CMSC 132: Object-Oriented Programming II

Graphs & Graph Traversal

Department of Computer Science
University of Maryland, College Park
Graph Data Structures

Many-to-many relationship between elements

- Each element has **multiple** predecessors
- Each element has **multiple** successors

![Graph Diagram]

*Image of a graph with multiple nodes and arrows indicating many-to-many relationships.*
Graph Definitions

**Node**
- Element of graph
- State
  - List of adjacent nodes

**Edge**
- Connection between two nodes
- State
  - Endpoints of edge
Graph Definitions

- Directed graph
  - Directed edges
- Undirected graph
  - Undirected edges

(a) Directed graph
(b) Undirected graph
Graph Definitions

- **Weighted graph**
  - *Weight (cost) associated with each edge*

![Graph with weighted edges](image)
Graph Definitions

Path

- Sequence of nodes $n_1, n_2, \ldots, n_k$
- 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
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
Graph Definitions

- **Reachable**
  - Path exists between nodes

- **Connected graph**
  - Every node is reachable from some node in graph

Unconnected graphs
Graph Operations

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

Example traversals starting from 1

Left to right
Right to left
Random
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
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...
Depth-first Tree Traversal

Example traversals from 1 (preorder)

Left to right
Right to left
Random
Traversal Algorithms

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

Approaches
- Record set of visited nodes
- Mark nodes as visited
Traversing a graph without revisiting nodes:

- Record set of visited nodes
- Initialize $\{\text{Visited}\}$ to empty set
- Add to $\{\text{Visited}\}$ as nodes is visited
- Skip nodes already in $\{\text{Visited}\}$

Initial graph:

\[ V = \emptyset \]

After visiting 1:

\[ V = \{1\} \]

After visiting 2:

\[ V = \{1, 2\} \]
Traversals – 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

```
F
```
```
T
```
```
F
```
Traversing Algorithm Using Sets

\[
\{ \text{Visited} \} = \emptyset \\
\{ \text{Discovered} \} = \{ \text{1st node} \} \\
\text{while } ( \{ \text{Discovered} \} \neq \emptyset ) \\
\quad \text{take node } X \text{ out of } \{ \text{Discovered} \} \\
\quad \text{if } X \text{ not in } \{ \text{Visited} \} \\
\quad \quad \text{add } X \text{ to } \{ \text{Visited} \} \\
\quad \text{for each successor } Y \text{ of } X \\
\quad \quad \text{if } (Y \text{ is not in } \{ \text{Visited} \} ) \\
\quad \quad \quad \text{add } Y \text{ to } \{ \text{Discovered} \} 
\]
Traversals Algorithm Using Tags

for all nodes $X$

set $X.tag = False$

$\{\text{Discovered}\} = \{1\text{st node}\}$

while ($\{\text{Discovered}\} \neq \emptyset$)

take node $X$ out of $\{\text{Discovered}\}$

if ($X.tag = False$)

set $X.tag = True$

for each successor $Y$ of $X$

if ($Y.tag = False$)

add $Y$ to $\{\text{Discovered}\}$
BFS vs. DFS Traversal

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