DAYANANDA SAGAR ACADEMY OF TECHNOLOGY & MANAGEMENT
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082
(Affiliated to Visvesvaraya Technological University, Belagavi and Approved by AICTE, New Delhi)
(CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi Accredited by NAAC,A+)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Academic Year: 2022-2023

DESIGN & ANALYSIS OF ALGORITHMS LABORATORY(21CS42)

DESIGN AND ANALYSIS OF ALGORITHM LABORATORY(21CS42) [As per Choice Based Credit System (CBCS) scheme]	
Course objectives: This course will enable students to • Design and implement various algorithms in JAVA. • Employ various design strategies for problem solving. • Measure and compare the performance of different algorithms.	
1	Sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case.
2	1. Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.
	2. Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.
3	1. To solve Knapsack problem using Greedy method.
	2. To find shortest paths to other vertices from a given vertex in a weighted connected graph, using Dijkstra's algorithm.
	3. To find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.
	4. To find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
4	1. Solve All-Pairs Shortest Paths problem using Floyd's algorithm.
	2. Solve Travelling Sales Person problem using Dynamic programming.
	3.Solve 0/1 Knapsack problem using Dynamic Programming method
5	1. Design and implement C++/Java Program to find a subset of a given set S = {Sl, 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}. Display a suitable message, if the given problem instance doesn't have a solution.
	2. Design and implement C++/Java Program to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.

1. **Sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case.**

```
package sort;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.Random;
public class Selecsort {

                    public static void selectionSort(int[] arr)
{
   int n = arr.length;
   for (int i = 0; i < n - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex])
            minIndex = j;
      }
      int temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
   }
}
public static int[] generateRandomNumbers(int n) {
   int[] nums = new int[n];
   Random rand = new Random();
   for (int i = 0; i < n; i++) {
    nums[i] = rand.nextInt(10000); // Generates numbers between 0 and 9999
   }
   return nums;
}
public static void main(String[] args) {
   int[] nValues = {5000, 10000, 15000, 20000, 25000};
   long[] timeTaken = new long[nValues.length];

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

```
                int n = nValues[i];
                int[] arr = generateRandomNumbers(n);

                long startTime = System.nanoTime();
                selectionSort(arr);
                long endTime = System.nanoTime();
long duration = (endTime - startTime) / 1000000; // Convert to milliseconds timeTaken[i] =
duration;
 System.out.println("Time taken to sort " + n + " elements: " + duration + " milliseconds");
  }
```

OUTPUT: Enter the elements

Time taken to sort 5000 elements in 2.5 milliseconds

Time taken to sort 10000 elements in 4.5 milliseconds

Time taken to sort 15000 elements in 7.5 milliseconds

Time taken to sort 20000 elements in 8.5 milliseconds

Time taken to sort 25000 elements in 11.5 milliseconds

2. **Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n > 5000 and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case**.

```
import java.util.Scanner;
import java.util.Arrays;
import java.util.Random;

public class QuickSortComplexity {
  static final int MAX = 200000;
  static int[] a = new int[MAX];
  public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.print("Enter Max array size: ");
        int n = input.nextInt();
        Random random = new Random();
        System.out.println("Enter the array elements: ");
        for (int i = 0; i < n; i++)
              // a[i] = input.nextInt(); // for keyboard entry
              a[i] = random.nextInt(10000); // generate
    // random numbers ñ uniform distribution

        // a = Arrays.copyOf(a, n); // keep only non zero elements
        // Arrays.sort(a); // for worst-case time complexity
```

```
        System.out.println("Input Array:");
        for (int i = 0; i < n; i++)
                System.out.print(a[i] + " ");
        // set start time
        long startTime = System.nanoTime();
        QuickSortAlgorithm(0, n - 1);
        long stopTime = System.nanoTime();
        long elapsedTime = stopTime - startTime;
        System.out.println("\nSorted Array:");
        for (int i = 0; i < n; i++)
                System.out.print(a[i] + " ");
        System.out.println();
        System.out.println("Time Complexity in ms for
            n=" + n + " is: " + (double) elapsedTime / 1000000);
}

public static void QuickSortAlgorithm(int p, int r) {
        int i, j, temp, pivot;
        if (p < r) {
                i = p;
                j = r + 1;
                pivot = a[p]; // mark first element as pivot
                while (true) {
                        i++;
                        while (a[i] < pivot && i < r)
                                i++;
                        j--;
                        while (a[j] > pivot)
                                j--;
                        if (i < j) {
                                temp = a[i];
                                a[i] = a[j];
                                a[j] = temp;
                        } else
                                break; // partition is over
                }
```

```
                    a[p] = a[j];
                    a[j] = pivot;
                    QuickSortAlgorithm(p, j - 1);
                    QuickSortAlgorithm(j + 1, r);
                }
            }
        }
```

Output

```
Enter Max array size: 20
Enter the array elements:
Input Array:
326 719 983 701 490 230 595 474 341 75 916 173 324 852 728 434 758 445 303 566
Sorted Array:
75 173 230 303 324 326 341 434 445 474 490 566 595 701 719 728 758 852 916 983
Time Complexity in ms for n=20 is: 0.023225

Enter Max array size: 20000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=20000 is: 4.953809

Enter Max array size: 30000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=30000 is: 7.141865

Enter Max array size: 40000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=40000 is: 8.698231

Enter Max array size: 50000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=50000 is: 9.103897

Enter Max array size: 60000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=60000 is: 12.380137

Enter Max array size: 70000
Enter the array elements:
Input Array:
Sorted Array:
```

Time Complexity in ms for n=70000 is: 24.719828

Enter Max array size: 80000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=80000 is: 21.150887

Enter Max array size: 90000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=90000 is: 35.894418

Enter Max array size: 100000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=100000 is: 31.430762

Enter Max array size: 200000
Enter the array elements:
Input Array:
Sorted Array:
Time Complexity in ms for n=200000 is: 47.498161

Plot Graph: time taken versus n on graph sheet

Time Complexity Analysis:

Quick Sort Algorithm
Average performance O(n log n)

3. Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n > 5000, and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.util.Random;
import java.util.Scanner;

public class MergeSort{
        static final int MAX = 200000;
        static int[] a = new int[MAX];

        public static void main(String[] args) {
                Scanner input = new Scanner(System.in);
                System.out.print("Enter Max array size: ");
                int n = input.nextInt();
                Random random = new Random();
                System.out.println("Enter the array elements: ");
                for (int i = 0; i < n; i++)
                {
                //        a[i] = input.nextInt(); // for keyboard entry
                 a[i] = random.nextInt(100000);
              System.out.print(a[i] + " ");
                }
                long startTime = System.nanoTime();
                MergeSortAlgorithm(0, n - 1);
                long stopTime = System.nanoTime();
                long elapsedTime = stopTime - startTime;
                System.out.println("Time Complexity (ms) for n = " +
n + " is : " + (double) elapsedTime / 1000000);
                System.out.println("Sorted Array (Merge Sort):");
                for (int i = 0; i < n; i++)
                        System.out.print(a[i] + " ");
                input.close();
        }

        public static void MergeSortAlgorithm(int low, int high) {
                int mid;
                if (low < high) {
                        mid = (low + high) / 2;
                        MergeSortAlgorithm(low, mid);
                        MergeSortAlgorithm(mid + 1, high);
                        Merge(low, mid, high);
                }
        }
```

```
        public static void Merge(int low, int mid, int high) {
                int[] b = new int[MAX];
                int i, h, j, k;
                h = i = low;
                j = mid + 1;
                while ((h <= mid) && (j <= high))
                        if (a[h] < a[j])
                                b[i++] = a[h++];
                        else
                                b[i++] = a[j++];

                if (h > mid)
                        for (k = j; k <= high; k++)
                                b[i++] = a[k];
                else
                        for (k = h; k <= mid; k++)
                                b[i++] = a[k];

                for (k = low; k <= high; k++)
                        a[k] = b[k];
        }
}
```

Output

```
Enter Max array size: 5
Enter the array elements:
856 604 528 287 321 Time Complexity (ms) for n = 5 is : 0.090071
Sorted Array (Merge Sort):
287 321 528 604 856

Enter Max array size: 10000
Enter the array elements:
Time Complexity (ms) for n = 10000 is : 1194.135767
Sorted Array (Merge Sort):

Enter Max array size: 20000
Enter the array elements:
Time Complexity (ms) for n = 20000 is : 2040.96632
Sorted Array (Merge Sort):

Enter Max array size: 30000
Enter the array elements:
Time Complexity (ms) for n = 30000 is : 3098.642188
Sorted Array (Merge Sort):

Enter Max array size: 40000
Enter the array elements:
Time Complexity (ms) for n = 40000 is : 3914.650313
Sorted Array (Merge Sort):
```

Enter Max array size: 50000
Enter the array elements:
Time Complexity (ms) for n = 50000 is : 4700.729745
Sorted Array (Merge Sort):

Enter Max array size: 60000
Enter the array elements:
Time Complexity (ms) for n = 60000 is : 5457.318457
Sorted Array (Merge Sort):

Enter Max array size: 70000
Enter the array elements:
Time Complexity (ms) for n = 70000 is : 6630.648568
Sorted Array (Merge Sort):

Enter Max array size: 80000
Enter the array elements:
Time Complexity (ms) for n = 80000 is : 7419.150889
Sorted Array (Merge Sort):

Enter Max array size: 90000
Enter the array elements:
Time Complexity (ms) for n = 90000 is : 8119.913672
Sorted Array (Merge Sort):

Enter Max array size: 100000
Enter the array elements:
Time Complexity (ms) for n = 100000 is : 8865.6302
Sorted Array (Merge Sort):

Plot Graph: time taken versus n on graph sheet

Time Complexity Analysis:

Merge Sort Algorithm

Average performance O(n log n)

4. Implement in Java, the 0/1 Knapsack problem using Greedy method.

```
import java.util.Scanner;
class KObject {                                          // Knapsack object details
        float w;
        float p;
        float r;
}
public class KnapsackGreedy {
        static final int MAX = 20;      // max. no. of objects
        static int n;                           // no. of objects
        static float M;                         // capacity of Knapsack

        public static void main(String args[]) {
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter number of objects:  ");
                n = scanner.nextInt();
                KObject[] obj = new KObject[n];
                for(int i = 0; i<n;i++)
                        obj[i] = new KObject();// allocate memory for members

                ReadObjects(obj);
                Knapsack(obj);
                scanner.close();
        }

        static void ReadObjects(KObject obj[]) {
                KObject temp = new KObject();
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter the max capacity of knapsack: ");
                M = scanner.nextFloat();

                System.out.println("Enter Weights: ");
                for (int i = 0; i < n; i++)
                        obj[i].w = scanner.nextFloat();

                System.out.println("Enter Profits: ");
                for (int i = 0; i < n; i++)
                        obj[i].p = scanner.nextFloat();

                for (int i = 0; i < n; i++)
                        obj[i].r = obj[i].p / obj[i].w;

                // sort objects in descending order, based on p/w ratio
                for(int i = 0; i<n-1; i++)
                        for(int j=0; j<n-1-i; j++)
                                if(obj[j].r < obj[j+1].r){
                                        temp = obj[j];
                                        obj[j] = obj[j+1];
                                        obj[j+1] = temp;
```

```
                }
            scanner.close();
        }

        static void Knapsack(KObject kobj[]) {
            float x[] = new float[MAX];
            float totalprofit;
            int i;
            float U; // U place holder for M
            U = M;
            totalprofit = 0;
            for (i = 0; i < n; i++)
                x[i] = 0;
            for (i = 0; i < n; i++) {
                if (kobj[i].w > U)
                    break;
                else {
                    x[i] = 1;
                    totalprofit = totalprofit + kobj[i].p;
                    U = U - kobj[i].w;
                }
            }
            System.out.println("i = " + i);
            if (i < n)
                x[i] = U / kobj[i].w;
            totalprofit = totalprofit + (x[i] * kobj[i].p);
            System.out.println("The Solution vector, x[]: ");
            for (i = 0; i < n; i++)
                System.out.print(x[i] + " ");
            System.out.println("\nTotal profit is = " + totalprofit);
        }
}
```

Output

```
Enter number of objects:
4
Enter the max capacity of knapsack:
5
Enter Weights: 1 2 2 1
Enter Profits:
15
20
10
12
i = 3
The Solution vector, x[]:
1.0 1.0 1.0 0.5
Total profit is = 52.0
```

5. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in Java.

```
import java.util.*;

public class DijkstrasClass {

        final static int MAX = 20;
        final static int infinity = 9999;
        static int n;               // No. of vertices of G
        static int a[][]; // Cost matrix
        static Scanner scan = new Scanner(System.in);

        public static void main(String[] args) {
                ReadMatrix();
                int s = 0;                  // starting vertex
                System.out.println("Enter starting vertex: ");
                s = scan.nextInt();
                Dijkstras(s);    // find shortest path
        }

        static void ReadMatrix() {
                a = new int[MAX][MAX];
                System.out.println("Enter the number of vertices:");
                n = scan.nextInt();
                System.out.println("Enter the cost adjacency matrix:");
                for (int i = 1; i <= n; i++)
                        for (int j = 1; j <= n; j++)
                                a[i][j] = scan.nextInt();
        }

        static void Dijkstras(int s) {
                int S[] = new int[MAX];
                int d[] = new int[MAX];
                int u, v;
                int i;
                for (i = 1; i <= n; i++) {
                        S[i] = 0;
                        d[i] = a[s][i];
                }
                S[s] = 1;
                d[s] = 1;
                i = 2;
                while (i <= n) {
                        u = Extract_Min(S, d);
                        S[u] = 1;
                        i++;
                        for (v = 1; v <= n; v++) {
                                if (((d[u] + a[u][v] < d[v]) && (S[v] == 0)))
                                        d[v] = d[u] + a[u][v];
```

```
                }
            }
            for (i = 1; i <= n; i++)
                if (i != s)
                    System.out.println(i + ":" + d[i]);
        }

        static int Extract_Min(int S[], int d[]) {
            int i, j = 1, min;
            min = infinity;
            for (i = 1; i <= n; i++) {
                if ((d[i] < min) && (S[i] == 0)) {
                    min = d[i];
                    j = i;
                }
            }
            return (j);
        }
}
```

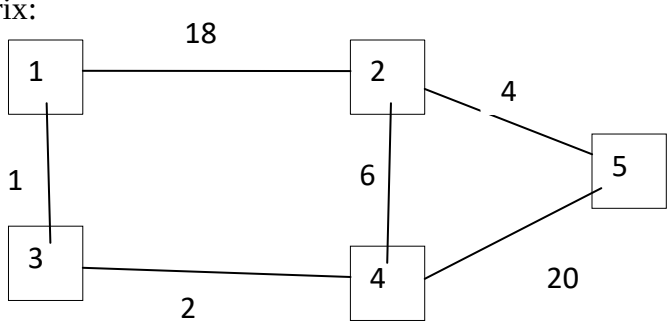
Output

Enter the number of vertices:

5

Enter the cost adjacency matrix:

0 18 1 9999 9999
18 0 9999 6 4
1 9999 0 2 9999
9999 6 2 0 20
9999 4 9999 20 0



Enter starting vertex:

1

2:9

3:1

4:3

5:13

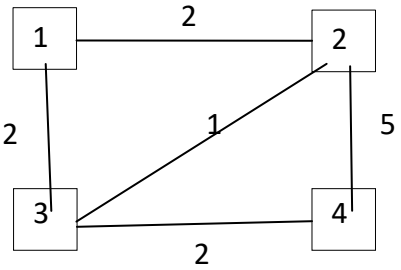
6. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal'salgorithm. Use Union-Find algorithms in your program.

```
import java.util.Scanner;

public class KruskalsClass {

        final static int MAX = 20;
        static int n; // No. of vertices of G
        static int cost[][]; // Cost matrix
        static Scanner scan = new Scanner(System.in);

        public static void main(String[] args) {
                ReadMatrix();
                Kruskals();
        }

        static void ReadMatrix() {

                int i, j;
                cost = new int[MAX][MAX];

                System.out.println("Implementation of Kruskal's algorithm");
                System.out.println("Enter the no. of vertices");
                n = scan.nextInt();

                System.out.println("Enter the cost adjacency matrix");
                for (i = 1; i <= n; i++) {
                        for (j = 1; j <= n; j++) {
                                cost[i][j] = scan.nextInt();
                                if (cost[i][j] == 0)
                                        cost[i][j] = 999;
                        }
                }
        }

        sstatic void Kruskals() {

                int a = 0, b = 0, u = 0, v = 0, i, j, ne = 1, min, mincost = 0;

                System.out.println("The edges of Minimum Cost Spanning Tree are");
```

```
        while (ne < n) {
            for (i = 1, min = 999; i <= n; i++) {
                for (j = 1; j <= n; j++) {
                    if (cost[i][j] < min) {
                        min = cost[i][j];
                        a = u = i;
                        b = v = j;
                    }
                }
            }
            u = find(u);
            v = find(v);
            if (u != v) {
                uni(a, v);
        System.out.println(ne++ + "edge (" + a + "," + b + ") =" + min);
                mincost += min;
            }
            cost[a][b] = cost[b][a] = 999;
        }
        System.out.println("Minimum cost :" + mincost);
}

static int find(int i) {

        while (parent[i] != 0)
            i = parent[i];
        return i;
}

static void uni(int i, int j) {

        parent[j] = i;
}}
```

Output

Enter the number of vertices: 4
Enter the cost adjacency matrix:
0 20 2 999
20 0 15 5
2 15 0 25
999 5 25 0
The edges of Minimum Cost Spanning Tree are
1edge (1,3) =2
2edge (2,4) =5
3edge (2,3) =15
Minimum cost :22



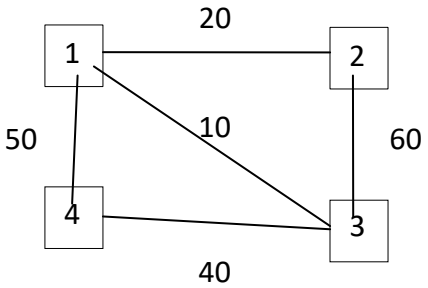
7. Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
import java.util.Scanner;

public class PrimsClass {

        final static int MAX = 20;
        static int n; // No. of vertices of G
        static int cost[][]; // Cost matrix
        static Scanner scan = new Scanner(System.in);

        public static void main(String[] args) {
                ReadMatrix();
                Prims();
        }

        static void ReadMatrix() {
                int i, j;
                cost = new int[MAX][MAX];

                System.out.println("\n Enter the number of nodes:");
                n = scan.nextInt();
                System.out.println("\n Enter the cost matrix:\n");
                for (i = 1; i <= n; i++)
                        for (j = 1; j <= n; j++) {
                                cost[i][j] = scan.nextInt();
                                if (cost[i][j] == 0)
                                        cost[i][j] = 999;
                        }
        }

        static void Prims() {

                int visited[] = new int[10];
                int ne = 1, i, j, min, a = 0, b = 0, u = 0, v = 0;
                int mincost = 0;

                visited[1] = 1;
                while (ne < n) {
                        for (i = 1, min = 999; i <= n; i++)
```

```
                    for (j = 1; j <= n; j++)
                        if (cost[i][j] < min)
                            if (visited[i] != 0) {
                                min = cost[i][j];
                                a = u = i;
                                b = v = j;
                            }
                    if (visited[u] == 0 || visited[v] == 0) {
        System.out.println("Edge" + ne++ + ":(" + a + "," + b + ")" + "cost:" + min);
                        mincost += min;
                        visited[b] = 1;
                    }
                    cost[a][b] = cost[b][a] = 999;
                }
                System.out.println("\n Minimun cost" + mincost);
        }

}
```

Output

```
 Enter the number of nodes: 4
Enter the cost matrix:
 0        20       10       50
20        0        60       999
10        60       0        40
50        999      40       0

Enter Source: 1
1 --> 3 = 10    Sum = 10
1 --> 2 = 20    Sum = 30
3 --> 4 = 40    Sum = 70

Total cost: 70
```



8. Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

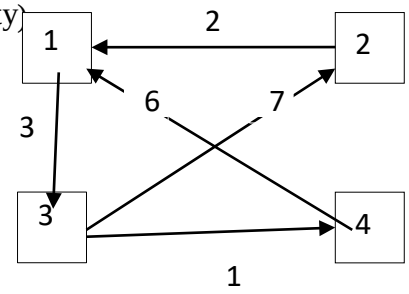
```
import java.util.Scanner;
public class FloydsClass {
static final int MAX = 20;      // max. size of cost matrix
static int a[][];               // cost matrix
static int n;                           // actual matrix size

public static void main(String args[]) {
        a = new int[MAX][MAX];
        ReadMatrix();
        Floyds();                               // find all pairs shortest path
        PrintMatrix();
}

static void ReadMatrix() {
        System.out.println("Enter the number of vertices\n");
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        System.out.println("Enter the Cost Matrix (999 for infinity) \n");
        for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= n; j++) {
                        a[i][j] = scanner.nextInt();
                }
        }
        scanner.close();
}

static void Floyds() {
        for (int k = 1; k <= n; k++) {
                for (int i = 1; i <= n; i++)
                        for (int j = 1; j <= n; j++)
                                if ((a[i][k] + a[k][j]) < a[i][j])
                                        a[i][j] = a[i][k] + a[k][j];
        }
}

static void PrintMatrix() {
        System.out.println("The All Pair Shortest Path Matrix is:\n");
        for(int i=1; i<=n; i++)
        {
                for(int j=1; j<=n; j++)
                        System.out.print(a[i][j] + "\t");
                System.out.println("\n");
        }
        }
}
```

Output

Enter the number of vertices: 4

EEnter the Cost Matrix (999 for intinity)

0	999	3	999
2	0	999	999
999	7	0	1
6	999	999	0



TThe All Pair Shortest Path Matrix is:

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

9. Implement Travelling Sales Person problem using Dynamic programming.

```
import java.util.Scanner;

public class TravSalesPerson {
static int MAX = 100;
static final int infinity = 999;

public static void main(String args[]) {
		int cost = infinity;
	int c[][] = new int[MAX][MAX];	// cost matrix
	int tour[] = new int[MAX];		// optimal tour
	int n;							// max. cities
	System.out.println("Travelling Salesman Problem using Dynamic
Programming\n");
	System.out.println("Enter number of cities: ");
	Scanner scanner = new Scanner(System.in);
	n = scanner.nextInt();
	System.out.println("Enter Cost matrix:\n");
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++) {
			c[i][j] = scanner.nextInt();
			if (c[i][j] == 0)
				c[i][j] = 999;
		}
	for (int i = 0; i < n; i++)
		tour[i] = i;
	cost = tspdp(c, tour, 0, n);
	// print tour cost and tour
	System.out.println("Minimum Tour Cost: " + cost);
	System.out.println("\nTour:");
	for (int i = 0; i < n; i++) {
		System.out.print(tour[i] + " -> ");
	}
	System.out.println(tour[0] + "\n");
	scanner.close();
}

static int tspdp(int c[][], int tour[], int start, int n) {
	int i, j, k;
	int temp[] = new int[MAX];
	int mintour[] = new int[MAX];
	int mincost;
	int cost;
	if (start == n - 2)
		return c[tour[n - 2]][tour[n - 1]] + c[tour[n - 1]][0];
	mincost = infinity;
	for (i = start + 1; i < n; i++) {
		for (j = 0; j < n; j++)
			temp[j] = tour[j];
```

```
                temp[start + 1] = tour[i];
                temp[i] = tour[start + 1];
                if (c[tour[start]][tour[i]] + (cost = tspdp(c, temp, start + 1, n)) <
mincost) {
                        mincost = c[tour[start]][tour[i]] + cost;
                        for (k = 0; k < n; k++)
                                mintour[k] = temp[k];
```

```
                }
        }
        for (i = 0; i < n; i++)
                tour[i] = mintour[i];
        return mincost;
}
}
```

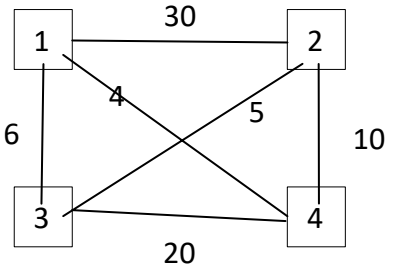
Output

Travelling Salesman Problem using Dynamic Programming

Enter number of cities:

Enter cost matrix:

0	30	6	4
30	0	5	10
6	5	0	20
4	10	20	0



Minimum Tour Cost: 25

Tour:

0 -> 2 -> 1 -> 3 -> 0

10. Solve 0/1 Knapsack problem using Dynamic Programming method.

```
import java.util.Scanner;

public class KnapsackDP {
        static final int MAX = 20; // max. no. of objects
        static int w[];  // weights 0 to n-1
        static int p[];    // profits 0 to n-1
        static int n;              // no. of objects
        static int M;              // capacity of Knapsack
        static int V[][];          // DP solution process - table
        static int Keep[][]; // to get objects in optimal solution

        public static void main(String args[]) {
                w = new int[MAX];
                p = new int[MAX];
                V = new int [MAX][MAX];
                Keep = new int[MAX][MAX];
                int optsoln;
                ReadObjects();
                for (int i = 0; i <= M; i++)
                        V[0][i] = 0;
                for (int i = 0; i <= n; i++)
                        V[i][0] = 0;
                optsoln = Knapsack();
                System.out.println("Optimal solution = " + optsoln);
        }
```

```
	static int Knapsack() {
		int r; // remaining Knapsack capacity
		for (int i = 1; i <= n; i++)
			for (int j = 0; j <= M; j++)
				if ((w[i] <= j) && (p[i] + V[i - 1][j - w[i]] > V[i - 1][j]))
				{
					V[i][j] = p[i] + V[i - 1][j - w[i]];
					Keep[i][j] = 1;
				} else {
					V[i][j] = V[i - 1][j];
					Keep[i][j] = 0;
				}

		// Find the objects included in the Knapsack
		r = M;
		System.out.println("Items = ");
		for (int i = n; i > 0; i--) // start from Keep[n,M]
			if (Keep[i][r] == 1) {
				System.out.println(i + " ");
				r = r - w[i];
			}
		System.out.println();
		return V[n][M];
	}

	static void ReadObjects() {
		Scanner scanner = new Scanner(System.in);
		System.out.println("Knapsack Problem - Dynamic Programming
Solution: ");
		System.out.println("Enter the max capacity of knapsack: ");
		M = scanner.nextInt();
		System.out.println("Enter number of objects:  ");
		n = scanner.nextInt();
		System.out.println("Enter Weights: ");
		for (int i = 1; i <= n; i++)
			w[i] = scanner.nextInt();
		System.out.println("Enter Profits: ");
		for (int i = 1; i <= n; i++)
			p[i] = scanner.nextInt();
		scanner.close();
	}
}
```

Output

```
Knapsack Problem - Dynamic Programming Solution:
Enter the max capacity of knapsack:
5
Enter number of objects:
4
Enter Weights:
```

1
2
2
1
Enter Profits:
15
20
10
12
Items =
4
2
1

Optimal solution = 47

11. Design and implement in Java to find a subset of a given set S = {S_1, S_2,. ,S_n} 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}. Display a suitable message, if the given problem instance doesn't have a solution.

```
import java.util.Scanner;

public class SumOfsubset {
        final static int MAX = 10;
        static int n;
        static int S[];
        static int soln[];
        static int d;

        public static void main(String args[]) {
                S = new int[MAX];
                soln = new int[MAX];
                int sum = 0;
                Scanner scanner = new Scanner(System.in);
                System.out.println("Enter number of elements: ");
                n = scanner.nextInt();

                System.out.println("Enter the set in increasing order: ");
                for (int i = 1; i <= n; i++)
                        S[i] = scanner.nextInt();
                System.out.println("Enter the max. subset value(d): ");
                d = scanner.nextInt();
                for (int i = 1; i <= n; i++)
                        sum = sum + S[i];
                if (sum < d || S[1] > d)
                        System.out.println("No Subset possible");
                else
                        SumofSub(0, 0, sum);
                scanner.close();
        }

        static void SumofSub(int i, int weight, int total) {
                if (promising(i, weight, total) == true)
                        if (weight == d) {
                                for (int j = 1; j <= i; j++) {
                                        if (soln[j] == 1)
```

```
                                        System.out.print(S[j] + "  ");
                    }
                    System.out.println();
                }
                else {
                    soln[i + 1] = 1;
                    SumofSub(i + 1, weight + S[i + 1], total - S[i + 1]);
                    soln[i + 1] = 0;
                    SumofSub(i + 1, weight, total - S[i + 1]);
                }
        }

        static boolean promising(int i, int weight, int total) {
        return ((weight + total >= d) && (weight == d || weight + S[i + 1] <= d));
        }
}
```

Output

```
Enter number of elements:
5
Enter the set in increasing order:
2
3
4
5
6
Enter the max. subset value(d): 9

2 3  4
3  6
4  5
```

12. Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of *n* vertices using backtracking principle.

```
package hamiltoniancycleexp;
import java.util.*;

public class HamiltonianCycleExp {
    public static void main(String[] args) {
        // TODO code application logic here
          HamiltonianCycle obj=new HamiltonianCycle();
 obj.getHCycle(1);
    }

}

class HamiltonianCycle
{
 private int adj[][],x[],n;
 public HamiltonianCycle()
 {
          Scanner src = new Scanner(System.in);
          System.out.println("Enter the number of nodes");
          n=src.nextInt();
          x=new int[n];
          x[0]=0;
          for (int i=1;i<n; i++)
                  x[i]=-1;
          adj=new int[n][n];

 System.out.println("Enter the adjacency matrix");
  for (int i=0;i<n; i++)
    for (int j=0; j<n; j++)
          adj[i][j]=src.nextInt();
 }

public void nextValue (int k)
{
int i=0;
 while(true)
{
          x[k]=x[k]+1;
```

```
        if (x[k]==n)
                x[k]=-1;
        if (x[k]==-1)
                return;
        if (adj[x[k-1]][x[k]]==1)
                for (i=0; i<k; i++)
                        if (x[i]==x[k])
                                break;
        if (i==k)
                if (k<n-1 || k==n-1 && adj[x[n-1]][0]==1)
                        return;
  }
}
public void getHCycle(int k)
{
        while(true)
        {
                 nextValue(k);
                 if (x[k]==-1)
                         return;
                if (k==n-1)
                {
                        System.out.println("\nSolution : ");
                        for (int i=0; i<n; i++)
                                System.out.print((x[i]+1)+" ");
                        System.out.println(1);
                }
                else getHCycle(k+1);
        }
  }
}
```

Output

```
Enter the number of nodes
4
Enter the adjacency matrix
0 1
1 1
1 0 1 1
1 1 0 1
1 1 1 0

Solution :
1 2 3 4 1
```

Solution :
1 2 4 3 1

Solution :
1 3 2 4 1

Solution :
1 3 4 2 1

Solution :
1 4 2 3 1

Solution :
1 4 3 2 1

VIVA-QUESTIONS

1. What is an algorithm?
An algorithm is a sequence of unambiguous instructions for solving a problem. i.e., for obtaining a required output for any legitimate input in a finite amount of time

2. What are important problem types? (or) Enumerate some important types of problems. 1. Sorting 2. Searching 3. Numerical problems 4. Geometric problems 5. Combinatorial Problems 6. Graph Problems 7. String processing Problems

3. Name some basic Efficiency classes
1. Constant 2. Logarithmic 3. Linear 4. nlogn 5. Quadratic 6. Cubic 7. Exponential 8. Factorial

4. What are algorithm design techniques?
Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmically, applicable to a variety of problems from different areas of computing. General design techniques are: (i) Brute force
(ii) divide and conquer
(iii) decrease and conquer
(iv) transform and concquer
(v) greedy technique
(vi) dynamic programming
(vii) backtracking
(viii) branch and bound

5. How is an algorithm"s time efficiency measured?
Time efficiency indicates how fast the algorithm runs. An algorithm's time efficiency is measured as a function of its input size by counting the number of times its basic operation (running time) is executed. Basic operation is the most time consuming operation in the algorithm's innermost loop.

6. How is the efficiency of the algorithm defined?
The efficiency of an algorithm is defined with the components. (i) Time efficiency -indicates how fast the algorithm runs (ii) Space efficiency -indicates how much extra memory the algorithm needs.

7. What are the characteristics of an algorithm?
Every algorithm should have the following five characteristics (i) Input (ii) Output (iii) Definiteness (iv) Effectiveness (v) Termination Therefore, an algorithm can be defined as a sequence of definite and effective instructions, which terminates with the production of correct output from the given input.

8. Write general plan for analyzing non-recursive algorithms.
i. Decide on parameter indicating an input's size.
ii. Identify the algorithm's basic operation
iii. Checking the no.of times basic operation executed depends on size of input.
iv. Set up sum expressing the no.of times the basic operation is executed. depends on some additional property,then best,worst,avg.cases need to be investigated (establishing order of growth)

9. Define the terms: pseudocode, flow chart.
A pseudocode is a mixture of a natural language and programming language like constructs. A pseudocode is usually more precise than natural language. A flowchart is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

10. write general plan for analyzing recursive algorithms.
i. Decide on parameter indicating an input's size.
ii. Identify the algorithm's basic operation iii. Checking the no.of times basic operation executed depends on size of input.if it depends on some additional property,then best,worst,avg.cases need to be investigated of times the basic operation is executed
iv. Set up the recurrence relation,with an appropriate initial condition,for the number
v. Solve recurrence (establishing order of growth)

11. Define the divide an conquer method.

Given a function to compute on 'n' inputs the divide-and-comquer strategy suggests splitting the inputs in to'k' distinct susbsets, 1<k<n, yielding 'k' subproblems. The subproblems must be solved recursively, and then a method must be found to combine subsolutions into a solution of the whole.

12. What is Merge sort?
Merge sort is divide and conquer strategy that works by dividing an input array in to two halves,sorting them recursively and then merging the two sorted halves to get the original array sorted.

13. What is general divide and conquer recurrence?
Time efficiency T(n)of many divide and conquer algorithms satisfies the equation T(n)=a.T(n/b)+f(n).This is the general recurrence relation.

14. Describe the recurrence relation for merge sort?
If the time for the merging operation is proportional to n, then the computing time of merge sort is described by the recurrence relation
T(n) = a n = 1, a a constant
2T (n/2) + n n >1, c a constant

15. The relation between order of growth of functions
O(1) < O(log n) < O(n) < O(n * log n) < O(n2) < O(n3) < O(2n)

16. Asymptotic notations
Big Oh(Worst Case), Big Theta (Average Case), Big Omega(Best Case)

17. Explain the greedy method
Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given 'n' inputs are required us to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one candevice an algorithm that works in stages considering one input at a time.

18.Define feasible and optimal solution.
Given n inputs and we are required to form a subset such that it satisfies some given constraints then such a subset is called feasible solution. A feasible solution either maximizes or minimizes the given objective function is called as optimal solution.

19. What are the constraints of knapsack problem?
To maximize ∑pixi
The constraint is : ∑wixi ≥ m and 0 ≤ xi ≤ 1 1≤ i ≤ n
where m is the bag capacity, n is the number of objects and for each object i wi and pi are the weight and profit of object respectively.

20. Specify the algorithms used for constructing Minimum cost spanning tree
a) Prim's Algorithm b) Kruskal's Algorithm

21. State single source shortest path algorithm (Dijkstra"s algorithm).
For a given vertex called the source in a weigted connected graph,find shotrtest paths to all its other vertices.Dijikstra's algorithm applies to graph with non-negative weights only.

22. State efficiency of prim"s algorithm.
O(|v|2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY) O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

23. State Kruskal Algorithm
The algorithm looks at a MST for a weighted connected graph as an acyclic subgraph with |v|-1 edges for which the sum of edge weights is the smallest.

24. . State efficiency of Dijkstra"s algorithm.
O(|v|2) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY)
O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

25. Define multistage graph

A multistage graph G =(V,E) is a directed graph in which the vertices are partitioned into K>=2 disjoint sets Vi,1<=i<=k.The multi stage graph problem is to find a minimum cost paths from s(source) to t(sink) Two approach(forward and backward)

26. Define All pair shortest path problem
Given a weighted connected graph, all pair shortest path problem asks to find the lengths of shortest paths from each vertex to all other vertices.

27.Define floyd"s algorithm.
To find all pair shortest path.

28. State the time efficiency of floyd"s algorithm
O(n3)

29. What is binary search?
It is an efficient method of finding out a required item from a given list, provided the list is in order. The process is:
1. First the middle item of the sorted list is found.
2. Compare the item with this element.
3. If they are equal search is complete. 4. If the middle element is greater than the item being searched, this process is repeated in the upper half of the list. 5. If the middle element is lesser than the item being searched, this process is repeated in the lower half of the list.

30. What is the another name for Quicksort?
Partition Exchange Sort

31. What is DFS traversal?
Consider an arbitrary vertex v and mark it as visited on each iteration, proceed to an unvisited vertex w adjacent to v. we can explore this new vertex depending upon its adjacent information. This process continues until dead end is encountered.(no adjacent unvisited vertex is available). At the dead end the algorithm backs up by one edge and continues the process of visiting unvisited vertices. This process continues until we reach the starting vertex.

32. What are the various applications of BFS & DFS tree traversal technique? Application of BFS.
Checking connectivity and checking acyclicity of a graph. Check whether there is only one root in the BFS forest or not. If there is only one root in the BFS forest, then it is connected graph otherwise disconnected graph. For checking cycle presence, we can take advantage of the graph's representation in the form of a BFS forest, if the latter vertex does not have cross edge, then the graph is acyclic.
Application of DFS
To check whether given graph is connected or not.
To check whether the graph is cyclic or not.
To find the spanning tree. Topological sorting

33. What is shortest path?
- It is the path which is having shortest length among all possible paths.

34. what is state space tree?
Constructing a tree of choices being made and processing is known as state-spacetree. Root represents an initial state before the search for solution begins.

35. What are the requirements that are needed for performing Backtracking?
Complex set of constraints. They are:
i. Explicit constraints.
ii. Implicit constraints.