SYLLABUS

Design & Analysis of Algorithms (CS501PC)

Unit - I (Chapters - 1,2)

Introduction : Algorithm definition, Algorithm specfication, Performance analysis - Space complexity, Time complexity, Randomized algorithms.

Divide and Conquer : General method, Applications - Binary search, Merge sort, Quick sort, Strassen's matrix multiplication.

Unit - II (Chapters - 3,6)

Disjoint set operations, Union and find algorithms, AND/OR graphs, Connected components, and Spanning trees, Bi-connected components, **Backtracking** -general method, applications, The 8-queen problem, sum of subsets problem, Graph coloring, Hamiltonian cycles.

Unit - III (Chapter - 4)

Greedy Method : General method, Applications - Knapsack problem, Job sequencing with deadlines, Minimum cost spanning trees, Single source shortest path problem.

Unit - IV (Chapter - 5)

Dynamic Programming : General method, Applications - Chained matrix multiplication, All pairs shortest path problem, Optimal binary search trees, 0/1 Knapsack problem, Reliabiltiy design, Travelling salesperson problem.

Unit - V (Chapters - 7,8)

Branch and Bound : General method, Applications - 0/1 Knapsack problem - LC branch and bound solution, FIFO branch and bound solution, Travelling salesperson problem.

NP-Hard and NP-Complete Problems : Basic concepts, Non-deterministic algorithms, NP-hard and NP-complete classes, Cook's theorem.

TABLE OF CONTENTS

Unit - II

Unit - V

UNIT - I

1 INTRODUCTION

1.1 Algorithm

Q.1 What is algorithm ?

Ans. : Definition of algorithm : The algorithm is defined as a collection of unambiguous instructions occurring in some specific sequence and such an algorithm should produce output for given set of input in finite amount of time.

Q.2 Define algorithm and describe the characteristics of the algorithm.

☞ **[JNTU : Part B, March-06, Nov.-06, Marks 8]**

Ans. : Algorithm – Refer Q.1.

Characteristics : Algorithm should posses following properties -

1) **Input** - The input of zero or more number of quantities should be given to the algorithm.
2) **Output** - The algorithm should produce at least one quantity, as an output.
3) **Definiteness** - Each instruction in algorithm should be specific and unambiguous.
4) **Finiteness** - The algorithm should be finite. That means after finite number of steps it should terminate.
5) **Effectiveness** - Every step of algorithm should be feasible. In other words, every step of algorithm should be such that it can be carried out by pen and pencil.

Q.3 Differentiate between profiling and debugging.

☞ **[JNTU : Part B, May-08, Marks 6]**

Ans :

- Debugging is a technique in which a sample set of data is tested to see whether faulty results occur or not. If any faulty result occurs then those results are corrected.
- But in debugging technique only presence of error is pointed out. Any hidden error can not be identified. Thus in debugging we cannot verify correctness of output on sample data. Hence profiling concept is introduced.
- Profiling or performance measurement is the process of executing a correct program on a sample set of data. Then the time and space required by the program to execute is measured.

1.2 Pseudo Code for Expressing Algorithms

Q.4 Write a recursive algorithm that converts a string of numerals to an integer, for example "34567" to 34567. ☞ **[JNTU : Part B, Dec.-11, Marks 7]**

Ans. :

```
Algorithm StrToNum(char *c, double *i)
{
  //initially i is initialised to 0
  if (*c == '\0' || !isdigit(*c)) then
   return *i;
   *i = *i * 10 + *c - '0';
   return StrToNum (c + 1, i);
}
```

Q.5 Write a recursive algorithm that calculates and returns the length of a list.

☞ **[JNTU : Part B, Dec.-11, Marks 8]**

Ans. :

```
Algorithm stringlen( char *p1,int *count)
{
    //initially count is intialised to 0
   if(*p1 == '\0') then
      return *count;
   *count=*count+1;
   return stringlen (p1+1,count);
  //finally the count contains the length of the string
//in main procedure simply print the value of count
}
```

Q.6 Write an algorithm to evaluate a polynomial using Honor's rule.
☞ [JNTU : Part B, May-09, Marks 8, Dec.-11, Marks 7]

Ans. :

```
Algorithm Poly_Honor(n,x)
{
  //initialize array a by coefficients of polynomial
   int k ← n;
   sum ← 0;
   for (i←k; i>0; i--)
   {
     sum ← (sum+a[i])*x;
   }
   sum ← sum+a[0];
   Write(sum);//evaluated result
}
```

Q.7 Write an algorithm in pseudocode to count the number of capital letters in a file of text. How many comparisons does it do ? What is fewest number of increments it might do ? What is the largest number ? Assume that N is number of characters in a file.
☞ [JNTU : Part B, April-11, Marks 15]

Ans. :

```
Algorithm Count_Capitals
{
  FILE *fp;
  Write("Enter file name: ");
  gets( filename );
  fp = fopen( filename, "r" );
  if ( fp == NULL ) then
  {
     Write("Cannot open for reading ");
     exit(1);     /* terminate program */
  }
  c = getc( fp ) ;
  while ( c != EOF ) do
  {
     if ((c >='A') AND (c<='Z')) then
   count++;
     c = getc ( fp );
  }
  fclose( fp );
  Write("Capital Letters in the file are "+count );
  }
```

If there are N number of characters in the file then, N comparisons are needed to obtain the count for capital letters.

Fewest Number - If the file is sorted and searched for the capital letter using binary search technique then in logN number of comparisons the capital letters can be obtained.

Largest Number - Comparing each letter for ensuring capital or not gives largest number of comparisons. It takes N number of comparisons.

1.2 Performance Analysis

Q.8 Define time complexity.
☞ [JNTU : Part A, Nov.-16, Marks 2]

Ans. : Time complexity is the amount of time taken by the computer to execute an algorithm. It is denoted by the asymptotic notations

Q.9 Define space complexity.
☞ [JNTU : Part A, Nov.-16, Marks 2]

Ans. : Space complexity is the amount of space required by an algorithm to execute. It is denoted by the asymptotic notation.

Q.10 What are the two factors used for defining the space complexity ? ☞ [JNTU : Part A, Marks 2]

Ans. : To compute the space complexity we use two factors : **constant and instance characteristics.** The space requirement S(p) can be given as :

$$S(p) = C + Sp$$

where C is a constant i.e. fixed part and it denotes the space of inputs and outputs. This space is an amount of space taken by instruction, variables and identifiers. And **Sp** is a space dependent upon instance characteristics. This is a variable part whose space requirement depends on particular problem instance.

Q.11 Describe the performance analysis in detail.
☞ [JNTU : Part B, May-08, May-12, Marks 8]

Ans. : The efficiency of an algorithm can be decided by measuring the performance of an algorithm. We

can measure the performance of an algorithm by computing two factors.

1. Amount of time required by an algorithm to execute.
2. Amount of storage required by an algorithm.

This is popularly known as **time complexity** and **space complexity** of an algorithm.

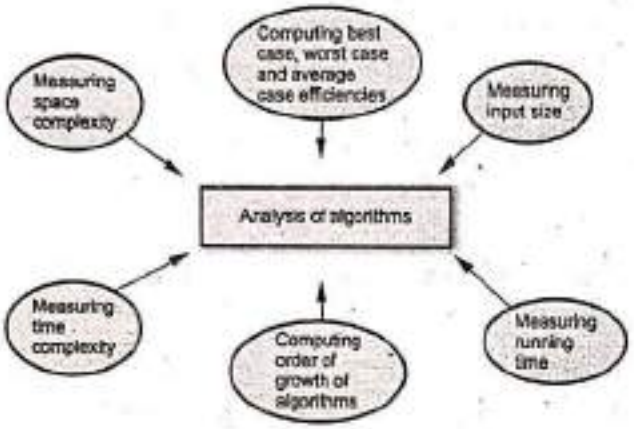


Fig. Q.11.1 Analysis of algorithms

Q.12 Why it is difficult to compute the time complexity in terms of physically clocked time ?

Ans. : It is difficult to compute the time complexity in terms of physically clocked time, because in multiuser system, executing time depends on many factors such as -

- System load
- Number of other programs running
- Instruction set used
- Speed of underlying hardware.

The time complexity is therefore given in terms of **frequency count.**

Frequency count is a count denoting number of times of execution of statement.

Q.13 What is frequency count ?

[JNTU : Part A, Marks 3]

Ans. : Frequency count is a count denoting number of times of execution of statement.

Consider following code for counting the frequency count.

```
void fun()
{
 int a;
 a=10;
 printf("%d",a);
}
```

For each statement except declaration statement, the frequency count will be -

```
void fun()
{
 int a;
 a=10; .....Executes once
 printf("%d",a); .....Execute Once
}
```

The frequency count of above program is 2.

Q.14 Obtain the frequency count for the following code.

```
i=1;
do
{
a++;
if(i==5)
break ;
i++;
}while(i<=n)
```

Ans. :

Statement	Frequency count
i=1;	1
a++;	5
if(i==5)	5
break;	1
++;	5
while(i<=n)	5
Total	22

Q.15 What is the smallest value of n such that an algorithm whose running time is $100\,n^2$ runs faster than the algorithm whose running time is 2^n on the same machine ? **[JNTU : Part B, Marks 5]**

Ans. : Finding the smallest value of n for an algorithm with $100n^2$ running time which runs faster than algorithm with 2^n running time means we have to obtain value of n such that $100\,n^2 < 2^n$.

If $n = 10$ then

$100(10)^2 \stackrel{?}{=} 2^{10}$

$10000 > 1024$

If $n = 12$ then

$100(12)^2 \stackrel{?}{=} 2^{12}$

$14400 > 4096$

If $n = 14$ then

$100(14)^2 \stackrel{?}{=} 2^{14}$

$19600 > 16384$

If $n = 16$ then

$100(16)^2 \stackrel{?}{=} 2^{16}$

$25600 < 65536$ ← Found

If $n = 15$ then

$100(15)^2 \stackrel{?}{=} 2^{15}$

$22500 < 32768$

As $n = 15$ the 2^n exceeds $100n^2$. Hence $n = 15$, is the smallest value satisfying the given condition.

Q.16 Arrange following rate of growth in increasing order.
2^n, nlogn, n^2, 1, n, logn, n!, n^3
[JNTU : Part A, Marks 2]

Ans. : 1, logn, n, nlogn, n^2, n^3, 2^n, n!

Q.17 Reorder the following complexity from smallest to largest
i) $n\log_2(n)$, $n + n^2 + n^3$, 2^4, sqrt(n)
ii) n^2, 2^n, $n\log_2(n)$, $\log_2(n)$, n^3
iii) nlog (n), n^8, $n^2/\log n$, $(n^2 - n + 1)$
iv) n!, 2^n, $(n+1)!$, 2^{2n}, n^n, $n^{\log n}$
[JNTU : Part A, Marks 3]

Ans. :

i) sqrt(n), $n \log_2 n$, $n + n^2 + n^3$, 2^4

ii) $\log n$, $n \log n$, n^2, n^3, 2^n

iii) $n \log n$, $\frac{n^2}{\log n}$, $(n^2 - n + 1)$, n^8

iv) $n^{\log n}$, 2^n, n!, $(n+1)!$, 2^{2n}, n^n

1.4 Asymptotic Notation

Q.18 List out the reasons for the difficulties that one faces while determining lower bound.
[JNTU : Part A, Nov.-16, Marks 2]

Ans. : A lower bound of a problem is the least time complexity required for any algorithm which can be used to solve this problem. The problem with computing lower bound is that - The lower bound for a problem is not unique. For example - $\Omega(1)$, $\Omega(n)$, $\Omega(n \log n)$ are all lower bounds for sorting.

Due to this – i) We always try to find higher lower bound ii) We always try to find better algorithm

Q.19 Define time complexity. Describe different notations used to represent these complexities.
[JNTU : Part B, May-09,13, Marks 10]

Ans. : Time complexity - Refer Q. 8.

Asymptotic notation is a shorthand way to represent the time complexity.

Using asymptotic notations we can give time complexity as "fastest possible", "slowest possible" or "average time".

Various notations such as Ω, Θ and O used are called **asymptotic notions.**

1. Big oh notation

The **Big oh** notation is denoted by 'O'. It is a method of representing the **upper bound of** algorithm's running time. Using big oh notation we can give longest amount of time taken by the algorithm to complete.

Definition

Let f(n) and g(n) be two non-negative functions.

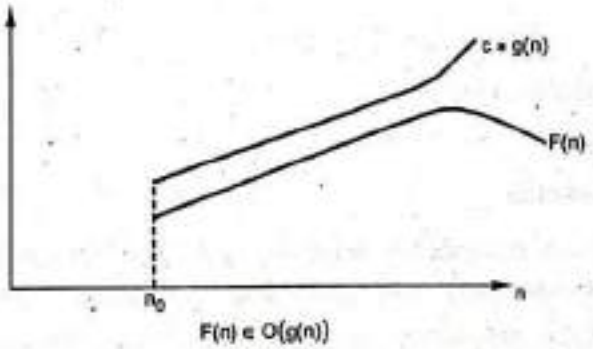


Fig. Q.19.1

Let n_0 and constant c are two integers such that n_0 denotes some value of input and $n > n_0$. Similarly c is some constant such that $c > 0$. We can write

$$f(n) \leq c * g(n)$$

then f(n) is big oh of g(n). It is also denoted as $f(n) \in O(g(n))$. In other words f(n) is less than g(n) if g(n) is multiple of some constant c.

For Example : If $f(n) = 2n + 2$ and $g(n) = n^2$ then $f(n) = O(g(n))$ for $n > 2$

2. Omega notation

Omega notation is denoted by 'Ω'. This notation is used to represent the **lower bound** of algorithm's running time. Using omega notation we can denote shortest amount of time taken by algorithm.

Definition

A function f(n) is said to be in Ω (g(n)) if f(n) is bounded below by some positive constant multiple of g(n) such that

$$f(n) \geq c * g(n) \qquad \text{For all } n \geq n_0$$

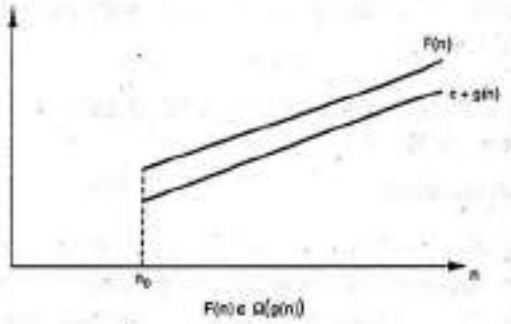


Fig. Q.19.2

It is denoted as $f(n) \in \Omega(g(n))$. Following graph illustrates the curve for Ω notation.

Consider $f(n) = 2n^2 + 5$ and $g(n) = 7n$

or $n > 3$ we get $f(n) > c * g(n)$.

It can be represented as

$$2n^2 + 5 \in \Omega(n)$$

3. Θ notation

The theta notation is denoted by Θ. By this method the running time is between upper bound and lower bound.

Definition

Let f(n) and g(n) be two non negative functions. There are two positive constants namely c_1 and c_2 such that

$$c_1 \leq g(n) \leq c_2 g(n)$$

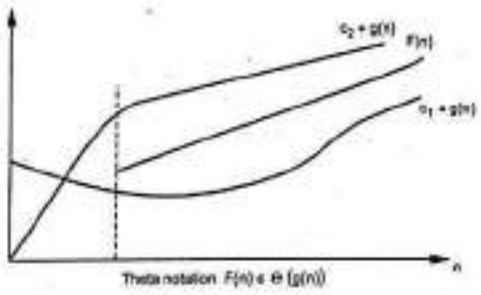


Fig. Q.19.3

Then we can say that

$$f(n) \in \Theta(g(n))$$

If $f(n) = 2n + 8$ and $g(n) = 7n$.

where $n \geq 2$

Similarly $f(n) = 2n + 8$

$g(n) = 7n$

i.e. $5n < 2n + 8 < 7n$ For $n \geq 2$

Here $c_1 = 5$ and $c_2 = 7$ with $n_0 = 2$.

Then $f(n) = \Theta(n)$

The theta notation is more precise with both big oh and omega notation.

Q.20 Compare Big-oh notation and Little-oh notation. Illustrate with an example.

[JNTU : Part B, May-12, Marks 7]

Ans. : Big- Oh notation – Refer Q. 19.

Little oh notation - The little Oh is denoted as o. It is defined as : Let, f(n) and g(n) be the non negative functions then

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

such that $f(n) = o(g(n))$ i.e f of n is little Oh of g of n.

$f(n) = o(g(n))$ if and only if $f(n) = o(g(n))$ and $f(n) \neq \theta(g(n))$

Q.21 Find Big-oh notation and Little-oh notation for $f(n) = 7n^3 + 50n^2 + 200$.

[JNTU : Part B, May-12, Marks 8]

Ans. : Let $f(n) = 7n^3 + 50n^2 + 200$

$$f(n) < 300\, n^3$$

For n larger than some crossing point. Say $n > 10$.

$$300 * n^3 < c * n^4$$

$$3n < cn^2$$

$$3 < c * n$$

$$n > \frac{3}{c}$$

For calculating little oh we have to pick up n such that $f(n) < c * n^4$ is true. So pick up n i.e. $\max\left(10, \frac{3}{c}\right)$.

Hence for given $f(n) = o(n^4)$.

The big-oh notation $f(n) = O(n^3)$.

Q.22 What are the basic efficiency classes ?

Ans. : The classification of different order of growth is as follows -

Name of efficiency class	Order of growth	Description	Example
Constant	1	As input size grows the we get larger running time.	Scanning array elements.
Logarithmic	log n	When we get logarithmic running time then it is sure that the algorithm does not consider all its input rather the problem is divided into smaller parts on each iteration.	Performing binary search operation.
Linear	n	The running time of algorithm depends on the input size n.	Performing sequential search operation.
nlogn	nlogn	Some instance of input is considered for the list of size n.	Sorting the elements using merge sort or quick sort.
Quadratic	n^2	When the algorithm has two nested loops then this type of efficiency occurs.	Scanning matrix elements.
Cubic	n^3	When the algorithm has three nested loops then this type of efficiency occurs.	Performing matrix multiplication.
Exponential	2^n	When the algorithm has very faster rate of growth then this type of efficiency occurs.	Generating all subsets of n elements.
Factorial	n!	When an algorithm is computing all the permutations then this type of efficiency occurs.	Generating all permutations.

Table Q.22.1 Basic asymptotic efficiency classes

Q.23 Write the non-recursive algorithm for finding the Fibonacci sequence and derive its time complexity.

[JNTU : Part B, Sept.-08, May-09, Marks 10, Nov.-16, Marks 5]

Ans. :

1. **Algorithm** Fibbo(n)
2. {
3. **//Problem Description** : This problem is to compute
4. //fibonacci sequence using non recursive method
5. **//Input** : Integer n for length of Fibbonacci sequence

```
6. //Output: The Fibonacci number present at n
location
7. if (n<=1) then
8.     Write(n);
9. else
10. {
11.    f2=0;f1=1;
12.    for(i=2 to n) do
13.    {
14.       f=f1+f2;
15.       f2=f1;f1=f;
16.    }
17.    Write(f);
18. }
19.}
```

Analysis

In above algorithm, basic operation is computing of next value in the Fibonacci sequence.

Line number	Frequency count
7	1
8	1
11	1
12	n
14	n – 1
15	n – 1
17	n
Total	4n + 1

By neglecting the constants, and by considering the order of magnitude we can denote the complexity of the above algorithm as **O(n)**.

Q.24 Show that f(n) + g(n) = O(n²) where f(n) = 3 n² – n + 4 and g(n) = nlogn + 5.
[JNTU : Part B, Sept.-08, May-09, Marks 8]

Ans. : This problem is for defining big oh notation. By definition big oh notation. We should have, $f(n) \leq c * g(n)$ where c is a constant. For the given problem we have to prove,

$$f(n) + g(n) \leq O(n^2).$$

We will select some positive constant c and value of n.

If n = 2 then

$$\text{L.H.S.} = f(n) + g(n)$$
$$= (3n^2 - n + 4) + (n \log_2 n + 5)$$
$$= (3(2)^2 - 2 + 4) + (2 \log 2 + 5)$$
$$= (12 - 2 + 4) + (2 + 5)$$
$$= 21$$
$$\text{R.H.S.} = n^2$$
$$= c * n^2$$
$$= 6 * (2)^2 \quad \text{where c is a constant with value 6.}$$
$$= 6 * 4$$
$$= 24$$

That is L.H.S. ≤ R.H.S.

Hence $f(n) + g(n) \leq c * (n^2)$

This proves that $f(n) + g(n) = O(n^2)$

Q.25 Derive the function $f(n) = 12n^2 + 6n$ is $O(n^3)$ and $\omega(n)$. **[JNTU : Part B, May-08, May-09, Marks 6]**

Ans. : The omega notation is denoted by 'Ω'. This notation is used to represent lower bound of algorithm's running time. Using omega notation we can denote shortest amount of time taken by algorithm. Hence we can write,

$$f(n) \geq c * g(n) \qquad \text{For all } n \geq n_0$$

Then $f(n) \in \Omega(g(n))$

That means we have to find out values of n and c such that,

$$f(n) \geq c * g(n)$$

Given that $f(n) = 12n^2 + 6n$

$$g(n) = n^3$$

If n = 3 then,

$$f(n) = 12n^2 + 6n$$
$$= 12(3)^2 + 6(3)$$
$$= 108 + 18$$
$$= 116$$
$$= \text{L.H.S.}$$

$g(n) = n^3$

$= (3)^3$

$= 27$

If we choose $c = 3$ then $c * g(n) = 3 * 27 = 81$.

Thus we get, $f(n) \geq c * g(n)$.

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

Now we have to prove that $f(n) \neq \Theta(g(n))$.

i.e. $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ is not true.

If $n = 3$, $c_1 = 2$ and $c_2 = 3$ then.

$$2 * (3)^3 \leq 12(3)^2 + 6(3) \leq 3 * (3)^3$$

$$2 * 27 \leq 116 \neq 81 \quad \therefore f(n) \neq \Theta(g(n))$$

Hence $f(n) = \omega(g(n))$ is proved.

Q.26 Show that $f_1(n) \times f_2(n) = O(g_1(n) \times g_2(n))$ where $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$.

[JNTU : Part B, Sept.-08, Marks 6]

Ans. : Let $f(n) \in O(g(n))$ when $f(n) \leq c_1 * g(n)$ is true.

We assume $f_1(n) \leq c_1 * (g_1(n))$.

Similarly, $f_2(n) \leq c_2 * (g_2(n))$.

Let, $c_3 = c_1 * c_2$ where c_3 is a new constant.

Hence we can write the functions as

$$f_1(n) \times f_2(n) = c_1 * (g_1(n)) \times c_2 * (g_2(n))$$

$$f_1(n) \times f_2(n) = c_1 * c_2 (g_1(n) \times g_2(n))$$

$$f_1(n) \times f_2(n) = c_3 * g_1((n) \times g_2(n)) \quad \ldots (Q.26.1)$$

If we assume

$f_1(n) \times f_2(n) = f(n)$ and

$g_1(n) \times g_2(n) = g(n)$ then $f(n) = c_3 * g(n)$.

$\therefore f(n) = O(g(n))$.

Hence we can say about eqution (Q.26.1) as

$$f_1(n) \times f_2(n) = O(g_1(n) \times g_2(n)).$$

Q.27 Show that $f(n) = 4n^2 - 64n + 288 = \Omega(n)^2$.

[JNTU : Part B, May-09, Marks 6]

Ans. : The given function $f(n) = \Omega(g(n))$ only when $f(n) \geq c * (g(n))$. Where c is some positive constant.

Let, $f(n) = 4n^2 - 64n + 288$

Assume, $n = 1$ and $c = 5$ then,

L.H.S. $= f(n) = 4n^2 - 64n + 288$

$= 4(1)^2 - 64(1) + 288$

$= 4 - 64 + 288$

$= 228$

R.H.S. $= c * g(n) = 5 * (n)^2$

$= 5 * (1)^2$

$= 5$

L.H.S. $\geq$ R.H.S.

i.e. $f(n) \geq c * g(n)$

Hence $f(n) = \Omega(g(n))$ is proved.

Q.28 Show that $n^3 \log n$ is $\omega(n^3)$.

[JNTU : Part B, Sept.-08, Marks 6]

Ans. : To prove $f(n) = \omega(g(n))$ we have to prove two cases.

Case 1 : $f(n) = \Omega(g(n))$

Case 2 : $f(n) \neq \Theta(g(n))$

Let us prove both of these cases -

Case 1 :

L.H.S. $= f(n)$

$= n^3 \log n$

Assume $n = 4$ then

$= (4)^3 \log 4$

$= 64 \times 2$

$= 128$

R.H.S. $= c * g(n)$

Assume $n = 4$ and $c = 1$

$\therefore$ R.H.S. $= 1 * (n^3)$

$= 1 * (4)^3$

$= 64$

As L.H.S. $\geq$ R.H.S.

i.e. $f(n) \geq c * g(n)$ we can say,

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

Case 2 : To prove $f(n) \neq \Theta(g(n))$ we have to prove that,

$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ **is not true.**

The $f(n) = n^3 \log n$ and $g(n) = n^3$.

Assume $n = 4$, $c_1 = 3$, $c_2 = 4$, then

$c_1 * g(n) = 3 * n^3 = 3 * 4^3 = 192$

$f(n) = n^3 \log n$

$= 4^3 (\log 4)$

$= 128$

$c_2 * g(n) = 4 * n^3$

$= 4 * (4)^3$

$= 256$

i.e. $192 \leq 128 \leq 256$ is not true.

Hence $c_1 * g(n) \leq f(n) \leq 256$ is not true.

$\therefore$ $f(n) = \Theta(g(n))$ is not true.

Hence **f(n) = ω(g(n))** is proved

Q.29 Given an array of n elements, (possibly with some of the elements are duplicates). Write an algorithm to remove all duplicates from the array in time O(n log n).

[JNTU : Part B, April-11, Marks 15]

Ans. : We will consider an array a[ ] in which all the elements with duplicates are stored. The logic to remove duplicates is as follows -

Store all the elements in **temp[ ]** array. The temp array will have two fields **index** and **value.**

1. Now sort the array **temp** based on value.
2. Remove duplicate elements (for searching duplicates just see the adjacent element in array). The duplicate with greater index value will be removed.
3. Now sort the array **temp** based on index.
4. Transfer the contents of **temp** array to array b.

For example consider an array a[ ]

0	1	2	3	4	5
10	5	10	3	5	2

array a

We will form **temp** array as,

	Value	index
0	10	0
1	5	1
2	10	2
3	3	3
4	5	4
5	2	5

temp

On sorting (Refer step 1)

	Value	index
0	2	5
1	3	3
2	5	1
3	5	4
4	10	0
5	10	2

temp

	Value	index
0	2	5
1	3	3
2	5	1
3	5	4 ← Delete
4	10	0
5	10	2 ← Delete

temp

On Deleting (Refer step 2)

	Value	index
0	2	5
1	3	3
2	5	1
4	10	0

temp

Now sorting **temp** based on index.

	Value	index
0	10	0
1	5	1
2	3	3
3	2	5

temp

Transfer to array a

0	10
1	5
2	3
3	2

b

Thus we get an array b without duplicates. Note that the order of elements in original array is maintained. The algorithm can be as given below -

```
Algorithm remove_duplicates (int a[1...n])
{
  // input : An array a[ ] with duplicates
  // output : An array b [ ] without duplicates
  Array temp [1 ... n] with two fields.
  for (i = 1 to n) do
  {
    temp[i].value = a[i];
    temp[i].index = i;
  }
```

sort temp array based on value. Remove duplicates with higher index. Sort temp array based on index. Transfer contents of array temp to b. Display array b.
}

The two sortings take O(n logn) time. Removing of duplicates takes O(n) time. Hence this algorithm takes **O(n logn)** time.

Q.30 If $T_1(n) = O(f(n))$ and $T_2(n) = O(g(n))$ then show that $T_1(n) + T_2(n) = O(max(f(n), g(n))$.
[JNTU : Part B, May-08, Marks 8]

Ans. : Let there be some constant c_1 such that

$$T_1(n) \leq c_1 g_1(n) \quad \text{for } n \geq n_1 \quad \ldots (Q.30.1)$$

Similarly there will be some constant c_2

Such that

$$T_2(n) \leq c_2 g_2(n) \quad \text{for } n \geq n_2 \quad \ldots (Q.30.2)$$

Let $c_3 = \max\{c_1, c_2\}$ such that $n \geq \max\{n_1, n_2\}$

Then we can write using equations (Q.30.1) and (Q.30.2) as :

$$T_1(n) + T_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$$
$$\leq (c_1 + c_2)(g_1(n) + g_2(n))$$
$$\leq c_3 (g_1(n) + g_2(n))$$
$$\leq c_3\, 2 \max(g_1(n) + g_2(n))$$

Hence

$T_1(n) + T_2(n) \in O(\max(g_1(n) + g_2(n))$ is true.

Q.31 Develop an algorithm to determine the minimum and maximum values in an array $a_1 a_2 \ldots a_n$ of integer (Here $n \geq 1$ and the entries in the array need not be distinct). Determine worst-case complexity function for this algorithm.

Ans. : Algorithm for minimum and maximum values in an array.

```
Algorithm Min-Max-element (a[0...n-1])
  //problem Description : This algorithm is for finding the
  //minimum and maximum element in an array.
  //input : An array a[] of some size n
  //output : Prints minimum and maximum
  //element of an array
  Max ← a[0]
  Min ← a[0]
  for (i ← 1 to n) do
  {
    if (a[i] > Max) then
      Max ← a [i]
    else if(a[i] < Max) then
      Min ← a [i]
  }
  Write ("The minimum element is", Min).
  Write ("The maximum element is", Max)
```

Mathematical analysis :

Step 1 : The **input size** is n i.e. total number of elements in array.

Step 2 : The **basic operation** is comparison in loop for finding maximum and minimum. Two such operations are there i.e. one give by **if** and other one is by **else if**.

Step 3 : The comparison is executed on each repetition of the loop. As comparison is made for each value of n, there is no need to find best case, worst case and average case analysis.

Step 4 : Let, C (n) be the number of times the comparison is executed. That means on each new value of i, the comparisons are made. Hence for i = 1 to n − 1 times the comparison are made. Then C(n) can be formulated as -

C(n) = Two comparisons made for each value of ;

Step 5 : Let us simplify the sum

$$C(n) = \sum_{i=1}^{n-1} 2$$

$$C(n) = 2 \cdot \sum_{i=1}^{n} 1$$

$$= 2(n - 1 + 1)$$

$$\because \left[\text{Using formula } \sum_{i=i}^{n} 1 = n - 1 + 1 = n\right]$$

$$= 2n$$

∴ The worst case efficiency is Ω (n).

Q.32 Write an algorithm to find the largest element in an array of n elements. Find the time complexity. [JNTU : Part B, Dec.-12, Marks 8]

Ans. :

```
Algorithm LargestElement(int a[],int n)
{
  int large;
  large ← a[0];
  for(i=0;i<n;i++) do
  {
    if(a[i]>large)
     large ← a[i];
  }
  Write("The largest element: "+large);
}
```

Time complexity = O(n)

1.5 Probabilistic Analysis

Q.33 Write short note on probabilistic analysis. [JNTU : Part B, Marks 5]

Ans. :

- In probability theory various "experiments" are carried out. The outcomes or results of those experiments determine the specific characteristics.
- **For example :** Picking up a card from a deck of 52 cards, tossing coin for five times, choosing a red ball from an urn containing red and white balls, rolling die four times. Each possible result of such experiment is called sample point. The set of all sample points is called **sample space.** The sample space is denoted by S. The sample space S is finite set. An event E occurs from sample space. For **m** sample points there are 2^m possible events.
- **Probability :** The probability of event E is the ratio of E to S. Hence

  $$\text{Probability} = \frac{|E|}{|S|}$$

- **For example :** When a coin is tossed, then we may get either head (H) or tail (T). Suppose, we have tossed four coins together then there are 16 possible outcomes : HHHH, HHHT, HHTH, HHTT, HTHH, HTHT, HTTH, HTTT, THHH, THHT, THTH, THTT, TTHH, TTHT, TTTH, TTTT. For 4 events (HHTH, THTT, TTHH, TTTT), the **probability** is $\frac{4}{16} = \frac{1}{4}$.
- **Mutual exclusion :** Two events A and B are said to be mutually exclusive if they do not have any common sample point. Hence $A \cap B = \phi$ For example A = {HHTH, HHTT}, B = {HTTT, THHT} are mutually exclusive.
- **Independence :** Two events A and B are said to be independent if

  $$P[A \cap B] = P[A] \cdot P[B]$$

- **Random variable :** The random variable is basically a function that maps elements of S to set of real numbers.
- For sample point $a \in S$ the F (a) denotes the mapping. If F denotes a finite set of elements then it is called **discrete.** Thus the random variables can be discrete random variables.
- **For example** - If we pick up four balls from an urn containing red and white balls then the number of red balls that get selected is F (RRRW) = 3 or F (RWWW) = 1 and so on.

Q.34 Explain the concept of probability distribution. [JNTU : Part A, Marks 3]

Ans. :

Probability distribution : If F is discrete random variable for sample space S then probability distribution can be defined for a range of elements $(a_1, a_2, \ldots a_n)$ such that $P[F=a_1]$, $P[F=a_2]$ ..., $P[F=a_n]$. For a probability distribution $\sum_{i=1}^{n} P[F=a_i] = 1$.

For example if we pick up four balls randomly from an urn containing red and white balls and F is number of red balls, then F can take on five values 0, 1, 2, 3 and 4 then

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

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

⇐ Here 1 means presence of red balls and 0 means absence of red ball from the four balls that are picked up.

Hence probability distribution of F is given by

$P[F = 0] = \frac{1}{16}$ i.e. no red ball present.

$P[F = 1] = \frac{4}{16}$ i.e. only one red ball present.

$P[F = 2] = \frac{6}{16}$ i.e. two red balls present from picked up balls.

$P[F = 3] = \frac{4}{16}$ i.e. when 3 red balls present from picked balls.

$P[F = 4] = \frac{1}{16}$ i.e. when 4 red balls are present from the picked balls.

Q.35 What is binomial distribution ?

☞ [JNTU : Part A, Marks 3]

Ans. : Suppose an experiment is conducted then the result of such experiment can be either **success** or **failure.**

Let, p represents success outcome.

Let n be number trials or experiments. These experiments are called **Bernoulli trial.**

The sample space S contains 2^n sample points. The random variable F has **binomial distribution** with (n, p) which can be given by -

$$P[F = i] = \binom{n}{i} p^i (1-p)^{n-i}$$

Q.36 Develop a probabilistic algorithm to find the value of the integral

$$\int_0^2 \sqrt{4-x^2}\, dx$$

☞ [JNTU : Part B, May-08, Marks 10]

Ans. : Assumption : Let, $f(x) = \sqrt{4 - x^2}$. The Lower bound = 0 and upper bound = 2.

```
Algorithm probabilistic (f(x), 0, 2)
{
   //Input : The function f(x) whose integration is to be
   obtained. The lower and
   //upper bound values
   //Output : The result of integration
      x = 2sin θ        // initialization
      dx = 2 cos θ dθ
   /* Find lower limit value */   } /* Computing low limit
      LOW = 0                      from x = 2 sin θ */
         0 = 2 sin θ;
         θ = 0;

/* Find upper limit value */
   Upper = 2
   2 = 2 sin θ;
   sin θ = 1;
   θ = π/2;

/* putting upper and lower limit values for integral */
```

$$I = \int_0^{\pi/2} \sqrt{4-4\sin^2\theta}\; 2\cos\theta\, d\theta$$

$$= 2\int_0^{\pi/2} \sqrt{1-\sin^2\theta}\; 2\cos\theta\, d\theta$$

$$= 4\int_0^{\pi/2} \cos^2\theta\, d\theta \quad // \text{As } \cos 2t = 2\cos^2 t - 1;$$

$$\cos^2 t = \frac{1+\cos 2t}{2}$$

$$= 4\int_0^{\pi/2} \left(\frac{1+\cos 2\theta}{2}\right) d\theta$$

$$= 2\left[\theta + \frac{\sin 2\theta}{2}\right]_0^{\pi/2}$$

$$= 2\left[\frac{\pi}{2} - 0\right] \quad //\sin\pi = 0$$

$$I = \pi$$

```
return I;
}
```

Q.37 What is randomizer [JNTU : Part A, Marks 2]

Ans. : A random number generator or randomizer makes use of randomized algorithms. In randomized algorithms the decision is always taken based current output.

Q.38 Explain the two types of randomized algorithm. [JNTU : Part A, Marks 2]

Ans. : There are two types of randomized algorithms.

Las Vegas algorithms : Are the type of randomized algorithms which produce same outcome on each execution for same input.

Monte Carlo algorithms : Are the type of randomized algorithms which produce different outcomes on each execution for same input.

Q.39 Give the advantages and disadvantages of randomized algorithm. [JNTU : Part A, Marks 3]

Ans. : There are two major advantages of randomized algorithms.

1. These algorithms are simple to implement.
2. These algorithms are many times efficient than traditional algorithms.

However randomized algorithms may have some drawbacks-

1. The small degree of error may be dangerous for some applications.
2. It is not always possible to obtain better results using randomized algorithm.

1.6 Amortized Complexity

Q.40 What is amortized analysis ? [JNTU : Part A, Marks 2]

Ans. :

- An amortized analysis means finding average running time per operation over a **worst case sequence** of operations.
- An amortized analysis indicates that average cost of a single operation is small if average of sequence of operations is obtained. This is true even if any one operation is expensive within the sequence.
- An amortized analysis guarantees the time per operation over worst case performance.

Q.41 What is the difference between average case analysis and amortized analysis?

Ans. : There is a difference between average case analysis and amortized analysis. In average - case analysis we are averaging over all possible inputs and in amortized analysis we are averaging over a sequence of operations. An amortized analysis assumes worst-case input.

Q.42 Explain about amortized analysis and probabilistic analysis. [JNTU : Part B,Dec.-12, Marks 7]

Ans. : Amortized analysis :

Definition : An amortized analysis means finding average running time per operation over a worst case sequence of operations.

There are 3 commonly used techniques used in amortized analysis -

i) Aggregate analysis ii) Accounting method iii) Potential method

i) Aggregate Analysis : Aggregate analysis is a kind of analysis in which the analysis is made over sequence of n operations and these operations actually take worst case time T(n). In worst case an amortized cost per operation is T(n)/n.

For example - Consider implementing stack. The two operations of the stack are 1. Push and 2. Pop

Each operation has running time as O(1). That means for n sequences of operations the total execution time will be (n).

Now if we write some function for performing **multiple pop** operations then k top objects will be removed from the stack.When the size of stack is at the most n. Hence a sequence of n operations costs $O(n^2)$. Hence average cost in amortized analysis is nothing but average cost of an operation = O(n)n.

ii) Accounting Method : The idea of accounting method is as follows -

a) Assign different charges to different operations.

b) The amount of charge is called amortized cost.

c) Then there is actual cost of each operation. The amortized cost can be more or less than actual cost.

d) When amortized cost > actual cost, the difference is saved in specific objects called credits.

e) When for particular operation amortized cost < actual cost the stored credits are utilized.

For example - In case of multiple_pop() operation the amortized cost for each operation is O(n)/n.

iii) Potential Method : The working of potential method is as follows -

- Let, D_0 be the initial data structure. Hence for n operations D_0, D_1, D_2.....D_n will be the data structure. Let c_1, c_2, ... c_n denotes the actual cost.
- Let Φ is a potential function which is a real number. $\Phi(D_i)$ is called potential of D_i.
- Let, c_i' be the amortized cost of i^{th} operation

$$c_i' = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{Potential change}}$$

Actual cost

$$\sum_{i=1}^{n} c_i' = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \quad \text{and}$$

$$\sum_{i=1}^{n} c_i' = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

- If $\Phi(D_n) \geq \Phi(D_0)$, then amortized cost is an upper bound of actual cost.

For example - The amortize cost of each stack operation is O(n).

Probabilistic analysis : Refer Q. 33.

1.7 Recurrence Relation

Q.43 What is recurrence relation ?

[JNTU : Part A, Marks 3]

Ans. : The recurrence equation is an equation that defines a sequence recursively. It is normally in following form -

$$T(n) = T(n-1) + n \text{ for } n > 0 \quad \ldots (Q.43.1)$$

$$T(0) = 0 \quad \ldots (Q.43.2)$$

Here equation (Q.43.1) is called **recurrence relation** and equation (Q.43.2) is called **initial condition.** The recurrence equation can have infinite number of sequences. The **general solution** to the recursive function specifies some formula.

Q.44 Explain the method of solving recurrence relation. **[JNTU : Part B, Marks 5]**

Ans. : There are two types of substitution -

- Forward substitution
- Backward substitution.

Forward substitution method - This method makes use of an initial condition in the initial term and value for the next term is generated. This process is continued until some formula is guessed. Thus in this kind of substitution method, we use recurrence equations to generate the few terms.

For example :

Consider a recurrence relation

$$T(n) = T(n-1) + n$$

with initial condition $T(0) = 0$.

Let,

$$T(n) = T(n-1) + n \quad \ldots (Q.44.1)$$

If n = 1 then

$$T(1) = T(0) + 1$$

$$= 0 + 1 \quad \ldots \because \text{Initial condition}$$

$$\therefore \quad \mathbf{T(1) = 1} \quad \ldots (Q.44.2)$$

If n = 2, then

$$T(2) = T(1) + 2$$

$$= 1 + 2 \quad \ldots \because \text{Equation (Q.44.2)}$$

$$\therefore \quad \mathbf{T(2) = 3} \quad \ldots (Q.44.3)$$

If n = 3 then

$$T(3) = T(2) + 3$$

$$= 3 + 3 \quad \ldots \because \text{Equation (Q.44.3)}$$

$$\therefore \quad \mathbf{T(3) = 6} \quad \ldots (Q.44.4)$$

By observing above generated equations we can derive a formula

$$T(n) = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$

We can also denote T(n) in terms of big oh notation as follows -

$$T(n) = O(n^2)$$

But in practice, it is difficult to guess the pattern from forward substitution. Hence this method is not very often used.

Backward substitution method - In this method backward values are substituted recursively in order to derive some formula.

For example -

Consider, a recurrence relation

$$T(n) = T(n-1) + n \quad \ldots (Q.44.5)$$

With initial condition $T(0) = 0$

$$T(n-1) = T(n-1-1) + (n-1) \quad \ldots (Q.44.6)$$

Putting equation (Q.44.6) in equation (Q.44.5) we get,

$$T(n) = T(n-2) + (n-1) + n \quad \ldots (Q.44.7)$$

Let

$$T(n-2) = T(n-2-1) + (n-2) \quad \ldots (Q.44.8)$$

Putting equation (Q.44.8) in equation (Q.44.7) we get,

$$T(n) = T(n-3) + (n-2) + (n-1) + n$$

$$\vdots$$

$$T(n) = T(n-k) + (n-k+1) + (n-k+2) + \ldots + n$$

If $k = n$ then

$$T(n) = T(0) + 1 + 2 + \ldots n$$

$$T(n) = 0 + 1 + 2 + \ldots + n \quad \ldots \because T(0) = 0$$

$$T(n) = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$$

Again we can denote T(n) in terms of big oh notation as

$$T(n) \in O(n^2).$$

Q.45 Solve the following recurrence relation $T(n) = T(n-1) + 1$ with $T(0) = 0$ as initial condition. Also find big oh notation.

[JNTU : Part B, Makrs 5]

Ans. : Let,

$$T(n) = T(n-1) + 1$$

By backward substitution,

$$T(n-1) = T(n-2) + 1$$

$$\therefore \quad T(n) = T(n-1) + 1$$

$$= (T(n-2) + 1) + 1$$

$$\mathbf{T(n) = T(n-2) + 2}$$

Again $T(n-2) = T(n-2-1) + 1$

$$= T(n-3) + 1$$

$\therefore$ $T(n) = T(n-2) + 2$ becomes

$$= (T(n-3) + 1) + 2$$

$$T(n) = T(n-3) + 3$$

$$\vdots$$

$$T(n) = T(n-k) + k \quad \ldots (Q.45.1)$$

If $k = n$ then equation (Q.45.1) becomes

$$T(n) = T(0) + n$$

$$= 0 + n$$

$\ldots \because$ Initial condition $T(0) = 0$

$$T(n) = n$$

$\therefore$ We can denote T(n) in terms of big oh notation as $\mathbf{T(n) = O(n)}$.

Q.46 Solve the recurrence relation

$$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + n}$$

[JNTU : Part B, Marks 5]

Ans. :

With $T(1) = 1$ as initial condition

$$T(n) = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2n$$

$$T(n) = 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$$

$$= 8T\left(\frac{n}{8}\right) + 3n$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 3n$$

$$\vdots$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$T(n) = n \cdot T\left(\frac{n}{n}\right) + \log n \cdot n$$

... If we assume $2^k = n$

$$= n \cdot T(1) + n \cdot \log n$$

$$T(n) = n + n \log (n) \quad \ldots \because T(1) = 1$$

i.e. $T(n) = n \log n$

Hence in terms of big oh notation -

$$T(n) = O(n \log n)$$

Q.47 What is Master's theorem ?

[JNTU : Part B, May-17, Marks 5]

Ans. : Consider the following recurrence relation -

$$T(n) = aT(n/b) + F(n)$$

where $n \geq d$ and d is some constant.

Then the Master theorem can be stated for efficiency analysis as -

If $F(n)$ is $\Theta(n^d)$ where $d \geq 0$ in the recurrence relation then,

1. $T(n) = \Theta(n^d)$ if $a < b^d$
2. $T(n) = \Theta(n^d \log n)$ if $a = b$
3. $T(n) = \Theta(n^{\log_b a})$ if $a > b^d$

Let us understand the Master theorem with some examples :

Example 1 :

$$T(n) = 4T(n/2) + n$$

We will map this equation with

$$T(n) = aT(n/b) + f(n)$$

Now $f(n)$ is n i.e. n^1. Hence $d = 1$.

$a = 4$ and $b = 2$ and

$a > b^d$ i.e. $4 > 2^1$

$$\therefore \quad T(n) = \Theta(n^{\log_b a})$$

$$= \Theta(n^{\log_2 4})$$

$$= \Theta(n^2) \quad \ldots \because \log_2 4 = 2$$

Hence time complexity is $\Theta(n^2)$.

Another variation of Master theorem is

For $T(n) = aT(n/b) + f(n)$ if $n \geq d$

1. If $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then
   $T(n) = \Theta(n^{\log_b a})$
2. If $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then
   $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
3. If $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then
   $T(n)$ is $= \Theta(f(n))$

Example 2 :

$$T(n) = 2T(n/2) + n \log n$$

Here $f(n) = n \log n$

$a = 2$, $b = 2$

$\log_2 2 = 1$

According to case 2 given in above Master theorem

$f(n) = \Theta(n^{\log_2 2} \log^1 n)$ i.e. $k = 1$

Then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

$= \Theta(n^{\log_2 2} \log^2 n) = \Theta(n^1 \log^2 n)$

$\therefore \quad T(n) = \Theta(n \log^2 n)$

Q.48 Let a, b, c be numbers such that $0 < a, b < 1$ and $c > 0$. Let $T(n)$ be defined by $T(n) = T(an) + T(bn) + cn$.
a) Show that if $(a + b) < 1$ then $T(n)$ is bounded by linear function.
b) Does there exist a, d such that, for all a, b, c above $T(n) = O(n^d)$? **[JNTU : Part B, Marks 5]**

Ans. : a) To show that $T(n)$ is bounded by linear function when $(a + b) < 1$, we will consider $a = \frac{3}{4}$, $b = \frac{1}{5}$.

The recurrence equation will then be

$$T(n) = T\left(\frac{3}{4}n\right) + T\left(\frac{1}{5}n\right) + cn$$

The recurrence tree will be -

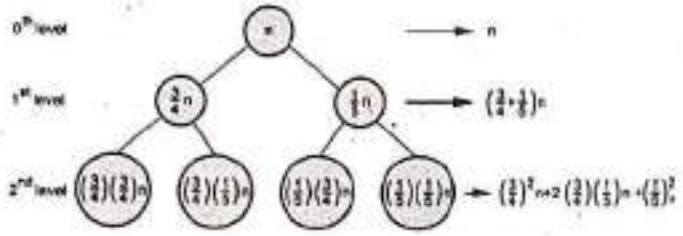


Fig. Q.48.1

At third level we will get -

$$\left(\frac{3}{4}\right)^3 n + 3\left(\frac{3}{4}\right)^2\left(\frac{1}{5}\right)n + 3\left(\frac{3}{4}\right)\left(\frac{1}{5}\right)^2 n + \left(\frac{1}{5}\right)^3 n$$

Thus $\left(\frac{3}{4}+\frac{1}{5}\right)^i n$ pattern works for i^{th} level

$\left(\frac{3}{4}+\frac{1}{5}\right)^i n = \left(\frac{19}{20}\right)^i n$ for i^{th} level. Then by summing up total amount of work done by T(n) will be

$$T(n) = \sum_{0}^{\log n}\left(\frac{19}{20}\right)^i n$$

$$T(n) = cn$$

Hence $T(n) = O(n)$

This proves that T(n) is bounded by linear function.

b) If a = b then, we can write, in following form

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

By Master's theorem

$T(n) = O(n^d)$ when $a < b^d$.

To obtain $T(n) = O(n^d)$ for all a, b, c

Such that $0 < a, b < 1$ and $c > 0$ the $a = b$, $a < b^d$ and $d < 1$.

Q.49 Solve the following recurrence relations and give a Ω bound for each of them

a) $T(n) = 2\,T(n/3) + 1$
b) $T(n) = 5\,T(n/4) + n$
c) $T(n) = 9\ \ T(n/3) + n^2$
d) $T(n) = 49\,T(n/25) + n^{3/2}\log n$
e) $T(n) = T(n-1) + n^c$, where $C \geq 1$, a constant.

[JNTU : Part B, Marks 5]

Ans. : a)

$$T(n) = 2T\left(\frac{n}{3}\right) + 1$$

$$T(n) = 2\left(2T\left(\frac{n}{9}\right)+1\right) + 1$$

$$= 4T\left(\frac{n}{9}\right) + 2$$

$$= 4\left(2T\left(\frac{n}{27}\right)+1\right) + 2$$

$$= 8\,T\left(\frac{n}{27}\right) + 3$$

$$T(n) = 2^3\,T\left(\frac{n}{3^3}\right) + 3$$

$$\vdots$$

$$T(n) = 2^k\,T\left(\frac{n}{3^k}\right) + k$$

If we assume $3^k = n$ then

$$T(n) = 2^k\,T\left(\frac{n}{n}\right) + k$$

$$T(n) = 2^k\,T(1) + k \quad \text{... Assuming } T(1) = 1$$

$$= 2^k + k$$

$$T(n) = 2^{\log_3 n} + \log_3 n$$

and the Ω bound of above recurrence will be

$$T(n) = \Omega\,(2^{\log_3 n})$$

b) Let

$$T(n) = 5\,T\left(\frac{n}{4}\right) + n$$

Here $a = 5$, $b = 4$, $f(n) = n$

$$\log_b a = \log_4 5 = 1.16$$

According to case 1 of Master's theorem

$$f(n) = O\,(n^{\log_b a - \varepsilon}) = O\,(n^{1.16 - 0.16})$$

then

$$T(n) = \Theta\,(n^{\log_b a}) = \Theta\,(n^{\log_4 5})$$

$$T(n) = \Theta\,(n^{1.16})$$

The omega bound can be

$$T(n) = \Omega\,(n^{\log_4 5})$$

c) $T(n) = 9\,T(n/3) + n^2$

By Master's Theorem,

Here $a = 9, b = 3, f(n) = n^2$

$$\log_b a = \log_3 9 = 2$$

By case 2 of Master's theorem $a = b^d$

i.e. $9 = 3^2$

Hence $T(n) = \Theta(n^d \log n)$

$T(n) = \Omega(n^2 \log n)$

d) $T(n) = 49\ T(n / 25) + n^{3/2} \log n$

$f(n) = n^{3/2} \log n = \Omega(n^{\log_{25} 49+\varepsilon})$

$\therefore$ $T(n) = \Theta(f(n))$

$T(n) = \Theta(n^{3/2} \log n)$

The omega bound can be

$T(n) = \Omega(n^{3/2} \log n)$

e) $T(n) = T(n-1) + n^c$

$T(n-1) = T(n-2) + (n-1)^c + n^c$

$T(n) = T(n-k) + (n-k-1)^c + n^c$

If $x = n - 1$ then

$T(n) = T(1) + (n - n + 1 - 1)^c + n^c$

$$= 1 + \sum_{i=2}^{n} i^c$$

$\therefore$ $T(n) = \Theta(n^{c+1})$

$\mathbf{T(n) = \Omega(n^{c+1})}$

Q.50 Solve the following recurrence relations and give a Ω bound for each of them.

a) $T(n) = 7\ T(n/7) + n$

b) $T(n) = 8\ T(n/2) + n^3$

c) $T(n) = T(n-1) + 2$

d) $T(n) = T(\sqrt{n}) + 1$

e) $T(n) = T(n-1) + c^n$, where $c > 1$, a constant

☞ [JNTU : Part B, Marks 5]

Ans. :

a) Let,

$T(n) = 7\ T(n/7) + n$

Here $a = 7, b = 7, f(n) = n$

$\log_b a = \log_7 7 = 0.845$

As $a = b$, by Master's theorem,

$T(n) = \Theta(n^d \log n)$

As $f(n) = \Theta(n^d) = n$ $\therefore d = 1$

Hence $\mathbf{T(n) = \Omega(n \log n)}$

b) Let,

$T(n) = 8\ T(n/2) + n^3$

Here $a = 8, b = 2, f(n) = n^3$

$\log_b a = \log_2 8 = 3$

$f(n) = n^3 = n^{\log_b a}$ i.e. case 2 of Master's theorem

Hence

$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

$\therefore$ $\mathbf{T(n) = \Omega(n^3 \log n)}$

c) $T(n) = T(n-1) + 2$

By backward substitution,

$T(n-1) = T(n-2) + 2$

$\therefore$ $T(n) = T(n-1) + 2$

$T(n) = (T(n-2) + 2) + 2$

$T(n) = T(n-2) + 4$

Similarly,

$T(n) = T(n-3) + 6$

$\vdots$

$T(n) = T(n-k) + 2k$

If $k = n - 1$ then

$T(n) = T(n - n + 1) + 2(n - 1)$

$T(n) = T(1) + (2n - 2)$

$T(n) = 1 + 2n - 2$ ... $\because T(1) = 1$

Then we can represent recurrence as

$\mathbf{T(n) = \Omega(n)}$

d) Let,

$T(n) = T(\sqrt{n}) + 1$

We will put $n = 2^m$. Then we get

$T(2^m) = T(2^{m/2}) + 1$

We will write $T(2^m) = S(m)$

$$S(m) = S(m/2) + 1$$

Here $a = 1, b = 2, f(n) = 1$

$$\log_b a = \log_2 1 = 0$$

$f(m) = 1 = m^0 = m^{\log_b a}$ i.e. Case 2 of Master's theorem

By Master's theorem

$\therefore$ $S(m) = \Theta(\log m)$

i.e. $T(n) = \Omega(\log \log n)$

e) Let,

$$T(n) = T(n-1) + c^n \quad \text{where} \quad c > 1$$

By backward substitution,

$$T(n) = T(n-1) + c^n$$

$$T(n) = (T(n-2) + c^{n-1}) + c^n$$

$$T(n) = T(n-2) + c^{n-1} + c^n$$

Similarly,

$$T(n) = T(n-3) + c^{n-2} + c^{n-1} + c^n$$

$$\vdots$$

$$T(n) = T(1) + \sum_{i=2}^{n} c^i$$

Thus $T(n) = \Omega(c^n)$ for $c > 1$

Q. 51 Suppose you are choosing between the following three algorithms :
a) Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
b) Algorithm B solves problems of size n by recursively solving two subproblems of size (n-1) and then combining the solutions in constant time.
c) Algorithm C solves problems of size n by dividing them into nine subproblems of size n=3, recursively solving each subproblem, and then combining the solutions in O(n2) time. What are the running times of each of these algorithms (in big-O notation), and which would you choose ?

[JNTU : Part B, Marks 5]

Ans. : We will write the recurrence relation for each problem for computing its running time.

a) $T(n) = 5T\left(\frac{n}{2}\right) + n$

Apply Master's theorem

$$T(n) = aT\left(\frac{n}{b}\right) + F(n)$$

Here, $a = 5, b = 2, F(n) = n^1 \quad \therefore d = 1$

As $a > b^d$ i.e. $5 > 2^1$

$$T(n) = \Theta(n^{\log_b a})$$

$$\mathbf{T(n) = \Theta(n^{\log_2 5})}$$

b) $T(n) = 2T(n-1) + C$

where C denotes a constant time. By solving recurrence by backward substitution method, we get.

$$T(n) = 2(2T(n-2) + C) + C$$

$$= 2^2 T(n-2) + 2C + C$$

$$= 2^2 (2T(n-3) + C) + 3C$$

$$= 2^3 T(n-3) + 2^2 C + 3C$$

$$= 2^n - 1\, C \quad \text{... assuming } T(0) = 1$$

$\therefore$ $\mathbf{T(n) = \Theta(2^n)}$

c) $T(n) = 9T\left(\frac{n}{3}\right) + Cn^2$

Apply Master's theorem,

$$T(n) = aT\left(\frac{n}{b}\right) + F(n)$$

Here, $a = 9, b = 3, F(n) = n^2 \quad \therefore d = 2$

As $a = b^d = 9 = 3^2$

$\therefore$ $T(n) = \Theta(n^d \log n)$

$\mathbf{T(n) = \Theta(n^2 \log n)}$

From above three complexities, the third algorithm takes the least running time. Hence third algorithm can be chosen.

END...

UNIT - I

2 Divide and Conquer

2.1 General Method

Important Points to Remember

- In divide and conquer method, a given problem is,
  i) Divided into smaller sub problems.
  ii) These sub problems are solved independently.
  iii) Combining all the solutions of sub problems into a solution of the whole.
- If the sub problems are large enough then divide and conquer is reapplied.
- Various applications of divide and conquer are - 1. Binary Search 2. Merge Sort 3. Quick Sort 4. Strassen's matrix 5. Convex Hull Problems.

Q.1 Write and explain the control abstraction for divide and conquer and give the time complexity.

[JNTU : Part B, May-09,13, Marks 8, April-11, 12, Marks 7]

Ans. : A control abstraction for divide and conquer is as given below - using control abstraction a flow of control of a procedure is given.

Algorithm DC(P)

```
{
    if P is too small then
    return solution of P.
    else
    {
    Divide (P) and obtain P1,P2,.....Pn
    where n ≥ 1
Apply DC to each subproblem
    return combine (DC(P1),DC(P2)... (DC(Pn));
    }
}
```

Generalized Recurrence : Let recurrence relation be

$$T(n) = aT(n/b) + f(n)$$

Consider $a \geq 1$ and $b \geq 2$. Assume $n = b^k$, where $k = 1, 2, \ldots$

$$T(b^k) = aT(b^k/b) + f(b^k)$$
$$= aT(b^{k-1}) + f(b^k)$$
$$= a[aT(b^{k-2}) + f(b^{k-1})] + f(b^k)$$
$$= a^2T(b^{k-2}) + af(b^{k-1}) + f(b^k)$$

Now substituting $T(b^{k-2})$ by using back substitution,

$$= a^2[aT(b^{k-3}) + f(b^{k-2})] + af(b^{k-1}) + f(b^k)$$
$$= a^3T(b^{k-3}) + a^2f(b^{k-2}) + af(b^{k-1}) + f(b^k)$$

Continuing in this fashion we get,

$$= a^kT(b^{k-k}) + a^{k-1}f(b^1) + a^{k-2}f(b^2) + \ldots + a^0f(b^k)$$
$$= [a^kT(1) + a^{k-1}f(b) + a^{k-2}f(b^2) + \ldots + a^0f(b^k)]$$

This can also be written as,

$$= a^kT(1) + \frac{a^k}{a}f(b) + \frac{a^k}{a^2}f(b^2) + \ldots + \frac{a^k}{a^k}f(b^k)$$

Taking a^k as common factor

$$= a^k\left[T(1) + \frac{f(b)}{a} + \frac{f(b^2)}{a^2} + \ldots + \frac{f(b^k)}{a^k}\right]$$
$$= a^k\left[T(1) + \sum_{j=1}^{k}\frac{f(b^j)}{a^j}\right]$$

By property of logarithm,

$$a^{\log_b x} = x^{\log_b a}$$

Hence we can write a^k as

$$a^k = a^{\log_b n} = n^{\log_b a}$$

We can rewrite the equation,

$$T(n) = a^k \left[T(1) + \sum_{j=1}^{k} f(b^j)/a^j \right]$$

$$T(n) = n^{\log_b a} \left[T(1) + \sum_{j=1}^{\log_b n} f(b^j)/a^j \right]$$

Thus order of growth of T(n) depends upon values of constants a and b and order of growth of function f(n).

Q.2 What are the applications of divide and conquer method ?

Ans. : Various applications that use the divide and conquer strategy are -

1. Binary search
2. Merge sort
3. Quick sort
4. Strassen's Matrix multiplication.
5. Finding maximum and minimum element.

2.2 Binary Search

Q.3 What is Binary Search ?

[JNTU : Part A, Marks 3]

Ans : Binary search is an efficient searching method. While searching the elements using this method the most essential thing is that the elements in the array should be **sorted one.**

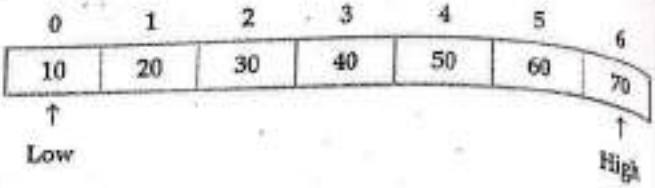
An element which is to be searched from the list of elements stored in array A[0...n – 1] is called KEY element.

Let A[m] be the mid element of array A. Then there are three conditions that needs to be tested while searching the array using this method

1. If KEY = A[m] then desired element is present in the list.
2. Otherwise if KEY < A[m] then search the left sub list
3. Otherwise if KEY > A[m] then search the right sub list.

Q.4 Search the element 60 from the list of elements as 10, 20, 30, 40, 50, 60, 70.

Ans. :



The KEY element (i.e. the element to be searched) is 60.

Now to obtain middle element we will apply formula :

$$m = (low + high)/2$$

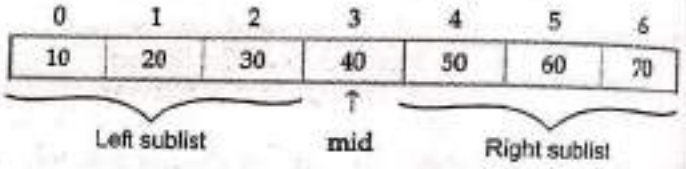
$$m = (0 + 6)/2$$

$$m = 3$$

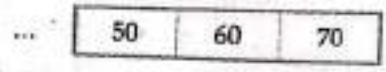
Then Check $A[m] \stackrel{?}{=} KEY$

i.e. $A[3] \stackrel{?}{=} 60$ NO $A[3] = 40$ and $40 < 60$

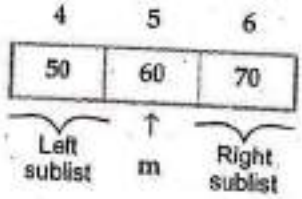
∴ Search the right sublist.



The right sublist is



Now we will again divide this list and check the mid element.



$m = (low + high)/2$

$m = (4 + 6)/2$

$\therefore m = 5$

is $A[m] \stackrel{?}{=} KEY$

i.e. $A[5] \stackrel{?}{=} 60$ Yes, i.e. the number is present in the list.

Thus we can search the desired number from the list of elements.

Q.5 Write recursive and non recursive algorithm for binary search using divide and conquer method. [JNTU : Part B, May-17, Marks 5]

Ans. : Algorithm (Non-recursive)

```
Algorithm BinSearch(A[0...n-1],KEY)
//Problem Description: This algorithm is for searching
//the element using binary search method.
//Input: An array A from which the KEY element is to be
//searched.
//Output: It returns the index of an array element if it is
//equal to KEY otherwise it returns -1
low ← 0
high ← n-1

while (low<high) do
{
 m ← (low+high)/2//mid of the array is obtained
   if (KEY=A[m])then
   return m
 else if(KEY<A[m])then
   high ← m-1        //search the left sub list
 else
   low ← m+1         //search the right sub list
}
return-1             //if element is not present in the list
```

Algorithm (Recursive)

```
  Algorithm BinSearch(A,KEY,low,high)
{
 //Problem Description: This algorithm is for searching
 //the element using binary search method
 //Input: A is an array of elements in which the desired
 //element is to be searched
 //KEY is the element that has to be searched
 //Output: It returns the index of the array element if
 //the KEY element is found.
 //initially  low=0; and high=n -1 where n is total
 //number
 //of elements in the list
  m=(low+high)/2; //mid of the array is obtained
  if(KEY=A[m])then
          return m;
  else if(KEY<A[m]) then
    BinSearch(A,KEY,low,m - 1);
    //search the left sub list
  else
    BinSearch(A,KEY,m+1,high);//search the right sub
//list
}
```

Q.6 Determine the time complexity of binary search method. [JNTU : Part B, May-17, Marks 5]

Ans. : Worst Case :

The worst case time complexity is given by

$$C_{worst}(n) = \underbrace{C_{worst}(n/2)}_{\text{Time required to compare left sublist or right sublist}} + \underbrace{1}_{\text{One comparison made with middle element}} \quad \text{for } n > 1$$

Also, $C_{worst}(1) = 1$

But as we consider the rounded down value when array gets divided the above equations can be written as

$$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1 \quad \ldots (Q.6.1)$$

$$C_{worst}(1) = 1 \quad \ldots (Q.6.2)$$

The above recurrence relation can be solved further. Assume $n = 2^k$ the equation (Q.6.1) becomes,

$$C_{worst}(2^k) = C_{worst}(2^k/2) + 1$$

$$C_{worst}(2^k) = C_{worst}(2^{k-1}) + 1 \quad \ldots (Q.6.3)$$

Using backward substitution method, we can substitute

$$C_{worst}(2^{k-1}) = C_{worst}(2^{k-2}) + 1$$

Then equation (Q.6.3) becomes,

$$C_{worst}(2^k) = [C_{worst}(2^{k-2}) + 1] + 1$$
$$= C_{worst}(2^{k-2}) + 2$$

Then $C_{worst}(2^k) = [C_{worst}(2^{k-3}) + 1] + 2$

$\therefore$ $C_{worst}(2^k) = C_{worst}(2^{k-3}) + 3$

...

...

...

$$C_{worst}(2^k) = C_{worst}(2^{k-k}) + k$$
$$= C_{worst}(2^0) + k$$

$$C_{worst}(2^k) = C_{worst}(1) + k \quad \text{... (Q.6.4)}$$

But as per equation (Q.6.2),

$$C_{worst}(1) = 1$$

put this value in equation (Q.6.4),

$$C_{worst}(2^k) = 1 + k$$

$$\therefore C_{worst}(n) = 1 + \log_2 n$$

$$\therefore C_{worst}(n) = \log_2 n \quad \text{for } n > 1$$

As we have assumed $n = 2^k$
Taking logarithm (base 2) on both sides
$\log_2 n = \log_2 2^k$
$\log_2 n = k \cdot \log_2 2$
$\log_2 n = k(1) \because \log_2 2 = 1$
$\therefore k = \log_2 n$

The worst case time complexity of binary search is $\Theta(\log_2 n)$.

Average Case

- In average case, the search target is equally likely to be located in any array position.
- The average number of probes made by binary search algorithm into an array of n elements is given by

$$(1.1+2.2+4.3+8.4+\ldots+2^{k-1}\cdot k)/n$$

$$= \left(\sum_{i=1}^{k}(i\cdot 2^{i-1})\right)\Big/(2^k-1) \quad \because \text{Assume } n = 2^k - 1$$

$$= \frac{1}{2}\left(\sum_{i=1}^{k}2\cdot(i\cdot 2^{i-1})\right)\Big/(2^k-1)$$

$$= \frac{1}{2}\left(\sum_{i=1}^{k}(i\cdot 2^{i})\right)\Big/(2^k-1)$$

$$= \left(\sum_{i=1}^{k}(i\cdot 2^{i})\right)\Big/(2\cdot(2^k-1))$$

$$= 2\cdot((k-1)\cdot 2^k+1)/(2\cdot(2^k-1))$$

$$= ((k-1)\cdot 2^k+1)/(2^k-1)$$

Substituting back in terms of n,

where $n = 2^k - 1 \quad \therefore 2^k = n+1$

i.e. $k = \log_2(n+1)$

We will get -

$$((\log_2(n+1)-1)\cdot(n+1)+1)/n$$

$$= (n\cdot\log_2(n+1)-n+\log_2(n+1)-1+1)/n$$

$$= \log_2(n+1)-1+(\log_2(n+1))/n$$

Thus

The average case complexity is $\Theta(\log_2 n)$

Best case

The best case of binary search occurs when the element you are searching for is the middle element of the list/array

Hence

The best case complexity is $\Theta(1)$

Q.7 What are advantages and disadvantages of binary search ?

Ans. :

Advantage of binary search :

1. Binary search is an optimal searching algorithm using which we can search the desired element very efficiently.

Disadvantage of binary search :

1. This algorithm requires the list to be sorted. Then only this method is applicable.

Q.8 Enlist the applications of binary search method.

Ans. :

1. The binary search is an efficient searching method and is used to search desired record from database.
2. For solving nonlinear equations with one unknown this method is used.

Q.9 Design Divide and conquer algorithm for computing the number of levels in a binary tree.

☞ [JNTU : Part B, May-09, Marks 8]

Ans. :

```
Algorithm Level_Count(root)
{
  if(root==NULL) then
    return 0;
  if(root->left== NULL AND root ->right==NULL)
then
    return 0;
   d1 ← Level_Count (root->left);
```

```
    //computing height of leftsubtree
    d2 ← Level_Count (root->right);
    //computing height of rightsubtree
  if(d1>d2) then
   return d1+1;
  else
   return d2+1;
    //maximum level = height of the tree
}
```

Q.10 Solve the recurrence relation of formula

$$T(n) = \begin{cases} g(n), & n \text{ is small} \\ 2T(n/2) + F(n), & \text{otherwise} \end{cases}$$

when

i) $g(n) = O(1)$ and $f(n) = O(n)$

ii) $g(n) = O(1)$ and $f(n) = O(1)$.

[JNTU : Part B, May-09, Marks 8, May-12, Marks 15]

Ans. : i) Let, $g(n)$ $O(1)$ and $f(n) = O(n)$.

Then the given recurrence relation is -

$$T(n) = 2T(n/2) + F(n)$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + F(n)\right] + F(n)$$

If we assume $F(n) = O(n)$ then

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + O(n)\right] + O(n)$$

$$= 4T\left(\frac{n}{4}\right) + O(n) + O(n)$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 2\,O(n)$$

$$T(n) = 4T\left(2T\left(\frac{n}{8}\right) + O(n)\right) + 2\,O(n)$$

$$T(n) = 2^3\,T\left(\frac{n}{8}\right) + 3\,O(n)$$

$$\vdots$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + k \cdot O(n)$$

But if we assume $2^k = n$ then $k = \log n$.

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + \log n \cdot O(n)$$

If $2^k = n$ then

$$T(n) = n \cdot T\left(\frac{n}{n}\right) + \log n \cdot O(n)$$

$$T(n) = n \cdot T(n) + \log n \cdot O(n)$$

$$T(n) = g(n) \quad \text{When n is small, and } g(n) = O(1)$$

$$\therefore \quad T(n) = n \cdot O(1) + \log n \cdot O(n)$$

$$\therefore \quad T(n) = n \log n$$

ii) $g(n) = O(1)$ and $F(n) = O(1)$ then the recurrence relation is

$$T(n) = 2\,T\left(\frac{n}{2}\right) + F(n) = 2\left[2T\left(\frac{n}{4}\right) + F(n)\right] + F(n)$$

$$= 4T\left(\frac{n}{4}\right) + 2\,F(n) = 2^2\,T\left(\frac{n}{2^2}\right) + 2\,F(n)$$

$$\vdots$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + k \cdot F(n)$$

If we assume $n = 2^k$ then $k = \log n$

$$= nT\left(\frac{n}{n}\right) + \log n \cdot F(n)$$

$$T(n) = nT(1) + \log n \cdot F(n)$$

$T(1)$ is small hence we assume $T(1) = g(n) = O(1)$

$$T(n) = n \cdot O(1) + \log n \cdot O(1)$$

$$T(n) = \log n$$

Q.11 Modify binary search of text so that in the case of unsuccessful search it returns the index i such that k(i) < key < k (i +1).

[JNTU : Part B, Dec.-09, Marks 8]

Ans. :

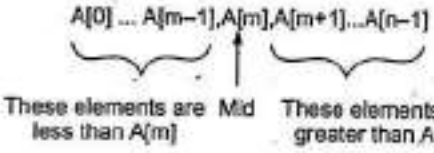
```
Algorithm BinSearch(A[0:n-1],KEY)
{
 //Problem Description : This algorithm returns index
//for an unsuccessful search using Binary search
 //Input : The array A from which the KEY element
//needs to be searched
 //Output : returns the index on unsuccessful search
 low←0;
 high←n-1;
 while(low<=high)do
 {
     m←(low+high)/2;  //mid of the array is obtained
     If(KEY==A[m]) then
```

```
        Write("The element is present in the list");
    else if(KEY<A[m]) then
        high←m-1;        //search the left sub list
    else
        low←m+1;         //search the right sub list
}
return m; //returning in an unsuccessful search
}
```

2.3 Quick Sort

Important Points to Remember

- Quick sort is a sorting algorithm that uses the divide and conquer strategy. In this method division is dynamically carried out. The three steps of quick sort are as follows :
- **Divide :** Split the array into two sub arrays that each element in the left sub array is less than or equal the middle element and each element in the right sub array is greater than the middle element. The splitting of the array into two sub arrays is based on pivot element. All the elements that are less than pivot should be in left sub array and all the elements that are more than pivot should be in right sub array.



- **Conquer :** Recursively sort the two sub arrays.
- **Combine :** Combine all the sorted elements in a group to form a list of sorted elements.

Q.12 Show how quick sorts the following sequences of keys in ascending order 22, 55, 33, 11, 99, 77, 55, 66, 54, 21, 32.

[JNTU : Part B, Marks 5, May-09, Marks 16]

Ans. : Step 1 :

22	55	33	11	99	77	55	66	54	21	32
↑	↑									↑
pivot	i									j

Assume the element at first position i.e. 22 as a pivot element. Now compare A[i] with pivot. If A[i] < pivot, increment i. If A[j] > pivot, then decrement j.

22	55	33	11	99	77	55	66	54	21	32
	i								j	

Swap A [i] and A [j]

Step 2 :

22	21	33	11	99	77	55	66	54	55	32
	i	j								

Swap A [i] and A [j]

22	21	11	33	99	77	55	66	54	55	32
		i	j							

Step 3 :

22	21	11	33	99	77	55	66	54	55	32
		i	j							

As j < i we will swap pivot element with A [j]

11	21	22	33	99	77	55	66	54	55	32
sublist 1		↑							sublist 2	
		pivot								

Step 4 : We need to sort two sublist independently.

11	21
↑	i / j
Pivot	

No swapping takes place.

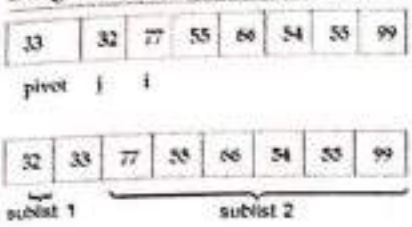
Step 5 : Consider sublist 2 for sorting.

33	99	77	55	66	54	55	32
↑	↑						↑
pivot	i						j

As A [i] > pivot element and A [j] < pivot element we need not have to increment or decrement i or j. Just swap A [j] and A [i] elements.

33	32	77	55	66	54	55	99
	i						j

Step 6 : Go on incrementing i when A [i] < pivot element and decrementing j when A [j] > pivot element.

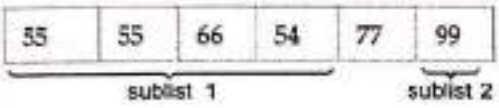


As j has crossed i, swap pivot element with A [j].

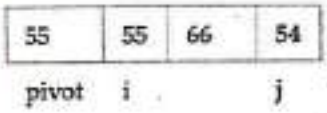
Step 7 : We will sort sublist 2,

Go on incrementing i when A[i] < pivot element and decrementing j when A[j] > pivot element.

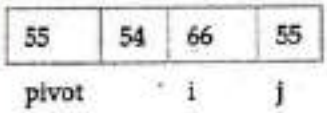
Step 8 : As j has crossed i, we will swap A[j] and pivot element.



Step 9 : Just sort sublist 1 by assuming 55 as pivot

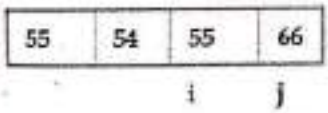


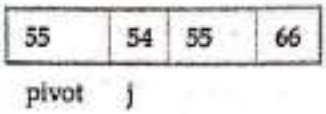
As i>= pivot element A[j] < A[piovt], just swap A[i] and A[j]



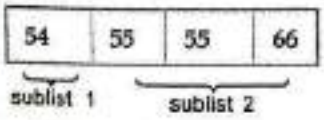
swap A[i] and A[j]

Step 10 :

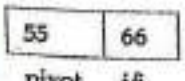




swap A[j] and A[pivot]



Step 11 : Consider sublist 2



There is no swapping

Finally we get a sorted list by combining all sorted sublists.

11	21	22	32	33	54	55	55	66	77	99

Q.13 Apply quick sort to sort the list.
E, X, A, M, P, L, E
In alphabetical order. Draw the tree of recursive calls made.

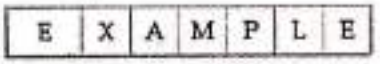
☞ **[JNTU : Part B, Marks 5, Dec.-11, Marks 8]**

Ans. : Let the list be

0	1	2	3	4	5	6
E	X	A	M	P	L	E
low						high

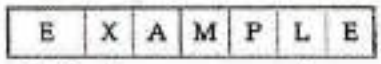
Array A[7]

We assume A[0] = pivot. ∴ The pivot is E . Also set i and j pointers as

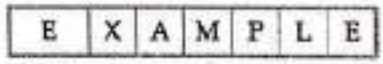


Is pivot > A[i]? = No. Then do not increment i.

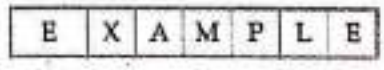
Now if A[j] ≥ pivot, go on decrementing j



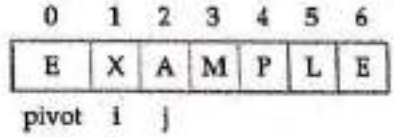
As L > E decrement j.



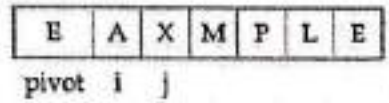
As P > E decrement j.



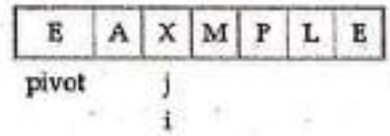
As M > E decrement j.



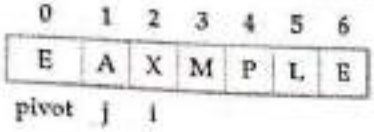
As A < E and X > E. Do not move i and j. But Swap A[i] and A[j]



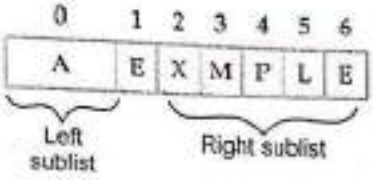
Now A < E. ∴ increment i.



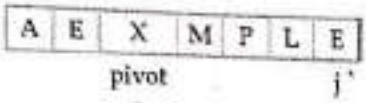
As X > E, decrement j.



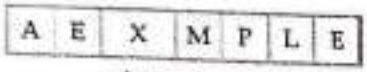
As j > i. We swap pivot and A[j]. Then



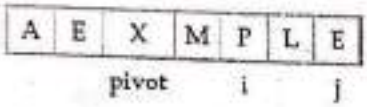
As there is only one element in the left sublist we do not sort left sublist.

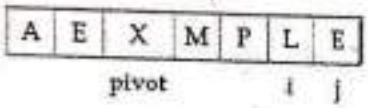


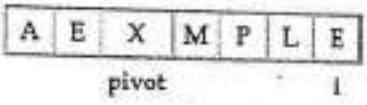
Now set first element of right sublist as pivot. Hence A[2] = pivot. And A[2] = i A[6] = j.

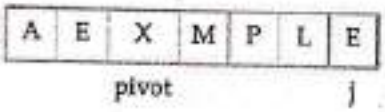


As pivot ≥ A[i] increment i. Go on incrementing i if this condition is satisfied.

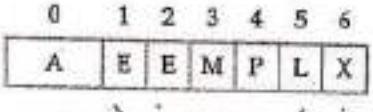




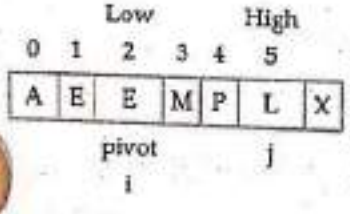




As i > j, swap pivot and A[j].

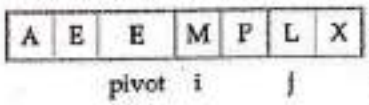


As pivot has occupied its position we get only left sublist and no element in the right sublist. Hence we consider only left sublist.

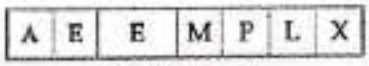


Assume the first element of left sublist i.e. A[2] = pivot. i.e. pivot = E. Similarly set i and j as A[Low] and A[High].

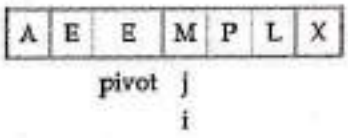
If A[i] ≤ pivot, increment i.



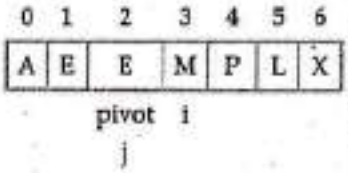
As M > E, do not increment i. But if A[j] > pivot then go on decrement j.



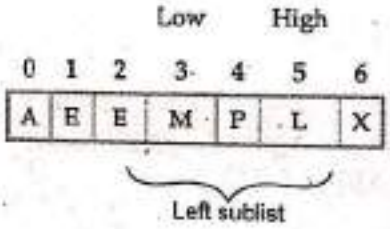
As P > E, decrement j.

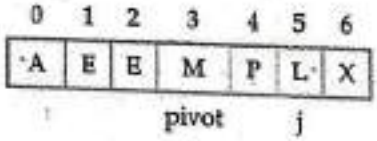


As A[j] > pivot decrement j.

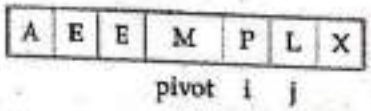


j < i. And A[j] and pivot are same. Thus partitioning is completed. There is no left sublist, we have only right sublist.

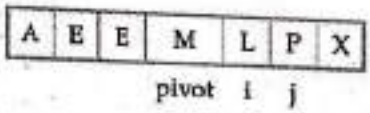


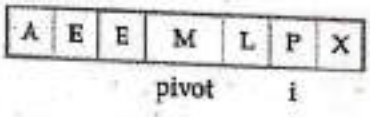


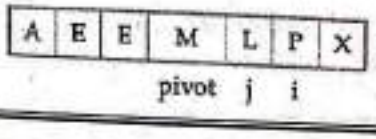
Set pivot, i and j.



As P > M, do not increment i, similarly L < M, do not decrement j. Swap A[i] and A[j].







Swap A[j] and pivot.

0	1	2	3	4	5	6
A	E	E	L	M	P	X

Since there is only one element each in left and right sublist. Further partitioning is not

Hence the sorted list is

A	E	E	L	M	P	X

The tree of recursive calls can be as shown below.

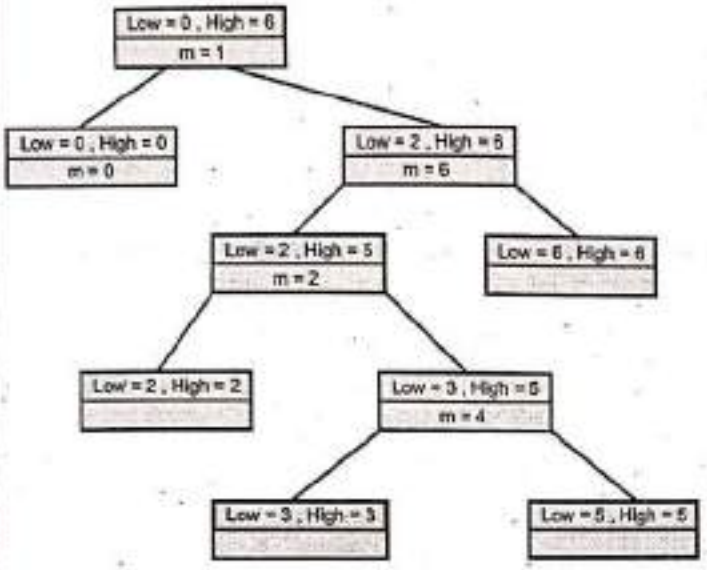


Fig. Q.13.1 Tree for recursive calls

Q.14 Write an algorithm for sorting n numbers using quick sort method. Determine its time complexity. **[JNTU : Part B, Dec.-11, Marks 8]**

Ans. : *Quick* and *partition*. Let us see them -

Algorithm Quick(A[0...n–1], low,high)
//Problem Description : This algorithm performs sorting
//of the elements given in Array A[0...n–1]
//Input : An array A[0...n–1] in which unsorted elements
//are
//given. The low indicates the leftmost element in the
//list
//and high indicates the rightmost element in the list
//Output : Creates a sub array which is sorted in
//ascending
//order
if(low<high)**then**
//split the array into two sub arrays
m ← partition(A[low...high])// m is mid of the array
Quick(A[low...m–1])
Quick(A[mid+1...high])

In above algorithm call to partition algorithm is given. The *partition* performs arrangement of the elements in ascending order. The recursive *quick* routine is for dividing the list in two sub lists. The pseudo code for *Partition* is as given below -

Algorithm Partition (A[low...high])
//Problem Description : This algorithm partitions the
//subarray using the first element as pivot element
//Input : A subarray A with low as left most index of the
//array and high as the rightmost index of the array.
//Output : The partitioning of array A is done and pivot
//occupies its proper position. And the rightmost index
//of the list is returned
pivot ← A[low]
i ← low
j ← high+1
while(i<=j) **do**
{
while(A[i]<=pivot) **do**
i ← i+1
while(A[j]>=pivot) **do**
j ← j–1;
if(i<=j) **then**
swap(A[i],A[j])//swaps A[i] and A[j]
}
swap(A[low],A[j])//when i crosses j swap A[low] and
A[j]
return j//rightmost index of the list

The Partition function is called to arrange the elements such that all the elements that are less than pivot are at the left side of pivot and all the elements that are greater than pivot are all at the right of pivot. In other words pivot is occupying its proper position and the partitioned list is obtained in an ordered manner.

Analysis

Best case (split in the middle)

If the array is always partitioned at the mid, then it brings the best case efficiency of an algorithm.

The recurrence relation for quick sort for obtaining best case time complexity is,

$$C(n) = \underbrace{C(n/2)}_{\text{Time required to sort left sub array}} + \underbrace{C(n/2)}_{\text{Time required to sort right sub array}} + \underbrace{n}_{\text{Time required for partitioning the sub array}}$$

and $C(1) = 0$

$C(n) = 2\,C(n/2) + n$

Here $f(n) \in n^1$ $\therefore d = 1$

Now, $a = 2$ and $b = 2$.

As from case 2 we get $a = b^d$ i.e. $2 = 2^1$, we get

$T(n)$ i.e. $C(n) = \Theta(n^d \log n)$

$\therefore$ $C(n) = \Theta(n \log n)$

Thus,

Best case time complexity of quick sort is $\Theta(n \log_2 n)$.

Worst case (sorted array)

The worst case for quick sort occurs when the pivot is a minimum or maximum of all the elements in the list. This can be graphically represented as

$C(n) = C(n-1) + n$

or $C(n) = n + (n-1) + (n-2) + \ldots + 2 + 1$

But as we know

$$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2} = \frac{1}{2} n^2.$$

$\therefore$ $C(n) = \Theta(n^2)$

The time complexity of worst case of quick sort is $\Theta(n^2)$.

Q.15 Derive the time complexity of the quick sort in an average case.

☞ [JNTU : Part B, Dec.-12, Marks 8]

Ans. : Let, $C_{avg}(n)$ denotes the average time of quick sort (A[1 ... n]).

The recurrence relation for random input array is

$C(n) = C(0) + C(n-1) + n$

or $C(n) = C(1) + C(n-2) + n$

or $C(n) = C(2) + C(n-3) + n$

or $C(n) = C(3) + C(n-4) + n$

...

...

...

$C(n) = C(n-1) + C(0) + n$

The average value of C(n) is sum of all the possible values divided by n.

$$\therefore \quad C_{avg}(n) = \frac{2}{n}\,(C(1)+C(2)+\ldots+C(n-1))+n$$

Multiplying both the sides by n we get,

$$n * C_{avg}(n) = 2\,(C(1)+C(2)+\ldots+C(n-1))+n^2$$

Now we put n = n – 1 then,

$$(n-1) * C_{avg}(n-1) = 2\,[C_{avg}(1)+C_{avg}(2)+\ldots+C_{avg}(n-2)]+(n-1)*(n-1)$$

$$n * C_{avg}(n) - (n-1) * C_{avg}(n-1) = 2[C_{avg}(1)+C_{avg}(2)+\ldots+C_{avg}(n-2)]+(n-1)*(n-1)$$

$$n * C_{avg}(n) - (n-1) * C_{avg}(n-1) = 2C_{avg}\,(n-1) + (2n-1)$$

$$n * C_{avg}(n) = (n+1) * C_{avg}\,(n-1) + (2n-1) < (n+1) * C_{avg}\,(n-1) + 2n$$

If we assume

$$(n+1) * C_{avg}\,(n-1)+2n = (n+1) * C_{avg}\,(n-1)+C'n$$

Then,

$$(n+1) * C_{avg}\,(n-1) < (n+1) * C_{avg}\,(n-1)+C'n$$

Divide this equation by n (n + 1) then

$$n * C_{avg}(n) < (n+1) * C_{avg}(n-1)+C'n$$

$$\frac{C_{avg}(n)}{(n+1)} < \frac{C_{avg}\,(n-1)}{n} + \frac{C'}{(n+1)}$$

If we assume

$$A(n) = \frac{C_{avg}(n)}{(n+1)} \text{ then}$$

$$A(n) < A\,(n-1) + \frac{C'}{(n+1)}$$

$$A(n-1) < A\,(n-2) + \frac{C'}{n}$$

$$A(n-2) < A\,(n-3) + \frac{C'}{n-1}$$

...

...

...

$$A(1) < A\,(0) + \frac{C'}{2}$$

Adding and cancelling these equations we get,

$$A(n) < C' * \left[\frac{1}{2}+\frac{1}{3}+\ldots.+\frac{1}{n}+\frac{1}{n+1}\right]$$

$$\therefore C_{avg}(n) = (n+1) * A(n)$$

i.e. $(n+1) * A(n) < C''(n+1)\log n$

$$\therefore C_{avg}(n) = \Theta\,(n \log n)$$

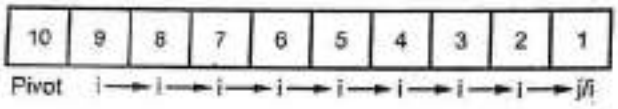
Hence average case time complexity of quick sort is $\Theta(n \log n)$.

Q.16 Assuming that quick sort uses the first item in the list as the pivot item :i) Give a list of n items (for example, an array of 10 integers) representing the worst-case scenario. ii) Give a list of n items (for example, an array of 10 integers) representing in the best-case scenario.

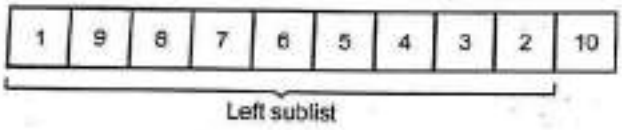
[JNTU : Part B, April-11, Dec.-11, Marks 8]

Ans. :

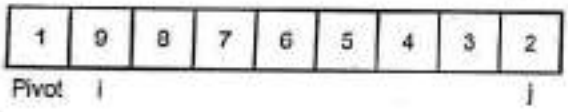
i) The worst case scenario for quick sort is when the pivot always goes to one of the ends of the array. Hence reverse sorted list of elements is usually worst case. For example - Consider following 10 elements that need to be sorted in ascending order.



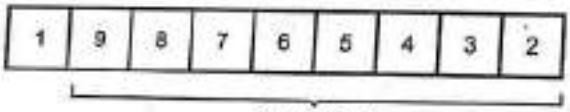
Swap A[j] and A[pivot]



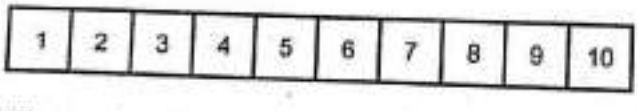
Now sort left sublist considering 1 as pivot element.



There will not be any swapping

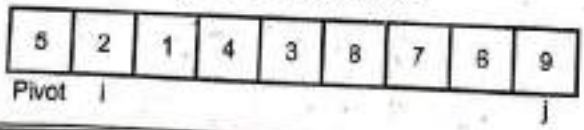


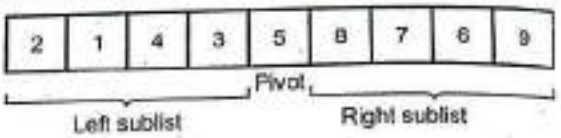
Continuing in this fashion we get finally the sorted list as



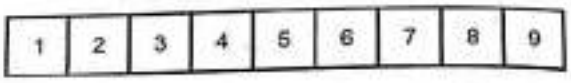
ii) The condition for best case of quick sort is that the pivot always goes in the middle of array.

Consider following list of elements as





Finally we get



Q.17 Is quick sort a stable sorting method ? Justify your answer.

[JNTU : Part B, Dec.-09, April-11, Marks 8]

Ans. : No, quick sort is not a stable sorting algorithm because after applying this sorting method the ordering of similar elements may not be preserved. (Since swapping is involved in this sorting with pivot element).

Q.18 Determine the running time of quick sort for
i) sorted input
ii) recerse - ordered input
iii) random input **[JNTU : Part B, May-09, Marks 8]**

Ans. : Assumption : For finding out the time complexities for given cases we assume that the pivot is somewhere at the middle of the list. Hence running time of quick sort differs based on selection of pivot.

a) **Sorted input :** In sorted input, the left and right pointers of the array simply move without any swapping. Thus the problem is divided into two equal problems. Hence th recurrence relation for this case is -

$$T(n) = 2 * T(n/2) + O(n)$$

Hence time complexity is **O(nlogn)**

b) **Reverse - ordered input :** This case is similar to previous case. But in this case on each increment of left and right pointer the swapping takes place. This affects **O(n)** part of the recurrence relation. But every swap is of O(1) time complexity The division of array still remains the same. Hence the time complexity is **O(nlogn).**

c) **Random input :** This is average case time complexity. The recurrence relation for random input array is -

$$T(n) = T(0) + T(n-1) + n$$

$$T(n) = T(1) + T(n-2) + n$$

$$T(n) = T(2) + T(n-3) + n$$

$$\vdots \quad \vdots$$

$$T(n) = T(n-1) + T(0) + n$$

This turn out to be O(nlog n) time complexity

2.4 Merge Sort

Q.19 Apply merge sort to sort the list E, X, A, M, P, L, E in alphabetical order.

[JNTU : Part B, Marks 5]

Ans. :

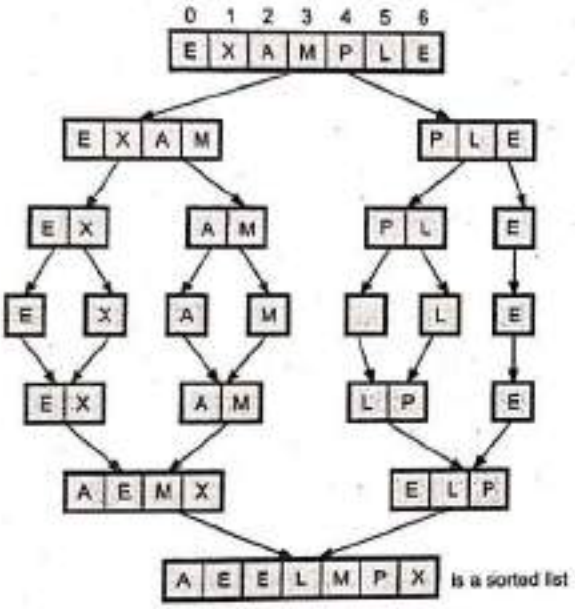


Fig. Q.18.1

Q.20 Explain the merge sort with the example. Write the algorithm of merge sort and the running time of the algorithm.

[JNTU : April-11, May-12, 17, Marks 7]

Ans. : Example – Refer Q. 19.

Algorithm and Time Complexity :

```
Algorithm MergeSort(int A[0...n-1],low,high)
if(low < high)then
  {
    mid ← (low+high)/2          // split the list at mid
    MergeSort(A,low,mid)        // first sublist
    MergeSort(A,mid+1,high)// second sublist
    Combine(A,low,mid,high)
    // merging of two sublists
  }
```

```
Algorithm Combine(A[0...n-1],low, mid, high)
{
  k ← low;// k as index for array temp
  i ← low;     // i as index for left sublist of array A
  j ← mid+1  // j as index for right sublist of array A
  while(i <= mid and j <= high)do
  {
  if(A[i]<=A[j])then
    // if smaller element is present in left sublist
    {
        // copy that smaller element to temp array
        temp[k] ← A[i]
        i ← i+1
        k ← k+1
    }
      else
      //smaller element is present in right sublist
    {
      //copy that smaller element to temp array
      temp[k] ← A[j]
      j ← j+1
      k ← k+1
    }
  }
    //copy remaining elements of left  sublist to temp
      while(i<=mid)do
  {
        temp[k] ← A[i]
        i ← i+1
      k ← k+1
  }
  //copy remaining elements of right sublist to temp
  while(j<=high)do
  {
    temp[k] ← A[j]
    j ← j+1
    k ← k+1
  }
```

Time Complexity Analysis

In merge sort algorithm the two recursive calls are made. Each recursive call focuses on n/2 elements of the list. After two recursive calls one call is made to combine two sublists i.e. to merge all n elements. Hence we can write recurrence relation as

$T(n) = T(n/2) + T(n/2) + cn$

Time taken by left sublist to get sorted + Time taken by right sublist to get sorted + Time taken for combining two sublists

where $n > 1$ $T(1) = 0$

Let, the recurrence relation for merge sort is

$T(n) = T(n/2) + T(n/2) + cn$

i.e. $T(n) = 2T(n/2) + cn$ ... (1)

$T(1) = 0$

Let $T(n) = aT(n/b) + f(n)$ ... (2)

As per Master theorem

$T(n) = \Theta(n^d \log n)$ if $a = b$

As in equation (1),

$a = 2$, $b = 2$ and $f(n) = cn$ i.e. n^d with $d = 1$.

and $a = b^d$

i.e. $2 = 2^1$

This case gives us,

$T(n) = \Theta(n \log_2 n)$

Hence the average and worst case time complexity of merge sort is $\Theta(n \log_2 n)$.

Q.21 Determine the running time of merge sort for
i) sorted input
ii) reverse - ordered input
II) random - ordered input.

[JNTU : Part B, May-09, Marks 8]

Ans. : i) The merge sort is the only sorting algorithm whose time complexity does not get affected by the ordering of the input. The ordering of the input has impact on merge procedure. And merge procedure anyway requires logN iterations. Hence :

i) **Sorted input :** In sorted input, the left subarray elements get combined with right subarray. Hence algorithm is of order **O(nlogn).**

ii) **Reverse - ordered input :** This case is similar to the case i) The only change is that right array gets copied first. Hence the time complexity is still **O(nlogn).**

iii) **Random - ordered input :** This is a normal case in which both the subarrays get combined and then get sorted. Hence time complexity still **O(nlogn).**

Q.22 Suggest refinements to merge sort to make it in place. **[JNTU : Part B, May-09, Marks 8]**

Ans. : The in-place merge sort algorithm is as given below -

```
Algorithm Merge_srt(low,high)
{
//array a[1:n] is for storing the unsorted list of
//elements declared globally,initially low is set at first
//location of array a
//Initially high is set to last location of array a
   if (n==1 ) then
   return;
   else
   {
      if(low<high) then
         mid:=(low+high)/2;
      //consider the floor value of mid.
      Merge_srt(low,mid);
      Merge_srt(mid+1,high);
      Merge(low,mid,high);
   }
}

Algorithm Merge(low,mid,high)
{
   //array a is globally declared
   //Another array aux[1:n] is required to do in-place
     merge
   // i,j are required for pointing out the locations in
     array a
   for (i=mid+1;i>low;i--) do
      aux[i-1]:=a[i--];
   for(j=mid;j<high;j++) do
      aux[high+mid-j]:=a[j+1];
   for(int k=low;k<=high;k++) do
      if(aux[j]<aux[i]) then
         a[k]:=aux[j--];
      else
         a[k]:=aux[i++];
}
```

Q.23 Is merge sort stable sorting algorithm ?
[JNTU : Part A, Marks 3]

Ans. : Yes, merge sort is the stable sorting algorithm. A sorting algorithm is said to be stable if it preserves the ordering of similar (equal) elements after applying sorting method. And merge sort is a method which preserves this kind of ordering. Hence merge sort is a stable sorting algorithm.

2.5 Strassen's Matrix Multiplication

Q.24 Explain in detail the Strassen's matrix multiplication. **[JNTU : Part B, Marks 5]**

Ans. : Strassen showed that 2×2 matrix multiplication can be accomplished in 7 multiplication and 18 additions or subtractions.

The divide and conquer approach can be used for implementing Strassen's matrix multiplication.

- **Divide :** Divide matrices into sub-matrices : A_0, A_1, A_2 etc.
- **Conquer :** Use a group of matrix multiply equations.
- **Combine :** Recursively multiply sub-matrices and get the final result of multiplication after performing required additions or subtractions.

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$S_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$S_2 = (A_{21} + A_{22}) \times B_{11}$$

$$S_3 = A_{11} \times (B_{12} - B_{22})$$

$$S_4 = A_{22} \times (B_{21} - B_{11})$$

$$S_5 = (A_{11} + A_{12}) \times B_{22}$$

$$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$

$$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$C_{11} = S_1 + S_4 - S_5 + S_7$$

$$C_{12} = S_3 + S_5$$

$$C_{21} = S_2 + S_4$$

$$C_{22} = S_1 + S_3 - S_2 + S_6$$

Now we will compare the actual our traditional matrix multiplication procedure with Strassen's procedure. In Strassen's multiplication

$$\begin{aligned} C_{11} &= S_1 + S_4 - S_5 + S_7 \\ &= (A_{11} + A_{22})(B_{11} + B_{22}) + A_{22} \times (B_{21} - B_{11}) \\ &\quad - (A_{11} + A_{12}) \times B_{22} + (A_{12} - A_{22}) \\ &\quad \times (B_{21} + B_{22}) \\ &= A_{11} B_{11} + A_{11} B_{22} + A_{22} B_{11} + A_{22} B_{22} \\ &\quad + A_{22} B_{21} - A_{22} B_{11} - A_{11} B_{22} - A_{12} B_{22} \\ &\quad + A_{12} B_{21} + A_{12} B_{22} - A_{22} B_{21} - A_{22} B_{22} \\ &= A_{11} B_{11} + A_{12} B_{21} \end{aligned}$$

Q.25 Compute the product of the following matrices of 4 × 4 size, using Strassen's multiplication method.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 8 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{bmatrix} \quad B = \begin{bmatrix} 8 & 9 & 1 & 2 \\ 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \end{bmatrix}$$

[JNTU : Part B, April-11, Marks 8]

Ans. : The given matrix is of order 4×4. Hence we will subdivide it in 2×2 submatrices. Then we will compute S_1, S_2, S_3, S_4, S_5 S_6 and S_7.

Let,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} & \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} \\ \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \end{bmatrix} \times \begin{bmatrix} \begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \\ \begin{bmatrix} 7 & 8 \\ 2 & 3 \end{bmatrix} & \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} \end{bmatrix}$$

Now

$$\begin{aligned} S_1 &= (A_{11} + A_{22}) * (B_{11} + B_{22}) \\ &= \left(\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \right) \left(\begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} \right) \\ &= \begin{bmatrix} 3 & 5 \\ 11 & 13 \end{bmatrix} \begin{bmatrix} 17 & 10 \\ 7 & 9 \end{bmatrix} \\ &= \begin{bmatrix} 51+35 & 30+45 \\ 187+91 & 110+117 \end{bmatrix} \end{aligned}$$

$$S_1 = \begin{bmatrix} 86 & 75 \\ 278 & 227 \end{bmatrix}$$

$$S_2 = (A_{21} + A_{22}) * B_{11}$$

$$= \left(\begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}\right)\begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 11 & 4 \\ 10 & 12 \end{bmatrix}\begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 88+12 & 99+16 \\ 80+36 & 90+48 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} 100 & 115 \\ 116 & 138 \end{bmatrix}$$

$$S_3 = A_{11} * (B_{12} - B_{22})$$

$$= \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}\left(\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} - \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}\right)\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} * \begin{bmatrix} -8 & 1 \\ 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -8+2 & 1+2 \\ -40+6 & 5+6 \end{bmatrix}$$

$$S_3 = \begin{bmatrix} -6 & 3 \\ -34 & 11 \end{bmatrix}$$

$$S_4 = A_{22} * (B_{21} - B_{11})$$

$$= \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} * \left(\begin{bmatrix} 7 & 8 \\ 2 & 3 \end{bmatrix} - \begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} * \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} -2-3 & -2-3 \\ -6-7 & -6-7 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} -5 & -5 \\ -13 & -13 \end{bmatrix}$$

$$S_5 = (A_{11} + A_{12}) * B_{22}$$

$$= \left(\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}\right) * \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 6 \\ 12 & 14 \end{bmatrix} * \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 36+24 & 4+30 \\ 108+56 & 12+70 \end{bmatrix}$$

$$S_5 = \begin{bmatrix} 60 & 34 \\ 164 & 82 \end{bmatrix}$$

$$S_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$$

$$= \left(\begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}\right) * \left(\begin{bmatrix} 8 & 9 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 8 & -1 \\ -1 & -1 \end{bmatrix} * \begin{bmatrix} 9 & 11 \\ 8 & 10 \end{bmatrix}$$

$$= \begin{bmatrix} 72-8 & 88-10 \\ -9-8 & -11-10 \end{bmatrix}$$

$$S_6 = \begin{bmatrix} 64 & 78 \\ -17 & -21 \end{bmatrix}$$

$$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$= \left(\begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix} - \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}\right)\left(\begin{bmatrix} 7 & 8 \\ 2 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 1 \\ 4 & 5 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * \begin{bmatrix} 16 & 9 \\ 6 & 8 \end{bmatrix}$$

$$= \begin{bmatrix} 16+6 & 9+8 \\ 16+6 & 9+8 \end{bmatrix}$$

$$S_7 = \begin{bmatrix} 22 & 17 \\ 22 & 17 \end{bmatrix}$$

$$C_{11} = S_1 + S_4 - S_5 + S_7$$

$$= \begin{bmatrix} 86 & 75 \\ 278 & 227 \end{bmatrix} + \begin{bmatrix} -5 & -5 \\ -13 & -13 \end{bmatrix} - \begin{bmatrix} 60 & 34 \\ 164 & 82 \end{bmatrix} + \begin{bmatrix} 22 & 17 \\ 22 & 17 \end{bmatrix}$$

$$C_{11} = \begin{bmatrix} 43 & 53 \\ 123 & 149 \end{bmatrix}$$

$$C_{12} = S_3 + S_5$$

$$= \begin{bmatrix} -6 & 3 \\ -34 & 11 \end{bmatrix} + \begin{bmatrix} 60 & 34 \\ 164 & 82 \end{bmatrix}$$

$$C_{12} = \begin{bmatrix} 54 & 37 \\ 130 & 93 \end{bmatrix}$$

$$C_{21} = S_2 + S_4$$

$$= \begin{bmatrix} 100 & 115 \\ 116 & 138 \end{bmatrix} + \begin{bmatrix} -5 & -5 \\ -13 & -13 \end{bmatrix}$$

$$C_{21} = \begin{bmatrix} 95 & 110 \\ 103 & 125 \end{bmatrix}$$

$$C_{22} = S_1 + S_3 - S_2 + S_6$$

$$= \begin{bmatrix} 86 & 75 \\ 278 & 227 \end{bmatrix} + \begin{bmatrix} -6 & 3 \\ -34 & 11 \end{bmatrix} - \begin{bmatrix} 100 & 115 \\ 116 & 138 \end{bmatrix} + \begin{bmatrix} 64 & 78 \\ -17 & -21 \end{bmatrix}$$

$$C_{22} = \begin{bmatrix} 44 & 41 \\ 111 & 79 \end{bmatrix}$$

The final product matrix will be -

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} 43 & 53 & 54 & 37 \\ 123 & 149 & 130 & 93 \\ 95 & 110 & 44 & 41 \\ 103 & 125 & 111 & 79 \end{bmatrix}$$

Q.26 Discuss Strassen's matrix multiplication and derive the time complexity.

[JNTU : Part B, April-11, Marks 8, May-13, Marks 7]

Ans. : Algorithm

```
Algorithm St_Mul(int *A, int *B, int *C, int n) {
 if (n == 1)then
 {
   (*C) =(*C)+ (*A) * (*B);
 }
 else
 {
   St_Mul(A,B,C,n/4);
   St_Mul(A, B+(n/4), C+(n/4), n/4);
   St_Mul(A+2 *(n/4), B, C+2 *(n/4), n/4);
   St_Mul(A+2 *(n/4), B+(n/4), C+3 *(n/4), n/4);
   St_Mul(A+(n/4), B+2*(n/4), C, n/4);
   St_Mul(A+(n/4), B+3*(n/4), C+(n/4), n/4);
   St_Mul(A+3*(n/4), B+2*(n/4), C+2*(n/4), n/4);
   St_Mul(A+3*(n/4), B+3*(n/4), C+3*(n/4), n/4);
 }
}
```

Time Complexity

$T(1) = 1$ ... assume $n = 2^k$

$T(n) = 7\ T(n/2)$

$T(n) = 7^k\ T(n/2^k)$

$T(n) = 7^{\log n}$

$T(n) = n^{\log 7} = n^{2.81}$

2.6 More examples on Divide and Conquer

Q.27 Write a pseudo code for a divide and conquer algorithm for finding values of both the largest and the smallest elements in an array of n numbers. **[JNTU : Part B, Dec.-11, 12, Marks 7]**

Ans. : In divide and conquer algorithm, the list of elements is divided at the mid in order to obtain two sublists. From both the sublist maximum and minimum elements are chosen. Two maxima and minima are compared and from them real maximum and minimum elements are determined. This process is carried out for entire list. The algorithm is as given below.

```
Algorithm Max_Min_Val (i, j, max, min)
//Problem Description : Finding min, max elements
recursively.
//Input : i, j are integers used as index to an array A.
The max and min will
//contain maximum and minimum value elements.
//Output : None
if (i = = j) then
{
   max ← A [i]
   min ← A [j]
}
else if (i = j–1) then
{
   if (A [i] < A [j]) then
   {
      max ← A [j]
      min ← A [i]
   }
   else
   {
      max ← A [i]
      min ← A [j]
   }      //end of else
}      //end of if
else
{
   mid ← (i+j) / 2
//divide the list handling two lists separately
   Max_Min_Val (i, mid, max, min)
   Max_Min_val (mid + 1, j, max_new, min_new)
   if (max < max_new) then
```

```
        max ← max_new        //combine solution
    If (min > min_new) then
    ) min ← min_new          //combine solution
```

Q.28 Consider a list of some elements from which maximum and minimum element can be found out.

1	2	3	4	5	6	7	8	9
50	40	– 5	– 9	45	90	65	25	75

[JNTU : Part B]

Ans. :

Step 1 :

1	2	3	4	5	
50	40	– 5	– 9	45	Sublist 1

6	7	8	9	
90	65	25	75	Sublist 2

We have divided the original list at mid and two sublists : Sublist 1 and sublist 2 are created. We will find min and max values respectively from each sublist.

Step 2 :

1	2	3		4	5	
50	40	– 5		– 9	45	Sublists

6	7		8	9	
90	65		25	75	Sublists

Again divide each sublist and create further sublists. Then from each sublist obtain min and max values.

Step 3 :

1	2		3		4	5
50	40		– 5		– 9	45

6	7		8	9
90	65		25	75

It is possible to divide the list (50, 40, – 5) further. Hence we have divided the list into sublists and min, max values are obtained.

Step 4 :

Now further division of the list is not possible. Hence we start combining the solutions of min and max values from each sublist.

1	2		3
50	40		– 5

Combine (1, 2) and (3) Min = – 5, Max = 50

1	2	3		4	5
50	40	– 5		– 9	45

Combine (1, ... 3) and (4, 5) Min = – 9, Max = 50

Now we will combine (1, ... 3) and (4, 5) and the min and max values among them are obtained. Hence,

min value = – 9

max value = 50

Step 5 :

6	7		8	9
90	65		25	75

Combine (6, 7) and (8, 9) Min = 25, Max = 90

Step 6 :

1	2	3	4	5		6	7	8	9
50	40	– 5	– 9	45		90	65	25	75

Combine the sublists (1, ... 5) and (6, ... 9). Find out min and max values which are

min = – 9 max = 90

Thus the complete list is formed from which the min and max values are obtained. Hence final min and max values are

min = – 9 and max = 90

UNIT - II

3 Searching and Traversal Techniques

3.1 Efficient Non-Recursive Binary Tree Traversal Algorithms

Important Points to Remember

- The binary tree contains at the most two child nodes.
- There are three traversal techniques - inorder, preorder and postorder technique.

Q.1 What is binary tree ? What are three traversals used in traversing binary tree ?
☞ [JNTU : Part B, May-17, Marks 5]

Ans. : A binary tree is a data structure that contains at the most two child nodes. For example -

There are three traversals used for traversing binary tree those are -

1. **Preorder traversal** - In this traversal the root/parent node is visited first, then the left child and then right child. For example for tree given in Fig. Q.1.1 the preorder traversal is,

   10, 8, 12, 11, 14.

2. **Inorder traversal** - In this traversal the left node is visited first, then parent node and then right node. For Fig. Q.1.1 the inorder traversal is,

   8, 10, 11, 12, 14.

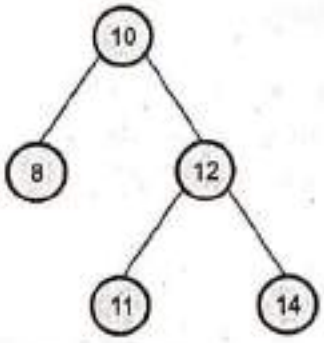


Fig. Q.1.1 Binary tree

3. **Postorder traversal** - In this traversal the left node, then right node and then the parent node is visited. For Fig. Q.1.1 the postorder sequence is,

   8, 11, 14, 12, 10.

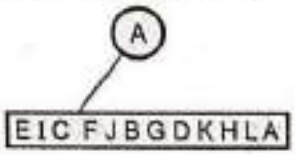
Q.2 Show that the inorder and postorder sequences of a binary tree uniquely define the binary tree. ☞ [Part B]

Ans. : For defining a binary tree uniquely atleast two sequences are required. Following example shows that using inorder and postorder sequences a binary tree can be uniquely defined. Let us write the sequences as follows

Inorder : E I C F J B G D K H L A

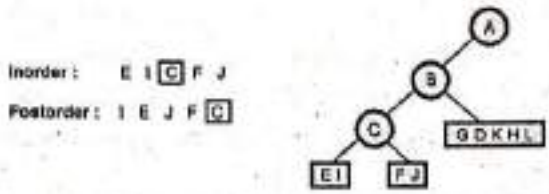
Postorder : I E J F C G K L H D B A

Step 1 : The last node in postorder sequence is root node. We will locate "A" in inorder sequence. The left sequence of "A" is the left subtree.

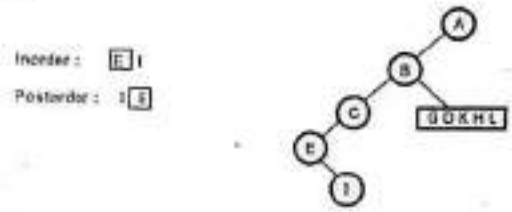


Step 2 :

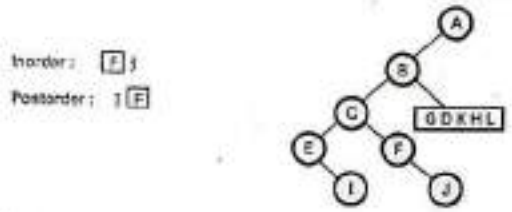
Step 3 :



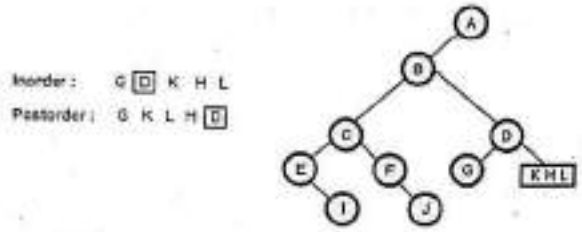
Step 4 :



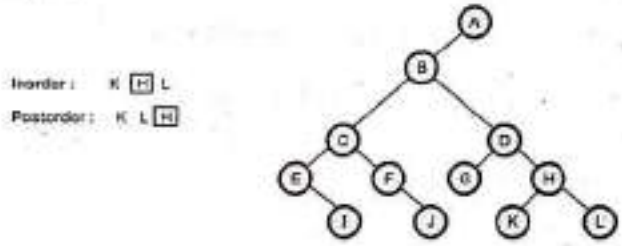
Step 5 :



Step 6 :



Step : 7



is the required binary tree.

Q.3 The preorder and postorder sequences of a binary tree do not uniquely define binary tree. Justify your answer. [Part B, Marks 5]

Ans. : In preorder sequence the first element denotes the root node. On the other hand in postorder sequence the last element denotes the root node. Thus determining a root node is very easy when pre and postorder sequences are given but decision on the remaining nodes for becoming left or right child is conflicting. Because left and right children get accumulated on one side in both these sequences. Along with either preorder or postorder sequence if inorder sequence is given then it partitions the left and right subsequences using the parent node. For example -

Consider

Preorder	10	8	12	11
Postorder	8	11	12	10

The element 10 denotes the root node.

From the sequences

Preorder	8,	12,	11
Postorder	8,	11,	12

It is not possible to determine which node should b attached as a left or right child of root 10. Thu building a unique binary tree is not possible usin preorder and postorder sequence.

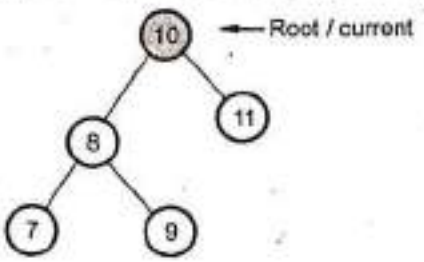
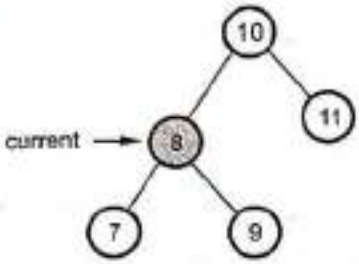
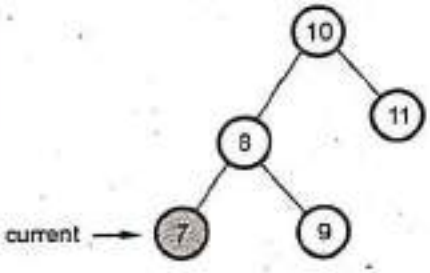
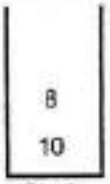
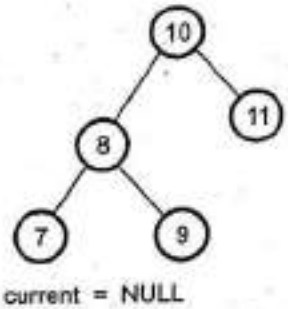
Q.4 Write and explain the non recursive inorder traversal.

Ans. :

```
void inorder(node *root)
{
 node *current,*s[10];
 int top=-1;
 if(root==NULL)
 {
    printf("\n Tree is empty\n");
    return;
 }
 current=root;
 for(;;)
 {
    while(current!=NULL)
    {
       push(current,&top,s);
       current=current->left;
    }
    if(!stempty(top))
    {
       pop(&top,s,&current);
       printf(" %d",current->data);
       current=current->right;
    }
    else
       return;
 }
}
```

Explanation :

We will consider following tree as an example.

		Initially we assume current = root = 10. As current != NULL the while statement from the routine **inorder** will get executed.
	Push 10 Move onto left branch ∴ current = current → left	
	Push 8 Move onto left branch ∴ current = current → left	
	Push 7 Move onto left branch ∴ current = current → left	

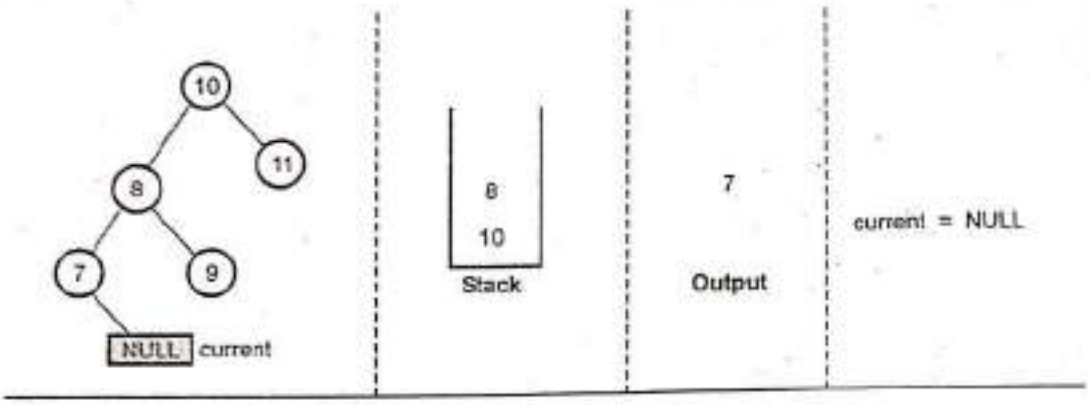
Now as **current** becomes **NULL**, we will come out of the while statements and following code fragment will get executed.

Code

```
if (! stempty (top))
{
 pop (&top,s, &current);
 cout <<" " << current → data;
 current = current → right;
}
```

Meaning

We will pop the topmost element from the stack i.e. 7. Then we will print it. And then we will move to right branch or right child of 7, calling it as new **current**.



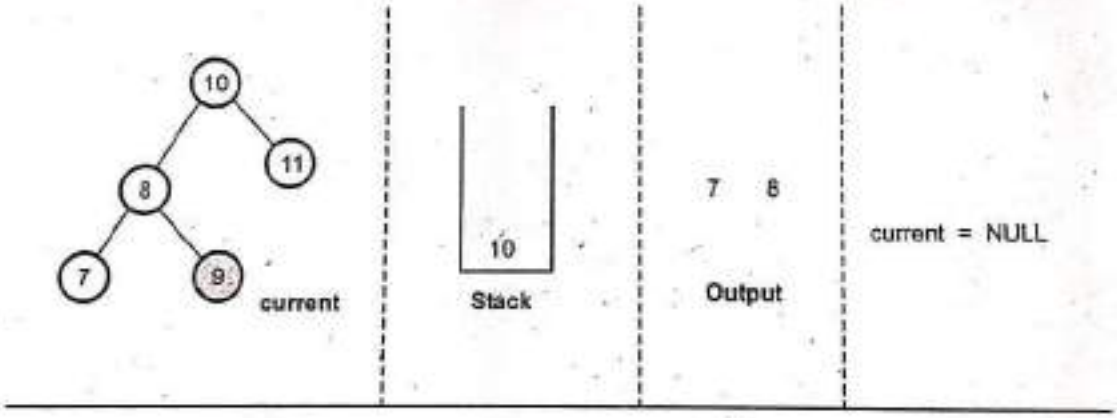
Now we will enter the for loop iteratively, but this time while statement will not get executed because current = NULL. Hence following code fragment will get executed.

Code

```
if (! stempty (top))
{
  pop (&top,s, &current);
  cout <<" " << current → data;
  current = current → right;
}
```

Meaning

We will pop the topmost element from the stack i.e. 8. Then we will print it. And then we will move onto right branch. That mean now current = right child of 8 = 9.



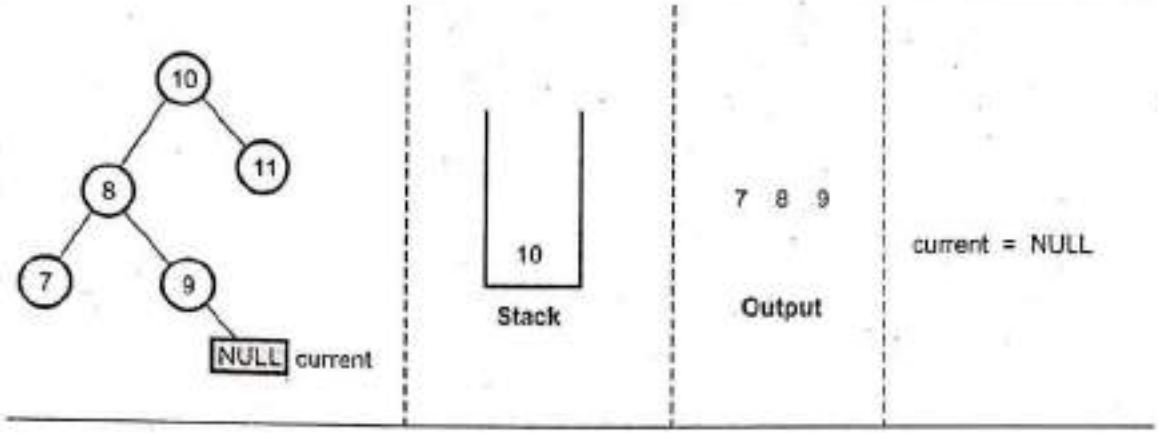
Now we will enter the for loop iteratively and while loop will be executed because current = 9. Hence 9 will be pushed onto the stack. And now current = current → left. As there is no left child for the node current; current = NULL. Hence following code fragment will be executed.

Code

```
if (! stempty (top))
{
  pop (&top,s, &current);
  cout <<" " << current → data;
  current = current → right;
}
```

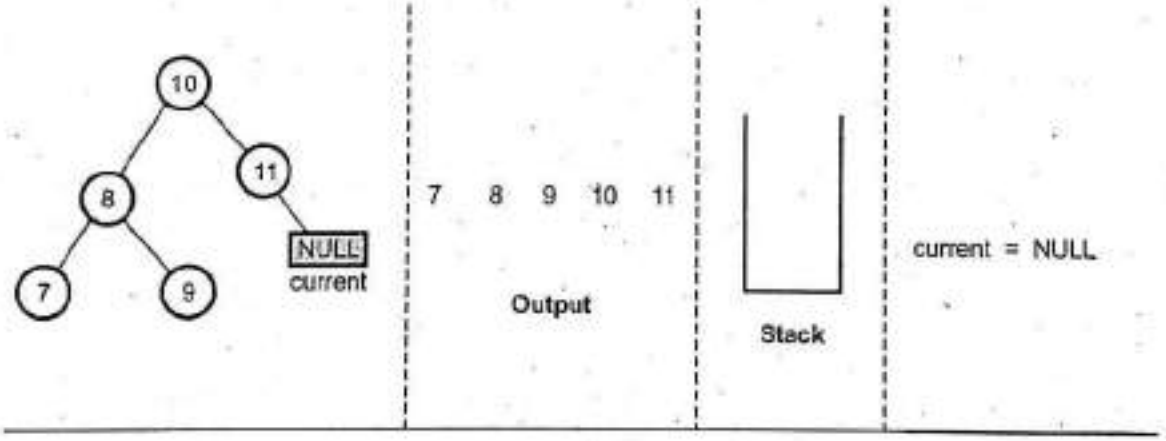
Meaning

Now 9 which lies on the top of the stack will b popped off. It will be printed as an output. And no current will set to right child of 9 which is NULL.

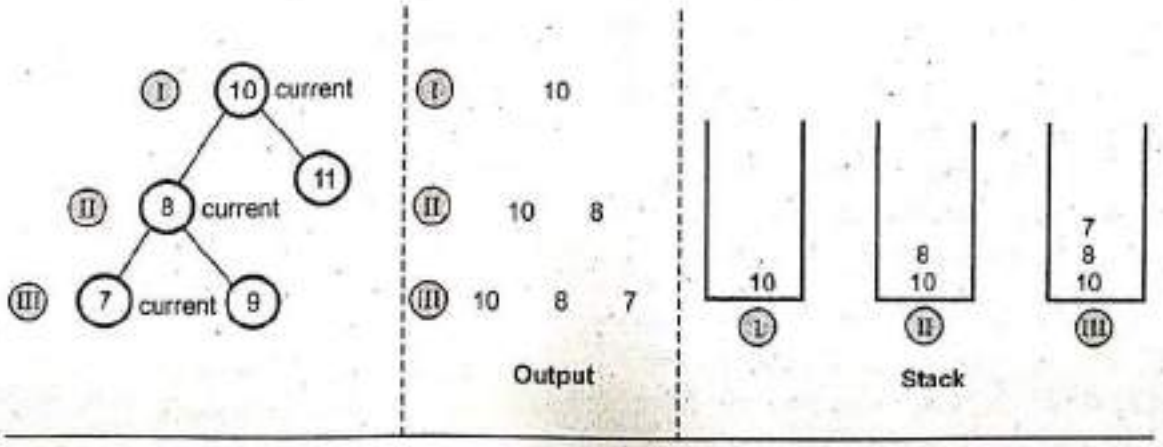


Now we will enter the for loop iteratively but this time while statement will not be executed. The if statement will be executed. According to this code fragment, 10 will be popped off, it will be printed as an output and we will move onto the right branch of 10. Hence now, current = 11 as an output we will get 7 8 9 10.

Again we will enter the for loop iteratively. The while statement will be executed because current = 12. We will push 11 onto the stack. And move onto the left branch of 11. But there is no left child to 11. Hence value of **current** becomes **NULL**. Then if statement will get executed. According to it, 11 will be popped off, it will be printed as an output and we will move onto right branch of 11.



Again we will enter the for loop iteratively. But current = NULL ; hence while statements will not be executed, stack is empty; hence if statements will not be executed. The **else** will be executed, by which control returns to main function. Thus we will get **7 8 9 10 11** as an output of non-recursive inorder traversal.



Q.5 Write and explain the non recursive preorder traversal.

Ans. :

```
void preorder(node *root)
{
node *current,*s[10];
int top=-1;
if(root==NULL)
{
   printf("\n The Tree is empty\n");
   return;
}
current=root;
for(;;)
{
   while(current!=NULL)
   {
    printf(" %d",current->data);
    push(current,&top,s);
    current=current->left;
   }
   if(!stempty(top))
   {
    pop(&top,s,&current);
    current=current->right;
   }
   else
    return;
}
}
/*
```

Explanation :

The logic for preorder traversal is similar to the inorder traversal. But the only difference is that we will print the value of each visited **current** node before pushing it onto the stack. Hence

Visiting each node, calling it as current, printing it as an output, pushing current node onto the stack and moving onto the left child. These are the sequence of operation which we must follow at every stage. Above is the scenario for stage I, II and III.

Now if we move onto left branch of 7 we will get current = NULL. We will pop 7 check if it has any right. As 7 has no right child we will pop the next element i.e. 8. As 8 has a right child i.e. 9, we will print 9 as an output and push it onto the stack. The stack will now be

Then 9 will be popped off, but 9 has no right child so we will pop 10. The 10 has right child i.e. 11. Hence print 11 as an output and push it onto the stack.

Finally 11 will be popped off. The node 11 has no left or right child so nothing will be pushed. As a result we get **10 8 7 9 11** as an output for non-recursive preorder traversal.

Q.6 Write a non-recursive algorithm of postorder traversal of tree and also analyze its time complexity.

[JNTU : Part B, April-09, Marks 10, Nov.-16, Marks 5]

Ans. :

```
Algorithm postorder(node *root)
{
//input: The root node
//output: prints the postorder sequence
top=-1; //initialise stack
if(root==NULL) then
{
   Write(\n The Tree is empty");
   return;
}
current ← root;
for(;;)
{
   while(current!=NULL) do
   {
     top++;
     st[top].element←current;
```

```
    //check 1 means visiting left subtree
    //check 0 means visiting right subtree
    st[top].check←-1;//visiting the left subbranch
    current←-current->left;
  }
  while(st[top].check==0) do
  {
    current←-st[top].element;
    top - -;
    Write(current->data);
    if(stempty(top)) then
     return;
  }
  current←-st[top].element;//pushing the element onto
                           //the stack
  current current->right;
  st[top].check←-0;//visiting right subtree
}
}
```

The above algorithm takes $O(n^2)$ time

Explanation

In the non-recursive postorder traversal we have to add extra field in the stack structure. This field keeps a check on whether the node is visited once or not. If we visit that node for the first time then we set that check to 1, if we again visits that node then we reset that check. This extra logic we have to put because we traverse to the left node first, then right node and then the parent node.

Q.7 Write a non recursive algorithm of post order tree traversal. **[JNTU : Part B, Nov.-16, Marks 5]**

Ans. : Refer Q.6.

3.2 Disjoint Set Operations

Q.8 What is disjoint set ? **[JNTU : Part A, Marks 3]**

Ans. :

- A disjoint set is a kind of data structure that contains partitioned sets. These partitioned sets are **separate non overlapping sets.**
- **For example :** If there are n=10 elements that can be partitioned into three disjoint sets. S_1, S_2 and S_3 such that no element is common among these sets.
- Let, S_1 = { 5, 4, 7, 9 }, S_2 = { 10, 12, 14 }, S_3 = { 2, 3, 6 }, then each set can be represented as a tree. It is as shown in Fig. Q.8.1.

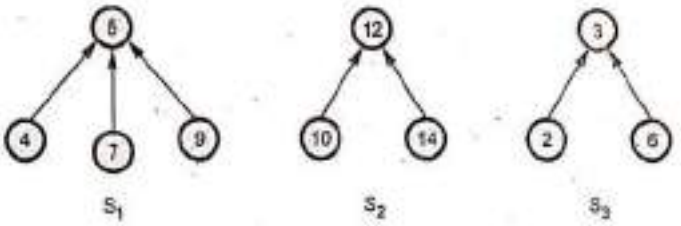


Fig. Q.8.1 Tree representation of sets

Q.9 Explain the disjoint set operations using trees ? **[JNTU : Part B, May-12, Marks 7]**

Ans. : There are two operations that can be performed on the data structure : disjoint set. These operations are **union** and **find.**

Let, S_1 = { 5, 4, 7, 9 }, S_2 = { 10, 12, 14 }, S_3 = { 2, 3, 6 },

1. **Disjoint set union** : If there exists two sets S_i and S_j then $S_i \cup S_j$ = {all the elements from set S_i and S_j }. In above example $S_1 \cup S_2$ = { 5, 4, 7, 9, 10, 12, 14 }.
2. **Find (i)** : For finding out the element i from given set, is done by this operation. For example - Element 7 is in S_1, element 14 is in S_2.

3.3 Union and Find Algorithms

Q.10 What is union operation ? Explain. **[JNTU : Part A, Marks 2]**

Ans. : The union operation combines the elements from two sets.

Consider S_1 = { 5, 4, 7, 9 } and S_2 = { 10, 12, 14 } then $S_1 \cup S_2$ is

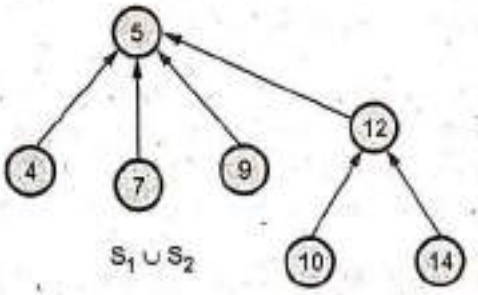


Fig. Q.10.1 Union operation

Q.11 Explain the method of representation of data of sets. **[JNTU : Part B, Marks 5]**

Ans. : To perform union or find operations efficiently on sets, it is necessary to represent the set elements in a proper manner. The data representation of sets is illustrated by following example -

Consider S_1 = { 5, 4, 7, 9 }, S_2 = { 10, 12, 14 } and S_3 = { 2, 3, 6 }

For finding out any desired element,

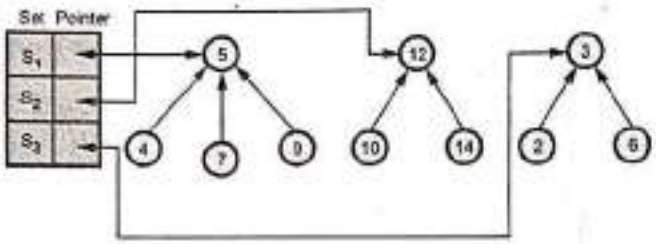


Fig. Q.11.1 Data representation

i. Find the pointer of the root node of corresponding set.

ii. Then find the desired element from that set.

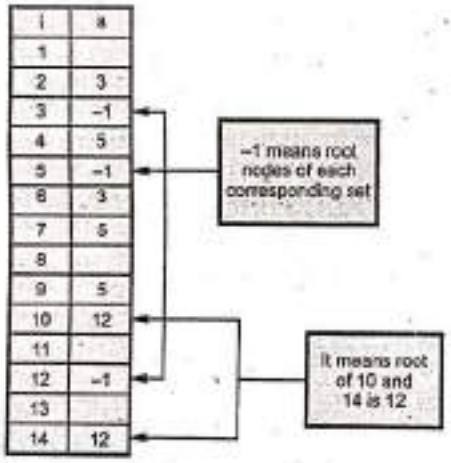


Fig. Q.11.2

The sets can be represented using arrays as follows -

The find operation can be performed as follows - **Find (9)** = 5. Now obtain a[5]= –1 means we have reached at the root node. Then node 5 is a root node whose child is 9. Thus element 9 is present in set S_1.

The union operation can be performed as follows - union (10, 20), union (20, 30), union (30, 40), union (40, 50).

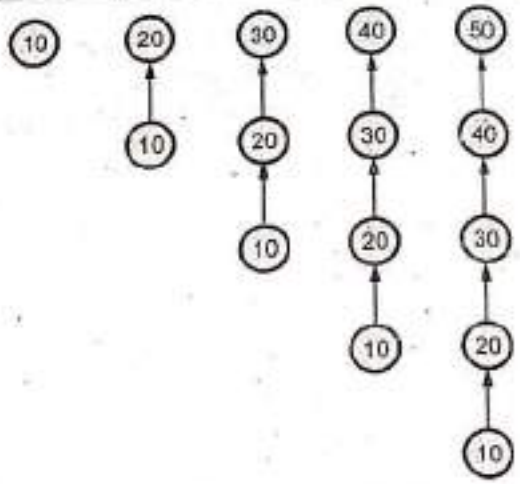
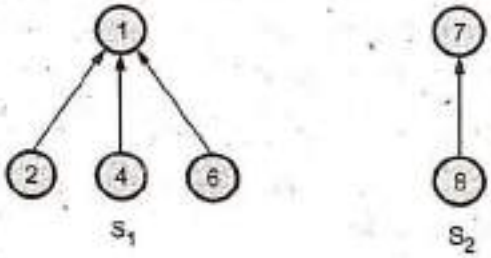


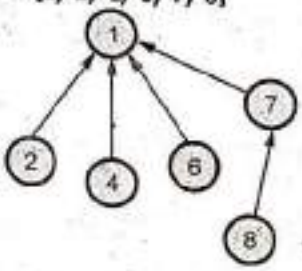
Fig. Q.11.3 Degenerated tree

Q.12 Two sets S_1 and S_2 are as given below S_1 = {1, 2, 4, 6} and S_2 = {7, 8}
a) Draw disjoint sets S_1 and S_2 using trees
b) Draw disjoint sets S_3 using trees such that $S_3 = S_1 \cup S_2$
c) Draw disjoint sets S_4 using trees such that $S_4 = S_2 \cup S_1$
d) Give pointer representation of S_1, S_2, S_3 and S_4. [JNTU : May-12, Part B, Marks 15]

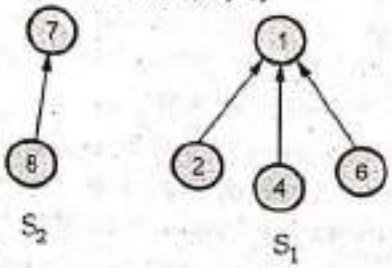
Ans. : a) The disjoint trees are -



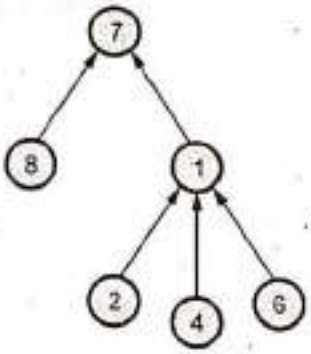
b) $S_3 = S_1 \cup S_2$ = {1, 2, 4, 6, 7, 8}



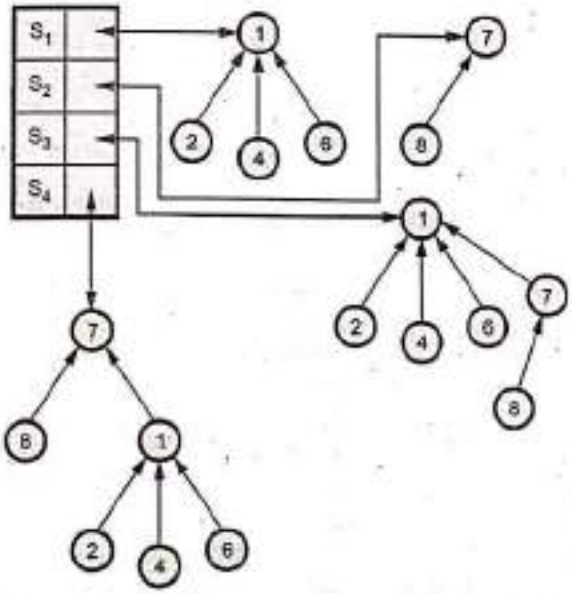
c) $S_4 = S_2 \cup S_1$ = {7, 8, 1, 2, 4, 6}



S_4 can be



d) Pointer representation -



Q.13 Explain the usefulness of the following fundamental operations on sets :
i) FIND ii) DELETE iii) UNION iv) INSERT.

[JNTU : Part B, May-12, Marks 15]

Ans. : Various operations used by disjoint set are - UNION, FIND, INSERT, DELETE and MAKE_SET. Following are the algorithms for these operations -

1) MAKESET(x) - This function is used for creating a new set.

```
MAKESET(x)
{
   a[x] ← x
   rank [x] ← 0
   child [x] ← 0 // stores number of children
   marked [x] ← 0 // indicates if node is marked for
   //deletion
}
```

The running time of this algorithm is Θ(1).

2) UNION

```
UNION (x1, x2)
{
   UNK (FIND (x1), FIND (x2))
}
LINK (x1, x2)
{
   if rank [x1] > rank [x2] then
      a[x2] ← x1
         child [x1] ← child [x2] + child [x1] + 1
   else
      a[x1] ← x2
      child [x2] ← child [x1] + child [x2] + 1
      if rank [x1] = rank [x2] then
         rank [x2] ← rank [x2] + 1
}
```

3) INSERT

```
INSERT (x1, x2)
{
   MAKESET (x1)
   UNION (x1, x2)
}
```

Using this function the elements of the set can be inserted in tree.

4) DELETE : Operation has following goals -

a) Either immediately remove a node or mark it for deletion.

b) Remove any nodes that are marked for deletion and having no child.

c) Decrement all **child** attributes for deleted nodes ancestors.

These goals can be satisfied by two oprations - DELETE and CLEAN

```
DELETE (x)
{
if (x ≠ a[x]) then
   CLEAN (a[x])
if child [x] ≠ 0 then
   marked [x] ← 1     // marked for deletion
else
```

```
    FREE (x)
}
CLEAN(x)
{
    child [x] ← child [x] - 1
        if x ≠ a [x] then
            CLEAN (a[x])
        if marked [x] = 1 then
            if child [x] = 0      // i.e. no child to x node
                FREE (x)
}
```

The worst case running time of DELETE operation is O(logn).

5) FIND

```
FIND (x)
{
    if x ≠ a[x] then
        a[x] ← FIND_SET (a[x], child [x] + 1)
    return a[x]
}
FIND_SET (x, ch)
{
    P ← a[x]      // making x parent
    if x ≠ a[x] then
        P ← FIND_SET (a[x], child [x] + 1)
        a[x] ← P
        child [x] ← child [x] – ch
        if marks [x] = 1 then
            if child [x] = 0 then
                FREE(x)
        return P
}
```

The purpose of FIND operation is -

a) to update child attribute correctly.

b) to remove any nodes that are marked for deletion.

Analysis

- The time required by (n–1) unions is **O(n)**.
- The time taken to process a find for an element at level i of tree is O(i). Hence total time required by find operation is $O(n^2)$.

Q.14 Write find and union algorithms.
[JNTU : Part B, May-12, Marks 8]

Ans. : Refer Q.13.

Q.15 Explain the set representation using tree and develop algorithms for UNION and FIND using weighing and collapsing rules.
[JNTU : Part B, May-09, Marks-16, May-13, Marks 15]

Ans. : Set representation using tree – Refer Q.11.

Algorithm using weighing rule - The performance of simple union operation can be improved by weighted union operation.

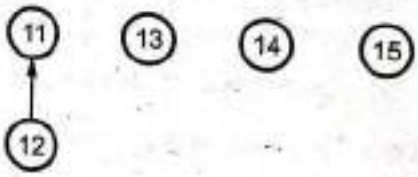
Definition

Weighting Rule : If the tree with root i has number of nodes that are less than a tree with root j then the tree with root j becomes parent of i otherwise (greater or equal to condition) make root i as parent node of root j.

For example

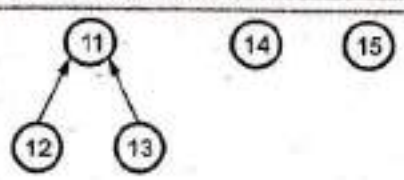
Step 1 : Initial

Step 2 :



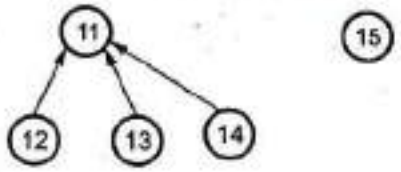
As root node 11 has no child and root node 12 also has no child (we get i=j condition of weighting rule), hence make 11 as parent of 12.

Step 3 :



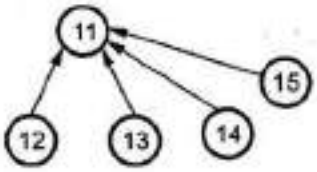
The tree with root 11 has one child and the tree with root 13 has no child, make the tree with root 11 as parent node of node 13.

Step 4 :



The tree with root 11 has two child nodes i.e. 12 and 13. But the tree with root 14 has no child node, hence tree with 11 becomes parent of node 14.

Step 5 :



The tree with root 11 has 12,13,14 nodes as child nodes but tree with root 15 has no child node. Therefore tree with node 11 becomes parent of node 15.

This is a tree obtained using weighing rule.

```
Algorithm Wt_Union ( i,j )
{
// Problem Description : This algorithm generates a
// tree using union operation.
// Input : trees with root i and j.
// Output : performs union of all nodes.
   a[i] ← – count[i]      // initially
   a[j] ← – count[j]
temp ← a[i] + a[j] ;
if (a[i] > a[j] ) then // i has few nodes than j
   {
      a[i] = j            // j becomes parent of i
      a[j] = temp ;
   }
      else                // lesser or equal to condition
   {
      a[j] = i            // i becomes parent of j.
      a[i] = temp
   }
}
```

Algorithm using collapsing rule -

Definition

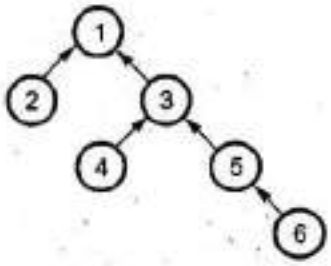
Collapsing Rule : If j is a node on the path from i to its root and a[i] ≠ root [i] then set a[j] to root [i]. Using collapsing rule, the find operation can be performed.

For example

Step 1 :

Consider a tree of unions.

Now we will find 6 using collapsing rule.



Let,
root = 1

Step 2 :

temp = parent[6]
As root ! = temp i.e. 1 ≠ 5
Set temp = parent [temp]
∴ temp = 5

Step 3 :

Again root ! = temp i.e. 1 ≠ 5
Set temp = parent [temp]
∴ temp = 3

Step 4 :

Again root ! = temp i.e. 1 ≠ 3
Set temp = parent [temp]
∴ temp = 1

Step 5 :

Now root = temp i.e. 1 = 1.

return root i.e. 1

Step 6 : Thus we can find (6) as 6 → 5 → 3 → 1

```
Algorithm Collapse_Find (i)
{
    // Problem Description : This algorithm collapse
    //all nodes
    // from i to root node.
       root = i
    while ( i != root ) do
    {
    temp = parent [i];
    parent [i] = root;
    i = temp;
    }
return root;
}
```

Q.16 For the following sequence of instructions
UNION (1,2,2)
UNION (2,3,3)
:
UNION (n – 1,n,n)
FIND (1)
FIND (2)
:
FIND (n)
a) Write the tree after (n – 1) UNION operations.
b) Compute the cost of n FIND instructions.
☞ [JNTU : Part B]

Ans. : a) Assume n = 5 then

Now if we want to find (5) then

Union (1, 2, 2)

i.e. Union (i, j, j) ⇒

parent [j] ← i

i.e. parent [2] ← 1

Union (2, 3, 3)

i.e. Union (i, j, j) ⇒

parent [j] ← i

i.e. parent [3] ← 2

Union (3, 4, 4)

i.e. Union (i, j, j) ⇒

parent [j] ← i

i.e. parent [4] ← 3

Union (4, 5, 5)

i.e. Union (i, j, j) ⇒

parent [j] ← i

i.e. parent [5] ← 4

Now if we want to find (5) then

Let root = 1

Find (5) = temp = parent [5] = 4

As root ≠ temp

Find (temp) = find (4) = parent [4] = temp = 3

As root ≠ temp

Find (temp) = find (3) = parent [3] = temp = 2

As root ≠ temp

Find (temp) = find (2) = parent [2] = temp = 1

As root = temp, return the value of root.

Thus Find(5) = 1 which denotes the root node o element 5.

If there are i number of levels then Find(i) = O(i Hence Find(5) = 5

Q.17 Write and explain the final algorithm for collapse rule with an example.
☞ [JNTU : Part B, May-12, Marks 15]

Ans. : Refer Q.15.

3.4 Spanning Trees

Q.18 What is spanning tree ? Explain with an example and also give applications of spanning tree.

[JNTU : Part B, Dec.-12, Marks 7]

Ans. : A spanning tree of a graph G is a subgraph which is basically a tree and it contains **all the vertices of G containing no circuit.**

Minimum spanning tree

A minimum spanning tree of a weighted connected graph G is a spanning tree with minimum or smallest weight.

Weight of the tree

A weight of the tree is defined as the sum of weights of all its edges.

For example :

Consider a graph G as given below. This graph is called weighted connected graph because some weights are given along every edge and the graph is a connected graph.

Applications of spanning trees

1. Spanning trees are very important in designing efficient routing algorithms.
2. Spanning trees have wide applications in many areas such as network design.

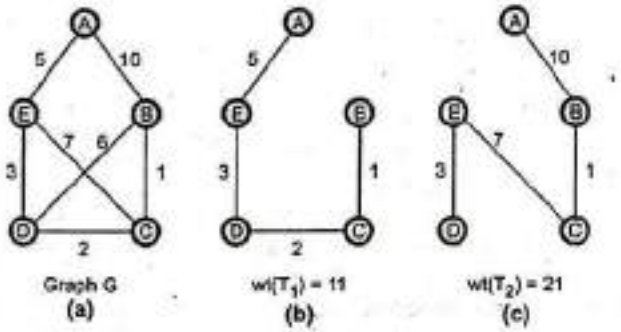


Fig. Q.18.1 Graph and two spanning trees out of which (b) is a minimum spanning tree

3.5 Graph Traversals

Important Points to Remember

A graph is a collection of two sets V and E. Where V is a finite non empty set of vertices and E is a finite non empty set of edges.

- **Vertices** are nothing but the nodes in the graph.
- Two adjacent vertices are joined by **edges.**
- Any graph is denoted as G = (V, E)

For example

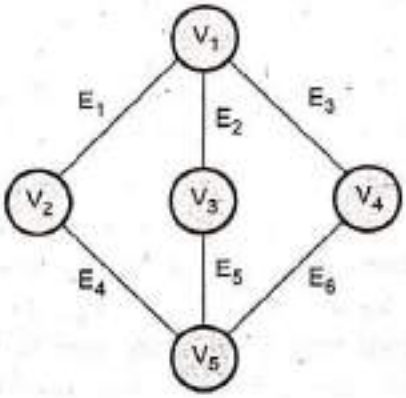


Fig. 3.1 Graph G

$G=\{(V_1, V_2, V_3, V_4, V_5), (E_1, E_2, E_3, E_4, E_5, E_6)\}$

- The graph traversal techniques are breadth first search and depth first search.

Q.19 Prove that every connected graph G on n vertices is union of at most (n + 1)/2 edge-disjoint paths. **[JNTU : Part B]**

Ans. : The edge-disjoint path is a path that contains no repeated edge. For a complete graph there are (n – 1)/2 Hamiltonian circuits, if n ≥ 3.

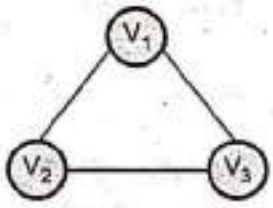
Consider a connected graph

There are n = 3 vertices

The edge disjoint paths are

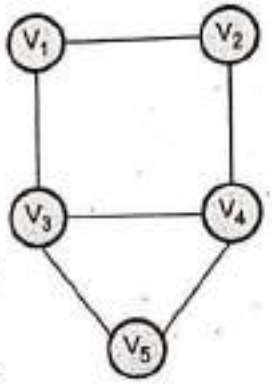
1. $V_1 - V_2 - V_3$
2. $V_2 - V_3 - V_1$

i.e. (n + 1) / 2 edge disjoint paths



There are n = 5 vertices.

The edge disjoint paths



1. $V_1 - V_2 - V_4 - V_3 - V_5$
2. $V_1 - V_3 - V_5 - V_4 - V_2$
3. $V_1 - V_2 - V_4 - V_5 - V_3$

i.e. (n + 1) / 2 edge disjoint paths

This shows that for n vertices of connected graph these are (n+1) / 2 edge disjoint paths.

Q.20 Explain breadth first search algorithm with suitable example. [JNTU : Part B, Marks 5]

Ans. : For the graph to traverse it by BFS, a vertex V_1 in the graph will be visited first, then all the vertices adjacent to V_1 will be traversed suppose adjacent to V_1 are (V_2 ,V_3, V_4,...V_n). So V_2, V_3,...V_n will be printed first. Then again from V_2 the adjacent vertices will be printed. This process will be continued for all the vertices to get encounterd.

Algorithm :

1. Create a graph. Depending on the type of graph i.e. directed or undirected set the value of the flag as either 0 or 1 respectively.
2. Read the vertex from which you want to traverse the graph say V_i.
3. Initialize the visited array to 1 at the index of V_i.
4. Insert the visited vertex V_i in the queue.
5. Visit the vertex which is at the front of the queue. Delete it from the queue and place its adjacent nodes in the queue.
6. Repeat the step 5, till the queue is not empty.
7. Stop.

```
void bfs(int v1)
{
   int v2;

   visit[v1] = TRUE;
   front = rear = -1;
   Q[++rear] = v1;
   while ( front != rear )
   {
      v1 = Q[++front];
      printf("%d\n", v1);
      for ( v2 = 0; v2 < n; v2++)
      {
         if ( g[v1][v2] == TRUE && visit[v2] ==
      FALSE )
         {
            Q[++rear] = v2;
            visit[v2] = TRUE;
         }
      }
   }
}
```

Explanation of logic of BFS

In BFS the queue is maintained for storing the adjacent nodes and an array 'visited' is maintaine for keeping the track of visited nodes i.e. once particular node is visited it should not be revisite again. Let us see how our program works.

Step 1 : Start with vertex 1.

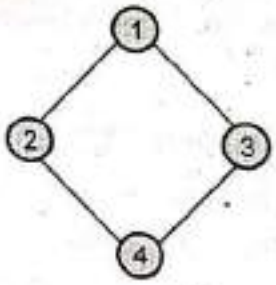
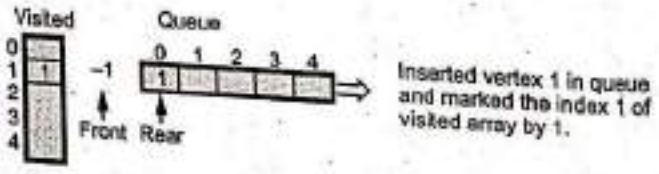
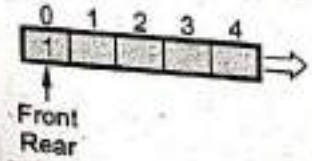


Fig. Q.20.1

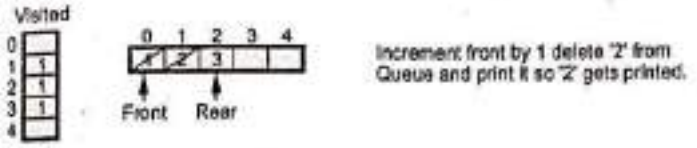
Step 2 :



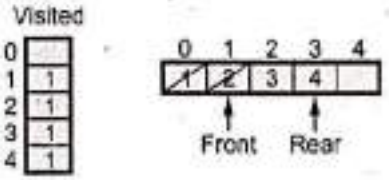
Step 3 : Find adjacent vertices of vertex 1 and ma them as visited, insert those in queue.



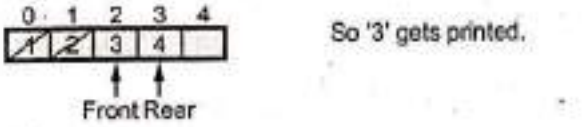
Step 4 : Find adjacent to '2' and insert those nodes in queue as well as mark them as visited.



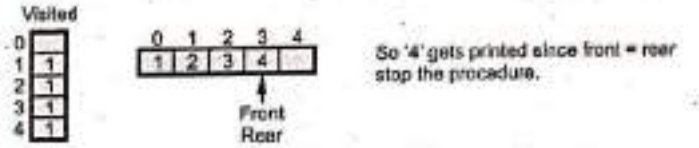
Step 5 : Increment front and delete the node print it.



Step 6 : Find adjacent to '3' i.e. 4 check whether it is marked as visited. If it is marked as visited do not insert in the queue.



Increment front, delete the node from queue and print it.



So output will be - BFS for above graph as

1 2 3 4

Q.21 Explain depth first search algorithm with suitable example. **[Part B, Marks 5]**

Ans. :

- In depth first search traversal, we start from one vertex, and traverse the path as deeply as we can go. When there is no vertex further, we traverse back and search for unvisited vertex.
- An array is maintained for storing the visited vertex.

For example :

- The DFS will be (if the source vertex is 0) 0-1-2-3-4.
- The DFS will be (if we start from vertex 3) 3-4-0-1-2.

```
void Dfs(int v1)
{
 int v2;
 printf("%d\n", v1);
 v[v1] = TRUE;
 for ( v2 = 0; v2 < MAX; v2++)
  if ( g[v1][v2] == TRUE && v[v2] == FALSE )
   Dfs(v2);
}
```

Explanation of logic for depth first traversal

In DFS the basic data structure for storing the adjacent nodes is stack. In our program we have used a recursive call to DFS function. When a recursive call is invoked actually push operation gets performed. When we exit from the loop pop operation will be performed. Let us see how our program works.

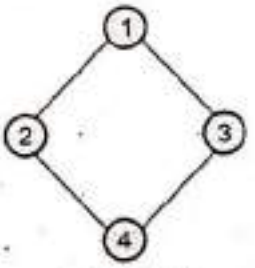
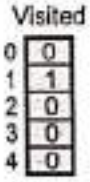
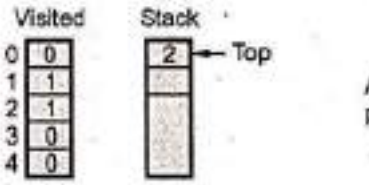


Fig. Q.21.1

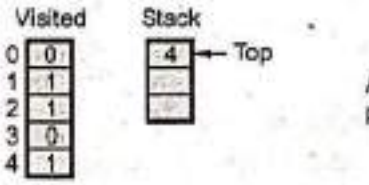
Step 1 : Start with vertex 1, print it so '1' gets printed. Mark 1 as visited.



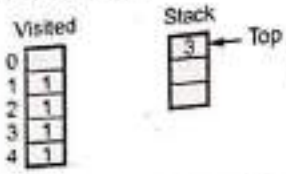
Step 2 : Find adjacent vertex to 1, say i.e. 2 if it is not visited, call DFS(2) i.e. 2 will get inserted in the stack, mark is as visited.



Step 3 : Find adjacent to '2' i.e. vertex 4 if it is not visited call DFS(4) i.e. 4 will get pushed onto the stack mark it as visited.



Step 4 : Find adjacent to '4' i.e. vertex 3 if it is not visited call DFS(3) i.e 3 will be pushed onto the stack mark it visited.

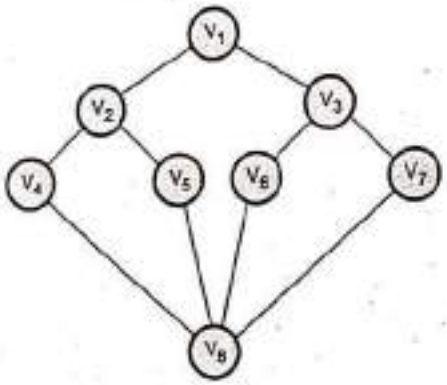


Since all the nodes are covered stop the procedure.

So output of DFS is

1 2 4 3

Q.22 Define DFS and BFS graph. Show DFS and BFS for the graph given below.
[JNTU : Part B, May-08, Marks 6]

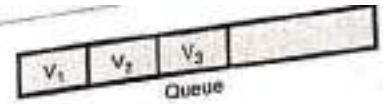


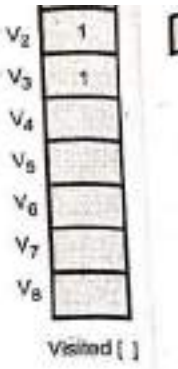
Ans. : DFS - Refer Q.21.
BFS - Refer Q.20.

BFS for given graph

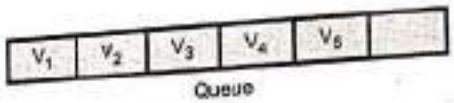
Step 1 : For BFS we start from vertex V_1. Insert V_1 in the queue, mark it as visited

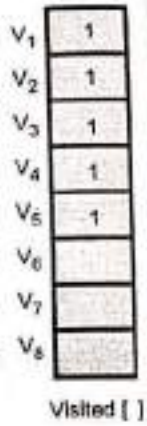
Step 2 : Now find the adjacent vertices of V_1 and insert them in the queue. Mark them as visited.



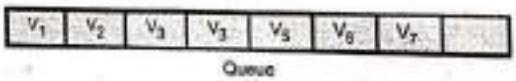


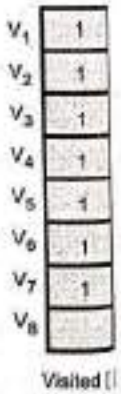
Step 3 : Now find adjacent vertices of V_2. These ar V_1, V_4, V_5. But as V_1 is already visited, just inse vertices V_4 and V_5 in queue.





Step 4 : Find adjacent vertices of V_3 which are $V_1, V_$ and V_7. But as V_1 is already visited, we will insert V and V_7 in the queue and mark them as visited.





Step 5 : Finally V_8 vertex will be inserted in queue.

Now delete each vertex from Queue and print it BFS sequence :

V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8

V_1	1
V_2	1
V_3	1
V_4	1
V_5	1
V_6	1
V_7	1
V_8	1

Visited [ ]

DFS for given graph

Step 1 : Start with vertex V_1, print it so V_1 gets printed. Mark V_1 as visited.

V_1	1
V_2	0
V_3	0
V_4	0
V_5	0
V_6	0
V_7	0
V_8	0

V_1

Step 2 : Find adjacent vertex to V_1, say i.e. V_2 if it is not visited call DFS (V_1) i.e. V_2 will get inserted in the stack mark as visited.

V_1	1
V_2	1
V_3	0
V_4	0
V_5	0
V_6	0
V_7	0
V_8	0

V_1, V_2

Step 3 : Find adjacent vertex to V_2, call DFS (V_2) i.e. V_4 will get inserted in the stack mark as visited.

V_1	1
V_2	1
V_3	0
V_4	1
V_5	0
V_6	0
V_7	0
V_8	0

V_1, V_2, V_4

Step 4 : Find adjacent vertex to V_4 call DFS (V_4)

V_1	1
V_2	1
V_3	0
V_4	1
V_5	0
V_6	0
V_7	0
V_8	1

V_1, V_2, V_4, V_8

Step 5 : Final adjacent vertex to V_8 call DFS (V_8).

V_1	1
V_2	1
V_3	0
V_4	1
V_5	1
V_6	0
V_7	0
V_8	1

V_1, V_2, V_4, V_8, V_5

Step 6 : Find adjacent to vertex V_5 i.e. V_2 which is already visited so next is V_1 it is already visited so next is V_5. So visit V_5 and call DFS (V_5).

V_1	1
V_2	1
V_3	0
V_4	1
V_5	1
V_6	1
V_7	0
V_8	1

$V_1, V_2, V_4, V_8, V_5, V_6$

Step 7 : Find adjacent to vertex V_6 call DFS (V_6).

V_1	1
V_2	1
V_3	1
V_4	1
V_5	1
V_6	1
V_7	0
V_8	1

Step 8 : Find adjacent to vertex V_3 call DFS (V_3)

V_1	1
V_2	1
V_3	1
V_4	1
V_5	1
V_6	1
V_7	1
V_8	1

As all nodes are visited so,

DFS of graph = V_1, V_2, V_4, V_8, V_5, V_6, V_7, V_3

Q.23 Explain the graph traversal with an example.

[JNTU : Part B, Nov.-16, Marks 5]

Ans. : Refer Q.22.

Q.24 Differentiate between BFS and DFS.

[JNTU : Part A, Nov.-16, May-17, Marks 5]

Ans. :

Sr. No.	BFS	DFS
1	BFS is simple to implement	DFS is complex to implement as it may suffer from infinite loop problem.
2	BFS will perform poor is for large numbers of vertices in graph.	DFS will perform better in case of large complex graph.
3	BFS requires more memory	DFS requires less memory
4	BFS will find the shortest path if the weight on the links are uniform.	DFs can not obtain shortest path.
5	BFS is not useful in sorting application	DFS is used in topological sorting.
6	This algorithm works in single stage. The visited vertices are removed from the queue and then displayed at once.	This algorithm works in two stages - in the first stage the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped-off.
7	If solution path is long, the whole tree must be searched up to that depth.	May settle for non-optimal solution.

3.6 AND/OR Graphs

Important Points to Remember

- While solving the complex problem, it is broken down into series of subproblems. These subproblems can be further decomposed into sub-subproblem. To obtain the solution to main problem the solution to any sub-tree can be sufficient. In fact we can choose the better solution and discard other solutions. This scenario can be represented by AND/OR graph.

Q.25 Explain the concept of AND/OR Graph in detail.

Ans. :

- AND-OR graph can be as shown in Fig. Q.25.1.

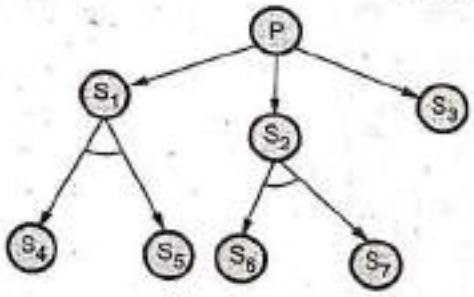


Fig. Q.25.1

- Here P is OR node i.e. to solve P we can get either S_1, S_2 or S_3 as solution. Node S_1 is AND node i.e. we can get solution S_1 only after solving both S_4 and S_5 (i.e. S_1 is dependent upon S_4 and S_5). Similarly S_2 is also AND node.
- AND-OR graphs are used in problem solving methods in which intelligent search is involved.

Definition of AND node and OR node.

i) **AND node** : A node is an AND node if all its successors have to be solved in an order for obtaining solution to that node.

ii) **OR node :** A node is an OR node if it is enough to solve any one of the successors to obtain solution to that node.

- AND-OR graphs operate on the graphs that do not involve any cycles.

Q.26 Write an algorithm for AND/OR graphs.

☞ [JNTU : Part B, Nov 16, Marks 5]

Ans. : Step 1 : Start from root node and follow the best path.

Step 2 : Choose one of the yet non-expanded nodes and expand it.

Step 3 : Update the estimation of newly expanded node, using the newly obtained information of the successors. Then decide which successors is most promising and mark this as part of the best path so far. This can change the currently selected path.

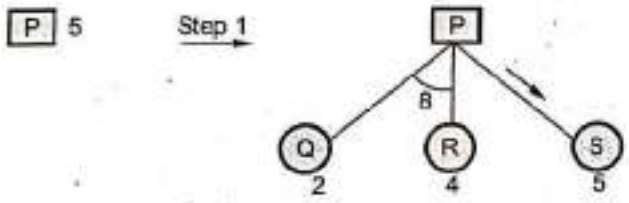


Fig. Q.26.1 (a)

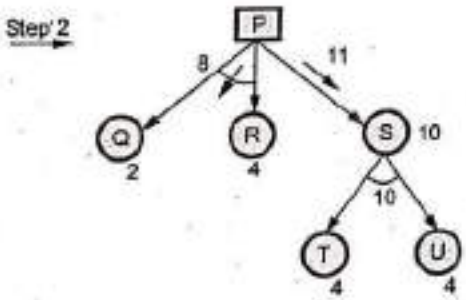


Fig. Q.26.1 (b)

Now we will obtain the optimum cost as 11. For solving problem P we will choose solution T and U then S.

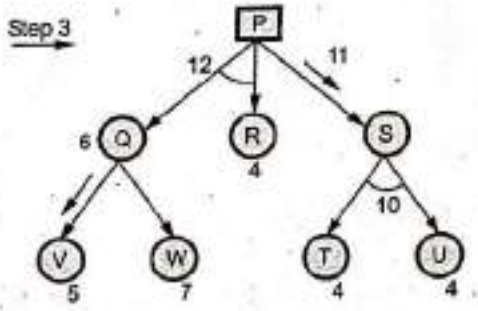


Fig. Q.26.1 (c)

- A node is solvable if
  - i) It is a terminal node.
  - ii) It is a non-terminal node and if its successors are AND nodes which are also solvable.
  - iii) It is a non-terminal node whose successors are OR nodes and if at least one of them is solvable.
- A node is unsolvable if
  - i) It is a non-terminal node having no successor.
  - ii) It is a non-terminal node whose successors are AND nodes and at least one of them is unsolvable.
  - iii) It is a non-terminal node whose successors are OR nodes and all these successor nodes are unsolvable.

3.7 Game Trees

Q.27 What are the applications of game tree ?

☞ [JNTU : Part A, Nov.-16, Marks 3]

Ans. :

- A game tree is a directed graph whose nodes are positions in a game and whose edges are moves.
- The complete game tree for a game is the game tree starting at the initial position and containing all possible moves from each position; the complete tree is the same tree as that obtained from the extensive-form game representation.
- Various applications of game tree are -

1. Building decision trees
2. Representing behavior trees
3. For creating state machines
4. Fuzzy logic application
5. Neural Network
6. Game of chess.

Q.28 Write short note on – game trees.

☞ [JNTU : Part B, Marks 5]

Ans. :

- Game tree is built to solve many interesting games such as tic-tac-toe, 15 puzzle 8 queen's problem. 15-puzzle problem is invented by Sam Loyd in 1878.
- In this puzzle there are 15 tiles which are numbered from 1 to 15.

- The **objective** of this problem is to transform the arrangement of tiles from initial arrangement to a goal arrangement.
- The initial and goal arrangement is shown by in Fig. Q.28.1.
- There is always one empty slot in the initial arrangement. This empty slot is denoted by ES.
- The legal moves are the moves in which the tiles adjacent to ES are moved to either left, right, up or down. These arrangements are called **states** of the puzzle.

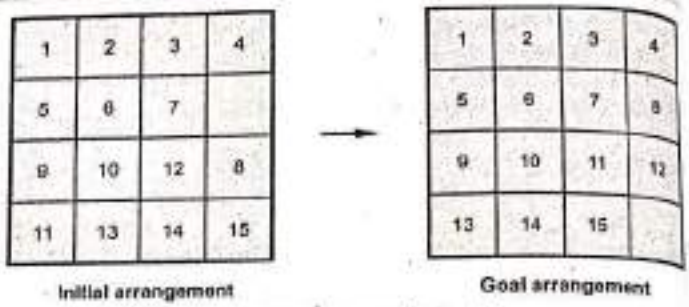


Fig. Q.28.1 15-puzzle problem

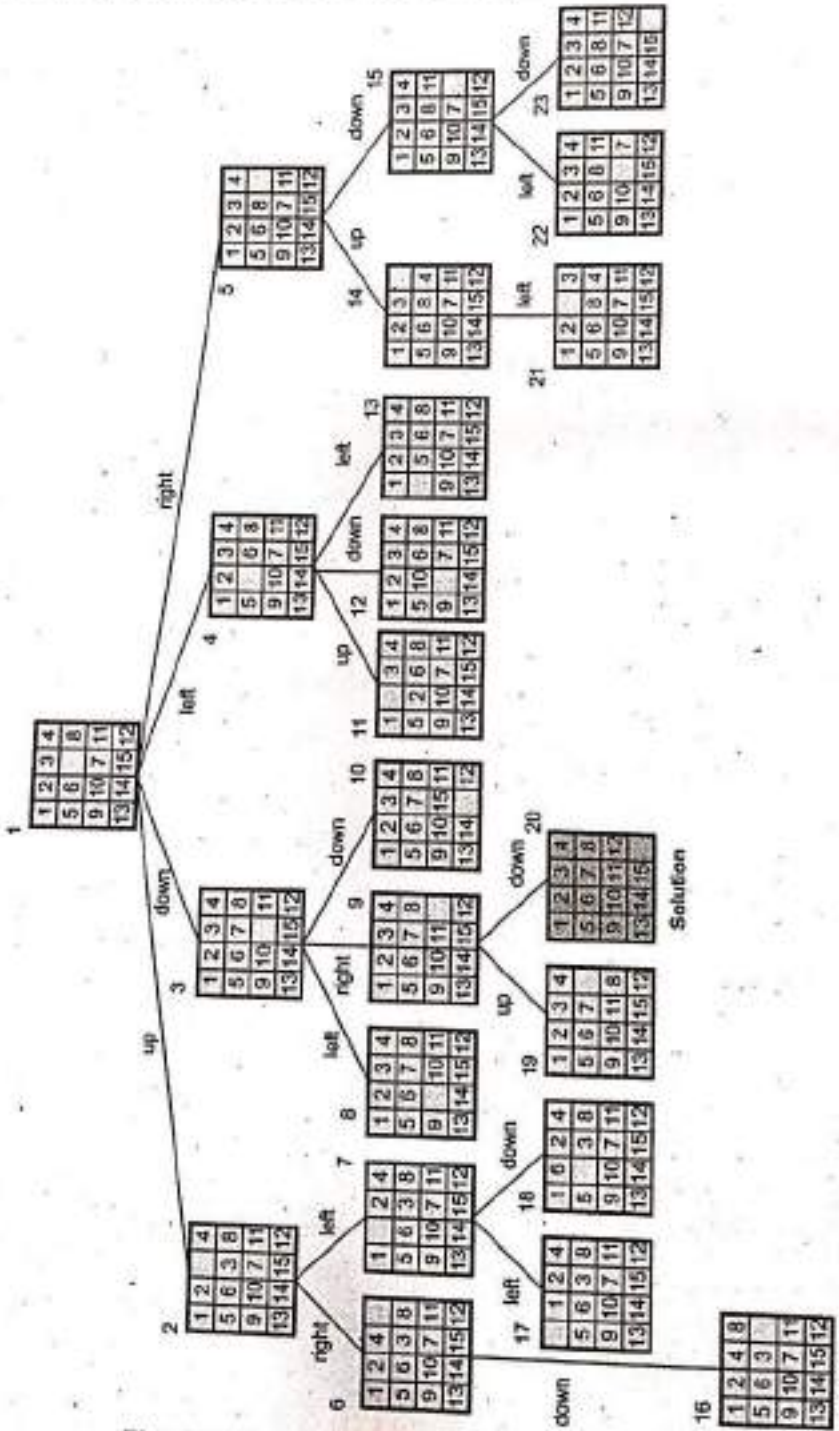


Fig. Q.28.2 Game tree for 15-puzzle problem

- The initial arrangement is called **initial state** and the goal arrangement is called **goal state.**
- A game tree can be drawn for representing various states.
- In the game tree the path from initial state to goal state represents the answer. A partial game tree can be as shown in Fig. Q.28.2.

3.8 Connected Components

Important Points to Remember

- A graph is said to be **connected** if there exists a path between any two vertices.
- If the graph G is connected undirected graph, then we can visit all the vertices of the graph in first call to Breadth First Search (BFS) or Depth First Search (DFS). The subgraph which we obtain after traversing the graph using BFS or DFS represents the connected component of the graph.

Q.29 Define the term - connected components.

☞ [JNTU : Part A, Marks 2]

Ans. : The connected components of a graph is a collection of vertices such that there is a path between every pair of vertices in the same component. That means there exists a path between V_i and V_j if V_i and V_j are in the same component and there is no path between V_i and V_k if V_i and V_k are in two different components.

- For obtaining path between any two vertices, the transitive closure is obtained.

Q.30 Describe the strongly connected components with an example.

☞ [JNTU : Part B, Dec.-12, Marks 8, May-17, Marks 5]

Ans. :

- Finding strongly connected components of a directed graph is one of the classic application of DFS.

In undirected graphs, two vertices are connected if they have a path connecting them But in case of directed graphs vertex a is strongly connected to b if there exists two paths one from a to b and another from b to a. We will use a b as a shorthand for " a is strongly connected to b" similarly b a as a shorthand for "b is strongly connected to a". This relation between the vertices is reflexive, symmetric and transitive.

For example

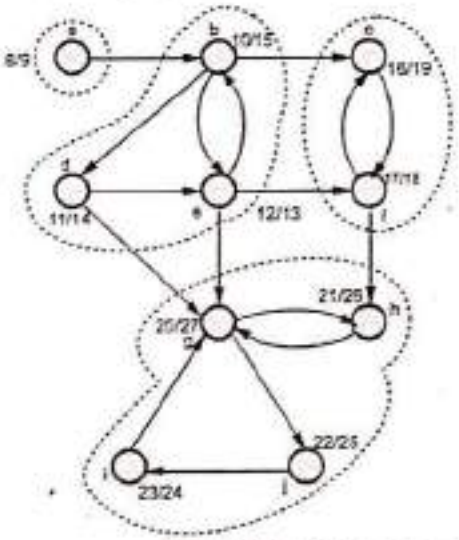


Fig. Q.30.1 A directed graph G

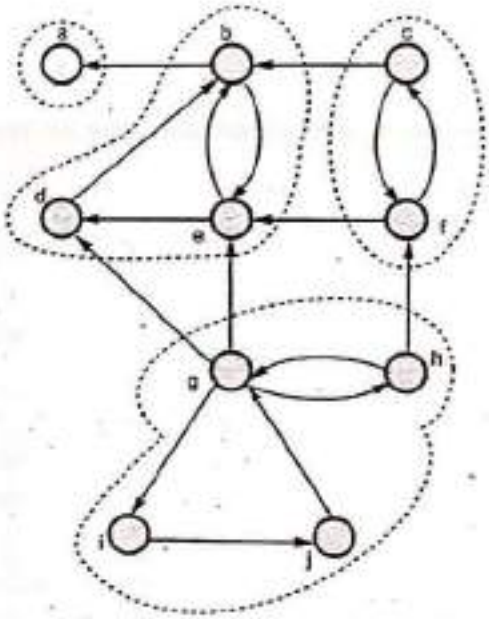


Fig. Q.30.2 A directed graph G^R

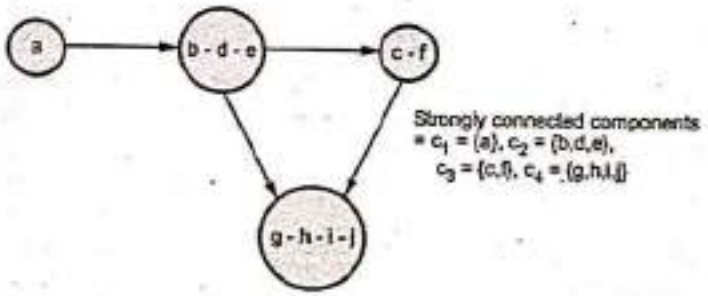


Fig. Q.30.3 Strongly connected components shown by a cyclic component graph

Q.31 Write an algorithm for obtaining strongly connected components of a graph.

Ans. :

Algorithm SCC(G)

{

// Problem Description: This algorithm obtains

// strongly connected components.

- Call DFS (G) compute finishing time for each vertex a denote it as finish [a]
- Compute G^R
- Call DFS (G^R) considering the vertices in the order of decreasing finish [a]
- Print the vertices of each tree obtained from DFS(G^R) as strongly connected component.

}

Q.32 Explain the properties of strongly connected components.

[JNTU : Part B, May-09, Marks 6, May-17, Marks 5]

Ans. : There are 3 properties of strongly connected components -

1. **Reflexive property** - Any vertex is strongly connected to itself.
2. **Symmetric property** - If any vertex u is connected to vertex v then vertex v is also connected to vertex u. That means there exists two paths that connects vertices u and v.
3. **Transitive property** - For a strongly connected component if vertex u is connected to v, v is connected to w with the paths u-v, v-u, v-w, w-v then there exists a path from u-w and w-v as well. This holds transitive property for strong connectivity.

Q.33 Give an algorithm to find the minimum number of edges that need to be removed from an undirected graph so that the resulting graph is acyclic.

[JNTU : Part B, April-11, Marks 7]

Ans. : Consider the connected components of a graph. Consider that there are k vertices. Then the graph must have at the most k - 1 edges to be acyclic. Then to obtain minimum number of edges that need to be removed for making the graph acyc will be -

Step 1 : Execute an algorithm that returns numb of connected components.

Step 2 : Count total number of edges (e) a vertices (v) from each connected component. Th minimum number of edges that must be removed f resulting graph being acyclic is $e - v + 1$.

- **Algorithm for finding connected components**

```
Algorithm components (G, n)
{
   for (i ← 1 to n)
   {
      visited [i] ← 0;
   }
   for (i ← 1 to n)
   {
      if (visited [i] == 0)
         DFS (i)
      output the newly visited vertices
      with adjacent edges
   }
}
```

Time complexity

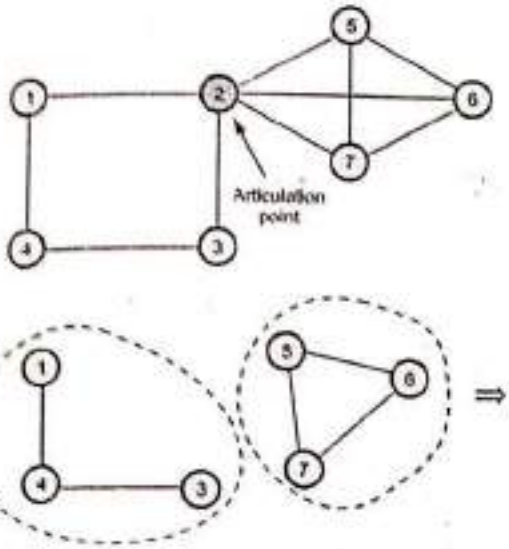
The time complexity of above algorithm is O(n+l The total time taken by DFS is O(E). And the for lo in which DFS routine is called takes O(n) time. Hen all the connected components get generated in O(n+ time.

3.9 Bi-connected Components

Q.34 Explain the concept of articulation poi with suitable example. **[JNTU : Part B, Marks 5]**

Ans. : Definition of Articulation Point : Let G = (V, be a connected undirected graph, then an articulati point of graph G is a vertex whose remov disconnects graph G. This articulation point is a ki of cut-vertex.

For example



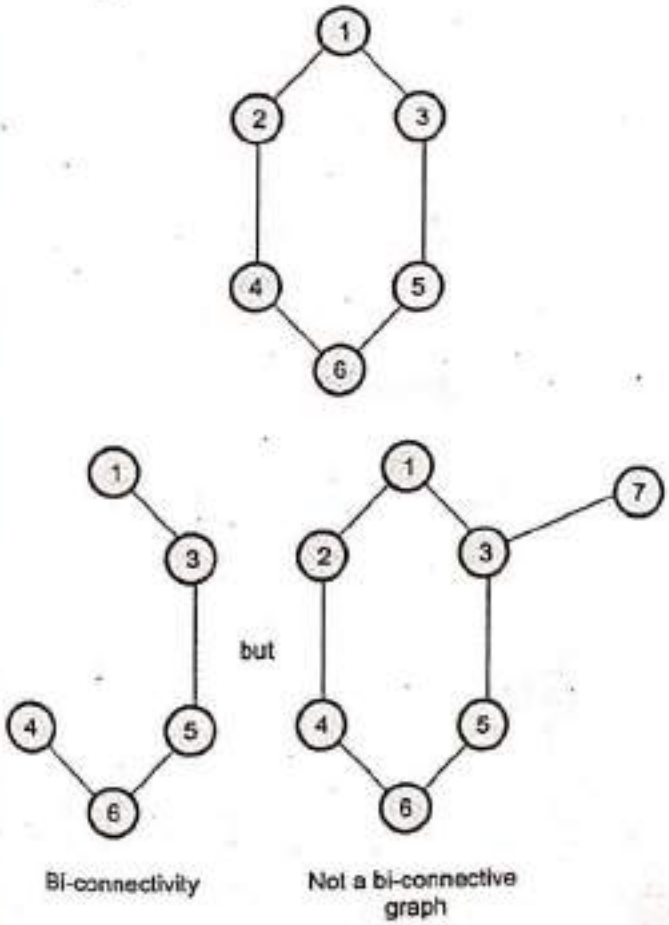
Q.35 Explain the concept of bi-connectivity of a graph with respect to articulation point.

[JNTU : Part B, Marks 5]

Ans. :

- A graph G is said to be bi-connected if it contains no articulation points.

For example



Eventhough we remove any single vertex we do not get disjoint graphs. Let us remove vertex 2 and we will get

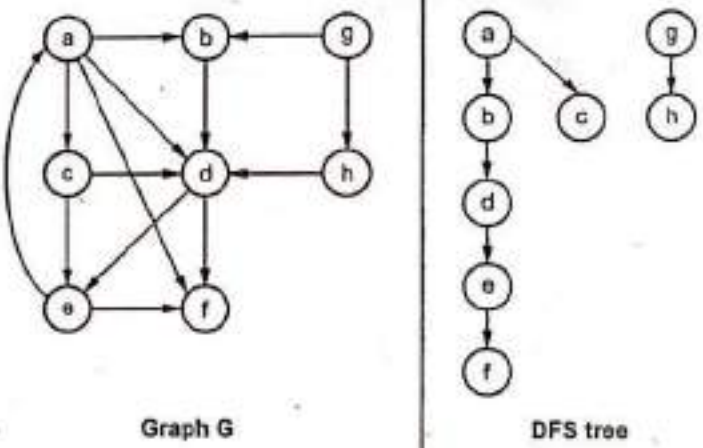
- If there exists any **articulation point** in the given graph then it is an **undesirable feature** of that graph. For instance in communication network the nodes or vertices represent the communication station x. If this communication station x is an articulation point then failure of this station makes the entire communication system down! And this is surely not desirable feature.

Q.36 What is DFS tree ? Explain with suitable example.

[JNTU : Part A, Marks 3]

Ans. : The depth first search algorithm involves exhaustive searches of all nodes by going ahead if possible, else backtrack. This leads to a tree like structure called DFS tree.

For example



Q.37 Explain classification of edges in DFS tree with suitable example.

[JNTU-H : Part B, Marks 5]

Ans. :

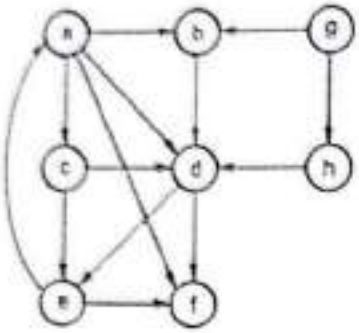
i) **Tree edge** : It is an edge in depth first search tree.

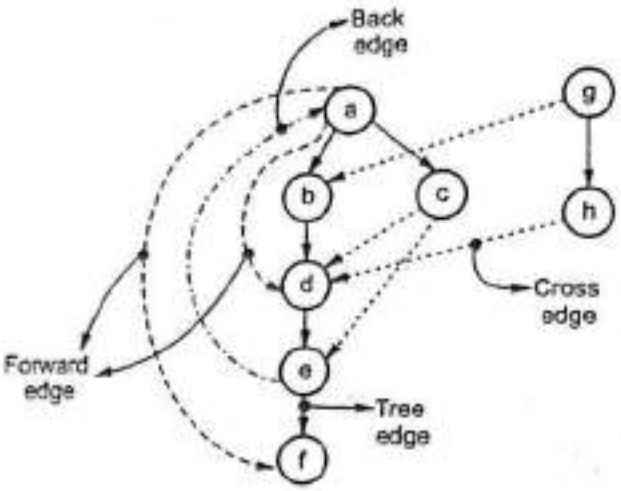
ii) **Back edge** : It is an edge (u,v) which is not in DFS tree and v is an ancestor of u. It basically indicates a loop.

iii) **Forward edge** : An edge (u,v) which is not in search tree and u is an ancestor of v.

iv) **Cross edge** : An edge (u,v) not in search tree and v is neither an ancestor nor a descendant of u.

For example





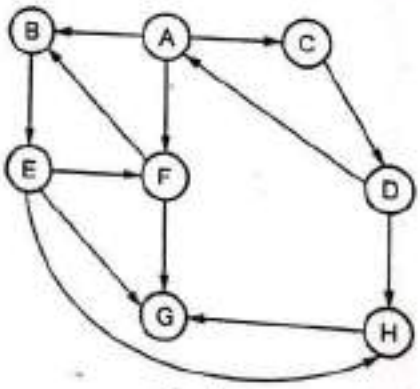
Tree edge
Cross edge
Forward edge
Back edge

Q.38 What are pre and post numbering in DFS. Explain with suitable example.

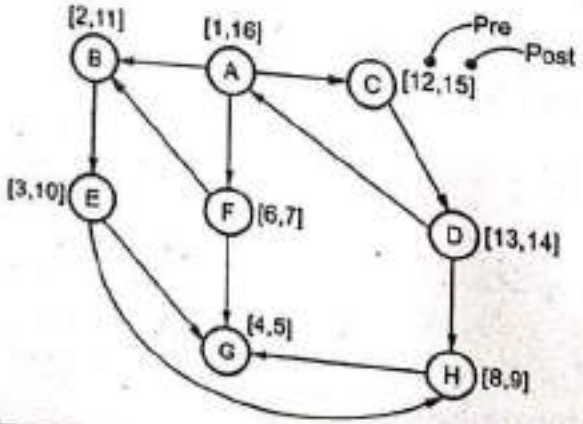
[JNTU-H : Part B, Marks 5

Ans. : The pre and post numbers in DFS are the values for start time of visit and end time of visit of ver respectively.

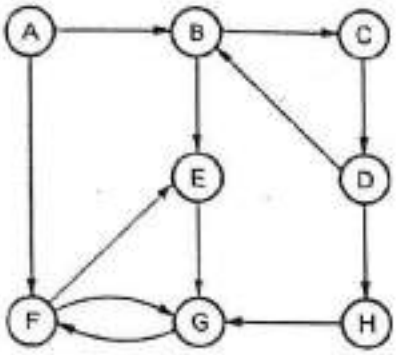
For example - Here we will start visiting from node A.



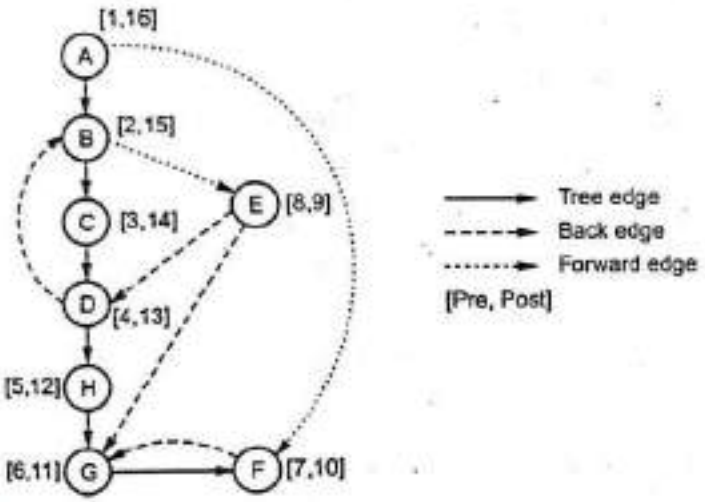
Graph G



Q.39 Perform depth-first search on following graph. Whenever there is a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge, forward edge, back edge or cross edge. Give the pre and post number of each vertex. **[JNTU-H : Part B, Marks 5]**



Ans. :



Q.40 Give an efficient algorithm that takes as input a directed graph G = (V, E) and determines whether or not there is a vertex s in V from which all other vertices are reachable. Show the time complexity of your algorithm.
[JNTU : Part B, Dec.-11, Marks 12]

Ans. :

Algorithm

Step 1 : Run DFS on graphs starting from any node

Step 2 : Run DFS again, starting from the vertex number that got highest post number.

Step 3 : If all the nodes are reachable from the second run of DFS, then return true else return false.

Time Complexity : The time complexity of the above algorithm is linear as DFS is executed twice. Hence it is O(V+E).

Q.41 Find a necessary and sufficient condition for the root of a depth first search for a connected graph to be an articulation point. Prove it.
[JNTU : Part B, May-09, Marks 8]

Ans. :

- To identify articulation points following observations can be made.
  - i) The root of the DFS tree is an articulation if it has two or more children.
  - ii) A leaf node of DFS tree is not an articulation point.
  - iii) If u is any internal node then it is not an articulation point if and only if from every child w of u it is possible to reach an ancestor of u using only a path made up of descendents of w and back edge.

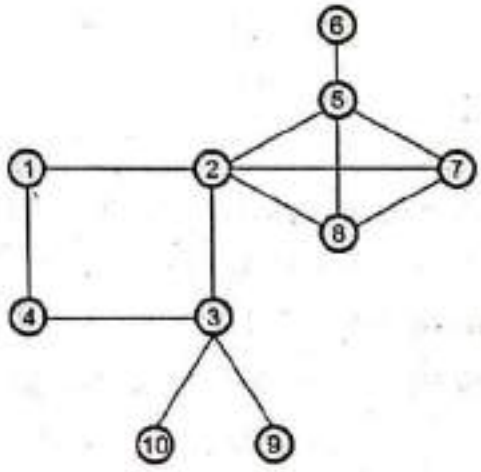
This observation leads to a simple rule as,

Low [u] = min { dfn [u], min {Low [w]/w is a child of u},
min {dfn [w]/ (u,w) is a back edge}}

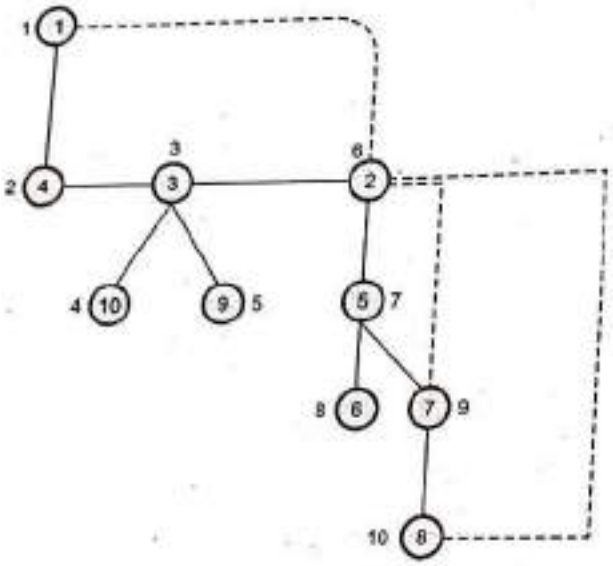
where Low [u] is the lowest depth first number that can be reached from u using a path of descendents followed by **at most one back edge**. The vertex u is an articulation point if u is child of w such that

$$L[w] \geq dfn[u].$$

Q.42 Obtain the articulation point for following graph. **[JNTU : Part B]**



Ans. : The DFS tree can be drawn as follows



Let us compute Low[u] using the formula

$$\text{Low [u]} = \min\left\{\begin{matrix}\text{dfn[u], min}\{\text{Low[w]} \mid \text{w is a child of u}\}, \\ \text{min}\{\text{dfn [w]} \mid \text{(u, w) is backedge}\}\end{matrix}\right\}$$

Low [1] = min { dfn[1] min {Low[4]}, dfn[2])

= min {1, Low [4], 6}

∴ Low [1] = 1

... ∵ nothing is less than 1

Low [2] = min { dfn[2], min {Low[5]}, dfn[1] }

= min { 6, ... Low [5], 1}

∴ Low [2] = 1

Low [3] = min {dfn[3], min {Low [10],

Low [9], Low [2]}, no backedge}

= min {3, min {Low[10], Low[9], 1) – }

= min {3, 1, –}

∴ Low [3] = 1

Low [4] = min {dfn [4], min {Low [3]} – }

= min {2, 1, – }

∴ Low [4] = 1

Low [5] = min {dfn [5], min {Low [6],

Low [7], – } }

= min {7, min{Low [6], Low [7] – } }

∴ Low [5] = Keep it as it is after getting value

Low[6] and Low [7] we will decid

Low [5]

Low [6] = min {dfn [6], – , – }

... ∵ leaf mod

∴ Low [6] = 8

Low [7] = min {dfn[8], – ,dfn [2]}

= min {10, –, 6}

∴ Low [7] = 6

As we have got Low [6] = 8 and Low [7] = 6. We w compute our incomplete computation Low [5].

∴ Low [5] = min {7, min {8,6}, – } = min {7, 6, –

∴ Low [5] = 6

Now Low [8] = min {dfn [8], – , dfn [2] }

= min {10, 6}

∴ Low [8] = 6

Low [9] = min {dfn [9], – , – }

∴ Low [9] = 5

Low [10] = min {dfn [10], – , – } = min {4, –, –

∴ Low [10] = 4

Hence Low values are Low [1:10] = {1, 1, 1, 1, 6, 8, 6, 5, 4}. Here vertex 3 is articulation point becaus child of 3 is 10 and Low [10] = 4. dfn[3] = 3. That i Low [w] ≥ dfn [u].

Similarly vertex 2 is an articulation point. Becaus child of 2 is 5 and

Low [5] = 6

dfn [2] = 6

i.e. Low [w] ≥ dfn[u]

Vertex 5 is articulation point because child of 5 is 6.

Low [6] = 8

dfn [5] = 7

i.e. Low [w] ≥ dfn [u]

Hence in above given graph vertex 2, 3 and 5 ar articulation points.

Q.43 Define an articulation point in a graph. Write algorithm to find articulation point. Write the time complexity of your algorithm.
[JNTU : Part B, April-11, Marks 15]

Ans. : Articulation point in a graph – Refer Q.34.

Algorithm for Articulation Points

The algorithm for obtaining the articulation point is as given below

```
Algorithm DFS_Art(u,v)
// the vertex u is a strating vertex for depth first
traversal.
//In depth first tree v is a parent(ancestor) of u.
//Initially an array dfn[ ] is initialized to 0.The def[ ]
stores the depth first search
//numbers
//Array Low[ ] is used to gives the lowest depth first
number that can be reached
//from u
{
//put the dfn number for u in dfn array
dfn[u]:=dfn_num;
Low[u]:=dfn_num;
def_num := dfn_num +1;
for ( each vertex w adjacent to u ) do
{
 // w is child of u and if w is not visited
 if(dfn[w]=0) then
 {
  DFS_Art(w,u);//finding the dfn of w
 Low[u]:=min(Low[u],Low[w]);
}
else if(w!=u) then //the edge u w can be a back edge
then update Low[u]
 Low[u]:=min(Low[u],dfn[w]);
}
}
```

Analysis

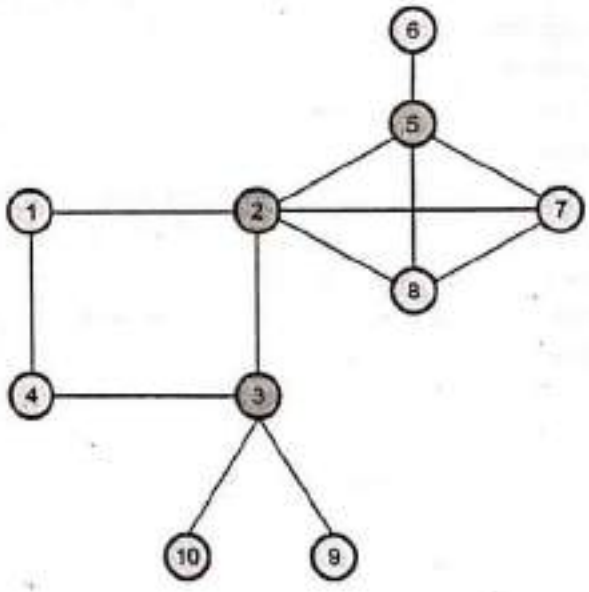
The DFS_Art has a complexity O(n+E) where E is number of edges in graph G and n is total number of nodes. Thus the articulation point can be determined in O(n+E) time.

Q.44 Explain how to identify Bi-connected components ? [JNTU : Part B, Marks 5]

Ans. :

- A bi-connected graph G = (V, E) is a connected graph which has no articulation points.
- A bi-connected component of a graph G is maximal bi-connected subgraph. That means it is not contained in any larger bi-connected subgraph of G.
- Some key observations can be made in regard to bi-connected components of graph.
  1) Two different bi-connected components should not have any common edges.
  2) Two different bi-connected components can have common vertex.
  3) The common vertex which is attaching two (or more) bi-connected components must be an articulation point of G.

For example : In following graph,



The articulation points are 2, 3, 5. Hence bi-connected components are

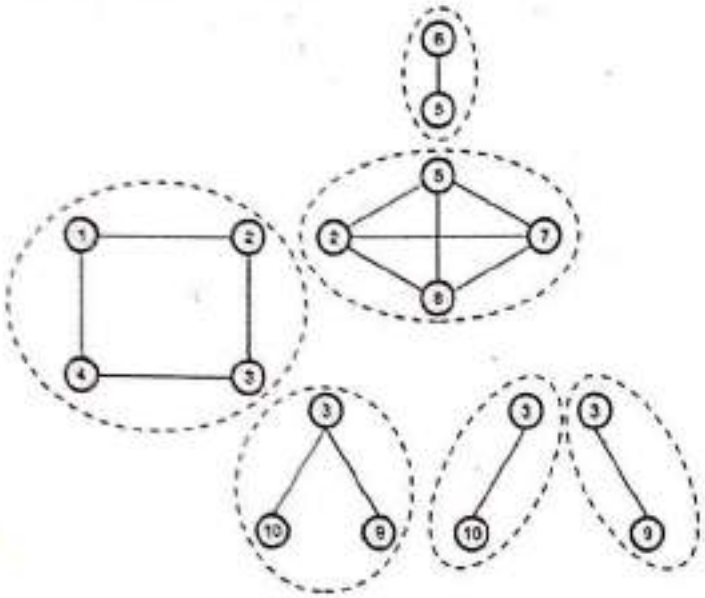


Fig. Q.44.1 Bi-connected components of G

Q.45 Give an algorithm for obtaining the bi-connected components. Also give the time complexity of your algorithm.

[JNTU : Part B, Marks 5]

Ans. : The algorithm for obtaining bi-connected components is as given below

```
Algorithm Bi_connect(u,v)
// the vertex u is a strating vertex for depth first
traversal.
//In depth first tree v is a parent(ancestor) of u.
//Initially an array dfn[ ] is initialized to 0.The def[ ]
stores the depth first search //numbers
//Array Low[ ] is used to gives the lowest depth first
number that can be reached
//from u
{
//put the dfn number for u in dfn array
dfn[u]←dfn_num;
Low[u]←dfn_num;
def_num ← dfn_num +1;

for ( each vertex w adjacent to u ) do
{
 // w is child of u and if w is not visited
if((v!=w) and (dfn[w]<dfn[u])) then
 push(u,w) onto the stack st;
 if(dfn[w]=0) then
{
 if(Low[w]>=dfn[u]) then
 {
   write ("Obtained Articulation Point");
   write("The new bi-connected component :");
   repeat
   {
    edge(x,y)← pop edge from the top of the stack;
    write(x,y);
   }until(((x,y)=(u,w)) OR ((x,y) = (w,u)));
}
 // if w is unvisited then
 Bi_connect(w,u)
 // update Low[u]
 Low[u]←min(Low[u],Low[w]);
}
else if(w!=u) then //the edge u w can be a back edg
then update Low[u]
 Low[u]←min(Low[u],dfn[w]);
}
}
```

For the above given algorithm the time comple remains O(n+E).

UNIT - III

4 Greedy Method

4.1 General Method

Important Points to Remember

- In an algorithmic strategy like Greedy, the decision of solution is taken based on the information available. The Greedy method is a straightforward method. This method is popular for obtaining the optimized solutions.
- In Greedy technique, the solution is constructed through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.
- At each step the choice made should be,

Choice of solution made at each step of problem solving in Greedy approach

- **Feasible** - It should satisfy the problem's constraints.
- **Locally optimal** - Among all feasible solutions the best choice is to be made.
- **Irrevocable** - Once the particular choice is made then it should not get changed on subsequent steps.

- In short, while making a choice there should be a Greed for the optimum solution.

Q.1 Explain the terms, feasible solution, optimal solution and objective function.

☞ [JNTU : Part B, May-09, Marks 8, April-11, Marks 7]

Ans. :

Feasible solution - For solving a particular problem there exists n inputs and we need to obtain a subset that satisfies some constraints. Then any subset that satisfies these constraints is called **feasible solution.**

Optimal solution - From a set of feasible solutions, particular solution that satisfies or nearly satisfies the objectives of the function such a solution is called optimal solution.

Objective function - A feasible solution that either minimizes or maximizes a given objective function is called objective function.

Q.2 Write the general characteristics of Greedy algorithm.

☞ [JNTU : Part B, Marks 5]

Ans. : There are two characteristics of Greedy algorithm

1. Greedy choice property : By this property, a globally optimal solution can be arrived at by making a locally optimal choice. That means, for finding the solution to the problem. We solve the subproblems, and whichever choice. We find best for that subproblem is considered. Then solve the subproblem is considered. Then solve the subproblem arising after the choice is made. This choice may depend upon the previously made choices but it does not depend on any future choice. Thus in greedy method, greedy choices are made one after the another reducing each given problem instance to smaller one. The greedy choice property brings efficiency in solving the problem with the help of subproblems.

2. Optimal substructure : A problem shows optimal substructure if an optimal solution to the problem contains optimal solution to the sub-problems. In other words a problem has optimal substructure if the best next choice always leads to the optimal solution.

In this chapter we will first understand the concept of Greedy method. Then we will discuss various examples for which Greedy method is applied.

Q.3 Write the control abstraction for the greedy method. **[JNTU : Part B, May-12, Marks 7]**

Ans. :

Algorithm

```
Greedy (D, n)
//In Greedy approach D is a domain
//from which solution is to be obtained of size n
//Initially assume
    Solution ← 0
    for i ← 1 to n do
        {
        s ← select (D) // section of solution from D
        If (Feasible (solution, s)) then
            solution ← Union (solution, s);
        }
    return solution
```

Check if the selected solution is feasible or not.

Make a feasible choices and select optimum solution.

In Greedy method following activities are performed.

1. First we select some solution from input domain.
2. Then we check whether the solution is feasible or not.
3. From the set of feasible solutions, particular solution that satisfies or nearly satisfies the objective of the function. Such a solution is called optimal solution.
4. As Greedy method works in stages. At each stage only one input is considered at each time. Based on this input it is decided whether particular input gives the optimal solution or not.

Q.4 Discuss the applications of Greedy Method. **[JNTU : Part B, May-13, Marks 7]**

Ans. :

1) Job sequencing with deadlines
2) Knapsack problem
3) Minimum spanning trees
4) Single source shortest path algorithm.

Q.5 Differentiate between divide and conquer and Greedy method. **[JNTU : Part B, Dec.-12, Marks 7, May-09, Marks 6]**

Ans. :

Sr.No.	Divide and conquer	Greedy algorithm
1.	Divide and conquer is used to obtain a solution to given problem.	Greedy method is used to obtain optimum solution.
2.	In this technique, the problem is divided into small subproblems. These subproblems are solved independently. Finally all the solutions of subproblems are collected together to get the solution to the given problem.	In greedy method a set of feasible solution is generated and optimum solution is picked up.
3.	In this method, duplications in subsolutions are neglected. That means duplicate solutions may be obtained.	In greedy method, the optimum selection is without revising previously generated solutions.
4.	Divide and conquer is less efficient because of rework on solutions.	Greedy method is compartively efficient but there is no as such guarantee of getting optimum solution.
5.	Examples : Quick sort binary search	Examples : Knapsack problem, finding minimum spanning tree.

Q.6 Explain the control abstraction of greedy method compare this with dynamic programming. **[JNTU : Part B, May-12, Marks 15]**

Ans. : Control abstraction- Refer Q.3.

Comparison with Dynamic Programming

Sr. No.	Greedy method	Dynamic programming
1.	Greedy method is used for obtaining optimum solution.	Dynamic programming is also for obtaining optimum solution.
2.	In Greedy method a set of feasible solutions and the picks up the optimum solution.	There is no special set of feasible solutions in this method.
3.	In Greedy method the optimum selection is without revising previously generated solutions.	Dynamic programming considers all possible sequences in order to obtain the optimum solution.
4.	In Greedy method there is no as such guarantee of getting optimum solution.	It is guaranteed that the dynamic programming will generate optimal solution using principle of optimality.

4.2 Job Sequencing with Deadlines

Q.7 State and explain Job sequencing with deadline problem.

Ans. : Consider that there are n jobs that are to be executed. At any time t = 1,2,3,.. only exactly one job is to be executed. The profits pi are given. These profits are gained by corresponding jobs. For obtaining feasible solution we should take care that the jobs get completed within their given deadlines.

Let n = 4

n	pi	di
1	70	2
2	12	1
3	18	2
4	35	1

We will follow following rules to obtain the feasible solution

- Each job takes one unit of time.
- If job starts **before or at its deadline**, profit is obtained, otherwise no profit.
- Goal is to schedule jobs to maximize the total profit.
- Consider all possible schedules and compute minimum total time in the system.

Q.8 Give an algorithm for job sequencing deadline. Also specify the time complexity of algorithm.

Ans. : The algorithm for job sequencing is as below

```
Algorithm Job_seq(D,J,n)
{
//Problem description : This algorithm is for job
sequencing using Greedy method.
//D[i] denotes ith deadline where 1 ≤ i ≤ n
//J[i] denotes ith job
// D[J[i]] ≤ D[J[i+1]]
//Initially
D[0]←0;
J[0]←0;
J[1]←1;
count←1;
for ←2 to n do
{
 t←count;
 while(D[J[t]] > D[i]) AND D[J[t]]!=t)) do t←t-1;
if((D[J[t]] ≤ D[i])AND(D[i] > t)) then
{
  //insertion of ith feasible sequence into J array
  for s←count to (t+1) step – 1 do
  J[s+1] ← J[s];
  J[t+1]←I;
  count←count+1;
 }//end of if
}//end of while
return count;
}
```

The sequence of J will be inserted if and only D[J[t]]!=t. This also means that the job J will processed if it is in within the deadline.

The computing time taken by above Job_ algorithm is $O(n^2)$, because the basic operation computing sequence in array J is within two nest for loops.

Q.9 Solve the following instance of "Jo sequencing with deadlines" problem : " n = 7, profits $(p_1, p_2, p_3, \ldots\ldots, p_7)$ = (3, 5, 2

18, 1, 6, 30) and deadlines $(d_1, d_2, \ldots, d_7)$ = (1, 3, 4, 3, 2, 1, 2).

[JNTU : Part B, May-13, Marks 8]

Ans. :

Step 1 : We will arrange the profits p_i in descending order. Then corresponding deadlines will appear.

Profit	30	20	18	6	5	3	1
Job	P_7	P_3	P_4	P_6	P_2	P_1	P_5
Deadline	2	4	3	1	3	1	2

Step 2 : Create an array J[ ] which stores the jobs. Initially J[ ] will be

1	2	3	4	5	6	7
0	0	0	0	0	0	0

J [7]

Step 3 : Add i^{th} job in array J[ ] at the index denoted by its deadline D_i.

First job is p_7. The deadline for this job is 2. Hence insert p_7 in the array j[ ] at 2^{nd} index.

1	2	3	4	5	6	7
	P_7					

Step 4 : Next job is p_3. Insert it in array J[ ] at index 4.

1	2	3	4	5	6	7
	P_7		P_3			

Step 5 : Next job is p_4. It has a deadline 3. Therefore insert it at index 3

1	2	3	4	5	6	7
	P_7	P_4	P_3			

Step 6 : Next job is p_6, it has deadline 1. Hence place p_6 at index 1.

1	2	3	4	5	6	7
P_6	P_7	P_4	P_3			

Step 7 : Next job is p_2, it has deadline 3. But as 3 is already occupied and there is no empty slot at index < J [3]. Just discard job p_2. Similarly job p_1 and p_5 will get discarded.

Step 8 : Thus the optimal sequence which we will obtain will be 6-7-4-3. The maximum profit will be 74.

Q.10 Explain job-sequencing with deadlines. Solve the following instance :

n = 5.

$(P_1, P_2, P_3, P_4, P_5)$ = (20, 15, 10, 5, 1)

$(d_1, d_2, d_3, d_4, d_5)$ = (2, 2, 1, 3, 3)

[JNTU : Part B, Dec.-12, Marks 8]

Ans. :

Step 1 : We will arrange the profits P_i in descending order, along with corresponding deadlines.

Profit	20	15	10	5	1
Job	P_1	P_2	P_3	P_4	P_5
Deadline	2	2	1	3	3

Step 2 : Create an array J [ ] which stores the jobs. Initially J [ ] will be

1	2	3	4	5
0	0	0	0	0

J [5]

Step 3 : Add i^{th} Job in array J[ ] at index denoted by its deadline D_i.

First job is P_1, its deadline is 2.

Hence insert P_1 in the array J[ ] at 2^{nd} index.

1	2	3	4	5
	P_1			

Step 4 : Next job is P_2 whose deadline is 2 but J[2] is already occupied. We will search for empty location < J[2]. The J[1] is empty. Insert P_2 at J[1].

Step 5 : Next job is P_3 whose deadline is 1. But as J[1] is already occupied discard this job.

Step 6 : Next job is P_4 with deadline 3. Insert P_4 at J[3].

1	2	3	4	5
P_2	P_1	P_4		

Step 7 : Next job is P_5 with deadline 3. But as there is empty slot at < = J[3], we will discard this job.

Step 8 : Thus the optimal sequence which we will obtain will be $P_2 - P_1 - P_4$. The maximum profit will be 40.

Q.11 Solve the following job sequencing problem (maximizing the profit by completing jobs before their deadlines) using greedy algorithm.
N (Number of jobs) = 4
Profits associated with jobs
$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$. Deadlines associated with jobs
$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$. **[JNTU : Part B]**

Ans. : Let Profits associated with jobs $(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$. Deadlines associated with jobs $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$.

Step 1 : We will arrange the profits P_i in descending order. Then corresponding deadlines will appear.

Profit	100	27	15	10
Job	P_1	P_4	P_3	P_2
Deadline	2	1	2	1

Step 2 : Create an array J [ ] which stores the jobs. Initially J [ ] will be

1	2	3	4
0	0	0	0

J [4]

Step 3 : Add i^{th} job in array J [ ] at the index denoted by its deadline d_i. First job is P_1.The deadline for this job is 2. Hence insert P_1 in J [ ] at 2^{nd} index.

1	2	3	4
	P_1		

Step 4 : Next job is P_4. Insert it at index 1.

1	2	3	4
P_4	P_1		

Step 5 : Next job is $P_3$with deadline 2. But the J[2] is already occupied. Hence discard P_3. Similarly job P_2 will also be discarded.

Step 6 : Thus the final optimal sequence which we will obtain will be $P_1 - P_4$. The maximum profit will be 127.

Q.12 How do you reduce/relate job sequencing problem with traveling sales person problem ?
[JNTU : Part B, April-11, Marks 15]

Ans. : The Job Sequencing Problem can be stated as follows -

Consider that there are n jobs that are to be executed At a time t only one job is to be executed. The profits are gained by executing the jobs within given deadlines. The goal is to obtain the sequence of jobs that will maximize total profit.

We can write a job sequence as a vector $(1, j_2, j_3, \ldots j_n)$ which implies that tour starts at city 1, goes to next city j_2 and so on until the final city j_n is reached. To complete the tour, the salesman must travel from j_n back to city 1. The cost of tour will then be

$$\text{cost} = c(1, j_2) + \sum_{k=2}^{n-1} c(j_k, j_{k+1}) + c(j_n, 1)$$

Where c(i, j) specifies the cost of traveling from city to city j when c(i, j) represents the distance i and j the objective of the problem is to minimize total distance traveled.

Q.13 Suppose there are n jobs to be executed but only k processors that can work in parallel. The time required by job i is ti. Write an algorithm that determines which jobs are to be run on which processors and the order in which they should be run so that the finish time of the last job is minimized. **[JNTU : Part B, April-11, Marks 15]**

Ans. : Refer Q.8.

4.3 The 0/1 Knapsack Problem

Q.14 Write an algorithm of greedy knapsack.
[JNTU : Part A, Nov.-16, Marks 2]

Ans. :

The Knapsack problem can be stated as follows :

Suppose there are n objects from i = 1, 2,...n. Each object i has some positive weight w_i and some profit value is associated with each object which is denoted as P_i. This Knapsack carry at the most weight W.

While solving above mentioned Knapsack problem we have the capacity constraint. When we try to solve this problem using Greedy approach our goal is,

1. Choose only those objects that give maximum profit.
2. The total weight of selected objects should be $\leq W$.

And then we can obtain the set of feasible solutions. In other words,

$$\text{maximized} \sum_{n}^{1} p_i x_i \text{ subject to } \sum_{n}^{1} w_i x_i \leq W$$

Where the Knapsack can carry the fraction x_i of an object i such that $0 \leq x_i \leq 1$ and $1 \leq i \leq n$.

Q.15 Write procedure for GREEDY KNAPSACK (P,W,M,X,N) where P and W contains profits and weights, M is Knapsack size and X is the solution vector. ☞ **[JNTU : Part B, May-03, 12, Marks 8]**

Ans. :

```
Algorithm GREEDY_KNAPSACK (P,W,M,X,N)
{
  //P[i] contains the profits of i items.
  //W[i] contains weights of i items.
  //where 1 ≤ i ≤ N.
  // X[i] is the solution vector.
  //M is Knapsack size.
    for (i:=1 to n) do
    {
      if (W[i]<M) then  //with the capacity
      {
        X[i]:=1.0;
        M:= M – W[i];
      }
    }
    if (i<= N) then
    X[i] := M/W[i];      // X[i] gives the output.
}
```

Q.16 Consider that there are three items. Weight and profit value of each item is as given below.

i	W_i	P_i
1	18	30
2	15	21
3	10	18

Also W = 20. Obtain the solution for the above given Knapsack problem. ☞ **[JNTU : Part B, Marks 5]**

Ans. : The feasible solutions are as given below.

X_1	X_2	X_3
1/2	1/3	1/4
1	2/15	0
0	2/3	1
0	1	1/2

Let us compute $\sum W_i X_i$

1. 1/2 *18 + 1/3 *15 + 1/4 *10 = 16.5
2. 1*18 + 2/15 *15 + 0*8
   = 20
3. 0*18 +2/3*15 +10
   = 20
4. 0*18 +1*15 +1/2*10
   = 20

Let us compute $\sum P_i X_i$

1. 1/2 *30 + 1/3 *21 + 1/4 *18
   = 26.5
2. 1*30 + 2/15 *21 + 0*18
   = 32.8
3. 0*30 +2/3*21 +18
   = 32
4. 0*30 +1*21 +1/2*18
   = 30

To summarize this

$\sum W_i X_i$	$\sum P_i X_i$
16.5	26.5
20	32.8
20	32
20	30

The solution 2 gives the maximum profit and hence it turns out to be optimum solution.

Q.17 Find the optimal solution of the Knapsack instance n = 7, M = 15, $(P_1,P_2,\ldots P_7)$ = (10, 5, 15, 7, 6, 18, 3) and $(w_1,w_2,w_3,\ldots w_7)$ = (2, 3, 5, 7, 1, 4, 1) ☞ **[JNTU : Part B, Marks 5]**

Ans. : For solving this problem, we will find out the fraction of weight that can be used to fill up the Knapsack satisfying the given capacity with

maximum weight. We will try various solutions for the same.

Feasible solutions can be

Solution	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1/2	1/3	1/4	1/7	1	3/4	1
2	1/2	1/3	1	1/7	0	1	1/2
3	1	1/3	1	1/7	1	1	1
4	1	1/3	1	1/7	0	1	1
5	1	2/3	1	0	1	1	1

Solution 1 :

$$\sum w_i x_i = \left(2*\frac{1}{2}\right)+\left(3*\frac{1}{3}\right)+\left(5*\frac{1}{4}\right)+\left(7*\frac{1}{7}\right)+(1*1)+\left(4*\frac{3}{4}\right)+1$$

$$= 1 + 1 + 1.25 + 1 + 1 + 3 + 1$$

$$\sum w_i x_i = 9.25$$

$$\sum p_i x_i = \left(10*\frac{1}{2}\right)+\left(5*\frac{1}{3}\right)+\left(15*\frac{1}{4}\right)+\left(7*\frac{1}{7}\right)+(6*1)+\left(18*\frac{3}{4}\right)+3$$

$$= 5 + 1.67 + 3.75 + 1 + 6 + 13.5 + 3$$

$$\sum p_i x_i = 33.92$$

Solution 2 :

$$\sum w_i x_i = \left(2*\frac{1}{2}\right)+\left(3*\frac{1}{3}\right)+(5*1)+\left(7*\frac{1}{7}\right)+(1*0)+(4*1)+\left(1*\frac{1}{2}\right)$$

$$= 1 + 1 + 5 + 1 + 0 + 4 + 0.5$$

$$\sum w_i x_i = 12.5$$

$$\sum p_i x_i = \left(10*\frac{1}{2}\right)+\left(5*\frac{1}{3}\right)+(15*1)+\left(7*\frac{1}{7}\right)+(6*0)+(18*1)+\left(3*\frac{1}{2}\right)$$

$$= 5 + 1.67 + 15 + 0 + 18 + 1.5$$

$$\sum p_i x_i = 42.17$$

Solution 3 :

$$\sum w_i x_i = (2*1)+\left(3*\frac{1}{3}\right)+(5*1)+\left(7*\frac{1}{7}\right)+(1*1)+(4*1)+(1*1)$$

$$= 2 + 1 + 5 + 1 + 1 + 4 + 1$$

$$\sum w_i x_i = 15$$

$$\sum p_i x_i = (10*1)+\left(5*\frac{1}{3}\right)+(15*1)+\left(7*\frac{1}{7}\right)+(6*1)+(18*1)+(3*1)$$

$$= 10 + 1.67 + 15 + 1 + 6 + 18 + 3$$

$$\sum p_i x_i = 54.67$$

Solution 4 :

$$\sum w_i x_i = (2*1)+\left(3*\frac{1}{3}\right)+(5*1)+\left(7*\frac{1}{7}\right)+(1*0)+(4*1)+(1*1)$$

$$\sum w_i x_i = 14$$

$$\sum p_i x_i = (10*1)+\left(5*\frac{1}{3}\right)+(15*1)+\left(7*\frac{1}{7}\right)+(6*0)+(18*1)+(3*1)$$

$$\sum p_i x_i = 48.67$$

Solution 5 :

$$\sum w_i x_i = (2*1)+\left(3*\frac{2}{3}\right)+(5*1)+(7*0)+(1*1)+(4*1)+(1*1)$$

$$= 2 + 2 + 5 + 1 + 4 + 1$$

$$\sum w_i x_i = 15$$

$$\sum p_i x_i = (10*1)+\left(5*\frac{2}{3}\right)+(15*1)+(7*0)+(6*1)+(18*1)+(3*1)$$

$$= 10 + 3.33 + 15 + 0 + 6 + 18 + 3$$

$$\sum p_i x_i = 55.33$$

From above computations, clearly solution 5 gi maximum profit. Hence solution 5 is considered optimum solution.

Q.18 If $p_1/w_1 \geq p_2/w_2 \ldots \geq p_n/w_n$ prove tha knapsack generates an optimal solution to t given instance of the knapsack problem.

☞ **[JNTU : Part B, May-08, Marks 8]**

OR

If objects are selected in order of decreasing v then prove that the algorithm knapsack finds optimal solution.

☞ **[JNTU : Part B, May-08, Dec.-11, April-11, Marks 8]**

Ans. : Proof : Let, $x = (x_1, x_2, \ldots x_n)$ be the set of solutions for the knapsack problem using Greedy approach. If all $x_i = 1$ then the solution of the problem itself is a optimal solution.

Now consider a least index j such that $x_j \neq 1$. From the algorithm, $x_i = 1$ when $1 \leq i \leq j$ and $x_i = 0$ when $j < i \leq n$ and x_i is either less than or equal to zero or may be less than 1.

Let, $y = (y_1, y_2, \ldots y_n)$ be an optimal solution such that $\sum w_i y_i = m$ where m denotes the capacity of knapsack.

Let k be a least index with three possibilities -

1. If $k < j$ then $x_k = 1$ but $y_k \neq x_k$ and $y_k < x_k$.
2. if $k = j$ then $\sum w_i x_i = m$ and $y_i = x_i$. It leads to the solution of knapsack.
3. If $k > j$ then $\sum w_i y_i > m$ and it cannot be a part of solution. There exists a new solution set $z = (z_1, z_2, \ldots, z_n)$ with $z_i = x_i$, $1 \leq i < k$ and $\sum_{k<i\leq n} w_i (y_i - z_i) = w_k (z_k - y_k)$.

This can be formulated as -

$$\sum_{1\leq i\leq n} p_i z_i = \sum_{1\leq i\leq n} p_i y_i + (z_k - y_k) w_k \frac{p_k}{w_k} - \sum_{k<i\leq n} (y_i - z_i) w_i \frac{p_i}{w_i}$$

$$\geq \sum_{1\leq i\leq n} p_i y_i + \left[(z_k - y_k) w_k - \sum_{k<i\leq n} (y_i - z_i) w_i \right] \frac{p_k}{w_k}$$

$$= \sum_{1\leq i\leq n} p_i y_i$$

If $\sum p_i z_i > p_i y_i$ then that means y is clearly greater than z and hence y cannot be optimal. Then remaining cases are either $z = x$ or $z \neq x$. If $z = x$ and x is optimal then transform y to x. This ultimately proves that x too is optimal.

4.4 Minimum Cost Spanning Trees

Important Points to Remember

- There are two algorithms for obtaining the minimum spanning tree namely – Prim's Algorithm and Kruskal's algorithm.
- Strategy to obtain minimum spanning tree using Prim's Algorithm –
  1. Start from the source vertex. Pick up the minimum weighted edge.
  2. Pick up the next edge by covering any of the already visited vertex.
  3. While selecting the edge - just make sure that the circuit is not getting formed.
  4. When all the vertices get visited then stop the procedure. Display the obtained minimum spanning tree.
- Strategy to obtain minimum spanning tree using Kruskal's Algorithm
  1. Start from the source vertex. Pick up the minimum weighted edge.
  2. Pick up the any minimum weighted edge each time.
  3. While selecting the edge - just make sure that the circuit is not getting formed.
  4. When all the vertices get visited then stop the procedure.

Q.19 Explain the concept of minimum spanning tree. ☞ [JNTU : Part A, Marks 3]

Ans. : Minimum spanning tree of weighted connected graph G is a spanning tree with minimum or smallest weight.

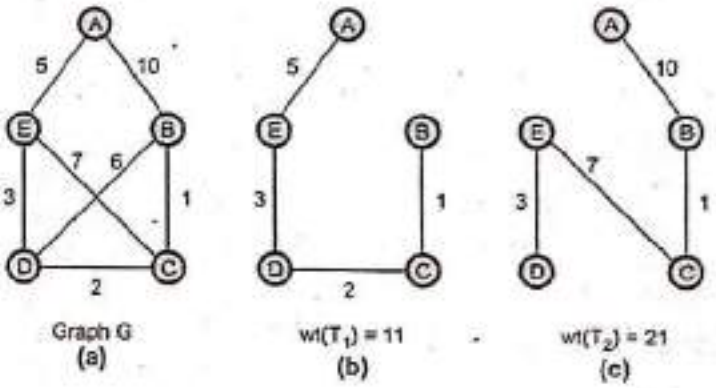


Fig. Q.19.1 Graph and two spanning trees out of which (b) is a minimum spanning tree

Q.20 Differentiate between Prims and Kruskal's algorithm. ☞ [JNTU : Part B, May-09, 12, Marks 6]

Ans. :

Sr. No.	Prims Algorithm	Kruskal's Algorithm
1.	Prim's algorithm initializes with node	Kriskal's algorithm initializes with edge

2.	In Prim's algorithm the next node is always selected from previously selected node.	In Kruskal's algorithm the minimum weight edge is selected independent of earlier selection.
3.	In Prim's algorithm graph must be connected.	The Kruskal's algorithm can work with disconnected graph.
4.	The time complexity of Prim's algorithm is $O(V^2)$	The time complexity of Kruskal's algorithm is $O(\log V)$.

Example of Prim's Algorithm – Refer Q.21, Q.22.

Example of Kruskal's Algorithm – Refer Q.24, Q.25.

Q.21 Explain the Prim's algorithm with the appropriate example.

[JNTU : Part B, April-11, Marks 7, Nov.-16, Marks 10]

Ans. : Consider the graph given below :

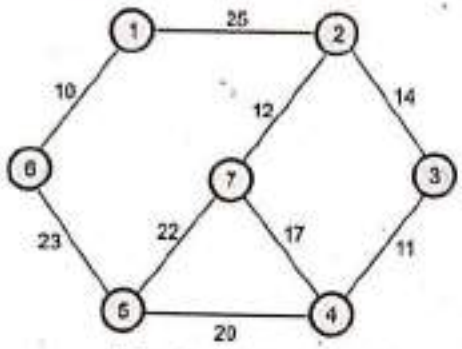
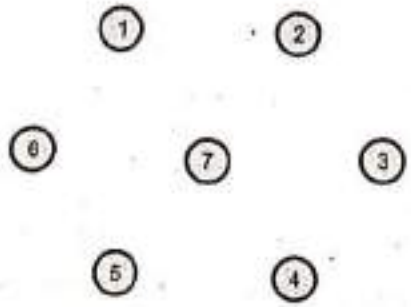


Fig. Q.21.1 Graph

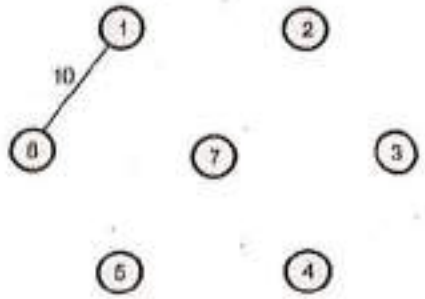
Now, we will consider all the vertices first. Then we will select an edge with minimum weight. The algorithm proceeds by selecting adjacent edges with minimum weight. Care should be taken for not forming circuit.

Step 1 :



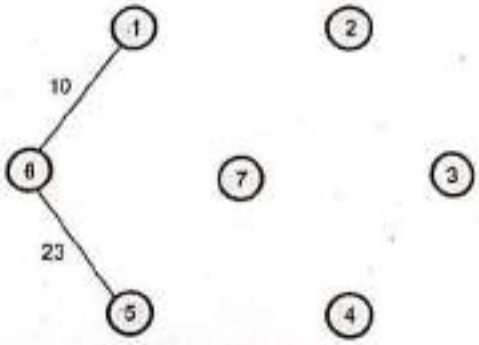
Total weight = 0

Step 2 :



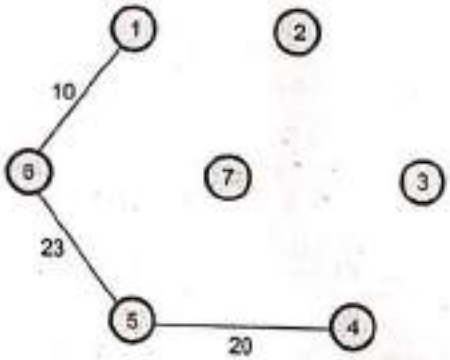
Total weight = 10

Step 3 :



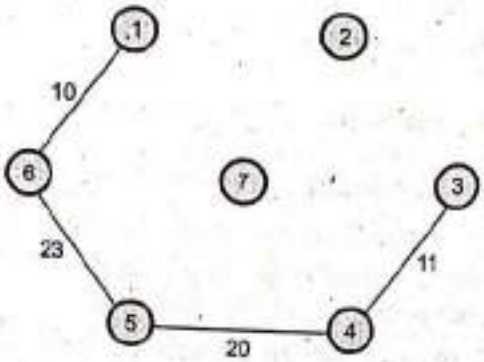
Total weight = 33

Step 4 :



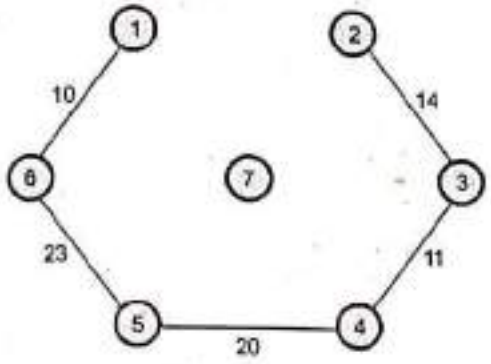
Total weight = 53

Step 5 :



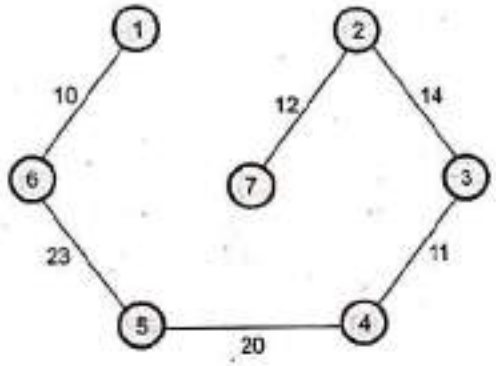
Total weight = 64

Step 6 :



Total weight = 78

Step 7 :



Total weight = 90

Q.22 Write Prim's Algorithm.
[JNTU : Part B, Marks 5]

Ans. :

Algorithm

```
Prim(G[0...Size – 1,0...Size – 1],nodes)
//Problem Description : This algorithm is for
implementing
//Prim's algorithm for finding spanning tree
//Input : Weighted Graph G and total number of nodes
//Output : Spanning tree gets printed with total path
length
   total=0;
  //Initialize the selected vertices list
    for i←0 to nodes–1 do
      tree[i] ← 0
      tree[0] = 1;//take initial vertex
       for k←1 to nodes do
       {
          min_dist ← ∞
          //initially assign minimum dist as infinity
          for i← 0 to nodes–1 do
          {
               for j←0 to nodes–1 do
          {
               if (G [i,j]AND ((tree[i]AND !tree[j])OR
                (!tree[i] AND tree[j])))then
                {
                  if(G[i,j] < min_dist)then
                        {
                        min_dist←G[i,j]
                        v1← i
                        v2← j
                      }
                }
          }
          }
Write(v1,v2,min_dist);
     tree[v1] ← tree[v2] ← 1
   total ← total+min_dist
   }
   Write("Total Path Length Is",total)
```

Select an edge such that one vertex is selected and other is not and the edge has the least weight.

Obtained edge with minimum wt.

Picking up those vertices yielding minimum edge.

Q.23 Give the time complexity of Prim's Algorithm.
[JNTU : Part B, Marks 5]

Ans. : The algorithm spends most of the time in selecting the edge with minimum length. Hence the **basic operation** of this algorithm is to find the edge with **minimum path length.** This can be given by following formula.

$$T(n) = \sum_{k=1}^{nodes-1} \left(\sum_{i=0}^{nodes-1} 1 + \sum_{j=0}^{nodes-1} 1 \right)$$

Time taken by for (k = 1 to nodes–1) loop — Time taken by for (i = 0 to nodes–1) loop — Time taken by for (j s= 0 to nodes–1) loop

We take variable n for 'nodes' for the sake of simplicity of solving the equation then

$$T(n) = \sum_{k=1}^{n-1} \left(\sum_{i=0}^{n-1} 1 + \sum_{j=0}^{n-1} 1 \right)$$

$$= \sum_{k=1}^{n-1} [((n-1)+0+1)+((n-1)+0+1)]$$

$$= \sum_{k=1}^{n-1} (2n) = 2n \sum_{k=1}^{n-1} 1$$

$$= 2n[(n-1)-1+1] = 2n(n-1)$$

$$= 2n^2 - 2n$$

$\therefore T(n) = n^2$

$\therefore T(n) = \Theta(n^2)$

But n stands for total number of nodes or vertices in the tree. Hence we can also state

> Time complexity of Prim's Algorithm is $\Theta(|V^2|)$.

But if the Prim's algorithm is implemented using binary heap with the creation of graph using adjacency list the its time complexity becomes $\Theta(E \log_2 V)$ where E stands for total number of edges and V stands for total numb of vertices.

Q.24 Explain the Kruskal's algorithm with the appropriate example. ☞ [JNTU : Part B, Marks 5]

Ans. : Consider the graph given below :

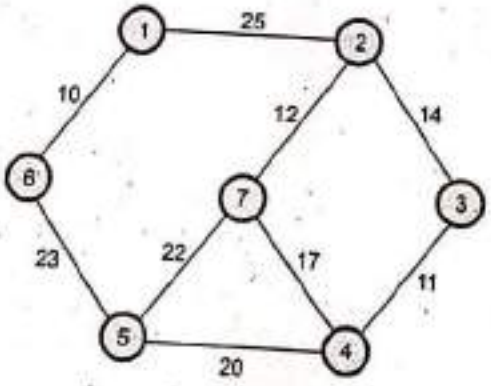
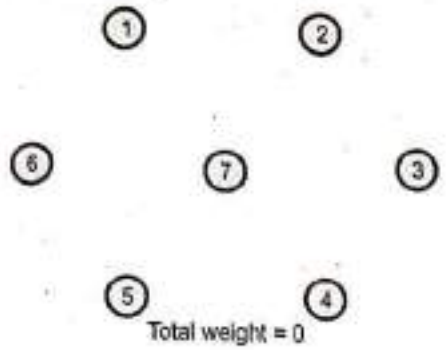


Fig. Q.24.1 Graph

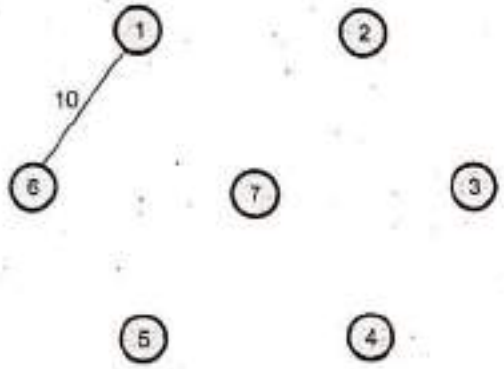
First we will select all the vertices. Then an edge with optimum weight is selected from heap, even though it not adjacent to previously selected edge. Care should be taken for not forming circuit.

Step 1 :



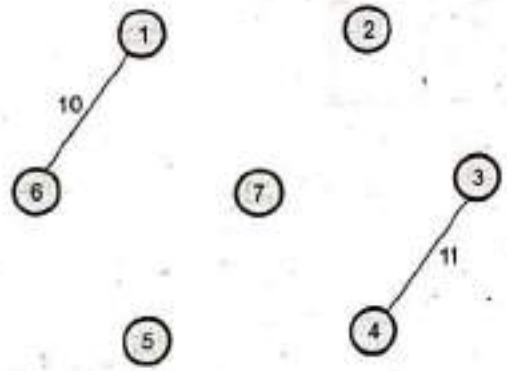
Total weight = 0

Step 2 :



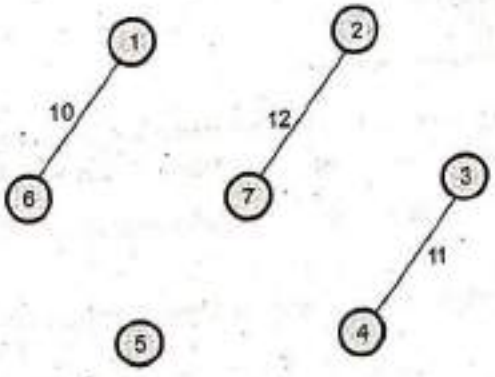
Total weight = 10

Step 3 :



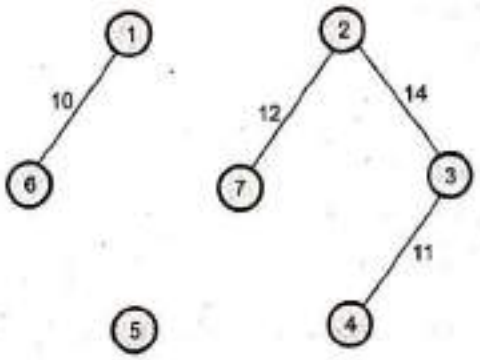
Total weight = 21

Step 4 :



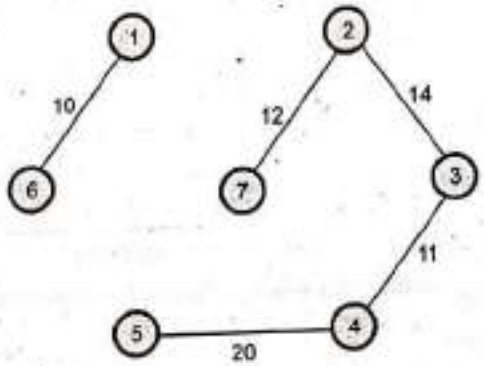
Total weight = 33

Step 5 :



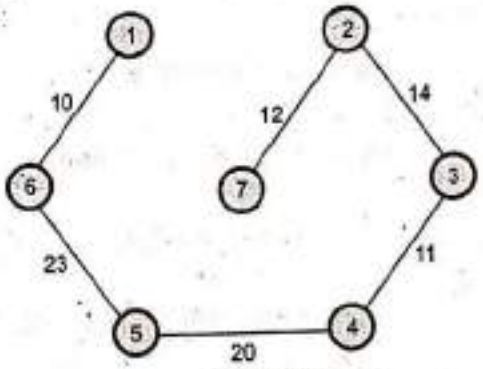
Total weight = 47

Step 6 :



Total weight = 67

Step 7 :



Total weight = 90

Q.25 Write Kruskals algorithm that generates minimum spanning tree for every connected undirected graph.

☞ **[JNTU : Part B, May-12, Marks 15, Dec.-12, Marks 7]**

Ans. :

Algorithm spanning_tree()

//Problem Description : This algorithm finds the

//minimum spanning tree using Kruskal's algorithm

//Input : The adjacency matrix graph G containing cost

```
//Output : prints the spanning tree with the total cost of
//spanning tree
count←0
k←0
sum←0
  for i←0 to tot_nodes do
        parent[i]←i
while(count!=tot_nodes-1)do
{
  pos←Minimum(tot_edges);//finding the minimum cost edge
      if(pos=-1)then//Perhaps no node in the graph
            break
      v1←G[pos].v1
      v2←G[pos].v2
      i←Find(v1,parent)
      j←Find(v2,parent)
    if(i!=j)then
    {
    tree[k][0] ←v1
//storing the minimum edge in array tree[]
      tree[k][1] ←v2
       k++
       count++;
       sum+←G[pos].cost
//accumulating the total cost of MST
       Union(i,j,parent);
       }
       G[pos].cost INFINITY
      }
      if(count=tot_nodes-1)then
      {
          for i←0 to tot_nodes-1
       {
          write(tree[i][0],tree[i][1])
       }
       write("Cost of Spanning Tree is ",sum)
    }
```

tree [ ] [ ] is an array in which the spanning tree edges are stored.

Computing total cost of all the minimum distances.

For each node of i, the minimum distance edges are collected in array **tree [ ] [ ]**. The spanning tree is printed here.

Q.26 Compute the time complexity of deriving minimum spanning tree from the weighted connected gra using Kruskal's algorithm.

[JNTU : Part B, April-11, Marks 7, Dec.-12, Marks

Ans. : The time complexity of Kruskal's algorithm is O(ElogV) or O(ElogE) where E is total number of edges, V total number of vertices.

Proof : Let, E is the total number of edges. In Kruskal's algorithm following are the two main functions that performed -

i) Determining edge with minimum cost.

ii) Delete the lowest cost edge from the list of edges.

Both these functions can be done efficiently if the list of edges E is maintained as a sorted list. But it is not necessary to sort all the edges. If the edges are maintained as a minheap then the next edge can be obtained in O(log|E|) time. The construction of heap takes O(|E|) time. Using union and find algorithm the MST can be constructed in O(|E|log|E|). This proves that the computing time of Kruskal's algorithm is O(ElogE).

Q.27 Show that in a complete graph with n vertices, the number of spanning trees generated can not be greater than $(2^{n-1} - 2)$.

[JNTU : Part B, April-11, Marks 8]

Ans. : For a complete graph K_n with n vertices, the total number of spanning trees are n^{n-2}.

For example consider a complete graph with 3 vertices.

i.e. $n^{n-2} = 3^{3-2} = 3^1 = 3$

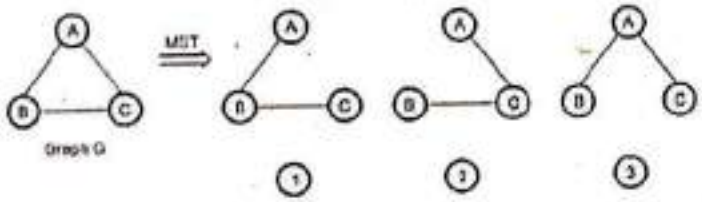


Fig. Q.27.1

Q.28 Does either Prim's or Kruskal's algorithm work if there are negative edge weights.

[JNTU : Part B, April-11, Marks 8]

Ans. : Yes the Prim's or Kruskal's algorithm work when there are negative weights in the graph. Because in finding minimum spanning tree using these algorithms. We select an edge that has minimum weight. The order in which vertices are added to minimum spanning tree depends only on ordering of the weight of the edge. Hence there is no effect of negative edge in generation of minimum spanning tree.

Q.29 Suppose we want to find the minimum spanning tree of the following graph 2.
a) Run Prim's algorithm; whenever there is a choice of nodes, always use alphabetic ordering (e.g., start from node A). Draw a table showing the intermediate values of the cost array.
b) Run Kruskal's algorithm on the same graph. Show how the disjoint-sets data structure looks at every intermediate stage (including the structure of the directed trees), assuming path compression is used.

[JNTU : Part B, Dec.-09, Marks 16, April-11, Marks 15]

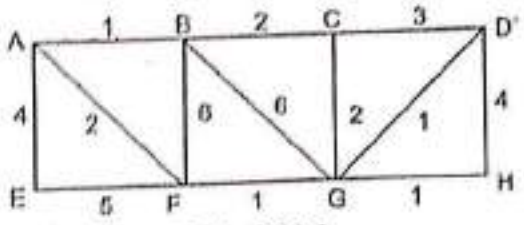
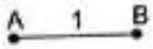


Fig. Q.29.1

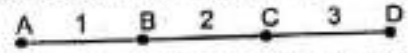
Ans. : a) Prim's algorithm

Step 1 : Select an edge AB with minimum weight

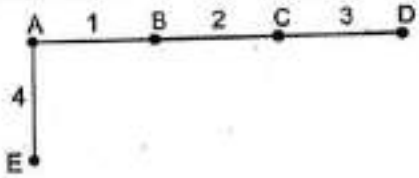


Step 2 : Select next minimum weight BC edge .

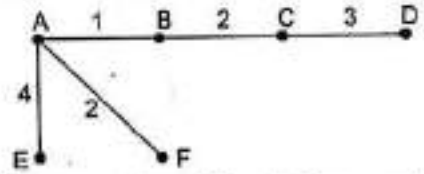
Step 3 : Select next minimum weight edge CD.



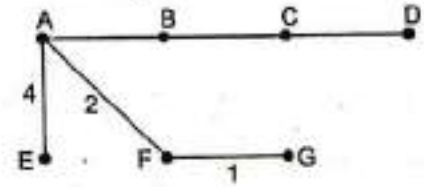
Step 4 : Select next minimum weight edge AE.



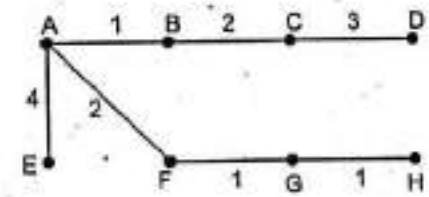
Step 5 : Select an edge with minimum weight.



Step 6 : Select an edge with minimum weight FG.

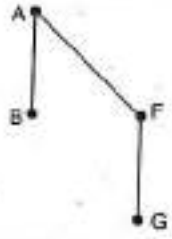
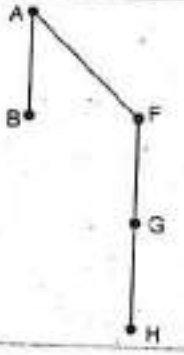
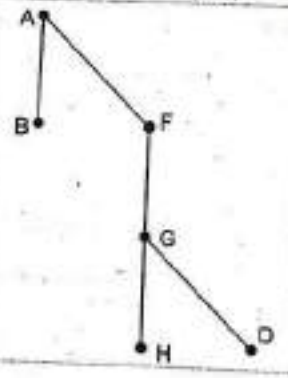
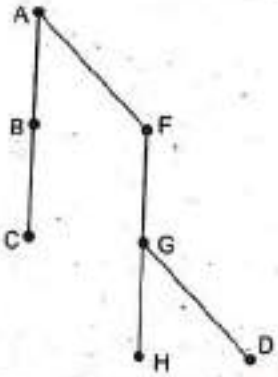
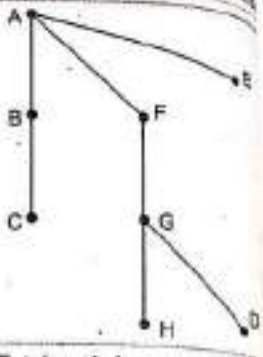


Step 7 : Select an edge with minimum weight i.e. GH



As all the vertices get visited the path length = 14 in an alphabetic order.

b) Kruskal's algorithm

Step 1 : Select an edge AB	Disjoint-set (path compression) 
Step 2 : Select an edge FG	
Step 3 : Select an edge GH	
Step 4 : Select an edge GD	
Step 5 : Select an edge BC	
Step 6 : Select an edge AE	
	Total path length = 12

4.5 Single Source Shortest Path Problem

Q.30 Write the pseudocode of a simpler version o Dijkstra's algorithm that finds only the distance (i.e. the lengths of shortest paths but not shortes paths themselves) from a given vertex to all othe vertices of a graph represented by its weigh matrix.

[JNTU : Part B, Dec.-11, Marks 15, May-17, Marks 5]

Ans. :

```
void Dijkstra(int tot_nodes,int cost[10][10],int source,in
dist[])
{
int i,j,v1,v2,min_dist;
int s[10];
for(i=0;i<tot_nodes;i++)
{
      dist[i]=cost[source][i];//initially put the
      s[i]=0; //distance from source vertex to i
      //i is varied for each vertex
      path[i]=source;//all the sources are put in path
}
s[source]=1;
for(i=1;i<tot_nodes;i++)
{
      min_dist=infinity;
      v1=-1;//reset previous value of v1
      for(j=0;j<tot_nodes;j++)
      {
      if(s[j]==0)
      {
         if(dist[j]<min_dist)
         {
          min_dist=dist[j];//finding minimum disatnc
          v1=j;
         }
```

```
        }
        }
        s[v1]=1;
        for(v2=0;v2<tot_nodes;v2++)
        {
        if(s[v2]==0)
        {
           if(dist[v1]+cost[v1][v2]<dist[v2])
           {
              dist[v2]=dist[v1]+cost[v1][v2];
              path[v2]=v1;
           }
        }
        }
    }
}
void display(int source,int destination,int dist[])
{
 int i;
 getch();
 printf("\n Step by Step shortest path is...\n");
 for(i=destination;i!=source;i=path[i])
 {
       printf("%d<--",i);
 }
 printf("%d",i);
 printf(" The length=%d",dist[destination]);
}
```

Q.31 Apply Dijkstraw's algorithm for finding all shortest paths from single source 'P' in a given graph.

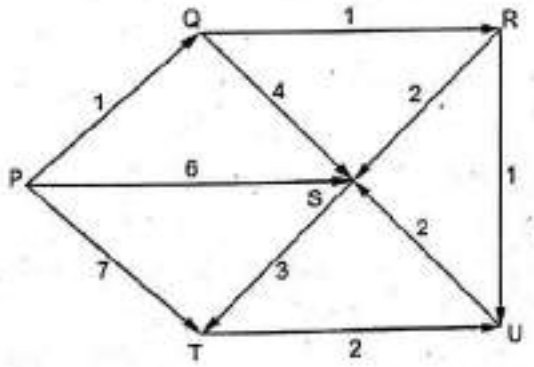


Fig. Q.31.1

Ans. :

Step 1 : We will always choose the vertex with minimum cost. We will start with vertex 'P'. Hence set S = {P}. The remainly distances are

$d(P - Q) = 1 \qquad d(P - R) = \infty$

$d(P - S) = 6 \qquad d(P - U) = \infty$

$d(P - T) = 7$

Step 2 : Choose vertex Q.

$\therefore \quad S = \{P, Q\}, T = \{R, S, T, U\}$

$d(P - S) = d(P - Q) + d(Q - S) = 5$

$d(P - T) = 7$

$d(P - R) = d(P - Q) + d(Q - R) = 2$

$d(P - U) = \infty$

Step 3 : Choose vertex R

$\therefore \quad S = \{P, Q, R\}, T = \{S, T, U\}$

$d(P - S) = d(P - Q) + d(Q - R) + d(R - S) = 4$

$d(P - T) = 7$

$d(P - U) = d(P - R) + d(R - U) = 2 + 1 = 3$

Step 4 : Choose vertex U.

$\therefore \quad S = \{P, Q, R, U\}, T = \{S, T\}$

$d(P - S) = 4$

$d(P - T) = 7$

Step 5 : Choose vertex S

$\therefore \quad S = \{P, Q, R, U, S\}, T = \{T\}$

$d(P - T) = 7$

Thus we get shortest distances to all other vertices from vertex P. These distances are represented as follows.

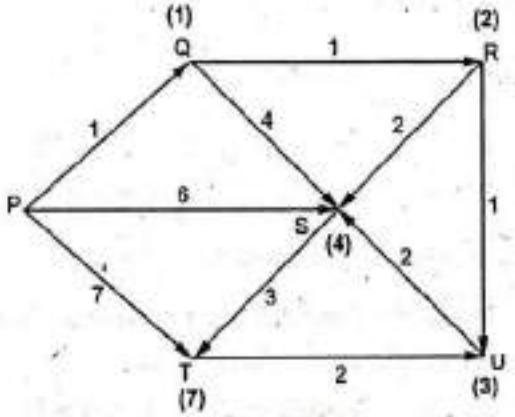


Fig. Q.31.2

UNIT - IV

5 Dynamic Programming

5.1 General Method

Important Points to Remember

- Dynamic programming is technique for solving problems with overlapping sub-problems.
- In this method each sub-problem is solved only once. The result of each sub-problem is recorded in a table from which we can obtain a solution to the original problem.
- Dynamic programming is typically applied to optimization problems.
- Examples of the problems that can be solved using dynamic programming are - finding Binomial coefficient, optimal binary search tree (OBST), 0/1 Knapsack problem, matrix chain multiplication.

Q.1 Write control abstraction (General Strategy Algorithm) of dynamic programming.

☞ [JNTU : Part A, Marks 3]

Ans. : Dynamic programming design involves four major steps :

1. Characterize the structure of optimal solution. That means develop a mathematical notation that can express any solution and sub-solution for the given problem.
2. Recursively define the value of an optimal solution.
3. By using bottom up technique compute value of optimal solution. For that you have to develop a recurrence relation that relates a solution to its sub-solutions, using the mathematical notation of step 1.
4. Compute an optimal solution from computed information.

Q.2 What is the difference between divide and conquer and dynamic programming ?

☞ [JNTU : Part A, Marks 3]

Ans. :

Sr. No.	Divide and conquer	Dynamic programming
1.	The problem is divided into small subproblems. These subproblems are solved independently. Finally all the solutions of subproblems are collected together to get the solution to the given problem.	In dynamic programming many decision sequences are generated and all the overlapping subinstances are considered.
2.	In this method duplications in subsolutions are neglected. i.e., duplicate subsolutions may be obtained.	In dynamic computing duplications in solution is avoided totally.
3.	Divide and conquer is less efficient because of rework on solutions.	Dynamic programming is efficient than divide and conquer strategy.
4.	The divide and conquer uses top down approach of problem solving (recursive methods).	Dynamic programming uses bottom up approach of problem solving (iterative method).
5.	Divide and conquer splits its input at specific deterministic points usually in the middle.	Dynamic programming splits its input at every possible split points rather than at a particular point. After trying all split points it determines which split point is optimal.

Q.3 What are the common steps in the dynamic programming to solve any problem ?

☞ [JNTU : Part A, Marks 2]

Ans. : Dynamic programming design involves major steps :

1. Characterize the structure of optimal solution That means develop a mathematical notation th

can express any solution and subsolution for the given problem.

2. Recursively define the value of an optimal solution.
3. By using bottom up technique compute value of optimal solution. For that you have to develop a recurrence relation that relates a solution to its subsolutions, using the mathematical notation of step 1.
4. Compute an optimal solution from computed information.

Q.4 Comment on the statement : "In dynamic programming, many decision sequences may be generated". **[JNTU : Part A, Marks 3]**

Ans. : Dynamic programming is typically applied to optimization problems for a given problem we may get any number of solutions. From all those solutions we seek for optimum solution and such an optimum solution becomes the solution to the given problem. In this method subproblem is solved only once. The result of each subproblem is recorded in a table from which we can obtain a solution to the original problem. Hence in dynamic programming many decision sequences are generated and all the overlapping sub instances are considered.

Q.5 Write the principle of optimality.
[JNTU : Part A, Nov.-16, Marks 3]

Ans. :

- The dynamic programming algorithm obtains the solution using principle of optimality.
- The principle of optimality states that "in an optimal sequence of decisions or choices, each subsequence must also be optimal."
- When it is not possible to apply the principle of optimality it is almost impossible to obtain the solution using the dynamic programming approach.
- For example : Finding of shortest path in a given graph uses the principle of optimality.

Q.6 Name the elements of dynamic programming.
[JNTU : Part A, Marks 3]

Ans. :

1) **Optimal substructure** - The dynamic programming technique makes use of principle of optimality to find the optimal solution from subproblems.
2) **Overlapping subproblems** - The dynamic programming is a technique in which the problem is divided into subproblems. The solutions of subproblems are shared to get the final solution to the problem. It avoids repetition of work and we can get the solution more efficiently.

Q.7 What are the applications of Dynamic programming ? **[JNTU : Part A, Marks 2]**

Ans. : Various applications that are implemented using dynamic programming are -

1. Multi-stage graph
2. Optimal binary search trees.
3. 0/1 Knapsack problem.
4. All pairs shortest path problem.
5. Traveling salesperson problem.
6. Reliability design.

5.2 Multistage Graphs

Q.8 Find a minimum cost path from 'S' to 't' in multistage graph using dynamic programming.
[JNTU : Part B, Marks 5]

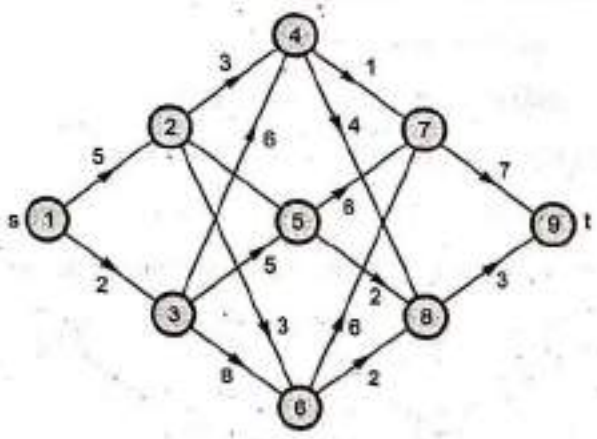


Fig. Q.8.1

Ans. : At stage 1 We will consider vertex 1 i.e. 's'

At stage 2 We will consider vertices 2 and 3

At stage 3 We will consider vertices 4, 5 and 6

At stage 4 We will consider vertices 7 and 8

At stage 5 We will consider vertex 9 i.e. 't'

Now, using backward approach, let us compute d(s, t) as follows -

$$d(s, t) = \min\{5 + d(2, t), 2 + d(3, t)\}$$

$$d(2, t) = \min\{3 + d(4, t), 3 + d(6, t)\}$$

$$d(3, t) = \min\{6 + d(4, t), 5 + d(5, t), 8 + d(6, t)\}$$

$$d(4, t) = \min\{1 + d(7, t), 4 + d(8, t)\}$$

$$d(5, t) = \min\{6 + d(7, t), 2 + d(8, t)\}$$

$$d(6, t) = \min\{6 + d(7, t), 2 + d(8, t)\}$$

$$d(7, t) = 7$$

$$d(8, t) = 3$$

$\therefore$ $d(6, t) = \min(6 + 7, 2 + 3)$

$\therefore$ **$d(6, t) = 5$ 6 – 8 – t**

Now, $d(5, t) = \min(6 + 7, 2 + 3)$

$\therefore$ **$d(5, t) = 5$ 5 – 8 – t**

Now, $d(4, t) = \min(1 + 7, 4 + 3)$

$d(4, t) = 7$ 4 – 8 – t

Now, $d(3, t) = \min(6 + 7, 5 + 5, 8 + 5)$

$\therefore$ **$d(3, t) = 10$ 3 – 5 – 8 – t**

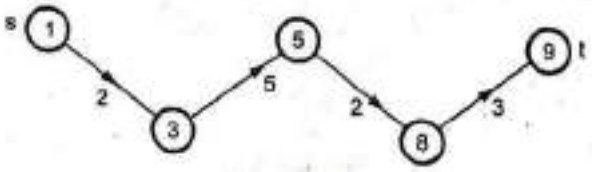
Now, $d(2, t) = \min(3 + 7, 3 + 5)$

$\therefore$ **$d(2, t) = 8$ 2 – 6 – 8 – t**

Now, $d(s, t) = \min(5 + 8, 2 + 10)$

$\therefore$ **$d(s, t) = 12$ 5 – 3 – 5 – 8 – t**

Thus we get the minimum cost path from s to t as



Cost = 12

Fig. Q.8.2

Q.9 Solve the given multistage graph.

[JNTU : Part B, Marks 5]

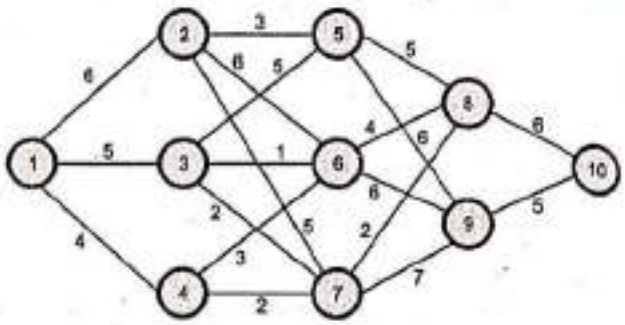


Fig. Q.9.1

Ans. :

At stage 1 we will consider vertex 1.

At stage 2 we will consider vertex 2, 3, 4.

At stage 3 we will consider vertex 5, 6, 7.

At stage 4 we will consider vertex 8, 9.

At stage 5 we will consider vertex 10.

Now using backward approach, let us complete d(1,10) i.e. source to destination.

$$d(1,10) = \min\{6 + d(2, 10), 5 + d(3, 10), 4 + d(4, 10)\}$$

$$d(2,10) = \min\{3 + d(5, 10), 6 + d(6, 10), 5 + d(7, 10)\}$$

$$d(3,10) = \min\{5 + d(5, 10), 1 + d(6, 10), 2 + d(7, 10)\}$$

$$d(4,10) = \min\{3 + d(6, 10), 2 + d(7, 10)\}$$

$$d(5,10) = \min\{5 + d(8, 10), 6 + d(9, 10)\}$$

$$d(6,10) = \min\{4 + d(8, 10), 6 + d(9, 10)\}$$

$$d(7,10) = \min\{2 + d(8, 10), 7 + d(9, 10)\}$$

$d(8,10) = 6$

$d(9,10) = 5$

$\therefore$ $d(7,10) = \min\{2 + d(8, 10), 7 + d(9, 10)\}$

$= \min\{2 + 6, \quad 7 + 5\}$

$d(7,10) = 8$ **7 - 8 - 10**

$\therefore$ $d(6,10) = \min\{4 + d(8, 10), 6 + d(9, 10)\}$

$= \min\{4 + 6, \quad 6 + 5\}$

$d(6,10) = 10$ **6 - 8 - 10**

$\therefore$ $d(5,10) = \min\{5 + d(8, 10), 6 + d(9, 10)\}$

$= \min\{5 + 6, \quad 6 + 5\}$

$d(5,10) = 11$ **5 - 8 - 10**

$\therefore$ $d(4,10) = \min\{3 + d(6, 10), 2 + d(7, 10)\}$

$= \min\{3+10,\ 2+8\}$

$d(4,10) = 10$ 4 - 7 - 8 - 10

$\therefore d(3,10) = \min\{5+d(5,10), 1+d(6,10), 2+d(7,10)\}$

$= \min\{5+11,\ 1+10, 2+8\}$

$d(3,10) = 10$ 3 - 7 - 8 - 10

$\therefore d(2,10) = \min\{3+d(5,10), 6+d(6,10), 5+d(7,10)\}$

$= \min\{3+11,\ 6+10, 5+8\}$

$d(2,10) = 13$ 2 - 7 - 8 - 10

$\therefore d(1,10) = \min(6+d(2,10), 5+d(3,10), 4+d(4,10))$

$= \min(6+13,\ 5+10, 4+10)$

$d(1,10) = 14$ 1 - 4 - 7 - 8 - 10

Thus we get minimum cost path as

The path with minimum cost is S - A - E - T with the cost 9.

This method is called **backward reasoning**. The algorithm for this method is as follows.

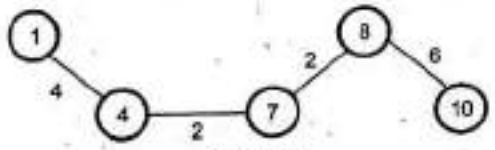


Fig. Q.9.2 (a)

Q.10 Write an algorithm for multistage Graph problem using backward approach.

[JNTU : Part B, Marks 5]

Ans. :

```
Algorithm Backward_Gr (G, stages, n, p)
// Problem description : This algorithm is for backward
// approach of multistage graph
// Input : The multistage graph G = (V, E),
// 'stages' is for total number of stages
// n is total number of vertices of G
// p is an array for restoring path
// Output : The path with minimum cost.
    back_cost[i] ← 0
    for (i ← 0 to n – 2) do
    {
        r ← Get_Min (i,n) // r is edge with min cost
        back_cost[i] ← back_cost[r] + c[r][i]
        d[i] ← r
    }
    // finding minimum cost path
    p[0] ← 0
    p[stages – 1] ← n – 1
    for (i ← stages – 1 to 1) do
        p[i] ← d[p[i + 1]];
```

Assigning minimum distance in 'd' array

Computing backward cost of each vertex

5.3 Optimal Binary Search Tree

Important Points to Remember

- Let $(a_1, a_2, \ldots a_n)$ be a set of identifiers such that $a_1 < a_2 < a_3$. Let p(i) be the probability with which we can search for a_i and q(i) be the probability of searching element x such that $a_i < x < a_{i+1}$ and $0 \le i \le n$. In other words p(i) is the probability of successful search and q(i) be the probability of unsuccessful search. Also $\sum_{1 \le i \le n} p(i) + \sum_{1 \le i \le n} q(i)$ then obtain a tree with minimum cost. Such a tree with optimum cost is called optimal binary search tree.
- We can obtain optimal binary search tree by building the optimal subtrees. This ultimately shows that optimal binary search tree follows the principle optimality.

Q.11 Consider n = 4 and (q_1, q_2, q_3, q_4) = (do, if, int, while). The values for p's and q's are given as p(1 : 4) = (3, 3, 1, 1) and q(0 : 4) = (2, 3, 1, 1, 1). Construct the optimal binary search tree.

[JNTU : Part B, Marks 15, Dec.-09, Marks 8]

Ans. : Let,

$$W_{i,i} = q_i,\ r_{i,i} = 0,\ C_{i,i} = 0$$

$$W_{i,i+1} = q_i + q_{(i+1)} + P_{(i+1)}$$

$$r_{i,i+1} = i + 1$$

$$C_{i,i+1} = q_i + q_{(i+1)} + P_{(i+1)}$$

$$W_{i,j} = W_{i,j-1} + p_j + q_j$$

$$r_{i,j} = k$$

$$C_{i,j} = \min_{i < k \le j} \{C_{(i,k-1)} + C_{k,j}\} + W_{i,j}$$

We will construct tables for values of W, C and r.

Let $i = 0$

$$W_{00} = q_0 = 2$$

When i = 1

$W_{11} = 3$

When i = 2

$W_{22} = 1$

When i = 3

$W_{33} = 1$

When i = 4

$W_{44} = q_4 = 1$

When i = 0 and j – i = 1 then

$W_{01} = q_0 + q_1 + p_1$

$= 2 + 3 + 3$

$W_{01} = 8$

When i =1 and j – i = 2

$W_{12} = q_1 + q_2 + p_2$

$= 3 + 1 + 3$

$W_{12} = 7$

When i = 2 and j – i = 3

$W_{23} = q_2 + q_3 + p_3$

$= 1 + 1 + 1$

$W_{23} = 3$

When i = 3 and j – i = 4

$W_{34} = q_3 + q_4 + p_4$

$= 1 + 1 + 1$

$W_{34} = 3$

Now, when i = 0 and j – i = 2

$W_{ij} = W_{i, j-1} + p_j + q_j$

$W_{02} = W_{01} + p_2 + q_2$

$= 8 + 3 + 1$

$W_{02} = 12$

When i =1 and j – i = 2

$W_{13} = W_{12} + p_3 + q_3$

$= 7 + 1 + 1$

$W_{13} = 9$

When i = 2 and j – i = 2 then

$W_{24} = W_{23} + p_4 + q_4$

$= 3 + 1 + 1$

$W_{24} = 5$

Now, when i = 0 and j – i = 3 then

$W_{03} = W_{02} + p_3 + q_3$

$= 12 + 1 + 1$

$W_{03} = 14$

When i = 1 and j – i = 3 then

$W_{14} = W_{13} + q_4 + p_4$

$= 9 + 1 + 1$

$W_{14} = 11$

When i = 0 and j – i = 4 then

$W_{04} = W_{03} + q_4 + p_4$

$= 14 + 1 + 1$

$W_{04} = 16$

The table for W can be represented as

j – i \ i	0	1	2	3	4
0	$W_{00} = 2$	$W_{11} = 3$	$W_{22} = 1$	$W_{33} = 1$	$W_{44} = 1$
1	$W_{01} = 8$	$W_{12} = 7$	$W_{12} = 3$	$W_{34} = 3$	
2	$W_{02} = 12$	$W_{13} = 9$	$W_{24} = 5$		
3	$W_{03} = 14$	$W_{14} = 11$			
4	$W_{04} = 16$				

We will now compute for C and r.

As $C_{i, i} = 0$ and $r_{i, i} = 0$

$C_{00} = 0$ $C_{11} = 0$ $C_{22} = 0$ $C_{33} = 0$ $C_{44} = 0$

$r_{00} = 0$ $r_{11} = 0$ $r_{22} = 0$ $r_{33} = 0$
$r_{44} = 0$

Similarly, $C_{i, i+1} = q_i + q_{(i+1)} + p_{(i+1)}$ and $r_{i, i+1} = i+1$

∴ When i = 0

$C_{01} = q_0 + q_1 + p_1$

$= 2 + 3 + 3$

$C_{01} = 8$

$r_{01} = 1$

When i = 1

$C_{12} = q_1 + q_2 + p_2$.

$= 3 + 1 + 3$

$C_{12} = 7$ and $r_{12} = 2$

When i = 2

$C_{23} = q_2 + q_3 + p_3$

$= 1 + 1 + 1$

$C_{23} = 3$ and $r_{23} = 3$

When i = 1

$C_{34} = q_3 + q_4 + p_4$

$= 1 + 1 + 1$

$C_{34} = 3$ and $r_{34} = 4$

Now we will compute C_{ij} and r_{ij} for $j - 1 \geq 2$.

As $C_{i,j} = \min_{i<k\leq j} \{C_{(i,k-1)} + C_{k,j}\} + W_{i,j}$

Hence we will find k.

For C_{02} we have i = 0 and j = 2. For $r_{i,j-1}$ to $r_{i+1,j}$ i.e. for r_{01} to $r_{1,2}$. We will compute minimum value of C_{ij}.

Let $r_{01} = 1$ and $r_{12} = 2$. Then we will assume value of k =1 and will compute C_{ij}. Similarly with k = 2 we will compute C_{ij} and will pick up minimum value of C_{ij} only.

Let us compute C_{ij} with following formula,

$C_{ij} = C_{i,k-1} + C_{kj}$

For k = 1, i = 0, j = 2.

$C_{02} = C_{00} + C_{12}$

$C_{02} = 0 + 7$

$C_{02} = 7$ ...(Q.11.1)

For k = 2, i = 0, j = 2

$C_{02} = C_{01} + C_{22} = 8 + 0$

$C_{02} = 8$ ...(Q.11.2)

From equations (Q.11.1) and (Q.11.2) we can select minimum value of C_{02} is 7. That means k = 1 gives us minimum value of $C_{i,j}$.

Hence $r_{ij} = r_{02} = k = 1$

Now $C_{02} = \min\{C_{(i,k-1)} + C_{kj}\} + W_{ij}$

$= 7 + W_{02}$

$= 7 + 12$

$C_{02} = 19$

Continuing in this fashion we can compute C_{ij} and r_{ij}. It is as given below.

j-i	0	1	2	3	4
0	$C_{00}=0$ $r_{00}=0$	$C_{11}=0$ $r_{11}=0$	$C_{22}=0$ $r_{22}=0$	$C_{33}=0$ $r_{33}=0$	$C_{44}=0$ $r_{44}=0$
1	$C_{01}=8$ $r_{01}=1$	$C_{12}=7$ $r_{12}=2$	$C_{23}=3$ $r_{23}=3$	$C_{34}=3$ $r_{34}=4$	
2	$C_{02}=19$ $r_{02}=1$	$C_{13}=12$ $r_{13}=2$	$C_{24}=8$ $r_{24}=3$		
3	$C_{03}=25$ $r_{03}=2$	$C_{14}=19$ $r_{14}=2$			
4	$C_{04}=32$ $r_{04}=2$				

Therefore T_{04} has a root r_{04}. The value of r_{04} is 2. From (a_1, a_2, a_3, a_4) = (do, if, int, while) a_2 becomes the root node.

$r_{ij} = k$

$r_{04} = 2$

Then r_{ik-1} becomes left child and r_{kj} becomes the right child. In other words r_{01} becomes the left child and r_{24} becomes right child of r_{ij}. Here $r_{01} = 1$ so a_1 becomes left child of a_2 and $r_{24} = 3$ so a_3 becomes right child of a_2.

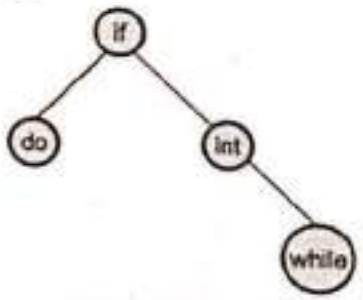


Fig. Q.11.1 OBST

For the node r_{24} i = 2, j = 4 and k =3. Hence left child of it is $r_{ik-1} = r_{22} = 0$. That means left child of $r_{24} = a_3$ is empty. The right child of r_{24} is $r_{34} = 4$. Hence a_4 becomes right child of a_3. Thus all n = 4 nodes are used to build a tree. The optimal binary search tree is

Q.12 Using algorithm OBST compute W(i, j), R(i, j) and C(i, j) $0 \le i \le j \le 4$, for the identifier set (a_1, a_2, a_3, a_4) = (end, goto, print, stop) with P(1) = 1/20, P(2) = 1/5, P(3) = 1/10, P(4) =1/20. Q(0) = 1/5, Q(1) = 1/10, Q(2) = 1/5, Q(3) = 1/20, Q(4) = 1/20. Using R(i, j) construct the optimal binary search tree.

☞ [JNTU : Part B, May-09, Marks 16]

Ans. : We will multiply values of P's and Q's by 20, for convenience. Hence,

P(1) = 1, P(2) = 4, P(3) = 2, P(4) = 1. Q(0) = 4, Q(1) = 2, Q(2) = 4, Q(3) = 1, Q(4) = 1

Now we will compute,

$W_{i,i} = q_i$, $R_{ii} = 0$, $C_{ii} = 0$

$W_{i,i+1} = q_i + q_{(i+1)} + P_{(i+1)}$

$R_{i,i+1} = i + 1$

$C_{i,i+1} = q_i + q_{(i+1)} + P_{(i+1)}$

Let i = 0,

$W_{00} = q_0 = 4$

$W_{11} = q_1 = 2$

$W_{22} = q_2 = 4$

$W_{33} = q_3 = 1$

$W_{44} = q_4 = 1$

$W_{01} = q_0 + q_1 + p_1$

$= 4 + 2 + 1$

$W_{01} = 7$

$W_{12} = q_1 + q_2 + p_2$

$= 2 + 4 + 4$

$W_{12} = 10$

$W_{23} = q_2 + q_3 + p_3$

$= 4 + 1 + 2$

$W_{23} = 7$

$W_{34} = q_3 + q_4 + p_4$

$= 1 + 1 + 1$

$W_{34} = 3$

Now we will use formula

$$W_{ij} = W_{ij-1} + p_j + q_j$$

Hence, let i = 0, j = 2 then

$W_{02} = W_{01} + p_2 + q_2$

$= 7 + 4 + 4$

$W_{02} = 15$

Let i = 1, j = 3

$W_{13} = W_{12} + p_3 + q_3$

$= 10 + 2 + 1$

$W_{13} = 13$

Let i = 2, j = 4

$W_{24} = W_{23} + p_4 + q_4$

$= 7 + 1 + 1$

$W_{24} = 9$

Let i = 0, j = 3

$W_{03} = W_{02} + p_3 + q_3$

$= 15 + 2 + 1$

$W_{02} = 18$

Let i = 1, j = 4

$W_{14} = W_{13} + p_4 + q_4$

$= 13 + 1 + 1$

$W_{14} = 15$

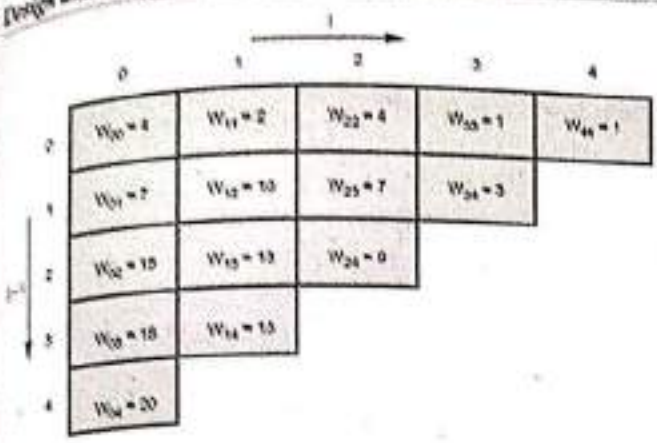
Let i = 0, j = 4

$W_{04} = W_{03} + p_4 + q_4$

$= 18 + 1 + 1$

$W_{04} = 20$

The tabular representation for values of W are as shown below.



The computation of C and R values is done as below.

$C_{00} = 0$

$R_{00} = 0$

$C_{11} = 0$

$R_{11} = 0$

$C_{22} = 0$

$R_{22} = 0$

$C_{33} = 0$

$R_{33} = 0$

$C_{44} = 0$

$R_{44} = 0$

For computation of $C_{i, i+1}$ and $R_{i, i+1}$

We will use following formula

$$C_{i, i+1} = q_i + q_{(i+1)} + P_{(i+1)}$$

$$R_{i, i+1} = i + 1$$

Let i = 0 then

$C_{01} = q_0 + q_1 + P_1$ $R_{01} = 1$

$= 4 + 2 + 1$

$C_{01} = 7$

$C_{12} = q_1 + q_2 + P_2$ $R_{12} = 2$

$= 2 + 4 + 4$

$C_{12} = 10$

$C_{23} = q_2 + q_3 + P_3$ $R_{23} = 3$

$= 4 + 1 + 2$

$C_{23} = 7$

$C_{34} = q_3 + q_4 + P_4$

$= 1 + 1 + 1$

$C_{34} = 3$ $R_{34} = 4$

To compute C_{ij} and R_{ij} we will use following formulae.

$$C_{ij} = \min_{i < k \le j} \{C(i, k-1) + C_{kj}\} + W_{ij}$$

$$R_{ij} = k$$

Value of k lies between $R_{i, j-1}$ and $R_{(i+1), j}$. Let us compute C_{02}.

Here i = 0 and j = 2. The value of k lies between $R_{01} = 1$ to $R_{12} = 2$.

When k = 1, i = 0, j = 2.

$C_{02} = C_{00} + C_{12}$

$C_{02} = 0 + 10$

$C_{02} = 10$

When k = 2, i = 0, j = 2.

$C_{02} = C_{01} + C_{22}$

$C_{02} = 7 + 0$

$C_{02} = 7 \rightarrow$ Minimum value of C.

We will select k = 2, as we obtain C value as minimum when k = 2. Hence $R_{02} = 2$ then $C_{02} = W_{02} + C_{02} = 15 + 7 = 22$.

Let us compute C_{13} the range for k is $R_{12} = 2$ to $R_{23} = 3$.

When k = 2, i = 1, j = 3.

$C_{13} = C_{11} + C_{23}$

$= 0 + 7$

$C_{13} = 7 \rightarrow$ Minimum value

When k = 3, i = 1, j = 3

$C_{13} = C_{12} + C_{33}$

$= 10 + 0$

$C_{13} = 10$

As we obtain minimum value of C_{13} with k = 2. Hence value of k becomes 2.

$\therefore \quad R_{13} = 2$

$C_{13} = W_{13} + \min \{ C_{13} \}$

$= 13 + 7$

$\therefore \quad C_{13} = 20$

Now we will compute C_{24}.

i = 2, j = 4.

Value of k lies between $R_{23} = 3$ to $R_{34} = 4$. When k = 3, i = 2 and j = 4.

$C_{24} = C_{22} + C_{34}$

$= 0 + 3$

$C_{24} = 3 \quad \rightarrow$ Minimum value

When k = 4, i = 2, j = 4

$C_{24} = C_{23} + C_{44}$

$= 7 + 0$

$C_{24} = 7$

Hence we set k = 3.

$\therefore \quad R_{24} = 3$

$C_{24} = W_{24} + \min \{C_{24}\}$

$= 9 + 3$

$\therefore \quad C_{24} = 12$

To compute C_{03}, i = 0, j = 3.

k lies between $R_{02} = 2$ to $R_{13} = 2$.

That means k = 2. $R_{03} = 2$.

$\therefore \quad C_{03} = \{C_{01} + C_{23}\} + W_{03}$

$= 7 + 7 + 18$

$\therefore \quad C_{03} = 32$

To compute C_{14}, i = 1, j = 4.

k lies between $R_{13} = 2$ to $R_{24} = 3$.

When k = 2, i = 1, j = 4

$C_{14} = C_{11} + C_{24}$

$= 0 + 12$

$\therefore \quad C_{14} = 12 \quad \rightarrow$ Minimum value

When k = 3, i = 1, j = 4

$C_{14} = C_{12} + C_{34}$

$= 10 + 3$

$\therefore \quad C_{14} = 13$

We will set k = 2

$\therefore \quad R_{14} = 2$

$C_{14} = W_{14} + \min \{C_{14}\}$

$= 15 + 12$

$\therefore \quad C_{14} = 27$

To compute C_{04}

i = 0, j = 4

k lies between $R_{03} = 2$ to $R_{14} = 2$

That means k = 2 $\quad \therefore R_{04} = 2$

$\therefore \quad C_{04} = \{C_{01} + C_{24}\} + W_{04}$

$C_{04} = \{7 + 12\} + 20$

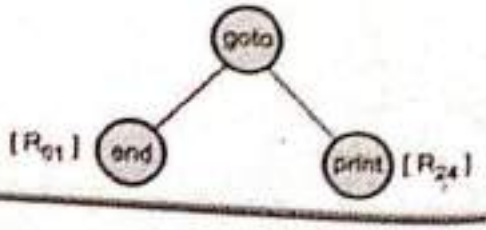
$\therefore \quad C_{04} = 39$

The values are summarized as below.

	0	1	2	3	4
0	$C_{00} = 0$ $R_{00} = 0$	$C_{11} = 0$ $R_{11} = 0$	$C_{22} = 0$ $R_{22} = 0$	$C_{33} = 0$ $R_{33} = 0$	$C_{44} = 0$ $R_{44} = 0$
1	$C_{01} = 7$ $R_{01} = 1$	$C_{12} = 10$ $R_{12} = 2$	$C_{23} = 7$ $R_{23} = 3$	$C_{34} = 3$ $R_{34} = 4$	
2	$C_{02} = 22$ $R_{02} = 2$	$C_{13} = 20$ $R_{13} = 2$	$C_{24} = 12$ $R_{24} = 3$		
3	$C_{03} = 32$ $R_{03} = 2$	$C_{14} = 27$ $R_{14} = 2$			
4	$C_{04} = 39$ $R_{04} = 2$				

Let us build tree using R_{ij}.

As $R_{04} = 2$, a_2 will be root.



Left child of $R_{ij} = k$ is R_{ik-1} i.e. R_{01}. As $R_{01} = 1$, a_1 becomes left child. Right child of $R_{ij} = k$ is R_{kj} i.e. $R_{24} = 3$, a_3 becomes right child.

For $R_{01} = 1$ left child is $R_{00} = 0$. That means there is no left child to node 'end'. For $R_{01} = 1$ right child is $R_{11} = 0$. That means there is no right child to node 'end'. Consider $R_{24} = 3$, left child is $R_{22} = 0$. That means there is no left child to node 'print'. For $R_{24} = 3$, right child is $R_{34} = 4$. That means a_4 = stop becomes right child of node 'print'. The final OBST is

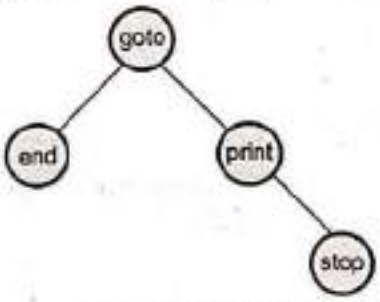


Fig. Q.12.1 OBST

Q.13 Consider 4 elements $a_1 < a_2 < a_3 < a_4$ with $q_0 = 0.25$, $q_1 = 3/16$, $q_2 = q_3 = q_4 = 1/16$. $p_1 = 1/4$, $p_2 = 1/8$, $p_3 = p_4 = 1/16$.

a) Construct the optimal binary search tree as a minimum cost tree.

b) Construct the table of values W_{ij}, C_{ij}, V_{ij} computed by the algorithm to compute the roots of optimal subtrees. [JNTU : Part B, Marks 5]

Ans. : For convenience we will multiply the probabilities q_i and p_i by 16. Hence p_i and q_i values are $(p_1, p_2, p_3, p_4) = (4, 2, 1, 1)$

$$(q_0, q_1, q_2, q_3, q_4) = (4, 3, 1, 1, 1)$$

Let us compute values of W first. For W_{ii} computation we will use,

$$W_{ii} = q_i$$

Hence $W_{00} = q_0$

$\therefore$ $W_{00} = 4$

$W_{11} = q_1$

$\therefore$ $W_{11} = 3$

$W_{22} = q_2$

$\therefore$ $W_{22} = 1$

$W_{33} = q_3$

$\therefore$ $W_{33} = 1$

$W_{44} = q_4$

$\therefore$ $W_{44} = 1$

Compute W_{ii+1} using formula

$$W_{ii+1} = q_i + q_{i+1} + p_{i+1}$$

Hence, $W_{01} = q_0 + q_1 + p_1 = 4 + 3 + 4$

$\therefore$ $W_{01} = 11$

$W_{12} = q_1 + q_2 + p_2 = 3 + 1 + 2$

$\therefore$ $W_{12} = 6$

$W_{23} = q_2 + q_3 + p_3 = 1 + 1 + 1$

$\therefore$ $W_{23} = 3$

$W_{34} = q_3 + q_4 + p_4 = 1 + 1 + 1$

$\therefore$ $W_{34} = 3$

For computing W_{ij} we will use following formula

$$W_{ij} = W_{ij-1} + p_j + q_j$$

$\therefore$ $W_{02} = W_{01} + p_2 + q_2 = 11 + 2 + 1$

$\therefore$ $W_{02} = 14$

$\therefore$ $W_{13} = W_{12} + p_3 + q_3 = 6 + 1 + 1$

$\therefore$ $W_{13} = 8$

$\therefore$ $W_{24} = W_{23} + p_4 + q_4$

$= 3 + 1 + 1$

$\therefore$ $W_{24} = 5$

Now, $W_{03} = W_{02} + p_3 + q_3$

$= 14 + 1 + 1$

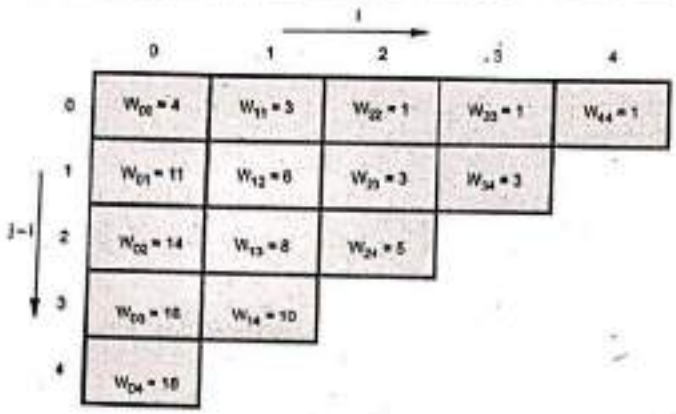
$\therefore$ $W_{03} = 16$

$W_{04} = W_{03} + p_4 + q_4$

$= 16 + 1 + 1$

$\therefore$ $W_{04} = 18$

To summarize W values,



To compute C and r values we will use $C_{ii} = 0$ and $r_{ii} = 0$.

Hence $C_{00} = 0$

$r_{00} = 0$

$C_{11} = 0$

$r_{11} = 0$

$C_{22} = 0$

$r_{22} = 0$

$C_{33} = 0$

$r_{33} = 0$

$C_{44} = 0$

$r_{44} = 0$

To compute C_{ii+1} and r_{ii+1} we will use following formulae

$$r_{ii+1} = i + 1$$

$$C_{ii+1} = q_i + q_{i+1} + p_{i+1}$$

Hence $r_{01} = 1$

$C_{01} = q_0 + q_1 + p_1$

$= 4 + 3 + 4$

$\therefore C_{01} = 11$

$r_{12} = 2$

$C_{12} = q_1 + q_2 + p_2$

$= 3 + 1 + 2$

$\therefore C_{12} = 6$

$r_{23} = 3$

$C_{23} = q_2 + q_3 + p_3$

$= 1 + 1 + 1$

$\therefore C_{23} = 3$

$r_{34} = 4$

$C_{34} = q_3 + q_4 + p_4 = 1 + 1 + 1$

$\therefore C_{34} = 3$

To compute C_{ij} and r_{ij} we will compute the values of k first.

The value of k lies between values of $r_{i, j-1}$ to $r_{i+1, j}$. The min $\{C_{i\,k-1} + C_{kj}\}$ decides value of k.

Consider i = 0 and j = 2.

To decide value of k, the range for k is $r_{01} = 1$ to $r_{12} = 2$ when k = 1 and i = 0, j = 2. We will compute C_{ij} using formula

$$C_{ij} = C_{ik-1} + C_{kj}$$

$\therefore C_{02} = C_{00} + C_{02}$

$C_{02} = 6 \rightarrow$ Minimum value of C_{02}.

When k = 2, and i = 0, j = 2.

$C_{02} = C_{01} + C_{22}$

$C_{02} = 11 + 0$

$\therefore C_{02} = 11$

In above computations we get minimum value of C_{02} when k = 1. Hence the value of k becomes 1.

As $r_{ij} = k$

$r_{02} = 1$

For $C_{ij} = W_{ij} + \min \{C_{i, k-1} + C_{kj}\}$

$\therefore C_{02} = W_{02} + C_{02}$

$C_{02} = 14 + 6 = 20$

Now for r_{13} and C_{13}

k is between $r_{12} = 2$ to $r_{23} = 3$ then when k = 2 and i = 1, j = 3.

$C_{13} = C_{11} + C_{23}$

$= 0 + 3$

$C_{13} = 3 \rightarrow$ Minimum value of C_{13}

$\therefore$

When k = 3 and i = 1, j = 3.

$$C_{13} = C_{12} + C_{33}$$
$$= 6 + 0$$

$\therefore$ $C_{13} = 6$

Hence k = 2 gives minimum value of C_{13}.

$\therefore$ $r_{13} = 2$

and $C_{13} = W_{13} + C_{13}$ (Minimum value)
$= 8 + 3$

$\therefore$ $C_{13} = 11$

Now for r_{24} and C_{24}

k is between $r_{23} = 3$ to $r_{34} = 4$.

$\therefore$ i = 2, j = 4 when k = 3.

$$C_{24} = C_{22} + C_{34}$$
$$= 0 + 3$$

$\therefore$ $C_{24} = 3$

When k = 4, i = 2, j = 4.

$$C_{24} = C_{23} + C_{44}$$
$$= 3 + 0$$

$\therefore$ $C_{24} = 3$

That means value of k can be 3. Let us consider k = 3. Then,

$r_{24} = 3$

and $C_{34} = W_{34} + C_{34}$
$= 5 + 3$

$\therefore$ $C_{34} = 8$

Now for r_{03} and C_{03}

Value of k lies between $r_{02} = 1$ to $r_{13} = 2$

When k = 1 and i = 0, j = 3.

$$C_{03} = C_{00} + C_{13}$$
$$= 0 + 11$$

$\therefore$ $C_{03} = 11 \rightarrow$ Minimum value of C_{03}.

When k = 2 and i = 0, j = 3.

$$C_{03} = C_{01} + C_{23}$$
$$= 11 + 3$$

$\therefore$ $C_{03} = 14$

Hence value of k = 1.

$\therefore$ $r_{03} = 1$

and $C_{03} = W_{03} + C_{03}$
$= 16 + 11$

$\therefore$ $C_{03} = 27$

Now for r_{14} and C_{14}

Value of k is between $r_{13} = 2$ to $r_{24} = 3$.

When k = 2 and i = 1, j = 4 then

$$C_{14} = C_{11} + C_{24}$$
$$= 0 + 8$$

$\therefore$ $C_{14} = 8 \rightarrow$ Minimum value of C_{14}

When k = 3 and i = 1, j = 4 then

$$C_{14} = C_{12} + C_{34}$$
$$= 6 + 3$$

$\therefore$ $C_{14} = 9$

Hence value of k = 2

$\therefore$ $r_{14} = 2$

and $C_{14} = W_{14} + C_{14}$
$= 10 + 8$

$C_{14} = 18$

Now for computing r_{04} and C_{04}

Value of k is between $r_{03} = 1$ to $r_{14} = 2$ then,

When k = 1 and i = 0, j = 4.

$$C_{04} = C_{00} + C_{14}$$
$$= 0 + 18$$

$\therefore$ $C_{04} = 18 \rightarrow$ Minimum value of C_{04}

When k = 2 and i = 0, j = 4.

$$C_{04} = C_{01} + C_{24}$$
$$= 11 + 8$$
$$\therefore \quad C_{04} = 19$$

Hence value of k = 1

$$r_{04} = 1$$
$$\therefore \quad C_{04} = W_{04} + C_{04}$$
$$= 18 + 18$$
$$\therefore \quad C_{04} = 36$$

To summarize

	0	1	2	3	4
0	$C_{00}=0$ $r_{00}=0$	$C_{11}=0$ $r_{11}=0$	$C_{22}=0$ $r_{22}=0$	$C_{33}=0$ $r_{33}=0$	$C_{44}=0$ $r_{44}=0$
1	$C_{01}=11$ $r_{01}=1$	$C_{12}=6$ $r_{12}=2$	$C_{23}=3$ $r_{23}=3$	$C_{34}=3$ $r_{34}=4$	
2	$C_{02}=20$ $r_{02}=1$	$C_{13}=11$ $r_{13}=2$	$C_{24}=8$ $r_{24}=3$		
3	$C_{03}=27$ $r_{03}=1$	$C_{14}=18$ $r_{14}=2$			
4	$C_{04}=36$ $r_{04}=1$				

To build the OBST

$$r_{04} = 1$$

Hence a_1 becomes root node

To compute children of a_1 we will apply following formula

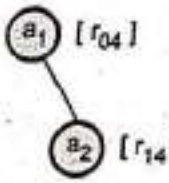
For node $r_{ij} = k$ then

left child is r_{ik-1} and right child is r_{kj}

Hence for $r_{04} = 1$, for node a_1

i = 0, j = 4 and k = 1.

Left child = $r_{00} = 0$. That is empty node.



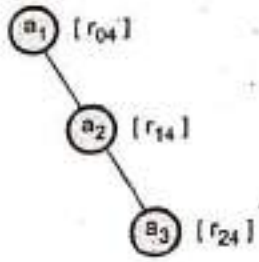
Right child = $r_{14} = 2$. That means a_2 is right child of a_1.

For node $r_{14} = 2$ for node a_2

i = 1, j = 4 and k = 2.

Left child = $r_{11} = 0$. That is empty node.

Right child = $r_{24} = 3$. That means a_3 is right child of a_2.



For node $r_{24} = 3$ i.e. for node a_3

i = 2, j = 4, and k = 3.

Left child = $r_{22} = 0$. That means no left child.

Right child = $r_{34} = 4$. That means a_4 is right child of node a_3.

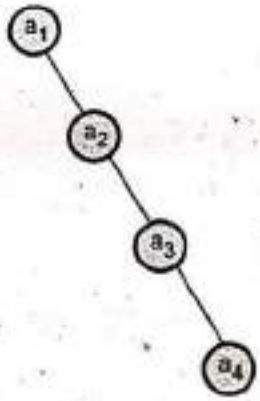


Fig. Q.13.1 OBST

is the required OBST.

Q.14 You are given a list of words, W1, W2, W3, ... Wn and their corresponding probabilities of occurrence p1, p2, p3, ... pn. The problem is to arrange these words in a binary search tree in a way that minimizes the expected total access time. Suggest a good algorithm to implement it. Also prove the complexity of the algorithm derived by you. **[JNTU : Part B, April-11, Marks 8]**

Ans. :

Algorithm Wt(p,q,W,n)

//p is an input array [1..n]

//q is an input array [0..n]

```
// W[i,j] will be output for this function such that
W[0..n,0..n]
{
 for i←1 to n do
   W[i,i] ← q[i]; // initialize
 for m←1 to n do
 {
 for i←0 to n-m do
  {
    k ← i+m;
    W[i,k]←W[i,k-1] + p[k] + q[k];
  }
 }
write(W[0:n]);//print the values of W
}
```

- For computing C_i, r_i following algorithm is used.

```
Algorithm OBST(p,q,W,C,r)
{
 //computation of first two rows
  for i=0 to n do
     {
        C[i,i] ← 0;
        r[i,i] ← 0;
        C[i,i+1] ← q[i]+q[i+1]+p[i+1];
        r[i,i+1]←i+1;
     }
     for m ← 2 to n do
     {
        For ← 0 to n-m do
     {
        j←i+m;
        k←Min_Value(C,r,i,j); //call for algortihm
                              //Min_Value
        // minimum value of C_i is obtained for
       //deciding value of k
        C[i,j]←W[i,j] + C[i,k-1] + C[k,j];
        r[i,j]←k;
     }
     }
     write(C[0:n],r[0:n]);//print values of C_ij and r_ij
     }

Algorithm Min_Value(C,r,i,j)
{
   minimum← ∞;
// finding the range of k
   for (m←r[i,j-1] to r[i+1,j]) do
   {
      if(C[i,m-1]+C[m,j])< minimum then
      {
         minimum ← C[i,m-1]+C[m,j];
         k←m;
      }
      return k;//This k gives minimum value of C
   }
}
```

Following algorithm is used for creating the tree T_{ij}.

```
Algorithm build_tree(r,a,i,j)
{
  T ← new(node);  //allocate memory for a new node
  k ← r[i,j];
  T -> data ← a[k];
  if (j==i+1)
     return;
  T->left=build_tree(r,a,i,k-1);
  T->right=build_tree(r,a,k,j);
}
```

In order to obtain the optimal binary search tree we will follow :

1. Compute the weight using Wt function.
2. Compute C_{ij} and r_{ij} using OBST.
3. Build the tree using build_tree function.

The optimal binary search tree for a set of n keys for all the keys that are equally likely to be searched for can be constructed as follows -

1. Compute the weight using Wt function.
2. Compute C_{ij} and r_{ij} using OBST algorithm.
3. Build the tree in such a way that average depth of its nodes can be minimized.

The average number of comparisons in the tree can be denoted by the cost of the tree, to find a key for given density function.

Following formula can be used to find average number of comparisons -

$$C[i, j] = i < \overset{\min}{k} < j \{C[i, k-1] + C[k, j] + W[i, j]\}$$

$$W[i, j] = W[i, j-1] + P[j] + q[j]$$

where $i = 1$ and $j = n - 2^k$

Analysis

The computation of each C and r can be done using three nested for loops. Hence the time complexity turns out to be $O(n^3)$.

5.4 The 0/1 Knapsack Problem

Q.15 Explain 0/1 Knapsack problem.

[JNTU : Part B, May-12, Marks 7]

Ans. : Problem Description : If we are given n objects and a Knapsack or a bag in which the object i that has weight w_i is to be placed. The Knapsack has a capacity W. Then the profit that can be earned is $p_i x_i$. The objective is to obtain filling of Knapsack with maximum profit earned.

$$\text{maximized } \sum_{n}^{1} p_i x_i \text{ subject to constraint } \sum_{n}^{1} w_i x_i \le W$$

where $1 \le i \le n$ and n is total number of objects. And $x_i = 0$ or 1.

Q.16 Define merging and purging rules of 0/1 Knapsack problem.

[JNTU : Part B, Dec.-09, May-12, Marks 8]

Ans. : To solve this problem using dynamic programming method we will perform following steps.

Step 1 : The notations used are

Let,

$f_i(y_j)$ be the value of optimal solution.

Then S^i is a pair (p, w) where $p = f(y_j)$ and $w = y_j$

Initially $S^0 = \{(0, 0)\}$

We can compute S^{i+1} from S^i

These computations of S^i are basically the sequence of decisions made for obtaining the optimal solutions.

Step 2 : We can generate the sequence of decisions in order to obtain the optimum selection for solving the Knapsack problem.

Let x_n be the optimum sequence. Then there are two instances $\{x_n\}$ and $\{x_{n-1}, x_{n-2} \ldots x_1\}$. So from $\{x_{n-1}, x_{n-2} \ldots x_1\}$ we will choose the optimum sequence with respect to x_n. The selection of sequence from remaining set should be such that we should be able to fulfill the condition of filling Knapsack of capacity W with maximum profit. Otherwise $x_n \ldots x_1$ is not optimal.

This proves that 0/1 Knapsack problem is solved using principle of optimality.

Step 3 : The formulae that are used while solving 0/1 Knapsack is

Let, $f_i(y_j)$ be the value of optimal solution. Then

$$f_i(y) = \max \{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\}$$

Initially compute

$$S^0 = \{(0, 0)\}$$

$$S_1^i = \{(P, W) \mid (P - p_i, W - w_i) S^i\}$$

S_1^{i+1} can be computed by merging S^i and S_1^i

Purging rule

If S^{i+1} contains (P_j, W_j) and (P_k, W_k); these two pairs such that

$P_j \le P_k$ and $W_j \ge W_k$, then (P_j, W_j) can be eliminated. This purging rule is also called as dominance rule. In purging rule basically the dominated tuples gets purged. In short, remove the pair with less profit and more weight.

Q.17 Solve Knapsack instance M = 6, and n = 3. Let P_i and W_i are as shown below.

[JNTU : Part B, Marks 5]

i	P_i	W_i
1	1	2
2	2	3
3	5	4

Ans. : Let us build the sequence of decision S^0, S^1, S^2.

$$S^0 = \{(0, 0)\} \text{ initially}$$

$$S_1^0 = \{(1, 2)\}$$

That means while building S_1^0 we select the next i^{th} pair. For S_1^0 we have selected first (P, W) pair which is (1, 2).

Now S^1 = {Merge S^0 and S_0^1}

= {(0, 0), (1, 2)}

S_1^1 = {Select next (P, W) pair and add it with S^1 }

= { (2, 3), (2+0, 3+0), (2+1, 3+2) }

S_1^1 = {(2, 3), (3, 5)}

∵ Repetition of (2, 3) is avoided.

S^2 = {Merge candidates from S^1 and S_1^1}

= {(0, 0), (1, 2), (2, 3),(3, 5)}

∴ S_1^2 = {Select next (P, W) pair and add it with S^2}

= {(5, 4), (6, 6), (7, 7), (8, 9)}

Now S^3 = {Merge candidates from S^2 and S_1^2}

S^3 = {(0, 0), (1, 2), (2, 3), (3, 5), (5, 4), (6, 6), (7, 7), (8, 9)}

S^3 = {(0, 0), (1, 2), (2, 3), (5, 4), (6, 6), (7, 7), (8, 9)}

Note that the pair (3, 5) is purged from S^3. This is because, let us assume (P_j, W_j) = (3, 5) and (P_k, W_k) = (5, 4). Here $P_j \le P_k$ and $W_j > W_k$ is true hence we will eliminate pair (P_j, W_j) i.e. (3, 5) from S^3.

As M = 6 we will find tuple denoting weight 6. It is $(6, 6) \in S^3$. Hence we will set $x_3 = 1$. Now $(6-P_3, 6-W_3) = (6-5, 6-4) = (1, 2)$. The pair (1, 2) is present in s^2 but it is originated in S^1.

Hence set $x_1 = 1$.

Finally, we get the solution $x_1 = 1$ and $x_3 = 1$.

∴ It is **(1, 0, 1)**

Q.18 Solve the following 0/1 Knapsack problem using dynamic programming P = (11, 21, 31, 33), W = (2, 11, 22, 15), C = 40, n = 4.

[JNTU : Part B, May-13, Marks 8]

Ans. : Given that

P = (11, 21, 31, 33)

W = (2, 11, 22, 15)

C = 40, n = 4

We will compute the subsequences for the solutions

$S^0 = \{(0,0)\}$

$S_1^0 = \{(11,2)\}$

$S^1 = \{(0,0),(11,2)\}$;

$S_1^1 = \{$Select next (P, W) pair and add it with $S^1\}$

$S_1^1 = \{(21,11),(21+0,\ 11+0),(21+11,11+2)\}$

$= \{(21,11),(21,11),(32,13)\}$

$S_1^1 = \{(21,11),(32,13)\}$

$S^2 = \{$Merge candidates from S^1 and $S_1^1\}$

$= \{(0,0),(11,2),(21,11),(32,13)\}$

$S_1^2 = \{$Select next (P, W) pair and add it with $S^2\}$

$S_1^2 = \{(31,22),(42,24),(52,33),(63,35)\}$

$S^3 = \{$Merge candidates from S^2 and $S_1^2\}$

$= \{(0,0),(11,2),(21,11),(32,13),(31,22),(42,24),(52,33),(63,35)\}$

$S_1^3 = \{$Select next (P, W) pair and add it with $S^3\}$

$= \{(33,15),(44,17),(54,26),(65,28),(64,37),(75,39),(85,48),(96,50)\}$

$S^4 = \{$Merge candidates from S^3 and $S_1^3\}$

$= \{(0,0),(11,2),(21,11),(32,13),(31,22),(42,24),(52,33),(65,35),(33,15),(44,17)(54,26),(65,28),(64,37),(75,39),(85,48),(96,50)\}$

But by purging rule we must eliminate (63, 35) because

$(P_j, W_j) = (63, 35)$

$(P_k, W_k) = (33, 15)$ and $W_j \ge W_k$

Similarly, (52, 33), (42, 24) and (31, 22) can be eliminated.

∴ $S^4 = \{(0,0),(11,2),(21,11),(32,13),(44,17),(33,15),(54,26),(65,28),(64,37),(75,39),(85,48),(96,50)\}$

With C = 40, we search for the tuple containing weight 40. There is no such tuple but we find (75,

39). The weight is just close the capacity. The tuple (75, 39) $\in S^4$ (Note that it was not present in S^3). Hence set,

$x_4 = 1$

$\therefore ((75-33),(39-15)) = (42, 24)$

The tuple (42, 24) $\in S^3$ (Note that is was not present in S^2). Hence set

$x_3 = 1$

$\therefore ((42-31), (24-22)) = (11, 2)$

The tuple (11, 2) is present in S^2 but it **originally** belongs to S^1. Hence set $x_1 = 1$

Thus the solution is {1, 0, 1, 1}.

Q.19 Find optimal solution for 0/1 Knapsack problem $(w_1, w_2, w_3, w_4) = (10, 15, 6, 9)$, $(p_1, p_2, p_3, p_4) = (2,5,8,1)$ and m = 30.

[JNTU : Part B, Marks 5]

Ans. : Given that,

$$(w_1, w_2, w_3, w_4) = (10, 15, 6, 9)$$

$$(p_1, p_2, p_3, p_4) = (2, 5, 8, 1)$$

$$m = 30$$

$$n = 4$$

We will compute the subsequences for the solutions

$S^0 = \{(0, 0)\}$

$\therefore S_1^0 = \{(2, 10)\}$

$S^1 = \{(0, 0), (2, 10)\}$; $S_1^1 = \{(5, 15), (7, 25)\}$

$S^2 = \{(0, 0), (2, 10), (5, 15), (7, 25)\}$;

$S_1^2 = \{(8, 6), (10, 16), (13, 21), (15, 31)\}$

$S^3 = \{(0, 0), (2, 10), (5, 15), (7, 25), (8, 6), (10, 16), (13, 21), (15, 31)\}$

But by purging rule we must eliminate (7, 25) because $(p_j, w_j) = (7, 25)$ and $(p_k, w_k) = (8, 6)$. As $p_j \le p_k$ but $w_j > w_k$. Similarly (2, 10) and (5, 15) can also be eliminated by purging rule.

$S^3 = \{(0, 0), (8, 6), (10, 16), (13, 21), (15, 31)\}$

$S_1^3 = \{(1, 9), (9, 15), (11, 25), (14, 30), (16, 40)\}$

$\therefore S^4 = \{(0, 0), (8, 6), (1, 9), (9, 15), (11, 25), (14, 30), (16, 40)\}$

Note that we have to eliminate pairs (10, 16), (13, 21), and (15, 31).

With m = 30, we will search for tuple containing value 30. A pair (14, 30) is present in S^4. Hence set $x_4 = 1$.

$\therefore ((14 - 1), (30 - 9)) = (13, 21)$. We will search for the pair (13, 21) and it is present in S^3. Therefore set $x_3 = 1$.

$\therefore ((13 - 8), (21 - 6)) = (5, 15)$. We will search for the pair (5, 15) and it $\in S^4$. Hence set $x_2 = 1$.

$\therefore ((5 - 5), (15 - 15)) = (0, 0)$

Thus we obtain the optimal solution as $(x_1, x_2, x_3, x_4) = (0, 1, 1, 1)$

Q.20 Consider -
n = 3, $(w_1, w_2, w_3) = (2, 3, 3)$, $(p_1, p_2, p_3) = (1, 2, 4)$ and m = 6.
Find optimal solution for the given data.

[JNTU : Part B, May-09, Marks 8]

Ans. : Given that -

i	p_i	w_i
1	1	2
2	2	3
3	4	3

Initially

$S^0 = \{(0, 0)\}$ $S_1^0 = \{(1, 2)\}$

$S^1 = \{(0, 0), (1, 2)\}$

$S_1^1 = \{(2, 3), (2 + 0, 3 + 0), (2 + 1, 3 + 2)\}$

$S_1^1 = \{(2, 3), (3, 5)\}$

$S^2 = \{$Merge candidates from S^1 and $S_1^1\}$

$S^2 = \{(0, 0), (1, 2), (2, 3), (3, 5)\}$

$\therefore S_1^2 = \{(4, 3), (5, 5), (6, 6), (7, 8)\}$

S^3 = {Merge candidates from S^2 and S_1^2}

= {(0, 0), (1, 2), (2, 3), (3, 5), (4, 3), (5, 5), (6, 6), (7, 8)}

By purging rule we will eliminate (3, 5) similarly (7, 8) can also be eliminate because m = 6

$\therefore$ S^3 = {(0, 0), (1, 2), (2, 3), (4, 3), (5, 5), (6, 6)}

The capacity m = 6 and tuple (6, 6) $\in S^3$. Hence $x_3 = 1$.

$\therefore$ ((6 – 4), (6 – 3)) = (2 – 3). As pair (2, 3) $\in S^2$. Set $x_2 = 1$.

$\therefore$ ((2 – 2), (3 – 3)) = (0, 0). Hence the optimal solution is (x_1, x_2, x_3) = **(0, 1, 1)**

Q.21 Write an Algorithm to solve knapsack problem using dynamic programming.

[JNTU : Part B, Marks 5]

Ans. :

```
struct PW
{
 float p;
 float w;
};
Algorithm DKnap(int n,        //number objects for
                              // knapsack
      float p1 [ ],           // profits of objects
      float w1[ ],            // weights of objects
      float Total_wt,         // Knapsack capacity
      int x[ ])       // solution indicated by 0s and 1s
{
  struct PW pair[size]; // for storing the tuples (p,w)
  int buffer[MAXSIZE];  // keeps track of locations in
                              S(i)
  int next;             //next free spot in array
  int start;            // start index of S(i–1)
  int end;              // end index of S(i–1)
  int i;                // index of object to be added
  int j; // index for adding next object to tuples in S(i–1)
  int k;                // index for traversing S(i–1)
  int up;       // largest index in  S(i–1)that fits obj
  float new_profit      // profit of new tuple in S(i)
  float new_wt;         // weight of new tuple in S(i)
  //Initially generate the first tuple S(0)= (0,0)
  pair[1].p=pair[1].w=0.0;
  buffer[0]=1;
  //Starting at end of S(0)
  start=1; end=1;
  buffer[1]=next=2;

     // Loop for adding successive objects to
        knapsack
  for (i=1; i<=n–1; i++)
  {
    k=start;        //k loops through S(i–1)
        // Return index of largest tuple in S(i–1) that
        // does
        // not overflow  the knapsack up=Big
        // (pair,w1,start,end,i,Total_wt);
        // Add object to next tuple in S(i–1) to form
        // (new_profit,new_wt)
        // and merge S(i–1) tuples into S(i)
    for (j=start; j<=up; j++)
    {
      new_profit=pair[j].p+ p1[i];
      new_wt=pair[j].w+ w1[i];
    //Copy the tuples in S(i–1) with less weight than
    //new
    //tuple.
      while ((k<=end) AND (pair[k].w<new_wt))
    {
      pair[next].p=pair[k].p;
      pair[next].w=pair[k].w;
      next++;
      k++;
    }
      //If next tuple has same weight (new_wt) take
      // the one
      //with largest profit then update new_profit
      if((k <= end) && (pair[k].w == new_wt)) then
      {
        if(pair[k].p > new_profit) new_profit=pair[k].p;
        k++;
      }
      //if new_profit > profit of previous tuple, insert
      //new_profit and new_wt
       if(new_profit > pair[next–1].p) then
       {
         pair[next].p=new_profit;
         pair[next].w=new_wt;
```

```
            next++;
        }
      // All remaining tuples in S(i-1) have weight >
    // new_wt
      //Skip over (pruning rule) any tuples with less
    // profit
        while ((k <= end)
        {
         if((pair[k].p <= pair[next-1].p)
        &&(pair[k].w >= pair[next-1].w))
            k++;
        }
    }//end of inner for loop
     //going through all tuples in S(i-1)that fit object in
    //knapsack. Then Merge in remaining tuples from
    // S(i-1)
        while (k<=end)
        {
            pair[next].p=pair[k].p;
            pair[next].w=pair[k].w;
            next++; k++;
        }
        //Initialize S(i+1) for the next object
        start=end+1;     // set index for start of
                         // S(i+1)
        end=next-1;      // set index for end of S(i+1)
        buffer[i+1]=next;
    }  //end of outer for loop
  //Trace back the solution.
  Trace_sol (p1,w1,pair,x,Total_wt,n,buffer);
}
```

Q.22 Explain 0/1 Knapsack problem and explain the approach to solve it using dynamic programming. Explain with an example.
[JNTU : Part B, April-11, Marks 15]

Ans. : Refer Q.20 and Q.21.

5.5 All Pairs Shortest Path Problem

Q.23 What the all pair shortest path problem ?
[JNTU : Part A, Marks 2]

Ans. : When a weighted graph, represented by its weight matrix W then objective is to find the distance between every pair of nodes.

Q.24 Write an algorithm of all pairs shortest path problem. **[JNTU : Part B, May-09, April-11, Marks 6]**

Ans. :

Step 1 : We will decompose the given problem into subproblems.

Let,

$A^k_{(i,j)}$ be the length of shortest path from node i to node j such that the label for every intermediate node will be <= k.

We will compute A^k for k = 1 ..... n for all nodes.

Step 2 : For solving all pair shortest path, the principle of optimality is used. That means any subpath of shortest path is a shortest path between the end nodes. Divide the paths from i node to j node for every intermediate node, say 'k'. Then there arises two cases.

i) Path going from i to j via k.

ii) Path which is not going via k. Select only shortest path from two cases.

Step 3 : The shortest path can be computed using bottom up computation method. Following is recursion method.

Initially : $A^0 = W[i, j]$

Next computations :

$$A^k_{(i,j)} = \min \left\{ A^{k-1}_{(i,j)}, A^{k-1}_{(i,k)} + A^{k-1}_{(k,j)} \right\}$$

where $1 \le k \le n$

```
Algorithm All_Pair(W,A)
{
 for i=1 to n do
    for j=1 to n do
      A[i,j] := W[i,j]; //copy the weights as it is in
matrix A
 for k=1 to n do
 {
 for i=1 to n do
    {
    for j=1 to n do
    {
     A[i,j]=min ( A[i,j],A[i,k] + A[k,j]);
```

```
    }
  }
}
}
```

Analysis

From above algorithm

The first double for-loop takes $O(n^2)$ time.

The nested three for loops take $O(n^3)$ time.

Thus, the whole algorithm takes $O(n^3)$ time.

Q.25 Obtain all pair shortest paths for following graph. **[JNTU : Part B, Dec.-12, Marks 8]**

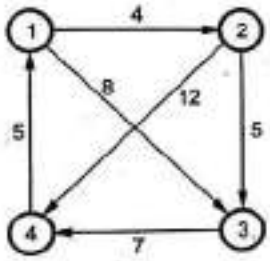


Fig. Q.25.1

Ans. :

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 4 & 8 & \infty \\ \infty & 0 & 5 & 12 \\ \infty & \infty & 0 & 7 \\ 5 & \infty & \infty & 0 \end{bmatrix}$$

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 4 & 8 & \infty \\ \infty & 0 & 5 & 12 \\ 12 & \infty & 0 & 7 \\ 5 & 9 & 13 & 0 \end{bmatrix}$$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 4 & 8 & 16 \\ 17 & 0 & 5 & 12 \\ 12 & \infty & 0 & 7 \\ 5 & 9 & 13 & 0 \end{bmatrix}$$

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 4 & 8 & 16 \\ 17 & 0 & 5 & 12 \\ 12 & 16 & 0 & 7 \\ 5 & 9 & 13 & 0 \end{bmatrix}$$

Thus shortest distances between all pairs is obtained.

Q.26 Find the shortest path between all pairs of nodes in the following graph. **[JNTU : Part B, May-09, Marks 8]**

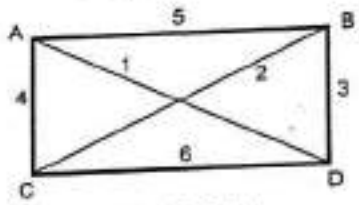


Fig. Q.26.1

Ans. : We will write adjacency matrix for given graph.

Assume A = 1, B = 2, C = 3, D = 4.

Now, let us compute A^1, A^2, A^3, A^4 by using a formula

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

i = 1, j = 1, k = 1

	A	B	C	D
A	0	5	4	1
B	5	0	2	3
C	4	2	0	6
D	1	3	6	0

Fig. Q.26.2

A [1, 1] = min (A [1, 1], A [1, 1] + A [1, 1])

= min (0, 0 + 0)

A [1, 1] = 0

i = 1, j = 2, k = 1

A [1, 2] = min (A [1, 2], A [1, 1] + A [1, 2])

= min (5, 0 + 5)

A [1, 2] = 5

i = 1, j = 3, k = 1

A [1, 3] = min (A [1, 3], A [1, 1] + A [1, 3])

= min (4, 0 + 4)

A [1, 3] = 4

i = 1, j = 4, k = 1

A [1, 4] = min (A [1, 4], A[1, 1] + A [1, 4]

= min (1, 0 + 1)

A [1, 4] = 1

i = 2, j = 1, k = 1

A [2, 1] = min (A [2, 1], A [2, 1] + A [1, 1])

= min (5, 5 + 0)

A [2, 1] = 5

i = 2, j = 2, k = 1

A [2, 2] = min (A [2, 2], A [2, 1] + A [1, 2])

= min (0, 5 + 5)

A [2, 2] = 0

i = 2, j = 3, k = 1

A [2, 3] = min (A [2, 3], A [2, 1] + A [1, 3])

= min (2, 5 + 4)

A [2, 3] = 2

i = 2, j = 4, k = 1

A [2, 4] = min (A [2, 4], A [2, 1] + A [1, 4])

= min (3, 5 + 1)

A [2, 4] = 3

i = 3, j = 1, k = 1

A [3, 1] = min (A [3, 1], A [3, 1] + A [1, 1])

= min (4, 4 + 0)

A [3, 1] = 4

i = 3, j = 2, k = 1

A [3, 2] = min (A [3, 2], A [3, 1] + A [1, 2])

= min (2, 4 + 5)

A [3, 2] = 2

i = 3, j = 3, k = 1

A [3, 3] = min (A [3, 3), A [3, 1] + A [1, 3])

= min (0, 4 + 4)

A [3, 3] = 0

i = 3, j = 4, k = 1

A [3, 4] = min (A [3, 4], A [3, 1] + A [1, 4])

= min (6, 4 + 1)

A [3, 4] = 5

i = 4, j = 1, k = 1

A [4, 1] = min (A [4, 1], A [4, 1] + A [1, 1]

= min (1, 1 + 0)

A [4, 1] = 1

i = 4, j = 2, k = 1

A [4, 2] = min (A [4, 2], A [4, 1] + A [1, 2])

= min (3, 1 + 5)

A [4, 2] = 3

i = 4, j = 3, k = 1

A [4, 3] = min (A [4, 3], A [4, 1] + A [1, 3])

= min (6, 1 + 4)

A [4, 3] = 5

i = 4, j = 4, k = 1

A [4, 4] = min (A [4, 4], A [4, 1] + A [1, 4])

= min (0, 1 + 1)

A [4, 4] = 0

The matrix is -

$A^{(1)}$ =

	A	B	C	D
A	0	5	4	1
B	5	0	2	3
C	4	2	0	5
D	1	3	5	0

Fig. Q.26.3

i = 1, j = 1, k = 2

A [1, 1] = min (A [1, 1], A [1, 2] + A [2, 1])

= min (0, 5 + 5)

A [1, 1] = 0

i = 1, j = 2, k = 2

A [1, 2] = min (A [1, 2], A [1, 2] + A [2, 2]

= min (5, 5 + 0)

A [1, 2] = 5

i = 1, j = 3, k = 2

A [1, 3] = min (A [1, 3], A [1, 2] + A [2, 3])

= min (4, 5 + 2)

A [1, 3] = 4

i = 1, j = 4, k = 2

A [1, 4] = min (A [1, 4], A [1, 2] + A [2, 4])

= min (1, 5 + 3)

A [1, 4] = 1

i = 2, j = 1, k = 2

A [2, 1] = min (A [2, 1], A [2, 2] + A [2, 1])

= min (5, 0 + 5)

A [2, 1] = 5

i = 2, j = 2, k = 2

A [2, 2] = min (A [2, 2], A [2, 2] + A [2, 2])

= min (0, 0 + 0)

A [2, 2] = 0

i = 2, j = 3, k = 2

A [2, 3] = min (A [2, 3], A [2, 2] + A [2, 3])

= min (2, 0 + 2)

A [2, 3] = 2

i = 2, j = 4, k = 2

A [2, 4] = min (A [2, 4], A [2, 2] + A [2, 4])

= min (3, 0 + 3)

A [2, 4] = 3

i = 3, j = 1, k = 2

A [3, 1] = min (A [3, 1], A [3, 2] + A [2, 1])

= min (4, 2 + 5)

A [3, 1] = 4

i = 3, j = 2, k = 2

A [3, 2] = min (A [3, 2], A [3, 2] + A [2, 2])

= min (2, 2 + 0)

A [3, 2] = 2

i = 3, j = 3, k = 2

A [3, 3] = min (A [3, 3], A [3, 2] + A [2, 3])

= min (0, 2 + 2)

A [3, 3] = 0

i = 3, j = 4, k = 2

A [3, 4] = min (A [3, 4], A [3, 2] + A [2, 4])

= min (5, 2 + 3)

A [3, 4] = 5

i = 4, j = 1, k = 2

A [4, 1] = min (A [4, 1], A [4, 2] + A [2, 1])

= min (1, 3 + 5)

A [4, 1] = 1

i = 4, j = 2, k = 2

A [4, 2] = min (A [4, 2], A [4, 2] + A [2, 2])

= min (3, 3 + 0)

A [4, 2] = 3

i = 4, j = 3, k = 2

A [4, 3] = min (A [4, 3], A [4, 2] + A [2, 3])

= min (5, 3 + 2)

A [4, 3] = 5

i = 4, j = 4, k = 2

A [4, 4] = min (A [4, 4], A [4, 2] + A [2, 4])

= min (0, 3 + 3)

A [4, 4] = 0

The matrix A^2 will be -

$A^{(2)}$ =

	A	B	C	D
A	0	5	4	1
B	5	0	2	3
C	4	2	0	5
D	1	3	5	0

Fig. Q.26.3

i = 1, j = 1, k = 3

A [1, 1] = min (A [1, 1], A [1, 3] + A [3, 1])

= min (0, 4 + 4)

A [1, 1] = 0

i = 1, j = 2, k = 3

A [1, 2] = min (A [1, 2], A [1, 3] + A [3, 2])

= min (5, 4 + 2)

A [1, 2] = 5

i = 1, j = 3, k = 3

A [1, 3] = min (A [1, 3], A [1, 3] + A [3, 3])

= min (4, 4 + 0)

A [1, 3] = 4

i = 1, j = 4, k = 3

A [1, 4] = min (A [1, 4], A [1, 3] + A [3, 4])

= min (1, 4 + 5)

A [1, 4] = 1

i = 2, j = 1, k = 3

A [2, 1] = min (A [2, 1], A [2, 3] + A [3, 1])

= min (5, 4 + 4)

A [2, 1] = 5

i = 2, j = 2, k = 3

A [2, 2] = min (A [2, 2], A [2, 3] + A [3, 2])

= min (0, 2 + 2)

A [2, 2] = 0

i = 2, j = 3, k = 3

A [2, 3] = min (A [2, 3], A [2, 3] + A [3, 3])

= min (2, 2 + 0)

A [2, 3] = 2

i = 2, j = 4, k = 3

A [2, 4] = min (A [2, 4], A [2, 3] + A [3, 4])

= min (3, 2 + 5)

A [2, 4] = 3

i = 3, j = 1, k = 3

A [3, 1] = min (A [3, 1], A [3, 3] + A [3, 1])

= min (4, 0 + 4)

A [3, 1] = 4

i = 3, j = 2, k = 3

A [3, 2] = min (A [3, 2], A [3, 3] + A [3, 2])

= min (2, 0 + 2)

A [3, 2] = 2

i = 3, j = 3, k = 3

A [3, 3] = min (A [3, 3], A [3, 3] + A [3, 3])

= min (0, 0 + 0)

A [3, 3] = 0

i = 3, j = 4, k = 3

A [3, 4] = min (A [3, 4], A [3, 3] + A [3, 4])

= min (5, 0 + 5)

A [3, 4] = 5

i = 4, j = 1, k = 2

A [4, 1] = min (A [4, 1], A [4, 3] + A [3, 1])

= min (1, 5 + 4)

A [4, 1] = 1

i = 4, j = 2, k = 3

A [4, 2] = min (A [4, 2], A [4, 3] + A [3, 2])

= min (3, 5 + 2)

A [4, 2] = 3

i = 4, j = 3, k = 3

A [4, 3] = min (A [4, 3], A [4, 3] + A [3, 3])

= min (5, 5 + 0)

A [4, 3] = 5

i = 4, j = 4, k = 3

A [4, 4] = min (A [4, 4], A [4, 3] + A [3, 4])

= min (0, 5 + 5)

A [4, 4] = 0

The matrix is -

$A^{(3)}$ =

	A	B	C	D
A	0	5	4	1
B	5	0	2	3
C	4	2	0	5
D	1	3	5	0

Fig. Q.26.4

i = 1, j = 1, k = 4

A [1, 1] = min (A [1, 1], A [1, 4] + A [4, 1])

= min (0, 1 + 1)

A [1, 1] = 0

i = 1, j = 2, k = 4

A [1, 2] = min (A [1, 2], A [1, 4] + A [4, 2])

= min (5, 1 + 3)

A [1, 2] = 4

i = 1, j = 3, k = 4

A [1, 3] = min (A [1, 3], A [1, 4] + A [4, 3])

= min (4, 1 + 5)

A [1, 3] = 4

i = 1, j = 4, k = 4

A [1, 4] = min (A [1, 4], A [1, 4] + A [4, 4])

= min (1, 1 + 0)

A [1, 4] = 1

i = 2, j = 1, k = 4

A [2, 1] = min (A [2, 1], A [2, 4] + A [4, 1])

= min (5, 3 + 1)

A [2, 1] = 4

i = 2, j = 2, k = 4

A [2, 2] = min (A [2, 2], A [2, 4] + A [4, 3])

= min (3, 3 + 0)

A [2, 2] = 3

i = 2, j = 3, k = 4

A [2, 3] = min (A [2, 3], A [2, 4] + A [4, 3])

= min (2, 3 + 5)

A [2, 3] = 2

i = 2, j = 4, k = 4

A [2, 4] = min (A [2, 4], A [2, 4] + A [4, 4])

= min (3, 3 + 0)

A [2, 4] = 3

i = 3, j = 1, k = 4

A [3, 1] = min (A [3, 1], A [3, 4] + A [4, 1])

= min (4, 5 + 1)

A [3, 1] = 4

i = 3, j = 2, k = 4

A [3, 2] = min (A [3, 2], A [3, 4] + A [4, 2])

= min (2, 5 + 3)

A [3, 2] = 2

i = 3, j = 3, k = 4

A [3, 3] = min (A [3, 3], A [3, 4] + A [4, 3])

= min (0, 5 + 5)

A [3, 3] = 0

i = 3, j = 4, k = 4

A [3, 4] = min (A [3, 4], A [3, 4] + A [4, 4])

= min (5, 5 + 0)

A [3, 4] = 5

i = 4, j = 1, k = 4

A [4, 1] = min (A [4, 1], A [4, 4] + A [4, 1])

= min (1, 0 + 1)

A [4, 1] = 1

i = 4, j = 2, k = 4

A [4, 2] = min (A [4, 2], A [4, 4] + A [4, 2])

= min (3, 0 + 3)

A [4, 2] = 3

i = 4, j = 3, k = 4

A [4, 3] = min (A [4, 3], A [4, 4] + A [4, 3])

= min (5, 0 + 5)

A [4, 3] = 5

i = 4, j = 4, k = 4

A [4, 4] = min (A [4, 4], A [4, 4] + A [4, 4])

= min (0, 0 + 0)

A [4, 4] = 0

Finally the matrix representing all pairs shortest path will be -

$A^{(4)}$ =

	A	B	C	D
A	0	4	4	1
B	4	0	2	3
C	4	2	0	5
D	1	3	5	0

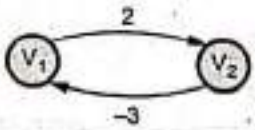
Fig. Q.26.5

This is the final solution.

Q.27 Give an example of a graph or digraph with negative weights for which Floyd's algorithm does not yield the correct result.

[JNTU : Part B, April-11, Marks 8]

Ans. : Consider following digraph with negative weights.



Now by Floyd's algorithm we will get -

$D^{(0)}$ =

	1	2
1	0	2
2	-3	0

$D^{(1)}$ =

	1	2
1	0	2
2	-3	-1

D^{2} =

	1	2
1	-1	1
2	-4	-4

The last matrix does not give the correct value of the shortest path. In fact we get $-\infty$ because repeating the cycle enough times makes the length of path arbitrarily small.

This shows that for a digraph with negative weights the Floyd's algorithm does not yield the correct result.

Q.28 Explain the problem of all pairs shortest path problem and write its algorithm using dynamic programming. Prove that it is working with a numerical example.

[JNTU : Part B, April-11, Marks 15]

Ans. : Refer Q.23, Q.24 and Q.25.

5.6 Travelling Salesperson Problem

Q.29 Discuss briefly the solution to the traveling salesperson problem using dynamic programming. Can it be solved by using divide and conquer method ?

[JNTU : Part B, May-09, Marks 6]

Ans. : Travelling salesperson problem

Problem Description

Let G be directed graph denoted by (V, E) where V denotes set of vertices and E denotes set of edges. The edges are given along with their cost C_{ij}. The cost $C_{ij} > 0$ for all i and j. If there is no edge between i and j then $C_{ij} = \infty$.

A tour for the graph should be such that all the vertices should be visited only once and cost of the tour is sum of cost of edges on the tour. The traveling salesperson problem is to find the tour of minimum cost.

Dynamic programming is used to solve this problem.

Step 1 : Let the function C(1, V–{1}) is the total length of the tour terminating at 1. The objective of TSP problem is that the cost of this tour should be minimum.

Let d[i, j] be the shortest path between two vertices i and j.

Step 2 : Let V_1, V_2...V_n be the sequence of vertices followed in optimal tour. Then (V_1, V_2,...V_n) must be a shortest path from V_1 to V_n which passes through each vertex exactly once.

Here the principle of optimality is used. The path V_i, V_{i+1},...V_j must be optimal for all paths beginning at V(i), ending at V(j), and passing through all the intermediate vertices $\{V_{(i+1)}..V_{(j-1)}\}$ once.

Step 3 : Following formula can be used to obtain the optimum cost tour.

Cost(i, S) = min {d[i, j] + Cost(j, S–{j})} where j ∈ S and i ∉ S.

Consider one example to understand solving of TSP using dynamic programming approach.

The traveling salesperson problem can not be solved using divide and conquer because it is based on principle of optimality. In divide and conquer method, the given problem is divided into subproblems and then simply the solutions of these subproblems are combined together to obtain the solution to given problem.

Q.30 For the given diagraph, obtain optimum cost Tour. **[JNTU : Part B, May-17, Marks 5]**

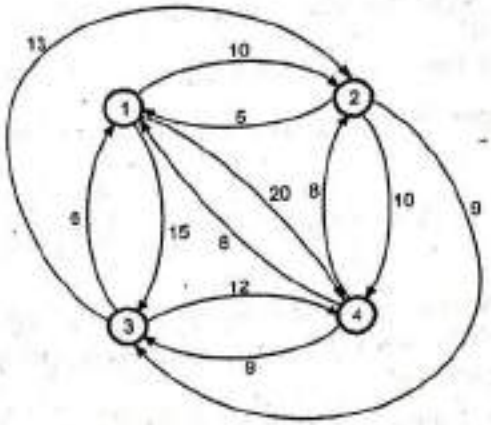


Fig. Q.30.1

Ans. : The distance matrix can be given by,

from ↓ / to →	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12

First we will select any arbitrary vertex say select 1.

Now process for intermediate sets with increasing size.

Step 1 : Let S = ϕ then,

Cost (2, ϕ 1) = d(2, 1) = 5
Cost (3, ϕ 1) = d(3, 1) = 6
Cost (4, ϕ 1) = d(4, 1) = 8

That means we have obtained dist (2, 1), dist (3, 1) and dist (4, 1).

Step 2 : Candidate (S) = 1

Apply the formula,

Cost (i, S) = min {d[i, j]+Cost (j, S–{j}}

Hence from vertex 2 to 1, vertex 3 to 1 and vertex 4 to 1 by considering intermediate path lengths we will calculate total optimum cost.

Cost (2, {3}, 1) = d (2, 3) + Cost (3, ϕ 1)
= 9 + 6
Cost (2, {3}, 1) = 15
Cost (2, {4}, 1) = d (2, 4) + Cost (4, ϕ 1)
= 10 + 8
Cost (2, {4}, 1) = 18
Cost (3, {2}, 1) = d (3, 2) + Cost (2, ϕ 1)
= 13 + 5
Cost (3, {2}, 1) = 18
Cost (3, {4}, 1) = d (3, 4) + Cost (4, ϕ 1)
= 12 + 8
Cost (3, {4}, 1) = 20
Cost (4, {2}, 1) = d (4, 2) + Cost (2, ϕ 1)
= 8 + 5
Cost (4, {2}, 1) = 13
Cost (4, {3}, 1) = d(4, 3) + Cost (3, ϕ 1)
= 9 + 6
Cost (4, {3}, 1) = 15

Step 3 : Consider candidate (S) = 2

Cost (2, {3, 4}, 1) = min {[d(2, 3) + Cost (3, {4}, 1)], [d(2, 4) + Cost (4, {3}, 1)]}
= min {[9 + 20], [10+15]}

Cost (2, {3, 4}, 1) = 25

Cost (3, {2, 4}, 1) = min {[d(3, 2) + Cost (2, {4}, 1)], [d(3, 4) + Cost (4, {2}, 1)]}
= min {[13+18], [12+13]}

Cost (3, {2, 4}, 1) = 25

Cost (4, {2, 3}, 1) = min {[d(4, 2) + Cost (2, {3}, 1)], [d(4, 3) + Cost (3, {2}, 1)]}
= min {[8 + 15], [9 + 18]}

Cost (4, {2, 3}, 1) = 23

Step 4 : Consider candidate (S) = 3. i.e. Cost (1, {2, 3, 4}) but as we have chosen vertex 1 initially the cycle should be completed i.e. starting and ending vertex should be 1.

∴ We will compute,

$$\text{Cost}(1,\{2,3,4\},1) = \min\begin{Bmatrix}[d(1,2)+\text{Cost}(2,\{3,4\},1)],\\ [d(1,3)+\text{Cost}(3,\{2,4\},1)],\\ [d(1,4)+\text{Cost}(4,\{2,3\},1)]\end{Bmatrix}$$

= min ([10 + 25], [15 + 25], [20 + 23])

= 35

Thus the optimal tour is of path length 35.

Consider step 4 now, from vertex 1 we obtain the optimum path as d(1, 2). Hence select vertex 2. Now consider step 3, in which from vertex 2 we obtain optimum cost from d(2, 4). Hence select vertex 4. Now in step 2 we get remaining vertex 3 as d(4, 3) is optimum. Hence optimal tour is 1, 2, 4, 3, 1.

5.7 Reliability Design

Q.31 Derive the mathematical formulation in reliability design. [JNTU : Part B, Dec.-12, Marks 7]

Ans. : When devices are connected together then it is a necessity that each device should work properly. The probability that device 'i' will work properly is called reliability of that device.

Let r_i be the reliability of device D_i then reliability of entire system is πr_i. It may happen that even if reliability of individual device is very good but reliability of entire system may not be good. Hence to obtain the good performance from entire system we can duplicate individual devices and can connect them in a series. To do so, we will attach switching circuits. The job of switching circuit is to determine which device in a group is working properly.

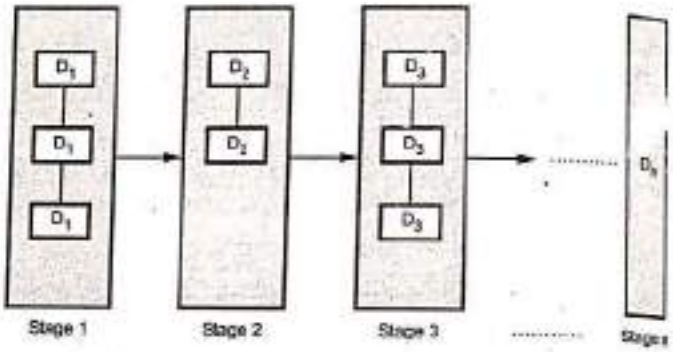


Fig. Q.31.1 Multiple devices connected together

Let m_i be the copies of device D_i then $(1 - r_i)^{mi}$ be the probability that all m_i have a malfunction. Hence the stage reliability is $1 - (1 - r_i)^{mi}$. We will denote reliability of stage i by $\phi_i(m_i)$.

Hence

$$\phi_i(m_i) = 1 - (1 - r_i)^{m_i}$$

In reliability design problem we except to get maximize reliability using device duplication. That means,

Maximize reliability $\prod_{1 \le i \le n} \phi_i(m_i)$ which is subjected to $\sum_{1 \le i \le n} C_i m_i \le C$ where C_i is the cost of each device and C is the maximum allowable cost of the system.

The upper bound u_i on the cost can be determined as :

$$u_i = \left\lfloor \left(C + C_i - \sum_{1}^{n} C_j \right) \Big/ C_i \right\rfloor$$

The dynamic programming solution S^i consist of tuples of the form (f, x) where $f = f_i(x)$. The $f_i(x)$ can be computed as

$$f_i(x) = \max_{1 \le m_i \le a_i} \{ \phi_i(m_i) f_{i-1}(x - C_i m_i) \}$$

Initially $f_o(x) = 1$

The dominance rule : (f_1x_1) dominates (f_2x_2) if $f_1 > f_2$ and $x_1 < x_2$. The dominated tuples can be discarded from S^i.

Q.32 Design a three stage system with device types D_1, D_2, D_3. The costs are ₹ 30, ₹ 15 and ₹ 20 respectively. The cost of the system is to be no more than ₹ 105. The reliability of each device type is 0.9, 0.8 and 0.5 respectively.

[JNTU : Part B, Dec.-04, June-07, Marks 10, May-12, Marks 15]

Ans. : We will first compute u_1, u_2, u_3 using following formula.

$$u_i = \left[\left(C + C_i - \sum_1^n C_j\right) \Big/ C_i\right]$$

Here $C = 105$, $C_i = C_1 = 30$,

$C_j = (C_1 + C_2 + C_3) = (30 + 15 + 20)$

$\therefore \quad u_1 = (105 + 30 - (30 + 15 + 20))/30 = 2.33$

$\therefore \quad u_1 = 2$

For computing u_2

$C = 105$, $C_i = C_2 = 20$,

$C_j = (C_1 + C_2 + C_3) = (30 + 15 + 20)$

$\therefore \quad u_3 = (105 + 15 - (30 + 15 + 20))/15$

$\therefore \quad u_2 = 3$

For computing u_3

$C = 105$, $C_i = C_3 = 20$, $C_j = (C_1 + C_2 + C_3)$

$= (30 + 15 + 20)$

$\therefore \quad u_3 = (105 + 20 - (30 + 15 + 20))/20$

$\therefore \quad u_3 = 3$

Hence $(u_1, u_2, u_3) = (2, 3, 3)$

How we will start computing subsequences

Initially $S^0 = (1, 0)$

Now we will obtain S_j^i and choose $m_{i,j}$. Let us start from S_1^1. In we will include {(reliability, cost)]

Now we will compute S^1, S^2, S^3 and so on.

Step 1 :

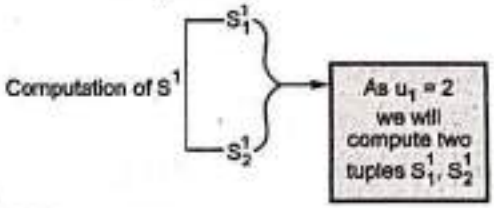
$$S_1^1 = (r_1, C_1) = \{(0.9, 30)\}$$

$$S_2^1 = S_j^i \text{ with } i = 1, j = 2$$

$$r_i = r_1 = 0.9, \quad m_i = j = 2$$

$\therefore \quad \phi_i(m_i) = 1 - (1 - r_i)^{m_i}$

$= 1 - (1 - 0.9)^2$



$\phi_i(m_i) = 0.99$

and $C_i m_i = C_1 * m_1 = 30 * 2$

$C_i m_i = 60$

$\therefore \quad S_2^1 = \{(0.99, 60)\}$

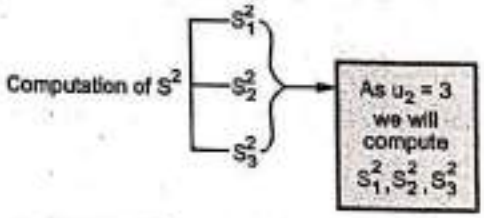
$\therefore \quad \boxed{S^1 = \{(0.9, 30), (0.99, 60)\}}$

Step 2 :

Now S_1^2

$$S_1^2 = S_j^i \text{ with } i = 2, \ j = 1$$

$$r_i = r_2 = 0.8$$



$m_i = m_2 = j = 1.$

$\therefore \quad \phi_i(m_i) = \left(1 - (1 - r_i)^{m_i}\right)$

$= 1 - (1 - r_2)^{m_2}$

$= 1 - (1 - 0.8)^1$

$\phi_i(m_i) = 0.8$

and $C_i m_i = C_2 m_2$

$= 15 * 1$

$C_i m_i = 15$

For computing the terms in S_1^2 we will use terms in S^1.

Sr. No	Using tuple (0.9, 30)	Using tuple (0.99, 60)
i)	Reliability = $0.9 * \phi_i(m_i)$ $\prod_{1 \le i \le n} \phi_i(m_i) = 0.9 * 0.8$ = 0.72	Reliability = $0.99 * \phi_i(m_i)$ = 0.99 * 0.8 = 0.792
ii)	Cost $\sum C_i m_i$ = 30 + 15 = 45 We get (reliability, cost) = (0.72, 45)	Cost = 60 + 15 = 75 We get (reliability, cost) = (0.792, 75)

$\therefore$ $S_1^2 = \{(0.72,45), (0.792,75)\}$

Now compute S_2^2

$S_2^2 = S_j^i$ with i = 2, j = 2.

$r_i = r_2 = 0.8$

$m_i = m_2 = j = 2$

$\therefore$ $\phi_i(m_i) = \left(1-(1-r_i)^{m_i}\right)$

$= 1-(1-0.8)^2$

$\phi_i(m_i) = 0.96$

and $C_i m_i = C_2 * m_2 = 15 * 2$

$\therefore$ $C_i m_i = 30$

For computing the terms in S_2^2 we will use terms in S^1.

Sr. No	Using tuple (0.9, 30)	Using tuple (0.99, 60)
i)	Reliability computation $\prod_{1 \le i \le n} \phi_i(m_i) = 0.9 * 0.96$ = 0.864	Reliability computation $\prod_{1 \le i \le n} \phi_i(m_i) = 0.99 * 0.96$ = 0.9504
ii)	Cost computation $\sum C_i m_i$ = 30 + 30 = 60 We get (reliability, cost) = (0.864, 60)	Cost computation $\sum C_i m_i$ = 60 + 30 = 90 We get (reliability, cost) = (0.9504, 90)

$\therefore$ $S_2^2 = \{(0.864, 60), (0.9504, 90)\}$

Now compute S_3^2

$S_3^2 = S_j^i$ with i = 2, j = 3

$r_i = r_2 = 0.8$

$m_i = m_2 = j = 3$

$\therefore$ $\phi_i(m_i) = 1-(1-r_i)^{m_i}$

$= 1-(1-r_2)^{m_2}$

$= 1-(1-0.8)^3$

$\phi_i(m_i) = 0.992$

and $C_i m_i = C_2 * m_2$

$= 15 * 3$

$C_i m_i = 45$

For computing the terms in S_3^2 we will use terms in S^1.

Sr. No	Using tuple (0.9, 30)	Using tuple (0.99, 60)
i)	Reliability computation $\prod_{1 \le i \le n} \phi_i(m_i) = 0.9 * 0.992$ = 0.8928	Reliability computation $\prod_{1 \le i \le n} \phi_i(m_i) = 0.99 * 0.992$ = 0.98208
ii)	Cost computation $\sum C_i m_i$ = 30 + 45 = 75 We get (0.8928, 75)	Cost computation $\sum C_i m_i$ = 60 + 45 = 105 We get (0.98208, 105)

$\therefore$ $S_3^2 = \{(0.8928, 75), (105, 0.98208)\}$

Eliminate because it exceeds 100 %

$\therefore$ $S_3^2 = \{(0.8928, 75)\}$

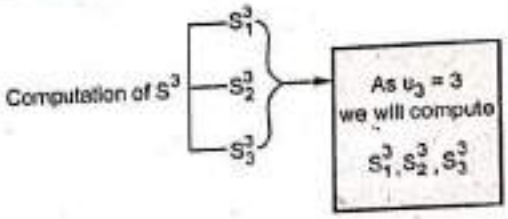
$S^2 = \{S_1^2, S_2^2, S_3^2\}$

$S^2 = \{(0.72,45), (0.792,75), (0.864,60), (0.9504,90), (0.8928,75)\}$

From this we need to eliminate (0.792, 75) and (0.9504, 90) due to dominance rule

Hence $S^2 = \{(0.72, 45), (0.864, 60), (0.8928, 75)\}$

step 3 :



Now S_1^3

$S_1^3 = S_j^i$ with $i = 3,\ j = 1$

$r_i = r_3 = 0.5$

$m_i = m_3 = j = 1.$

$\therefore \quad \phi_i(m_i) = 1-(1-r_i)^{m_i}$

$= 1-(1-r_3)^{m_3}$

$= 1-(1-0.5)^1$

$\phi_i(m_i) = 0.5$

and $C_i m_i = C_3 * m_3$

$= 20 * 1$

$C_i m_i = 20$

For computing the terms is S_1^3 we will use terms in S^2.

Sr. No	Using tuple (0.72, 45)	Using (0.864, 60)	Using (0.8928, 75)
i)	$\prod \phi_i(m_i) = 0.72 * 0.5$ $= 0.36$	i) $\prod \phi_i(m_i) = 0.864 * 0.5$ $= 0.432$	i) $\prod \phi_i(m_i) = 0.8928 * 0.5$ $= 0.4464$
ii)	$\sum C_i m_i = 45 + 20$ $= 65$ We get (0.36, 65)	$\sum C_i m_i = 60 + 20$ $= 80$ We get (0.432, 80)	$\sum C_i m_i = 75 + 20$ $= 95$ We get (0.4464, 95)

$\therefore \quad S_1^3 = \{(0.36, 65), (0.432, 80), (0.4465, 95)\}$

Now compute S_2^3

$S_2^3 = S_j^i$ with $i = 3, j = 2$

$r_i = r_3 = 0.5$

$m_3 = j = 2$

$\therefore \quad \phi_i(m_i) = 1-(1-r_i)^{m_i}$

$= 1-(1-0.5)^2$

$\phi_i(m_i) = 0.75$

and $C_i m_i = C_3 * m_3$

$= 20 * 2$

$C_i m_i = 40$

For computing the terms in S_2^3 we will use terms in S^2.

Sr. No	Using (0.72, 45)	Using (0.864, 60)	Using (0.8928, 75)
i)	$\prod \phi_i(m_i) = 0.72 * 0.75$ $= 0.54$	$\prod \phi_i(m_i) = 0.864 * 0.75$ $= 0.648$	$\prod \phi_i(m_i) = 0.8928 * 0.75$ $= 0.6696$
ii)	$\sum C_i m_i = 45 + 40$ $= 85$ We get (0.54, 85)	$\sum C_i m_i = 60 + 40$ $= 100$ We get (0.648, 100)	$\sum C_i m_i = 75 + 40$ $= 115$ We get (0.6696, 115)

$\therefore S_2^3 = \{(0.54, 85), (0.648, 100), (0.6696, 115)\}$

↑

Eliminate because it exceeds 100 %

$\therefore S_2^3 = \{(0.54, 85), (0.648, 100)\}$

Now compute S_3^3

$S_3^3 = S_i^1$ with $i = 3$, $j = 3$.

$r_i = r_3 = 0.5$

$m_i = m_3 = j = 3$

$\phi_i(m_i) = 1-(1-r_i)^{m_i}$

$= 1-(1-0.5)^{m_3}$

$= 1-(1-0.5)^3$

$\therefore \phi_i(m_i) = 0.875$

and $C_i m_i = C_3 m_3$

$= 20 * 3$

$C_i m_i = 60$

For computing the terms in S_3^3, we will use terms in S^2.

Sr. No	Using (0.72, 45)	Using (0.864, 60)	Using (0.8928, 75)
i)	$\prod \phi_i(m_i) = 0.72 * 0.875$ $= 0.63$	$\prod \phi_i(m_i) = 0.864 * 0.875$ $= 0.756$	$\prod \phi_i(m_i) = 0.8928 * 0.875$ $= 0.7812$
ii)	$\sum C_i m_i = 45 + 60$ $= 105$ As cost > 100 % discard this tuple	$\sum C_i m_i = 60 + 60$ $= 120$ As cost > 100 % discard this tuple	$\sum C_i m_i = 75 + 60$ $= 135$ As cost > 100 % discard this tuple.

Hence there will not be any term for S_3^3

$S^3 = \{S_1^3, S_2^3\}$

$\therefore S^3 = \{(0.36, 65), (0.432, 80), (0.4464, 95), (0.54, 85), (0.648, 100)\}$

For this we need to eliminate (0.4464, 95) due to dominance rule, as 95 > 85 of (0.54, 85) tuple.

$\therefore$ $S^3 = \{(0.36, 65), (0.432, 80), (0.54, 85), (0.648, 100)\}$

Observe that the reliability design will be with reliability 0.648 and cost 100.

Now tuple (0.648, 100) is obtained from S_2^3 i.e. S_j^i when i = 3, j = 2. At that time $m_i = j$ i.e. $m_3 = 2$.

We have computed (0.648, 100) from (0.864, 60). The tuple (0.864, 60) is present in S_2^2 i.e. S_i^j when i = 2, j = 2. Then $m_i = j$ i.e. $m_2 = 2$.

We have computed (0.864, 60) from (0.9, 30). The tuple (0.9, 30) is present in S_1^1 i.e. S_j^i when i = 1, j = 1. Then $m_1 = 1$.

Thus we get $m_1 = 1$, $m_2 = 2$ and $m_3 = 2$ as a solution to reliability design.

Q.33 Consider three stages of a system with $r_1 = 0.3$ $r_2 = 0.5$ $r_3 = 0.2$ and $C_1 = 30$, $C_2 = 20$ and $C_3 = 30$ where the total cost of system is C = 80 and $u_1 = 2$, $u_2 = 3$, $u_3 = 2$. Find the reliability design. [JNTU : Part B, May-13, Marks 7]

Ans. : Given that - $r_1 = 0.3$, $r_2 = 0.5$, $r_3 = 0.2$

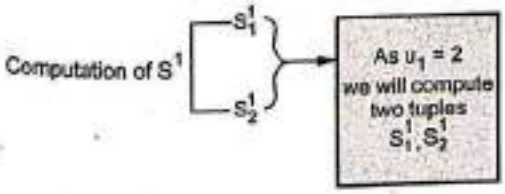
$C_1 = 30$, $C_2 = 20$ and $C_3 = 30$

Total cost 80

$u_1 = 2$, $u_2 = 3$ and $u_3 = 2$.

Now we will compute S^1, S^2 and S^3.

Step 1 :



$S_1^1 = (r_1, C_1) = \{(0.3, 30)\}$

$S_2^1 = S_j^i$ with i = 1 and j = 2

$r_i = r_1 = 0.3$

$m_i = m_1 = j = 2$

$\therefore$ $\phi_i(m_i) = 1-(1-r_i)^{mi}$

$= 1-(1-0.3)^2$

$\phi_i(m_1) = 0.51$

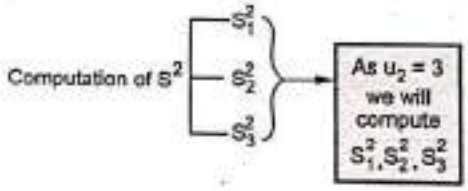
and $C_i m_i = C_1 * m_1 = 30 * 2$

$C_i m_i = 60$

$\therefore$ $S_2^1 = \{0.51, 60\}$

$\therefore$ $s^1 = \{(0.3, 30), (0.51, 60)\}$

Step 2 :



Now S_1^2

$S_1^2 = S_j^i$ with i = 2, j = 1

$r_i = r_2 = 0.5$

$m_i = m_2 = j = 1$

$\therefore$ $\phi_i(m_i) = \left(1-(1-r_i)^{mi}\right)$

$\phi_i(m_i) = 0.5$

and $C_i m_i = C_2 * m_2 = 20 * 1$

$C_i m_i = 20$

For computing the terms in S_1^2 we will use terms in S^1.

Sr. No	Using tuple (0.3, 30)	Using tuple (0.51, 60)
i)	Reliability $\prod_{1 \le i \le n} \phi_i(m_i) = 0.3 * \phi_i(m_i)$ $= 0.3 * 0.5$ $= 0.15$	Reliability $\prod_{1 \le i \le n} \phi_i(m_i) = 0.5 * \phi_i(m_i)$ $= 0.5 * 0.5$ $= 0.25$
ii)	Cost $\sum = C_i m_i = 30 + 20$ $= 50$ We get tuple (0.15, 50)	Cost $\sum C_i m_i = 60 + 20$ $= 80$ We get tuple (0.25, 80)

$\therefore \quad S_1^2 = \{(0.15, 50), (0.25, 80)\}$

Now compute S_2^2

$S_2^2 = S_j^i$ with $i = 2, j = 2$

$r_i = r_2 = 0.5$

$m_i = m_2 = j = 2.$

$\therefore \quad \phi_i(m_i) = 1-(1-r_i)^{mi}$

$= 1-(1-0.5)^2$

$\phi_i(m_i) = 0.75$

and $C_i m_i = C_2 * m_2$

$= 20 * 2$

$C_i m_i = 40$

For computing the terms in S_2^2 we will use terms in S^1.

Sr. No	Using tuple (0.3, 30)	Using tuple (0.51, 60)
i)	Reliability $\phi_i(m_i) = 0.3 * \phi_i(m_i)$ $\prod_{1 \le i \le n} = 0.3 * 0.96$ $= 0.288$	Reliability $\prod_{1 \le i \le n} \phi_i(m_i) = 0.51 * \phi_i(m_i)$ $= 0.51 * 0.96$ $= 0.4896$
ii)	Cost $\sum C_i m_i = 30 + 40$ $= 70$ We get (0.288, 70)	Cost $\sum C_i m_i = 60 + 40$ $= 100$ We get (0.4896, 100)

$\therefore \quad S_2^2 = \{(0.288, 70), (0.4896, 100)\}$

Now compute S_3^2

$S_3^2 = S_j^i$ with $i = 2$ and $j = 3$

$r_i = r_2 = 0.5$

$m_i = m_2 = j = 3$

$\therefore \quad \phi_i(m_i) = 1-(1-r_i)^{mi}$

$= 1-(1-0.5)^3$

$\phi_i(m_i) = 0.875$

and $C_i m_i = C_2 * m_2$

$= 20 * 3$

$C_i m_i = 60$

For computing the terms in S_3^2 we will use terms in S^1.

Sr. No.	Using tuple (0.3, 30)	Using tuple (0.51, 60)
i)	Reliability $\prod_{1 \le i \le n} \phi_i(m_i) = 0.3 * \phi_i(m_i)$ $= 0.3 * 0.875$ $= 0.2625$	Reliability $\prod \phi_i(m_i) = 0.51 * \phi_i(m_i)$ $= 0.51 * 0.875$ $= 0.44625$
ii)	Cost $\sum C_i m_i = 30 + 60$ $= 90$ We get (0.2625, 90)	Cost $\sum C_i m_i = 60 + 60$ $= 120$ We get (0.44625, 120)

$\therefore$ $S_3^2 = \{(0.2625, 90), (0.44625, 120)\}$

Eliminate because it exceeds 100 %.

$S_3^2 = \{(0.2625, 90)\}$

$\therefore$ $S^2 = \{S_1^2, S_2^2, S_3^2\}$

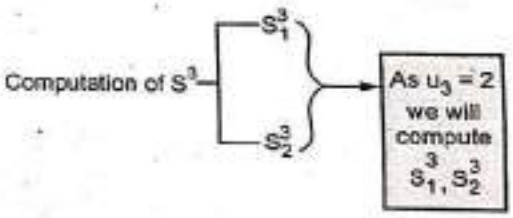
$= \{(0.15,50), (0.25, 80), (0.288, 70), (0.4896, 100), (0.2625, 90)\}$

From this we will eliminate (0.4896, 100) and (0.25, 80) due to dominance rule.

Hence

$$S^2 = \{(0.15,50), (0.288,70), (0.2625, 90)\}$$

Step 3 : Now S_1^3



$S_1^3 = S_j^i$ with $i = 3$ and $j = 1$

$r_i = r_3 = 0.2$

$m_i = m_3 = j = 1$

$\therefore$ $\phi_i(m_i) = 1-(1-r_i)^{m_i} = 1-(1-0.2)^1$

$\phi_i(m_i) = 0.2$

and $C_i m_i = C_3 * m_3 = 30 * 1$

$C_i m_i = 30$

For computing the terms in S_1^3 we will use terms in S^2.

Sr. No	Using tuple (0.15, 50)	Using (0.288, 70)	Using (0.2625, 90)
i)	$\prod \phi_i(m_i) = 0.15 * 0.2$ $= 0.030$	$\prod \phi_i(m_i) = 0.288 * 0.2$ $= 0.0576$	$\prod \phi_i(m_i) = 0.2625 - 0.2$ $= 0.00525$
ii)	Cost $\sum C_i m_i = 50 + 30$ $= 80$ We get (0.030, 80)	ii) Cost $\sum C_i m_i = 70 + 30$ $= 100$ We get (0.576, 100)	Cost $\sum C_i m_i = 90 + 30$ $= 120$ We get (0.0525, 120)

$\therefore \quad S_1^3 = \{(0.030, 80), (0.576, 100), (0.0525, 120)\}$

Now compute S_2^3

$S_2^3 = S_j^i$ with i = 3 and j = 2

$r_i = r_3 = 0.2$

$m_i = m_3 = j = 2.$

$\therefore \quad \phi_i(m_i) = 1-(1-r_i)^{m_i}$

$= 1-(1-0.2)^2$

$\phi_i(m_i) = 0.36$

and $C_i m_i = C_3 * m_3$

$= 30 * 2$

$\therefore \quad C_i m_i = 60$

For computing the terms in S_2^3 we will use terms in S^2

Sr. No	Using (0.15, 50)	Using (0.288, 70)	Using (0.2625, 90)
i)	$\prod_{1 \le i \le n} \phi_i(m_i) = 0.3 * \phi_i m_i$ $= 0.15 * 0.36$ $= 0.054$	$\prod_{1 \le i \le n} \phi_i(m_i) = 0.288 * \phi_i(m_i)$ $= 0.288 * 0.36$ $= 0.103.68$	$\prod_{1 \le i \le n} \phi_i(m_i) = 0.2625\ \phi_i(m_i)$ $= 0.2625 * 0.36$ $= 0.0945$
ii)	$\sum c_i m_i = 50 + 60$ $= 110$ We get (0.054, 110)	$\sum C_i m_i = 70 + 60$ $= 130$ We get (0.10368, 130)	$\sum C_i m_i = 90 + 60$ $= 150$ We get (0.0945, 150)

$\therefore \quad S_2^3 = \{(0.054, 110), (0.10368, 130), (0.945, 150)\}$

Eliminate these terms as cost > 100 %.

$S^3 = \{S_1^3\}$

$\therefore \quad S^3 = \{(0.030, 80), (0.576, 100), (0.0525, 120)\}$

From this we need to eliminate (0.525, 120) and (0.576, 100) as, maximum allowed cost is 80.

$\therefore \quad \boxed{S^3 = \{(0.030, 80)\}}$

From S^3 tuple is **(0.030, 80)**. This tuple is actually obtained from S_1^3.

$\therefore$ S_1^3 with i = 3, j = 1, $m_3 = j = 1$.

Therefore set $\boxed{m_3 = 1}$

We have computed (0.030, 80) from (0.15, 50)

But $(0.15, 50) \in S_1^2$ originally.

$\therefore S_1^2$ with $i = 2, j = 1, m_2 = j - 1$.

$\therefore$ Set $m_2 = 1$

We have compute (0.15, 50) from (0.3, 30)

But $(0.3, 30) \in S_1^1$ with $i = 1, j = 1, m_1 = j = 1$.

$\therefore$ Set $m_1 = 1$

Thus we get $m_1 = 1$, $m_2 = 1$ and $m_3 = 1$ as a solution to reliability design.

Multiple Choice Questions

Q.1 Which method can be used when the solution to a problem can not be viewed as the result of a sequence of decisions ?

[JNTU : Nov.-09]

a Greedy method
b Divide and conquer
c Dynamic Programming
d Backtracking

Q.2 The _____ states that whenever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence.

[JNTU : Nov.-09]

a principle of locality
b principle of optimality
c principle of backtracking
d none of the above

Q.3 Order profit-weight ratios of all objects _____.

a $p_i/w_i >= (p_i+1)/(w_i+1)$ for $1<=i<=n-1$
b $i/w_i<=(p_i+1)/(w_i+1)$ for $1 <=i<=n-1$
c $p_i/w_i>=(p_i+1)/(w_i+1)$ for $1 >=i<=n-1$
d $p_i/w_i<=(p_i+1)/(w_i+1)$ for $i>=n$

[JNTU : August-16]

Q.4 The goal is obtained at _____ in 0/1 Knapsack using dynamic programming.

a table[0,0] b table[0,n]
c table[n,w] d table[n,n]

Fill in the Blanks

Q.1 The principle of optimality is used by _____.

Q.2 _____ and _____ are two variations of Knapsack problem.

Q.3 The purging rule is used to solve _____ problem.

Q.4 Reliability design application makes use of _____ algorithmic strategy.

Answer Key Multiple Choice Questions

1.	c	2.	b
3.	b	4.	c

Answer Key Fill in the Blanks :

1.	dynamic programming	2.	Fractional Knapsack, 0/1 Knapsack
3.	0/1 Knapsack problem	4.	dynamic programming

END...

UNIT - II

6 Backtracking

6.1 General Method

Important Points to Remember

- 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.
- Backtracking algorithm determines the solution by systematically searching the solution space (i.e. set of all feasible solutions) for the given problem.
- Backtracking is a depth first search with some bounding function.
- Typical applications of backtracking are - Eight Queen's Problem, Sum of Subset Problem, Graph Coloring Problem

Q.1 What is backtracking ? What is the advantage of this method ? ☞ [JNTU : Part A, Marks 2]

Ans. : In the backtracking method

1. The desired solution is expressible as an n tuple $(x_1, x_2,...x_n)$ where x_i is chosen from some finite set S_i.
2. The solution maximizes or minimizes or satisfies a criterion function C $(x_1, x_2,...x_n)$.

The major advantage of backtracking 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.

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

Q.2 Enlist the characteristics of backtracking strategy. ☞ [JNTU : Part A, Marks 2]

Ans. : Following are the characteristics of backtracking strategy -

1. In backtracking technique an organized and exhaustive search that avoids searching all possibilities is made.
2. The solution space is organized into state space tree.
3. It uses depth first search method.
4. The search tree is pruned using bonding functions.

Q.3 What are the constraints that must be satisfied while solving any problem using backtracking ? Explain briefly. ☞ [JNTU : Part B, Marks 2]

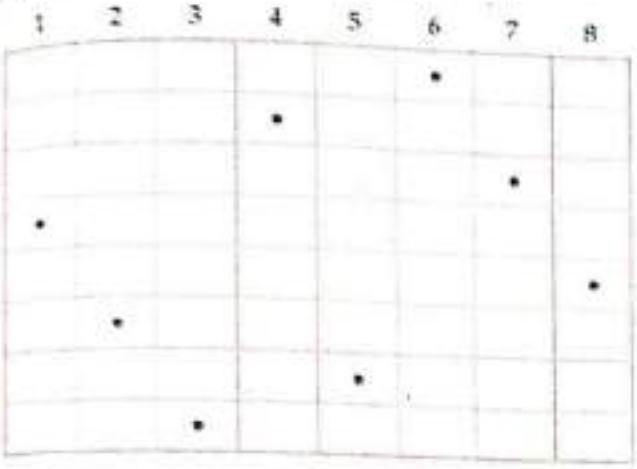
Ans. : Definition : Explicit constraints are the rules that restrict each element xi has to be chosen from given set only. Explicit constraints depends on particular instance I of the problem. All the tuples from solution set must satisfy the explicit constraints.

Definition : Implicit constraints are the rules that decide which tuples in the solution space of I satisfy the criterion function. Thus the implicit constraints represent by which xi in the solution set must be related with each other.

For example - 8-Queen's problem - The 8-queen's problem can be stated as follows. Consider a chessboard of order 8×8. The problem is to place 8 queens on this board such that no two queens can attack each other. That means no two queens can be placed on the same row, column or diagonal.

The solution to 8-queens problem can be obtained using backtracking method.

The solution can be given as below -



This 8 Queen's problem is solved by applying implicit and explicit constraints.

The **explicit constraints** show that the solution space S_i must be {1, 2, 3, 4, 5, 6, 7, 8} with $1 \le i \le 8$. Hence solution space consists of 8^8 8-tuples.

The **implicit constraint** will be - 1) No two x_i will be same. That means all the queens must lie in different columns.

2) No two queens can be on the same row, column or diagonal.

Hence the above solution can be represented as an 8-tuple {4, 6, 8, 2, 7, 1, 3, 5}.

Q.4 Define following terms : State space, explicit constraints, implicit constraints, problem state, solution states, answer states, live node, E-node, dead node, bounding functions.

☞ **[JNTU : Part B, Marks 5]**

Ans. :

State space : All paths from root to other nodes define the state space of the problem. The tree organization for 4-queen's problems is shown by given Fig. Q.4.1. (Refer Fig. Q.4.1 on next page)

Explicit constraints : Explicit constraints are rules, which restrict, each vector element to be chosen from given set.

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

Problem states : Each node in the state space tree is called problem state.

Solution states : The solution states are those problem states s for which the path from root to s defines a tuple in the solution space. In some trees the leaves define solution states.

Answer states : These are the leaf nodes which correspond to an element in the set of solutions. These are the states which satisfy the implicit constraints.

Live node : A node which is generated and whose children have not yet been generated is called live node.

E-node : The live node whose children are currently being expanded is called E-node.

Dead node : A dead node is a generated node which is not to be expanded further or all of whose children have been generated.

Bounding functions : Bounding functions will be used to kill live nodes without generating all their children. A care should be taken while doing so, because atleast one answer node should be produced or even all the answer nodes be generated if problem needs to find all possible solutions.

Q.5 Define state space tree.

☞ **[JNTU : Part A, Nov.-16, Marks 2]**

Ans. : Refer Q.4.

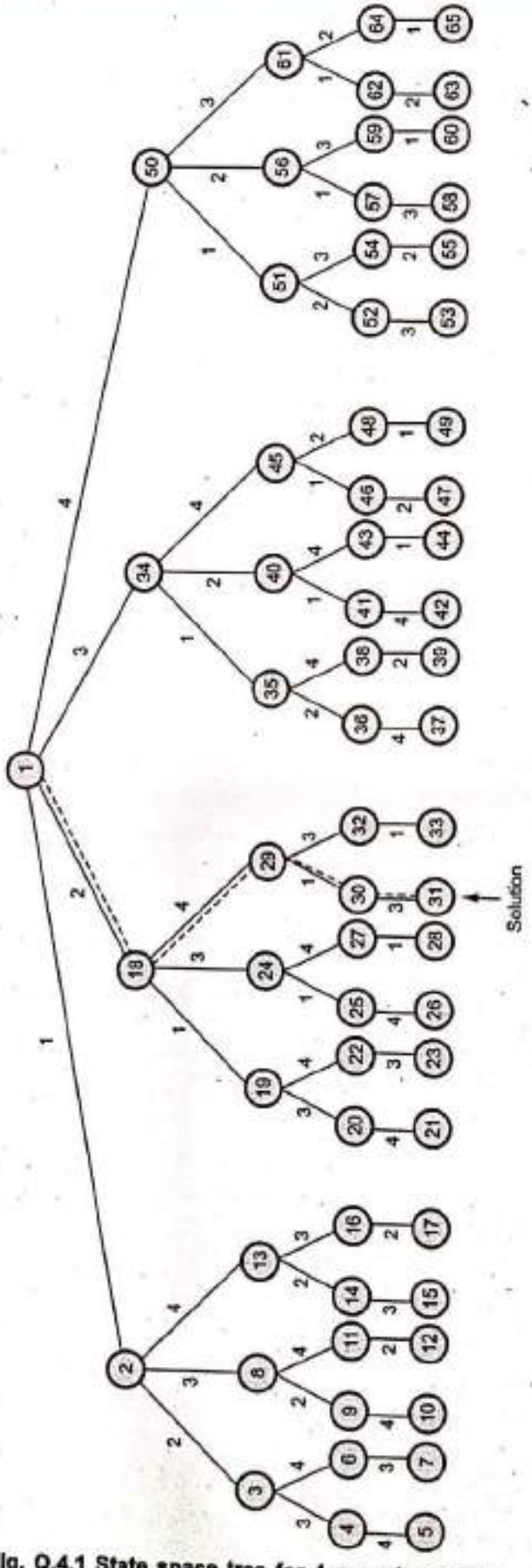


Fig. Q.4.1 State space tree for 4-queen's problem

Q.6 What are different applications of backtracking ? [JNTU : Part A, Marks 2]

Ans. : Various applications that are based on backtracking method are -

1. 8 Queen's problem : This is a problem based on chess games. By this problem, it is stated that arrange the queens on chessboard in such a way that no two queens can attack each other.
2. Sub of subset problem.
3. Graph coloring problem.
4. Finding Hamiltonian cycle.
5. Knapsack problem.

Q.7 Explain the recursive backtracking algorithm. [JNTU : Part B, May-09, Marks 8]

Ans. : Algorithm Backtrack()

```
//This is recursive backtracking algorithm
//a[k] is a solution vector. On entering (k–1) remaining next
//values can be computed.
//T(a1,a2,...ak) be the set of all values for a(i+1), such that
//a1,a2,...ai+1) is a path to problem state.
// Bi+1 is a bounding function such that if
Bi+1(a1,a2,...ai+1) is false
//for a path (a1,a2,...ai+1) from root node to a problem state then
//path can not be extended to reach an answer node
{
for ( each ak that belongs to T((a1,a2,...ak–1) do
{
if(Bk(a1,a2,...ak) = true) then // feasible sequence
{
   if ((a1,a2,...ak) is a path to answer node then
     print(a[1],a[2],...a[k]);
   if (k<n) then
     Backtrack(k+1); //find the next set.
 }
}
}
```

Q.8 What is backtracking ? Write general iterative algorithm for backtracking. [JNTU : Part B, Marks 5]

Ans. : Backtracking : Refer Q.1.

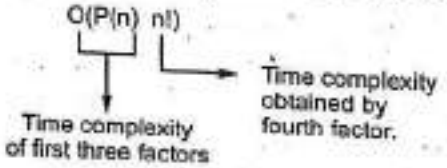
Iterative algorithm for backtracking :

```
Algorithm Non_Rec_Back(n)
// This is a non recursive version of backtracking
//a[1,...,n] is a solution vector and each a1,a2,...ak will
be used to print solution.
{
 k:=1;
 while(k ≠ 0) do
 {
    if(any a[k] that belongs to
T(a[1],a[2],...a[k–1])remains untried)
    AND (Bk (a[1],a[2]...,a[k] is true) then
   {
    if((a[1],a[2]...a[k]) is a path to answer node) then
     write(a[1],a[2],..a[k]); // solution printed
    //consider next element of the set
    k:=k+1;
    else
      k : =k–1;  //backtrack to most recent value
   }
 }
}
```

Q.9 Give the efficiency analysis of backtracking algorithm. [JNTU : Part B, Marks 5]

Ans. : The efficiency of both the backtracking algorithm depends upon four factors -

1) The time required to generate a[k].
2) The number of a[k] elements which satisfy the explicit constraints.
3) The time required by bounding functions B_k to generate a feasible sequence.
4) The number of elements a[k] that satisfy the bounding functions B_k for all the values of k.



The first three factors are **independent** on the given **problem instance**. And the time complexity of these three factors is polynomial time. But the fourth factor varies according to the size of problem instance. That means, if there are n! nodes which satisfy the bounding function then the worst case time complexity of backtracking.

6.2 The n-queen Problem

Q.10 State n queeen's problem.

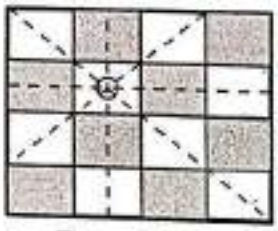
[JNTU : Part A, Marks 2]

Ans. : The n-queen's problem can be stated as follows.

Consider a n × n chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

For example

Consider 4 × 4 board



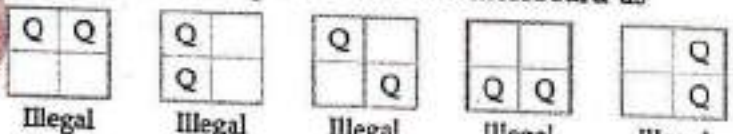
The next queen - if is placed on the paths marked by dotted lines then they can attack each other

Q.11 Explain why 2 Queen's problem is unsolvable ?

[JNTU : Part A, Marks 2]

Ans. : In n-queen's problem, we have to place the queens in such a way that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

- 2-Queen's problem is not solvable - Because 2-queens can be placed on 2 × 2 chessboard as



Q.12 Explain the applications of backtracking.

[JNTU : Part B, May-09, Marks 6]

OR

Explain in detail how the technique of backtracking can be applied to solve the 8-queens problem.

[JNTU : Part B, Dec.-12, Marks 7]

OR

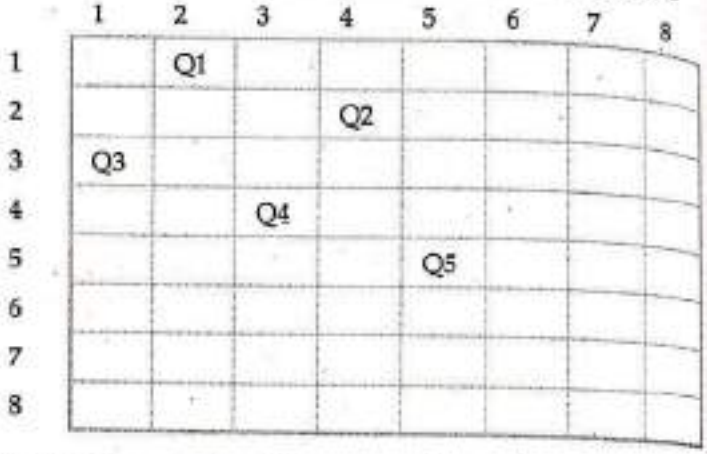
State and explain the n-Queen problem using backtracking.

[JNTU : Part B, May-13, Marks 7]

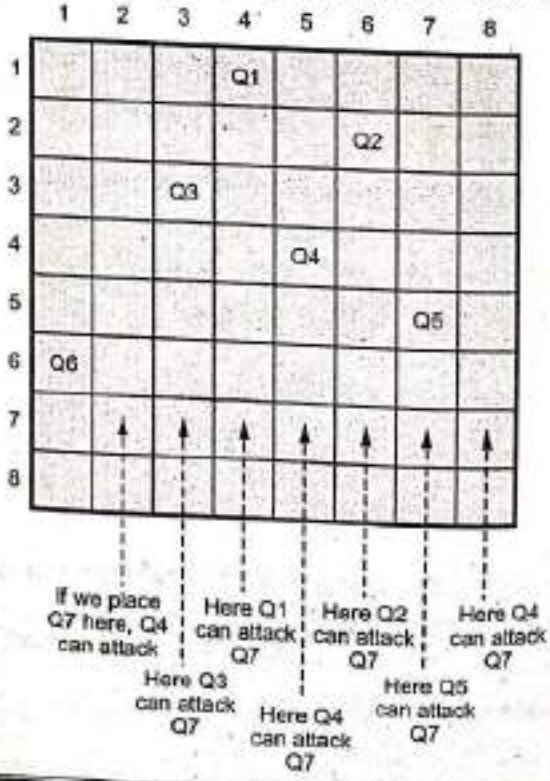
Ans. : The n-queen's problem can be stated as follows.

Consider a n × n chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

We will start placing the queens on the chessboard.



Thus we have placed 5 queens in such a way that no two queens can attack each other. Now if we want to place Q6 at location (6, 6) then queen Q5 can attack, if Q6 is placed at (6, 7) then Q1 can attack, if Q6 is placed at (6, 8) then Q2 can attack it. Similarly at (6, 5) the Q5 will attack it. At (6, 4) the Q2 will attack, at (6, 3) the Q4 will attack, at (6, 2) the Q1 will attack and at (6, 1) Q3 will attack the queen Q6. This shows that we need to backtrack and change the previously placed queens positions. It could then be -



Hence we have to backtrack to adjust already placed queens.

	1	2	3	4	5	6	7	8
1	Q1							
2								Q2
3						Q3		
4			Q4					
5							Q5	
6		Q6						
7				Q7				
8								

But again Q8 cannot be placed at any empty location safely. Hence we need to backtrack. Finally the successful placement of all the eight queens can be shown by following figure.

	1	2	3	4	5	6	7	8
1			Q1					
2						Q2		
3		Q3						
4							Q4	
5	Q5							
6				Q6				
7								Q7
8					Q8			

Q.13 Describe the 4-queens problem using backtracking.

[JNTU : Part B, May-09, May-12,13, Marks 8, Nov 16, Marks 5]

OR

Draw and explain the portion of the tree for 4-queens problem that is generated during backtracking. **[JNTU : Part B, May-09, Marks 10]**

Ans. : Let us take 4-queens and 4×4 chessboard.

- Now we start with empty chessboard.
- Place queen 1 in the first possible position of its row, i.e. on 1st row and 1st column.

Q			

- Then place queen 2 after trying unsuccessful place - 1(1, 2), (2, 1), (2, 2) at (2, 3) i.e. 2nd row and 3rd column.

Q			
		Q	

- This is the dead end because a 3rd queen cannot be placed in next column, as there is no acceptable position for queen 3. Hence algorithm **backtracks** and places the 2nd queen at (2, 4) position.

Q			
			Q

- The place 3rd queen at (3, 2) but it is again another dead end as next queen (4th queen) cannot be placed at permissible position.

Q			
			Q
	Q		

- Hence we need to backtrack all the way upto queen 1 and move it to (1, 2).
- Place queen 1 at (1, 2), queen 2 at (2, 4), queen 3 at (3, 1) and queen 4 at (4, 3).

The state space tree of 4-queen's problem is shown in Fig. Q.13.1.

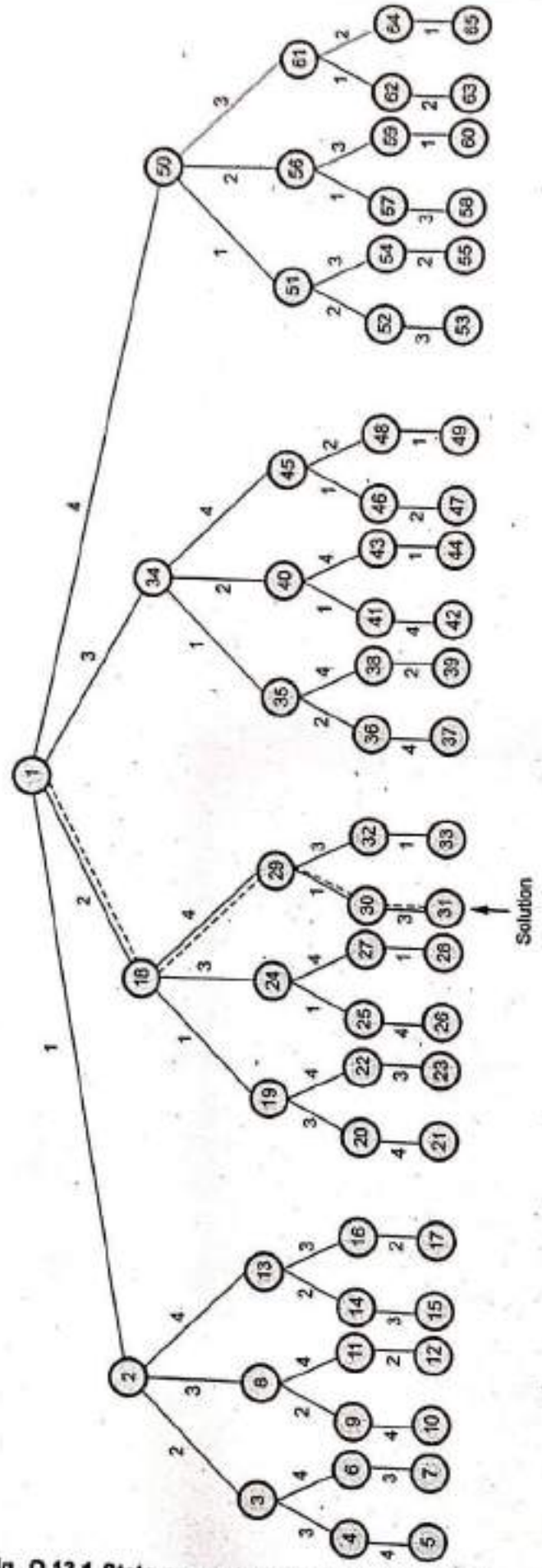
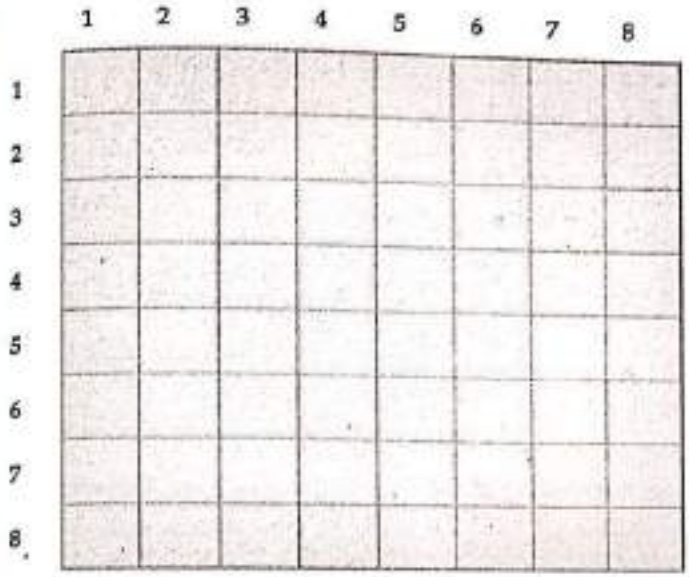


Fig. Q.13.1 State space tree for 4-queen's problem

Now we will consider how to place 8-Queen's on the chessboard.

Initially the chessboard is empty.



```
Algorithm Queen(n)
//Problem description : This algorithm is for
implementing n
//queen's problem
//Input : total number of queen's n.
    for column ←1 to n do
    {
      if(place(row,column))then
      {
       board[row]column//no conflict so place queen
      if(row=n)then//dead end
     print_board(n)
          //printing the board configuration
      else//try next queen with next position
          Queen(row+1,n)
      }
    }
```

This function checks if two queens are on the same diagonal or not.

Row by row each queen is placed by satisfying constraints.

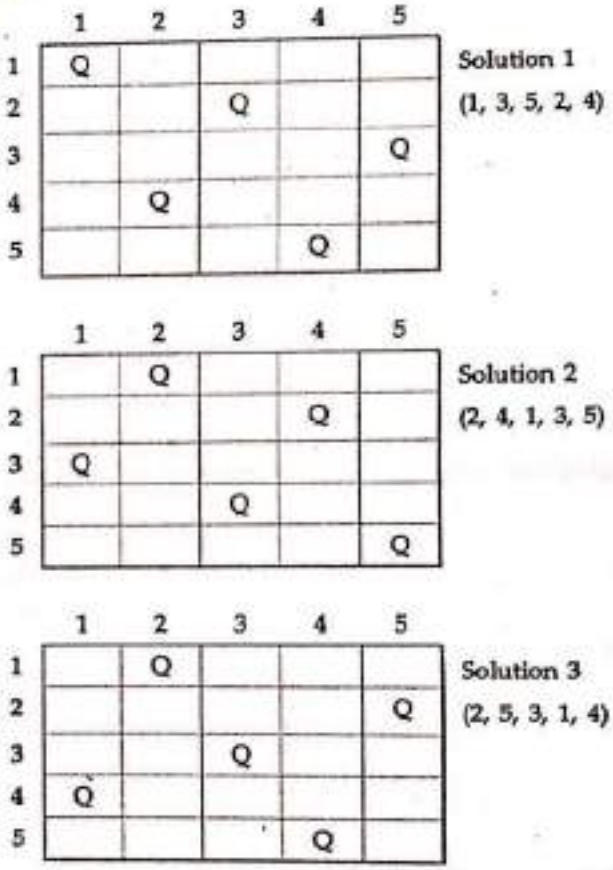
```
Algorithm place(row,column)
//Problem Description : This algorithm is for placing the
//queen at appropriate position
//Input : row and column of the chessboard
//output : returns 0 for the conflicting row and column
//position and 1 for no conflict.
    for i ← 1 to row-1 do
    { //checking for column and diagonal conflicts
     if(board[i] = column)then
       return 0
     else if(abs(board[i]- column) = abs(i - row))then
       return 0
    }
    //no conflicts hence Queen can be placed
   return 1
```

Same column by 2 queen's

This formula gives that 2 queens are on same diagonal

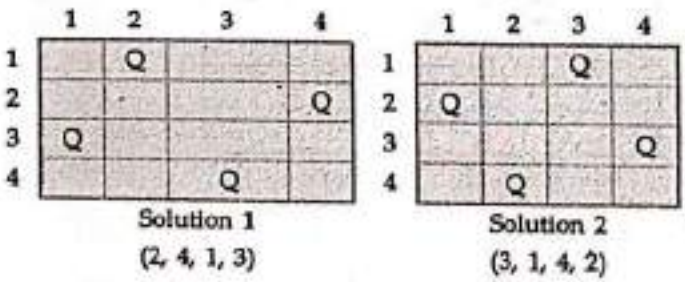
Q.14 Generate at least 3 solutions for 5-queen's problem. [JNTU : Part B, Marks 5]

Ans. : The solutions are as given below.



Q.15 Obtain any two solutions to 4-queen's problem. Establish the relationship between them. [JNTU : Part B, Marks 5]

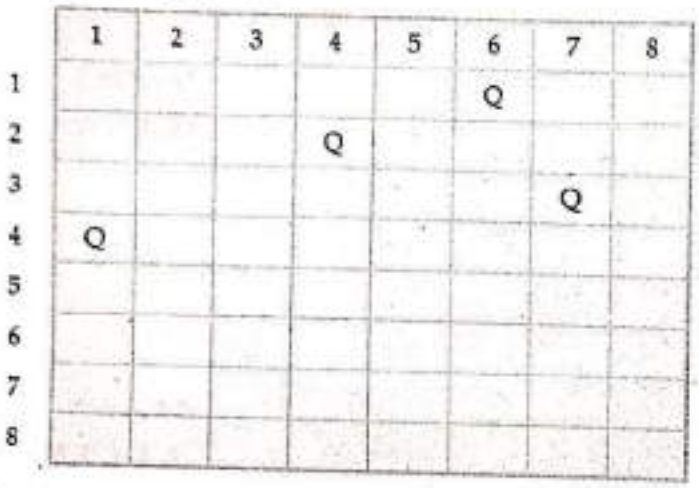
Ans. : The solutions to 4-queen's problem are as given below.



If these two solutions are observed then we can say that second solution can be simply obtained by reversing the first solution.

Q.16 Solve 8-queen's problem for a feasible sequence (6, 4, 7, 1). [JNTU : Part B, Marks 5]

Ans. : As the feasible sequence is given, we will place the queens accordingly and then try out the other remaining places.



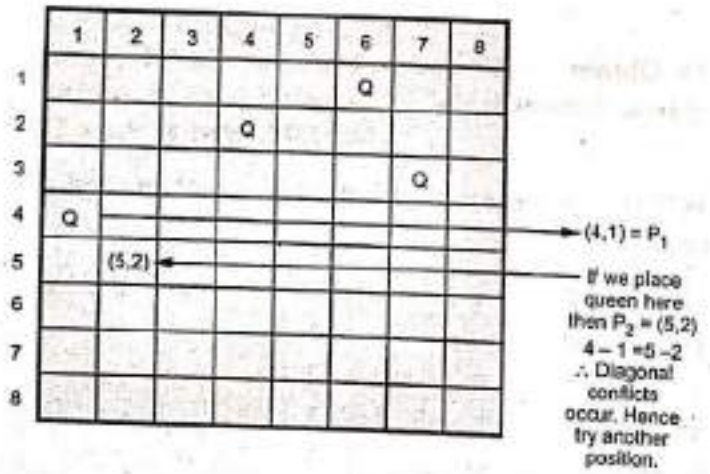
The diagonal conflicts can be checked by following formula -

Let, $P_1 = (i, j)$ and $P_2 = (k, l)$ are two positions. Then P_1 and P_2 are the positions that are on the same diagonal, if

$$i + j = k + l \quad \text{or}$$

$$i - j = k - l$$

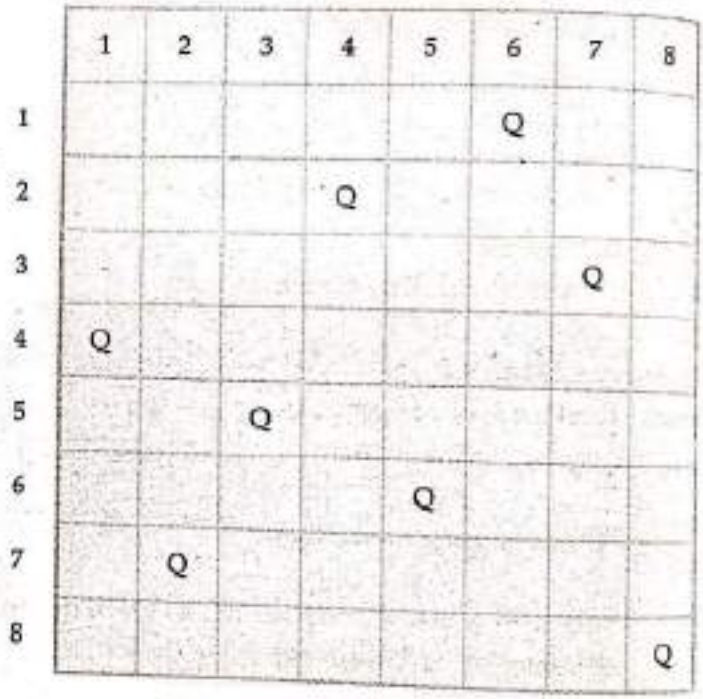
Now if next queen is placed on (5, 2) then



It can be summarized below.

Queen Positions								Action
1	2	3	4	5	6	7	8	
6	4	7	1					Start
6	4	7	1	2				As 4 – 1 = 5 – 2 conflict occurs.
6	4	7	1	3				5 – 3 ≠ 4 – 1 or 5 + 3 ≠ 4 + 1 ∴ Feasible
6	4	7	1	3	2			As 5 + 3 = 6 + 2. It is not feasible.
6	4	7	1	3	5			Feasible
6	4	7	1	3	5	2		Feasible
6	4	7	1	3	5	2	8	List ends and we have got feasible sequence.

The 8-queens on 8 × 8 board with this sequence is -



8-queens with feasible solution (6, 4, 7, 1, 3, 5, 2, 8)

6.3 Sum of Subsets Problem

Q.17 State and explain sum of subsets problem with suitable example. [JNTU : Part B, Marks 5]

Ans. : Problem Statement :

Let, $S = [S_1, \ldots S_n]$ be a set of n positive integers, then we have to find a subset whose sum is equal to given positive integer d.

It is always convenient to sort the set's elements in ascending order. That is,

$$S_1 \leq S_2 \leq \ldots \leq S_n$$

Example - Refer Q.19.

Q.18 Write a procedure to solve sum of subset problem. [JNTU : Part A, Marks 3]

Ans. :

Step 1 : Start with an empty set.

Step 2 : Add to the subset, the next element from the list.

Step 3 : If the subset is having sum d then stop with that subset as solution.

Step 4 : If the subset is not feasible or if we have reached the end of the set then backtrack through the subset until we find the most suitable value.

Step 5 : If the subset is feasible then repeat step 2.

Step 6 : If we have visited all the elements without finding a suitable subset and if backtracking is possible then stop without solution. This problem can be well understood with some example.

Q.19 Consider a set S = {5, 10, 12, 13, 15, 18} and d = 30. Solve it for obtaining sum of subset. [JNTU : Part B, Marks 5]

Ans. :

Initially subset = { }	Sum = 0	-
5	5	Then add next element.
5, 10	15 ∵ 15 < 30	Add next element.
5, 10, 12	27 ∵ 27 < 30	Add next element.
5, 10, 12, 13	40	Sum exceeds d = 30 hence backtrack.
5, 10, 12, 15	42	Sum exceeds d = 30 ∴ Backtrack
5, 10, 12, 18	45	Sum exceeds d ∴ Not feasible. Hence backtrack.
5, 10		
5, 10, 13	28	
5, 10, 13, 15	33	Not feasible ∴ Backtrack.
5, 10		
5, 10, 15	30	Solution obtained as sum = 30 = d

∴ The state space tree can be drawn as follows.

{5, 10, 12, 13, 15, 18}

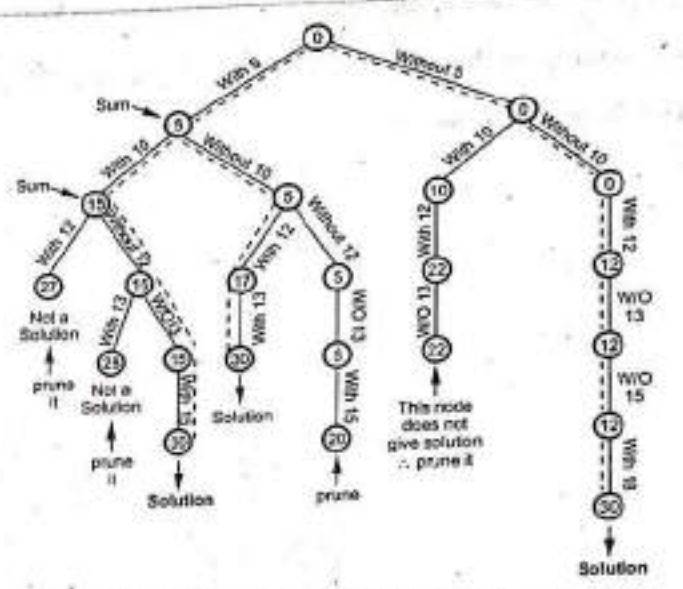


Fig. Q.19.1 State space tree for sum of subset

Q.20 Let w = {7; 4; 10; 23; 35; 20; 32} and m = 55. Find all possible subsets of w that sum to m. Draw the portion of the state space tree that is generated. [JNTU : Part B, May-12, Marks 15]

Ans. :

Initially subset = { }	Sum = 0	-
7	7	Add next element
7, 4	11 < 55	Add next element
7, 4, 10	21 < 55	Add next element
7, 4, 10, 23	44 < 55	Add next element
7, 4, 10, 23, 35	79 > 55	Sum exceeds ∴ No feasible Hence backtrack
7, 4, 10, 23, 20	64 > 55	Sum exceeds ∴ backtrack
7, 4, 10, 23, 32	76 > 55	Sum exceeds ∴ backtrack
7, 4, 10, 32	53 < 55	Add next element
7, 4, 10, 32, 20	73 > 55	Backtrack
7, 4, 10, 35	56 > 55	Backtrack
7, 10	17 < 55	Add next element
7, 10, 23	40 < 55	Add next element
7, 10, 23, 35	75 > 55	Backtrack

Continuing in this manner, we get {35, 20} {23, 32} as the subsets having d = 55.

The state space tree is -

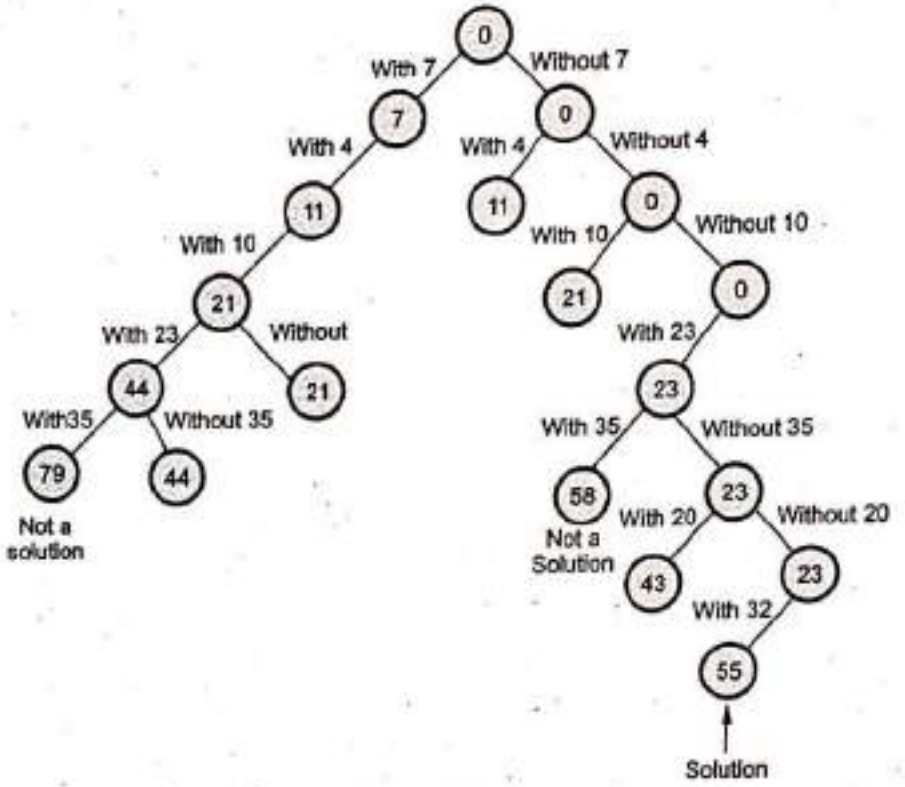


Fig. Q.20.1

Q.21 Write a backtracking algorithm for the sum of subsets problem using the state space tree corresponding to the variable tuple size formulation. ☞ [JNTU : Part B, April-11, Dec.-11, Marks 15]

Ans. :

Algorithm Sum_Subset(sum, index, Remaining_sum)

//sum is a variable that stores the sum of all the

//selected elements

//index denotes the index of chosen element from the given

//Remaining_sum is initially sum of all the elements.

//selection of some element from the set subtracts the

//chosen value

//from Remaining_sum each time.

//w[1...n] represents the set containing n elements

//a[j] represents the subset where $1 \le j \le$ index

//sum = $\sum_{j=1}^{index-1} w[j]*a[j]$

//Remaining_sum = $\sum_{j=index}^{n} = w[j]$.

// w[j] is sorted in non-decreasing order

// For a feasible sequence assume that $w[1] \le d$ and

$\sum_{i=1}^{n} w[i] \ge d$

// Generate left child until sum + w[index] is $\le d$

a[index]←1

```
if(sum + w[index] = d) then
                                    ← The subset is printed
  write(a[1...index]) //subset is found
else if (sum + w[index] + w[index+1] d) then
  Sum_Subset((sum+w[index]), (index+1),(Remaining_sum – w[index]));
        // Generate right child
if(sum+Remaining_sum – w[index] ≥ d) AND
        (sum+w[index+1]≤ d)) then
{
      a[index] ← 0
  Sum_Subset(sum, (index+1), (Remaining_sum – w[index]));
}
```

Search the next element which can make sum ≤ d

6.4 Graph Coloring

Important Points to Remember

- Graph coloring is a problem of coloring each vertex in graph in such a way that no two adjacent vertices have same color and yet m-colors are used.
- This problem is also called **m-coloring problem.**
- If the degree of given graph is d then we can color it with d + 1 colors.
- The least number of colors needed to color the graph is called its **chromatic number.**

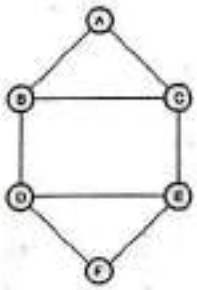
Q.22 What is m-coloring problem ? **[JNTU : Part A, Marks 2]**

Ans. : Graph coloring is a problem of coloring each vertex in graph in such a way that no two adjacent vertices have same color and yet m-colors are used. This problem is also called m-coloring problem.

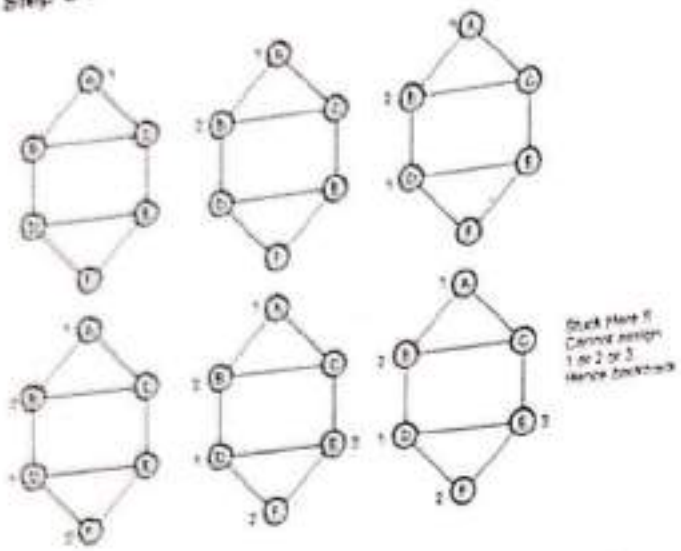
Q.23 Explain graph coloring problem in detail. **[JNTU : Part B, Marks 5]**

Ans. :

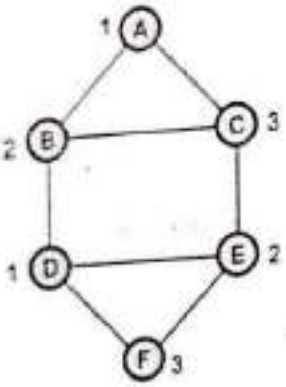
Step 1 : A Graph G consists of vertices from A to F. There are three colors used Red, Green and Blue. We will number them out. That means 1 indicates Red, 2 indicates Green and 3 indicates Blue color.



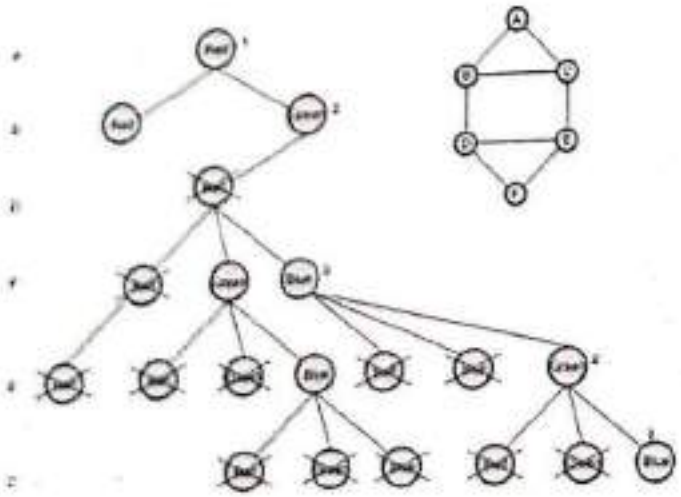
Step 2 :



Step 3 :



Thus the graph coloring problem is solved. The state-space tree can be drawn for better understanding of graph coloring technique using backtracking approach -



Here we have assumed, color index Red = 1, Green = 2 and Blue = 3.

Q.24 What is graph coloring ? Design an algorithm which finds m-coloring of a graph.

GTU [JNTU : Part B, May-09, Marks 8]

Ans. : Graph Coloring Problem - Refer Q.22.

Algorithm - The algorithm for graph coloring uses an important function Gen_Col_Value() for generating color index. The algorithm is -

Algorithm Gr_color(k)

```
//The graph G[1 : n,1 : n] is given by adjacency matrix
//Each vertex is picked up one by one for coloring
//x[k] indicates the legal color assigned to the vertex
{
 repeat
 {
   // produces possible colors assigned
   Gen_Col_Value(k);
 if(x[k] = 0) then
   return; //Assignment of new color is not possible
 if(k=n) then // all vertices of the graph are colored
   write(x[1:n]); //print color index
 else
  Gr_color(k+1) // choose next vertex
 }until(false);
}
```

Takes O(m) time

The algorithm used assigning the color is given as below

Algorithm Gen_Col_Value(k)

```
 //x[k] indicates the legal color assigned to the vertex
 // If no color exists then x[k] = 0
 {
// repeatedly find different colors
  repeat
   {
     x[k] ← (x[k]+1) mod (m+1) ; //next color index when it is                    //highest index
     if(x[k] = 0) then // no new color is remaining
       return;
     for (j ← 1 to n) do
     {
      // Taking care of having different colors for
     // adjacent
    // vertices by using two conditions i) edge should be
```

```
// present between two vertices
// ii) adjacent vertices should not have same color
If((G[k,j]!=0) AND (x[k]→ x[j])) then
  break;
}
//If there is no new color remaining
If ( j= n+1) then
  return;
}until(false);
}
```

Q.25 Describe the backtracking technique to m-coloring graph. Explain with an example.

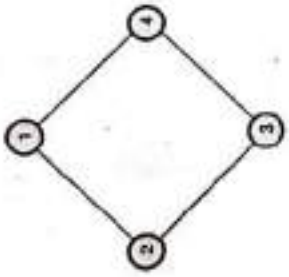
[JNTU : Part B, May-12, Marks 7]

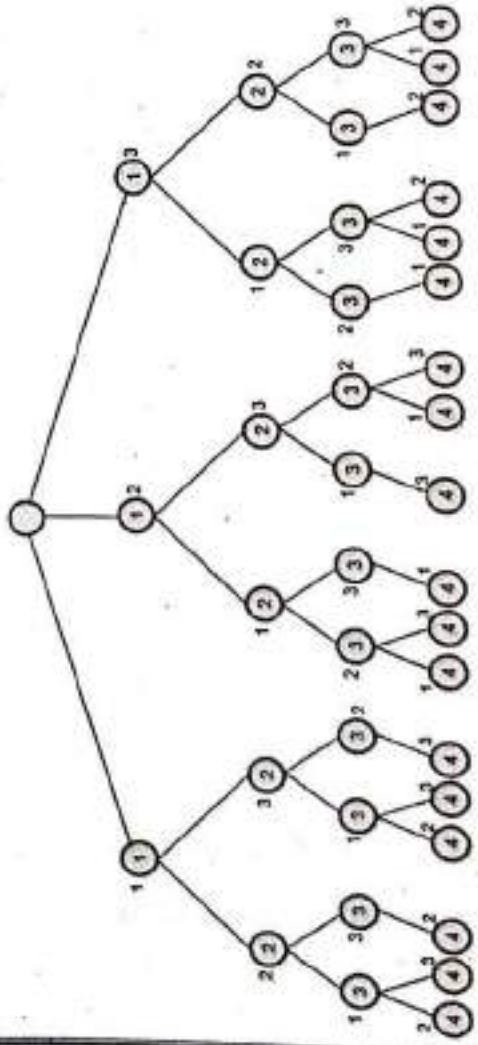
Ans. : Refer Q.24.

Q.26 Draw the portion of the state space tree for m-colorings of a graph.

[JNTU : Part B, May-12, Marks 8, Nov.-16, Marks 5]

Ans. : Consider, a graph given below can be solved using three colors.





Consider,

Red = 1, Green = 2 and Blue = 3.

The number inside the node indicates vertex number and the number outside the node indicates color index.

6.5 Hamiltonian Cycles

Q.27 What is Hamiltonian path ?

[JNTU : Part A, Marks 2]

Ans. : Problem - Given an undirected connected graph and two nodes x and y then find a path from x to y visiting each node in the graph exactly once.

Q.28 Find the Hamitonion cycle by using backtracking approach for given graph.

[JNTU : Part B, Marks 5]

Ans. : The state space tree is generated in order to find Hamiltonian cycles. Using backtracking the Hamiltonian cycle can be obtained.

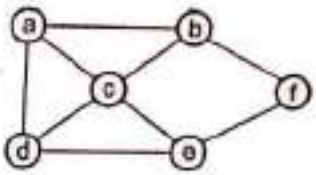


Fig. Q.28.1

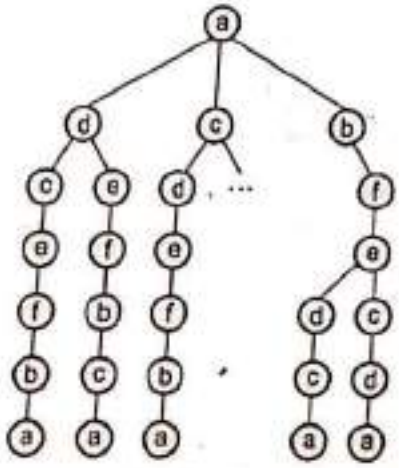


Fig. Q.28.2

Q.29 Write a recursive backtracking algorithm for finding the Hamiltonian Cycle.

[JNTU : Part B, Marks 5]

Ans. :

```
Algorithm H_Cycle(k)
 // This is recursive backtracking algorithm for finding
Hamiltonian cycle
//The graph G[1:n,1:n] is adjacency matrix used for
graph G
//The cycle begins from the vertex 1
{
 repeat
 {
  Next_Vertex(k);//generates next legal vertex for
determining cycle
  if(x[k]=0) then
   return;
  if(k=n) then
   write(x[1:n]) // print the Hamiltonian cycle
  else
    H_Cycle(k+1);
 } until(false);
}
```

The algorithm for generating next vertex which helps in finding Hamiltonian cycle is as given below -

```
Algorithm Next_Vertex(k)
// This algorithm finds the Hamiltonian path by
choosing appropriate
// vertex each time. The x[1:k-1] is a Hamiltonian path.
//Each time the x[k] is assigned to a next highest
numbered vertex
// which is not visited earlier. If x[k]=0 then no vertex is
assigned
// to x[k].
//n denotes total number of vertices
{
 while(1)
 {
     //obtain next vertex
  x[k] : =x[k+1] mod (n+1);
  if(x[k]=0) then
     return;
  if(G[x[k-1],x[k]]=1) then // if edge between vertex k
and
     // k-1 is present
  {
   for (j=1 to k - 1 ) do //for every adjacent vertex
    if(x[j]=x[k]) then
     break;// not a distinct vertex
```

```
if(j=k) then // obtained a distinct vertex
{
  if((k<n) OR ((k=n) AND G[x[n],x[1]]=1)) then
    return;//return a distinct vertex
}
}
}
```

END ...

UNIT - V

7 Branch and Bound

7.1 General Method

Important Points to Remember

- Branch and bound is a general algorithmic method for finding optimal solutions of various optimization problems.
- Branch and bounding method is a general optimization technique that applies where the greedy method and dynamic programming fail. However, it is much slower.
- Applications of Branch and Bound method are - i) Backtracking and ii) Branch and Bound

Q.1 Compare backtracking and branch and bound method. ☞ [JNTU : Part A, Marks 3]

Ans. :

Backtracking	Branch and bound
Solution for backtracking is traced using depth first search.	In this method, it is not necessary to use depth first search for obtaining the solution, even the breadth first search, best first search can be applied.
Typically decision problems can be solved using backtracking.	Typically optimization problems can be solved using branch and bound.
While finding the solution to the problem bad choices can be made.	It proceeds on better solutions. So there cannot be a bad solution.
The state space tree is searched until the solution is obtained.	The state space tree needs to be searched completely as there may be chances of being an optimum solution any where in state space tree.
Applications of backtracking are - M coloring, Knapsack,	Applications of branch and bound are - Job sequencing, TSP.

Q.2 Define the term branch and bound technique explain it with an example. ☞ [JNTU : Part B, May-09, Marks 8]

Ans. :

- In branch and bound method a state space tree is built and all the children of E nodes (a live node whose children are currently being generated) are generated before any other node can become a live node.
- For exploring new nodes either a BFS or D-search technique can be used.
- In Branch and bound technique, BFS-like state space search will be called FIFO (First In First Out) search. This is because the list of live node is First In First Out list (queue). On the other hand the D-search like state space search will be called LIFO search because the list of live node is Last in First out list (stack).
- In this method a space tree of possible solutions is generated. Then partitioning (called as branching) is done at each node of the tree. We compute lower bound and upper bound at each node. This computation leads to selection of answer node.
- Bounding functions are used to avoid the generation of subtrees that do not contain an answer node.

Q.3 Explain the general method of branch and bound. ☞ [JNTU : Part B, May-09, Marks 8]

Ans. : Algorithm Branch_Bound()

```
{
//E is a node pointer;
  E← new(node);  // This is the root node which is the
                 // dummy start node
  //H is heap for all the live nodes.
  // H is a min-heap for minimization problems.
  //H is a max-heap for maximization problems.
  while (true)
  {
   If (E is a final leaf) then
```

```
{
   //E is an optimal solution
   write( path from E to the root);
   return;
}
Expand(E);
if (H is empty) then //if no element is present in
                     heap
   {
      write(" there is no solution");
      return;
   }
E ← delete_top(H);
}
}
```

Following is an algorithm named Expand is for generating state space tree -

```
Algorithm Expand(E)
{
Generate all the children of E;
Compute the approximate cost value of each child;
Insert each child into the heap H;
}
```

7.2 LC Search

Q.4 What is LC search ? ☞ **[JNTU : Part A, Marks 3]**

Ans. :

- For speeding up the search process we need to intelligent ranking function for live nodes. Each time, the next E-node is selected on the basis of this ranking function. For this ranking function additional computation (normally called as cost) is needed to reach to answer node from the live node.
- The Least Cost (LC) search is a kind of search in which least cost is involved for reaching to answer node. At each E-node the probability of being an answer node is checked.
- BFS and D-search are special cases of LC search.

Q.5 Write control abstraction algorithm for LC search. ☞ **[JNTU : Part A, Nov.-16, Marks 3]**

Ans. : The control abstraction for Least cost search is given as follows.

```
Algorithm LC_search()
//tr is a state space tree and x be the node in the tr
//E represents the E-node
//initialize the list of live nodes to empty
{
      if( tr is answer node) then
      {
         write(tr);//output the answer node
         return;
      }
      E←tr //set the E-node
      repeat
      {
         for(each child x of E) do
         {
            if(x is answer node) then
            {
               output the path from x to root;
               return;
            }
//the new live node x will be added in the list of live
//nodes
            Add_Node(x);
            x → parent ← E;
//pointing the path from x to root;
      }
      if(no more live nodes) then
      {
         write("Can not have answer node!!");
         return;
      }
// E points to current E-node
      E← Least_cost(); //finds the least cost node to set
                       // next E-node
   }until(false);
}
```

In above algorithm if x is in tr then c(x) be the minimum cost answer node in tr. The algorithm uses two functions Least_cost and Add_Node() function to delete or add the live node from the list of live nodes. Using above algorithm we can obtain path from answer node to root. We can use parent to trace the parent of node x. Initially root is the E node. The for loop used in the algorithm examines all the children of E-node for obtaining the answer node. The function Least_cost() looks for the next possible E-node.

Q.6 Explain properties of LC-search.

[JNTU : Part B, May-09, Dec.-11, Marks 8]

Ans. :

- In branch and bound method the basic idea is selection of E-node. The selection of E-node should so perfect that we will reach to answer node quickly.
- Using FIFO and LIFO branch and bound method the selection of E-node is very complicated and somewhat blind.
- For speeding up the search process we need to intelligent ranking function for live nodes. Each time, the next E-node is selected on the basis of this ranking function. For this ranking function additional computation (normally called as cost) is needed to reach to answer node from the live node.
- The Least Cost (LC) search is a kind of search in which least cost is involved for reaching to answer node. At each E-node the probability of being an answer node is checked.
- BFS and D-search are special cases of LC search.
- Each time the next E-Node is selected on the basis of the ranking function (smallest c^(x)). Let g^(x) be an estimate of the additional effort needed to reach an answer node from x. Let h(x) to be the cost of reaching x from the root and f(x) to be any non-decreasing function such that

$$c^(x) = f(h(x)) + g^(x)$$

- If we set g^(x)=0 and f(h(x)) to be level of node x then we have BFS.
- If we set f(h(x))=0 and g^(x) ≤ g^(y) whenever y is a child of x then the search is a D-search.
- An LC search with bounding functions is known as LC Branch and Bound search.
- In LC search, the cost function c(.) can be defined as

  i) If x is an answer node then c(x) is the cost computed by the path from x to root in the state space tree.

  ii) If x is not an answer node such that subtree of x node is also not containing the answer node then c(x) = ∞.

  iii) Otherwise c(x) is equal to the cost of minimum cost answer node in subtree x.

- c^(.) with f(h(x))=h(x) can be an approximation of c(.).

7.3 Bounding

Q.7 Explain the concept of Bounding.

[JNTU : Part A, Marks 3]

Ans. :

- The bounding functions are used to avoid the generation of sub trees that do not contain the answer nodes. In bounding lower bounds and upper bounds are generated at each node.
- A cost function c^(x) is such that c^(x) ≤ c(x) is used to provide the lower bounds on solution obtained from any node x.
- Let *upper* is an upper bound on cost of minimum-cost solution. In that case, all the live nodes with c^(x)> upper can be killed.
- At the start the upper is usually set to ∞. After generating the children of current E-node, *upper* can be updated by minimum cost answer node. Each time a new answer node can be obtained.

Q.8 Explain the principles of :
a) Control Abstraction for LC-search. b) Bounding

[JNTU : Part B, Dec.-11, May-09, Marks 8]

Ans. : (a) Refer Q. 5.

(b) Refer Q. 7.

Q.9 Let n = 4

Job index	P_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

Then select an optimal subset J with optimal penalty. What will be the penalty corresponding to optimal solution ?

[JNTU : Part B, April-11, Marks 15]

Ans. : The state space tree can be drawn for proper selection of job i for creating J. There are two ways by which the state space tree can be drawn : Fixed tuple size formulation and variable tuple size

formulation. The variable tuple size formulation can be as shown below.

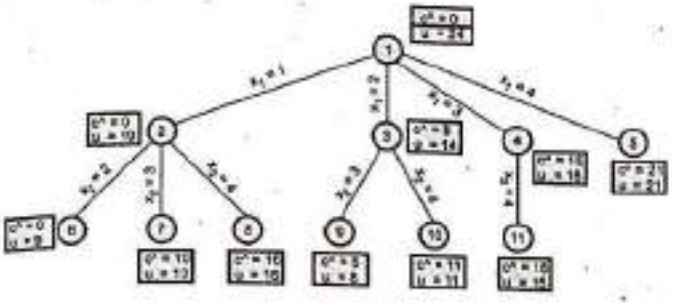


Fig. Q.9.1 State space tree with variable tuple size formulation

The function c^(x) can be computed for each node x as

$$c\hat{}(x) = \sum_{i<m \; i \in S_x} P_i$$

where $m = \max\{ i \mid i \in S_x \}$

and S_x be subset of jobs selected for J at node x.

The upper bound can be computed using function u(x). Then,

$$u(x) = \sum_{i \notin S_x} P_i$$

Here u(x) corresponds to the cost of the solution S_x for node x.

For node 1 (root) there are 4 childrens. These children are for selection of either 1 or 2 or 3 or for job 4. Hence $x_1 = 2$ or $x_1 = 3$ or $x_1 = 4$.

For node 2 we have with $x_1 = 1$ and therefore we can select either 2, 3 or 4. Hence $x_2 = 2$ or $x_2 = 3$ or $x_2 = 4$ corresponds to node 6, 7 or 8. Continuing in this fashion, we have drawn the state space tree.

At node 1 :

$c\hat{} = 0$

$u = 24$ ∵ $\sum_{i=1}^{n} P_i = 5 + 10 + 6 + 3$

At node 2 :

$c\hat{} = 0$ ∵ $\sum P_i$ where $i < 1 = 0$

$u = 19$ ∵ the sum of P_i excluding $x_1 = 1$, i.e. $P_1 = 5$.

At node 3 :

$c\hat{} = 5$ ∵ $\sum P_i$ where $i < 2$

$u = 14$ ∵ $\sum P_i$ without $x_1 = 2$

i.e. $24 - 10 = 14$

At node 4 :

$c\hat{} = 15$ ∵ $\sum P_i$ where $i < 3$

$u = 18$ ∵ $\sum P_i$ without P_3

i.e. $24 - 6 = 18$

At node 5 :

$c\hat{} = 21$ ∵ $\sum P_i$ where $i < 4$

$u = 21$ ∵ $\sum P_i$ without P_4

i.e. $24 - 3 = 21$

At node 6 :

$c\hat{} = 0$ ∵ $\sum_{i<2} P_i$ and not containing $x_1 = 1$ i.e. P_1

$u = 9$ ∵ $\sum_{i=1}^{n} P_i$ such that $i \neq 2$ and $i \neq 1$. Hence $6 + 3 = 9$

At node 7 :

$c\hat{} = 10$ ∵ $\sum_{i<3} P_i$ where $i \neq 1$ and $i \neq 3$. Hence $P_2 = 10$.

$u = 13$ ∵ $\sum_{i=1}^{n} P_i$ such that $i \neq 1$ and $i \neq 3$.

Hence $P_2 + P_4 = 13$

At node 8 :

$c^\wedge = 16$ ∵ $\sum_{i<4} P_i$ and $i \neq 1$ and $i \neq 4$.

Hence $P_2 + P_3 = 16$.

$u = 16$ ∵ $\sum_{i=1}^{n} P_i$ and $i \neq 1$ and $i \neq 4$.

Hence $P_2 + P_4 = 16$

At node 9 :

$c^\wedge = 5$ ∵ $\sum_{i<3} P_i$ but $i \neq 2$. Hence $P_1 = 5$.

$u = 8$ ∵ $\sum_{i=1}^{n} P_i$ but $i \neq 2$ and $i \neq 3$.

That is $P_1 + P_4 = 8$

At node 10 :

$c^\wedge = 11$ ∵ $\sum_{i<4} P_i$ but $i \neq 2$.

Hence $P_1 + P_3 = 5 + 6 = 11$.

$u = 11$

At node 11 :

$c^\wedge = 15$ ∵ $\sum_{i<4} P_i$ and $i \neq 3$.

Hence $P_1 + P_2 = 5 + 10 = 15$

$u = 15$ ∵ $\sum_{i=1}^{n} P_i$ and $i \neq 3$ and $i \neq 4$.

Hence $P_1 + P_2 = 5 + 10 = 15$

Now we will draw fixed tuple size formulation.

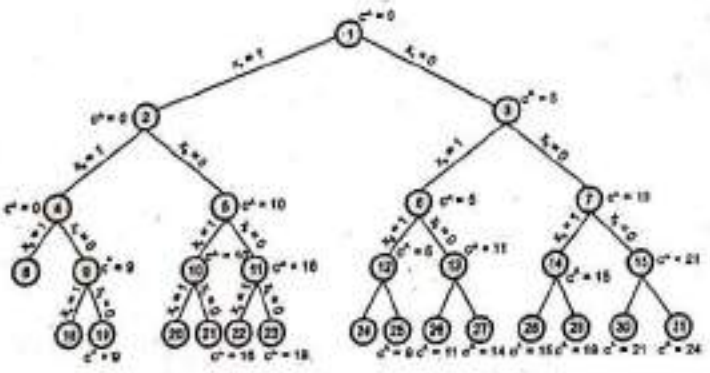


Fig. Q.9.2 State space tree for fixed tuple size formulation

In fixed tuple size formulation. x_i=1 means job i is selected $x_i = 0$ means job i is not selected for formation of J (the optimal subset): Initially, $c^\wedge = 0$ for node 1 as no job is selected. For node 3, it indicates omission of job 1. Hence, penalty is 5. For node 5 there is omission of job 2. Hence $c^\wedge$ = penalty = 10. For node 7 there is omision of job 1 and job 2. Hence $c^\wedge$=15. For node 13 there is omission of job 1 and job 3. Hence penalty is 11. Therefore $c^\wedge = 11$. Continuing in this fashion $c^\wedge$ are computed.

Q.10 What is bounding ? Explain the principles of bounding. [JNTU : Part B, May-13, Marks 7]

Ans. : Refer Q. 7 and 8.

7.4 Travelling Salesperson Problem

Important Points to Remember

- **Problem Statement for TSP :** 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 starting city, completes the tour.
- Typically travelling salesperson problem is represented by weighted graph.
- **Tour(x)** is the path that begins at root and reaching to node x in a state space tree and returns to root. In branch and bound strategy cost of each node x is computed. The travelling salesperson problem is solved by choosing the node with optimum cost.

Q.11 Explain the terms row minimization, column minimization and full reduction. [JNTU : Part B, Marks 5]

Ans. : Row Minimization : Row minimization is a technique in which each row of the matrix is reduced by the choosing minimum value from each row. For example -

We will find minimum of each row.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \quad \begin{matrix} 10 \\ 2 \\ 2 \\ 3 \\ 4 \end{matrix}$$

21 Total reduced cost.

We will subtract the row_minimum value from corresponding row. Hence,

$$Red\_Row(M) = \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

Column Minimization : Column minimization is a technique in which each column of the matrix is reduced by the choosing minimum value from each row. For example -

Consider matric M as

$$\begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

min value: 1 0 3 0 0

It is ignored | It is ignored | It is ignored

The reduced column matrix will be -

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Full Reduction : It is a technique in which the matrix is firstly reduced using row minimization technique and then reduced using column minimization technique.

Q.12 The edge length of a directed graph are given by the below matrix. Using the traveling salesperson algorithm, calculate the optimal tour.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

[JNTU : Part B, May-09, Marks 16]

Ans. : Fully reduced matrix is,

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

Optimum cost = 21 + 4 = 25.

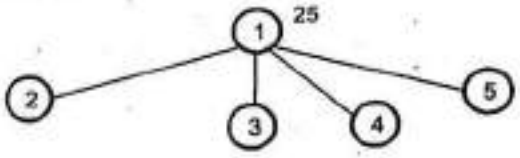


Fig. Q.12.1 Portion of state space tree

Consider path 1, 2. Make 1st row and 2nd column ∞. And set M [2] [1] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

ignore | ignore | ignore | ignore | ignore

ignore ignore ignore ignore ignore

Cost of node 2 is

25 + 0 + 10

↑ optimum cost; ↑ reduced cost; ↑ old value of M[1] [2]

= 35

Consider path 1, 3. Make 1st row = ∞, 3rd column = ∞ and M[3] [1] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix}$$

↓

11

Cost of node 3 is

= 25 + 11 + 17

↑ optimum cost; ↑ reduced cost; ↑ M[1][3]

= 53

Consider path 1, 4.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

Cost of node 4 is = 25 + 0 + 0

= 25

Consider path 1, 5.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix} \begin{matrix} \\ \rightarrow 2 \\ \\ \rightarrow 3 \\ \\ \end{matrix}$$

Cost of node 5 is = 25 + 5 + 1

= 31

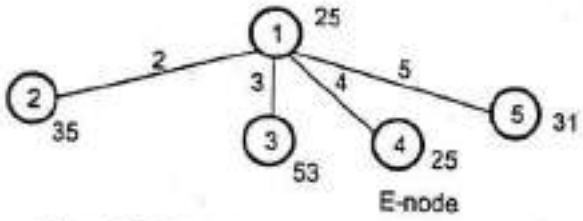


Fig. Q.12.2 Portion of state space tree

Now, as cost of node 4 is optimum, we will set node 4 as E-node and generate its children nodes 6, 7, 8.

Consider path 1, 4, 2 for node 6. Set 1st row, 4th row to ∞. Set 2nd column to ∞. Also M[4] [1] = ∞, M[2] [1] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

Cost of node 6 = 25 + M [4, 2]

= 25 + 3

= 28

Consider path 1, 4, 3 for node 7. Set 1st row, 4th row to ∞. Set 4th column, 3rd column to ∞. Set M [4] [1] = M[3] [1] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} \rightarrow 2$$

$$\uparrow$$

11

Cost of node 7 = 25 + 13 + M [4] [3]

= 25 + 13 + 12

= 50

Consider path 1, 4, 5 for node 8.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix} \rightarrow 11$$

Cost of node 8 = 25 + 11

= 36

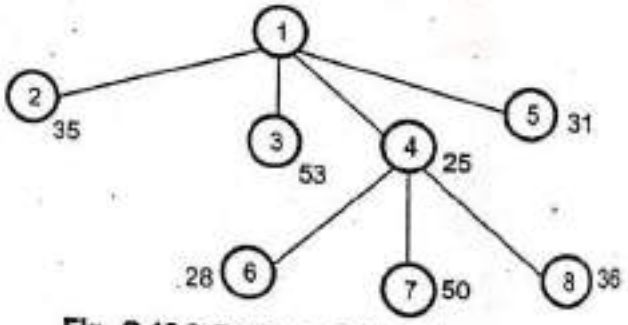


Fig. Q.12.3 Portion of state space tree

Now as cost of node 6 is minimum, node 6 becomes an E-node. Hence generate children for node 6. Node 9 and 10 are children nodes of node 6.

Consider path 1, 4, 2, 3 for node 9. Set 1st row, 4th row, 2nd row to ∞. Set 4th column, 2nd column and 3rd column to ∞.

Set M[1,4], M[1,2], M[1,3] to ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix} \rightarrow 2$$

$$\downarrow$$

11

Cost of node 9

= 28 + 13 + 11
↑ optimum cost ↑ reduced cost ↑ M[2][3]

= 52

Consider path 1, 4, 2, 5 for node 10. Set 1 st row, 4th row , 2nd row to ∞. Set 4th column, 2nd column and 5th column to ∞. Set M [1] [4] = M[1] [2] = M [1] [5] = ∞.

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

Cost of node 10 = 28 + M[2] [5] = 28 + 0 = 28

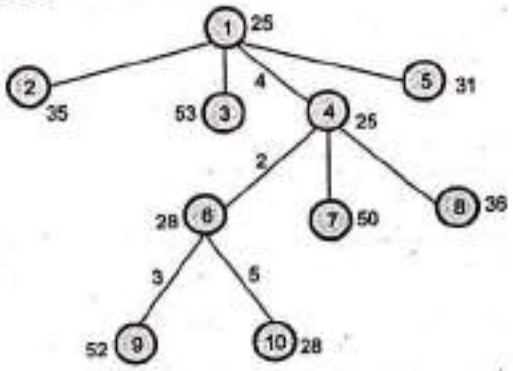


Fig. Q.12.4 Portion of state space tree

Node 10 becomes E-node now.

As for node 10 only child being generated is node 11. We set path as 1, 4, 2, 5, 3. To complete the tour we return to 1. Hence the state space tree is,

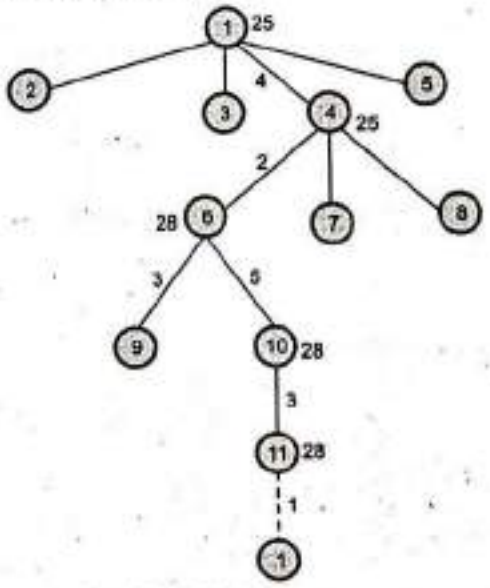


Fig. Q.12.5 State space tree

Hence the optimum cost of the tour is 28.

Q.13 Apply the branch and bound algorithm to solve the TSP for the following cost matrix.

$$\begin{bmatrix} \infty & 11 & 10 & 9 & 6 \\ 8 & \infty & 7 & 3 & 4 \\ 8 & 4 & \infty & 4 & 8 \\ 11 & 10 & 5 & \infty & 5 \\ 6 & 9 & 5 & 5 & \infty \end{bmatrix}$$

[JNTU : Part B, May-09, Marks 16, May-12, Marks 15]

Ans. : Step 1 : We will find the minimum value from each row and subtract the value from corresponding row.

row

∞	11	10	9	6	→ 6
8	∞	7	3	4	→ 3
8	4	∞	4	8	→ 4
11	10	5	∞	5	→ 5
6	9	5	5	∞	→ 5
					23

Minvalue ⟹

∞	5	4	3	0
5	∞	4	0	1
4	0	∞	0	4
6	5	0	∞	0
1	4	0	0	∞

reduced Matrix

Fig. Q.13.1

Now we will obtain minimuim value from each column. If any column contains 0 then ignore that column and a fully reduced matrix can be obtained.

∞	5	4	3	0
5	∞	4	0	1
4	0	∞	0	4
6	5	0	∞	0
1	4	0	0	∞
↑ 1	↑ Ignore	↑ Ignore	↑ Ignore	↑ Ignore

Subtracting ⟹ 1 from 1st column

∞	5	4	3	0
4	∞	4	0	1
3	0	∞	0	4
5	5	0	∞	0
0	4	0	0	∞

Total reduced cost = Total reduced row cost + Total reduced column cost
= 23 + 1
= 24

Now we will set 24 as the optimum cost.

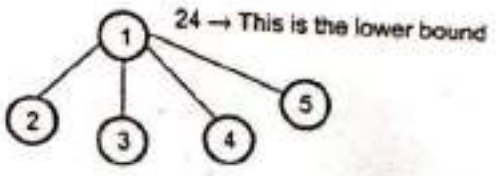


Fig. Q.13.2

∞	∞	∞	∞	∞	→ ignore
4	∞	4	∞	1	→ 1
3	0	∞	∞	4	→ ignore
∞	5	0	∞	0	→ ignore
0	4	0	∞	∞	→ ignore

Subtracting ⇒ 1 from 2nd row

∞	∞	∞	∞	∞
4	∞	3	∞	0
3	0	∞	∞	4
∞	5	0	∞	0
0	4	0	∞	∞
↑	↑	↑	↑	↑
Ignore	Ignore	Ignore	Ignore	Ignore

The total cost for node 4 is

= Optimum cost + M [1] [4] + Minimum row cost

= 24 + 3 + 1

= 28

Consider the path (1, 5). Make 1st row, 5th column to be ∞. Set M [5] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
4	∞	4	0	∞	→ ignore
3	0	∞	0	∞	→ ignore
5	5	0	∞	∞	→ ignore
∞	4	0	0	∞	→ ignore
↑	↑	↑	↑	↑	
3	Ignore	Ignore	Ignore	Ignore	

Subtracting 3 from 1st column.

∞	∞	∞	∞	∞
1	∞	4	0	∞
0	0	∞	0	∞
2	5	0	∞	∞
∞	4	0	0	∞

The total cost at node 5 is

= Optimum cost + Reduced column cost + Old value M [1] [5]

= 24 + 3 + 0

= 27

The partial state space tree will be -

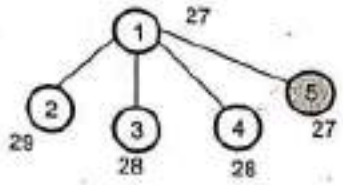


Fig. Q.13.3

The node 5 shows minimum cost. Hence node 5 will be an E node. That means we will select node 5 for expansion.

Step 3 : Now we will consider the paths [1, 5, 2], [1, 5, 3] and [1, 5, 4], of above state space tree do the further computations. Consider path 1, 5, 2. Make 1st row, 5th row and second column as ∞. Set M [5] [1] and M [2] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
∞	∞	4	0	1	→ ignore
3	∞	∞	0	4	→ ignore
5	∞	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore
↑	↑	↑	↑	↑	
3	ignore	ignore	ignore	ignore	

Now subtracting 3 from 1st column, we get -

∞	∞	∞	∞	∞
∞	∞	4	0	1
0	∞	∞	0	4
2	∞	0	∞	0
∞	∞	∞	∞	∞

Hence total cost for node 6 will be -

= Optimum cost at node 5 + Column-reduced cost + M [5] [2]

= 27 + 3 + 4

= 34

Consider path [1, 5, 3]. Make 1st row, 5th row and 3rd column as ∞.

Set M [5] [1] = M [3] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
4	∞	∞	0	∞	→ ignore
∞	0	∞	0	∞	→ ignore
5	5	∞	∞	∞	→ 5
∞	4	∞	0	∞	→ ignore

Subtracting ⇒ 5 from 4th row

∞	∞	∞	∞	∞
4	∞	∞	0	∞
∞	0	∞	0	∞
1	1	∞	∞	∞
∞	4	∞	0	∞
↑	↑	↑	↑	↑
1	ignore	ignore	ignore	ignore

Subtract 1 from 1st column

∞	∞	∞	∞	∞
3	∞	∞	0	∞
∞	0	∞	0	∞
0	1	∞	∞	∞
∞	4	∞	0	∞

The total cost for node 7 will be -

= Optimum cost at node 5 + Column reduced cost + M [5] [3]

= 27 + 1 + 0

= 28.

Consider the path [1, 5, 4]. Make first row, fifth row and forth column to be ∞.

Set M [5] [1] = M [4] [1] = ∞.

∞	∞	∞	∞	∞	→ ignore
4	∞	4	∞	1	→ 1
3	0	∞	∞	4	→ ignore
∞	5	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore

Subtracting 1 from ⇒ 2nd row

∞	∞	∞	∞	∞
3	∞	3	∞	0
3	0	∞	∞	4
∞	5	0	∞	0
∞	∞	∞	∞	∞
↑	↑	↑	↑	↑
3	ignore	ignore	ignore	ignore

Subtract 3 from ⇒ 1st column

∞	∞	∞	∞	∞
0	∞	3	∞	0
0	0	∞	∞	4
∞	5	0	∞	0
∞	∞	∞	∞	∞

The total cost for node 8 will be

= Optimum cost at node 5 + Column-reduced + Row-reduced cost + M [5] [4]

= 27 + (1 + 3) + 0

= 31

The partial state space tree will be -

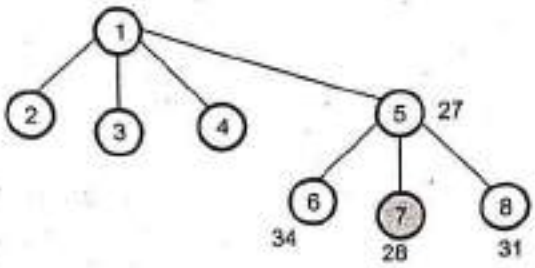


Fig. Q.13.4

The node 7 has optimum cost. Hence 7 becomes an E node and it will be expanded further.

Step 4 : Now we will consider paths [1, 5, 3, 2] and [1, 5, 3, 4] for further computations. Consider path 1, 5, 3, 2 and make 1st, 5th, 3rd row as ∞ and 2nd column as ∞.
Set M [2] [1] = M [3] [1] = M[5][1] = ∞.

The matrix becomes -

∞	∞	∞	∞	∞	→ ignore
∞	∞	4	0	1	→ ignore
∞	∞	∞	∞	∞	→ ignore
5	∞	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore
↑	↑	↑	↑	↑	
5	ignore	ignore	ignore	ignore	

Subtract 5 from 1st column.

∞	∞	∞	∞	∞
∞	∞	4	0	1
∞	∞	∞	∞	∞
0	∞	0	∞	0
∞	∞	∞	∞	∞

The reduced cost at node 9 [i.e. path [1, 5, 3, 2]] will be

= Optimum cost at node 7 + Column-reduced cost + M [3] [2]

= 28 + 5 + 0

= 33

Consider the path [1, 5, 3, 4]. Make 1st row, 5th row, 3rd row and 4th column as ∞.

Set M [5] [1] = M [3] [1] = M [4] [1] = ∞,

∞	∞	∞	∞	∞	→ ignore
4	∞	4	∞	1	→ 1
∞	∞	∞	∞	∞	→ ignore
∞	5	0	∞	0	→ ignore
∞	∞	∞	∞	∞	→ ignore

Subtracting 1 from second row,

∞	∞	∞	∞	∞
3	∞	3	∞	0
∞	∞	∞	∞	∞
∞	5	0	∞	0
∞	∞	∞	∞	∞
↑ 3	↑ 5			

Subtracting 3 from 1st column and 5 from 2nd column

∞	∞	∞	∞	∞
0	∞	3	∞	0
∞	∞	∞	∞	∞
∞	0	0	∞	0
∞	∞	∞	∞	∞

The total reduced cost at node 10 [i.e. path 1, 5, 3, 4] is

= Optimum cost at node 7 + Row-reduced cost + M [3] [4]

= 28 + (3 + 5) + 0 = 36

The partial state space tree will be -

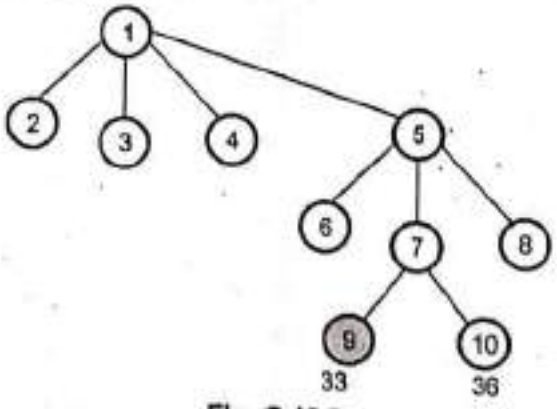


Fig. Q.13.5

Step 5 : Now consider path [1, 5, 3, 2, 4]

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	5	0	∞	0
∞	∞	∞	∞	∞
	↓ 5 minvalue			

Now subtract 5 from 2nd column

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	0	0	∞	0
∞	∞	∞	∞	∞

The reduced cost at node 11 will be -

= Optimum cost at node 9 + column reduced cost + M [2] [4]

= 33 + 5 + 0 = 38

The final state space tree will be -

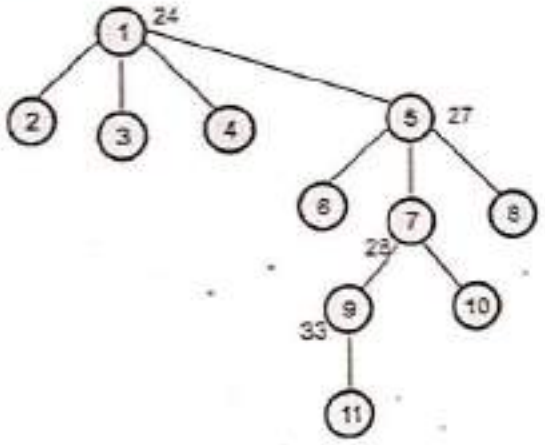


Fig. Q.13.6

The tour with minimum cost is 29. The path will be 1, 5, 3, 2, 4, 1.

Q.14 Solve TSP problem having the following cost-matrix using branch - and - bound technique.

[JNTU : Part B, Dec.-12, Marks 15]

	A	B	C	D
A	X	5	2	3
B	4	X	1	5
C	4	2	X	3
D	7	6	8	X

Ans. :

Step 1 : We will find minimum from each row.

∞	5	2	3	2
4	∞	1	5	1
4	2	∞	3	2
7	6	∞	∞	6

11 Total reduced cost

We will subtract row_minimum value from each corresponding row.

Now we will find minimum from each column and then subtract that minimum amount from each corresponding column. **Reduced column cost = 2**

∞	3	0	1
3	∞	0	4
2	0	∞	1
1	0	2	∞

If the column contains 0 then do not look for any minimum value. The reduced matrix will be,

∞	3	0	0
2	∞	0	3
1	0	∞	0
0	0	2	∞

This is fully reduced matrix.

∴ Total reduced cost = Reduced row cost + Reduced column cost = 11+2

∴ Optimum cost = 13

The partial state space tree will be

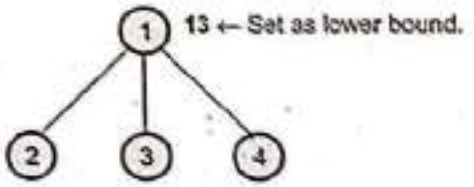


Fig. Q.14.1

Step 2 : We will take fully reduced matrix. Consider **Path 1-2.** Make 1st row ∞, 2nd column = ∞ and set M[2][1] = ∞.

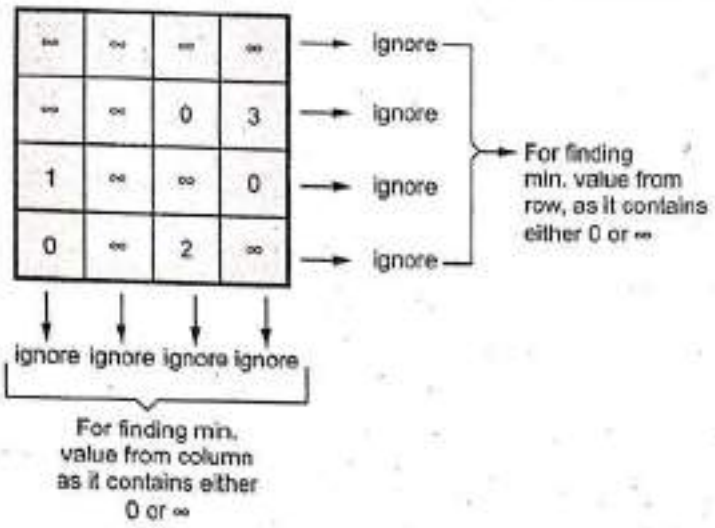


Fig. Q.14.2

∴ Cost of node 2 = Optimum cost + Min. row cost + Min. column cost + Old value M[1][2] from fully reduce matrix

∴ Cost of node 2 = 13 + 0 + 0 + 3

∴ Cost of node 2 = 16

Similarly consider path 1-3. Make 1st row ∞, 3rd column ∞ and M[3][1] = ∞

Obtain min row and min column value

∞	∞	∞	∞
2	∞	∞	3
∞	0	∞	0
0	0	∞	∞

→ 2

∴ Cost of node 3 = Optimum cost + Reduced cost + Old value of M[1][3]

= 13 + 2 + 0

∴ Cost of node 3 = 15

Consider path 1-4. Make 1st row = ∞, 4th column = ∞, M[4][1] = ∞

∴ Cost of node 4 = Optimum cost + Reduced cost + Old value of M [1] [4]

∴ = 13 + 1 + 0

∴ Cost of node 4 = 14

∞	∞	∞	∞
2	∞	0	∞
1	0	∞	∞
∞	0	2	∞

↓

1

The partial state space tree

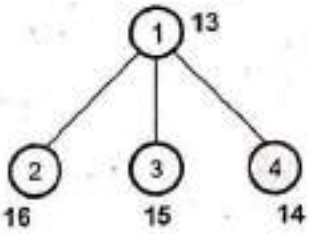


Fig. Q.14.3

We will expand **node 4.** Now we will generate the children 5,6.

∞	∞	∞	∞
∞	∞	0	3
1	∞	∞	0
∞	∞	∞	∞

↓

Min. value 1

Consider **path 1-4-2 for node 5.**

Set 1st row = 4th row = ∞.

The 2nd column = ∞

Also set M[4] [1] = M [2] [1] = ∞.

∴ Cost of node 5 = Optimum cost + Reduced cost of column + Old value of M[4][2]

= 14+1+0

Cost of node 5= 15

Consider **Path 1-4-3** for **node 6** set 1st row = 4th row = 3rd column = ∞

Also set M[4][1] = M[3][1] = ∞

∴ Cost of node 6 = Optimum cost + Reduced row and column value + Old value of M[4][3]

= 14 + (2+2) + 2

Cost of node 6 = 20

∞	∞	∞	∞
2	∞	∞	3
∞	0	∞	0
∞	∞	∞	∞

→ Min. value 2

↓

Min. value 2

The partial state space tree will be,

Clearly, we will expand node 5 as it has optimum cost.

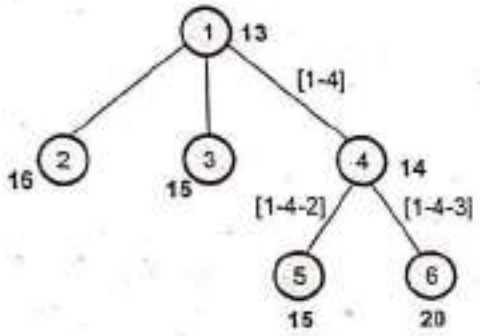


Fig. Q.14.4

Consider path 1-4-2-3 for node 7.

∞	∞	∞	∞
∞	∞	∞	∞
∞	0	∞	0
∞	∞	∞	∞

Set 1st row = 4th row = 2nd row = 3rd column = ∞

Set M[4][1] = M[2][1] = M[3][1] = ∞

There is no min. value from row or column.

∴Cost for node 7 = Optimum cost + Reduced row or column value + Old value M[2-3]

Cost for node 7 = 15 + 0 + 0

cost for node 7 = 15

The partial state space tree

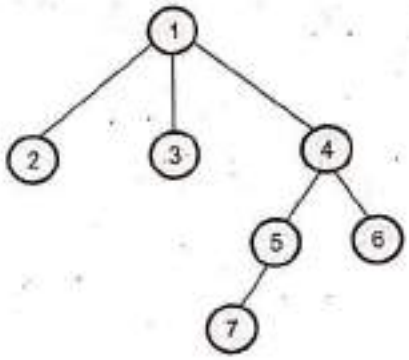


Fig. Q.14.5

The complete state space tree for completing the tour will be by visiting node 1 again

The optimum cost tour is 1-4-2-3-1 with cost 14.

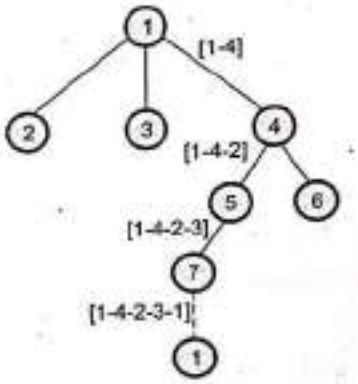


Fig. Q.14.6

Q.15 Describe the Traveling Salesperson Problem in branch and bound.

☞ **[JNTU : Part B, May-12, Marks 7]**

Ans. : Refer Q.4.

7.5 The 0/1 Knapsack Problem

Important Points to Remember

Problem statement :

- The 0/1 Knapsack problem states that - There are 'n' objects given and capacity of Knapsack is 'm'. Then select some objects to fill The knapsack in such a way that it should not exceed the capacity of Knapsack and maximum profit can be earned.
- The Knapsack problem is a maximization problem. That means we will always seek for maximum $P_i x_i$ (where P_i represents profit of object x_i).
- Minimize profit – $\sum_{i=1}^{n} P_i x_i$ subject to $\sum_{i=1}^{n} W_i x_i$

  Such that $\sum W_i x_i \leq m$ and $x_i = 0$ or 1 where $1 \leq i \leq n$

Q.16 Write an algorithm to solve the knapsack problem with the branch and bound.

☞ [JNTU : Part B, May-09, Marks 10]

Ans. :

```
Algorithm C_Bound(total_profit,total_wt,k)
//total_profit denotes the current total profit
//total_wt denotes the current total weight
// k is the index of last removed object
//w[i] represents the weight of object i
//p[i] represents the profit of object i
//m is the weight capacity of knapsack
{
   pt←total_profit;
   wt←total_wt;
   for(i←k+1 to n) do
   {
      wt←wt+w[i];
      if(wt<m) then  pt←pt+p[i];
      else return (pt+(1 – (wt-m)/w[i])*p[i]);
   }
   return pt;
}
```

The algorithm for computing u(x) is as given below

```
Algorithm U_Bound(total_profit,total_wt,k,m)

//total_profit denotes the current total profit
//total_wt denotes the current total weight
// k is the index of last removed object
//w[i] represents the weight of object i
//p[i] represents the profit of object i
//m is the weight capacity of knapsack

   pt←total_profit;
   wt←total_wt;
   for(i←k+1 to n) do
   {
      if(wt+w[i]<=m) then
      {
         pt←pt – p[i];
         wt←wt+w[i];
      }
   }
   return pt;
```

Q.17 Consider Knapsack instance n = 4 with capacity m = 15 such that,

Object i	P_i	W_i
1	10	2
2	10	4
3	12	6
4	18	9

☞ [JNTU : Part B, Marks 5]

Ans. : Let us design state space tree using fixed tuple size formulation. The computation of c^(x) and u(x) for each node x is done.

In Fig. Q.17.1 at each node a structure is drawn in which computation of c^(·) and u(·) is given. The tag field is useful for tracing the path.

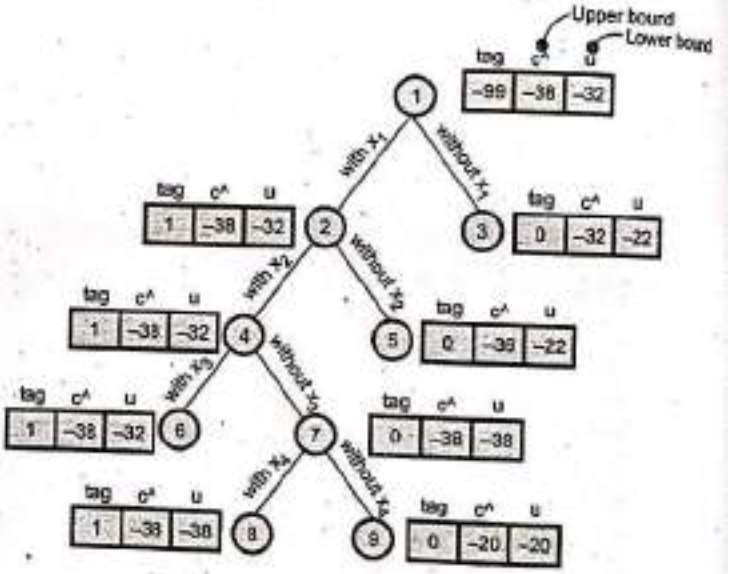


Fig. Q.17.1 LCBB solution space tree

Using algorithm U_Bound u(x) is computed and using algorithm C_Bound c^(x) is computed.

u(1) can be computed as -

for i = 1, 2 and 3

$$\therefore \quad -\sum P_i = -(10 + 10 + 12)$$

$$\therefore \quad u(1) = -32$$

If we select i = 4, then it will exceed capacity of Knapsack.

The computation of c^(1) can be done as follows :

$$c\hat{}(1) = -\left[u(1)+\left[\frac{m-\text{Current total weight}}{\text{Actual weight of remaining object}}\right]*\left[\text{Actual profit of remaining object}\right]\right]$$

$$c\hat{}(1) = -\left[32+\left[\frac{15-(2+4+6)}{9}\right]*18\right]$$

$$= -\left(32+\frac{3}{9}*18\right)$$

$$c\hat{}(1) = -38$$

Now at node 2,

Since the item 1 is selected

The values of $c\hat{}$ and u will be the same i.e. $c\hat{} = -38$, $u = -32$

Now at node 3

Do not select item 1.

$\therefore$ u(3) can be computed as

$$-\sum pi = -(10+12) \text{ i.e. with item 2 and 3.}$$

$$= -22$$

$$\therefore \quad u = -22$$

For computation of $c\hat{}$

$$c\hat{} = -\left[u+\left(\frac{m-\text{total selected wt}}{\text{remaining objects weight}}\right)*\text{profit of remaining object}\right]$$

Here remaining object is object 4.

It has P = 18 and W = 9

$$\therefore \quad c\hat{} = -\left[22+\left(\frac{15-(4+6)}{9}\right)*18\right] = -\left[22+\left(\frac{5}{9}\right)*18\right]$$

$$c\hat{} = -32$$

Now consider node 2. The difference between – 38 and – 32 is 6.

Similarly for node 3. The difference between – 32 and – 22 is 10.

The node having minimum difference between $C\hat{}$ and u becomes E node.

Hence node 2 becomes E node.

Therefore we expand node 2.

In this way considering each possibility of object being in Knapsack or not being in Knapsack with least cost $c\hat{}(x)$ and u(x) is computed. Each time minimum $-\sum P_i x_i$ will become E-node an we will get the answer node as node 8. (Refer Fig. Q.17.1). If we trace the tag field we will get tag(2) – tag (4) – tag (7) – tag (8). i.e. 1101. Hence $x_4 = 1$, $x_3 = 0$, $x_2 = 1$ and $x_1 = 1$. We will select object x_4, x_2 and x_1 to fill up the Knapsack and gain maximum profit.

Q.18 Find an optimal solution for the following 0/1 Knapsack instance using branch and bound method :
Number of objects n = 5, capacity of knapsack m = 100
Profits = (10, 20, 30, 40, 50), weights = (20, 30, 66, 40, 60) ☞ **[JNTU : Part B, Marks 5]**

Ans. : We will arrange all the items in such a way that

$$\frac{P_i}{W_i} > \frac{P_{i+1}}{W_{i+1}} > \frac{P_{i+2}}{W_{i+2}} \cdots$$

The arrangement will be

Item	P_i	W_i
4	40	40
5	50	60
2	20	30
1	10	20
3	30	66

- Here we will select item 4, item 5. Now if we select item 2 then total weight > m. Hence by selecting item 4 and item 5 the upper bound for profit is obtained i.e. 40 + 50 = 90 ∴ we will write ub = – 90 for root node, of BB tree. As by choosing item 4 and item 5 the total capacity of knapsack is fulfilled.

Hence $c^{\wedge} = ub = -90$

Now if we don't select item 4. Then consider selection of item 5 and item 2. Then upper bound for profit is ub = – 70. By this selection the total weight = 50 + 30 = 90. Now we need only weight 10 to reach to maximum capacity of knapsack.

Just consider next item for selection i.e. item 1 but we need its fractional weight

$$c^{\wedge} = ub + \text{fractional profit of next item}$$

$$= (70) + \left(\frac{10}{20} * 10\right) = 75$$

$$c^{\wedge} = -75$$

The partial state space tree will be -

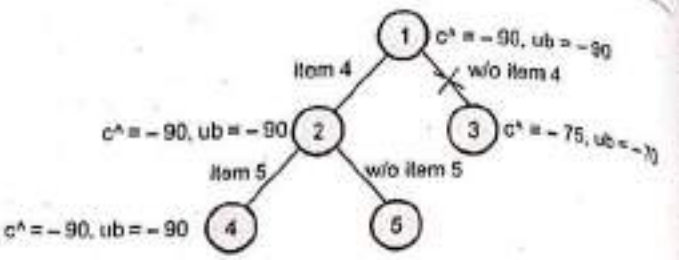


Fig. Q.18.1

We will choose node 2 because it is having less upper bound than node 3. The node 4 and node 5 has the same c^ and ub as that of node 2 because it considers selection of item 4 and item 5.

Now consider node 5 i.e. select item 4 and item 2 ub = – (40 + 20) = – 60. The next remaining weight of item 1 is 20 which can be added as a whole.

$$\therefore \quad c^{\wedge} = -(\text{ub} + \text{weight of item 1})$$

$$= -(60 + 10)$$

$$\therefore \quad c^{\wedge} = -70$$

∴ At node 5 $c^{\wedge} = -70$, ub = – 60

- But as node 4 is having lesser upper bound than node 5. Select node 4.

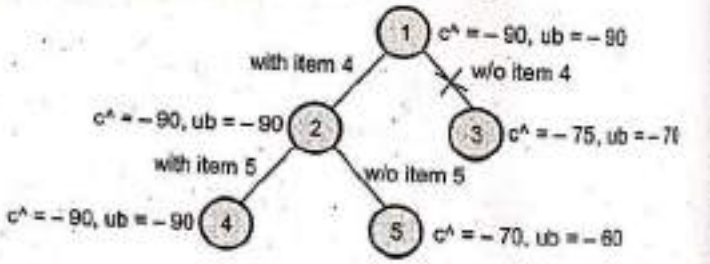


Fig. Q.18.2

- Now can not further select item 2, item 1 or item 3 because by selecting item 4 and item 5 the knapsack has reached to its capacity.
- Hence solution = {item 4, item 5} with maximum profit = 90.

Q.19 Differentiate between Dynamic knapsack and Branch and bound knapsack problem.
☞ **[JNTU : Part B, May-09, Marks 6]**

Ans. :

Sr. No.	Dynamic knapsack	Branch and bound knapsack
1.	In dynamic Knapsack method, the principle of optimality is used.	In branch and bound method a state space tree is built using bounding function.
2.	A sequence of decision $S^{(i)}$ is made, while solving the problem using this option.	A root node is further expanded using E node.
3.	By merging and purging rules the pairs (P, W) can be eliminated.	If particular node posses C^(·) value which is greater than upper then the node is not expanded further.
4.	In dynamic programming the ordering is done using P and W. No two tuples have the same P or W value.	In FIFOBB, many nodes may lie on the same level and there might be same P and W values.
5.	This algorithm is slower.	This algorithm runs faster.
6.	The time complexity is $O(2^n)$.	The LCBB algorithm has O (log m) time complexity where m is number of live nodes.

7.6 FIFO and LC Branch and Bound Solution

Q.20 What is FIFO branch and bound search ? [JNTU : Part A, Marks 3]

Ans. : (1) FIFO branch and bound stands for First In First Out branch and bound. (2) In this search method the children of E node (Expanded node) are inserted in queue. (3) This method makes use of breadth First Search.

Q.21 Write an algorithm for FIFO branch and bound. [JNTU : Part B, Marks 5]

Ans. :

```
Algorithm FIFOBB()
//tr is a state space tree and x be the node in the tr
//E represents the E-node
//Initialize the list of live nodes to empty
{
   if( tr is answer node) then
   {
      u:=min(cost[tr],(upper(tr)+E))
      write(tr);//output the answer node
      return;
   }
   E←tr //set the E-node
   repeat
   {
   for(each child x of E) do
   {
      if(x is answer node) then
      {
         output the path from x to root;
         return;
      } //the new live node x will be added in the list
      //of live nodes
   Insert_Q(x);
   x.parent← E://pointing the path from x to root;
      if((x is answer node) AND cost(x)<u) then
      {
         u←min(cost[tr],(upper(tr)+E))
      }
   }
   if(no more live nodes) then
   {
      write("No more Live nodes now");
      write("least Cost is =",u);
      return;
   }          // E points to current E-node
   E← Del_Q(); // deletes the least cost node from Q
                  // to set next E-node
   } until(false);
}
```

In above algorithm if x is in tr then c(x) be the minimum cost answer node in *tr*. The computation of u value is done. We always choose the minimum values of cost of the children generated from the current E node.

The algorithm uses two functions Del_Q and () and Insert_Q() function to delete or add the live node from the list of live nodes stored in queue. Using above algorithm we can obtain path from answer node to root.

Q.22 Explain the principles of FIFO branch and bound. [JNTU : Part B, Dec.-11, Marks 8]

Ans. : Refer Q.20 and Q.21.

Q.23 What is LIFO branch and bound search ?

Ans. : (1) LIFO stands for last in first out branch and bound technique. (2) Children of E node are inserted in a stack. (3) The search is made using the Depth First Search Technique.

Q.24 Explain the LIFO branch and bound algorithm.

Ans. :

```
Algorithm LIFOBB()
//tr is a state space tree and x be the node in the tr
//E represents the E-node
//initialize the list of live nodes to empty
{
    if( tr is answer node) then
        {
        u←min(cost[tr],(upper(tr)+E))
        write(tr);//output the answer node
        return;
        }
        E←tr //set the E-node
        repeat
        {
        for(each child x of E) do
        {
        if(x is answer node) then
        {
        output the path from x to root;
        return;
        }
//the new live node x will be added in the list of live
nodes
   PUSH(x);
   x.parent← E;//pointing the path from x to root;
   if( (x is answer node) AND cost(x)<u) then
      {
         u←min(cost[tr],(upper(tr)+E))
      }
    }
   if(no more live nodes) then
   {
      write("No more Live nodes now");
      write("least Cost is =",u);
      return;
   }
// E points to current E-node
   E← POP();
// deletes the least cost node from stack to set next
E-node
} until(false);
}
```

In above algorithm if x is in tr then c(x) be the minimum cost answer node in tr. The computation of u value is done. We always choose the minimum values of cost of the children generated from the current E node.

The algorithm uses two functions POP () and PUSH() function to delete or add the live node from the list of live nodes stored in stack. Using above algorithm we can obtain path from answer node to root.

Q.25 Explain the principles of LIFO branch and bound. ☞ **[JNTU : Part B, Dec.-11, Marks 8]**

Ans. : Refer Q.23 and Q.24.

END ...

UNIT - V

8 NP-Hard and NP-Complete Problems

8.1 Basic Concepts

Q.1 Explain the following : i) Computational complexity ii) Decision problems. iii) Deterministic and non-deterministic algorithms. iv) Complexity classes. ☞ **[JNTU : Part A, Marks 4]**

Ans. : i) Computational Complexity : The computational problems is an infinite collection of instances with a solution for every instance. The computational complexity problems can be decision problem or functional problem

ii) **Decision Problem :** In computational complexity the solution to the problem can be "yes" or "no" type. Such type of problems are called decision problem.

iii) **Deterministic and Non Deterministic Algorithm :** The problems that can be solved in deterministic polynomial time are called deterministic algorithms. On the other hand, the problems that can be solved in non-deterministic polynomial time are called NP problems.

iv) **Complexity Classes :** The complexity classes is a set of problems of related complexity. It includes function problems, P classes, NP classes, optimization problem.

Q.2 Explain the classes of P and NP. ☞ **[JNTU : Part B, May-09, Marks 8]**

Ans. :

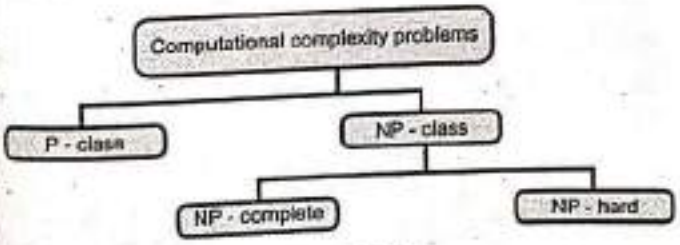


Fig. Q.2.1

i) **P Class Problems :** The problems that can be solved in polynomial time are called P class problems. For example - Searching a key from the list of elements, sorting the elements, All pair shortest path.

ii) **NP Class Problems :** The class of problems that can be solved in non deterministically polynomial time are called NP class problem.

For example - Knapsack problem, Traveling Salesperson Problem.

Q.3 Differentiate between P class and NP Class Problems. ☞ **[JNTU : Part A, Marks 2]**

Ans. :

P Class Problems	NP Class Problems
An algorithm in which for given input the definite output gets generated is called Polynomial time algorithm(P class)	An algorithm is called non deterministically polynomial time algorithm (NP class) when for given input there are more than one paths that the algorithm can follow. Due to which one can not determine which path is to be followed after particular stage.
All the P class problems are basically deterministic.	All the NP class problems are basically non deterministic.
Every problem which is a P class is also in NP class.	Every problem which is in NP is not the P class problem.
P class problems can be solved efficiently.	NP class problems can not be solved efficiently as efficiently as P class problems.
Examples : Binary Search, Bubble Sort	**Examples :** Knapsack problem, Traveling Salesperson problem.

Q.4 List three problems that have polynomial time algorithms. Justify your answer.

[JNTU : Part A, Marks 3]

Ans. : The problems that can be solved in polynomial time are called P - class problems. For example -

1. **Binary search** - In searching an element using binary search method, the list is simply divided at the mid and either left or right sublist is searched for key element. This process is carried out in O(logn).
2. **Evaluation of polynomial** - In a polynomial evaluation we make out the summation of each term of polynomial. The evaluated result is simply an integer. This process is carried out in O(n) time.
3. **Sorting a list** - The elements in a list can be arranged either in ascending or descending order. This procedure is carried out in O(nlogn) time.

This shows that all the above problems can be solved in polynomial time.

Q.5 Explain the differences between decision and optimization problems.

[JNTU : Part B, April-11, Marks 8]

Ans. :

- The decision problem is a problem that can be posed as a yes-no question of the input values. For example - is the given number n is a prime number. The answer for this problem could be either yes or no. Hence is it a decision problem
- The optimization problem asks us to find, among all feasible solutions, one that maximizes or minimizes a given objective. For example – Finding shortest path between two cities is an optimization problem. The possible answer is there exists some from source s to t or there is no path that exists between s and t.
- If one can solve the optimization problem in polynomial time then it is possible to solve the decision problem in polynomial time.
- Similarly by doing binary search on the bound b, one can transform a polynomial answer to a decision version into a polynomial time algorithm for the corresponding optimization problem.
- Normally the decision problems are NP Complete while optimization problems are NP hard.

Q.6 Describe some classic NP problems and why they are important [JNTU : Part B, Dec.-11, Marks 7]

Ans. : • An algorithm is called non deterministic polynomial time algorithm(NP Class) when for given input there are more than one paths that the algorithm can follow. Due to this one can not determine which path is to be followed at particular stage.

Travelling Salesman's Problem (TSP) : This problem can be stated as " Given a set of cities and cost to travel between each pair of cities, determine whether there is a path that visits every city once and returns to the first city. Such that the cost travelled is less".

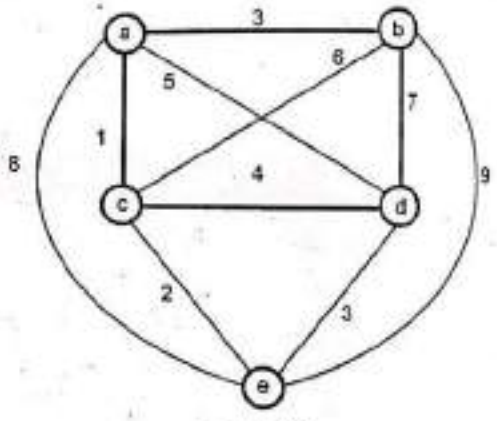


Fig. Q.6.1

For example :

- The tour path will be **a-b-d-e-c-a** and total cost of tour will be **16.**
- This problem is NP problem as there may exist some path with shortest distance between the cities. If you get the solution by applying certain algorithm then Travelling Salesman problem is **NPComplete Problem**. If we get no solution at all by applying an algorithm then the travelling salesman problem belongs to **NP hard class.**
- The NP problems are important because it contains the class of problems that does can not be solved in polynomial time. For example – Knapsack problem, Traveling Salesman Problem, Sum of subset problem. To get the solution of these problems the only Polynomial time class of problems is not sufficient.

Q.7 Write about tractable and intractable problems. [JNTU : Part B, Dec.-11, Marks 8]

Ans. : Tractable Problem: a problem that is solvable by a polynomial-time algorithm. The upper bound is polynomial. For example

1. Searching an unordered list
2. Searching an ordered list
3. Sorting a list
4. Multiplication of integers (even though there's a gap)
5. Finding a minimum spanning tree in a graph

Intractable Problem : a problem that cannot be solved by a polynomial-time algorithm. The lower bound is exponential.

1. Towers of Hanoi : we can prove that any algorithm that solves this problem must have a worst-case running time that is at least $2n - 1$.
2. List all permutations (all possible orderings) of n numbers

Q.8 Explain about optimization problem with example. [JNTU : Part B, Dec.-12, Marks 8]

Ans. : The optimization problem asks us to find, among all feasible solutions, one that maximizes or minimizes a given objective. For example – Finding shortest path between two cities is an optimization problem. The possible answer is there exists some from source s to t or there is no path that exists between s and t.

Shortest Path Problem – Refer Q.6.

8.2 Non-deterministic Algorithms

Q.9 Distinguish between deterministic and non deterministic algorithm. [JNTU : Part A, Nov.-16, Marks 3, May-17, Marks 5]

Ans. : • The algorithm in which every operation is uniquely defined is called deterministic algorithm.

• The algorithm in which every operation may not have unique result, rather there can be specified set of possibilities for every operation. Such an algorithm is called non-deterministic algorithm. Non-deterministic means that no particular rule is followed to make the guess.

Q.10 Write a non-deterministic algorithm for searching element. [JNTU : Part B, Marks 5]

Ans. : • The non-deterministic algorithm is a two stage algorithm -

i) **Non-deterministic (Guessing) stage** - Generate an arbitrary string that can be thought of as a candidate solution.

ii) **Deterministic ("Verification") stage** - In this stage it takes as input the candidate solution and the instance to the problem and returns *yes* if the candidate solution represents actual solution.

```
Algorithm Non_Determin()
// A[1:n] is a set of elements
// we have to determine the index i of A at which
element x is
//located.
{
  // The following for-loop is the guessing stage
     for i=1 to n do
       A[i] := choose(i);

  // Next is the verification(deterministic) stage
   if (A[i] = x) then
   {
      write(i);
      success();
   }
   write(0);
   fail();
}
```

In the above given non-deterministic algorithm there are three functions used -

1) **Choose** - Arbitrarily chooses one of the element from given input set.
2) **Fail** - Indicates the unsuccessful completion.
3) **Success** - Indicates successful completion.

The algorithm is of non-deterministic complexity O(1), when A is not ordered then the deterministic search algorithm has a complexity $\Omega(n)$.

Q.11 Give the non-deterministic algorithm for sorting elements in non decreasing order. [JNTU : Part B, Marks 5]

Ans. :

```
Algorithm Non_DSort(A,n)
// A[1:n] is an array that stores n elements which are
positive integers B[1:n] be an auxiliary
// array in which elements are put at appropriate
positions
{
// guessing stage
   for i←1 to n do
      B[i]←0;//initialize B to 0
   for i←1 to n do
   {
      j←choose(1,n)
//select any element from the input set
      if(B[j]!=0) then
         fail();
      B[j]←A[i];
   }
//verification stage
   for i←1 to n–1 do
      if(B[i]>B[i+1]) then
         fail();// not sorted elements
      write(B[1:n]);// print the sorted list of elements
      success();
}
```

The time required by a non-deterministic algorithm with some input set is minimum number of steps needed to reach to a successful completion if there are multiple choices from input set leading to successful completion. In short, a non-deterministic algorithm is of complexity O(f(n)) where n is the total size of input.

Q.12 Write a non-deterministic Knapsack algorithm.
☞ [JNTU : Part B, Marks 5]

Ans. : The *0/1 Knapsack Decision Problem* is to determine, for a given profit P, whether it is possible to load the knapsack so as to keep the total weight no greater than m, while making the total profit at least equal to Pt. This problem has the same parameters as the 0-1 Knapsack Optimization Problem (as discussed in earlier chapters) plus the additional parameter Pt.

The non deterministic algorithm for 0/1 knapsack problem is as given below -

```
Algorithm Non_DKnaps()
//p[1:n] is a profit each object i where 1 ≤ i ≤ n
//w[1:n] is the weight of each object i where 1 ≤ i ≤ n
//Profit and Wt represent the current total profit and
weight
// m is the capacity of knapsack
{
//initially no item is selected
 Wt:=0;
Profit:=0;
// guessing stage
   for i:=1 to n do
   {
      x[i]:=choose(0,1);
      Wt:=Wt+x[i]*w[i];
      Profit:=Profit+x[i]*p[i];
   }
//verification stage
// selected items exceed the capacity or less profit is
earned

   if((Wt>m) or (Profit<pt)) then
      fail();
   else
      success();//maximum profit earned
}
```

Q.13 Explain the satisfiability problem.
☞ [JNTU : Part B, May-09, Marks 8, Dec.-11, Marks 15]

OR Prove CNF satisfiability of AND/OR Graph decision problem. ☞ [JNTU : Part B, Nov.-16, Marks 5]

OR Write short note on 3-SAT problem.
☞ [JNTU : Part B, May-17, Marks 5]

Ans. : 1. CNF - SAT problem

This problem is based on Boolean formula. The Boolean formula has various Boolean operations such as OR(+), AND (·) and NOT. There are some notations such as → (means implies) and ↔ (means if and only if).

A Boolean formula is in **Conjuctive Normal Form (CNF)** if it is formed as collection of subexpressions. These subexpressions are called clauses.

For example

$(\bar{a} + b + d + \bar{g})\ (c + \bar{e})\ (\bar{b} + d + \bar{f} + h)\ (a + c + e + \bar{h})$

This formula evaluates to 1 if b, c, d are 1.

The CNF-SAT is a problem which takes Boolean formula in CNF form and checks whether any assignment is there to Boolean values so that formula evaluates to 1.

Theorem : CNF-SAT is in NP complete.

Proof : Let S be the Boolean formula for which we can construct a simple non-deterministic algorithm which can guess the values of variables in Boolean formula and then evaluates each clause of S. If all the clauses evaluate S to 1 then S is satisfied. Thus CNF-SAT is in NP-complete.

2. A 3SAT problem

A 3SAT problem is a problem which takes a Boolean formula S in CNF form with each clause having **exactly three literals** and check whether S is satisfied or not. [Note that CNF means each literal is ORed to form a clause, and each clause is ANDed to form Boolean formula S].

Following formula is an instance of 3SAT problem :

$(\bar{a} + b + \bar{g})\ (c + \bar{e} + f)\ (\bar{b} + d + \bar{f})\ (a + e + \bar{h})$

8.3 NP-hard and NP-complete Classes

Q.14 What is the relationship between NP hard and NP complete ? **[JNTU : Part A, Nov.-16, Marks 2]**

Ans. : A problem (a language) is said to NP-hard if every problem in NP can be poly time reduced to it.

Q.15 Explain the P, NP, NP-hard and NP-complete classes. Give relationship between them.
[JNTU : Part B, May-12, Marks 15, Dec.-11, Marks 8, May-17, Marks 5]

Ans. :
- Problems which are known to lie in P are often called as *tractable*. Problems which lie outside of P are often termed as *intractable*. Thus, the question of whether P = NP or P ≠ NP is the same as that of asking whether there exist problems in NP which are intractable or not.

The relationship between P and NP is depicted by Fig. Q.15.1.

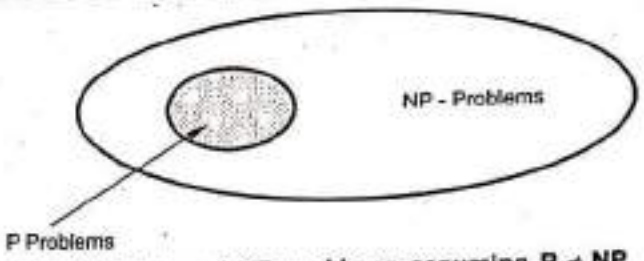


Fig. Q.15.1 P and NP problems assuming P ≠ NP

- We don't know if P = NP. However in 1971 S. A. Cook proved that a particular NP problem known as SAT(Satisfiability of sets of Boolean clauses) has the property that, if it is solvable in polynomial time, so are all NP problems. This is called a "NP-complete" problem.
- Let A and B are two problems then problem A reduces to B if and only if there is a way to solve A by deterministic polynomial time algorithm using a deterministic algorithm that solves B in polynomial time.
- A reduces to B can be denoted as $A \propto B$. In other words we can say that if there exists any polynomial time algorithm which solves B then we can solve A in polynomial time. We can also state that if $A \propto B$ and $B \propto C$ then $A \propto C$.
- A NP problem such that, if it is in P, then NP = P. If a (not necessarily NP) problem has this same property then it is called "NP-hard". Thus the class of NP-complete problem is the intersection of the NP and NP-hard classes.

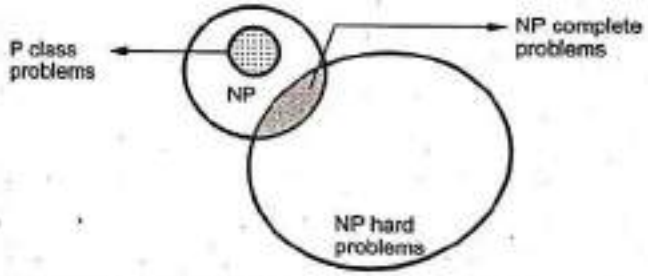


Fig. Q.15.2 Relationship between P, NP, NP-complete and NP-hard problems

- Normally the decision problems are NP-complete but optimization problems are NP-hard. However if problem A is a decision problem and B is optimization problem then it is possible that $A \propto B$. For instance the Knapsack decision problem can be Knapsack optimization problem.

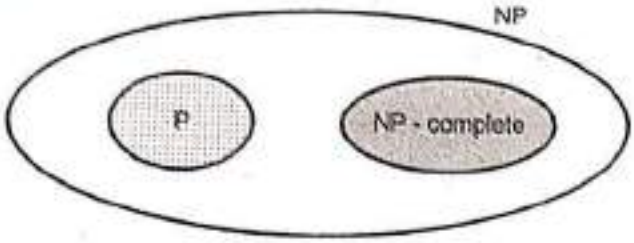


Fig. Q.15.3

- There are some NP-hard problems that are not NP-complete. For example *halting problem*. The halting problem states that : "Is it possible to determine whether an algorithm will ever halt or enter in a loop on certain input?"
- Two problems P and Q are said to be polynomially equivalent if and only if $P \propto Q$ and $Q \propto P$.

Q.16 Explain about different types of NP problem.
☞ [JNTU : Part B, May-12, Marks 8]

Ans. : Refer Q.15.

Q.17 Show that HAMILTONIAN CYCLE problem on directed graph is NP complete
☞ [JNTU : Part B, Nov.-16, Marks 5]

Ans. :

Theorem : HAMILTONIAN CYCLE is in NP.

Proof : Let A be some non-deterministic algorithm to which graph G is given as input. The vertices of graph are numbered from 1 to N. We have to call the algorithm recursively in order to get the sequence S. This sequence will have all the vertices without getting repeated. The vertex from which the sequence starts must be ended at the end. This check on the sequence S must be made in polynomial time n. Now if there is a Hamiltonian cycle in the graph then algorithm will output "yes". Similarly if we get output of algorithm as "yes" then we could guess the cycle in G with every vertex appearing exactly once and the first visited vertex getting visited at the last. That means A non-deterministically accepts the language HAMILTONIAN CYCLE. It is therefore proved that HAMILTONIAN CYCLE is in NP.

Q.18 Prove that vertex cover problem is NP Complete.
☞ [JNTU : Part B, Marks 5]

Ans. : To prove that vertex cover is NP-complete we will apply reduction technique. That means reduce 3-SAT to vertex cover. As we know 3-SAT to basically a Boolean expression s containing 3 variables in each clause and value of S is true.

To prove this theorem consider S be some Boolean formula in CNF (conjunctive Normal Form) having 3 variables in each clause no for each variable a we will create,

Fig. Q.18.1

If the clause is a + b + c we will create a triangle (as there are 3 - variables)

Using 3-SAT we will build a graph as follows for given expression.

$(a+b+c)\ (a+b+\bar{c})\ (\bar{a}+c+\bar{d})$

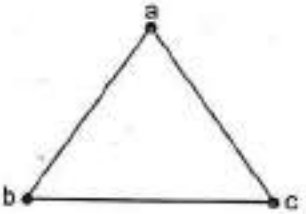


Fig. Q.18.2

The graph has vertex cover of size K = n + 2m, where n is number of variables in 3-SAT, m is number of clauses in CNF, K is total number of vertices that are present in the set of vertex cover.

For a clause (a + b + c) we can build a graph as :

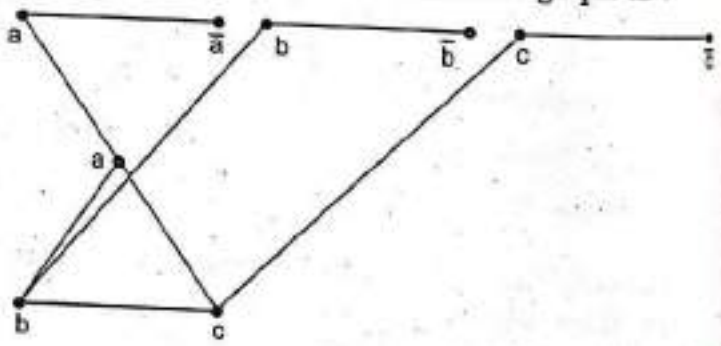


Fig.Q.18.3

Thus for given expression

$(a+b+c)\ (a+b+\bar{c})\ (\bar{a}+c+\bar{d})$ we get following graph.

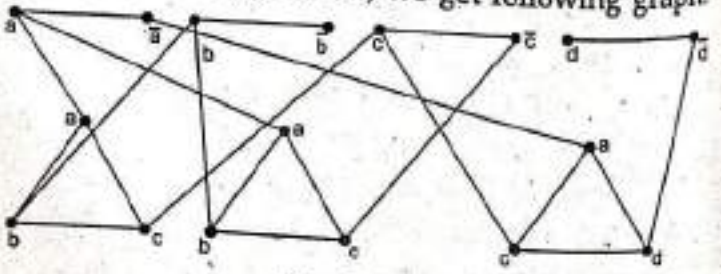


Fig. Q.18.4

Here n = Number of variable = 4

m = Number of clauses = 3

K = n + 2m = 4 + 2(3) = 10

In vertex cover we get K = 10 vertices which cover all the vertices. The vertex cover is shown by following graph in which the covering vertices are rounded.

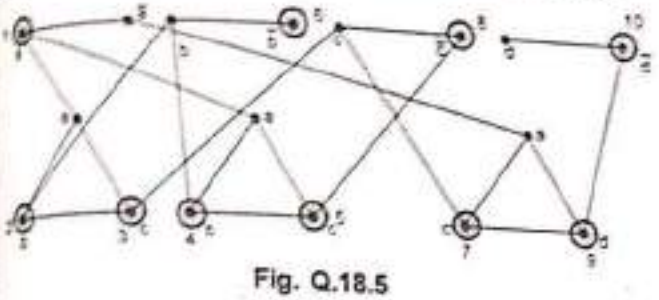


Fig. Q.18.5

Thus K = n + 2m is proved as K = 10. This is how 3-SAT can be reduced to vertex cover. The 3-SAT is NP-complete. Hence vertex is NP-complete is proved.

Thus K = n + 2m is proved as K = 10. This is how 3-SAT can be reduced to vertex cover. The 3-SAT is NP-complete. Hence vertex is NP-complete is proved.

8.4 NP-hard Problems

Q.19 Briefly explain the concepts of the NP-hard and NP-complete ?

[JNTU : Part B, May-12, Marks 7, Nov.-16, Marks 5, Dec.-12, Marks 7]

OR What are differences between NP-hard and NP-complete classes ? Explain with examples.

[JNTU : Part B, May-12, Marks 15]

Ans. : The NP complete has a property that it can be solved in polynomial time if and only if all other NP complete problems can also be solved in polynomial time.

If an NP-hard problem can be solved in polynomial time then all the NP complete problems can be solved in polynomial time.

All the NP complete problems are NP-hard but there are some NP hard problems that are not known to be NP complete.

Example of NP Complete problem – Traveling Salesman problem

Example of NP Hard Problem – Halting Problem

Q.20 What is meant by halting problem ? Explain with an example. **[JNTU : Part B, May-09, Marks 8]**

Ans. : The halting problem can be stated as follows - "It is not possible to determine for an arbitrary deterministic algorithm A and input I whether algorithm A with input I ever terminates or enters in an infinite loop." This problem is undecidable problem. Hence there is no algorithm to solve this problem. To prove that halting problem is NP hard which is not in NP, we will make use of satisfiability.

Construct an algorithm A with an input w. The input w is a propositional formula with n variables. Then algorithm A tries all 2^n possible truth assignments, and determines whether w is satisfiable. If w is satisfiable then A halts successfully but if it is not then A enters in an infinite loop. If we had a polynomial time algorithm for the halting problem then we could solve the satisfiability problem in polynomial time with the help of algorithm A and input w. This proves that halting problem is NP hard that is not in NP.

Q.21 Write about graph colouring problem and subset sum probelm. Are they NP problems. If yes, Justify your answer to include them in to NP problems. **[JNTU : Part B, Dec.-11, Marks 15]**

Ans. : **Graph coloring problem :** Refer Q.23.

Subset sum problem : Refer Q.17.

To show that graph coloring problem is NP hard. The proof will be -

Proof : In this proof we have to prove if the formula F for CNF (with 3 literals) is satisfiable then graph G is n + 1 colorable.

Let, F be a CNF formula with 3 literals per clause $C_1, C_2 \ldots C_m$ be some clauses.

Let, x_i be the variable in F where $1 \le i \le n$.

L.H.S. → i.e. Formula F is satisfiable

1. Let, $f(y_i) = i$
2. If x_i = True then $f(x_i) = i$, $f(\overline{x}_i) = n + 1$ else $f(x_i) = n + 1$, $f(\overline{x}_i) = i$
3. If x_i is in C_j and $\overline{x}_i$ = True then $f(C_j) = f(x_i)$

If $\bar{x}_i$ is in C_j and $\bar{x}_i$ = True then $f(C_j) = f(\bar{x}_i)$

R.H.S. → i.e. G is n + 1 colorable

1. y_i is assigned with color i.
2. $f(x_i) \neq f(\bar{x}_i)$. That means either $f(x_i) = i$ and $f(\bar{x}_i) = n + 1$ or $f(x_i) = n + 1$ and $f(\bar{x}_i) = i$.
3. If $f(C_j) = i = f(x_i)$, then assign x_i = T. If $f(C_j) = i = f(\bar{x}_i)$ then assign $\bar{x}_i$ = T.
4. For atleast one x_i, the x_i and $\bar{x}_i$ are not in C_j. The $f(C_j) \neq n + 1$
5. If $f(C_j) = i = f(x_i)$ then $(C_j, x_i) \neq E$. If x_i is in C_j, then C_j is true. If $f(C_j) = i = f(\bar{x}_i)$ then $(C_j, \bar{x}_i) \neq E$. If $\bar{x}_i$ is in C_j then C_j is true.

This proves that if F is satisfiable then G is n + 1 colorable.

As the 3 SAT problem is NP hard the graph coloring is also NP Hard.

To, show that the sum of subsets problem is NP hard, given that exact cover problem is NP hard.

Proof :

- The exact cover problem is NP hard.
- In this problem there exist a collection of sets $F = \{S_1, S_2, ... S_k\}$ let T be the subset. We have to determine whether $T \subseteq F$. Such that

$$\bigcup_{S_i \in T} S_i = \bigcup_{S_i \in F} S_i = \{u_1, u_2, u_3, ..., u_n\}$$

- Let us build the sum of subset problem A. Here $A = \{a_1, a_2, ... a_k\}$

where $a_j = \sum_{1 \le i \le n} \varepsilon_{ji} (k+1)^{i-1}$

- The $\varepsilon_{ji} = 1$ if $u_i \in S_j$ and $\varepsilon_{ji} = 0$ otherwise.
- Let M be the sum.

$$M = \sum_{0 \le i < n} (k+1)^i = \left((k+1)^n - 1\right) / k$$

- Thus F has exact cover if and only if $A = \{a_1, a_2, ..., a_k\}$ has a subset with sum M. But M and A can be constructed in polynomial time from F, the exact cover problem α sub of subsets.
- As exact cover problem is NP Hard, sum of subsets problem is also NP hard.

Q.22 Explain the strategy to prove that a problem is NP hard. [JNTU : May-09, Marks 8]

Ans. : The strategy to prove that a problem is NP hard is as given below -

1. Pick a problem P_1. The problem P_1 is already known to be NP hard.
2. Show the process of obtaining an instance I' of P_2 from any instance I of P_1. This instance can be obtained in polynomial deterministic time.

This newly obtained instance I' should determine the solution of instance I from P_1.

3. From step 1 and 2 conclude that $P_1 \propto P_2$.
4. From above steps conclude that P_2 is NP-hard.

Q.23 Specify any one example of NP-hard problem. Also mention that why it is NP hard ? [JNTU : Part B, Marks 5]

Ans. : NP - Hard Problem - Hamiltonian cycle problem.

If we want to find Hamiltonian cycle with length less than k then this is a determinist problem. It is NP complete as well. Because it is easy to determine Hamiltonian cycle with length less than k. But (optimization version) i.e. "what is the shortest Hamiltonian cycle ?" is NP - Hard Problem. Because it is not easy to determine if cycle obtained is shortest or not.

8.5 Cook's Theorem

Q.24 Explain about Cook's theorem. [JNTU : Part B, May-09, Marks 8, May-12, Marks 7, Nov.-16, Marks 5]

Ans. : Any instance of Boolean satisfiability problem is a boolean expression in which Boolean variables are combined using Boolean operators. An expression is satisfiable if its value results to be true on some assignments of Boolean variables.

The Boolean satisfiability problem is in NP. This is because a non-deterministic algorithm can guess an assignment of truth values of variables. This algorithm can also determine the value of expression for corresponding assignement and can accept if entire expression is true .

The algorithm is composed of -

- Input tape wherein tape is divided in finite number of cells.
- The read/write head which reads each symbol from tape.
- Each cell contain only one symbol at a time.
- Computation is performed in number of states.
- The algorithm terminates when it reaches to accept state.

The conjunction clauses for Boolean expression are given in following table

Clauses	Meaning	Time
T_{ij0}	Cell i of input tape contains symbol j.	O(P(n))
Q_{s0}	Initial state.	O(1)
H_{00}	Initial position of tape head.	O(1)
$T_{ijk} = T_{ij(k+1)} \vee H_{ik}$	Tape remains unchanged unless written.	$O(P(n^2))$
$Q_{qk} \rightarrow \neg Q_{qk}$	One state at a time.	O(P(n))
$H_{ik} \rightarrow \neg H_{ik}$	One read/ write head position at a time.	$O(P(n^2))$
$T_{ijk} \rightarrow \neg T_{ijk}$	One symbol per tape cell at a time.	$O(P(n^2))$

Note that H denotes head, Q denotes states and T denotes tape. The disjunction clause for this algorithm can be written as :

Clause	Meaning	Time
$(H_{ik} \wedge Q_{qk} \wedge T_{ijk}) \rightarrow (H_{(i+j)(k+1)} \wedge Q_{q(K+1)} \wedge T_{ij(K+1)})$	Possible transitions	$O(P(n^2))$
Disjunction of all clauses Q_f	Moving to accept state	O(1)

If B is satisfiable, then there is an accepting state in the algorithm.

Thus the proof shows that boolean satisfiability problem can be solved in polynomial time. Hence all problems in NP could be solved in polynomial time. And hence the complexity class NP could be equal to P.

C) p[i] = j but not p[j] = i D) p[i] = j or p[j] = i

Q.10 Time taken to perform (n – 1) UNIONS is _____

A) $O(n^2)$ B) $O(n^3)$ C) O(n) D) O(1)

II **Fill in the Blanks**

Q.11 Time complexity of MST is _____.

Q.12 AOE means _____.

Q.13 A path from s to a node x outside Y is called special if every intermediary node on the path belongs to Y called _____ problem.

Q.14 The function _____ iff there exist positive constants c and n_0 such that $f(n) \leq c^* g(n)$ for all n, $n \geq n_0$

Q.15 _____ notation provides an asymptotic lower bound on a function.

Q.16 Merging 4 sorted files containing 50, 10, 25 and 15 records will take _____ time.

Q.17 _____ also called partition exchange sort.

Q.18 A graph G is biconnected if and only if it contains no _____.

Q.19 _____ rule is used for FIND algorithm.

Q.20 Total time of optimal merge patterns is _____.

NOVEMBER - 2016 [114CS]
Design and Analysis of Algorithms

Question Paper
II - II [CSE] B.Tech.

Time : 3 Hours] **[Maximum Marks : 75**

Note : This equation paper contains two parts A and B.

Part A is compulsory which carries 25 marks. Answer all questions in Part A. Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

PART - A (25 Marks)

Q.1 a) Define the time complexity. [2]

b) List out the reasons for the difficulties that one faces while determining the lower bound. [3]

c) Write an algorithm of simple union. [3]

d) What are the applications of game tree ? [3]

e) Write an algorithm of greedy knapsack. [2]

f) State the principle of optimality. [3]

g) Define state space tree. [2]

h) Write the control abstraction algorithm for LC search. [3]

i) What is the relation between NP-hard and NP-complete ? [2]

j) Distinguish between deterministic and non deterministic algorithm. [3]

PART - B (50 Marks)

Q.2 a) *Trace the quick sort algorithm to sort the list C, O, L, L, E, G, E in alphabetical order.*

b) *Solve the following recurrence :* [5 + 5]

$T(n) = 4T(n/2) + n$, where $n \geq 1$ and is a power of 2

OR

Q.3 a) *Write the non-recursive algorithm for finding the Fibonacci sequence and define its time complexity.*

b) *Consider the following recurrence equation :* [5 + 5]

$$T(n): T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n-1)+n^n & \text{otherwise} \end{cases}$$

Q.4 a) *Explain the graph traversal with an example.*

b) *Write an algorithm for AND/OR graphs.* [5 + 5]

OR

Q.5 a) *Write a non recursive algorithm of post order tree traversal.*

b) *Differentiate between BFS and DFS.* [5 + 5]

Q.6 *Write an algorithm of prim's minimum cost spanning tree.* [10]

OR

Q.7 *Consider 4 elements $a1 < a2 < a3 < a4$ with $q(0) = \frac{1}{8}$, $q(1) = \frac{1}{16}$, $q(2) = q(3) = q(4) = \frac{1}{16}$: $p(1) = \frac{1}{4} = p(2) = \frac{1}{8}$, $p(3) = p(4) = \frac{1}{16}$. Construct the table of values of W(i, j), R(i, j) and C(i, j) computed by the algorithm to compute the roots of optimal sub trees.* [10]

Q.8 *Draw the portion of the state space tree generated by LC branch and bound for an instance n = 4, (P1, P2, P3, P4) = (10, 10, 12, 18), (w1, w2, w3, w4) = (2, 4, 6, 9) and m = 15.* [10]

OR

Q.9 a) *Explain the 4-queen problem using backtracking.*

b) *Draw the state space tree for m-coloring graph.* [5 + 5]

Q.10 a) *Show that the HAMILTONIAN_CYCLE problem on directed graphs is NP-complete.*

b) *State the cook's theorem. What is the significance of this theorem ?* [5 + 5]

OR

Q.11 a) *Prove that CNF satisfiability of AND/OR graph decision problem.*

b) *Explain the classes of NP-hard and NP-complete.* [5 + 5]

Fill in the Blanks

Q.11 A path in G which contains every vertex of G once and only once is _____. [1]

Q.12 The running time of merge sort can be recursively represented by _____. [1]

Q.13 Any vertex is strongly connected to itself is called _____. [1]

Q.14 By using potential method, amortized cost = actual cost + _____. [1]

Q.15 _____ mean to computing time exponential. [1]

Q.16 Quick sort running time depends on the _____. [1]

Q.17 _____ Approach is divide the problem into sub problems and then each sub problem solved independently. [1]

Q.18 Worst case time complexity of dynamic array is _____. [1]

Q.19 Recurrence relation of tower Hanoi with n disks _____. [1]

Q.20 A graph with no edges known as empty graph. Empty graph is also known as _____. [1]

MAY - 2017 [124CB][114CS]
Design and Analysis of Algorithms

Question Paper
II - II [CSE] B.Tech.

Time : 3 Hours] [Maximum Marks : 75

PART - A (25 Marks)

Q.1 a) Define order of growth. [2]

b) If $f(n) = 5n^2 + 6n + 4$ then prove that $f(n)$ is $O(n^2)$. [3]

c) Define a spanning tree and minimum spanning tree. [2]

d) Define articulation point. [3]

e) Define greedy method. [2]

f) State the principle of optimality. [3]

g) List the application of Backtracking. [2]

h) Define E - node. [3]

i) Define class P. [2]

j) Explain briefly about optimization problem. [3]

PART - B (50 Marks)

Q.2 a) Write the pseudo code that input of n integers and output them in non decreasing order.

b) Describe the Master's theorem. Solve the following recurrence relations by using Master's theorem.

i) $T(n) = 4\ T(n/2) + n$ ii) $T(n) = 2T(n/2) + n \log n$ [5 + 5]

OR

Q.3 a) Define recurrence equation ? Find the time complexity of merge sort from recurrence relation using substitution method.

b) Write the pseudo code for binary search and analyze the time complexity. [5 + 5]

Q.4 a) *Compare and contrast BFS and DFS.*

b) *Define strongly connected components. Explain the properties of strongly connected components.* [5+5]

OR

Q.5 a) *Discuss about various binary tree traversal methods with example.*

b) *Differentiate greedy and dynamic programming.* [5+5]

Q.6 a) *Discuss about fractional knapsack problem. Consider the following instance of knapsack problem n = 3, m = 20, profits* $(p_1, p_2, p_3) = (25, 24, 15)$ *and weights* $(w_1, w_2, w_3) = (18, 15, 10)$*. Obtain the optimal solution using greedy approach.*

b) *Compute all pair shortest path for following graph shown in Fig. 1.* [5+5]

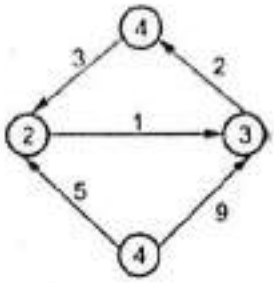


Fig. 1

OR

Q.7 a) *Write the pseudo code for Dijkstra's algorithm for single source shortest path problem.*

b) *Describe traveling sales person problem. Find the minimum cost tour for the following graph using dynamic programming. Costs of the edges are given by matrix shown in Fig. 2.* [5+5]

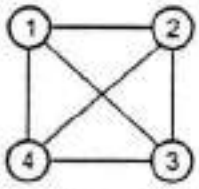


Fig. 2

$$\begin{vmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{vmatrix}$$

Q.8 *What is graph coloring problem ? Describe the back tracking technique to m - coloring with following planar graph shown in Fig. 3.* [10]

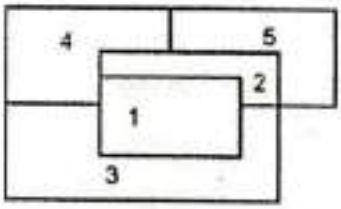


Fig. 3

OR

Q.9 *Write about Hamiltonian cycle. Draw portion state space tree for the following graph shown in Fig. 4.* [10]

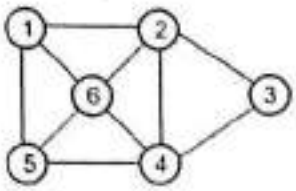


Fig. 4

Q.10 a) *Write short notes on 3 - SAT problem.*

b) *Briefly explain deterministic and non deterministic algorithms with example.* [5 + 5]

OR

Q.11 a) *Describe about clique problem.*

b) *Give the relation between NP Hard and NP Complete.* [5 + 5]

END...

DECEMBER - 2018 [135AF] (R16)
Design and Analysis of Algorithms

Solved Paper
B.Tech., III - I [CSE/IT]

Time : 3 Hours] [Maximum Marks : 75

Note : This question paper contains two parts A and B.

Part A is compulsory which carries 25 marks. Answer all questions in Part A. Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

PART - A **(25 Marks)**

Q.1 a) *Write an algorithm to find the number of digits in the binary representation of a positive decimal integer.* [2]

Ans. : Algorithm Binary(n)

```
count := 1;
while(n>1)
{
    count := count+1;
    n := floor(n/2);
}
return count ;
```

b) *How can we measure an algorithm's running time ?* **(Refer Q.12 of Chapter - 1)** [3]

c) *What is a set ? List the operations that can be performed on it.* **(Refer Q.8 and Q.9 of Chapter - 3)** [2]

d) *Give brief note on graph coloring.* **(Refer Q.23 of Chapter - 6)** [3]

e) *State the Job - Sequencing Deadline Problem.* **(Refer Q.7 of Chapter - 4)** [2]

f) *Find an optimal solution to the knapsack instance n = 4 objects and the capacity of knapsack m = 15, profits (10, 5, 7, 11) and weight are (3, 4, 3, 5).* [3]

Ans. : We will solve this problem using Greedy method. First of all let us arrange the objects in decreasing order of pi/wi.

x	Pi	wi	pi/wi
1	10	3	10/3 = 3.33
3	7	3	7/3 = 2.33
4	11	5	11/5 = 2.2
2	5	4	5/4 = 1.25

Capacity of the knapsack m = 15

Number of objects = 4

Step 1 : Select object 1 with profit 10 and weight as 3

∴ Total profit = 10, Total weight = 3

Step 2 : Select object 3 with profit 7 and weight as 3

∴ Total profit = 17, Total weight = 6

Step 3 : Select object 4 with profit 11 and weight as 5

∴ Total profit = 28, Total weight = 11

Step 4 : Select object 2 with profit 5 and weight as 4

∴ Total profit = 33, Total weight = 15

As Total weight is exactly equal to capacity of knapsack, the **solution is (1,1,1,1) with profit = 33**

g) *What is Travelling Sales Man Problem ?* **(Refer important points to remember of section 7.4)** [2]

h) *Give the statement of reliability design problem.* **(Refer Q.31 of Chapter - 5)** [3]

i) *State the methodology of Branch and Bound.* **(Refer section 7.1)** [2]

j) *Define bounding function ? Give the statement of 0/1 Knapsack FIFO BB.*
(Refer Q. 7 and important points to remember of section 7.5) [3]

PART B **(50 Marks)**

Q.2 a) *Explain recursive binary search algorithm with suitable examples.* **(Refer Q.5 of Chapter - 2)**

b) *Distinguish between merge sort and quick sort.* [5 + 5]

Ans. :

Sr. No.	Merge sort	Quick sort
1.	Sorts the elements by dividing the array into two, repeatedly until one element is left.	Sorts the element by comparing each element with pivot.
2.	It is suitable for any size of the array.	It is suitable for small array
3.	Works with constant speed.	Works efficiently for small array.
4.	It requires more space.	It requires minimum space.
5.	O(nlogn)	The worst case time complexity is $O(n^2)$

OR

Q.3 a) *What is stable sorting method ? Is merge sort a stable sorting method ? Justify your answer.*

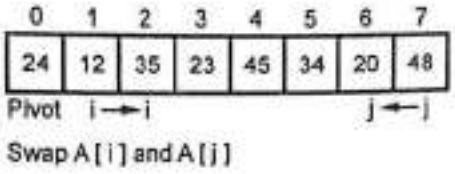
Ans. : A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.

Yes, merge sort is a stable sorting algorithm. Because in this algorithm the sorting is done by partitioning the array into two halves. When merging two halves, we use "Left Element <= Right Element" so that there is a favor of left-half values over right-half values, if they are equal.

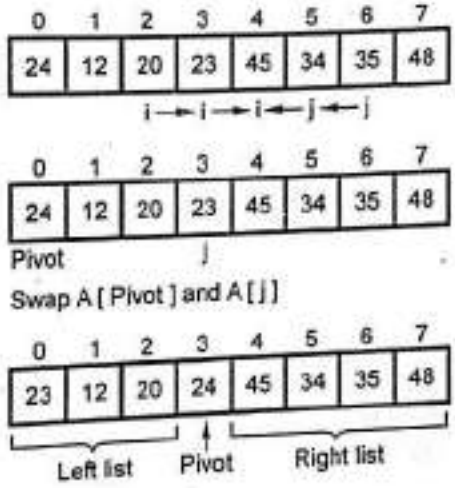
b) *Explain partition exchange sort algorithm and trace this algorithm for n = 8 elements : 24, 12, 35, 23, 45, 34, 20, 48.* [5 + 5]

Ans. : Let us arrange the elements in an array

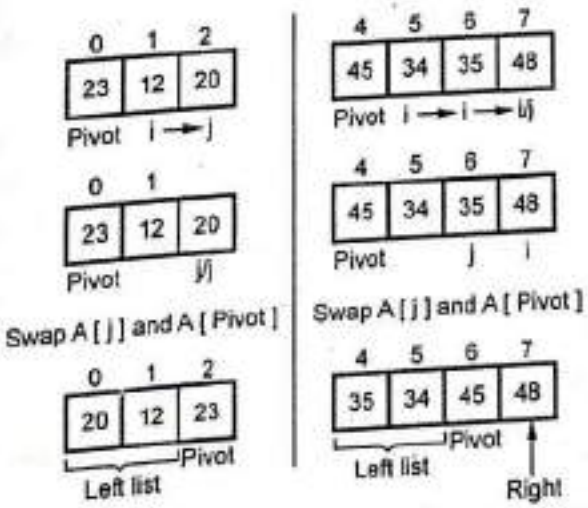
Step 1 :



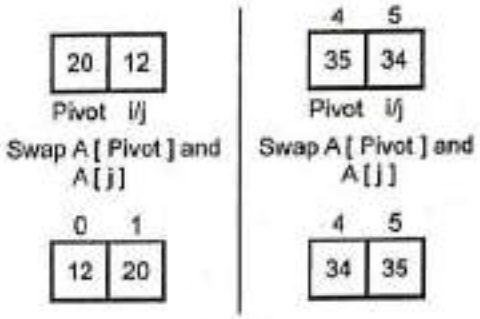
Step 2 :



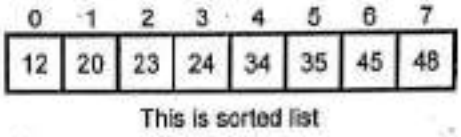
Step 3 :



Step 4 :



Step 5 : Combining all lists from above steps we get



Q.4 *Write and explain the algorithm of Bi connected components with an example.* **(Refer Q.35 of Chapter - 3)** [10]

OR

Q.5 *Give the solution to the 8-queens problem using backtracking.* **(Refer Q.12 of Chapter - 6)** [10]

Q.6 *What is minimum cost spanning tree? Explain an algorithm for generating minimum cost spanning tree and list some applications of it.* **(Refer Q.19 and Q.25 of Chapter - 4)** [10]

OR

Q.7 a) *Explain the greedy technique for solving the Job sequencing problem.* **(Refer Q.7 and Q.9 of Chapter - 4)**

b) *Write with an example of Prim's algorithm.* **(Refer Q.21 of Chapter - 4)** [5 + 5]

Q.8 a) *Discuss the time and space complexity of dynamic programming travelling sales person algorithm.*

Ans. : Refer Q.29 of Chapter - 5

The strategy of solving TSP using dynamic programming makes use of finding good subproblem always. For a subset $R \subseteq V$ of the vertices, we can consider the subproblem to be the least path that enters R, visits all the vertices and then leaves. For $R \subseteq V$ and $j \notin R$, define C(R, j) = cost of cheapest path that leaves 1 for a vertex in R, visits all of R exactly once and no others, and then leaves R for j.

Therefore an equivalent value for the traveling salesman problem is C(R = {2, . . . , n}, 1), as we can start the cycle from any vertex arbitrarily. The recursive solution for Cost is

$$\text{Cost}(R, j) \leftarrow \min i \in R \, (dij + \text{Cost}(R - \{i\}, i))$$

We are optimizing over all $i \in R$ for the node in the path prior to j. The cost would be the cost to go from 1 to all the vertices in R – {i} then to i and then to j.

That cost is precisely Cost(R – {i}, i) + dij .

Note that any set $R \subseteq V$ can be expressed by a vector of bits of length n. Thus each subproblem is O(n) time plus the subsubproblem time. As there are $O(n^* 2^n)$ subproblems, the total time complexity is $O(n^2 2^n)$.

Space required is also exponential.

b) *Write an algorithm of matrix chain multiplication.* [5 + 5]

Ans. :

```
Algorithm Matrix_chain(p[0...n])
{
    //Problem Description: This algorithm is to build the table m[i,j] and s[i,j]
    for (i ← 1 to n) do
    m[i,i] ← 0
    for(len ← 2 to n) do
    {   for (i←2 to (n – len+1) ) do
        {
            j ← i+len – 1
            m[i,j]=infinity
            for (k ← i to j – 1) do
            {
                q ← m [i,k] +m[k+1,j]+p[i–1]*p[k]*p[j]
                if (q<m[i,j]) then
                {
                    m[i,j]← q
                    s[i,j] ← k
                }//end of if
            }//end of k's for loop
        }//end of i's for loop
    }//end of len's for loop
    return m[1,n]
}//end of algorithm
```

OR

Q.9 *With the help of suitable example explain the all pairs shortest path problem.* **(Refer Q.25 of Chapter - 5)** [10]

Q.10 a) *Give the 0/1 Knapsack LCBB algorithm.* **(Refer Q.16 of Chapter - 7)**

b) *Differentiate between deterministic and non deterministic algorithm.* **(Refer Q.9 of Chapter - 8)** [5 + 5]

OR

Q.11 *Draw and portion of state space tree generated by LCBB for the 0/1 Knapsack instance :* $n = 5$, $(p_1, P_2, \ldots, p_5) = (10, 15, 6, 8, 4)$ $(w_1, w_2, \ldots, w_5) = (4, 6, 3, 4, 2)$ *and* $m = 12$. *And also find an optimal solution of the same.* [10]

Ans. : We will compute u(x) and c^(x) for each node in a state space tree. The fixed tuple size formulation is considered to construct LCBB space tree. Following formulae will be used to compute u(.) and c^(.).

$$u(x) = -\sum P_i x_i$$

$$c^{\wedge}(x) = u(x) - \left[\frac{m - \text{Current total weight}}{\text{Actual weight of remaining object}}\right] * \left[\begin{matrix}\text{Actual profit of} \\ \text{remaining object}\end{matrix}\right]$$

For $x = 1$ i.e for root node.

$$u(x) = u(1) = -\sum P_i \text{ where } i = 1, 2 \text{ and } 5.$$
$$= -(P_1 + P_2 + P_5)$$
$$= -(10 + 15 + 4)$$
$$= -29$$

As by selecting P_1, P_2 and P_5 we get the total weight $W_1 + W_2 + W_5$ = Actual size of Knapsack. Hence,

$$c\hat{}(x) = c\hat{}(1) = u(1) - 0$$
$$c\hat{}(1) = -29$$

Again for node 2 we select $x_1 = 1$. Hence computation of u(2) is

$$u(2) = -\sum P_i$$
$$= -(P_1 + P_2 + P_5)$$
$$u(2) = -29$$

Hence $c\hat{}(2) = -29.$

For node 3, when $x_1 = 0$. That means we want to find an upper bound u(x) without first object. Then,

$$u(3) = -(P_2 + P_3 + P_5)$$
$$= -(15 + 6 + 4)$$
$$u(3) = -25$$

The total current weight is $W_2 + W_3 + W_5 = 11$.

Hence, $$c\hat{}(3) = u(3) - \left(\frac{m-11}{W_4} * P_4\right)$$

When $x_1 = 0$ and x_2, x_3 and x_5 are selected the only. Remaining object is x_4.

$$= -25 - \left(\frac{1}{4} * 8\right)$$
$$c\hat{}(3) = -27.$$

Continuing in this way the state space tree can be drawn as,

In Fig. 1, the upper is – 29. As node 2 has $c\hat{}(2) = -29$. Hence node 2 becomes an E node. Therefore generate children of node 2. Node 5 has $c\hat{}(5) >$ upper, hence we kill node 5 immediately. Node 4 becomes an E node. The children 6 and 7 of node 4 are generated. Node 6 does not form a feasible solution. Hence node 7 becomes an E-node. Continuing in this fashion the state space tree is generated.

We get the solution to this Knapsack instance at node 10. Hence solution is

$(x_1, x_2, x_3, x_4, x_5) = (1, 1, 0, 0, 1)$.

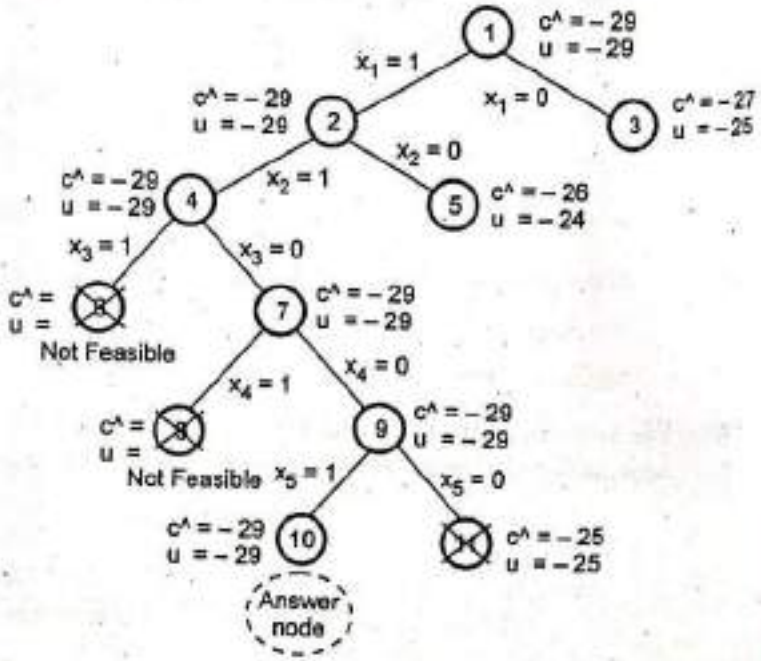


Fig. 1 Portion of state space tree

JUNE - 2019 [135AF] (R16)
Design and Analysis of Algorithms

Solved Paper
B.Tech., III - I [CSE/IT]

Time : 3 Hours] **[Maximum Marks : 75**

Note : This question paper contains two parts A and B.

Part A is compulsory which carries 25 marks. Answer all questions in Part A, Part B consists of 5 Units. Answer any one full question from each unit. Each question carries 10 marks and may have a, b, c as sub questions.

PART - A (25 Marks)

Q.1 a) *In what way a time complexity differs from space complexity.* **(Refer Q.8 and Q.9 of Chapter - 1)** [2]

b) *Give the general plan of divide and conquer algorithms.* **(Refer Q.1 of Chapter - 2)** [3]

c) *Write an algorithm for collasping find.* **(Refer Q.15 of Chapter - 3)** [2]

d) *Define backtracking ? List the applications of backtracking.* **(Refer Q.1 and Q.6 of Chapter - 6)** [3]

e) *What is the importance of knapsack algorithm in our daily life ?* [2]

Ans. : The principle of knapsack problem is useful in our daily life. For example - A student can score more by choosing the questions having more weightage than choosing the questions with less weightage. Thus the student have to solve less number of questions to score well. This strategy is based on knapsack problem.

f) *Write control abstraction of Greedy method ?* **(Refer Q.3 of Chapter - 4)** [3]

g) *What do you mean by dynamic programming.* **(Refer section 5.1)** [2]

h) *Define optimal binary search tree with an example.* **(Refer Q.11 of Chapter - 5)** [3]

i) *State the difference between FIFO and LC Branch and Bound algorithms.* **(Refer Q.4 and Q.20 of Chapter 7)** [2]

j) *Write the control abstraction of least cost Branch and Bound.* **(Refer Q.5 of Chapter - 7)** [3]

PART B (50 Marks)

Q.2 a) *What are the different mathematical notations used for algorithm analysis.* **(Refer Q.19 of Chapter - 1)**

b) *Write Divide and Conquer recursive quick sort algorithm and analyze the algorithm for average time complexity.* **(Refer Q.14 of Chapter - 2)** [10]

OR

Q.3 *Write Randomized algorithm of quick sort.* **(Refer Q.18 of Chapter - 2)** [10]

Q.4 *Write an algorithm to determine the Hamiltonian cycle in a given graph using backtracking.* **(Refer Q.29 of Chapter - 6)** [10]

OR

Q.5 *Explain the AND/OR graph problem with an example.* **(Refer Q.25 of Chapter - 3)** [10]

Q.6 a) *Explain the Knapsack problem with an example.* **(Refer Q.14 of Chapter - 4)**

b) *Write a greedy algorithm for sequencing unit time jobs with deadlines and profits.* **(Refer Q.8 of Chapter - 4)** [10]

OR

Q.7 *State the job - Sequencing with deadlines problem. Find an optimal sequence to the n = 5 Jobs where profits (P1, P2, P3, P4, P5) = (20, 15, 10, 5, 1) and deadlines (d1, d2, d3, d4, d5) = (2, 2, 1, 2, 3)* **(Refer Q.10 of Chapter - 4)** [10]

Q.8 *Draw an Optimal Search Tree for n = 4 identifiers (a1, a2, a3, a4) = (do, if, read, while) P (1 : 4) = (3, 3, 1, 1) and Q (0 : 4) = (2, 3, 1, 1, 1).* **(Refer Q.11 of Chapter - 5)** [10]

OR

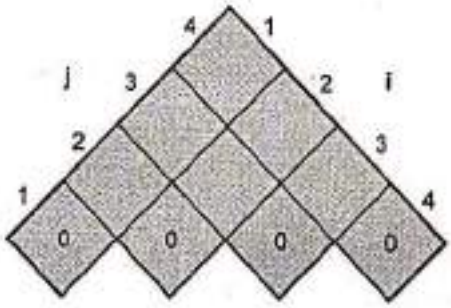
Q.9 *Explain how Matrix- chain Multiplication problem can be solved using dynamic programming with suitable example.* [10]

Ans :

Consider the matrices as : A1 [10 × 100], A2 [100 × 5], A3 [5 × 50] and A4 [50 × 1].

Let, $P_0 = 10$, $P_1 = 100$, $P_2 = 5$, $P_3 = 50$, $P_4 = 1$

Let, m[1, 1], m[2, 2], m[3, 3] and [4, 4] = 0



Now let i = 1, j = 2, k = 1

$$m[i, j] = m[i,k] + m[k+1, j] + P_{i-1} P_k P_j$$

$$= m[1,1] + m[2,2] + P_0 P_1 P_2$$

$$= 0 + 0 + 10 * 100 * 5$$

$$\mathbf{m[1, 2] = 5000}$$

Let i = 2, j = 3, k = 2

$$m[i, j] = m[i,k] + m[k+1, j] + P_{i-1} P_k P_j$$

$$= m[2,2] + m[3,3] + P_1 P_2 P_3$$

$$= 0 + 0 + 100 * 5 * 50$$

$$\mathbf{m[2, 3] = 25000}$$

Let i = 3, j = 4, k = 3

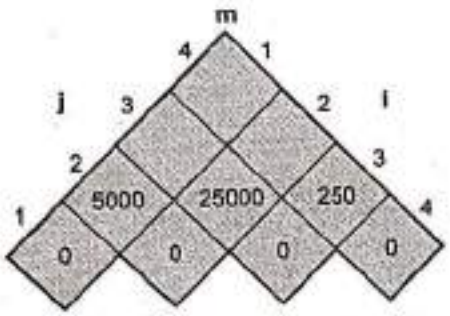
$$m[i, j] = m[i, k] + m[k+1, j] + P_{i-1} P_k P_j$$

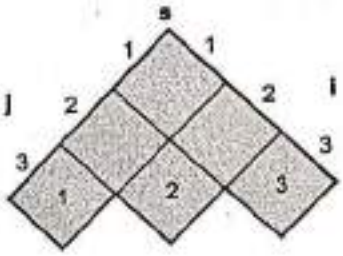
$$m[3, 4] = m[3, 3] + m[4, 4] + P_2 P_3 P_4$$

$$= 0 + 0 + 5 * 50 * 1$$

$$\mathbf{m[3, 4] = 250}$$

The tables can be partially filled as





Let i = 1, j = 3, k = 1 or k = 2

$$m[i, j] = \min\{m[i, k] + m[k+1, j] + P_{i-1} P_k P_j\}$$

$$m[1, 3] = \min\begin{Bmatrix} m[1,1] + m[2,3] + P_0 P_1 P_3 \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \end{Bmatrix}$$

$$= \min\begin{Bmatrix} 0 + 25000 + 10 * 100 * 50 \\ 5000 + 0 + 10 * 5 * 50 \end{Bmatrix}$$

$$= \min\begin{Bmatrix} 75000 \\ 7500 \end{Bmatrix}$$

$$\mathbf{m[1, 3] = 7500} \quad \text{with} \quad \mathbf{k = 2}$$

Let i = 2, j = 4, k = 2 or k = 3

$$m[i, j] = \min\{m[i, k] + m[k+1, j] + P_{i-1} P_j P_k\}$$

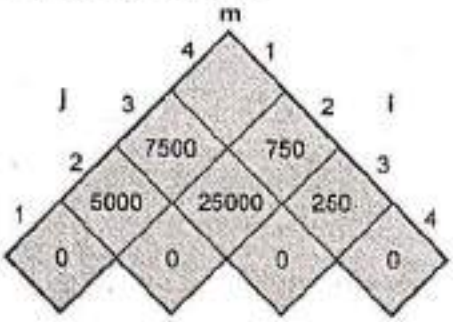
$$m[2, 4] = \min\begin{Bmatrix} m[2,2] + m[3,4] + P_1 P_4 P_2 \\ m[2,3] + m[4,4] + P_1 P_4 P_3 \end{Bmatrix}$$

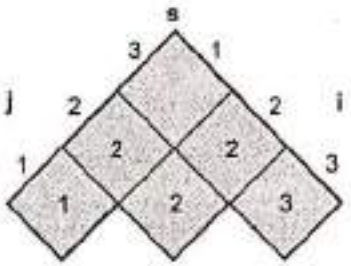
$$= \min\begin{Bmatrix} 0 + 250 + 100 * 1 * 5 \\ 25000 + 0 + 100 * 1 * 50 \end{Bmatrix}$$

$$= \min\begin{Bmatrix} 750 \\ 30{,}000 \end{Bmatrix}$$

$$\mathbf{m[2, 4] = 750} \quad \text{with} \quad \mathbf{k = 2}$$

The tables can be partially filled as





Let i = 1, j = 4, k = 1 or k = 2 or k = 3

$$m[i, j] = \min\{m[i, k] + m[k+1, j] + P_{i-1} P_j P_k\}$$

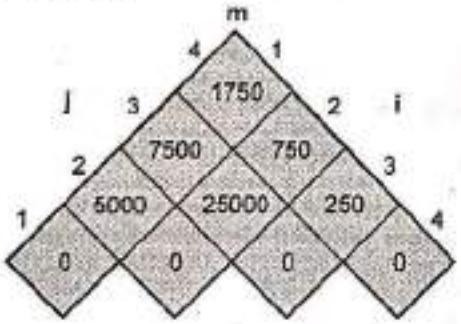
$$= \min\begin{Bmatrix} m[1,1] + m[2,4] + P_0 P_4 P_1 \\ m[1,2] + m[3,4] + P_0 P_4 P_2 \\ m[1,3] + m[4,4] + P_0 P_4 P_3 \end{Bmatrix}$$

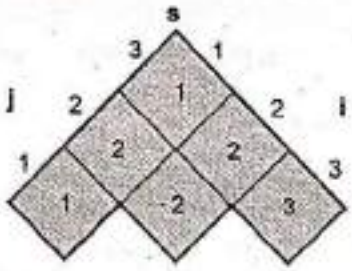
$$m[1, 4] = \min\begin{Bmatrix} 0 + 750 + 10*1*100 \\ 5000 + 250 + 10*1*5 \\ 7500 + 0 + 10*1*50 \end{Bmatrix}$$

$$= \min\begin{Bmatrix} 1750 \\ 5300 \\ 8000 \end{Bmatrix}$$

m[1, 4] = 1750 with k = 1

The tables are as follows





The optimal chained matrix order = **(A1(A2(A3A4)))**

Q.10 *Solve the Travelling Salesman problem using branch and bound algorithms.* **(Refer Q.12 of Chapter - 7)** [10]

OR

Q.11 *Explain the FIFO BB 0/1 Knapsack problem procedure with the knapsack instance for n = 4, m = 15, (p1, p2, p3, p4) = (10, 10, 12, 18), (w1, w2, w3, w4) = (2, 4, 6, 9). Draw the portion of the state space tree and find optimal solution.* **(Refer Q.17 of Chapter - 7)** [10]

END...

DECEMBER - 2019 [135AF] (R16)
Design and Analysis of Algorithms

Solved Paper
B.Tech., III - I [CSE/IT]

Time : 3 Hours] [Maximum Marks : 75

PART - A (25 Marks)

Q.1 a) *Define θ - notation.* **(Refer Q.19 of Chapter - 1)** [2]

b) *Explain "Divide and Conquer Technique".* **(Refer Important points to remember of section 2.1)** [3]

c) *State and explain graph coloring problem.* **(Refer Q.23 of Chapter - 6)** [2]

d) *Differentiate between breadth first search and Depth first search.* **(Refer Q.24 of Chapter - 3)** [3]

e) *Define minimum spanning tree.* **(Refer Q.18 of Chapter - 3)** [2]

f) *Write any two characteristics of Greedy Algorithm.* **(Refer Q.2 of Chapter - 4)** [3]

g) *What are the drawbacks of dynamic programming ?* [2]

Ans. : **Drawbacks of Dynamic programming :**

1) As the solution of every subproblem is calculated and stored, memory requirement for dynamic programming is more.

2) Many times, the output value of subproblem is stored which is never utilized in next subsequent subproblem.

h) *Write the difference between the Greedy method and Dynamic programming.* **(Refer Q.6 of Chapter - 4)** [3]

i) *Define NP-Complete.* **(Refer Q.15 of Chapter 8)** [2]

j) *Why the search path in a state-space tree of a branch and bound algorithm is terminated ?* [3]

Ans. : We terminate a search path at the current node in a state-space tree of a branch-and-bound algorithm for any one of the following three reasons :

1. The value of the node's bound is not better than the value of the best solution seen so far.
2. The node represents no feasible solutions because the constraints of the problem are already violated.
3. The subset of feasible solutions represented by the node consists of a single point and hence no further choices can be made.

PART - B (50 Marks)

Q.2 *Apply and explain merge sort to sort the following list : 8, 3, 2, 9, 7, 1, 5, 4. How efficient is merge sort ?* [10]

Ans. : Let the numbers to be stored in an array as follows - We will divide and merge the numbers.

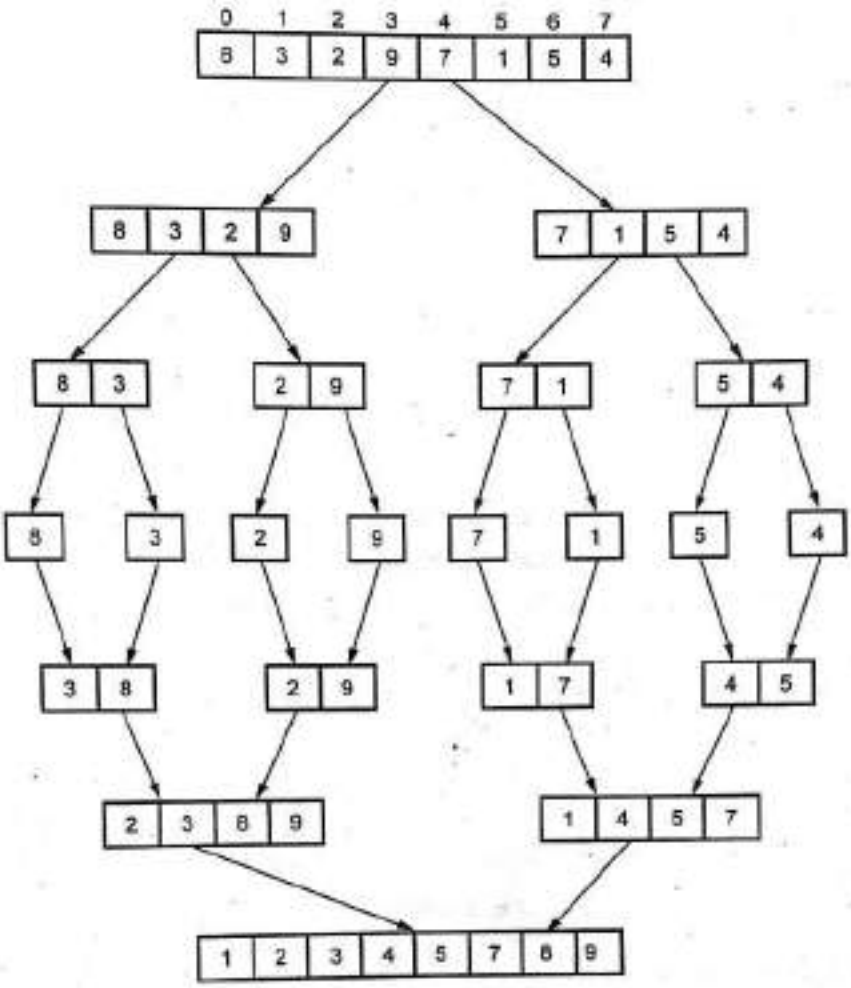


Fig. 1

OR

Q.3 a) *Consider the following recurrence. T(n) = 8 T (n/2) + n, Obtain asymptotic bound using substitution method.* **[10]**

Ans. : Let us assume T(1) = 1 as initial condition.

Let

$$= 8\left(8T\left(\frac{n}{4}\right)+\frac{n}{2}\right)+n$$

$$= 8^2T\left(\frac{n}{4}\right)+8\left(\frac{n}{2}\right)+n$$

$$= 8^2T\left(\frac{n}{4}\right)+5n$$

$$= 8^2\left(8T\left(\frac{n}{8}\right)+\frac{n}{2}\right)+5n$$

$$= 8^3 T\left(\frac{n}{8}\right) + 16n + 5n$$

$$= 8^3 T\left(\frac{n}{8}\right) + 21n$$

$$= \left(2^3\right)^3 T\left(\frac{n}{2^3}\right) + \left(3.2^3 - 1\right)n \quad \ldots (1)$$

If we put $2^k = n$ then $k = \log n$ in equation (1)

$$\therefore \quad T(n) = \left(2^k\right)^3 T(1) + k(2^k - 1)n$$

$$\therefore \quad = n^3 + \log n(n-1)n = n^3 + n\log n(n-1)$$

$$T(n) = \theta(n^3)$$

Q.4 *Consider the following graph 1. If there is ever a decision between multiple neighbor nodes in the BFS or DFS algorithms, assume we always choose the letter closest to the beginning of the alphabet first. In what order will the nodes be visited using a Breadth First Search and Depth First Search with start vertex as 0?* **[10]**

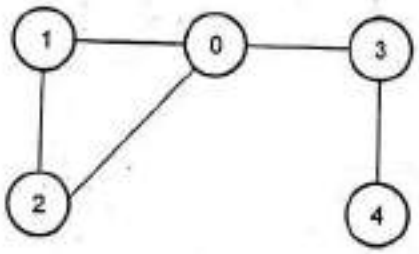


Fig. 2 Graph 1

Ans. : Refer Q. 20 and Q.21 of Chapter 3

DFS : 0-1-2-3-4

BFS : 0-1-2-3-4

OR

Q.5 *Given a set of non-negative integers {10, 7, 5, 18, 12, 20, 15}, and a value sum 35, determine if there is a subset of the given set with sum equal to given sum.* **[10]**

Ans. : The given set is {10,7,5,18,12,20,15} For sum 35

Initially subset = {}	Sum = 0	
10	10	Add next element
10,7	17<35	Add next element
10,7,5	22<35	Add next element
10,7,5,18	40>35	Sum exceeds. Hence backtrack.
10,7,5,	22<35	Add next element
10,7,5,12	34<35	Add next element

10,7,5,12,20	54>35	Sum exceeds. Hence backtrack
10,7,5,12,	34<35	Choose next element
10,7,5,12,15	49>35	Sum, exceeds.Hence backtrack.
10,7,5,12	34	Backtrack
10,7	17	Backtrack
10,	10	Add next element
10,5	15<35	Add next element 20
10,5,20	35=35	Solution Found.

Similarly we can get the solutions **{20,15}** and **{5,18,12}**

Q.6 *Write down Prim's Algorithm for finding the Minimum Spanning Tree of a connected graph. Execute your algorithm on the following graph 2.* **(Refer Q.21 of Chapter - 4)** **[10]**

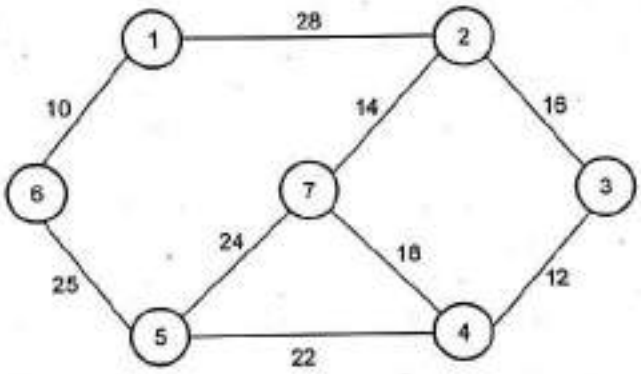


Fig. 3 Graph 2

OR

Q.7 *Given the jobs $(J_1 \ldots J_6)$, their deadlines {5,3,3,2,4,2} and associated profits as {200, 180, 190, 300, 120, 100}.*
a) Write the optimal schedule that gives maximum profit.
b) Are all the jobs completed in the optimal schedule ?
c) What is the maximum earned profit ? **[4 + 4 + 2]**

Ans. : Let the profits associated with the jobs (P1,P2,P3,P4,P5,P6) = (200,180,190,300,120,100). Deadlines associated with jobs (d1,d2,d3,d4,d5,d6) = (5,3,3,2,4,2)

Step 1 : We will arrange profits Pi in descending order. Then corresponding deadlines will appear.

Profit	300	200	190	180	120	100
Job	P4	P1	P3	P2	P5	P6
Deadline	2	5	3	3	4	2

Step 2 : Create an array J[ ] which stores the jobs. Initially J[] will be

1	2	3	4	5	6
0	0	0	0	0	0

Step 3 : Add i^{th} job in array J[ ] at the index denoted by its deadline di. First job is P4. The deadline for this job is 2. Hence insert P4 in J[ ] at 2^{nd} index.

1	2	3	4	5	6
0	P4	0	0	0	0

Step 4 : Next job is P1 with deadline 5. Hence insert P1 at index 5.

1	2	3	4	5	6
0	P4	0	0	P1	0

Step 5 : Next job is P3 with deadline 3. Hence insert P3 at index 3.

1	2	3	4	5	6
0	P4	P3	0	P1	0

Step 6 : Next job is with profit P2 with deadline 3. But as 3 is already occupied, just discard P2.

Step 7 : Next job is with profit P5 with deadline 4

1	2	3	4	5	6
0	P4	P3	P5	P1	0

Step 8 : Next job is P6 with deadline 2. As slot at index 2 is already occupied. Hence discard it.

Step 9 : The optimal sequence which we will obtain in P4-P3-P5-P1. With the maximum profit is 810 All the jobs are not allocated in this sequence. The jobs with profit P2 and P6 are discarded.

Q.8 *Write down the Floyd Warshall algorithm to solve the all pairs shortest paths problem on a directed graph. Run your algorithm on the following weighted directed graph 3 and show the matrix D_k that results for each iteration of the outer loop.* **[10]**

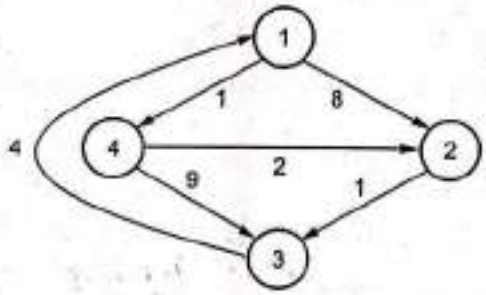


Fig. 4 Graph 3

Ans. : Let us build the adjacency matrix for the given graph.

D^0 =

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

Step 1 : Now let us compute D1,D2,.. using the following formula

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

i=1, j=1, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,1] = min(D[1,1],D[1,1]+D[1,1]) = min(0, 0+0) **D[1,1] = 0**	i=1, j=2, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,2] = min(D[1,2],D[1,1]+D[1,2]) = min(8, 0+8) **D[1,2] = 8**
i=1, j=3, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,3] = min(D[1,3],D[1,1]+D[1,3]) = min(∞, 0+∞) **D[1,3] = ∞**	i=1, j=4, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,4] = min(D[1,4],D[1,1]+D[1,4]) = min(1, 0+1) **D[1,4] = 1**
i=2, j=1, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,1] = min(D[2,1],D[2,1]+D[1,1]) = min(∞, ∞+0) **D[2,1] = ∞**	i=2, j=2, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,2] = min(D[2,2],D[2,1]+D[1,2]) = min(0, ∞+8) **D[2,2] = 0**
i=2, j=3, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,3] = min(D[2,3],D[2,1]+D[1,3]) = min(1, ∞+∞) **D[2,3] = 1**	i=2, j=4, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,4] = min(D[2,4],D[2,1]+D[1,4]) = min(∞,∞+1) **D[2,4] = ∞**
i=3, j=1, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,1] = min(D[3,1],D[3,1]+D[1,1]) = min(4,4+0) **D[3,1] = 4**	i=3, j=2, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,2] = min(D[3,2],D[3,1]+D[1,2]) = min(∞,4+8) **D[3,2] = 12**
i=3, j=3, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,3] = min(D[3,3],D[3,1]+D[1,3]) = min(0,4+∞) **D[3,3] = 0**	i=3, j=4, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,4] = min(D[3,4],D[3,1]+D[1,4]) = min(∞,4+1) **D[3,4] = 5**
i=4, j=1, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,1] = min(D[4,1],D[4,1]+D[1,1]) = min(∞,∞+0) **D[4,1] = ∞**	i=4, j=2, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,2] = min(D[4,2],D[4,1]+D[1,2]) = min(2, ∞+8) **D[4,2] = 2**

i=4, j=3, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,3] = min(D[4,3],D[4,1]+D[1,3]) = min(9, ∞+∞) **D[4,3] = 9**	i=4, j=4, k=1 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,4] = min(D[4,4],D[4,1]+D[1,4]) = min(0, ∞+1) **D[4,4] = 0**

Hence the D^1 is as follows :

D^1 =

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

Step 2 :

i=1, j=1, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,1] = min(D[1,1],D[1,2]+D[2,1]) = min(0,8+∞) **D[1,1] = 0**	i=1, j=2, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,2] = min(D[1,2],D[1,2]+D[2,2]) = min(8, 0+8) **D[1,2] = 8**
i=1, j=3, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,3] = min(D[1,3],D[1,2]+D[2,3]) = min(∞, 8+1) **D[1,3] = 9**	i=1, j=4, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,4] = min(D[1,4],D[1,2]+D[2,4]) = min(1, 8+∞) **D[1,4] = 1**
i=2, j=1, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,1] = min(D[2,1],D[2,2]+D[2,1]) = min(∞, 0+∞) **D[2,1] = ∞**	i=2, j=2, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,2] = min(D[2,2],D[2,2]+D[2,2]) = min(0, 0+0) **D[2,2] = 0**
i=2, j=3, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,3] = min(D[2,3],D[2,2]+D[2,3]) = min(1, 0+1) **D[2,3] = 1**	i=2, j=4, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,4] = min(D[2,4],D[2,2]+D[2,4]) = min(∞,0+∞) **D[2,4] = ∞**
i=3, j=1, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,1] = min(D[3,1],D[3,2]+D[2,1]) = min(4,12+∞) **D[3,1] = 4**	i=3, j=2, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,2] = min(D[3,2],D[3,2]+D[2,2]) = min(12,12+0) **D[3,2] = 12**
i=3, j=3, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,3] = min(D[3,3],D[3,2]+D[2,3]) = min(0,12+1) **D[3,3] = 0**	i=3, j=4, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,4] = min(D[3,4],D[3,2]+D[2,4]) = min(5,12+∞) **D[3,4] = 5**

i=4, j=1, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,1] = min(D[4,1],D[4,2]+D[2,1]) = min(∞,2+∞) **D[4,1] = ∞**	i=4, j=2, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,2] = min(D[4,2],D[4,2]+D[2,2]) = min(2, 2+0) **D[4,2] = 2**
i=4, j=3, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,3] = min(D[4,3],D[4,2]+D[2,3]) = min(9, 2+1) **D[4,3] = 3**	i=4, j=4, k=2 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,4] = min(D[4,4],D[4,2]+D[2,4]) = min(0, 2+∞) **D[4,4] = 0**

D^2 =

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

Step 3 :

i=1, j=1, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,1] = min(D[1,1],D[1,3]+D[3,1]) = min(0,9+4) **D[1,1] = 0**	i=1, j=2, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,2] = min(D[1,2],D[1,3]+D[3,2]) = min(8, 9+12) **D[1,2] = 8**
i=1, j=3, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,3] = min(D[1,3],D[1,3]+D[3,3]) = min(9, 9+0) **D[1,3] = 9**	i=1, j=4, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,4] = min(D[1,4],D[1,3]+D[3,4]) = min(1, 9+5) **D[1,4] = 1**
i=2, j=1, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,1] = min(D[2,1],D[2,3]+D[3,1]) = min(∞, 1+4) **D[2,1] = 5**	i=2, j=2, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,2] = min(D[2,2],D[2,3]+D[3,2]) = min(0, 1+12) **D[2,2] = 0**
i=2, j=3, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,3] = min(D[2,3],D[2,3]+D[3,3]) = min(1, 1+0) **D[2,3] = 1**	i=2, j=4, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,4] = min(D[2,4],D[2,3]+D[3,4]) = min(∞,1+5) **D[2,4] = 6**
i=3, j=1, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,1] = min(D[3,1],D[3,3]+D[3,1]) = min(4,0+4) **D[3,1] = 4**	i=3, j=2, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,2] = min(D[3,2],D[3,3]+D[3,2]) = min(12,0+12) **D[3,2] = 12**

i=3, j=3, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,3] = min(D[3,3],D[3,3]+D[3,3]) = min(0,0+0) **D[3,3] = 0**	i=3, j=4, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,4] = min(D[3,4],D[3,3]+D[3,4]) = min(5,0+5) **D[3,4] = 5**
i=4, j=1, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,1] = min(D[4,1],D[4,3]+D[3,1]) = min(∞,3+4) **D[4,1] = 7**	i=4, j=2, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,2] = min(D[4,2],D[4,3]+D[3,2]) = min(2, 3+12) **D[4,2] = 2**
i=4, j=3, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,3] = min(D[4,3],D[4,3]+D[3,3]) = min(3, 3+0) **D[4,3] = 3**	i=4, j=4, k=3 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,4] = min(D[4,4],D[4,3]+D[3,4]) = min(0, 3+∞) **D[4,4] = 0**

D^3 =

	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	12	0	5
4	7	2	3	0

Step 4 :

i=1, j=1, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,1] = min(D[1,1],D[1,4]+D[4,1]) = min(0,1+7) **D[1,1] = 0**	i=1, j=2, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,2] = min(D[1,2],D[1,4]+D[4,2]) = min(8, 1+2) **D[1,2] = 3**
i=1, j=3, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,3] = min(D[1,3],D[1,4]+D[4,3]) = min(9, 1+3) **D[1,3] = 4**	i=1, j=4, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[1,4] = min(D[1,4],D[1,4]+D[4,4]) = min(1, 1+0) **D[1,4] = 1**
i=2, j=1, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,1] = min(D[2,1],D[2,4]+D[4,1]) = min(5, 6+7) **D[2,1] = 5**	i=2, j=2, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,2] = min(D[2,2],D[2,4]+D[4,2]) = min(0, 6+2) **D[2,2] = 0**
i=2, j=3, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,3] = min(D[2,3],D[2,4]+D[4,3]) = min(1, 6+3) **D[2,3] = 1**	i=2, j=4, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[2,4] = min(D[2,4],D[2,4]+D[4,4]) = min(6,6+0) **D[2,4] = 6**

i=3, j=1, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,1] = min(D[3,1],D[3,4]+D[4,1]) = min(4,5+7) **D[3,1] = 4**	i=3, j=2, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,2] = min(D[3,2],D[3,4]+D[4,2]) = min(12,5+2) **D[3,2] = 7**
i=3, j=3, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,3] = min(D[3,3],D[3,4]+D[4,3]) = min(0,5+3) **D[3,3] = 0**	i=3, j=4, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[3,4] = min(D[3,4],D[3,4]+D[4,4]) = min(5,5+0) **D[3,4] = 5**
i=4, j=1, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,1] = min(D[4,1],D[4,4]+D[4,1]) = min(7,0+7) **D[4,1] = 7**	i=4, j=2, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,2] = min(D[4,2],D[4,4]+D[4,2]) = min(2, 0+2) **D[4,2] = 2**
i=4, j=3, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,3] = min(D[4,3],D[4,4]+D[4,3]) = min(3, 0+3) **D[4,3] = 3**	i=4, j=4, k=4 D[i,j] = min(D[i,j],D[i,k]+D[k,j]) D[4,4] = min(D[4,4],D[4,4]+D[4,4]) = min(0, 0+0) **D[4,4] = 0**

$D^4 =$

	1	2	3	4
1	0	3	4	1
2	5	0	1	6
3	4	7	0	5
4	7	2	3	0

This is the final solution, which is representing the all pair shortest path.

OR

Q.9 *Determine the cost and structure of an optimal binary search tree for a set of n = 7 keys with the following probabilities. Where (p1...p7) = { .04 .06 .08 .02 .10 .12 .14} (q0....q7) = {.06 .06 .06 .06 .05 .05 .05 .05}* **[10]**

Ans. : Let (p1, ..., p7) = (0.04, 0.06, 0.08, 0.02, 0.10, 0.12, 0.14) and (q0, ..., q7) = (0.06, 0.06, 0.06, 0.06, 0.05, 0.05, 0.05, 0.05)

We will use following formulae :

$$W_{i,i} = q_i,\ r_{i,i} = 0,\ C_{i,i} = 0$$

$$W_{i,i+1} = q_i + q_{(i+1)} + p_{(i+1)}$$

$$r_{i,i+1} = i + 1$$

$$C_{i,i+1} = q_i + q_{(i+1)} + p_{(i+1)}$$

$$W_{i,j} = W_{i,j-1} + p_j + q_j$$

$$r_{i,j} = k$$

$$C_{i,j} = i<\overset{\min}{k}\leq j\{C_{(i,k-1)} + C_{k,j}\} + W_{i,j}$$

We will construct tables for the values W, C, and r

Let $i = 0, W_{00} = q_0 = 0.06$

$i = 1, W_{11} = q_1 = 0.06$

$i = 2, W_{22} = q_2 = 0.06$

$i = 3, W_{33} = q_3 = 0.06$

$i = 4, W_{44} = q_4 = 0.05$

$i = 5, W_{55} = q_5 = 0.05$

$i = 6, W_{66} = q_6 = 0.05$

$i = 7, W_{77} = q_7 = 0.05$

Step 1 : For computation of W_{ij} we use following formula -

i) $W_{i,j+1} = Q_i + Q_{i+1} + p_{i+1}$

ii) $W_{i,j} = W_{ij-1} + p_j + Q_j$

When i = 0 and j - i = 1 $W_{01} = q_0 + q_1 + p_1$ = 0.06 + 0.06 + 0.04 $W_{01} = 0.16$	When i=1 and j - i = 1 $W_{12} = q_1 + q_2 + p_2$ = 0.06 + 0.06 + 0.06 $W_{12} = 0.18$	When i = 2 and j - i = 1 $W_{23} = q_2 + q_3 + p_3$ = 0.06 + 0.06 +0.06 $W_{23} = 0.18$	When i = 3 and j - i = 1 $W_{34} = q_3 + q_4 + p_4$ = 0.06 + 0.05 + 0.02 $W_{34} = 0.13$	When i = 4 and j - i = 1 $W_{45} = q_4 + q_5 + p_5$ = 0.05 + 0.05 + 0.10 $W_{45} = 0.20$	When i=5 and j - i = 1 $W_{56} = q_5 + q_6 + p_6$ = 0.05 + 0.05 +0.12 $W_{56} = 0.22$	When i=6 and j - i = 1 $W_{67} = q_6 + q_7 + p_7$ = 0.05+0.05+0.14 $W_{67} = 0.24$
When i = 0 and j - i = 2 $W_{02} = W_{01} + p_2 + q_2$ = 0.16 + 0.06 + 0.06 $W_{02} = 0.28$	When i = 1 and j - i = 20 $W_{13} = W_{12} + p_3 + q_3$ = 0.18 + 0.06 + 0.06 $W_{13} = 0.32$	When i = 2 and j - i = 2 $W_{24} = W_{23} + p_4 + q_4$ = 0.20 + 0.02 + 0.05 $W_{24} = 0.27$	When i = 3 and j - i = 2 $W_{35} = W_{34} + p_5 + q_5$ = 0.13 + 0.10 + 0.05 $W_{35} = 0.28$	When i = 4 and j - i = 2 $W_{46} = W_{45} + p_6 + q_6$ = 0.20 + 0.12 + 0.05 $W_{46} = 0.37$	When i = 5 and j - i = 2 $W_{57} = W_{56} + p_7 + q_7$ = 0.22 + 0.14 + 0.05 $W_{57} = 0.41$	
When i = 0 and j - i = 3 $W_{03} = W_{02} + p_3 + q_3$ = 0.28 + 0.06 + 0.06 $W_{03} = 0.42$	When i = 1 and j - i = 3 $W_{14} = W_{13} + p_4 + q_4$ = 0.32 + 0.02 + 0.05 $W_{14} = 0.39$	When i = 2 and j - i = 3 $W_{25} = W_{24} + p_5 + q_5$ = 0.27+ 0.10 + 0.05 $W_{25} = 0.42$	When i = 3 and j - i = 3 $W_{36} = W_{35} + p_6 + q_6$ = 0.28 + 0.12 + 0.06 $W_{36} = 0.46$	When i = 4 and j - i = 3 $W_{47} = W_{46} + p_7 + q_7$ = 0.37 + 0.14 + 0.05 $W_{47} = 0.56$		
When i = 0 and j - i = 4 $W_{04} = W_{03} + p_4 + q_4$ = 0.42 + 0.02 + 0.05 $W_{04} = 0.49$	When i = 1 and j - i = 4 $W_{15} = W_{14} + p_5 + q_5$ = 0.39 + 0.10 + 0.05 $W_{15} = 0.54$	When i = 2 and j - i = 4 $W_{26} = W_{25} + p_6 + q_6$ = 0.42+0.12+0.05 $W_{26} = 0.59$	When i = 3 and j - i = 4 $W_{37} = W_{36} + p_7 + q_7$ = 0.46 + 0.14 + 0.05 $W_{37} = 0.65$			
When i = 0 and j - i = 5 $W_{05} = W_{04} + p_5 + q_5$ = 0.49 + 0.12 + 0.05 $W_{05} = 0.66$	When i = 1 and j - i = 5 $W_{16} = W_{15} + p_6 + q_6$ = 0.54 + 0.12 + 0.05 $W_{16} = 0.71$	When i = 2 and j - i = 5 $W_{27} = W_{26} + p_7 + q_7$ = 0.59 + 0.14 + 0.05 $W_{27} = 0.78$				
When i = 0 and j - i = 6 $W_{06} = W_{05} + p_6 + q_6$ = 0.66 + 0.12 + 0.05 $W_{06} = 0.83$	When i = 1 and j - i = 6 $W_{17} = W_{16} + p_7 + q_7$ = 0.71 + 0.14 + 0.05 $W_{17} = 0.9$					
When i = 0 and j - i = 7 $W_{07} = W_{06} + p_7 + q_7$ = 0.83 + 0.14 + 0.05 $W_{07} = 1.02$						

The summerized table of W can be represented as

i →

$W_{00} = 0.06$	$W_{11} = 0.06$	$W_{22} = 0.06$	$W_{33} = 0.06$	$W_{44} = 0.05$	$W_{55} = 0.05$	$W_{66} = 0.05$	$W_{77} = 0.05$
$W_{01} = 0.16$	$W_{12} = 0.18$	$W_{23} = 0.20$	$W_{34} = 0.13$	$W_{45} = 0.20$	$W_{56} = 0.22$	$W_{67} = 0.24$	
$W_{02} = 0.28$	$W_{13} = 0.32$	$W_{24} = 0.27$	$W_{35} = 0.28$	$W_{46} = 0.37$	$W_{57} = 0.41$		
$W_{03} = 0.42$	$W_{14} = 0.39$	$W_{25} = 0.42$	$W_{36} = 0.46$	$W_{47} = 0.56$			
$W_{04} = 0.49$	$W_{15} = 0.54$	$W_{26} = 0.59$	$W_{37} = 0.65$				
$W_{05} = 0.66$	$W_{16} = 0.71$	W27 = 0.78					
$W_{06} = 0.83$	$W_{17} = 0.9$						
$W_{07} = 1.02$							

j − i ↓

Step 2 : Now we will compute C and r

The formula is, $C_{ii} = 0$ and $r_{ii} = 0$

Hence

$C_{00} = C_{11} = C_{22} = C_{33} = C_{44} = C_{55} = C_{66} = C_{77} = 0$

and $r_{00} = r_{11} = r_{22} = r_{33} = r_{44} = r_{55} = r_{66} = r_{77} = 0$

For computation of C_{ij} and r_{ij} we use following formula -

1) $$C_{i,i+1} = q_i + q_{i+1} + P_{i+1}$$

2) $$r_{ij} = i + 1$$

i = 0 and j − i = 1 $C_{01} = q_0 + q_1 + P_1$ = 0.06 + 0.06 + 0.04 $C_{01} = 0.16$	i = 1 and j − i = 1 $C_{12} = q_1 + q_2 + P_2$ = 0.06 + 0.06 + 0.06 $C_{12} = 0.18$	i = 2 and j − i = 1 $C_{23} = q_2 + q_3 + P_3$ = 0.06 + 0.06 + 0.08 $C_{23} = 0.20$	i = 3 and j − i = 1 $C_{34} = q_3 + q_4 + P_4$ = 0.06 + 0.05 + 0.02 $C_{34} = 0.13$	i = 4 and j − i = 1 $C_{45} = q_4 + q_5 + P_5$ = 0.05 + 0.05 + 0.02 $C_{45} = 0.12$	i = 5 and j − i = 1 $C_{56} = q_5 + q_6 + P_6$ = 0.05 + 0.05 + 0.12 $C_{56} = 0.22$	i = 6 and j − i = 1 $C_{67} = q_6 + q_7 + P_7$ = 0.05 + 0.05 + 0.14 $C_{67} = 0.24$
i = 0 and j − i = 1 $r_{01} = i + 1$ $r_{01} = 1$	i = 1 and j − i = 1 $r_{12} = i + 1$ $r_{12} = 2$	i = 2 and j − i = 1 $R_{23} = i + 1$ $r_{23} = 3$	i = 2 and j − i = 1 $R_{34} = i + 1$ $r_{34} = 3$	i = 4 and j − i = 1 $R_{45} = i + 1$ $r_{45} = 5$	i = 5 and j − i = 1 $R_{56} = i + 1$ $r_{56} = 6$	i = 6 and j − i = 1 . $R_{67} = i + 1$ $r_{67} = 7$

Step 3 : Now we will compute C_{ij} and r_{ij} for $j - i >= 2$

The formula used is

$$C_{i,j} = i < \overset{\min}{k} < j \{C_{i,k-1} + C_{k,j}\} + W_{i,j}$$

There fore we need to find value of k

When i = 0 and j = 2	When i = 1 and j = 3	When i = 2 and j = 4	When i = 3 and j = 5	When i = 4 and j = 6	When i = 5 and j = 7
• For k = 1 $C_{02} = C_{1k-1} + C_{kj}$ $= C_{00} + C_{12}$ $= 0 + 0.18$ $C_{02} = 0.18$ • For k = 2 $C_{02} = C_{ik-1} + C_{kj}$ $= C_{01} + C_{22}$ $= 0.16 + 0$ $C_{02} = 0.16$ From above computations $Min(C_{02}) = 0.16$ $\therefore$ k = 2 • $r_{02} = k = 2$ $C_{02} = Min(C_{02}) + W_{02}$ $= 0.16 + 0.28$ $C_{02} = 0.44$	• For k = 2 $C_{13} = C_{ik-1} + C_{kj}$ $= C_{11} + C_{23}$ $= 0 + 0.20$ $C_{13} = 0.20$ • For k = 3 $C_{13} = C_{ik-1} + C_{kj}$ $= C_{12} + C_{33}$ $= 0.18 + 0$ $C_{13} = 0.18$ From above computations $Min(C_{13}) = 0.18$ $\therefore$ k = 3 • $r_{13} = k = 3$ $C_{13} = Min(C_{13}) + W_{13}$ $= 0.18 + 0.32$ $C_{13} = 0.5$	• For k = 3 $C_{24} = C_{ik-1} + C_{kj}$ $= C_{22} + C_{34}$ $= 0 + 0.13$ $C_{24} = 0.13$ • For k = 4 $C_{24} = C_{ik-1} + C_{kj}$ $= C_{23} + C_{44}$ $= 0.20 + 0$ $C_{24} = 0.20$ From above computations $Min(C_{24}) = 0.13$ $\therefore$ k = 3 • $r_{24} = k = 3$ $C_{24} = Min(C_{24}) + W_{24}$ $= 0.13 + 0.27$ $C_{24} = 0.4$	• For k = 4 $C_{35} = C_{ik-1} + C_{kj}$ $= C_{33} + C_{45}$ $= 0 + 0.12$ $C_{35} = 0.12$ • For k = 5 $C_{35} = C_{ik-1} + C_{kj}$ $= C_{34} + C_{55}$ $= 0.13 + 0$ $C_{35} = 0.13$ From above computations $Min(C_{35}) = 0.12$ $\therefore$ k = 4 • $r_{35} = k = 4$ $C_{35} = Min(C_{35}) + W_{35}$ $= 0.12 + 0.28$ $C_{35} = 0.4$	• For k = 5 $C_{46} = C_{ik-1} + C_{kj}$ $= C_{44} + C_{56}$ $= 0 + 0.22$ $C_{46} = 0.22$ • For k = 6 $C_{46} = C_{ik-1} + C_{kj}$ $= C_{45} + C_{66}$ $= 0.12 + 0$ $C_{46} = 0.12$ From above computations $Min(C_{46}) = 0.12$ $\therefore$ k = 6 • $r_{46} = k = 6$ $C_{46} = Min(C_{46}) + W_{46}$ $= 0.12 + 0.37$ $C_{46} = 0.49$	• For k = 6 $C_{57} = C_{ik-1} + C_{kj}$ $= C_{55} + C_{67}$ $= 0 + 0.24$ $C_{57} = 0.24$ • For k = 7 $C_{57} = C_{ik-1} + C_{kj}$ $= C_{56} + C_{77}$ $= 0.22 + 0$ $C_{57} = 0.22$ From above computations $Min(C_{57}) = 0.22$ $\therefore$ k = 7 • $r_{57} = k = 7$ $C_{57} = Min(C_{57}) + W_{57}$ $= 0.22 + 0.41$ $C_{57} = 0.63$
When i = 0 and j = 3, • For k = 1 $C_{03} = C_{ik-1} + C_{kj}$ $= C_{00} + C_{13}$ $= 0 + 0.5$ $C_{03} = 0.5$ • For k = 2 $C_{03} = C_{ik-1} + C_{kj}$ $= C_{01} + C_{23}$ $= 0.16 + 0.20$ $C_{03} = 0.36$ • For k = 3 $C_{03} = C_{ik-1} + C_{kj}$ $= C_{02} + C_{33}$ $= 0.44 + 0$ $C_{03} = 0.44$ From above computations $Min(C_{03}) = 0.36$ $\therefore$ k = 2 • $r_{03} = k = 2$ $C_{03} = Min(C_{03}) + W_{03}$	When i = 1 and j = 4, • For k = 2 $C_{14} = C_{ik-1} + C_{kj}$ $= C_{11} + C_{24}$ $= 0 + 0.4$ $C_{14} = 0.4$ • For k = 3 $C_{14} = C_{ik-1} + C_{kj}$ $= C_{12} + C_{34}$ $= 0.18 + 0.13$ $C_{14} = 0.31$ • For k = 4 $C_{14} = C_{ik-1} + C_{kj}$ $= C_{13} + C_{44}$ $= 0.5 + 0$ $C_{14} = 0.5$ From above computations $Min(C_{14}) = 0.31$ $\therefore$ k = 3 • $r_{14} = k = 3$ $C_{14} = Min(C_{14}) + W_{14}$	When i = 2 and j = 5, • For k = 3 $C_{25} = C_{ik-1} + C_{kj}$ $= C_{22} + C_{35}$ $= 0 + 0.4$ $C_{25} = 0.4$ • For k = 4 $C_{25} = C_{ik-1} + C_{kj}$ $= C_{23} + C_{45}$ $= 0.20 + 0.12$ $C_{25} = 0.32$ • For k = 5 $C_{25} = C_{ik-1} + C_{kj}$ $= C_{24} + C_{55}$ $= 0.4 + 0$ $C_{25} = 0.4$ From above computations $Min(C_{25}) = 0.32$ $\therefore$ k = 4 • $r_{25} = k = 4$ $C_{25} = 0.32$ Min $(C_{25}) + W_{25}$	When i = 3 and j = 6, • For k = 4 $C_{36} = C_{ik-1} + C_{kj}$ $= C_{33} + C_{46}$ $= 0 + 0.49$ $C_{36} = 0.49$ • For k = 5 $C_{36} = C_{ik-1} + C_{kj}$ $= C_{34} + C_{56}$ $= 0.13 + 0.22$ $C_{36} = 0.35$ • For k = 6 $C_{36} = C_{ik-1} + C_{kj}$ $= C_{35} + C_{66}$ $= 0.4 + 0$ $C_{36} = 0.4$ From above computations $Min(C_{36}) = 0.35$ $\therefore$ k = 5 • $r_{36} = k = 5$ $C_{36} = 0.35$ Min $(C_{36}) + W_{36}$	When i = 4 and j = 7, • For k = 5 $C_{47} = C_{ik-1} + C_{kj}$ $= C_{44} + C_{57}$ $= 0 + 0.63$ $C_{47} = 0.63$ • For k = 6 $C_{47} = C_{ik-1} + C_{kj}$ $= C_{45} + C_{67}$ $= 0.12 + 0.24$ $C_{47} = 0.36$ • For k = 7 $C_{47} = C_{ik-1} + C_{kj}$ $= C_{46} + C_{77}$ $= 0.49 + 0$ $C_{47} = 0.49$ From above computations $Min(C_{47}) = 0.36$ $\therefore$ k = 6 • $r_{47} = k = 6$ $C_{47} = 0.36$ $Min(C_{47}) + W_{47}$	

= 0.36 + 042

C_{03} = 0.78

= 0.31 + 0.39

C_{14} = 0.7

= 0.32 + 0.42

C_{25} = 0.74

= 0.35+0.46

C_{36} = 0.81

= 0.36 + 0.56

C_{47} = 0.92

When i = 0 and j = 4,

- For k = 1

$C_{04} = C_{k-1} + C_{kj}$

$= C_{00} + C_{14}$

= 0 + 0.7

C_{04} = 0.7

- For k = 2

$C_{04} = C_{k-1} + C_{kj}$

$= C_{01} + C_{24}$

= 0.16 + 0.4

C_{04} = 0.56

- For k = 3

$C_{04} = C_{k-1} + C_{kj}$

$= C_{02} + C_{34}$

= 0.44 + 0.13

C_{04} = 0.57

- For k = 4

$C_{04} = C_{k-1} + C_{kj}$

$= C_{03} + C_{44}$

= 0.78 +0

C_{04} = 0.78

From above computations

Min(C_{04}) = 0.56

∴ k = 2

- r_{04} = k = 2

C_{04} = Min (C_{04}) + W_{04}

= 0.56 + 0.49

C_{04} = 1.05

When i = 1 and j = 5,

- For k = 2

$C_{15} = C_{k-1} + C_{kj}$

$= C_{11} + C_{25}$

= 0 + 0.74

C_{15} = 0.74

- For k = 3

$C_{15} = C_{k-1} + C_{kj}$

$= C_{12} + C_{35}$

= 0.18 + 0.4

C_{15} = 0.58

- For k = 4

$C_{15} = C_{k-1} + C_{kj}$

$= C_{13} + C_{45}$

= 0.5 + 0

C_{15} = 0.50

- For k = 5

$C_{15} = C_{k-1} + C_{kj}$

$= C_{14} + C_{55}$

= 0.7 + 0

C_{15} = 0.70

From above computations

Min(C_{15}) = 0.50

∴ k = 4

- r_{15} = k = 4

C_{15} = Min(C_{15}) + W_{15}

= 0.50 + 0.54

C_{15} = 1.04

When i = 2 and j = 6,

- For k = 3

$C_{26} = C_{k-1} + C_{kj}$

$= C_{22} + C_{36}$

= 0 + 0.81

C_{26} = 0.81

- For k = 4

$C_{26} = C_{k-1} + C_{kj}$

$= C_{23} + C_{46}$

= 0.20 + 0.49

C_{26} = 0.69

- For k = 5

$C_{26} = C_{k-1} + C_{kj}$

$= C_{24} + C_{56}$

= 0.4 + 0.22

C_{26} = 0.62

- For k = 6

$C_{26} = C_{k-1} + C_{kj}$

$= C_{25} + C_{66}$

= 0.74 + 0

C_{26} = 0.74

From above computations

Min(C_{26}) = 0.62

∴ k = 5

- r_{25} = k = 5

C_{26} = Min(C_{26}) + W_{26}

= 0.62+0.59

C_{26} = 1.21

When i = 3 and j = 7,

- For k = 4

$C_{37} = C_{k-1} + C_{kj}$

$= C_{33} + C_{47}$

= 0 + 0.92

C_{37} = 0.92

- For k = 5

$C_{37} = C_{k-1} + C_{kj}$

$= C_{34} + C_{57}$

= 0.13 + 0.63

C_{37} = 0.76

- For k = 6

$C_{37} = C_{k-1} + C_{kj}$

$= C_{35} + C_{67}$

= 0.4 + 0.24

C_{37} = 0.64

- For k = 7

$C_{37} = C_{k-1} + C_{kj}$

$= C_{36} + C_{77}$

= 0.81 + 0

C_{37} = 0.81

From above computations

Min(C_{37}) =0.64

∴ k = 6

- r_{37} = k = 6

C_{37} = Min(C_{37}) + W_{37}

= 0.64 + 0.65

C_{37} = 1.29

When i = 0 and j = 5,

- For k = 1

$C_{05} = C_{k-1} + C_{kj}$

$= C_{00} + C_{15}$

= 0 + 1.04

C_{05} = 1.04

- For k = 2

$C_{05} = C_{k-1} + C_{kj}$

$= C_{01} + C_{25}$

= 0.16+0.74

C_{05} = 0.90

- For k = 3

When i = 1 and j = 6,

- For k = 2

$C_{16} = C_{k-1} + C_{kj}$

$= C_{11} + C_{26}$

= 0 + 1.21

C_{16} = 1.21

- For k = 3

$C_{16} = C_{k-1} + C_{kj}$

$= C_{12} + C_{36}$

= 0.18 + 0.81

C_{16} = 0.99

- For k = 4

When i = 2 and j = 7,

- For k = 3

$C_{27} = C_{k-1} + C_{kj}$

$= C_{22} + C_{37}$

= 0 + 1.29

C_{27} = 1.29

- For k = 4

$C_{27} = C_{k-1} + C_{kj}$

$= C_{23} + C_{47}$

= 0.20 + 0.92

C_{27} = 1.12

- For k = 5

$C_{05} = C_{1k-1} + C_{kj}$
$= C_{02} + C_{35}$
$= 0.44 + 0.4$
$C_{05} = 0.84$

- For k = 4

$C_{05} = C_{1k-1} + C_{kj}$
$= C_{03} + C_{45}$
$= 0.78+0.12$
$C_{05} = 0.90$

- For k = 5

$C_{05} = C_{1k-1} + C_{kj}$
$= C_{04} + C_{55}$
$= 1.05 + 0$
$C_{05} = 1.05$

From above computations

$Min(C_{05}) = 0.84$

k = 3

- $r_{05} = k = 3$

$C_{05} = Min(C_{05}) + W_{05}$
$= 0.84 + 0.66$
$C_{05} = 1.50$

$C_{16} = C_{1k-1} + C_{kj}$
$= C_{13} + C_{46}$
$= 0.5 + 0.49$
$C_{16} = 0.99$

- For k = 5

$C_{16} = C_{1k-1} + C_{kj}$
$= C_{14} + C_{56}$
$= 0.7 + 0.22$
$C_{16} = 0.92$

- For k = 6

$C_{16} = C_{1k-1} + C_{kj}$
$= C_{15} + C_{66}$
$= 1.04 + 0$
$C_{16} = 1.04$

From above computations

$Min(C_{16}) = 0.92$

k = 5

- $r_{16} = k = 5$

$C_{16} = Min(C_{16}) + W_{16}$
$= 0.92+0.71$
$C_{16} = 1.63$

$C_{27} = C_{1k-1} + C_{kj}$
$= C_{24} + C_{57}$
$= 0.4 + 0.63$
$C_{27} = 1.03$

- For k = 6

$C_{27} = C_{1k-1} + C_{kj}$
$= C_{25} + C_{67}$
$= 0.74 + 0.24$
$C_{27} = 0.98$

- For k = 7

$C_{27} = C_{1k-1} + C_{kj}$
$= C_{26} + C_{77}$
$= 1.21+0$
$C_{27} = 1.21$

From above computations

$Min(C_{27}) = 0.98$

k = 6

- $r_{27} = k = 6$

$C_{27} = Min(C_{27}) + W26$
$= 0.98 + 0.59$
$C_{27} = 1.57$

When i = 0 and j = 6,

- For k = 1

$C_{06} = C_{1k-1} + C_{kj}$
$= C_{00} + C_{16}$
$= 0 + 1.63$
$C_{06} = 1.63$

- For k = 2

$C_{06} = C_{1k-1} + C_{kj}$
$= C_{01} + C_{26}$
$= 0.16 + 1.21$
$C_{06} = 1.37$

- For k = 3

$C_{06} = C_{1k-1} + C_{kj}$
$= C_{02} + C_{36}$
$= 0.44 + 0.81$
$C_{06} = 1.25$

- For k = 4

$C_{06} = C_{1k-1} + C_{kj}$
$= C_{03} + C_{46}$
$= 0.78+ 0.49$
$C_{06} = 1.27$

- For k = 5

When i = 1 and j = 7,

- For k = 2

$C_{17} = C_{1k-1} + C_{kj}$
$= C_{11} + C_{27}$
$= 0 + 1.57$
$C_{17} = 1.57$

- For k = 3

$C_{17} = C_{1k-1} + C_{kj}$
$= C_{12} + C_{37}$
$= 0.18+1.29$
$C_{17} = 1.47$

- For k = 4

$C_{17} = C_{1k-1} + C_{kj}$
$= C_{13} + C_{47}$
$= 0.5 + 0.92$
$C_{17} = 1.42$

- For k = 5

$C_{17} = C_{1k-1} + C_{kj}$
$= C_{14} + C_{57}$
$= 0.7 + 0.63$
$C_{17} = 1.33$

- For k = 6

$C_{06} = C_{(k-1)} + C_{kj}$
$= C_{04} + C_{56}$
$= 1.05 + 0.22$
$C_{06} = 1.27$

- For k = 6

$C_{06} = C_{(k-1)} + C_{kj}$
$= C_{05} + C_{66}$
$= 1.50 + 0$
$C_{06} = 1.50$

From above computations
$\text{Min}(C_{06}) = 1.25$
k = 3

- $r_{06} = k = 3$

$C_{06} = \text{Min}(C_{06}) + W_{06}$
$= 1.25 + 0.83$
$C_{06} = 2.08$

$C_{17} = C_{(k-1)} + C_{kj}$
$= C_{15} + C_{67}$
$= 1.04 + 0.24$
$C_{17} = 1.28$

- For k = 7

$C_{17} = C_{(k-1)} + C_{kj}$
$= C_{15} + C_{77}$
$= 1.04 + 0$
$C_{17} = 1.04$

From above computations
$\text{Min}(C_{17}) = 1.04$
k = 7

- $r_{17} = k = 7$

$C_{17} = \text{Min}(C_{17}) + W_{17}$
$= 1.04 + 0.9$
$C_{17} = 1.94$

When i = 0 and j = 7,

- For k = 1

$C_{07} = C_{(k-1)} + C_{kj}$
$= C_{00} + C_{77}$
$= 0 + 1.94$
$C_{07} = 1.94$

- For k = 2

$C_{07} = C_{(k-1)} + C_{kj}$
$= C_{01} + C_{27}$
$= 0.16 + 1.57$
$C_{07} = 1.73$

- For k = 3

$C_{07} = C_{(k-1)} + C_{kj}$
$= C_{02} + C_{37}$
$= 0.44 + 1.29$
$C_{07} = 1.73$

- For k = 4

$C_{07} = C_{(k-1)} + C_{kj}$
$= C_{03} + C_{47}$
$= 0.78 + 0.92$
$C_{07} = 1.70$

- For k = 5

$C_{07} = C_{(k-1)} + C_{kj}$
$= C_{04} + C_{57}$
$= 1.05 + 0.63$
$C_{07} = 1.68$

- For k = 6

$C_{07} = C_{ik-1} + C_{kj}$

$= C_{05} + C_{67}$

$= 1.50+0.24$

$C_{07} = 1.74$

- For k = 7

$C_{07} = C_{ik-1} + C_{kj}$

$= C_{06} + C_{77}$

$= 2.08 + 0$

$C_{07} = 2.08$

From above computations

$Min(C_{07}) = 1.68$

$\therefore k = 5$

$r_{07} = k = 5$

$C_{07} = Min(C_{07}) + W_{07}$

$= 1.68 + 1.02$

$C_{07} = 2.70$

Let us build tree using r_{ij}

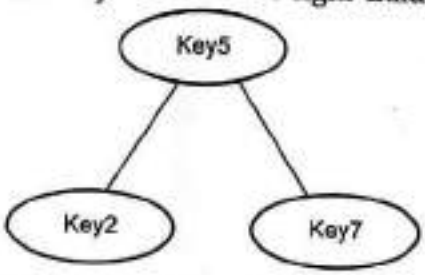
We consider $r_{07} = 5$. Hence key 5 will be the root node.

For $r_{07} = 5$

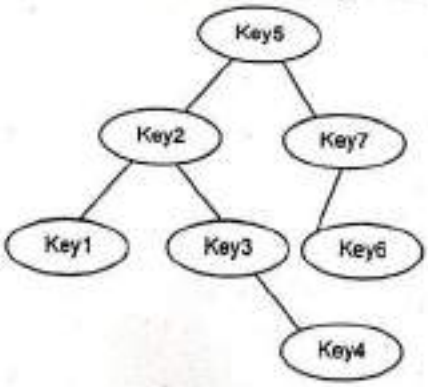
I=0,j=7,k=5

Left child = $r_{ik-1} = r_{04} = 2$. That means Key 2 will be left child

Right child = $r_{kj} = r_{57} = 7$. That means key 7 will be the right child



Thus now finding the left and right child for each node we get the final OBST as



This is the required optimal Binary Search Tree with the cost 2.70.

Q.10 a) *Prove, If any NP-complete problem belongs to class P, then is P = NP ?* **(Refer Q.15 of Chapter - 8)**

b) *Write a non deterministic algorithm of sorting the list of elements.* **(Refer Q.11 of Chapter - 8)** **[5 + 5]**

OR

Q.11 *Solve the Travelling Salesman Problem for the following graph 4 by using the Branch and Bound Algorithm.* **[10]**

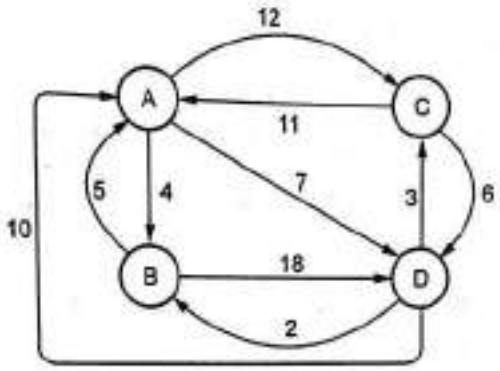


Fig. 5 Graph 4

Ans. :

Step 1 : We will create an adjacency matrix for the given graph

	A	B	C	D
A	∞	0	8	3
B	0	∞	∞	13
C	5	∞	∞	0
D	8	0	1	∞

Original Matrix

Now as every row contains 0, there is no need to reduce the rows.

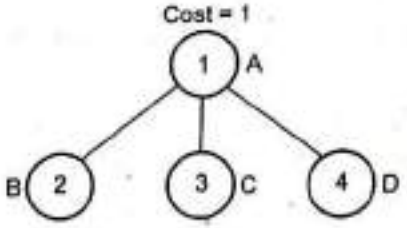
The column 3 is reduced by 1 as it contains 1 as minimum value. After performing reduction (i.e. subtracting 1 from column 3) we get the reduced matrix as

	A	B	C	D
A	∞	0	7	3
B	0	∞	∞	13
C	5	∞	∞	0
D	8	0	0	∞

Reduced Matrix

We get the reduced cost as $= r = 1$

Step 2 : Now we will construct the state space tree as follows -



Now we will find the cost of A-B, A-C and A-D

Finding the cost of A-B

Make first row and second column as ∞ . Similarly M[B-A] = ∞. Hence

	A	B	C	D	
A	∞	∞	∞	∞	
B	∞	∞	∞	13	13
C	5	∞	∞	0	
D	8	∞	0	∞	
	5				

∴ Total reduced cost r̂ = 18

Hence cost of A–B = c(A – B) + r + r̂

= 0 + 1 + 18 = 19

C(A–B) = 19

Finding the cost of A-C

Make first row and third column as ∞ . Similarly M[C- A] = ∞. Hence

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	13
C	∞	∞	∞	0
D	8	0	∞	∞

∴ Total reduced cost r̂ = 0

Hence cost of A-C = c(A-C)+ r + r̂

= 7 + 1 + 0

C(A–C) = 8

Finding the cost of A-D

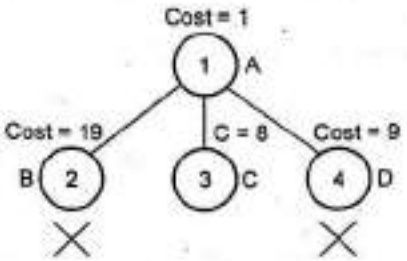
Make first row and fourth column as ∞. Similarly M[D-A] = ∞. Hence

	A	B	C	D	
A	∞	∞	∞	∞	
B	0	∞	∞	∞	
C	5	∞	∞	∞	5
D	∞	0	∞	∞	

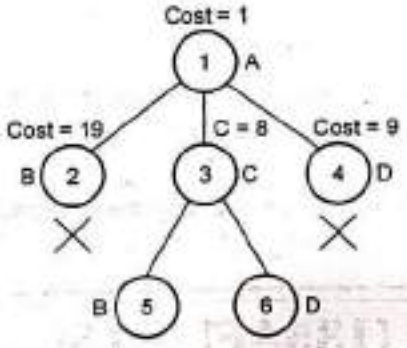
$\therefore$ $\hat{r} = 5$

Hence cost of (A-D) = c(A−D)+r+$\hat{r}$ = 3 + 1 + 5 = 9

Hence the state space tree is



Step 3 : Now we will expand node 3. The space tree is



Now we will find the cost of C-B, C-D

Finding the cost of C-B. Consider the reduced matrix obtained in step 2

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	13
C	∞	∞	∞	0
D	8	0	∞	∞

Make third row and second column as ∞. Also make B - C as ∞

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	13
C	∞	∞	∞	∞
D	8	∞	∞	∞

Here $\hat{r} = 5$

Hence cost of C-B = = c(C-B) + c(3) + $\hat{r}$ = ∞ + 8 + 0 = ∞

Similarly we will find the cost of C-D. Consider the reduced matrix obtained in step 2

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	13
C	∞	∞	∞	0
D	8	0	∞	∞

Make third row and fourth column as ∞. Also make D-C as ∞

	A	B	C	D
A	∞	∞	∞	∞
B	0	∞	∞	∞
C	∞	∞	∞	∞
D	8	∞	∞	∞

Hence cost of C-D = c(C-D) + c(3) + 0 = 0 + 8 + 0.

Hence we will select the vertex D for expansion. The state space tree is

$\hat{r}$ = 0

Now from this node clearly we have with us the only node remaining and i.e. B. Thus the optimal tour is A-C-D-B-A

OCTOBER - 2020 [135AF] (R16)

Design and Analysis of Algorithms

Solved Paper
B.Tech., III - I [CSE/IT]

Time : 2 Hours] **[Maximum Marks : 75**

Answer any five questions

All questions carry equal marks

Q.1 **a)** *Explain in the general method of divide and conquer with an example.* **(Refer Q.1 of Chapter - 2)**

b) *Write an algorithm for stressan's matrix multiplication and analyze the complexity of your algorithm.* **(Refer Q.26 of Chapter - 2)** **[7+8]**

Q.2 **a)** *List the disjoint set operations and explain with examples.* **(Refer Q.9 of Chapter - 3)**

b) *Explain GRAPH coloring problem with example. Analyze the running time for that problem/algorithm.* **(Refer Q.23 of Chapter - 6 (Running Time : $O(2^n n)$)** **[7+8]**

Q.3 *Differentiate between prim's algorithm and krushkars algorithm for finding the minimum cost spanning tree.* **(Refer Q.20, Q.21 and Q.24 of Chapter - 4)** **[15]**

Q.4 a) *Write an algorithm of all pairs shortest path problem.* **(Refer Q.24 of Chapter - 5)**

b) *Solve the following 0/1 Knapsack problem using dynamic programming*
$P = (11, 21, 31, 33)$, $W = (2, 12, 23, 15)$, $C = 42$, $n = 4$. **(Refer Similar Q.18 of Chapter - 5)** **[7+8]**

Q.5 a) *Compare NP Hard and NP Complete.* **(Refer Q.19 of Chapter - 8)**

b) *Explain about 0/1 Knapsack Problem using branch and bound with example.* **(Refer Q.17 of Chapter - 7)** **[7+8]**

Q.6 *Explain the merge sort algorithm with an example. Design an algorithm for merge sort.* **(Refer Q.20 of Chapter - 2)** **[15]**

Q.7 a) *Explain the major drawbacks of backtracking method with example.*

Ans. : **Following are some major drawbacks of backtracking algorithm**

1) The backtracking suffers from the problem of **thrashing.** Thrashing means repeated failures due to same reason. Thrashing occurs because the standard backtracking algorithm does not identify the real reason of conflicts. Hence during backtracking technique the search is made in different parts of the space tree with failure for the same reason.

2) The overall runtime of backtracking is normally slow.

3) For solving large problem, backtracking technique is not sufficient, it needs to take help of branch and bound method.

4) Backtracking approach is not efficient for solving strategic problems. For example - Consider the traveling salesman problem. It can be efficiently solved using branch and bound.

b) *Write an algorithm for sum of subsets problem.* **(Refer Q.21 of Chapter - 6)** **[8+7]**

Q.8 a) *Write an algorithm for Knapsack problem using Greedy method.* **(Refer Q.14 of Chapter - 4)**

b) *Find the optimal solution by using prims minimum cost spanning of the following graph.* **[7+8]**

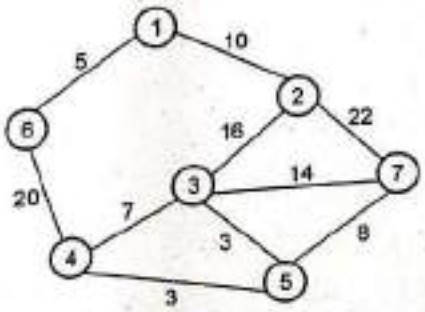
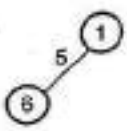


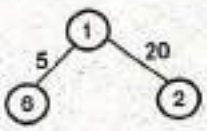
Fig. 1

Ans. :

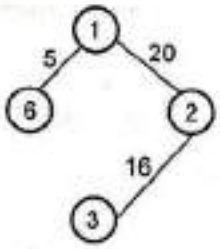
Step 1 : Select edge 1-6



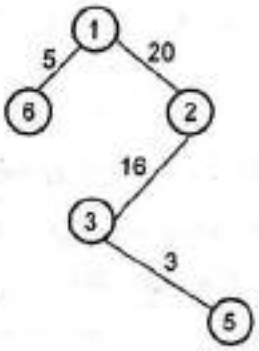
Step 2 : Select edge 1-2



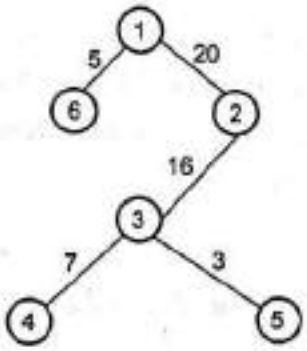
Step 3 : Select edge 2-3



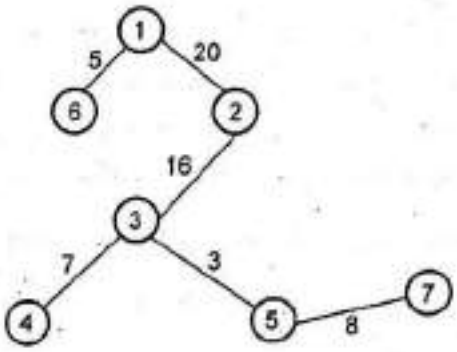
Step 4 : Select edge 3-5



Step 5 : Select edge 3-4



Step 6 : Select edge 5-7



This is the required spanning tree with total cost as 59.

END...