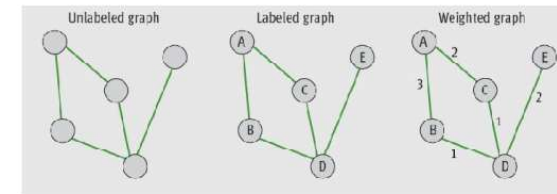# Graph Data Structure

1

Introduction

- Mathematically, a graph is a set V of vertices and a set E of edges, such that each edge in E connects two of the vertices in V
- We use node as a synonym for vertex

Unlabeled graph   Labeled graph   Weighted graph

2

Terminology

- One vertex is **adjacent** to another vertex if there is an edge connecting the two vertices (neighbors)
- **Path**: Sequence of edges that allows one vertex to be reached from another vertex in a graph
- **Length of a path**: Number of edges on the path
- **Degree of a vertex**: Number of edges connected to it

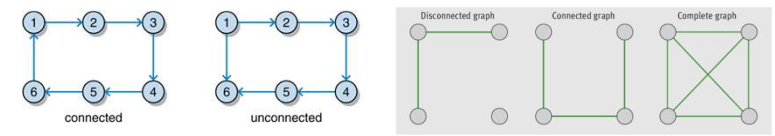Labeled graph   Weighted graph
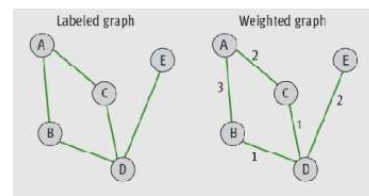
3

Terminology

- **Simple path**: Path that does not pass through the same vertex more than once
- **Cycle**: Path that begins and ends at the same vertex

Simple path: ABC   Cycle: BCD
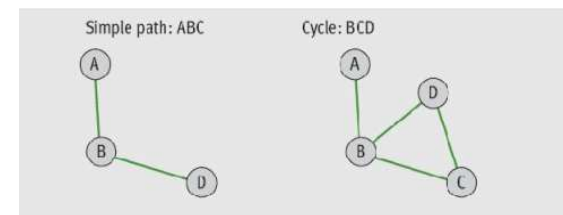
4

Terminology

- A graph is **connected** if there is a path from each vertex to every other vertex
- A graph is **complete** if there is an edge from each vertex to every other vertex

5

Graph Terminology

- Graphs can be undirected or directed (**digraph**)
- A directed edge has a source vertex and a destination vertex
- Edges emanating from a given source vertex are called its incident edges
- To convert undirected graph to equivalent directed graph, replace each edge in undirected graph with a pair of edges pointing in opposite directions
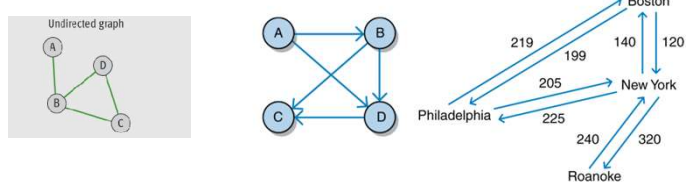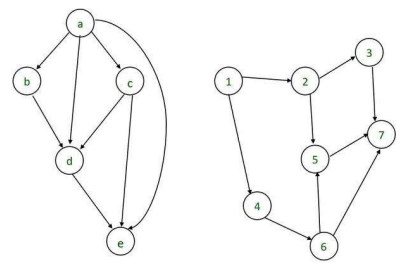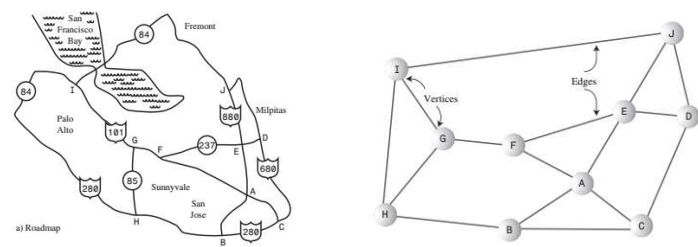

6

Terminology

- A special case of digraph (directed graph) that contains no cycles is known as a **directed acyclic graph**, or **DAG**
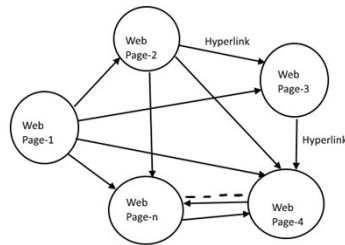

7

Graph Applications – Example - Roadmap

8

Graph Applications – Example – Web Links

9

Graph Applications

- Graphs serve as models of a wide range of objects:
  - A roadmap
  - A map of airline routes
  - A schematic of the computers and connections that make up the Internet
  - The links between pages on the Web
  - The relationship between students and courses i.e courses and their pre-requisites
  - A diagram of the flow capacities in a communications or transportation network

10

Representing Graphs

- To represent graphs, you need a convenient way to store the vertices and the edges that connect them
- Two commonly used representations of graphs:
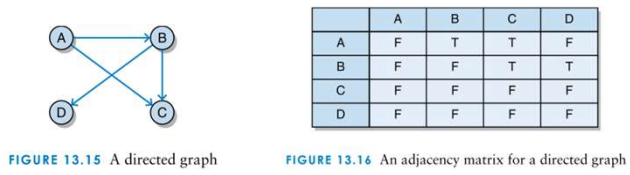 - The adjacency matrix
  - The adjacency list

11

Adjacency Matrix

- If a graph has N vertices labeled 0, 1, . . . , N – 1:
- The adjacency matrix for the graph is a grid G with N rows and N columns
- Cell G[i][j] = 1 (or F) if there's an edge from vertex i to j
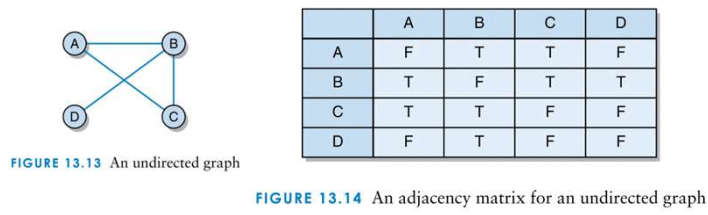- Otherwise, there is no edge and that cell contains 0

12

Adjacency Matrix – A Directed Graph

FIGURE 13.15 A directed graph

	A	B	C	D
A	F	T	T	F
B	F	F	T	T
C	F	F	F	F
D	F	F	F	F

FIGURE 13.16 An adjacency matrix for a directed graph

Cell G[i][j] = T if there's an edge from vertex i to j
Cell G[i][j] = F if there's no edge from vertex i to j

1-13
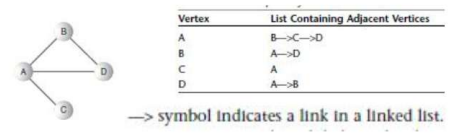
13

Adjacency Matrix - An Undirected Graph

FIGURE 13.13 An undirected graph

	A	B	C	D
A	F	T	T	F
B	T	F	T	T
C	T	T	F	F
D	F	T	F	F

FIGURE 13.14 An adjacency matrix for an undirected graph

Cell G[i][j] = T if there's an edge from vertex i to j
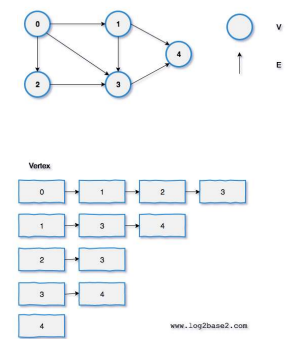Cell G[i][j] = F if there's no edge from vertex i to j

1-14

14

Adjacency List

- If a graph has N vertices labeled 0, 1, . . . , N – 1,
- Each individual list shows what vertices a given vertex is adjacent to.
- Adjacency list shows which vertices are adjacent to—that is, one edge away from—a given vertex, not paths from vertex to vertex.

Vertex	List Containing Adjacent Vertices
A	B—>C—>D
B	A—>D
C	A
D	A—>B

—> symbol indicates a link in a linked list.

15

15

Adjacency List – Directed Graph

16

16

4

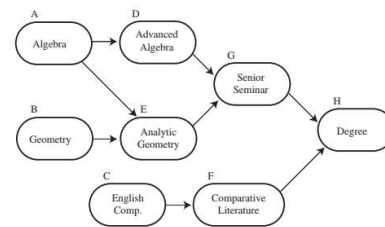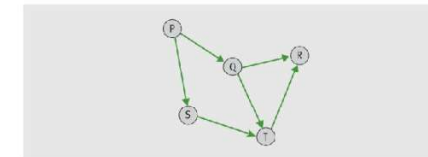## Topological Sort

- A directed acyclic graph (DAG) has an order among the vertices
- A **topological order** assigns a rank to each vertex such that the edges go from lower- to higher-ranked vertices
- **Topological sort**: process of finding and returning a topological order of vertices in a graph
- **Topological sort algorithm** → Does topological sort.

Example: Course Pre-requisites

17

Topological Sort Example

[FIGURE 20.14] A graph of courses

[FIGURE 20.15] The first topological ordering of the graph

[FIGURE 20.16] The second topological ordering of the graph

18

Graph Traversals

- In addition to the insertion and removal of items, important graph-processing operations include: – Traversing all of the items in a graph
- One starts at a given vertex and, from there, visits all vertices to which it connects
- Different from tree traversals, which always visit all of the nodes in a given tree
- There are two algorithms (breadth first traversal/search and depth first search/algorithm)

19

Graph Traversal

- A depth-first search (DFS), uses a stack
- A breadth-first search (BFS), uses a queue
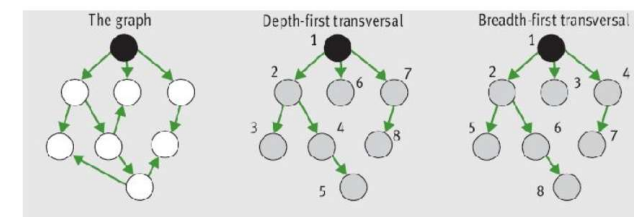
20

Depth First Search

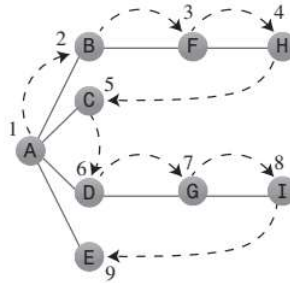
Depth First Search

- To carry out the depth-first search, you pick a starting point—in this case, vertex A.
- You then do three things: visit this vertex, push it onto a stack so you can remember it, and mark it so you won't visit it again.
- Next, you go to any vertex adjacent to A that hasn't yet been visited. We'll assume the vertices are selected in alphabetical order, so that brings up B. You visit B, mark it, and push it on the stack.
- Now what? You're at B, and you do the same thing as before: go to an adjacent vertex that hasn't been visited. This leads you to F. We can call this process Rule 1.

Depth First Search

- **RULE 1**: If possible, visit an adjacent unvisited vertex, mark it, and push it on the stack.
- Applying Rule 1 again leads you to H. At this point, however, you need to do something else because there are no unvisited vertices adjacent to H. Here's where Rule 2 comes in.
- **RULE 2**: If you can't follow Rule 1, then, if possible, pop a vertex off the stack.

Depth First Search

- Following this rule, you pop H off the stack, which brings you back to F. F has no unvisited adjacent vertices, so you pop it. Ditto B. Now only A is left on the stack.
- A, however, does have unvisited adjacent vertices, so you visit the next one, C. But C is the end of the line again, so you pop it and you're back to A. You visit D, G, and I, and then pop them all when you reach the dead end at I. Now you're back to A. You visit E, and again you're back to A.
- This time, however, A has no unvisited neighbors, so we pop it off the stack. But now there's nothing left to pop, which brings up Rule 3.
- RULE 3: If you can't follow Rule 1 or Rule 2, you're done
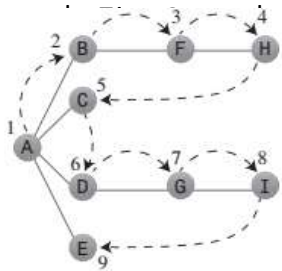
Depth First Search

Event	Stack
Visit A	A
Visit B	AB
Visit F	ABF
Visit H	ABFH
Pop H	ABF
Pop F	AB
Pop B	A
Visit C	AC
Pop C	A
Visit D	AD
Visit G	ADG
Visit I	ADGI
Pop I	ADG
Pop G	AD
Pop D	A
Visit E	AE
Pop E	A
Pop A	
Done	

- **RULE 1**: If possible, visit an adjacent unvisited vertex, mark it, and push it on the stack.

- Applying Rule 1 again leads you to H. At this point, however, you need to do something else because there are no unvisited vertices adjacent to H. Here's where Rule 2 comes in.

- **RULE 2**: If you can't follow Rule 1, then, if possible, pop a vertex off the stack.

- **RULE 3**: If you can't follow Rule 1 or Rule 2, you're done

25

Depth First Search Python Code

- https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm

26

Breadth First Search

- In the breadth-first search, on the other hand, the algorithm likes to stay as close as possible to the starting point.

- It visits all the vertices adjacent to the starting vertex, and only then goes further afield. This kind of search is implemented using a queue instead of a stack.
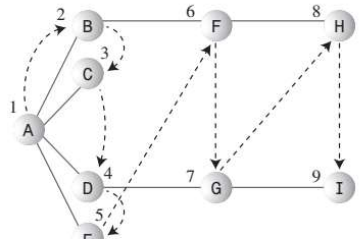
27

BFS

RULE 1

Visit the next unvisited vertex (if there is one) that's adjacent to the current vertex, mark it, and insert it into the queue.

RULE 2

If you can't carry out Rule 1 because there are no more unvisited vertices, remove a vertex from the queue (if possible) and make it the current vertex.

RULE 3

If you can't carry out Rule 2 because the queue is empty, you're done.
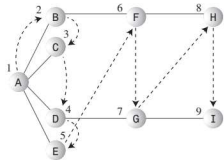
28

Breadth First Search

- Thus, you first visit all the vertices adjacent to A, inserting each one into the queue as you visit it. Now you've visited A, B, C, D, and E. At this point the queue (from front to rear) contains BCDE.

- There are no more unvisited vertices adjacent to A, so you remove B from the queue and look for vertices adjacent to it. You find F, so you insert it in the queue. There are no more unvisited vertices adjacent to B, so you remove C from the queue. It has no adjacent unvisited vertices, so you remove D and visit G. D has no more adjacent unvisited vertices, so you remove E. Now the queue is FG. You remove F and visit H, and then you remove G and visit I.

- Now the queue is HI, but when you've removed each of these and found no adjacent unvisited vertices, the queue is empty, so you're done. Table below shows this sequence.

29

Breadth First Search

RULE 1

Visit the next unvisited vertex (if there is one) that's adjacent to the current vertex, mark it, and insert it into the queue.

RULE 2

If you can't carry out Rule 1 because there are no more unvisited vertices, remove a vertex from the queue (if possible) and make it the current vertex.

RULE 3

If you can't carry out Rule 2 because the queue is empty, you're done.

Event	Queue (Front to Rear)
Visit A	
Visit B	B
Visit C	BC
Visit D	BCD
Visit E	BCDE
Remove B	CDE
Visit F	CDEF
Remove C	DEF
Remove D	EF
Visit G	EFG
Remove E	FG
Remove F	G
Visit H	GH
Remove G	H
Visit I	HI
Remove H	I
Remove I	
Done	

30

Breadth First Search Python Code

- https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm#

31

Minimum Spanning Tree

- In a weighted graph, you can sum the weights for all edges in a spanning tree and attempt to find a spanning tree that minimizes this sum

- **There are several algorithms for finding a minimum spanning tree for a component. Please check our reference book.**

- For example, to determine how an airline can service all cities, while minimizing the total length of the routes it needs to support

32

8
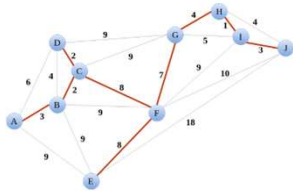
Dijkstra's Algorithm

- This is an algorithm used to find the **shortest distance from one start node to one other specific node.**
- It works by choosing the next vertex to be the closest unvisited vertex from the starting point.
- Graph data structure is useful in doing this.
- This is applied to weighted graphs.
- Protocols like RIP, OSPF, and BGP use Dijkstra's algorithm to calculate the best path for data transmission between network nodes.
- Digital Mapping Services in Google Maps also make use of this algorithm.
- Check this link: https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php
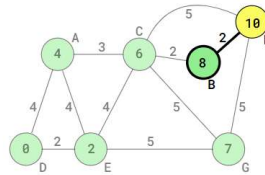
33

Summary

- Graphs have many applications
- A graph consists of one or more vertices (items) connected by one or more edges
- **Path**: Sequence of edges that allows one vertex to be reached from another vertex in the graph
- A graph is connected if there is a path from each vertex to every other vertex
- A graph is complete if there is an edge from each vertex to every other vertex

34

Summary

- A subgraph consists of a subset of a graph's vertices and a subset of its edges
- **Connected component**: a subgraph consisting of the set of vertices reachable from a given vertex
- Directed graphs allow travel along an edge in just one direction
- Edges can be labeled with weights, which indicate the cost of traveling along them
- Graphs have two common implementations: Adjacency matrix and adjacency list

35

Summary

- Graph traversals explore tree-like structures within a graph, starting with a distinguished start vertex e.g., depth-first traversal/search and breadth-first traversal/search
- A minimum spanning tree is a spanning tree whose edges contain the minimum weights possible
- A topological sort generates a sequence of vertices in a directed acyclic graph

36