

CMSC 132: Object-Oriented Programming II



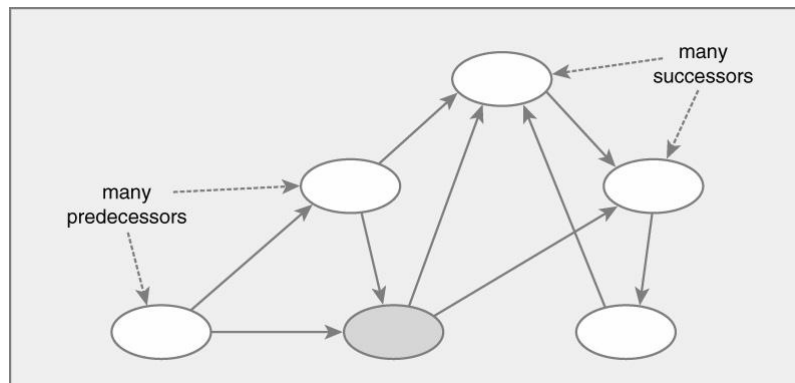
Graphs & Graph Traversal

Department of Computer Science
University of Maryland, College Park

1

Graph Data Structures

- Many-to-many relationship between elements
 - Each element has **multiple** predecessors
 - Each element has **multiple** successors



2

Graph Definitions

■ Node

- Element of graph
- State
 - List of adjacent nodes



■ Edge

- Connection between two nodes
- State
 - Endpoints of edge



3

Graph Definitions

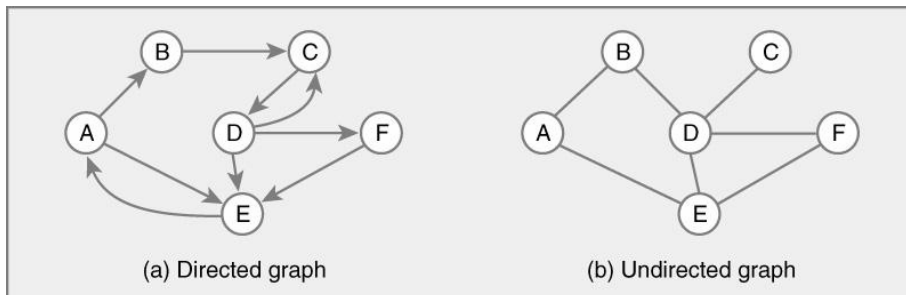
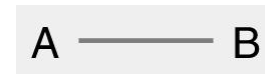
■ Directed graph

- Directed edges



■ Undirected graph

- Undirected edges

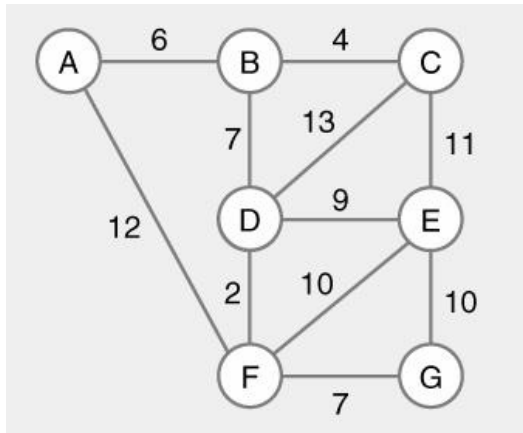


4

Graph Definitions

■ Weighted graph

- Weight (cost) associated with each edge



5

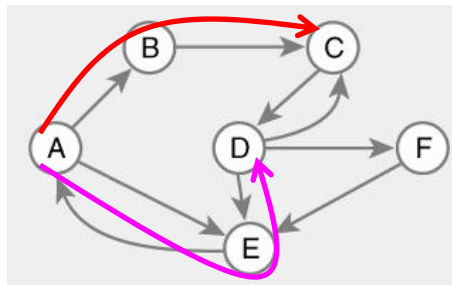
Graph Definitions

■ Path

- Sequence of nodes n_1, n_2, \dots, n_k
- Edge exists between each pair of nodes n_i, n_{i+1}

■ Example

- A, B, C is a path
- A, E, D is not a path



6

Graph Definitions

■ Cycle

- Path that ends back at starting node

■ Example

■ A, E, A

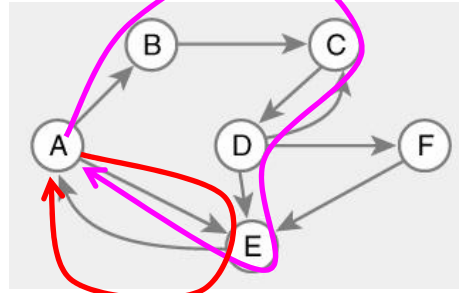
■ A, B, C, D, E, A

■ Simple path

- No cycles in path

■ Acyclic graph

- No cycles in graph



7

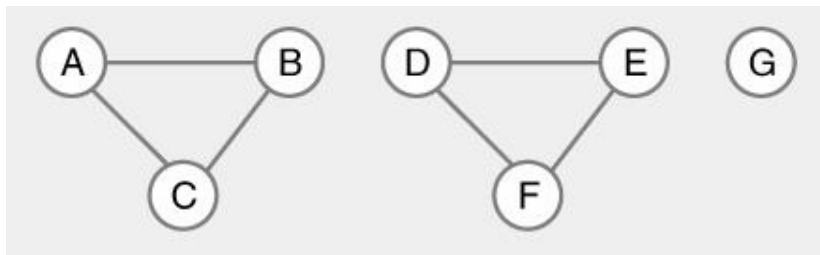
Graph Definitions

■ Reachable

- Path exists between nodes

■ Connected graph

- Every node is reachable from some node in graph



Unconnected graphs

8

Graph Operations

■ Traversal (search)

- Visit each node in graph exactly once
- Usually perform computation at each node
- Two approaches
 - Breadth first search (BFS)
 - Depth first search (DFS)

9

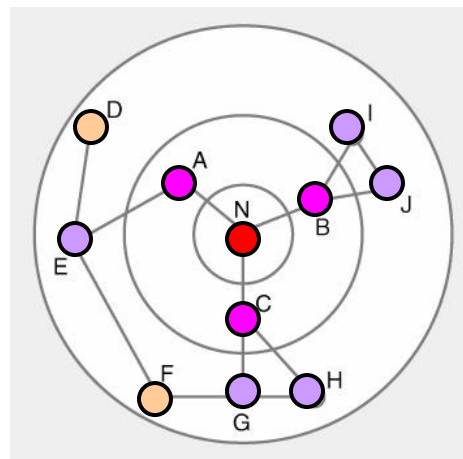
Breadth-first Search (BFS)

■ Approach

- Visit all neighbors of node first
- View as series of expanding circles
- Keep list of nodes to visit in queue

■ Example traversal

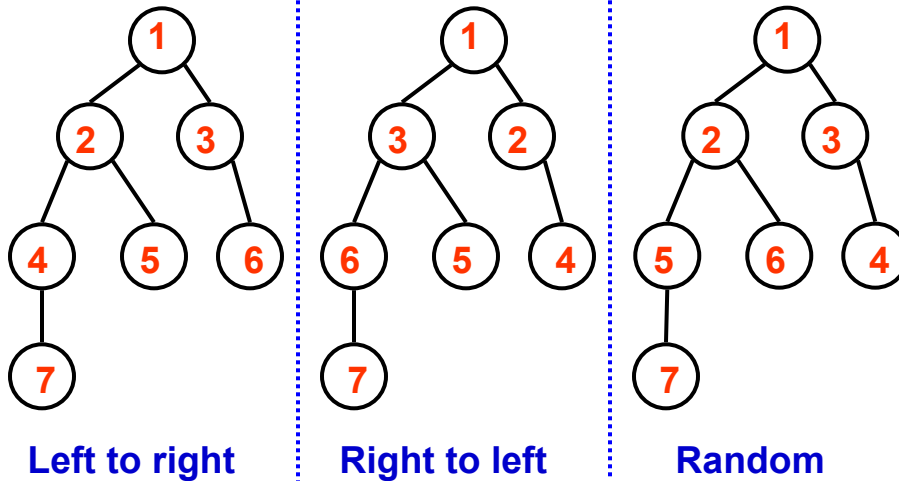
1. n
2. a, c, b
3. e, g, h, i, j
4. d, f



10

Breadth-first Tree Traversal

■ Example traversals starting from 1



11

Traversals Orders

■ Order of successors

■ For tree

- Can order children nodes from left to right

■ For graph

- Left to right doesn't make much sense
- Each node just has a set of successors and predecessors; there is no order among edges

■ For breadth first search

- Visit all nodes at distance k from starting point
- Before visiting any nodes at (minimum) distance $k+1$ from starting point

12

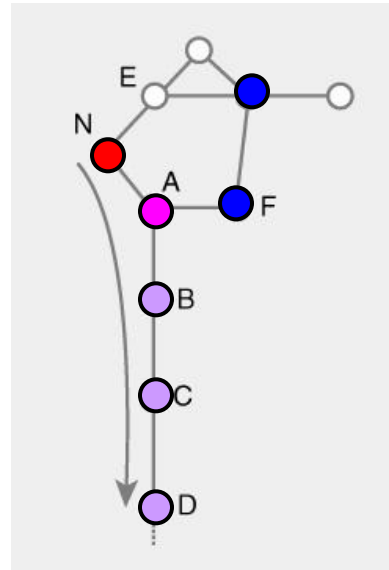
Depth-first Search (DFS)

■ Approach

- Visit all nodes on path first
- **Backtrack** when path ends
- Keep list of nodes to visit in a stack

■ Example traversal

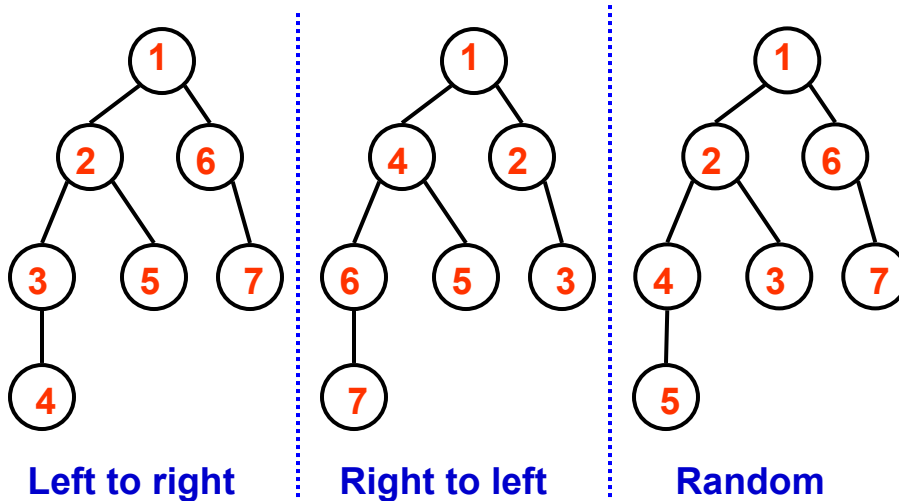
1. **N**
2. **A**
3. **B, C, D, ...**
4. **F...**



13

Depth-first Tree Traversal

■ Example traversals from 1 (preorder)



14

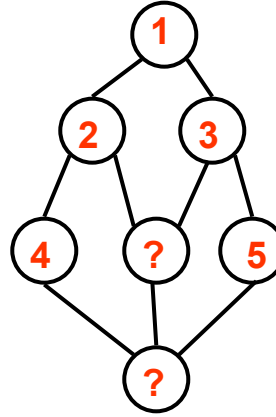
Traversal Algorithms

■ Issue

- How to avoid revisiting nodes
- Infinite loop if cycles present

■ Approaches

- Record set of visited nodes
- Mark nodes as visited

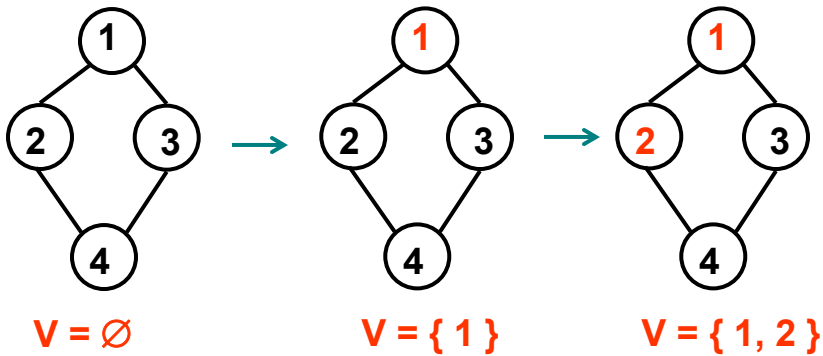


15

Traversal – Avoid Revisiting Nodes

■ Record set of visited nodes

- Initialize { Visited } to empty set
- Add to { Visited } as nodes is visited
- Skip nodes already in { Visited }

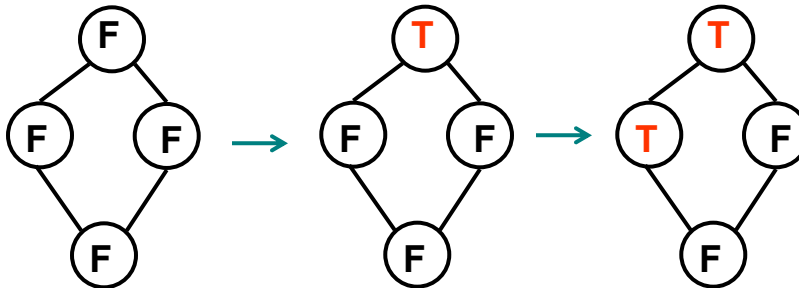


16

Traversal – Avoid Revisiting Nodes

■ Mark nodes as visited

- Initialize tag on all nodes (to False)
- Set tag (to True) as node is visited
- Skip nodes with tag = True



17

Traversal Algorithm Using Sets

```
{ Visited } = ∅  
{ Discovered } = { 1st node }  
while ( { Discovered } ≠ ∅ )  
  take node X out of { Discovered }  
  if X not in { Visited }  
    add X to { Visited }  
    for each successor Y of X  
      if ( Y is not in { Visited } )  
        add Y to { Discovered }
```

18

Traversal Algorithm Using Tags

```
for all nodes X
  set X.tag = False
{ Discovered } = { 1st node }
while ( { Discovered } ≠ ∅ )
  take node X out of { Discovered }
  if (X.tag = False)
    set X.tag = True
    for each successor Y of X
      if (Y.tag = False)
        add Y to { Discovered }
```

19

BFS vs. DFS Traversal

- Order nodes taken out of { Discovered } key
- Implement { Discovered } as Queue
 - First in, first out
 - Traverse nodes breadth first
- Implement { Discovered } as Stack
 - First in, last out
 - Traverse nodes depth first

20

BFS Traversal Algorithm

```
for all nodes X
  X.tag = False
put 1st node in Queue
while ( Queue not empty )
  take node X out of Queue
  if (X.tag = False)
    set X.tag = True
    for each successor Y of X
      if (Y.tag = False)
        put Y in Queue
```

21

DFS Traversal Algorithm

```
for all nodes X
  X.tag = False
put 1st node in Stack
while (Stack not empty )
  pop X off Stack
  if (X.tag = False)
    set X.tag = True
    for each successor Y of X
      if (Y.tag = False)
        push Y onto Stack
```

22

Recursive Graph Traversal

- Can traverse graph using recursive algorithm
 - Recursively visit successors
- Approach
 - Visit (X)
 - for each successor Y of X
 - Visit (Y)
- Implicit call stack & backtracking
 - Results in depth-first traversal

23

Recursive DFS Algorithm

```
Traverse( )  
  for all nodes X  
    set X.tag = False  
    Visit ( 1st node )  
Visit ( X )  
  set X.tag = True  
  for each successor Y of X  
    if (Y.tag = False)  
      Visit ( Y )
```

24