Manjula Nannuri

Day-7&8_assignment

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

```
package sortings;

import java.util.Scanner;

public class BalancedBinaryTree {

int val;

BalancedBinaryTree left;

BalancedBinaryTree right;

BalancedBinaryTree(int val) {

this.val = val;

this.left = null;

this.right = null;

}

private static boolean isBalanced(BalancedBinaryTree root) {

if (root == null) {

return true;

}

int leftHeight = getHeight(root.left);

int rightHeight = getHeight(root.right);

if (Math.abs(leftHeight - rightHeight) > 1) {

return false;

}

return isBalanced(root.left) && isBalanced(root.right);

}

private static int getHeight(BalancedBinaryTree node) {
```

```
if (node == null) {

return 0;

}

int leftHeight = getHeight(node.left);

int rightHeight = getHeight(node.right);

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the structure of the binary tree (Enter -1 for
null nodes):");

BalancedBinaryTree root = buildTree(scanner);

System.out.println("Input binary tree:");

printTree(root);

boolean balanced = isBalanced(root);

if (balanced) {

System.out.println("The binary tree is balanced.");

} else {

System.out.println("The binary tree is not balanced.");

}

scanner.close();

}

private static BalancedBinaryTree buildTree(Scanner scanner) {

int val = scanner.nextInt();

if (val == -1) {

return null;//1 2 -1 -1 3 -1 -1

}

BalancedBinaryTree node = new BalancedBinaryTree(val);
```

```
node.left = buildTree(scanner);

node.right = buildTree(scanner);

return node;

}

public static void printTree(BalancedBinaryTree root) {

if (root == null) {

System.out.print("-1 ");

return;

}

System.out.print(root.val + " ");

printTree(root.left);

printTree(root.right);

}

}
```

OUTPUT:

```
Enter the structure of the binary tree (Enter -1 for null nodes):

1

2

-1

-1

3

-1

-1

Input binary tree:

1 2 -1 -1 3 -1 -1 The binary tree is balanced.
```

Task 2: Trie for Prefix Checking

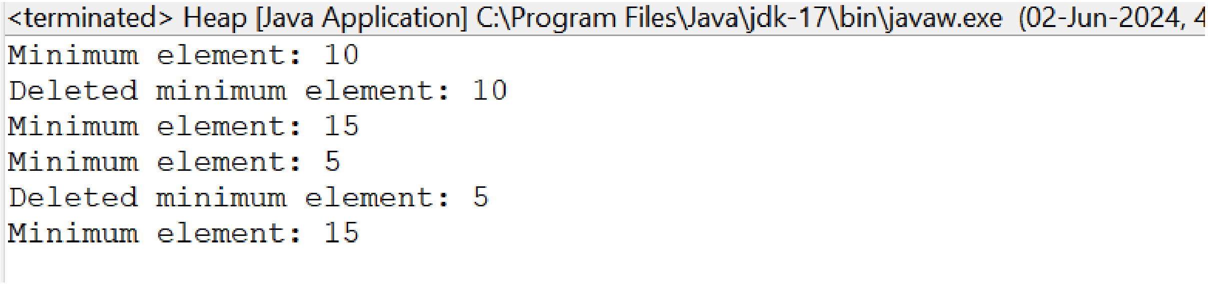
Implement a trie data structure in C# that supports insertion of strings and provides a method to check if a given string is a prefix of any word in the trie.

```
package com.sorting;
import java.util.ArrayList;
import java.util.List;
public class Heap {
private List<Integer> heap;
public Heap() {
heap = new ArrayList<>();
}
public void insert(int val) {
heap.add(val);
heapifyUp(heap.size() - 1);
}
public int deleteMin() {
if (heap.isEmpty()) {
throw new IllegalStateException("Heap is empty");
}
int minVal = heap.get(0);
int lastVal = heap.remove(heap.size() - 1);
if (!heap.isEmpty()) {
heap.set(0, lastVal);
heapifyDown(0);
}
```

```
return minVal;

}

public int getMin() {

if (heap.isEmpty()) {

throw new IllegalStateException("Heap is empty");

}

return heap.get(0);

}

private void heapifyUp(int index) {

while (index > 0) {

int parentIndex = (index - 1) / 2;

if (heap.get(index) >= heap.get(parentIndex)) {

break;

}

swap(index, parentIndex);

index = parentIndex;

}

}

private void heapifyDown(int index) {

int size = heap.size();

while (index < size) {

int leftChildIndex = 2 * index + 1;

int rightChildIndex = 2 * index + 2;

int smallest = index;

if (leftChildIndex < size && heap.get(leftChildIndex) < heap.get(smallest))
{

smallest = leftChildIndex;

}
if (rightChildIndex < size && heap.get(rightChildIndex) <
heap.get(smallest)) {

smallest = rightChildIndex;

}

if (smallest == index) {

break;

}

swap(index, smallest);

index = smallest;

}

}

private void swap(int index1, int index2) {

int temp = heap.get(index1);

heap.set(index1, heap.get(index2));

heap.set(index2, temp);

}

public static void main(String[] args) {

Heap minHeap = new Heap();

minHeap.insert(10);

minHeap.insert(15);

minHeap.insert(20);

minHeap.insert(17);

minHeap.insert(25);

System.out.println("Minimum element: " + minHeap.getMin()); // Should print
10

System.out.println("Deleted minimum element: " + minHeap.deleteMin()); //
Should print 10

System.out.println("Minimum element: " + minHeap.getMin()); // Should print
15

minHeap.insert(5);
```

```
System.out.println("Minimum element: " + minHeap.getMin()); // Should print
5

System.out.println("Deleted minimum element: " + minHeap.deleteMin()); //
Should print 5

System.out.println("Minimum element: " + minHeap.getMin()); // Should print
15

}

}
```

OUTPUT:



```
<terminated> Heap [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (02-Jun-2024, 4
Minimum element: 10
Deleted minimum element: 10
Minimum element: 15
Minimum element: 5
Deleted minimum element: 5
Minimum element: 15
```

Task 4: Graph Edge Addition Validation

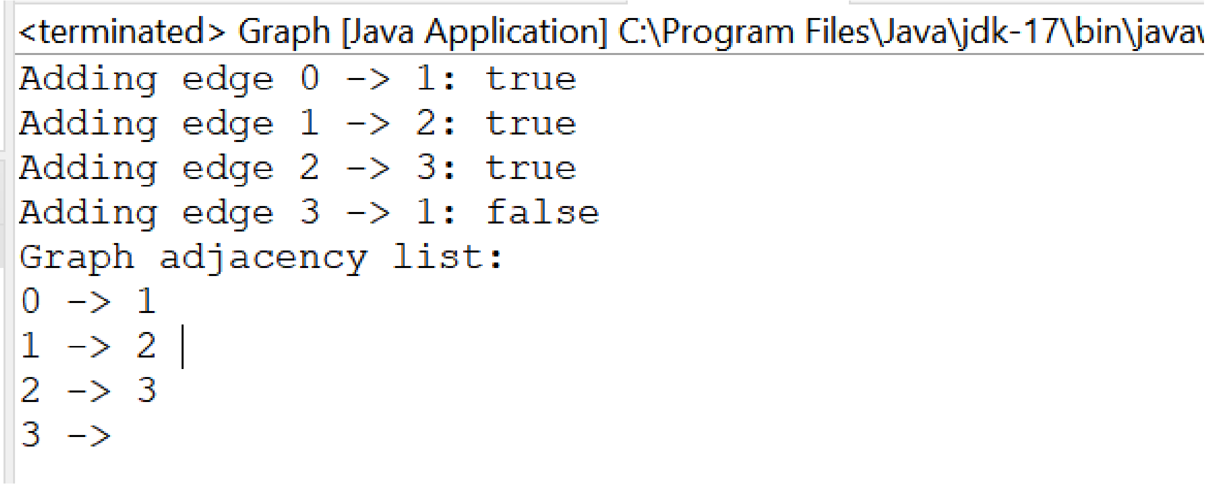
Given a directed graph, write a function that adds an edge between two nodes and then checks if the graph still has no cycles. If a cycle is created, the edge should not be added

```
package com.sorting;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class Graph {

private Map<Integer, List<Integer>> adjacencyList;

public Graph() {

this.adjacencyList = new HashMap<>();

}

public void addNode(int node) {

adjacencyList.putIfAbsent(node, new ArrayList<>());
```

```
}
public boolean addEdge(int from, int to) {
adjacencyList.get(from).add(to);
if (hasCycle()) {
adjacencyList.get(from).remove((Integer) to);
return false;
}
return true;
}
private boolean hasCycle() {
Map<Integer, Boolean> visited = new HashMap<>();
Map<Integer, Boolean> recursionStack = new HashMap<>();
for (int node : adjacencyList.keySet()) {
if (isCyclic(node, visited, recursionStack)) {
return true;
}
}
return false;
}
private boolean isCyclic(int node, Map<Integer, Boolean> visited,
Map<Integer, Boolean> recursionStack) {
if (recursionStack.getOrDefault(node, false)) {
return true;
}
if (visited.getOrDefault(node, false)) {
return false;
}
visited.put(node, true);
```

```java
recursionStack.put(node, true);
for (int neighbor : adjacencyList.getOrDefault(node, new ArrayList<>())) {
if (isCyclic(neighbor, visited, recursionStack)) {
return true;
}
}
recursionStack.put(node, false);
return false;
}
public void displayGraph() {
for (Map.Entry<Integer, List<Integer>> entry : adjacencyList.entrySet()) {
System.out.print(entry.getKey() + " -> ");
for (int neighbor : entry.getValue()) {
System.out.print(neighbor + " ");
}
System.out.println();
}
}
public static void main(String[] args) {
Graph graph = new Graph();
graph.addNode(0);
graph.addNode(1);
graph.addNode(2);
graph.addNode(3);
System.out.println("Adding edge 0 -> 1: " + graph.addEdge(0, 1));
System.out.println("Adding edge 1 -> 2: " + graph.addEdge(1, 2));
System.out.println("Adding edge 2 -> 3: " + graph.addEdge(2, 3));
System.out.println("Adding edge 3 -> 1: " + graph.addEdge(3, 1));
System.out.println("Graph adjacency list:");

graph.displayGraph();

}

}
```

OUTPUT:



```
<terminated> Graph [Java Application] C:\Program Files\Java\jdk-17\bin\java
Adding edge 0 -> 1: true
Adding edge 1 -> 2: true
Adding edge 2 -> 3: true
Adding edge 3 -> 1: false
Graph adjacency list:
0 -> 1
1 -> 2
2 -> 3
3 ->
```

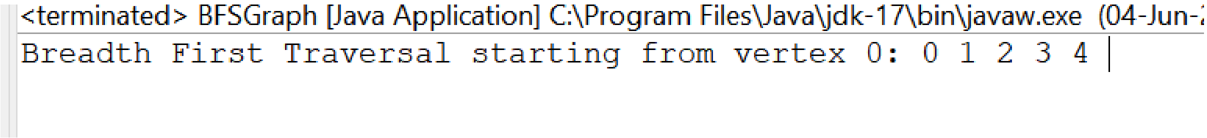
Task 5: Breadth-First Search (BFS) Implementation

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.

```
package com.sorting;


import java.util.ArrayList;

import java.util.HashMap;

import java.util.HashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Queue;

import java.util.Set;


public class BFSGraph {
```

```
int vertices;
LinkedList<Integer>[] adjList;

@SuppressWarnings("unchecked") BFSGraph(int vertices)
{
    this.vertices = vertices;
    adjList = new LinkedList[vertices];
    for (int i = 0; i < vertices; ++i)
        adjList[i] = new LinkedList<>();
}


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);
        }
      }
    }

}

  public static void main(String[] args)
  {

    int vertices = 5;
    BFSGraph graph = new BFSGraph(vertices);
    graph.addEdge(0, 1);
    graph.addEdge(0, 2);
    graph.addEdge(1, 3);
    graph.addEdge(1, 4);
    graph.addEdge(2, 4);

    System.out.print(
      "Breadth First Traversal starting from vertex 0: ");
    graph.bfs(0);

}
  }
```

Output:



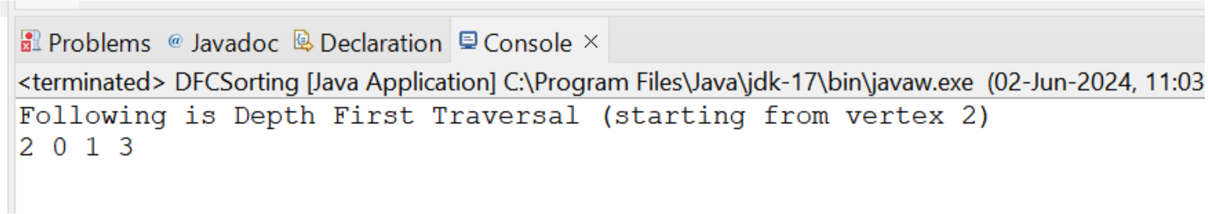
```
<terminated> BFSGraph [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (04-Jun-
Breadth First Traversal starting from vertex 0: 0 1 2 3 4
```

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.

```
package com.sorting;
import java.util.Iterator;
import java.util.LinkedList;
public class DFCSorting {
private int V;
private LinkedList<Integer> adj[];
@SuppressWarnings("unchecked") DFCSorting(int v)
{
V = v;
adj = new LinkedList[v];
for (int i = 0; i < v; ++i)
adj[i] = new LinkedList();
}
// Function to add an edge into the graph
void addEdge(int v, int w)
{
// Add w to v's list.
adj[v].add(w);
}
void DFSUtil(int v, boolean visited[])
{
visited[v] = true;
System.out.print(v + " ");
Iterator<Integer> i = adj[v].listIterator();
while (i.hasNext()) {
int n = i.next();
if (!visited[n])

DFSUtil(n, visited);

}

}

void DFS(int v)

{

boolean visited[] = new boolean[V];

DFSUtil(v, visited);

}

public static void main(String args[])

{

DFCSorting g=new DFCSorting(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

System.out.println(

"Following is Depth First Traversal "

+ "(starting from vertex 2)");

// Function call

g.DFS(2);

}

}
```

OUTPUT:



<terminated> DFCSorting [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-Jun-2024, 11:03

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```