

Graphs.

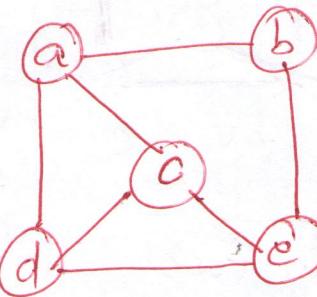
Graphs → Definitions
 → Examples
 → The Graph ADT

What is a Graph?

A graph $G = (V, E)$ is composed of:

V : set of vertices

E : set of edges connecting the vertices in V .



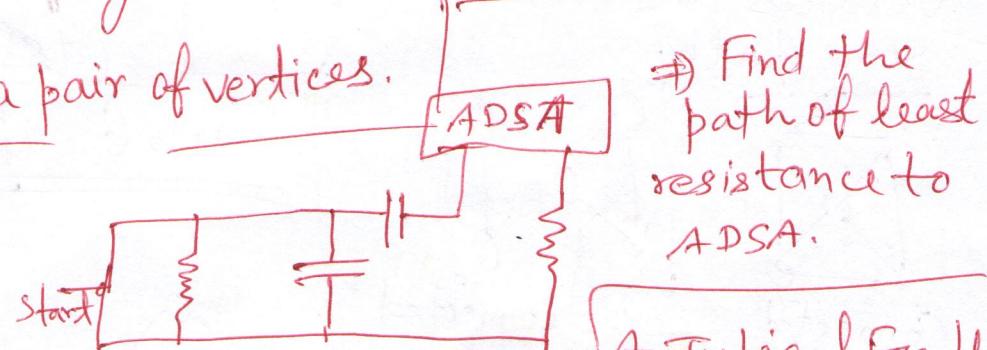
$$V = \{a, b, c, d, e\}$$

$$E = \{(a,b), (a,d), (a,c), (b,e), (c,d), (c,e), (d,e)\}$$

An edge $e = (u,v)$ is a pair of vertices.

Applications

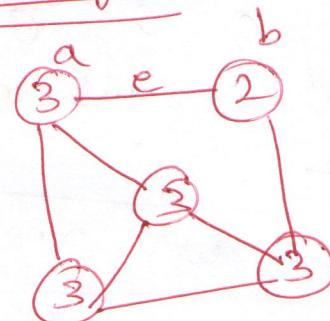
Electronic circuits



Networks (roads, flights)

Communication networks

Graph Terminology.



Adjacent vertices: vertices connected by an edge.

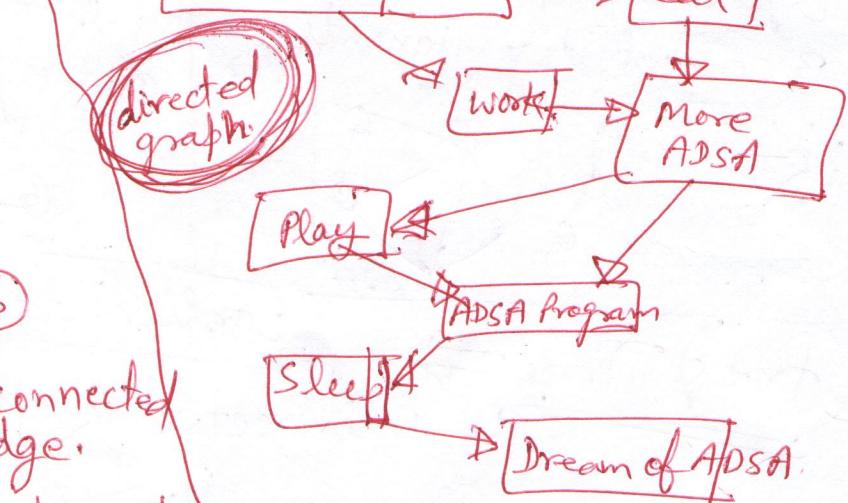
Degree (of a vertex): # of adjacent vertices.

→ $e = (a,b)$ is incident to vertices a & b .

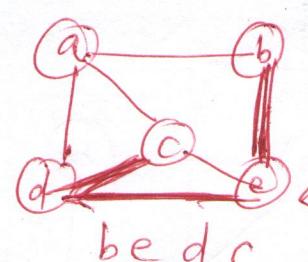
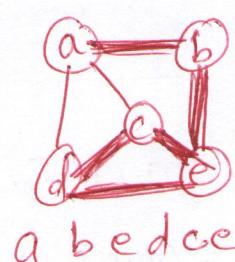
What is the sum of the degrees of all vertices?

→ Twice the number of edges.

→ since adjacent vertices each count the adjoining edge, it will be counted twice.



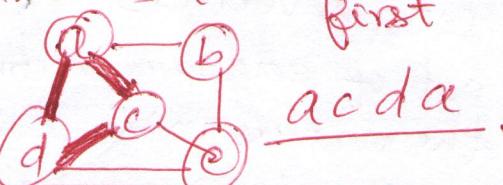
② Path: sequence of vertices v_1, v_2, \dots, v_k such that consecutive vertices v_i and v_{i+1} are adjacent.



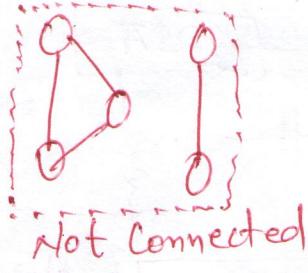
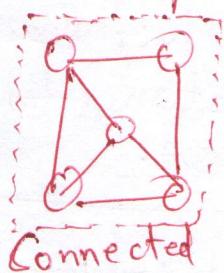
Simple Path:

No repeated vertices

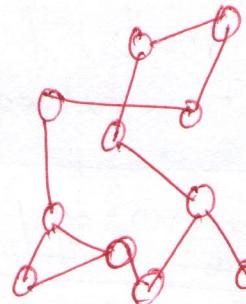
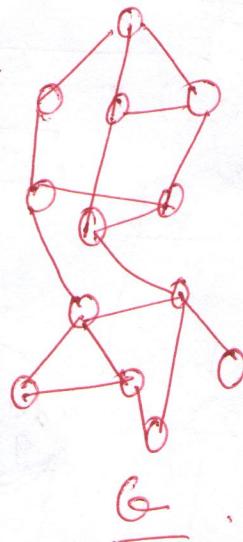
Cycle Path: simple path, except that the last vertex is the same as the first vertex.



Connected Graph: any two vertices are connected by some path.



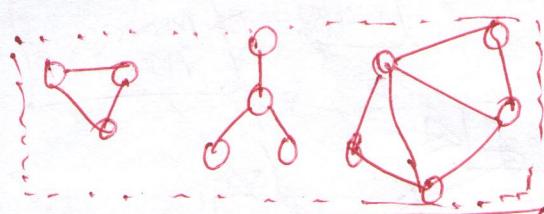
Sub Graph: subset of vertices and edges forming a graph.



Subgraph of G.

Connected Component \Rightarrow

maximal connected subgraph,
E.g. the graph below has 3 connected components.



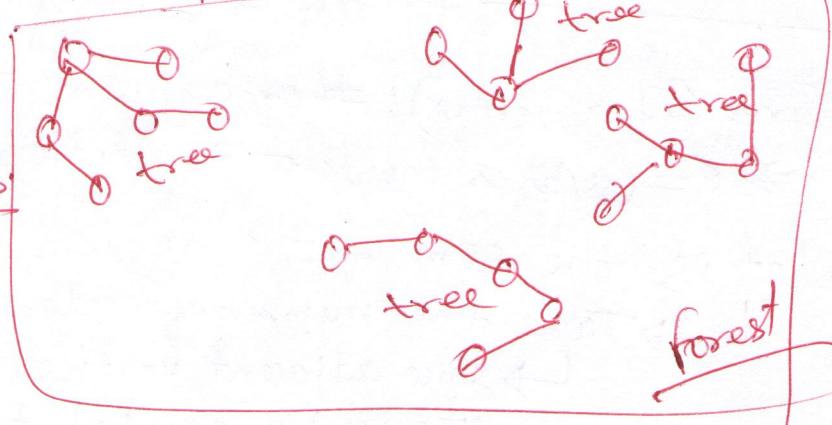
(Earlier tree was rooted tree)

(free) Tree \Rightarrow

Connected Graph without cycles.

#Forest

\Leftrightarrow collection of trees.

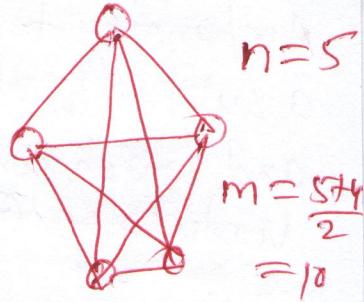


Connectivity

Let $n = \# \text{vertices}$, and $m = \# \text{edges}$

Complete Graph: one in which all pairs of vertices are adjacent

(i.e., every pair of vertices has an edge).



How many edges does a complete graph have?

There are $n(n-1)/2$ pairs of vertices and so $m = n(n-1)/2$.

For a tree $m = n-1$, where $n = \# \text{vertices}$ and $m = \# \text{edges}$.
 ⇒ A tree on n vertices has $n-1$ edges.

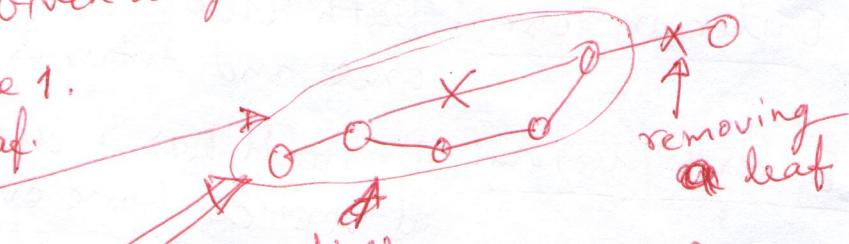
Proof by induction ⇒

$$\text{Base Case: } n=2 \quad \text{---} \quad \text{no. of edges} = n-1 = 1$$

Induction Hypothesis: statement is true for all $n \leq k$

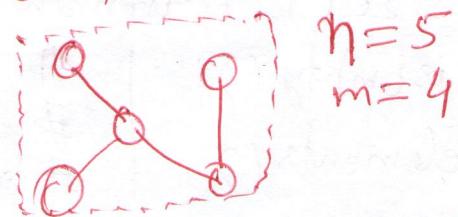
Induction Step: Given a graph(tree) on $k+1$ vertices

A leaf is a vertex of degree 1.
 Every tree has at least 1 leaf.



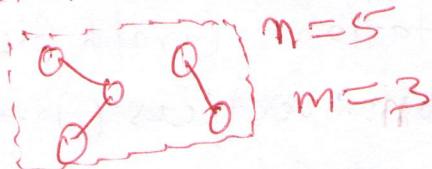
If $m < n-1$, then G_1 is not connected.

edges # Elements (vertices)



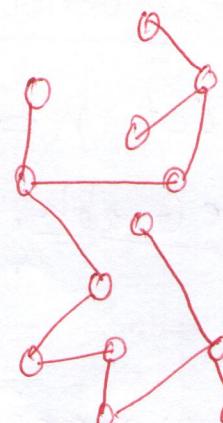
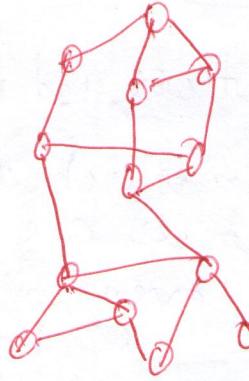
If a graph has $n-k$ number of edges

then it can have $\frac{k}{\text{without cycle}}$ or $\frac{\text{more number}}{\text{with cycle}}$ of connected components.



④ Spanning Tree

A spanning tree of G_1 is a subgraph which is a tree and which contains all vertices of G_1 .



failure on any edge leads to disconnects system.

↳ least fault tolerance.

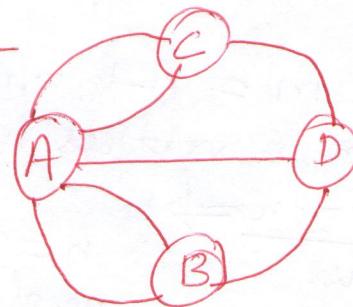
Graph Model (with parallel edges)

Suppose you are a postman, and you do not want to retrace your steps.

In 1736, Euler proved that this is not possible.

Eulerian Tour : path that traverses every edge exactly once and returns to the first vertex.

Euler's Theorem: A graph has a Eulerian tour if and only if all vertices have even degree.



The Graph ADT \Rightarrow The graph ADT is a positional container whose positions are the vertices and the edges of the graph.

size() Return the number of vertices + number of edges of G_i .

isEmpty() | # positions()

elements() | # swap()

| # replaceElement()

Notation: Graph G_i ; Vertices V, W; Edge e; Object o

\Rightarrow numVertices() — Return the number of vertices of G_i .

\Rightarrow numEdges() — Return the number of edges of G_i .

\Rightarrow vertices() — Return an enumeration of the vertices of G_i . \rightarrow To iterate across all vertices

\Rightarrow edges() — Return an enumeration of the edges of G_i .

- # directed Edges () - enumeration of all directed edges in G.
- # undirected Edges () - enumeration of all undirected edges in G.
- # incident Edges(v) - enumeration of all edges incident on v.
- # inIncident Edges(v) - enumeration of all edges entering v.
- # outIncident Edges(v) - enumeration of all edges leaving v.
- # Opposite(v, e) - an end point of e ~~distinct~~ distinct from v.
- # degree(v) - the degree of v.
- # inDegree(v) - the in-degree of v.
- # outDegree(v) - the out-degree of v.
- # adjacentVertices(v) - enumeration of vertices adjacent to v.
- # inAdjacentVertices(v) - enumeration of vertices adjacent to v along incoming edges.
- # outAdjacentVertices(v) - enumeration of vertices adjacent to v along outgoing edges.
- # areAdjacent(v, w) - whether vertices v and w are adjacent.
- # endVertices(e) - the end vertices of e.
- # origin(e) - the end vertex from which e leaves.
- # destination(e) - the end vertex at which e arrives.
- # isDirected(e) - true iff e is directed.] In mixed graph

Update Methods →

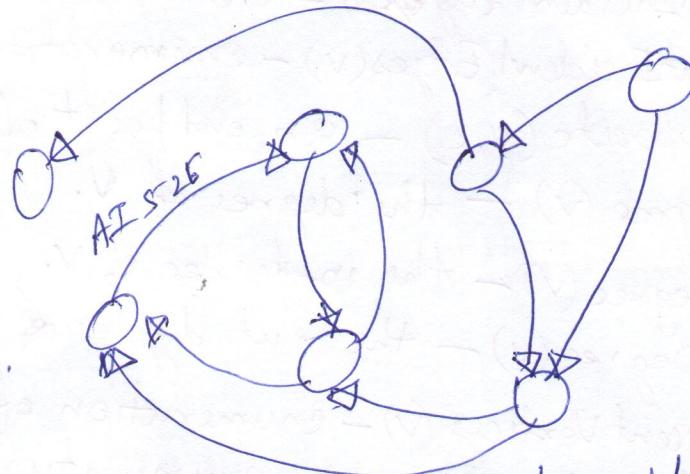
- # makeUndirected(e) - Set e to be an undirected edge.
- # reverseDirection(e) - Switch the origin and destination vertices of e.
- # setDirectionFrom(~~e~~(e, v) - sets the direction of e away from v, one of its end vertices.
- # setDirectionTo(e, v) - sets the direction of e toward v, one of its end vertices.
- # insertEdge(v, w, o) - Insert and return an undirected edge between v & w, storing o at this position.
- # insertDirectedEdge(v, w, o) - Insert and return a directed edge between v & w, storing o at this position.
- # insertVertex(o) - Insert and return a new (isolated) vertex storing o at this position.

⑥ Data Structures for Graphs

Edge List Adjacency list Adjacency matrix.

Edge List \Rightarrow

We store the vertices and the edges into two containers, and each ~~edge~~ edge object has references to the vertices it connects.



Flights b/w cities.

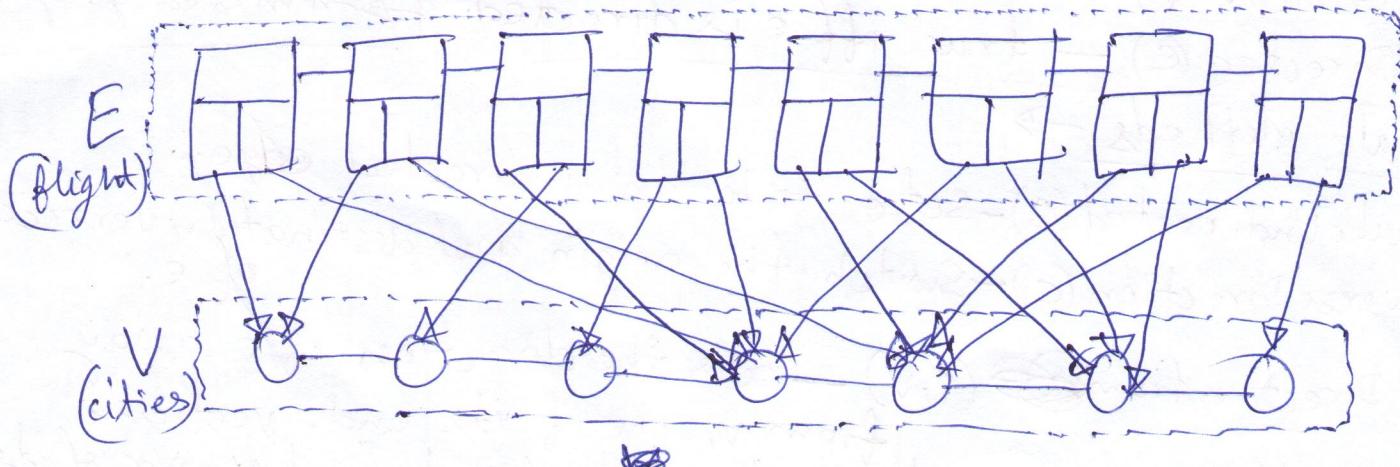
Vertex \rightarrow City
Edge \rightarrow Flight

\Rightarrow The edge list structure

simply stores the vertices and the edges in two unsorted sequences.

\Rightarrow Easy to implement.

\Rightarrow Finding the edges incident on a given vertex is inefficient since it requires examining the entire edge sequence.



finding the adjacent vertices is costly.

Performance

size, isEmpty, replaceElement, swap $\xrightarrow{\text{numVertices}} O(1)$
 $\xrightarrow{\text{numEdges}}$

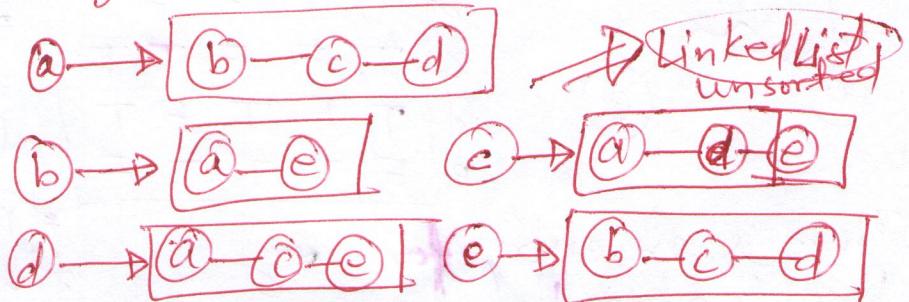
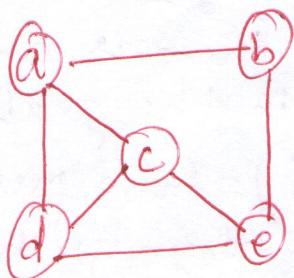
vertices $\rightarrow O(n)$, edges, directEdges, undirectEdges $\rightarrow O(m)$
elements, positions $\rightarrow O(ntm)$, endVertices, opposite, origin, destination, isDirected $\rightarrow O(1)$

incidentEdges, adjacentVertices, areAdjacent, degree $\rightarrow O(m)$

insertVertex, insertEdge, removeEdge, reverseDirection, setDirectionFrom, removeVertex $\rightarrow O(m)$
setDirectionTo $\rightarrow O(1)$

Adjacency List (traditional)

- # adjacency list of a vertex V ; sequence of vertices adjacent to V .
- # Represent the graph by the adjacency lists of all the vertices.

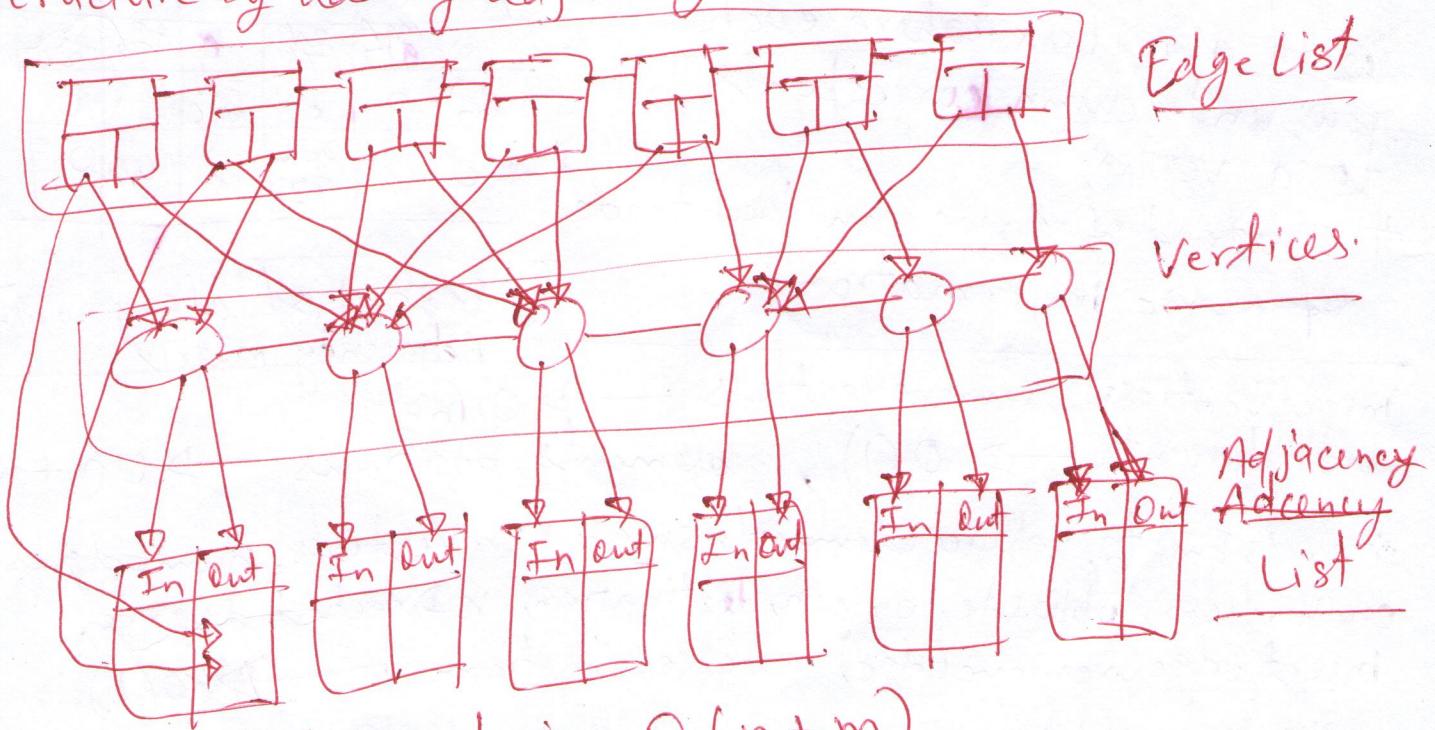


$$\boxed{\text{Space} = O(N + \sum \deg(v)) = O(N + M)}$$

↓ array of size N . → $2M$

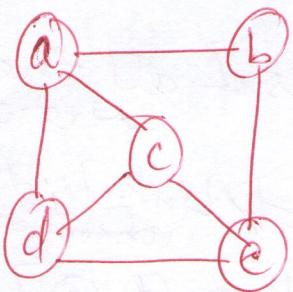
Adjacency List (modern)

- # The adjacency list structure extends the edge list structure by adding adjacency lists to each vertex.



- # The space requirement is $O(n+m)$
- size, isEmpty, replaceElement, swap, numVertices, numEdges, endVertices, opposite, origin, destination, isDirected, degree, inDegree, outDegree, insertVertex, insertEdge, removeEdge, reverseDirection → $O(1)$
- vertices → $O(n)$, edges, directed edges, undirected edges → $O(m)$
- elements, positions → $O(n+m)$ [removeVertices(v) → $O(\deg(v))$]
- incidentEdges(v), adjacentVertices(v), areAdjacent(u, v) → $O(\min(\deg(u), \deg(v)))$

④ Adjacency matrix (traditional)



| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | F | T | T | T | F |
| b | T | F | F | F | T |
| c | T | F | F | T | T |
| d | T | F | T | F | T |
| e | F | T | T | T | F |

Matrix M with entries for all pairs of vertices

$M[i, j] = \text{true}$ means that there is an edge (i, j) in the graph.

There is an entry for every possible edge, therefore:

$$\text{Space} = \Theta(N^2)$$

$M[i, j] = \text{false}$ means that there is no edge (i, j) in the graph.

Adjacency Matrix (modern).

The adjacency matrix structures augments the edge list structure with a matrix where each row and column corresponds to a vertex.

Instead of q , we can keep track of edge information.

0 1 2 3 4 5

| | | | | | | |
|---|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | DTW | 0 | MAA | 0 |
| 1 | 0 | 0 | 0 | SRD | SRI | 0 |
| 2 | CXG | 0 | 0 | UZB | 14 | DEL |
| 3 | 259 | 0 | 0 | 0 | AIS | 25 |
| 4 | 0 | MAA | IN | 0 | AZB | DEL |
| 5 | SPD | 0 | 76 | 0 | 26 | 0 |
| 6 | 987 | 0 | 0 | 970 | 0 | 0 |
| 7 | 0 | 0 | DEL | 0 | 0 | 0 |

Edge list along with Adjacency matrix

insertVertices, removeVertices $\rightarrow O(n^2)$

areAdjacent $\rightarrow O(1)$, elements, positions $\rightarrow O(n+m)$

size, isEmpty, replaceElement, swap, numVertices, numEdges, endVertices, opposite, origin, destination, isDirected, degree, insertEdge, removeEdge, reverseDirection $\rightarrow O(1)$.

vertices $\rightarrow O(n)$, edges, directedEdges, undirectedEdges $\rightarrow O(m)$

incidentEdges, inIncidentEdges, outIncidentEdges, adjacentVertices, inAdjacentVertices, outAdjacentVertices $\rightarrow O(n)$.