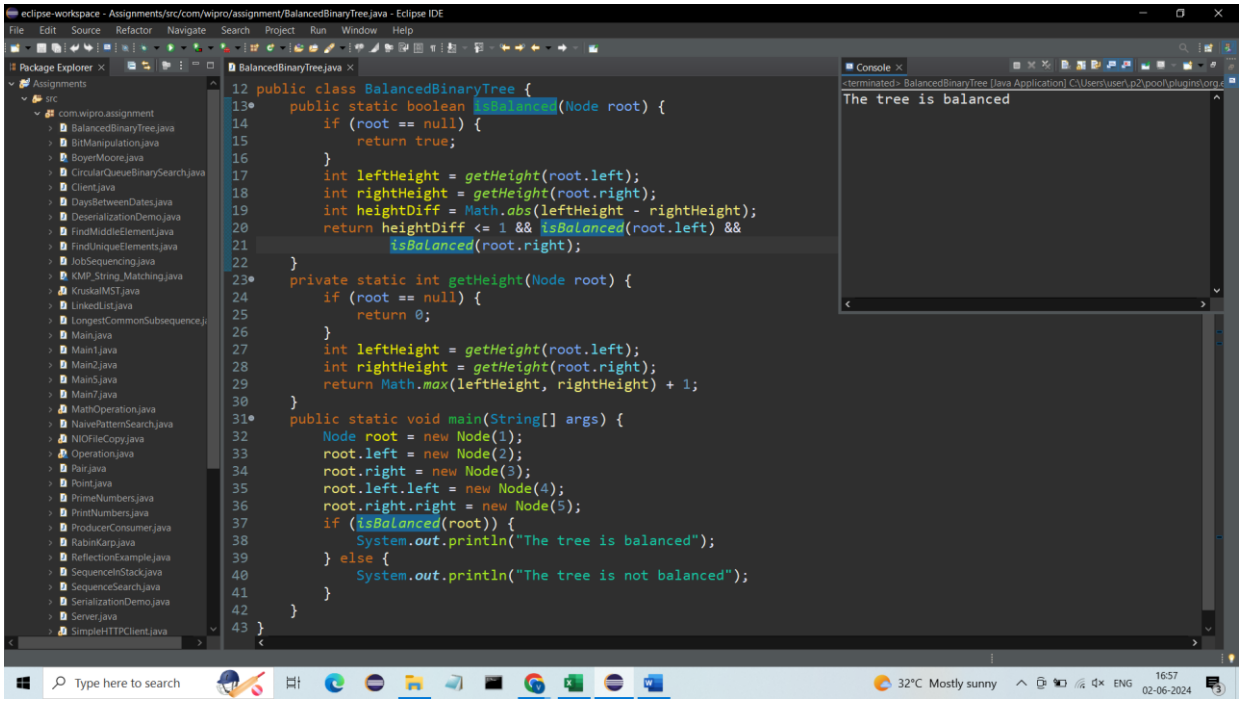
Task 1: Balanced Binary Tree Check

Write a function to check if a given binary tree is balanced. A balanced tree is one where the height of two subtrees of any node never differs by more than one.

Function: -

```java
public static boolean isBalanced(Node root) {
    if (root == null) {
        return true;
    }

    int leftHeight = getHeight(root.left);
    int rightHeight = getHeight(root.right);

    int heightDiff = Math.abs(leftHeight - rightHeight);

    return heightDiff <= 1 && isBalanced(root.left) && isBalanced(root.right);
}

private static int getHeight(Node root) {
    if (root == null) {
        return 0;
    }

    int leftHeight = getHeight(root.left);
    int rightHeight = getHeight(root.right);

    return Math.max(leftHeight, rightHeight) + 1;
}
```

Output: -



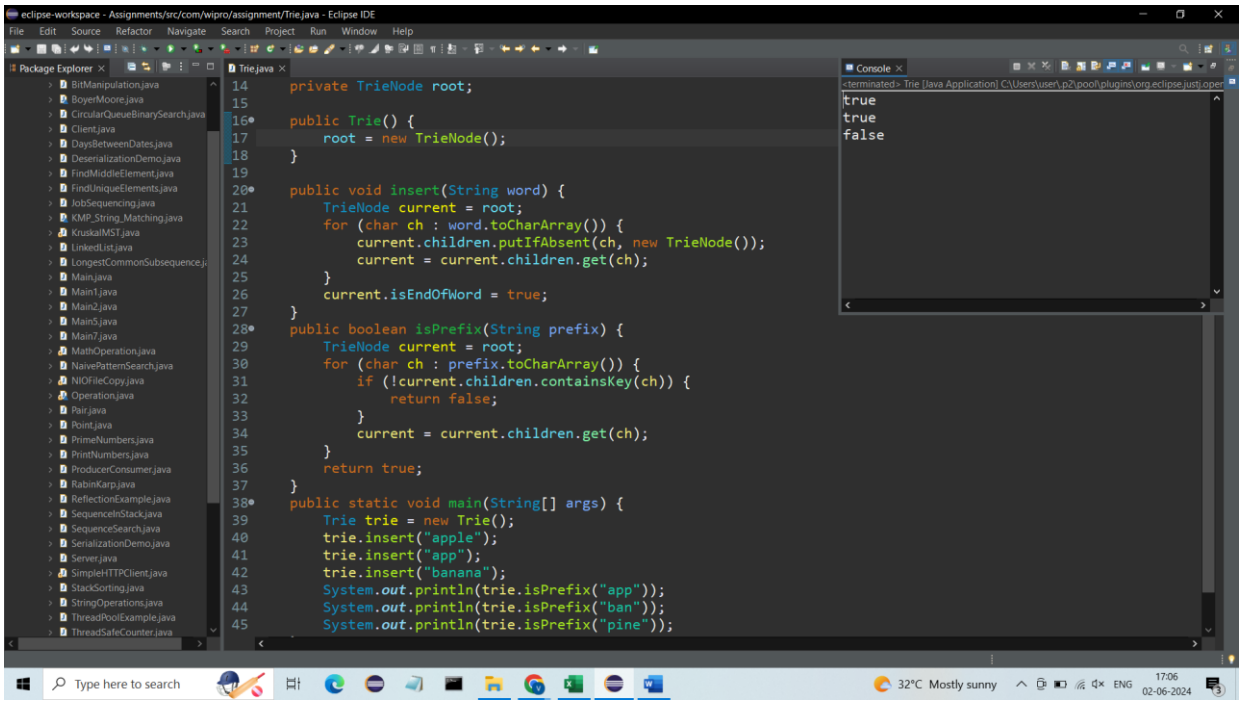
Task 2: Trie for Prefix Checking

Implement a trie data structure in java that supports insertion of strings and provides a method to check if a given string is a prefix of any word in the trie.

Function: -

```java
public void insert(String word) {
    TrieNode current = root;
    for (char ch : word.toCharArray()) {
        current.children.putIfAbsent(ch, new TrieNode());
        current = current.children.get(ch);
    }
    current.isEndOfWord = true;
}
public boolean isPrefix(String prefix) {
    TrieNode current = root;
    for (char ch : prefix.toCharArray()) {
        if (!current.children.containsKey(ch)) {
            return false;
        }
        current = current.children.get(ch);
    }
    return true;
}
```

Output: -



Task 3: Implementing Heap Operations

Code a min-heap in java with methods for insertion, deletion, and fetching the minimum element. Ensure that the heap property is maintained after each operation.

Function for Insertion: -

```java
public void insert(int value) {
    if (size >= capacity) {
        throw new IllegalStateException("Heap is full");
    }
    heap[size] = value;
    size++;
    heapifyUp();
}
```

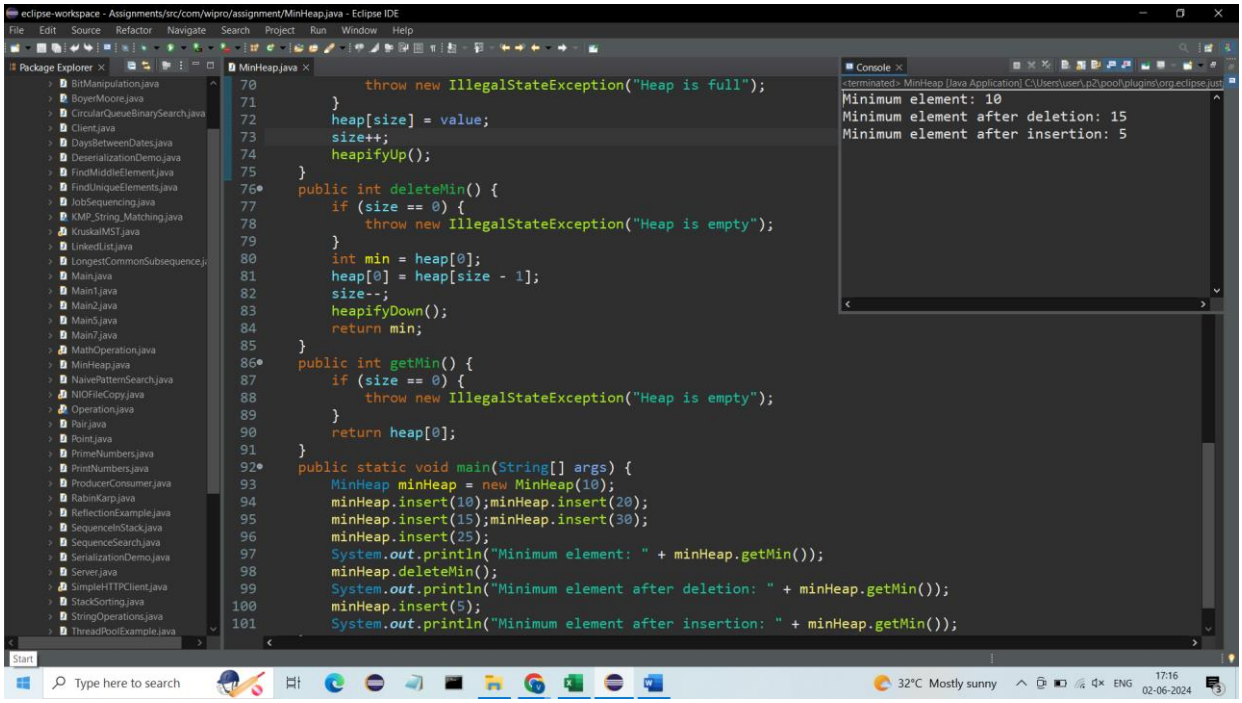
Function for Deletion: -

```java
  public int deleteMin() {
      if (size == 0) {
          throw new IllegalStateException("Heap is empty");
      }
      int min = heap[0];
      heap[0] = heap[size - 1];
      size--;
      heapifyDown();
      return min;
  }
```

Function for Minimum: -

```java
 public int getMin() {
     if (size == 0) {
         throw new IllegalStateException("Heap is empty");
     }
     return heap[0];
 }
```

Output: -



Task 4: Graph Edge Addition Validation

Given a directed graph, write a function that adds an edge between two nodes and then checks if the graph still has no cycles. If a cycle is created, the edge should not be added.

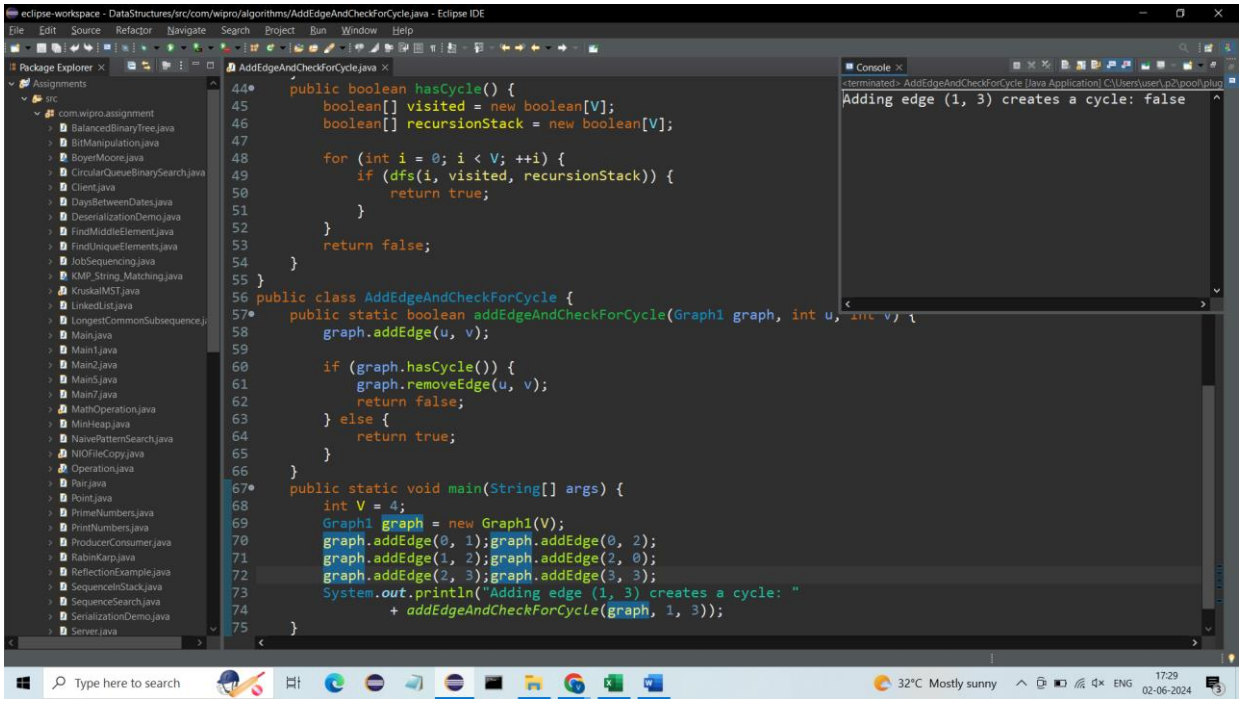
Function for checking graph has cycles or not: -

```
public boolean hasCycle() {
    boolean[] visited = new boolean[V];
    boolean[] recursionStack = new boolean[V];

    for (int i = 0; i < V; ++i) {
        if (dfs(i, visited, recursionStack)) {
            return true;
        }
    }
    return false;
}
```

Function for adding cycles to the graph: -

```java
public class AddEdgeAndCheckForCycle {
    public static boolean addEdgeAndCheckForCycle(Graph1 graph, int u, int v) {
        graph.addEdge(u, v);

        if (graph.hasCycle()) {
            graph.removeEdge(u, v);
            return false;
        } else {
            return true;
        }
    }
```

Output: -



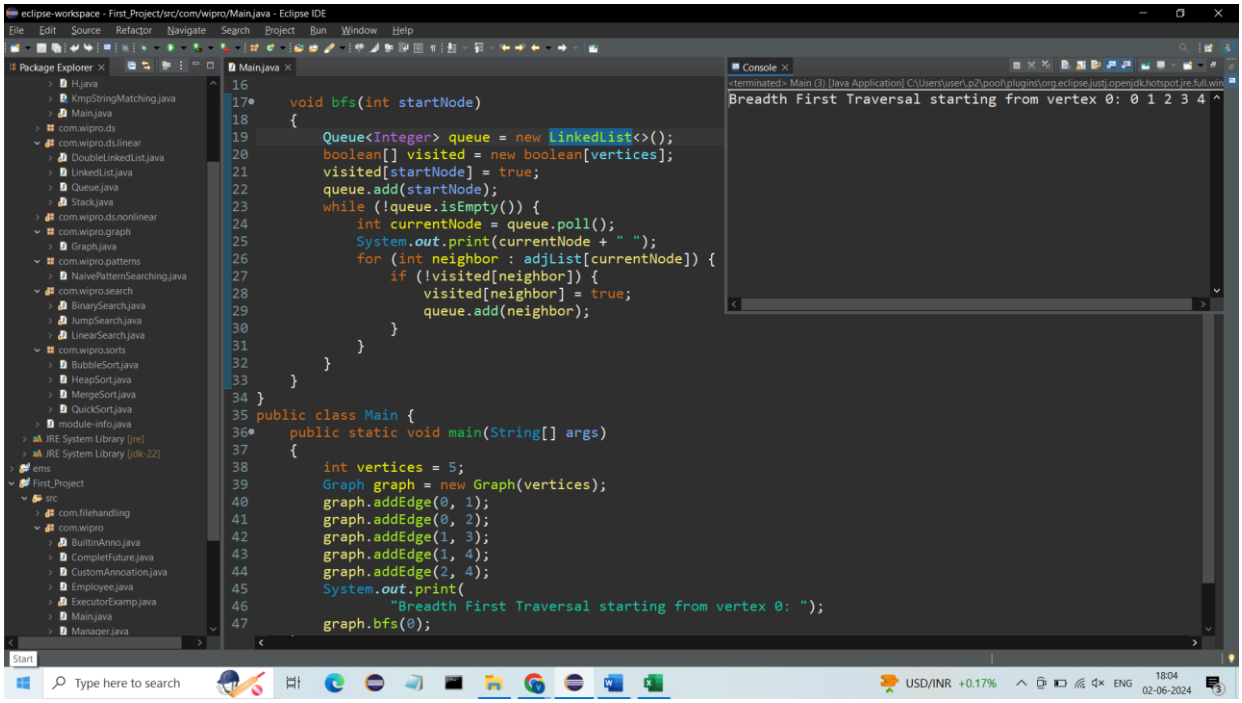
Task 5: Breadth-First Search (BFS) Implementation

For a given undirected graph, implement BFS to traverse the graph starting from a given node and print each node in the order it is visited.

Function to perform Breadth First Search on a graph: -

```
    void addEdge(int u, int v) { adjList[u].add(v); }

    void bfs(int startNode)
    {
        Queue<Integer> queue = new LinkedList<>();
        boolean[] visited = new boolean[vertices];
        visited[startNode] = true;
        queue.add(startNode);
        while (!queue.isEmpty()) {
            int currentNode = queue.poll();
            System.out.print(currentNode + " ");
            for (int neighbor : adjList[currentNode]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    queue.add(neighbor);
                }
            }
        }
    }
}
```

Output: -



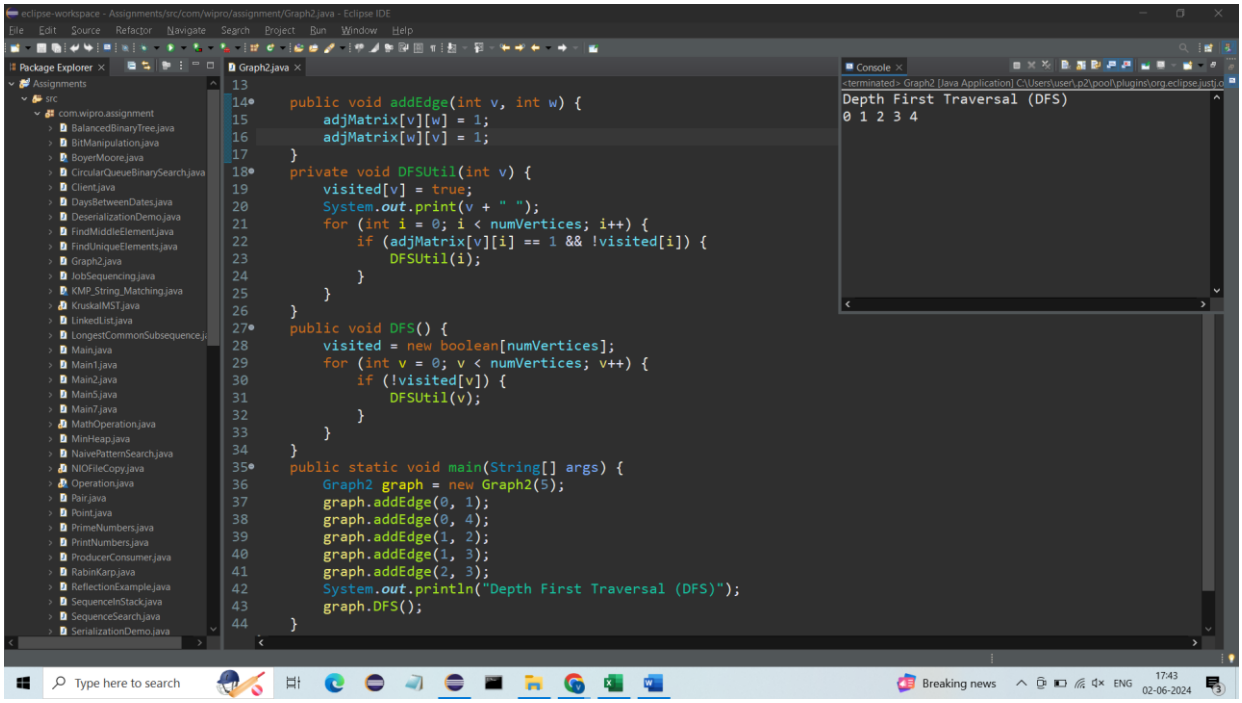
Task 6: Depth-First Search (DFS)

Recursive Write a recursive DFS function for a given undirected graph. The function should visit every node and print it out.

Function: -

```
    private void DFSUtil(int v) {
        visited[v] = true;
        System.out.print(v + " ");
        for (int i = 0; i < numVertices; i++) {
            if (adjMatrix[v][i] == 1 && !visited[i]) {
                DFSUtil(i);
            }
        }
    }
    public void DFS() {
        visited = new boolean[numVertices];
        for (int v = 0; v < numVertices; v++) {
            if (!visited[v]) {
                DFSUtil(v);
            }
        }
    }
```

Output: -


eclipse-workspace - Assignments/src/com/wipro/assignment/Graph2.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Assignments
src
com.wipro.assignment
BalancedBinaryTree.java
BitManipulation.java
BoyerMoore.java
CircularQueueBinarySearch.java
Client.java
DaysBetweenDates.java
DeserializationDemo.java
FindMiddleElement.java
FindUniqueElements.java
Graph2.java
JobSequencing.java
KMP_String_Matching.java
KruskalMST.java
LinkedList.java
LongestCommonSubsequence.j
Main.java
Main1.java
Main2.java
Main5.java
Main7.java
MathOperation.java
MinHeap.java
NaivePatternSearch.java
NIOFileCopy.java
Operation.java
Pair.java
Point.java
PrimeNumbers.java
PrintNumbers.java
ProducerConsumer.java
RabinKarp.java
ReflectionExample.java
SequenceInStack.java
SequenceSearch.java
SerializationDemo.java
Graph2.java
13
14 public void addEdge(int v, int w) {
15 adjMatrix[v][w] = 1;
16 adjMatrix[w][v] = 1;
17 }
18 private void DFSUtil(int v) {
19 visited[v] = true;
20 System.out.print(v + " ");
21 for (int i = 0; i < numVertices; i++) {
22 if (adjMatrix[v][i] == 1 && !visited[i]) {
23 DFSUtil(i);
24 }
25 }
26 }
27 public void DFS() {
28 visited = new boolean[numVertices];
29 for (int v = 0; v < numVertices; v++) {
30 if (!visited[v]) {
31 DFSUtil(v);
32 }
33 }
34 }
35 public static void main(String[] args) {
36 Graph2 graph = new Graph2(5);
37 graph.addEdge(0, 1);
38 graph.addEdge(0, 4);
39 graph.addEdge(1, 2);
40 graph.addEdge(1, 3);
41 graph.addEdge(2, 3);
42 System.out.println("Depth First Traversal (DFS)");
43 graph.DFS();
44 }
Console
<terminated> Graph2 [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.o
Depth First Traversal (DFS)
0 1 2 3 4
Type here to search
Breaking news
ENG
17:43
02-06-2024