06/01/2020

Definition of Algorithm

An algorithm can be define as the set of instruction when followed in specific sequence accomplishes the desire tasks. In other words the sequencial set of instruction to accomplish any other task is called algorithm.

* Properties of Algorithm

1) Input
2) Output → processed data or processed raw.
3) Definitness → code → ambiguity.
4) Finitness → every algorithm should get terminated;
5) Effectiveness → it can be simply executable using Pen & Paper

1) Input

Ayush.

Whatever a raw that is provided to a system to work upon it is known as input to the system "Each algorithm should be provided with one or more inputs".

Output
After processing raw in the System the outcome of the System is known as output "At least one output must result from any algorithm"

3) Definitness
Each instruction of the algorithm should be clear and unabigious (error free) and should provide the finite result.
For eg: the instruction like Substract 2 or 3 from 5 or 2 by 0 are abigious and provide unpredictiable result.

4) Finitess
Each algorithm must contain finite no. of statements in other words the algorithm must terminate some where.

5) Effectiveness

Each algorithm must be simple to understand & can be runned or executed mannually with the help of pen and paper

imp

* Various Stages of Algorithm

I How to devise the algorithm?
Devising or designing the algorithm is an art in terms of program

II How to validate the algorithm?
Once the algorithm is devised it must be validated to check weather the algorithm will provide the desired output or not for the given set of input data.

III Analysis of the Algorithm
We can have more than one algorithm for a given task out of which we have to choose the best one to achieve this we must analyse various algorithm with respect to memory consumption as well as the execution time

Each program when executed on any computer system requires memory for its execution. The relative space required for the algorithm is known as its space complexity.

Each execution of instruction requires the system time. The time complexity of algorithm is the overall time required to its completion.

IV How to test the Program?

Once the best algorithm is decided with respect to space & time complexity. The algorithm is translated into a program using specific language which can be run on a computer system. This program must be tested for ~~variation~~ various set of input values to get the variation in the output for the performance analysis. The testing involves compilation & execution

* Algorithm Specification

1) Assisgment operator (:=)
(:=) is called the assisgnment operator

2) Each executable statement is terminated by (;) semicolon.

3) Comment (//) :
It is not included in the program it is only used for documentation

4) The data types of the variable must not be specified explicitly in other words variables need not to be declared like (int n).

8/01/2020

For representing the loop do keyword is compulsory.

5) The user define data type are represented using record.

eg
```
Student = record
{
    int rollno;
    char name [10];
};
```

6) The logical operators are : AND, NOR & NOT
Relational operators are : > < ≥ ≤ ≠

Here in algorithm, the array index starts from one.

7) The Syntax for the for loop

```
for var : = initial to final step Stepcount
    do
```

Keyword

Stepcount → increment/decrement

by default value is (+1)

```
for (i=0; i≤n; i++)
for i : = 1 to n step (Stepcount) do
                      +1

for i : = 1 to n step do
for i : = 1 to n step +2 do
        [for increment of +2]
```

for decrement

for i := n to 1 step -1 do

* while loop

Syntax for while loop

```
initialization
while (condition) do
{ ------ Statement 1
  ------ Statement 2
     :
  ------ ..... n
}
```

Example

```
i = 1;
while (i < 10) do
{
  Display ("i");
  i = i + 1;
}
```

1, 2, 3 9

* do while loop

Syntax for do while loop

```
do
{ ------ Statements
  ------
}
while (condition)
```

* Conditional Statements

→ If Syntax :

```
if (condition) then
{
    Statement 1;
    Statement 2;
}
```

→ If Else Syntax :

```
if (condition) then
{
    Statement;
}
else
{
    Statement;
}
```

→ If else if Syntax :

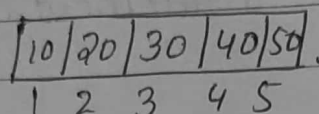
```
if (condition) then
{
    Statements;
}
else if (condition) then
{
    Statements;
}
else
{
    Statements;
}
```

→ How to specify the algorithm heading

Algorithm Algorithm Name (Argument List)

for eg: Algorithm for Linear Search

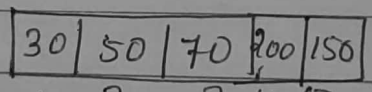
num = 40



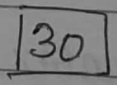
Algorithm Linear Search (A, n, num)

* First Algorithm

Write an Algorithm to find out the maximum element from an array.



max = Arr[1]




```
Algorithm Max-No (Arr, n=5, max)
{ // Arr is an array having n no. of elements
  // max is the output parameter to hold the max value
  max := Arr[1];
  for i := 2 to n step step count do
```

```
{
   if (Arr[i] > max) then
   {
      max := Arr[i];
   }
}
return (max);
}
```

- write an algorithm for the linear search.

50	60	80	10	3	5
1	2	3	4	5	6

num = 3 or n = 2

```
Algorithm Linear_Search (a, n, num)
{ // a is an array having n no. of elements
  // num is the no. to be searched.
     found := 0;
     for i := 1 to n step 1 do
     { if (a[i] = num) then
       { found := 1
         Display ("no. found");
         break;
       }
     }
     if (not found) then
     { Display ("no. is not found");
     }
```

1/01/2020

For push operation check for overflow condition and for POP operation check for underflow cond.

Q Algorithm for Push & POP operation

For this algorithm top should initialize by zero.

```
Algorithm push (S, top, maxsize, n)
{ // maxsize is the max. size of output
  // S is for stack
    if (top = maxsize) then
    {
        Display ("Stack is full");
        return (false);
        break;
    }
    else
    {
    top = top + 1;
     S[top] := val;
    }
    return (true);
  }
}
```

Combination Prob. - we make data set (we make the set of two or more than 2 prob. [eg sorting & searching]

Combinatorial Prob - work in a ~~graph~~ group (eg. graph) (ordered set, cond^n should be applied)

For pop

```
Algorithm Pop (S, top, n, x)
{
    if (top = 0) then
    {
        Display ("Stack is EMPTY.");
        return false;
        break;
    }
    else
    {   x := S[top]
        top := top - 1;
        S[top] = val;
    }
    return (true);
}
}
```

Note

S[top] = val.

val = S[top]

both the Syntax are different

Greedy - when we have one prob by ... dynamic - condition should be applied in algo, i) it ... then only we will go for this technique.
Backtracking - when we are stuck in a prog. then we find the soln like searching diff. route

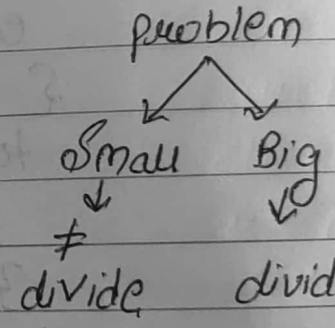
10/01/2020

Algorithm for / Push

There are certain Technique to design an algorithm.

1) Divide & conquer Technique
2) Greedy Strategy.
3) Dynamic Programming.
4) Backtracking.

* 1) Divide & conquer Technique



Problem Statement for divide & conquer Strategy

If we are having a problem 'P' then the divide & conquer method ~~in~~ enable us to divide the problem 'P' in possible ~~case~~ k Sub problem $(P_1, P_2 \dots P_k)$, such that it cannot be divided more or further at this stage. Each Sub problem will be having the same format as that of the basic Problem 'P'. When no further Subdivision

10/01/11

can be obtain for the problem. It provides the Solu^n. to the sub prob. which can be combined again to get the overall problem solved.
The divide & conquered strategy results the recursive solution to the problem. Some example based on this strategy are: binary search, merge sort, quick sort, min & max no., Strassen's Matrix multiplication

Algorithm for divide & conquire

The general algorithm for divide & conquire is ~~are~~ as follows

```
Algorithm D & C (P)
//P is a problem
{
if Small (P) then return (S(P));
}
else
{ Divide problem P into subproblems
i.e P1, P2, .... Pk where k ≥ 1
Apply D & C technique to each subproblem
return (combine S(P1), S(P2), ... S(Pk));
}
```

where P is the basic problem. & S(P) (Small of P) represent the smallest partition of the problem P. S(P) represent the solution to the smallest

13/01/2020

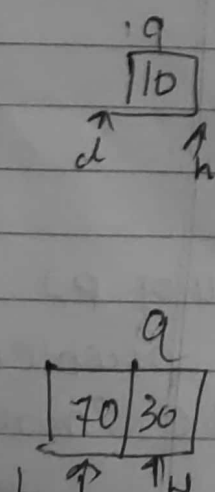
problem. $P_1, P_2, P_3, \ldots P_K$ are the sub-problems. The combine represent the combine solution after applying divide and conquere over all the sub problem. The general time complexity for divide and conquere can be given by the recurrance relationship

$$T(P) = \begin{cases} g(n) & \text{for small problem} \\ T(P_1) + T(P_2) + T(P_3) + \cdots T(P_K) + f(n) & \text{Otherwise} \end{cases}$$

$T(P_1) + T(P_2) + \cdots + T(P_K)$ are the time complexity for the respective subproblems. $f(n)$ is the time required to divide & combine the solution. $g(n)$ represent the time for the smallest Problem.

→ Algorithm for min & max no using Divide & conquer strategy

```
Algorithm MinMax(a, l, h, min, max)
{ // a is the array, l is the lower index
    h is the higher index, min & max are
    the output to be found
  if (l = h) then // small(P)
  { min := a[l];
    max := a[l];
  }
  else if (l = h-1) then // Another case for
                            small(P)
```

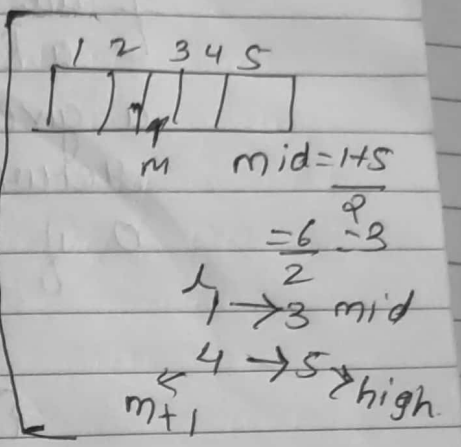


30 70

30/70

```
{
  if (a[i] < a[h]) then
  {
      min := a[i];
      max := a[h];
  }
else
{   min := a[h];
    max := a[i];
}}
else
{ // Divide P into subproblem P1, P2 .... Pk
    mid := (i+h)/2;

    min max (a, i, mid, min, max);
    min max (a, mid+1, h, min1, max1);
    if (min > min1) then // combine the subproblem
    {   min := min1;
    }
    if (max > max1) then
    {
        max := max;
    }
```




```
or  if (max1 > max)
    {  max := max1;
    }
```

Problem Statement for binary Search

If we are provided with list of elements a[1------n], the problem is to Search a given element x in the list a, and to return the value if found and return 0 if not found.

```
Algorith Binary Search (a, l, h, x)
{   // a is the array
    // l is the lower index
    // h is the higher index
    // x is the no. to be found.
case1 : if (l=h) then
    {
        if (a[l] =x) then return (l);
    else
    {
        return 0;
    }
}
else
{ mid:= (l+h)/2;
    if (a[mid] =x) then
    {
        return (mid);
    }
```

```
else if (a[mid] < x) then
{ }
    Binary Search (a, mid+1, high, x);
}
else
{
    Binary Search (a, l, mid-1, x);
}
}
```

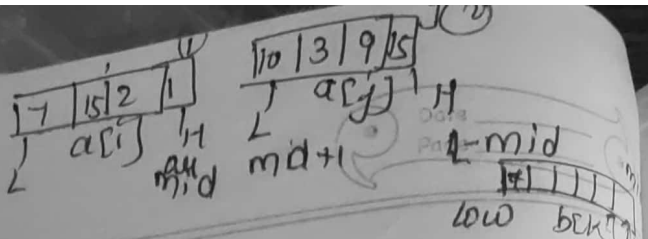
→ Devise an algorithm for merge Sort using divide and Conqure technique.

```
Algorithm Merge Sort (a, low, high)
{
    if (low < high) then
    { }
        mid := low + high / 2;
        merge Sort (a, low, mid);
        merge Sort (a, mid+1, high);
        merge (a, low, mid, high);
    }
}
```

Divide

```
Algorithm Merge (a, low, mid, high)
{ i := a[low];
  j := mid+1;
  k := b[low];
```

Con
qu




```
while (i ≤ mid && j ≤ high) do
{
  if (a[i] ≤ a[j]) then
  {
     b[k] := a[i];
     i := i+1;  k := k+1;
  }
else
{    b[k] := a[j];
     j := j+1;
  }
     k := k+1;
  }
if (j ≥ high) then
  while (i ≤ mid)
{
   b[k] := a[i];
   i := i+1;
   k := k+1;
}
if (i ≥ mid) then
{  while (j ≤ high) do
   {
     b[k] := a[j];
     j := j+1;
     k := k+1;
   }
```

conquer

for k:= low to high step 1 step count do
// transferring elements from temperary to original array.
{
a[k] := b[k];
}

9 Devise an algorithm for quick Sort using divide & Conquer technique

Algorithm Quick Sort (a, ~~high~~ low, high)
{
low := l;
high := h;
mid or key := a[low+high]/2;
do
{
while (key > a[low]) do
{
low := low + 1;
}
while (key < a[high]) do
{
high := high - 1;
}
if (low < high)
{
temp := a[low];
a[low] := a[high];
a[high] := a[temp];

```
        low := low + 1;
        high := high - 1;
    }
}
while (low <= high);
    if (low < high)
{
    Quick Sort (a, l, high);
}
if (low < h)
{
    Quick Sort (a, low, h);
}
}
```

20/01/2020

*

20/01/2020

* Problem Statement

Quick Sort is a Sorted algorithm which is commonly used in computers science. It give result using divide & conquire technique. It creates a 2 empty arrays to hold in element less than the pivot value & the elements greater then the pivot value and then recursively Sort the Sub arrays the other thing for it is the Partition or the exchange Sort.

* Perform Quick Sort on the following data values
35, ~~33~~, 42, 10, 14, 19, ~~27~~, 44, ~~26~~, 31, ~~44~~,

(35) 33 42 10 14 19 44 31 ←

→ 31 33 42 10 14 19 44 (35)

31 33 (35) 10 14 19 44 42 ←

31 33 19 10 14 (35) 44 42

14 33 19 10 31 (35) 42 44

10 33 19 14 31 (35) 42 44

↑

21/01/2020

Vimp

10 14 19 33 31 (35) 42 44,

10 14 19 31 33 (35) 42 44

10 14 19 31 33 35 42 44

21/01/2020

Imp Quick Sort using Divide & conquer strategy

```
Algorithm Quick Sort (l, h, a)
{
  if (low < high) then
{
  j := partition (a, low, high);
    // j is pivot
  Quick Sort (a, low, j-1);
  Quick Sort (a, j+1, high);
}
}
Algorithm Partition (a, low, high);
{
  temp := a[l];
    i := low;
    j := high;
  repeat
  {
    repeat ;
  {   i := i+1;
  }
 while (a[i] ≤ temp);
  repeat
    { j := j-1
  }
    until (a[j] ≥ temp);
}
```

Strassen's matrix multiplication

```
if (i<j)
{
  Interchange (a,i,j)
}
} until (i>j);
  a[i]:= a[j];
  a[j]:= temp;
  return j;
}
```

Algorithm interchange (a, i, j)

```
{
  temp:= a[i];
  a[i]:= a[j];
  a[j] := temp;
}
```

Strasses Matrix Multiplication - (Applicable for only 2x2 matrix)

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

$$\begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}_{2\times 2}$$

Total mul = 8
Add = 4

22/01/2020

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

A(N×N) B(N×N)

$\frac{N}{2} \times \frac{N}{2}$ $\frac{N}{2} \times \frac{N}{2}$

D&C

C $8 \to M$

$4 \to A$

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

Strassen's Matrix Multiplication

$$7T\left(\frac{n}{2}\right)$$

→ Algorithm for Strassen's matrix multiplication

1) Divide the input matrices A & B into $n/2 \times n/2$ sub matrices

2) Using Scalar addition and Substraction each of which is $n/2 \times n/2$

3) Recursively, compute seven matrix product. (i.e) $P_i = A_i B_i$ for all $i = 1$ to 7

4) Compute the desired sub matrices (R, S, t, U) of the resultant matrix (C) by adding or Substracting various combination of P_i matrices.

$AB = C$ of $n/2 \times n/2$

Submatrices

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_{2\times2} \begin{bmatrix} e & f \\ g & h \end{bmatrix}_{2\times2} = \begin{bmatrix} r & s \\ t & u \end{bmatrix}_{2\times2}$$

eg

Note This method is used to multiply (2x2) matrices o
In this we perform 7 multiplication and 18
addition or substraction because addition
or substraction is easier than multiplicat

The formula use to compute the resultant is as follows

$P = (A_{11} + A_{22})(B_{11} + B_{22})$
$B = (A_{21} + A_{22}) B_{11}$
$T = (A_{12} + A_{11}) B_{22}$
$U = (A_{21} - A_{11})(B_{12} + B_{11})$

$R = A_{11}(B_{12} - B_{22})$
$S = A_{22}(B_{21} - B_{11})$
$V = (A_{12} - A_{22})(B_{21} + B_{22})$
C_{11} or $r = P + S - T + V$
C_{12} or $s = R + T$
C_{21} or $t = Q + S$
C_{22} or $u = P + R - Q + U$

through d/c technice $T\frac{n}{2}+O(n^2)$
Strassens matrix mult $7T\frac{n}{2}$ bez it focus on add & sub bez it is valid for 2x2 matrix

eg

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

A B Pore

1 Solve the recurrence relation ship

$$T(n) = \begin{cases} 1 & n=0 \rightarrow \text{best} \\ T(n-1)+1 & n>0 \rightarrow \text{Av/worst} \end{cases}$$

↳ Recursive func^n

when the term is form then it form recurrence relationship.

$T(n) = T(n-1)+1$ — (1)

put $n = n-1$ in eq^n (1)

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

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

Substitute $T(n-1)$ in eq^n (1)

$T(n) = T(n-2)+2$ — (2)

put $n = n-2$ in eq^n (1)

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

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

Substitute in eq^n (2)

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

$T(n) = T(n-3)+3$ — (3)

Kth Stage

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

Assume, $n-k=0$

$n=k$

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

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

$T(n) = 1+n$

for n.

$O(n)$

8 $T_n = \begin{cases} 2T(n-1)+3 & \\ 1 & n=1 \end{cases}$

$T_n = 2T(n-1)+3$ — (1)

Put $n = n-1$ in eqn (1)

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

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

Substitute $T(n-1)$ in eqn (1)

$T_n = 4T(n-2)+6$ — (2)

put $n = n-2$ in eqn (1)

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

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

Substitute $T(n-2)$ in eqn (2)

$T(n) = 4T\,2T(n-3)+9$

$T(n) = 8T(n-3)+9$

$2^k(n-k)$

Put $n = n-3$ in eqn (1)

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

$T(n-3) =$

k^{th} stage

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

let $n-k=0$

$n=k.$

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

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

Put $n = n/2$

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

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

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

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

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

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

Put $n = \frac{n}{4}$ in eq ①

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

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

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

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

k^{th} iteration

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

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

$$= n \cdot 1 + n \log n$$

$$= n + n \log n$$

$$2^k T\left(\frac{n}{2^k}\right) = T(1)$$

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

Taking log on both side

$\therefore \log n = k.$

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

$T(1) = 1$

$Tn = 3T(n-1) + 1$ — (1)

Put $n = n-1$ in eqn (1)

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

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

Substituting $T(n-1)$ in eqn (1)

$T(n) = 9T(n-2) + 4$ — (2)

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

Put $n = n-2$ in eqn (1)

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

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

Substituting T(n-2) in eq 2

$T(n) = 3T(n-3) + 3 + 4$ — (3)

$3^2 T(n-3) + 3^2 + 3^1 + 3^0$

Put $n = n-3$ in eqn (1)

$T(n-3) = 3T(n-4) + 1$

Substituting T(n-3) in eqn (3)

$T(n) = 3^4 T(n-4) + 4$ — (4)

Kth iteration.

$3^K + T(n-K) + \sum_{i=0}^{K-1} 3^i$

$n - K = 1$

$\therefore K = n-1$

$T(K) = 3^{n-1} T(n-n+1) + \sum_{i=0}^{n-1-1} 3^i$

$= 3^{n-1} T(1) + \sum_{i=0}^{n-2} 3^i$

$= 3^{n-1} \cdot 1 + \sum_{i=0}^{n-2} 3^i$

$3^0 + 3^1 + 3^2 +$

Sum of n terms of GP

$S_n = \frac{a \cdot (r^n - 1)}{r - 1}$

ratio

$\frac{3}{1} = \frac{3^2}{3} = 3$

24/01/20

$$T(K) = 3^{n-1} \cdot T(1) + \sum_{i=0}^{n-2} 3^i$$

$$3^0 + 3^1 + 3^2 + \cdots + 3^{n-2}$$

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

$$r > 1$$

$$a = 1, \quad r = 3$$

$$S_n = \frac{1(3^{n-1} - 1)}{3 - 1}$$

$$S_n = \frac{3^{n-1} - 1}{2}$$

$$T(n) = 3^{n-1} \cdot 1 + \frac{3^{n-1} - 1}{2}$$

$$= \frac{2 \cdot 3^{n-1} + 3^{n-1} - 1}{2}$$

$$\frac{3^{n-1}(2+1) - 1}{2}$$

$$= \frac{3 \cdot 3^{n-1} - 1}{2}$$

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

$$T(n) = \frac{3^n - 1}{2}$$

Greedy
↓
without backtracking

Feasible Soln (set of shops online store)
Optimal Soln (selected by user)

UNIT - 2

GREEDY STRATEGY

```
Algorithm Greedy (a, n)
{
    Solution := ∅;
     x := Select (a);
for i := 1 to n step 1 do
    {
      if feasible (Solution, x) then
      {
        Solution := Union (Solution, x);
      }
}
  return (Solution);
}
```

* Greedy Strategy

If we are provided with finite set of value than the greedy strategy specifies the selection of all the i/p one by one checking them wether their physical solution for the given problem or not. Is they are possible they will be added to the solution to the problem and some optamization condition can

be applied to get optimized Solution. The general greedy algorithm is explain above

Devise an algorithm for Searching an element in Binary Search tree using greedy Strategy.

Non recursive approach for Binary Search tree

```
Algorithm Search BST (A, x)
A is an array x is the element to be Search.
{
    temp := root;
   found := false;
while ((temp ≠ 0) and not found)
  {
   if (temp [info] = x) then
   { {
       found := true
       break;
    }
  else if (temp [info] > x) then
   { }
    temp := temp [lchild];
   }
  else
   {
    temp := temp [rchild];
   }
}
```

```
if (not found) then
{
Display ("No. not found");
}
else
{
return x;
}
```

* Recursive approach for Binary Search

```
Algorithm Search BST (root x)
{
if ((root ≠ x) AND root = 0) then
{
return (0);
}
else if (root [info] = x)
{
return (root);
}
else if (root [info] > x) then
{
Search BST (root [child], x);
}
else
{
Search BST (root [r child], x);
}
```

* Devise a recursive Algorithm for insert value in BST

```
Algorithm Insert BST (root x)
{
if (root = 0) then
  {
    root := new free nodes);
  OR
    root := (Tree *) malloc
                (sizeof (Tree));
      root [info] := x;
      root [lchild] := 0;
      root [rchild] := 0;
  }
}
Insert BST (root, x)
  {
    if root
```

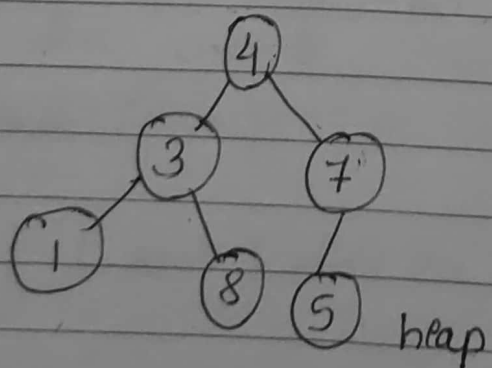
It is the type of BST

* Heap Sort
Heap is also known as priority queue it can be represent by binary tree with the following properties

* Structure
A heap is completely filled Binary tree which is filled from left to right.

II) Heap order property :-
For every node x. In the heap the parent x <= the value of x it is.

Ex Apply heap sort for the following element
4, 3, 7, 1, 8, 5

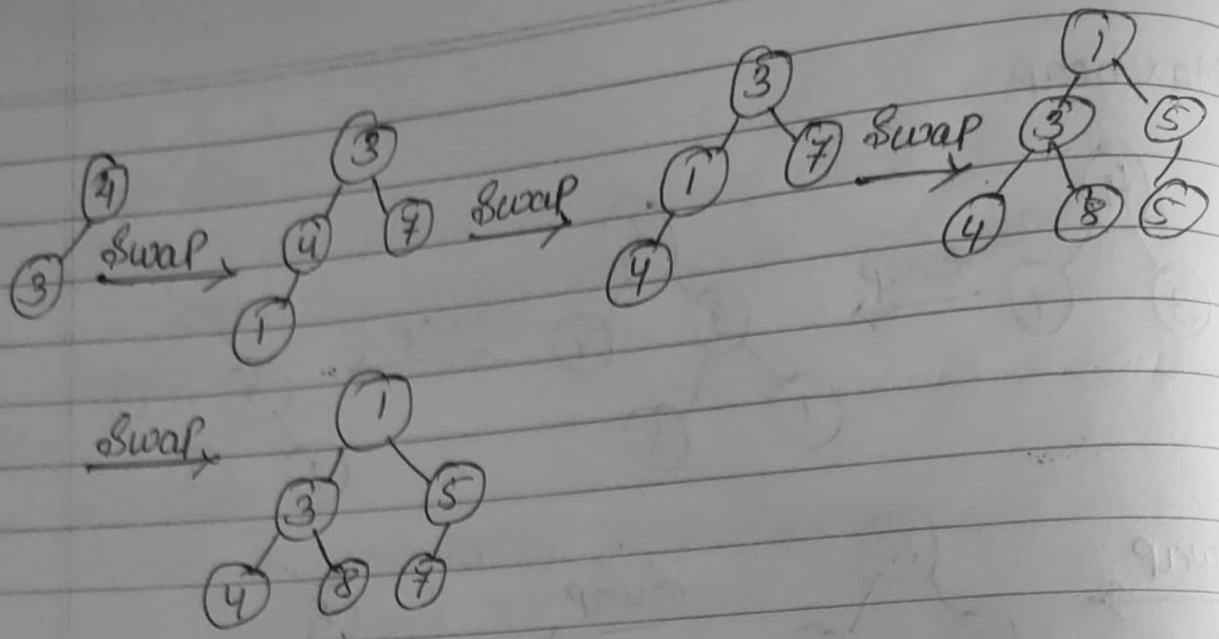


heap

1) Max heap

2) Min Heap :-

Deletion → Last level
Structure → Right to left
Property



Min Heap

Q1) Algorithm Heap Sort

① Build Heap :- complexity $-O(n)$

Build Binary Tree taking an items as i/p, and inserting the heap structure properly held. In other words build complete binary search tree.

Heapify the binary tree making sure the tree satisfies the heap order property

② perform delete Max operation :-

The complexity of tree is for deletion $O(\log n)$

Booting process System & what is the diff b/w bomb & cold booting

Delete the maximum element from the heap which is the root node one place this elements at

Advantage

① M/u efficient is faster than the merge Sort but Slow than quick Sort algorithm Specially for very large value of n.

Disadvantage

① It is non-Stable case is inefficient and relatively Slow than quick Sort means uses of heap

There are two main uses of Heap.

① Is to implement a Special kind of queue called the priority queue in ordinary queue elements are added and one end and deleted in other end. In priority queue each element has priority when an element is remove it must have highest priority. In heap Sort each node has higher priority then below. The highest priority in the top

② Sortings:

Heap Sorted uses heap data Structure to Sort the elements. To Sort an array. There are two Steps

i) Insert each value in the heap

ii) Remove each value in the Heap

(1) Highest the no. highest
(2) Lowest the no. highest the priority

* Algorithm Heap - Sort (a, n)
{
Heap_ify (a, n); // create Heap
// x from array to heap

for i := n to 2 step -1 do
{
t := a[i]; //for Swapping new max
a[i] := a[1]; element
a[1] := t;
adjust (a, 1, i-1);
}
}

* Creating Heap out of n elements

Algorithm - Heapify (a, n)
// n is the current no. of elements
{
for i := n/2 to 1 step -1 do
{
Adjust (a, i, n);
}
}

Deletion from Heap
Algorithm Adjust (a, i, n)

```
{
 j := 2*i // j is the left child

 val := a[i];
 while (j<=n)
{
 if (j<n and a[j] < a[j+1])
     // which child is greatest
{
   j := j+1;
}
 if (a[j] < val)
 {
      a[j/2] := a[j];
 // I] value is already greater than I will not get
swap
 // I] this condition get false that means it still
want swaping. Here we are performing
swaping of elements.
   j := 2*j;
}
  a[j/2] := val;
}
```

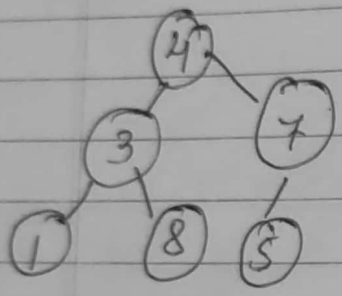
Left child = $2i+1$

Right child = $2i+2$

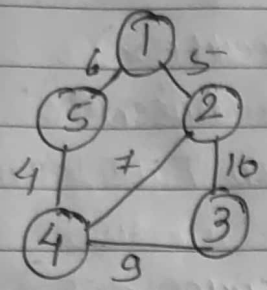
parent (root) = $2i$

parent = $\frac{i-1}{2}$.

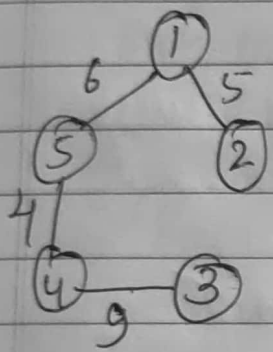
4, 3, 7, 1, 8, 5



* Minimum Spanning trees



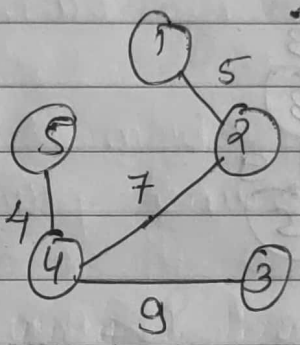
(I) Kruskal



Cost = 6+5+4+9
= 24

(II) PRIM'S

Source node



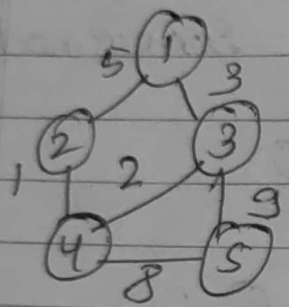
Cost = 4+5+7+9
= 25

* Minimum Spanning Tree :-
The method process or algorithm are use to solve this problem is known as prim's and kruskal Algorithm.

1) PRIM'S Algorithm

① Label all the vertices as unchosen
② Let T consist of n, m of nodes initially ~~with~~ with no edge

③ Chose any arbitary vertex 'is' & level it is choosen.
& while (there is an unchosen vertex)
pick the lightest edge b/w any chosen 'u'
unchosen 'v' label vs, as choosen $T = T + \langle u, v \rangle$
if ($\langle u, v \rangle$ does not give rise to cycle in T)



* Krushkal Algorithm

Krushkal Algo uses greedy approach because it goes on choosing the highest edge and choose the next highest edges all edges are sorting in ascending order whi choosen the next edge

$T = \phi$ // T is solution vertex or tree

Vs = Set of vertices

iii) Construct a priority queue containing all the edges in E in ascending order

iv) For each V of Vs add {V} to Vs

v) { while mode of (|Vs| > 1)
choose <v,w> an edge E of Q with least weight delete <v,w> from Q.
if (<v,w> are in different sets w_1 and w_2 in Vs)

{
Replace w_1 & w_2 in Vs by $w_1 \cup w_2$ add <v,w> to J;
}

if (v & w are in same set)
it suggest a cycle
discard <v,w>
}

100

Date
Page

Knapsack Problem : (Fractional Knapsack)

If we are given with n object & a bag or a knapsack of capacity m, each object is allocated own the weight (wi) such that $i \leq i \leq n$. If a fraction (x_i) where x_i s to the range is inserted in the knapsack then the profit ($P_i x_i$) is gain on own carried

The problem is to fill the bag satisfying the following property.

i) Maximize $\sum P_i x_i$
$1 \leq i \leq n$

ii) Subjected to $\sum w_i x_i \leq m$
$1 \leq i \leq n$

where x_i is the fraction pack value $0 \leq x_i$

Knapsack Fraction

Profit by weight Ratio

Q) capacity = 20

	P	W	P/W
n_1	25	18	1.38
n_2	29	15	1.6
n_3	15	10	1.5

$x_2 = 1.0$ $V = 5kg$

$x_3 = 0.5 = \frac{5}{10} = 0.5$

Total profit

$= 29 + 0.5 \times 10 \times 0.5$

$= 29 + 7.5$

$= 31.5$

Fraction = $\frac{\text{Remaing weight}}{\text{Actual weight}}$

Q)

	P	W	P/W
n_1	28	15	1.866
n_2	20	12	1.666
n_3	15	10	1.5

capacity = 30

$x_1 = 1.0$

$V = 15kg$

$x_2 = 1.0$

$V = 3kg$

$x_3 = 0.3 = \frac{3}{10} = 0.3$

Total profit $= 28 + 20 + 0.3 \times 10 \times 1.5$

$= 52.5$

$m = 60$

	Item	Profit	weight	P/w
			50	7
	A	280	10	10
	B	100	20	6
	C	120	24	5
	D	120		

Arrange the in decending order

	P	W	P/w
B	100	10	10
A	280	40	7
C	120	20	6
D	120	24	5

B = 1.0 U = 50kg

A = 1.0 U = 10kg

C = 0.5 U = fraction $\frac{10}{20} = 0.5$

Total Profit = $100 + 280 + [illegible] \times 120 \times \frac{10}{20}$

Total Profit = 440 Rs

Total weight = $10 + 40 + 20 \times \frac{10}{20}$

$= 60$

* Algorithm Greedy Knapsack (n, m)

```
{
for i := 1 to n step 1 do
{
    x[i] = 0.0;
}
    U := m;
for i := 1 to n step 1 do
{
  if (w[i] ≥ U) then break;
  // After false condition calculate fraction
    x[i] := 1.0;
    U := U - w(i);
}
  x[i] := U / w(i);
}
```

Spanning tree is the minimum tree of sub graph.
If and only if they follows following condition.

(i) No. of vertices in spanning tree equals to no. of vertices of graph.

(ii) Tree has minimum no. of edges.

(iii) It should be connected i.e. It should use all vertices and there should be no cycle.

(iv) Any connected graph with n vertices must have atleast n-1 edges and all these connected graph with n-1 edges are in the tree.

So the feasible solution for the spanning tree the final graph must connected with no cycle.

(vi) The optimal solution for spanning tree it is the feasible solution with the minimum cost

* Difference b/w prims and kruskal

Prims	Kruskal
1) prims algorithm initiate with a node	1) The algorithm initiate with an edge
2) In prim's algorithm graph must be connected	2) while a known function can work on disconnected to graph
3) The prim's have the complexity O(V2)	3) Kruskal's have O (log V)

* Disadvantage of prims

1) The list of edges has to be search from the begining get added.

11) If there are more than one edge have same weight then all possible spanning tree are require to found for the final tree

	P	w	P/w
	1	2	0.5
	2	3	0.66
	5	4	1.25

	P	w	P/w	
n_3	5	4	1.25	$x_3 = 1.0$ $m = 6$ $U = 2$
n_2	2	3	0.66	$x_2 = 2/3 = 0.66$
n_1	1	2	0.5	X

Total weight $= 4 + 3 \times 0.66$

$= 5.98 \approx 6$

Total Profit $= 5 + 2 \times 0.66$

$= 6.32$

$= 6$ Rs.

* Job Sequencing ~~Grade Line~~ with dead line

If we are provided with n job & with each & job two parameters are associated first is the profit & other is the dead line. We are provided with a single machine only The problem is to find a set of jobs which can be carried out with maximum profit in the given dead line. The profit (P_i) can be achieved only if the respective task is completed in the given dead line (d_i)

For given set of 'n' jobs there are 'n' deadli
& 'n' profits to be earned for any i^{th} jo
there is a deadline $(d_i) > 0$ and $(P_i) > 0$
Profit (P_i) is gained if job is completed with
its deadline.

* Feasible Solution
Feasible Solution = Set of jobs such that each job that can be completed by its deadline and value of feasible solution is sum of the points

* Optimal Soln
It is the feasible Soln with the maximum Profit

Eg

Jobs	J_1	J_2	J_3	J_4
Profit	100	10	15	27
Deadline	2	1	2	1

Step I Arrange in decending order according to Profit

Job	J_1	J_4	J_3	J_2
Profit	100	27	15	10
Deadline	2	1	2	1

Step 2 Now
the
the
with
line
Se
w
ha

42

Step 1

Step 2: Now maximum deadline is 2 So we select the jobs in such a way that we can get the optimal solution So, select first job with the deadline 1 and other job with dead line 2. To get the optimal solution we select job J_4 with dead line 1 and job J_1 with deadline 2, because these jobs are having maximum profit then others

Jobs	J_4	J_1
Profit	27	100
Deadlines	1	2

$J = \{J_4, J_1\}$

∴ Total profit $= 100 + 27$
$= 127$

Q 42. $n = 6$ $\{J_1, J_2, J_3, J_4, J_5, J_6\}$
Profit $\{20, 15, 10, 7, 5, 3\}$
Deadline $\{2, 1, 1, 3, 1, 3\}$

Step 1: ~~Arrange in decending order~~ ~~are to profit~~

Job	J_1	J_2	J_3	J_4	J_5	J_6
profit	20	15	10	7	5	3
Deadline	2	1	1	3	1	3

Step1 Arrange the jobs in ascending order

Job	J_6	J_5	J_4	J_3	J_2	J_1
Profit	3	5	7	10	15	20
Deadline	3	1	3	1	1	2

Step2

Job	J_2	J_1	J_4
Profit	15	20	7
Deadline	1	2	3

$J = \{J_2, J_1, J_4\}$

Total profit = 15 + 20 + 7
= 42

* Algorithm for job sequencing

Algorithm Job Sequencing (d[1..n], P[1..n], n)

// d[1..n] - represents the deadline array
// P[1..n] - Profit array
{it should be in decending order}
// n - no. of jobs

{

S = {1}; // Solution vector.

```
for i := 2 to N step 1 do
{
    if all jobs in SU {i} scheduled by
        deadline S = SU {i}
    return S;
}
}
```

* Huffman Coading

It is a lossless data compression technique generating variable length code for different symbols, it is based on greedy approach which consider frequency of alphabets for generating the codes. It has the complexity $n \log n$, where

n is the no. of unique characters

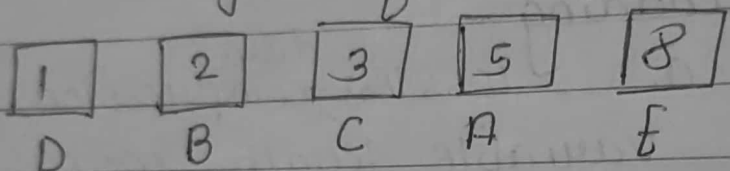
The length of the code is inversely proportional to frequency of its occurance.

Huffman code is use to convert fixed length codes into variable length codes which results in lossless compression. Variable length code may be further compressed using JPEG (Joint picture Group) and MPEG (Moving picture Expert Group) techniques to get desired compression ratio.

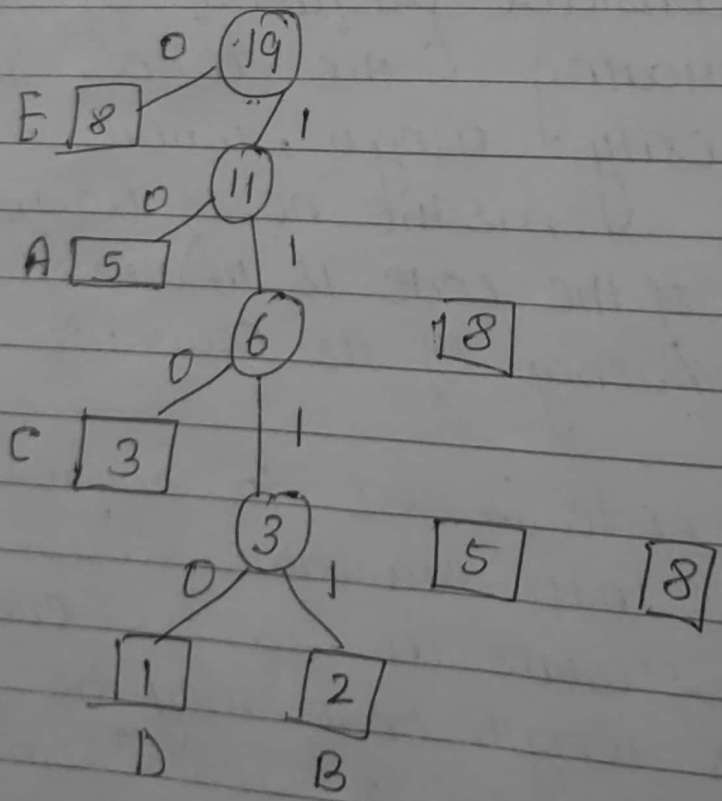
Q Using Huffman technique find out the codes for each characters

Data	A	B	C	D	E
frequency	5	2	3	1	8

Arrange elements in ascending order according to frequency



→ External file



L → 0

R → 1

Code

A → 10
B → 1111
C → 110
D → 1110
E → 0

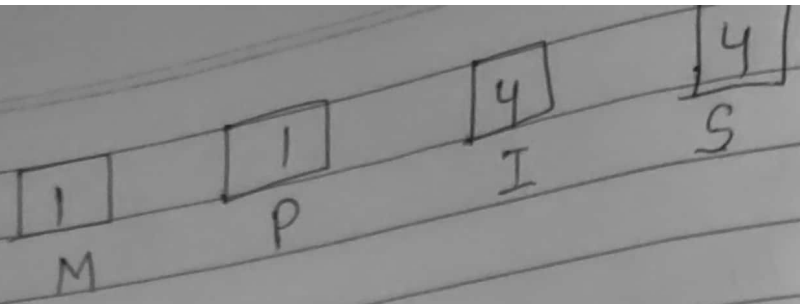
Code length

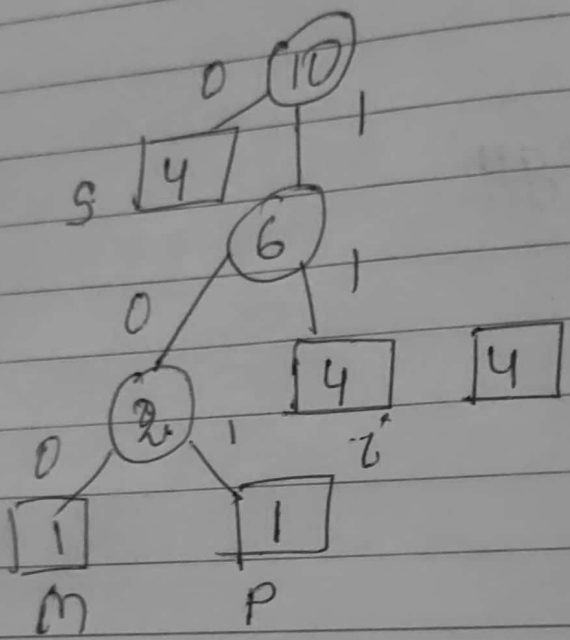
A → 2
B → 4
C → 3
D → 4
E → 1

Q Apply the Huffmann Encoding for the word MISSISSIPPI.

DATA	M	I	S	S	I	S	S	I	P	I
frequency.	1	4	4	4	4	4	4	4	1	4

Arrange elements in ascending order.





code	code length
S → 0	1
I → 11	2
M → 100	3
P → 101	3

Total no. of bits = frequency of X code length
character

Total no. of bits = $1 \times 3 + 4 \times 2 + 4 \times 1 + 1 \times 3$

$= 3 + 8 + 4 + 3$

$= 18$

Average no. of bits = $\frac{\text{total no. of bits}}{\text{total no. of freq. of chars.}}$

$= \frac{18}{10}$

$= 1.8$

8) The characters A to H have set of frequencies

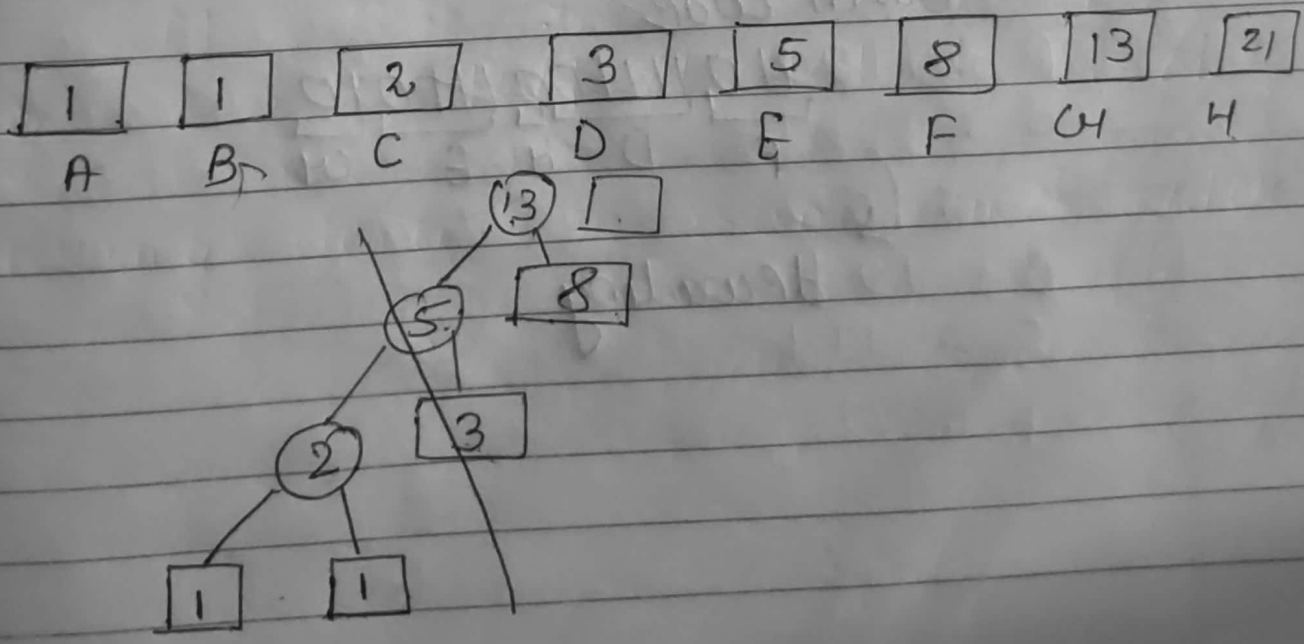
A = 1 E = 5
B = 1 F = 8
C = 2 G = 13
D = 3 H = 21

what is the sequence of characters corresponding to following code

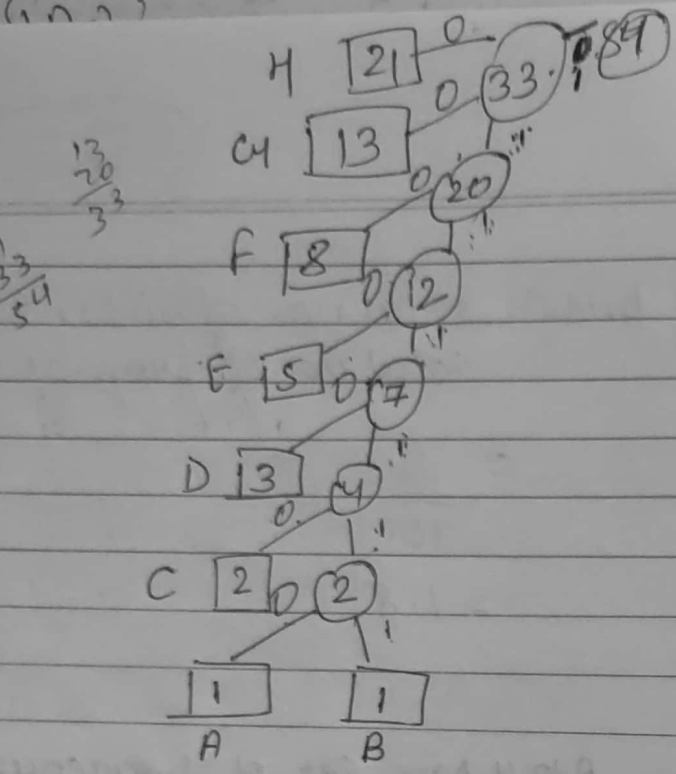
110, 1111, 00, 111, 010

D

Arrange elements in Ascending Order.



13
13
26



code

A = 1111110
B = 1111111
D = 111110
E = 1110
F = 110
CH = 10
H = 0

Given code

110 1111 00 1110 10

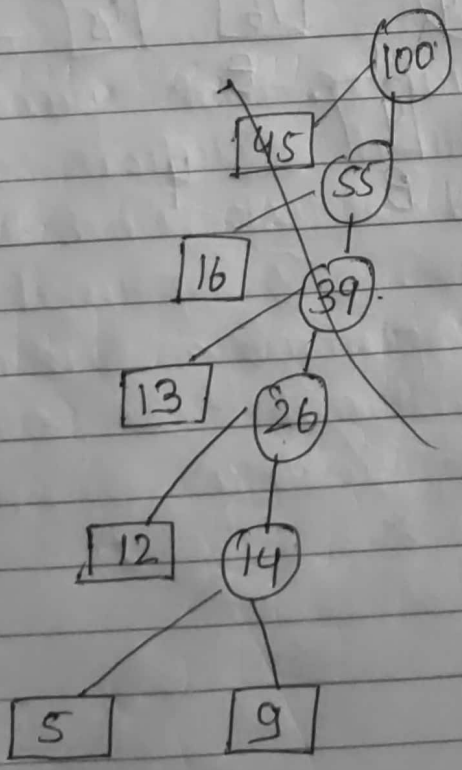
F D H E CH

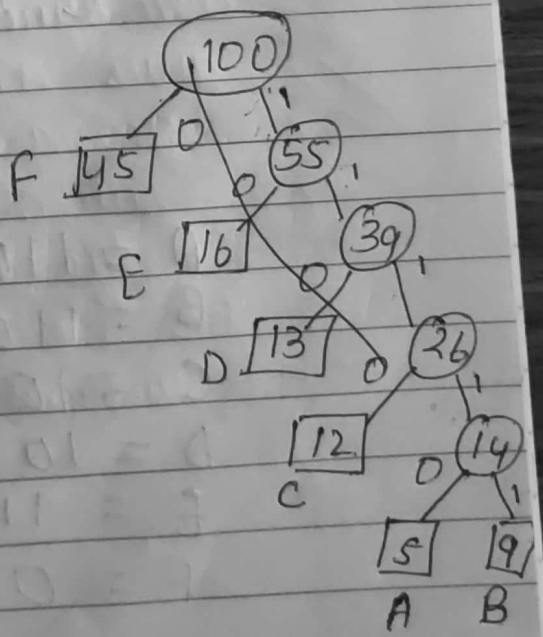
decoding

A networking company uses a compression technique to encode the message before transmitting over the network. Suppose the message contain the following characters with their freq. Char. are A, B, C, D & E & F & freq. are 5, 9, 12, 13, 16, 45 Note that each character in input message take 1 bits. If compression technique is used, how many bits will be saved in the message.

Arrange the element in ascending order.

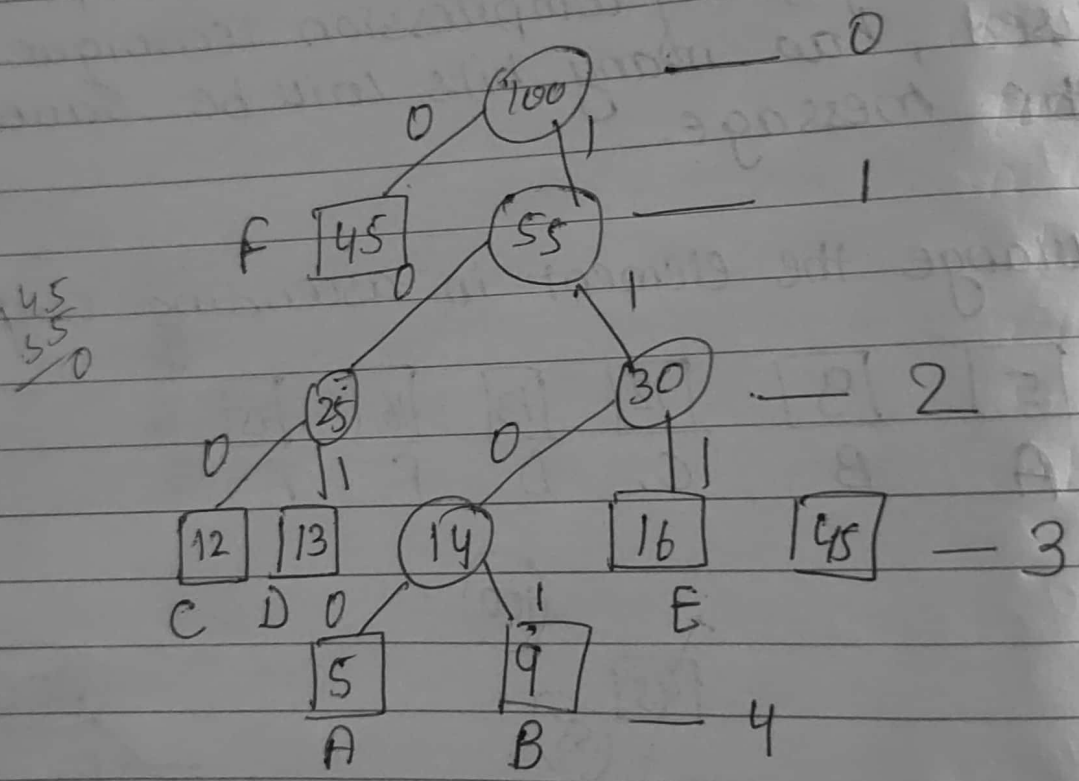
5	9	12	13	16	45
A	B	C	D	E	F





Code

A = 11110
B = 11111
C = 1110
D = 110
E = 10
F = 0



45
55
100

A = 1100
B = 1101
C = 100
D = 101
E = 111
F = 0

If we are not using Huffmann coding.

Total no. of char. = Total no. of freq = 100

1 byte = 8 bits

Total memory = 100 x 8

= 800 bits

Using Huffmann coding

(45 x 1) + (12 x 3) + (13 x 3) + (16 x 3) + (5 x 4) + (9 x 4)

= 224 bits.

Remaining = 800 - 224

= 576 bits save.

UNIT-3

DYNAMIC PROGRAMMING

Difference b/w Greedy Strategy & dynamic Programming

Both are build there Soln by collection of choices of individual elements. Both known as optimization technique:. The greedy method uses the top down approach to solve the given problem. It finds the solution in straight forward fashion. On the other hand it finds the solution by using previous smaller subinstances

We cannot garuntee that greedy method gives an optimal solution. i.e there is no any kind of test which tells the solution obtained is optimal or not, but in dynamic Programming we can find out the soln is optimal or not by the principal of optimality.

Dynamic Programming

It is one of the optimization technique, improvement over divide & conquer, greedy strategy. It is most popular method in designing algo. bez greedy strategy may not produce optimal solution every time and in divide and conquer subinstences are created and combined and there is a possibility that these subinstances may overlap

The basic idea is to avoid calculating the same subinstances ~~to a~~ two, or more no. of times by keeping a table of known result & table is filled up as subinstances are solved. It is bottom up technique. The dynamic programming starts with the smallest and simplest subinstances.

* D/b divide & conquer and Dynamic Program.

Both technique divide their input problem into parts as subproblem and obtain the solution by combining all solution of subproblem

Divide & conquer method divides the given problem at specific point usually at the middle. while there is no specific pt. to

decide to divide the problem. It splits the problem around optimal point.

* Principle of optimality

It is the method for finding an optimal soln used by the dynamic programming. The principle of optimality applies only when the optimal soln to a problem always contain optimal solution to all the subproblem. The real life application of dynamic programming are in robotic control and in air traffic controller. It is also used to maximize the CPU efficiency.

* problems that can be solved using dynamic programming are:-

1) Multistage graph
2) Travelling sales man problem
3) Zero / one knapsack problem
4) All pair shortest path problem.
5) Floyd warshall problem.

mcl

* Multistage graph problem

Problem Statement for the multistage graph

A multistage graph $(G) = (V, E)$ is a directed graph in which the vertices are subdivided into $K \geq 2$ disjoint subset (V_i) where i ranges from $1 \leq i \leq K$. Also if $\langle v, v \rangle$ is an edge of the graph G where $v \in V_i$ then v must belongs to V_{i+1} $(v \in V_{i+1})$. In multistage graph the condition $|V_1| = |V_K| = 1$ must satisfy i.e the first and the last stage of the graph must contain single vertex

Let s is the source such that $s \in V_i$ and t is the sink (destination) which belongs to V_K, then we have to find the shortest path from source s to sink t.

To obtain the solution K-2 decisions must be taken and at each stage (V_i) we have to decide the vertex at stage (V_{i+1}) which should be on the shortest path. The cost can be calculate by the given formula.

$$Cost(i, j) = m\{c[j, l) + cost[i+1, l]\}$$

where

$l \in V_{i+1}$

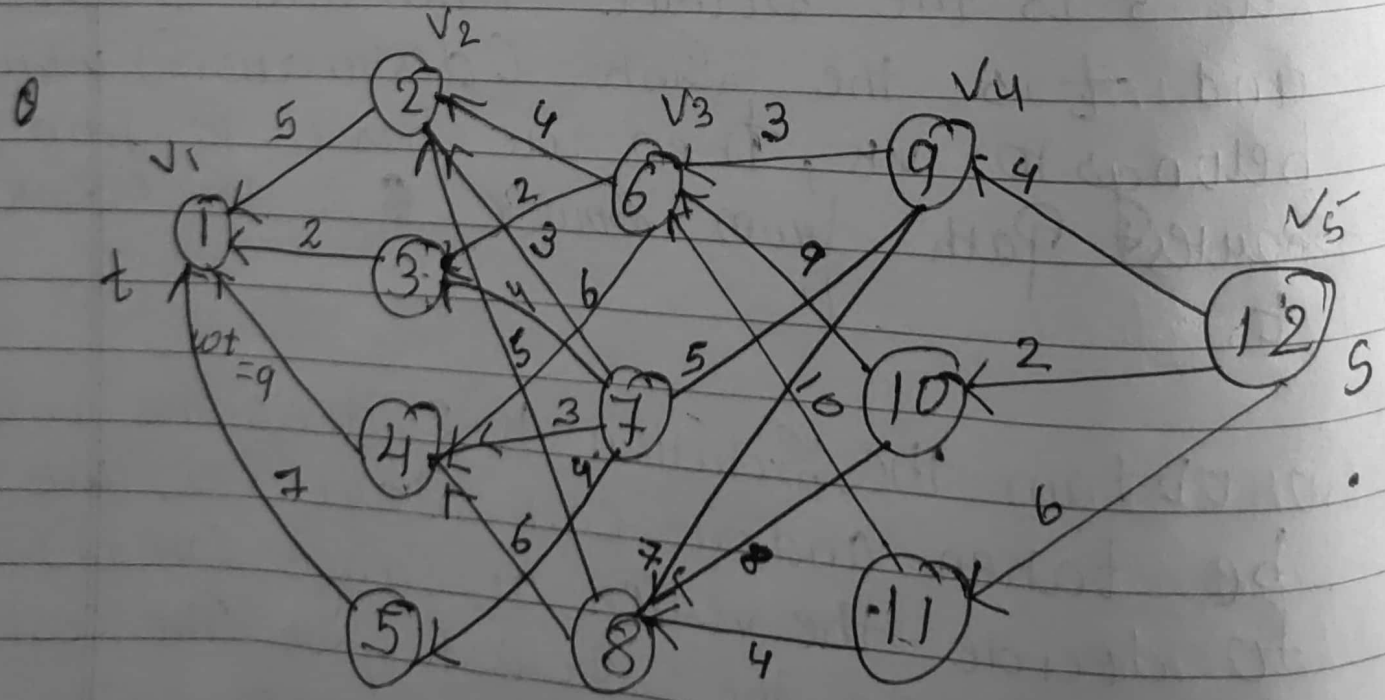
and $\langle j, l \rangle \in E$

There are two approaches to solve the multistage graph

1) Forward approach
2) Backward approach.

Its application are :-

1) Project management
2) Resource allocation & deadlock
3) Evaluation of expenditure for the project



5/02/2020

MSC4

Back to front — forward

Backward — First to last

S - Source

t - Sink / Destination.

Solving with forward Approach

vs

cost (5, 12) = 0

cost (4, 9) = 4

cost (4, 10) = 2

cost (4, 11) = 6

cost (3, 6) = min {4 + cost (3, 6), 4 + cost (3, 7),
4 + cost (3, 8)}

= min {4 + 3, 4 + 5, 4 + 7}

= min {7, 9, 11}

cost (3, 6) = 7

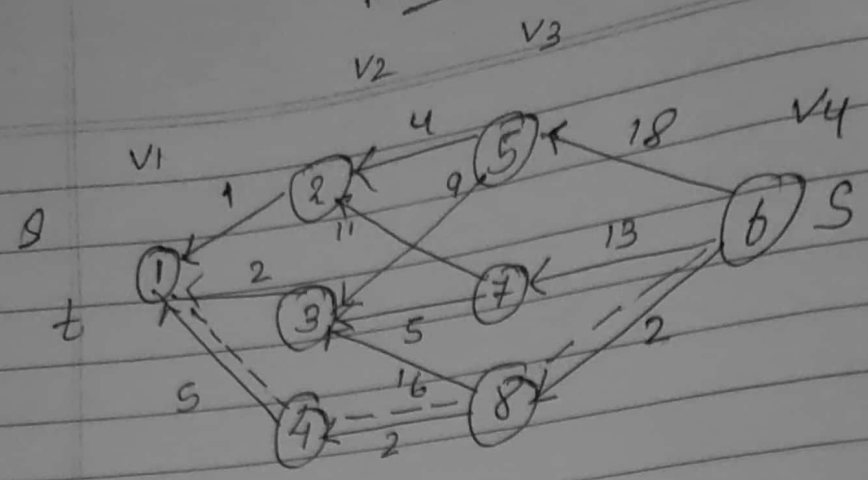
cost (2, 2) = min (4 + cost (2, 2), + 4 +

cost (3, 6) = min {3 + cost (4, 9), 5 + cost (4, 9),
7 + cost (4, 9)}

= min {3 + 4, 5 + 4, 7 + 4}

= min {7, 9, 11}

Forward



v_4
cost (4, 6) = 0
v_3
cost (3, 5) = 18
v_3
cost (3, 7) = 13
v_3
cost (3, 8) = 2

cost(2,2) = min {4 + cost (3,5), 11 + cost (3,7)}
= min {4 + 18, 11 + 13}
= min {22, 24}
cost (2, 2) = 22

cost (2,3) = min {9 + cost (3,5), 5 + cost (3,7),
16 + cost (3, 8)}
= min {9 + 18, 5 + 13, 16 + 2}
= min {27, 18, 18}
cost (2,3) = 18

cost (2,4) = 2 + cost (3,8)
cost (2,4) = 2 + 2 = 4

cost (1,1) = min {1 + cost (2,2), 2 + cost (2,3)
5 + cost (2,4)}
= min {1 + 22, 2 + 18, 5 + 4}
cost (1,1) = 9 min {23, 20, 9

distances

$d(4,6) = 6$

$d(3,5) = 5$

$d(3,7) = 7$

$d(3,8) = 8$

$d(2,2) = 5$

It is not 2 bez in cost of (2,2) is 22 & 22 has arrived from 4 + cost (3,5) there fore 5. & Similarly,

$d(2,3) = 7$

$d(2,3) = 8$

$d(2,4) = 8$

$d(1,1) = 4$

$v_1 = 1$

$v_2 = d(1,1) = 4$ (1st level crossed)

$v_3 = d(2,4) = 8$

$v_4 = 6$

Min distence $= 5 + 2 + 2$

$= 9$

Backward

cost(1,1) = 0 d(1,1) = 1
cost(2,2) = 1 d(2,2) = 2
cost(2,3) = 2 d(2,3) = 3
cost(2,4) = 5 d(2,4) = 5

cost(3,5) = min{4 + cost(2,2), 9 + cost(2,3)}
= min{4+1, 9+2}
= min{5, 11}
cost(3,5) = 5
d(3,5) = 2.

cost(3,7) = min{11 + cost(2,2), 5 + cost(2,3)}
= min{11+1, 5+2}
= min{12, 7}
cost(3,7) = 7.
d(3,7) = 3

cost(3,8) = min{16 + cost(2,3), 9 + cost(2,4)}
= min{16+2, 9+5}
= min{18, 14}
cost(3,8) = 14
d(3,8) = 4.

cost(4,6) = min{18 + cost(3,5), 13 + cost(3,7), 2 + cost(3,8)}
= min{18+5, 13+7, 2+14}
= min{23, 20, 16}
cost(4,6) = 9
d(4,6) =

Note

*

$v_4 = 6$

$v_3 = d(4,6) = 8$

$v_2 = d(3,8) = 4$

$v_1 = 1$

$6 \rightarrow 8 \rightarrow 4 \rightarrow 1$

Minimum distance $= 2 + 2 + 5$

$= 9$

Note In case of forward approach the distances on the nodes are selected from front i.e v_1 to v_n where v_1 is the Stage 1 and v_n is the Stage n while,

In case of backward approach we will find the nodes from v_n to v_1

* Algorithem for we forward graph

Algorithem F Graph (g, k, n, P)

// P is the output Parameter and it is the path from S to t

// g is the K Stage graph having n elements

// n as number from 1 to n

{

6/02/2020

```
cost[0] = 0;
for (j = n-1 to 1 step -1 do)
{
// Let r be a vertex such that
   <r, j> is in edge of G and
    c[j, r] + cost[r] is minimum
cost[j] := c[j, r] + cost[r];
   d[j] := r;
}
```

cost

```
P[1] := 1;
P[k] := n;
// where P[1] and P[k] are stage no.
   and 1 & n are the vertex.
for k := 2 to k-1 step 1 do
{
  P[j] := d[P[j-1]];
}
```

Path

* Algorithem B graph

Algorithem B graph (G, k, n, P)

```
   cost[1] := 0;
for ( j := 2 to n step 1 do)
   {
Let r be a vertex such th
```

cost

cost

Path

cost

$u \langle u, j \rangle$ is an edge of G and
c[u] + cost[u, j] is minimum

cost[j] := c[u] + cost[u, j];
d[j] := u;
// u is the vertex
}

path

P[k] := n;
P[1] := 1;
for (k := k-1 to 2 step -1 do)
{
P[j] := d[P[j+1]];
}

0 m = 6, w => 6 kg profit = 1, 2, 5

profit	weight
1	2
2	3
5	4

$P_i \leq P_{i+1}$
$w_i \leq w_{i+1}$
weight and profit is always in the ascending order
n = 3

P W

$S^0 = \{0,0\}$ $\quad S_1^0 = \{1,2\}$

$S^1 = \{(0,0)\ (1,2)\}$

$S_1^1 = \{(2,3)\ (3,5)\}$

$S^2 = \{(0,0)\ (1,2)\ (2,3)\ (3,5)\}$

$S_1^2 = \{(2,3)(3,5)\}$

$S^3 = \{(0,0)\ (1,2)\ (2,3)\ (5,4)\ (6,6)\ (7,7)\ (8,9)$

$S^3 = \{(0,0)\ (1,2)\ (2,3)$
$(3,5)\ (5,4)\ (6,6)$
$(7,7)\ (0,9)\}$

$m = 6$

$P_i < P_{i+1}$

$w_i < w_{i+1}$

After pruning

$S^3 = \{(0,0)\ (1,2)\ (2,3)\ (5,4)\ \underline{(6,6)}\}$

$(6,6) \in S^3$

$(6,6) \notin S^2 \quad x_3 = 1$

$(1,2) \in S^2$

$(1,2) \in S^1 \quad x_2 = 0$

$(0,0) \notin S^0$

$x_1 = 0$

Ans (0,1)

$P = 1, 2, 5$

$w = 2, 3, 4$

$P = 1+5 = 6$

$w = 2+4 = 6$

Left (as it) Right (add)

Q) P = 1, 3, 6, 7
w = 2, 4, 5, 6
m = 10

Profit	weight
1	2
3	4
6	5
7	6

$P_i < P_{i+1}$

$w_i < w_{i+1}$

weight & profit is always in ascending order

n = 4 (P → w)

$S^0 = \{ 0,0 \}$

$S^1 = \{ (0,0) (1,2) \}$ $S_1^1 = \{ (1,2) (4,$

$S_1^1 = \{ (0,0) (1,2) (4,6) \}$

$S^2 = \{ (0,0) (1,2) (4,6) (10,11) \}$

$S^3 = \{ (0,0) (1,2) (4,6) (10,11$

~~0,0, 1,2 3,4 6,5 4,9 7,6, 8,8, 13,11~~

$S^0 = \{(0,0)\}$

$S^1 = \{(0,0)\ (1,2)\}$

$S^2 = \{(0,0)\ (1,2)\ (3,4)\ (4,6)\}$

$S^3 = \{(0,0)\ (1,2)\ (3,4)\ (4,6)\ (6,5)\ (7,7)\ (9,9)\ (10,11)\}$

$S_1^0 = \{(1,2)\}$

$S_1^1 = \{(3,4)\ (4,6)\}$

$S_2^1 = \{(6,5)\ (7,7)\ (9,9)\ (10,11)\}$

$S_3^1 = \{(7,6)\ (8,8)\ (10,10)\ (11,12)\ (13,11)\ (14,13)\ (16,15)\ (17,17)\}$

$S_4 = \{(0,0)\ (1,2)\ (3,4)\ (4,6)\ (6,5)\ (7,7)\ (9,9)\ (10,11)\ (7,6)\ (8,8)\ (10,10)\ (11,12)\ (13,11)\ (14,13)\ (16,15)\ (17,17)\}$

$S4 = \{(0,0)\ (1,2)\ (3,4)\ (6,5)\ (7,7)\ (7,6)\ (8,8)\ (10,10)\}$

$(10,10) \in S4$

$(10,10) \notin S3$ $\boxed{x4 = 1}$

$(3,4) \in S3$

$(3,4) \in S2$ $\boxed{x3 = 0}$

$(0,0) \in S1$

$(0,0) \in S0$ $\boxed{x2 = 0}$

$\boxed{x1 = 1}$

Ans 1001

$P = 1, 3, 6, 7$

$w = 2, 4, 5, 6$

$P = 8$

$w = 0$

Solve 0, 1 Knapsick prob. where m=10
items = 4, profit = 10, 40, 30, 50
weight = 5, 4, 6, 3, ~~cap~~

profit	weight
10	5
40	4
30	6
50	3

$P_i < P_{i+1}$

$w_i < w_{i+1}$

$S^0 = \{0,0\}$, $S_0' = \{(10,5)\}$

$S^1 = \{(0,0)\ (10,5)\}$, $S_1' = \{(40,4)\ (50,9)\}$

$S^2 = \{(0,0)\ (10,5)\ (40,4)\ (50,9)\}$, $S_2' = \{(30,6), (40,11)\ (70,10)\ (80,15)\}$

$S^3 = \{(0,0)\ (10,5)\ (40,4)\ (50,9)\ (30,6)\ (40,11)\ (70,10)\ (80,15)\}$, $S_3' = \{(50,3)\ (60,8)\ (90,7)\ (100,12)\ (80,9)\ (120,13)\ (130,18)\}$

$S^4 = \{(0,0)\ (10,5)\ (40,4)\ (50,9)\ (30,6)\ (40,11)\ (70,10)\ (80,15)\ (50,3)\ (60,8)\ (90,7)\ (100,12)\ (80,9)\ (120,13)\ (130,18)\}$

$S^4 = \{(0,0)\ (10,5)\ (40,4)\ (50,9)\ (30,6)\ (70,10)\ (50,3)\ (60,8)\ (90,7)\ (90,9)\}$

$S^4 = \{(0,0)\ (40,4)\ (50,6)\ (70,10)\ (50,3)\ (60,8)\ (90,9)\}$

(0,0) (40,4) (30,6) (50,3)

(40,4) $\in S_3$
(40,4) $\in S_2$
(0,0) $\in S_1$
(0,0) $\in S_1$

(90,9) $\in S^4$ | $x_4 = 1$
(90,9) $\notin S^3$

$S_4 = \{(0,0)\ (40,4)\ (30,6)\ (70,10)\ (50,3)$
$(90,7)\ (80,9)\ (120,13)\}$

$S_4 = \{(0,0)\ (40,4)\ (30,6)\ (50,3)\ (80,9)\}$

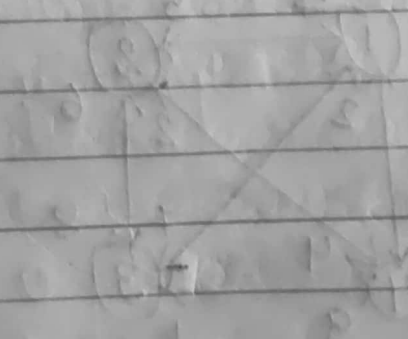
(80,9) $\in S^4$
(80,9) $\notin S^3$ | $x_4 = 1$
(40,4) $\in S^3$
(40,4) $\in S^2$ | $x_3 = 0$
(0,0) $\in S^1$ | $x_2 = 0$
(0,0) $\in S^0$ | $x_1 = 1$

1001

Always apply the pruning at last.

(1001)

Profit = 40, 42, 25, 12
weight = 4, 7, 5, 3
capacity = 10



* Travelling Salesman Problem

Let graph $G = \{V, E\}$ with vertices is given and the cost matrix C for the graph G is defined as (cost) $c(ij)$ of i to j & $c_{ij} = \infty$ if there is no edge b/w i & j.

A tour for the graph G is the simple directed cycle having all the vertices which starts from source vertices & end with the same vertex.

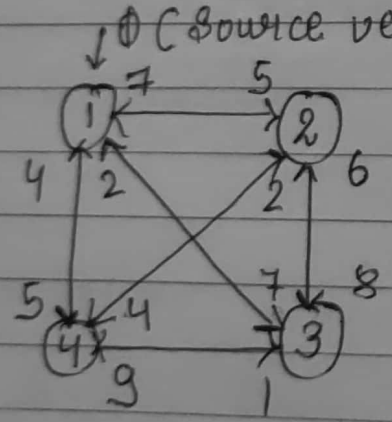
This problem is tour to a minimum cost.

The general formula to calculate tour is -

$$g(i, S) = \min_{j \in S} \{ c_{ij} + g(j, S - \{j\}) \}$$

where $S \in$ Set of vertices

Ex



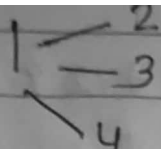
1 is the source vertex

$c(i) = 0$

$c_{ij} = \infty$

Adjacency matrix

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



$g(2,\phi) = c(2,1) = 7$

$g(3,\phi) = c(3,1) = 2$

$g(4,\phi) = c(4,1) = 4$

$g(2,3) = c_{23} + g(3,\phi)$

$g(2,3) = 8 + 2 = 10$

$g(2,4) = c_{24} + g(4,\phi)$

$= 4 + 4 = 8$

$g(3,2) = c_{32} + g(\phi,\phi)$

$= 6 + 7 = 13$

$g(3,4) = c_{34} + g(4,\phi)$

$= 9 + 4 = 13$

$g(4,2) = c_{42} + g(2,\phi)$

$= 2 + 7 = 9$

$g(4,3) = c_{43} + g(3,\phi)$

$= 1 + 2 = 3$

Distributive law

$g\{2(3,4)\} = \min\{c_{23} + g(3,4), c_{24} + g(4,3)\}$

$= \min\{8+13, 4+3\}$

$= \min\{21, 7\}$

$g\{3(2,4)\} = \min\{c_{32} + g(2,4), c_{34} + g(4,2)\}$

$= \min\{6+4, 9+9\}$

$= \min\{10, 18\}$

$g\{4(2,3)\} = \min\{c_{42} + g(2,3), c_{43} + g(3,2)\}$

$= \min\{9+10, 1+13\}$

$= \min\{19, 14\}$

$g\{1(2,3,4)\} = \min\{c_{12} + g\{2\{3,4\}\}; c_{13} + g\{3(2,4)\}, c_{14} + g\{4(2,3)\}$

$= \min\{5+4, 7+14, 5+12\}$

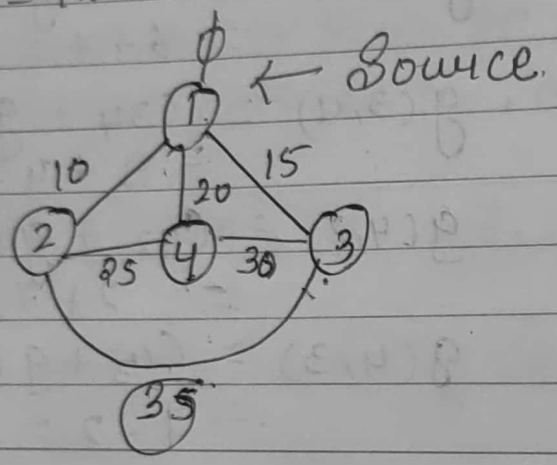
$= \min\{12, 21, 17\}$

$g\{1,(2,3,4)\} = 12$

$1 \to 2 \to 4 \to 3 \to 1$

$5+4+1+2$

$= 12$



Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix} \end{array}$$

$g(2,\phi) = C(2,1) = 10$

$g(3,\phi) = C(3,1) = 15$

$g(4,\phi) = C(4,1) = 20$

$g(2,3) = C_{23} + g(3,\phi)$
$= 35 + 15 = 50$

$g(2,4) = C_{24} + g(4,\phi)$
$= 25 + 20 = 45$

$g(3,2) = C_{32} + g(2,\phi)$
$= 35 + 10 = 45$

$g(3,4) = C_{34} + g(4,\phi)$
$= 30 + 20 = 50$

$g(4,2) = C_{42} + g(2,\phi)$
$= 25 + 10 = 35$

$g(4,3) = C_{43} + g(3,\phi)$
$= 30 + 15 = 45$

Distributive law

$g\{2(3,4)\} = \min\{C_{23} + g(3,4), C_{24} + g(4,3)\}$
$= \min\{35 + 50, 25 + 45\}$
$= \min\{85, 70\}$

$g\{3(2,4)\} = \min\{C_{32} + g(2,4), C_{34} + g(4,2)\}$
$= \min\{35 + 25, 30 + 35\}$
$= \min\{60, 65\}$

$g\{4(2,3)\} = \min\{C_{42} + g(2,3), C_{43} + g(3,2)\}$
$= \min\{25 + 35, 30 + 45\}$
$= \min\{60, 75\}$

15
60
75

$g\{1(2,3,4)\} = \min\{c_{12} + g\{2(3,4)\}, c_{13} + g\{3(2,4)\}, c_{14} + g\{4(2,3)\}\}$

$= \min\{10+70, 15+60, 20+60\}$

$= \min\{80, 75, 80\}$

$g\{1(2,3,4)\} = 80$ FIFO

$1 \to 2 \to 4 \to 3 \to 1$

$10 + 25 + 30 + 15$

$= 80$

10
25
30
15
80

Minimum distance = 80

Q1 Solve the recurrance relationship

$a_n = 2a_{n-1} + 1 \quad n \geq 1$

$a_1 = 1$

Put $n=1$, $n=2$

$\frac{n(n+1)}{2}$

complexity

Q2 Tp. $T(0) = c_1$

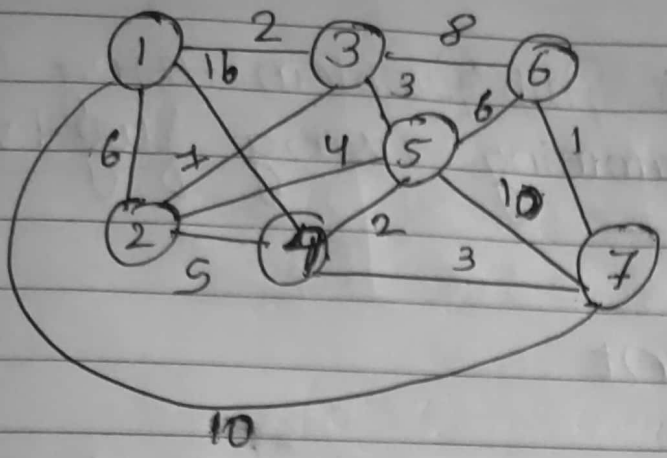
$T(n) = T(n-1) + c_2$

$n = n-1$

$n = n-2$

Q3 $T(n) = 2T\left(\frac{n}{2}\right) + c_3$

Q4)



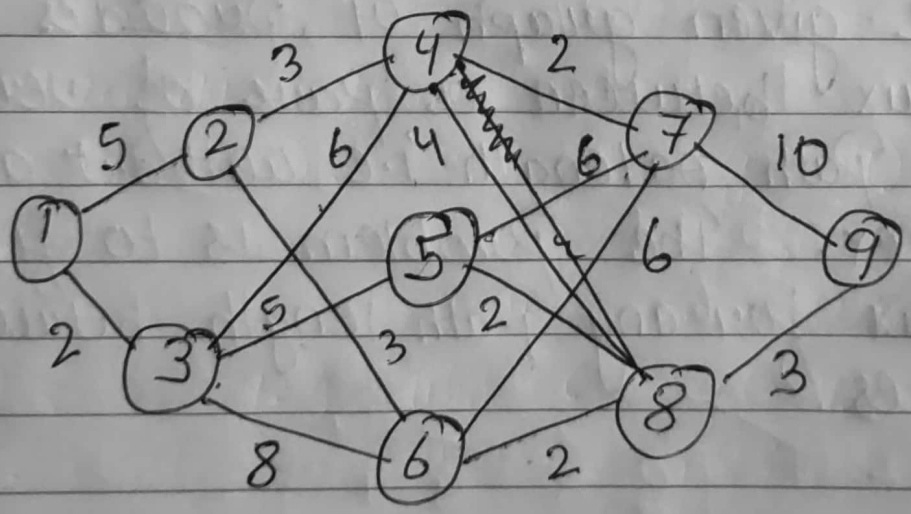
Using Travelling Salesman Problem findout the Shortest path

Q5)

Multi Stage Graph

S → 1

D → 9



* All Pair Shortest Path Problem using in dynamic programming using Floyd Warshall Algorithm

Problem Statement

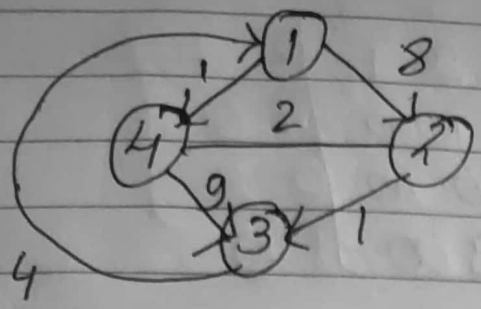
Let the directed graph (G = V, E) is given with 'n' no. of vertices then the adjacency matrix representation will be n×n square matrix whose each element aij is given as

$$a_{ij} \begin{cases} 1 & \text{(if } v_i \text{ \& } v_j \text{ are adjacent)} \\ 0 & \text{(otherwise)} \end{cases}$$

i.e aii = 0. The adjacency matrix for the given graph G, provides a path matrix between a pair of vertices with one path length i.e without any intermediate vertex. The Problem is to find a path matrix for any path length between two vertices.

Note → For calculation purpose, if there is no edge b/w the vertex then, it will be treated as infinity. For the self loop it is considered one or the given weight otherwise zero.

Q1)



Source i
destination j

node i.e near to destination will be intermidiate node

$A[0] =$

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

Case-1

$i=1 \quad j=3 \quad K=2$

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$= d_{12} + d_{23}$

$d_{13}^{2} = 8+1$

$= 9$

Case 2

$i=1, \quad j=3 \quad K=4$

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$= d_{14} + d_{43}$

$d_{13} = 1+9$

$= 10$

case 3

$i=1, \quad j=3 \quad K=2$

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$d_{13}^{2} = d_{12} + d_{23}$

$= 8+1$

$\boxed{d_{13}^{2} = 9.}$

$A[1] =$

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

Case 4

$i = 2, j = 1, k = 3$

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$= d_{23} + d_{31}$ $\Rightarrow d_{21} = d_{23} + d_{31}$

$= 1 + 4$

$d_{21} = 5$

Case 5

$i = 2, j = 4, k = 1$

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$d_{24} = d_{21} + d_{14}$

$= 5 + 1$

$d_{24} = 6$

$A[2] =$

	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	∞	0	∞
4	∞	2	9	0

$$A[3] = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

case 6

- $i = 3, j = 2, k = 1$

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$d_{32} = d_{31} + d_{12}$

$= 4 + 8$

$d_{32} = 12$

$$A[4] = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

case 7

$i = 3, j = 4, k = 1$

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$d_{34} = d_{31} + d_{14}$

$= 4 + 1$

$= 5$

$d_{34} = 5$

$$A[5] = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 14 & 6 \\ 4 & 12 & 0 & 5 \\ \infty & 9 & 9 & 0 \end{bmatrix} \end{array}$$

Case 8

$i = 4, \; j = 1, \; k = 3$

$$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$$

$$d_{41} = d_{43} + d_{31}$$
$$= 9 + 4$$
$$= 13$$

$$A[6] = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 14 & 6 \\ 4 & 12 & 0 & 5 \\ 13 & 9 & 9 & 0 \end{bmatrix} \end{array}$$

UNIT-5

TREE

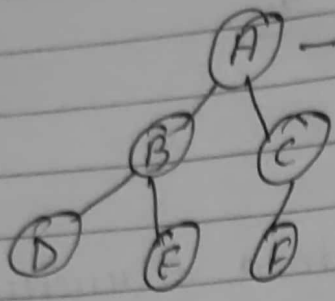
It is a non linear data structure used to represent parent child relationship among its data element In other words it is used to represent the hierarical information. The tree is a collection of finite no. of data elements known as nodes such that there exist a special node known as the root node & all other remaining nodes are divided into disjoint subset, such that, each subset will again be a tree.

*) Binary Tree

This tree is having a special node known as the root node and all other remaining node are divided into atmost two disjoint subset (0, 1, 2), such that each subset will again be a binary tree.

→ Level

1) The root node is always present at level zero.
2) Then each child node will be numbered with the level no. one more than its Parent level number.



→ Height

The height or depth of the tree is one more than its ~~max~~ level number

$$H = l_{max} + 1$$

for above example $= 2 + 1$

$= 3$

2.) Binary Search Tree

A binary tree will be binary search tree if root node contain larger value than that of its left subtree node value and the smaller value than that of its right subtree node value for each node

→ Tree traversal

There are 3 types of traversal :-

1) Inorder (LRoR)
2) Preorder (RoLR)
3) Post order (LRoR)

* Algorithm for Inorder traversal (RRLRoR)

```
Algorithm Inorder (root)
{
  if (root != 0) then
  {
    Inorder (root [Lchild]);
    Display (root [info]);
    Inorder (root [Rchild]);
  }
}
```

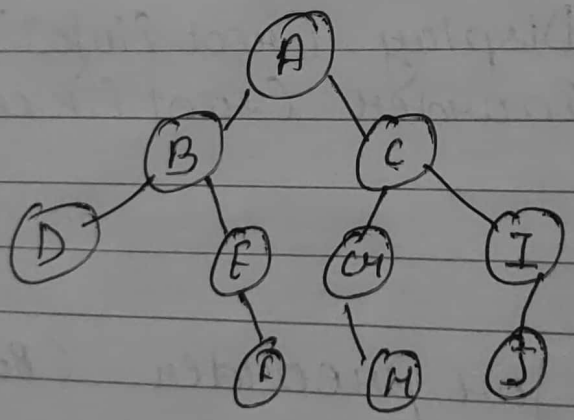
* Algorithm for Preorder (RoLR)

```
Algorithm Preorder (root)
{
  if (root != 0) then
  {
    Display (root [info]);
    Preorder (root [Lchild]);
    Preorder (root [Rchild]);
  }
}
```

Post Algorithm for Post order (L R Ro)

```
Algorithm post order (L R Ro)
{ if (root != 0) then
  {
    postorder (root [L child]);
    Post order (root [R child]);
     Display (root [info]);
  }
}
```

Q) Inorder, Preorder, postorder



Inorder (L Ro R)

~~I, G, I, G, H, B, E~~ F, D, B, E, A
D B E F A G H C J I

Preorder (Ro L R)

A, B, D, E, F, G, H, I, J

Postorder (L R Ro)

D F E B H G J I C A

Q) To make tree

Inorder D B E F A G H C J I

Preorder A B D E F C G H I J

postorder D F E B H G J I C A

* AVL Tree / Height balance tree

Significance of Height in Binary Search tree.

It is found that if the height of the tree is larger than it will take more time to search a given element. On the other hand if the height of the tree is minimum the search will take a minimum time. Hence, to make the search more faster the tree must be balance w.r.t its height. A term balance factor is used for balancing the binary search tree. The balance factor can be define as, the difference of the no. of child nodes in the longest left subtree path & the longest right subtree path. If the balance factor for each node comes out to be zero then the tree will be perfectly balance binary search tree but such trees are rare. Practically in the height balance tree the balance factor lies in the range of -1, 0, +1 i.e ± 1. If the insertion of new node result unbanced binary tree then the following rotations can used to balanced it.

1) Lea
2) Ri
3) Le
4) Ri

01) Co

02) S

1) Left rotation
2) Right rotation
3) Left Right rotation
4) Right Left rotation

Q1) Construct an AVL tree for the following elemet
63, 9, 19, 27, 18, 108, 99, 81

Q2) 50, 20, 60, 10, 8, 15, 32, 46, 11, 48

02/2020

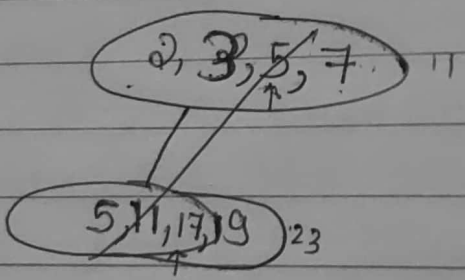
① Construction B-Tree order 5
2, 3 5 7 11 17 19 23 29 31

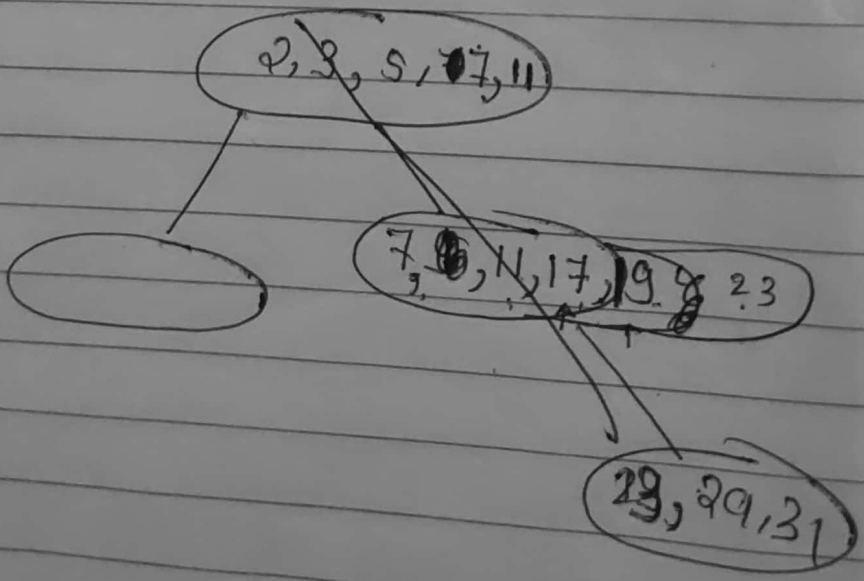
order = 4
key = 3 } mid = key/2

② Order 4
R, 4, F, X, A, M, C, D, E, T, H, [illegible], L, [illegible], U

Soln

Order = 5
Key = order − 1
= 5 − 1
= 4



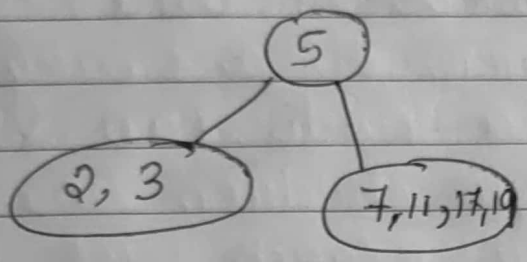


Soln

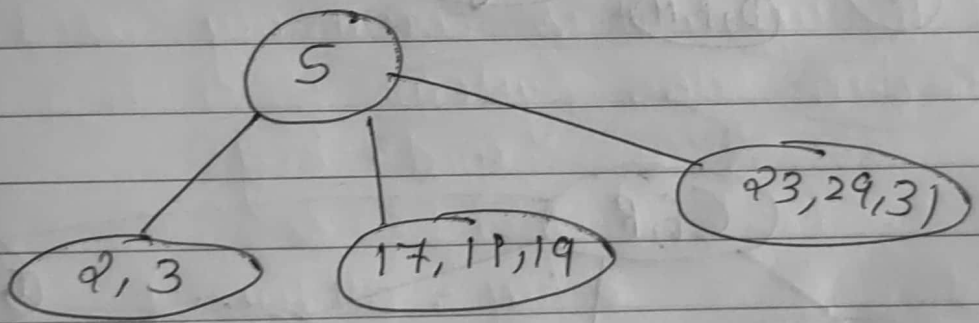
I

I) 2, 3, 5, 7

II)



III)



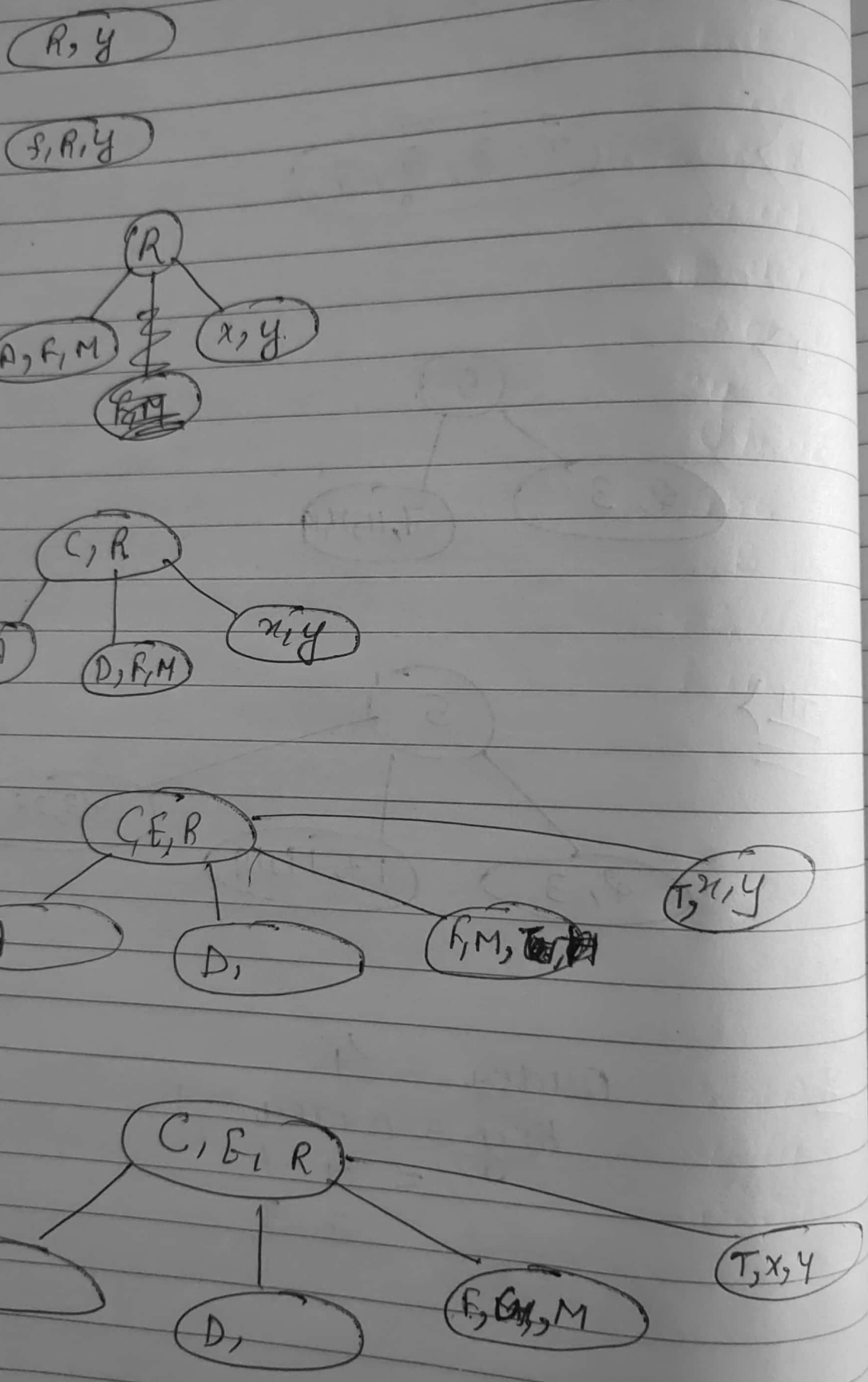
Solⁿ 2

order = 4

key = order − 1

= 4 − 1

= 3



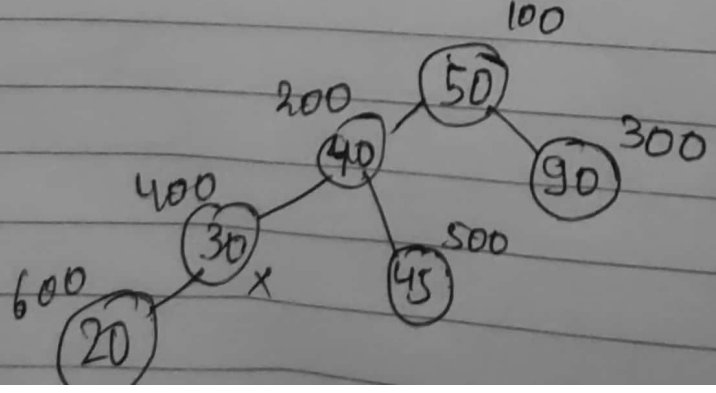
B-Tree

A B-tree is a data structure that keeps data sorted & allows searches insertion and deletion in logarithmic time unlike self balancing tree. Binary search is optimize for system that read & write large block of data. The B-tree is a generalization of a binary search tree, in that a node can have more than two children unlike self balancing binary search tree, B-tree is a good example for the external memory. It is commonly used in database and in file system.

$$mid = \frac{key}{2}$$

- key = order − 1

* write an algorithm to delete for an element from the tree and firstly you have to search the element.



Algorithm Search for delete (root, Left, Right)

while (root

while C

Algorithm Search for delete (root, num, pnode)

```
{ found := false;
  dnode := root;
  pnode := 0;
while (dnode != 0 & not found ) do
  {
   if (num = dnode [info]) then
    {
      found := true;
    }
  else if ( num < dnode [info] ) then
  {
     dnode := dnode [Lchild];
  }
else
  {
    dnode := dnode [Rchild];
  }
return (false);
}
```

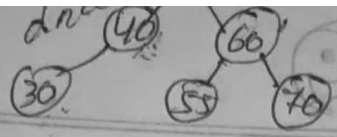
Case I Algorithm to Delete an element from the tree having no child

```
Algorithm Delete if no child (root, num)
{
  if (! search for Delete (root, num, Pnode)
  { 
  Cwrite "No to be deleted not found");
}
else
{ // No. to be deleted has no child
  if (dnode [Lchild] = 0 && dnode [Rchild] = 0) the
  { 
    if (Pnode = dnode [Lchild])
  }
    Pnode [Lchild] := 0;
}
else
{
   Pnode [R child] := 0;
}
   delete (dnode);
}
};
```

Case II Alg

```
Else
{
  if
{ 
}
else
{
```

Case III

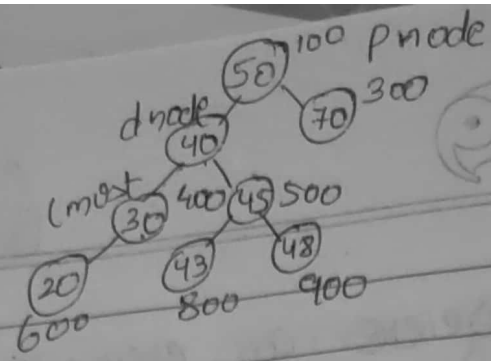


Case II ~~Alg~~ Delete an element having only one child

```
Else if (dnode [Lchild] ≠0 and dnode [rchild] =0)
{
  if (Pnode = dnode [Lchild])
{
    Pnode [Lchild] := dnode [Lchild];
    Pnode [Lchild] := 0;
}
else
{
  Pnode [Lchild] := dnode [rchild];
    P [Lchild] := 0;
}
```

Case III Delete an element having only one child & that is right child.

```
Else if (dnode [Lchild] =0 and dnode [rchild] ≠0)
{  if (Pnode = dnode [rchild])
{
    Pnode [Lchild] := dnode [rchild];
}
else
  {  pnode [rchild] := dnode [rchild];
}
  delete (dnode);
}
```

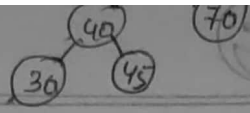


1st rmost
2nd lmost

Case IV

Case IV Delete the node having both child i.e Left child & Right child

```
else if (dnode [Lchild] ≠ 0 & dnode [Rchild] ≠ 0)
{ if (dnode = Pnode [Lchild]) then
  {
    Pnode [Lchild] := dnode [Lchild];
    rmost := dnode;
  while (rmost [Rchild] ≠ 0) do
    {
      Rmost := rmost [Rchild];
      break;
    }
  Lmost := dnode [Lchild];
  while [Lmost [Rchild] ≠ 0]
  {
    Lmost := Lmost [Rchild];
  }
  Lmost [Rchild] := rmost;
  }
}
  delete (dnode);
}
```



Case V Delete the node having both child i.e right child

```
{Else (dnode = Pnode [R child]) then
  {
   Pnode [Rchild] := dnode [Rchild];
```