Graph Data Structures

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

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

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

**Directed graph**
- Directed edges

**Undirected graph**
- Undirected edges

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

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

![Weighted Graph Diagram]

- **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

![Path Diagram]
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
  - $A, E, A$
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

**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)

---

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 Search (BFS)

- Example traversals

1. Left to right
2. Right to left
3. Random

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, a, b, c, d, ...
  2. f …
Depth-first Search (DFS)

Example traversals

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
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 \}

\[ V = \emptyset \]
\[ V = \{ 1 \} \]
\[ V = \{ 1, 2 \} \]

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

\[ F \]
\[ T \]
\[ T \]
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 }

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 }
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

BFS Traversal Algorithm

```plaintext
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 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 )