LIST OF PROGRAMS

PART – A

Sl. No.	Program
1	Write a program to sort a given set of elements using Merge sort method and find the time required to sort the elements.
2	Write a program to obtain the Topological ordering of vertices in a given digraph using DFS method
3.	Write a program to find element uniqueness using presorting
4.	Write a program to implement Horspool's algorithm for String Matching.
5.	Write a program to implement 0/1 Knapsack problem using dynamic programming
6.	Write a program to find the shortest path using Dijkstra's algorithm for a weighted connected graph.
7.	Write a program to find a subset of a given set S = {S1, S2,.....,Sn} of npositive 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.
8.	Write a program to implement N -queens problem using backtracking.

SOLUTIONS TO LAB PROGRAMS WITH EXPECTED INPUT AND OUTPUT

1. **Write a program to sort a given set of elements using Merge sort method and find the time required to sort the elements.**

```
#include<stdio.h>
#define MAX 1000

int count;

void merge(int a[MAX], int low, int mid, int high)
{
        int i, j, k, b[MAX];

        i = low;
        j = mid+1;
        k = low;
        while( (i<=mid) && (j<=high))
        {
                if(a[i] < a[j])
                {
                        b[k++] = a[i++];
                        count++;
                }
                else
                {
                        b[k++] = a[j++];
                        count++;
                }
        }

        while(i <= mid)
    {
                b[k++] = a[i++];
         }

        while(j <= high)
           {
                 b[k++] = a[j++];
                }

        for(i=low; i<k; i++)
                a[i] = b[i];
}
void mergesort(int a[MAX], int low, int high)
{
        int mid;
        if(low < high)
        {
                mid = (low + high)/2;/*find the mid-point and divide the array into two halves*/
                mergesort(a,low,mid);//call mergesort for first half
                mergesort(a,mid+1,high);//call mergesort for second half
                merge(a,low,mid,high);// merge the two halves sorted
        }
```

```
}

int main()
{
        int i,j,n,a[MAX],b[MAX],c[MAX];
    int c1,c2,c3;
        printf("\n Enter n: ");
        scanf("%d",&n); //read the number of elements

        printf("\nEnter elements: ");
        for(i=0;i<n;i++)
           scanf("%d",&a[i]);// read the input elements to sort

        count=0;
        mergesort(a,0,n-1);

        printf("\n Sorted elements: \n");
        for(i=0;i<n;i++)
                printf("%d\n",a[i]);//display sorted elements
    printf("\n Number of counts : %d\n",count);
        printf("\n SIZE\t ASC\t DESC\t RAND\n");
    for(i=16; i<550;i=i*2)
        {
                for(j=0;j<i;j++)
                {
                        a[j]=j;
                        b[j]=i-j;
                        c[j]=rand() % i;
                }
          count=0;
          mergesort(a,0,i-1);
          c1=count;
          count=0;
          mergesort(b,0,i-1);
          c2=count;
          count=0;
          mergesort(c,0,i-1);
          c3=count;
          printf("\n %d\t%d\t%d\t%d",i,c1,c2,c3);
    }
        return 0;
}
```

Output

```
Enter n: 5
Enter elements:1 3 7 2 0
Sorted Elements:
0
1
2
3
7
```

2. Write a program to obtain the Topological ordering of vertices in a given digraph using DFS method

```
#include<stdio.h>
#include<stdlib.h>
int j=0;pop[10],v[10];

void dfs(int source,int n,int a[10][10])
{
        int i,k,top=-1,stack[10];
        v[source]=1;
        stack[++top]= source+1;
        while(top!=-1)//check if stack is not empty
         {
                for(k=0;k<n;k++)
                 {
                        .if( a[source][k] == 1 && v[k] == 1 )
                         {
                          for(i=top; i>=0;i--)
                           if(stack[i] == k+1 )
                                {
                                 printf("\n Topological order not possible");
                                  exit(0);
                                }
                         }
                    else
                        {
                         if( a[source][k] == 1 && v[k] == 0)
                          {
                                v[k]=1;
            stack[++top]= k+1;
                                source = k;
                                k=0;
                          }
                        }
                 }
          pop[j++]=source+1;
          top --;
          source = stack[top] – 1;
  }
}

void topo(int n , int a[10][10])
{
   int i,k;
   for(i=0;i<n;i++)
      v[i]=0;
   for(k=0;k<n;k++)
              if(v[k]== 0)
                      dfs(k,n,a);//dfs function call
}
```

```
int main()
{
  int n,i,j,a[10][10];
  printf("\n Enter the no of Vertices : ");
  scanf("%d",&n);
  printf("\n Enter the Adjacency matrix\n");
  for(i=0;i<n;i++)
        for(j=0;j<n;j++)
                scanf("%d",&a[i][j]);
  topo(n,a);
  printf("\n The topological ordering is\n");
  for(i=n-1;i>=0;i--)
                printf("%d\t",pop[i]);
}
```

Output

Enter the number of vertices:8
Enter the adjacency matrix

0 1 1 0 1 0 0 0
0 0 1 0 0 1 1 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0

The topological ordering is
3 0 1 6 5 2 4 7

4. Write a program to implement Horspool's algorithm for String Matching.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

#define MAX 256
int t[MAX];
int count=1;

void shifttable(char pat[])
{
  int i,j,m;
  m=strlen(pat);
  for(i=0;i<MAX;i++)
  t[i]=m;
  for(j=0;j<m-1;j++)
  t[pat[j]]=m-1-j;

}

int horspool(char src[],char pat[])
{
  int i,j,k,m,n;
  n=strlen(src);
  m=strlen(pat);
  i=m-1;

  while(i<n)
  {  k=0;
    while((k<m) && (pat[m-1-k]==src[i-k]))
     k++;
    if(k==m)
          return (i-m+1);
    else
    {
          i=i+t[src[i]];
          count=count+1;
    }
  }
  return -1;
}

int main()
{
  char src[100],pat[10];
  int pos;
  printf("\n Enter the main source string\n");
  gets(src);
  printf("\n Enter the pattern to be searched\n");
  gets(pat);
  shifttable(pat);
```

```
pos=horspool(src,pat);
if(pos>=0)
 {
   printf("\n Found at %d position ",pos+1);
   printf("\n number of shifts are %d",count);
 }
 else
   printf("\n String match failed");
 return 0;
}
```

Output

Enter the Text: ABAABCD

Enter the pattern: ABC

Found at 4 position

5. Write a program to implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>

#define MAX 150

//Function declarations
int knap(int n,int m);
int big(int a,int b);

//Global variables
int w[MAX];                          //Array to store weights of each item
int p[MAX];                          //Array to store profits of each item
int v[MAX][MAX];          //Optimal solution of 'i' items with 'j' capacity

int big(int a,int b)//compute largest of given two numbers
{
        if (a > b) return a;
    else return b;
}

int knap(int n,int m)
{
        int i, j;
        for(i = 1; i <= n; i++)
        for(j = 1; j <= m; j++)
        {
                if( (j - w[i]) < 0)
                        v[i][j] = v[i-1][j];
                else
                        v[i][j] = big(v[i-1][j], p[i] + v[i-1][j-w[i]] );
        }

        return v[n][m];
}


int main()
{
        int i, j, profit, n, m;

        printf("\n Enter n (no. of items): ");
        scanf("%d",&n);

        printf("\n Enter the knapsack capacity:");
        scanf("%d",&m);

        printf("\n enter the weights and profits :\n");
        for(i=1;i<=n;i++)
    {
                printf("w[%d] = ",i);
      scanf("%d",&w[i]);
       printf("p[%d] = ",i);
```

```
        scanf("%d",&p[i]);
          }

        for(i=0; i<=n; i++)
                v[i][0]=0;

        for(j=0; j<=m; j++)
                v[0][j]=0;

        profit = knap(n,m);//call knap function to compute profit

        printf("\n goal = %d\n\n",profit);//display profit

        return 0;
}
```

Output

Enter n (no. of items): 4

Enter the knapsack capacity: 5

enter the weights and profits:

2 12
1 10
3 20
2 15

goal = 37

6. Write a program to find the shortest path using Dijkstra's algorithm for a weighted connected graph.

```
#include <stdio.h>

#define INFINITY 999

void dijk(int cost[10][10], int n, int source, int v[10], int d[10]);

int main()
{
        int n;                                  //no. of nodes
        int cost[10][10];                       //Adjacency matrix of graph
        int source;                             //source node
        int v[10];          //visited array. keeps track to nodes visited
        int d[10];          //distance array.shortest distance from source node
        int i, j;                               //index variables

        //1. Read no. of nodes
        printf("Enter n: ");
        scanf("%d",&n);

        //2. Read cost adjacency matrix of graph
        printf("Enter  Cost matrix: \n");
        for(i=1; i<=n; i++)
                for(j=1; j<=n; j++)
                        scanf("%d",&cost[i][j]);

        //3. Read source
        printf("\n enter source:");
        scanf("%d",&source);

        //4. Initialise d[] to distance from source to each node
        //Initialise v[] to 0, indicating none of the nodes are visited
        for(i=1; i<=n; i++)
        {
                d[i] = cost[source][i];
                v[i] = 0;
        }

        //5. Call function to compute shortest distance
        dijk(cost, n, source, v, d);

        //6. Print Shortest distance from source to all other nodes
        printf("Shortest distance from source %d\n\n",source);
        for(i=1; i<=n; i++)
                printf("%d --> %d = %d\n\n",source,i,d[i]);

        return 0;
}

//Function to implement dijkstra algorithm
void dijk(int cost[10][10],int n,int source,int v[10],int d[10])
{
```

```
    int least, i, j, u;

    //A. Mark source node as visited
    v[source] = 1;

    //B. From each node find shortest distance to nodes not visited
    for(i=1; i<=n; i++)
    {
        //B1. Assume least as infinity
        least = INFINITY;

        //B2. Find u and d(u) such that d(u) is minimum i.e., Find //the next nearest node
        for(j=1; j<=n; j++)
        {
            if(v[j] == 0 && d[j] < least)
            {
                least = d[j];
                u = j;
            }
        }

        //B3. Mark u as visited (mark nearest node as visited)
        v[u] = 1;

        //B4. For remaining nodes, find shortest distance through u
        for(j=1; j<=n; j++)
        {
            if(v[j] == 0 && (d[j] > (d[u] + cost[u][j])) )
                d[j] = d[u] + cost[u][j];
        }
    }//end for outer
}//end function
```

Output

```
Enter n: 5

Enter the cost matrix:
0 3 999 7 999
3 0 4 2 999
999 4 0 5 6
7 2 5 0 4
999 999 4 6 0

Enter source: 2

Shortest distance from source 2
2-->1 =3
2-->2 =0
2-->3 =4
2-->4 =2
2-->5 =6
```

7. Write a program to find a subset of a given set S = {S1, S2,.....,Sn} of npositive 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.

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
int s[MAX],x[MAX];
int d;
void sumofsub(int p, int k, int r)
{
  int i;
  x[k]=1;
  if ((p + s[k]) == d)
   {
    for(i=1; i<=k; i++)
    if (x[i] == 1)
    printf("%d ",s[i]);
    printf("\n");
   }
   else
    if (p+ s[k] + s[k+1] <= d)
      sumofsub(p + s[k],k+1, r-s[k]);
  if ((p + r - s[k] >= d) && (p + s[k+1] <= d))
   {
    x[k]=0;
    sumofsub(p,k+1,r-s[k]);
   }
}
void main()
{
 int i,n,sum=0;
 clrscr();
 printf("\n Enter max. number : ");
 scanf("%d",&n);
 printf("\n Enter the set in increasing order :\n");
 for(i=1;i<=n;i++)
 scanf("%d",&s[i]);
 printf("\n Enter the max. subset value : ");
 scanf("%d",&d);
 for(i=1;i<=n;i++)
  sum=sum+s[i];
 if (sum < d || s[1] > d)
  printf("\n No subset possible");
 else
 sumofsub(0,1,sum);
 getch();
}
```

Output :

```
Enter max. number : 5
Enter the set in increasing order :
1 2 5 6 8

Enter the max. subset value : 9
1  2  6
1  8
```

8. Write a program to implement n-queens problem.

```
#include <stdio.h>

//Function declarations
void nqueens(int n);
int can_place(int c[10],int r);
void display(int c[10],int r);

//Global variable
int count = 0;

int main()
{
  int n;

      //1. Read no. of queens
  printf("Enter n (no of queens): ");
  scanf("%d",&n);

  //2. Call function if solution exist
  if(n == 2 || n == 3)
    printf("Solution does not exist.");
  else
      {
    nqueens(n);
      printf("Total no. of solutions: %d\n",count);
  }

  return 0;
}

void nqueens(int n)
{
  int r;                    //Contains row no.
  int c[10];                //Stores queens positions in each row
      int i;

  r = 0;                //Select first queen (place queen in first row)
  c[r] = -1;        //Initial position of queen

  while(r >= 0)         //As long as there are solutions
  {
    c[r]++;             //Place queen in r th coloumn


  //verify there is no attack from any of t previous queens placed
    while(c[r] < n && !can_place(c,r))
      c[r]++;

    if(c[r] < n)
    {
      if(r == n-1)    /if all n queens - display
      {
```

```
            printf("Solution %d: ",++count);
            for(i=0;i<n;i++)
               printf("%4d",c[i]+1);

            display(c,n);
         }
         else      //else place the next queen in next row
         {
            r++;
            c[r] = -1;
         }
      }
      else
         r--;      //backtracking (go to previous row)
   }
}

//Function to check attack on queen r from 0-(r-1) queens
//return 0: if there is attack, other wise return 1;

int can_place(int c[10],int r)
{
   int i;

   for(i=0; i<r; i++)
   {
      if( (c[i] == c[r]) || (abs(i-r) == abs(c[i] - c[r])) )
         return 0;
   }
   return 1;
}

//Function to create chessboard with queens placed and display
void display(int c[10],int n)
{
   char cb[10][10];
   int i, j;

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         cb[i][j] = '-';

   for(i=0;i<n;i++)
      cb[i][c[i]] = 'Q';

   //Display the chess board
   printf("\n\nChessboard: \n");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
         printf("%4c",cb[i][j]);
      printf("\n\n");
   }
}
```

Output

Run1

Enter the total number of queens: 4
Total number of solutions: 2
Solution 1:
Chessboard
_Q_ _
_ _ _ Q
Q_ _ _
_ _ Q _
Solution 2
Chessboard
_ _ Q _
Q_ _ _
_ _ _ Q
_ Q _ _

Run2
Enter the number of queens: 3
Solution does not exist

DAA LAB VIVA QUESTION BANK

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 conquer

(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

$O(1) < O(\log n) < O(n) < O(n * \log n) < O(n^2) < O(n^3) < O(2^n)$

16. Asymptotic notations

Big Oh(Worst Case), Big Theta (Average Case), Big Omega(Best Case)

17. What is the time complexity of linear search?

$\Theta(n)$

18. What is the time complexity of binary search?

Θ(log2n) **3) What is the major requirement for binary search?**

The given list should be sorted.

19. 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.

20. What is parental dominance?

The key at each node is greater than or equal to the keys at its children.

21. What is heap?

A **heap** can be defined as a binary tree with keys assigned to its nodes (one key per node) provided the following two conditions are met:

1 The tree's shape requirement – The binary tree is essentially complete (or simply complete), that is, all its levels are full except possibly the last level, where only some rightmost leaves may be missing.

2. The parental dominance requirement – The key at each node is greater than or equal to the keys at its children

22. What is height of Binary tree?

It is the longest path from the root to any leaf.

23. What is the time complexity of heap sort?

Θ(n logn)

24. What is Merge Sort?

Merge sort is an O (n log n) comparison-based sorting algorithm. Where the given array is divided into two equal parts. The left part of the array as well as the right part of the array is sorted recursively. Later, both the left sorted part and right sorted part are merged into a single sorted array.

25. Who invented Merge Sort?

John Von Neumann in 1945.

26. On which paradigm is it based?

Merge-sort is based on the divide-and-conquer paradigm.

27. What is the time complexity of merge sort?

n log n.

28. What is the space requirement of merge sort?

Space requirement: $\Theta(n)$ (not in-place).

29. When do you say an algorithm is stable and in place?

Stable – if it retains the relative ordering. In – place if it does not use extra memory.

30. Who invented Dijkstra's Algorithm?

Edsger Dijkstra invented this algorithm.

31. What is the other name for Dijkstra's Algorithm?

Single Source Shortest Path Algorithm.

32. Which technique the Dijkstra's algorithm is based on?

Greedy Technique.

33. What is the purpose of Dijkstra's Algorithm?

To find the shortest path from source vertex to all other remaining vertices

34. Name the different algorithms based on Greedy technique.

Prim's Algorithm, Kruskal's algorithm

35. What is the constraint on Dijkstra's Algorithm?

This algorithm is applicable to graphs with non-negative weights only.

36. What is a Weighted Graph?

A weighted graph is a graph with numbers assigned to its edges. These numbers are called weights or costs.

37. What is a Connected Graph?

A graph is said to be connected if for every pair of its vertices u and v there is a path from u to v.

38. What is the time complexity of Dijkstra's algorithm?

For adjacency matrix, It is
(IVI*IVI)

For adjacency list with edges stored as min heap, it is O(IEIlogIVI)

39. Differentiate b/w Traveling Salesman Problem(TSP) and Dijkstra's Algorithm.

In TSP, given n cities with known distances b/w each pair , find the shortest tour that passes through all the cities exactly once before returning to the starting city.

In Dijkstra's Algorithm, find the shortest path from source vertex to all other remaining vertices

40. Differentiate b/w Prim's Algorithm and Dijkstra's Algorithm?

Prim's algorithm is to find minimum cost spanning tree.

Dijkstra's algorithm is to find the shortest path from source vertex to all other remaining vertices.

41. What are the applications of Dijkstra's Algorithm?

It finds its application even in our day to day life while travelling. They are helpful in routing applications.

42. Who was the inventor of the Quicksort?

C.A.R.HOARE, a British Scientist.

43. Which design strategy does Quicksort uses?

Divide and Conquer

44. What is Divide and Conquer Technique?

(I) Divide the instance of a problem into two or more smaller instances

(II) Solve the smaller instances recursively

(III) Obtain solution to original instances by combining these solutions

45. What is the another name for Quicksort?

Partition Exchange Sort

46. Is Quicksort Stable as well as In place?

Not Stable but In place.

47. When do you say that a Quick sort having best case complexity?

When the pivot exactly divides the array into equal half

48. When do you say that Quick sort having worst case complexity?

When any one of the subarray is empty

49. How many more comparisons are needed for average case compared to best case?

38% more

50. When do you say that the pivot element is in its final position?

When all the elements towards the left of pivot are <= pivot and all the elements towards the right of pivot are >= pivot.

51. What technique does kruskal's algorithm follow.

Greedy technique

37) What is the time complexity of kruskal algorithm.

With an efficient sorting algorithm, the time efficiency of an kruskal's algorithm will be in

O(|E|log|E|).

52. Who is the inventor of kruskal's algorithm.

Joseph Kruskal.

53. Which technique is used to solve BFS & DFS problems?

Decrease and Conquer

54. 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.

55. 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.

56. Which data structures are used in BFS & DFS tree traversal technique?

We use Stack data structure to traverse the graph in DFS traversal.

We use queue data structure to traverse the graph in BFS traversal.

57. What is Dynamic Programming?

It is a technique for solving problems with overlapping sub problems.

58. What does Dynamic Programming have in common with Divide and- Conquer?

Both solve the problems by dividing the problem into sub problems. Using the solutions of sub problems, the solutions for larger instance of the problem can be obtained.

59. What is a principle difference between the two techniques?

Only one instance of the sub problem is computed & stored. If the same instance of sub problem is encountered, the solution is retrieved from the table and never recomputed. Very precisely re - computations are not preformed.

60. What is a spanning tree ?

A spanning tree of a weighted connected graph is a connected acyclic(no cycles) subgraph (i.e. a tree)that contains all the vertices of a graph and number of edges is one less than number of vertices.

61. What is a minimum spanning tree ?

Minimum spanning tree of a connected graph is its spanning tree of smallest weight.

62. Prim's algorithm is based on which technique.

Greedy technique.

63. Name some of the application greedy technique

They are helpful in routing applications: In the case of network where large number of computers are connected, to pass the data from one node to another node we can find the shortest distance easily by this algorithm. Communication Networks: Suppose there are 5 different cities and we want to establish computer connectivity among them, then we take into the account of cost factor for communication links. Building a road network that joins n cities with minimum cost.

64. Does Prim's Algorithm always yield a minimum spanning tree ?

Yes

65. What is the purose of Floyd's algoreithm?

- To find the all pair shortest path.

66. What is the time efficiency of floyd's algorithm? - It is same as warshall's algorithm that is $\theta(n^3)$.

67. What is shortest path?

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

68. Warshall's algorithm is optimization problem or non-optimization problem ?

Non-optimization problem

69. For what purpose we use Warshall's algorithm?

Warshall's algorithm for computing the transitive closure of a directed graph

70. Warshall's algorithm depends on which property?

Transitive property

71. What is the time complexity of Warshall's algorithm?

Time complexity of warshall's algorithm is $\theta(n^3)$.

72. What is state space tree?

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

73. What are promising and non-promising state-space-tree?

Node which may leads to solution is called promising.

Node which doesn't leads to solution is called non-promising.

74. What are the requirements that are needed for performing Backtracking?

Complex set of constraints. They are: i. Explicit constraints. ii. Implicit constraints.

75. 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.

76. 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

77. What are the constraints of knapsack problem?

To maximize $\sum pixi$

The constraint is : $\sum wixi \geq m$ and $0 \leq xi \leq 1$ $1 \leq i \leq n$

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.

78. Specify the algorithms used for constructing Minimum cost spanning tree.

a) Prim's Algorithm

b) Kruskal's Algorithm

79. 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.

80. 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)

81. State Kruskal Algorithm.

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

82. 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)