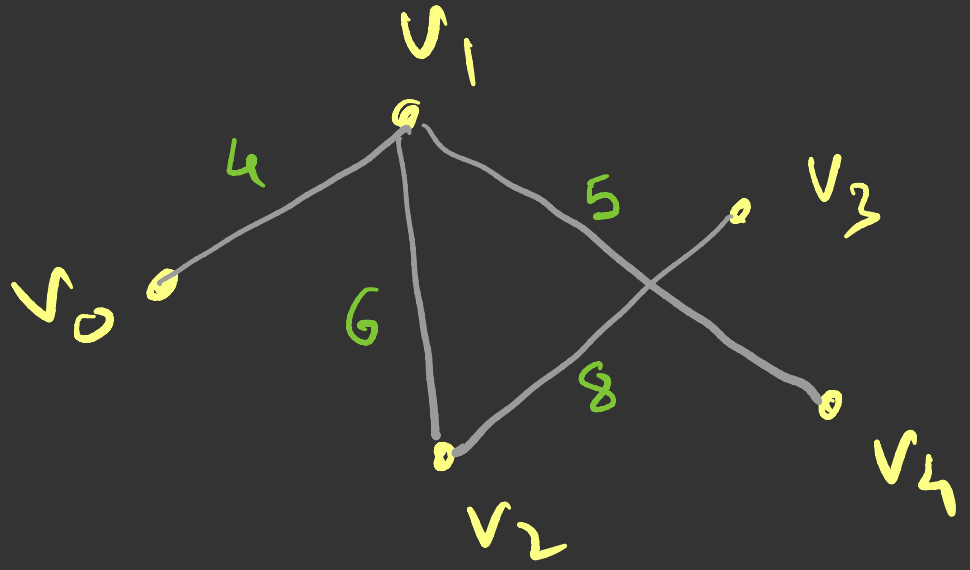


	V_0	V_1	V_2	V_3	V_4
V_0	0	1	1	0	0
V_1	1	0	0	0	1
V_2	1	0	0	1	1
V_3	0	0	1	0	1
V_4	0	1	1	1	0

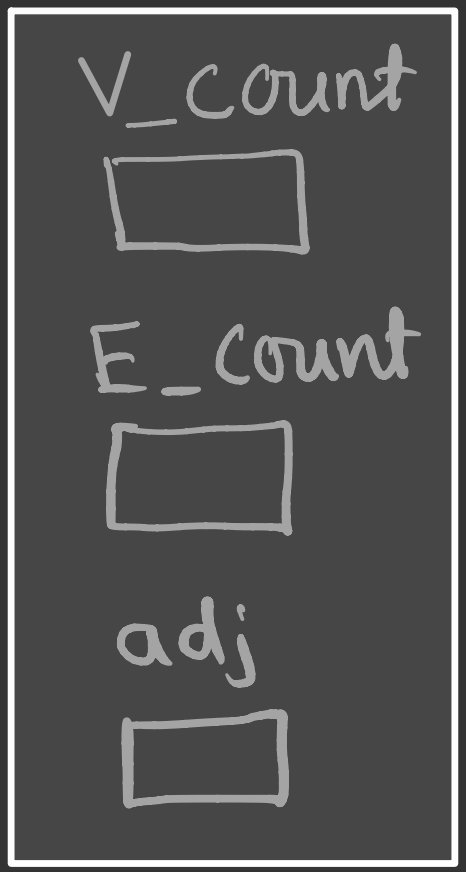


	v_0	v_1	v_2	v_3
v_0	0	1	0	0
v_1	1	0	0	1
v_2	0	0	0	1
v_3	0	1	1	0

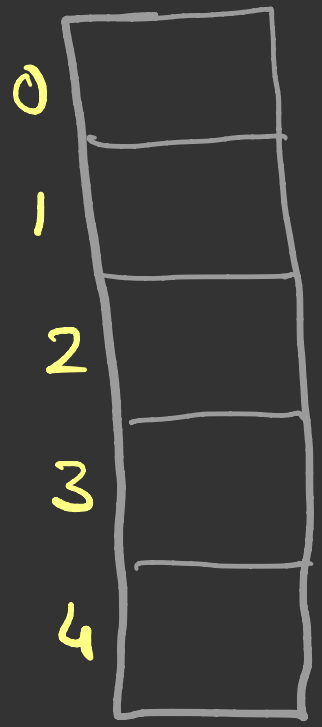
Weighted Graph

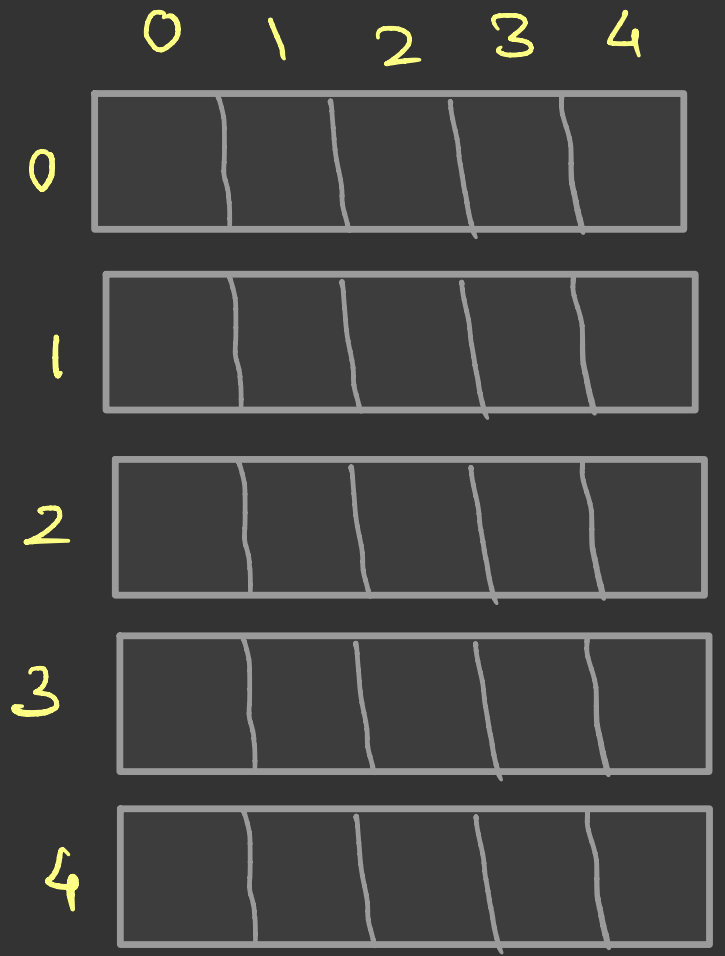


	v_0	v_1	v_2	v_3	v_4
v_0	0	4	0	0	0
v_1	4	0	6	0	5
v_2	0	6	0	8	0
v_3	0	0	8	0	0
v_4	0	5	0	0	0



Graph object





Adjacency List Representation

The graph is stored as a linked structure.

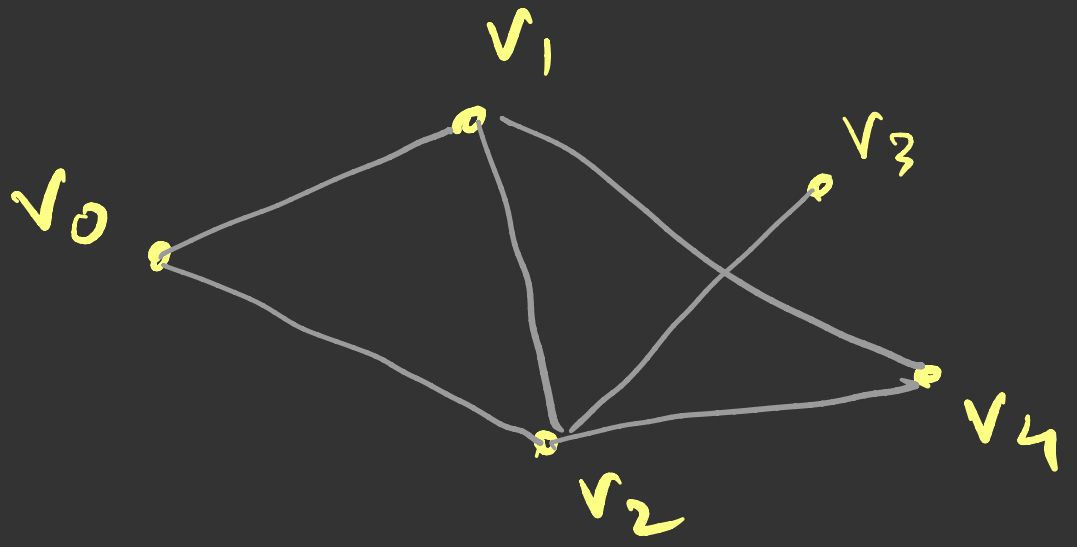
The adjacency list stores information about only those edges that exists.

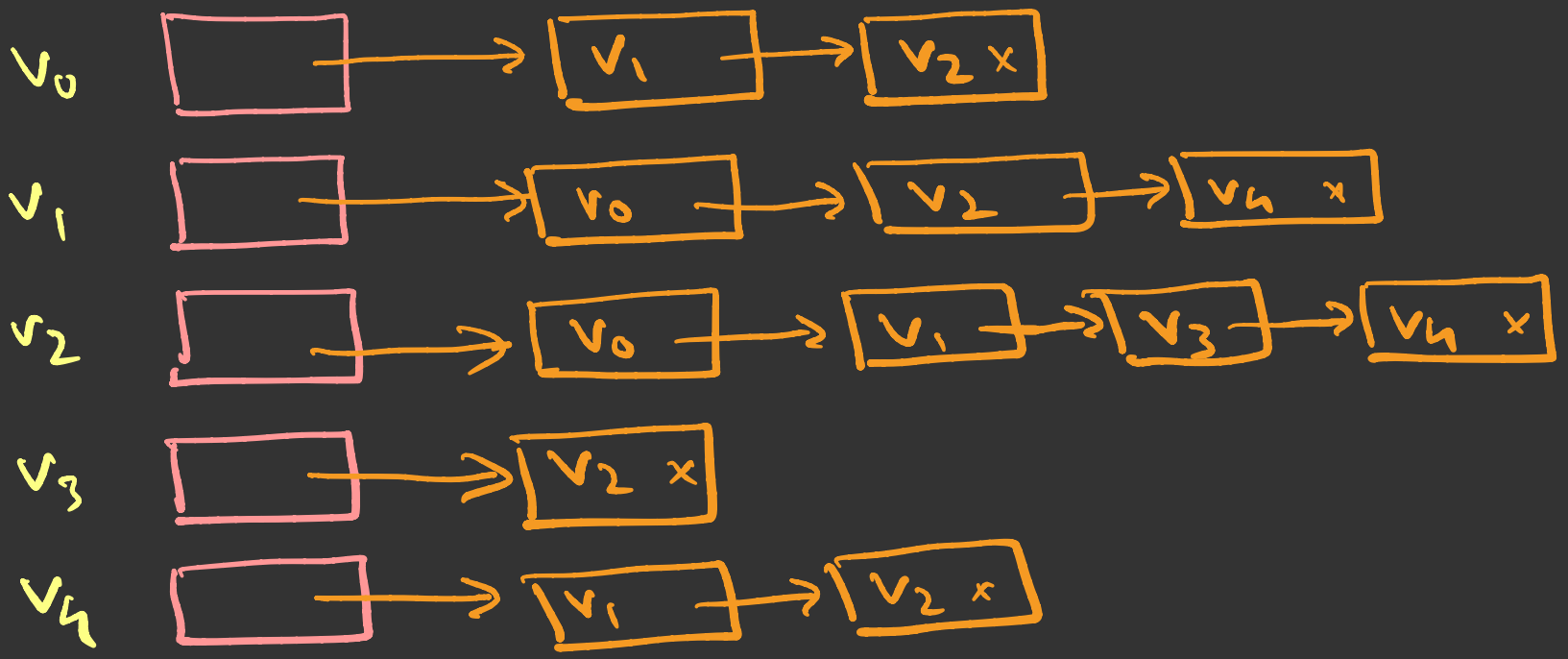
The adjacency list contains a directory and a set of linked lists.

The directory contains one entry for each node of the graph.

Each entry in the directory points to a linked list that represents the nodes that are connected to that node

Each node of the linked list has three fields, one is node identifier, second is the link to the next field and the third is an optional weight field which contains the weights of the edges.


v_1
v_3
v_0
v_4
v_2


v_0
v_1
v_2 x
v_1
v_0
v_2
v_4 x
v_2
v_0
v_1
v_3
v_4 x
v_3
v_2 x
v_4
v_1
v_2 x

```
struct node
{
    int vertex;
    node *next;
};
```

```
class AdjList
{
    private:
        node *start;
};
```

```
class Graph
{
    private:
        int V_count;
        AdjList *arr;
};
```