INFORMATION SCIENCE

ANALYSIS AND DESIGN OF ALGORITHM

M. SUDHA RANI, B.Tech.
Lecturer
Department of Information Science
M. N. TECHNICAL INSTITUTE
Kamma Gondana Halli, Bengaluru - 560015

UNIT 1

INTRODUCTION

SYLLABUS

1. Introduction

1.1 Algorithm Definitions

1.1.1 Fundamental of algorithm problem solving

1.1.2 The efficiency of algorithms.

1.1.3 Best, Average and worst case analysis

1.2 Methodologies for Analyzing algorithm

1.2.1. Pseudocode

1.2.2. Counting the primitive operations

1.2.3. Algorithm Complexities.

1.2.3.1 Space Complexity

Analysis of space complexity

How to calculate space complexity.

Time complexity.

1.3. Asymtotic Notations

The Big-oh Notation.

The Big- theta Notation.

Ordering functions by their Growth rates.

SYNOPSIS

Introduction

Algorithm plays an important role in both science and computing. The word algorithm comes from the name of a persian author "Abu Jafar Mohammed Ibu Musa al Khowarismi". It provides tools for designing algorithms for new problems.

It is a well-organized, pre-arranged and well-defined step by step procedure that specifies the solution to a given problem, which will be used to perform some task in a finite amount of time.

An algorithm should consist of characteristics such as, input, output, definiteness, finiteness and effectiveness.

Fundamentals of algorithm problem solving: The sequence of steps which are to be considered for solving a problem are

- Understanding the problem.
- Ascertaining the capabilities of computational device choosing between exact and approximate algorithms.
- Deciding an appropriate datastructure
- Designing an algorithm
- Method of specifying an algorithm
- Proving the correctness of an algorithm
- Analyzing an algorithm
- Coding
- Debugging

Efficiency of an algorithm: Efficiency means how effectively the algorithm can be executed within less time and less space. It can be measured by using time and space efficiency with best, average and worst case analysis

Best Case: an algorithm will take very least time to execute.

Worst Case: an algorithm takes very longest time to get executed.

Average Case: an algorithm takes average time to get executed.

Methodologies for Analyzing an algorithm.

There are different types of methodologies to analyze an algorithm. They are

- Pseudocode: It is a method of representing an algorithm using natural and programming language constructs.

It is similar to the higher languages such as C, pascal or Java etc.

- Counting Primitive Operations: It is a method to analyze the running time of an algorithm for this, the primitive operations or basic operations should be identified and the execution time of each primitive operation should be determined and counted.
- Algorithm Complexities: An algorithm can be analyzed by using two complexities:

→ Space complexity: It is the amount of memory that may be required to run a program.

→ Time Complexity: It is the amount of time required for the program or an algorithm execution. i. e. how fast an algorithm runs.

Ordering functions by their growth rates:

Algorithms are expected to work fast for all values of n. But some algorithms execute faster for smaller values of n. As value of n increases they tend to become slow. This change is called order of growth.

Asymptotic Notation:

The word "asymptotic" means "study of functions". Functions with parameter n, can analyzed by using three notations.

Big-oh Notation: It is a method of expressing upper bound with function.

$$f(n) \le c^* g(n), \text{ for all } n \ge n_o$$
$$f(n)\ \varepsilon\ O(n)$$

Big-Omega Notation: It is a method of expressing lower bound with function,

$$f(n) \ge c * g(n), \text{ for all } n \ge n_o$$
$$f(n)\ \varepsilon\ \Omega(n)$$

Big-Theta Notation: It is method of expressing both upper and lower bound with function,

$$C_1 * g(n) \le f(n) \le C_2 * g(n), \text{ for all } n \ge n_o$$
$$f(n)\ \varepsilon\ \Theta(n)$$

5 MARK QUESTIONS

1. Define Algorithm. Explain with an example

Ans. An algorithm is a step-by-step procedure to perform some task in a finite amount of time in a finite number of steps.

Example: Sum of two numbers

step 1:start

step 2:Input A,B

step 3:compute sum

sum←A + B

step 4:output sum

step 5:stop

2. List the various steps involved in designing an algorithm.

Ans. An algorithm design technique is a general method of solving a problem in the form of algorithms. It can be specified in three ways such as,

1. Natural language: It is a method in which an algorithm can be written in english like statements.
Eg:

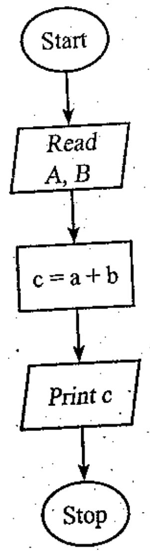
step 1: Read 2 numbers

step 2: Add 2 numbers

step 3: Print results

2. Flow chart: It is a pictorial representation, in which geometrical shapes such as boxes,circles etc. are used to represent an algorithm.

Eg;



3. Pseudocode: It is a method of representing an algorithm using natural language and programming language.Eg:

step 1: Start

step 2: [Read a,b]

    int a,b

step 3: [compute c]

    c = a + b

step 4: Print c

step 5 : Stop

3. Write an algorithm to find sum and average of three numbers.

Ans Algorithm: Sum and Avrage of three numbers

Input: Input 3 + ve integers

Output: Output the sum and Avg.

```
    Read A, B, C
    Sum ← A + B + C
    Avg ← sum/3
```

4. Write an algorithm to find largest of three numbers.

Ans. Algorithm: Largest of three numbers

Input: Read three +ve integers

Output: Print the largest of three numbers

```
     if a > b then
     if a > c then
    Print a
else
    Print c
end if
else
     if b > c then
      Print b
else
    Print c
  end if
end if
```

5. Write the advantages and disadvantages of an algorithm.

Ans. Advantages

1. step-by-step representation of a solution
2. Easy to understand
3. It has got a definite procedure
4. Easy to convert into flowchart and program from algorithm

5. Easy to debug
6. Independent of programming language

Disadvantages

1. Time consuming
2. Complicated to create program

6. Write a note on efficiency of algorithms.

Ans. The efficiency of an algorithm can be measured by using time and space efficient.

Time Efficiency: It indicates how fast an algorithm can be executed. It depends on factors such as:

- Speed of the computer
- Compiler used
- Choice of programming language
- Choice of algorithm
- Number of i/p's and o/p's

Space Efficiency: It indicates how much space is used by algorithm during execution. It depends on factors such as,

- Program space
- Data space
- Stack space

7. Explain best, worst and average case Analysis

Ans. An algorithm can require different times to solve different problems of the same size.

Best-case Analysis:

The minimum amount of time that an algorithm require to solve a problem of size n is called best case. The best case behavior of an algorithm is Not so useful.

Worst-case Analysis:

The maximum amount of time that an algorithm require to solve a problem of an size n is called worst case. It gives an upper bound for the time complexity of an algorithm.

Average case Analysis:

The average amount of time that an algorithm require to solve a problem of size n is called average case: Sometimes, it is difficult to find the average-case behavior of an algorithm

8. What is a pseudo code? Explain with an example

Ans. A pseudocode is a method of representing an algorithm using natural language and programming language constructs.

Example: To find large number in an array.

Alogithm:Max_Array(A,n)

Input : An array storing n integers

Output: Large element in an array A

```
        large←A[0]
        for i←1 to n-1 do
            large<A[i] then
            large←A[i]
return large
```

9. Explain counting primitive operations with suitable example

Ans. Algorithm: Inner product

Input : the integer n, and two arrays A and B of size n.

Output: inner product of two arrays

```
        prod←0
            for i ← 0 to n-1 do
                prod ← prod + A[i] * B[i]
        return prod
```

1. Line 1 is one op (assigning a value)
2. Loop initializing is one op (assigning a value)
3. Line 3 is five ops per iteration (mult, add, 2 array references,assign)
4. Line 3 is executed n times: total is 5n
5. Loop incrementation is two ops (an addition and an assignment)
6. Loop incrementation is done n times,total is 2n
7. Loop termination test is one op (a comparison $i < n$) each time
8. Loop termination is done n + 1 times (n successes,one failure) total n + 1
9. Return is one op

Total is thus,

$1 + 1 + 5n + 2n + (n + 1) + 1 = 8n + 4$

10. What is space complexity? Explain with suitable example

Ans. Space complexity of a program is the amount of memory that may be required to run a program. The components required to calculate space are

Instruction space: This is a space required to store the machine code generated by the compiler

Data space: This is a space required to store variables i.e. constants, static or dynamic variables, etc.

Stack space: This is a space required to store return addresses and return values.

The overall space requirement is the sum of statically allocated storage for the code segment and dynamically allocated storage for activation records. Hence,

$S(P) = C_P + S_P$

where,

C_P = Space required for code segment (static part)

S_P = Space required for activation records(dynamic part)

Example: Finding avg of three integer numbers

Algorithm: Avg of three nos.

Input: Three +ve integers

Output: Average of 3 positive integers

```
Read a, b, c, avg
avg = a + b + c/3
print avg
```

Space occupied by,

Integer variables are,

= 4 * 2 (i.e a, b, c and each of size 2bytes)

= 8 bytes

Constant variable = 2 bytes (i.e 3 is a constant)

Total space = 8 + 2 = 10bytes.

11. What is time complexity?

Ans. It is the amount of time a program or an algorithm takes for execution i.e how fast an algorithm runs.

Input n

All problems will be having one thing as common i.e input size 'n' when 'n' increases the time taken by the algorithm also increases

Eg: To find max element in an array, the time required is n, because until all the elements in the array are compared we can't find out max element.

Unit of Algorithms Runtime

The unit of time possibly seconds,or milliseconds or any other similar unit and it may also depends on the type of computer used.

The standard method of computing time efficiency of an algorithm are

1. Operation counts
2. Step counts
3. Mathematical Analysis
4. Practical method

The simplest way is the first method

The running time of an algorithm T (n) is

T(n) = t*c (n)

where,t = time taken by the basic operation

c (n) = no.of times the code need to be executed

Eg: (i)for i ← 1 to n do

```
{.....
}
```

Assume t as 1,then

T(n) = t* (cn)

= 1 * n

= n.

(ii) for i ← 1 to n do

```
for j ← 1 to n do
{
....}
```

Assume t as 1, then

T(n) = t* c(n)

= 1*n^2 (i.e two loops)

= n^2

12. Explain big-oh notation with an example.

Ans. Big-oh is the formal method of expressing the upper bound of an algorithm's running time. It is a measure of the largest amount of time it could possibly take for an algorithm to complete

Let f(n) be the time efficiency.Then f(n) is said to be Big-oh of g(n), denoted by

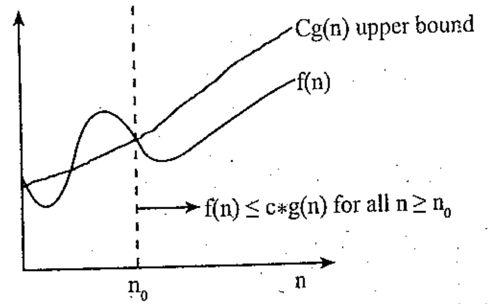
f(n) Ɛ O(n)

or

f(n) = O (g(n))

Such that there exists a positive constant 'c', and non-negative integer 'n_0' satisfying the constraint

$f(n) \le c*g(n)$ for all $n \ge no$



The above graph indicates that f(n) will not consume more than specified time c*g(n).

Eg: $f(n) = 10n^3 + 8$. Express f(n) using Big-oh.

Given that $f(n) = 10n^3 + 8$

Replacing 8 with n^3 (so that next higher order term is obtained) i.e.

$c.g(n) = 10n^3 + n^3$

$= 11n^3$　　　for $n = 8$

Thus, $f(n) \le c * g(n)$ for $n \ge no$

$10n^3 + 8 \le 11 * n^3$ for $n \ge 8$

$c = 11, g(n) = n^3$ and $n_0 = 8$ By definition,

$f(n)\ \mathcal{E}\ 0\ (g(n))$ i.e. $f(n)\ \mathcal{E}\ 0(n^3)$

13. Explain big - omega notation with an example.

Ans: It gives the lower - bound on a function f (n) within a constant factor. The lower bound indicates that f (n) will consume atleast the specified time c * g(n).

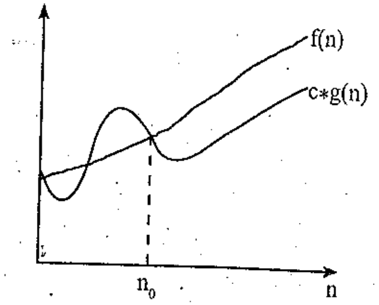
Let f(n) be the time complexity of an algorithm, then the function f(n) is said to be Big-omega of g(n), denoted by

$f(n)\ \mathcal{E}\ \Omega\ (g\ (n))$

$f(n) = \Omega\ (g\ (n))$.

such that there exists a positive constant 'c', and non-negative integer 'n_0' satisfying the constant.

$f(n) \ge c * g(n)$ for $n \ge n_o$



Eg: Let $f(n) = 10n^3 + 8$. Express using Big omega notation.

$f(n) \ge c * g(n)$　　for all $n \ge = 0$

i.e $10n^3 + 8 \ge 10 * n^3$　　for $n >= 0$

Thus, $c = 10$, $g(n) = n^3$ and n_o

Hence $f(n)\ \mathcal{E}\ \Omega\ (n^3)$

14. Explain big-theta notation with an example.

Ans: It is used to denote both lower and upper bound on a function f(n) within a constant factor.

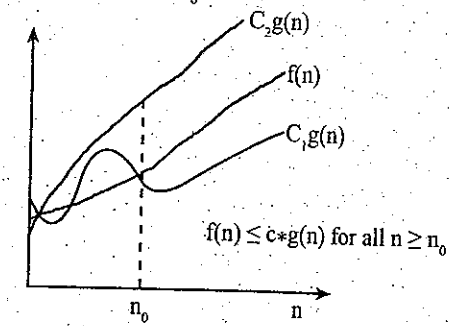
Let f(n) be the time complexity of an algorithm. The function f(n) is said to be Big-theta of g(n) which is denoted by

$f(n)\ \mathcal{E}\ \theta\ (g(n))$

$f(n) = \theta\ (g(n))$

Such that there exists a the constants "C1, C2" and non-negative integer no satisfying the constraint

$C_1 * g(n) \le f(n) \le C_2 * g(n)$　　for all $n \ge n_o$



Such upper bound on f(n) indicates that f(n) will not consume more than specified time $C_2 * g(n)$. The lower bound on f(n) indicates that the function f(n) will consume at leat the specified time $C_1 * g(n)$.

Eg: Let $f(n) = 10n^3 + 5$ Express f(n) using Big-omega. The constraint to be satisfied is

$C_1 * g(n) \le f(n) \le C_2 * g(n)$ for all $n \ge n_o$

$10 * n^3 \le 10 n^3 + 5 \le 11 * n^3$ for $n \ge 2$

It is clear that,

$C_1 = 10$, $C_2 = 11$, $n_o = 2$, $g(n) = n^3$

By definition,

$f(n) \,\mathcal{E}\, \theta(g(n))$

i.e $f(n) \,\mathcal{E}\, \theta(n^3)$

15. f(n) = 100n + 5, analyze for worst case.

Ans. Let f(n) be the time complexity. The worst case is that the function f(n) will not consume more than the specified time. Thus it should be proved that,

$f(n) \,\mathcal{E}\, o(n)$

Given,

$f(n) = 100n + 5$

Replacing 5 with n (So that next higher order term is obtained). thus we get

$*cg(n) = 100n + n$ for $n = 5$

$= 101n$ for $n = 5$

The constraint is satisfied as,

$f(n) \le c * g(n)$ for $n \ge n_o$

i.e $100n + 5 \le 101 n$ for $n \ge 5$

Thus, $c = 101$, $g(n) = n$, $n_o = 5$. By definition,

$f(n) \,\mathcal{E}\, O(g(n))$ Hence, $f(n) \,\mathcal{E}\, O(n)$

16. f(n) = 100n + 5, Analyse for best case.

Let f(n) be the time complexity. The best case to be considered is that atleast it should take minimum specified time.

The constraint to be satisfied is

$f(n) \ge C * g(n)$ for $n \ge n_o$

$100n + 5 \ge 100 * n$ for $n \ge n_o$

Thus, $c = 100$, $g(n) = n$, $n_o = 0$

so by definition, $f(n) \,\mathcal{E}\, \Omega(n)$

17. f(n) 100n + 5 Analyse for average case.

Ans: Let f(n) be the time complexity of an algorithm. The average case is that the f(n) will not consume more than upper bound and it will consume for best atleast the lower bound value.

Thus,

$C_1 * g(n) \le f(n) \le C_2 * g(n)$ for all $n \ge n_o$

$100 * n \le 100n + 5 \le 105 * n$ for $n \ge 5$

Thus,

$C_1 = 100$, $C_2 = 105$, $n_o = 5$, $g(n) = n$

So By definition,

$f(n) \,\mathcal{E}\, \Theta(g(n))$

$f(n) \,\mathcal{E}\, \Theta(n)$

18. Write a note on ordering functions by their growth rates.

Ans. Algorithms are expected to work fast for all values of n. Some algorithms execute faster for smaller values of n. But, as the value of n increases, they tend to be very slow.

The change in behavior as the value of n increases is called "order of growth".

The table shows the values of some of the functions.

N	log N	N log N	N^2	N^3	2^N	N!
1	0	0	1	1	2	1
2	1	2	4	8	4	2
4	2	8	16	64	16	24
8	3	24	64	512	256	40320
16	4	64	256	4096	65536	high
32	5	160	1024	32768	4294967216	very high

1: It indicates that running time of a program is constant.

log N: It indicates that running time of program is logarithmic. It occurs in programs that solve larger problems.

N: It indicates that running time of a program is linear. i.e when N is 100, running time is 100 units when N is doubled, so does the running time.

N log N: It indicates that running time of a program is N log N. The divide- and - conquer algorithms will have this running time.

N^2: It indicates that running time of a program is quadratic. The algorithms which have two loops such as sorting algorithms, addition and subtraction of matrices will have this running time.

N^3: It indicates that running time of a program is cubic. These algorithms will have three loops. Eg: matrix multiplication, simultaneous equation solving.

2^N: Indicates running time of an algorithm is exponential The algorithms that generate subsets of a given set will have this running time.

N!: It indicates that running time of an algorithm is factorial. The algorithms that generate all permutations of set will have this running time.

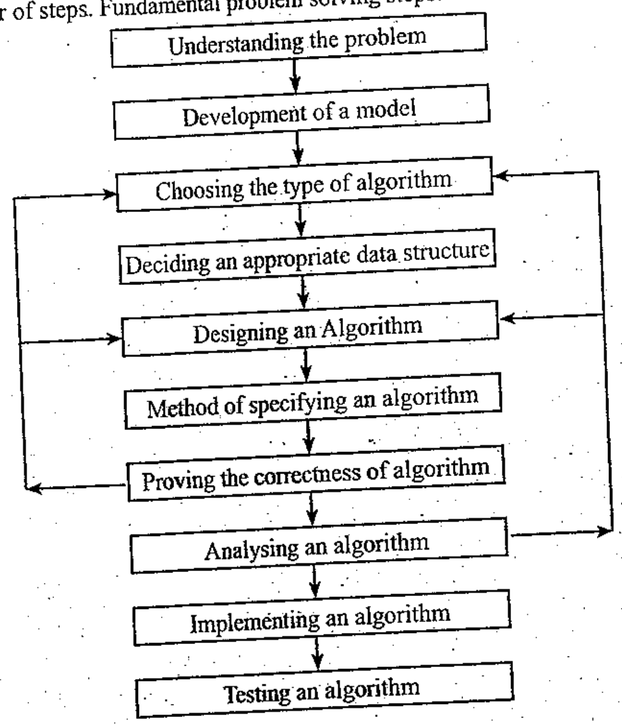
10 MARK QUESTIONS

1. F(n) = 100n + 5, Analyse for best, worst and average cases.

Ans: Ref. Q. 15, Q. 16, Q. 17.

2. Define algorithm. Explain the fundamental steps in solving any algorithm.

Ans: An algorithm is a step - by - step procedure to perform some task in a finite amount of time in a finite number of steps. Fundamental problem solving steps:



*** Understanding the problem:**

The given problem description,

→ Should be read carefully.

→ Should be understood clearly and completely.

→ Special cases have to be thought about

→ Any doubts should be clarified.

→ Missing information should be identified.

→ Should determine the output.

→ Exact range of input should be specified.

*** Development of a model:**

To ascertain the capabilities of a computational device, an algorithm is based on 3 factors such as,

1. Architecture of the device: If it is based on Von- neuman architecture, then sequential algorithm must be designed. If it is based on parallel processing, then parallel algorithms must be designed.

2. Speed of the device: It is important only for the military and real time applications where time is a critical factor.

3. Memory Space: Large memory space is required for the problems which are complex and which involve large volumes of data.

*** Choosing the type of Algorithm.**

Based on the output of an algorithm, there are 2 types. They are

1. Exact Algorithms: These solve the problems exactly.

Eg: Sorting, Searching, pattern matching, TSP etc.

2. Approximate Algorithms: These solve the problem approximately.

Eg: Finding sqrt, solving non-linear equations etc.

*** Deciding an appropriate data structure.**

In this phase, a well - defined algorithm and data structure is selected to produce an efficent program.

Algorithms + data structure = program

*** Designing an Algorithm:**

It is a general method of solving a problem in the form of an algorithm. The various design techniques are,

- Brute force Method
- Divide and conquer
- Decrease and conquer
- Iterative Improvement
- Transform and conquer
- Dynamic Programming
- Greedy technique.

*** Method of specifying an Algorithm:**

An algorithm can be specified in 3 ways such as,

1. Natural language - An algorithm can be written in english like statements with very little

mathematical expressions.

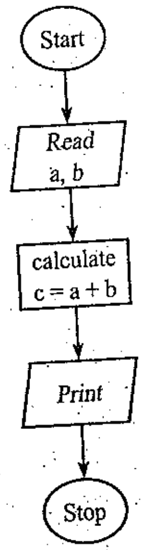
Eg: Read 2 numbers.

Add 2 numbers.

display

2. Flow Chart: It is a pictorial representation of an algorithm. All the steps are drawn by using geometrical shapes, such as circles, boxes etc.

Eg:



3. Pseudocode: It is a method of representing an, algorithm using natural language and programming language.

Eg: step 1: Start

step 2: [Read a, b]

int a, b

step 3: [compute c]

$c \leftarrow a + b$

step 4: print C

step 5: Stop

*** Proving the correctness of an Algorithm: (Validation)**

After designing an algorithm for a specific problem the correctness should be proved i.e it must compute the correct result for all possible inputs. This is **called** algorithm validation. A technique

called mathematical induction is used for validation.

*** Analysing an Algorithm:**

It is a process of determining the computing time and storage. It helps to improve the efficiency of an algorithm. For each algorithm the user has to predict the best case, average case and worst cases by using space and time efficiencies of an algorithm.

There are two phases, which are used for analyzing the computing time of an algorithm. They are

1. Priori Analysis: This is the first phase, in which the frequency count of the statement is determined. It is defined as "the number of times the statement is executed".

2. Posteriori Analysis:

This is the second phase, and it is the process of executing the program and finding out the time and space taken by an algorithm. Hence it depends on the machine and language used.

*** Implementing an algorithm:**

An algorithm should be converted into a program by using different languages such as c, c++, c#, Java etc. The syntax may vary but the selected language should support the features specified in design phase.

*** Testing an algorithm:**

It is the process of identifying errors in a program. It executes the problem on correct data sets and measures the time and space it takes to compute the results.

3. Explain best, worst and average case analysis with an example.

Ans: Consider searching an element in an array.

Worst case Analysis:

In worst case, an algorithm takes longest time to execute among all possible inputs. It requires maximum number of comparisons.

The worst cases for searching an element are,

- When there are no matching element found.
- The first matching element happens to be the last one in the list.

Thus the algorithm makes n number of key comparisons among all possible inputs for the worst case.

Hence $C_{worst}(n) = n$

Best case Analysis:

In best case, an algorithm for the input of size "n" for which it takes least time to execute among all possible inputs is called best case efficiency.

In this case, the algorithm runs fastest among all possible inputs i.e. the element can be found at first itself.

Hence $C_{Best}(n) = 1$

Average Case Analysis:

The average case indicates that the location of the element may be any where in the list.

for linear search the assumptions are,

1). The probability of successful search is p or 1.

"$0 \le p \le 1$".

2). The probability of a match occuring in the i^{th} position of a list is "p/n" for every "i".

3). For unsuccessful search, the number of comparisons made is "n" with probability of (1-p).

Hence, $C_{avg}(n) = 1. p/n + 2. p/n + ... i. p/n + ...n\ p/n] + n(1-p)$

$= p/n\,[1 + 2 + ... + i + ... + n] + n(1-p)$

$$= p/n\left[\frac{n(n+1)}{2}\right] + n(1-p)$$

$$= \frac{p(n+1)}{2} + n(1-p)$$

$$C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p)$$

If p = 1, i.e search is successful.

$$C_{avg}(n) = \frac{n+1}{2}$$

If p = 0 ; i.e search is unsuccessful

$C_{avg}(n) = n$

4. Explain the following.

Ans: (a) Algorithm with an example.

Ref. Q1. (5 Marks)

(b) Best, worst and average case of linear search

Ref. Q3. (10 Marks)

4. Explain the following with suitable examples.

Ans. (a) Big-ohm notation

Ref. Q12. (5 Marks)

(b) Big-omega notation

Ref. Q13. (5 Marks)

UNIT 2

GRAPHS OPTIMIZATION PROBLEMS

SYLLABUS

2.1 Graphs.

Definitions and Representations.

Different types of graph

Searching Methods: DFS and BFS

Introduction to Trees

Applications.

2.2 Optimization Problem.

Feasible solutions.

Optimal solutions.

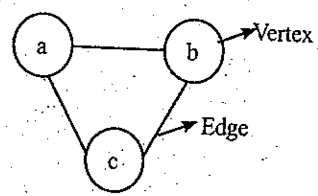
Important problem types: Sorting, searching, string processing graph, problems, combinational problems, Genometic problems.

Numeric problems.

SYNOPSIS

Graph:

A graph is an abstract representation of a set of objects where some pair of the objects are connected by using links. An object is called as a node or vertex which is represented by a circle. A link is called as an edge which is a line between two vertices.



Definition: A graph is defined as a pair of 2 sets.

V and E, denoted by G = (V, E)

where, V- set of vertices

E - set of edges.

Graph representation: A directed or undirected graph can be represented in two ways.

1. Adjacency Matrix and 2. Adjacency linked list.

Types of Graph: Based on how a graph is drawn, there are different types of graphs, such as

- Connected graph
- Planar and non-planar graphs
- Weighted graphs
- Labelled graphs
- Cyclic and Acyclic graphs.

Searching Methods.

A graph search or traversal is a method, which visits every node exactly once in a systematic fastion. There are two standard graph searching methods:

DFS: (Depth-First Search)

It is a method of traversing a graph depth-wise. As the name implies, the search is deeper and deeper in the graph.

BFS: (Breadth-First Search)

It is a method of traversing a graph breadth-wise i.e. the search continues horizontally of breadth-wise level by level.

Introduction to trees:

A tree is a connected acyclic graph which is also called as a free tree.

A tree is a non-linear type of data structure. A graph is called as a tree if and only if it doesn't contain a circuit and it is connected.

The graph applications are used in real life applications.

Optimization Problems:

Optimization refers to choosing the best element from some set of available alternatives. Feasible and optimal solutions are the ones which will be used to find the optimal solutions.

Important problem types: These come across while dealing with optimization problems.

Sorting: Process of arranging items in some sequence.

Searching: Process of finding a particular item in the large amount of data.

String Processing: It deals with manipulation, matching, replacing, searching, deleting a string in a text.

Graph Problems: These are used to model real time applications.

Combinational Problems: It involves permutations, combinations or sub set constructions, calendar Problems.

Geometric Problems: It deals with geometric object problems such as points, curves, lines etc.

Numerical Problems: It deals with mathematical manipulations such as solving equations, computing differentiation and integration etc.

5 MARK QUESTIONS

1. Define graph. Explain its applications.

Ans. A graph G is defined as a pair of 2 sets V and E, which is denoted by G = (V,E).

where, V = set of vertices

V = set of edges

Applications:

1. Used to represent link structure of a website.
2. Weighted graphs with pairwise connection represents road network, communication network etc.
3. Graph theory is used to study molecules is physics and chemistry.
4. It is also widely used in sociology, biology and conservation effects.

2. Explain node, vertex, edge in a graph with an example.

Ans. Vertex - It is a synonym for a "node", and a node is represented by a circle.

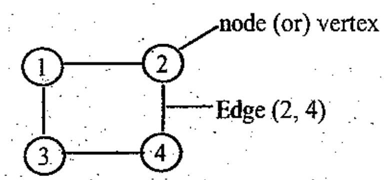
Eg: 1 2

3 4

The nodes are identified as 1, 2, 3, 4 which are also called as "vertices" and is denoted by a set

V = {1, 2, 3, 4}

Edge - If u and v are vertices, then a line joining these two vertices is called as an "edge", and is denoted as (u, v)



Hence from the above graph,

Vertices V = {1, 2, 3, 4}

Edges E = {(1, 2), (1,3), (2,1), (2, 4), (3,1), (3,4), (4,3), (4,2)}\

3. Explain adjacency matrix representation of a graph with an example.

Ans: Let G = (V,E) be a graph, where V is set of vertices and E is set of edges. Let N be the number of vertices is a graph G, then the adjacency matrix A of a graph G is defined as,

$$A[i][j] = \begin{cases} 1, \text{ if there is an edge from vertex i to vertex j} \\ 0, \text{ if there is no edge from vertex i to vertex j} \end{cases}$$

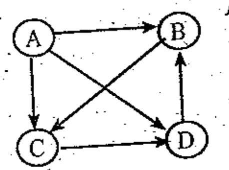
Hence, it is clear from the definition, i.e an adjacency matrix of a graph with "N" vertices is a "boolean N x N square matrix".

For both the directed and undirected graphs,

→ If there is an edge between i and j, then the element is i^{th} row and j^{th} column will be "1".

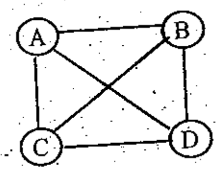
→ If ther is no edge between i and j, then the element is i^{th} row and j^{th} column will be "0".

Eg: Directed graph



$$\begin{array}{c c} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Undirected graph.



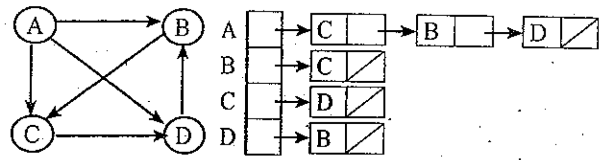
$$\begin{array}{c c} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

4. Explain adjacency lists representation of a graph with example.

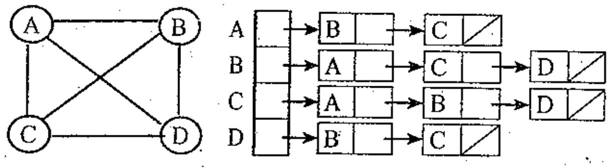
Ans: Let G = (V, E) be a graph, where V is set of vertices and E is set of edges.

An adjacency linked list is an array if "n" linked lists, where "n" is the number of vertices in a graph G. Each location of an array represents a vertex of the graph. For each vertex u Ɛ v, a linked list consisting of all the vertices adjacent to u is created and stored in A [u]. Hence the resulting array A is called as an adjacency list.

Eg: Directed Graph Array.



Undirected graph Arrays.



5. Explain DFS and traverse the given graph using DFS.

Ans: DFS is a method of traversing a graph by visiting each mode of the graph in a systematic order. As the name implies, it means "to search deeper in the graph".

In this, a vertex u is picked as source vertex and is visited. Next a vertex V adjacent to us is picked and visited. The search continues deeper and deeper in the graph until no vertex is adjacent or all the vertices are visited.

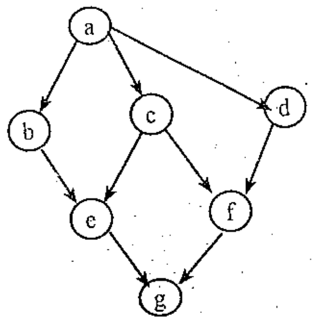
The different types of edges are,

* Tree Edge: During traversal, when a new unvisited vertex say V is reached for the first time from a current vertex say u, then (u,v) is called as a tree edge". It is represented by using solid lines.

* Back Edge: When an already visited vertex say V is reached from the current vertex u, then (u,v) is called as a "back edge". It is represented by using dotted lines.

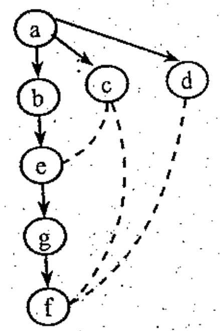
DFS uses a stack data structure which provides LIFO property, Which is useful while traversing the graph.

Method: When a vertex is reached for the first time it is pushed on to the stack, and when a dead end is reached, pop the vertex from the stack. Each vertex is numbered in the order in which it is pushed on to the stack and also numbered in the order in which it is deleted from the stack. Ignore the vertex already visited.



Stack	V = adj (s [Top])	Noes visited	Pop (stack)	o/p T (u,v)
-	-	a	-	-
a_1	b	a,b	-	a-b
a_1, b_2	e	a, b, e,	-	b-e
a_1, b_2, e_3	g	a, b, e, g	-	e-g
a_1, b_2, e_3, g_4	f	a, b, e, g, f	-	g-f
a_1, b_2, e_3, g_4, f_5	-	a, b, e, g, f	$f_{5,1}$	-
a_1, b_2, e_3, g_4	-	a, b, e, g, f	$g_{4,2}$	-
a_1, b_2, e_3	-	a, b, e, g, f	$e_{3,3}$	-
a_1, b_2	-	a, b, e, g, f	$b_{2,4}$	-
a_1	c	a, b, e, g, f, c	-	a-c
a_1, c_6	-	a, b, e, g, f, c	$c_{6,5}$	-
a_1	d	a, b, e, g, f, c, d	-	**a-d**
a_1, d_7	-	a, b, e, g, f, c, d	$d_{7,6}$	-
a_1	-	a, b, c, d, e, f, g	$a_{1,7}$	-

Spanning tree using DFS



6. Explain BFS and traverse the graph using BFS.

Ans: BFS is a method of traversing a graph by visiting each mode of the graph in a systematic order. In this, the graph is traversed in the "order of a level of a vertex" or " breadth - wise level-by-level".

For example, assume u as the start vertex, and it is said to be at level 0. Then in the first stage, all the vertices at level 1, will be visited. In the second stage, vertices at distance 2 will be visited and so on.

At each level, the vertices are visited from left to right. Thus the search continues horizontally or breadth wise. Hence the name BFS.

The different types of edges are,

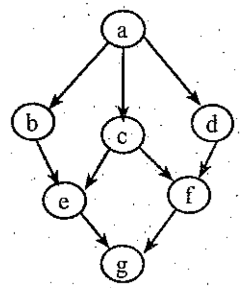
* Tree edge: When a new unvisited vertex say V is reached for the first time from a current vertex say u, then the edge (u, v) is called as a tree edge, which is represented by using solid lines.

* Cross Edge: When an already visited vertex say V is reached from the current vertex u then the edge (u, v) is called as a cross edge, which is represented by using dotted lines.

BFS uses a Queue data structure which provides FIFO property, while traversing the graph.

Procedure:

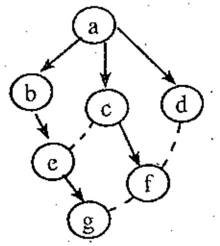
When a vertex is reached for the first time, it is inserted into a rear end of a queue. When a dead end is reached a vertex is deleted from a front end of a queue.



u = del (Q)	v = adj to u	Nodes visited	Queue	o/p (u, v)
-	-	a	a	-
a	b, c, d	a, b, c, d	b, c, d	a-b a-c a-d
b	e	a, b, c, d, e	c, d, e	b-e
c	e,f	a, c, d, e, f, b	d, e, f	c- ef, c-f
d	f	a, c, d, e, f, b	e, f	d - f

c	g	a, b, c, d, e, f, g	f, g	e→g
f	-	a, b, c, d, e, f, g	g	-
g	f	a, b, c, d, c, f, g	-	g- f

Spanning tree using BFS



7. Explain the pseudo code of BFS.

Ans: Step 1: Initialize Q with start vertex and mark this vertex as visited.

Step 2: While queue is not empty.

Delete a vertex u from queue.

Identify all the vertices V adjacent to u.

If the vertices adjacent to u are not visited

then

Mark them as visited

Insert all the marked vertices into queue Q.

output u, v

end if

end while.

8. Explain the pseudocode of DFS.

Ans: Step 1: Select node u as the start vertex and push it on to the stack and mark it as visited.

Step 2: While stack is not empty,

for vertex u on top of the stack

find the next adjacent vertex.

If V is adjacent

If V is not visited, then

push it on to stack and

number it in the order it is pushed.

Mark it as visited by adding V to S.

else

Ignore the vertex

end if

else

Remove the vertex V from the stack

Number it in the order it is popped.

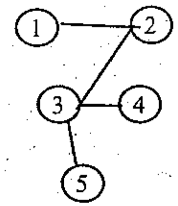
end if

end while

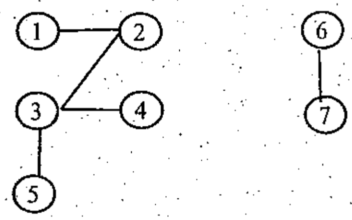
Step 3: Repeat steps 1 and 2 until all the vertices in the graph are traversed.

9. Explain different types of trees with example.

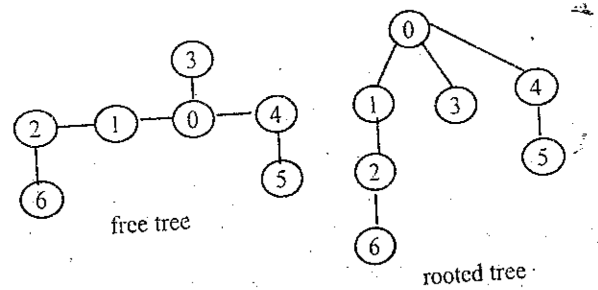
Ans. Tree: A tree which is also called as a free tree, is a connected acyclic graph.



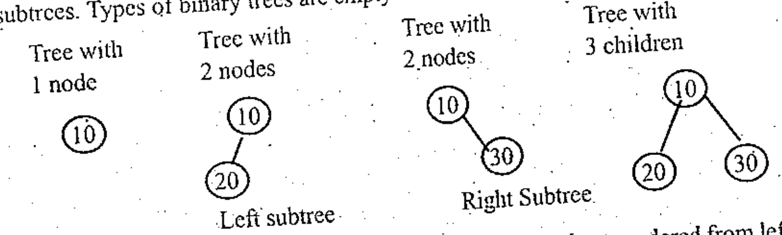
Forest: A collection of one or more trees is called as a forest.



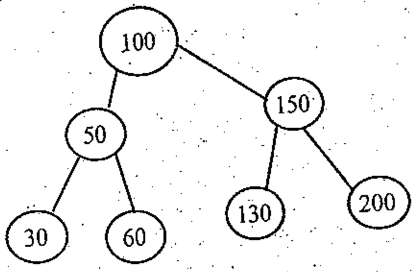
Rooted Tree: A rooted tree is obtained by placing the root at level 0. For this, consider an arbitrary vertex in a free tree as the root of the rooted tree. The vertices adjacent to the root are placed at level 1. The vertices two edges apart from the root are at level 2 and so on.



Binary tree: A node in a binary tree has at most two children. i.e. each node has either zero one or two subtrees. Types of binary trees are empty tree



Ordered Tree: It is a rooted tree in which the children of each node are ordered from left to right.
Binary Search Tree: It is a binary tree in which for each node x in the tree, elements in the left sub tree are less than info(x) and elements in tree are greater or equal to info(x).



10. Explain feasible and optimal solutions with example.

Ans: Feasible Solutions: These are the values of decision variables that simultaneously satisfy all the restrictions of a linear programming (LP) problem. Feasible solutions found in feasible region.

Optimal Solutions: It is the most profitable or the least costly solution that simultaneously satisfies all the constraints of LP problem.

10 MARK QUESTIONS

1. Explain the following.

Ans: (a) Sorting: It is the process of arranging items in some sequence, accordingly it has 2 common yet distinct meaning.

1. Ordering: Arranging items of the same kind, class, nature.

2. Categorizing: Grouping and labelling items with similar properties.

Eg: Arranging numbers, strings, record of students etc. The records can be arranged based on some key information. The different sorting algorithm are,

- Bubble sort.
- Insertion sort.
- Shell sort.
- Merge sort.
- Heap sort.
- Quick sort.
- Counting sort.
- Bucket sort.
- Radix sort.
- Selection sort.

Properties of sorting algorithms are,

1. Stable: If an algorithm preserves the relative order of any 2 equal elements, then it is stable.

Eg: If a[i] and a[j] are at positions i and j, where i < j.

After sorting, a [i] and a [j], are moved to i' and j' where i' < j'. Thenit is stable.

2. Inplace: If an algorithm does not require an extra memory space except for few memory units then it is said to be inplace.

(b). Searching:

The process of finding a particular item in the large amount of data is called as searching. Searching algorithms are,

- Linear search
- Binary search
- Interpolation search

• Hashing etc

No searching technique fits for all the situations. Some algorithms works faster on sorted items, some require extra space, some are complex to understand and some are very easy to understand and so on. Based on the situation, an appropriate technique can be used.

(c). Graph Problems:

A graph is just a data structure that consists of a set of vertices and a set of edges linking vertices.

A tree is a special kind of graph. Graph algorithms operate on a graph and search a graph for path between two nodes or order of vertices etc. These are used to model real life applications such as,

• Transportation and communication networks.

• Scheduling projects and games.

• Web's diameter.

The graph algorithms are,

1. Graph traversal algorithms: BFS and DFS

2. Shortest path problem: Floyd's algorithm, Dijikstra's algorithm, to pological sorting problem.

3. Prim's and Kruskal's algorithm: For constructing minimum spanning tree of a weighted, connected graph.

2. Explain the following with example.

Ans: (a) DFS

Ref. Q5 (5 Marks)

(b) BFS

Ref. Q6 (5 Marks)

3. Define graph. Explain different types of graph representations with example.

Ans: A graph is defined as a set of vertices and Edges, G = (V, E)

where, V - set of vertices

E - set of edges.

There are two types of graph representation.

Adjacency Matrix - Ref. Q3 (5 Marks)

Adjacency Linked List - Ref. Q4 (5 Marks)

4. Write an algorithm of DFS and explain.

Ans: Algorithm: DFS (V).


```
Input: G (V, E) || A graph with set of vertices and edges.
Output: DFS spanning tree
|| G is undirected graph with n vertices.
|| Visited [1 : n] is an array to remember the visited information.
|| u is the starting vertex.
Visited [u] = 1 || Mark the starting vertex as visited
for (each vertex v (adjacent to u))
    if visited [v] = 0 then
        DFS (v)
    end if
end
```

5. Write an algorithm of BFS and explain.

Ans: Algorithm: BFS

```
Input: G = (V, E) || A graph with set of vertices and edges.
Output: BFS spanning tree.
Vis [v] = 1 || Mark the starting vertex as visited.
T = 0 ; || Initialize spanning tree
Initialize Queue ( ) || set front and rear Pointers
while (1)
for (all vertices V adjacent to u)
if (vis [v] = 0) then
    Q _ insert (v) ;
    Vis [v] = 1;
    T = Union (T, <u, v >) ; || add edge to spanning tree
end if
 || (Q _ empty (,)) return T ;
 r = Q _ delete ( ) ;
end for
end while.
```

6. Define graph. Explain different types of graphs with example.

Ans: A graph G is defined as a pair of 2 sets V and E, which is denoted by $G = (V, E)$.

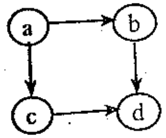
where, V - set of vertices

E - set of edges.

Types of graphs:

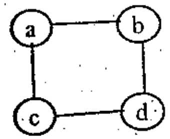
1. Directed graph: A graph $G = (V, E)$, in which every edge is directed is called as a directed graph or diagraph.

Eg:

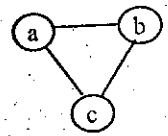


2. Undirected graph: A graph $G = (V, E)$, in which every edge is not directed is called as an undirected graph.

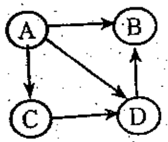
Eg:



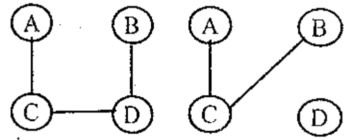
3. Connected graph: In an undirected graph if every vertex is connected to 1 or more vertices is called connected graph.



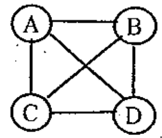
4. Strongly connected graph: In a directed graph, where every pair of vertices or connected by two or more links is called strongly connected graph.



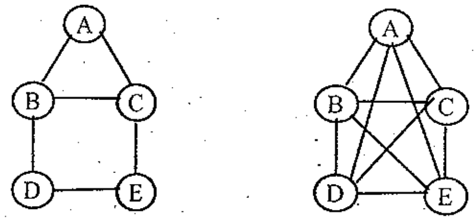
5. Disconnected Graph: It is a graph with no link between a pair of vertices. Also called as connectionless graph.



6. Complete graph: It is a graph in which every vertex is connected to every other vertex.

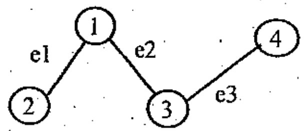


7. Planar Graph: It is the graph, which can be drawn in a plane with no edges crossing.

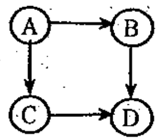


Planar graph Non-Planar graph

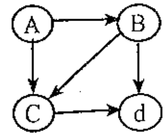
8. Labelled Graph: It is the graph where each vertex or edge is assigned a label or numerical weight.



9. Acyclic Graph: A graph which contains no cycles.

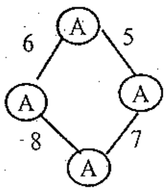


10. Cyclic Graph: A graph which contains atleast one cycle.



11. Weighted Graph: It is a graph. Which contains a number or weight to each edge. Such weights may represent cost, length or capacity etc. The weight of the graph is the sum of weights given to all edges.

weight = 6 + 5 + 7 + 8 = 26



UNIT 3

BRUTE FORCE METHOD

SYLLABUS

3.1 Selection Sort.

3.2 Bubble Sort.

3.3 Sequential Search

3.4 Exhaustive Search.

3.4.1 Travelling Salesman Problem.

3.4.2 o/1 Knapsack problem.

SYNOPSIS

Brute Force Method

It is a straight forward method of solving a given problem based on the problem's statement and definition. The different methods that use brute force technique are selection sort.

It is a straight selection sort or push down sort, in which the smallest element should be found and exchanged with first element, then second smallest should be exchanged with second item in the list and so on.

Bubble Sort: In this method, the adjacent elements in the list are compared and exchanged if they are out of order.

Sequential Search: It is also called as linear search, in which the search element is to be compared with each item in the list sequentially one after the other.

Exhaustive Search: It is a straight forward approach to solve combinatorial problems. It is a time consuming process. This search generates all possible solutions for a problem, from which an optimal solution is selected.

5 MARK QUESTIONS

1. Sort the following numbers using selection sort 5 7 2 9 1.

Unsorted List of elements	After pass 1	After pass 2	After pass 3	After pass 4
5 7 2 9 1	1 7 2 9 5	1 2 7 9 5	1 2 5 9 7	1 2 5 7 9

Sorted of elements are 1 2 5 7 9

2. Sort the following numbers using Bubble sort 20 30 40 10 5.

Unsorted List	Pass 1				Pass 2		
20	20	20	20	20	20	20	20
30	30	30	30	30	30	10	10
40	40	40	10	10	10	30	5
10	10	10	40	5	5	5	30
5	5	5	5	40	40	40	40

	Pass 3		Pass 4	Sorted List is
20	10	10	5	5
10	20	5	10	10
5	5	20	20	20
30	30	30	30	30
40	40	40	40	40

3. Write the algorithm for selection sort.

Ans: Algorithm: Selection sort (A [0.... n-1], n)

Input: Array of n numbers, A

Output: Sorted Array

```
    for i ← 0 to n-1 do
      min ← i
    for j ← i + 1 to n do
      if ( a [j] < a [min]) then
        min ← j
      end if
    end for
    emp ← a [i]
    a [i] ← a [min]
    a [min] ← temp
  end for
end selection sort.
```

4. Write an algorithm for Bubble sort.

Ans: Algorithm: Bubble sort (A [0,.....n-1], n)

Input: Array of n numbers

Output: Sorted array A


```
    for i ← 0 to n-1 do
      for j ← 0 to n-i-1 do
        if (A [j] ≤ A [j + 1]) then
          temp ← A [j]
          A [j] ← A [j + 1]
          A [j + 1] ← temp
    end if
    end for
end Bubble sort
```

5. Explain sequential search algorithm with suitable example.

Ans: In sequential search, the search key element is compared with each element sequentially one after the other in the list. If the element is found the search is successful, else the search is unsuccessful.

Algorithm: Sequential search (A [o0,....n-1], key, n)

Input: Array of n numbers, key

Output: Key found or not found

```
    for i ← 0 to n-1 do
      if (A [i] = key) then
        return i ;
      end if
    return -1 ;
  end for
end sequential search
```

Eg: List = 10 20 30 40 50

Key = 40.

a [0] = 10   10 = = 40

a [1] = 20   20 = = 40

a [2] = 30   30 = = 40

a [3] = 40

a [4] = 50

a [3] = Key

Key found.

6. Explain Travelling salesman problem with suitable example using Brute force technique.

Ans: Given n cities, a salesman starts at a specified city [source], and visiting all n-1 cities only once and returning to the city where he started. The objective is to find a route through the cities that minimizes the cost, there by maximizing the profit.

The graph can be modelled by a directed weighted graph as follows:

1. The vertices of the graph represent various cities.
2. The weights associated with edges represent the distance between 2 cities or the cost while traveling from one city to other.

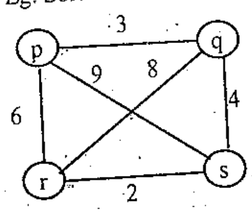
The graph can be solved as follows.

1. Get all the routes from on city to other by taking various permutations.
2. Compute the route length or route cost for each permutation and select the shortest among them.

In this technique, the execution increases very rapidly as the size of the TSP problem increases.

If number of cities equal to n, then the possible routes are (n-1)!

Eg: Solve the following TSP which is represented as a graph.



$$\begin{array}{c|cccc|} & p & q & r & s \\ p & 0 & 3 & 6 & 9 \\ q & 3 & 0 & 8 & 4 \\ r & 6 & 8 & 0 & 2 \\ s & 9 & 4 & 2 & 0 \end{array}$$

Assume that the salesperson starts from city P, and the various routes are,

$p \xrightarrow{3} q \xrightarrow{4} s \xrightarrow{2} r \xrightarrow{6} p = 15$

$p \xrightarrow{3} q \xrightarrow{8} r \xrightarrow{2} s \xrightarrow{9} p = 22$

$p \xrightarrow{6} r \xrightarrow{2} s \xrightarrow{4} q \xrightarrow{3} p = 15$

$p \xrightarrow{6} r \xrightarrow{8} q \xrightarrow{4} s \xrightarrow{9} p = 27$

$p \xrightarrow{9} s \xrightarrow{4} q \xrightarrow{8} r \xrightarrow{6} p = 27$

$p \xrightarrow{9} s \xrightarrow{2} r \xrightarrow{8} q \xrightarrow{3} p = 22$

To get the maximum profit, the route with minimum cost is considered. Hence two routes are selected as

$p \to q \to s \to r \to p$

$p \to r \to s \to q \to p$

7. Explain 0/1 Knapsack with suitable example using Brute.

Ans: Given a knapsack (Bag or container) of capacity M and n objects of weights $w_1, w_2, \ldots, w_n$ with profits, $p_1, p_2, \ldots, p_n$. Let $x_1, x_2, x_3 \ldots x_n$ be the fractions of the objects that are supposed to be added into the knapsack.

The main objective is to place the objects into the knapsack so that maximum profit is obtained and the weights of objects chosen should not exceed the capacity of knapsack.

The problem can be stated as

$$\text{Maximize } \sum_{i=1}^{n} p_i x_i$$

$$\text{Subject to the constraint } \sum_{i=1}^{n} w_i x_i \leq M$$

Eg: Solve the following knapsack problem.

Given $M = 40$

$n = 3$

w1, w2, w3 = {20, 25, 10} which represent weight of 3 objects.

p1, p2, p3 = {30, 40, 35} are the profits of 3 objects.

Solution: The various feasible solutions are,

Maximum weight of Bag = 40.

Objects Selected	Total weight of objects selected	Feasible or Not	Profit Earned
{ }	0	-	0
{1}	20	feasible	30
{2}	25	feasible	40
{3}	10	feasible	35
{1, 2}	20 + 25 = 45	Not feasible	70
{1, 3}	20 + 10 = 30	feasible	65
{2, 3}	25 + 10 = 35	feasible	75
{1, 2, 3}	20 + 25 + 10 = 65	Not feasible	105

The subset which leads to maximum profit should be selected, which is 75 in this problem with {2, 3} set.

10 MARK QUESTIONS

1. Write and explain selection sort algorithm with suitable example.

Ans: Selection sort, as the name indicates first the smallest element should be found and it should be exchanged with first element. Then obtain the second smallest in the list and exchange with second element and so on finally all the elements get sorted.

Algorithm: Ref. Q3. (5 Marks)

Example: Ref. Q1. (5 Marks)

2. Write and explain Bubble sort algorithm with suitable example.

Ans: Bubble sort is one of the simplest and most straight forward method of sorting. In this, the adjacent elements in the list are compared and exchanged if they are out of order.

To sort n elements, n-1 passes are required. The result of the first pass is that the largest element is placed in the last location of an array. Then by the end of second pass the second largest element is placed in the list and so on.

Algorithm: Ref. Q4. (5 Marks)

Example: Ref. Q2. (5 Marks)

3. Explain the following with suitable examples using Brute force technique.

Ans: (a) Travelling Salesman problem.

Ref. Q6. (5 Marks)

(b) 0/1 Knapsack problem.

Ref. Q7. (5 Marks).

UNIT 4

DIVIDE AND CONQUER, DECREASE AND CONQUER

SYLLABUS

Divide and conquer, Decrease and conquer.

4.1 Divide and conquer.

4.1.1 Merge sort.

4.1.2 Quick sort.

4.1.3 Strassen's matrix multiplication.

4.2 Decrease and conquer.

4.2.1 Insertion sort

4.2.1.1 Analysis of Insertion sort.

4.2.1.2 Implementation.

4.2.2 Topological sorting.

SYNOPSIS

4. Divide and conquer, Decrease and conquer.

Divide-and conquer is a top down technique for designing algorithms that consist of dividing the problem into smaller sub-problems hoping that the solutions of the sub-problems are easier to find. The solutions of all smaller problems are then combined to get a solution for the original problem.

Merge Sort: The steps are

1. Divide: Divide the given array of n elements into 2 parts of n/2 elements each.
2. Conquer: Sort the left part and the right part of the array recursively using merge sort.
3. Combine, merge the sorted left part and sorted right part to get a single sorted array.

Quick Sort.

This is another popular type of sorting comes under divide and conquer category.

It is also known as portion exchange sort.

1. Divide: Divide the array A [0] A[1] ----A[n-1] into two sub array.
2. Conquer: Sort the left part of the array recursively and sort the right part of the array recursively.
3. Combine: Since two sub-arrays are in the previous step, no need to combine because the resulting array is sorted.

Strassen's Matrix Multiplication.

It is more efficient than the conventional matrix multiplication for sufficiently large value of n, but not for smaller value of n.

$$T(n)=\begin{cases}1 & \text{if } n=1\\ 7T(n/2) & \text{otherwise}\end{cases}$$

Time complexity = $T(n) = \vartheta(n^{2.807})$

Decrease and Conquer.

The decrease and conquer method can be applied to problems such that they can be solved either by using top down bottom-up (non- recursively).

Insertion Sort

This sorting procedure is similar to the way we play cards. After shuffling the cards each cards will be picked and inserted into the proper place, so that the cards in the hand are arranged in ascending order. The same techniques is being followed while arranging the elements in ascending order.

Topological Sorting.

Topological sort is a directed a cyclic graph G = (vV, E). A topological sort of a graph can be viewed as an ordering of vertices along a horizontal line so that all directed edges go from left to right. For a cyclic graph no linear ordering is possible.

The topological sorting can be done by using 2 methods.

1. DFS Method
2. Source removal Method.

5 MARK QUESTIONS

1. Write the algorithm for Merge sort.

Ans. Merge sort (A, low, high)

```
// Input: A is an unsorted vector with low and high as lower bound and upper bound.
// Output: A is a sorted vector.
        if (low < high)
        mid ← (low + high)/2 // Divide the array into equal parts.
        merge sort (a, low, mid) // Sort the left part of array.
        merge sort (a, mid + 1, high) // Sort the right part of the array
        simple merge (a, low, mid, high) // merge the left part and right part.
end if
simple merge (A, low, mid, high)
// Input: A is sorted from the index position low to mid.
```



```
A is sorted from the index position mid + 1 to high.
// Output: A is sorted from index Low to high
   i ← low, J ← mid + 1, k ← Low.
   while (i < mid and j < = high)
      if (A [i] < A [j]) then
      C [k] ← A [i]  // Copy the lowest element from first part of A to C.
      i ← i + 1 // Point to next item in the left part of A
      k ← k + 1 // Point to next item in C
else.
      C [k] ← A [j] // Copy the lowest element from second part of A to C
      j ← j + 1 // Point to next item in the second part of A.
      k ← k + 1; // Point to next item in C
   end if
end while
while (i < = mid) // Copy the remaining items from left part of A to C
  c [k] ← A [i]
   k ← k + 1, i ← i + 1
end while
   while (j < = high) // Copy the remaining items from right part of A to C
      C [k] ← A [j]
      k ← k + 1, j ← j + 1
      end while
for i = Low to high // Copy the elements from vector ς to vector A.
   a [i] ← c [i]
end for
```

2. Write the algorithm for Quick sort.

Ans. Algorithm Quick sort (A, Low, high)

```
// Input: Low: Position of first element of array
      high: Position of last element of array A
        A: It is an array consisting of unsorted elements.
// Output: Array of sorted elements.
      if (Low < high).
      k ← Partition (A, Low, high)
```

```
        Quick - sort (a, Low, k - 1)
        Quick - sort (A, k + 1, high)
end if
end Quick sort
Algorithm: Partition (A, low, high)
// Input: Low, high, A
// Output: Partitioned array
Pivot ← A [Low]
i ← low
j ← high + 1
while (i < = j)
        do i ← i + 1 while (pivot > = a [i])
        do j ← j-1 while (pivot < a [j])
        if (i < j) exchange (a [i], a[j])
end while
exchange (a[j], a [Low])
return j
end partition
```

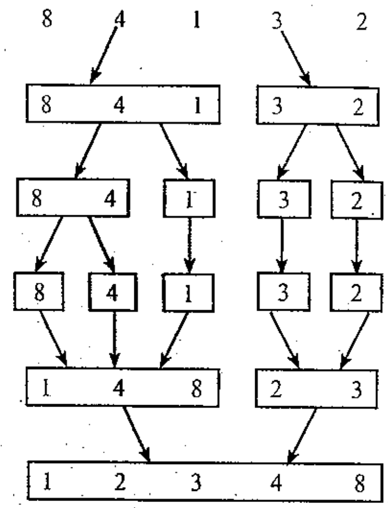
3. Write the algorithm for Insertion sort insertion sort (A, n)

Ans. Algorithm: Insertion-Sort (A, n)

```
Input - A-the list to be sorted.
        n - the total number of elements in the list to be sorted.
// Output - A - the list is sorted.
for i ← 1 to n-1 do
        item ← a [i]
            j ← i-1
   while (j > = 0 and item < a [j])
        a [j + 1] ← a
            j ← j-1
   end while
        a [j + 1] ← item.
end for
```

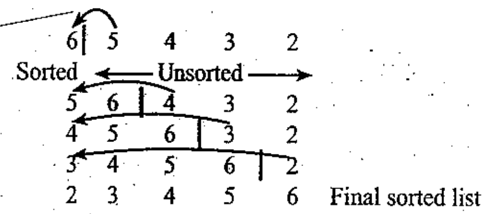

4. Sort the following numbers using merge sort.

Ans.



5. Sort the following numbers using Insertion sort.

Ans.



6. Write the applications of divide and conquer technique.

Ans. 1. Divide and conquer algorithm is used, to improve the detection speed of high interaction client honeypot.

2. In the molecular science.

3. To handle the Big-data traffic using parallel processing in network.

4. Where if the search space is reduced by a constant factor at each step.

7. Write the applications of decrease and conquer technique.

Ans. (i) Tower of Hanoi.

(ii) Generating all n! permutations of {1, 2, – n}.

(iii) Insertion sort.

(iv) Euclid's algorithm.

(v) Binary search and bisection method.

(vi) Topological sorting.

(vii) Graph traversal.

8. Sort the following numbers using Quick sort.

7 4 2 9 1

$C_{worst}(n) = n$

$i \le p$ and $p \le j$

```
7|4  2  9  1      4 ≤ 7 condition true so increment i.
P|i           j

7|4  2  9  1      2 ≤ 7 increment i
P|   i        j

7|4  2  9  1      9 ≤ 7 condition false, so stop and start comparing j with pivot
P|      i     j

7|4  2  9  1      1 ≥ 7 condition false so swap i and j.
P|      i     j

7|4  2  1  9      if i > j; swap pivot with j and split.
P|      i     j

7|4  2  1  9
P|      j     i

1  4  2  7  9
1  4  2|  7  9
```

Then run quick sort recursively with two partitioned arrays.

1 4 2   7 9

After sorting, Merge the two arrays.

1 2 4 7 9

9. Explain divide and conquer technique.

Ans. Divide and conquer is a top-down technique for designing algorithms that consist of dividing the problem into smaller sub problems hoping that the solutions of the sub problems are easier to find. The solutions of all smaller problems are then combined to get a solution for the original problem. This technique involves three steps at each level of the recursion.

Divide: The problem is divided into a number of sub problems.

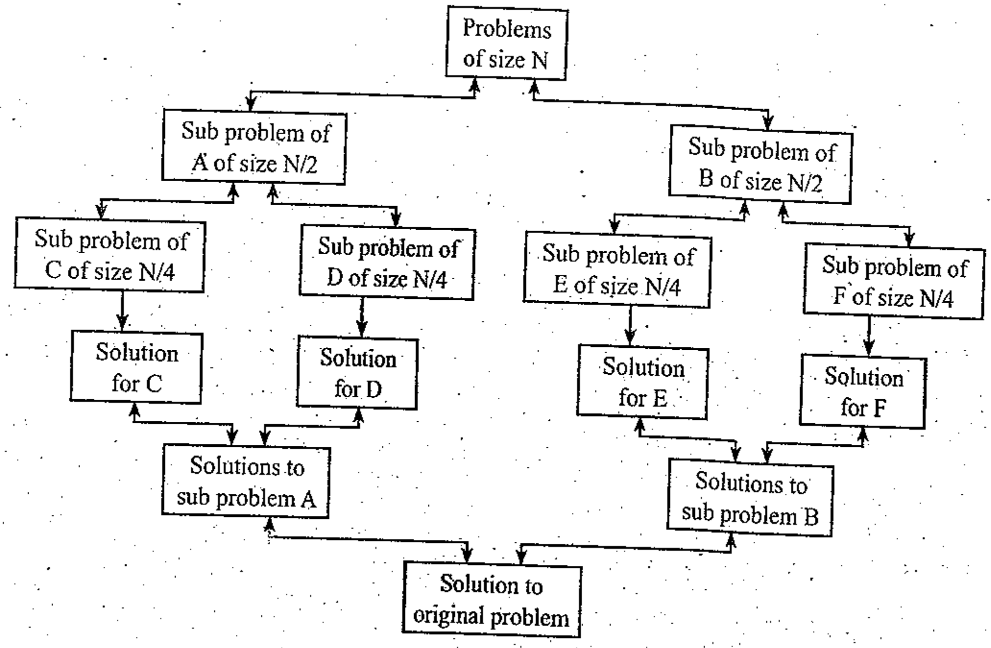
Conquer: The sub problems are conquered by solving them recursively. If the sub problems are smaller in size, the problem can be solved using straight forward method.

Combine: The solution of sub problems are combined to get the solution for the larger problem.

The various techniques using this approach for solving problems are,

• Merge Sort • Quick Sort • Tree Traversals • Binary Search.

Matrix multiplication using Strassesm's algorithm.



10. Explain decrease and conquer technique.

Ans. The decrease and conquer method can be solved applied to problems such that they can be solved either by using top-down (recursively) or bottom-up (non - recursively).

The decrease and conquer is a method of solving a problem by

(i) Changing the problem size from n to smaller size of n-1, n/2 etc. In other words, change the problems from larger instance into smaller instance.

(ii) Conquer (or solve) the problem of smaller size.

(iii) Convert the solution of smaller size problem into a solution for large size problem.

There are three strategies in decrease and conquer method.

(i) Decrease by a constant.

The size of an instance is reduced by the same constant at each iteration of the algorithm.

(ii) Decrease by a constant factor.

The size of a problem instance is reduced by the same constant factor on each iteration of the algorithm.

(iii) Variable size decrease.

A size reduction pattern varies from one iteration to another.

Algorithm: Decrease and Conquer (A [0----n-1])

Input: Array A of size n.

Output: Sum, product etc. Depending upon the problem.

if small ( ) return G ( ) // return the solution of original problem.

n ← n-1 // decrease by one

return decrease and conquer (A);

end decrease and conquer.

10 MARK QUESTIONS

1. Write the formulas and apply Strassen's matrix multiplication for the following.

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \qquad B=\begin{bmatrix}3 & 4\\ 1 & 2\end{bmatrix}$$

$$\begin{bmatrix}C_1 & C_2\\ C_3 & C_4\end{bmatrix}=\begin{pmatrix}A1 & A2\\ A3 & A4\end{pmatrix}*\begin{pmatrix}B1 & B2\\ B3 & B4\end{pmatrix}=\begin{pmatrix}m1+m4-m5+m7 & m3+m5\\ m2+m4 & m1+m3-m2+m6\end{pmatrix}$$

Where m1 = (A1 + A4) * (B1 + B4)

m2 = (A3 + A4) * B1

m3 = A1 * (B2-B4)

m4 = A4 * (B3-B1)

m5 = (A1 + A2) * B4

m6 = (A3-A1) * (B1 + B2)

m7 = (A2-A4) * (B3 + B4)

$$A=\begin{bmatrix}A1 & A2\\ A3 & A4\end{bmatrix} \qquad B=\begin{bmatrix}B1 & B2\\ B3 & B4\end{bmatrix}$$

$$=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} * \begin{bmatrix}3 & 4\\ 1 & 2\end{bmatrix}$$

m1 = (1 + 4) * (3 + 2) = 5 * 5 = 25

m2 = (3 + 4) * 3 = 7 * 3 = 21

m3 = 1 * (4 - 2) = 1 * 2 = 2

m4 = 4 * (1 - 3) = 4 * -2 = -8

m5 = (1 + 2) * 2 = 3 * 2 = 6

m6 = (3 - 1) * (3 + 4) = 2 * 7 = 14

m7 = (2 - 4) * (1 + 2) = -2 * 3 = -6

$$\begin{bmatrix}25+(-8)-6+(-6) & 2+6\\ 21+(-8) & 25+2-21+14\end{bmatrix}=\begin{bmatrix}25-8-6-6 & 8\\ 13 & 20\end{bmatrix}=\begin{bmatrix}5 & 8\\ 13 & 20\end{bmatrix}$$

2. Explain Strassen's matrix multiplication with suitable example.

Ans. Strassen's $\theta(n^{2.807})$

The time complexity of an algorithm to multiply two matrices using brute force method is $\theta(n^3)$. But Strassen V has found out a new method where number of multiplications can be reduced there by increasing the efficiency. This algorithm is called Strassen's matrix multiplication.

Ref. Q3. (10 Marks)

3. Write and explain Quick sort with suitable example using divide and conquer technique.

Ans. Quick sort is another popular type of sorting comes under divide and conquer category. It is also known as partition exchange sort.

(i) Divide: Divide the array A[0], A[1], ---A[n-1] into two sub-array.

$$\underbrace{A[0]\,A[1]\,---\,A[1<-1]}_{\text{all are} \le A[k]} \quad \underset{\text{Pivot Element}}{A[k]} \quad \underbrace{A[k+1]\,A[k+2]\,-\,A[n-1]}_{\text{all are} \ge A[k]}$$

With respect to A[k] the elements are divided into two parts. So A[k] is called pivot element.

(ii) Conquer: Sort the left part of the array recursively, sort the right part of the array recursively.

(iii) Combine: Since two sub arrays are sorted in the previous step, no need to combine and the resulting array is sorted.

Example

In quick sort algorithm pick an element from array of elements. This element is called pivot. Then compare the values from left to right until a greater element is found then swap the values.

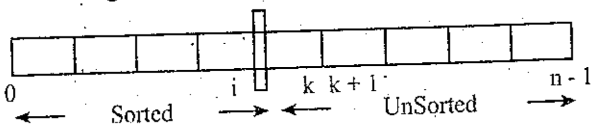
Again start comparison from right with pivot. When lesser element is found then swap the values. The same steps will be followed until all the elements which are less than the pivot come before the pivot and all elements greater than the pivot come afferit.

This is called partition operation. The sub-array of lesser and greater elements will be sorted recursively.

Example Ref. Q8. (5 Marks)

4. Write and explain. Insertion sort with suitable example using decrease and conquer technique.

Ans. The sorting procedure is similar to the way we play cards. After shuffling the cards, each card will be picked and inserted the it into the proper place so that cards in hand are arranged in ascending order. The same technique is being followed while arranging the elements in ascending order. The given list is divided into two parts. Sorted part and unsorted part.



All the elements from 0 to i are sorted and elements from K to n-1 are not sorted. The Kth item can be inserted into any of the positions from o to i so that elements towards left of boundary are sorted. As each item is inserted towards sorted left part, the boundary moves to the right decreasing the unsorted list. Once the boundary moves to the right most position, the elements towards the left of boundary represent the sorted list.

Example, Ref Q(5) (5 Marks)

5. Explain Merge sort with suitable example using divide and conquer technique.

Ans. The various steps that are involved while sorting using merge sort are.

(i) Divide: Divide the given array of n elements into 2 parts of n/2 elements each.

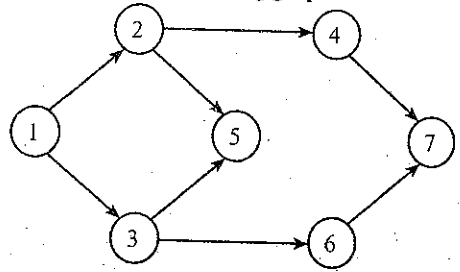
(ii) Conquer: Sort the left part and the right part of the array recursively using merge sort.

(iii) Combine: Merge the sorted left part and sorted right part to get a single sorted array.

The key operation in merge sort is combining the sorted, left part and sorted right part into a single sorted array. This process of merging two sorted vectors into a single sorted vector is called simple merge sort. The only necessary condition for this problem is that both arrays should be sorted.

Eg: Ref. Q4. (5 Marks)

6. Obtain Topological ordering for the following graph.



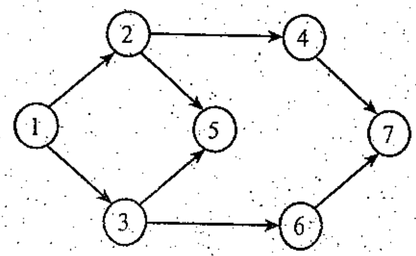
(i) Apply DFS based algorithm to solve the topological sorting problem for a graph.

Stack	Adj. Nodes	Nodes Visited	Pop
-	-	1	-
1	2	1,2	-
1,2	4	1,2,4	-
1,2,4	7	1,2,4,7	-
1,2,4,7	-	1,2,4,7	7
1,2,4	-	1,2,4,7	4
1,2	5	1,2,4,7,5	-
1,2,5	-	1,2,4,7,5,	5
1,2	-	1,2,4,7,5	2
1	3	1,2,4,7,5	-
1,3	6	1,2,4,7,5	-
1,3,6	7	1,2,4,7,5, 3	6
1,3	5	1,2,4,7,5,3,6	3
1	-	1,2,4,7,5,3,6	1

In DFS that, the topological order is the reverse of pop out order

$1 \rightarrow 3 \rightarrow 6 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7$

(ii) Topological sort using source removed method.



The adjacency matrix is,

$$\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline & 0 & 1 & 1 & 1 & 2 & 1 & 2 \end{array} \rightarrow \text{Sum of columns i.e. In degrees of each node.}$$

Note: Every time the adjacent node is found, then respective in degree of that node is decreased by one, till it becomes zero.

Stack	M = Pop ( )	Solution T	V-adj (u)	In degree of nodes [1] [2] [3] [4] [5] [6] [7]
1	1	1	2,3	0 1 1 1 2 1 2 0 0 0 1 2 1 2 0 0 0 1 1 0 2
2,3	3	1,3	5,6	
E,2,6	6	1,3, 6	7	0 0 0 1 1 0 1
2,	2	1,3, 6, 2	4,5	0 0 0 0 0 0 1
4,5	5	1,3,6, 2,5	-	-
4	4	1,3,6, 2,5,4	7	0 0 0 0 0 0 0
7	7	1,3,6,2,5, 4,7	-	-

Thus the order is $1 \rightarrow 3 \rightarrow 6 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	25	25
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	24	24	25	25	25
3	0																				

UNIT 5

DYNAMIC PROGRAMMING, GREEDY TECHNIQUE

SYLLABUS

5. Dynamic Programming, Greedy Technique.
5.1 Dynamic Programming.
5.1.1. Warshall's algorithm.
5.1.2 Floyd's algorithm.
5.1.3 Oll Knapsack problem.
5.2 Greedy Technique.
5.2.1 Prim's algorithm.
5.2.2 Kruskal's algorithm.
5.2.3 Dijikstra's algorithm.

SYNOPSIS

Dynamic programming is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping substances. Here programming means "Planning".

Warshall's Algorithm.

Floyd - Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights. (but with no negative cycle)

$$P[i,j] = \begin{cases} 1 & \text{if path exists from } i \text{ to } j \\ 0 & \text{else} \end{cases}$$

Floyd's Algorithm.

In all pairs shortest path problem the shortest distance from all nodes to all other nodes should be found.

D [i, j] = min (D[i, j], D[i, k] + D[k, j])

0/1 Knapsack Problem.

Given a knapasack with following.

m - Capacity of the knapsack.

n - number of objects.

w - An array consisting of profits

p1, p2, ----pn

x - An array consisting of either 0 or 1.

A o in x_i represent ith object has not been selected and a 1 in x_i i^{th} object has

The main objective is to place the objects into the knapsack so that maximum profit is obtained and the weights of object chosen should not exceed the capacity of knapsack.

$$V[i,j] = \begin{vmatrix} 0 & \text{if } i = j = 0 \\ v[i-a, j & \text{if } w_i > j \end{vmatrix}$$

max (v[i-1, j], v[i-1, j - w_i] + p_i) if wi ≤ j

Greedy Technique.

A greedy algorithm is an algorithm that always tries to find the best solution for each sub problem.

Prim's Algorithm.

This algorithm is used to find minimum spanning tree. A spanning tree is a tree in which all nodes are connected without forming a circuit.

Kruskal's Algorithm.

Kruskal's algorithm is another algorithm of obtaining minimum spanning tree. In this the minimum cost edge has to be selected. It is not necessary that selecting optimum edge is adjacent.

Dijikstra's Algorithm.

To find shortest distance from given source to destination.

5 MARK QUESTIONS

1. Write a note on Dynamic Programming.

Ans. It is a method of solving the problem with overlapping sub problems

It works by dividing the problems into sub-problems and getting the solution for the sub problems using which solution for the given problem can be obtained.

Once a sub problems is solved the result is stored in a table and never recalculated.

The various problems that uses this concept are fibonacci number, Warshall's algorithm, computing binomial co-efficient.

Dynamic Programming uses bottom-up approach. It is more efficient because re-computations are not done.

2. Write the Warshall's algorithm.

Ans. Algorithm Warshall (n, A, P)

```
// Input: Adjacency matrix A.
// Output: Transitive closure.
   Step 1: [make a copy of adjacency of matrix].
        for i ← 0 to n-1 do.
        for j ← 0 to n-1 do
        p [i, j] = A [i, j]
        end for
end for
Step 2: [Find the transitive closure. (path matrix)].
        for k ← 0 to n-1 do.
            for i ← 0 to n-1 do.
                for j ← 0 to n-1 do.
if (P [i, j] = 0 and if (P [i, k] = 1 and P [k, j] = 1) then.
                P [i, j] = 1
        end if
      end for
   end for
end for.
Step 3: Return n.
```

3. Write the Floyd's algorithm.

Ans. Algorithm: Floyd (n, cost, D)

```
// Input: Cost adjacency matrix of size n x n.
// Output: Shortest distance matrix of size n x n.
        for i ← 0 to n-1 do.
            for j ← 0 to n-1 do.
        D [i, j] = cost [i, j]
      end for
end for
```

```
for k ← 0 to n-1 do.
    for i ← 0 to n-1 do.
        for j ← 0 to n-1 do.
D [i, j] = min (D [i, j], D[i, k] + D[k, j])
        end for.
    end for.
end for.
```

4. Write the Prim's algorithm.

Ans. . Algorithm: Prim's (n, w)

```
//Input n: Number of vertices in a graph
    w: Cost adjacency matrix.
//Output d: Shortest distance from source to all other nodes.
    p: Shortest path from source to destination.
    s: Gives the nodes that are so for visited and the nodes that are not visited.
for i ← 0 to n-1 do.
    s [i] = 0
    d [i] = w [src, i]
p [i] = src.
end for
S [src] = 1    // add src to S.
sum ← 0 // Initial cost of min spanning tree.
    k ← 0 // used as index to share the edges selected.
for i ← 0 to n-1 do.
find u and d[u] such that d [u] is minimum and u ∑ v-s
Add u to s.
Select an edge with least cost
Add cost associated with edge to get total cost of min spanning tree.
for every V ∑ v-s do
    if (w [u, v] < d[v])
        d [v] = w [u, v]
        p [v] = u
    end if
```



```
    end for
end for
if (sum > = 9999)
Write " spanning tree does not exist"
else
Write "spanning tree exist and print the min spanning tree"
end if
```

5. Write the Kruskal's algorithm.

Ans. Algorithm: Kruskal (n, m, E)

```
// Input: n - number of vertices in the graph.
        m - number of edges in the graph.
        E - edge list consisting of set of edge along with equiralent weights.
// Output: Minimum spanning tree (MST).
Count ← 0      //Initial nor of edges.
    K ← 0      // Points to the first selected edge of MST.
sum ← 0        // Initial cost of MST.
// Create a forest with n vertices.
        for i ← 0 to n do.
            parent [i] ← i
end for
while (count ≠ n-1 and E ≠ ϕ)
Select an edge (u, v) with least cost
i ← find (u, parent)   // Find the root for vertex u
j ← find (v, parent)...// Find the root for vertex v
if (i! = j)            // If roots of u and v are different.
// Select the edge (u, v) as the edge of MST.
        t [k] [0] ← u
        t [k] [1] ← v
        k + +
        Count + + // update no of edges selected for MST
        sum ← sum + cost (u, v) // update the cost of MST
        union (i, j, parent); // Merge the two trees with roots i and j.
```

```
        end if
// Delete the edge (u, v) from the list.
end while.
if (count ! = n-1).
Write " spanning tree does not exist";
return
end if
Write "spanning tree is shown below."
    for i ← 0 to n-2 do.
    write [t [i] [0],   t[i] [1]]
end for
Write "cost of spanning tree is", sum.
```

6. Write the Dijkstra's algorithm.

Ans. Algorithm: Dijkstra (n, w, source, destination, D, P)

```
//Input n: nor of vertices in the graph.
        w: cost adjacency matrix with values.
        src: source vertex.
        destn: Destination vertex.
// Output  D: Shortest distance from source to all other nodes.
           P: Shortest path from source to destination.
           S: Gives the nodes that are so for visited and the nodes that are not visited.
S [src] = 1     // add  src to S.
for i ← o to n-1 do.
find u and d[u] such that d[u] is minimum and
    u ∑ v-s
    add u to s.
    if (u – destination) break;
for every v ∑ v-s do (i.e for v = 0 to n-1)
        if (d[u] + w[u, v] < d[v])
          d[v] = d [u] + w[u, v]
            p[v] = u
      end if
```



```
    end for
end for
```

7. Write the applications of Dynamic Programming.

Ans. 1. System optimization of environmental problem.

2. The water resources allocation problem.

3. The optimum scheme problem of water treatment.

4. The shortest path problem.

8. Write the applications of Greedy technique.

Ans. 1. Dijkstra's algorithm for finding short path to all nodes given a start node.

2. Prim's algorithm for finding a minimum spanning tree.

3. Huffman trees for data compression.

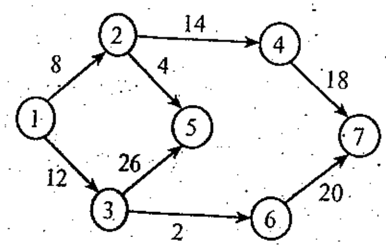
4. Travelling salesman problem.

5. Combinatorial problems (mathematical optimization).

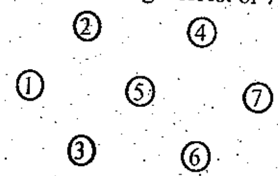
6. Graph-coloring problem.

9. Obtain the minimum cost spanning tree for the graph in figure using Kruskal's algorithm.

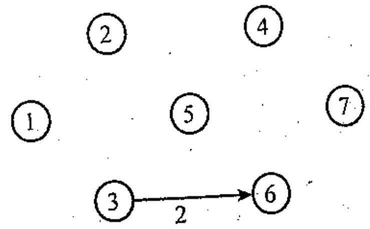
Ans.



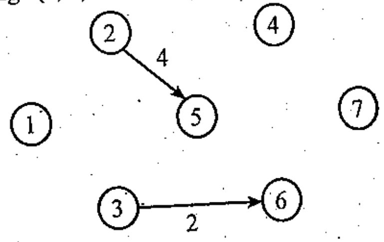
In the beginning no edge is selected thus forming a forest of 7 trees.



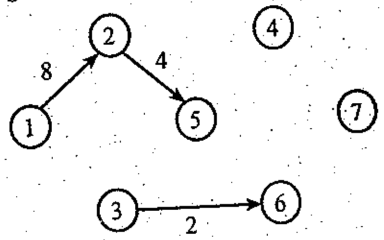
Step 1: The cost of the edge. (3, 6) is least and hence it is selected. Thus forming trees for the trees 3 and 6, node 3 is the root.



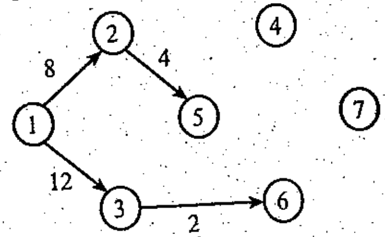
Step 2: The next least cost edge (2, 5).



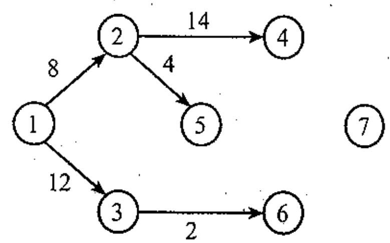
Step 3: The next least cost edge is (1, 2).



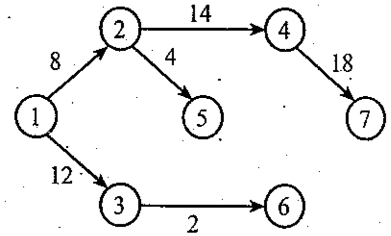
Step 4: The next least cost edge is (1, 3).



Step 5: The next least cost edge is (2, 4).



Step 6: The next least cost edge is (4, 7)



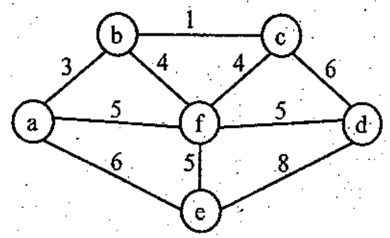
Hence, all the nodes are visited.

The cost of this minimum spanning trees is given by the sum of the weights of the selected edge i.e. $8 + 12 + 2 + 4 + 14 + 18 = 58$

10 MARK QUESTIONS

1. Obtain the minimum cost spanning tree for the following graph using Prim's.

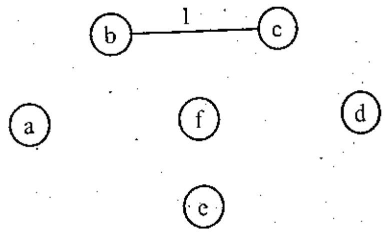
Ans.



The cost adjacency matrix is.

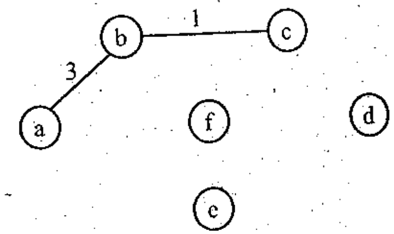
	a	b	c	d	e	f
a	0	3	∞	∞	6	5
b	3	0	1	∞	∞	4
c	∞	1	0	6	∞	4
d	∞	∞	6	0	8	5
e	6	∞	∞	8	0	5
f	5	4	4	5	5	0

Step 1: The least cost is 1 and hence the edge (b, c) is considered and src = b



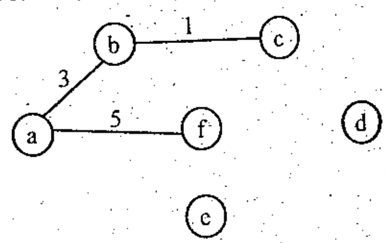
Total weight = 1

Step 2: The next least cost is 3 and hence the edge (b, a) is considered.



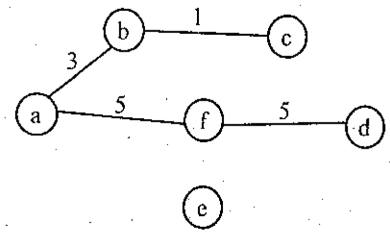
Total weight = 4

Step 3: The next least cost is 5 and hence the edge (a, f) is selected.



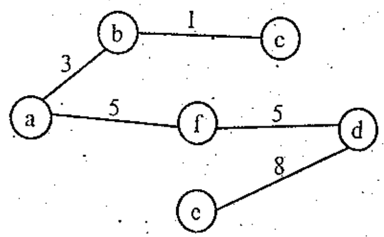
Total weight = 9

Step 4: The next least cost is 5 and hence the edge (f,d) is selected.



Total weight = 14

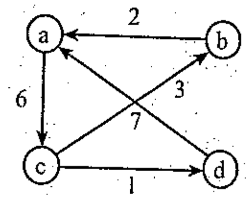
Step 5. The next least cost is 8 and hence the edge (d, e) is selected.



Total weight = 22

Note: The next least cost edge should be selected with the previous edge nodes

2. Obtain all pairs shortest path for following graph.



	a	b	c	d
a	0	∞	6	∞
b	2	0	∞	∞
c	∞	3	0	1
d	6	∞	∞	0

Formula:

D[i, j] = min [D (i, j), D(i, k) + D(k, j)]

Step 1:

	a	b	c	d
a	0	∞	6	∞
b	2	0	∞	∞
c	∞	3	0	1
d	6	∞	∞	0

$$\left.\begin{matrix}(b, a) = 2 \\ (a, c) = 6\end{matrix}\right\} = (b,c) = \infty$$

= min [(b, c), (b, a) + (a, c)]

= min [∞, (2 + 6)]

(b, c) = 8

$\boxed{(b, c) = 8}$

Step 2:

	a	b	c	d
a	0	∞	6	∞
b	2	0	8	∞
c	∞	3	0	1
d	6	∞	∞	0

$$\left.\begin{matrix}(c, b) = 3 \\ (b, a) = 2\end{matrix}\right| = (c,a) = \infty$$

$= \min[\infty,(3+2)]$

$= \min(\infty,5)$

$\boxed{(c,a) = 5}$

$$\left.\begin{matrix}(c,b) = 3 \\ (b,c) = 8\end{matrix}\right| = (c,c) = 0$$

$= \min[1,(3+8)]$

$= \min(0,11)$

$\boxed{(c,c) = 0}$

Step 3:

	a	b	c	d
a	0	∞	6	∞
b	2	0	8	∞
c	5	3	0	1
d	6	∞	∞	0

$$\left.\begin{matrix}(a,c) = 6 \\ (c,a) = 5\end{matrix}\right| = (a,a) = 0$$

(c,a) = 5

$= \min[0,(6+5)] = 0$

$= \min(0,11)$

$\boxed{(a,a) = 0}$

$$\left.\begin{matrix}(a,c) = 6 \\ (c,b) = 3\end{matrix}\right| = (a,b) = \infty$$

$= \min[\infty,(6+3)]$

$= \min(\infty,9)$

$\boxed{(a,b) = 9}$

$$\left.\begin{matrix}(a,c)=6 \\ (c,c)=0\end{matrix}\right| = (a,c) = 6$$

$= \min[6,(6+0)]$

$= \min(6,6)$

$\boxed{(a,c) = 6}$

$$\left.\begin{matrix}(a,c)=6\\(c,d)=1\end{matrix}\right| = (a,d) = \infty$$

$= \min[\infty,(6+1)]$

$= \min(\infty,7)$

$\boxed{(a,d)=7}$

$$\left.\begin{matrix}(b,c)=8\\(c,a)=5\end{matrix}\right| = (b,a) = 2$$

$= \min[2,(8+5)]$

$= \min(2,13)$

$\boxed{(b,a)=2}$

$$\left.\begin{matrix}(b,c)=8\\(c,b)=3\end{matrix}\right| = (b,b) = 0$$

$\min[0,(8+3)]$

$= \min(0,11)$

$\boxed{(b,b)=0}$

$$\left.\begin{matrix}(b,c)=8\\(c,c)=0\end{matrix}\right| = (b,c) = 8$$

$= \min[8,(8+0)]$

$= \min(8,8)$

$\boxed{(b,c)=8}$

$$\left.\begin{matrix}(b,c)=8\\(c,d)=1\end{matrix}\right| = (b,d) = \infty$$

$= \min[\infty,(8+1)]$

$= \min(\infty,9)$

$\boxed{(b,d)=9}$

Step 4:

	a	b	c	d
a	0	9	6	7
b	2	0	8	9
c	5	3	0	1
d	6	∞	∞	0

$$\left.\begin{matrix}(a,d)=7\\(d,a)=6\end{matrix}\right| = (a,a) = 0$$

$= \min[0,(7+6)]$

$= \min(0,13)$

$\boxed{(a,a)=0}$

$$\left.\begin{matrix}(a,d)=7\\(d,d)=0\end{matrix}\right| = (a,d) = 7$$

$= \min[7,(7+0)]$

$= \min(7,7)$

$\boxed{(a,d)=7}$

$$\left.\begin{matrix}(b,d)=9\\(d,a)=6\end{matrix}\right| = (b,a) = 2$$

$= \min[2,(9+6)]$

$= \min(2,15)$

$\boxed{(b,a)=2}$

$$\left.\begin{matrix}(b,d)=9\\(d,d)=0\end{matrix}\right| = (b,d) = 9$$

$= \min[9,(9+0)]$

$= \min(9,9)$

$\boxed{(b,d)=9}$

$$\left.\begin{matrix}(c,d)=1\\(d,a)=6\end{matrix}\right| = (c,a)=5$$

$= \min[5,(1+6)]$

$= \min(5,7)$

$\boxed{(c,a)=5}$

$$\left.\begin{matrix}(c,d)=1\\(d,d)=0\end{matrix}\right| = (c,d)=1$$

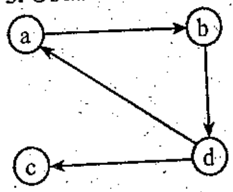
$= \min[1,(1+0)]$

$= \min(1-1)$

$\boxed{(c,d)=1}$

	a	b	c	d
a	0	9	6	7
b	2	9	8	9
c	5	3	0	1
d	6	∞	∞	0

3. Obtain the transitive closure for the following graph.



Adjacency matrix.

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

The pair i, j is made 1 if and only if (i, k) = 1 and (k, j) = 1

Step 1: Consider 1st row and col

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

$$\left.\begin{matrix}(d,a)=1\\(a,b)=1\end{matrix}\right| = (d,b)=1$$

Step 2:

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	1	1	0

$$\left.\begin{matrix}(a,b)=1\\(b,d)=1\end{matrix}\right| = (a,d)=1$$

$$\left.\begin{matrix}(d,b)=1\\(b,d)=1\end{matrix}\right| = (d,d)=1$$

Step 3:

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

As, all the values are zero, hence no change.

Step 4

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

$$\left.\begin{array}{l}(a,d)=1\\(d,a)=1\end{array}\right|=(a,a)=1$$

$$\left.\begin{array}{l}(a,d)=1\\(d,b)=1\end{array}\right|=(a,b)=1$$

$$\left.\begin{array}{l}(a,d)=1\\(d,c)=1\end{array}\right|=(a,c)=1$$

$$\left.\begin{array}{l}(a,d)=1\\(d,d)=1\end{array}\right|=(a,d)=1$$

$$\left.\begin{array}{l}(b,d)=1\\(d,a)=1\end{array}\right|=(b,a)=1$$

$$\left.\begin{array}{l}(b,d)=1\\(d,b)=1\end{array}\right|=(b,b)=1$$

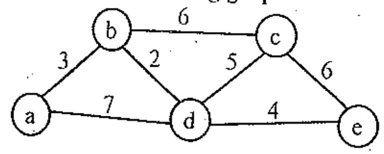
$$\left.\begin{array}{l}(b,d)=1\\(d,c)=1\end{array}\right|=(b,c)=1$$

$$\left.\begin{array}{l}(b,d)=1\\(d,d)=1\end{array}\right|=(b,d)=1$$

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

Final matrix obtained.

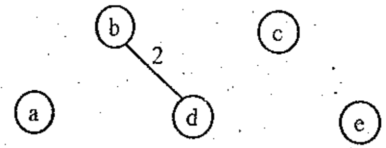
4. Find the single shortest path for the following graph.



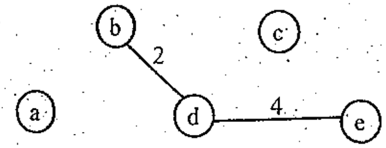
cost adjacency matrix is,

$$\begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 3 & \infty & 7 & \infty \\ b & 3 & 0 & 6 & 2 & \infty \\ c & \infty & 6 & 0 & 5 & 6 \\ d & \infty & \infty & 5 & 0 & 4 \\ e & \infty & \infty & 6 & 4 & 0 \end{array}$$

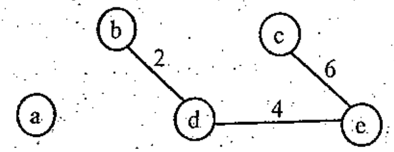
1. Consider the shortest distance in the graph. Select b to d0 with distance 4 with minimum distance 2 vertex



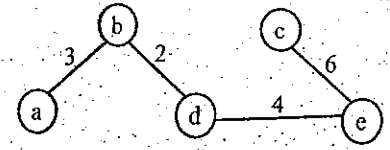
2. d to e selected with distance 4. Go to e through d. Vertex e is selected with distance 6.



3. Select e to c with distance 6. Go to c through e. Vertex C is selected with distance 12



4. Select through b to a with distance 3 vertex a is selected from b with distance 3.



Thus, the shortest distance is

a→b→d→e→c

3 + 2 + 4 + 6 = 15

5. Solve he following instance for 0/1 knapsack problem using Dynamic Programming.

n = 4, P = [4, 2, 1, 8] w = [3, 1, 7, 9] m = 10

i	w	p
1	3	4
2	1	2
3	7	1
4	9	8

Step 1: n = 4, m = 10

Matrix: consists of n + 1 rows, j and m + 1 col s

i \ j	0	1	2	3	4	5	6	7	8	9	10
0											
1											
2											
3											
4											

i =0, j=0, v[i, j] = 0 if i = j = 0

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										
4	0										

0 if i = j = 0

v [i, j] = v [i - 1, j] if wi > j

max (v [i - 1, j], v[i - 1, j - wi] + pi) if wi ≤ j

Step 2: i = 1, j = 1......10, w = 3, p = 4.

w	i	j	
3	1	1	v [1-1,,] = v[0,1] = 0
3	1	2	v [0, 2]
3	1	3	max (v[0, 3], v[0, 0] + 4) = 4
3	1	4	max (v[0.4], v[0, 1] + 4) = 4
3	1	5	max (v [0, 5], v [0, 2] + 4) = 4
3	1	6	max (v [0, 6], v [0, 3] + 4) = 4
3	1	7	max (v [0,7], v [0, 4] + 4) = 4
3	1	8	max (v [0, 8], v [0, 5] + 4) = 4
3	1	9	max (v [0, 9], v [0, 6] + 4) = 4
3	1	10	max (v [0, 10], v [0, 7] + 4) = 4

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	4	4	4	4	4
2	0										
3	0										
4	0										

Step 3: i = 2, w = 1, p = 2, j = 1.........,10

w	i	j	
1	2	1	max (v [1, 1], v [1, 0] + 2) = 2
1	2	2	max (v [1, 2], v [1, 1] + 2) = 2
1	2	3	max (v [1, 3], v [1, 2] + 2) = 4
1	2	4	max (v [1, 4], v [1, 3] + 2) = 6
1	2	5	max (v [1, 5], v [1, 4] + 2) = 6
1	2	6	max (v [1, 6], v [1, 5] + 2) = 6
1	2	7	max (v [1, 7], v [1, 6] + 2) = 6
1	2	8	max (v [1, 8], v [1, 7] + 2) = 6
1	2	9	max (v [1, 9], v [1, 8] + 2) = 6
1	2	10	max (v [1, 10], v [1, 9] + 2) = 6

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	4	4	4	4	4
2	0	2	2	4	6	6	6	6	6	6	6
3	0										
4	0										

Step 4: i = 3, w = 7, p = 1, j = 1, 2, ...10

w	i	j	
7	3	1	v [2, 1] = 2
7	3	2	v. [2, 2] = 2
7	3	3	v [2, 3] = 4
7	3	4	v [2, 4] = 6
7	3	5	v [2, 5] = 6
7	3	6	v [2, 6] = 6
7	3	7	v [2, 7] = 6
7	3	8	v [2, 8] = 6
7	3	9	v [2, 9] = 6
7	3	10	v [2, 10] = 6

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	4	4	4	4	4
2	0	2	2	4	6	6	6	6	6	6	6
3	0	2	2	4	6	6	6	6	6	6	6
4	0										

Step 5: i = 4, w = 9, p = 8, j = 1, 2, ...10

w	i	j		
9	4	1	v [3, 1] = 2	↓ Thus, optimal solution v [n, m] = v [4, 10] = 6
9	4	2	v [3, 2] = 2	
9	4	3	v [3, 3] = 4	
9	4	4	v [3, 4] = 6	
9	4	5	v [3, 5] = 6	
9	4	6	v [3, 6] = 6	
9	4	7	v [3, 7] = 6	
9	4	8	v [3, 8] = 6	
9	4	9	v [3, 9] = 6	
9	4	10	v [3, 10] = 6	

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	4	4	4	4	4
2	0	2	2	4	6	6	6	6	6	6	6
3	0	2	2	4	6	6	6	6	6	6	6
4	0	2	2	4	6	6	6	6	6	6	6

Profit table. Optimal solution

Hence, optimal solution is 6.

Note: For each step v[i, j] values, consider the previous steps matrix values.

6. Solve the following instance for 0/1 Knapsack problem using dynamic programming.

n = 3, p = [25, 24, 15] w = [18, 15, 10] m = 20

n = 3

m = 20

i	w	p
1	18	25
2	15	24
3	10	16

0 if i = j = 0

v [i, j] = v [i-1, j] if wi > j

max (v [i-1, j], v [i-1, j-wi] + pi) if wi ≤ j)

Step 1: Consider the matrix with n + 1 rows and m + 1 columns.

i ↓ \ j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0																					
1																					
2																					
3																					

Step 2: i = 0, j = 0

v [i, j] = 0 if i = j = 0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
↓1	0																				
2	0																				
3	0																				

Step 3: $p_i = 25$, $w_1 = 18$ i = 1, j = 1,............,20

v [i, j] = v [i-1, j] if $w_i > j$

18 > 0

j	i	w_i	
0	1	18	v [0, 0] = 0
1	1	18	v [0, 1] = 0
2	1	18	v [0, 2] = 0
3	1	18	v [0, 3] = 0
4	1	18	v [0, 4] = 0
5	1	18	v [0, 5] = 0
6	1	18	v [0, 6] = 0
7	1	18	v [0, 7] = 0
8	1	18	v [0, 8] = 0
9	1	18	v [0, 9] = 0
10	1	18	v [0, 10] = 0
11	1	18	v [0, 11] = 0
12	1	18	v [0, 12] = 0
13	1	18	v [0, 13] = 0
14	1	18	v [0, 14] = 0
15	1	18	v [0, 15] = 0
16	1	18	v [0, 16] = 0
17	1	18	v [0, 17] = 0
18	1	18	max (v [0, 18], v [0, 0] + 25) = v [0, 25] = 25
19	1	18	max (v [0, 19], v [0, 1] + 25) = 25
20	1	18	max (v [0, 20], v[0, 2] + 25) = 25

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	25	25
2	0																				
3	0																				

Step 4: i = 2, w = 15, p = 24, j = 1, ...20

i	i	w	p = 24
2	1	15	v [i-1, j] = v [1, 1] = 0
2	2	15	v [1, 2] = 0
2	3	15	v [1, 3] = 0
2	4	15	v [1, 4] = 0
2	5	15	v [1, 5] = 0
2	6	15	v [1, 6] = 0
2	7	15	v [1, 7] = 0
2	8	15	v [1, 8] = 0
2	9	15	v [1, 9] = 0
2	10	15	v [1, 10] = 0
2	11	15	v [1, 11] = 0
2	12	15	v [1, 12] = 0
2	13	15	v [1, 13] = 0
2	14	15	v [1, 14] = 0
2	15	15	max [(1, 15) v (1, 0) + 24] = 24 max [0 , 0 + 24]=24
2	16	15	max [(1, 16) v (1, 1) + 24] = 24 max [0 , 0 + 24]=24
2	17	15	max [(1, 17) v (1, 2) + 24] = 24 max [0 , 0 + 24]=24
2	18	15	max [(1, 18) v (1, 3) + 24] = 25 max [25 , 0 + 24]=25
2	19	15	max [(1, 19) v (1, 4) + 24] = 25 max [25 , 0 + 24]=25
2	20	15	max [(1, 20) v (1, 5) + 24] = 25 max [25 , 0 + 24]=25

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	25	25
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	24	24	25	25	25
3	0																				

Step 5: i = 3, w = 10, p = 15, j = 1,......20

w	i	j	p = 15
10	3	1	v [2, 1] = 0
10	3	2	v [2, 2] = 0
10	3	3	v [2, 3] = 0
10	3	4	v [2, 4] = 0
10	3	5	v [2, 5] = 0
10	3	6	v [2, 6] = 0
10	3	7	v [2, 7] = 0
10	3	8	v [2, 8] = 0
10	3	9	v [2, 9] = 0
10	3	10	max [v (2, 10), v(2, 0) = 15] = 15 max [0 , 0+15]=15
10	3	11	max [v (2, 11), v(2, 1) = 15] = 15 max [0 , 0+15]=15
10	3	12	max [v (2, 12), v(2, 2) = 15] = 15 max [0 , 0+15]=15
10	3	13	max [v (2, 13), v(2, 3) = 15] = 15 max [0 , 0+15]=15
10	3	14	max [v (2, 14), v(2, 4) = 15] = 15 max [0 , 0+15]=15
10	3	15	max [v (2, 15), v(2, 5) = 15] = 24 max [24 , 0 + 15] = 24
10	3	16	max [v (2, 16), v(2, 6) = 15] = 24 max [24 , 0 + 15] = 24
10	3	17	max [v (2, 17), v(2, 7) = 15] = 24 max [24 , 0 + 15] = 24
10	3	18	max [v (2, 18), v(2, 8) = 15] = 25 max [25 , 0 + 15] = 25
10	3	19	max [v (2, 19), v(2, 9) = 15] = 25 max [25 , 0 + 15] = 25
10	3	20	max [v (2, 20), v(2, 10) = 15] = 25 max [25 , 0 + 15] = 25

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	25	25
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	24	24	25	25	25
3	0	0	0	0	0	0	0	0	0	0	15	15	15	15	15	24	24	24	25	25	(25)

Optimal solution

Thus, optimal solution v [n, m] = v [3, 20] = 25

UNIT 6

BACKTRACKING, BRANCH AND BOUND

SYLLABUS

Backtracking, Branch and Bound.

6.1 Backtracking

6.1.1 The method

6.1.2 Explicit and Implicit constraints.

6.1.3 Solution spad.

6.1.4 n-queens problem.

6.1.5 Travelling salesman problem.

6.2 Branch and Bound.

6.2.1 Traveling salesman problem.

SYNAPOSIS

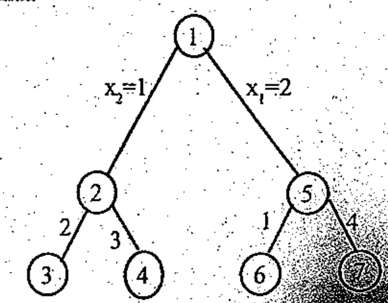
In backtracing method the desired solution is expressible as an n' tuple (x_1, x_2, -----x_n) where xi is chosen from some finite set s_i.

The basic idea of backtracking is to build up a vector, one component at a time and to test whether the vector being formed has any chance of success.

The major advantage of this algorithm is that we can realize the fact that the partial vector generated does not lead to an optimal solution. In such a situation that vector can be ignored.

n- queens problem

Consider a 4x4 chess board and 4 queen's. The objective is to place these 4 queen's in each row such that no 2 queens attack each other. Thus a systematic approach is used to solve this problem. The solution space tree is that.



Travelling salesman problem.

If there are n cities and cost of travelling from any city to any other city is given then we have to obtain the cheapest round trip such that each city is visited exactly once and then returning to the starting city completes the tour.

5 MARK QUESTIONS

1. Explain the method of backtracking.

Ans. 1. The desired solution is expressible as an n^{t} tuple (x1, x2, -----xn) where xi is chosen from some finite set si.

2. The solution maximizes or minimizes or satisfies a criteria on function c(x1, x2 ----xn).

3. The problem can be categorized into 3 types.

(i) Finding Feasible solution? Is decision problem.

(ii) Finding best solution? Is optimization problem.

(iii) Listing all feasible solution: enumeration problem.

4. The basic idea of back tracking is to build up a vector, one component at a time and to test whether the vector being formed has any chance of success.

5. The major advantage of this algorithm is that the fact can be realise the fact that the partial vector generated does not lead to an optimal solution, In such a situation that vector can be ignored.

6. It determines the solution by systematically, searching the solution space (i.e set of all feasible solutions) for the given problem.

7. It is a DFS search with some bounding function.

2. Explain the method of branch and bound.

Ans. In this method a space tree of all possible solutions is generated. The partitioning or branching is done at each node of the tree and a bound value is computed at each node. If the bound value of some node is not better than the best solution then that corresponding node is not expanded further. Such a node is called non-promising node and is terminated over there. It is also called that the branch is pruned over there. This is done so because such a branch will never lead to solution. This is principle idea of branch and bound. Thus computation of bounding value at each node comparing it with best solution and then expanding the node further can lead to final solution node. In Branch and bound technique assigning the best bounding value is a difficult task So FIFO method is used for Branch and bound. In this technique, a queue contains set of live nodes and each live node is taken from the queue for exploration depending upon its lower bound value.

3. Write the applications of Backtracking.

Ans. (i) Queen's Problem.

(ii) Binary Lock Problem.

(iii) Container Landing Problem.

(iv) Knapsack Problem.

(v) Travelling Salesman Problem.

4. Write the applications of Branch and bound.

Ans. (i) Integer programming.

(ii) Nonlinear programming.

(iii) Quadratic assignment problem.

(iv) Nearest neighbor search.

(v) Travelling salesman problem.

5. Explain explicit and implicit constraints with example.

Ans. 1. Explicit constraints are the rules that restrict each x1 to take on values only from a given set.

→ These depend on the particular instance I of problem being solved.

→ All tuples that satisfy the explicit constraints define a possible solution space for I.

Eg:- x_i = o or all non-negative real numbers.

$x_i = \{0, 1\}$

$L_i \leq x_i \leq u_i$

2. Implicit constraints.

Implicit constraints are rules that determine which of the tuples in the solution space of I satisfy the criterion function.

→ It describes the way in which the x_i must relate to each other.

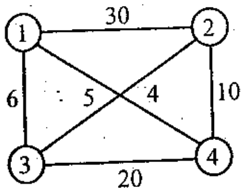
* Determine problem solution by systematically searching the solution space for the given problem instant.

→ Use a tree organization for solution space

10 MARK QUESTIONS

1. Explain travelling salesman problem using Backtracking.

Ans. Problem statement: If there are n cities and cost of travelling from any city to any other city is given, then we have to obtain the cheapest round trip such that each city is visited exactly once and then returning to the city where started completes the tour.



$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 30 & 6 & 4 \\ 2 & 30 & 0 & 5 & 10 \\ 3 & 6 & 5 & 0 & 20 \\ 4 & 4 & 10 & 20 & 0 \end{array}$$

$1 \to 2 \to 4 \to 3 \to 1 = 30 + 10 + 20 + 6 = 76$

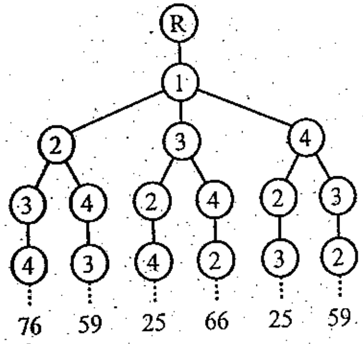
$1 \to 2 \to 3 \to 4 \to 1 = 30 + 5 + 20 + 4 = 59$

$1 \to 3 \to 4 \to 2 \to 1 = 6 + 20 + 10 + 30 = 66$

$1 \to 3 \to 2 \to 4 \to 1 = 6 + 5 + 10 + 4 = 25$

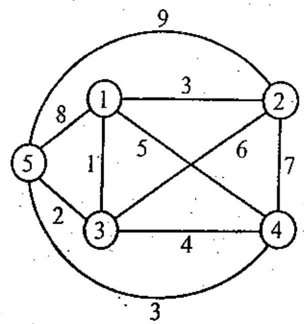
$1 \to 4 \to 3 \to 2 \to 1 = 4 + 20 + 5 + 30 = 59$

$1 \to 4 \to 2 \to 3 \to 1 = 4 + 10 + 5 + 6 = 25$



The optimal solution is 25

2. Explain travelling salesman problem using Branch and bound.



1	1, 2	1, 3	1, 4	1, 5
	3	1	5	8
2	2, 1	2, 3	2, 4	2, 5
	3	6	7	9
3	3, 1	3, 2	3, 4	3, 5
	1	6	4	2
4	4, 1	4, 2	4, 3	4, 5
	5	7	4	3
5	5, 1	5, 2	5, 3	5, 4
	8	9	2	3

vertex 1 min $(3, 1, 5, 8) \to 1$.

vertex 2 min $(3, 6, 7, 9) \to 3$.

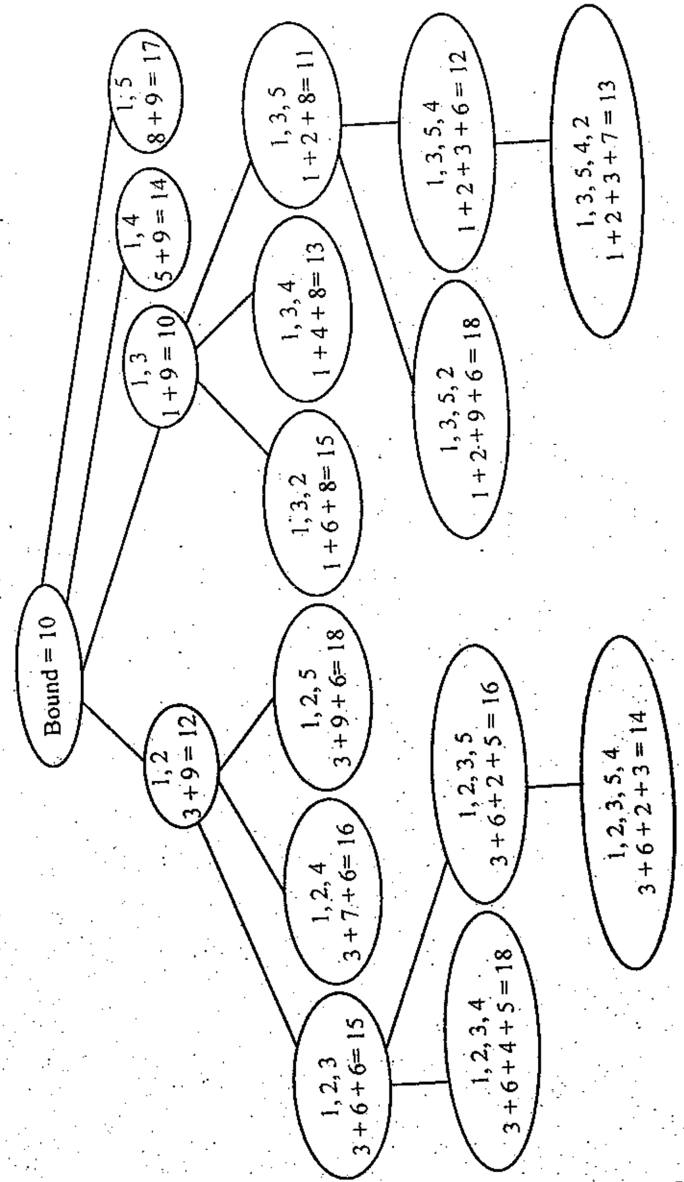
vertex 3 min $(1, 6, 4, 2) \to 1$.

vertex 4 min $(5, 7, 4, 3) \to 3$.

vertex 5 min $(8, 9, 2, 3) \to 2$.

Total $= 1 + 3 + 1 + 3 + 2$

Bound value $= 10$



The optimal solution for TSP using Branch and bound is $1 \to 3 \to 5 \to 4 \to 2$

$1 + 2 + 3 + 7 = 13$

QUESTION PAPERS

DTE SUPER MODEL QUESTION PAPER (WITH ANSWERS)

Diploma in Information Science & Engineering

IV- Semester

ANALYSIS AND DESIGN OF ALGORITHM

Time: 3 Hours **Max Marks: 100**

PART-A

Answer any SIX questions each carries 5 marks. **5 x 6 = 30 Marks**

1. Write an algorithm to find sum and average of three numbers.

Ans. Ref. Unit 1, Q3. (5 Marks)

2. What is time complexity?. Explain with suitable example.

Ans. Ref. Unit 1, Q11. (5 Marks)

3. Explain node vertex, edge in a graph with an example.

Ans. Ref. Unit 2, Q2. (5 Marks)

4. Explain Travelling salesman problem with suitable example using Brute force technique.

Ans. Ref. Unit 3, Q6. (5 Marks)

5. Explain and write an algorithm using decrease and conquer techinque.

Ans. Ref. Unit 4, Q3. (5 Marks)

6. Sort the following numbers using Insertion sort.

6 5 4 3 2

Ans. Ref. Unit 4. Q5. (5 Marks)

7. Write the prim's algorithm.

Ans. Ref. Unit 5, Q4. (5 Marks)

8. Write the applications of Dynamic programming.

Ans. Ref. Unit 5, Q7. (5 Makrs)

9. Explain explicit and implicit constraints with example.

Ans. Ref. Unit 6, Q5. (5 Marks)

PART-B

Answer any SEVEN full questions each carries 10marks. **10 x 7 = 70 Marks**

1. Explain the follwing with suitable examples.

(a) Big-oh notation.

(b) Big-omega notation.

Ans. Ref Unit 1, Q5. (10 Marks)

2. f(n) = 100n + 5, analyse for best, worst and average cases. (10 Marks)

Ans. Ref. Unit 1. Q1.

3. Explain the following.

(a) Sorting.

(b) Searching

(c) Graph problems. (10Marks)

Ans. Unit 2, Q1.

4. Write an algorithm of DFS and explain. (10 Marks)

Ans. Ref. Unit 2, Q4.

5. Write and explain bubble sort algorithm with suitable example. (10 Marks)

Ans. Ref. Unit 3, Q2.

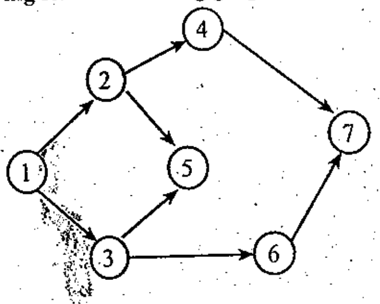
6. Write the formulas and apply strassen's matrix multiplication for the following.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}$$

(10 Marks)

Ans. Ref. Unit 4, Q3.

7. Obtain Topological ordering for the following graph.



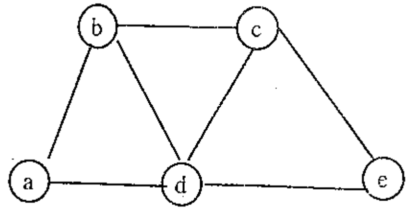
(10 Marks)

Ans. Ref. Unit 4, Q8.

8. Solve the following instance for 0/1 knapsack problem using dynamic programming, n = 4, p= [4, 2, 1, 8] w = [3, 1, 7, 9] m=10. (10 Marks)

Ans. Ref. Unit 5, Q9.

9. Find the single source shortest peth for the following graph.



Ans. Ref. Unit 5, Q4. (10 Marks)

10. Explain n-queens problem and write the solution space tree for 4 queens.

Ans. Consider a 4 x 4 chess board and 4 queens. The objective is to place these 4 queens in each row such that no 2 queens attack each other. Thus a systematic approach is used to solve this problem. The solution space is tree is drawn.

1. Level of the nod represents row of a chess board and X_i on the edge represents column of the chess board.

2. We start with root as 1st line node, this node becomes E node andn generates node no.2, which implies X_1=1 i.e 1[st] Q is placed in the column of 1[st] row.

Q_1			

3. Node 2 becomes E node and node 3 is generated since it attacks with Q_1 there is no use of further generation. Hence node 3 is skipped with all its children.

Q_1			
	Q_2		

4. Now node 8 is generated from node 2 i.e X_2 =3 and it is accepted.

5. Now node 8 becomes E node and next node is generated X_3 -2. Since if attacks, there is no use of further generation, Hence node 9 is killed.

6. Generate node 11 i.e X_3 = 4 from node 8, since it attacks. And even node 11 is killed. Since there are no other possible childs for node 8, thus backtrack to node 2 again.

7. Now node 13 is generated i.e X_2 = 4, since it is acceptable it becomes E node and node 14 is generated i.e X_3 = 2.

8. Now node 14 becomes E node and node i.e X_4 = 3 is generated and is killed because it gets attacked by Q_3 since there are no further nodes backtrack to 13.

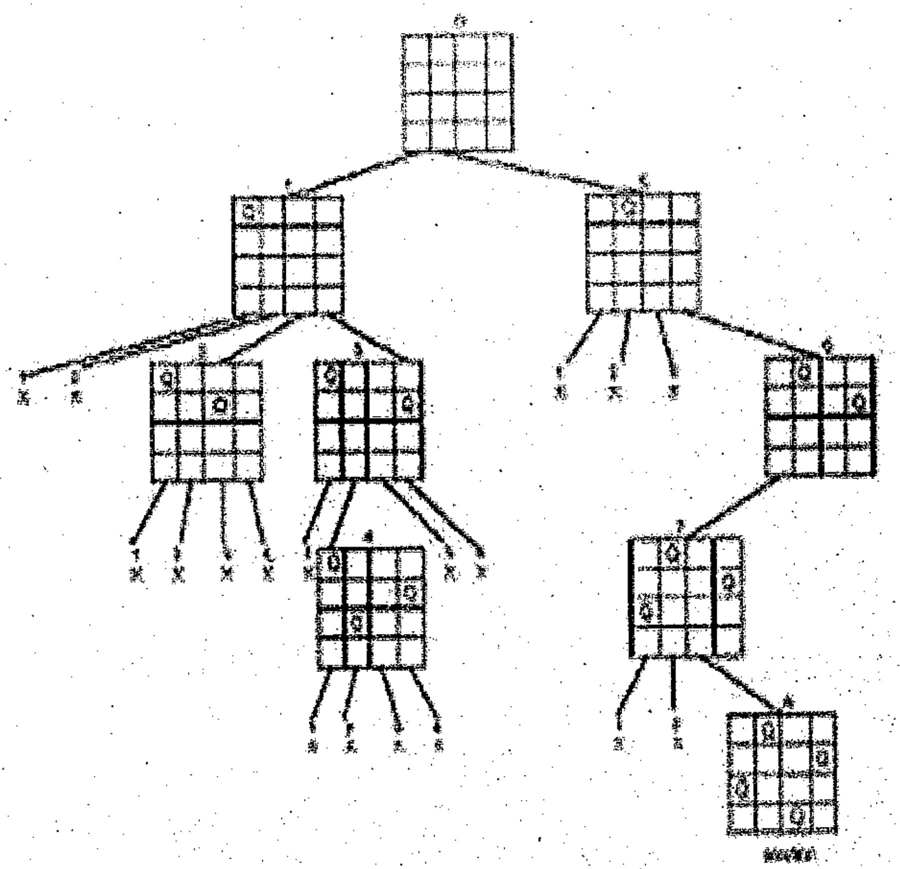
9. Now node 16 i.e $X_3 = 3$ is generated and is killed because it is attacked by Q_1 and Q_2

Q_1			
		Q_2	

10. Since all nodes generated from node 2 thus we have to backtrack to root node and try node 18 and soon.

11. Finally the solution with X_1, X_2, X_3, X_4 is 2, 4, 1, 3.

State Space Tree of the Four-queens Problem



SUPER MODEL QUESTION PAPER (WITH ANSWERS)

Diploma in Information Science & Engineering

IV- Semester

ANALYSIS AND DESIGN OF ALGORITHM

Time: 3 Hours **Max Marks: 100**

PART-A

Answer any SIX questions each carries 5 marks. **5 x 6 = 30 Marks**

1. Define algorithm. Explain with an example.

Ans. Ref. Unit 1, Q1. (5 Marks)

2. Write a note on efficiency of algorithms.

Ans. Ref. Unit 1, Q6. (5 Marks)

3. Explain feasible and optimal solution with example.

Ans. Ref. Unit 2, Q10. (5 Marks)

4. Explain sequential search algorithm with suitable example.

Ans. Ref. Unit 3, Q5. (5 Marks)

5. Write the algorithm for Merge sort.

Ans. Ref. Unit 4, Q1. (5 Marks)

7.Write the Warshall's Algorithm.

Ans. Ref. Unit 5, Q2 (5 Marks)

8. Write the application of Greedy method.

Ans. Ref. Unit 5, Q7. (5 Marks)

9. Explain the method of backtracking.

Ans. Ref. Unit 6 Q1. (5 Marks)

PART-B

Answer any SEVEN full questions each carries 10marks. **10 x 7 = 70 Mark**

1. Define algorithm. Explain the fundamental steps in solving any algorithm.

Ans. Ref. Unit 1, Q2. (10 Marks)

2. Explain best, average and worst case analysis with an example.

Ans. Ref. Unit 1, Q3. (10 Marks)

3. Define graph. Explain different types of graph representations with example.

Ans. Ref. Unit 2, Q3. (10 Marks)

4. Write an algorithm of BFs and explain.

Ans. Ref. Unit 2, Q5. (10 Marks)

5. Write and explain selection sort with example. (10 Marks)

Ans. Ref. Unit 3, Q1.

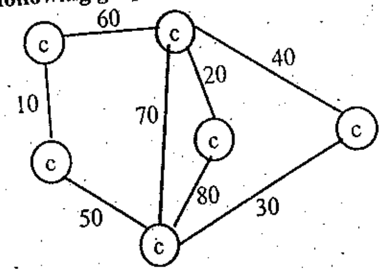
6. Write and explain Quick sort with suitable example using divide and conquer method. (10 Marks)

Ans. Ref. Unit 4, Q5.

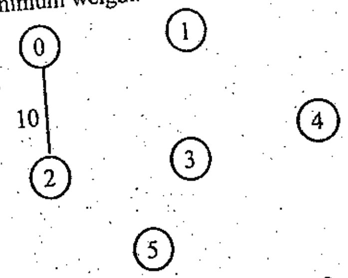
7. Write and explain Insertion sort with example. (10 Marks)

Ans. Ref. Unit 4, Q5.

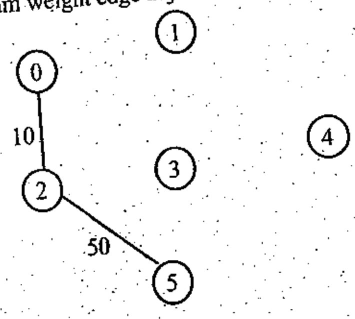
8. Obtain the MST for the following graph using Prim's Algorithm.



Step 1: Select an edge with minimum weight.

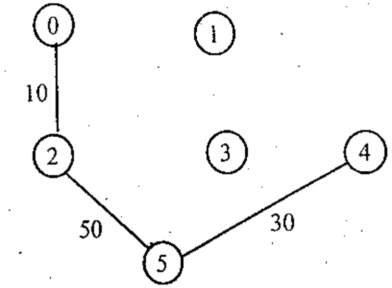


Step 2: Select the next minimum weight edge adjacent to 0 or 2.



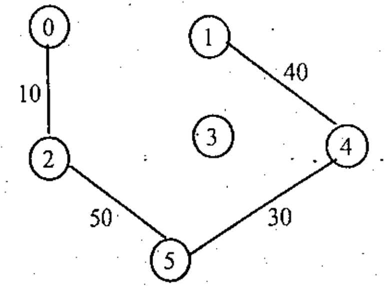
Total weight = 10 + 50 = 60

Step 3: Select the next minimum weight and so on.



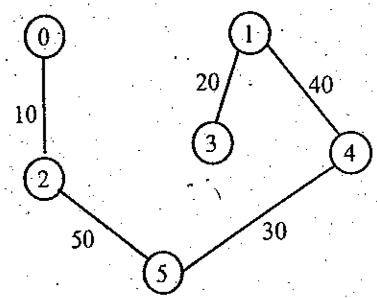
Total weight = 10+50+30=90.

Step 4:



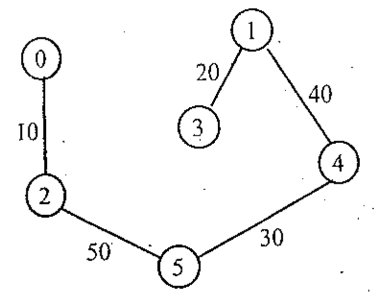
Total weight = 10+50+30+40=130

Step 5:

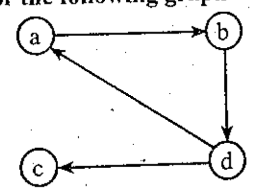


Total weight = 10+50+30+40+20=150

Finally, the MST is



9. Obtain the transitive closure for the following graph.



Ans. Ref. Unit-V Q 3

10 Explain TSP using backtracking. **(10 Marks)**

Ans. Ref. Unit 6, Q1.

SUPER MODEL PRACTICE QUESTION PAPER

Diploma in Information Science & Engineering

IV- Semester

ANALYSIS AND DESIGN OF ALGORITHM

Time: 3 Hours **Max Marks: 100**

PART-A

Answer any SIX questions each carries 5 marks. **5 x 6 = 30 Marks**

1. Write the advantages and disadvantages of an algorithm.

2. Explain counting primitive operations with example.

3. Explain different types of trees with example.

4. Write an algorithm for selection sort.

5. Sort the following numbers using Merge sort

8 4 1 3 2

6. Write an algorithm for insertion sort.

7. Write Kruskal's algorithm.

8. Write a note on dynamic programming.

9. Explain the method of branch and bound.

PART-B

Answer any SEVEN full questions each carries 10marks. **10 x 7 = 70 Mark**

1. Explain best, worst and average cases of linear search.

2. Explain with example.

(a) Big-oh Notation

(b) Big- theta Notation

3. Explain the following with example.

(a) DFS **(b) BFs**

4. Define graph. Explain different types of graphs with example.

5. Explain the following with example using brute force technique

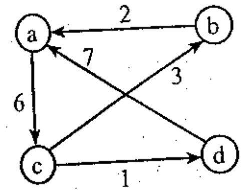
(a) TSP **(b) 0/1 Knapsack.**

6. Explain Merge sort with example.

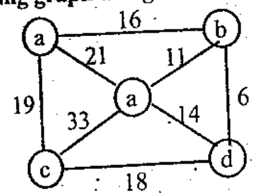
7. Solve using strassen's matrix multiplication.

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 1 \\ 3 & 1 \end{bmatrix}$$

8. **Obtain all pairs shortest path for the following graph.**



9. **Obtain the MST for the following graph using Kruskal's algorithm.**



10. **Explain TSP using branch and bound.**