UNIVERSITY OF ALLAHABAD

PRACTICAL RECORDS
OF
DESIGN AND ANALYSIS OF ALGORITHMS LAB

Name: PARIGYAN

Class: B.TECH (CSE)

Semester: 4^{TH}

Roll No: 24483311

Enrollment No: U2251028

1. Write a program that implements Bubble sort

```
public class bubbleSort {
  static void bubbleSort(int[] arr){
    int n = arr.length;
    int temp = 0;
    for(int i=0;i<n;i++){
      for(int j=1;j<(n-i);j++){
        if(arr[j-1]>arr[j]){
          temp = arr[j-1];
          arr[j-1] = arr[j];
          arr[j] = temp;
        }
      }
    }
  }

  public static void main(String[] args){
    int[] arr = {3,60,35,2,45,320,5};

    System.out.println("Array before Bubble Sort");
    for(int i=0;i<arr.length;i++){
      System.out.print(arr[i]+" ");
    }
    System.out.println();

    bubbleSort(arr);

    System.out.println("Array after Bubble Sort");
    for(int i=0;i<arr.length;i++){
      System.out.print(arr[i]+" ");
    }
  }
}
```

2. Write a program that implements insertion sort

```
public class insertion_sort {
  static void insertionSort(int[] arr){
    int n = arr.length;
    for (int j=1;j<n;j++) {
      int key = arr[j];
      int i = j-1;
      while((i>-1) && (arr[i]>key)){
        arr [i+1] = arr[i];
        i--;
      }
```

```
            arr[i+1] = key;
        }
    }

    public static void main(String[] args){
        int[] arr = {9,14,3,2,43,11,58,22};
        System.out.println("Before Insertion Sort");
        for(int i:arr){
            System.out.print(i+" ");
        }
        System.out.println();

        insertionSort(arr);

        System.out.println("After Insertion Sort");
        for(int i:arr){
            System.out.print(i+" ");
        }
    }
}
```

3. Write a program that implements selection sort

```
public class selection_sort {
    public static void selectionSort(int[] arr){
        for (int i = 0; i < arr.length - 1; i++)
        {
            int index = i;
            for (int j = i + 1; j < arr.length; j++){
                if (arr[j] < arr[index]){
                    index = j;
                }
            }
            int smallerNumber = arr[index];
            arr[index] = arr[i];
            arr[i] = smallerNumber;
        }
    }

    public static void main(String[] args){
        int[] arr = {9,14,3,2,43,11,58,22};
        System.out.println("Before Selection Sort");
        for(int i:arr){
            System.out.print(i+" ");
        }
        System.out.println();
```

```
        selectionSort(arr);

        System.out.println("After Selection Sort");
        for(int i:arr){
            System.out.print(i+" ");
        }
    }
}
```

4. Write a program to implement merge sort

```
public class merge_sort {
    void merge(int[] arr, int l, int m, int r){
        int n1 = m-l+1;
        int n2 = r-m;

        int[] L = new int[n1];
        int[] R = new int[n2];

        for(int i=0;i<n1;i++){
            L[i] = arr[l+i];
        }
        for(int j=0;j<n2;j++){
            R[j] = arr[m+1+j];
        }

        int i=0, j=0;
        int k=l;
        while(i<n1 && j<n2){
            if(L[i]<=R[j]){
                arr[k] = L[i];
                i++;
            } else{
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        while(i<n1){
            arr[k] = L[i];
            i++;
            k++;
        }

        while(j<n2){
            arr[k] = R[j];
```

```
            j++;
            k++;
        }
    }

    void sort(int[] arr,int l,int r){
        if(l<r){
            int m = (l+r)/2;

            sort(arr,l,m);
            sort(arr,m+1,r);

            merge(arr,l,m,r);
        }
    }

    static void printArray(int[] arr){
        int n = arr.length;
        for(int i=0;i<n;i++){
            System.out.print(arr[i]+" ");
        }
        System.out.println();
    }

    public static void main(String[] args){
        int[] arr = {12,11,13,5,6,7};

        System.out.println("Given Array");
        printArray(arr);

        merge_sort obj = new merge_sort();
        obj.sort(arr,0,arr.length-1);

        System.out.println("\nSorted array");
        printArray(arr);
    }
}
```

5. Write a program to Sort a given set of elements using the Quick sort

```
public class quick_sort {
    int partition(int[] arr, int low, int high){
        int pivot = arr[high];
        int i=(low-1);
        for(int j=low;j<high;j++){
            if(arr[j]<=pivot){
                i++;
```

```
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }

      int temp = arr[i+1];
      arr[i+1] = arr[high];
      arr[high] = temp;

      return i+1;
   }

   void sort(int[] arr, int low, int high){
      if(low<high){
         int pi = partition(arr, low, high);

         sort(arr,low,pi-1);
         sort(arr,pi+1,high);
      }
   }

   static void printArray(int[] arr){
      int n = arr.length;
      for(int i=0;i<n;i++){
         System.out.print(arr[i]+" ");
      }
      System.out.println();
   }

   public static void main(String[] args){
      int[] arr = {10,7,8,9,1,5};
      int n = arr.length;

      quick_sort obj = new quick_sort();
      obj.sort(arr, 0, n-1);

      System.out.println("sorted array");
      printArray(arr);
   }
}
```

6. Write a program that implements Linear search.

```
public class linearSearch {
   static int search(int[] arr, int n, int x){
      for(int i=0;i<n;i++){
```

```
      if(arr[i]==x)
        return i;
    }
    return -1;
  }

  public static void main(String[] args){
    int[] arr = {3,4,1,7,5};
    int n = arr.length;
    int x = 4;
    int index = search(arr, n, x);
    if(index==-1)
      System.out.println("Element is not present in the array");
    else
      System.out.println("Element found at position " + index);
  }
}
```

7. Write a program that implements binary search.

```
public class binarySearch {
  public static void binarySearch(int[] arr,int first,int last,int key){
    int mid = (first+last)/2;
    while(first<=last){
      if(arr[mid]<key){
        first = mid+1;
      } else if(arr[mid]==key){
        System.out.println("Element is found at index: " + mid);
        break;
      } else{
        last = mid-1;
      }
      mid = (first+last)/2;
    }
    if(first>last){
      System.out.println("Element is not found");
    }
  }

  public static void main(String[] args){
    int[] arr = {10,20,30,40,50};
    int key = 30;
    int last = arr.length-1;
    binarySearch(arr, 0, last, key);
  }
}
```

8. Write a program to implement Binary search tree

```
class Node{
  int key;
  Node left, right;
  public Node(int item){
    key = item;
    left = right = null;
  }
}

public class BST {
  Node root;
  BST(){
    root = null;
  }

  Node insert(Node node, int key){
    if(node==null){
      node = new Node(key);
      return node;
    }

    if(key<node.key)
      node.left = insert(node.left, key);
    else if(key>node.key)
      node.right = insert(node.right, key);

    return node;
  }

  Node search(Node root, int key){
    if(root==null||root.key==key)
      return root;

    if(root.key<key)
      return search(root.right, key);

    return search(root.left, key);
  }

  public static void main(String[] args){
    BST tree = new BST();

    tree.root = tree.insert(tree.root, 50);
    tree.insert(tree.root, 50);
    tree.insert(tree.root, 20);
    tree.insert(tree.root, 40);
```

```
    tree.insert(tree.root, 70);
    tree.insert(tree.root, 60);
    tree.insert(tree.root, 80);

    int key = 6;

    if(tree.search(tree.root, key)==null)
      System.out.println(key+" not found");
    else
      System.out.println(key+" found");

    key = 60;

    if(tree.search(tree.root, key)==null)
      System.out.println(key+" not found");
    else
      System.out.println(key+" found");
  }
}
```

9. Write a program to find optimal ordering of matrix multiplication

```
public class MatrixChainMultiplication {
  static int[][] m;
  static int[][] s;
  static void matrixChainOrder(int[] p) {
    int n = p.length - 1;
    m = new int[n + 1][n + 1];
    s = new int[n + 1][n + 1];

    for (int i = 1; i <= n; i++) {
      m[i][i] = 0;
    }
    for (int l = 2; l <= n; l++) {
      for (int i = 1; i <= n - l + 1; i++) {
        int j = i + l - 1;
        m[i][j] = Integer.MAX_VALUE;
        for (int k = i; k < j; k++) {
          int q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
          if (q < m[i][j]) {
            m[i][j] = q;
            s[i][j] = k;
          }
        }
      }
    }
  }
```

```java
    static void printOptimalParentheses(int i, int j) {
        if (i == j) {
            System.out.print("A" + i);
        } else {
            System.out.print("(");
            printOptimalParentheses(i, s[i][j]);
            printOptimalParentheses(s[i][j] + 1, j);
            System.out.print(")");
        }
    }

    public static void main(String[] args) {
        int[] matrixSizes = {30, 35, 15, 5, 10, 20, 25};
        matrixChainOrder(matrixSizes);

        System.out.println("Optimal Parenthesization:");
        printOptimalParentheses(1, matrixSizes.length - 1);
        System.out.println("\nMinimum number of scalar multiplications: " +
m[1][matrixSizes.length - 1]);
    }
}
```

10. Implement 0/1 Knapsack problem using Dynamic Programming

```java
public class Knapsack {
    static int knapsack(int W, int[] wt, int[] val, int n) {
        int[][] dp = new int[n + 1][W + 1];

        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= W; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0;
                } else if (wt[i - 1] <= w) {
                    dp[i][w] = Math.max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
        return dp[n][W];
    }

        public static void main(String[] args) {
        int[] val = {60, 100, 120};
        int[] wt = {10, 20, 30};
        int W = 50;
        int n = val.length;
```

```
        System.out.println("Maximum value that can be put in knapsack: " + knapsack(W,
wt, val, n));
    }
}
```

11. Write a program that implements knapsack using greedy.

```
import java.util.Arrays;
import java.util.Comparator;
public class KnapsackGreedy {
    static class Item {
        int value;
        int weight;

        public Item(int value, int weight) {
            this.value = value;
            this.weight = weight;
        }
    }

    static double knapsack(int W, Item[] items) {
        Arrays.sort(items, Comparator.comparingDouble((Item item) -> (double) item.value /
item.weight).reversed());

        double totalValue = 0;
        int remainingCapacity = W;

        for (Item item : items) {
            if (remainingCapacity >= item.weight) {
                totalValue += item.value;
                remainingCapacity -= item.weight;
            } else {
                totalValue += item.value * ((double) remainingCapacity / item.weight);
                break;
            }
        }
        return totalValue;
    }

    public static void main(String[] args) {
        int W = 50;
        Item[] items = {new Item(60, 10), new Item(100, 20), new Item(120, 30)};
        double maxValue = knapsack(W, items);
        System.out.println("Maximum value that can be put in knapsack: " + maxValue);
    }
}
```

12. Write a program to implement file compression (and un-compression) using Huffman's algorithm.

```
import java.util.*;
public class HuffmanCompression {
  static class HuffmanNode {
    char character;
    int frequency;
    HuffmanNode left, right;

    HuffmanNode(char character, int frequency) {
      this.character = character;
      this.frequency = frequency;
    }
  }

  static class HuffmanComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode x, HuffmanNode y) {
      return x.frequency - y.frequency;
    }
  }

  public static Map<Character, String> buildHuffmanCodes(String input) {
    Map<Character, Integer> frequencyMap = new HashMap<>();
    for (char c : input.toCharArray()) {
      frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
    }

    PriorityQueue<HuffmanNode> pq = new PriorityQueue<>(new
HuffmanComparator());
    for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
      pq.add(new HuffmanNode(entry.getKey(), entry.getValue()));
    }

    while (pq.size() > 1) {
      HuffmanNode left = pq.poll();
      HuffmanNode right = pq.poll();
      HuffmanNode parent = new HuffmanNode('\0', left.frequency + right.frequency);
      parent.left = left;
      parent.right = right;
      pq.offer(parent);
    }

    Map<Character, String> codes = new HashMap<>();
    buildCodes(pq.peek(), "", codes);
    return codes;
  }
```

```
    private static void buildCodes(HuffmanNode node, String code, Map<Character,
String> codes) {
        if (node == null) return;
        if (node.left == null && node.right == null) {
            codes.put(node.character, code);
        }
        buildCodes(node.left, code + "0", codes);
        buildCodes(node.right, code + "1", codes);
    }

    public static String compress(String input) {
        Map<Character, String> codes = buildHuffmanCodes(input);
        StringBuilder compressed = new StringBuilder();
        for (char c : input.toCharArray()) {
            compressed.append(codes.get(c));
        }
        return compressed.toString();
    }

    public static String decompress(String compressed, Map<Character, String> codes) {
        StringBuilder decompressed = new StringBuilder();
        StringBuilder currentCode = new StringBuilder();
        for (char bit : compressed.toCharArray()) {
            currentCode.append(bit);
            for (Map.Entry<Character, String> entry : codes.entrySet()) {
                if (entry.getValue().equals(currentCode.toString())) {
                    decompressed.append(entry.getKey());
                    currentCode = new StringBuilder();
                    break;
                }
            }
        }
        return decompressed.toString();
    }

    public static void main(String[] args) {
        String input = "My name is Parigyan.";

        String compressed = compress(input);
        System.out.println("Compressed: " + compressed);

        Map<Character, String> codes = buildHuffmanCodes(input);
        String decompressed = decompress(compressed, codes);
        System.out.println("Decompressed: " + decompressed);
    }
}
```

13. Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
import java.util.*;
class Edge implements Comparable<Edge> {
  int src, dest, weight;

  public Edge(int src, int dest, int weight) {
    this.src = src;
    this.dest = dest;
    this.weight = weight;
  }

  @Override
  public int compareTo(Edge other) {
    return this.weight - other.weight;
  }
}

class DisjointSet {
  int[] parent, rank;
  public DisjointSet(int size) {
    parent = new int[size];
    rank = new int[size];
    for (int i = 0; i < size; i++) {
      parent[i] = i;
      rank[i] = 0;
    }
  }

  public int find(int x) {
    if (parent[x] != x) {
      parent[x] = find(parent[x]);
    }
    return parent[x];
  }

  public void union(int x, int y) {
    int rootX = find(x);
    int rootY = find(y);
    if (rootX == rootY) return;
    if (rank[rootX] < rank[rootY]) {
      parent[rootX] = rootY;
    } else if (rank[rootX] > rank[rootY]) {
      parent[rootY] = rootX;
    } else {
      parent[rootY] = rootX;
      rank[rootX]++;
```

```
        }
    }
}

public class KruskalsAlgorithm {
    public static List<Edge> kruskalMST(List<Edge> edges, int V) {
        List<Edge> result = new ArrayList<>();
        Collections.sort(edges);

        DisjointSet ds = new DisjointSet(V);

        for (Edge edge : edges) {
            int srcParent = ds.find(edge.src);
            int destParent = ds.find(edge.dest);
            if (srcParent != destParent) {
                result.add(edge);
                ds.union(srcParent, destParent);
            }
        }
        return result;
    }

    public static void main(String[] args) {
        int V = 4;
        List<Edge> edges = new ArrayList<>();
        edges.add(new Edge(0, 1, 10));
        edges.add(new Edge(0, 2, 6));
        edges.add(new Edge(0, 3, 5));
        edges.add(new Edge(1, 3, 15));
        edges.add(new Edge(2, 3, 4));

        List<Edge> mst = kruskalMST(edges, V);
        System.out.println("Edges in the Minimum Spanning Tree:");
        for (Edge edge : mst) {
            System.out.println(edge.src + " - " + edge.dest + " : " + edge.weight);
        }
    }
}
```

14. Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
import java.util.*;
class Edge {
    int src, dest, weight;
    public Edge(int src, int dest, int weight) {
        this.src = src;
```

```
        this.dest = dest;
        this.weight = weight;
    }
}

public class PrimsAlgorithm {
    public static void primMST(List<List<Edge>> graph, int V) {
        boolean[] inMST = new boolean[V];
        int[] parent = new int[V];
        int[] key = new int[V];

        Arrays.fill(key, Integer.MAX_VALUE);
        Arrays.fill(parent, -1);

        key[0] = 0;

        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, inMST, V);
            inMST[u] = true;

            for (Edge edge : graph.get(u)) {
                int v = edge.dest;
                int weight = edge.weight;
                if (!inMST[v] && weight < key[v]) {
                    parent[v] = u;
                    key[v] = weight;
                }
            }
        }
        printMST(parent, V, graph);
    }

    private static int minKey(int[] key, boolean[] inMST, int V) {
        int min = Integer.MAX_VALUE, minIndex = -1;
        for (int v = 0; v < V; v++) {
            if (!inMST[v] && key[v] < min) {
                min = key[v];
                minIndex = v;
            }
        }
        return minIndex;
    }

    private static void printMST(int[] parent, int V, List<List<Edge>> graph) {
        System.out.println("Edges in the Minimum Spanning Tree:");
        for (int i = 1; i < V; i++) {
            System.out.println(parent[i] + " - " + i + " : " + graph.get(i).get(parent[i]).weight);
        }
```

```
    }

    public static void main(String[] args) {
        int V = 5;

        List<List<Edge>> graph = new ArrayList<>();
        for (int i = 0; i < V; i++) {
            graph.add(new ArrayList<>());
        }

        addEdge(graph, 0, 1, 2);
        addEdge(graph, 0, 3, 6);
        addEdge(graph, 1, 2, 3);
        addEdge(graph, 1, 3, 8);
        addEdge(graph, 1, 4, 5);
        addEdge(graph, 2, 4, 7);
        addEdge(graph, 3, 4, 9);

        primMST(graph, V);
    }

    private static void addEdge(List<List<Edge>> graph, int src, int dest, int weight) {
        graph.get(src).add(new Edge(src, dest, weight));
        graph.get(dest).add(new Edge(dest, src, weight));
    }
}
```

15. Write a program to implements Dijkstra's algorithm.

```
import java.util.*;
public class DijkstrasAlgorithm {
    static class Node implements Comparable<Node> {
        int vertex;
        int distance;

        public Node(int vertex, int distance) {
            this.vertex = vertex;
            this.distance = distance;
        }

        @Override
        public int compareTo(Node other) {
            return this.distance - other.distance;
        }
    }

    public static void dijkstra(List<List<Node>> graph, int source) {
```

```
        int V = graph.size();
        boolean[] visited = new boolean[V];
        int[] distance = new int[V];
        Arrays.fill(distance, Integer.MAX_VALUE);
        distance[source] = 0;

        PriorityQueue<Node> pq = new PriorityQueue<>();
        pq.offer(new Node(source, 0));

        while (!pq.isEmpty()) {
            Node node = pq.poll();
            int u = node.vertex;

            if (visited[u]) continue;
            visited[u] = true;

            for (Node neighbor : graph.get(u)) {
                int v = neighbor.vertex;
                int weight = neighbor.distance;

                if (!visited[v] && distance[u] != Integer.MAX_VALUE && distance[u] + weight <
distance[v]) {
                    distance[v] = distance[u] + weight;
                    pq.offer(new Node(v, distance[v]));
                }
            }
        }
        printSolution(distance, V, source);
    }

    public static void printSolution(int[] distance, int V, int source) {
        System.out.println("Shortest distances from source vertex " + source + " to all other
vertices:");
        for (int i = 0; i < V; i++) {
            System.out.println("Vertex " + i + ": Distance = " + distance[i]);
        }
    }

    public static void main(String[] args) {
        int V = 5;
        int source = 0;

        List<List<Node>> graph = new ArrayList<>();
        for (int i = 0; i < V; i++) {
            graph.add(new ArrayList<>());
        }

        addEdge(graph, 0, 1, 2);
```

```
        addEdge(graph, 0, 3, 1);
        addEdge(graph, 1, 2, 4);
        addEdge(graph, 1, 3, 3);
        addEdge(graph, 1, 4, 7);
        addEdge(graph, 2, 4, 1);
        addEdge(graph, 3, 4, 5);

        dijkstra(graph, source);
    }
    public static void addEdge(List<List<Node>> graph, int src, int dest, int weight) {
        graph.get(src).add(new Node(dest, weight));
        graph.get(dest).add(new Node(src, weight));
    }
}
```

16. Write a program to implement All-Pairs Shortest Paths Problem using Floyd's algorithm.

```
public class FloydsAlgorithm {
    static final int INF = 99999;

    public static void floydWarshall(int[][] graph, int V) {
        int[][] dist = new int[V][V];
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                dist[i][j] = graph[i][j];
            }
        }
        for (int k = 0; k < V; k++) {
            for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                    if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
        printSolution(dist, V);
    }

    public static void printSolution(int[][] dist, int V) {
        System.out.println("Shortest distances between all pairs of vertices:");
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                if (dist[i][j] == INF) {
                    System.out.print("INF\t");
                } else {
                    System.out.print(dist[i][j] + "\t");
```

```java
            }
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    int V = 4;
    int[][] graph = {
        {0, 5, INF, 10},
        {INF, 0, 3, INF},
        {INF, INF, 0, 1},
        {INF, INF, INF, 0}
    };
    floydWarshall(graph, V);
}
}
```

17. Find a subset of a given set S = {s1, s2 ,..,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1, 8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.

```java
import java.util.*;
public class SubsetSum {
    public static void findSubset(int[] set, int target) {
        List<Integer> subset = new ArrayList<>();
        if (findSubsetHelper(set, target, subset, 0)) {
            System.out.println("Subset with sum " + target + " found: " + subset);
        } else {
            System.out.println("No subset found with sum " + target);
        }
    }

    private static boolean findSubsetHelper(int[] set, int target, List<Integer> subset, int
index) {
        if (target == 0) {
            return true;
        }
        if (index == set.length) {
            return false;
        }
        if (set[index] <= target) {
            subset.add(set[index]);
            if (findSubsetHelper(set, target - set[index], subset, index + 1)) {
                return true;
            }
```

```
            subset.remove(subset.size() - 1);
        }
        return findSubsetHelper(set, target, subset, index + 1);
    }

    public static void main(String[] args) {
        int[] set = {1, 2, 5, 6, 8};
        int target = 9;
        findSubset(set, target);
    }
}
```

18. Implement N Queen's problem using back tracking.

```
import java.util.*;
public class NQueens {
    public static List<List<String>> solveNQueens(int n) {
        List<List<String>> result = new ArrayList<>();
        char[][] board = new char[n][n];
        for (char[] row : board) {
            Arrays.fill(row, '.');
        }
        solveNQueensHelper(board, 0, result);
        return result;
    }

    private static void solveNQueensHelper(char[][] board, int col, List<List<String>>
result) {
        if (col == board.length) {
            result.add(constructSolution(board));
            return;
        }

        for (int row = 0; row < board.length; row++) {
            if (isValid(board, row, col)) {
                board[row][col] = 'Q';
                solveNQueensHelper(board, col + 1, result);
                board[row][col] = '.';
            }
        }
    }

    private static boolean isValid(char[][] board, int row, int col) {
        for (int i = 0; i < col; i++) {
            if (board[row][i] == 'Q') {
                return false;
            }
```

```
        }
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }
        for (int i = row, j = col; i < board.length && j >= 0; i++, j--) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }
        return true;
    }

    private static List<String> constructSolution(char[][] board) {
        List<String> solution = new ArrayList<>();
        for (char[] row : board) {
            solution.add(String.valueOf(row));
        }
        return solution;
    }

    public static void main(String[] args) {
        int n = 4;
        List<List<String>> solutions = solveNQueens(n);
        for (List<String> solution : solutions) {
            for (String row : solution) {
                System.out.println(row);
            }
            System.out.println();
        }
    }
}
```

19. Write a program to implement Graph Colouring using backtracking method.

```
import java.util.*;
public class GraphColoring {
    public static boolean isSafe(int[][] graph, int[] colors, int vertex, int color) {
        for (int i = 0; i < graph.length; i++) {
            if (graph[vertex][i] == 1 && colors[i] == color) {
                return false;
            }
        }
        return true;
    }
```

```
  public static boolean graphColoringUtil(int[][] graph, int numColors, int[] colors, int
vertex) {
    if (vertex == graph.length) {
      return true;
    }

for (int color = 1; color <= numColors; color++) {
      if (isSafe(graph, colors, vertex, color)) {
        colors[vertex] = color;
        if (graphColoringUtil(graph, numColors, colors, vertex + 1)) {
          return true;
        }
        colors[vertex] = 0;
      }
    }
    return false;
  }

  public static void graphColoring(int[][] graph, int numColors) {
    int[] colors = new int[graph.length];
    if (graphColoringUtil(graph, numColors, colors, 0)) {
      System.out.println("Graph coloring possible with " + numColors + " colors:");
      for (int i = 0; i < graph.length; i++) {
        System.out.println("Vertex " + i + ": Color " + colors[i]);
      }
    } else {
      System.out.println("Graph coloring not possible with " + numColors + " colors.");
    }
  }

  public static void main(String[] args) {
    int[][] graph = {
      {0, 1, 1, 1},
      {1, 0, 1, 0},
      {1, 1, 0, 1},
      {1, 0, 1, 0}
    };
    int numColors = 3;
    graphColoring(graph, numColors);
  }
}
```

20. Write a program to implement Travelling sales person using branch and bound.

```
import java.util.*;
public class TSPBranchAndBound {
  static class Node implements Comparable<Node> {
```

```
        int vertex;
        int level;
        int cost;
        ArrayList<Integer> path;

        public Node(int vertex, int level, int cost, ArrayList<Integer> path) {
            this.vertex = vertex;
            this.level = level;
            this.cost = cost;
            this.path = new ArrayList<>(path);
            this.path.add(vertex);
        }

        @Override
        public int compareTo(Node other) {
            return this.cost - other.cost;
        }
    }

    static int tsp(int[][] graph) {
        int n = graph.length;
        PriorityQueue<Node> pq = new PriorityQueue<>();
        ArrayList<Integer> path = new ArrayList<>();
        int minCost = Integer.MAX_VALUE;

        Node root = new Node(0, 0, 0, path);
        root.path.add(0);
        pq.add(root);

        while (!pq.isEmpty()) {
            Node current = pq.poll();
            if (current.level == n - 1) {
                current.path.add(0);
                if (current.cost < minCost) {
                    minCost = current.cost;
                    path = current.path;
                }
                continue;
            }

            for (int i = 0; i < n; i++) {
                if (!current.path.contains(i) && graph[current.vertex][i] != 0) {
                    Node child = new Node(i, current.level + 1, current.cost +
graph[current.vertex][i], current.path);
                    pq.add(child);
                }
            }
        }
```

```
        System.out.println("Optimal Path: " + path);
        return minCost;
    }

    public static void main(String[] args) {
        int[][] graph = {
            {0, 10, 15, 20},
            {10, 0, 35, 25},
            {15, 35, 0, 30},
            {20, 25, 30, 0}
        };
        int minCost = tsp(graph);
        System.out.println("Minimum cost: " + minCost);
    }
}
```

21. Write a program to implement Travelling sales person using dynamic programming.

```
import java.util.*;
public class TSPDynamicProgramming {
    static final int INF = Integer.MAX_VALUE;

    public static int tsp(int[][] graph) {
        int n = graph.length;
        int[][] dp = new int[n][1 << n];
        for (int[] row : dp) {
            Arrays.fill(row, INF);
        }
        dp[0][1] = 0;
        for (int mask = 1; mask < (1 << n); mask++) {
            for (int u = 0; u < n; u++) {
                if ((mask & (1 << u)) != 0) {
                    for (int v = 0; v < n; v++) {
                        if (u != v && (mask & (1 << v)) != 0) {
                            dp[v][mask] = Math.min(dp[v][mask], dp[u][mask ^ (1 << v)] + graph[u][v]);
                        }
                    }
                }
            }
        }

        int minCost = INF;
        for (int v = 1; v < n; v++) {
            minCost = Math.min(minCost, dp[v][(1 << n) - 1] + graph[v][0]);
        }
```

```
        return minCost;
    }

    public static void main(String[] args) {
        int[][] graph = {
            {0, 10, 15, 20},
            {10, 0, 35, 25},
            {15, 35, 0, 30},
            {20, 25, 30, 0}
        };
        int minCost = tsp(graph);
        System.out.println("Minimum cost: " + minCost);
    }
}
```

22. Write a program to implement the backtracking algorithm for the Hamiltonian Circuits problem.

```
import java.util.*;
public class HamiltonianCircuit {
    public static boolean isSafe(int v, int[][] graph, List<Integer> path, int pos) {
        if (graph[path.get(pos - 1)][v] == 0) {
            return false;
        }
        for (int i = 0; i < pos; i++) {
            if (path.get(i) == v) {
                return false;
            }
        }
        return true;
    }

    public static boolean hamiltonianCircuitUtil(int[][] graph, List<Integer> path, int pos) {
        int n = graph.length;
        if (pos == n) {
            if (graph[path.get(pos - 1)][path.get(0)] == 1) {
                return true;
            } else {
                return false;
            }
        }
        for (int v = 1; v < n; v++) {
            if (isSafe(v, graph, path, pos)) {
                path.add(v);
                if (hamiltonianCircuitUtil(graph, path, pos + 1)) {
                    return true;
                }
```

```
            path.remove(pos);
          }
        }
        return false;
      }

      public static boolean hamiltonianCircuit(int[][] graph) {
        int n = graph.length;
        List<Integer> path = new ArrayList<>();
        path.add(0);

        if (hamiltonianCircuitUtil(graph, path, 1)) {
          System.out.println("Hamiltonian Circuit found: " + path);
          return true;
        }
        else {
          System.out.println("No Hamiltonian Circuit exists.");
          return false;
        }
      }

      public static void main(String[] args) {
        int[][] graph = {
            {0, 1, 0, 1, 0},
            {1, 0, 1, 1, 1},
            {0, 1, 0, 0, 1},
            {1, 1, 0, 0, 1},
            {0, 1, 1, 1, 0}
        };
        hamiltonianCircuit(graph);
      }
    }
```

23. Write a program to implement greedy algorithm for job sequencing with deadlines.

```
import java.util.*;
public class JobSequencing {
  static class Job {
    char id;
    int deadline;
    int profit;

    public Job(char id, int deadline, int profit) {
      this.id = id;
      this.deadline = deadline;
      this.profit = profit;
    }
```

```
    }

    public static List<Character> jobSequence(Job[] jobs) {
        Arrays.sort(jobs, (a, b) -> b.profit - a.profit);
        int maxDeadline = 0;
        for (Job job : jobs) {
            maxDeadline = Math.max(maxDeadline, job.deadline);
        }

        boolean[] slots = new boolean[maxDeadline + 1];
        Arrays.fill(slots, false);

        List<Character> sequence = new ArrayList<>();
        int totalProfit = 0;

        for (Job job : jobs) {
            for (int i = Math.min(maxDeadline, job.deadline); i > 0; i--) {
                if (!slots[i]) {
                    slots[i] = true;
                    sequence.add(job.id);
                    totalProfit += job.profit;
                    break;
                }
            }
        }
        System.out.println("Total Profit: " + totalProfit);
        return sequence;
    }

    public static void main(String[] args) {
        Job[] jobs = {
                new Job('a', 2, 100),
                new Job('b', 1, 19),
                new Job('c', 2, 27),
                new Job('d', 1, 25),
                new Job('e', 3, 15)
        };

        List<Character> sequence = jobSequence(jobs);
        System.out.println("Job Sequence: " + sequence);
    }
}
```