**IDSS Project Documentation**

**Breadth-First Strategy & Depth-First Strategy**

**Supervised by:**

**DR. Abeer Amer**

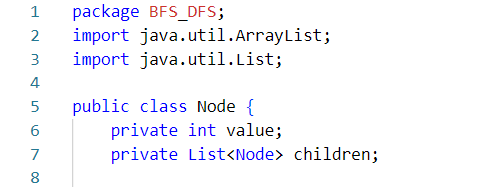
**MS. Nourhan Nasr**

**Team members:**

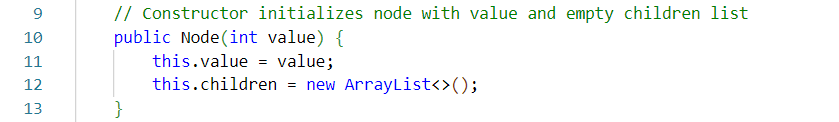
|  |  |  |
| --- | --- | --- |
| Name | Sec. | Seat n. |
| Saad Abd El-Samei Mohamed | **G4** | **64221** |
| Mahmoud Mohsen Ali El-Saftawy | **G5** | **64202** |
| Mohamed Gamal Mohamed | **G5** | **64227** |
| Mohamed Abd El-Raouf El-Fawi | **G3** | **64283** |
| Yousef Ahmed El-sayed | **G5** | **74508** |
| Ro'ya Ahmed Mohamed anwar | **G4** | **64220** |
| Salma Ebrahim El-sayed Ebrahim | **G4** | **64296** |
| Christina Sobhy Abdo Rezek allah George | **G5** | **64260** |
| Esraa Ali Ahmed sleem | **G3** | **64300** |
| Yara Khaled Ahmed Mohamed Okasha | **G5** | **64265** |

**Breadth-First Strategy & Depth-First Strategy**

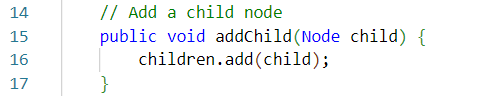
1. **Node Class:**

****

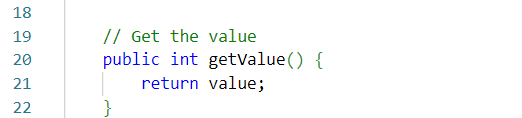
* private int value: Stores the integer value of the node.
* private List<Node> children: Stores the list of child nodes.



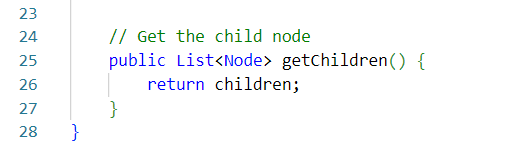
* Initializes a node with a given value and an empty list of children.



* Adds a child node to the current node.

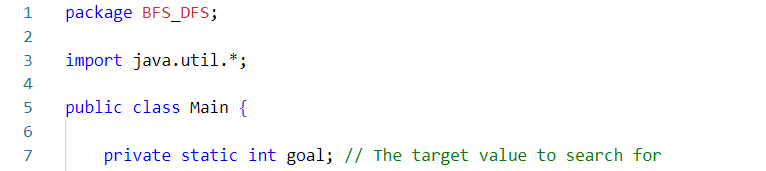


* Returns the value of the node.

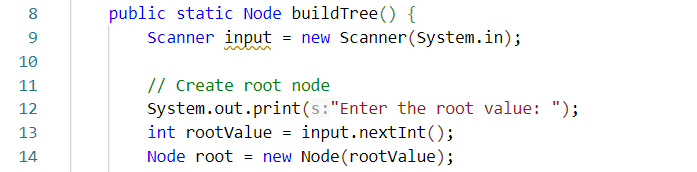


* Returns the list of child nodes.

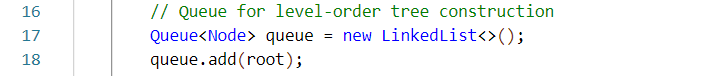
1. **Main Class:**

****

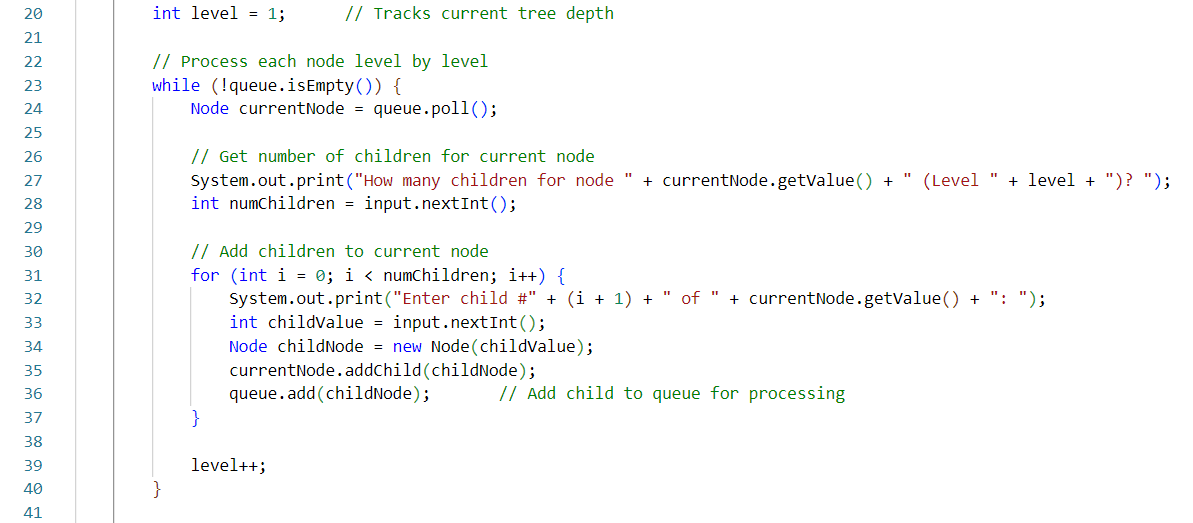
* Imports necessary Java utilities (Scanner, Queue, Stack, etc.)
* private static int goal   
  - Class-level variable storing search target

****

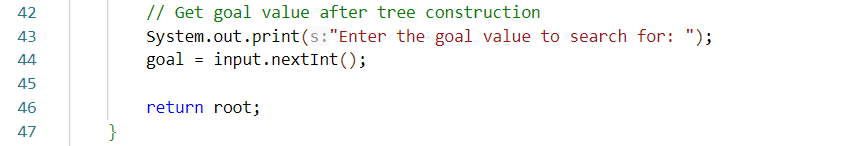
* Creates root node from user input



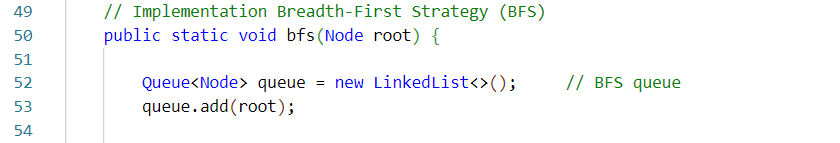
* Uses queue for BFS-style tree building



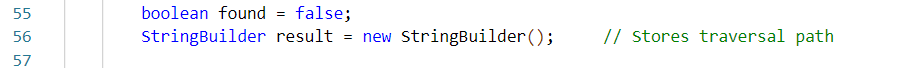
* Processes nodes level by level
* Collects children count and values through user input

****

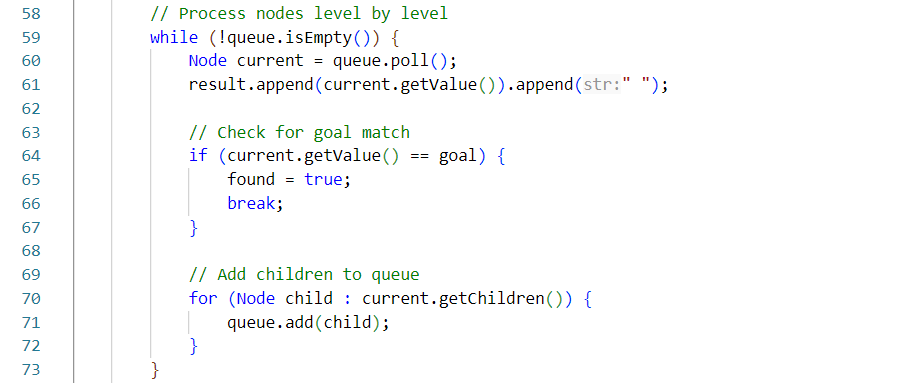
* Stores search target after tree construction

****

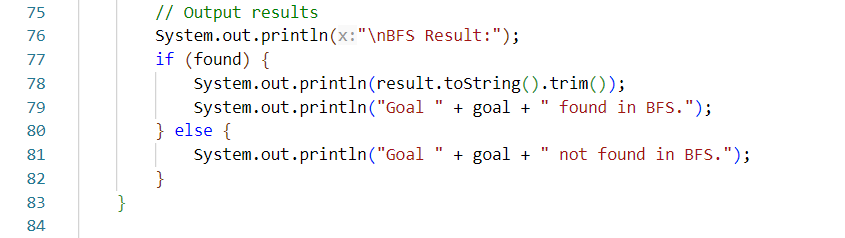
* Initializes a FIFO (First-In-First-Out) queue to process nodes level by level.
* Starts traversal by adding the root node.

****

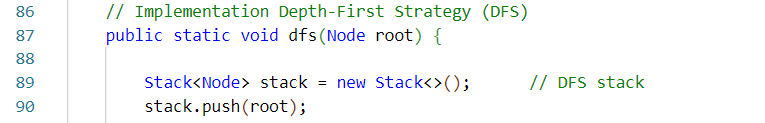
* found: Tracks whether the goal value has been found.
* result: Stores the sequence of visited node values.



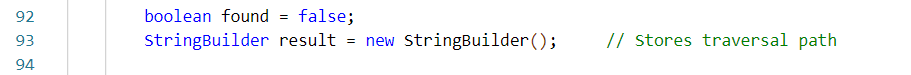
* **Steps**:
  1. **Dequeue Node**: Removes the front node from the queue ( poll( ) ).
  2. **Record Value**: Appends the node’s value to result.
  3. **Goal Check**:
     + If the node’s value matches the goal, sets found = true and exits the loop.
  4. **Enqueue Children**: Adds all children of the current node to the queue.



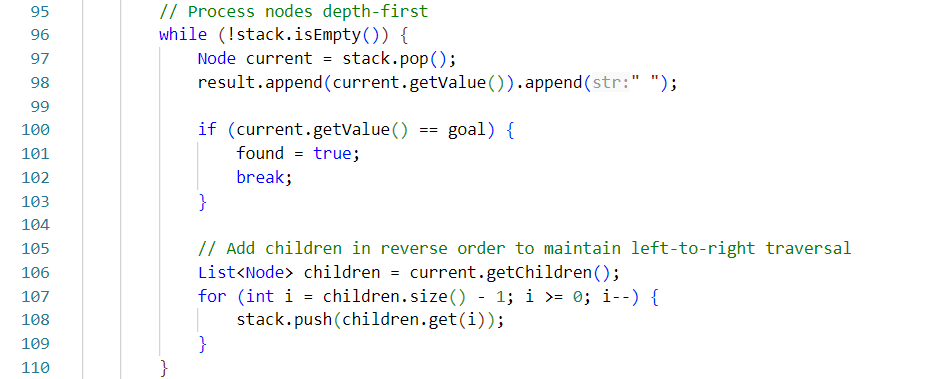
* **Output Logic**:
  + **Found**: Prints the traversal path (trimmed to remove trailing spaces) and a success message.
  + **Not Found**: Prints a failure message.



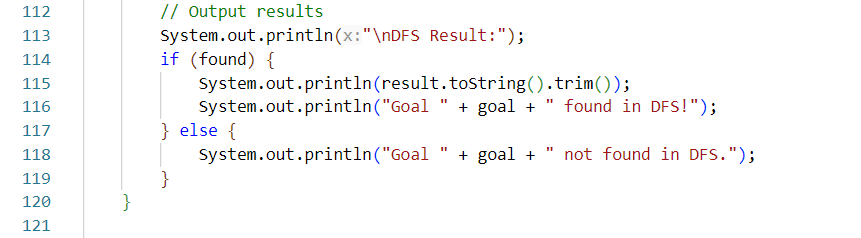
* Initializes a LIFO (Last-In-First-Out) stack to process nodes depth-first.
* Starts traversal by adding the root node.



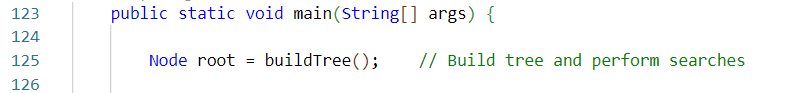
* found: Tracks whether the goal value has been found.
* result: Stores the sequence of visited node values.



* **Steps**:
  1. **Pop Node**: Removes the top node from the stack (pop()).
  2. **Record Value**: Appends the node’s value to result.
  3. **Goal Check**:
     + If the node’s value matches the goal, sets found = true and exits the loop.
  4. **Push Children in Reverse Order**:
     + Iterates over children from **right to left** (reverse order).
     + Pushes them onto the stack to ensure **left-to-right traversal** when popped.
     + *Why?*: Stacks are LIFO. By pushing the rightmost child first, it will be popped last, preserving the left-to-right order.

****

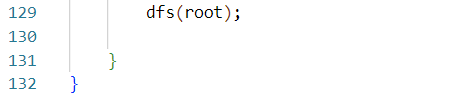
* **Found**: Prints the traversal path (trimmed to remove trailing spaces) and a success message.
* **Not Found**: Prints a failure message.



* Calls the buildTree() method to dynamically construct the tree based on **user input**.
* buildTree():
  + Prompts the user to enter values for the root and child nodes.
  + Returns the root node of the constructed tree.

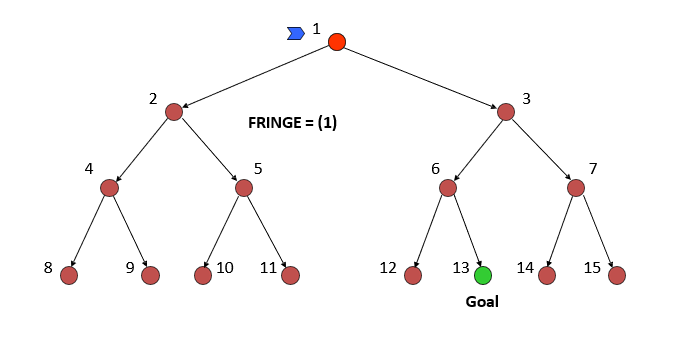
****

* Executes the BFS traversal algorithm starting from the root node.
* **Key Actions**:
  1. Visits nodes level by level.
  2. Checks each node for a match with the predefined goal value.
  3. Prints the traversal path and search result.

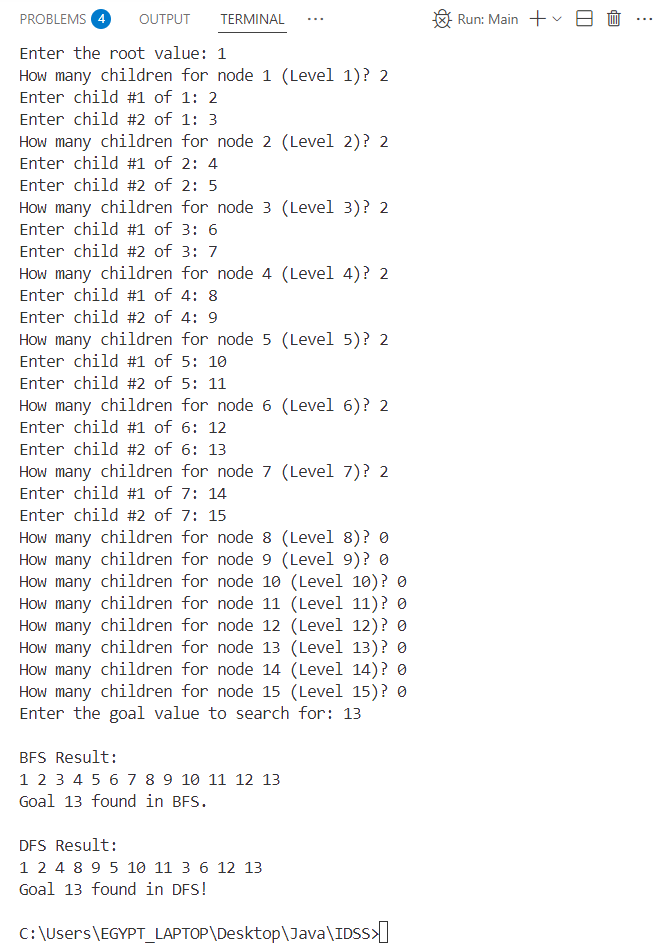


* Executes the DFS traversal algorithm starting from the root node.
* **Key Actions**:
  1. Visits nodes by exploring branches deeply before backtracking.
  2. Checks each node for a match with the predefined goal value.
  3. Prints the traversal path and search result.

**Example 1**:



**Run The Code:**

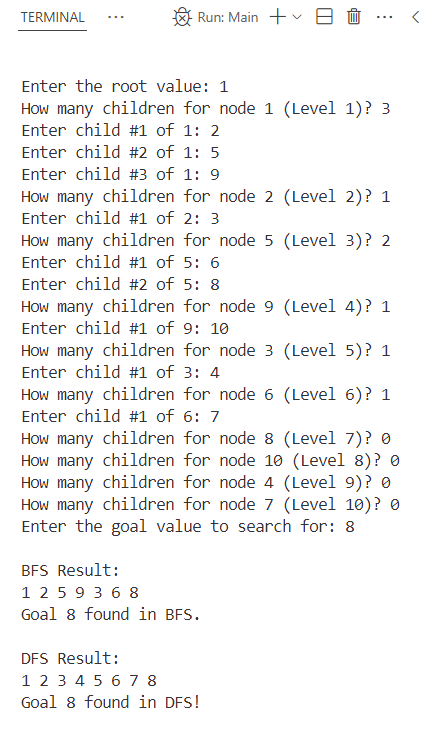
****

**Example 2:**

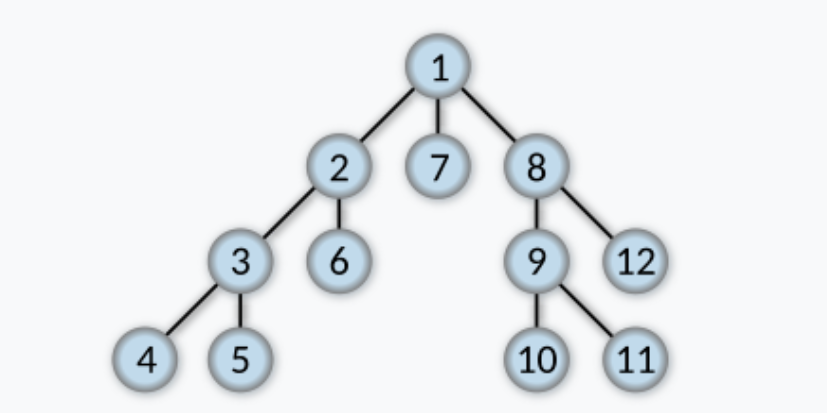
**A screenshot of a computer

Description generated with very high confidence**

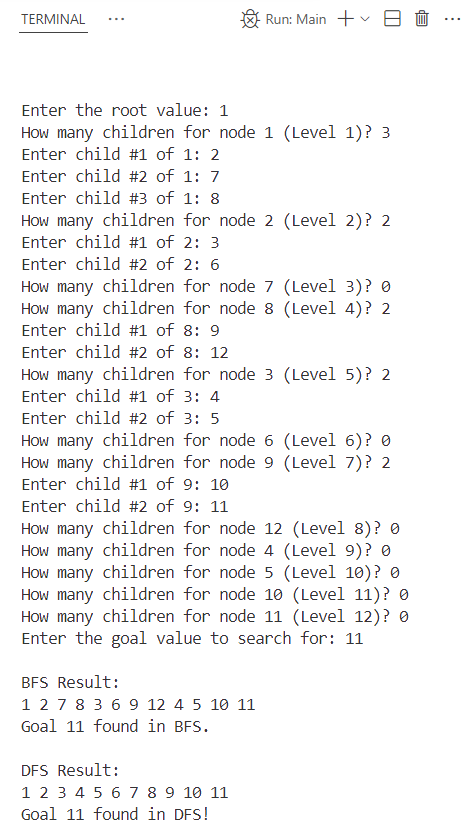
**Run The Code:**

****

**Example 3:**

****

**Run The Code:**

****