CS 1332R
WEEK 13

Introduction to Graphs

Breadth-First Search

Depth-First Search

Exam 3 Review

ANNOUNCEMENTS

- ❑

Introduction to Graphs

- ❑ We define a graph **(G)** by its **vertices** and **edges**.
- ❑ We define an edge **(e)** by the two vertices it connects **(u, v)** and its weight **(w)**.

V = the set of vertices

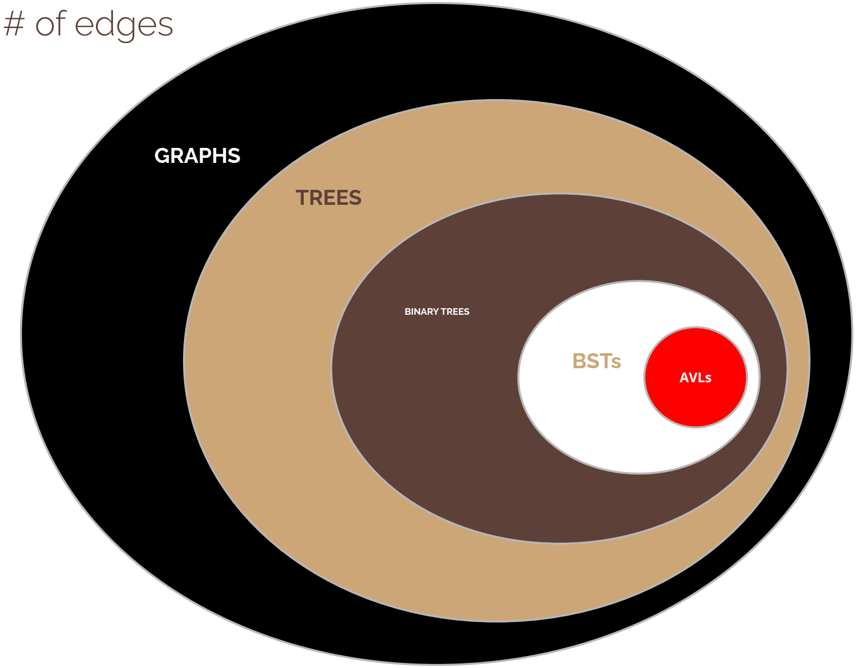
|V| = the # of vertices

E = the set of edges

|E| = the # of edges

RELEVANT TERMINOLOGY

ORDER(G) = |V|
SIZE(G) = |E|
Indegree(v) = # of edges going into a vertex
Outdegree(v) = # of edges going out of a vertex
Adjacent = describes two vertices that are connected by an edge
Connected Graph = every vertex has a path to every other vertex
Unconnected Graph = not all vertices have a path to every other vertex
Directed Graph = edges have a direction, they are distinct <u>ordered pairs</u>
Undirected Graph = edges do not have a direction, e(u, v) is the same as e(v, u)
Subgraph = a graph G’ such that V’ is a subset of V and E’ is a subset of E
Cycle = a path with the same start and end vertex
Acyclic Graph = a graph containing no cycles
Tree = a minimally connected, acyclic graph



Introduction to Graphs

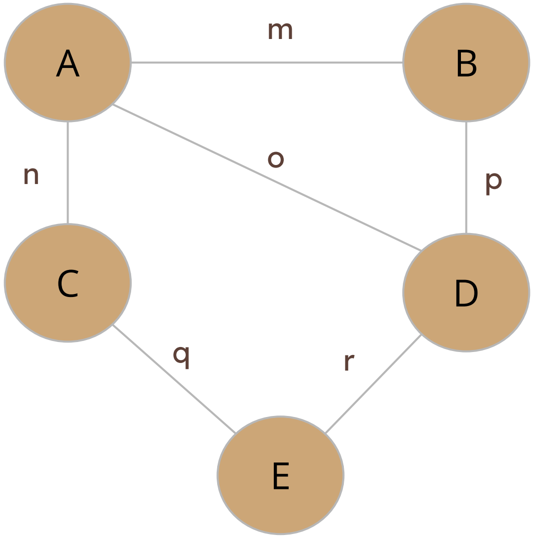
- We define a graph **(G)** by its **vertices** and **edges**.
- We define an edge **(e)** by the two vertices it connects **(u, v)** and its weight **(w)**.

V = the set of vertices

|V| = the # of vertices

E = the set of edges

|E| = the # of edges



HOW WE REPRESENT A GRAPH

1. Adjacency Matrix
   a. |V| x |V| 2D array
   b. Space: O(|V|^2)
   c. Matrix[v1, v2] = edge(v1, v2, w)

2. Adjacency List (***used in your homework***)
   a. Map of each vertex to its incident edges
   b. Space: O(|V| + |E|)
   c. ***In your homework, we map each vertex to a list of VertexDistance objects. The VertexDistance contains an adjacent vertex and the distance to that adjacent vertex, a.k.a the weight of the edge connecting the two vertices.***

3. Edge Set (***used in your homework***)
   a. A set of all edges in the graph
   b. Space: O(|E|)

Breadth-First Search (BFS)

- **PURPOSE**: Traversing a graph from a starting vertex.
- Finds the shortest distance from the start vertex to other vertices ***on an unweighted graph***.
- Use cases: level order traversal, many future algorithms (e.g. Dijkstra's)

STRUCTURES WE NEED

- **Queue<VertexDistance<T>> :** *a priority queue of VertexDistance objects ordered by the distance, which is the cumulative distance from the start vertex to the vertex*
- **Set<Vertex<T>>** : *a **visited set** containing vertices we have found the shortest path to*
- **List<Vertex<T>>** : *a **list** of the vertices in the graph in the order they were visited*

Why do we need a List and a Set to keep track of the visited vertices?

The List is ordered while a Set is unordered. The List has our actual final traversal.

BFS: Implementation

STRUCTURES

- **Queue<VertexDistance<T>> :** *a priority queue of VertexDistance objects ordered by the distance, which is the cumulative distance from the start vertex to the vertex*
- **Set<Vertex<T>>** : *a **visited set** containing vertices we have found the shortest path to*
- **List<Vertex<T>>** : *a **list** of the vertices in the graph in the order they were visited*

ALGORITHM

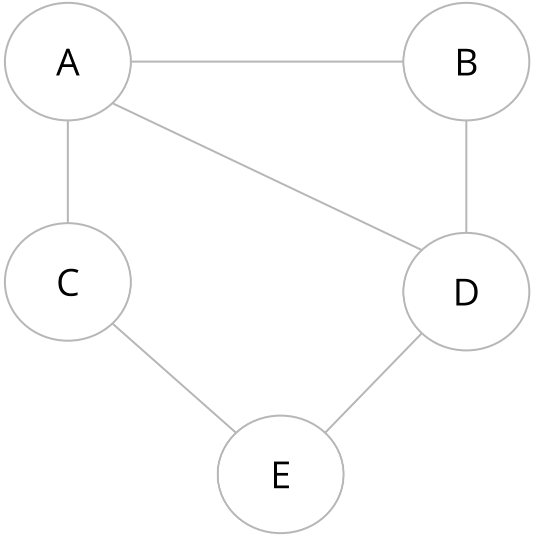
```
initialize s as start vertex
initialize queue, visitedSet, list
queue.enqueue(s)                      2 termination conditions
while queue is not empty and visitedSet is not full:
    v = queue.dequeue()
    if v is not in visitedSet:
        visitedSet.add(v)
        list.add(v)
        for all u adjacent to v and u not in visitedSet:
            queue.enqueue(u)
```

Efficiency: O(|V| + |E|)

- Accessing every vertex and edge once
- In Java, Set is implemented as a HashSet → .contains() is O(1).

BFS: Practice

Start at vertex A.

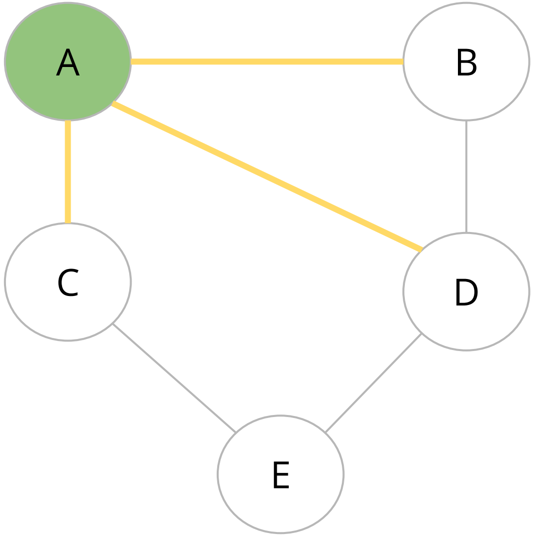


DIAGRAMMING SETUP

QUEUE	VISITED SET
A	

BFS: Practice

Start at vertex A.

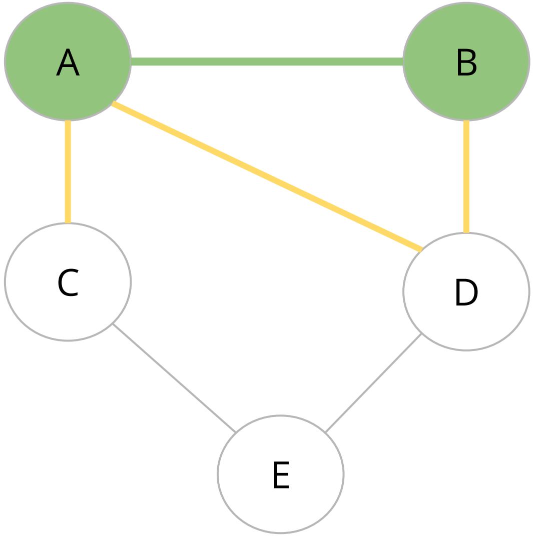


DIAGRAMMING SETUP

QUEUE	VISITED SET
~~A~~	A
B	
C	
D	

BFS: Practice

Start at vertex A.

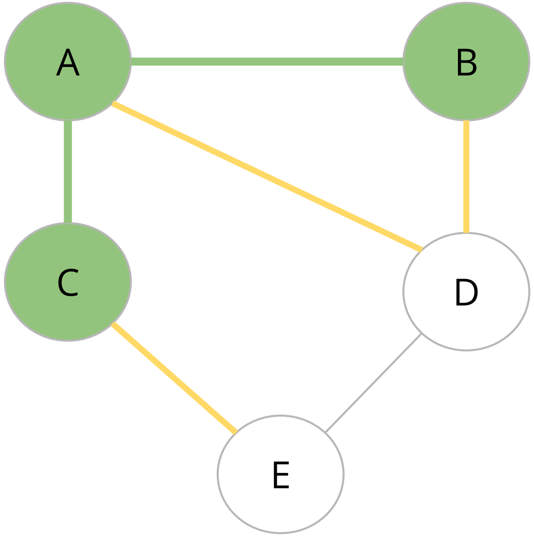


DIAGRAMMING SETUP

QUEUE	VISITED SET
~~A~~	A
~~B~~	B
C	
D	
D	

BFS: Practice

Start at vertex A.

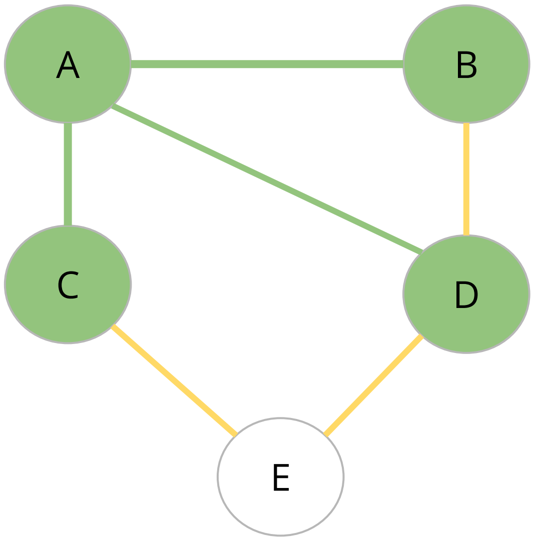


DIAGRAMMING SETUP

QUEUE	VISITED SET
~~A~~	A
~~B~~	B
~~C~~	C
D	
D	
E	

BFS: Practice

Start at vertex A.

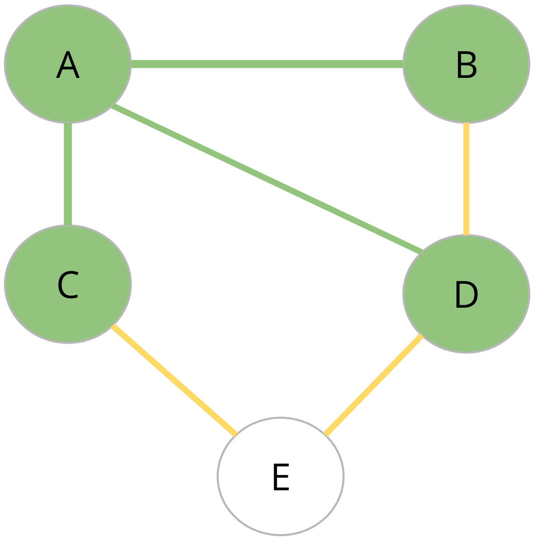


DIAGRAMMING SETUP

QUEUE	VISITED SET
~~A~~	A
~~B~~	B
~~C~~	C
~~D~~	D
D	
E	

BFS: Practice

Start at vertex A.

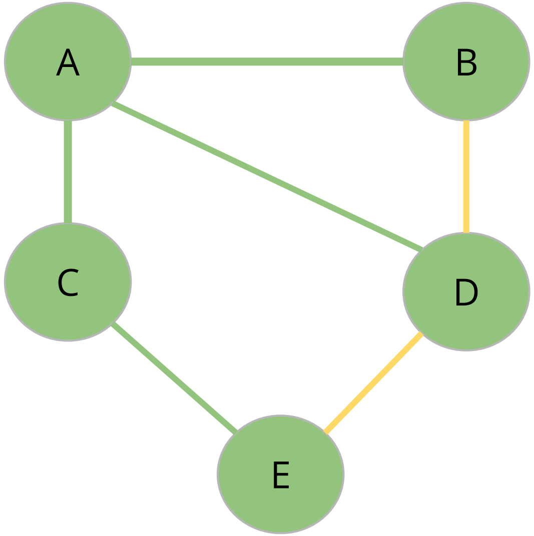


DIAGRAMMING SETUP

QUEUE	VISITED SET
~~A~~	A
~~B~~	B
~~C~~	C
~~D~~	D
~~D~~	
E	

BFS: Practice

Start at vertex A.



DIAGRAMMING SETUP

QUEUE	VISITED SET
~~A~~	A
~~B~~	B
~~C~~	C
~~D~~	D
~~D~~	E
~~E~~	

Depth-First Search (DFS)

- **PURPOSE**: Traversing a graph from a starting vertex.
- Implemented with a stack, we will use the recursive stack
- Use Cases: pre-, post-, in-order traversals, recursion

STRUCTURES WE NEED

****implicit use of a stack through recursion****

- **Set<Vertex<T>>** : *a **visited set** containing vertices we have found the shortest path to*
- **List<Vertex<T>>** : *a **list** of the vertices in the graph in the order they were visited*

DFS: Implementation

STRUCTURES

- **Set<Vertex<T>>** : *a **visited set** containing vertices we have found the shortest path to*
- **List<Vertex<T>>** : *a **list** of the vertices in the graph in the order they were visited*

ALGORITHM

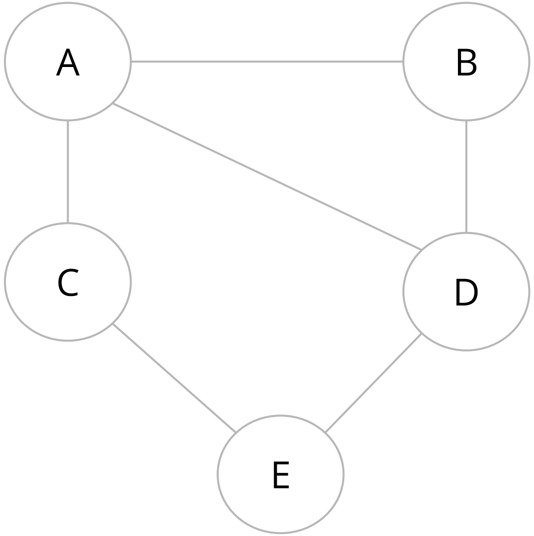
```
initialize s as start vertex
initialize visitedSet, list
dfs(s)

dfs(v):                     base case
    if v not in visitedSet:
        visitedSet.add(v)
        list.add(v)
        for all w adjacent to v and not in visitedSet:
            dfs(w)
```

Efficiency: O(|V| + |E|)

- Accessing every vertex and edge once (same as BFS)
- In Java, Set is implemented as a HashSet → .contains() is O(1).

DFS: Practice

Start at vertex A.

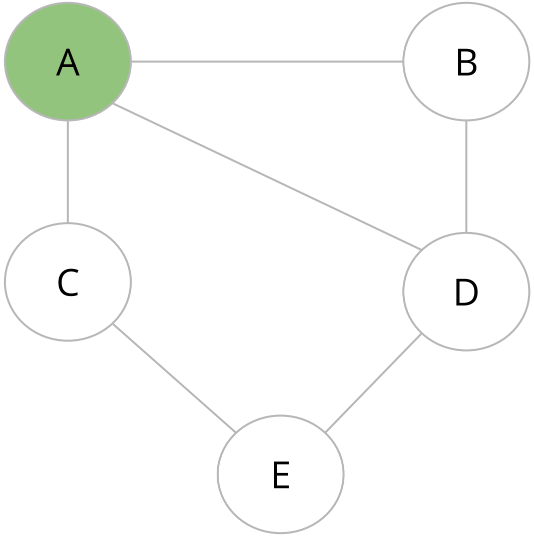


DIAGRAMMING SETUP

STACK	VISITED SET
dfs(A)	

DFS: Practice

Start at vertex A.

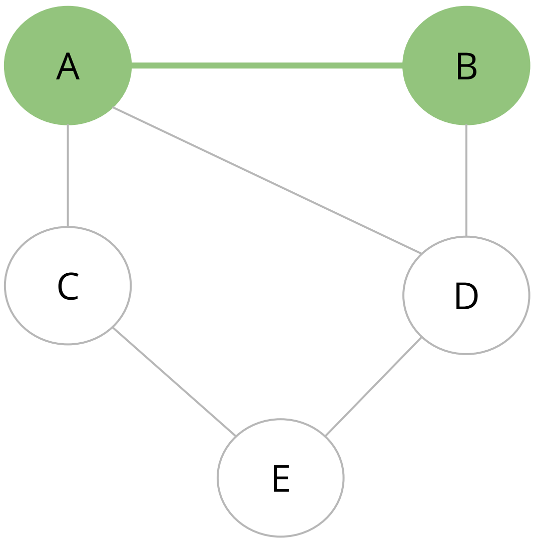


DIAGRAMMING SETUP

STACK	VISITED SET
~~dfs(A)~~	A
dfs(B)	
dfs(C)	
dfs(D)	

DFS: Practice

Start at vertex A.

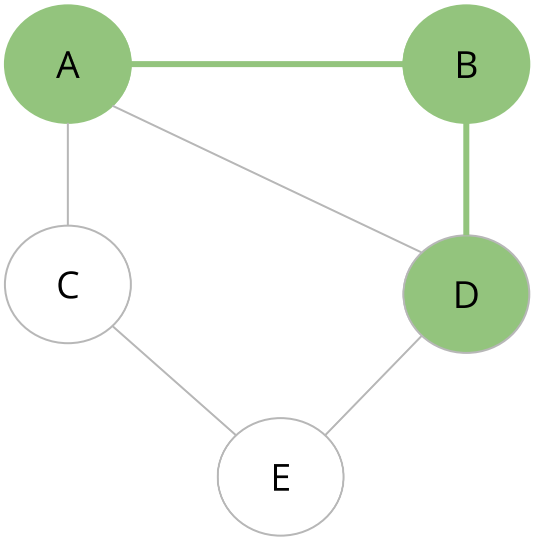


DIAGRAMMING SETUP

STACK	VISITED SET
~~dfs(A)~~	A
~~dfs(B)~~	B
dfs(C)	
dfs(D)	
dfs(D)	

DFS: Practice

Start at vertex A.

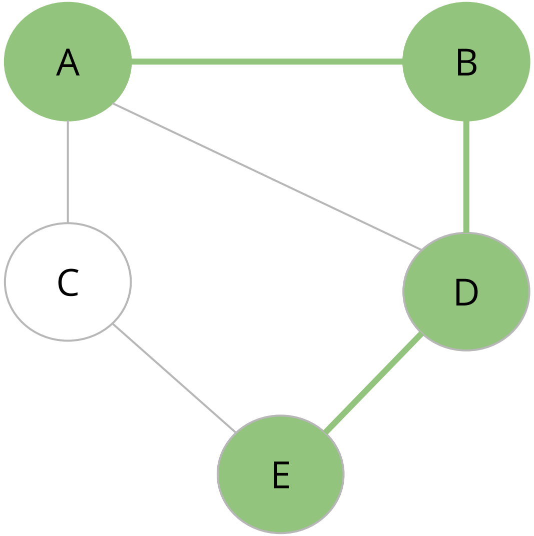


DIAGRAMMING SETUP

STACK	VISITED SET
~~dfs(A)~~	A
~~dfs(B)~~	B
dfs(C)	D
dfs(D)	
~~dfs(D)~~	
dfs(E)	

DFS: Practice

Start at vertex A.

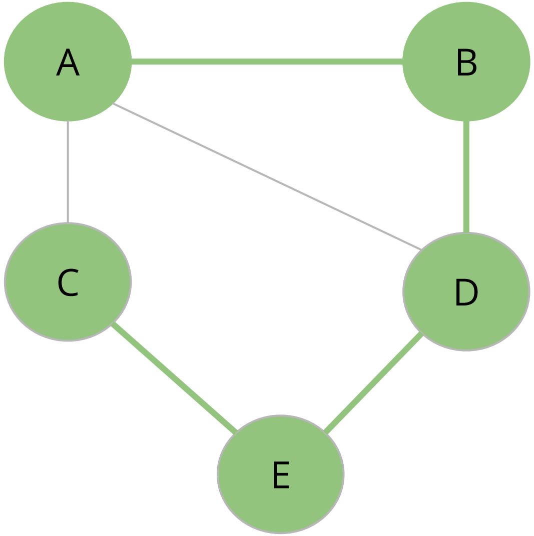


DIAGRAMMING SETUP

STACK	VISITED SET
~~dfs(A)~~	A
~~dfs(B)~~	B
dfs(C)	D
dfs(D)	E
~~dfs(D)~~	
~~dfs(E)~~	
dfs(C)	

DFS: Practice

Start at vertex A.



DIAGRAMMING SETUP

STACK

~~dfs(A)~~
~~dfs(B)~~
dfs(C)
dfs(D)
~~dfs(D)~~
~~dfs(E)~~
~~dfs(C)~~

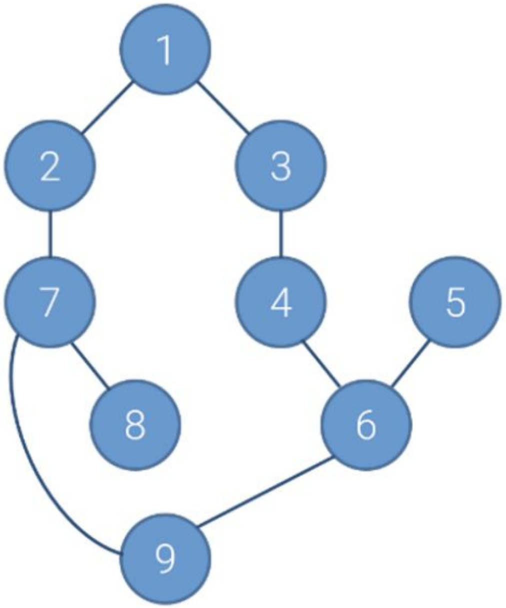
VISITED SET

A
B
D
E
C

BFS/DFS: Practice

Perform BFS & DFS on this graph starting at vertex 1. Iterate through neighbors in numerical order.

Return visited order for each search.



DFS: 1 2 7 8 9 6 4 3 5

BFS: 1 2 3 7 4 8 9 6 5

LEETCODE PROBLEMS

994. Rotting Oranges

785. Is Graph Bipartite?

130. Surrounded Regions

EXAM 3 REVIEW

Kahoot

Socrative: CS1332

Practice exams in Canvas: Files -> Resources -> Recitation Materials -> Recitation Practice Exams

Any questions?

Name
Office Hours
Contact

Name
Office Hours
Contact

Let us know if there is anything specific you want out of recitation!