LAB ASSIGNMENT

5

21BCE9905

K. Madhu Sudhana Rao

1. Write a program to implement brute-force approach to solve Travelling Salesman problem

```
import java.util.*)
Public class TSPBruteforce {
   private static final int INF = Integer.MAX_VALUE;

  Public static void main(String[] args) {
     int[] [] distances = {{0, 10, 15, 20},
                          {10 0 35 25},
                          {15, 35, 0, 30},
                          {20, 25, 30, 0}};
  System.out.println("Thee Shortest distance is "+
                    tsp_bruteforce(distances));
  }
 private static int tsp_bruteforce( int[] [] distances) {
  int n = distances.length;
  int minDistance = INF;
 List<List<Integer>> tours = new ArrayList<>();
  ~~tour.add~~().
 for (int i=0; i<n; i++){
   List<integer> tour = new ArrayList<>();
   tour. add (i)
```

```
    generateTours (distances, i, currLength
                                     1, n, tour);

    if ( ! tour. is Empty()){.
            tours.add(tour);

    }

}
for (List <Integer> tour : tours) {.
    int dist = calculate Total Distance (distances, tour);

    if (dist < minDistance){.
          minDistance = dist;

    }
}
return min Distance;

}
Private static  void generateTours (int[][] distances, int
        currlength, int n, List<Integer> currentTour){.

  if (currLength = = n){.
     current Tour. add (startIndex);
     return;

  }
  for (int i=0; i<n; i++){.
   if (distances[startIndex][ i ] != 0 && !currentTour.contain (i)){
      currentTour.add(i);
   generateTour( distances, i, currLength+1, n, currentTour);
   currentTour.remove (currentTour. size() -1);
   }
  }
}
```

```
private static int calculateTotalDistance(int[][] distances,
                    List <Integer> tour){

    int result = 0;

    for(int i=0; i< tour.size(); i++){

        int j = (i+1) % tour.size();

        result += distances[tour.get(i)][tour.get(j)];

    }
    return result;
    }
}
```

output:

Thee shortest distance is 20

2. Write a program to implement brute force approach to solve 0/1 knapsack problem.

```
class Knapsack {
    Static int max(int a, int b) {
        return (a > b) ? a : b;
    }
    Static int Knapsack(int W, int[] wt, int val[], int n, int[][]
                                                      memo) {
        if (n==0 || w==0)
            return 0;
        if (memo[n][w] != -1)
            return memo[n][w];
        if (wt[n-1] > w)
            memo[n][w] = Knapsack(w, wt, val, n-1, memo);
        else
        memo[n][w] = max(val[n-1] + knapSack(w-wt[n-1], wt,
                val, n-1, memo), Knapsack(w, wt, val, n-1, memo));
        return memo[n][w];
    }
    public static void main(String args[]) {
        int val[] = new int[]{60, 100, 120};
        int wt[] = new int[]{10, 20, 30};
        int w = 50;
        int n = val.length;
        int[][] memo = new int[n+1][w+1];
        for (int i=0; i<=n; i++) {
            for (int j=0; j<=w; j++) {
                memo[i][j] = -1;
            }
        }
        System.out.println("maximum value that can be put is" +
                Knapsack(W, wt, val, n, memo));
    }
}
```

Output :- maximum value that can be Put is 220

3. Write a program to implement brute-force approach to solve Assignment problem.

```
import jav. util.ArrayList;
import java. util. Arrays;
Public  class Assignment Problem1.
    Public Static int calculateTotalCost(int[] assignment,
                         int[][] costMatrix){.
   int totalCost =0;
   for(int i=0; i<assignment.length; i++){
      totalCost += costMatrix[i][assignment[i]];
   }
   return totalCost;
   }
 Public Static int[] bruteForceAssignment (int[][] costMatri
   int n = costMatrix.length;
   int[] Assignment = new int[n];
  for (int i=0; i<n; i++){.
      assignment[i] =i;
   }.
 int [] optimalAssignment = Arrays.Copyof(assignment,n);
 int  minCost =Integer.MAX_VALUE;
  do{
    int totalcost = calculateTotalcost(assignment, costMatrix);
    if (totalcost< min cost){.
       min cost = total cost;
      optimalAssignment = Arrays.copyof(assignment,n);
    }
  } while (next Permutation (assignment));
   return optimal Assignment;
}
```

```
Public Static boolean nextPermutation(int[] array) {
    int i = array.length - 1;
   while (i>0 && array[i-1] >= array[i]){
        i--;
    }
   if (i<=0){
        return false;
    }
   int j = array.length-1;
   while (array[j] <= array[i-1]){
        j--;
    }
  int temp = array[i-1];
  array[i-1] = array[j];
  array[j] = temp;
  j = array.length-1:
  while (i<j){
      temp = array[i];
      array[i] = array[j];
      array[j] = temp;
      i++;
      j++;   return true;
Public Static void main(String[] args){
   int[][] costMatrix = {{3, 2, 7},
                         {2, 4, 6};
                         {1, 3, 5} };
  int[] optimalAssignment = bruteForce(Assignment(costMatrix);
 System.out.println("optimal Assignment" +
                Arrays.toString(optimalAssignment));
  }
}
```

output:-

```
Optimal Assignment: [1, 0, 2]
```