\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Krushkals Algorithm

public class Kruskal {

    public void KruskalAlgo(Edge edges[],int vertices) {

        int mst[][] = new int[vertices][vertices];

        Arrays.sort(edges);

        //i

        int edgeCounter = 0;

        int edgeTaken = 0;

        int parent[] = new int[vertices];

        int rank[] = new int[vertices];

        for(int i=0;i<vertices;i++) {

            parent[i]=-1;

            rank[i] = 0;

        }

        while(edgeTaken != vertices-1){

            Edge e = edges[edgeCounter];

            if(!isCyclic(e.u,e.v,parent)) {

                union(findParent(e.u, parent), findParent(e.v, parent), parent, rank);

                mst[e.u][e.v] = e.w;

                edgeTaken++;

            }

            edgeCounter++;

        }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Topological sorting

public class TopologicalSorting {

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

    public boolean isCyclic(int n,ArrayList<ArrayList<Integer>> adj) {

        int indegree[] = new int[n];

        for(int i=0;i<n;i++) {

            for(Integer j : adj.get(i)) {

                indegree[j]++;

            }

        }

        Queue<Integer> q = new LinkedList<Integer>();

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

            int degree = indegree[i];

            if(degree==0) {

                q.add(degree);

            }

        }

        int count=0;

        while(!q.isEmpty()) {

            int node = q.poll();

            Sorted.add(node);

            count++;

            for(Integer j : adj.get(node)) {

                indegree[j]--;

                if(indegree[j]==0) {

                    q.add(j);

                }

            }

        }

        if(count==n) {

            System.out.println(Sorted);

            return false;

        }else {

            return true;

        } } }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Balanced Paranthesis

public boolean balancedParanthesis(String exp) {

        Stack<Character> st = new Stack<Character>();

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

            char c = exp.charAt(i);

            if(c=='{' || c=='(' || c=='[') {

                st.push(c);

            }

            else if(c=='}' || c==')' || c==']' ) {

                if(!st.empty() && (c=='}' && st.peek()=='{') || (c==')' && st.peek()=='(') || (c==']' && st.peek()=='[') ) {

                    st.pop();

                }

                else {

                    st.push(c);

                }

            }

        }

        if(st.empty()) {

            return true;

        }

        else {

            return false;

        }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Alternate Alphabets

public class AlphabetsAlternate {

    public void printOdd() {

        try {

            char c = 'a';

            while(c<='z') {

                System.out.println(Thread.currentThread().getName()+" "+c);

                c++;

                c++;

                Thread.sleep(1000);

            }

        }catch (Exception e) {

            System.out.println(e);

        }

    }

    public void printEven() {

                    try {

                        char c = 'b';

                        while(c<='z') {

                            System.out.println(Thread.currentThread().getName()+" "+c);

                            c++;

                            c++;

                            Thread.sleep(1500);

                        }

                    } catch (InterruptedException e) {

                        System.out.println(e);

                    }

    }

    public static void main(String[] args) {

        AlphabetsAlternate obj = new AlphabetsAlternate();

Thread t1 = new Thread(new Runnable() {

            public void run() {

                //print odd

                obj.printOdd();

            }

        },"Thread-1");

        Thread t2 = new Thread(new Runnable() {

            public void run() {

                //print even

                obj.printEven();

            }

        },"Thread-2");

        t1.start();

        t2.start();

    }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Odd Even two threads

public class OddEvenThreads {

    public void printOdd(int limit) {

        try {

            for(int i=1;i<limit;i++) {

                if(i%2!=0) {

                    System.out.println(Thread.currentThread().getName()+" "+ i);

                }

                Thread.sleep(1000);

            }

        }catch (Exception e) {

            System.out.println(e);

        }

    }

    public void printEven(int limit) {

                    try {

                        for(int i=1;i<limit;i++) {

                            if(i%2==0) {

                                System.out.println(Thread.currentThread().getName()+" " + i );

                            }

                            Thread.sleep(1000);

                        }

                    } catch (InterruptedException e) {

                        System.out.println(e);

                    }

    }

    public static void main(String[] args) {

        OddEvenThreads obj = new OddEvenThreads();

        Thread t1 = new Thread(new Runnable() {

            public void run() {

                //print odd

                obj.printOdd(15);

            }

        },"Thread-1");

        Thread t2 = new Thread(new Runnable() {

            public void run() {

                //print even

                obj.printEven(15);

            }

        },"Thread-2");

        t1.start();

        t2.start();

    }

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Quick Sort

Text

Description automatically generated

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bubble Sort

Table

Description automatically generated

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Merge Sort

Table

Description automatically generated with medium confidence

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Binary Search

Table

Description automatically generated

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Binary Search Tree

list insertBST(list node,list root) {

        if(root==null) {

            root=node;

        }

        else if(node.data<root.data) {

            root.left = insertBST(node, root.left);

        }

        else if(node.data>root.data){

            root.right = insertBST(node, root.right);

        }

        return root;

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LRU

public void put(int key,int value) {

        list node = new list(key,value);

        if(!map.containsKey(node.data)) {

            if(map.size()==cachesize) {

                map.remove(head.data);

                remove(head);

                insert(node);

                map.put(node.data, node);

            }

            else {

                insert(node);

                map.put(node.data, node);

            }

        }

        else {

            remove(node);

            insert(node);

        }

    }

    public int get(int key) {

        list node = map.get(key);

        if(node==null) {

            return -1;

        }

        else{

return node.value;

        } }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Doubly Linked List

public void add(int data) {

        list node = new list(data);

        if(head==null) {

            head=node;

            tail=head;

        }

        else {

            tail.right=node;

            node.left=tail;

            tail=node;

        }

    }

    public void delete(int data) {

        list temp =head;

        if(head==null) {

            System.out.println("nothing there");

        }

        else if(temp.data==data) {

            head=head.next;

        }

        else {

            while(temp.right!=null) {

                list prev=temp.left;

                list next = temp.right;

                if(temp.data==data) {

                    prev.right = next;

                    next.left =prev;

                }

            }

        }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MultiThreading Example

class Mutlithreading\_by\_extending\_thread\_class extends Thread{

    public void run(){

try {

     for(int i=1;i<6;i++) {

         System.out.println(i);

         Thread.sleep(2000);

         }

}

catch (Exception e) {

System.out.println("Exception is caught");

}

}

}

class Multithreading\_by\_implementing\_runnable\_interface implements Runnable{

        public void run()

     {

     try {

          for(int i=55;i<61;i++) {

             System.out.println(i);

             Thread.sleep(2000);

             }

     }

     catch (Exception e) {

     System.out.println("Exception is caught");

     }

     }

    }

public class Multithreading {

    public static void main(String[] args) {

            Thread obj1= new Thread(new Multithreading\_by\_implementing\_runnable\_interface());

            obj1.start();

            Thread obj2 = new Thread(new Mutlithreading\_by\_extending\_thread\_class());

            obj2.start();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Circular Queue

public void enqueue(int data) {

        Node entry = new Node(data);

        if(head==null) {

            head=entry;

            tail=head;

            tail.next=head;

        }

        else {

            tail.next=entry;

            tail = entry;

            tail.next = head;

        }

    }

    public void dequeue() {

        if(head==null) {

            System.out.println("Nothing present");

            return;

        }

        else {

            head=head.next;

            tail.next=head;

        }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Single Queue

public void enqueue(int data) {

        Node entry = new Node(data);

        if(head==null) {

            head=entry;

            tail=head;

        }

        else {

            tail.next=entry;

            tail = entry;

        }

    }

    public void dequeue() {

        if(head==null) {

            System.out.println("Nothing present");

            return;

        }

        else {

            head=head.next;

        }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Stack

public void push(int data) {

            Node entry = new Node(data);

            if(head==null) {

                head = entry;

            }

            else {

                entry.next = head;

                head= entry;

            }

        }

        public void pop() {

             if(head==null) {

                System.out.println("Stack Underflow");

                return;

             }

             else {

                 head = head.next;

             }

        }

        public int peek() {

            if(head==null) {

                return -1;

            }

            else {

                return head.data;

            }

        }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Linked List add and remove node

public void addNode(int data) {

        list node = new list(data);

        if(head==null) {

            head = node;

tail = head;

        }

        else {

            tail.next=node;

tail=node;

        }

    }

    public void remove(int data) {

        list temp=head;

        list prev=null;

        if(head==null) {

            return ;

        }

        else if(temp.data==data) {

            head = head.next;

        }

        else {

            while(temp.next!=null) {

                prev=temp;

                temp=temp.next;

                if(temp.data==data) {

                    prev.next = temp.next;

                    break;

                }

            }

        }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Modified Bfs

public static void bfs(int graph[][],int source,int destination) {

        int v = graph.length;

        boolean visited[] = new boolean[v];

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

        int parent[] = new int[v];

        queue.add(source);

        visited[source]=true;

        while( queue.size()>0) {

            int popped = queue.poll();

                for(int j=0;j<v;j++) {

                    if(graph[popped][j]==1 && !visited[j]) {

                        visited[j]=true;

                        queue.add(j);

                        parent[j]=popped;

                    }

                }

        }

        Queue<Integer> path = new LinkedList<Integer>();

        path.add(destination);

        int prev = destination;

        for(int i=0;i<v;i++) {

             prev = parent[prev];

            if(prev==source) {

                path.add(prev);

                break;

            }

            path.add(prev);

        }

        while(path.size()>0) {

            System.out.println(path.poll());

        }

        }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bfs

public static void bfs(int graph[][],int source) {

        int v = graph.length;

        boolean visited[] = new boolean[v];

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

        queue.add(source);

        visited[source]=true;

        while( queue.size()>0) {

            int popped = queue.poll();

            System.out.println(popped+" ");

                for(int j=0;j<v;j++) {

                    if(graph[popped][j]==1 && !visited[j]) {

                        visited[j]=true;

                        queue.add(j);

                    }

                }

        }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Dfs

    public static void dfs(int graph[][],int source) {

        boolean visited[] = new boolean[graph.length];

        dfsRecursive(graph,source,visited);

    }

    public static void dfsRecursive(int graph[][],int source,boolean visited[]) {

         visited[source] = true;

        System.out.println(source + " ");

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

             if(graph[source][i]!=0 && !visited[i]) {

                 dfsRecursive(graph,i,visited);

             }

         }

    }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Dijkstra ALgorithm

public static void dijikstra(int adjMatrix[][]) {

int v = adjMatrix.length;

boolean visited[] = new boolean[v];

int distance[]=new int[v];

int parent[] = new int[v];

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

distance[i]=Integer.MAX\_VALUE;

}

distance[0]=0;

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

//find vertex with min vertex

int minVetex = findMinVertex(distance,visited);

visited[minVetex] = true;

//explore neighbors

for(int j=0;j<v;j++) {

if(adjMatrix[minVetex][j]!=0 && !visited[j]) {

int newDistance = distance[minVetex]+adjMatrix[minVetex][j];

if(newDistance<distance[j]) {

distance[j]=newDistance;

parent[j]=minVetex;

}

}

}

}

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

System.out.println(i + " "+distance[i]+ " " + parent[i]);

}

}

public static int findMinVertex(int distance[],boolean visited[]) {

int minVertex=-1;

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

if(!visited[i] && (minVertex==-1 || distance[i] < distance[minVertex])) {

minVertex = i;

}

}

return minVertex;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Postfix to Infix

public static void postfix(String expression) {

        Stack<String> stack = new Stack<String>();

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

            char symbol = expression.charAt(i);

            if( (symbol>='a' && symbol<='z') || (symbol>='A' && symbol<='Z') ) {

                stack.add(""+symbol);

            }

            else if(symbol=='+' || symbol=='-' || symbol=='\*' || symbol=='/') {

                String op2=stack.pop();

                String op1= stack.pop();

                String newExpr = "(" + op1+symbol+op2 + ")";

                stack.add(newExpr);

            }

        }

        String result = stack.pop();

        System.out.println(result);

    }