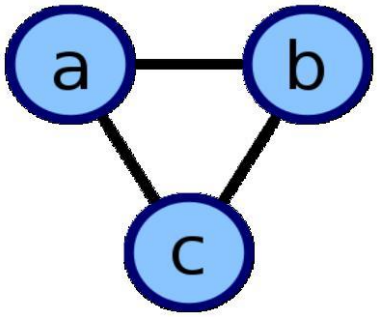
Name : Mufaddal Diwan
Class : SE-3
Roll no : 21340

ASSIGNMENT NO. 4

TITLE	Adjacency list representation of the graph or use adjacency matrix representation of the graph.
PROBLEM STATEMENT /DEFINITION	There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representation used.(Operation to be performed adding and deleting edge, adding and deleting vertices, calculated in-degree and out-degree for both directed and undirected graph)
OBJECTIVE	1. Define a graph (undirected and directed), a vertex/node, and an edge. 2. Given the figure of a graph, give its set of vertices and set of edges. 3. Given the set of vertices and set of edges of a graph, draw a figure to show the graph. 4. Given a graph, show its representations using an adjacency list and adjacency matrix. Also give the space required for each of those representations. 6. Given a DAG, show the steps in topologically sorting the vertices, and give the time complexity of the algorithm.
OUTCOME	The adjacency list easily find all the links that are directly connected to a particular vertex i.e. edge between the cities.
S/W PACKAGES AND HARDWARE	• (64-bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS
APPARATUS USED	• Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++.

Concepts related Theory:

An adjacency list representation for a graph associates each vertex in the graph with the collection of its neighboring vertices or edges. There are many variations of this basic idea, differing in the details of how they implement the association between vertices and collections, in how they implement the collections, in whether they include both vertices and edges or only vertices as first class objects, and in what kinds of objects are used to represent the vertices and edges.



This undirected cyclic graph can be described by the three unordered lists {b, c}, {a, c}, {a, b}

f

The graph pictured above has this adjacency list representation:

a	adjacent to	bc
b	adjacent to	ac
c	adjacent to	ab

OPERATION:

The main operation performed by the adjacency list data structure is to report a list of the neighbors of a given vertex. this can be performed in constant time per neighbor. In other words, the total time to report all of the neighbors of a vertex *v* is proportional to the degree of *v*.

It is also possible, but not as efficient, to use adjacency lists to test whether an edge exists or does not exist between two specified vertices. In an adjacency list in which the neighbors of each vertex are unsorted, testing for the existence of an edge may be performed in time proportional to the minimum degree of the two given vertices, by using a sequential search through the neighbors of this vertex. If the neighbors are represented as a sorted array, binary search may be used instead, taking time proportional to the logarithm of the degree.

Trade-offs:

The main alternative to the adjacency list is the adjacency matrix, a matrix whose rows and columns are indexed by vertices and whose cells contain a Boolean value that indicates whether an edge is present between the vertices corresponding to the row and column of the cell. For a sparse graph (one in which most pairs of vertices are not connected by edges) an adjacency list is significantly more space-efficient than an adjacency matrix (stored as an array): the space usage of the adjacency list is proportional to the number of edges and vertices in the graph, while for an adjacency matrix stored in this way the space is proportional to the square of the number of vertices. However, it is possible to store adjacency matrices more space-efficiently, matching the linear space usage of an adjacency list, by using a hash table indexed by pairs of vertices rather than an array.

The other significant difference between adjacency lists and adjacency matrices is in the efficiency of the operations they perform. In an adjacency list, the neighbors of each vertex may be listed efficiently, in time proportional to the degree of the vertex. In an adjacency matrix, this operation takes time proportional to the number of vertices in the graph, which may be significantly higher than the degree. On the other hand, the adjacency matrix allows testing whether two vertices are adjacent to each other in constant time; the adjacency list is slower to support this operation.

Graph Representation — Adjacency Matrix and Adjacency List:

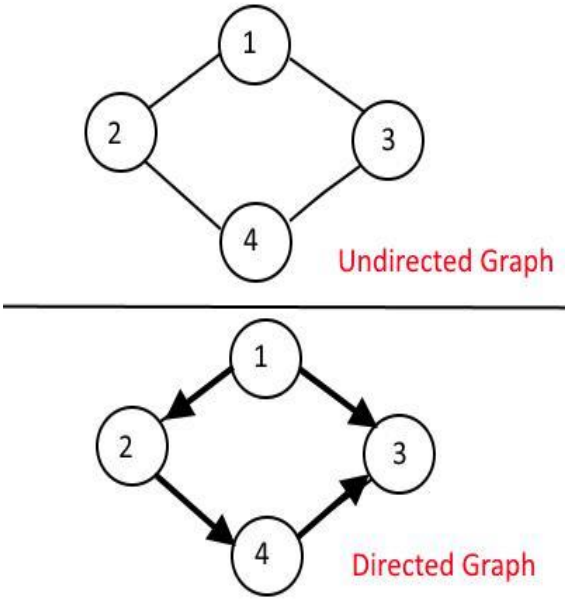
What is Graph ?

G = (V,E)

Graph is a collection of nodes or vertices (V) and edges(E) between them. We can traverse these nodes using the edges. These edges might be weighted or non-weighted.

There can be two kinds of Graphs

- Un-directed Graph — when you can traverse either direction between two nodes.
- Directed Graph — when you can traverse only in the specified direction between two nodes.



Now how do we represent a Graph, There are two common ways to represent it:

- Adjacency Matrix
- Adjacency List

Adjacency Matrix:

Adjacency Matrix is 2-Dimensional Array which has the size VxV, where V are the number of vertices in the graph. See the example below, the Adjacency matrix for the graph shown above.

adjMaxtrix[i][j] = 1 when there is edge between Vertex i and Vertex j, else 0.

	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

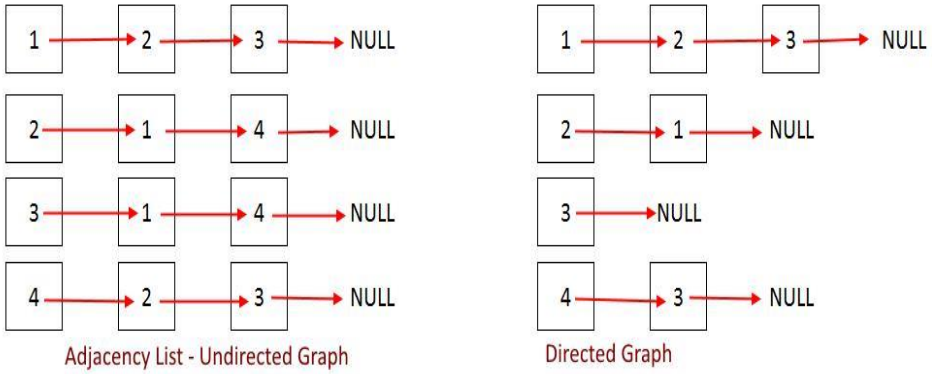
	1	2	3	4
1	0	1	1	0
2	0	0	0	1
3	0	0	0	0
4	0	0	1	0

It’s easy to implement because removing and adding an edge takes only O(1) time.

But the drawback is that it takes $O(V^2)$ space even though there are very less edges in the graph.

Adjacency List:

Adjacency List is the Array[] of Linked List, where array size is same as number of Vertices in the graph. Every Vertex has a Linked List. Each Node in this Linked list represents the reference to the other vertices which share an edge with the current vertex. The weights can also be stored in the Linked List Node.



Algorithm:

```
class Edge

{
private:
Vertex *source;
Vertex *destination;
int distance;
public:
Edge(Vertex *s, Vertex *d, int dist)
{
source = s;
destination = d;
distance = dist;
}

Vertex *getSource()
{
return source;
}
Vertex * getDestination()
{
```

```
return destination;
}
int getDistance()
{
return distance;
}
};
class Vertex
{
private : string city;
vector<Edge> edges;//vector edges created for edges
public:
Vertex(string name)

{
city = name;
}
void addEdge(Vertex *v, int dist)
{
Edge newEdge(this,v,dist);// creating object for edge
edges.push_back(newEdge);//creating  adjusting  list
}
void showEdge()
{
cout<<"From"<<city<<"to"<<endl;
for(int i=0; i<(int)edges.size();i++)
{
Edge
e = edges[i];
cout<<e.getDestination()
-
>getCity()<<"requires"<<e.getDistance()<<"hrs"<<endl;
}
cout<<endl;
}
string getCity()
{
return city;
}
vector<Edge> getEdges()
{
return edges;
}
};
class Graph
{
vector<Vertex*> v;
```

```
public:
Graph(){}
void insert(Vertex *val)
{
v.push_back(val);
}
void Display()
{
for(int i=0;i<(int)v.size();i++)
{
// v[i].showEdge();
v[i]
-
>showEdge();
}
}
};
int main()
{
Graph g;
// crea
ting verticex ot nodes for each city
Vertex v1 = Vertex("Mumbai");
Vertex v2 = Vertex("Pune");
Vertex v3 = Vertex("Kolkata");
Vertex v4 = Vertex("Delhi");
//creating pointers to nodes
Vertex *vptr1 = &v1;
Vertex *vptr2 = &v2;
Vertex *vptr3 = &v3;
Vertex *vptr4 = &v4;
//attaching the nodes by adding edges
v1.addEdge(vptr4,2);
v2.addEdge(vptr1,1);
v3.addEdge(vptr1,3);
v4.addEdge(vptr2,2);
v4.addEdge(vptr3,3);
//cretaing graph
g.insert(vptr1);
g.insert(vpt
r2);
g.insert(vptr3);
g.insert(vptr4);
cout<<"
\
n
```

```
\
t Displaying City Transport Map Using Adjacency List"<<endl;
g.Display();
return 1;
}
```

TEST CASES:

Check The is there adjacency list find all the edges that are directly connected to a cities or not.

Test-Cases

Desciption	Input	Output	Result
Create Airport	Cities: 5 City 1: A City 2: B City 3: C City 4: D City 5: E Src Dest Dist 1 2 1 1 4 2 2 1 1 2 3 2 3 2 2 3 4 1 4 1 2 4 3 1	-	Pass
Display Graph	-	Source Destination Duration A B 1 D 2 B A 1 C 2 C B 2 D 1 D A 2 C 1	Pass
Indegree Of a vertex	Enter index: 1	2	Pass
Outdegree Of a vertex	Enter index : 4	2	Pass

Conclusion:

Thus we have studies adjacency list representation of the graph successfully for cities.