Graphs

CSC-114 Data Structure and Algorithms

slides credit: Ms. Saba Anwar, CIIT Lahore

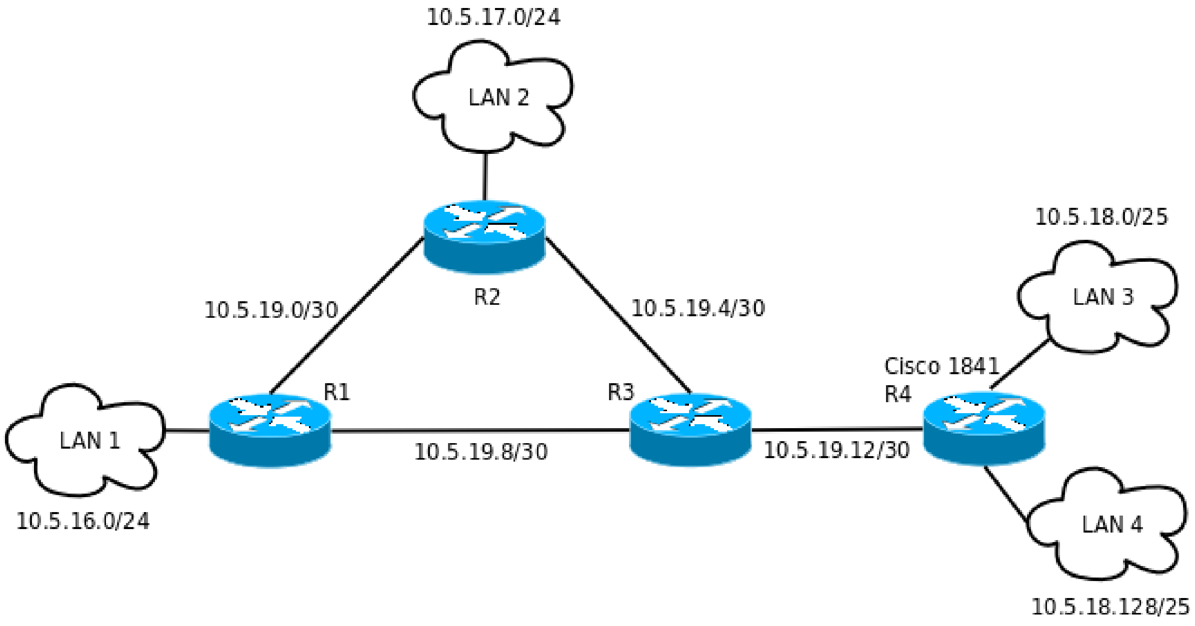
Outline

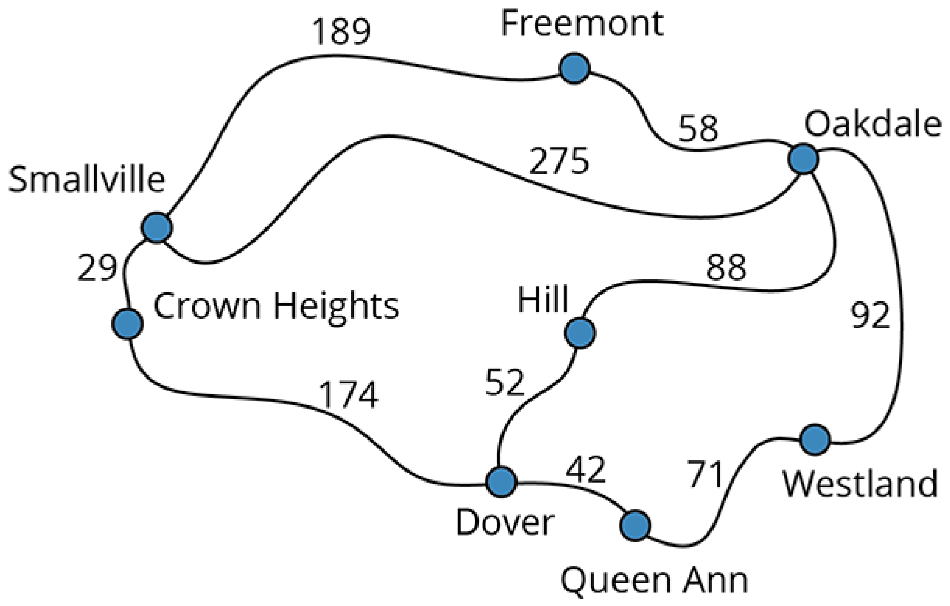
- Non-Linear Data Structures
  - Graphs
    - Intro
    - Application
    - Terminologies
    - Representation
      - Adjacency List
      - Adjacency Matrix

Graph

A graph is a way of representing relationships between pairs of objects.

It is a set of objects, called vertices, together with a collection of pairwise connections between them, called edges





Graph

Graph is a mathematical structure that is defined as G= (v, e), where v is a set of vertices{v_1, v_2, …v_n} and e is a set of edges {e_1,e_2,e_3,…e_m}

Where edge e is an ordered pair of two vertices, represents a connection between two vertices

Graph can be directed, or undirected

Example:

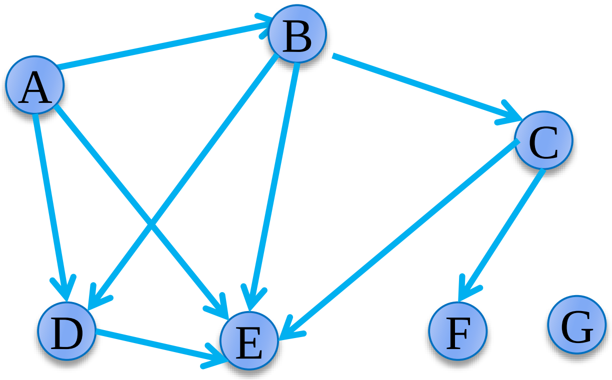
V = {A, B, C, D, E, F, G}

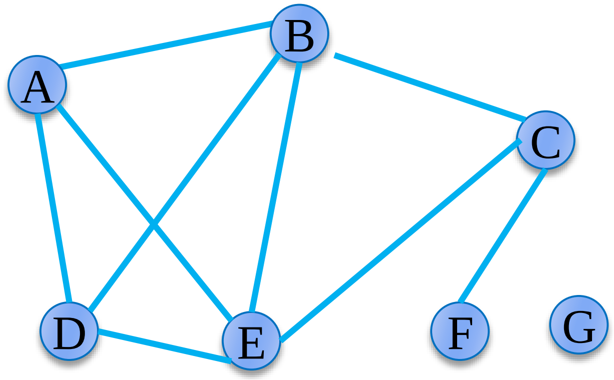
E = { {A, B}, {A, D}, {A, E}, {B, C}, {B, D}, {B, E}, {C, E}, {C, F}, {D, E} }

Then the graph G=(V,E) is:

|V|= 7

|E|=9

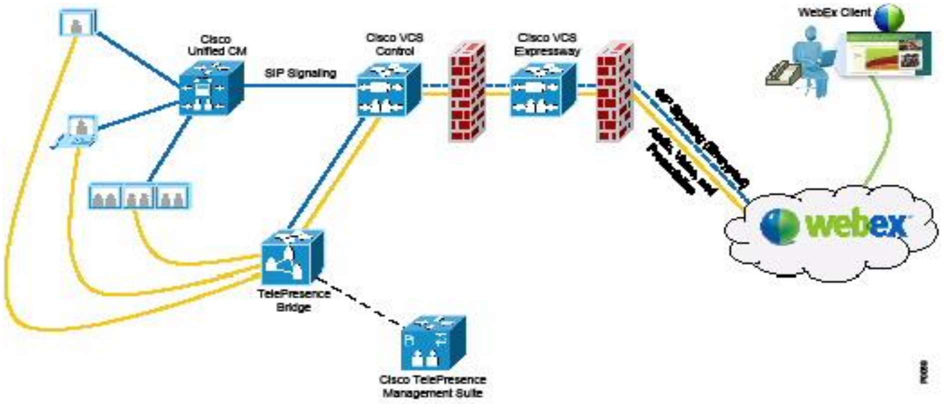




Applications

Maps

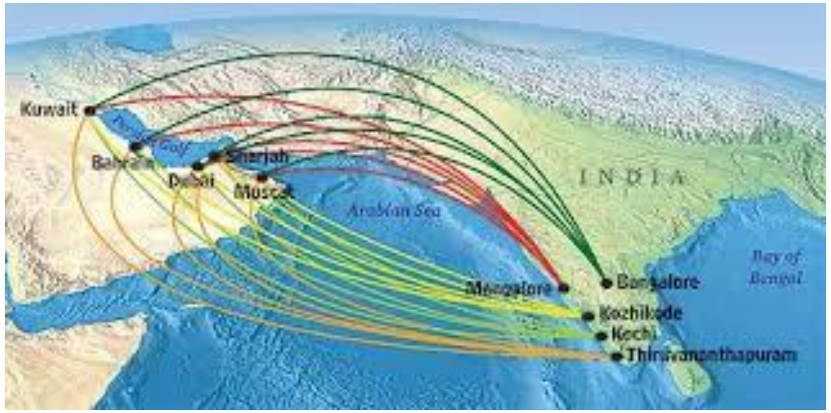
Communication Networks



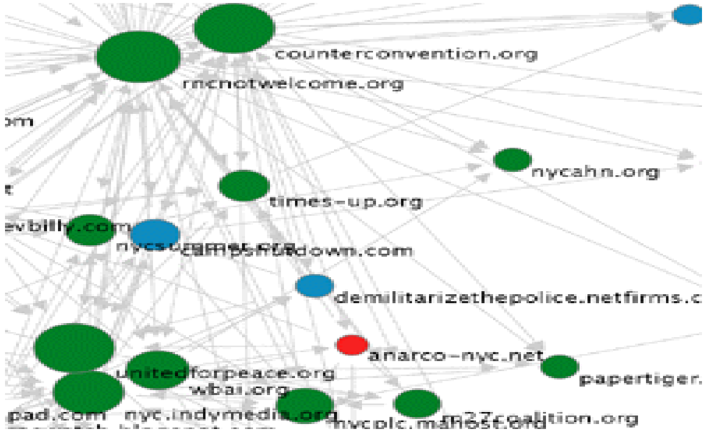
Social networks

Applications

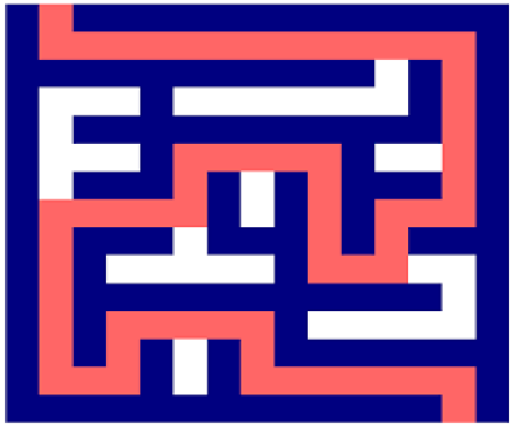
Flight routes



Web graphs



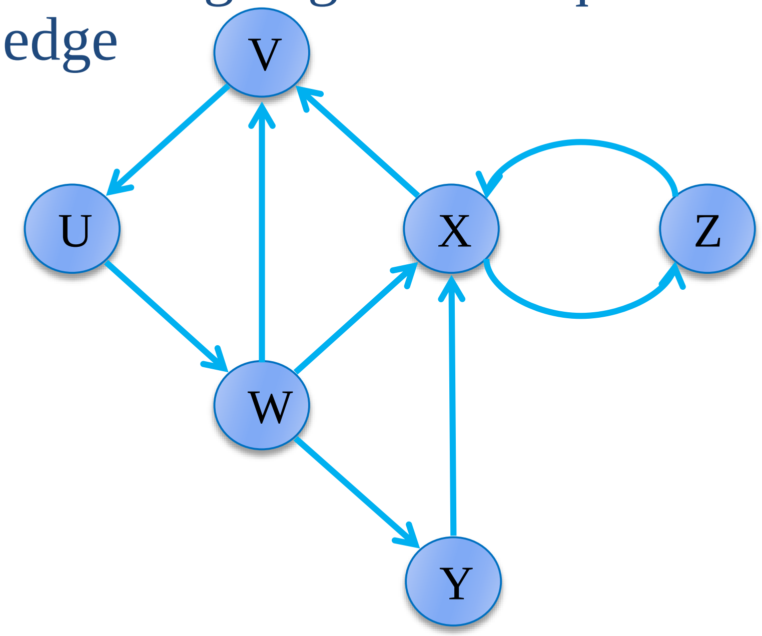
Maze (games)



Terminologies

Directed Graph

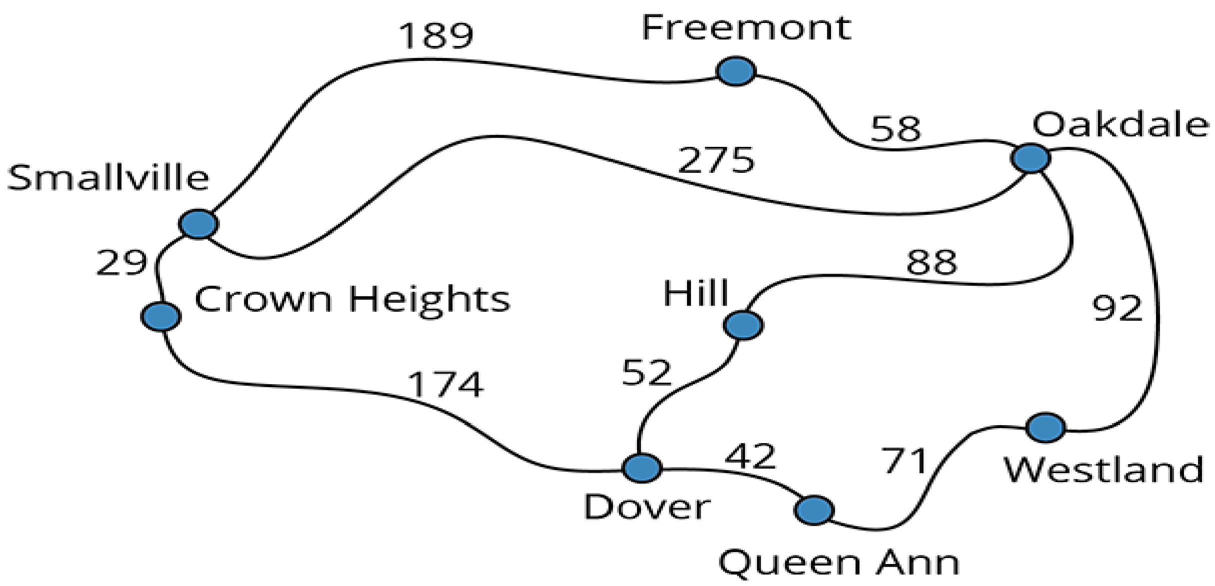
- Also called digraph
- Every edge has a direction, an incoming edge is not equal to outgoing edge



Vertex order is important

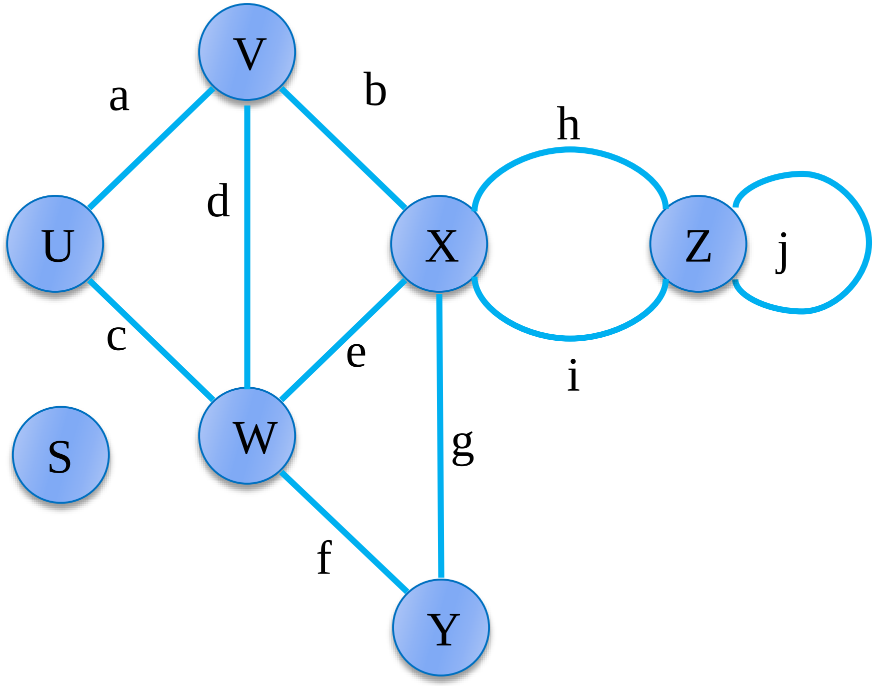
Un-Directed Graph

- Edge has no direction and considered as two-ways
  - Vertex order is not important



Terminologies

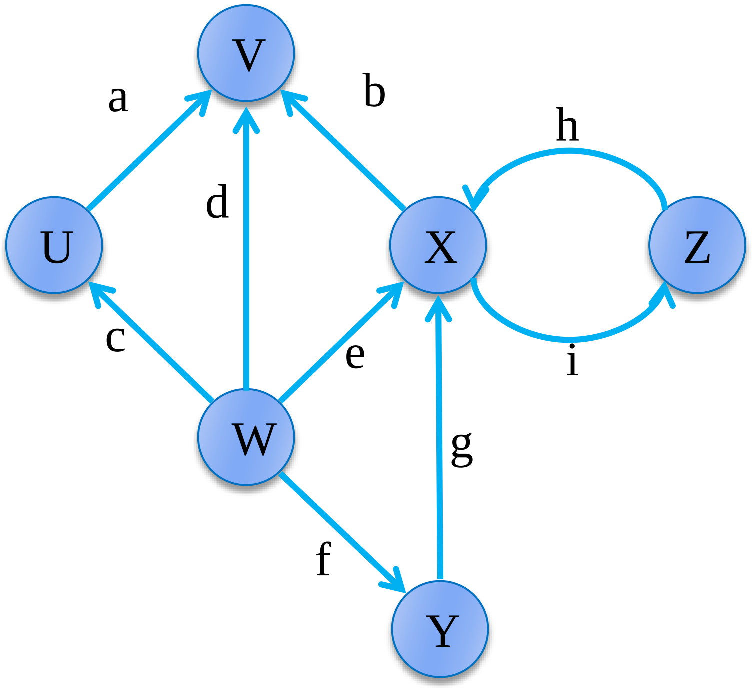
- End Points
  - Vertices at both ends of an edge
- Incident Edge
  - If vertex is an end point of edge, edge is incident on that vertex
    - a is incident on u and v
- Adjacent/Neighbor Vertices
  - Vertices that are end points of same edge
    - u, v and w are adjacent
- Self Loop
  - Node connected to itself
    - Z is in self loop and j is self edge
- Degree of Node
  - Number of incident edges
    - U has degree 2, X has degree 5
    - Self edge is considered twice, so z has degree 4
- Parallel Edges
  - Edges with same end points
    - h and i are parallel edges



Terminologies

In a directed graph we can distinguish between

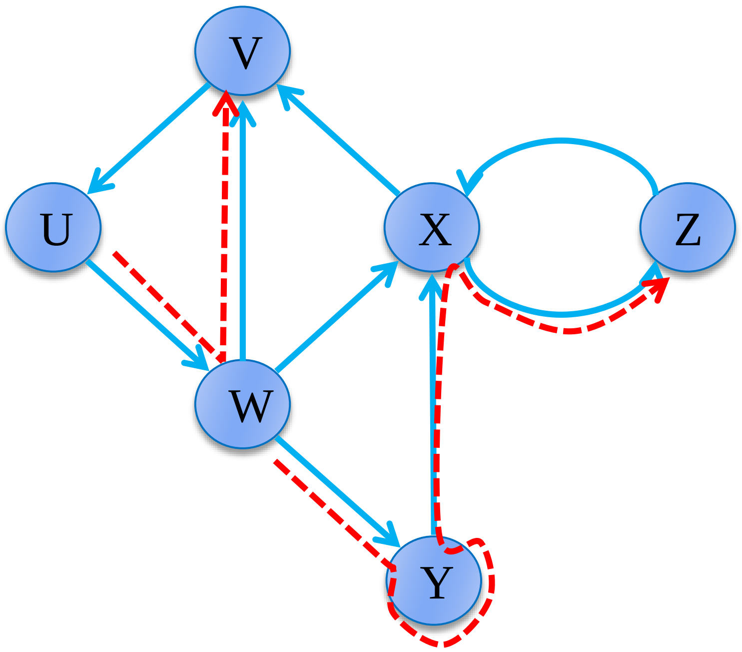
- Incoming Edges
  - Directed edges for which the given vertex is destination
    - c is incoming edge of U
- Outgoing Edges
  - Directed edges for which the given vertex is origin
  - a is outgoing edge of U
- In-Degree of Vertex
  - Number of incoming edges
- Out-Degree of Vertex
  - Number of outgoing edges
- Source Vertices
  - Vertices with an in-degree of zero
    - w is source vertex
- Sink Vertices
  - Vertices with an out-degree of zero
    - v is sink vertex

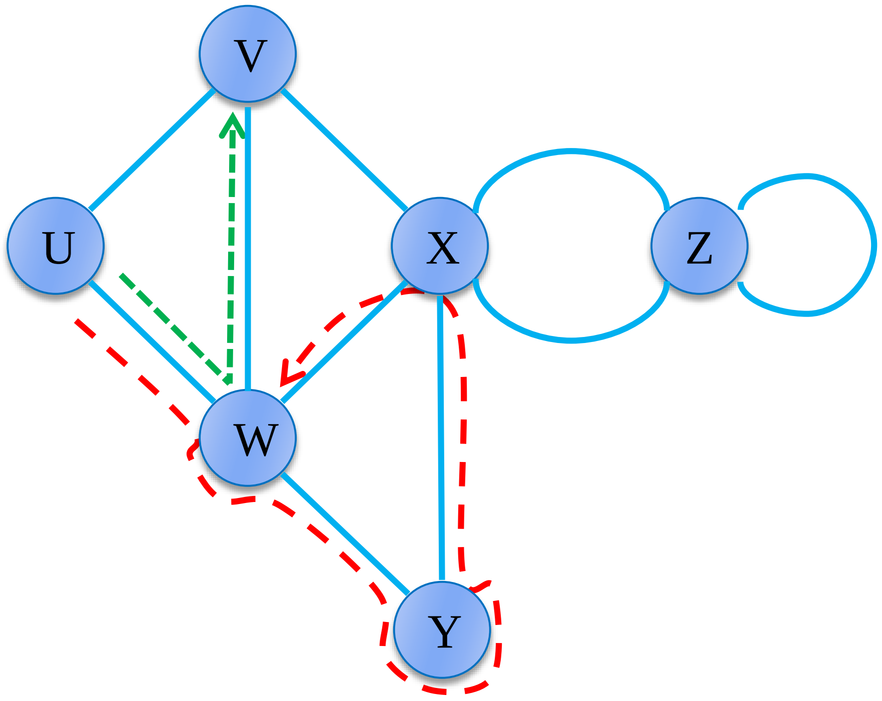


Terminologies

Path

A path between two vertices is sequence of alternating vertices and edges, where each successive vertex is connected.





Terminologies

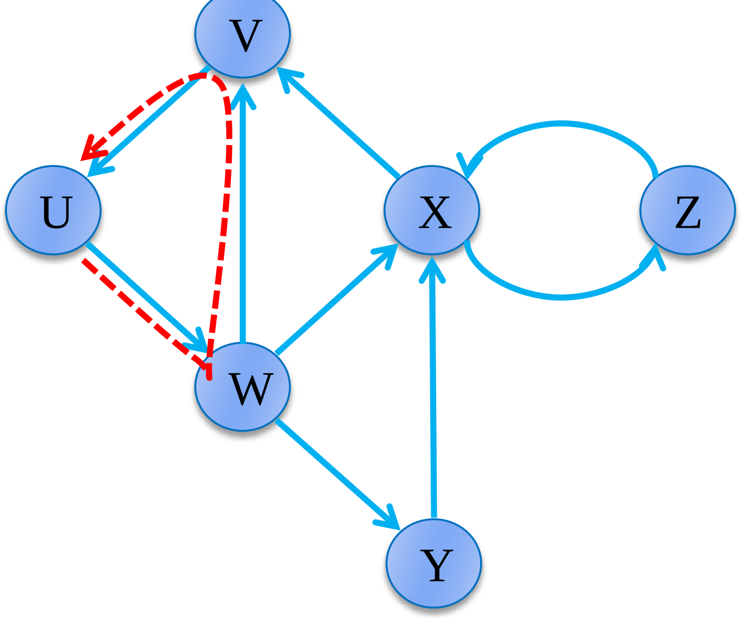
Simple Path

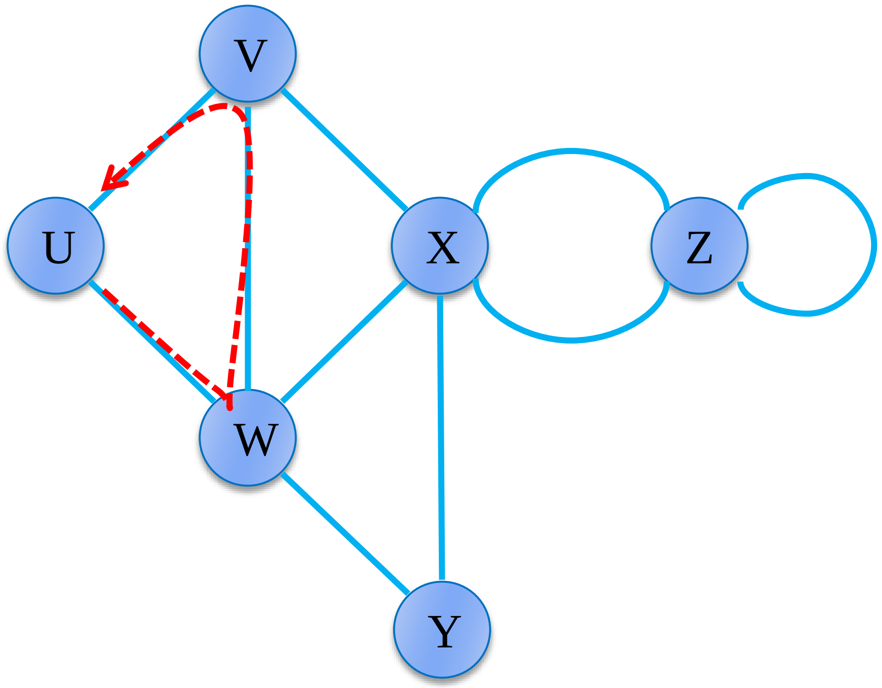
A path with distinct nodes

One of the possible paths from u to v is u-w-v

Cycle

A path that starts and finish at same vertex



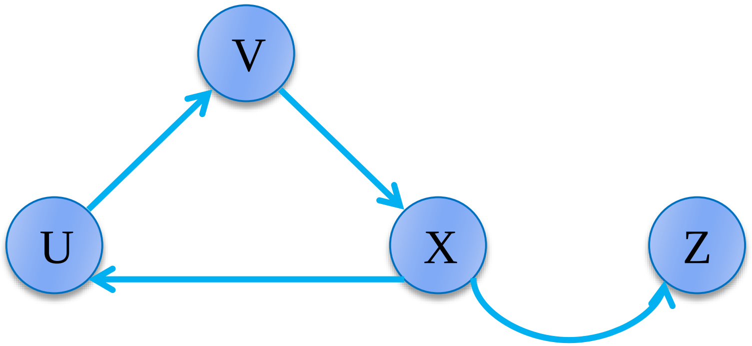


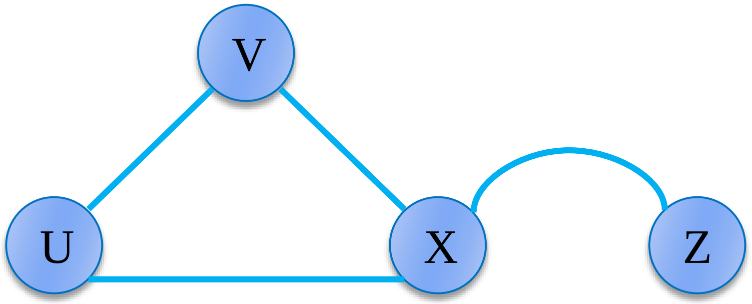
Terminologies

Reachability of Vertex

A vertex is v reachable from other vertex if there exists a path from other vertex to vertex v

In undirected graph, edge is considered as 2-ways, so every vertex is reachable in following example.





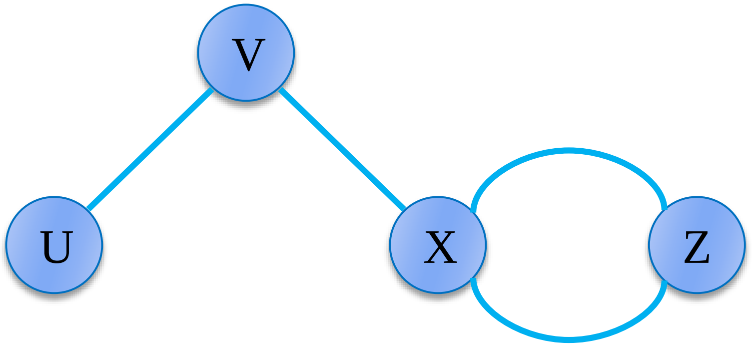
U , V and X are not reachable from Z

all vertices are reachable from each other

Terminologies

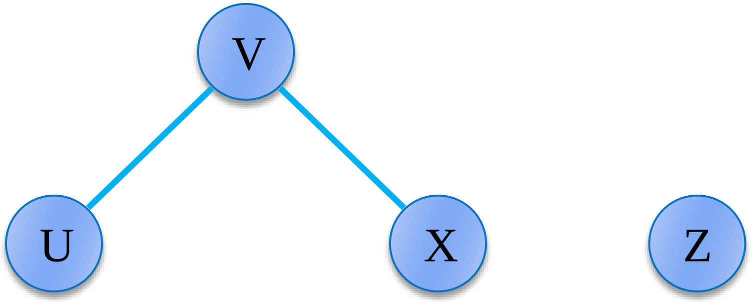
Connected Graph

Each vertex is reachable from every other vertex, in other words there exists a path from each vertex to every other vertex



Disconnected Graph

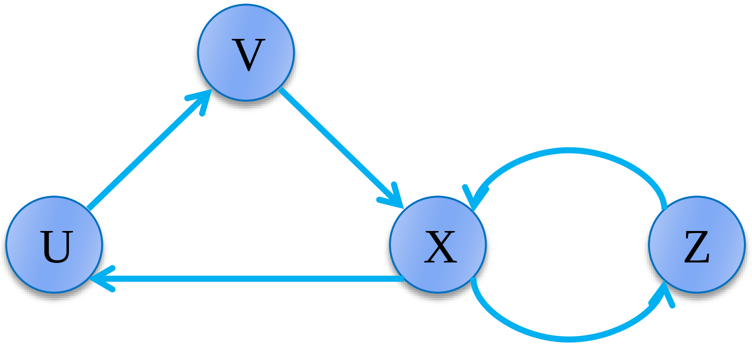
A graph that is not connected



Terminologies

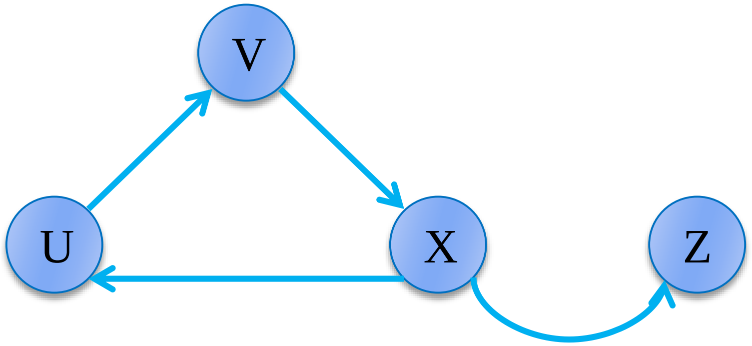
Strongly Connected Graph

If there exists a directed path between each pair of vertices



Weakly Connected Graph

If there exists a path between each pair of vertices which ignores direction

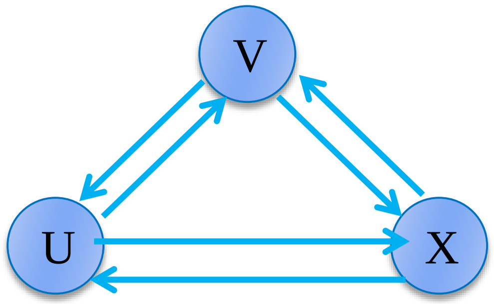


Terminologies

Completely Connected Graph

If there exists an edge between each pair of vertices

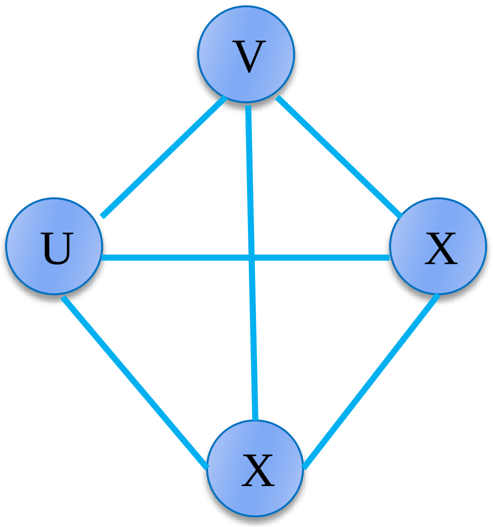
Directed



Maximum Edges?

$|E|=|V|*|V-1| =O(|V|^2)$

Undirected



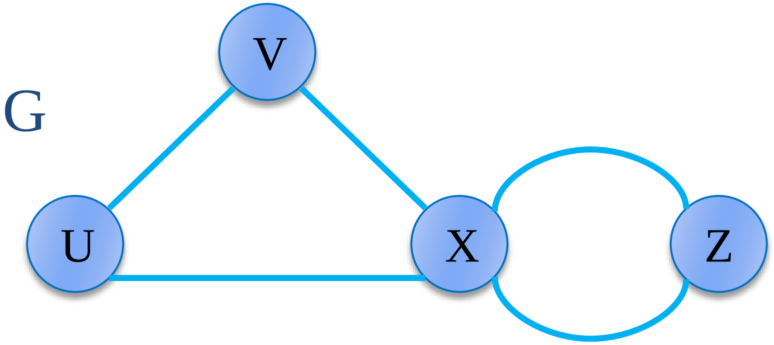
Maximum Edges

$|E|=|V|*|V-1|/2= O(|V|^2)$

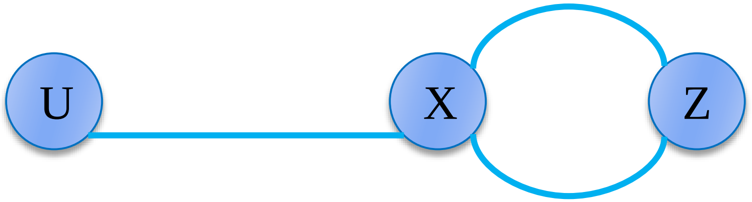
Terminologies

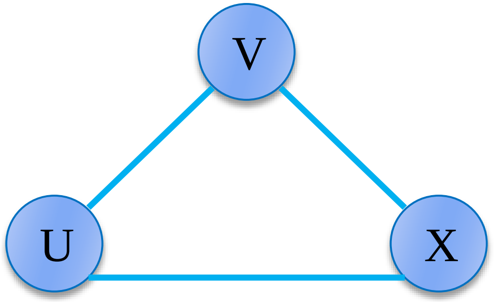
Sub Graph

A graph that is consists of subset of vertices and edges of G



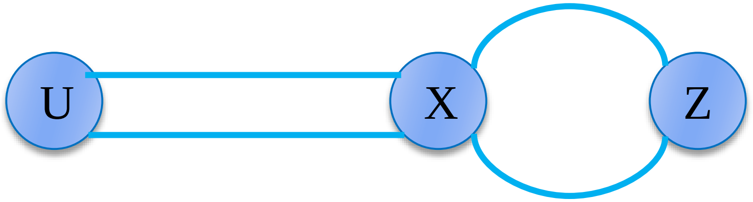
Two possible sub-graphs





Following is not valid sub-graph of G, why?

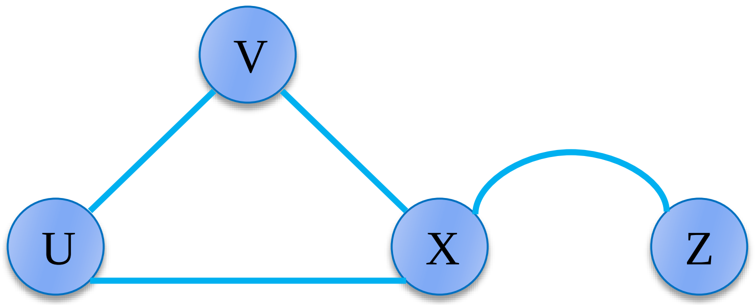
Look at definition of sub-graph



Terminologies

Simple Graph

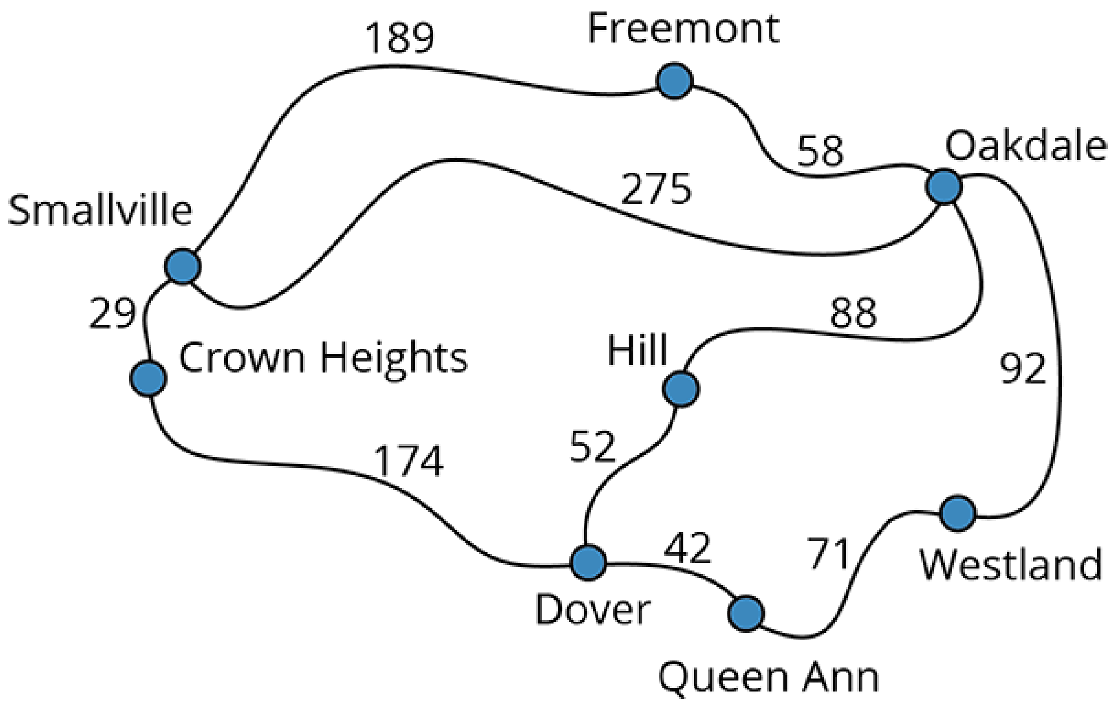
A graph with no parallel and self edges



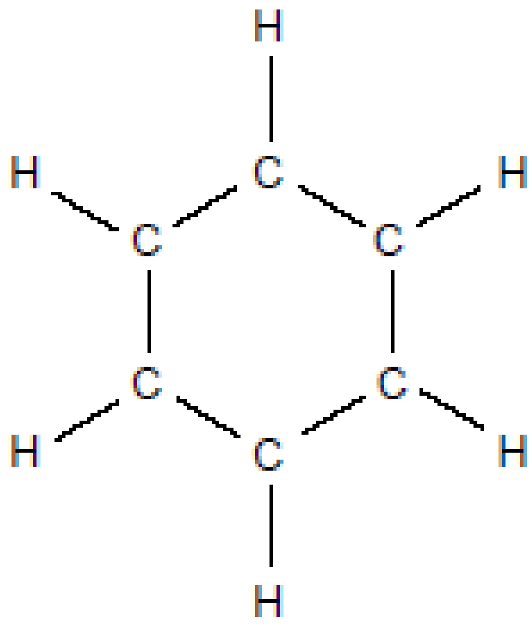
Terminologies

Weighted Graph

Weights on edge means cost or distance between end points of edge



Non-Weighted Graph

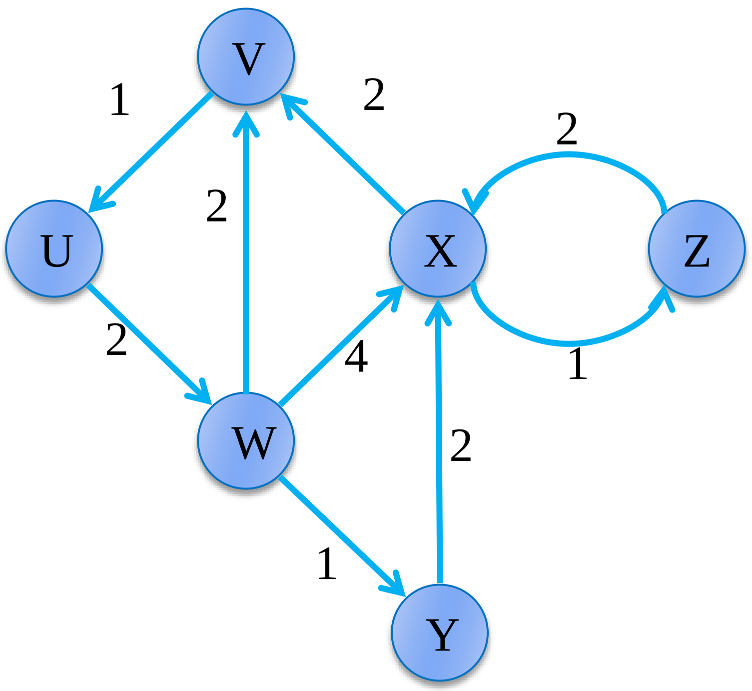


Benzene molecule

Terminologies

Path Length

- Sum of weights of edges on path from one vertex to other.
  - Length of path between u and w is 2
  - Length of path between u and y is 3
  - There are two paths from u to x
    - Path u-w-x has length 6 and path u-w-y-x has length 5
- Shortest Path
  - Path with minimum length
  - From u to x is 5
  - From u to v is 4

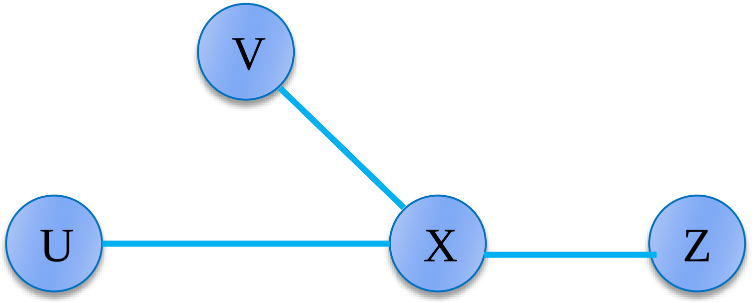


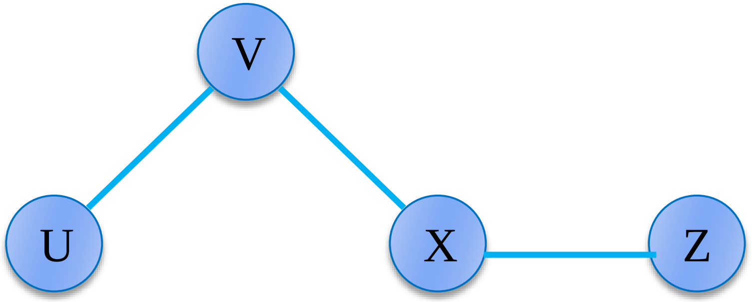
Terminologies

Tree

An undirected graph G is tree if it fulfills any of the following condition:

- G has V-1 edges and no cycles
- G has V-1 edges and is connected
- G is connected, but removing any edge disconnects it
- G is acyclic, but adding any edges creates a cycle
- Exactly one simple path between each pair of vertices in G

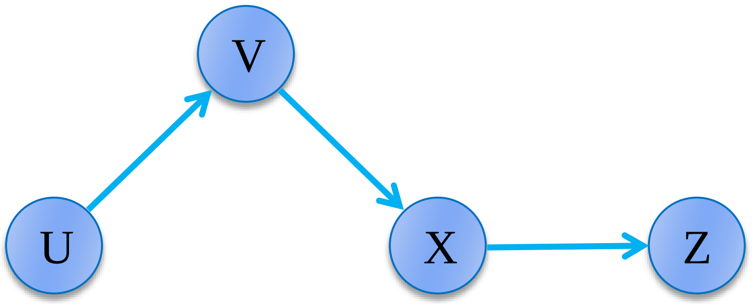




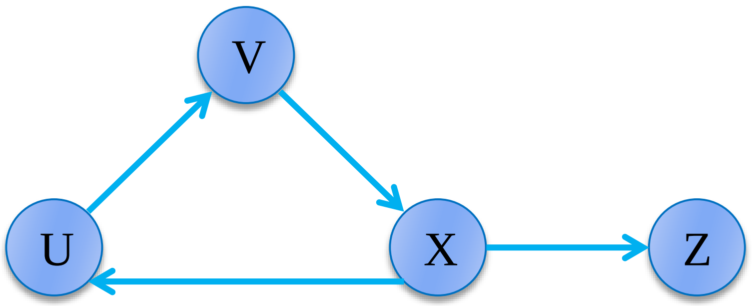
Terminologies

Directed Acyclic Graph (DAG)

Directed graph without cycles



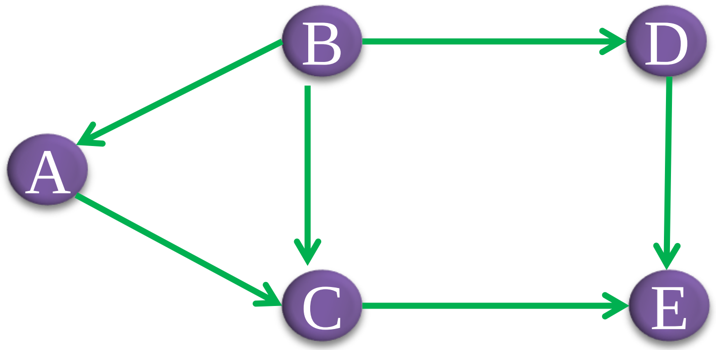
Following graph is not DAG, as it contains a cycle



Terminologies

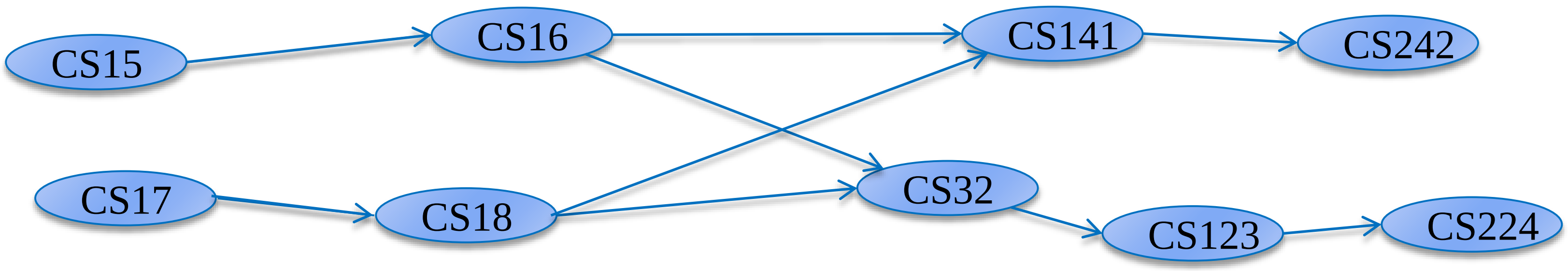
Directed Acyclic Graph (DAG)

A directed graph without cycles



Applications:

- The parse tree constructed by a compiler to execute sequential statements
- Dependency graphs for task scheduling
- Dependency graphs between classes formed by inheritance relationships in object-oriented programming languages
- Information categorization systems, such as folders in a computer
- Course pre-requisites



Graph ADT

Common methods for Graph ADT can be:

- numVertices()
- vertices()
- numEdges()
- edges()
- outgoingEdges(v)
- incomingEdges(v)
- getEdge(v1, v2)
- endVertices(e)
- opposite(v, e)
- insertVertex(value)
- insertEdge(v1, v2, value)
- removeVertex(v)
- removeEdge(e)

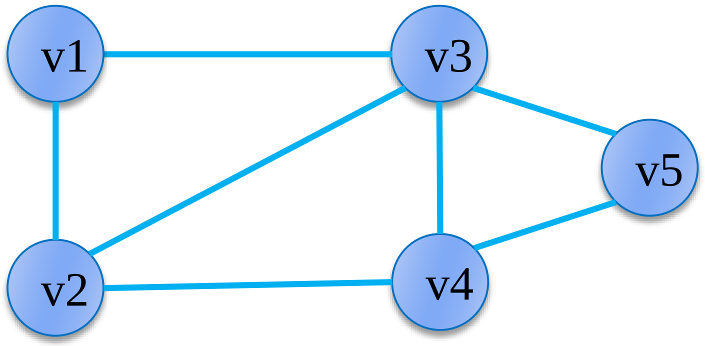
Run time depends upon underlying implementation

Memory Representation

There are multiple ways to represent a graph in memory:

- Adjacency Matrix
- Adjacency List
- Edge List
- Adjacency Map

Assume following example for all representations



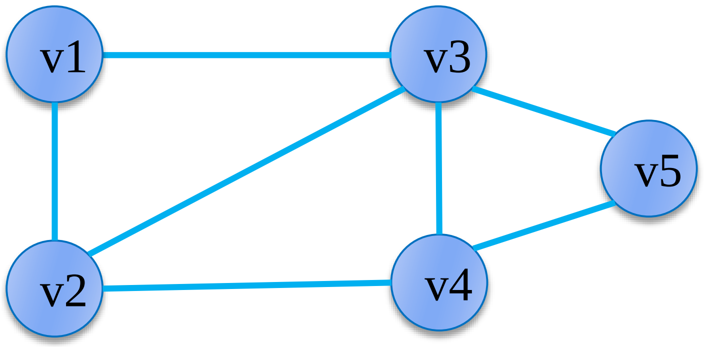
Adjacency Matrix

The adjacency matrix of a graph G = (V, E) is a |V | × |V | matrix E, where each entry E_{ij} is equal to 1 if there exists an edge e = (vi, vj) ∈ E and 0 otherwise.

1 and 0 can also be replaced with true/false.

Vertex list itself is stored in 1D array

0	1	2	3	4
"V1"	"V2"	"V3"	"V4"	"V5"



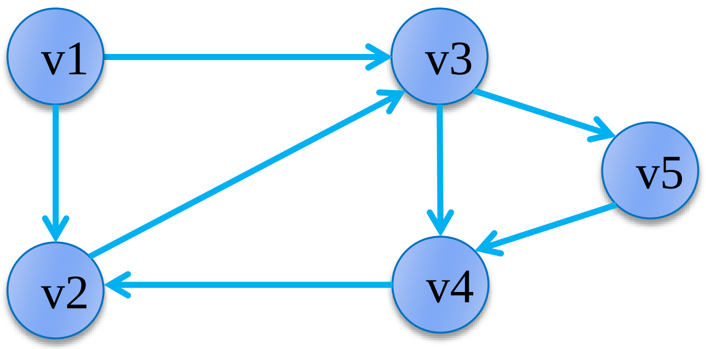
	V1	V2	V3	V4	V5
V1	0	1	1	0	0
V2	1	0	1	1	0
V3	1	1	0	1	1
V4	0	1	1	0	1
V5	0	0	1	1	0

Adjacency Matrix

If graph is directed?

Then entry E_{ij} is equal to 1 if there exists an edge e = from vi to vj and 0 otherwise.

	V1	V2	V3	V4	V5
V1	0	1	1	0	0
V2	0	0	1	0	0
V3	0	0	0	1	1
V4	0	1	0	0	0
V5	0	0	0	1	0

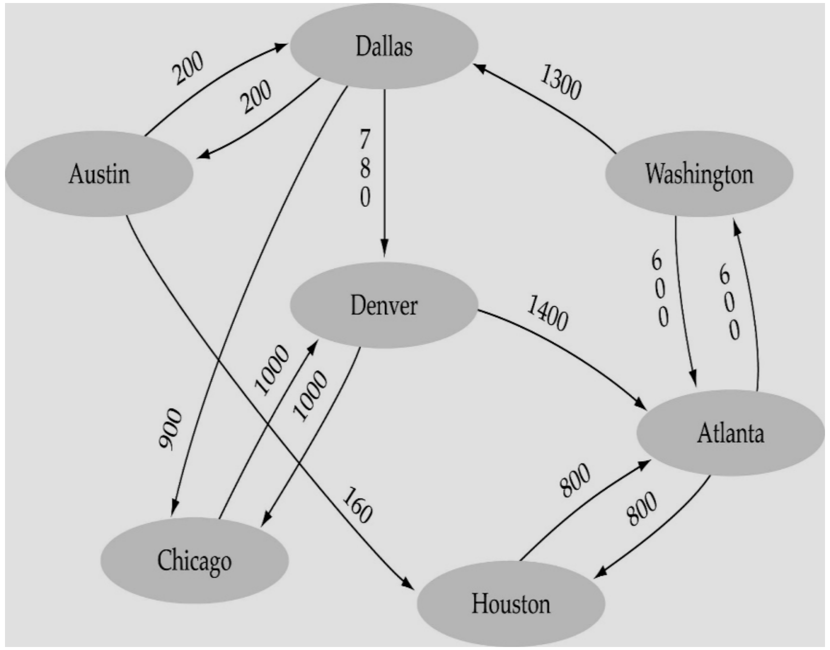


Weighted Graph:

1 and 0 are replaced with respective weights

0 or -1 presents no edge

Example



graph
.numVertices 7
.vertices

[0]	"Atlanta "
[1]	"Austin "
[2]	"Chicago "
[3]	"Dallas "
[4]	"Denver "
[5]	"Houston "
[6]	"Washington"
[7]	
[8]	
[9]	

.edges

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[0]	0	0	0	0	0	800	600	•	•	•
[1]	0	0	0	200	0	160	0	•	•	•
[2]	0	0	0	0	1000	0	0	•	•	•
[3]	0	200	900	0	780	0	0	•	•	•
[4]	1400	0	1000	0	0	0	0	•	•	•
[5]	800	0	0	0	0	0	0	•	•	•
[6]	600	0	0	1300	0	0	0	•	•	•
[7]	•	•	•	•	•	•	•	•	•	•
[8]	•	•	•	•	•	•	•	•	•	•
[9]	•	•	•	•	•	•	•	•	•	•

(Array positions marked '•' are undefined)

Adjacency Matrix

- Vertex list and Edge list are given
- Running Time?
  - Get a vertex's out-edges
  - Get a vertex's in-edges
  - Decide if some edge exists
  - Insert an edge
  - Delete an edge
  - Inset a vertex
  - Remove a vertex
- Memory
  - $O(|V|^2)$
- Good for?
  - Dense graphs
    - Edges are close to $|V|^2$

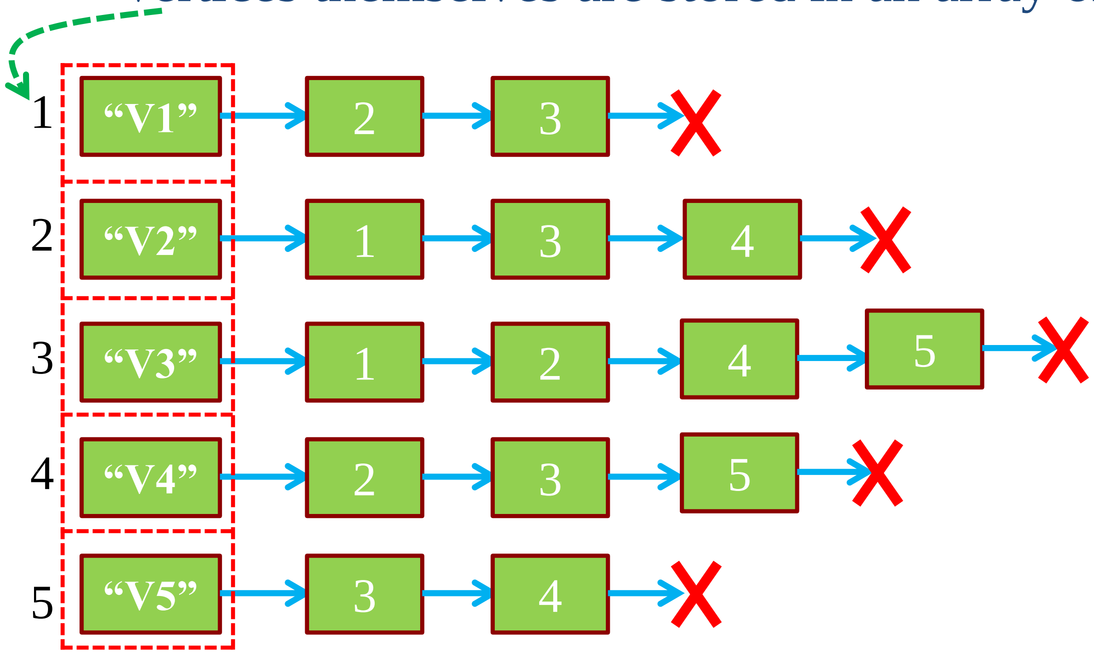
0	1	2	3	4
"V1"	"V2"	"V3"	"V4"	"V5"

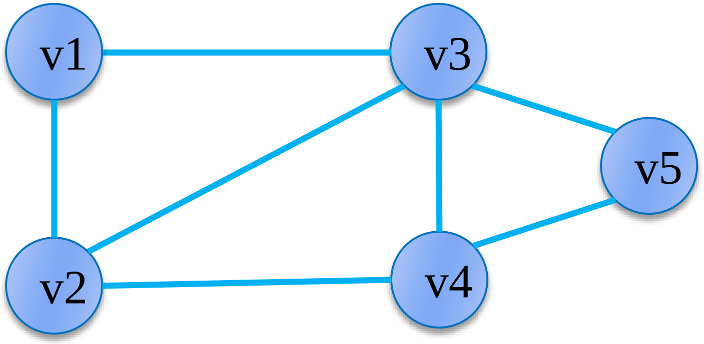
	V1	V2	V3	V4	V5
V1	0	1	1	0	0
V2	0	0	1	0	0
V3	0	0	0	1	1
V4	0	1	0	0	0
V5	0	0	0	1	0

Adjacency List

Each vertex is associated with its list of adjacent nodes.

Vertices themselves are stored in an array or hashmap

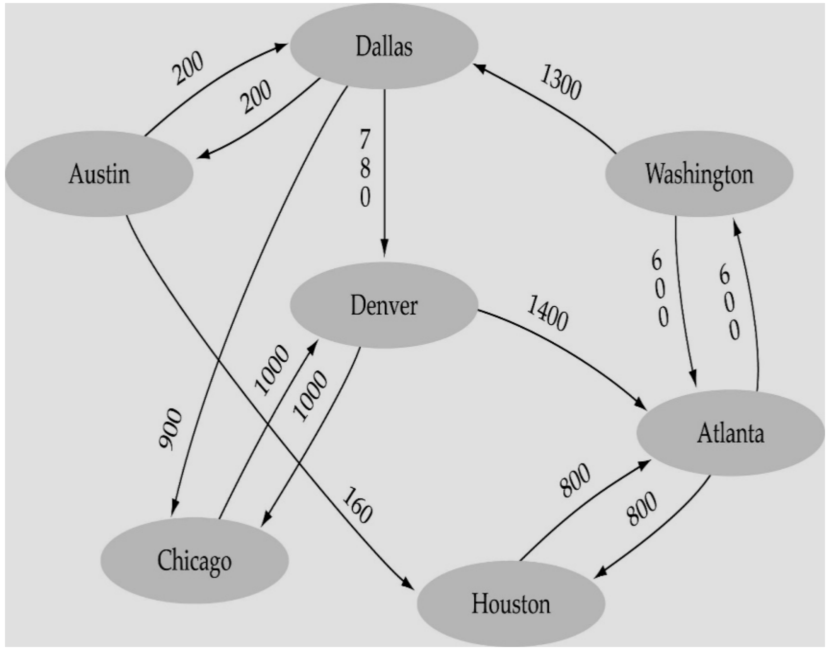


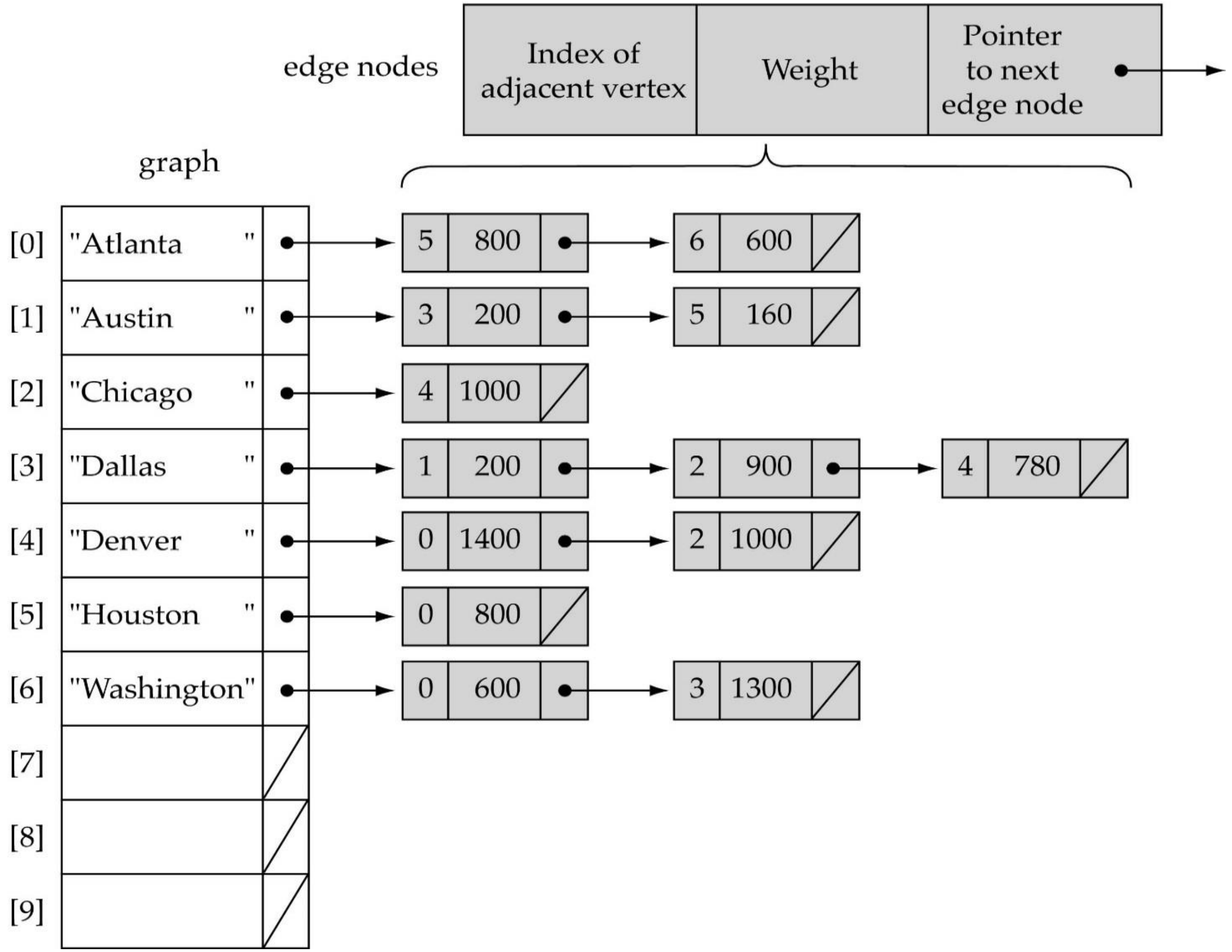


List can be of simple integers/ characters/ strings which represents vertex label or number

Here assumes a hashmap with <key,value> pair where key is vertex label and values is list of its connected vertices

(a)


Dallas
Austin
Washington
Denver
Atlanta
Chicago
Houston
200
200
1300
780
600
600
1400
1000
1000
900
160
800
800


edge nodes
Index of adjacent vertex
Weight
Pointer to next edge node
graph
[0] "Atlanta "
[1] "Austin "
[2] "Chicago "
[3] "Dallas "
[4] "Denver "
[5] "Houston "
[6] "Washington"
[7]
[8]
[9]
5 800
6 600
3 200
5 160
4 1000
1 200
2 900
4 780
0 1400
2 1000
0 800
0 600
3 1300

Adjacency List

Running Time?

- Get a vertex's out-edges
- Get a vertex's in-edges
- Decide if some edge exists
- Insert an edge
- Delete an edge
- Inset a vertex
- Remove a vertex

Memory

- O(|V|+|E|)

Good for?

- Sparse
  - Edges are significantly less than $|V|^2$