19BCE301 – Practical 11 – Branch And Bound

Aim: To implement the JOB Assignment problem using Branch and Bound approach.

Code:

```
import java.util.*;
public class JobAssignmentBranchAndBound19BCE301 {

    static int[][] costMatrix;
    static boolean available[];
    static Node leaf;
    static boolean assigned[];

    static class Node{
        int worker;
        int job;
        int cost;
        int sumtillnow;
        Node parent;
        Node(int a,int b,int c,int e,Node d){
            worker=a;
            job=b;
            cost=c;
            sumtillnow=e;
            parent=d;
        }
    }

    static class sortPQ implements Comparator<Node>{
        @Override
        public int compare(Node a,Node b){
            return a.sumtillnow-b.sumtillnow;
        }
    }

    static int calcLB(int i,int n,int worker){
        if(worker==n) return 0;
        int LB=0;
        available[i]=false;
        for(int j=worker;j<n;j++){
            int min=Integer.MAX_VALUE;
```

```
            for(int k=0;k<n;k++){
                if(available[k]){
                    if(costMatrix[j][k]<min){
                        min=costMatrix[j][k];
                    }
                }
            }
            LB+=min;
        }
        available[i]=true;
        return LB;
    }

    static void assign(){
        int n=available.length;
        PriorityQueue<Node> pq = new PriorityQueue<>(new sortPQ());
        pq.add(new Node(-1,-1,0,0,null));
        while(pq.size() > 0){
            Node curr = pq.poll();
            if(curr.worker!=-1 && assigned[curr.worker]){
                continue;
            }
            if(curr.job!=-1){
                available[curr.job]=false;
                assigned[curr.worker]=true;
            }
            int worker = curr.worker+1;
            if(worker==n){
                leaf=curr;
                return;
            }
            for(int i=0;i<n;i++){
                if(available[i]){
                    pq.add(new
Node(worker,i,costMatrix[worker][i],curr.sumtillnow+costMatrix[worker][i]+calc
LB(i,n,worker+1),curr));
                }
            }
        }
    }

    static void display(){
        ArrayList<Node> list=new ArrayList<Node>();
        int cost = 0;
        while(leaf.parent!=null){
            cost+=leaf.cost;
            list.add(leaf);
            leaf=leaf.parent;
```

```
        }
        Collections.reverse(list);
        System.out.println("Minimum cost is: "+cost);
        System.out.println("Jobs assigned are as follows: ");
        for(Node i:list){
            System.out.println("Worker "+i.worker+" is assigned the job
"+i.job);
        }
    }

    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter number of jobs(same as workers): ");
        int n=sc.nextInt();
        System.out.println("Enter the cost matrix: ");
        costMatrix=new int[n][n];
        available=new boolean[n];
        assigned=new boolean[n];
        for(int i=0;i<n;i++){
            available[i]=true;
            assigned[i]=false;
            for(int j=0;j<n;j++){
                costMatrix[i][j]=sc.nextInt();
            }
        }
        assign();
        System.out.println("The Jobs are assigned as follows: ");
        display();
        sc.close();
    }
}
```

Output:

```
PS D:\Design and analysis of algorithms\Practical 10> javac JobAssignmentBranchAndBound19BCE301.java
PS D:\Design and analysis of algorithms\Practical 10> java JobAssignmentBranchAndBound19BCE301
Enter number of jobs(same as workers):
4
Enter the cost matrix:
9 2 7 8
6 4 3 7
5 8 1 8
7 6 9 4
The Jobs are assigned as follows:
Minimum cost is: 13
Jobs assigned are as follows:
Worker 0 is assigned the job 1
Worker 1 is assigned the job 0
Worker 2 is assigned the job 2
Worker 3 is assigned the job 3
PS D:\Design and analysis of algorithms\Practical 10>
```