Author : Imtiaz Adar

Phone : +8801778-767775

Email : imtiaz-adar@hotmail.com

DATA STRUCTURE AND ALGORITHMS

BFS :

```
/*
 * Imtiaz Adar
 * BFS
 */
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class BFS {
        private ArrayList<ArrayList<Integer>> adj;
private int vertex;

BFS(int vertex){

    this.vertex = vertex;

    adj = new ArrayList<ArrayList<Integer>>();

    for(int i = 0; i < vertex; i++) {

        adj.add(new ArrayList<Integer>());

    }

}

public void addEdge(int u, int v) {

    adj.get(u).add(v);

    adj.get(v).add(u);

}

public void bfs(int source) {

    boolean[] visited = new boolean[vertex];

    for(int i = 0; i < visited.length; i++)

        visited[i] = false;

    LinkedList<Integer> queue = new LinkedList<Integer>();

    visited[source] = true;

    queue.add(source);

    while(queue.size() != 0) {

        int node = queue.poll();

        System.out.print(node + " ");

        Iterator<Integer> it = adj.get(source).iterator();

        while(it.hasNext()) {
            int val = it.next();
            if(!visited[val]) {
                visited[val] = true;
                queue.add(val);
            }
        }
    }
    System.out.println();
}
public static void main(String[] args) {
    FastScanner scan = new FastScanner();
    System.out.println("Enter how many nodes you want to add : ");
    int nodes = scan.nextInt();
    BFS obj = new BFS(nodes);
    obj.addEdge(0, 2);
    obj.addEdge(1, 2);
    obj.addEdge(2, 0);
    obj.addEdge(2, 3);
    obj.addEdge(3, 3);
    System.out.println("Enter source : ");
    int source = scan.nextInt();
    System.out.println("\n< - BFS - >\n");
    obj.bfs(source);
}
static class FastScanner{
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
    StringTokenizer st = new StringTokenizer("");
    String next(){
        while(!st.hasMoreTokens()) {
            try {
                st = new StringTokenizer(br.readLine());
            }
            catch(IOException e) {
                e.printStackTrace();
            }
        }
        return st.nextToken();
    }
    String nextLine() {
        String str = "";
        try{
            str = br.readLine();
        }
        catch(IOException e) {
            e.printStackTrace();
        }
        return str;
    }
```

```
		int nextInt() {
			return Integer.parseInt(next());
		}
		long nextLong() {
			return Long.parseLong(next());
		}
		double nextDouble() {
			return Double.parseDouble(next());
		}
		int[] readIntArray(int size) {
			int[] x = new int[size];
			for(int i = 0; i < x.length; i++) {
				x[i] = nextInt();
			}
			return x;
		}
	}
}
```

DFS :

```
/*
 * Imtiaz Adar
 * DFS
 */
```

```
import java.util.ArrayList;
import java.util.Iterator;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class DFS {
    private ArrayList<ArrayList<Integer>> adj;
    private int vertex;
    DFS(int vertex){
        this.vertex = vertex;
        adj = new ArrayList<ArrayList<Integer>>();
        for(int i = 0; i < vertex; i++) {
            adj.add(new ArrayList<Integer>());
        }
    }
    public void addEdge(int u, int v) {
        adj.get(u).add(v);
        adj.get(v).add(u);
    }
    public void dfsUtil(boolean[] visited, int source) {
        visited[source] = true;
        System.out.print(source + " ");
        Iterator<Integer> it = adj.get(source).iterator();
        while(it.hasNext()) {
            int node = it.next();
            if(!visited[node]) {
                dfsUtil(visited, node);
            }
        }
    }
    public void dfs() {
        boolean[] visited = new boolean[vertex];
        for(int i = 0; i < visited.length; i++) {
            if(!visited[i]) {
                dfsUtil(visited, i);
            }
        }
    }
    public static void main(String[] args) {
        FastScanner scan = new FastScanner();
        System.out.println("Enter how many nodes you want to add : ");
        int nodes = scan.nextInt();
        DFS obj = new DFS(nodes);
        obj.addEdge(0, 2);
        obj.addEdge(1, 2);
        obj.addEdge(2, 0);
        obj.addEdge(2, 3);

        obj.addEdge(3, 3);

        System.out.println("\n< - DFS - >\n");

        obj.dfs();

        System.out.println();

    }

    static class FastScanner{

        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

        StringTokenizer st = new StringTokenizer("");

        String next(){

            while(!st.hasMoreTokens()) {

                try {

                    st = new StringTokenizer(br.readLine());

                }

                catch(IOException e) {

                    e.printStackTrace();

                }

            }

            return st.nextToken();

        }

        String nextLine() {

            String str = "";

            try{

                str = br.readLine();
            }
            catch(IOException e) {
                e.printStackTrace();
            }
            return str;
        }
        int nextInt() {
            return Integer.parseInt(next());
        }
        long nextLong() {
            return Long.parseLong(next());
        }
        double nextDouble() {
            return Double.parseDouble(next());
        }
        int[] readIntArray(int size) {
            int[] x = new int[size];
            for(int i = 0; i < x.length; i++) {
                x[i] = nextInt();
            }
            return x;
        }
    }
}
```

BINARY SEARCH :

```
public class Binary_Search {

    public static int binary_search(int[] arr, int ele) {
        int left = 0;
        int right = arr.length - 1;
        while(left <= right) {
            int mid = left + (right - left) / 2;
            if(arr[mid] == ele)
                return mid;
            else if(arr[mid] < ele)
                left = mid + 1;
            else right = mid - 1;
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50, 60, 70};
        int pos = binary_search(arr, 50);
        if(pos != -1)
            System.out.println("Element exists in position : " + pos);
        else System.out.println("Element Not exists");
    }
}
```

LINEAR SEARCH :

```
/*
 * Imtiaz Adar
 * Linear Search
 */
public class Linear_Search {

	public static int linear_search(int[] arr, int ele) {
		for(int i = 0; i < arr.length; i++) {
			if(arr[i] == ele) {
				return i;
			}
		}
		return -1;
	}
	public static void main(String[] args) {
		int[] arr = {10, 20, 30, 40, 50, 60, 70};
		int pos = linear_search(arr, 70);
		if(pos != -1)
			System.out.println("Element exists in position : " + pos);
		else System.out.println("Element Not exists");
	}
}
```

BINARY SEARCH TREE :

```
/*
 * Imtiaz Adar
 * Binary Search Tree
 */
public class Binary_Search_Tree {
    class BST{
        int data;
        BST left, right;
    }

    BST NEWNODE(int data) {
        BST temp = new BST();
        temp.data = data;
        temp.left = temp.right = null;
        return temp;
    }

    BST Insert(BST root, int data) {
        if(root == null) {
            root = NEWNODE(data);
        }
        if(data < root.data) {
            root.left = Insert(root.left, data);
        }
        else if(data > root.data) {
            root.right = Insert(root.right, data);
        }
        return root;
    }

    BST Find_Min(BST root) {
        while(root.left != null) {
            root = root.left;
        }
        return root;
    }

    BST Find_Max(BST root) {
        while(root.right != null) {
            root = root.right;
        }
        return root;
    }

    void Inorder(BST root) {
            if(root != null) {
                    Inorder(root.left);
                    System.out.print("--> " + root.data);
                    Inorder(root.right);
            }
    }

    void Preorder(BST root) {
            if(root != null) {
                    Preorder(root.left);
                    Preorder(root.right);
                    System.out.print("--> " + root.data);
            }
    }

    void Postorder(BST root) {
            if(root != null) {
                    System.out.print("--> " + root.data);
                    Postorder(root.left);
                    Postorder(root.right);
            }
    }

    BST Search(BST root, int data) {
        if(root == null)
            return null;
        else if(data < root.data) {
            return Search(root.left, data);
        }
        else if(data > root.data) {
            return Search(root.right, data);
        }
        return root;
    }

    BST Remove(BST root, int data) {
        if(root == null)
            return null;
        else if(data < root.data) {
            root.left = Remove(root.left, data);
        }
        else if(data > root.data) {
            root.right = Remove(root.right, data);
        }
        else {
            if(root.left == null) {
                return root.right;
            }
            else if(root.right == null) {
                return root.left;
            }
            BST temp = Find_Min(root.right);
            root.data = temp.data;
            root.right = Remove(root.right, root.data);
        }
        return root;
    }

    void Find(BST root, int num) {
        if(Search(root, num) != null) {
            System.out.println(num + " FOUND");
        }
        else {
            System.out.println(num + " NOT FOUND");
        }
    }

    public static void main(String[] args) throws NullPointerException{
        Binary_Search_Tree tree = new Binary_Search_Tree();
        BST node = null;
        String inorder = "## INORDER ##\n";
        String preorder = "## PREORDER ##\n";
```

```
        String postorder = "## POSTORDER ##\n";
        node = tree.Insert(node, 15);
        tree.Insert(node, 24);
        tree.Insert(node, 12);
        tree.Insert(node, 33);
        tree.Insert(node, 46);
        System.out.println(inorder);
        tree.Inorder(node);
        System.out.println("\n");
        System.out.println(preorder);
        tree.Preorder(node);
        System.out.println("\n");
        System.out.println(postorder);
        tree.Postorder(node);
        System.out.println("\n");
        tree.Remove(node, 33);
        System.out.println(inorder);
        tree.Inorder(node);
        System.out.println("\n");
        tree.Find(node, 12);
    }
}
```

BUBBLE SORT :

```java
/*
 * Imtiaz Adar
 * Bubble Sort
 */

public class Bubble_Sort {
    public static void bubblesort(int[] arr, int n) {
        for(int i = 0; i < n - 1; i++) {
            for(int j = 0; j < n - i - 1; j++) {
                if(arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }
    public static void printArray(int[] arr) {
        for(int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        int[] arr = {12, 5, 22, 88, 44, 33};
        int size = arr.length;
        bubblesort(arr, size);
        printArray(arr);
    }
}
```

INSERTATION SORT :

```
/*
 * Imtiaz Adar
 * Insertation Sort
 */
public class Insertation__Sort {

    public static void insertationsort(int[] arr) {
        for(int i = 1; i < arr.length; i++) {
            int temp = arr[i];
            int j = i - 1;
            while(j >= 0 && arr[j] > temp) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = temp;
        }
    }
    public static void printArray(int[] arr) {
        for(Integer it : arr) {
            System.out.print(it + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        int[] arr = {2, 44, 12, 23, 52, 77, 33};
        insertationsort(arr);
        printArray(arr);
    }
}
```

MERGE SORT :

```
/*
 * Imtiaz Adar
 * Merge Sort
 */
public class Merge_Sort {

    public static void merge_sort(int[] arr, int l, int mid, int r) {
        int n1 = mid - l + 1;
        int n2 = r - mid;
        int[] left = new int[n1];
```

```
int[] right = new int[n2];
for(int i = 0; i < n1; i++) {
    left[i] = arr[l + i];
}
for(int i = 0; i < n2; i++) {
    right[i] = arr[mid + i + 1];
}
int i = 0, j = 0, k = l;
while(i < n1 && j < n2) {
    if(left[i] <= right[j]) {
        arr[k] = left[i];
        i++;
    }
    else {
        arr[k] = right[j];
        j++;
    }
    k++;
}
while(i < n1) {
    arr[k] = left[i];
    i++;
    k++;
}
```

```
    while(j < n2) {
        arr[k] = right[j];
        j++;
        k++;
    }
}
public static void merge(int[] arr, int l, int r) {
    if(l < r) {
        int mid = l + (r - l) / 2;
        merge(arr, l, mid);
        merge(arr, mid + 1, r);
        merge_sort(arr, l, mid, r);
    }
}
public static void printArray(int[] arr) {
    for(Integer item : arr) {
        System.out.print(item + " ");
    }
    System.out.println();
}
public static void main(String[] args) {
    int[] arr = {4, 12, 7, 23, 25, 24, 55, 33, 37, 9};
    merge(arr, 0, arr.length - 1);
    printArray(arr);
```

```
	}
}
```

QUICK SORT :

```
/*
 * Imtiaz Adar
 * Quick Sort
 */
public class Quick_Sort {

	public static void swap(int[] arr, int i, int j) {
		int temp = arr[i];
		arr[i] = arr[j];
		arr[j] = temp;
	}
	public static int partial(int[] arr, int low, int high) {
		int higher = arr[high];
		int i = low - 1;
		for(int j = low; j <= high - 1; j++) {
			if(arr[j] < higher) {
				i++;
				swap(arr, i, j);
			}
		}
```

```
        swap(arr, i + 1, high);
        return i + 1;
    }
    public static void quicksort(int[] arr, int low, int high) {
        if(low < high) {
            int part = partial(arr, low, high);
            quicksort(arr, low, part - 1);
            quicksort(arr, part + 1, high);
        }
    }
    public static void printArray(int[] arr) {
        for(Integer it : arr) {
            System.out.print(it + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        int[] arr = {2, 44, 12, 23, 52, 77, 33};
        quicksort(arr, 0, arr.length - 1);
        printArray(arr);
    }
}
```

SELECTION SORT :

```
/*
 * Imtiaz Adar
 * Selection Sort
 */

public class Selection_Sort {
    public static void selectionsort(int[] arr, int size) {
        for(int i = 0; i < size - 1; i++) {
            int minindex = i;
            for(int j = i + 1; j < size; j++) {
                if(arr[minindex] > arr[j]) {
                    minindex = j;
                }
            }
            int temp = arr[minindex];
            arr[minindex] = arr[i];
            arr[i] = temp;
        }
    }
    public static void printArray(int[] arr) {
        for(int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        int[] arr = {66, 3, 22, 11, 13, 77};
        int size = arr.length;
        selectionsort(arr, size);
        printArray(arr);
    }
}
```

SINGLY LINKED LIST :

```
/*
 * Imtiaz Adar
 * Linked List [Singly]
 */
public class Singly_Linked_List {
    Node head;

    class Node{
        int data;
        Node next;
        Node(int data){
            this.data = data;
            this.next = null;
        }
    }

    public void insert_head(int data) {
        Node newnode = new Node(data);
        if(this.head == null) {
            this.head = newnode;
            return;
        }
        newnode.next = this.head;
        this.head = newnode;
    }

    public void insert_tail(int data) {
        Node newnode = new Node(data);
        if(this.head == null) {
            this.head = newnode;
            return;
        }
        Node currentNode = this.head;
        while(currentNode.next != null) {
            currentNode = currentNode.next;
        }
        currentNode.next = newnode;
    }
```

```
    }

public void delete_head() {
    if(this.head == null) {
        System.out.println("This Linked List Is Empty");
        return;
    }
    this.head = this.head.next;
}

public void delete_tail() {
    if(this.head == null) {
        System.out.println("This Linked List Is Empty");
        return;
    }
    if(this.head.next == null) {
        this.head = null;
        return;
    }
    Node secondLast = this.head;
    Node last = this.head.next;
    while(last.next != null) {
        last = last.next;
        secondLast = secondLast.next;
    }
    secondLast.next = null;
}
public void delete_by_value(int val) {
    if(this.head == null) {
        System.out.println("Linked List Is Empty");
        return;
    }
    if(this.head.data == val) {
        this.head = this.head.next;
        return;
    }
    Node node = this.head;
    while(node.next != null) {
        if(node.data == val)
            break;
        node = node.next;
    }
    if(node.next == null) {
        System.out.println("Element Not Found");
    }
    else {
        node.next = node.next.next;
    }
```

```
}
public void displayLinkedList() {
        Node temp = this.head;
        System.out.println("Displaying Linked List");
        while(temp != null) {
                System.out.print(temp.data + "-> ");
                temp = temp.next;
        }
        System.out.print("NULL" + "\n");
}
public static void main(String[] args) {
        Singly_Linked_List list = new Singly_Linked_List();
        list.insert_head(56);
        list.insert_tail(16);
        list.insert_tail(54);
        list.insert_tail(76);
        list.insert_tail(82);
        list.insert_head(113);
        list.insert_head(25);
        list.displayLinkedList();
        list.delete_head();
        list.delete_tail();
        list.displayLinkedList();
        list.insert_tail(22);
        list.delete_by_value(113);
        list.displayLinkedList();
    }
}
```

DOUBLY LINKED LIST :

```
/*
 * Imtiaz Adar
 * Linked List [Doubly]
 */
public class Doubly_Linked_List {
    Node head;

    class Node{
        Node next;
        Node prev;
        int data;
        Node(int data){
            this.data = data;
            this.next = this.prev = null;
        }
    }

    void insert_head(int data) {
        Node newNode = new Node(data);
        newNode.next = this.head;
        if(this.head == null) {
            this.head = newNode;
        }
        else {
            newNode.next = this.head;
            this.head.prev = newNode;
            this.head = newNode;
        }
    }

    void insert_tail(int data) {
        Node newNode = new Node(data);
        if(this.head == null) {
            this.head = newNode;
        }
        else {
            Node currNode = this.head;
            while(currNode.next != null) {
                currNode = currNode.next;
            }
            currNode.next = newNode;
            newNode.prev = currNode;
    }
}

void insert_after(int data, int given) {
    if(this.head == null) {
        System.out.println("This Is an Empty Linked List");
    }
    else {
        Node node = this.head;
        while(node != null) {
            if(node.data == given)
                break;
            node = node.next;
        }
        if(node == null) {
            System.out.println("Node Not Found");
        }
        else {
            Node newnode = new Node(data);
            newnode.next = node.next;
            newnode.prev = node;
            if(node.next != null) {
                node.next.prev = newnode;
            }
            node.next = newnode;
        }
    }
}

void insert_begin(int data, int given) {
    if(this.head == null) {
        System.out.println("This Is an Empty Linked List");
    }
    else {
        Node node = this.head;
        while(node != null) {
            if(node.data == given)
                break;
            node = node.next;
        }
        if(node == null) {
            System.out.println("Node Not Found");
        }
        else {
            Node newnode = new Node(data);
            newnode.next = node;
            newnode.prev = node.prev;
            if(node.prev != null) {
                node.prev.next = newnode;
            }
            else {
                this.head = newnode;
                node.prev = newnode;
            }
        }
    }
}

void delete_head() {
    if(this.head == null) {
        System.out.println("Linked List Is Empty");
        return;
    }
    if(this.head.next == null) {
        this.head = null;
    }
    else {
        this.head = this.head.next;
        this.head.prev = null;
    }
}
```

```
void delete_tail() {
    if(this.head == null) {
        System.out.println("Linked List Is Empty");
        return;
    }
    if(this.head.next == null) {
        this.head = null;
    }
    else {
        Node node = this.head;
        while(node.next != null) {
            node = node.next;
        }
        node.prev.next = null;
    }
}

void delete_by_value(int value) {
    if(this.head == null) {
        System.out.println("Linked List Is Empty");
        return;
    }
    if(this.head.next == null) {
        if(value == this.head.data) {
            this.head = null;
        }
        else {
            System.out.println("Element Not Present In The Linked List");
        }
        return;
    }
    if(this.head.data == value) {
        this.head = this.head.next;
        this.head.prev = null;
        return;
    }
    Node node = this.head;
    while(node.next != null) {
        if(node.data == value)
            break;
        node = node.next;
    }
    if(node.next != null) {
        node.next.prev = node.prev;
        node.prev.next = node.next;
    }
    else {
        if(node.data == value) {
```

```
            node.next.prev = null;
        }
        else {
            System.out.println("Not Present");
        }
    }
}

void display_Linked_List() {
    Node temp = this.head;
    System.out.println("Displaying Linked List");
    while(temp != null) {
        System.out.print(temp.data + "-> ");
        temp = temp.next;
    }
    System.out.print("NULL" + "\n");
}

public static void main(String[] args) {
    Doubly_Linked_List doublelist = new Doubly_Linked_List();
    doublelist.insert_head(12);
    doublelist.insert_tail(121);
    doublelist.insert_tail(24);
    doublelist.insert_tail(44);
```

```
		doublelist.insert_tail(33);
		doublelist.insert_begin(55, 33);
		doublelist.insert_tail(17);
		doublelist.insert_after(444, 24);
		doublelist.display_Linked_List();
		doublelist.delete_head();
		doublelist.delete_tail();
		doublelist.delete_by_value(444);
		doublelist.display_Linked_List();

	}
}
```

RECURSION [Fibonacci] :

```
/*
 * Imtiaz Adar
 * Fibonacci (Recursion)
 */
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class Fibonacci_Recursion {
	public static int fibonacci(int n) {
		if(n < 2) return n;
		return fibonacci(n - 1) + fibonacci(n - 2);
	}

	public static void main(String[] args) {
		FastScanner scan = new FastScanner();
		System.out.println("Enter the limit : ");
		int lim = scan.nextInt();
		for(int i = 0; i < lim; i++) {
			int ans = fibonacci(i);
			System.out.print(ans + " ");
		}
		System.out.println();
	}
	static class FastScanner{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		StringTokenizer st = new StringTokenizer("");
		String next(){
			while(!st.hasMoreTokens()) {
				try {
					st = new StringTokenizer(br.readLine());
				}
				catch(IOException e) {
```

```
                e.printStackTrace();
            }
        }
        return st.nextToken();
    }
    String nextLine() {
        String str = "";
        try{
            str = br.readLine();
        }
        catch(IOException e) {
            e.printStackTrace();
        }
        return str;
    }
    int nextInt() {
        return Integer.parseInt(next());
    }
    long nextLong() {
        return Long.parseLong(next());
    }
    double nextDouble() {
        return Double.parseDouble(next());
    }
```

```
        int[] readIntArray(int size) {
            int[] x = new int[size];
            for(int i = 0; i < x.length; i++) {
                x[i] = nextInt();
            }
            return x;
        }
    }
}
```

RECURSION [TOWER OF HANOI] :

```
/*
 * Author : Imtiaz Adar
 * Tower Of Hanoi
 */
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class Tower_Of_Hanoi_Recursion {
    public static void towerOfHanoi(int n, char from, char aux, char to) {
        if(n == 1) {
            System.out.println("MOVE DISK 1 FROM " + from + " TO " + to);
            return;
        }
        towerOfHanoi(n - 1, from, aux, to);
        System.out.println("MOVE DISK " + n + " FROM " + from + " TO " +
to);
        towerOfHanoi(n - 1, aux, to, from);
    }
    public static void main(String[] args) {
        FastScanner scan = new FastScanner();
        System.out.println("Enter number of disks : ");
        int n = scan.nextInt();
        towerOfHanoi(n, 'A', 'C', 'B');
    }
    static class FastScanner{
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer("");
        String next(){
            while(!st.hasMoreTokens()) {
                try {
                    st = new StringTokenizer(br.readLine());
                }
                catch(IOException e) {
                    e.printStackTrace();
                }
            }
            return st.nextToken();
        }
        String nextLine() {
            String str = "";
            try{
                str = br.readLine();
            }
            catch(IOException e) {
                e.printStackTrace();
            }
            return str;
        }
        int nextInt() {
            return Integer.parseInt(next());
        }
        long nextLong() {
            return Long.parseLong(next());
        }
        double nextDouble() {
            return Double.parseDouble(next());
        }
        int[] readIntArray(int size) {
            int[] x = new int[size];
```

```
            for(int i = 0; i < x.length; i++) {
                x[i] = nextInt();
            }
            return x;
        }
    }
}
```