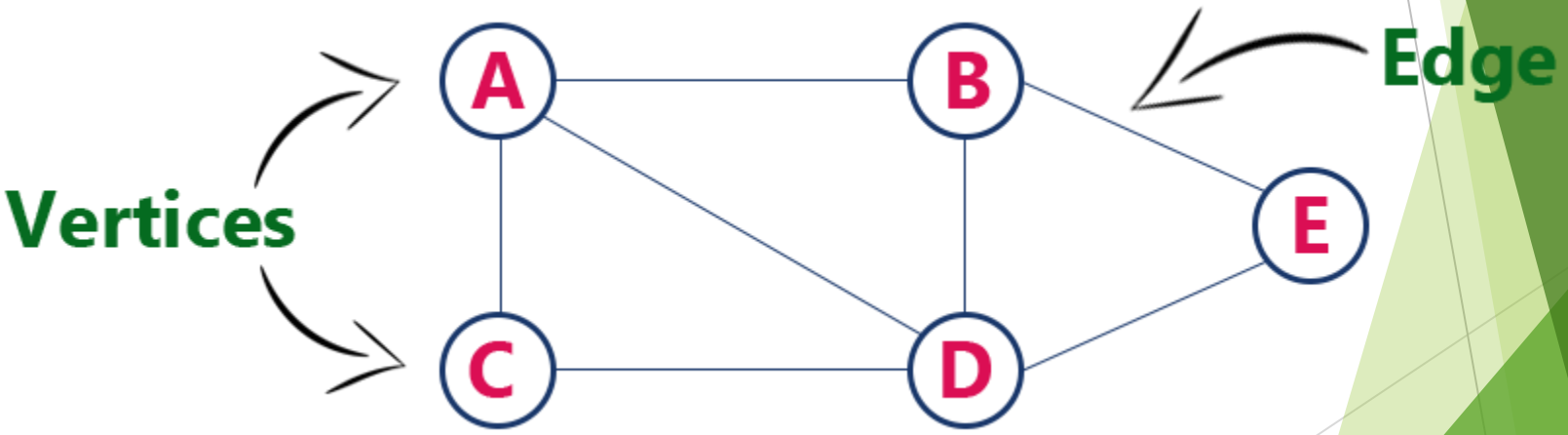
01418231
Data Structure

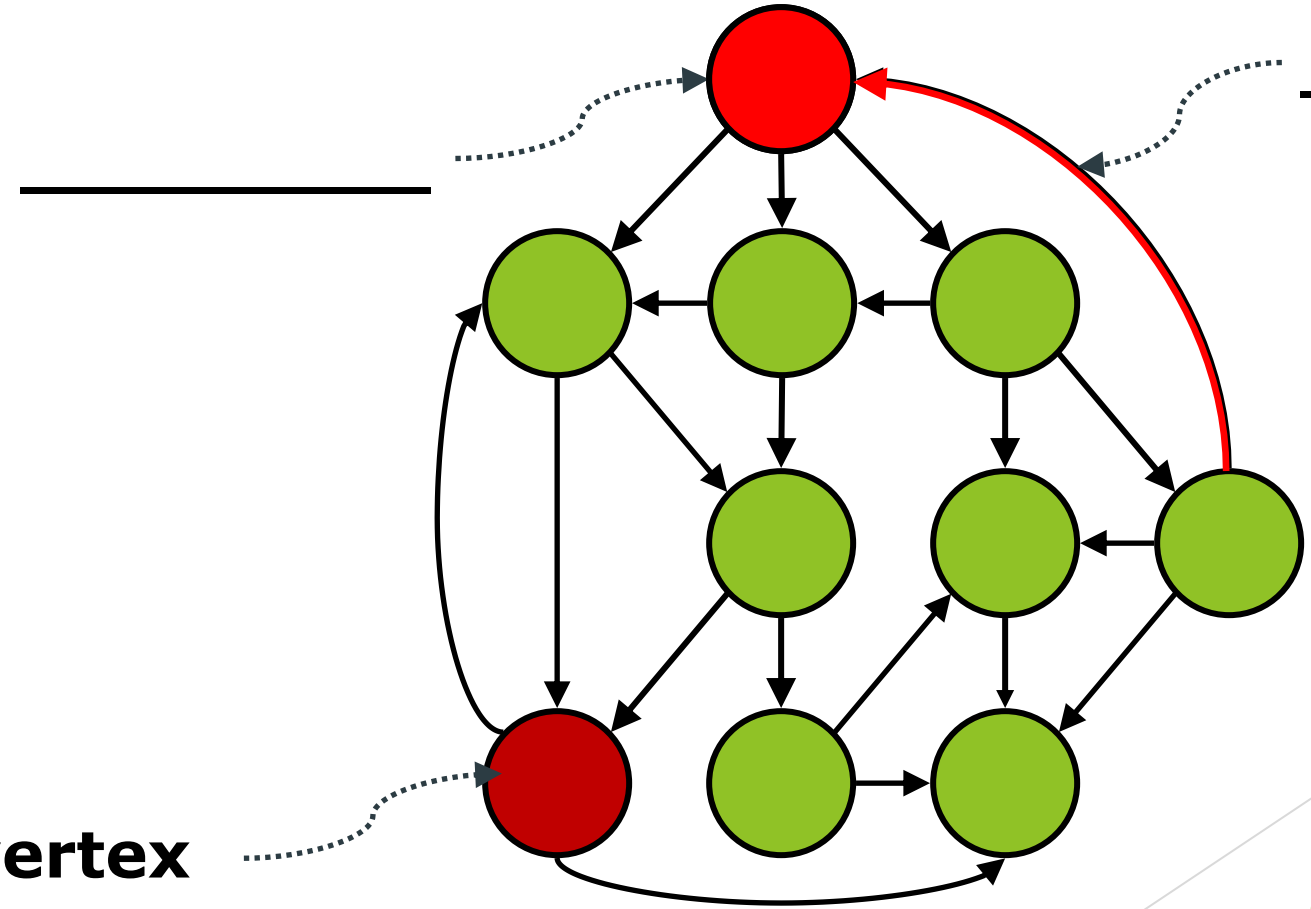
Graph



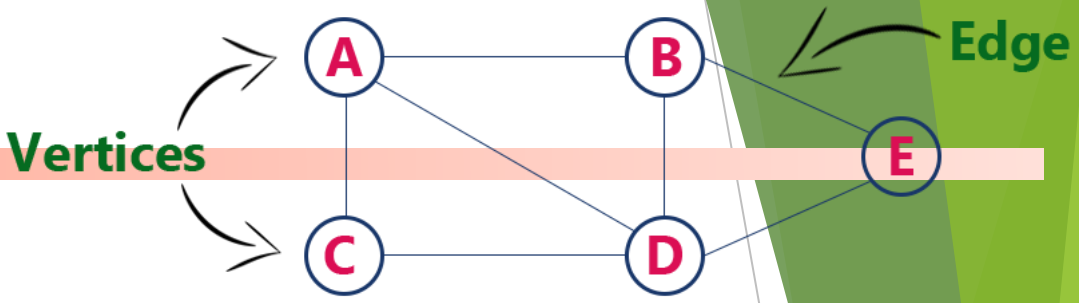
http://btechsmartclass.com/DS/U3_T8.html

Graph and Traversal

Graph



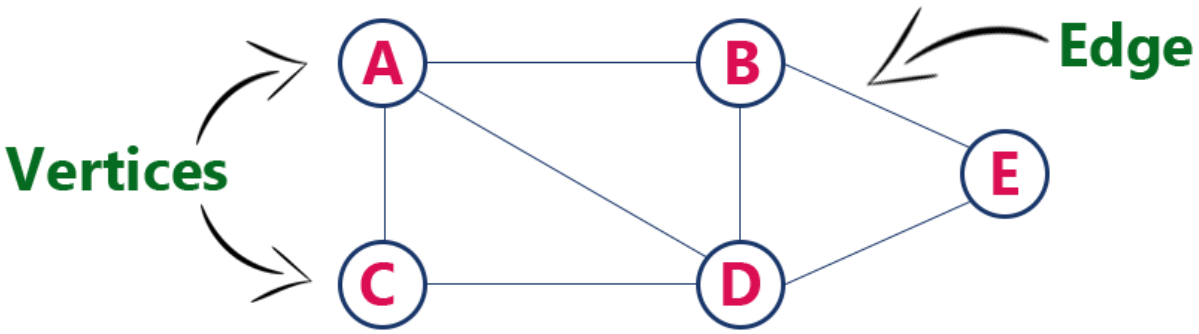
Graphs



- G = (V,E)
- V is the vertex set
  - Vertices are also called nodes and points.
  - A, B, C, D & E are known as vertices.
- E is edge or arc that connects two between vertices.
  - (A,B) is the link between vertices A and B
- Edges are three types.
  1. _____ - An undirected egde is a bidirectional edge. If there is a undirected edge between vertices A and B then edge (A , B) is equal to edge (B , A).
  2. _____- A directed egde is a unidirectional edge. If there is a directed edge between vertices A and B then edge (A , B) is not equal to edge (B , A).
  3. _____ - A weighted egde is an edge with cost on it.

Graphs

- This graph G can be defined as G = (V , E)
- Where V = {A,B,C,D,E}
- and E = {(A,B)_____}



http://btechsmartclass.com/DS/U3_T8.html

Graphs

- Undirected edge has no orientation (u,v).

u v

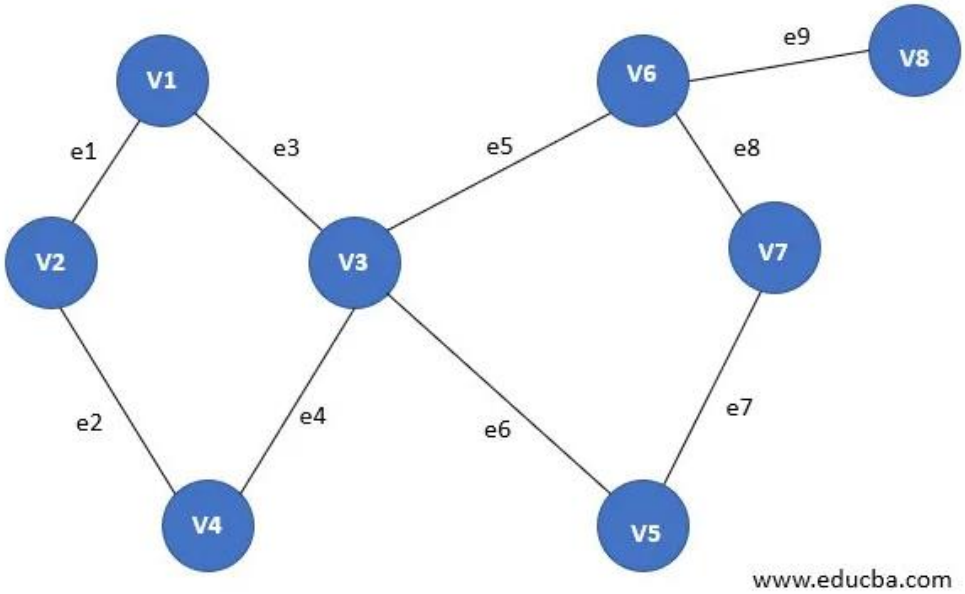
- Directed edge has an orientation (u,v)

u v

- Weighted Edge has an orientation (u,v) or no orientation

u v

Graphs



- _____ = The number of vertices in the graph.
- _____ = The number of edges in the graph.
- _____ = Number of edges incident to the vertex.

https://www.educba.com/types-of-graph-in-data-structure/

Types of Graphs in Data Structures

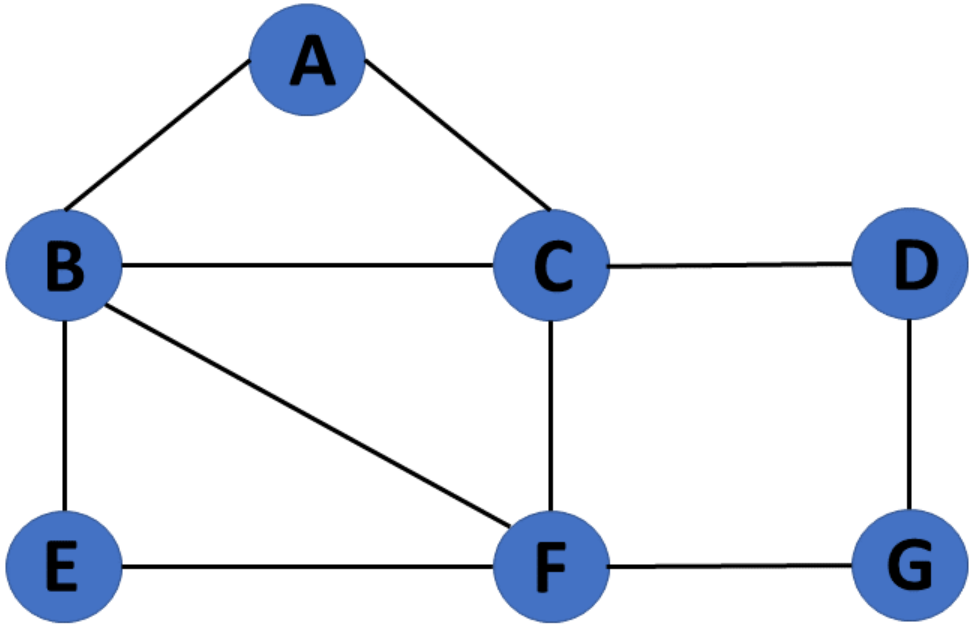
Types of Graphs in Data Structures

1. Finite Graph
2. Infinite Graph
3. Trivial Graph
4. Simple Graph
5. Multi Graph
6. Null Graph
7. Complete Graph
8. Pseudo Graph
9. Regular Graph
10. Weighted Graph
11. Directed Graph
12. Undirected Graph
13. Connected Graph
14. Disconnected Graph
15. Cyclic Graph
16. Acyclic Graph
17. Directed Acyclic Graph
18. Subgraph

https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

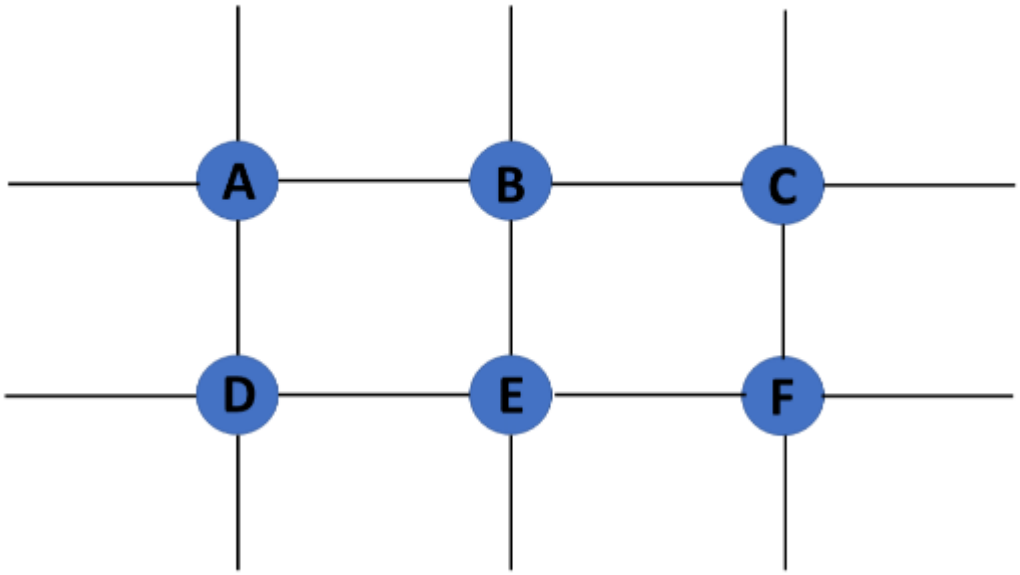
- The graph G=(V, E) is called a finite graph if the number of vertices and edges in the graph is limited in number



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

- The graph G=(V, E) is called a infinite graph if the number of vertices and edges in the graph is interminable.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

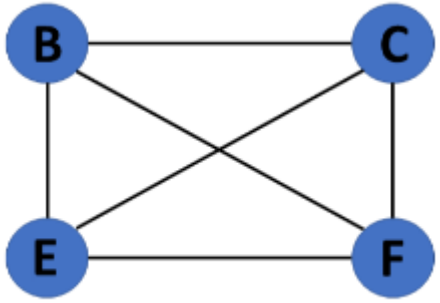
Graph

- A graph G= (V, E) is trivial if it contains only a single vertex and no edges.

https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

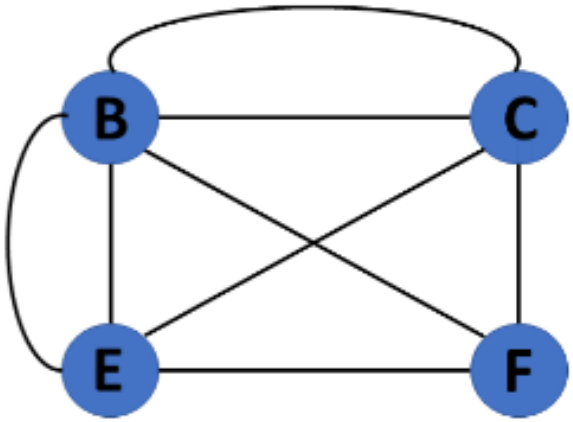
- If each pair of nodes or vertices in a graph G=(V, E) has only one edge, it is a simple graph.
- As a result, there is just one edge linking two vertices, depicting one-to-one interactions between two elements.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

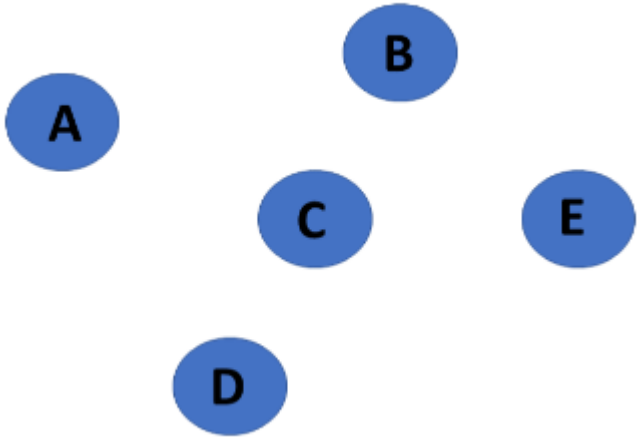
- If there are numerous edges between a pair of vertices in a graph G= (V, E), the graph is referred to as a multigraph.
- There are no self-loops in a Multigraph.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

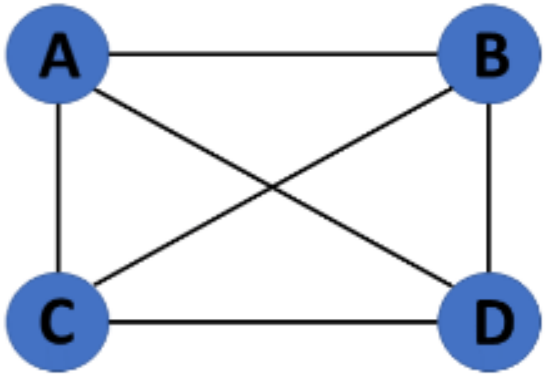
- It's a reworked version of a trivial graph. If several vertices but no edges connect them, a graph G= (V, E) is a null graph.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

- If a graph G= (V, E) is also a simple graph, it is complete.
- Using the edges, with n number of vertices must be connected.
- It's also known as a full graph because each vertex's degree must be n-1.

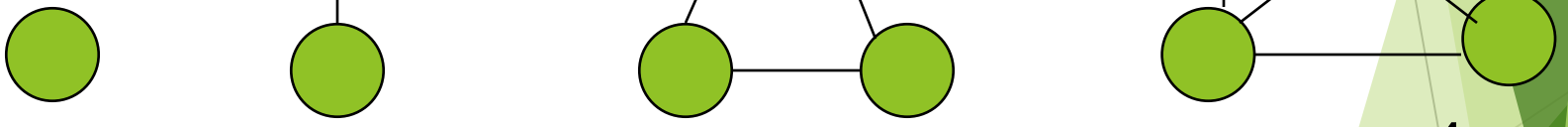


https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

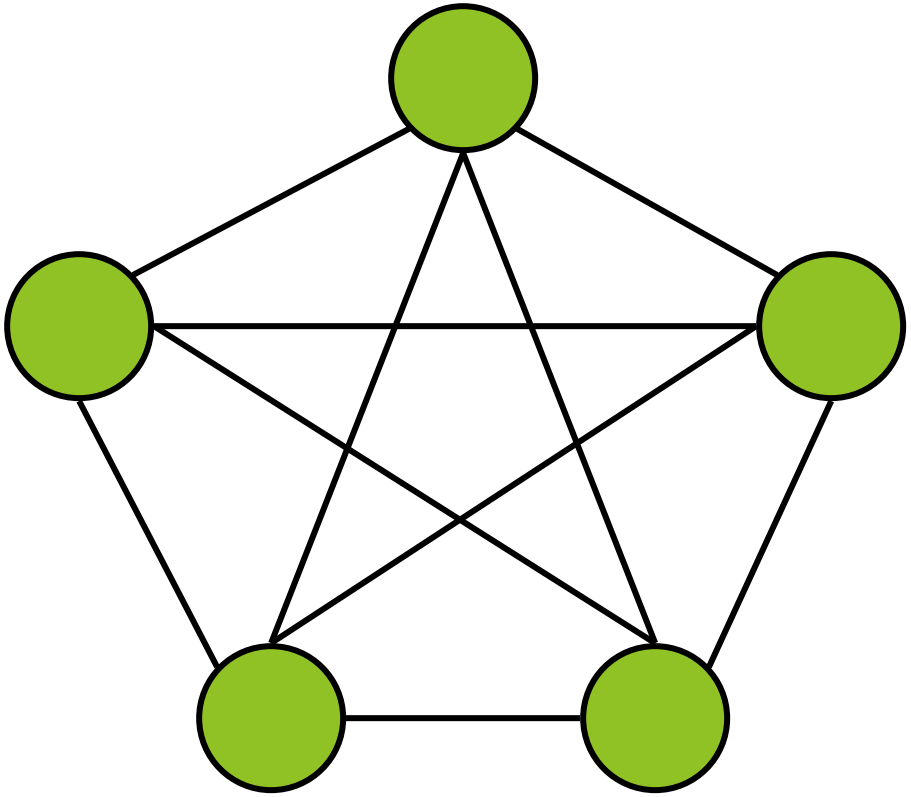
Have all possible edges.

- Undirected Graph ,Directed Graph



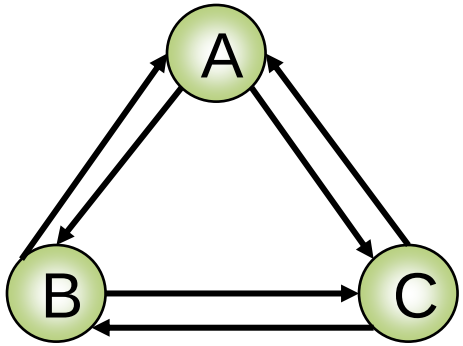
n = 1 n = 2 n = 3 n = 4

Graph (Clique)



Graph (Clique)

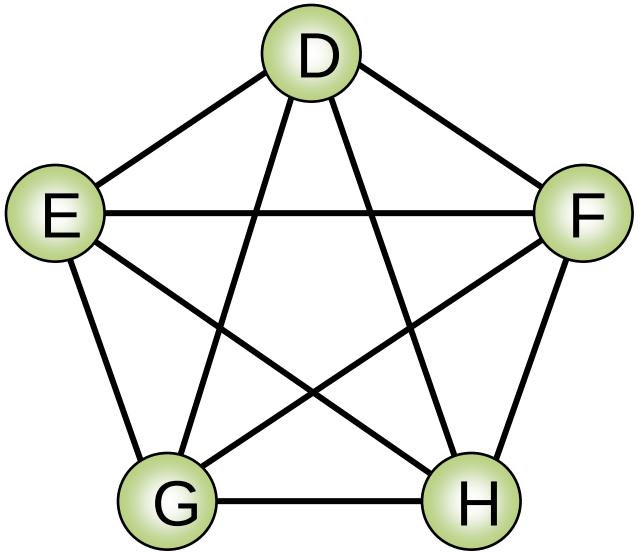
- Number of edges of graph



Directed Graph

Number of Edges = N*(N-1)

3*(3-1) =6



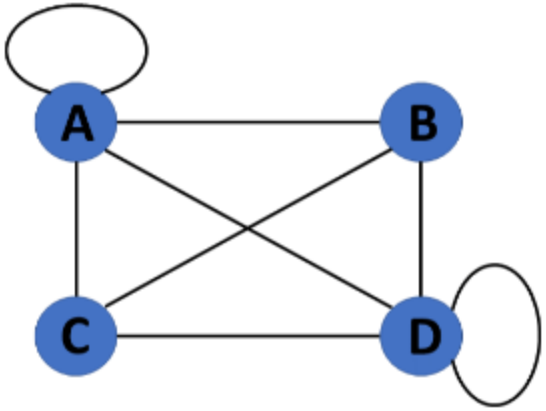
Undirected Graph

Number of Edges = [N*(N-1)] /2

5*(5-1) / 2 = 10

Graph

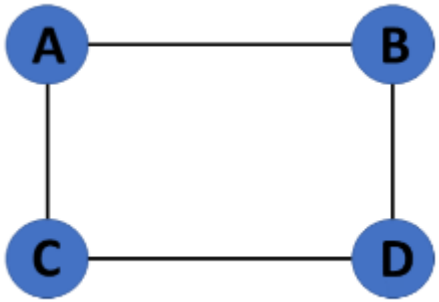
- If a graph G= (V, E) contains a self-loop besides other edges, it is a pseudograph.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

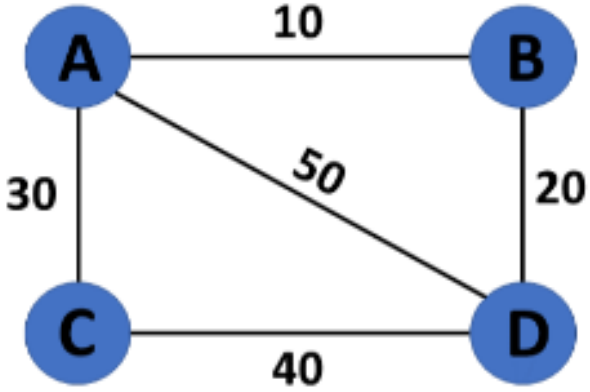
- If a graph G= (V, E) is a simple graph with the same degree at each vertex, it is a regular graph.
- As a result, every whole graph is a regular graph.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

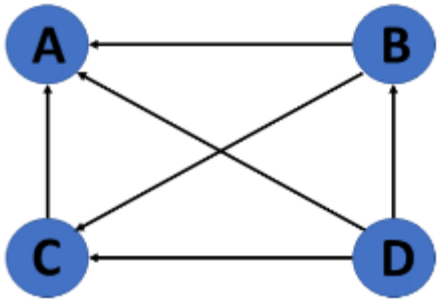
- A graph G= (V, E) is called a labeled or weighted graph because each edge has a value or weight representing the cost of traversing that edge.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

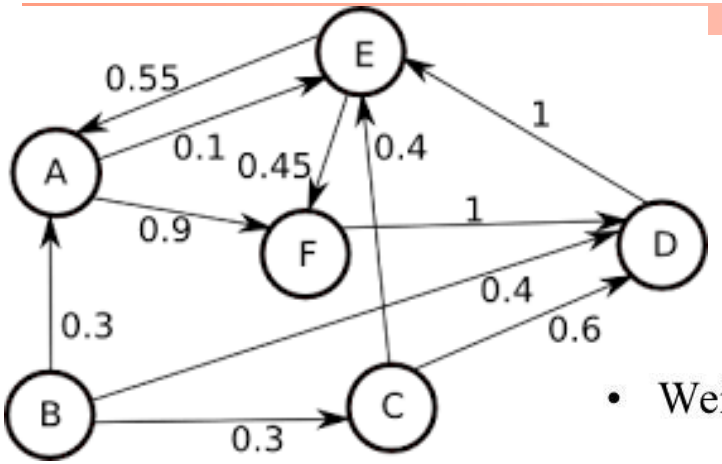
Graph

- A directed graph also referred to as a digraph, is a set of nodes connected by edges, each with a direction.



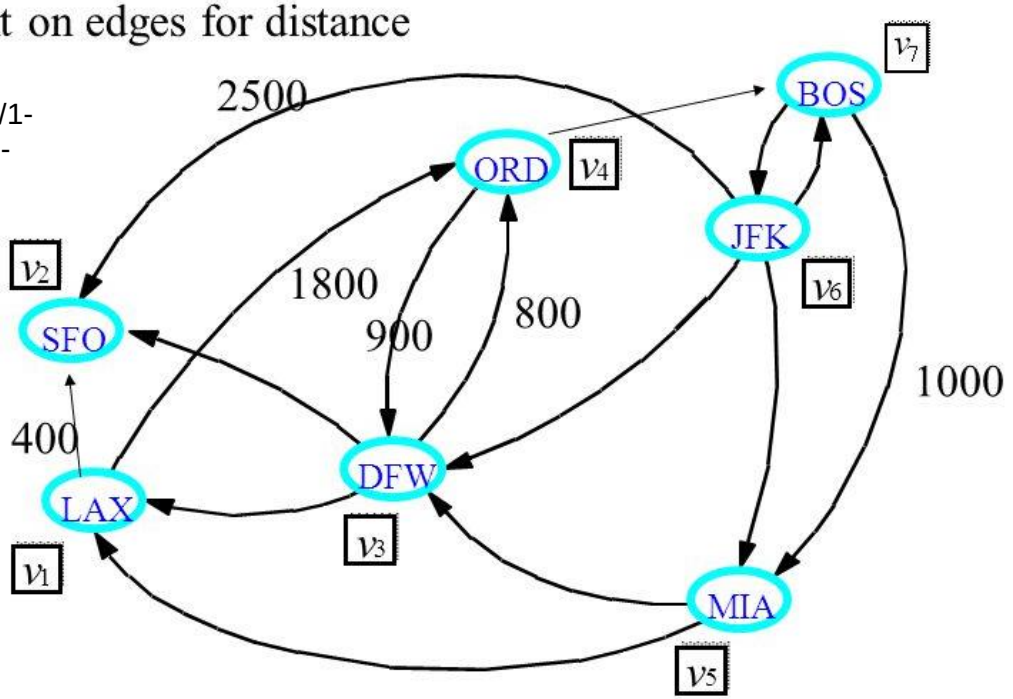
https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph with weighted edges



https://www.chegg.com/homework-help/questions-and-answers/1-consider-directed-graph-weighted-edges-represent-graph-using-adjacency-list-adjacency-ma-q20254763

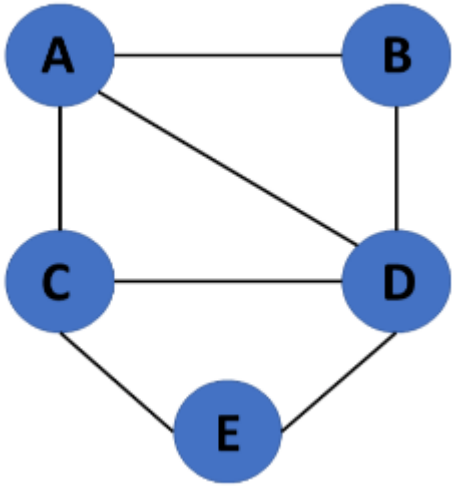
- Weight on edges for distance



https://slideplayer.com/slide/4953310/

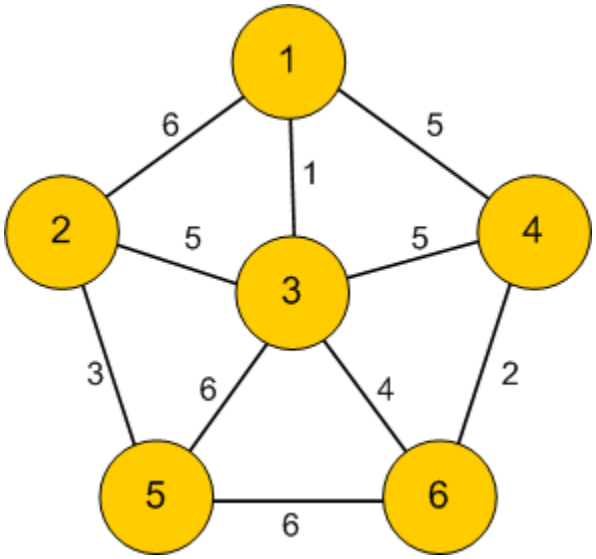
Graph

- An undirected graph comprises a set of nodes and links connecting them.
- The order of the two connected vertices is irrelevant and has no direction.
- You can form an undirected graph with a finite number of vertices and edges.

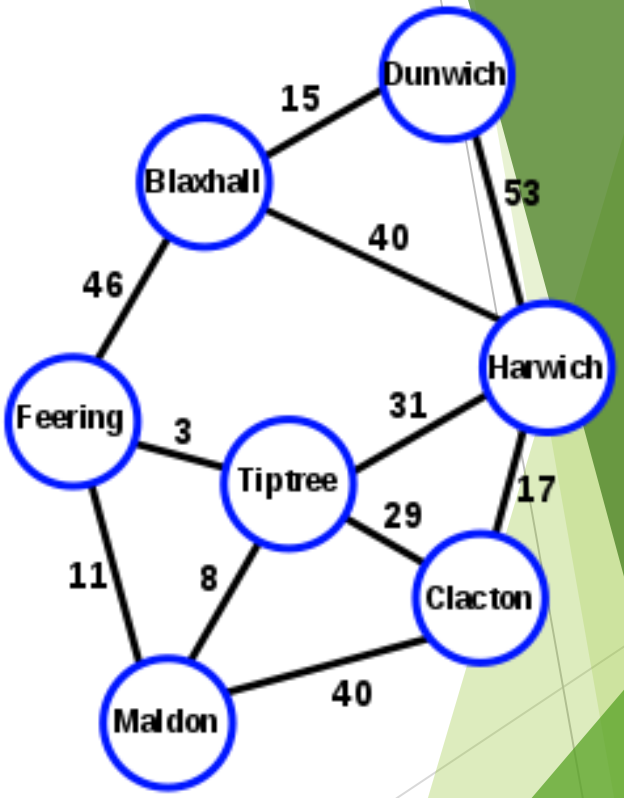


https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph with weighted edges



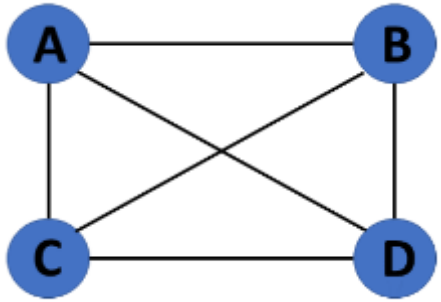
https://forum.qt.io/topic/77736/qt-simple-library-for-drawing-undirected-weighted-graph



https://en.wikipedia.org/wiki/Widest_path_problem

Graph

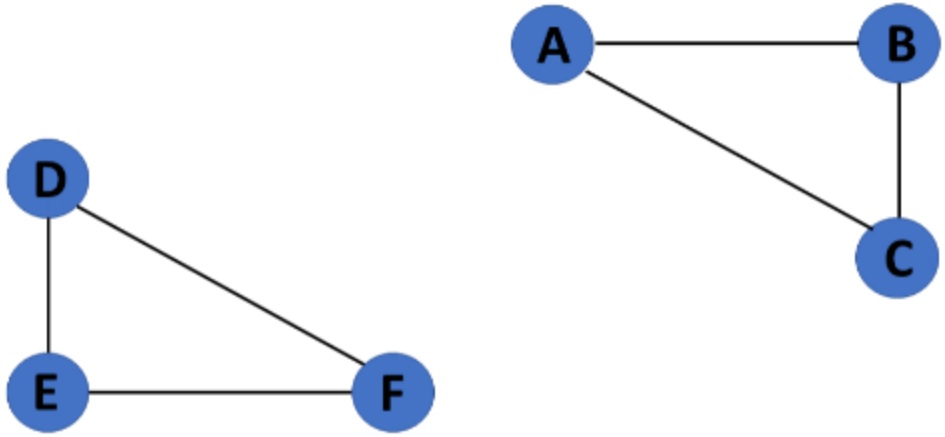
- If there is a path between one vertex of a graph data structure and any other vertex, the graph is connected.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

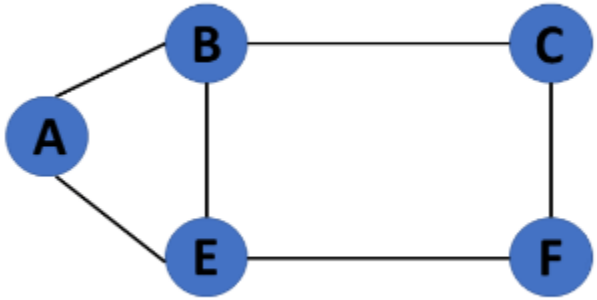
- When there is no edge linking the vertices, you refer to the null graph as a disconnected graph.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

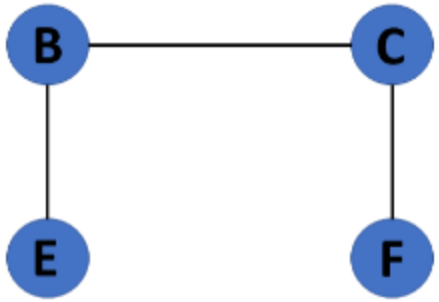
- If a graph contains at least one graph cycle, it is considered to be cyclic.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

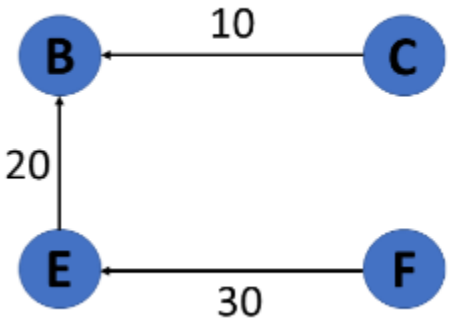
- When there are no cycles in a graph, it is called an acyclic graph.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

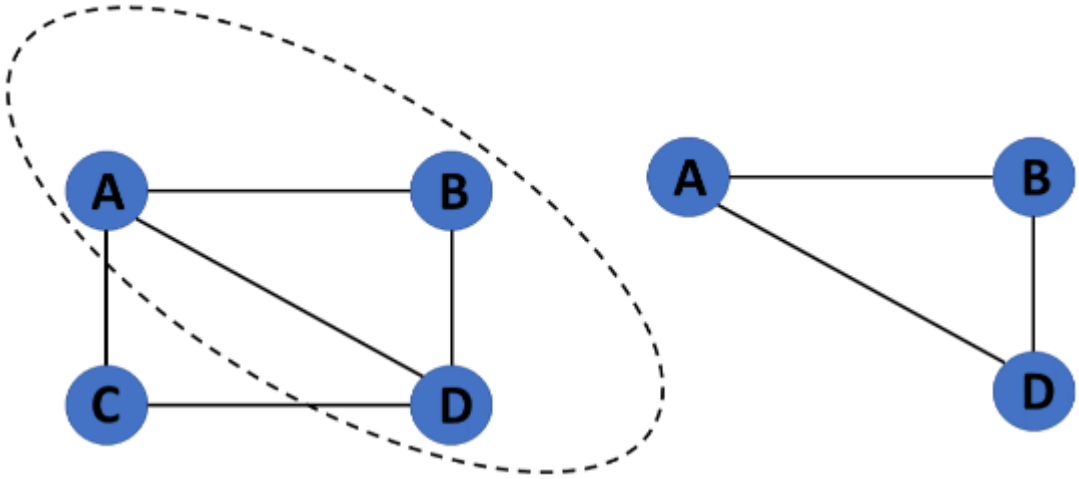
- It's also known as a directed acyclic graph (DAG), and it's a graph with directed edges but no cycle. It represents the edges using an ordered pair of vertices since it directs the vertices and stores some data.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

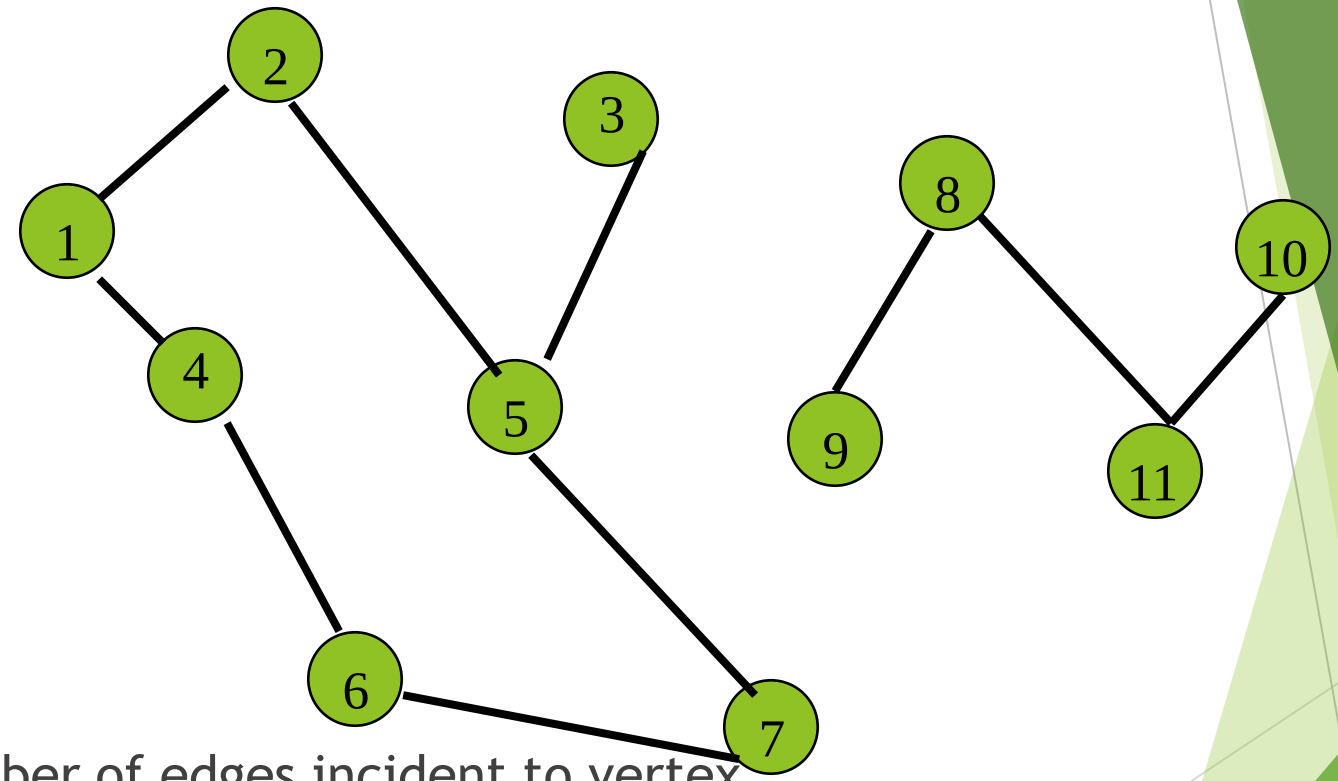
- The vertices and edges of a graph that are subsets of another graph are known as a subgraph



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Graph

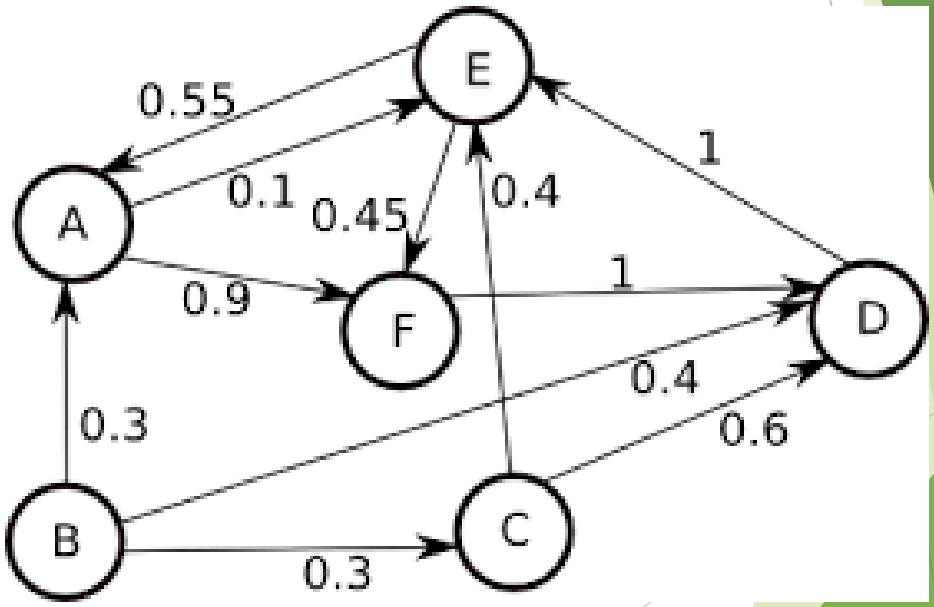
Sum of degrees = 2e (e is number of edges)



- Number of edges incident to vertex.
- Undirected graph
- degree(2) = 2, degree(5) = 3, degree(3) = 1

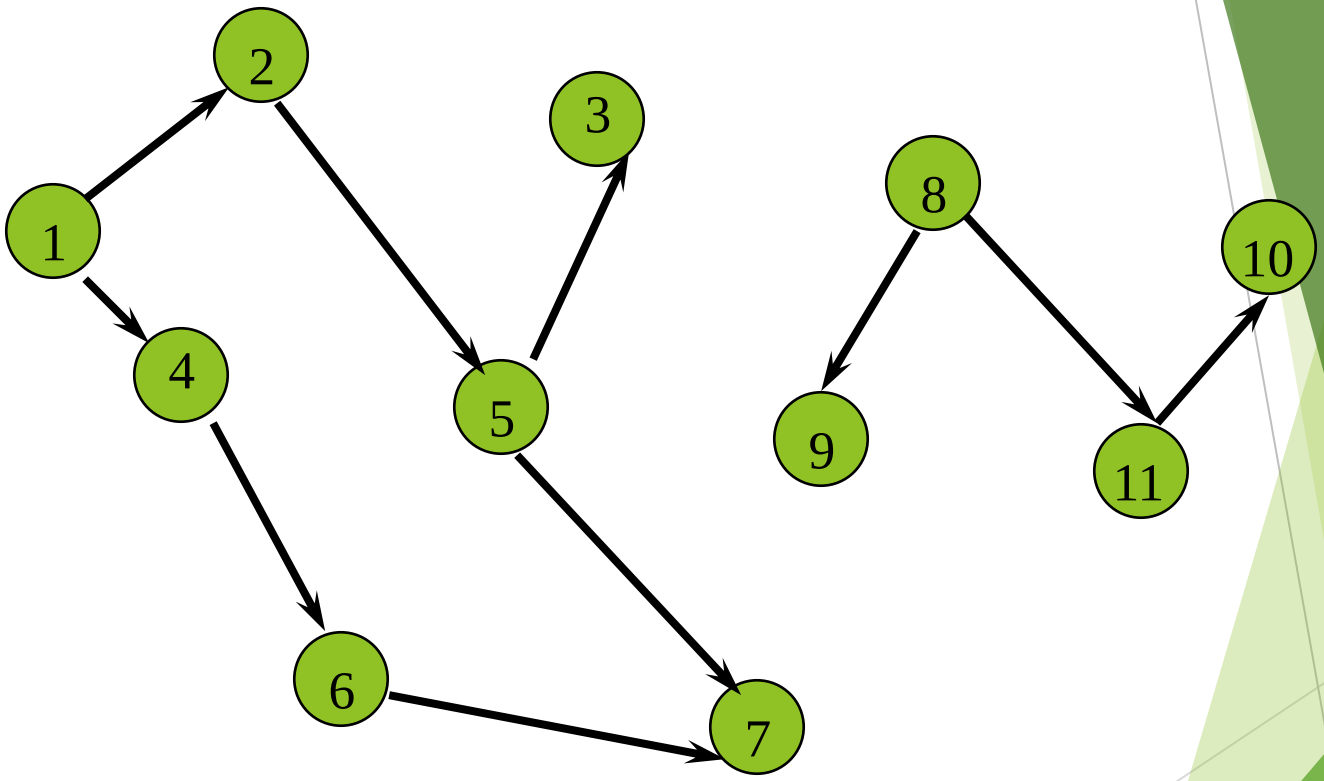
Vertex Degree

- Directed graph
  - Sum of Degree (degree)= _____
    - In-Degree
    - Out-Degree



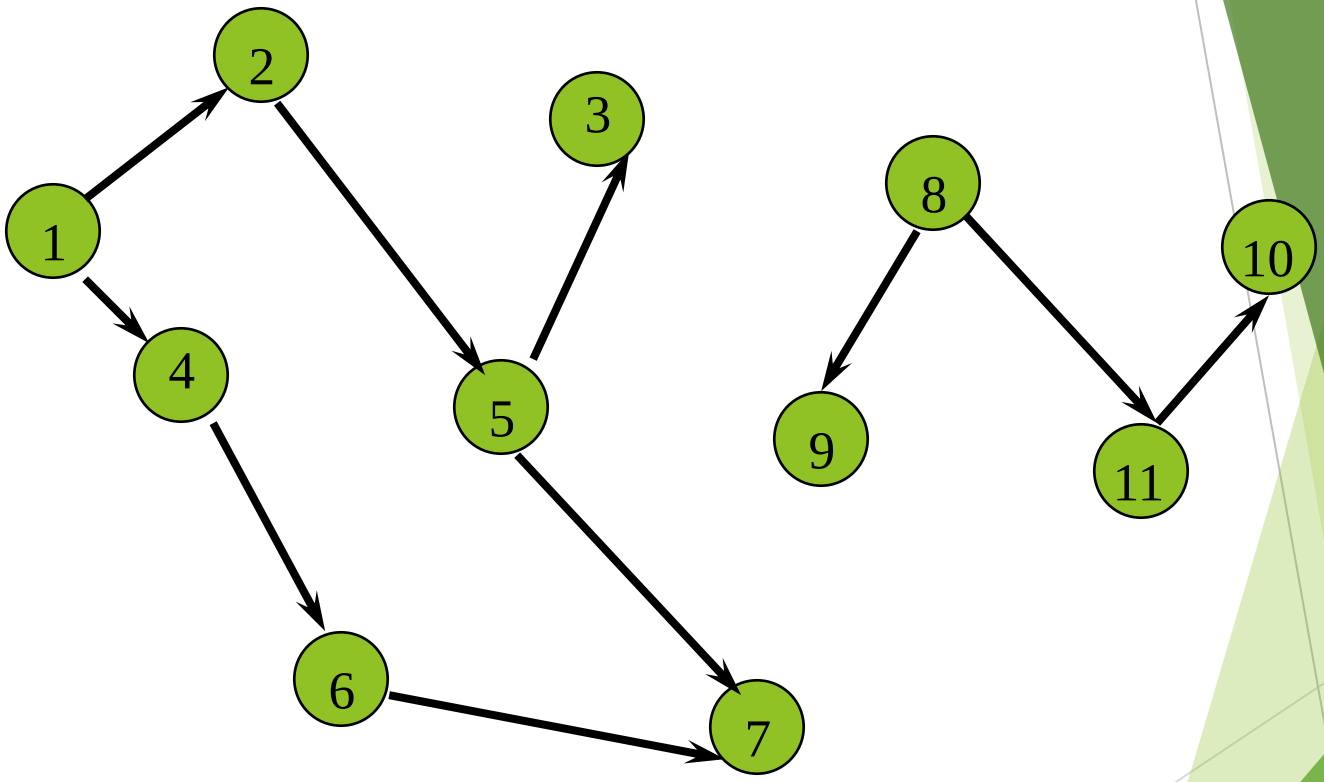
https://www.chegg.com/homework-help/questions-and-answers/1-consider-directed-graph-weighted-edges-represent-graph-using-adjacency-list-adjacency-ma-q20254763

In-Degree



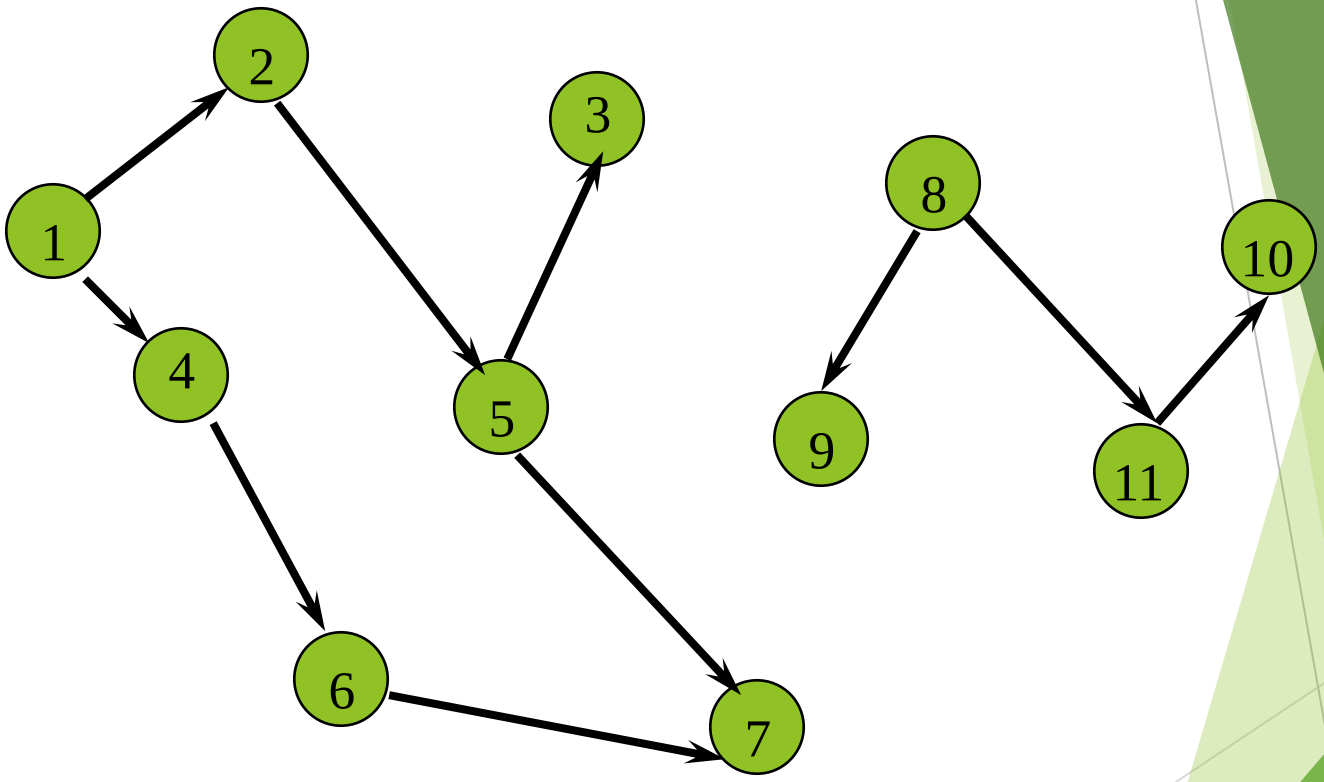
- in-degree is number of incoming edges
- indegree_____

Out-Degree



- out-degree is number of outbound edges
- outdegree(2) = 1, outdegree(8) = 2

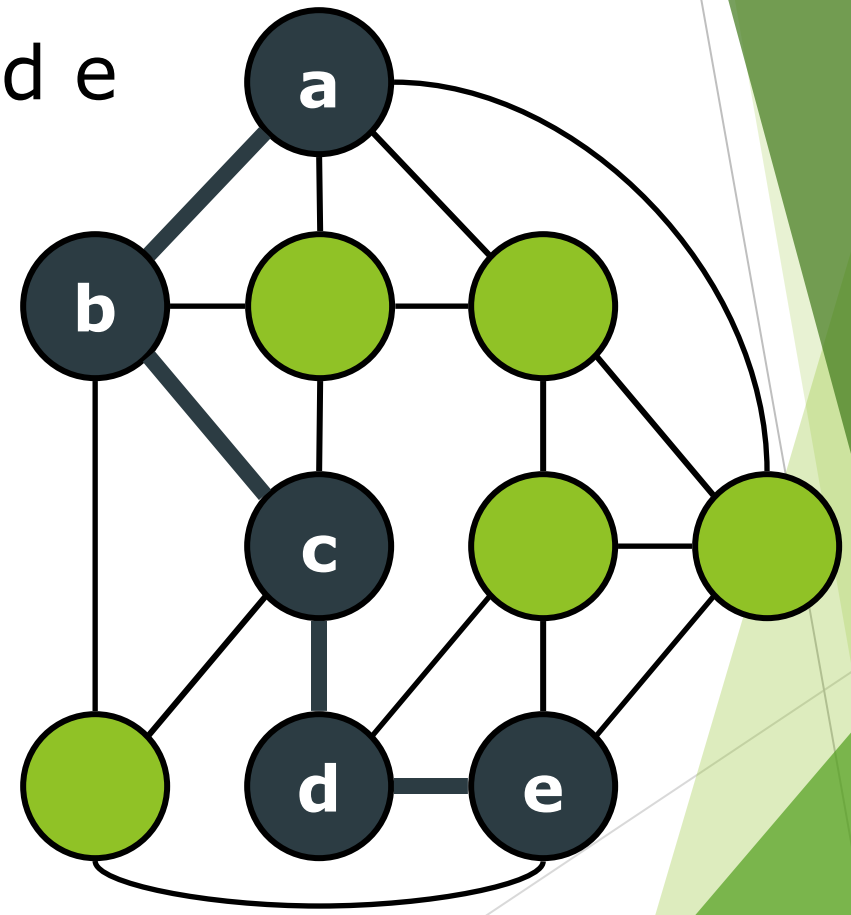
Degree



- Degree (2) = _____
- Degree (5) = _____

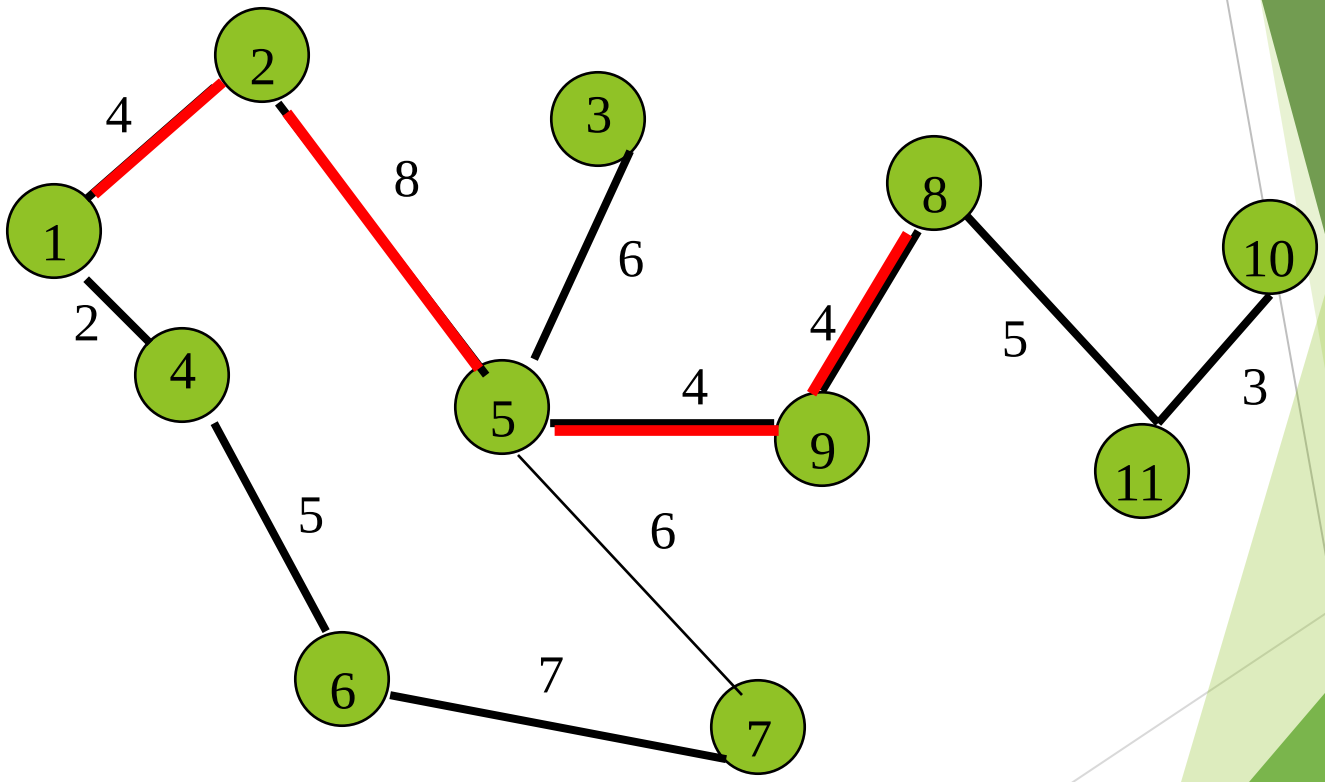
Path

- Path between a and e
- Length =



Path Finding

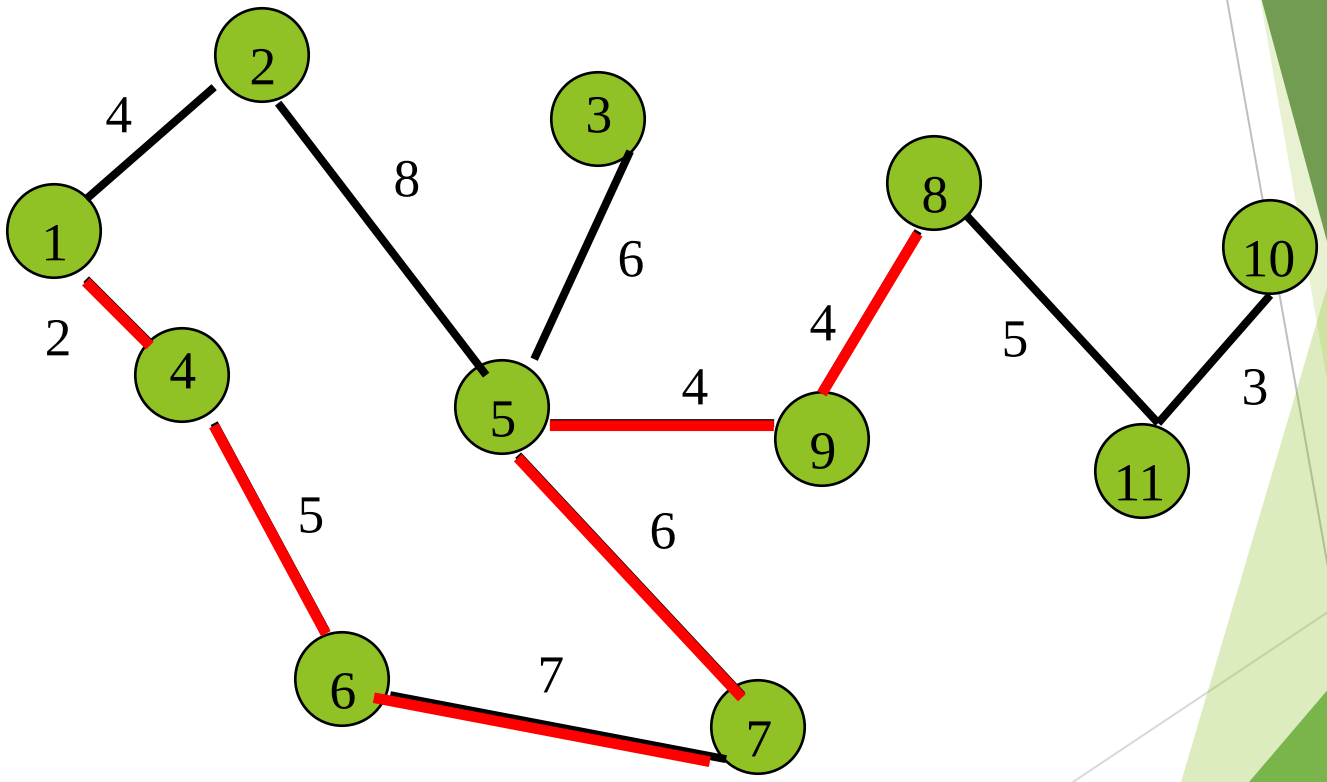
- Path between 1 and 8.



Path length

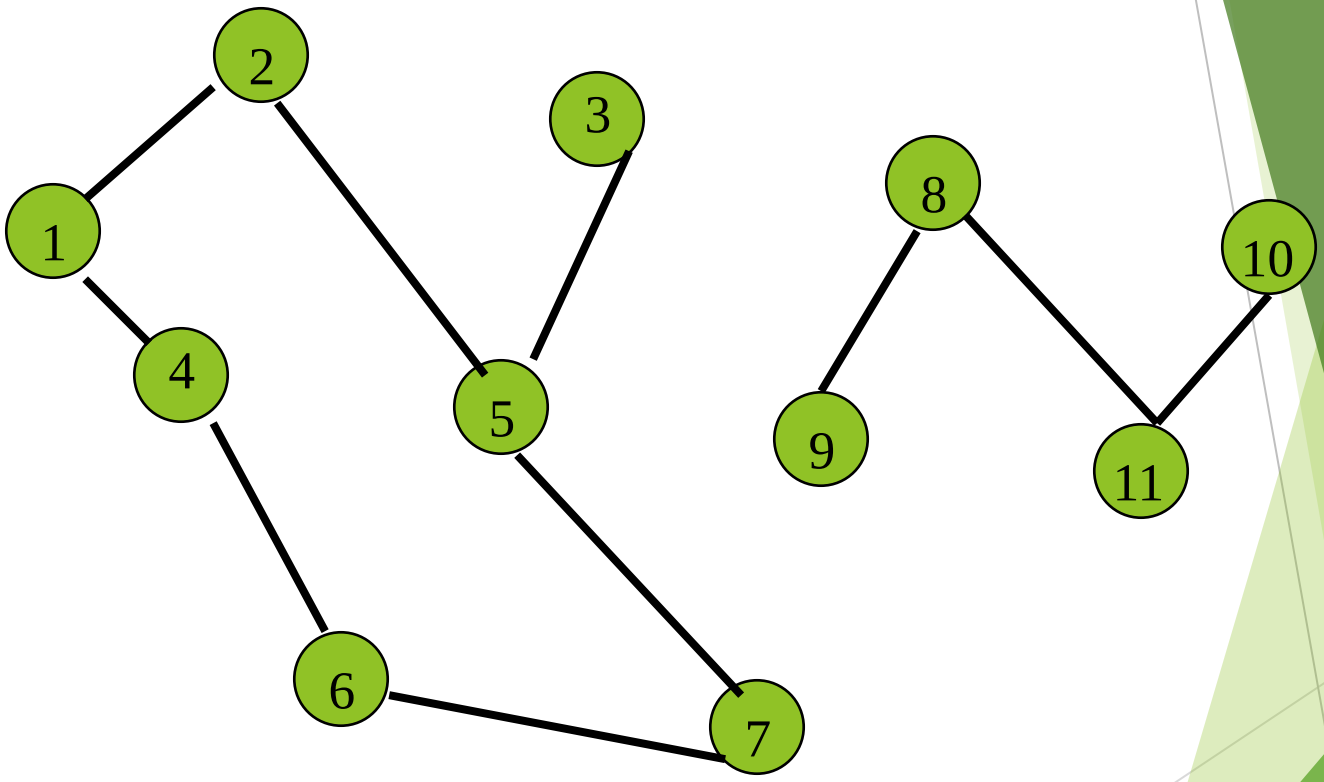
Path Finding

- Path between 1 and 8.



Path length is _____

Path Finding

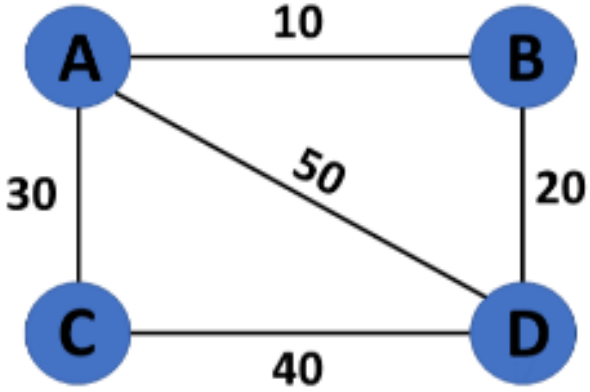


No path between 2 and 9.

Weighted Graph

Weighted Graph

- A graph G= (V, E) is called a labeled or weighted graph because each edge has a value or weight representing the cost of traversing that edge.



https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure

Formally

A weighted graph G = _____ where

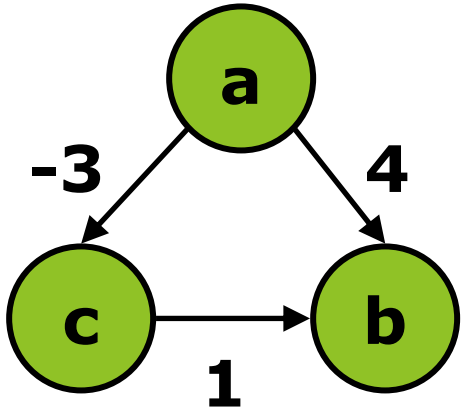
- V is the set of vertices
- E is the set of edges
- W is the weight function

Example

V = { a, b, c }

E = { (a,b), _____ }

W = { ((a,b), 4), _____ }



Adjacent Vertices

- adj(v) = set of vertices adjacent to v

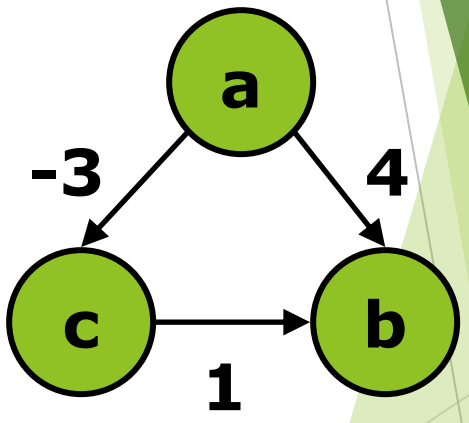
  adj(a) = {_____

  adj(b) = {_____

  adj(c) = {_____

- $\sum_v |adj(v)| = |E|$

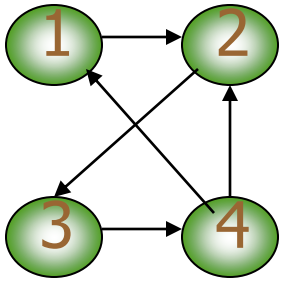
- adj(v): Neighbours of v



Graph Representation

1. Adjacency Matrix
2. Incidence Matrix
3. Adjacency Lists
    1. Array Adjacency Lists
    2. Linked Adjacency Lists

Adjacency Matrix

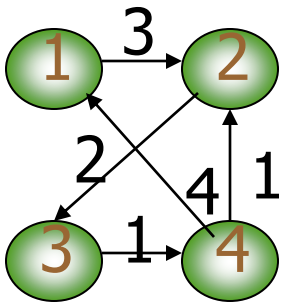


graph

	1	2	3	4
1		T		
2			T	
3				T
4	T	T		

OR

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



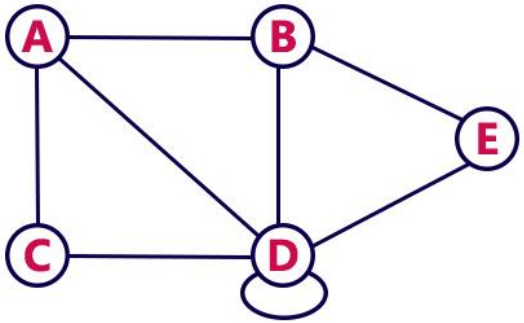
graph

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

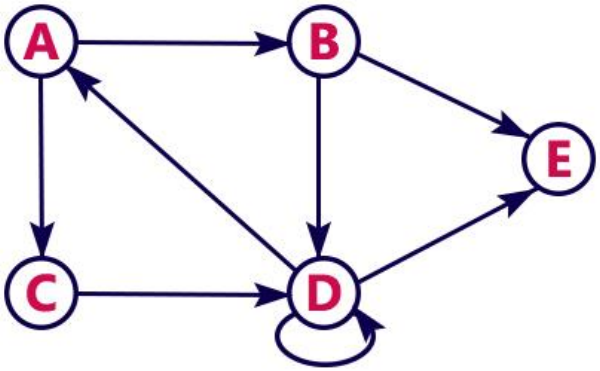
Ideal view

Computerized view

Adjacency Matrix



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0



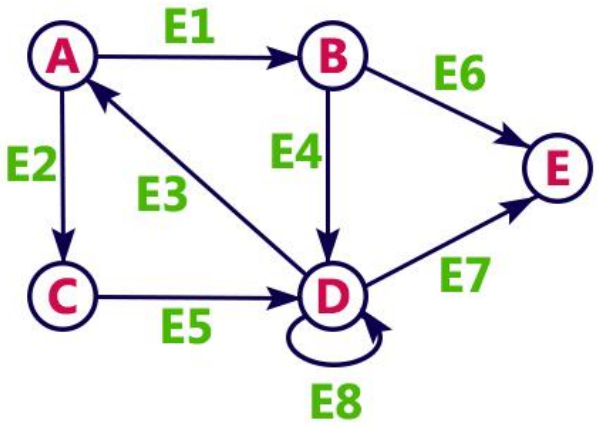
	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

http://btechsmartclass.com/DS/U3_T9.html

Incidence Matrix

- graph can be represented using a matrix of size total number of vertices by total number of edges.
- That means if a graph with 4 vertices and 6 edges can be represented using a matrix of 4X6 class.
- In this matrix, rows represents vertices and columns represents edges.
  - This matrix is filled with either 0 or 1 or -1.
  - 0 represents row edge is not connected to column vertex,
  - 1 represents row edge is connected as outgoing edge to column vertex
  - -1 represents row edge is connected as incoming edge to column vertex.

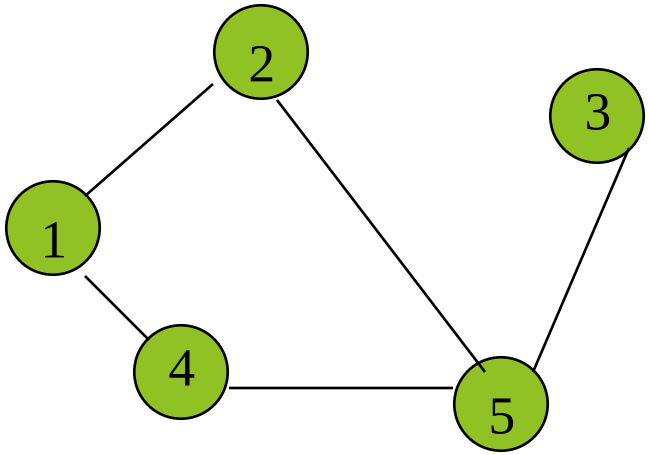
Incidence Matrix



	E1	E2	E3	E4	E5	E6	E7	E8
A	1	1	-1	0	0	0	0	0
B	-1	0	0	1	0	1	0	0
C	0	-1	0	0	1	0	0	0
D	0	0	1	-1	-1	0	1	1
E	0	0	0	0	0	-1	-1	0

Array Adjacency Lists

- Adjacency list for vertex i is a linear list of vertices adjacent from vertex i.
- An array of n adjacency lists.



aList[1] = _____

aList[2] = _____

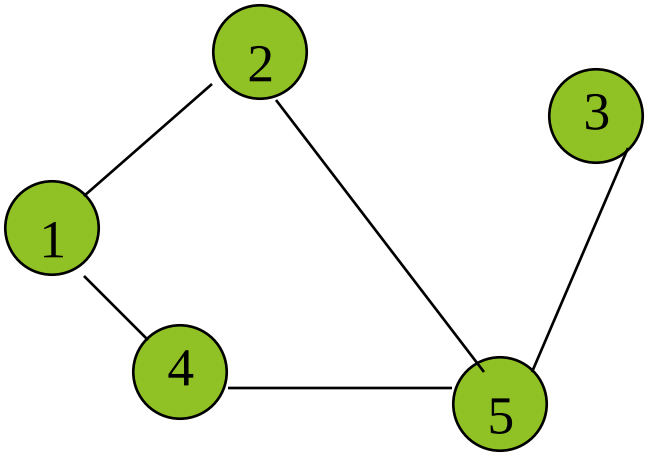
aList[3] = _____

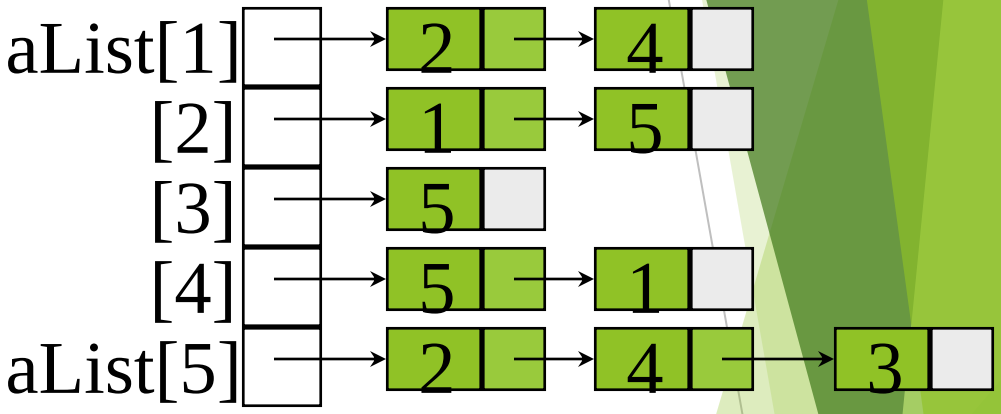
aList[4] = _____

aList[5] = _____

Linked Adjacency Lists

- Each adjacency list is a chain.





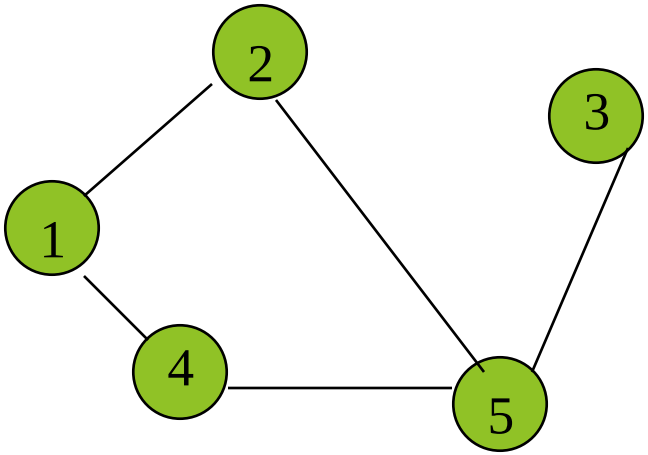
Array Length = n

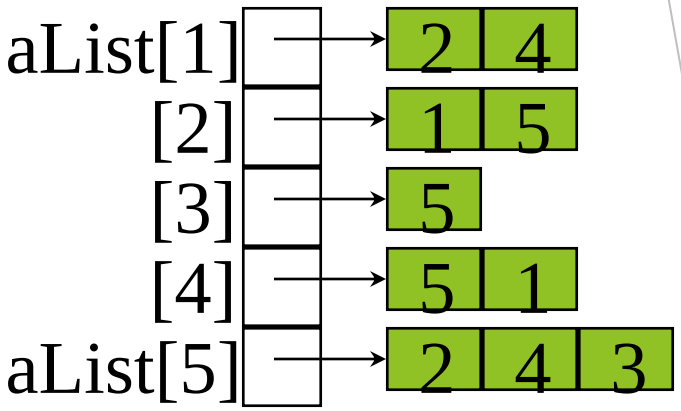
# of chain nodes = 2e (undirected graph)

# of chain nodes = e (digraph)

Array Adjacency Lists

- Each adjacency list is an array list.





Array Length = n

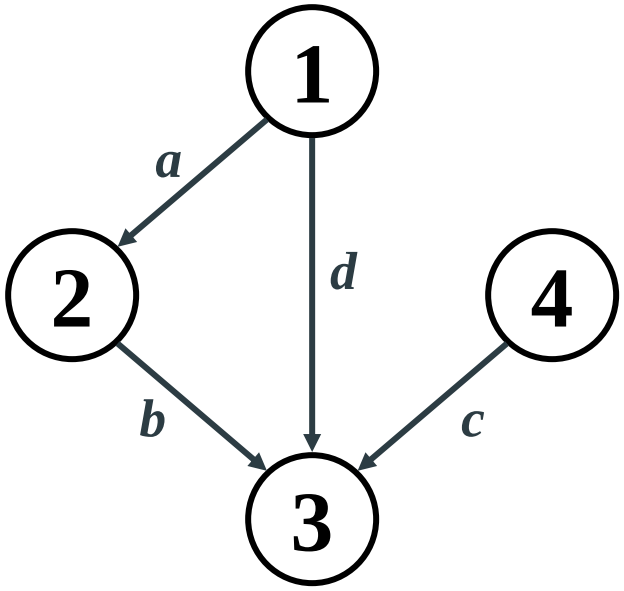
# of list elements = 2e (undirected graph)

# of list elements = e (digraph)

Question?

Graphs: Adjacency Matrix

- Example:

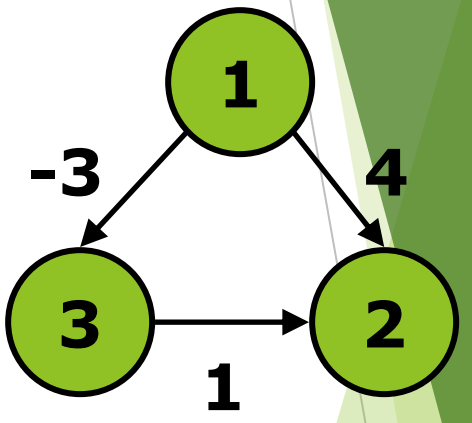


A	1	2	3	4
1				
2				
3			??	
4				

Adjacency Matrix (weight)

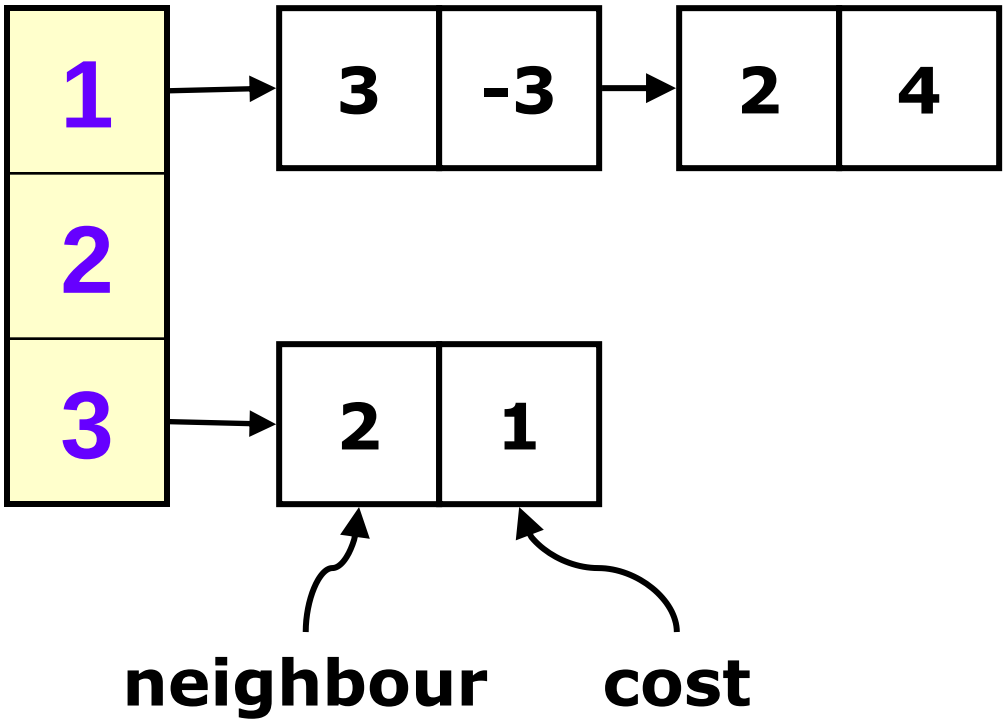
double vertex[][];

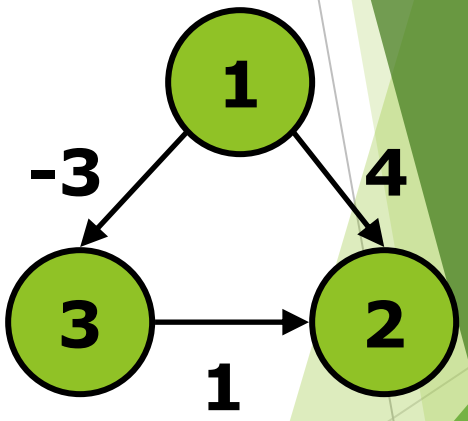
	1	2	3
1			
2			
3			



Adjacency List (weight)

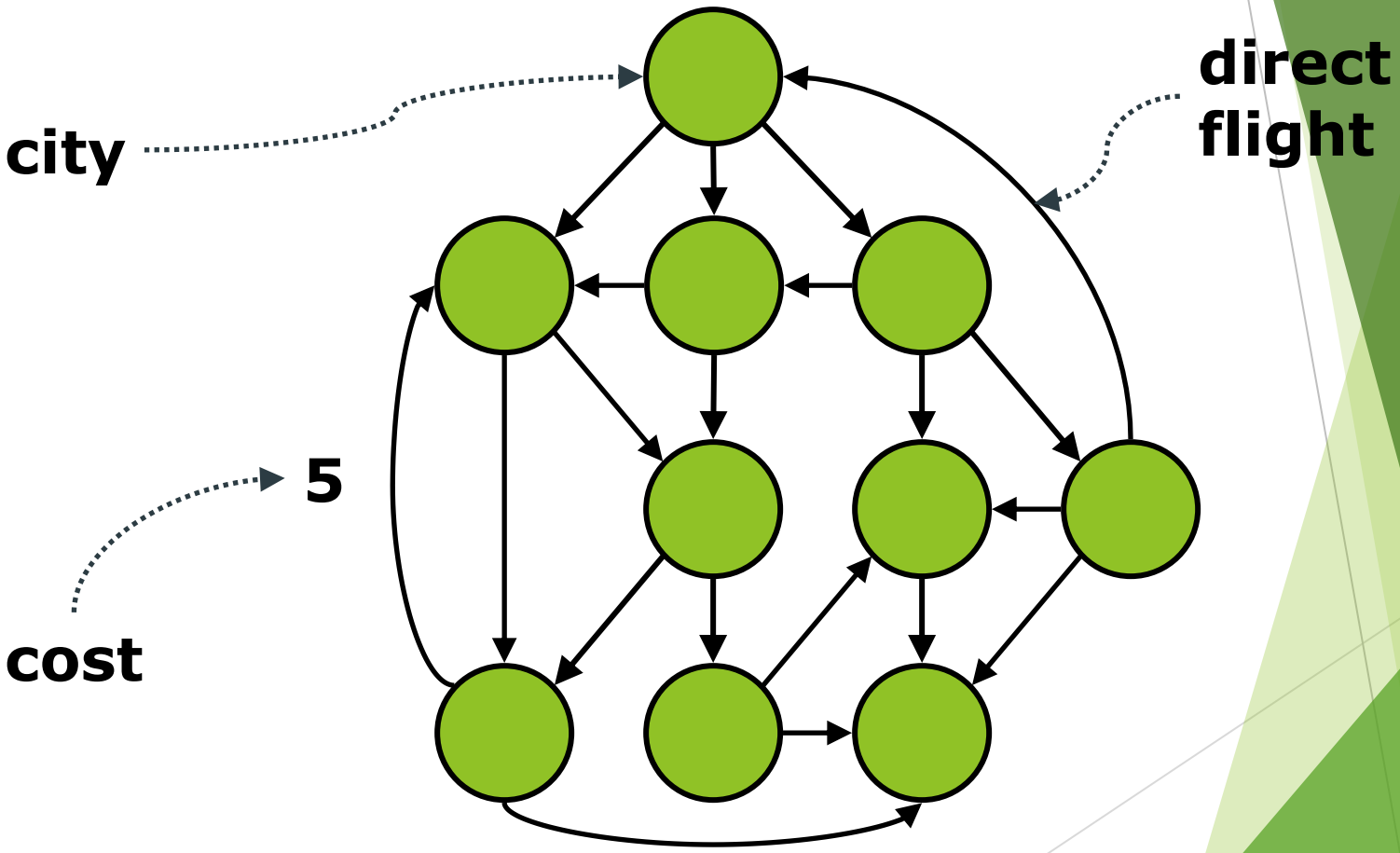
EdgeList vertex[];





Applications

Travel Planning



Question

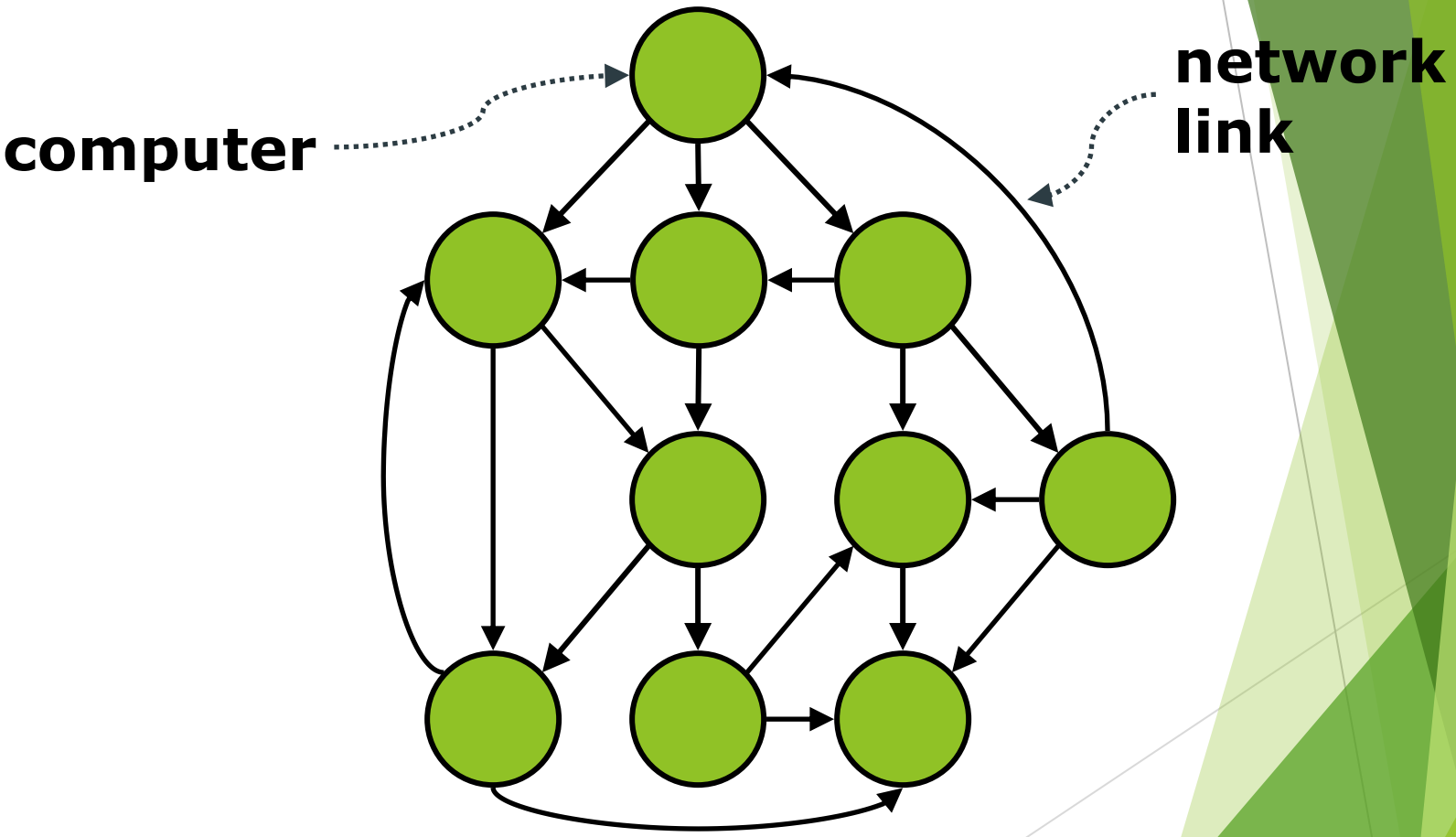
- What is the shortest way to travel between A and B?

  “SHORTEST PATH PROBLEM”

- How to mimimize the cost of visiting n cities such that we visit each city exactly once, and finishing at the city where we start from?

  “TRAVELING SALESMAN PROBLEM”

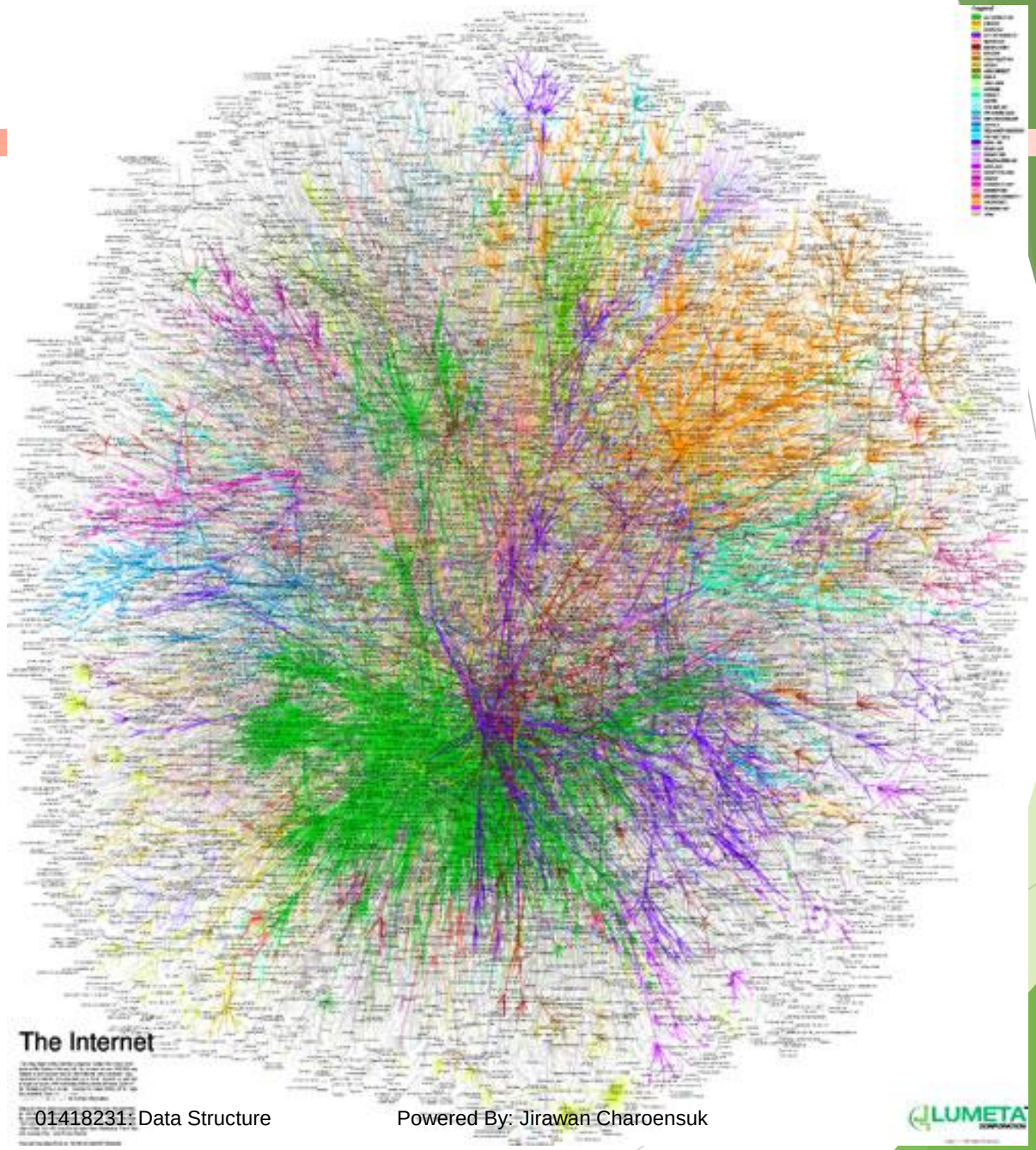
Internet



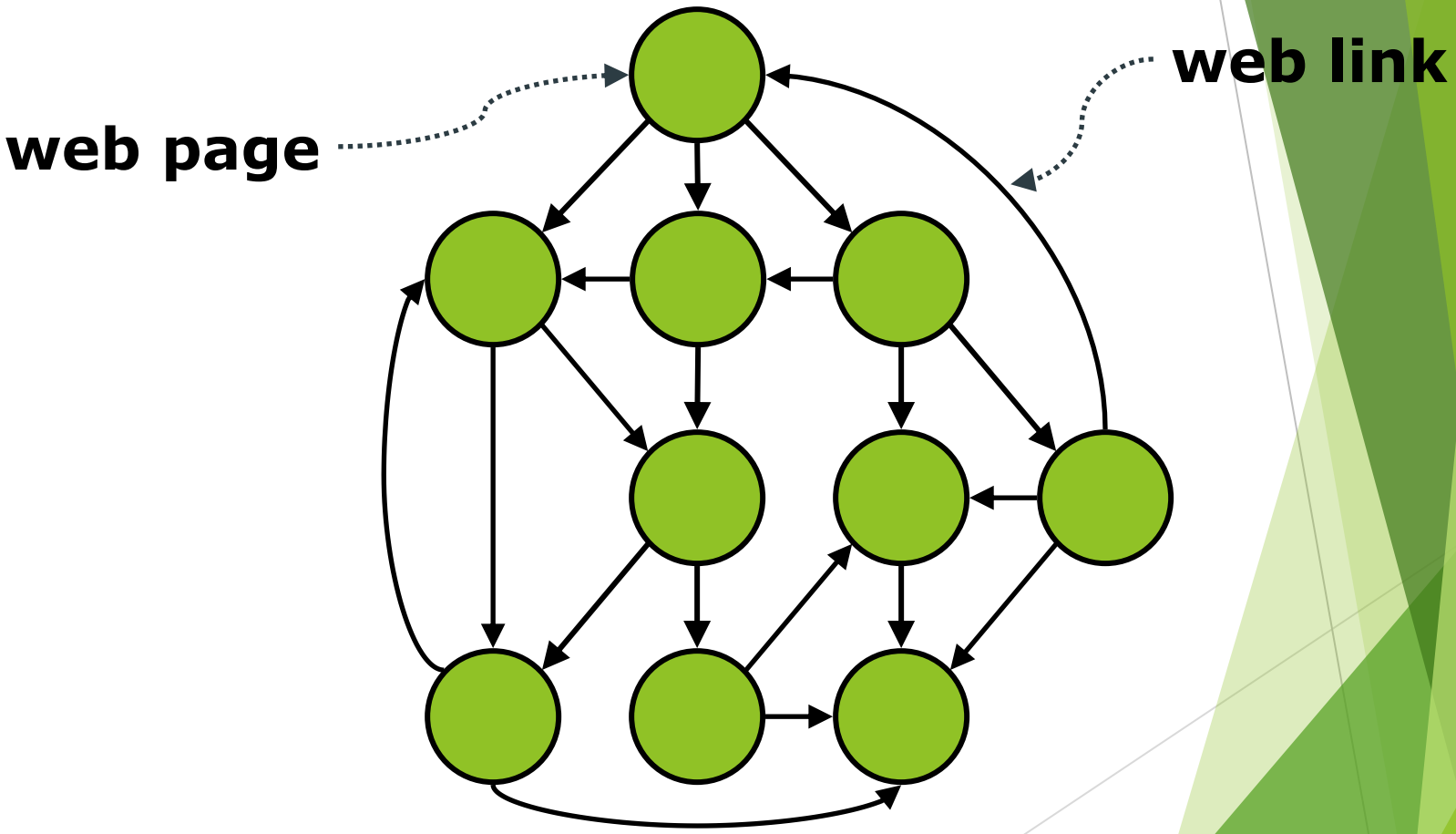
Question

- What is the shortest route to send a packet from A to B?

"SHORTEST PATH PROBLEM"


The Internet

The Web

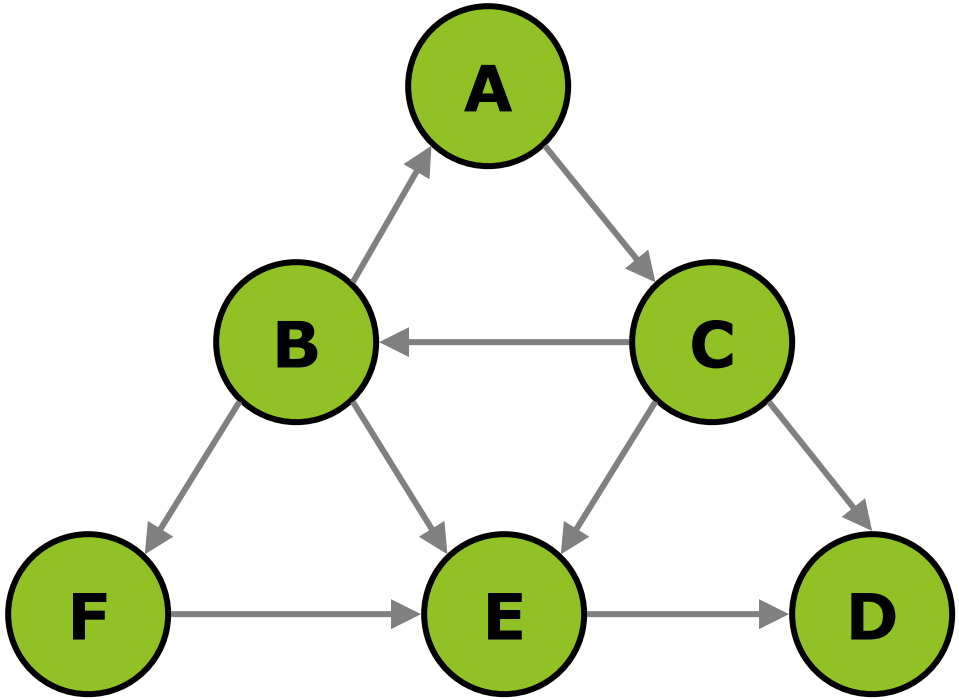


Question

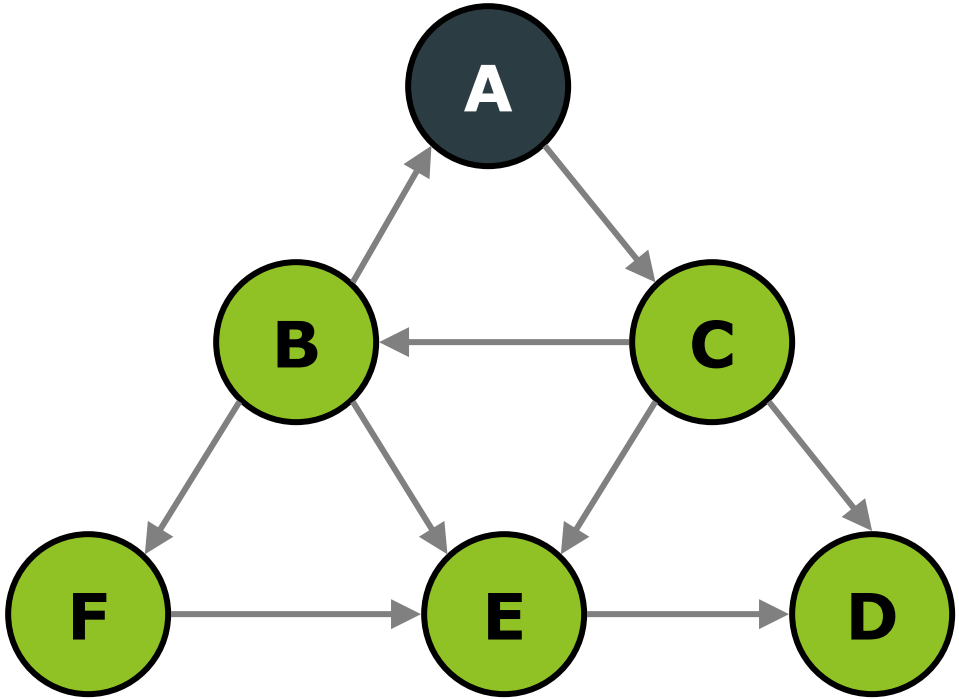
- Which web pages are important?
- Which set of web pages are likely to be of the same topic?

Breadth-First Search

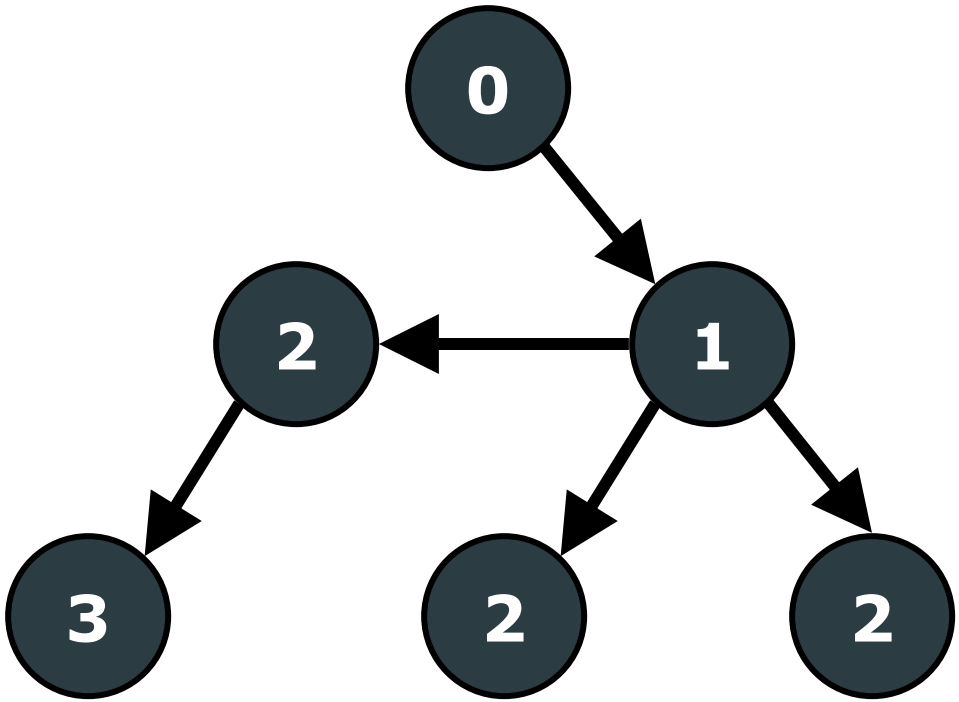
Breadth-First Search



Breadth-First Search

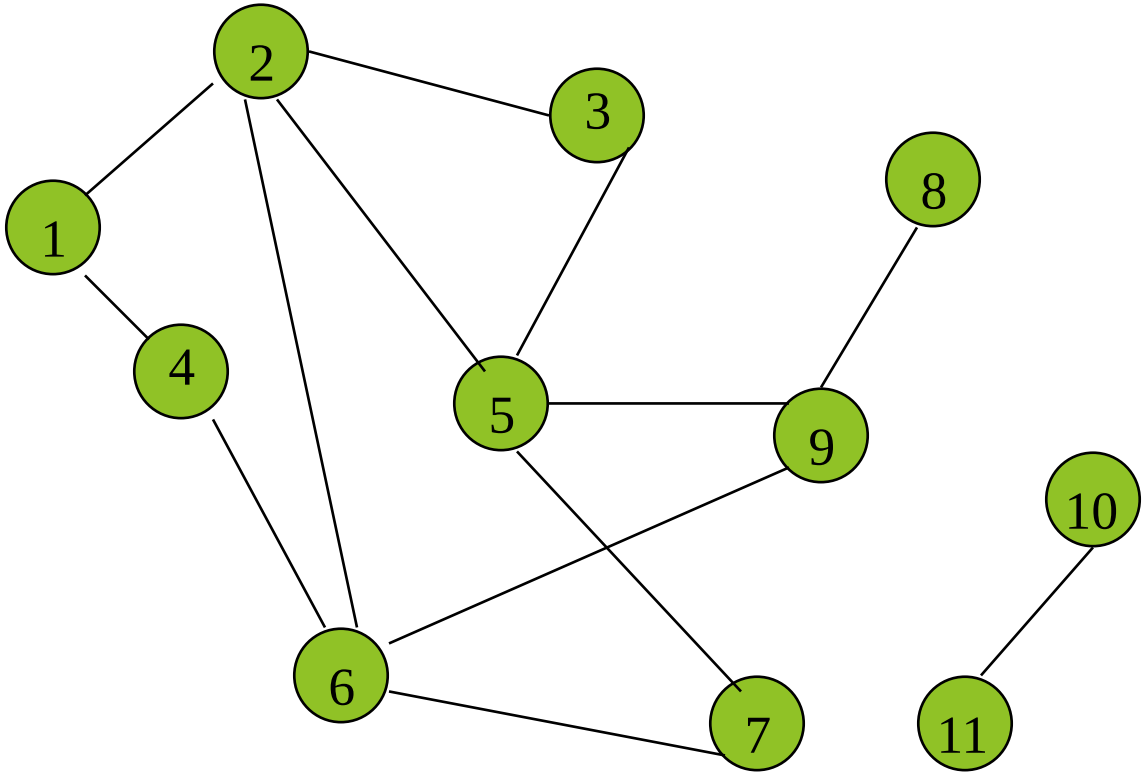


Breadth-First Search



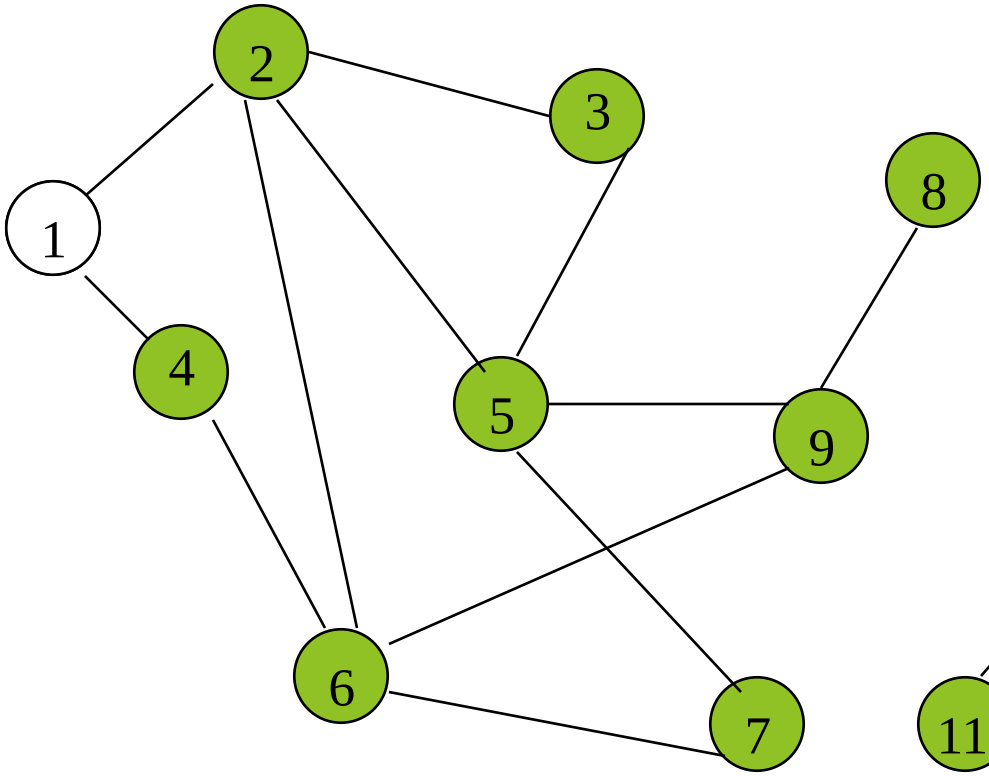
Breadth-First Search Example 2

Breadth-First Search



Start search at vertex 1.

Breadth-First Search

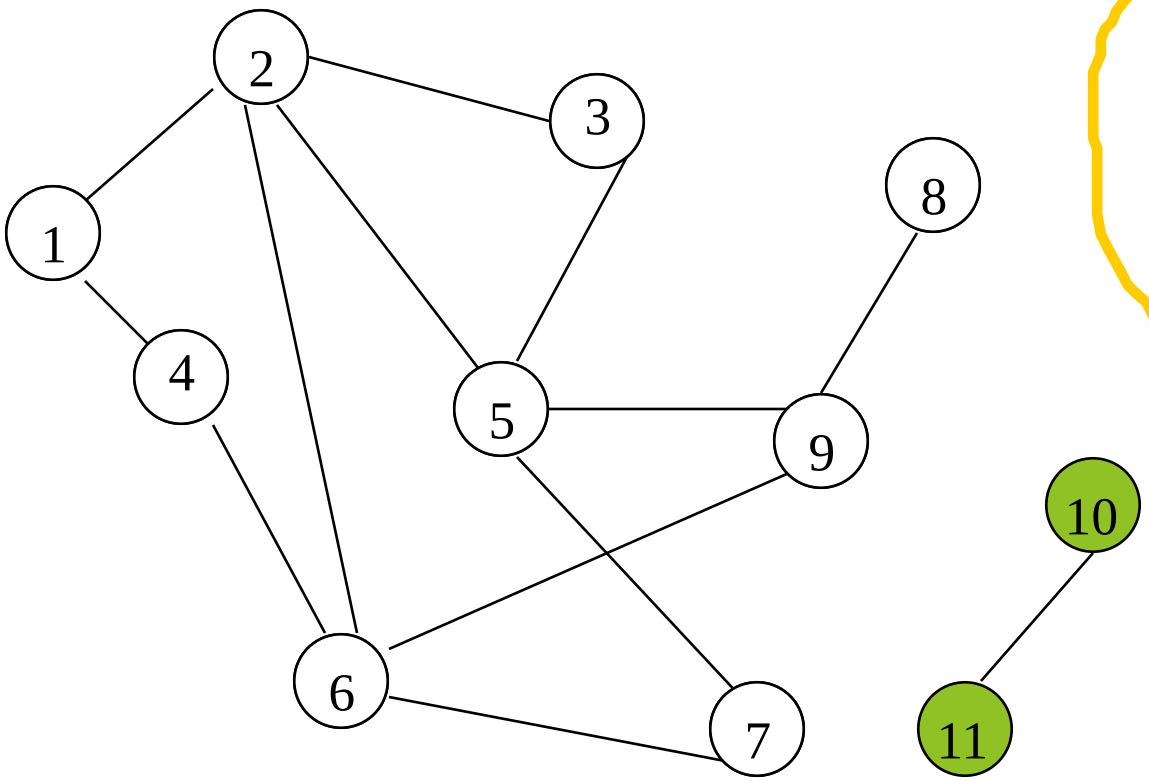


FIFO Queue

1

Mark/label start vertex and put in a FIFO queue.

Breadth-First Search



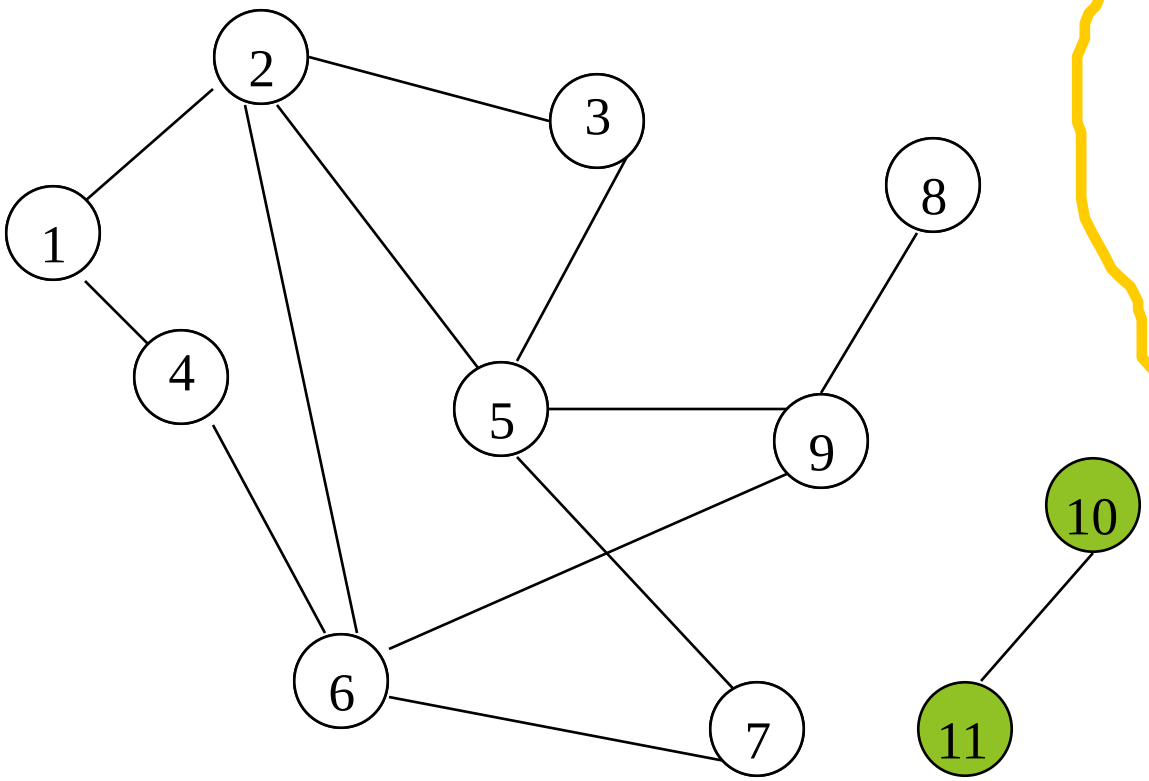
FIFO Queue

8

Remove 8 from Q; visit adjacent unvisited vertices;

put in Q.

Breadth-First Search



FIFO Queue

Queue is empty. Search terminates.

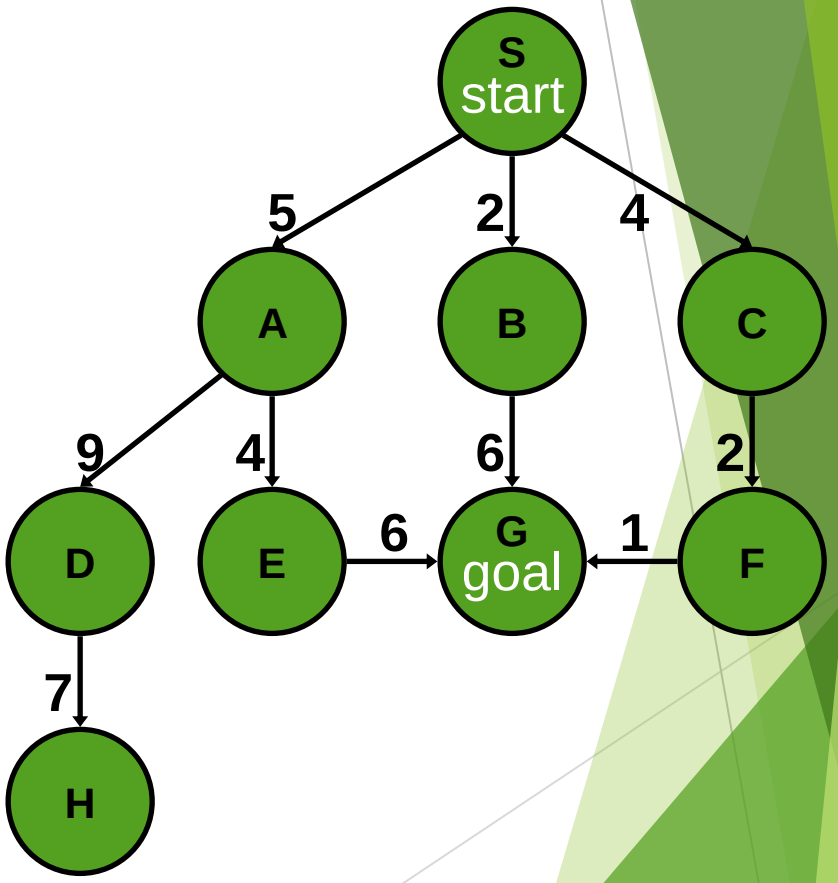
Breadth-First Search Example 3

Breadth-First Search (BFS)

```
generalSearch(problem, queue)
```

# of nodes tested: 0, expanded: 0

current	nodes list
	{S}

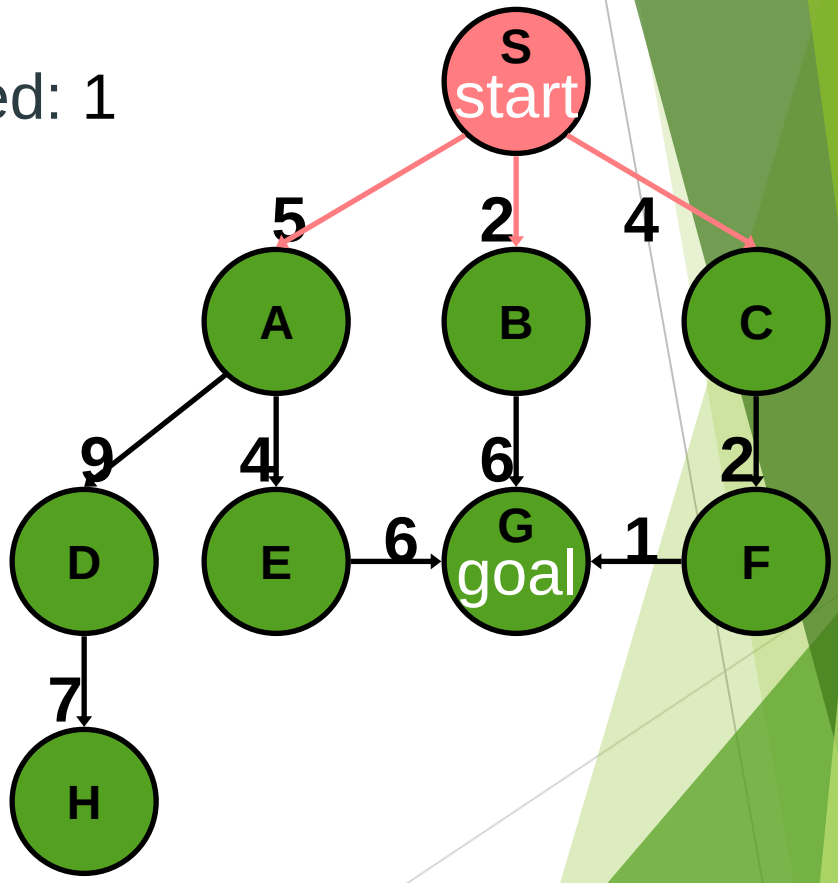


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 1, expanded: 1

current	nodes list
	{S}
S not goal	**{A,B,C}**

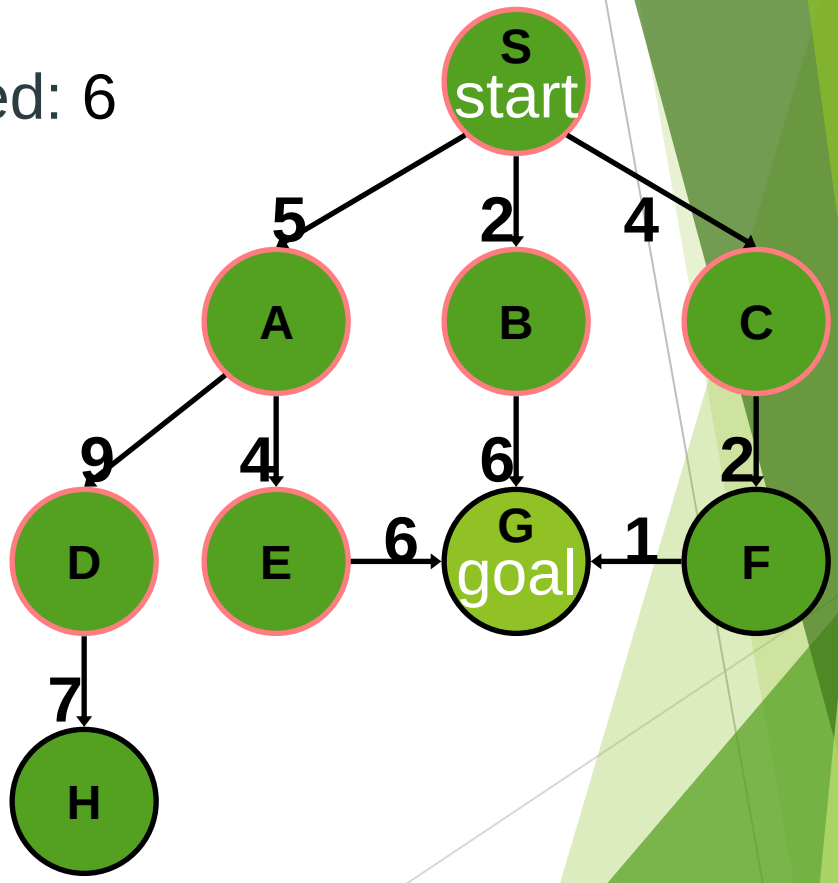


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

current	nodes list
	{S}
S	**{A,B,C}**
A	**{B,C,D,E}**
B	**{C,D,E,G}**
C	**{D,E,G,F}**
D	**{E,G,F,H}**
E	**{G,F,H}**
G goal	**{F,H}** **no expand**

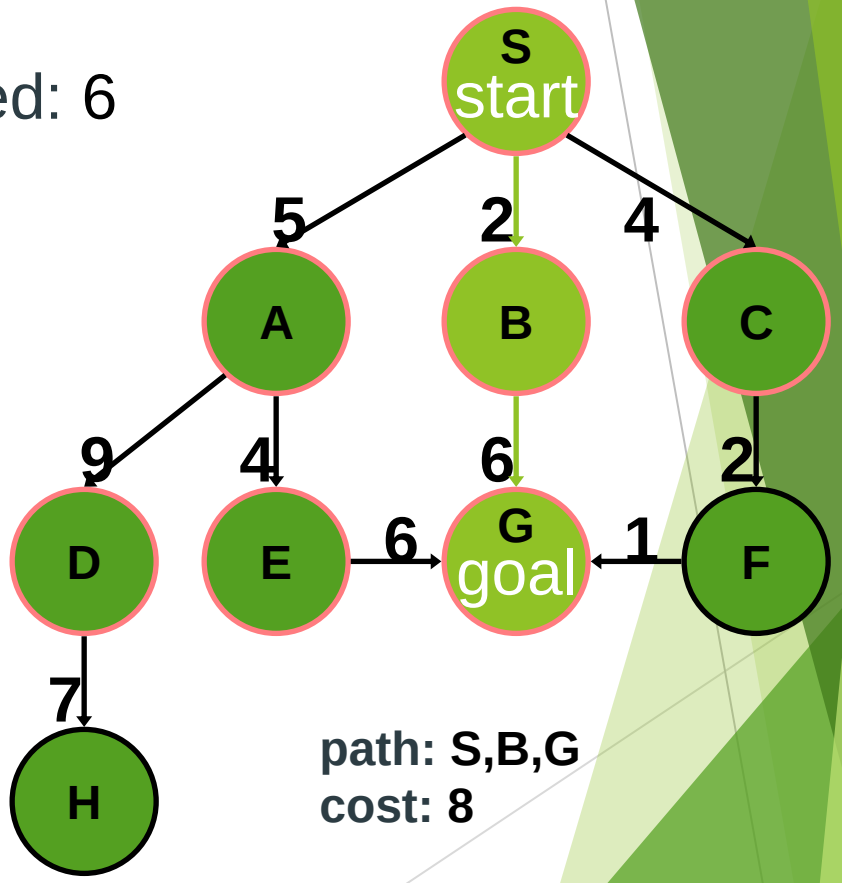


Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

current	nodes list
	{S}
S	**{A,B,C}**
A	**{B,C,D,E}**
B	**{C,D,E,G}**
C	**{D,E,G,F}**
D	**{E,G,F,H}**
E	**{G,F,H,G}**
G	**{F,H,G}**



Breadth-First Search

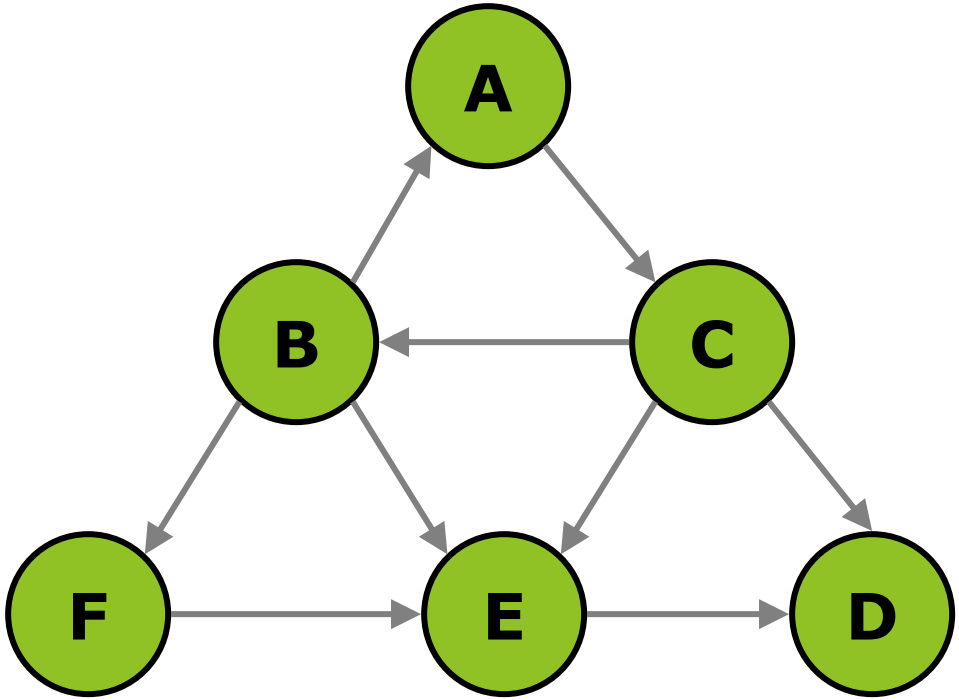
- "Explore" a graph, turning it into a tree
  - One vertex at a time
  - Expand frontier of explored vertices across the *breadth* of the frontier
- Builds a tree over the graph
  - Pick a *source vertex* to be the root
  - Find ("discover") its children, then their children, etc.

Breadth-First Search

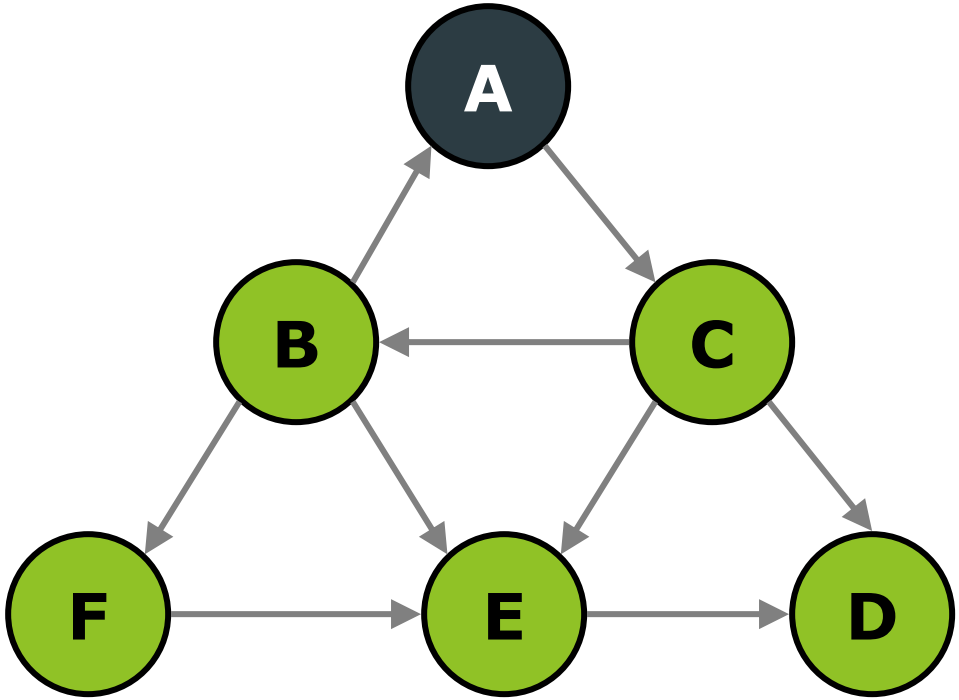
- Visit start vertex and put into a FIFO queue.
- Repeatedly remove a vertex from the queue, visit its unvisited adjacent vertices, put newly visited vertices into the queue.

Depth-First Search

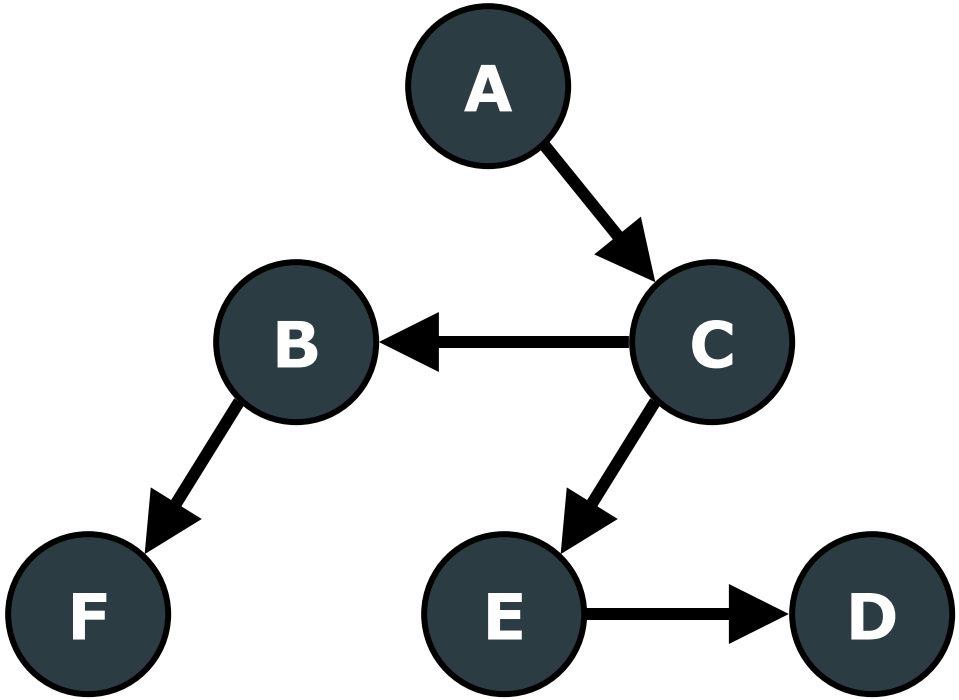
Depth-First Search



Depth-First Search

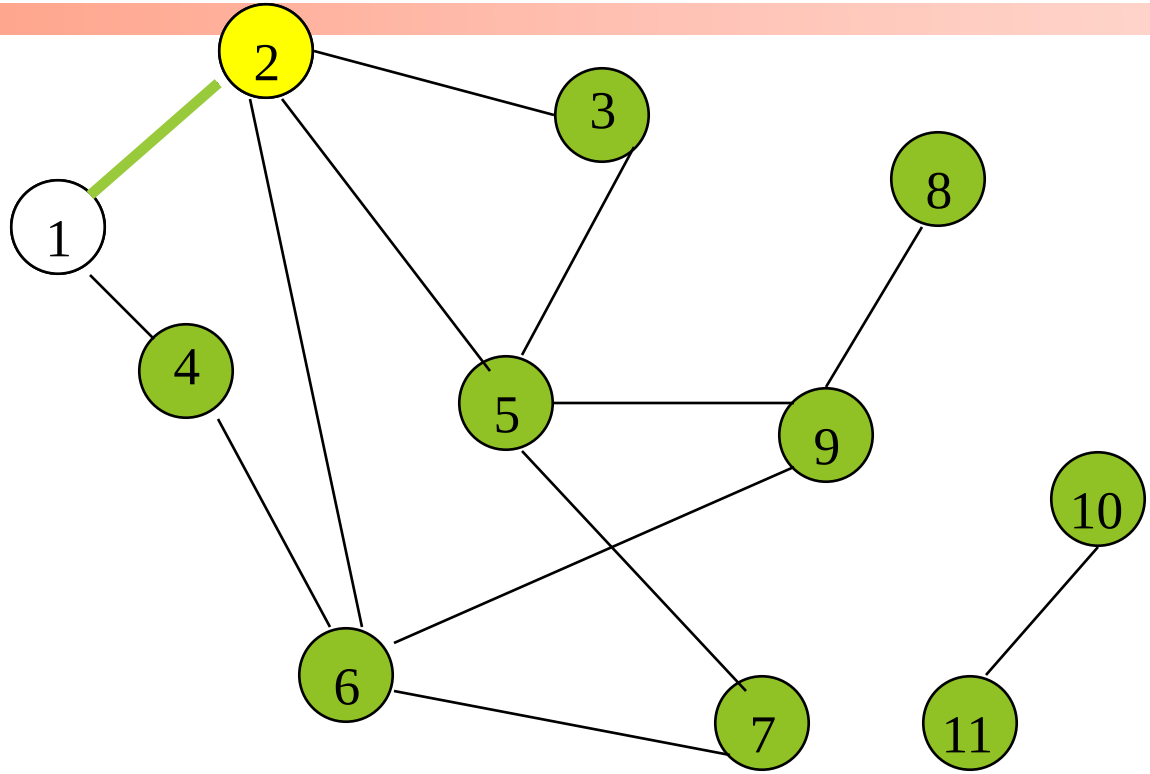


Depth-First Search



Depth-First Search Example 2

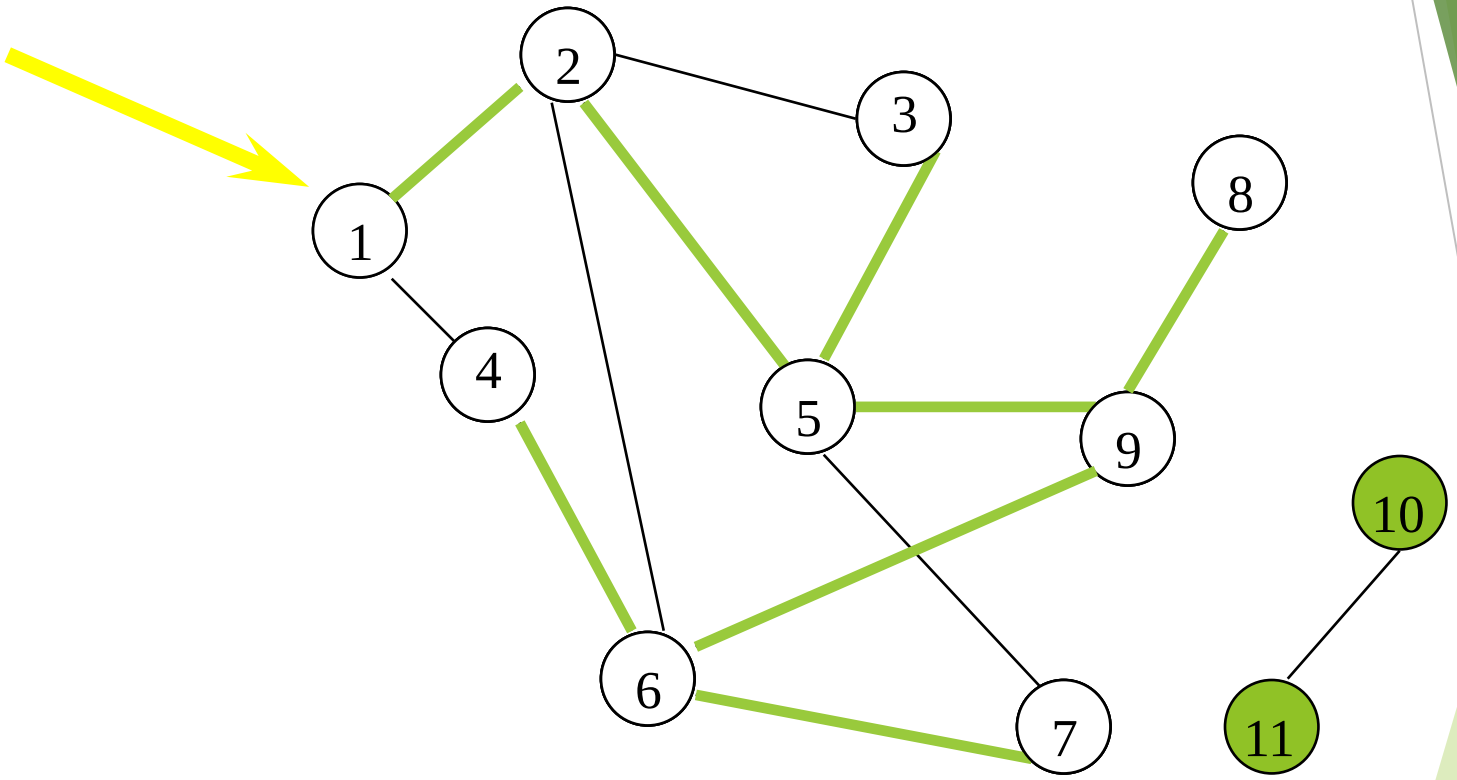
Depth-First Search



Start search at vertex 1.

Label vertex 1 and do a depth first search from either 2 or 4.

Depth-First Search



Return to invoking method.

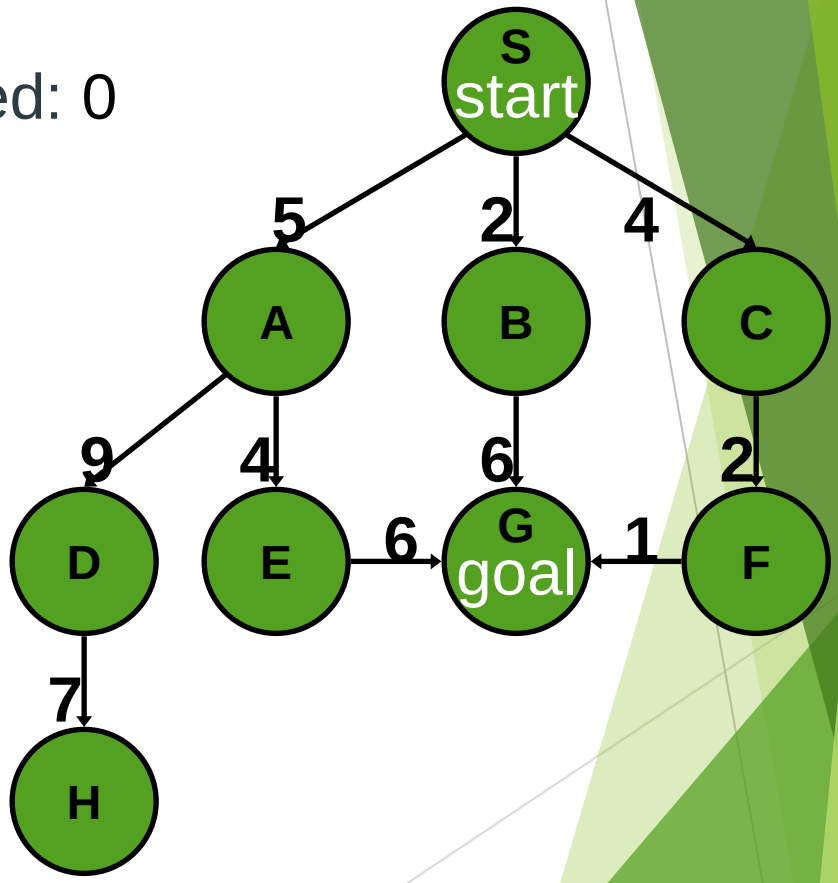
Depth-First Search Example 3

Depth-First Search (DFS)

```
generalSearch(problem, stack)
```

# of nodes tested: 0, expanded: 0

current	nodes list
	{S}

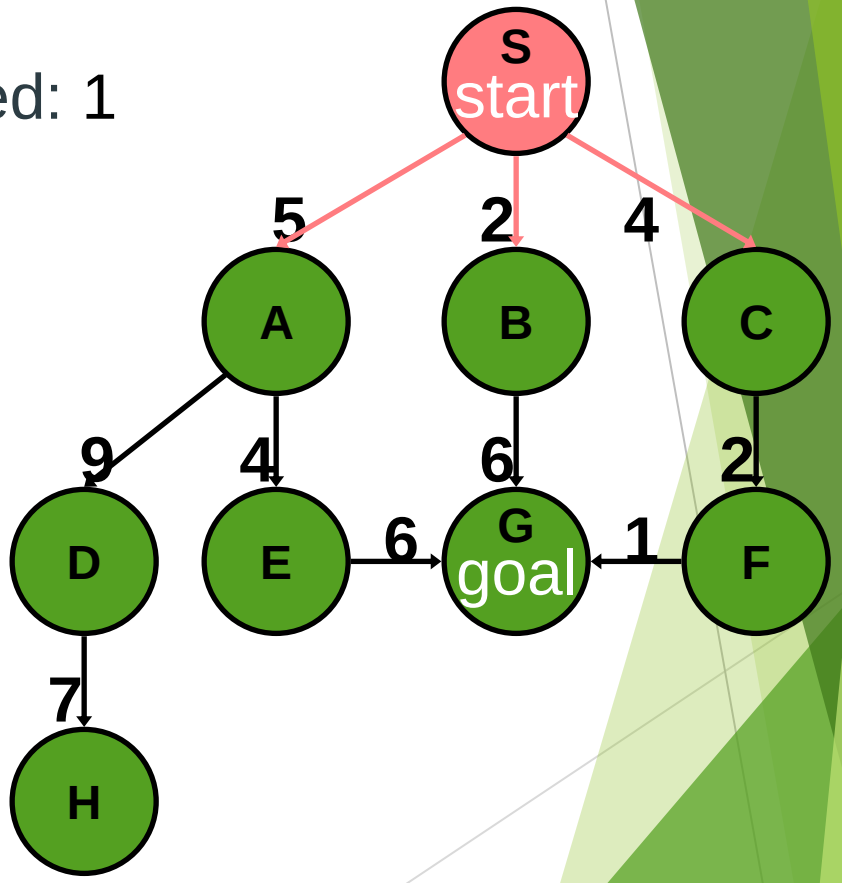


Depth-First Search (DFS)

`generalSearch(problem, stack)`

# of nodes tested: 1, expanded: 1

current	nodes list
	{S}
S not goal	**{A,B,C}**

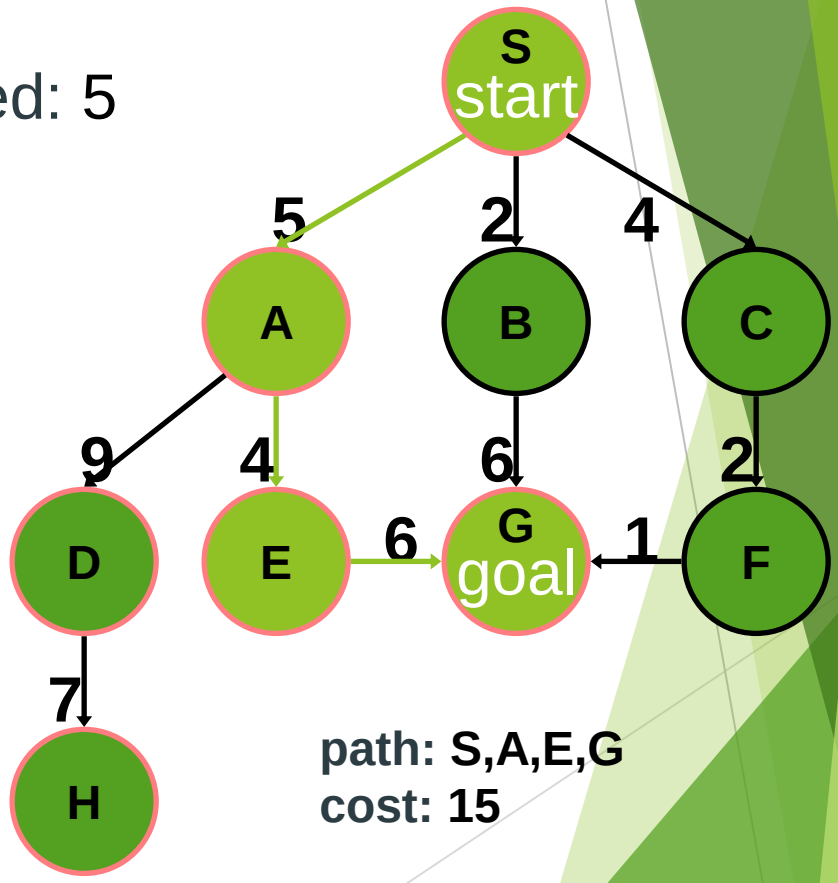


Depth-First Search (DFS)

`generalSearch(problem, stack)`

# of nodes tested: 6, expanded: 5

current	nodes list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G	{B,C}



Summary

Breadth-First Search

- Visit start vertex and put into a FIFO queue.
- Repeatedly remove a vertex from the queue, visit its unvisited adjacent vertices, put newly visited vertices into the queue.

Depth-First Search

- A depth-first search (DFS) in an undirected graph G is like wandering in a labyrinth with a string and following one path to the end
- We then backtrack by rolling up our string until we get back to a previously visited vertex *v*.
- *v* becomes our current vertex and we repeat the previous steps

Question