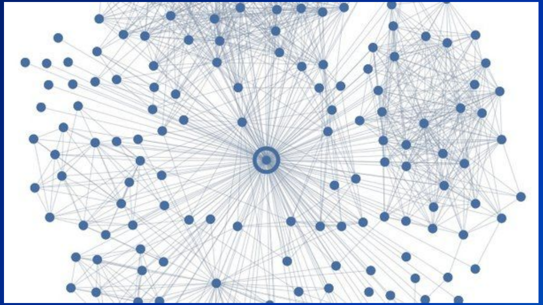
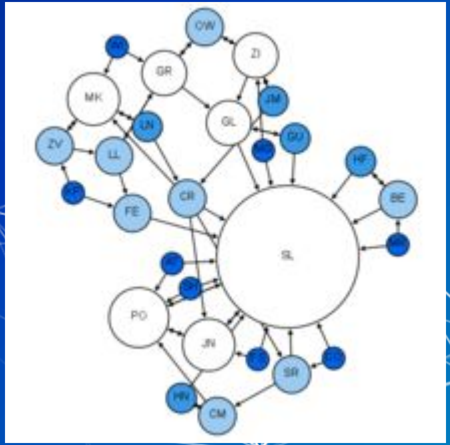
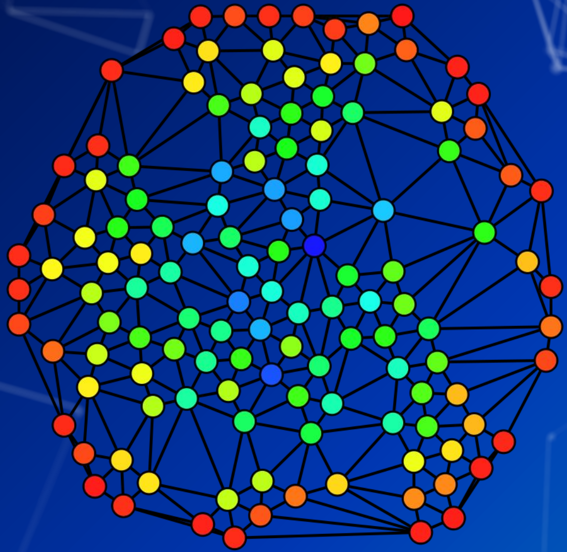
Graph Theory Part 1

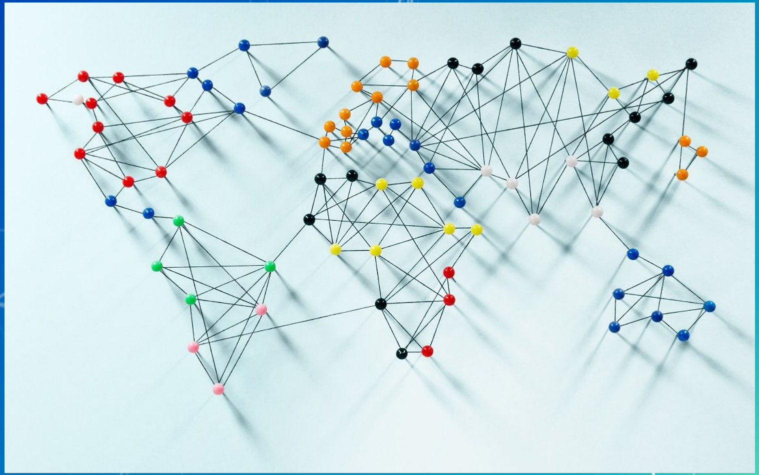
Week 7 AI Inspire Fall 2019



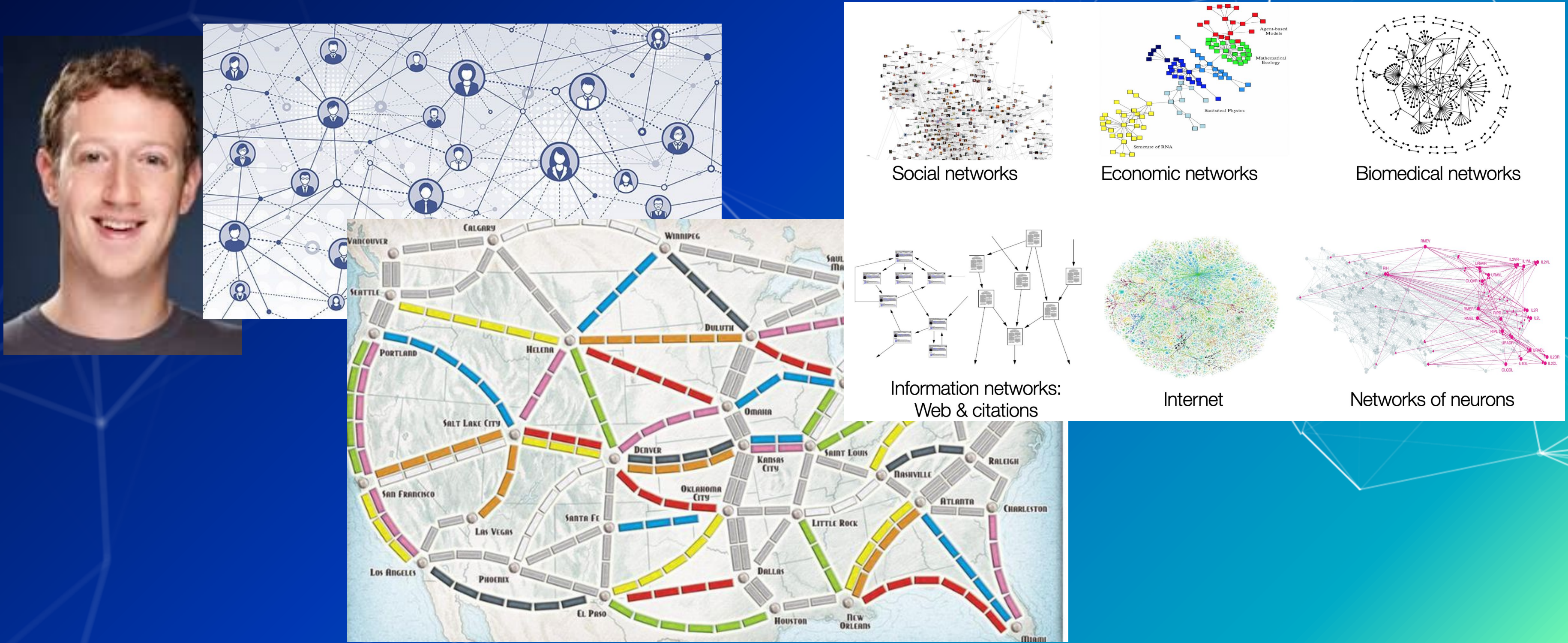
Introduction







Real World Graph Applications



What is a Graph?

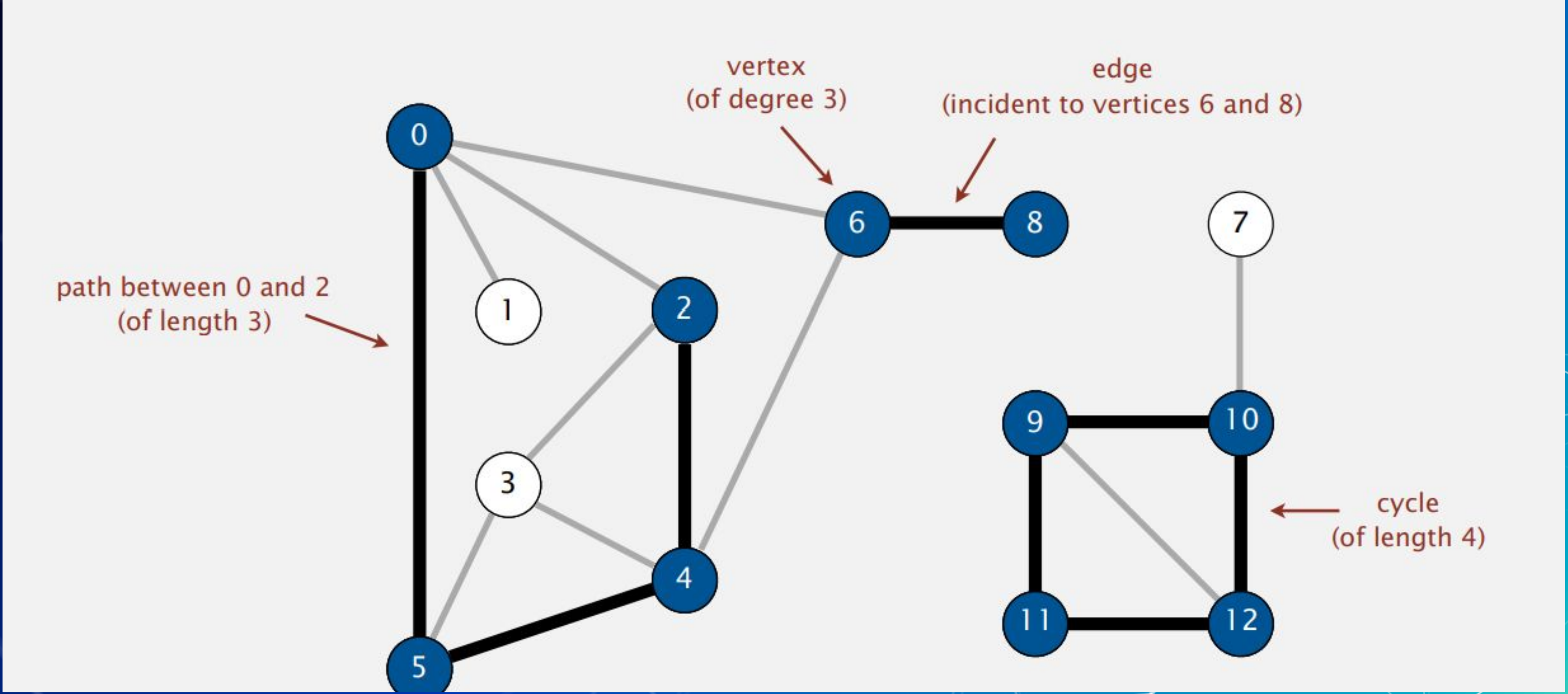
- Type of data structure to store data
  - More sophisticated than other structures learnt so far
- Set of nodes / vertices connected pairwise by edges (edge joins 2 nodes)
  - Node ⇒ stores some type of data
  - Edge ⇒ connection
- Total # vertices = V and total # edges = E

Vocabulary Part 1

- Connected vertices u and v = there exists some **path** between u and v
- Path = some sequence of nodes connected by edges s.t. no edge repeats (can repeat nodes)
  - Adj nodes in path seq are adj. to each other in real graph
- Cycle = pather where first and last nodes are same
- Degree of vertex = # edges touching vertex

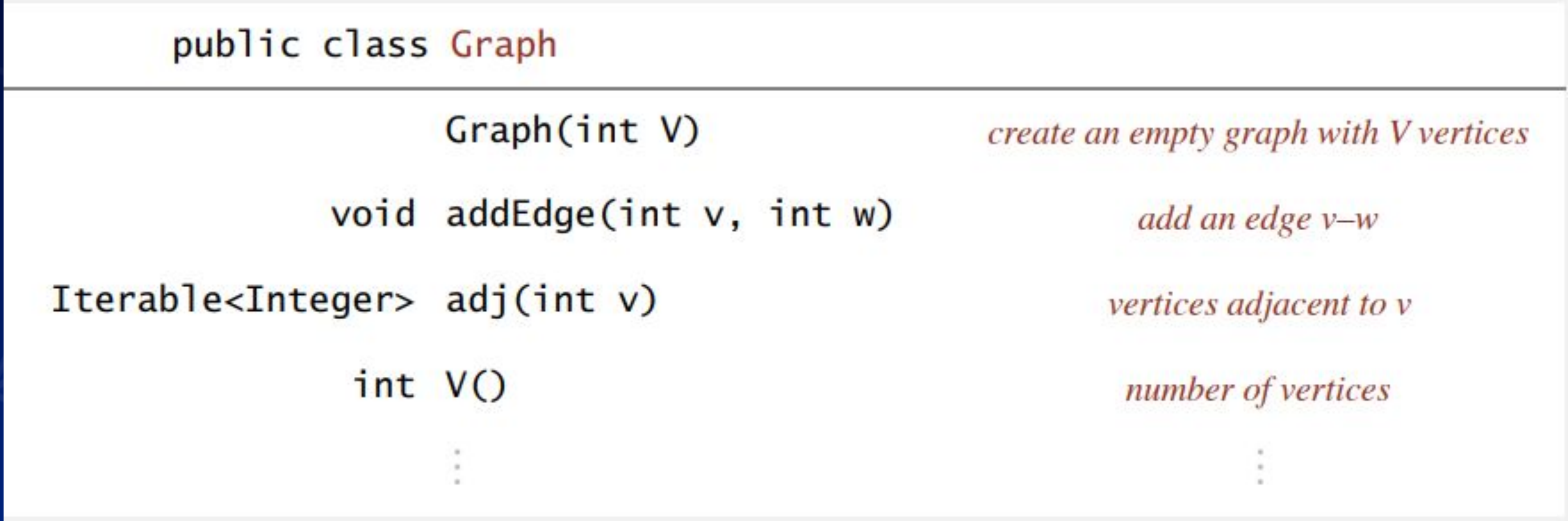
Vocabulary Part 2

- Adjacent nodes = nodes connected by an edge
- Incident edges = edges that share vertex
- Incident vertex u & edge e IF u is one of the two vertices e connects
- Undirected Graph = graph with NO DIRECTION
  - Financial transaction graph may need direction
  - Some types of social networks may not need direction



Credits - Princeton University COS 226 Lecture

Graph API



public class Graph

	`Graph(int V)`	*create an empty graph with V vertices*
`void`	`addEdge(int v, int w)`	*add an edge v–w*
`Iterable<Integer>`	`adj(int v)`	*vertices adjacent to v*
`int`	`V()`	*number of vertices*
	⋮	⋮

Credits - Princeton University COS 226 Lecture

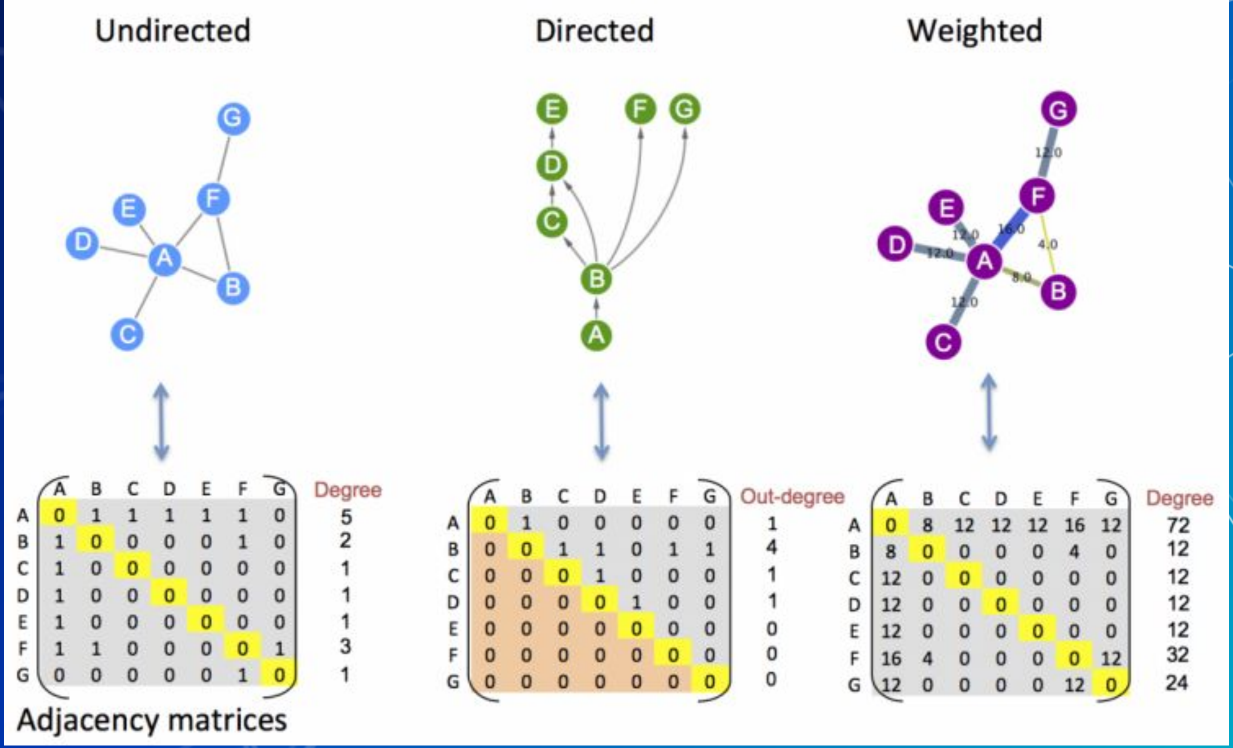
Task 1

How can we compute the degree of a vertex v in the graph G?

Representing a Graph

Understand diff graph rep. & analysis

Method 1 - Adjacency Matrix



	A	B	C	D	E	F	G	Degree
A	0	1	1	1	1	1	0	5
B	1	0	0	0	0	1	0	2
C	1	0	0	0	0	0	0	1
D	1	0	0	0	0	0	0	1
E	1	0	0	0	0	0	0	1
F	1	1	0	0	0	0	1	3
G	0	0	0	0	0	1	0	1

	A	B	C	D	E	F	G	Out-degree
A	0	1	0	0	0	0	0	1
B	0	0	1	1	0	1	1	4
C	0	0	0	1	0	0	0	1
D	0	0	0	0	1	0	0	1
E	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0

	A	B	C	D	E	F	G	Degree
A	0	8	12	12	12	16	12	72
B	8	0	0	0	0	4	0	12
C	12	0	0	0	0	0	0	12
D	12	0	0	0	0	0	0	12
E	12	0	0	0	0	0	0	12
F	16	4	0	0	0	0	12	32
G	12	0	0	0	0	12	0	24

Adjacency matrices

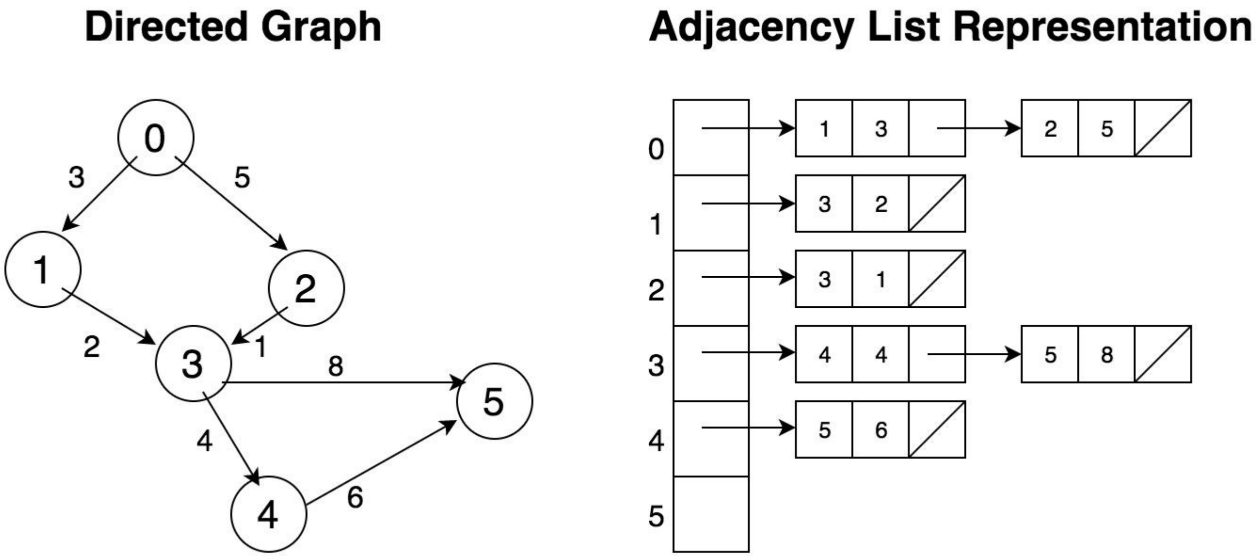
Method 1 - Adjacency Matrix ANALYSIS

Task - print out which which vertices are adjacent.

Write code and analyze runtime.

How could we reduce space complexity?

Method 2- Adjacency List



Directed Graph

Adjacency List Representation

www.kodefork.com

Method 2- Adjacency List ANALYSIS

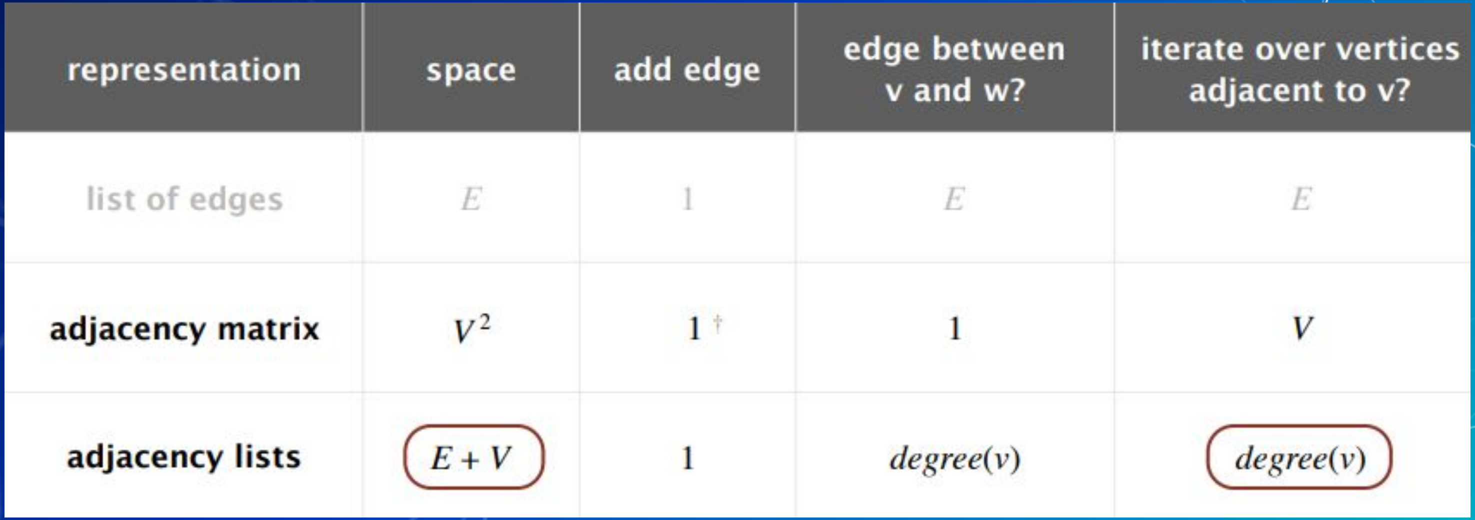
Task - print out which which vertices are adjacent.

Write code and analyze runtime.

Note - harder analysis than adj. matrix

Summary of Graph Rep

- Use adjacency list in real life because much more efficient runtime



representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 †	1	V
adjacency lists	$\boxed{E + V}$	1	$degree(v)$	$\boxed{degree(v)}$

Credits - Princeton University COS 226 Lecture

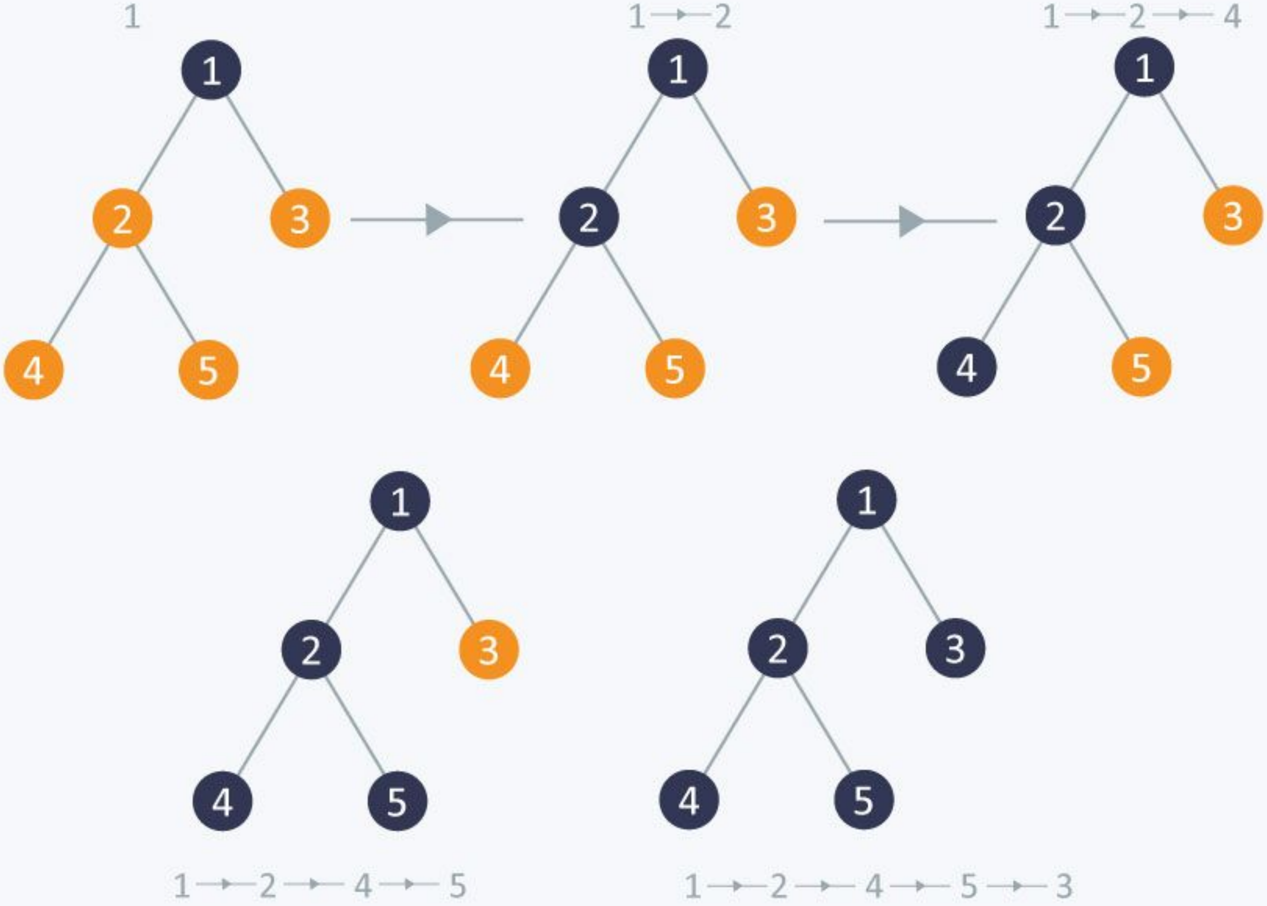
Homework

1) Write Java implementation of API for adjacency list

Depth First Search

DFS - a popular traversal

DFS



Task 1

Write code or pseudocode which performs depth first search (RECURSIVELY)

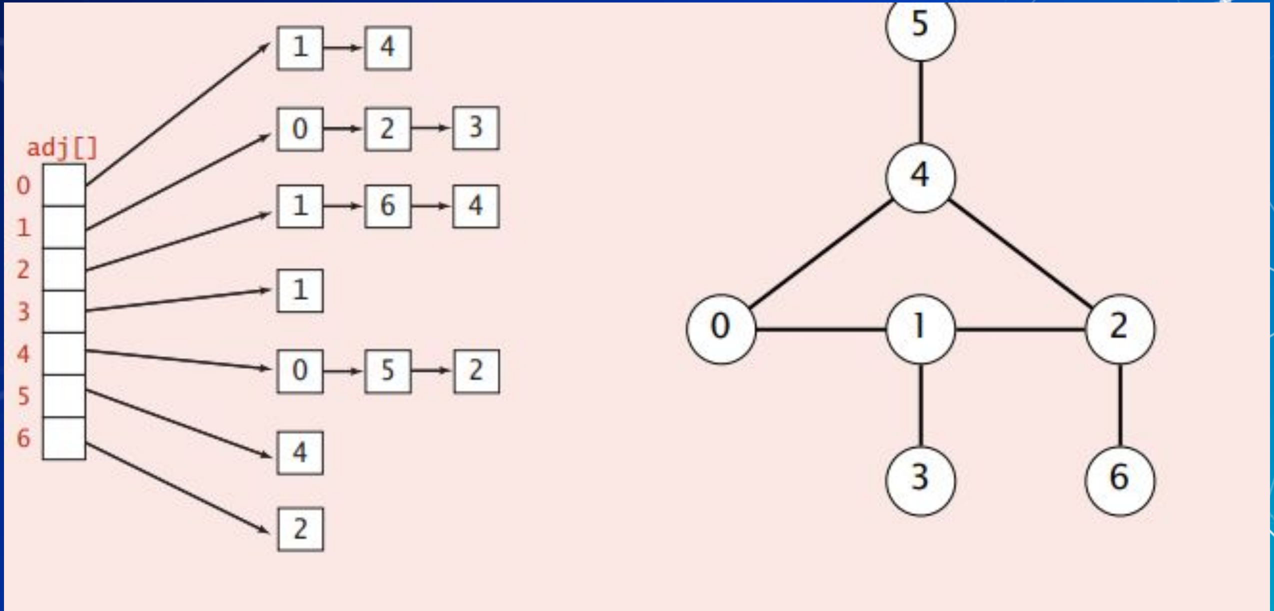
Solution

1) Mark the current vertex v
2) Recursively visit ALL unmarked nodes w that are adjacent to v

Make sure to use the adjacency list AND an array to mark visited nodes

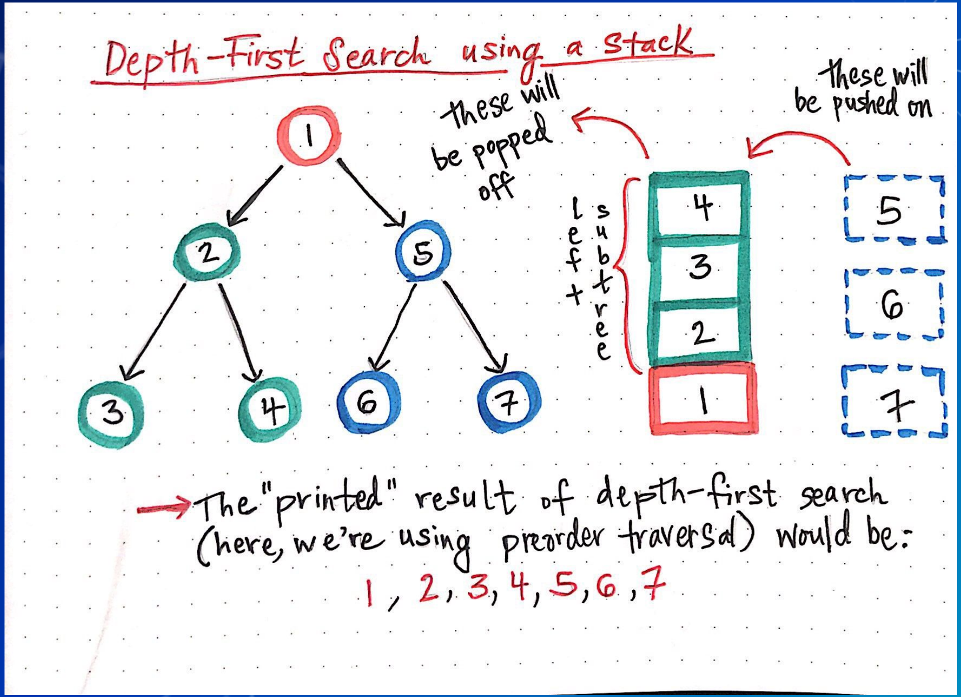
ALSO, can store another array which helps create paths

Demonstration



Credits - Princeton University COS 226 Lecture

Demonstration



Depth-First Search using a Stack

→ The "printed" result of depth-first search (here, we're using preorder traversal) would be:

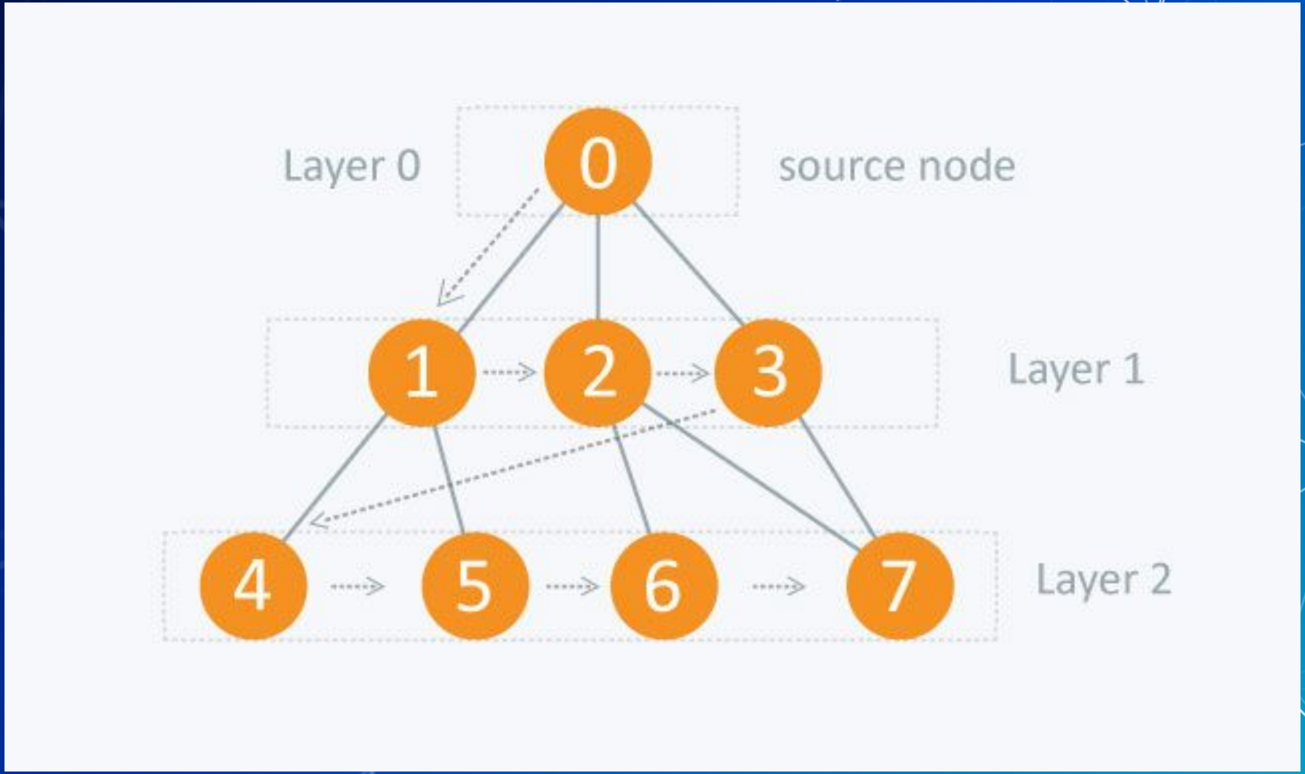
1, 2, 3, 4, 5, 6, 7

DFS Analysis

What is the runtime of the algorithm?

Breadth First Search

BFS - a popular traversal


Layer 0
0
source node
1
2
3
Layer 1
4
5
6
7
Layer 2

Task 1

Write code or pseudocode which performs breadth first search (using a queue)

Solution

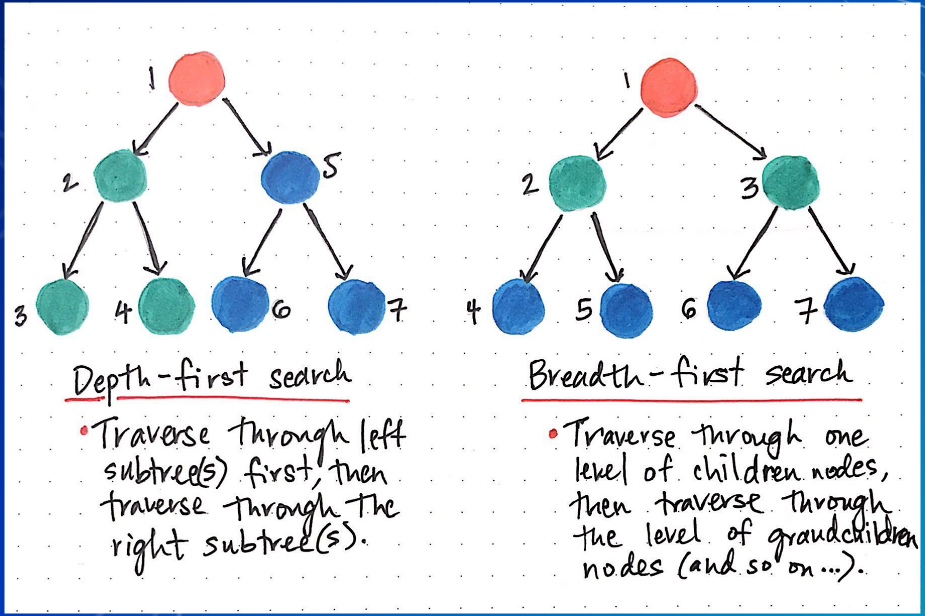
1) Put first node in queue
2) Iterate till queue is EMPTY
    a) Remove least recently added (FIFO) vertex
    b) Add all of v's UNMARKED NEIGHBORS to queue AND mark them

Remember to use adjacency list to generate all possible neighbors ⇒ then from these, check which are unmarked to add them to queue (queue stores UNEXPLORED nodes)

Homework

1) Analyze algorithm to get runtime complexity of BFS

Summary



Depth-first search

- Traverse through left subtree(s) first, then traverse through the right subtree(s).

Breadth-first search

- Traverse through one level of children nodes, then traverse through the level of grandchildren nodes (and so on...).

Summary

https://www.youtube.com/watch?v=zaBht0DEL0w