Design and Analysis of Algorithms
Chapter 4

Dr. M. G. Abbas Malik
Assistant Professor
mgmalik@kau.edu.sa
abbas.malik@gmail.com

Faculty of Computing and Information Technology
King Abdulaziz University (North Jeddah Branch)
Decrease-and-Conquer

1. Reduce problem instance to smaller instance of the same problem
2. Solve smaller instance
3. Extend solution of smaller instance to obtain solution to original instance

- Can be implemented either top-down or bottom-up
- Also referred to as *inductive* or *incremental* approach
3 Types of Decrease and Conquer

- **Decrease by a constant** (usually by 1):
  - insertion sort
  - graph traversal algorithms (DFS and BFS)
  - topological sorting
  - algorithms for generating permutations, subsets

- **Decrease by a constant factor** (usually by half)
  - binary search and bisection method
  - exponentiation by squaring
  - multiplication à la russe

- **Variable-size decrease**
  - Euclid’s algorithm
  - selection by partition
  - Nim-like games
What’s the difference?

Consider the problem of exponentiation: Compute \( a^n \)

- Brute Force
- Divide and conquer
  - Decrease by one
  - Decrease by constant factor
What’s the difference? – Brute Force

BruteForce_Algo_A_power_N(a, n)

powern ← 1
for I ← 1 to n
    powern ← powern * a
return powern

Analysis of this brute-force algorithm?
What’s the difference? – Brute force

BruteForce_Algo_A_power_N(a, n)
  powern ← 1
  for I ← 1 to n
    powern ← powern * a
  return powern

\[ T(n) = c_1 + c_2 \times (n + 1) + c_3 \times n + c_4 \]

\[ \mathcal{O}(n) \]
What’s the difference?

Consider the problem of exponentiation: Compute $a^n$

- Brute Force
- Divide and conquer
  - Decrease by one
  - Decrease by constant factor
What’s the difference? – Divide and conquer

Decrease by 1

- Top-down approach

\[ f(n) = a^n \]

- This function can recursively be defined as

\[ f(n) = \begin{cases} f(n - 1) \cdot a & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases} \]

- Can we develop a recursive algorithm for this?
What’s the difference? – Divide and conquer

Decrease by 1
- Top-down approach

DivideNConquer_TopDown_AlgA_power_N(a, n)
if n = 0
  return 1
return a * DivideNConquer_TopDown_AlgA_power_N(a, n-1)

Analysis of this top-down divide and conquer algorithm?
What’s the difference? – Divide and conquer

Decrease by 1
- Top-down approach

\[
\text{DivideNConquer\_TopDown\_Algo\_A\_power\_N}(a, n) =
\begin{cases}
    c_1 & \text{if } n = 0 \\
    c_2 & \text{return } 1 \\
    c_3 & \text{return } a \times \text{DivideNConquer\_TopDown\_Algo\_A\_power\_N}(a, n-1)
\end{cases}
\]

\[
T(n) = \begin{cases}
    c_1 + c_2 & \text{if the condition is true} \\
    c_1 + c_3 & \text{if the condition is false}
\end{cases}
\]

Constant time complexity – \(O(c)\)

Thus the complexity depends upon the total number of recursive calls.
What’s the difference? – Divide and conquer

**Decrease by 1**
- **Top-down approach**

DivideNConquer_TopDown_Algo_A_power_N(a, n)

if n = 0
    return 1
return a * DivideNConquer_TopDown_Algo_A_power_N(a, n-1)

**Total Number of recursive calls = n+1**
The complexity of first n recursive calls is \((c_1 + c_3) \times n\)
The complexity of the last call is \(c_1 + c_2\)

\[ T(n) = (c_1 + c_3) \times n + c_1 + c_2 \]

\( O(n) \)
What’s the difference? – Divide and conquer

Increase by 1
- Bottom-up approach
  - Multiplying 1 by a n-times
  - Yes it is the same as the brute-force algorithm, but we have come to it by a different thought process.

$O(n)$
What’s the difference?

Consider the problem of exponentiation: Compute $a^n$

- Brute Force
- Divide and conquer
  - Decrease by one
  - Decrease by constant factor
What’s the difference? – Divide and conquer

Decrease by constant factor

- Top-down approach
- Suppose the constant factor is 2 – half of the original problem size
- Can the exponentiation problem $f(n) = a^n$ be recursively defined by reducing the size $n$ by the factor 2?
  - If $n$ is ODD?
  - If $n$ is EVEN?
What’s the difference? – Divide and conquer

Decrease by constant factor

- Top-down approach
- The exponentiation problem \( f(n) = a^n \) can be recursively defined as

\[
a^n = \begin{cases} 
(a^{n/2})^2 & \text{if } n \text{ is even} \\
(a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd} \\
1 & \text{if } n = 0
\end{cases}
\]
What’s the difference? – Divide and conquer

Decrease by constant factor

DivideNConquer_Algo_A_power_N(a, n)
    if n = 1
        return a
    else if n mod 2 = 0
        return DivideNConquer_Algo_A_power_N(a,n/2) * DivideNConquer_Algo_A_power_N(a,n/2)
    else
        return DivideNConquer_Algo_A_power_N(a,(n-1)/2) * DivideNConquer_Algo_A_power_N(a,(n-1)/2) * a

Analysis of this algorithm also depends on the number of recursive calls

Abbas Malik: FCIT, King Abdulaziz University
What’s the difference? – Divide and conquer

Decrease by constant factor

DivideNConquer_Algo_A_power_N(a, 8)

n is even

Recursion Tree
What’s the difference? – Divide and conquer

Decrease by constant factor

DivideNConquer_Algo_A_power_N(a, 8)

Recursion Tree

\( n \text{ is even} \)

\( a^8 \)
What’s the difference? – Divide and conquer

Decrease by constant factor

DivideNConquer_Algo_A_power_N(a, 7)

Recursion Tree

n is odd
What’s the difference? – Divide and conquer

Decrease by constant factor

DivideNConquer_Algotype_N(a, 7)

Recursion Tree

n is odd

$a^6 \times a$

$a, 7$

$a^2 \times a = a^3$

$a, 3$

$a, 1$

$a, 3$

$a, 1$

$a, 1$

$a, 1$

$a, 1$
What’s the difference? – Divide and conquer

Decrease by constant factor

DivideNConquer_Alg0_A_power_N(a, 9)

Recursion Tree
What’s the difference? – Divide and conquer

Decrease by constant factor

DivideNConquer_Algo_A_power_N(a, 9)

Recursion Tree

n is odd

$a^8 \ast a$
Insertion Sort

To sort array $A[1...n]$, sort $A[1..n-1]$ recursively (top down approach) and then insert $A[n]$ in its proper place among the sorted $A[1...n-1]$

- Usually implemented bottom up (non-recursively)

Example: Sort 6, 4, 1, 8, 5

```
6 | 4 1 8 5
4 6 | 1 8 5
1 4 6 | 8 5
1 4 6 8 | 5
1 4 5 6 8
```
Pseudocode of Insertion Sort

```plaintext
InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1
        while (j > 0) and (A[j] > key) {
            j = j - 1
        }
        A[j+1] = key
    }
}
```
# Insertion Sort

<table>
<thead>
<tr>
<th>Statement</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsertionSort(A, n) {</td>
<td></td>
</tr>
<tr>
<td>for i = 2 to n {</td>
<td>$c_1n$</td>
</tr>
<tr>
<td>key = A[i]</td>
<td>$c_2(n-1)$</td>
</tr>
<tr>
<td>j = i - 1;</td>
<td>$c_3(n-1)$</td>
</tr>
<tr>
<td>while (j &gt; 0) and (A[j] &gt; key) {</td>
<td>$c_4\sum t_i$</td>
</tr>
<tr>
<td>A[j+1] = A[j]</td>
<td>$c_5 \sum (t_{i-1})$</td>
</tr>
<tr>
<td>j = j - 1</td>
<td>$c_6 \sum (t_{i-1})$</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>A[j+1] = key</td>
<td>$c_7(n-1)$</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
Analysis of Insertion Sort

- **Time efficiency**
  
  \[ C_{worst}(n) = \frac{n(n-1)}{2} \in \Theta(n^2) \]
  
  \[ C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2) \]
  
  \[ C_{best}(n) = n - 1 \in \Theta(n) \] (also fast on almost sorted arrays)

- **Space efficiency**: in-place

- **Stability**: yes

- **Best elementary sorting algorithm overall**

- **Binary insertion sort**
Graphs

A graph is an ordered pair $G = (V, E)$ where $V \rightarrow$ a set of vertices or nodes and $E \rightarrow$ a set of edges or links between nodes of $E$, where elements are sub set of $V \times V$ considered either as ordered pairs or not.
Graphs

- A Mathematical structure used to model pair-wise relations between objects.
  
  “A graph is collection of vertices and edges that connect pairs of vertices.”
Graphs

- **Types**
  - Directed graph
Graphs

- **Types**
  - Directed graph
  - Un-directed graph
Graphs

- **Types**
  - Directed graph
  - Un-directed graph
  - Complete graph
    - Every two nodes are connected
Graphs

- **Types**
  - Directed graph
  - Un-directed graph
  - Complete graph
    - Every two nodes are connected
  - Disconnected graph
Graphs

- Properties
  - Adjacency
    - Two connected nodes are adjacent to each other.
Graphs

- Properties
  - Adjacency
    - Two connected nodes are adjacent to each other.
    - In complete graph, every two vertices are adjacent to each other.
## Graphs

- **Properties**
  - Adjacency
  - Path in graph
Graphs

- Properties
  - Adjacency
  - Path in graph
Graphs

- Properties
  - Adjacency
  - Path in graph
  - Circuit
    - A circular path
Graphs

- **Properties**
  - Adjacency
  - Path in graph
  - Circuit
    - A circular path
Trees

- A tree is a connected undirected graph with no simple circuits.
- Unique path between two vertices.
- A connected graph where each node has zero or more children and at most one parent.
Root

- A vertex which does not have a parent
Trees

- **Root**
  - A vertex which does not have a parent
- **Parent**
Trees

- **Root**
  - A vertex which does not have a parent
- **Parent**
- **Child**
Trees

- **Root**
  - A vertex which does not have a parent
- **Parent**
- **Child**
- **Internal vertex**
Trees

- **Root**
  - A vertex which does not have a parent
- **Parent**
- **Child**
- **Internal vertex**
- **Leaf**
  - A vertex with no child
Trees

- **Root**
  - A vertex which does not have a parent
- **Parent**
- **Child**
- **Internal vertex**
- **Leaf**
  - A vertex with no child
- **Sibling**
  - Children of the same parent
Trees

- **Degree of a node**
  - No. of children of a node

- Degree of node 1: 2
- Degree of node 2: 2
- Degree of node 3: 1
Trees

- **Degree of a node**
  - No. of children of a node

- **Depth or Level of a node**
  - No. of edges from the root

![Tree Diagram]

- Root
- Depth = 1
- Nodes 2, 3
- Depth = 2
- Nodes 4, 5, 6
- Depth = 3
Trees

- **Degree of a node**
  - No. of children of the node

- **Depth or Level of a node**
  - No. of edges from the root

- **Height of Tree**
  - No. of edges in the longest path from root to a leaf.

Tree Height = 2
Trees

- **Properties**
  - A tree with \( v \) vertices has \( v - 1 \) edges or links.

- **m-ary Tree**
  - Every internal node has not more than \( m \)-children.
  - A rooted m-ary tree of height \( h \) is balanced, if all leafs are at level \( h \) or \( h - 1 \).
  - A complete m-ary tree with \( i \) internal vertices contains \( v = mi + 1 \) vertices.
  - A rooted m-ary tree of height \( h \) has leaves \( l \leq m^h \).
Binary Tree

- A tree with degree of every vertex not more than 2.
  - A vertex cannot have more than 2 children.
- A Binary Tree with \( i \) internal vertices contains how many vertices \( v \)?
  - \( v \leq 2i + 1 \)
- What is the height \( h \) of a binary tree with \( v \) vertices?
  - \( h = \lfloor \log_2 v \rfloor \)
- How many leaves \( l \) a binary tree of height \( h \) may have?
  - \( l \leq 2^h \)
Search Tree

- Data structures that support many dynamic-set operations:
  - SEARCH
  - MINIMUM
  - MAXIMUM
  - PREDECESSOR
  - SUCCESSOR
  - INSERT
  - DELETE

- Examples
  - Heap, Priority Queues, Dictionary, etc.
Binary Search Tree

- An organized Binary Tree.
- Binary Search Tree Property
  - Key of a Parent will be greater than the key of left child (if any)
  - Key of a Parent will be smaller than the key of right child (if any)

Let $x$ be a node in Binary Search Tree.

If $y$ is node in the left sub-tree of $x$,
then $\text{Key}[y] \leq \text{Key}[x]$.

If $y$ is node in the right sub-tree of $x$,
then $\text{Key}[y] \geq \text{Key}[x]$. 
Binary Search Tree - Implementation

Data Structure

<table>
<thead>
<tr>
<th>Parent</th>
<th>Data</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>25</td>
<td>16</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>35</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>53</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>null</td>
<td>77</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Binary Search Tree

- **Operations**
  - Traverse
  - Search
  - Minimum
  - Maximum
  - Insert
  - Successor
  - Predecessor
  - Delete
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - In-order – Root is printed between the left and the right sub trees.

**Inorder-Tree-Walk**($x$)

If $x \neq \text{null}$ then
- Inorder-Tree-Walk(Left($x$))
- Print Key($x$)
- Inorder-Tree-Walk(Right($x$))
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - In-order – Root is printed between the left and the right subtrees.

**Inorder-Tree-Walk(x)**

If $x \neq \text{null}$ then
- Inorder-Tree-Walk(Left($x$))
- Print Key($x$)
- Inorder-Tree-Walk(Right($x$))

```
Inorder-Tree-Walk(x)
```
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - In-order – Root is printed between the left and the right subtrees.

**Inorder-Tree-Walk**(\(x\))
If \(x \neq \text{null}\) then
  - Inorder-Tree-Walk(Left(\(x\)))
  - Print Key(\(x\))
  - Inorder-Tree-Walk(Right(\(x\)))
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - **In-order** – Root is printed between the left and the right subtrees.

\[
\text{Inorder-Tree-Walk}(x) \\
\text{If } x \neq \text{null then} \\
\text{Inorder-Tree-Walk(Left}(x)) \\
\text{Print Key}(x) \\
\text{Inorder-Tree-Walk(Right}(x))
\]
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - In-order – Root is printed between the left and the right sub trees.

**Inorder-Tree-Walk**\( (x) \)

- If \( x \neq \text{null} \) then
  - Inorder-Tree-Walk\( (\text{Left}(x)) \)
  - Print Key\( (x) \)
  - Inorder-Tree-Walk\( (\text{Right}(x)) \)

**Output**

14

Abbas Malik: FCIT, King Abdulaziz University
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - Types
    - In-order – Root is printed between the left and the right sub trees.

\[
\text{Inorder-Tree-Walk}(x) \quad \text{If } x \neq \text{null then} \\
\phantom{\text{Inorder-Tree-Walk}(x)} \text{Inorder-Tree-Walk(Left}(x)) \\
\phantom{\text{Inorder-Tree-Walk}(x)} \text{Print Key}(x) \\
\phantom{\text{Inorder-Tree-Walk}(x)} \text{Inorder-Tree-Walk(Right}(x))
\]

Output

\[14, 25\]
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - In-order – Root is printed between the left and the right subtrees.

**Inorder-Tree-Walk**(x)
If \( x \neq \text{null} \) then
  Inorder-Tree-Walk(Left(x))
  Print Key(x)
  Inorder-Tree-Walk(Right(x))

**Output**
14, 25
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - **In-order** – Root is printed between the left and the right subtrees.

**Inorder-Tree-Walk(x)**

If \( x \neq \text{null} \) then

Inorder-Tree-Walk(Left(x))

Print Key(x)

Inorder-Tree-Walk(Right(x))

Output

14, 25, 35
Binary Search Tree

- Traverse – Tree Walk (In Order)
  - Types
    - In-order – Root is printed between the left and the right sub trees.

**Inorder-Tree-Walk(x)**
If $x \neq \text{null}$ then
Inorder-Tree-Walk(Left(x))
Print Key(x)
Inorder-Tree-Walk(Right(x))

Output
14, 25, 35, 43
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - Types
    - In-order – Root is printed between the left and the right sub trees.

**Inorder-Tree-Walk** ($x$)

If $x \neq$ null then

Inorder-Tree-Walk(Left($x$))

Print Key($x$)

Inorder-Tree-Walk(Right($x$))

**Output**

14, 25, 35, 43
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - In-order – Root is printed between the left and the right sub trees.

\[
\text{Inorder-Tree-Walk}(x)
\]

If \( x \neq \text{null} \) then

- Inorder-Tree-Walk(Left(\( x \)))
- Print Key(\( x \))
- Inorder-Tree-Walk(Right(\( x \)))

Output

14, 25, 35, 43
Binary Search Tree

- Traverse – Tree Walk (In Order)
  - Types
    - In-order – Root is printed between the left and the right sub trees.

```plaintext
INORDER-TREE-WALK(x)
If x ≠ null then
  Inorder-Tree-Walk(Left(x))
  Print Key(x)
  Inorder-Tree-Walk(Right(x))
```

Output

14, 25, 35, 43, 55
Binary Search Tree

- **Traverse – Tree Walk (In Order)**
  - **Types**
    - **In-order** – Root is printed between the left and the right sub trees.

\[
\text{Inorder-Tree-Walk}(x)
\]

If \( x \neq \text{null} \) then
\[
\text{Inorder-Tree-Walk}(\text{Left}(x))
\]
Print Key(x)
\[
\text{Inorder-Tree-Walk}(\text{Right}(x))
\]

Output

14, 25, 35, 43, 55, 65

In-order Tree Walk gives you a sorted output.
### Binary Search Tree

- **Traverse – Tree Walk (Pre-Order)**
  - **Types**
    - Pre-order – Root is printed first then its left and right subtrees.

```plaintext
PREORDER-TREE-WALK(x)
If x ≠ null then
  Print Key(x)
  Preorder-Tree-Walk(Left(x))
  Preorder-Tree-Walk(Right(x))
```

Output

43, 25, 14, 35, 65, 55
Binary Search Tree

- Traverse – Tree Walk (Post-Order)
  - Types
    - Post-order – Root is printed after its left and right sub trees.

```plaintext
Postorder-Tree-Walk(x)
If x ≠ null then
  Postorder-Tree-Walk(Left(x))
  Postorder-Tree-Walk(Right(x))
  Print Key(x)
```

Output

14, 35, 25, 55, 65, 43
Binary Search Tree

• Traverse – Tree Walk (In-Order) Analysis
  - Complexity of the algorithm depends upon?
    - No. of recursive calls
    - $n$ recursive calls for a tree with $n$ nodes

\[
\text{Inorder-Tree-Walk}(x) \\
\text{If } x \neq \text{null then} \\
\text{Inorder-Tree-Walk(Left(x))} \\
\text{Print Key(x)} \\
\text{Inorder-Tree-Walk(Right(x))}
\]

Output
14, 25, 35, 43, 55, 65
Binary Search Tree

• **Operations**
  - Traverse
  - Search
  - Minimum
  - Maximum
  - Insert
  - Successor
  - Predecessor
  - Delete
Binary Search Tree

• **Search**
  - Searching an element in a binary search tree.
    - **Recursive method**

```
SEARCH-TREE (x, k)
  If x = null or Key(x) = k then
    Return x
  If k < Key(x) then
    Return Search-TREE (Left(x), k)
  Else
    Return Search-TREE (Right(x), k)
```

Abbas Malik: FCIT, King Abdulaziz University
Binary Search Tree

- **Search**
  - Searching an element in a binary search tree.
    - Recursive method

**Search-Tree** \((x, k)\)

- If \(x = \text{null} \) or \(\text{Key}(x) = k\) then
  - Return \(x\)
- If \(k < \text{Key}(x)\) then
  - Return Search-Tree (Left\((x)\), \(k\))
- Else
  - Return Search-Tree (Right\((x)\), \(k\))

Search 55 in this tree
Binary Search Tree

• Search
  - Searching an element in a binary search tree.
    - Recursive method

**Search-Tree** \((x, k)\)

- If \(x = \text{null} \) or \(\text{Key}(x) = k\) then
  - Return \(x\)
- If \(k < \text{Key}(x)\) then
  - Return Search-Tree (Left\((x)\), \(k\))
- Else
  - Return Search-Tree (Right\((x)\), \(k\))

Search 55 in this tree
Binary Search Tree

- **Search**
  - Searching an element in a binary search tree.
    - **Recursive method**

**Search-Tree** \((x, k)\)
- If \(x = \text{null} \) or \(\text{Key}(x) = k\) then
  - Return \(x\)
- If \(k < \text{Key}(x)\) then
  - Return Search-Tree (Left\((x)\), \(k\))
- Else
  - Return Search-Tree (Right\((x)\), \(k\))

Search 55 in this tree

K=55
**Binary Search Tree**

- **Search**
  - Searching an element in a binary search tree.
  - **Recursive method**

  \[
  \text{SEARCH-TREE} (x, k)
  \begin{align*}
  &\text{If } x = \text{null or } \text{Key}(x) = k \text{ then} \\
  &\quad \text{Return } x \\
  &\text{If } k < \text{Key}(x) \text{ then} \\
  &\quad \text{Return } \text{Search-Tree} (\text{Left}(x), k) \\
  &\text{Else} \\
  &\quad \text{Return } \text{Search-Tree} (\text{Right}(x), k)
  \end{align*}
\]

Search 55 in this tree

Number is found!!

K=55
Binary Search Tree

- **Search**
  - Searching an element in a binary search tree.
    - Recursive method

**SEARCH-TREE** \((x, k)\)
- If \(x = \text{null} \) or \(\text{Key}(x) = k\) then
  - Return \(x\)
- If \(k < \text{Key}(x)\) then
  - Return **SEARCH-TREE** \((\text{Left}(x), k)\)
- Else
  - Return **SEARCH-TREE** \((\text{Right}(x), k)\)

Check your self how it works when we want to search 26 in this tree!!!!!!!!!!!!!
Binary Search Tree

- **Search**
  - Searching an element in a binary search tree.
    - Iterative method

```
ITERATIVE-SEARCH-TREE (x, k)
    While x ≠ null and k ≠ Key(x)
        If k < Key(x) then
            x = Left(x)
        Else
            x = Right(x)
    Return x
```
Binary Search Tree

- Operations
  - Traverse
  - Search
  - Minimum
  - Maximum
  - Insert
  - Successor
  - Predecessor
  - Delete
Binary Search Tree

- **Minimum or Maximum**
  - Searching the minimum or maximum element in a binary search tree.

  ```
  TREE-MINIMUM (x)
  While Left(x) \neq null
  x = Left(x)
  Return x
  
  TREE-MAXIMUM (x)
  While Right(x) \neq null
  x = Right(x)
  Return x
  ```
Binary Search Tree

• Operations
  ▪ Traverse
  ▪ Search
  ▪ Minimum
  ▪ Maximum
  ▪ Insert
  ▪ Successor
  ▪ Predecessor
  ▪ Delete
Binary Search Tree

- **Insert a node**
  - **Input**
    - A Binary Search Tree $T$ or Root node of the Binary Search Tree
    - A node $z$ with Key[$z$] and Left[$z$] = null and Right[$z$] = null
  - **Output**
    - A Binary tree with added node $z$

- **Algorithmic Steps**
  - First we need to search the right place of $z$ in the input Binary Search Tree $T$.
  - Then we will insert the node $z$ at the right place.
Binary Search Tree

- Insert a node
- Design an Algorithm!!!

```
TREE-INSERT (T, z)
  y = null
  x = Root(T)
  While x ≠ null
    y = x
    If Key(z) < Key(x) then
      x = Left(x)
    Else
      x = Right(x)
  Parent(z) = y
  If y = null then
    Root(T) = z  // Tree is empty
  Else
    If Key(z) < Key(y)
      Left(y) = z
    Else
      Right(y) = z
```
Binary Search Tree

- **Insert a node**
  Insert a node with key = 60

**Tree-Insert** \((T, z)\)

1. \(y = \text{null}\)
2. \(x = \text{Root}(T)\)
3. While \(x \neq \text{null}\)
   - \(y = x\)
   - If \(\text{Key}(z) < \text{Key}(x)\) then
     - \(x = \text{Left}(x)\)
   - Else
     - \(x = \text{Right}(x)\)
4. Parent\((z) = y\)
5. If \(y = \text{null}\) then
   - \(\text{Root}(T) = z\)  // Tree is empty
   Else
     - If \(\text{Key}(z) < \text{Key}(y)\)
       - \(\text{Left}(y) = z\)
     - Else
       - \(\text{Right}(y) = z\)
**Binary Search Tree**

- **Insert a node**

  Insert a node with key = 60

  Tree-Insert \((T, z)\)

  \[
  \begin{align*}
  y &= \text{null} \\
  x &= \text{Root}(T) \\
  \text{While } x \neq \text{null} & \\
  y &= x \\
  \text{If } \text{Key}(z) < \text{Key}(x) & \\
  x &= \text{Left}(x) \\
  \text{Else} & \\
  x &= \text{Right}(x) \\
  \text{Parent}(z) &= y \\
  \text{If } y = \text{null} & \\
  \text{Root}(T) &= z \quad \text{// Tree is empty} \\
  \text{Else} & \\
  \text{If } \text{Key}(z) < \text{Key}(y) & \\
  \text{Left}(y) &= z \\
  \text{Else} & \\
  \text{Right}(y) &= z
  \end{align*}
  \]
**Binary Search Tree**

- **Insert a node**
  - Insert a node with key = 60

```
TREE-INSERT (T, z)
  y = null
  x = Root(T)
  While x ≠ null
    y = x
    If Key(z) < Key(x) then
      x = Left(x)
    Else
      x = Right(x)
    Parent(z) = y
    If y = null then
      Root(T) = z // Tree is empty
    Else
      If Key(z) < Key(y)
        Left(y) = z
      Else
        Right(y) = z
```
Binary Search Tree

- **Insert a node**

  Insert a node with key = 60

  Tree-Insert \((T, z)\)
  
  \[
  \begin{align*}
  y &= \text{null} \\
  x &= \text{Root}(T) \\
  \text{While } x \neq \text{null} \quad \text{do} \\
  & \quad y = x \\
  & \quad \text{If Key}(z) < \text{Key}(x) \quad \text{then} \\
  & \quad \quad x = \text{Left}(x) \\
  & \quad \text{Else} \\
  & \quad \quad x = \text{Right}(x) \\
  & \quad \text{Parent}(z) = y \\
  & \quad \text{If } y = \text{null} \quad \text{then} \\
  & \quad \quad \text{Root}(T) = z \\
  & \quad \text{Else} \\
  & \quad \quad \text{If Key}(z) < \text{Key}(y) \\
  & \quad \quad \quad \text{Left}(y) = z \\
  & \quad \quad \text{Else} \\
  & \quad \quad \quad \text{Right}(y) = z
  \end{align*}
  \]
Binary Search Tree

- **Insert a node**
  
  Insert a node with key = 60

```
TREE-INSERT (T, z)
    y = null
    x = Root(T)
    While x ≠ null
        y = x
        If Key(z) < Key(x) then
            x = Left(x)
        Else
            x = Right(x)
        End If
    End While
    Parent(z) = y
    If y = null then
        Root(T) = z // Tree is empty
    Else
        If Key(z) < Key(y)
            Left(y) = z
        Else
            Right(y) = z
        End If
    End If
```

![Binary Search Tree Diagram](image)
Binary Search Tree

- **Insert a node**

  Insert a node with key = 60

  ![Binary Search Tree Diagram]

  **TREE-INSERT (T, z)**

  ```
  y = null
  x = Root(T)
  While x ≠ null
    y = x
    If Key(z) < Key(x) then
      x = Left(x)
    Else
      x = Right(x)
    Parent(z) = y
    If y = null then
      Root(T) = z // Tree is empty
    Else
      If Key(z) < Key(y)
        Left(y) = z
      Else
        Right(y) = z
  ```
Binary Search Tree

- **Insert a node**
  - Insert a node with key = 60

**Tree-Insert** \((T, z)\)

- \(y = \text{null}\)
- \(x = \text{Root}(T)\)
- While \(x \neq \text{null}\)
  - \(y = x\)
  - If \(\text{Key}(z) < \text{Key}(x)\) then
    - \(x = \text{Left}(x)\)
  - Else
    - \(x = \text{Right}(x)\)
- \(\text{Parent}(z) = y\)
- If \(y = \text{null}\) then
  - \(\text{Root}(T) = z\) // Tree is empty
- Else
  - If \(\text{Key}(z) < \text{Key}(y)\)
    - \(\text{Left}(y) = z\)
  - Else
    - \(\text{Right}(y) = z\)
Binary Search Tree

- Insert a node

Insert a node with key = 60

```plaintext
TREE-INSERT (T, z)
    y = null
    x = Root(T)
    While x ≠ null
        y = x
        If Key(z) < Key(x) then
            x = Left(x)
        Else
            x = Right(x)
        Parent(z) = y
        If y = null then
            Root(T) = z // Tree is empty
        Else
            If Key(z) < Key(y)
                Left(y) = z
            Else
                Right(y) = z
```
Binary Search Tree

- **Insert a node**
  
  Insert a node with key = 60

---

**Tree-Insert** \((T, z)\)

\[
\begin{align*}
y &= \text{null} \\
x &= \text{Root}(T) \\
\text{While } x \neq \text{null} \\
\quad y &= x \\
\quad \text{If } \text{Key}(z) < \text{Key}(x) \text{ then} \\
\quad \quad x &= \text{Left}(x) \\
\quad \text{Else} \\
\quad \quad x &= \text{Right}(x) \\
\text{Parent}(z) &= y \\
\text{If } y = \text{null} \text{ then} \\
\quad \text{Root}(T) &= z \quad \text{// Tree is empty} \\
\text{Else} \\
\quad \text{If } \text{Key}(z) < \text{Key}(y) \\
\quad \quad \text{Left}(y) &= z \\
\quad \text{Else} \\
\quad \quad \text{Right}(y) &= z
\end{align*}
\]
Binary Search Tree

- **Insert a node**
  
  Insert a node with key = 60

```
Tree-Insert (T, z)
  y = null
  x = Root(T)
  While x ≠ null
    y = x
    If Key(z) < Key(x) then
      x = Left(x)
    Else
      x = Right(x)
    Parent(z) = y
  If y = null then
    Root(T) = z // Tree is empty
  Else
    If Key(z) < Key(y)
      Left(y) = z
    Else
      Right(y) = z
```

```
<table>
<thead>
<tr>
<th>Parent</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>null</td>
</tr>
</tbody>
</table>
```
Binary Search Tree

- **Insert a node**
  - Insert a node with key = 60

**Tree-Insert** \((T, z)\)

\[
\begin{align*}
y &= \text{null} \\
x &= \text{Root}(T) \\
\text{While } x \neq \text{null} & \\
&\quad y = x \\
&\quad \text{If } \text{Key}(z) < \text{Key}(x) \text{ then} \\
&\quad&\quad x = \text{Left}(x) \\
&\quad\text{Else} \\
&\quad&\quad x = \text{Right}(x) \\
&\quad\text{Parent}(z) = y \\
&\text{If } y = \text{null} \text{ then} \\
&\quad \text{Root}(T) = z \quad \text{// Tree is empty} \\
&\text{Else} \\
&\quad \text{If } \text{Key}(z) < \text{Key}(y) \text{ then} \\
&\quad&\quad \text{Left}(y) = z \\
&\quad\text{Else} \\
&\quad&\quad \text{Right}(y) = z
\end{align*}
\]
Binary Search Tree

- **Operations**
  - Traverse
  - Search
  - Minimum
  - Maximum
  - Insert
  - Successor
  - Predecessor
  - Delete
Binary Search Tree

- **Successor**
  - If all keys are distinct, the successor of a node $x$ is the node with the smallest key greater than $\text{Key}[x]$.
Binary Search Tree

• **Successor**
  - If all keys are distinct, the successor of a node $x$ is the node with the smallest key greater than $\text{Key}[x]$. 

![Binary Search Tree Diagram]

Abbas Malik: FCIT, King Abdulaziz University
Binary Search Tree

- **Successor**
  - If all keys are distinct, the successor of a node $x$ is the node with the smallest key greater than $\text{Key}[x]$
Binary Search Tree

- **Successor**
  - If all keys are distinct, the successor of a node $x$ is the node with the smallest key greater than $\text{Key}[x]$. 

```
        43
       /   \
      25   65
     /   /   \
    14  35   55
```
Binary Search Tree

- **Successor**
  - If all keys are distinct, the successor of a node $x$ is the node with the smallest key greater than Key[$x$]
Binary Search Tree

- **Successor**

```plaintext
TREE-SUCCESSOR (x)
If Right(x) ≠ null then
    Return Tree-Minimum(Right(x))
y = Parent(x)
While y ≠ null and x = Right(y)
    x = y
    y = Parent(y)
Return y
```
Binairy Search Tree

- **Predecessor**
  - If all keys are distinct, the predecessor of a node $x$ is the node with the largest key smaller than $\text{Key}[x]$.
Binary Search Tree

- **Predecessor**
  - If all keys are distinct, the predecessor of a node $x$ is the node with the largest key smaller than Key[$x$]
Binary Search Tree

• Predecessor
  ▪ If all keys are distinct, the predecessor of a node $x$ is the node with the largest key smaller than Key[$x$]
Binary Search Tree

- **Operations**
  - Traverse
  - Search
  - Minimum
  - Maximum
  - Insert
  - Successor
  - Predecessor
  - Delete
Binary Search Tree

- **Delete a node**
  - **Input**
    - A Binary Search Tree \( T \) or Root node of the Binary Search Tree
    - A node \( z \) that we want to delete
  - **Output**
    - A Binary tree with deleted node \( z \)

- **Algorithmic Steps**
  - First we need to search the node \( z \) in the input Binary Search Tree \( T \).
  - Then we will delete the node \( z \) and rearrange our Binary Search Tree.
**Tree-Delete** (T, z)

If Left(z) = null or Right(z) = null then
    y = z
Else
    y = Tree-Successor(z)
If Left(y) ≠ null then
    x = Left(y)
Else
    x = Right(y)
If x ≠ null then
    Parent(x) = Parent(y)
If Parent(y) = null then
    Root(T) = x
Else
    If y = Left(Parent(y)) then
        Left(Parent(y)) = x
    Else
        Right(Parent(y)) = x
If y ≠ z then
    Key(z) = Key(y)
Return y
Graphs

A graph is an ordered pair $G = (V, E)$ where
$V \rightarrow$ a set of vertices or nodes and
$E \rightarrow$ a set of edges or links between nodes of $E$,
where elements are sub set of $V \times V$
considered either as ordered pairs or not.
Representation of Graphs
Representation of Graphs

Adjacency Matrix Representation

- A matrix with rank $|V| \times |V|$
- $a_{ij} = 1$ if there exists an edge $E(v_i, v_j)$, otherwise $a_{ij} = 0$ (if there is not edge between vertices $V_i$ and $V_j$)

Undirected Graph Adjacency Matrix Representation

![Graph and Adjacency Matrix](image)
Representation of Graphs

Adjacency Matrix Representation

- A matrix with rank $|V| \times |V|$
- $a_{ij} = 1$ if there exists an edge $E(v_i, v_j)$, otherwise $a_{ij} = 0$ (if there is not edge between vertices $V_i$ and $V_j$)

Undirected Graph Adjacency Matrix Representation
Representation of Graphs

Adjacency Matrix Representation

- A matrix with rank $|V| \times |V|$
- $a_{ij} = 1$ if there exists an edge $E(v_i, v_j)$, otherwise $a_{ij} = 0$ (if there is not edge between vertices $V_i$ and $V_j$)

Undirected Graph Adjacency Matrix Representation

Undirected Graph

Adjacency Matrix Representation

\[
\begin{pmatrix}
1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 \\
\end{pmatrix}
\]
Representation of Graphs

Adjacency Matrix Representation

- A matrix with rank $|V| \times |V|
- $a_{ij} = 1$ if there exists an edge $E(v_i, v_j)$, otherwise $a_{ij} = 0$ (if there is not edge between vertices $V_i$ and $V_j$)

Directed Graph Adjacency Matrix Representation
Representation of Graphs

Adjacency List Representation

- Consists of an array Adj of size $|V|$ lists, one for each vertex in $V$
- For each $u \in V$, the adjacency list Adj[$u$] consists all the vertices $v$ such that there is an edge $(u,v) \in E$

Undirected Graph Adjacency List Representation
Representation of Graphs

Adjacency List Representation

- Consists of an array Adj of size $|V|$ lists, one for each vertex in $V$
- For each $u \in V$, the adjacency list Adj$[u]$ consists all the vertices $v$ such that there is an edge $(u,v) \in E$

Directed Graph Adjacency Matrix Representation
Representation of Graphs

Matrix Representation
- Good for dense or complete graph
- Time efficient: it is easy to determine that an edge between two vertices exist or not.
- Space complexity $\Theta(V^2)$

List Representation
- Good for sparse graph
- Space efficient: it uses asymptotically less memory to represent the graph
- Space complexity $\Theta(V+E)$
Graph Traversal

Many problems require processing all graph vertices (and edges) in systematic fashion.

Graph traversal algorithms:

- Depth-first search (DFS)
- Breadth-first search (BFS)
Breadth First Search

- A simplest method for searching a graph
- We assume that the input graph is represented in Adjacency List Representation
- Given a graph $G(V, E)$ and distinguished source vertex $s$, Breadth First Search algorithm explores the edges of $G$ to discover every vertex that is reachable from the source $s$.
- It also produces a “Breadth First Tree” with root $s$ that contains all reachable vertices
- For any vertex $v$ reachable from $s$, the path in the Breadth First Tree from $s$ to $v$ corresponds to the shortest path from $s$ to $v$ in the graph $G$ (the path containing smallest number of edges)
Breadth First Search

- Algorithm works both on directed or undirected graphs
- *Breadth First Search* algorithm first discovers nodes that are closer to the source $s$.
- It discovers all vertices at distance $k$ from the source $s$ before discovering any vertices at distance $k+1$
- To keep track of progress, BFS algorithm colors each vertex white (undiscovered), gray (discovered first time) or black.
- All vertices that are adjacent to black vertex have been discovered
Breadth First Search

- The node in the Adjacency List Representation contains following information:
  1. \( \text{color} \rightarrow \) to contain color information (white, gray or black)
  2. \( \text{d} \rightarrow \) to contain the distance between the source and the vertex
  3. \( \pi \rightarrow \) to contain the address of the parent vertex

![Diagram showing adjacency list representation with vertex names, \( \pi \), Vertex Name, \( d \), and \( \text{color} \).]
Breadth First Search

BFS(G, s)

1. for each vertex \( u \in V[G] - \{s\} \)
   2. do \( \text{color}[u] \leftarrow \text{WHITE} \)
   3. \( d[u] \leftarrow \infty \)
   4. \( \pi[u] \leftarrow \text{NIL} \)
   
5. \( \text{color}[s] \leftarrow \text{GRAY} \)
6. \( d[s] \leftarrow 0 \)
7. \( \pi[s] \leftarrow \text{NIL} \)
8. \( Q \leftarrow \emptyset \)
9. \( \text{ENQUEUE}(Q, s) \)
10. while \( Q \neq \emptyset \)
    11. do \( u \leftarrow \text{DEQUEUE}(Q) \)
    12. for each \( v \in \text{Adj}[u] \)
    13. do if \( \text{color}[v] = \text{WHITE} \)
        then \( \text{color}[v] \leftarrow \text{GRAY} \)
    14. \( d[v] \leftarrow d[u] + 1 \)
    15. \( \pi[v] \leftarrow u \)
    16. \( \text{ENQUEUE}(Q, v) \)
    17. \( \text{color}[u] \leftarrow \text{BLACK} \)

Initialize the color, distance and parent of all vertices except the source.
Initialize the color, distance and parent of all vertices except the source

Initialize the color, distance and parent of the source vertex

Breadth First Search

BFS(G, s)

for each vertex \( u \in V[G] - \{s\} \)

do \( color[u] \leftarrow \) WHITE

d[u] \leftarrow \infty

\( \pi[u] \leftarrow \) NIL

\( color[s] \leftarrow \) GRAY

d[s] \leftarrow 0

\( \pi[s] \leftarrow \) NIL

\( Q \leftarrow \emptyset \)

ENQUEUE(\( Q, s \))

while \( Q \neq \emptyset \)
do \( u \leftarrow \) DEQUEUE(\( Q \))

for each \( v \in Adj[u] \)
do if \( color[v] = \) WHITE

then \( color[v] \leftarrow \) GRAY

d[v] \leftarrow d[u] + 1

\( \pi[v] \leftarrow u \)

ENQUEUE(\( Q, v \))

\( color[u] \leftarrow \) BLACK
Breadth First Search

BFS(G, s)

1. for each vertex \( u \in V[G] - \{s\} \)
   do \( \text{color}[u] \leftarrow \text{WHITE} \)
   \( d[u] \leftarrow \infty \)
   \( \pi[u] \leftarrow \text{NIL} \)

2. \( \text{color}[s] \leftarrow \text{GRAY} \)
3. \( d[s] \leftarrow 0 \)
4. \( \pi[s] \leftarrow \text{NIL} \)
5. \( Q \leftarrow \emptyset \)
6. \( \text{ENQUEUE}(Q, s) \)

while \( Q \neq \emptyset \)
   do \( u \leftarrow \text{DEQUEUE}(Q) \)
   for each \( v \in \text{Adj}[u] \)
      do if \( \text{color}[v] = \text{WHITE} \)
      then \( \text{color}[v] \leftarrow \text{GRAY} \)
      \( d[v] \leftarrow d[u] + 1 \)
      \( \pi[v] \leftarrow u \)
      \( \text{ENQUEUE}(Q, v) \)
6. \( \text{color}[u] \leftarrow \text{BLACK} \)
Breadth First Search

\[
BFS(G, s)
\]

1. for each vertex \( u \in V[G] - \{s\} \)
   - do \( \text{color}[u] \leftarrow \text{WHITE} \)
   - \( d[u] \leftarrow \infty \)
   - \( \pi[u] \leftarrow \text{NIL} \)

2. \( \text{color}[s] \leftarrow \text{GRAY} \)
3. \( d[s] \leftarrow 0 \)
4. \( \pi[s] \leftarrow \text{NIL} \)
5. \( Q \leftarrow \emptyset \)
6. \( \text{ENQUEUE}(Q, s) \)
7. while \( Q \neq \emptyset \)
   - do \( u \leftarrow \text{DEQUEUE}(Q) \)
   - for each \( v \in \text{Adj}[u] \)
     - do if \( \text{color}[v] = \text{WHITE} \)
       - then \( \text{color}[v] \leftarrow \text{GRAY} \)
       - \( d[v] \leftarrow d[u] + 1 \)
       - \( \pi[v] \leftarrow u \)
       - \( \text{ENQUEUE}(Q, v) \)
   - \( \text{color}[u] \leftarrow \text{BLACK} \)

The main breadth first searching loop that search each and every vertex in an increasing order of its distance from the source vertex \( s \) and build Breadth First Tree during the search process.
**Breadth First Search**

BFS$(G, s)$

```plaintext
1. for each vertex $u \in V[G] - \{s\}$
   
2. do $\text{color}[u] \leftarrow \text{WHITE}$
   
3. $d[u] \leftarrow \infty$
   
4. $\pi[u] \leftarrow \text{NIL}$

5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$

8. $Q \leftarrow \emptyset$
9. $\text{ENQUEUE}(Q, s)$
10. while $Q \neq \emptyset$
11. do $u \leftarrow \text{DEQUEUE}(Q)$
12. for each $v \in \text{Adj}[u]$
13. do if $\text{color}[v] = \text{WHITE}$
14. then $\text{color}[v] \leftarrow \text{GRAY}$
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. $\text{ENQUEUE}(Q, v)$
18. $\text{color}[u] \leftarrow \text{BLACK}$
```
Breadth First Search

BFS(G, s)

1. for each vertex \( u \in V[G] \setminus \{s\} \)
   2. do \( \text{color}[u] \leftarrow \text{WHITE} \)
   3. \( d[u] \leftarrow \infty \)
   4. \( \pi[u] \leftarrow \text{NIL} \)
   5. \( \text{color}[s] \leftarrow \text{GRAY} \)
   6. \( d[s] \leftarrow 0 \)
   7. \( \pi[s] \leftarrow \text{NIL} \)
   8. \( Q \leftarrow \emptyset \)
   9. \( \text{ENQUEUE}(Q, s) \)

10. while \( Q \neq \emptyset \)
    11. do \( u \leftarrow \text{DEQUEUE}(Q) \)
    12. for each \( v \in \text{Adj}[u] \)
    13. do if \( \text{color}[v] = \text{WHITE} \)
    14. then \( \text{color}[v] \leftarrow \text{GRAY} \)
    15. \( d[v] \leftarrow d[u] + 1 \)
    16. \( \pi[v] \leftarrow u \)
    17. \( \text{ENQUEUE}(Q, v) \)
    18. \( \text{color}[u] \leftarrow \text{BLACK} \)
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. do $color[u] \leftarrow$ WHITE
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow$ NIL
5. $color[s] \leftarrow$ GRAY
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow$ NIL
8. $Q \leftarrow \emptyset$
9. ENQUEUE($Q, s$)
10. while $Q \neq \emptyset$
11. do $u \leftarrow$ DEQUEUE($Q$)
12. for each $v \in Adj[u]$
13. do if $color[v] =$ WHITE
14. then $color[v] \leftarrow$ GRAY
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. ENQUEUE($Q, v$)
18. $color[u] \leftarrow$ BLACK
Breadth First Search

BFS($G, s$)

1. for each vertex $u \in V[G] - \{s\}$
2. do $\text{color}[u] \leftarrow \text{WHITE}$
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow \text{NIL}$
5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. $\text{ENQUEUE}(Q, s)$
10. while $Q \neq \emptyset$
11. do $u \leftarrow \text{DEQUEUE}(Q)$
12. for each $v \in \text{Adj}[u]$
13. do if $\text{color}[v] = \text{WHITE}$
14. then $\text{color}[v] \leftarrow \text{GRAY}$
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. $\text{ENQUEUE}(Q, v)$
18. $\text{color}[u] \leftarrow \text{BLACK}$
Breadth First Search

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\)
2. \(\text{do color}[u] \leftarrow \text{WHITE}\)
3. \(d[u] \leftarrow \infty\)
4. \(\pi[u] \leftarrow \text{NIL}\)
5. \(\text{color}[s] \leftarrow \text{GRAY}\)
6. \(d[s] \leftarrow 0\)
7. \(\pi[s] \leftarrow \text{NIL}\)
8. \(Q \leftarrow \emptyset\)
9. \(\text{ENQUEUE}(Q, s)\)
10. while \(Q \neq \emptyset\)
11. \(\text{do } u \leftarrow \text{DEQUEUE}(Q)\)
12. \(\text{for each } v \in \text{Adj}[u]\)
13. \(\text{do if color}[v] = \text{WHITE}\)
14. \(\text{then color}[v] \leftarrow \text{GRAY}\)
15. \(d[v] \leftarrow d[u] + 1\)
16. \(\pi[v] \leftarrow u\)
17. \(\text{ENQUEUE}(Q, v)\)
18. \(\text{color}[u] \leftarrow \text{BLACK}\)
Breadth First Search

BFS(G, s)
1   for each vertex u ∈ V[G] − {s}
2       do color[u] ← WHITE
3       d[u] ← ∞
4       π[u] ← NIL
5   color[s] ← GRAY
6   d[s] ← 0
7   π[s] ← NIL
8   Q ← Ø
9   ENQUEUE(Q, s)
10  while Q ≠ Ø
11     do u ← DEQUEUE(Q)
12        for each v ∈ Adj[u]
13           do if color[v] = WHITE
14              then color[v] ← GRAY
15                 d[v] ← d[u] + 1
16                 π[v] ← u
17                 ENQUEUE(Q, v)
18                 color[u] ← BLACK
Breadth First Search

BFS(G, s)

1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← Ø
9 ENQUEUE(Q, s)
10 while Q ≠ Ø
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS(G, s)
1  for each vertex u ∈ V[G] − {s}
2    do color[u] ← WHITE
3     d[u] ← ∞
4     π[u] ← NIL
5  color[s] ← GRAY
6  d[s] ← 0
7  π[s] ← NIL
8  Q ← Ø
9  ENQUEUE(Q, s)
10  while Q ≠ Ø
11    do u ← DEQUEUE(Q)
12      for each v ∈ Adj[u]
13        do if color[v] = WHITE
14           then color[v] ← GRAY
15             d[v] ← d[u] + 1
16             π[v] ← u
17             ENQUEUE(Q, v)
18  color[u] ← BLACK
**Breadth First Search**

\[
\text{BFS}(G, s) \\
1 \quad \text{for each vertex } u \in V[G] - \{s\} \\
2 \quad \text{do } color[u] \leftarrow \text{WHITE} \\
3 \quad d[u] \leftarrow \infty \\
4 \quad \pi[u] \leftarrow \text{NIL} \\
5 \quad color[s] \leftarrow \text{GRAY} \\
6 \quad d[s] \leftarrow 0 \\
7 \quad \pi[s] \leftarrow \text{NIL} \\
8 \quad Q \leftarrow \emptyset \\
9 \quad \text{ENQUEUE}(Q, s) \\
10 \quad \text{while } Q \neq \emptyset \\
11 \quad \text{do } u \leftarrow \text{DEQUEUE}(Q) \\
12 \quad \text{for each } v \in \text{Adj}[u] \\
13 \quad \text{do if } color[v] = \text{WHITE} \\
14 \quad \text{then } color[v] \leftarrow \text{GRAY} \\
15 \quad \quad d[v] \leftarrow d[u] + 1 \\
16 \quad \pi[v] \leftarrow u \\
17 \quad \text{ENQUEUE}(Q, v) \\
18 \quad color[u] \leftarrow \text{BLACK}
\]
Breadth First Search

BFS\( (G, s) \)
1. for each vertex \( u \in V[G] - \{s\} \)
2. do \( \text{color}[u] \leftarrow \text{WHITE} \)
3. \( d[u] \leftarrow \infty \)
4. \( \pi[u] \leftarrow \text{NIL} \)
5. \( \text{color}[s] \leftarrow \text{GRAY} \)
6. \( d[s] \leftarrow 0 \)
7. \( \pi[s] \leftarrow \text{NIL} \)
8. \( Q \leftarrow \emptyset \)
9. \( \text{ENQUEUE}(Q, s) \)
10. while \( Q \neq \emptyset \)
11. do \( u \leftarrow \text{DEQUEUE}(Q) \)
12. for each \( v \in \text{Adj}[u] \)
13. do if \( \text{color}[v] = \text{WHITE} \)
14. then \( \text{color}[v] \leftarrow \text{GRAY} \)
15. \( d[v] \leftarrow d[u] + 1 \)
16. \( \pi[v] \leftarrow u \)
17. \( \text{ENQUEUE}(Q, v) \)
18. \( \text{color}[u] \leftarrow \text{BLACK} \)
Breadth First Search

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\)
2. do \(color[u] \leftarrow \text{WHITE}\)
3. \(d[u] \leftarrow \infty\)
4. \(\pi[u] \leftarrow \text{NIL}\)
5. \(color[s] \leftarrow \text{GRAY}\)
6. \(d[s] \leftarrow 0\)
7. \(\pi[s] \leftarrow \text{NIL}\)
8. \(Q \leftarrow \emptyset\)
9. \texttt{ENQUEUE}(Q, s)
10. while \(Q \neq \emptyset\)
11. do \(u \leftarrow \texttt{DEQUEUE}(Q)\)
12. for each \(v \in \text{Adj}[u]\)
13. do if \(color[v] = \text{WHITE}\)
14. then \(color[v] \leftarrow \text{GRAY}\)
15. \(d[v] \leftarrow d[u] + 1\)
16. \(\pi[v] \leftarrow u\)
17. \texttt{ENQUEUE}(Q, v)
18. \(color[u] \leftarrow \text{BLACK}\)
Breadth First Search

```
BFS(G, s)
1  for each vertex u ∈ V[G] − {s}
2       do color[u] ← WHITE
3       d[u] ← ∞
4       π[u] ← NIL
5  color[s] ← GRAY
6  d[s] ← 0
7  π[s] ← NIL
8  Q ← ∅
9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11    do u ← DEQUEUE(Q)
12       for each v ∈ Adj[u]
13          do if color[v] = WHITE
14             then color[v] ← GRAY
15               d[v] ← d[u] + 1
16               π[v] ← u
17               ENQUEUE(Q, v)
18       color[u] ← BLACK
```
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. \hspace{1em} do $\text{color}[u] \leftarrow \text{WHITE}$
3. \hspace{1em} $d[u] \leftarrow \infty$
4. \hspace{1em} $\pi[u] \leftarrow \text{NIL}$
5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. $\text{ENQUEUE}(Q, s)$
10. while $Q \neq \emptyset$
\hspace{1em} do $u \leftarrow \text{DEQUEUE}(Q)$
\hspace{2em} for each $v \in \text{Adj}[u]$
\hspace{3em} do if $\text{color}[v] = \text{WHITE}$
\hspace{4em} then $\text{color}[v] \leftarrow \text{GRAY}$
\hspace{4em} $d[v] \leftarrow d[u] + 1$
\hspace{4em} $\pi[v] \leftarrow u$
\hspace{4em} $\text{ENQUEUE}(Q, v)$
\hspace{4em} $\text{color}[u] \leftarrow \text{BLACK}$
Breadth First Search

BFS(G, s)
1 for each vertex $u \in V[G] - \{s\}$
2 do $color[u] \leftarrow \text{WHITE}$
3 $d[u] \leftarrow \infty$
4 $\pi[u] \leftarrow \text{NIL}$
5 $color[s] \leftarrow \text{GRAY}$
6 $d[s] \leftarrow 0$
7 $\pi[s] \leftarrow \text{NIL}$
8 $Q \leftarrow \emptyset$
9 $\text{ENQUEUE}(Q, s)$
10 while $Q \neq \emptyset$
11 do $u \leftarrow \text{DEQUEUE}(Q)$
12 for each $v \in Adj[u]$
13 do if $color[v] = \text{WHITE}$
14 then $color[v] \leftarrow \text{GRAY}$
15 $d[v] \leftarrow d[u] + 1$
16 $\pi[v] \leftarrow u$
17 $\text{ENQUEUE}(Q, v)$
18 $color[u] \leftarrow \text{BLACK}$
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← ∅
9 ENQUEUE(Q, s)
10 while Q ≠ ∅
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS\( (G, \, s) \)

1. for each vertex \( u \in V[G] - \{s\} \)
2. \( \text{do} \) color[\( u \)] \( \leftarrow \) WHITE
3. \( d[u] \leftarrow \infty \)
4. \( \pi[u] \leftarrow \text{NIL} \)
5. color[\( s \)] \( \leftarrow \) GRAY
6. \( d[s] \leftarrow 0 \)
7. \( \pi[s] \leftarrow \text{NIL} \)
8. \( Q \leftarrow \emptyset \)
9. Enqueue\( (Q, \, s) \)
10. while \( Q \neq \emptyset \)
11. \( \text{do} \) \( u \leftarrow \text{DEQUEUE}(Q) \)
12. for each \( v \in \text{Adj}[u] \)
13. \( \text{do if} \) color[\( v \)] = WHITE
14. \( \text{then} \) color[\( v \)] \( \leftarrow \) GRAY
15. \( d[v] \leftarrow d[u] + 1 \)
16. \( \pi[v] \leftarrow u \)
17. Enqueue\( (Q, \, v) \)
18. color[\( u \)] \( \leftarrow \) BLACK
Breadth First Search

BFS(G, s)
1. for each vertex u ∈ V[G] − {s}
2. do color[u] ← WHITE
3. d[u] ← ∞
4. π[u] ← NIL
5. color[s] ← GRAY
6. d[s] ← 0
7. π[s] ← NIL
8. Q ← ∅
9. ENQUEUE(Q, s)
10. while Q ≠ ∅
11. do u ← DEQUEUE(Q)
12. for each v ∈ Adj[u]
13. do if color[v] = WHITE
14. then color[v] ← GRAY
15. d[v] ← d[u] + 1
16. π[v] ← u
17. ENQUEUE(Q, v)
18. color[u] ← BLACK
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← ∅
9 ENQUEUE(Q, s)
10 while Q ≠ ∅
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS($G, s$)
1  for each vertex $u \in V[G] - \{s\}$
2      do $color[u] \leftarrow$ WHITE
3          $d[u] \leftarrow \infty$
4          $\pi[u] \leftarrow$ NIL
5  $color[s] \leftarrow$ GRAY
6  $d[s] \leftarrow 0$
7  $\pi[s] \leftarrow$ NIL
8  $Q \leftarrow \emptyset$
9  ENQUEUE($Q, s$)
10  while $Q \neq \emptyset$
11      do $u \leftarrow$ DEQUEUE($Q$)
12          for each $v \in Adj[u]$
13              do if $color[v] =$ WHITE
14                  then $color[v] \leftarrow$ GRAY
15                      $d[v] \leftarrow d[u] + 1$
16                      $\pi[v] \leftarrow u$
17                      ENQUEUE($Q, v$)
18      $color[u] \leftarrow$ BLACK
Breadth First Search

BFS\((G, s)\)

1. for each vertex \( u \in V[G] - \{s\} \)
2. \( \text{do color}[u] \leftarrow \text{WHITE} \)
3. \( d[u] \leftarrow \infty \)
4. \( \pi[u] \leftarrow \text{NIL} \)
5. \( \text{color}[s] \leftarrow \text{GRAY} \)
6. \( d[s] \leftarrow 0 \)
7. \( \pi[s] \leftarrow \text{NIL} \)
8. \( Q \leftarrow \emptyset \)
9. \( \text{ENQUEUE}(Q, s) \)
10. while \( Q \neq \emptyset \)
11. \( \text{do } u \leftarrow \text{DEQUEUE}(Q) \)
12. \( \text{for each } v \in \text{Adj}[u] \)
13. \( \text{do if color}[v] = \text{WHITE} \)
14. \( \text{then color}[v] \leftarrow \text{GRAY} \)
15. \( d[v] \leftarrow d[u] + 1 \)
16. \( \pi[v] \leftarrow u \)
17. \( \text{ENQUEUE}(Q, v) \)
18. \( \text{color}[u] \leftarrow \text{BLACK} \)

Abbas Malik: FCIT, King Abdulaziz University
Breadth First Search

BFS\( (G, s) \)
1. for each vertex \( u \in V[G] - \{s\} \)
2. do \( \text{color}[u] \leftarrow \text{WHITE} \)
3. \( d[u] \leftarrow \infty \)
4. \( \pi[u] \leftarrow \text{NIL} \)
5. \( \text{color}[s] \leftarrow \text{GRAY} \)
6. \( d[s] \leftarrow 0 \)
7. \( \pi[s] \leftarrow \text{NIL} \)
8. \( Q \leftarrow \emptyset \)
9. \( \text{ENQUEUE}(Q, s) \)
10. while \( Q \neq \emptyset \)
11. do \( u \leftarrow \text{DEQUEUE}(Q) \)
12. for each \( v \in \text{Adj}[u] \)
13. do if \( \text{color}[v] = \text{WHITE} \)
14. then \( \text{color}[v] \leftarrow \text{GRAY} \)
15. \( d[v] \leftarrow d[u] + 1 \)
16. \( \pi[v] \leftarrow u \)
17. \( \text{ENQUEUE}(Q, v) \)
18. \( \text{color}[u] \leftarrow \text{BLACK} \)
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. do $color[u] \leftarrow \text{WHITE}$
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow \text{NIL}$
5. $color[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. ENQUEUE($Q, s$)
10. while $Q \neq \emptyset$
11. do $u \leftarrow \text{DEQUEUE}(Q)$
12. for each $v \in \text{Adj}[u]$
13. do if $color[v] = \text{WHITE}$
14. then $color[v] \leftarrow \text{GRAY}$
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. ENQUEUE($Q, v$)
18. $color[u] \leftarrow \text{BLACK}$
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. \hspace{1em} do $\text{color}[u] \leftarrow \text{WHITE}$
3. \hspace{1em} $d[u] \leftarrow \infty$
4. \hspace{1em} $\pi[u] \leftarrow \text{NIL}$
5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. ENQUEUE($Q, s$)
10. while $Q \neq \emptyset$
11. \hspace{1em} do $u \leftarrow \text{DEQUEUE}(Q)$
12. \hspace{1em} for each $v \in \text{Adj}[u]$
13. \hspace{1em} \hspace{1em} do if $\text{color}[v] = \text{WHITE}$
14. \hspace{1em} \hspace{1em} then $\text{color}[v] \leftarrow \text{GRAY}$
15. \hspace{1em} \hspace{1em} $d[v] \leftarrow d[u] + 1$
16. \hspace{1em} \hspace{1em} $\pi[v] \leftarrow u$
17. \hspace{1em} \hspace{1em} ENQUEUE($Q, v$)
18. $\text{color}[u] \leftarrow \text{BLACK}$
Breadth First Search

\( \text{BFS}(G, s) \)
1. for each vertex \( u \in V[G] - \{s\} \)
2. do \( \text{color}[u] \leftarrow \text{WHITE} \)
3. \( d[u] \leftarrow \infty \)
4. \( \pi[u] \leftarrow \text{NIL} \)
5. \( \text{color}[s] \leftarrow \text{GRAY} \)
6. \( d[s] \leftarrow 0 \)
7. \( \pi[s] \leftarrow \text{NIL} \)
8. \( Q \leftarrow \emptyset \)
9. \( \text{ENQUEUE}(Q, s) \)
10. while \( Q \neq \emptyset \)
11. do \( u \leftarrow \text{DEQUEUE}(Q) \)
12. for each \( v \in \text{Adj}[u] \)
13. do if \( \text{color}[v] = \text{WHITE} \)
14. then \( \text{color}[v] \leftarrow \text{GRAY} \)
15. \( d[v] \leftarrow d[u] + 1 \)
16. \( \pi[v] \leftarrow u \)
17. \( \text{ENQUEUE}(Q, v) \)
18. \( \text{color}[u] \leftarrow \text{BLACK} \)
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← Ø
9 ENQUEUE(Q, s)
10 while Q ≠ Ø
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. do $\text{color}[u] \leftarrow \text{WHITE}$
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow \text{NIL}$
5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. ENQUEUE($Q, s$)
10. while $Q \neq \emptyset$
11. do $u \leftarrow \text{DEQUEUE}(Q)$
12. for each $v \in \text{Adj}[u]$
13. do if $\text{color}[v] = \text{WHITE}$
14. then $\text{color}[v] \leftarrow \text{GRAY}$
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. ENQUEUE($Q, v$)
18. $\text{color}[u] \leftarrow \text{BLACK}$
Breadth First Search

BFS(G, s)
1   for each vertex u ∈ V[G] − {s}
2      do color[u] ← WHITE
3          d[u] ← ∞
4          π[u] ← NIL
5   color[s] ← GRAY
6   d[s] ← 0
7   π[s] ← NIL
8   Q ← ∅
9   Enqueue(Q, s)
10  while Q ≠ ∅
11     do u ← Dequeue(Q)
12        for each v ∈ Adj[u]
13           do if color[v] = WHITE
14              then color[v] ← GRAY
15              d[v] ← d[u] + 1
16              π[v] ← u
17              Enqueue(Q, v)
18          color[u] ← BLACK
Breadth First Search

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\)
2. \(\color{u} \leftarrow \text{WHITE}\)
3. \(d[u] \leftarrow \infty\)
4. \(\pi[u] \leftarrow \text{NIL}\)
5. \(\color{s} \leftarrow \text{GRAY}\)
6. \(d[s] \leftarrow 0\)
7. \(\pi[s] \leftarrow \text{NIL}\)
8. \(Q \leftarrow \emptyset\)
9. \(\text{ENQUEUE}(Q, s)\)
10. while \(Q \neq \emptyset\)
11. \(\text{do } u \leftarrow \text{DEQUEUE}(Q)\)
12. \(\text{for each } v \in \text{Adj}[u]\)
13. \(\text{do if } color[v] = \text{WHITE}\)
14. \(\text{then } color[v] \leftarrow \text{GRAY}\)
15. \(d[v] \leftarrow d[u] + 1\)
16. \(\pi[v] \leftarrow u\)
17. \(\text{ENQUEUE}(Q, v)\)
18. \(\color[u] \leftarrow \text{BLACK}\)
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← Ø
9 ENQUEUE(Q, s)
10 while Q != Ø
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\)
2. do color\([u]\) \(\leftarrow\) WHITE
3. \(d[u] \leftarrow \infty\)
4. \(\pi[u] \leftarrow NIL\)
5. color\([s]\) \(\leftarrow\) GRAY
6. \(d[s] \leftarrow 0\)
7. \(\pi[s] \leftarrow NIL\)
8. \(Q \leftarrow \emptyset\)
9. Enqueue\((Q, s)\)
10. while \(Q \neq \emptyset\)
11. do \(u \leftarrow Dequeue(Q)\)
12. for each \(v \in Adj[u]\)
13. do if color\([v]\) = WHITE
14. then color\([v]\) \(\leftarrow\) GRAY
15. \(d[v] \leftarrow d[u] + 1\)
16. \(\pi[v] \leftarrow u\)
17. Enqueue\((Q, v)\)
18. color\([u]\) \(\leftarrow\) BLACK
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← ∅
9 ENQUEUE(Q, s)
10 while Q ̸= ∅
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← Ø
9 ENQUEUE(Q, s)
10 while Q ≠ Ø
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\)
2. do \(\text{color}[u] \leftarrow \text{WHITE}\)
3. \(d[u] \leftarrow \infty\)
4. \(\pi[u] \leftarrow \text{NIL}\)
5. \(\text{color}[s] \leftarrow \text{GRAY}\)
6. \(d[s] \leftarrow 0\)
7. \(\pi[s] \leftarrow \text{NIL}\)
8. \(Q \leftarrow \emptyset\)
9. \(\text{ENQUEUE}(Q, s)\)
10. while \(Q \neq \emptyset\)
11. do \(u \leftarrow \text{DEQUEUE}(Q)\)
12. for each \(v \in \text{Adj}[u]\)
13. do if \(\text{color}[v] = \text{WHITE}\)
14. then \(\text{color}[v] \leftarrow \text{GRAY}\)
15. \(d[v] \leftarrow d[u] + 1\)
16. \(\pi[v] \leftarrow u\)
17. \(\text{ENQUEUE}(Q, v)\)
18. \(\text{color}[u] \leftarrow \text{BLACK}\)
Breadth First Search

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\) do color\([u]\) ← WHITE
2. \(d[u] \leftarrow \infty\)
3. \(\pi[u] \leftarrow \text{NIL}\)
4. color\([s]\) ← GRAY
5. \(d[s] \leftarrow 0\)
6. \(\pi[s] \leftarrow \text{NIL}\)
7. \(Q \leftarrow \emptyset\)
8. \(\text{ENQUEUE}(Q, s)\)
9. while \(Q \neq \emptyset\) do
   10. \(u \leftarrow \text{DEQUEUE}(Q)\)
   11. for each \(v \in \text{Adj}[u]\) do
       12. if color\([v]\) = WHITE then
           13. color\([v]\) ← GRAY
           14. \(d[v] \leftarrow d[u] + 1\)
           15. \(\pi[v] \leftarrow u\)
           16. \(\text{ENQUEUE}(Q, v)\)
       17. color\([u]\) ← BLACK
Breadth First Search

\[ \text{BFS}(G, s) \]
1. for each vertex \( u \in V[G] - \{s\} \)
   2. do \( \text{color}[u] \leftarrow \text{WHITE} \)
   3. \( d[u] \leftarrow \infty \)
   4. \( \pi[u] \leftarrow \text{NIL} \)
   5. \( \text{color}[s] \leftarrow \text{GRAY} \)
   6. \( d[s] \leftarrow 0 \)
   7. \( \pi[s] \leftarrow \text{NIL} \)
   8. \( Q \leftarrow \emptyset \)
   9. \( \text{ENQUEUE}(Q, s) \)
10. while \( Q \neq \emptyset \)
    11. do \( u \leftarrow \text{DEQUEUE}(Q) \)
    12. for each \( v \in \text{Adj}[u] \)
    13. do if \( \text{color}[v] = \text{WHITE} \)
    14. then \( \text{color}[v] \leftarrow \text{GRAY} \)
    15. \( d[v] \leftarrow d[u] + 1 \)
    16. \( \pi[v] \leftarrow u \)
    17. \( \text{ENQUEUE}(Q, v) \)
18. \( \text{color}[u] \leftarrow \text{BLACK} \)
Breadth First Search

\[
BFS(G, s)
\]

\[
\begin{align*}
&1 \quad \text{for each vertex } u \in V[G] - \{s\} \\
&2 \quad \text{do color}[u] \leftarrow \text{WHITE} \\
&3 \quad d[u] \leftarrow \infty \\
&4 \quad \pi[u] \leftarrow \text{NIL} \\
&5 \quad \text{color}[s] \leftarrow \text{GRAY} \\
&6 \quad d[s] \leftarrow 0 \\
&7 \quad \pi[s] \leftarrow \text{NIL} \\
&8 \quad Q \leftarrow \emptyset \\
&9 \quad \text{ENQUEUE}(Q, s) \\
&10 \quad \text{while } Q \neq \emptyset \\
&11 \quad \text{do } u \leftarrow \text{DEQUEUE}(Q) \\
&12 \quad \text{for each } v \in \text{Adj}[u] \\
&13 \quad \text{do if color}[v] = \text{WHITE} \\
&14 \quad \quad \text{then color}[v] \leftarrow \text{GRAY} \\
&15 \quad \quad d[v] \leftarrow d[u] + 1 \\
&16 \quad \quad \pi[v] \leftarrow u \\
&17 \quad \quad \text{ENQUEUE}(Q, v) \\
&18 \quad \quad color[u] \leftarrow \text{BLACK}
\end{align*}
\]
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← ∅
9 ENQUEUE(Q, s)
10 while Q ≠ ∅
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS\( (G, s) \)
1 for each vertex \( u \in V[G] - \{s\} \)
2 do color[\( u \)] ← WHITE
3 \( d[u] \leftarrow \infty \)
4 \( \pi[u] \leftarrow \text{NIL} \)
5 color[\( s \)] ← GRAY
6 \( d[s] \leftarrow 0 \)
7 \( \pi[s] \leftarrow \text{NIL} \)
8 \( Q \leftarrow \emptyset \)
9 ENQUEUE\( (Q, s) \)
10 while \( Q \neq \emptyset \)
11 do \( u \leftarrow \text{DEQUEUE}(Q) \)
12 for each \( v \in \text{Adj}[u] \)
13 do if color[\( v \)] = WHITE
14 then color[\( v \)] ← GRAY
15 \( d[v] \leftarrow d[u] + 1 \)
16 \( \pi[v] \leftarrow u \)
17 \( \text{ENQUEUE}(Q, v) \)
18 color[\( u \)] ← BLACK
Breadth First Search

BFS($G$, $s$)
1. for each vertex $u \in V[G] - \{s\}$
2. do $color[u] \leftarrow \text{WHITE}$
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow \text{NIL}$
5. $color[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. $\text{ENQUEUE}(Q, s)$
10. while $Q \neq \emptyset$
11. do $u \leftarrow \text{DEQUEUE}(Q)$
12. for each $v \in Adj[u]$
13. do if $color[v] = \text{WHITE}$
14. then $color[v] \leftarrow \text{GRAY}$
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. $\text{ENQUEUE}(Q, v)$
18. $color[u] \leftarrow \text{BLACK}$
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← ∅
9 ENQUEUE(Q, s)
10 while Q ≠ ∅
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS\( (G, s) \)
1. for each vertex \( u \in V[G] - \{s\} \)
2. do \( \text{color}[u] \leftarrow \text{WHITE} \)
3. \( d[u] \leftarrow \infty \)
4. \( \pi[u] \leftarrow \text{NIL} \)
5. \( \text{color}[s] \leftarrow \text{GRAY} \)
6. \( d[s] \leftarrow 0 \)
7. \( \pi[s] \leftarrow \text{NIL} \)
8. \( Q \leftarrow \emptyset \)
9. \( \text{ENQUEUE}(Q, s) \)
10. while \( Q \neq \emptyset \)
11. do \( u \leftarrow \text{DEQUEUE}(Q) \)
12. for each \( v \in \text{Adj}[u] \)
13. do if \( \text{color}[v] = \text{WHITE} \)
14. then \( \text{color}[v] \leftarrow \text{GRAY} \)
15. \( d[v] \leftarrow d[u] + 1 \)
16. \( \pi[v] \leftarrow u \)
17. \( \text{ENQUEUE}(Q, v) \)
18. \( \text{color}[u] \leftarrow \text{BLACK} \)
Breadth First Search

BFS(G, s)
1. for each vertex u ∈ V[G] − {s}
2. do color[u] ← WHITE
3. d[u] ← ∞
4. π[u] ← NIL
5. color[s] ← GRAY
6. d[s] ← 0
7. π[s] ← NIL
8. Q ← Ø
9. ENQUEUE(Q, s)
10. while Q ≠ Ø
11. do u ← DEQUEUE(Q)
12. for each v ∈ Adj[u]
13. do if color[v] = WHITE
14. then color[v] ← GRAY
15. d[v] ← d[u] + 1
16. π[v] ← u
17. ENQUEUE(Q, v)
18. color[u] ← BLACK
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. \hspace{1em} do $\text{color}[u] \leftarrow \text{WHITE}$
3. \hspace{2em} $d[u] \leftarrow \infty$
4. \hspace{2em} $\pi[u] \leftarrow \text{NIL}$
5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. $\text{ENQUEUE}(Q, s)$
10. while $Q \neq \emptyset$
11. \hspace{1em} do $u \leftarrow \text{DEQUEUE}(Q)$
12. \hspace{2em} for each $v \in \text{Adj}[u]$
13. \hspace{3em} do if $\text{color}[v] = \text{WHITE}$
14. \hspace{4em} then $\text{color}[v] \leftarrow \text{GRAY}$
15. \hspace{4em} $d[v] \leftarrow d[u] + 1$
16. \hspace{4em} $\pi[v] \leftarrow u$
17. \hspace{4em} $\text{ENQUEUE}(Q, v)$
18. $\text{color}[u] \leftarrow \text{BLACK}$
Breadth First Search

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\) do
2. \(color[u] \leftarrow \text{WHITE}\)
3. \(d[u] \leftarrow \infty\)
4. \(\pi[u] \leftarrow \text{NIL}\)
5. \(color[s] \leftarrow \text{GRAY}\)
6. \(d[s] \leftarrow 0\)
7. \(\pi[s] \leftarrow \text{NIL}\)
8. \(Q \leftarrow \emptyset\)
9. \(\text{ENQUEUE}(Q, s)\)
10. while \(Q \neq \emptyset\)
11. do \(u \leftarrow \text{DEQUEUE}(Q)\)
12. for each \(v \in \text{Adj}[u]\) do
13. if \(color[v] = \text{WHITE}\) then
14. \(color[v] \leftarrow \text{GRAY}\)
15. \(d[v] \leftarrow d[u] + 1\)
16. \(\pi[v] \leftarrow u\)
17. \(\text{ENQUEUE}(Q, v)\)
18. \(color[u] \leftarrow \text{BLACK}\)
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. do $color[u] \leftarrow$ WHITE
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow$ NIL
5. $color[s] \leftarrow$ GRAY
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow$ NIL
8. $Q \leftarrow \emptyset$
9. ENQUEUE($Q, s$)
10. while $Q \neq \emptyset$
11. do $u \leftarrow$ DEQUEUE($Q$)
12. do for each $v \in Adj[u]$
13. do if $color[v] =$ WHITE
14. then $color[v] \leftarrow$ GRAY
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. ENQUEUE($Q, v$)
18. $color[u] \leftarrow$ BLACK
Breadth First Search

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. do $\text{color}[u] \leftarrow \text{WHITE}$
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow \text{NIL}$
5. $\text{color}[s] \leftarrow \text{GRAY}$
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow \text{NIL}$
8. $Q \leftarrow \emptyset$
9. $\text{ENQUEUE}(Q, s)$
10. while $Q \neq \emptyset$
11. do $u \leftarrow \text{DEQUEUE}(Q)$
12. for each $v \in \text{Adj}[u]$
13. do if $\text{color}[v] = \text{WHITE}$
14. then $\text{color}[v] \leftarrow \text{GRAY}$
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. $\text{ENQUEUE}(Q, v)$
18. $\text{color}[u] \leftarrow \text{BLACK}$

Abbas Malik: FCIT, King Abdulaziz University
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] − {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← ∅
9 ENQUEUE(Q, s)
10 while Q ≠ ∅
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS(G, s)
1 for each vertex u ∈ V[G] – {s}
2 do color[u] ← WHITE
3 d[u] ← ∞
4 π[u] ← NIL
5 color[s] ← GRAY
6 d[s] ← 0
7 π[s] ← NIL
8 Q ← Ø
9 ENQUEUE(Q, s)
10 while Q ≠ Ø
11 do u ← DEQUEUE(Q)
12 for each v ∈ Adj[u]
13 do if color[v] = WHITE
14 then color[v] ← GRAY
15 d[v] ← d[u] + 1
16 π[v] ← u
17 ENQUEUE(Q, v)
18 color[u] ← BLACK
Breadth First Search

BFS(G, s)
1    for each vertex u ∈ V[G] − {s}
2        do color[u] ← WHITE
3        d[u] ← ∞
4        π[u] ← NIL
5    color[s] ← GRAY
6    d[s] ← 0
7    π[s] ← NIL
8    Q ← ∅
9    ENQUEUE(Q, s)
10   while Q ̸= ∅
11      do u ← DEQUEUE(Q)
12      for each v ∈ Adj[u]
13          do if color[v] = WHITE
14             then color[v] ← GRAY
15             d[v] ← d[u] + 1
16             π[v] ← u
17             ENQUEUE(Q, v)
18          color[u] ← BLACK
Breadth First Search

BFS(G, s)
1   for each vertex \( u \in V[G] - \{s\} \)
2     do color[\( u \)] \( \leftarrow \) WHITE
3     \( d[\( u \)] \leftarrow \infty \)
4     \( \pi[\( u \)] \leftarrow \text{NIL} \)
5     color[\( s \)] \( \leftarrow \) GRAY
6     \( d[\( s \)] \leftarrow 0 \)
7     \( \pi[\( s \)] \leftarrow \text{NIL} \)
8     \( Q \leftarrow \emptyset \)
9     ENQUEUE(\( Q, s \))
10    while \( Q \neq \emptyset \)
11       do \( u \leftarrow \text{DEQUEUE}(Q) \)
12          for each \( v \in \text{Adj}[\( u \)] \)
13             do if color[\( v \)] = WHITE
14                then color[\( v \)] \( \leftarrow \) GRAY
15                  \( d[\( v \)] \leftarrow d[\( u \)] + 1 \)
16                  \( \pi[\( v \)] \leftarrow u \)
17                  ENQUEUE(\( Q, v \))
18             color[\( u \)] \( \leftarrow \) BLACK
Analysis of Breadth First Search

Initialization cost is $O(V)$, because we are initializing all vertices in the graph.

```plaintext
BFS(G, s)
for each vertex $u \in V[G] - \{s\}$
do $color[u] \leftarrow$ WHITE
d[u] $\leftarrow \infty$
$\pi[u] \leftarrow$ NIL
$color[s] \leftarrow$ GRAY
d[s] $\leftarrow 0$
$\pi[s] \leftarrow$ NIL
$Q \leftarrow \emptyset$
ENQUEUE($Q$, s)
while $Q \neq \emptyset$
do $u \leftarrow$ DEQUEUE($Q$)
for each $v \in Adj[u]$
do if $color[v] =$ WHITE
then $color[v] \leftarrow$ GRAY
d[v] $\leftarrow$ d[u] + 1
$\pi[v] \leftarrow u$
ENQUEUE($Q$, v)
color[u] $\leftarrow$ BLACK
```
Analysis of Breadth First Search

Initialization cost is $O(V)$, because we are initializing all vertices in the graph.

In this part, the complexity depends on the total number of edges in the graph, i.e. $O(E)$. 

```
BFS(G, s)

for each vertex $u \in V[G] - \{s\}$
do $color[u] \leftarrow WHITE$
d[$u] \leftarrow \infty$
$\pi[u] \leftarrow NIL$

$color[s] \leftarrow GRAY$
d[$s] \leftarrow 0$
$\pi[s] \leftarrow NIL$

$Q \leftarrow \emptyset$

ENQUEUE($Q, s$)

while $Q \neq \emptyset$
do $u \leftarrow DEQUEUE(Q)$
for each $v \in Adj[u]$
do if $color[v] = WHITE$
then $color[v] \leftarrow GRAY$
d[$v] \leftarrow d[u] + 1$
$\pi[v] \leftarrow u$
ENQUEUE($Q, v$)

$color[u] \leftarrow BLACK$
```
Analysis of Breadth First Search

Total Running Time = O(V+E) = Linear

Initialization cost is O(V), because we are initializing all vertices in the graph.

In this part, the complexity depends on the total number of edges in the graph, i.e. O(E).
**Shortest Path Distance**

BFS$(G, s)$

1. for each vertex $u \in V[G] - \{s\}$
2. \hspace{5mm} do color$[u] \leftarrow$ WHITE
3. \hspace{5mm} $d[u] \leftarrow \infty$
4. \hspace{5mm} $\pi[u] \leftarrow$ NIL
5. color$[s] \leftarrow$ GRAY
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow$ NIL
8. $Q \leftarrow \emptyset$
9. ENQUEUE$(Q, s)$
10. while $Q \neq \emptyset$
11. \hspace{5mm} do $u \leftarrow$ DEQUEUE$(Q)$
12. \hspace{10mm} for each $v \in Adj[u]$
13. \hspace{15mm} do if color$[v] = \text{WHITE}$
14. \hspace{20mm} then color$[v] \leftarrow$ GRAY
15. \hspace{15mm} $d[v] \leftarrow d[u] + 1$
16. \hspace{15mm} $\pi[v] \leftarrow u$
17. \hspace{15mm} ENQUEUE$(Q, v)$
18. \hspace{10mm} color$[u] \leftarrow$ BLACK
Shortest Path Distance

BFS\((G, s)\)
1. for each vertex \(u \in V[G] - \{s\}\) 
2. \(\text{do } color[u] \leftarrow \text{WHITE}\) 
3. \(d[u] \leftarrow \infty\) 
4. \(\pi[u] \leftarrow \text{NIL}\) 
5. \(color[s] \leftarrow \text{GRAY}\) 
6. \(d[s] \leftarrow 0\) 
7. \(\pi[s] \leftarrow \text{NIL}\) 
8. \(Q \leftarrow \emptyset\) 
9. \(\text{ENQUEUE}(Q, s)\) 
10. while \(Q \neq \emptyset\) 
11. \(\text{do } u \leftarrow \text{DEQUEUE}(Q)\) 
12. \(\text{for each } v \in Adj[u]\) 
13. \(\text{do if } color[v] = \text{WHITE}\) 
14. \(\text{then } color[v] \leftarrow \text{GRAY}\) 
15. \(d[v] \leftarrow d[u] + 1\) 
16. \(\pi[v] \leftarrow u\) 
17. \(\text{ENQUEUE}(Q, v)\) 
18. \(color[u] \leftarrow \text{BLACK}\)
Shortest Path Distance

BFS($G, s$)
1. for each vertex $u \in V[G] - \{s\}$
2. do $color[u] \leftarrow$ WHITE
3. $d[u] \leftarrow \infty$
4. $\pi[u] \leftarrow$ NIL
5. $color[s] \leftarrow$ GRAY
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow$ NIL
8. $Q \leftarrow \emptyset$
9. Enqueue($Q, s$)
10. while $Q \neq \emptyset$
11. do $u \leftarrow$ Dequeue($Q$)
12. for each $v \in Adj[u]$
13. do if $color[v] =$ WHITE
14. then $color[v] \leftarrow$ GRAY
15. $d[v] \leftarrow d[u] + 1$
16. $\pi[v] \leftarrow u$
17. Enqueue($Q, v$)
18. $color[u] \leftarrow$ BLACK
Breadth-first search (BFS)

- Visits graph vertices by moving across to all the neighbors of last visited vertex
- BFS uses a queue
- Similar to level-by-level tree traversal
- “Redraws” graph in tree-like fashion (with tree edges and cross edges for undirected graph)
Pseudocode of BFS

ALGORITHM \textit{BFS}(G)

//Implements a breadth-first search traversal of a given graph
//Input: Graph \(G = (V, E)\)
//Output: Graph \(G\) with its vertices marked with consecutive integers
//in the order they have been visited by the BFS traversal
mark each vertex in \(V\) with 0 as a mark of being “unvisited”
\(count \leftarrow 0\)
\textbf{for} each vertex \(v\) in \(V\) \textbf{do}
\hspace{1em} if \(v\) is marked with 0
\hspace{1.5em} \textit{bfs}(v)
\textit{bfs}(v)
//visits all the unvisited vertices connected to vertex \(v\) by a path
//and assigns them the numbers in the order they are visited
//via global variable \(count\)
\(count \leftarrow count + 1;\) \hspace{1em} mark \(v\) with \(count\) and initialize a queue with \(v\)
\textbf{while} the queue is not empty \textbf{do}
\hspace{1em} \textbf{for} each vertex \(w\) in \(V\) adjacent to the front vertex \textbf{do}
\hspace{2em} if \(w\) is marked with 0
\hspace{3em} \(count \leftarrow count + 1;\) \hspace{1em} mark \(w\) with \(count\)
\hspace{3em} add \(w\) to the queue
\hspace{2em} remove the front vertex from the queue
Notes on BFS

• BFS can be implemented with graphs represented as:
  • adjacency matrices: $\Theta(V^2)$
  • adjacency lists: $\Theta(|V|+|E|)$

• Yields single ordering of vertices (order added/deleted from queue is the same)

• Applications:
  • checking connectivity, finding connected components
  • checking acyclicity
  • finding articulation points and biconnected components
  • searching state-space of problems for solution (AI)

Abbas Malik: FCIT, King Abdulaziz University
Depth First Search

• To search deeper in the graph whenever possible
• Edges are explored out of the most recently discovered vertex \( v \) that still has unexplored edges
• When all edges of \( v \) are explored, the search “backtracks” to explore edges leaving the vertex from which \( v \) was discovered
• It discovers all the vertices that are reachable from the original source vertex
• If any undiscovered vertices remain, then one of them is selected as a new source and the search is repeated from that source
Depth-First Search (DFS)

- Visits graph’s vertices by always moving away from last visited vertex to unvisited one, backtracks if no adjacent unvisited vertex is available.

- Uses a stack
  - a vertex is pushed onto the stack when it’s reached for the first time
  - a vertex is popped off the stack when it becomes a dead end, i.e., when there is no adjacent unvisited vertex

- “Redraws” graph in tree-like fashion (with tree edges and back edges for undirected graph)
Depth First Search

- Whenever a vertex $v$ is discovered during a scan of the adjacency list of an already discovered vertex $u$, depth first search records this event by setting $v$’s parent field (predecessor) $\pi(v) = u$
- Unlike Breadth First Search whose predecessor graph forms a tree (Breadth First Tree), the predecessor graph produced by a Depth First Search may be composed of several trees because the search is repeated from multiple sources (recursive algorithm)
- DFS forms a Depth First Forest composed of several Depth First Trees
Depth First Search

- To keep track of progress, DFS algorithm colors each vertex white (undiscovered), gray (discovered first time) or black (a vertex whose all adjacent vertices have been discovered).
- We assume that the input graph is represented in Adjacency List Representation.
- DFS timestamps each vertex during search.
  - $d[\nu] \rightarrow$ timestamp that records when $\nu$ is first discovered
  - $f[\nu] \rightarrow$ timestamp that records when the search finishes examining $\nu$’s adjacency list and blackens $\nu$
Depth First Search

- The node in the Adjacency List Representation contains following information:
  1. color → to contain color information (white, gray or black)
  2. \( d \) → records the first discovered timestamp
  3. \( \pi \) → to contain the address of the parent vertex (predecessor)
  4. \( f \) → records the finishing of examination of \( v \)'s adjacency list

<table>
<thead>
<tr>
<th>( \pi )</th>
<th>Vertex Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>color</td>
</tr>
<tr>
<td>( f )</td>
<td></td>
</tr>
</tbody>
</table>
Depth First Search

**DFS (G)**

- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - \( \text{time} = 0 \)

- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - **DFS-Visit** (\( u \))

**DFS-Visit** (\( w \))

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - **DFS-Visit** (\( v \))

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$

For each vertex $u \in V[G]$
  - If $\text{color}[u] = \text{WHITE}$ then
    - **DFS-Visit** ($u$)

**Initialize the color, distance and parent of all vertices except the source**

**DFS-Visit** ($w$)

- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$

- For each vertex $v \in \text{Adj}[w]$
  - If $\text{color}[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - **DFS-Visit** ($v$)

- $f[w] = \text{time} = \text{time} + 1$
**Depth First Search**

### DFS (G)

- For each vertex \( u \in V[G] \)
  - color\[u\] = WHITE
  - \( \pi[u] = \text{NIL} \)
  - time = 0

- For each vertex \( u \in V[G] \)
  - If color\[u\] = WHITE then
    - DFS-Visit (u)

### Initialize

- Initialize the color, distance and parent of all vertices except the source

- Searches the graph by changing the source and repeating the search for each new source vertex using DFS-Visit

### DFS-Visit (w)

- color\[w\] = GRAY
- time = time + 1
- d[w] = time
- For each vertex \( v \in \text{Adj}[w] \)
  - If color\[v\] = WHITE then
    - \( \pi[v] = w \)
    - DFS-Visit(\( v \))
- color\[w\] = BLACK
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
  
  If \( \text{color}[u] = \text{WHITE} \) then
  
  DFS-Visit \((u)\)

Decorating the current discovered node with color and discovered timestamp

**Initialize the color, distance and parent of all vertices except the source**

**Searches the graph by changing the source and repeating the search for each new source vertex using DFS-Visit**

**DFS-Visit \((w)\)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
  
  If \( \text{color}[v] = \text{WHITE} \) then
  
  \( \pi[v] = w \)
  
  DFS-Visit \((v)\)

**color[w] = BLACK**

\( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

- For each vertex \( u \in V[G] \):
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - \( \text{time} = 0 \)

- For each vertex \( u \in V[G] \):
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit \( (u) \)

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)

- For each vertex \( v \in \text{Adj}[w] \):
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit \( (v) \)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)

**Initialization**

- Initialize the color, distance and parent of all vertices except the source

**Search**

- Searches the graph by changing the source and repeating the search for each new source vertex using DFS-Visit

**Decorating**

- Decorating the current discovered node with color and discovered timestamp

**Adjacency List**

- Discovering the adjacency list of vertex \( u \) by recursively calling DFS-Visit
Depth First Search

DFS (G)

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
- \( \text{DFS-Visit}(u) \)

Initialize the color, distance and parent of all vertices except the source

Searches the graph by changing the source and repeating the search for each new source vertex using DFS-Visit

DFS-Visit (w)

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
- \( \pi[v] = w \)
- \( \text{DFS-Visit}(v) \)

Blacken the vertex \( u \) as all its adjacent vertices have been discovered and finished discovering timestamp

Decorating the current discovered node with color and discovered timestamp

Discovering the adjacency list of vertex \( u \) by recursively calling DFS-Visit

Abbas Malik: FCIT, King Abdulaziz University
**Depth First Search**

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit (u)

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
  - \( \text{color}[w] = \text{BLACK} \)
  - \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex \( u \in V[G] \)

- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)

If \( \text{color}[u] = \text{WHITE} \) then:

- **DFS-Visit** (\( u \))

\[
\begin{array}{ll}
\text{time} & 0 \\
\text{u} & \text{nil}
\end{array}
\]

**DFS-Visit** (\( w \))

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  
  If \( \text{color}[v] = \text{WHITE} \) then:

  - \( \pi[v] = w \)
  
  - **DFS-Visit** (\( v \))

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - time = 0

  For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - \( \text{DFS-Visit}(u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- time = time + 1
- \( d[w] = \text{time} \)
- For each vertex \( \nu \in \text{Adj}[w] \)
  - If \( \text{color}[
u] = \text{WHITE} \) then
    - \( \pi[
u] = w \)
    - \( \text{DFS-Visit}(\nu) \)
- \( f[w] = \text{time} = \text{time} + 1 \)

---

Abbas Malik: FCIT, King Abdulaziz University
**Depth First Search**

**DFS (G)**

For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$

For each vertex $u \in V[G]$

If $\text{color}[u] = \text{WHITE}$ then
- $\text{DFS-Visit}(u)$

**DFS-Visit (w)**

- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$

For each vertex $v \in \text{Adj}[w]$

If $\text{color}[v] = \text{WHITE}$ then
- $\pi[v] = w$
- $\text{DFS-Visit}(v)$

$\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - **DFS-Visit (u)**

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - **DFS-Visit(v)**

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
  color[\( u \)] = WHITE
  \( \pi[\( u \)] = \text{NIL} \)
  time = 0
For each vertex \( u \in V[G] \)
  If color[\( u \)] = WHITE then
    DFS-Visit (\( u \))

DFS-Visit (\( w \))
  color[\( w \)] = GRAY
  time = time + 1
  d[\( w \)] = time
  For each vertex \( v \in \text{Adj}[\( w \)] \)
    If color[\( v \)] = WHITE then
      \( \pi[\( v \)] = \( w \) \)
      DFS-Visit(\( v \))
  color[\( w \)] = BLACK
  f[\( w \)] = time = time + 1
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit (u)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)
- For each vertex \( \nu \in \text{Adj}[w] \)
  - If \( \text{color}[
u] = \text{WHITE} \) then
    - \( \pi[
u] = w \)
    - DFS-Visit(\( \nu \))
  - \( \text{color}[w] = \text{BLACK} \)
  - \( \text{f}[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit (u)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
For each vertex \( v \in \text{Adj}[w] \)
- If \( \text{color}[v] = \text{WHITE} \) then
  - \( \pi[v] = w \)
  - DFS-Visit(v)
- \( f[w] = \text{time} = \text{time} + 1 \)

- \( a \)
- \( nil \)
- \( d \)
- \( b \)
- \( nil \)
- \( e \)
- \( c \)
- \( nil \)
- \( g \)
Depth First Search

**DFS (G)**
For each vertex $u \in V[G]$
  - $\text{color}[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
- $\text{time} = 0$
For each vertex $u \in V[G]$
  - If $\text{color}[u] = \text{WHITE}$ then
    - DFS-Visit ($u$)

**DFS-Visit ($w$)**
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
  - For each vertex $v \in \text{Adj}[w]$
    - If $\text{color}[v] = \text{WHITE}$ then
      - $\pi[v] = w$
      - DFS-Visit($v$)
- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
**Depth First Search**

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
DFS-Visit (\( u \))

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(\( v \))
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit (u)

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
  DFS-Visit (u)

DFS-Visit (w)
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
For each vertex \( \nu \in \text{Adj}[w] \)
  If \( \text{color}[
u] = \text{WHITE} \) then
    \( \pi[
u] = w \)
    DFS-Visit(\nu)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
- DFS-Visit (u)

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
- \( \text{DFS-Visit}(u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
- \( \pi[v] = w \)
- \( \text{DFS-Visit}(v) \)

- \( \text{color}[w] = \text{BLACK} \)
- \( \text{f}[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - \( \text{DFS-Visit}(u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)
  - For each vertex \( v \in \text{Adj}[w] \)
    - If \( \text{color}[v] = \text{WHITE} \) then
      - \( \pi[v] = w \)
      - \( \text{DFS-Visit}(v) \)
    - \( \text{color}[w] = \text{BLACK} \)
    - \( \text{f}[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - \( \pi[w] = u \)
  - \( \text{DFS-Visit}(u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
For each vertex \( v \in \text{Adj}[w] \)
- If \( \text{color}[v] = \text{WHITE} \) then
  - \( \pi[v] = w \)
  - \( \text{DFS-Visit}(v) \)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit (u)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$
For each vertex $u \in V[G]$
- If $\text{color}[u] = \text{WHITE}$ then
  - DFS-Visit ($u$)

**DFS-Visit ($w$)**
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
For each vertex $v \in \text{Adj}[w]$
- If $\text{color}[v] = \text{WHITE}$ then
  - $\pi[v] = w$
  - DFS-Visit($v$)
- $f[w] = \text{time} = \text{time} + 1$

**Example**
- $\text{time} = 2$
- $w \rightarrow b$
- $d[w] = 1$
- For each vertex $v \in \text{Adj}[w]$
- If $\text{color}[v] = \text{WHITE}$ then
  - $\pi[v] = w$
  - DFS-Visit($v$)
- $f[w] = 2$
Depth First Search

**DFS (G)**

For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$

For each vertex $u \in V[G]$
If $\text{color}[u] = \text{WHITE}$ then
- $\text{DFS-Visit}(u)$

**DFS-Visit (w)**
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$

For each vertex $v \in \text{Adj}[w]$
If $\text{color}[v] = \text{WHITE}$ then
- $\pi[v] = w$
- $\text{DFS-Visit}(v)$

- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
Depth First Search

DFS (G)
For each vertex $u \in V[G]$
  color[$u$] = WHITE
  $\pi[u] = \text{NIL}$
  time = 0
For each vertex $u \in V[G]$  
  If color[$u$] = WHITE then  
    DFS-Visit ($u$)

time | 2 
-----|---
$u$   | a
w     | b
v     | e

DFS-Visit (w)
  color[w] = GRAY
  time = time + 1
  d[w] = time
  
  For each vertex $v \in \text{Adj}[w]$
      If color[$v$] = WHITE then
          $\pi[v] = w$
          DFS-Visit($v$)
  
  color[w] = BLACK
  $f[w] = \text{time} = \text{time} + 1$
**Depth First Search**

**DFS (G)**
For each vertex $u \in V[G]$  
- $\text{color}[u] = \text{WHITE}$  
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$
For each vertex $u \in V[G]$  
- If $\text{color}[u] = \text{WHITE}$ then  
- DFS-Visit ($u$)

**DFS-Visit (w)**
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
For each vertex $v \in \text{Adj}[w]$
- If $\text{color}[v] = \text{WHITE}$ then  
  - $\pi[v] = w$
  - DFS-Visit($v$)
- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit (u)

DFS-Visit (w)
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
- \( \text{f}[w] = \text{time} = \text{time} + 1 \)

**Sample Graph**

```
DFS-Visit (w)
```

```
DFS-Visit (u)
```
Depth First Search

**DFS (G)**
For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- time = 0
For each vertex $u \in V[G]$
  If $\text{color}[u] = \text{WHITE}$ then
    DFS-Visit (u)

**DFS-Visit (w)**
- $\text{color}[w] = \text{GRAY}$
- time = time + 1
- $d[w] = \text{time}$
  For each vertex $v \in \text{Adj}[w]$
    If $\text{color}[v] = \text{WHITE}$ then
      $\pi[v] = w$
    DFS-Visit(v)
- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit \( (u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit\( (v) \)
  - \( \text{color}[w] = \text{BLACK} \)
  - \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- time = 0

For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit (u)

**DFS-Visit (w)**

1. \( \text{color}[w] = \text{GRAY} \)
2. time = time + 1
3. \( d[w] = \text{time} \)
4. For each vertex \( v \in \text{Adj}[w] \)
   - If \( \text{color}[v] = \text{WHITE} \) then
     - \( \pi[v] = w \)
     - DFS-Visit(v)
5. \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- time = 0
For each vertex \( u \in V[G] \)
  If \( \text{color}[u] = \text{WHITE} \) then
    DFS-Visit (u)

DFS-Visit (w)
- \( \text{color}[w] = \text{GRAY} \)
- time = time + 1
- \( d[w] = \text{time} \)
  For each vertex \( v \in \text{Adj}[w] \)
    If \( \text{color}[v] = \text{WHITE} \) then
      \( \pi[v] = w \)
      DFS-Visit(v)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$
For each vertex $u \in V[G]$
  - If $\text{color}[u] = \text{WHITE}$ then
    - DFS-Visit ($u$)

DFS-Visit ($w$)
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
For each vertex $\nu \in \text{Adj}[w]$
  - If $\text{color}[
u] = \text{WHITE}$ then
    - $\pi[\nu] = w$
    - DFS-Visit($\nu$)
- $f[w] = \text{time} = \text{time} + 1$

Abbas Malik: FCIT, King Abdulaziz University
Depth First Search

DFS (G)
For each vertex $u \in V[G]$
  - $\text{color}[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
  - $\text{time} = 0$
For each vertex $u \in V[G]$
  - If $\text{color}[u] = \text{WHITE}$ then
    - DFS-Visit ($u$)

DFS-Visit ($w$)
  - $\text{color}[w] = \text{GRAY}$
  - $\text{time} = \text{time} + 1$
  - $d[w] = \text{time}$
  - For each vertex $v \in \text{Adj}[w]$
    - If $\text{color}[v] = \text{WHITE}$ then
      - $\pi[v] = w$
      - DFS-Visit($v$)
  - $\text{color}[w] = \text{BLACK}$
  - $f[w] = \text{time} = \text{time} + 1$

Abbas Malik: FCIT, King Abdulaziz University
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- time = 0

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
- DFS-Visit (u)

DFS-Visit (w)
- \( \text{color}[w] = \text{GRAY} \)
- time = time + 1
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
- \( \pi[v] = w \)
- DFS-Visit(v)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex $u \in V[G]$
  - $color[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
- $time = 0$
- For each vertex $u \in V[G]$
  - If $color[u] = \text{WHITE}$ then
    - DFS-Visit ($u$)

**DFS-Visit (w)**
- $color[w] = \text{GRAY}$
- $time = time + 1$
- $d[w] = time$
- For each vertex $v \in \text{Adj}[w]$
  - If $color[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - DFS-Visit($v$)
- $color[w] = \text{BLACK}$
- $f[w] = time = time + 1$
Depth First Search

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - \( \text{DFS-Visit} (u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - \( \text{DFS-Visit}(v) \)
- \( \text{color}[w] = \text{BLACK} \)
- \( \text{f}[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - \( \text{DFS-Visit}(u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - **DFS-Visit(v)**
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex $u \in V[G]$
  $\text{color}[u] = \text{WHITE}$
  $\pi[u] = \text{NIL}$
$time = 0$
For each vertex $u \in V[G]$
  If $\text{color}[u] = \text{WHITE}$ then
    DFS-Visit (u)

DFS-Visit (w)
$\text{color}[w] = \text{GRAY}$
$time = time + 1$
$d[w] = time$
For each vertex $v \in \text{Adj}[w]$
  If $\text{color}[v] = \text{WHITE}$ then
    $\pi[v] = w$
    DFS-Visit($v$)
$\text{color}[w] = \text{BLACK}$
$f[w] = time = time + 1$
Depth First Search

DFS (G)
For each vertex $u \in V[G]$
   $\text{color}[u] = \text{WHITE}$
   $\pi[u] = \text{NIL}$
   $\text{time} = 0$
For each vertex $u \in V[G]$
   If $\text{color}[u] = \text{WHITE}$ then
     DFS-Visit (u)

DFS-Visit (w)
$\text{color}[w] = \text{GRAY}$
   $\text{time} = \text{time} + 1$
   $d[w] = \text{time}$
   For each vertex $v \in \text{Adj}[w]$
     If $\text{color}[v] = \text{WHITE}$ then
       $\pi[v] = w$
       DFS-Visit(v)
   $\text{color}[w] = \text{BLACK}$
   $f[w] = \text{time} = \text{time} + 1$
Depth First Search

DFS (G)
For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$
For each vertex $u \in V[G]$
  If $\text{color}[u] = \text{WHITE}$ then
    DFS-Visit ($u$)

DFS-Visit ($w$)
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
  For each vertex $v \in \text{Adj}[w]$
    If $\text{color}[v] = \text{WHITE}$ then
      $\pi[v] = w$
      DFS-Visit($v$)
  $\text{color}[w] = \text{BLACK}$
  $f[w] = \text{time} = \text{time} + 1$
**Depth First Search**

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
  If \( \text{color}[u] = \text{WHITE} \) then
  DFS-Visit (\( u \))

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
  If \( \text{color}[v] = \text{WHITE} \) then
    \( \pi[v] = w \)
    DFS-Visit (\( v \))

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- color\[u\] = WHITE
- \( \pi[u] = \text{NIL} \)
- time = 0

For each vertex \( u \in V[G] \)
- If color\[u\] = WHITE then
  - DFS-Visit (u)

**DFS-Visit (w)**

- color\[w\] = GRAY
- time = time + 1
- d\[w\] = time
- For each vertex \( v \in \text{Adj}[w] \)
  - If color\[v\] = WHITE then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
- color\[w\] = BLACK
- f\[w\] = time = time + 1
Depth First Search

DFS (G)
For each vertex $u \in V[G]$ 
- $\text{color}[u] = \text{WHITE}$ 
- $\pi[u] = \text{NIL}$ 
- $\text{time} = 0$
For each vertex $u \in V[G]$ 
If $\text{color}[u] = \text{WHITE}$ then 
- DFS-Visit ($u$)

DFS-Visit ($w$)
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$ 
  If $\text{color}[v] = \text{WHITE}$ then 
    $\pi[v] = w$
    DFS-Visit($v$)
- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
  DFS-Visit (u)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)
For each vertex \( v \in \text{Adj}[w] \)
  If \( \text{color}[v] = \text{WHITE} \) then
    \( \pi[v] = w \)
    DFS-Visit(v)
  If \( \text{color}[w] = \text{BLACK} \)
    \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit (u)

DFS-Visit (w)
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( \text{d}[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit (u)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
  - \( \text{color}[w] = \text{BLACK} \)
  - \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex $u \in V[G]$
  - $color[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
- $\text{time} = 0$
- For each vertex $u \in V[G]$
  - If $color[u] = \text{WHITE}$ then
    - DFS-Visit ($u$)

**DFS-Visit ($w$)**
- $color[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$
  - If $color[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - DFS-Visit($v$)
- $color[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - \( \text{DFS-Visit} (u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
For each vertex \( v \in \text{Adj}[w] \)
- If \( \text{color}[v] = \text{WHITE} \) then
  - \( \pi[v] = w \)
  - \( \text{DFS-Visit}(v) \)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
- DFS-Visit (u)

\[
\begin{align*}
\text{DFS-Visit (w)} \\
\text{color}[w] = \text{GRAY} \\
\text{time} = \text{time} + 1 \\
\text{d}[w] = \text{time} \\
\text{For each vertex } \nu \in \text{Adj}[w] \\
\text{If } \text{color}[
u] = \text{WHITE} \text{ then} \\
\pi[
u] = w \\
\text{DFS-Visit}(
u) \\
\text{color}[w] = \text{BLACK} \\
/ [w] = \text{time} = \text{time} + 1
\end{align*}
\]
Depth First Search

DFS (G)

For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- time = 0

For each vertex $u \in V[G]$
If $\text{color}[u] = \text{WHITE}$ then
  DFS-Visit ($u$)

DFS-Visit ($w$)
- $\text{color}[w] = \text{GRAY}$
- time = time + 1
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$
  If $\text{color}[v] = \text{WHITE}$ then
    $\pi[v] = w$
    DFS-Visit($v$)
- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit (u)

---

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
  - \( \pi[v] = w \)
  - DFS-Visit(v)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- time = 0
For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit (u)

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- time = time + 1
- \( d[w] = \text{time} \)
- For each vertex \( \nu \in \text{Adj}[w] \)
  - If \( \text{color}[\nu] = \text{WHITE} \) then
    - \( \pi[\nu] = w \)
    - DFS-Visit(\nu)
    - \( \text{color}[w] = \text{BLACK} \)
    - \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
  If \( \text{color}[u] = \text{WHITE} \) then
  DFS-Visit (u)

DFS-Visit (w)
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
  For each vertex \( v \in \text{Adj}[w] \)
    If \( \text{color}[v] = \text{WHITE} \) then
      \( \pi[v] = w \)
      DFS-Visit(v)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( color[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
  If \( color[u] = \text{WHITE} \) then
    DFS-Visit (u)

**DFS-Visit (w)**
- \( color[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
  For each vertex \( v \in \text{Adj}[w] \)
    If \( color[v] = \text{WHITE} \) then
      \( \pi[v] = w \)
      DFS-Visit(v)
  \( color[w] = \text{BLACK} \)
  \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$

For each vertex $u \in V[G]$
- If $\text{color}[u] = \text{WHITE}$ then
  - DFS-Visit $(u)$

**DFS-Visit (w)**

- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$

For each vertex $v \in \text{Adj}[w]$
- If $\text{color}[v] = \text{WHITE}$ then
  - $\pi[v] = w$
  - DFS-Visit$(v)$
- $f[w] = \text{time}$
- $\text{time} = \text{time} + 1$
Depth First Search

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - **DFS-Visit (u)**

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - **DFS-Visit(v)**
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$

For each vertex $u \in V[G]$
If $\text{color}[u] = \text{WHITE}$ then
  - $\text{DFS-Visit (u)}$

<table>
<thead>
<tr>
<th>time</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>a</td>
</tr>
</tbody>
</table>

**DFS-Visit (w)**
- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$
  - If $\text{color}[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - $\text{DFS-Visit}(v)$
  - $\text{color}[w] = \text{BLACK}$
  - $f[w] = \text{time} = \text{time} + 1$
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
\[
\text{color}[u] = \text{WHITE} \\
\pi[u] = \text{NIL} \\
time = 0
\]
For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
DFS-Visit (u)

DFS-Visit (w)
\[
\text{color}[w] = \text{GRAY} \\
time = \text{time} + 1 \\
d[w] = \text{time} \\
\text{For each vertex } v \in \text{Adj}[w] \\
\text{If } \text{color}[v] = \text{WHITE} \text{ then } \\
\pi[v] = w \\
\text{DFS-Visit}(v) \\
\text{color}[w] = \text{BLACK} \\
f[w] = \text{time} = \text{time} + 1
\]
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- color\([u]\) = WHITE
- \( \pi[u] = \text{NIL} \)
- time = 0

For each vertex \( u \in V[G] \)
- If color\([u]\) = WHITE then
  - DFS-Visit (\( u \))

**DFS-Visit (\( w \))**

- color\([w]\) = GRAY
- time = time + 1
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If color\([v]\) = WHITE then
    - \( \pi[v] = w \)
    - DFS-Visit(\( v \))
    - color\([w]\) = BLACK
    - \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
  If \( \text{color}[u] = \text{WHITE} \) then
  \[ \text{DFS-Visit} (u) \]

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
  If \( \text{color}[v] = \text{WHITE} \) then
  \[ \pi[v] = w \]
  \[ \text{DFS-Visit}(v) \]

\( \text{color}[w] = \text{BLACK} \)
\( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit \( (u) \)

**DFS-Visit \( (w) \)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit \( (v) \)
    - \( \text{color}[w] = \text{BLACK} \)
    - \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \)
- \( \text{DFS-Visit}(u) \)

**DFS-Visit (w)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \)
- \( \pi[v] = w \)
- \( \text{DFS-Visit}(v) \)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
DFS-Visit (u)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
- \( \pi[v] = w \)
- DFS-Visit(v)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$

For each vertex $u \in V[G]$
- If $\text{color}[u] = \text{WHITE}$ then
  - DFS-Visit ($u$)

---

**DFS-Visit ($w$)**

- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$

For each vertex $v \in \text{Adj}[w]$
- If $\text{color}[v] = \text{WHITE}$ then
  - $\pi[v] = w$
  - DFS-Visit($v$)
- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$

---

**Example Graph**

- $a$ connected to $e$, $b$, $c$
- $b$ connected to $e$, $d$
- $c$ connected to $g$
- $e$ connected to $d$, $b$
- $d$ connected to $c$
- $g$ connected to $c$

**Example Values**

- $a = 1/8$
- $b = 2/7$
- $c = 9$
- $e = 4/5$
- $b = 3/6$

**Time Values**

- $u = 9$
- $c = 4$
- $v = 0$

---

Abbas Malik: FCIT, King Abdulaziz University
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
- DFS-Visit \((u)\)

**DFS-Visit \((w)\)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
- \( \pi[v] = w \)
- DFS-Visit\((v)\)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit \((u)\)

**DFS-Visit \((w)\)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
- If \( \text{color}[v] = \text{WHITE} \) then
  - \( \pi[v] = w \)
  - DFS-Visit \((v)\)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - \( \text{DFS-Visit} (u) \)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
- If \( \text{color}[v] = \text{WHITE} \) then
  - \( \pi[v] = w \)
  - \( \text{DFS-Visit}(v) \)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - time = 0
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit (u)

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex $u \in V[G]$
  - $\text{color}[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
- time = 0
- For each vertex $u \in V[G]$
  - If $\text{color}[u] = \text{WHITE}$ then
    - DFS-Visit ($u$)

**DFS-Visit (w)**
- $\text{color}[w] = \text{GRAY}$
- time = time + 1
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$
  - If $\text{color}[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - DFS-Visit($v$)
- $f[w] = \text{time} = \text{time} + 1$

<table>
<thead>
<tr>
<th>$u$</th>
<th>$c$</th>
<th>$w$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>9</td>
<td>w</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$v$</th>
<th>$g$</th>
</tr>
</thead>
</table>

Abbas Malik: FCIT, King Abdulaziz University
### Depth First Search

**DFS (G)**

- For each vertex $u \in V[G]$
  - $\text{color}[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
  - $\text{time} = 0$
- For each vertex $u \in V[G]$
  - If $\text{color}[u] = \text{WHITE}$ then
    - $\text{DFS-Visit} (u)$

**DFS-Visit (w)**

- $\text{color}[w] = \text{GRAY}$
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$
  - If $\text{color}[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - $\text{DFS-Visit}(v)$
- $f[w] = \text{time} = \text{time} + 1$
Depth First Search

**DFS (G)**
For each vertex $u \in V[G]$
- $\text{color}[u] = \text{WHITE}$
- $\pi[u] = \text{NIL}$
- $\text{time} = 0$
For each vertex $u \in V[G]$
  If $\text{color}[u] = \text{WHITE}$ then
    DFS-Visit ($u$)

DFS-Visit ($w$)
- $\text{color}[w] = \text{GRAY}$
  - $\text{time} = \text{time} + 1$
  - $d[w] = \text{time}$
  - For each vertex $v \in \text{Adj}[w]$
    If $\text{color}[v] = \text{WHITE}$ then
      $\pi[v] = w$
      DFS-Visit($v$)
    - $\text{color}[w] = \text{BLACK}$
      $f[w] = \text{time} = \text{time} + 1$
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
 DFS-Visit (u)

DFS-Visit (w)
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
  
  For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(v)
  - \( \text{color}[w] = \text{BLACK} \)
  
  \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**
For each vertex $u \in V[G]$  
- $\text{color}[u] = \text{WHITE}$  
- $\pi[u] = \text{NIL}$  
- $\text{time} = 0$
For each vertex $u \in V[G]$  
If $\text{color}[u] = \text{WHITE}$ then  
DFS-Visit ($u$)

**DFS-Visit ($w$)**
- $\text{color}[w] = \text{GRAY}$  
- $\text{time} = \text{time} + 1$
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$  
  If $\text{color}[v] = \text{WHITE}$ then  
  - $\pi[v] = w$
  - DFS-Visit($v$)
  - $\text{color}[w] = \text{BLACK}$
  - $f[w] = \text{time} = \text{time} + 1$
**Depth First Search**

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
- DFS-Visit \((u)\)

**DFS-Visit \((w)\)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
- \( \pi[v] = w \)
- DFS-Visit\((v)\)

- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
  \( \text{color}[u] = \text{WHITE} \)
  \( \pi[u] = \text{NIL} \)
  \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
  If \( \text{color}[u] = \text{WHITE} \) then
    DFS-Visit (u)

def DFS-Visit (w)
  \( \text{color}[w] = \text{GRAY} \)
  \( \text{time} = \text{time} + 1 \)
  \( d[w] = \text{time} \)
  For each vertex \( v \in \text{Adj}[w] \)
    If \( \text{color}[v] = \text{WHITE} \) then
      \( \pi[v] = w \)
      DFS-Visit(v)
    \( \text{color}[w] = \text{BLACK} \)
    \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit (\( u \))

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit(\( v \))
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit \((u)\)

**DFS-Visit \((w)\)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( \nu \in \text{Adj}[w] \)
  - If \( \text{color}[
u] = \text{WHITE} \) then
    - \( \pi[
u] = w \)
    - DFS-Visit\((\nu)\)
  - \( \text{color}[w] = \text{BLACK} \)
  - \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
- If \( \text{color}[u] = \text{WHITE} \) then
  - DFS-Visit \( u \)

DFS-Visit \( (w) \)
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( \nu \in \text{Adj}[w] \)
  - If \( \text{color}[
u] = \text{WHITE} \) then
    - \( \pi[
u] = w \)
    - DFS-Visit(\( \nu \))
  - \( \text{color}[w] = \text{BLACK} \)
  - \( f[w] = \text{time} = \text{time} + 1 \)
**Depth First Search**

**DFS (G)**

- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
  - \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit \( (u) \)

**DFS-Visit \( (w) \)**

- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit \( (v) \)
  - \( \text{color}[w] = \text{BLACK} \)
  - \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

**DFS (G)**

For each vertex \( u \in V[G] \)
- \( \text{color}[u] = \text{WHITE} \)
- \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)

For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
- **DFS-Visit** (\( u \))

**DFS-Visit (w)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)

For each vertex \( v \in \text{Adj}[w] \)
If \( \text{color}[v] = \text{WHITE} \) then
- \( \pi[v] = w \)
- **DFS-Visit(v)**
- \( \text{color}[w] = \text{BLACK} \)
- \( f[w] = \text{time} = \text{time} + 1 \)
Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
\[
\text{color}[u] = \text{WHITE} \\
\pi[u] = \text{NIL} \\
\text{time} = 0
\]
For each vertex \( u \in V[G] \)
If \( \text{color}[u] = \text{WHITE} \) then
DFS-Visit (u)

DFS-Visit (w)
\[
\text{color}[w] = \text{GRAY} \\
\text{time} = \text{time} + 1 \\
\text{d}[w] = \text{time} \\
\text{For each vertex } v \in \text{Adj}[w] \\
\text{If } \text{color}[v] = \text{WHITE} \text{ then} \\
\pi[v] = w \\
\text{DFS-Visit}(v) \\
\text{color}[w] = \text{BLACK} \\
\text{f}[w] = \text{time} = \text{time} + 1
\]
Depth First Search

**DFS (G)**
- For each vertex $u \in V[G]$
  - $\text{color}[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
  - time = 0
- For each vertex $u \in V[G]$
  - If $\text{color}[u] = \text{WHITE}$ then
    - DFS-Visit ($u$)

**DFS-Visit ($w$)**
- $\text{color}[w] = \text{GRAY}$
- time = time + 1
- $d[w] = \text{time}$
- For each vertex $v \in \text{Adj}[w]$
  - If $\text{color}[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - DFS-Visit($v$)
- $\text{color}[w] = \text{BLACK}$
- $f[w] = \text{time} = \text{time} + 1$

---

Abbas Malik: FCIT, King Abdulaziz University
**Depth First Search**

**DFS (G)**
- For each vertex \( u \in V[G] \)
  - \( \text{color}[u] = \text{WHITE} \)
  - \( \pi[u] = \text{NIL} \)
- \( \text{time} = 0 \)
- For each vertex \( u \in V[G] \)
  - If \( \text{color}[u] = \text{WHITE} \) then
    - DFS-Visit \( (u) \)

**DFS-Visit \( (w) \)**
- \( \text{color}[w] = \text{GRAY} \)
- \( \text{time} = \text{time} + 1 \)
- \( d[w] = \text{time} \)
- For each vertex \( v \in \text{Adj}[w] \)
  - If \( \text{color}[v] = \text{WHITE} \) then
    - \( \pi[v] = w \)
    - DFS-Visit \( (v) \)
  - \( \text{color}[w] = \text{BLACK} \)
  - \( f[w] = \text{time} = \text{time} + 1 \)
### Depth First Search

**DFS (G)**
- For each vertex $u \in V[G]$
  - $color[u] = WHITE$
  - $\pi[u] = NIL$
  - $time = 0$
- For each vertex $u \in V[G]$
  - If $color[u] = WHITE$ then
    - DFS-Visit ($u$)

**DFS-Visit (w)**
- $color[w] = GRAY$
- $time = time + 1$
- $d[w] = time$
- For each vertex $v \in Adj[w]$
  - If $color[v] = WHITE$ then
    - $\pi[v] = w$
    - DFS-Visit($v$)
- $color[w] = BLACK$
- $f[w] = time = time + 1$
**Depth First Search**

**DFS (G)**
- For each vertex $u \in V[G]$
  - $color[u] = \text{WHITE}$
  - $\pi[u] = \text{NIL}$
  - $time = 0$
- For each vertex $u \in V[G]$
  - If $color[u] = \text{WHITE}$ then
    - DFS-Visit $(u)$

**DFS-Visit (w)**
- $color[w] = \text{GRAY}$
- $time = time + 1$
- $d[w] = time$
- For each vertex $v \in \text{Adj}[w]$
  - If $color[v] = \text{WHITE}$ then
    - $\pi[v] = w$
    - DFS-Visit $(v)$
- $color[w] = \text{BLACK}$
- $f[w] = time = time + 1$
Analysis – Depth First Search

DFS (G)
For each vertex \( u \in V[G] \)
  \( \text{color}[u] = \text{WHITE} \)
  \( \pi[u] = \text{NIL} \)
  \( \text{time} = 0 \)
For each vertex \( u \in V[G] \)
  If \( \text{color}[u] = \text{WHITE} \) then
    DFS-Visit (u)

DFS-Visit (w)
  \( \text{color}[w] = \text{GRAY} \)
  \( \text{time} = \text{time} + 1 \)
  \( d[w] = \text{time} \)
  For each vertex \( v \in \text{Adj}[w] \)
    If \( \text{color}[v] = \text{WHITE} \) then
      \( \pi[v] = w \)
      DFS-Visit(v)
  \( \text{color}[w] = \text{BLACK} \)
  \( f[w] = \text{time} = \text{time} + 1 \)

It searches all vertices \( V \) and all edges \( E \), thus its complexity is \( O(V + E) \)
ALGORITHM   $DFS(G)$

// Implements a depth-first search traversal of a given graph
// Input: Graph $G = (V, E)$
// Output: Graph $G$ with its vertices marked with consecutive integers
// in the order they've been first encountered by the DFS traversal
mark each vertex in $V$ with 0 as a mark of being “unvisited”

$\text{count} \leftarrow 0$

for each vertex $v$ in $V$ do
    if $v$ is marked with 0
        $dfs(v)$

$dfs(v)$
// visits recursively all the unvisited vertices connected to vertex $v$ by a path
// and numbers them in the order they are encountered
// via global variable count
$\text{count} \leftarrow \text{count} + 1$; mark $v$ with $\text{count}$

for each vertex $w$ in $V$ adjacent to $v$ do
    if $w$ is marked with 0
        $dfs(w)$
Notes on DFS

- DFS can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(V^2)$
  - adjacency lists: $\Theta(|V|+|E|)$

- Yields two distinct ordering of vertices:
  - order in which vertices are first encountered (pushed onto stack)
  - order in which vertices become dead-ends (popped off stack)

- Applications:
  - checking connectivity, finding connected components
  - checking acyclicity
  - finding articulation points and biconnected components
  - searching state-space of problems for solution (AI)
Dags and Topological Sorting

A *dag*: a directed acyclic graph, i.e. a directed graph with no (directed) cycles

![Diagram of a dag and not a dag]

Vertices of a dag can be linearly ordered so that for every edge its starting vertex is listed before its ending vertex (*topological sorting*). Being a dag is also a necessary condition for topological sorting be possible.

Arise in modeling many problems that involve prerequisite constraints (construction projects, document version control)
Topological Sorting Example

Order the following items in a food chain

- fish
- human
- shrimp
- sheep
- plankton
- wheat
- tiger

280 Abbas Malik: FCIT, King Abdulaziz University
**DFS-based Algorithm**

**DFS-based algorithm for topological sorting**

- Perform DFS traversal, noting the order vertices are popped off the traversal stack
- Reverse order solves topological sorting problem
- Back edges encountered? → NOT a dag!

Example:

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>f</td>
<td>g</td>
</tr>
<tr>
<td>e</td>
<td></td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>
```

Efficiency:

Abbas Malik: FCIT, King Abdulaziz University
Source Removal Algorithm

Repeatedly identify and remove a source (a vertex with no incoming edges) and all the edges incident to it until either no vertex is left (problem is solved) or there is no source among remaining vertices (not a dag).

Example:

Efficiency: same as efficiency of the DFS-based algorithm
Decrease-by-Constant-Factor Algorithms

In this variation of decrease-and-conquer, instance size is reduced by the same factor (typically, 2).

Examples:
• Binary search and the method of bisection
• Exponentiation by squaring
• Multiplication à la russe (Russian peasant method)
• Fake-coin puzzle
• Josephus problem
Exponentiation by Squaring

• The problem: Compute \( a^n \) where \( n \) is a nonnegative integer
• The problem can be solved by applying recursively the formulas:

  For even values of \( n \)
  \[
  a^n = (a^{n/2})^2 \quad \text{if } n > 0 \text{ and } a^0 = 1
  \]

  For odd values of \( n \)
  \[
  a^n = (a^{(n-1)/2})^2 a
  \]

Recurrence: \( M(n) = M(\lfloor n/2 \rfloor) + f(n) \), where \( f(n) = 1 \) or \( 2 \), \( M(0) = 0 \)

Master Theorem: \( M(n) \in \Theta(\log n) = \Theta(b) \) where \( b = \lceil \log_2(n+1) \rceil \)
Russian Peasant Multiplication

The problem: Compute the product of two positive integers

Can be solved by a decrease-by-half algorithm based on the following formulas.

For even values of \( n \):

\[
 n \times m = \frac{n}{2} \times 2m
\]

For odd values of \( n \):

\[
 n \times m = \frac{n-1}{2} \times 2m + m \quad \text{if} \quad n > 1 \quad \text{and} \quad m \quad \text{if} \quad n = 1
\]
Example of Russian Peasant Multiplication

Compute $20 \times 26$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>104</td>
</tr>
<tr>
<td>2</td>
<td>208</td>
</tr>
<tr>
<td>1</td>
<td>416</td>
</tr>
</tbody>
</table>

$\text{Note: Method reduces to adding } m\text{'s values corresponding to odd } n\text{'s.}$
Fake-Coin Puzzle (simpler version)

There are $n$ identically looking coins one of which is fake. There is a balance scale but there are no weights; the scale can tell whether two sets of coins weigh the same and, if not, which of the two sets is heavier (but not by how much). Design an efficient algorithm for detecting the fake coin. Assume that the fake coin is known to be lighter than the genuine ones.

Decrease by factor 2 algorithm

Decrease by factor 3 algorithm
Variable-Size-Decrease Algorithms

In the variable-size-decrease variation of decrease-and-conquer, instance size reduction varies from one iteration to another.

Examples:

- Euclid’s algorithm for greatest common divisor
- Partition-based algorithm for selection problem
- Interpolation search
- Some algorithms on binary search trees
- Nim and Nim-like games
Euclid’s Algorithm

Euclid’s algorithm is based on repeated application of equality

\[ \text{gcd}(m, n) = \text{gcd}(n, m \mod n) \]

Ex.: \( \text{gcd}(80, 44) = \text{gcd}(44, 36) = \text{gcd}(36, 12) = \text{gcd}(12, 0) = 12 \)

One can prove that the size, measured by the second number, decreases at least by half after two consecutive iterations.

Hence, \( T(n) \in O(\log n) \)
Selection Problem

Find the $k$-th smallest element in a list of $n$ numbers

- $k = 1$ or $k = n$

- **median**: $k = \lceil n/2 \rceil$

  Example: $4, 1, 10, 9, 7, 12, 8, 2, 15$ median = ?

The median is used in statistics as a measure of an average value of a sample. In fact, it is a better (more robust) indicator than the mean, which is used for the same purpose.
Digression: Post Office Location Problem

Given $n$ village locations along a straight highway, where should a new post office be located to minimize the average distance from the villages to the post office?
Algorithms for the Selection Problem

The sorting-based algorithm: Sort and return the $k$-th element
Efficiency (if sorted by mergesort): $\Theta(n \log n)$

A faster algorithm is based on using the quicksort-like partition of the list. Let $s$ be a split position obtained by a partition:

\[
\begin{array}{c|c}
\text{all are } \leq A[s] & \text{all are } \geq A[s] \\
\hline
s & \\
\end{array}
\]

Assuming that the list is indexed from 1 to $n$:
If $s = k$, the problem is solved;
if $s > k$, look for the $k$-th smallest elem. in the left part;
if $s < k$, look for the $(k-s)$-th smallest elem. in the right part.

Note: The algorithm can simply continue until $s = k$. 
**Tracing the Median / Selection Algorithm**

Example: 4 1 10 9 7 12 8 2 15  
Here: $n = 9$, $k = \lceil 9/2 \rceil = 5$

<table>
<thead>
<tr>
<th>Array Index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>4</td>
<td>1</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Sorted</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>s &lt; k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Sorted</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>s &gt; k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution: median is 8
Efficiency of the Partition-based Algorithm

Average case (average split in the middle):

\[ C(n) = C(n/2) + (n+1) \quad C(n) \in \Theta(n) \]

Worst case (degenerate split):

\[ C(n) \in \Theta(n^2) \]

A more sophisticated choice of the pivot leads to a complicated algorithm with \( \Theta(n) \) worst-case efficiency.
Interpolation Search

Searches a sorted array similar to binary search but estimates location of the search key in $A[l..r]$ by using its value $v$. Specifically, the values of the array’s elements are assumed to grow linearly from $A[l]$ to $A[r]$ and the location of $v$ is estimated as the $x$-coordinate of the point on the straight line through $(l, A[l])$ and $(r, A[r])$ whose $y$-coordinate is $v$:

$$x = l + \left\lfloor \frac{(v - A[l])(r - l)}{(A[r] - A[l])} \right\rfloor$$
Analysis of Interpolation Search

- Efficiency
  
  average case: $C(n) < \log_2 \log_2 n + 1$

  worst case: $C(n) = n$

- Preferable to binary search only for VERY large arrays and/or expensive comparisons

- Has a counterpart, the method of false position (regula falsi), for solving equations in one unknown (Sec. 12.4)
Binary Search Tree Algorithms

Several algorithms on BST requires recursive processing of just one of its subtrees, e.g.,

- Searching
- Insertion of a new key
- Finding the smallest (or the largest) key
Searching in Binary Search Tree

Algorithm $BTS(x, v)$
//Searches for node with key equal to $v$ in BST rooted at node $x$
    if $x = \text{NIL}$ return -1
    else if $v = K(x)$ return $x$
    else if $v < K(x)$ return $BTS(left(x), v)$
    else return $BTS(right(x), v)$

Efficiency

    worst case: $C(n) = n$
    average case: $C(n) \approx 2\ln n \approx 1.39\log_2 n$
One-Pile Nim

There is a pile of $n$ chips. Two players take turn by removing from the pile at least 1 and at most $m$ chips. (The number of chips taken can vary from move to move.) The winner is the player that takes the last chip. Who wins the game – the player moving first or second, if both player make the best moves possible?

It’s a good idea to analyze this and similar games “backwards”, i.e., starting with $n = 0, 1, 2, \ldots$
Partial Graph of One-Pile Nim with $m = 4$

Vertex numbers indicate $n$, the number of chips in the pile. The losing position for the player to move are circled. Only winning moves from a winning position are shown (in bold).

**Generalization:** The player moving first wins iff $n$ is not a multiple of 5 (more generally, $m+1$); the winning move is to take $n \mod 5$ ($n \mod (m+1)$) chips on every move.