Graph

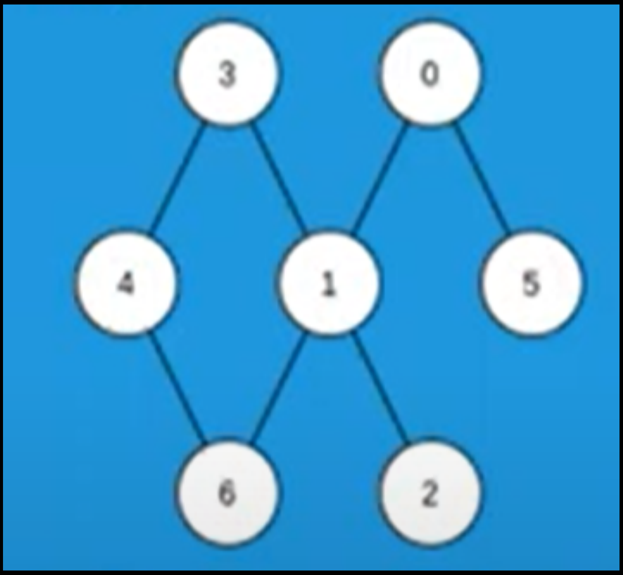
A way of representing relationships between pairs of objects. Visualized as circles connected by lines representng nodes and edges respeectively

Nodes/Vertices (n)

Usually used to represent object in a given proplem

Nodes/Vertices (n)

Usually used to represent relationships between those objects (nodes)



Adjacency relation :

Nodes a,b are called adjacents (Neighbors) , if there is a direct edge between them

Path :

A sequence of nodes connected by edges .

Cycle :

A path whose first and last nodes are the same .

Multi Graph :

A graph having multiple edges between the same pair of nodes .

Self loop :

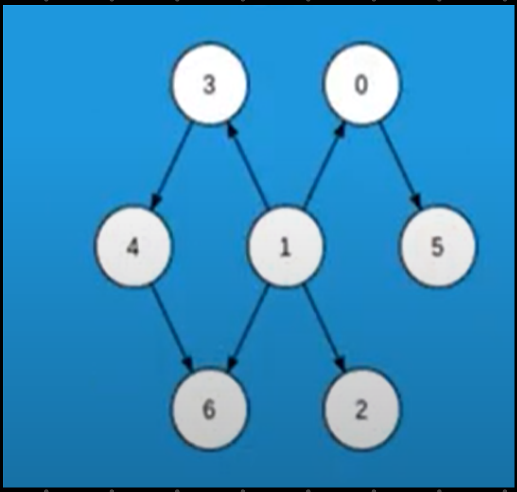
An edge connecting a node to itself .

Simple graph :

A graph that's neither multigraph nor having self-loops

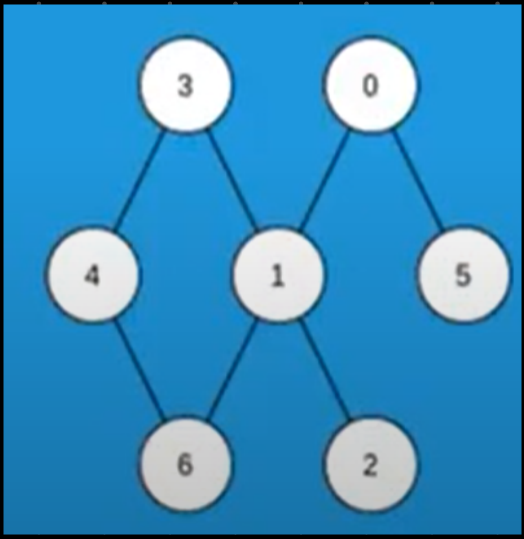
Directed

A graph whose all edges are having directions



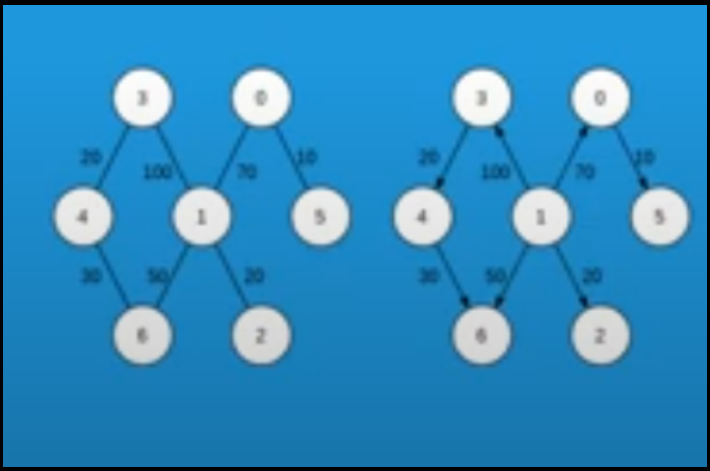
Undirected

A graph whose all edges area NOT having directions



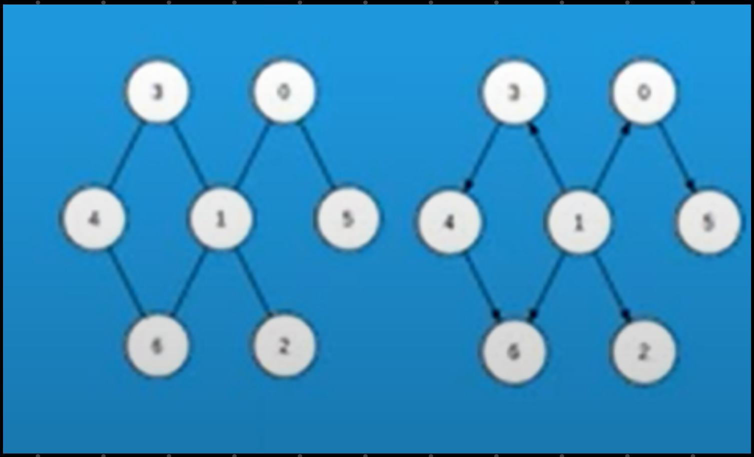
Weighted

A graph having a weight (number) associated with each edge



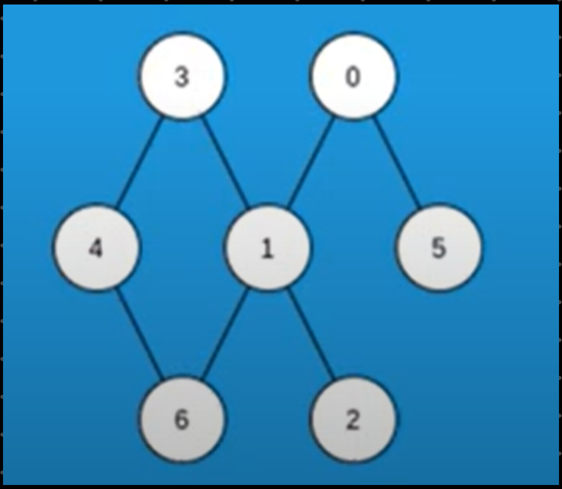
Unweighted

A graph whose all edges are considered equivalent in length (weight)



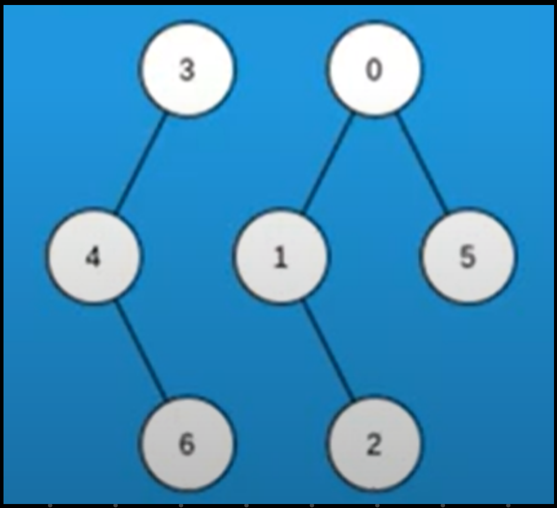
Connected

A graph in which there is a path between every pair of nodes



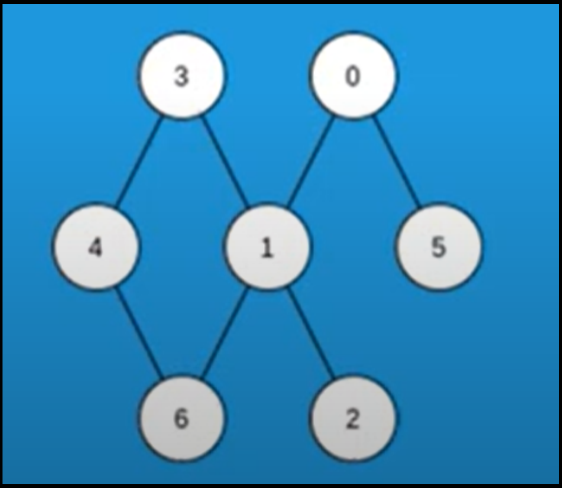
Disconnected

A graph consisted of a set of connected components (subgraphs)



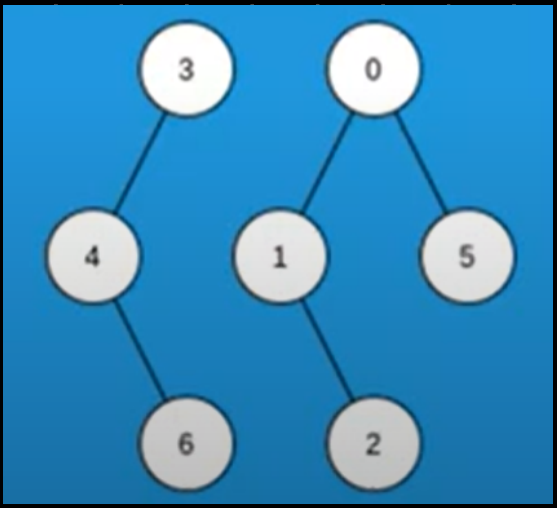
Cyclic

A graph that is / contains a cycle



Acyclic

A graph with no cycles



Explicit

دا الشغل العادي اللي احنا عارفينه

هيكون مديك ال nodes وال edges
واضحين

Implicit

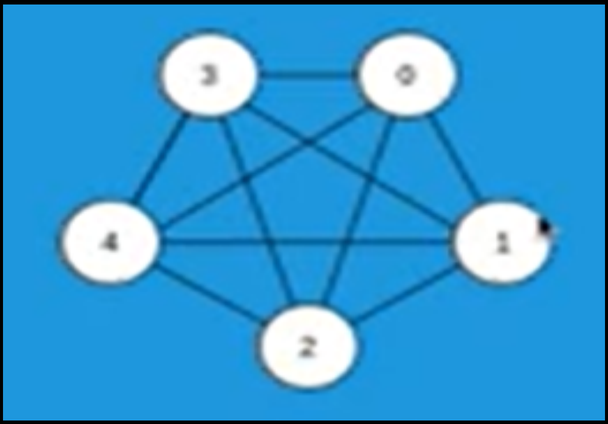
هديك معادله او شرط لو في كذا يبقي

في edge بين ال node دي ودي

A graph whose nodes or edges aren't explicitly represented as objects in computer memory ,but are determined algorithmically in runtime from some I/P .

Complete graph :

A graph which each pair of ndes is directly connected by an edge



$n \rightarrow$ number of nodes (5)

$m \rightarrow$ number of edges (?)

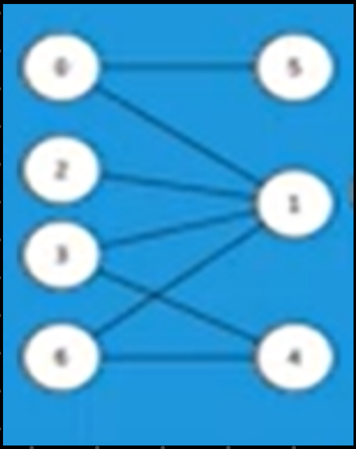
$$m = \frac{n(n-1)}{2}$$

$$m = \underset{n0}{4} + \underset{n1}{3} + \underset{n2}{2} + \underset{n2}{1} + \underset{n3}{0} = 10$$

اكتر جراف فيه عدد edges بالتالي
اكتر جراف بياخد تايم !!!!!!!!!

Bipartite graph :

A graph that can be divided into 2 sets such that there's no edge between nodes within the same set



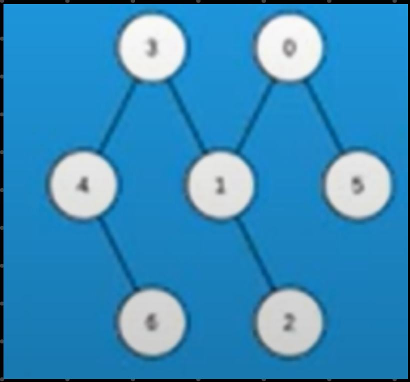
اهم حاجه يبقي مفيش edge بين اي 2nodes في
نفس الجروب

في الرسمه الجروب اللي علي اليمين والي علي الشمال كل جروب
ال nodes اللي جواه مفيش بينهم اي edge

Tree Graph :

** Connected Acyclic graph

** Undirected graph in which there's excatly one unique path between each pair of nodes



الشروط

1. one connected component .
2. Edges = nodes - 1
3. Acyclic .

DAG : Directed Acyclic Graph .

Representation

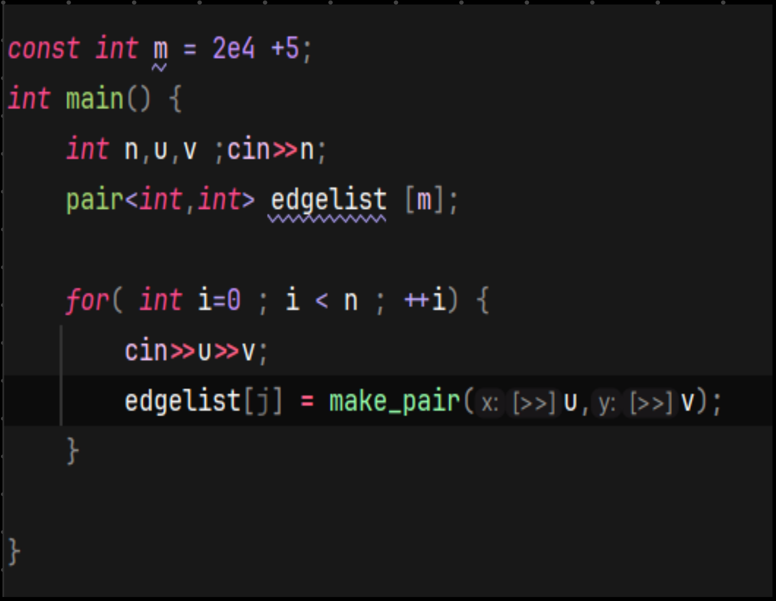
edge list

adj matrix

adj list

Edge List :

memory : O(n)

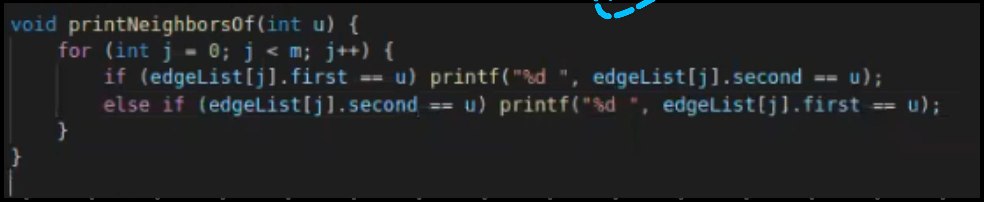


```
const int m = 2e4 +5;
int main() {
    int n,u,v ;cin>>n;
    pair<int,int> edgelist [m];

    for( int i=0 ; i < n ; ++i) {
        cin>>u>>v;
        edgelist[j] = make_pair( x: [>>] u, y: [>>] v);
    }

}
```

* لو عايز اعرف كل

ال Neighbors ل node معينة هتاخد

Time : O(n)

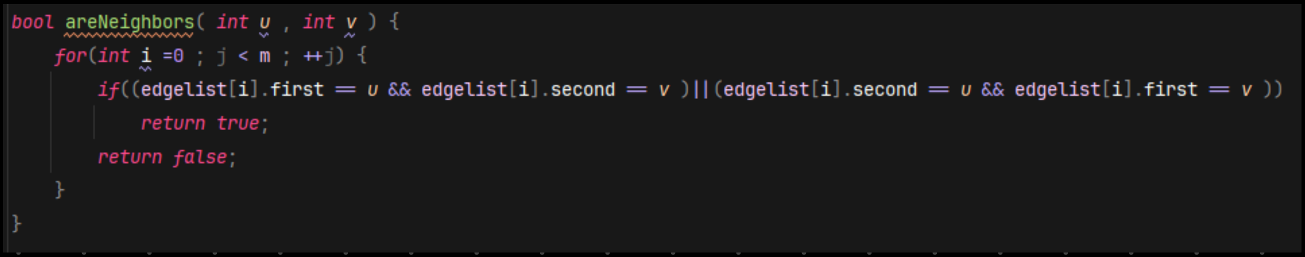


```
void printNeighborsOf(int u) {
    for (int j = 0; j < m; j++) {
        if (edgeList[j].first == u) printf("%d ", edgeList[j].second == u);
        else if (edgeList[j].second == u) printf("%d ", edgeList[j].first == u);
    }
}
```

* لو عايز اعرف هل ال2 nodes اللي

معايا دول neighbors ولا لأ

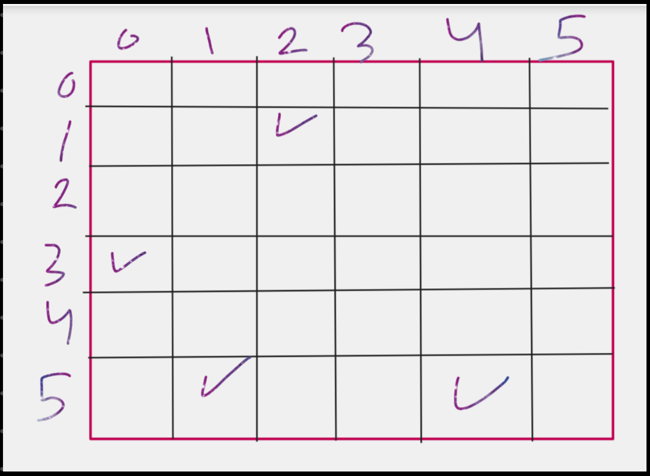
Time : O(n)



```
bool areNeighbors( int u , int v ) {
    for(int i =0 ; j < m ; ++j) {
        if((edgelist[i].first == u && edgelist[i].second == v )||(edgelist[i].second == u && edgelist[i].first == v ))
            return true;
        return false;
    }
}
```

ADJ Matrix :

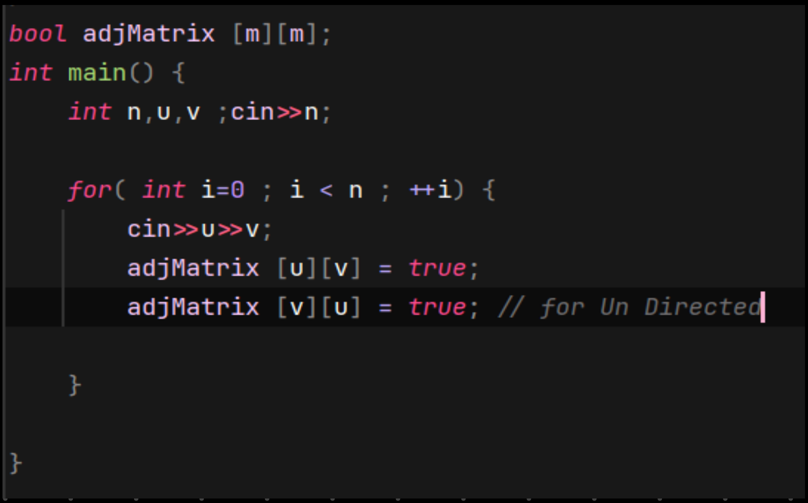
عباره عن bool matrix [n][n]
حيث ان n هي عدد ال nodes كلها في
الاول false كل edge بين 2 nodes
هخليه ب true



	0	1	2	3	4	5
0						
1			✓			
2						
3	✓					
4						
5		✓			✓	

هيظهر معايا مشكله بسيطه فالكود لو الجراف directed او لأ !!!

لو directed هخزن في الماتركس 2,3 لكن لو undirected هخزن في 2,3 وفي 3,2

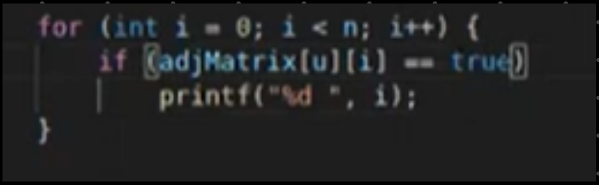


```
bool adjMatrix [m][m];
int main() {
    int n,u,v ;cin>>n;

    for( int i=0 ; i < n ; ++i) {
        cin>>u>>v;
        adjMatrix [u][v] = true;
        adjMatrix [v][u] = true; // for Un Directed

    }

}
```

Time : O(n^2)

Time : O(n)

* لو عايز اعرف كل ال neighbors ل node معينه
ولتكن ال node u



```
for (int i = 0; i < n; i++) {
    if (adjMatrix[u][i] == true)
        printf("%d ", i);
}
```

* لو معايا 2nodes وعايز اعرف هل هما connected ام لا

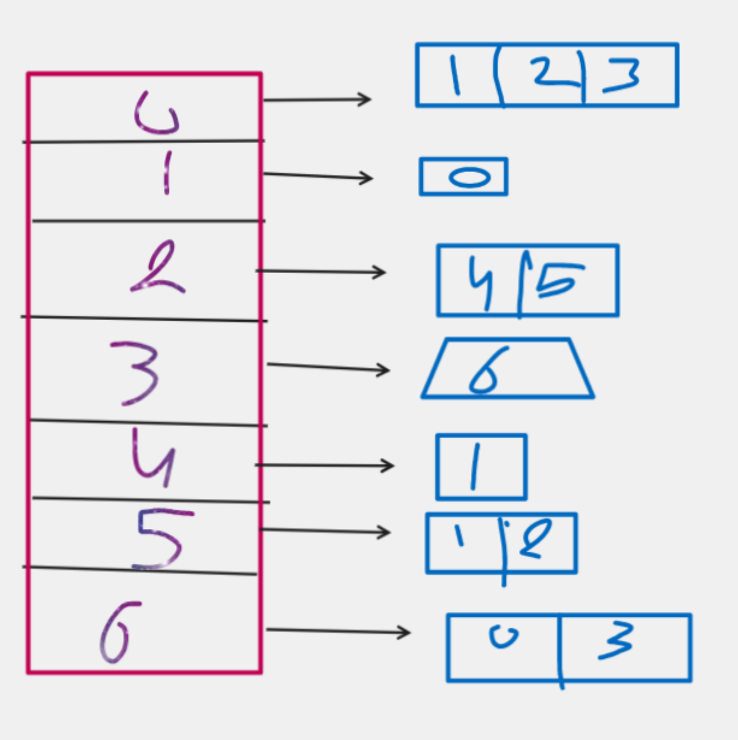
Time : O(1)

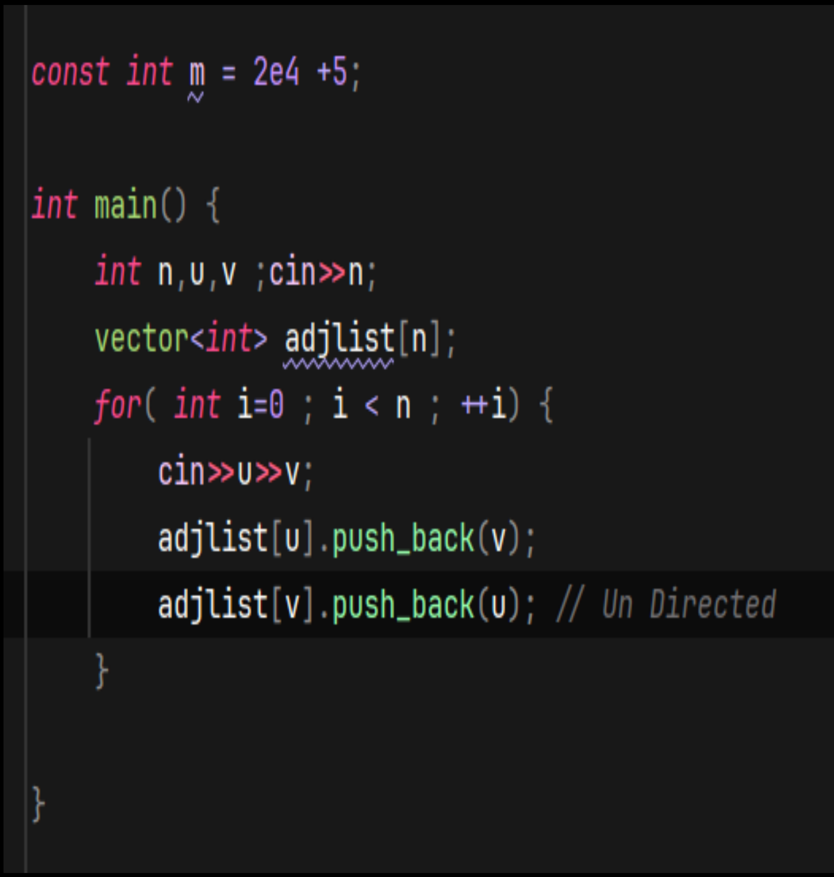
اهم ميزه!!!

ADJ List :

* هما لاحظوا مشكله وحشه اوي في النوع اللي قبله وهي استخدام الميموري الكبير انا مخزن حتي الnodes اللي مفيش بينهم edge وانا مش مستفيد منها بحاجه

انا هعمل list بارقام ال nodes اللي عندي وكل عنصر يمثل dynamic vector بارقام ال nodes اللي متصله بال node دي





```
const int m = 2e4 +5;

int main() {
    int n,u,v ;cin>>n;
    vector<int> adjlist[n];
    for( int i=0 ; i < n ; ++i) {
        cin>>u>>v;
        adjlist[u].push_back(v);
        adjlist[v].push_back(u); // Un Directed
    }

}
```

موجود بردو الحته بتاع ال undirected

لو عايز ال neighbors بتوع اي نود هي الفيكتور بتاع النود دي O(1)

اهم ميزه !!!