11/2/16

MODULE - I

Algorithm :- It is an unambigious, sequence of steps to solve the problem,

ie it generates the required o/p by accepting legitimate i/p in finite amount of time.

Characteristics of Algorithm :-

1. Non-ambigious steps
2. Range of inputs [It should accept]
3. Legitimate [valid] input.
4. Finiteness.
5. Definiteness [correctness, it definitely gives the O/p]
6. Effectiveness

Same Algorithm can be represented in different ways

i) Natural language rep^{-n} [English]

ii) pseudo code rep^{-n}

iii) Flow chart method

7. Same problem can be solved using many Idea ie Design techniques.

GCD :-

1) Euclids method
2) Consecutive integer check method.
3) middle school method.

Effectiveness :- Every step should be effective [Important] to generate the required o/p

1. Euclids - GCD

Algorithm :- Euclids - GCD (m, n)

// Input : non-negative integer m, n and both should not be zero

// output : GCD of m, n.

while (n ≠ 0)

r ← m % n

m ← n

n ← r.

endwhile

return (m)

2. Consecutive integer check ~~language~~ method

Algorithm :- consecutive integer check ~~language~~ method - GCD

// Input :- ~~non-negative~~ positive (non zero) integers m, n and both should ~~be~~ not be zero

// ouput :- ~~output~~ GCD of m, n.

m = 30, 4

t	m%t	n%t
4	2	-
3	0	1
$\boxed{2}$	0	0
GCD		

step 1 : t ← min(m, n)

step 2 : Divide m by t. if remainder is non-zero, then go to step 4

step 3 : Divide n by t. if remainde is zero, return t as GCD

step 4 : Decrease t by 1 ; Goto step 2.

Pseudo-Code

1. select minimum value of two inputs assign it to t.

2. divide m and n by t, if both remainder zero, then t is GCD.

3. else decrement t by 1 and repeat step 2 until both remainder are zero.

3. Middle school method

Algorithm : middle school- GCD(m,n)

// Input : non-zero positive integer

// output : GCD of m,n.

step1 : find the factors of m

step2 : Find the factors of n

step3 : Find the common factors from step1 and step2

step4 : return largest common factor from step3 as GCD

#include <iostream.h>

Sieve of Eratosthemes

used to generate prime numbers for given range.

n=2

Initial -

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

Sieve (2) 2, 3, 5, 7, 9, 11, 13, 15, 17, 19

Sieve (3) 2, 3, 5, 7, 11, 13, 17, 19.

Algorithm sieve (n)

// Input : An integer $n \geq 2$.

// output : list A contain prime number upto n.

```
for p ← 2 to n do
    L[p] ← p
end for
for p ← 2 to ⌊sqrt(n)⌋ do
    j ← p*p ;
```

```
if(L[j] ≠ 0)
    while(j ≤ n) do
        L[j] ← 0
        j ← j + p
    end while
end if
end for

j ← 1
for p ← 2 to n do
    if(L[p] ≠ 0)
        A[j] ← L[p]
        j ← j+1
    end if
end for
return (A)
```

Algorithm specification :-

1) comment is specified through //
2) Algorithm header is specified as

   Algorithm Name (parameter list)

3) ~~Alg~~ specify algorithms legitimate inputs and the required O/P
4) compound Statements are specified with { } or begin ..... end
5) Record data type are specified through { } {like struct}

   Ex :- record student
   {
       name datatype
       - - - - - -
   }

   (compound data types) each of them are accessed their Intances.

6) if (condition) then
    statement 1
    Statement 2
    ⋮
else
    statement
    ⋮
endif.

7. loop statement

① while (condition) do
    ⋮
  endwhile.

② repeat
    ⋮   } Do while
  until (condition)

③ for variable1 ← value1 to value2 do
    ⋮
  end for

④ for variable ← value1 downto value2 do

⑤ for variable ← value1 to value increase by value do.

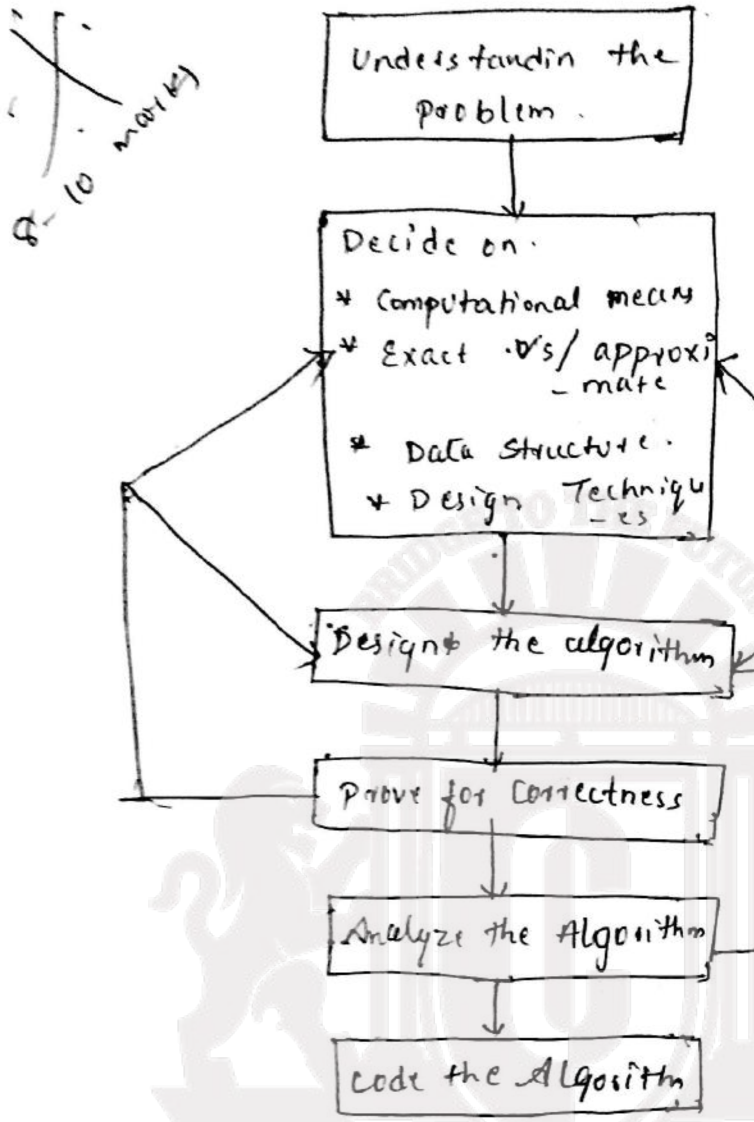
6) Assignment operator (←)
   eg:- A ← B.

⑦ Relational operator (<, ≤, >, ≥, =, ≠).

⑧ Multidimentional array index are specified ~~in [and]~~. in [and]

   Ex:- A[i, j]

Algorithm design and analysis process/Frame work



1. The first step involves in the throughly understanding the problem statement and trying to solve the problem manually for the different instances (copies) of the problem.

2. Decide on

→ Computational means

The computational capability judges the computing power of the device on which the algorithm is made to run, common factors considered are speed and memory. The type of the algorithm can be considered as sequential algorithm and parallel algorithm

→ Exact v/s Approximate solving: It decides on whether the problem should generate exact sol^n or approximate

sol^{-n}. generating a exact sol^{-n} may be difficult for the following Instances

a) generating o/p for non-linear eq^{-n}'s, solving integrals, calculating square roots.

b) Due to complexity of solving problem exactly which requires high computation capability outcomes are compromised with approx values.

→ Data Structure

The choice of data structure affects the performance of the algorithm. choose the data structure that are ~~comprised with approx values~~ approximate for current problem.

ex:- choosing array instead of linked list.

→ Algorithm Design techniques

Designing an Algorithm is defined as the method or the strategy to solve the given problem and get the desired o/p.. A few design techniques are listed below.

i) Brute force - linear search, bubble sort, ...

ii) Divide & conquer

iii) Decrease and conquer.

iv) Greedy technique.

v) Dynamic programming.

vi) Back tracking, Branch and Bound.

3) Design of Algorithm

Designing algorithm involves representing each and every step with non-ambiguity features & simple and basis. Algorithm can be represented using natural language, flowchart or pseudocode.

4) Prove the correctness / Algorithm valuation

It is the process of checking a correctness of the algorithm for the range of legitimate input and their respective output. Approximate sol^{-n} may be verified by checking their error way, which should not exceed to a certain limit.

Analysis Framework :-

Based on the resource of computing machines.

1) CPU time. → Time complexity Analysis.
2) RAM space. → .space .complexity Analysis.

---

1) measuring input size

Identify potential input for the algorithm which effects the time efficiency.

ex:- linear search (A [1 ....n], key)

In this size n is a prominant i/p parameter which effects the efficieny of the algorithm.

2) Unit of measuring time
   i) clock rate - device time [we dont conside] ✗
   ii) primitive operation - counting [— " —] ✗
   iii) Basic operation ✓

5) Analyses of the algorithm :-

The algorithm is analysed based on time and space constrained and an algorithm which suits the best for the application choosen.

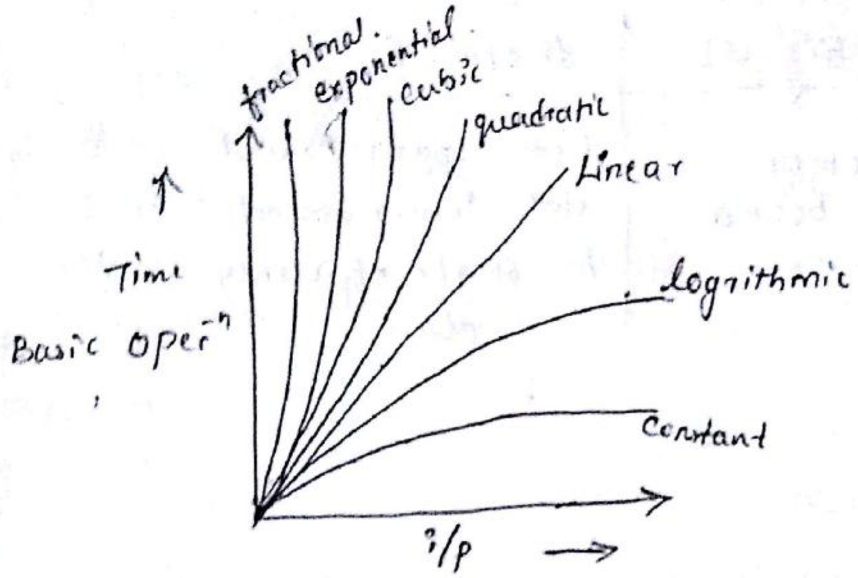
6) Code the Algorithm :-

Algorithm is implemented on machine using one of the programming languages.

3. Order of growth

It is the relationship b/w i/p size and the time consumed for running an algorithm as the input size increases, the time consumed propomately increases in a particular order.

The order in which time increases w.r.t input is called as order of the growth.



4. efficiency classes [standard orders of growth]

Name	Representation	Input n=10
constant	c	c
logrithmic (binary search)	$\log_A$	$\log_2 10 = 3.32$
Linear (addition)	n	n = 10
n-log-n	nlogn	Time complexity $10\log_2 10 = 33.2$
Quadratic [Bubble sort]	n^2	100
cubic [matrix mul]	n^3	1000
exponential	2^n	1024
Factorial	n!	3628800

5. Asymptotic notations

It is a symbolic representation to categorise the algorithmic effeciency at different instances of input of the same size, comparing in the standard Efficiency classes

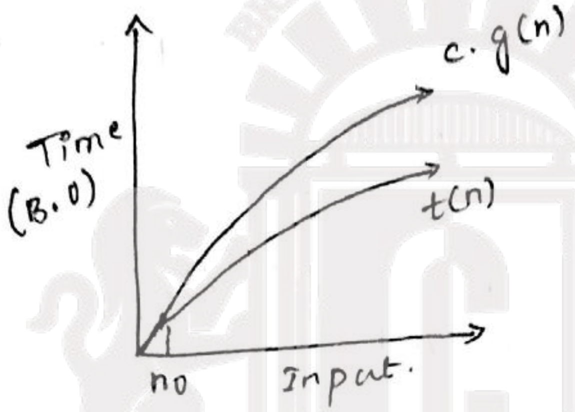
Big-O

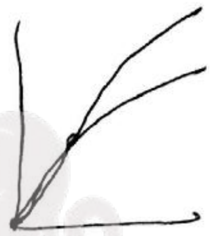
A fn $t(n)$ is said to be in $O(g(n))$, denoted by $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ $\forall$ large 'n', ie if there exist some positive constant 'c' and some non negative integer 'n_0' such that $t(n) \le c\, g(n)$ $\forall n \geq n_0$

$$\boxed{t(n) \le c \cdot g(n) \quad \forall n \geq n_0}$$

$t(n) \rightarrow$ Order of growth of algorithm.

$g(n) \rightarrow$ Refrence order of growth.





Ex:-

P.T $100n+5 \in O(n)$

$t(n) = 100n+5$

$g(n) = n$

Proof:-

If $100n+5 \in O(n)$, then

$100n+5 \le c \cdot n \quad \forall n \geq n_0$.

$100n+5 \le 100n+n$

$100n+5 \le 101n$.

$\boxed{c = 101}$

$100n+5 \le 101n$.

$n=1 \quad 100+5 \le 101 \quad \times$

$n=2 \quad 100\times2+5 \le 101\times2 \quad \times$

$n=5 \quad 100\times5+5 \le 101\times5$

$n=6 \quad 100 \times 6+5 \leq 101 \cdot 6$ ✓

$\boxed{n_0=5}$

$\therefore \; 100n+5 \leq 101n \quad \forall n \geq 5.$

$\therefore \; \boxed{100n+5 \in O(n)}$

<u>Ex 2</u> $\cdot \; 3n^2+n+2 \in O(n^2)$

$t(n) = 3n^2+n+2$.

$g(n) = n^2$.

<u>Proof</u> $\cdot$ If $3n^2+n+2 \in O(n^2)$, then

$3n^2+n+2 \in c \cdot n^2 \quad \forall n \geq n_0$

$3n^2+n+2 \in 3n^2 \cdots$

$3n^2+2n \in (3n^2+n+n^2)$

$3n^2+2n \in 4n^2+n$

$\boxed{c=4}$

$n=1 \quad 3n^2+2n \in 4n^2+n$.

$6 \qquad 5$

$n=2 \quad 16 \qquad 18$

$\boxed{n_0=2}$

$3n^2+n+2 \leq 4n^2+n \quad \forall n \geq 2$.

$\boxed{3n^2+n+2 \in O(n^2)}$

<u>Ex 3</u> :- $4n^3+3 \in O(n^3)$

$t(n) = 4n^3+3$

$g(n) = n^3$

Proof, if $4n^3+3 \in O(n^3)$ then.

$4n^3+\cancel{3n^3} \in \cancel{4}n^3+n^3$
$5n^3$

$\boxed{c = 5}$

$\cancel{5n^3}$ $4n^3+3 \in \cancel{4n}^3+n^3$

$4n^3+3 \in 5n^3$

$\boxed{c = 5}$

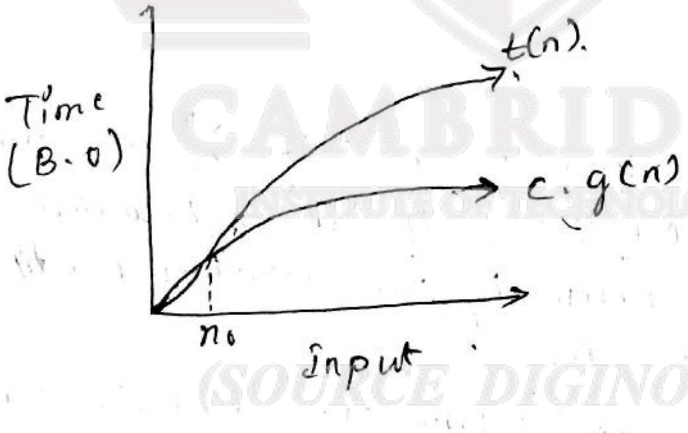
$\boxed{n_0 = 2}$

Big - Ω [omega]

a function $t(n)$ is said to be in $\Omega(g(n))$ denoted by $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some constant multiple of $g(n)$ $\forall$ large n.

ie. if there exists some +ve' constant 'c' and some non-negative integer n_0 such that

$$\boxed{t(n) \geq c \cdot g(n) \quad \forall\, n \geq n_0}$$



Ex:-

P.T $3n^2+n \in \Omega(n)$

$t(n) = 3n^2+n$.

$g(n) = n$.

$3n^2+n \geq c.n. \quad \forall n \geq n_0$

$3n^2+n \geq 3n+n$.

$3n^2+n \geq 4n$.

$\boxed{c=4}$

if $n_0 = 1$. $3n^2+n \geq 4n \quad \forall n \geq 1$

hence $\boxed{3n^2+n \in \Omega(n)}$.

<u>Ex:</u> . P.T $4n^3+2n^2+n+5 \in \Omega(n^3)$

$t(n) = 4n^3+2n^2+n+5$

$g(n) = n^3$

$4n^3+2n^2+n+5 \geq c.n^3$

$4n^3+2n^2+n+5 \geq 4n^3+2n^2+n^3$

$4n^3+2n^2+n+5 \geq 7n^3$

$c = 7$.

$n=1$, $4+2+1+5 \geq 7$

$12 \geq 7$

$n_0 = 1$

$4n^3+2n^2+n+5 \geq 7 \quad \forall n \geq 1$

$4n^3+2n^2+n+5 \geq \Omega(n^3)$.

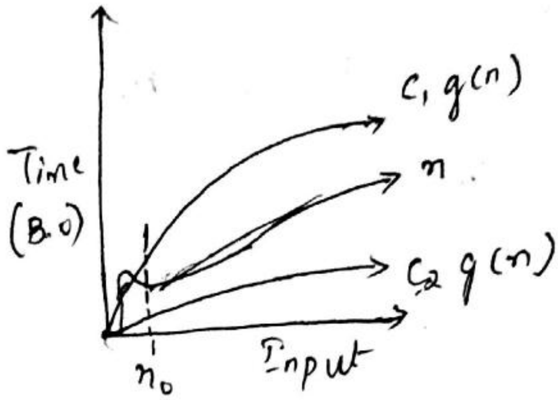
21/2/17

<u>Theta - Θ</u>

A fn $t(n)$ is said to be $\Theta(g(n))$ denoted by $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some constant multiples of $g(n)$ $\forall$ large n.

~~$\forall$ do~~ <u>ie</u> if there exist some +ve constants c_1 and c_2 and some non negative integer n_0 such that

$$c_2\, g(n) \leq t(n) \leq c_1\, g(n) \quad \forall n \geq n_0$$



P.T. $\frac{1}{2} n(n-1) \in \Theta(n^2)$

$t(n) = \frac{1}{2} \cdot n(n-1)$.

$g(n) = n^2$.

$$\boxed{c_2 \cdot n^2 \leq \underbrace{\boxed{\tfrac{1}{2} n(n-1)}}_{1} \underbrace{\leq c_1 n^2}_{2}} \quad \forall n \geq n_0$$

eq^n 1.

$c_2 n^2 \leq \frac{1}{2} n(n-1)$

$\frac{1}{2} n(n-1) \geq c_2 \cdot n^2$

$\frac{1}{2} n^2 - \frac{1}{2} n \geq c_2 n^2$

$\frac{1}{2} n^2 - \frac{1}{2} n \geq \frac{1}{2} n^2 - \frac{1}{2} \cdot \frac{1}{2} n^2$.

$\geq \frac{1}{2} n^2 - \frac{1}{4} n^2$

$\frac{1}{2} n(n-1) \geq \frac{1}{4} n^2$

$c_2 = \frac{1}{4}$

$n = 1$ $\frac{1}{2} 1(1-1) \geq \frac{1}{4} \cdot 1$ ✗

$n = 2$ $\frac{1}{2} 2(2-1) \geq \frac{1}{4} 2^2$ ✓

$n = 3$ $\frac{1}{2} 3(3-1) \geq \frac{1}{4} 3^2$ ✓

$\boxed{n_{01} = 2}$

consider eq^n (2)

$\frac{1}{2} n(n-1) \leq c_1 n^2 \quad \forall \; n \geq n_{02}$

$\frac{1}{2}n^2 - \frac{1}{2}n \leq c_1 \cdot n^2$

$\frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2$

$c_1 = \frac{1}{2}$

$n = 1, \; \frac{1}{2} - \frac{1}{2} \leq \frac{1}{2}$

$n = 2 \cdot \; 2 - 1 \leq 2$

$\boxed{n_{0_2} = 1}$

$n_0 = \max\{n_0, n_{0_2}\}$

$\max\{2, 1\}$

$= 2.$

$\therefore \frac{1}{4}n^2 \leq \frac{1}{2} n(n-1) \leq \frac{1}{2}n^2 \quad \forall \; n \geq 2.$

$\therefore \boxed{\frac{1}{2} n(n-1) \in \Theta(n^2)}$

2. $100n + 5 \in \Theta(n).$

~~$c_2 \cdot n^2 \leq \frac{1}{2} n(n$~~

$\boxed{c_2 n \leq 100n + 5} \leq c_1 n$

$c_2 n \leq 100n + 5$

$100n + 5 \geq c_2 n.$

$100n + 5 \geq 100n$

$100n + 5 \geq 100n.$

$c_2 = 100$

$n = 1$

$n_{01} =$

notes4free.in

$\therefore 100n + 5 \leq c_1 n$.

$c_1 n \geq 100n + 5$

~~100~~ $100n + 5 \leq c_1 n$.

$100n + 5 \leq 100n + n$

$\boxed{c_1 = 101.}$

$n = 5,$ ✓

$\boxed{n_{02} = 1.}$

$n_0 = \max\{n_{01}, n_{02}\}$

$\max\{5, 1\}$

$\{5\}$

$\therefore 100n \leq 100n + 5 \leq 101n$.

$\boxed{100n + 5 \in \Theta n}$

Properties of Assymptotic notations

1. If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$
then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Proof :-

If $t_1(n) \in O(g_1(n))$ then $t_1(n) \leq c_1 g_1(n)$
$\forall\ n \geq n_{01}$

lly

$= t_2(n) \in O(g_2(n))$ then, $t_2(n) \leq c_2 . g_2(n)\ \forall\ n \geq n_{02}$.

$= t_1(n) + t_2(n) \leq c_1 . g(n) + c_2\, g_2(n)$

To maximize RHS

$c_3 \ \max\{c_1\ c_2\}$

Replace c_1 and c_2 by c_3

$= t_1(n) + t_2(n) \leq c_3 \cdot g_1(n) + c_3 \cdot g_2(n)$

Choose $\max\{g_1(n), g_2(n)\}$ to replace $g_1(n)$ and $g_2(n)$.

$= t_1(n) + t_2(n) \leq c_3 \cdot \max\{g_1(n), g_2(n)\} +$

$c_3 \cdot \max\{g_1(n), g_2(n)\}$.

$\leq 2 c_3 \max\{g_1(n), g_2(n)\}$

$c = 2c_3$

$= t_1(n) + t_2(n) \leq c \cdot \max\{g_1(n), g_2(n)\}$.

$n_0 = \max\{n_{01}, n_{02}\}$.

$= \therefore t_1(n) + t_2(n) \leq \max\{g_1(n), g_2(n)\} \; \forall n \geq n_0$

hence.

$$\boxed{t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})}$$

2. comparing 2 different order of growth $(t(n), g(n))$ using limits

ie $\lim_{n \to \infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & t(n) \text{ is smaller order of groth than } g(n) \\ c, \text{ where } c > 0 & t(n) \text{ is same order of grolwth as } g(n) \\ \infty & t(n) \text{ has higher order growth than } g(n) \end{cases}$

17/2/17

Ex:- compare $n!$ with 2^n using limits

$$\lim_{n \to \infty} \frac{n!}{2^n}$$

using sterling formula for $n!$

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\lim_{n \to \infty} \frac{n!}{2^n} = \lim_{n \to \infty} \frac{\sqrt{2\pi n}\,(n/e)^n}{2^n}$$

$$= \sqrt{2\pi} \lim_{n \to \infty} \sqrt{n}\left(\frac{n}{2e}\right)^n$$

$$= \infty$$

$\therefore$ $n!$ has higher order of growth compared to 2^n (exponent)

Mathematical Analysis on non-Recursive Algorithm

1) Decide on input parameter
2) Identify the basic operation
3) check whether the algorithm depends only on input or if there are any variation, if so estimate Best case, worst case and average case time efficiency separately.
4) Build summation equation for no of Basic operation executed.
5) solve the equation & ascertain it to one of the standard efficiency class

Ex:-

Algorithm liner-search (A[1....n], key).
//Input : An array of Integers A[] with n elements and key
//output : returns true if found else false.

```
for i ← 1 to n do.
        if (A[i] == key) then
                return True
        end if
end for
return false.
```

Analysis

1. Input parameter - 'n' size of array A.
2. Basic operation - search /comparision - A[i] = key
3. Apart from input size 'n' the algorithm produces varying order of growth, therefore estimate time analysis for best case, worst case and average cas seperately

4) Best case : if the key is found at first position. comparision is one

$C_{Best}(n) = 1$

5 i] $\boxed{C_{Best}(n) \in \Omega(1)}$ ⇒ constant order of growth

Worst case : if the key is found in last position.

$C_{worst}(n) = \sum_{i=1}^{n} 1$

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

$= n$ //

$\left\{ \sum_{l}^{u} \text{constant} = \text{constant} * (u-l+1) \right.$

5ii] $\boxed{C_{worst}(n) \in O(n)}$ ⇒ linear order of growth.

Average case :

$C_{Avg}(n) = \frac{(1+2+3+4+\ldots+n)}{n} * p + (1-p) * n.$

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

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

If key is found, p = 1.

$C_{Avg\text{-}found}(n) = \left(\frac{n+1}{2}\right) * 1 + (1-1) * n.$

$= \frac{n+1}{2} \approx n/2 + 1/2.$

For last value of n.

$C_{Avg\text{-}found}(n) \approx 1/2\, n.$

$\boxed{C_{Avg\text{-}found}(n) \in \Theta(n)}$

if key is not found , $P = 0$.

$C_{avg-false}(n) = \left(\frac{n+1}{2}\right) * 0 + (1-0) * n$

$= n$.

$\boxed{C_{avg-false}(n) \in \Theta(n)}$ => Constant. Order of growth.

Ex 2 Finding the largest timing in given list.

Algorithm max-element (A[1...n])
//Input: An array A[ ] of n Integers (positive)
//output : return MAX.

```
MAX ← 0.
  for i ← 0 to n do
    if [A[i] > MAX]
        MAX ← A[i]
    end if
  end for
return MAX
```

Analysis

1) Input parameter → 'n' size of array 'A'
2) Basic operation → search / comparision - A[i] > MAX
3) the algorithm completely depends on 'n'
   Hence no variations.
4) $c(n) = \sum_{1}^{n} 1$

$= n$ linear order of growth.

$C(n) \in \Theta(n)$.

4. $$c(n) = \sum_{i=1}^{n} 1$$

$$= n - 1 + 1$$

$$= n \qquad \boxed{c(n) \in \Theta(n)}$$

Exp:- To find the uniqueness of a given list

Algorithm Uniqueness (A [1...n]).

// Input : An Array A [ ] of 'n' integer

// output : unique list / duplicate list .

for $i \leftarrow 1$ to $n-1$ do .

for $j \leftarrow i+1$ to n do .

if (A[i] = A [j]) then

return (duplicate).

end if

end for

return (unique).

Analysis :- (check uniqueness) .

(1) Iput parmeter :- 'n' Size of array

(2) Basic operation:- comparission - $A[j] = A[j]$

(3) for Uniqueness check algorithm depends only on 'n' :- no variation .

(4) $$c(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1$$

(5) $$c(n) = \sum_{i=1}^{n-1} (n-(i+1)+1) \Rightarrow \sum_{i=1}^{n-1} (n - i - \cancel{1} + \cancel{1})$$

$$c(n) = \sum_{i=1}^{n-1} (n-i) \Rightarrow \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$$

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

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

$$c(n) = n(n-1) - \frac{n(n-1)}{2}.$$

$$c(n) = \frac{n(n-1)}{2}.$$

$$c(n) = \frac{n^2}{2} - \frac{n}{2}.$$

for large value of n.

$$\boxed{c(n) \in \Theta(n^2)}$$

eg Matrix multiplication

$$\underset{A}{\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}} \quad \underset{B}{\begin{bmatrix} 3 & 2 \\ 1 & 7 \end{bmatrix}}$$

Algorithm :-

```
matrix-multiplication (A[n,n], B[n,n]).
//Input :- matrix A[n,n], B[n,n] of integers.
//output :- Resultant matrix C[n,n].
// Assumed that matrix C is initialized to zero
  for i<-1 to n do
   for j<-1 to n do.
    for k<-1 to n do
     C[i,j] <- C[i,j] + A[i,k] * B[k,j].
    end for
   end for
  end for
  return C.
```

Analysis :-

1) Input Parameter - 'n' nxn order of matrix (Both A or B).

2) Basic operation :- multiplication - $A[i,k] * B[k,j]$.

3) Depends only on order of matrix - 'n' no variations.

4) $M(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} 1$.

$m\cancel{(n)=}($

$\boxed{m(n) \in (n^3)}$

cubic order of growth.

Ex: counting no of Bits required to represent Decimal no

Algorithm :- Bits(n)

Input : Decimal no 'n' → non-negative integer

Output : count ↔ no of bits to represent 'n'

```
count ← 1
while (n>1) do
  n ← n/2
  count ← count++
end while
return (count)
```

Analysis :-

1) Input parameter :- 'n'

2) Basic operation :- Addition

3) Depends only on 'n' - no variations

4) $A(n) = \sum_{i=1}^{\lfloor \log_2 n \rfloor} 1$

$\lfloor \log_2 n \rfloor$ = upper lm − lower lm + 1

$= \lfloor \log_2 n \rfloor - 1 + 1$ $\lfloor \log_2 n \rfloor$

$\lfloor \log_2 n \rfloor$ $\boxed{A(n) \in \Theta(\log n)}$

logrithmic order of growth.

Mathematical analysis for recursive algorithm :-

1) Decide on input parameter
2) Identify Basic operation
3) Check if the order of growth depends only on 'n' or if there are any variation, if so, then estimate Best case, worst case and Average case seperately
4) Build Recurrance relation based on the Basic operation.
5) Solve the recurrence relation and ascertain it to one of the standard efficiency class

Ex :-

Algorithm Factorial (n)
// Input : non negative integer n
// output : factorial of n

```
if(n≤1)
    return (1)
else
    return (n * factorial (n-1))
```

Analysis :-

1) Input parameter is 'n'
2) B.O → multiplication − n * factorial(n−1)

3. Depends only on 'n' - no variation

4.

$$\text{factorial}(n) = \begin{cases} 1 & n \le 1 \\ n * \text{factorial}(n-1) & n > 1 \end{cases}$$

~~Base~~ ~~one~~ basic operation.

$$M(n) = \begin{cases} 0 & n \le 1 \\ 1 + M(n-1) & n > 1 \end{cases}$$

$M(n) = 1 + M(n-1)$ until $M(1) = 0$.

5) Solve using Backward substitution method

{ Backward substitution method }

$$\begin{aligned} m(n) &= 1 + m(n-1). \\ &= 1 + 1 + m(n-2). \\ &= 2 + m(n-2) \\ &= 2 + 1 + m(n-3). \\ &= 3 + m(n-3) \\ &\vdots \\ & n-1 + m(\cancel{n-(n-1)})^{0}. \quad \{ m(1) = 0 \\ &= n-1 \end{aligned}$$

For value of n, $m(n) \approx n$.

or

$$\boxed{m(n) \in \Theta(n)}$$

linear order of growth.

Finding no of bits to represent Decimal numbers

Algorithm :- Bit-count (n).
//Input :- non-negative integer 'n'
//output :- count of no of bits required.

```
if (n ≤ 1) then.
    return (1)
else
    return(1 + Bit-count(⌊n/2⌋)).
end if
```

$(\lfloor n/2 \rfloor))$

Analysis :-

1) Input parameter - 'n'
2) Basic operation - addition " 1+ Bit-count(n/2)
3) Depends only on 'n' - no variations
4) Algorithms recurrence relation.

$$\text{Bit-count}(n) = \begin{cases} 1, & \text{for } n \le 1 \\ 1 + \text{Bit-count}(n/2) & n > 1 \end{cases}$$

B.O recurrence relation.

$$A(n) = \begin{cases} 0 & n \le 1 \\ 1 + A(n/2) & n > 1 \end{cases}$$

5) $A(n) = 1 + A(n/2)$ until $A(1) = 0$.

$1 + [1 + A(n/4)]$

$1 + [1 + [1 + A(n/8)]]$

⋮

complicated.

consider $n = 2^k \longrightarrow (1)$

$$\therefore A(n) = 1 + A\left(2^k/2\right)$$
$$= 1 + A\left(2^{k-1}\right)$$
$$= 1 + \left[1 + A\left(2^{k-2}\right)\right]$$
$$= 2 + A\left(2^{k-2}\right)$$
$$3 + A\left(2^{k-3}\right)$$
$$\vdots$$
$$k + A\left(\cancel{2^{k-k}}\right)$$

$$A(n) = k.$$

Apply $\log_2$ on eq^n (1)

$$\log_2 n = k \cancel{\log_2 2}$$
$$\log_2 n = k.$$
$$\log_2 n = A(n)$$

$$\boxed{A(n) \in \Theta \log n}$$

logrithmic order of growth.

~~Tim~~

Tower of Hanoi

Algorithm :- Tower (n, S, T, D)

// Input : no of discs - n' ~~inpu~~ 3 poles - S, T, D

// output : n - discs on destination pole

if $(n = 1)$ then.

move disc 1 from S to D

else

Tower $(n-1, S, D, T)$

move nth disk from S to D

Tower (n-1, T, S, D)

end if.

Analysis

1) Input parmeter - 'n' no of disks.

2) B.O -> moving discs.

3) Depends only on 'n' - no variation.

4) Algorithm's .. recurrence relation.

$$\text{Tower}(n, S, T, D) = \begin{cases} \text{move disc 1 from S to D} & n = 1 \\ \text{Tower}(n-1, S, D, T), & \\ \text{move } n^{th} \text{ disc from S to D} & n > 1 \\ \text{Tower}(n-1, T, S, D) & \end{cases}$$

B.O. recurrence relation

$$M(n) = \begin{cases} 1 & n = 1 \\ M(n-1) + 1 + M(n-1) & n > 1 \end{cases}$$

5) $m(n) = 1 + 2M(n-1)$ until $M(1) = 1$

$$= 1 + 2[1 + 2M(n-2)]$$

$$= 1 + 2 + 2^2 \cdot M(n-2)$$

$$= 1 + 2 + 2^2 [1 + 2M(n-3)]$$

$$= 2^0 + 2^1 + 2^3 M(n-3)$$

$$= 2^0 + 2^1 + 2^3 + 2^4 M(n-4)$$

$$= 2^0 + 2^1 + 2^3 + \cancel{2^4} + \cdots\, 2^{n-1} M(n - \cancel{(n-1)})$$

$$= 2^0 + 2^1 + 2^3 + 2^4 + \cdots 2^{n-1}$$

= GP

= sum of Geometric progession.

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

$a = 1$, $r = 2$.

$$\therefore M(n) = \frac{1(2^n - 1)}{2 - 1} = 2^n - 1$$

For large value of n.

$M(n) \approx 2^n$.

$\therefore \boxed{M(n) \in \Theta(2^n)}$

4/3/17 Exponential order of growth.

<u>Fibonacci series</u>

Seed elements - 0, 1

Algorithm Fib-iterative (n)

//input : an integer $n \geq 2$.

//output : n^{th} term of Fib series

// Auxillary array F[0...n]

F[0] ← 0

F[1] ← 1

for i ← 2 to n do

F[i] ← F[i-1] + F[i-2]

end for.

return;

<u>Analysis</u>

1) Input parameter - n.

2) B.O ⇒ Addition - F[i-1] + F[i-2].

3) Depends only on 'n' no variations

4) $A(n) = \sum_{i=2}^{n} 1$

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

$= n-1$

For large value of n.

$A(n) \approx n$.

$\boxed{A(n) \in \Theta(n)}$

linear order of growth

Recursive method

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

$f(n) - f(n-1) - f(n-2) = 0 \longrightarrow (1)$

similar to $a\,x(n) + b\,x(n-1) + c\,x(n-2) = 0$.

homogenous linear second order equation.

Its characteristic eq^{n}

$$ar^2 + br + c = 0.$$

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Apply to eq^{n} (1).

$a = 1$, $b = -1$, $c = -1$

characteristic eq^{n} of $f(n)$

$$= r^2 - r - 1 = 0.$$

$$= \frac{-(-1) \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2(1)}.$$

$$= \frac{+1 \pm \sqrt{1+4}}{2}$$

$$r = \frac{+1 \pm \sqrt{5}}{2}.$$

$r_1 = \frac{1+\sqrt{5}}{2}$ $\quad r_2 = \frac{1-\sqrt{5}}{2}.$

$x(n) = \alpha r_1^n + \beta r_2^n.$

$f(n) = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^n + \beta\left[\frac{1-\beta\sqrt{5}}{2}\right] \longrightarrow (2)$

We know that

$f(0) = 0$, $f(1) = 1$

$f(0) = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^0 + \beta\left[\frac{1-\sqrt{5}}{2}\right]^0 = 0.$

$\alpha + \beta = 0$ $\quad \boxed{\beta = -\alpha}$

consider $f(1) = 1$ in eq^n ②.

$$f(1) = \alpha\left[\frac{1+\sqrt{5}}{2}\right]^1 + \beta\left[\frac{1-\sqrt{5}}{2}\right]^1 = 1.$$

Replace $\beta = -\alpha$.

$$f^1 = \alpha\left[\frac{1+\sqrt{5}}{2}\right]^1 - \alpha\left[\frac{1-\sqrt{5}}{2}\right] = 1$$

$$\alpha\left[\frac{\cancel{1}+\sqrt{5}-\cancel{1}+\sqrt{5}}{\cancel{2}}\right] = 1$$

$$\alpha\left[\frac{\cancel{2}\sqrt{5}}{\cancel{2}}\right] = 1$$

$$= \alpha\left[\sqrt{5}\right] = 1$$

$$\boxed{\alpha = \frac{1}{\sqrt{5}}}$$

$$\boxed{\beta = -\frac{1}{\sqrt{5}}}$$

$$f(n) = \frac{1}{\sqrt{5}}\left[\frac{1+\sqrt{5}}{2}\right]^n - 1/\sqrt{5}\left[\frac{1-\sqrt{5}}{2}\right]^n.$$

$$= 1/\sqrt{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

Golden Ratio $\phi = \frac{1+\sqrt{5}}{2}$.

$$\hat{\phi} = \frac{1-\sqrt{5}}{2}.$$

$$f(n) = 1/\sqrt{5}\left[\phi^n - \hat{\phi}^n\right]$$

~~Analysis~~

Algorithm :- Fib-rec (n)

// Input : An integer $n \geq 2$.

if $(n \leq 1)$

return(n)

else
return (fib_rec(n-1) + fib_rec(n-2)).
endif

Analysis

1) Input parameter - 'n'
2) B.O. Addition fib_rec(n-1) + Fib_rec(n-2).
3) Depending only on 'n' - no variations.
4) Algorithm recurrence relation.

$$Fib\_rec(n) = \begin{cases} 0, 1 & n \le 1 \\ Fib\_rec(n-1) + Fib\_rec(n-2) & n > 1 \end{cases}$$

Based on B.O.

$$A(n) = \begin{cases} 0 & n \le 1 \\ A(n-1) + A(n-2) + 1 & n > 1 \end{cases}$$

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

$$A(n) - A(n-1) - A(n-2) - 1 = 0.$$

$$(A(n)+1) - (A(n-1)+1) - (A(n-2)+1) = 0.$$

$B(n) = A(n) + 1$, $B(n-1) = A(n-1) + 1$

$B(n-2) = A(n-2+1)$.

$\therefore B(n) - B(n-1) - B(n-2) = 0.$

Same as $f(n) - f(n-1) - f(n-2) = 0.$

sol^n for $f(n) = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$

$\therefore$ Applying the same solution

$$B(n) = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$$

$B(n) = A(n) + 1$

$A(n) = B(n) - 1$

$$\therefore A(n) = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right) - 1$$

For large values of n, subtracting 1 becomes negligible and $\hat{\phi}^n$ is inverse of ϕ^n, so it can be negligible.

$$A(n) \approx \frac{1}{\sqrt{5}}\,\phi^n$$

$$\boxed{A(n) \in \Theta(\phi^n)}$$

Exponential order of growth.

7/3/17

Problem types (Refer text book)

1) Sorting

Ordering of elements based required manner.

Ex:- Bubble sort, merge sort, selection sort, Radix sort, Insertion sort.

2→ properties of sorting algorithm.

1. Stable property. [maintains first come first ser
2. In-place property.

1. stable property :-

A sorting Algorithm is said to be stable if it maintains the relative order of the duplicate elements even after sorting.

example

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

1	2	2	3	5	5	7

2. In-place property :-

An sorting Algorith is said to be in-place if the algorithm does not consume extra memory

Ex:- Bubble sort, selection sort, Insertion sort

→ merge sort is sorting by distributed counting

. Searching problem

Find an element in the given list is known as Searching problem.

Numeric Searching
 ↳ simple/set of number (pattern)

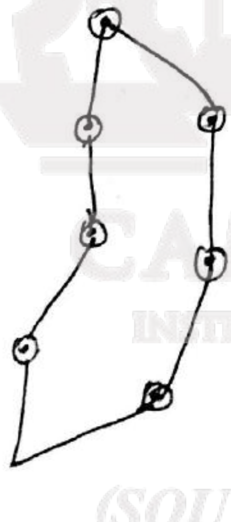
Non-Numeric
 ↳ characters / sub-string.

Ex:- linear, binary, interpolation search, Hashing
Horspool, Boyer - more.

3. String Processing

ex :- Adding 2 string
Finding length of str.
Copying 2 - strings.
Searching sub-strings.

4. Graph Problems



Shortest path.
Hamiltonian circuit.
Traversals Techniques.
Travelling sales person Problems (TSP)
Distance algorithm.
Spanning tree graph.

5.) Geometric Problems
problems regarding plotting all geometric shapes

6) Numerical problems :

→ equations which are continous in nature.
→ Definite integral.

→ sine series,
→ Runge kutta method
→ Simpson's method

7) Combinatorial problem :-

permutation, combination, sub set which grows exponentially

Data structure

location
{ Linear :- array, stack, queue, structure, list
{ Non-linear :- Tree, graph.

Based on access { sequential - Linked list, stack, queue, tree
{ Dynamic - array, indexed-Linked list.

Array :- homogeneous collection of objects.

Structure :- homogeneous or heterogeneous collection of object.

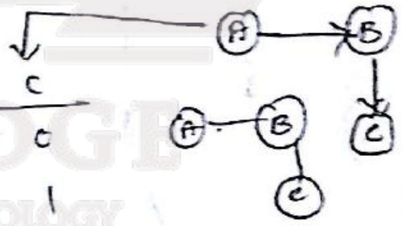
stack :- FILO, LIFO

Queue :- FIFO, LIFO

graph :- $G\{V, E\}$
↳ Directed
↳ Undirected.

1. Adjacency matrix → $n \times n$ Symmetrix matrix.

2) Adjacency List



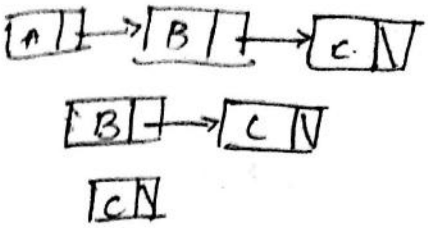
Assymmetric

	A	B	C
A	0	1	0
B	0	0	1
C	0	0	0

symmetric

	A	B	C
A	0	1	0
B	1	0	1
C	0	1	0.

Adjacency list:



sparse Graph

-> no of edges are less than no of nodes

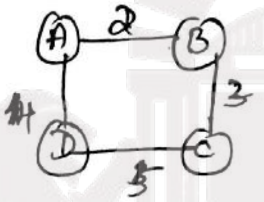
-> graph containing few edges. [Adjacency list]

Dense Graph

-> opposite of sparse graph [Adjacency matrix]

1. weigted graph -> edge with value.

ex:- Distance b/w two cities.



weight matrix or cost matrix

	A	B	C	D
A	∞	2	∞	4
B	2	∞	3	∞
C	∞	3	∞	5
D	4	∞	5	∞

Adju

connected graph.

comple graph

cylic graph

Acylic graph

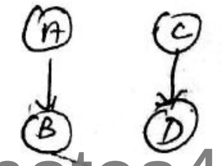
(DAG).

Tree (CDAG)

1) connected graph
2) Acyclic graph
3) Directed graph

Forest

(Not connected DAG)



indegree

outdegree

vertex - indegree-0-root.
- outdegree - 0 - leaf.

Vertex - non-zero indegree and non-zero outdegree. } Branch.

→ Heap tree → ascending order heap tree
descending ——"——

BST.

Set :- collection of non-duplicate elements

multiset → collection of elements with duplicate value.

Dictionary. pair of elements
<name , value>

General method
Binary method
Merge sort
Quick sort
Min-max algorithm
strassen's matrix multiplication.

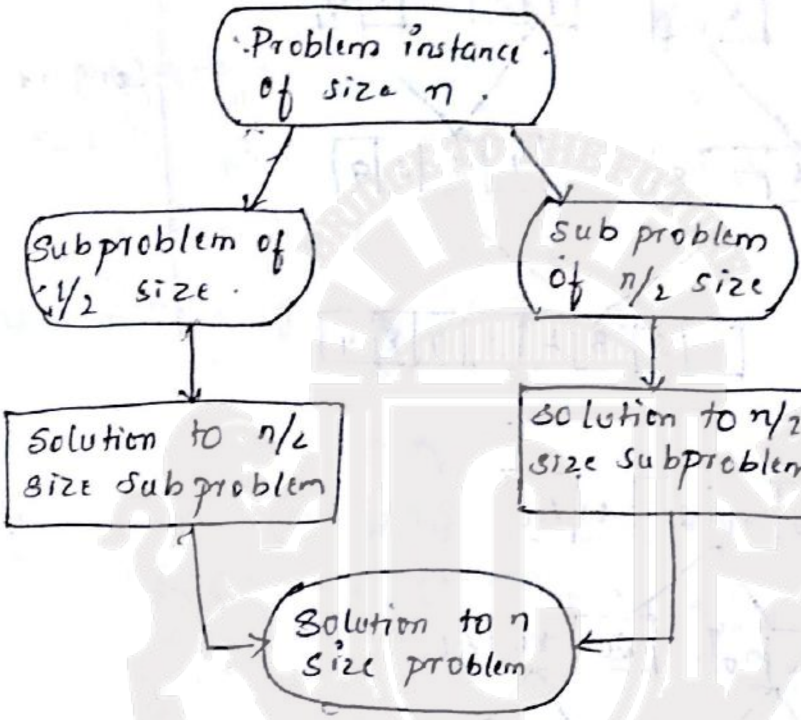
7/10/17

MODULE - 2

DIVIDE AND CONQUER

1. Problem is divided into sub-problems [until no further division is required]. → Divide
2. solve the sub-problem with known method [simple]
3. If required, combine the solution of the sub-problem instances to find the solution of overall problem

conquer.

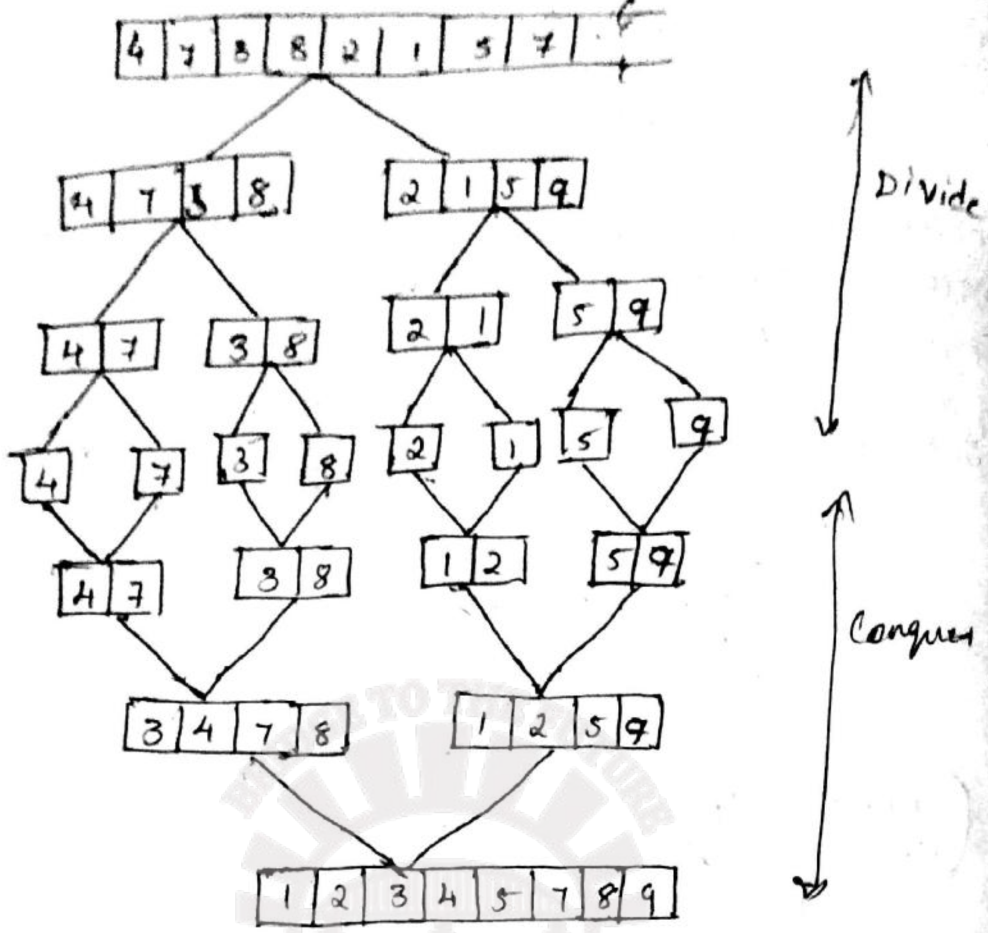


Merge sort

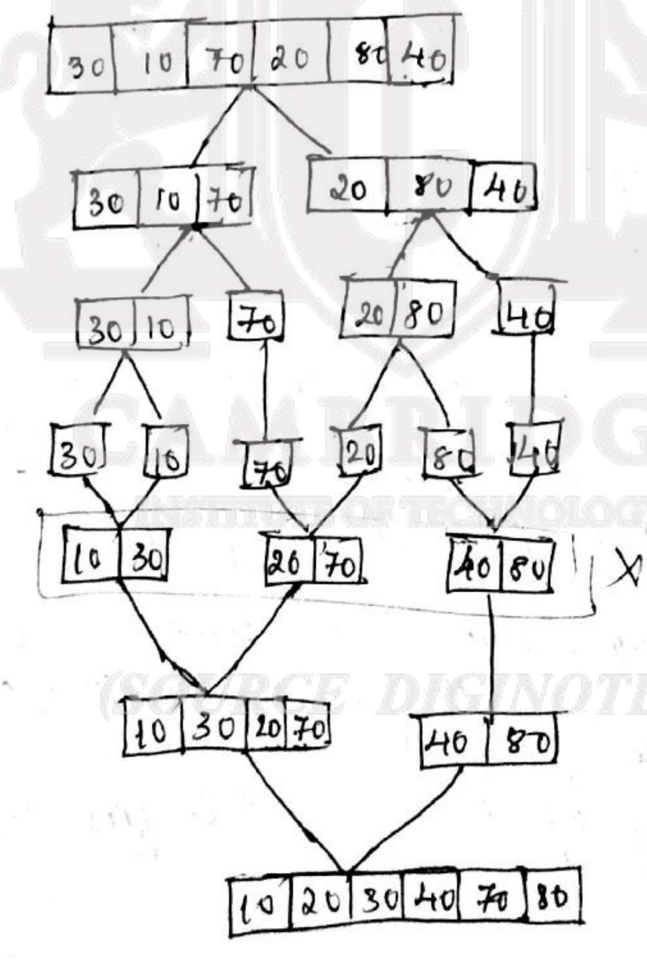
→ The given list is considered to be as unsorted file.
→ Divide the file into two part [approximately equal]
→ continue dividing until file reaches to a non divisible state. [one element file]

→ One element file by default is considered as sorted file
→ Combine the sorted files through simple comparision b/w the elements of files.
→ The combining of two file satisfies conquer part.

ex 1



ex 2



Algorithm merge_Sort (A[0, ... n-1])

//Input : An array A[0,.... n-1] of n oderable elements

//output : Array A[0, ... n-1] of ordered elements

if (n > 1) then

copy A[0, ... ⌊n/2⌋] to B[0, ... ⌊n/2⌋]

copy A[⌊n/2⌋+1 ..... n-1] to C[0.... ⌈n/2⌉]

mergesort (B[0.... ⌊n/2⌋])

mergesort (C[0.... ⌈n/2⌉])

merge (B, C, A)

end if

Algorithm merge (B[0 ....p-1], C[0 ....., q-1], A[0 ..... p+q-1])

//Input : sorted array B[ ] and C[ ]

//output : sorted array A[ ].

i ← 0, j ← 0, k ← 0

while (i < p and j < q)

if (B[i] < C[j]) then

A[k] ← B[i]

i ← i+1

else

A[k] ← C[j]

j ← j+1

end if

k ← k+1

end while

if (i < p)

copy B[i..... p-1] to A[k ..... p+q-1]

else

copy C[j .... q-1] to A[k ..... p+q-1]

end if

16/3/17

Analysis

1. Input parameter → 'n' size of input array.
2. Basic operation → comparison $B[i] < C[j]$
3. Minimal variation (within same order of growth)
4. Recurren relation Based on B.O.

$$C(n) = \begin{cases} 0 & n=1 \\ C(n/2) + C(n/2) + n-1 & n>1 \end{cases}$$

5. $C(n) = 2C(n/2) + \underbrace{n-1}_{\text{upper bound}}$ until $C(1) = 0$

[Backward substitution moving from n to 1]

consider $n = 2^k$

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

$= 2 \cdot C(2^{k-1}) + 2^k - 1$

$= 2 \cdot [2 \cdot C(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1$

$= 2[2 \cdot [2C(2^{k-3}) + 2^{k-2} - 1] + 2^{k-1} - 1] + 2^k - 1$

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

$= 2^2 C(2^{k-2}) + 2 \cdot 2^k - 2^1 - 2^0$

$= 2^3 C(2^{k-3}) + 3 \cdot 2^k - 2^2 - 2^1 - 2^0$.

⋮

$= 2^k \cdot \cancel{C(2^{k-k})}^{0} + k \cdot 2^k - 2^{k-1} - 2^{k-2} \ldots \ldots - 2^1 - 2^0$.

$= k \cdot 2^k - [2^{k-1} - 2^{k-2} \ldots - 2^1 - 2^0]$

Sum of n^{th} term of Geometric series

$$S_n = \frac{ar^n - 1}{r - 1}$$

$=$

$= k \cdot 2^k - \left[\frac{1 \cdot 2^k - 1}{2 - 1}\right]$

$= k \cdot 2^k - [1 \cdot 2^k - 1]$

$= k \cdot 2^k - [2^k - 1] = k \cdot 2^k - 2^k + 1$

$= \boxed{2^k [k-1] + 1}$ // $\approx 2^k \cdot k$.

upper bound

Apply $\log_2$ on both sides $n = 2^k$.

$\log_2 n = \log_2 (2^k) = k \log_2 2$

$= k$.

$c(n) \approx n \cdot \log_2 n$

$\boxed{c(n) \in \Theta(n \log_2 n)}$

Theoretical upper bound

$c(n) = n \cdot [\log_2 n - 1] + 1$

$n = 10$

$10[\log_2 10 - 1] + 1$

$= 24.21 \approx 25$//

To get the lower bound.

$C(n) = 2C(n/2) + n/2$ until $c(1) = 0$

$n = 2^k$

$= 2\, c(2^k/2) + 2^k/2$

$= \cancel{2\left[2c\left(\frac{2^{k-2}}{2}\right) + 2^{k-1}/2\right] + 2^k/2}$

$= \cancel{2^2 c\left[2^{k-2}/2\right] + 2^{k-1} + 2^k/2}$

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

$= 2^2 \cdot c[2^{k-2}] + 2^{k-1} + 2^{k-1}$

$= 2^2 c[2^{k-2}] + 2[2^{k-1}]$

$= 2^3 \cdot c[2^{k-3}] + 3[2^{k-1}]$

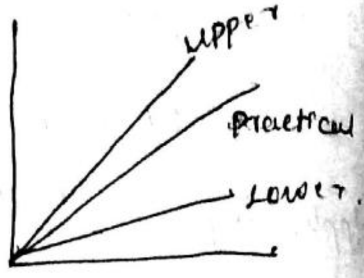
$\vdots$

$= 2^k c(\cancel{2^{k-2}}) + k(2^{k-1})$

$= k(2^{k-1})$

$$= K \frac{2^k}{2}.$$

$$\boxed{C(n) = \frac{n \log_2 n}{2}.}$$ lower bound.



$n = 10$

$C(n) = 16.6$

$\approx 17.$

Master's Theorem

If there is Recurrence relation

$$T(n) = a\,T(n/b) + f(n)$$

Where a -- no of subproblems to be solved

$b \rightarrow$ no. of fractions of input size 'n'

$f(n)$ = time consumed for Divide / conquer

d = degree of n in $f(n)$.

then,

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_b n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Ex1 for Merge sort $C(n) = 2 \cdot C(n/2) + n - 1$

According to master's theorem.

$T(n) = a \quad T(n/b) + f(n).$

$a = 2, \ b = 2, \ f(n) = n - 1, \ d = 1$

Relationship b/w $\quad a \quad b^d$

$2 = 2^1$

$$\therefore \boxed{C(n) \in \Theta(n^1 \log_2 n).}$$

It is applicable for divide and conquer algorithm.

Ex2 $C(n) = 2 \cdot C(n/2) + n/2$

According to master's theorem.

$T(n) = a, \ T(n/b) + f(n)$

$a = 2, \ b = 2, \ f(n) = n/2, \ d = 1$

relationship b/w a b^d

$2 = 2^1$

$$\boxed{C(n) \in \Theta(n^1 \log_2 n)}$$

example 3. $T(n) = 3 \cdot T(n/4) + n^2$

$a = 3,\ b = 4,\ f(n) = n^2,\ d = 2$

relationship b/w a b^d

$3 < 4^2$

$$C(n) \in \Theta(n^2).$$

Example 4: $T(n) = T(n/2) + 1$

$a = 1,\ b = 2\ \ f(n) = 1\ \ d = 0$

relationship b/w a b^d

1 $2^0 = 1$

$$C(n) \in \Theta(n^0 \log_2 n)$$

Ex 5: $T(n) = 4T(n/2) + n$

$a = 4,\ b = 2,\ f(n) = n,\ d = 1$

a b^d

$4 > 2$

$$\boxed{T(n) \in \Theta(n^{\log_2 4})}$$

$T(n) \in \Theta(2n)$

Ex 6 $T(n) = 5T(n/2) + n.$

$a = 5,\ b = 2,\ f(n) = n,\ d = 1$

a b^d

$5 > 4$

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

$T(n) \in \Theta(n^{\log_2 5}).$

$T(n) = \Theta(n^{2.3}).$

$C(n) \in \Theta(n^{2.3}).$

Quick sort

→ One of the divide and conquer algorithm
→ partition results in un-even sub groups.
→ partition occurs based on pivot / Anchor element.
→ left group < pivot element &
pivot < Right group

Left group < pivot < Right group.

p.
ex   8 $\boxed{5}$ 2 4 7 6

2 4 |5| 7 6.

17/3/17

Ex:

0 1 2 3 4 5 6 7
Pivot i i i j j
$\boxed{5}$ 3 1 9 8 2 4 7

swap

$\boxed{5}$ 3 1 4 8 2 9 7

swap

$\boxed{5}$ 3 1 4 2 8 9 7

swap

j i
$\boxed{2}$ 3 1 4 5 8 9 7

$\boxed{2}$ 3 1 4 |5| 8 9 7

swap

j i
$\boxed{2}$ 1 3 4 5 8 9 7

swap

Pivot i j
|1| |2| $\boxed{3}$ 4 5 8 9 7

1 2 3 4 5 $\boxed{8}$ 9 7
Pivot i j

i j

1 2 3 4 5 $\boxed{8}$ 9 7

swap

j i

1 2 3 4 5 $\boxed{8}$ 7 9

swap

1 2 3 4 5 7 8 9

i i i j j

2 x 2 $\boxed{8}$ 4 3 9 2 7 10

i j i

$\boxed{8}$ 4 3 7 2 9 10

$\boxed{2}$ 4 3 7 $\boxed{8}$ 9 10

i i j j j j j

$\boxed{2}$ 4 3 7 $\boxed{8}$ 9 10

if A[i] $\leq$ pivot

i++

if A[j] $\geq$ pivot

j--

if (i<j) then

swap(A[i], A[j])

if (i>j) then

swap(pivot, A[j])

if (i=j)

i i j

8 $\boxed{4}$ 3 9 2 7 10

Algorithm Quicksort (A[l....r])

//Input : An orderable ~~sub~~-array A[l...r] of A[0...n-1]

// output : sorted array A[l...r]

    if (l<r) then

        s← partition (A[l...r])

        Quicksort (A[l...s-1])

        Quicksort (A[s+1...r])

    end if.

Algorithm partition (A[l...r])

//Input : An sub-array A[l...r] of A[0...n-1].

//output : partition position j

    p← A[l]

    i← l+1

    j ← r

    repeat

        repeat i←i+1 until A[i] ≥ p

        repeat j← j+1 until A[j] ≤ p.

        swap (A[i], A[j]) // last swap is Invalid

        until i>j

        swap (A[i], A[j]) // to avoid Invalid swap

        swap (A[l], A[j])

        return j

i
3

2

Time Analysis.

1) Input parameter – 'n' size of array A.
2) Basic operation – $A[i] \geq P$ $A[j] \leq P$ – comparision.
3) There is variation of time for the same size 'n'.

∴ find Bestcase, worstcase and averagecase individually

4. **Best case :-**

Partition results in approximately equal sub-groups

ex:-

5 2 4 7 8.

$\underbrace{2 \quad 4}_{n/2} \quad \boxed{5} \quad \underbrace{7 \quad 8}_{n/2}$

$$\therefore C_B(n) = C_B(n/2) + C_B(n/2) + \underset{\substack{\downarrow \\ \text{min. no. of comparision.}}}{(n+1)}$$

$$C_B(n) = 2C_B(n/2) + (n+1)$$

until $C_B(1) = 0$

5) Applying Backward substitution method $n = 2^k$

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

$$= 2 \cdot C_B(2^{k-1}) + (2^k+1).$$

$$= 2(2 \cdot C_B(2^{k-2}) + 2^{k-1}+1) + 2^k+1$$

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

$$= 2^2 C_B 2^{k-2} + 2 \cdot 2^k + 2^1 + 2^0$$

$$= 2^3 C_B 2^{k-3} + 3 \cdot 2^k + 2^2 + 2^1 + 2^0$$

$$= 2^k C_B(\cancel{2^{k-k}}) + k \cdot 2^k + 2^{k-1} + \dots + 2^1 + 2^0$$

$$= k \cdot 2^k + 2^k + 2^{k-1} + \dots + 2^1 + 2^0$$

$$C_B(n) = k \cdot 2^k - \frac{[1 \cdot 2^k - 1]}{2-1}$$ $\left\{ S_n = \frac{ar^n - 1}{r-1} \right.$

$$C_B(n) = k \cdot 2^k - [2^k - 1].$$

$$2^k[k+1] + 1$$

$$C_B(n) \approx 2^k[k]$$ [for larger input value)

Apply $\log_2$ on both sides of $n = 2^k$.

$$\log_2 n = k \log_2 2$$

$$k = \log_2 n$$

$$\log_2(2^k) = k$$

$$C_B(n) = n \cdot \log_2 n$$

$$\boxed{C_B(n) \in \Omega(n \log_2 n)}$$ lower bound.

Best
15
7

worst
15
0 14

n
(n-1)

<u>Worst case</u> :-

If the partition creates one heavy group and the other empty group

$$C_w(n) = C_w(0) + C_w(n-1) + (n+1)$$ until $C_w(1) = 0$

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

$$[C_w(n-2) + (n-1+1)] + n+1$$

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

$$C_w(n-3) + (n-2+1) + n + (n+1)$$

$$C_w(n-3) + (n-1) + n + (n+1)$$

$c(0) + c(n+)n-1$

$n = n-1$

$$= C_w(n-(n-1)) + 3 + 4 + 5 + \dots + n + (n+1)$$

$$C_w(n) = 3 + 4 + 5 + \dots + n + (n+1).$$

$$= [1 + 2 + 3 + 4 + 5 + \dots + n + (n+1)] - 3$$

↳ Arithmetic progression.

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

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

$$\boxed{C_w(n) = \frac{(n+1)(n+2)}{2} - 3}$$

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

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

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

For large value of input (n)

$$C_w(n) \approx n^2/2$$

$$\boxed{C_w(n) \in \Theta(n^2)}$$ Quadratic Order of growth

Average case

$C_A(n) \approx 2n\log_e n$. $2n\log_e n$

$C_A(n) \approx 1.33\, n\log_e n$

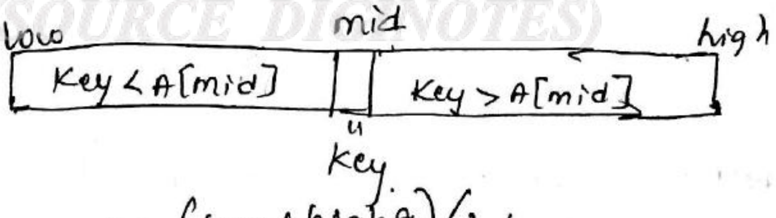
i.e 33 % more than Best case.

$$\boxed{C_A(n) \in \Theta(n\log_e n)}$$

18/3/17

Binary search

Ordered list - A



mid = (low+high)/2;

Algorithm Binary-search (A[low....high], key).

//Input : An ordered array A[low....high] of 'n' elements and search element - key.

//output : return 1 if key is found else 0.

```
if (low ≤ high) then
    mid ← (low + high)/2
    if (key = A[mid]) then
        return(1)
    else if (key < A[mid]) then
        return(Binary-search(A[low....mid-1], key))
    else
        return(Binary-search(A[mid+1....high], key))
    end if
else
    return(0)
end if
```

Time Analysis

1) Input parameter - n size of array A
2) Basic operation - comparision = key = A[mid]
3) No of comparison vary based on the position of key ∴ Estimate best / worst / Average case individually

4) Best case

$C_B(n) = 1.$ found at mid.

$\boxed{C_B(n) \in \Omega(1)}$ constant order of growth

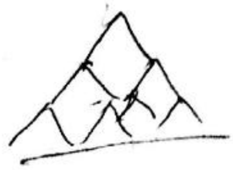
Worst case

$C_W(n) = C_W(n/2) + 1$ till $C_W(1) = 1$.

$n = 2^k$.

$C_W(n) = C_n(2^k/2) + 1$

$$C_w(n) = C_w(2^{k-1}) + 1$$
$$= [C_w(2^{k-2}) + 1] + 1$$
$$= C_w(2^{k-2}) + 2.$$
$$= C_w(2^{k-3}) + 3.$$



$$\vdots$$

$$C_w(\cancel{2^{k-k}}) + k.$$
$$= 1 + k.$$
$$= k + 1$$

$= n = 2^k$

= apply log on B.S. $k = \log_2 n$.

$= \log_2 n + 1.$

For large value of n:

$C_w(n) \approx \log_2 n.$

$$\boxed{C_w(n) \approx O(\log_2 n)}$$

logrithmic order of growth.

Average case

$\lceil \log_2 n \rceil$

if $n=10$ - $\lceil \log_2 10 \rceil = 4$

$$\therefore C_a(n) = \frac{1}{\log_2 n} \sum_{i=1}^{\log_2 n} i$$

$$= \frac{1}{\log_2 n}[1 + 2 + 3 \cdots + \log_2 n]$$

$$= \frac{1}{\log_2 n} \cdot \frac{\log_2 n(\log_2 n+1)}{2}.$$

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

$$C_A(n) = \frac{\log_2 n+1}{2}.$$

$$C_A(n) \approx \frac{1}{2}\log_2 n.$$

$$\boxed{C_A(n) \in \Theta(\log n)}.$$

Maximum and minimum problem

Bruteforce technique

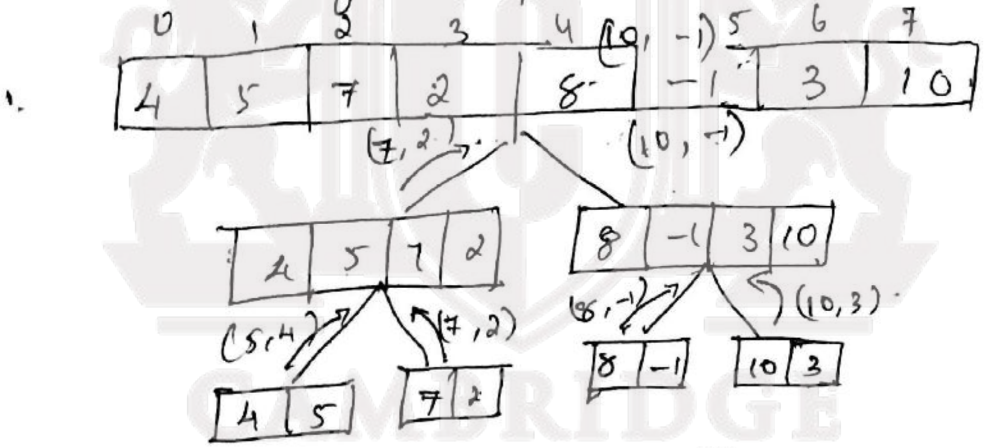
5 7 2 8 -1 3 10.

max: -999 ~~5~~ ~~7~~ ~~8~~ $\boxed{10}$

min: 999 ~~5~~ ~~7~~ -1 $\boxed{-1}$

2n no of comparision.

in divide and conquer technique, we reduce no of comparision.



comparision = 10

① (0, 7, 10, -1)

② (0, 3, 7, 2) ⑤ (4, 7, 10, -1).

③ (0, 1, 5, 4) (2, 3, 7, 2) ④ (4, 5, 8, -1) ⑥ (6, 7, 10, 3) ⑦

(32, 7)

Exa 20 17 10 25 32 11 7 22

(25, 10) (32, 7)

20 17 10 25 32 11 7 22.

(20, 17) (25, 10) (32, 11) (22, 7)

20, 17 25, 10 32, 11 22, 7.

Algorithm : (i, j, Max, min)
// input : limits of array i, j
// output : ~~maximum~~ max and ~~Minimum~~ min values of array.

```
if (i = j) then,        // one element
    max ← min ← A[i]
else if ((j - i) = 1) then    // two element
    if (A[i] < A[j]) then
        max ← A[j], min ← A[i].
    else
        Max ← A[i], min ← A[j]
    end if
else
    mid ← (i + j)/2.
    MaxMin( i, mid, max, min )
    MaxMin ( mid+1, j , max1, min1)
    if (max1 > max ) then
        max ← max1
    end if
    if (min1 < min) then
        min ← min1
    end if
end if
```

<u>Time Analysis</u> :-

1) Input parameter - no of elements b/w i - j as n.

2) Basic operation - comparision
A[i] < A[j], max1 > max, min1 < min

3) Depending only on array size -'n'
no variations

Date :
Page No :

4. $T(n) \begin{cases} 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ T(n/2) + T(n/2) + 2 & \text{if } n > 2 \end{cases}$

5) $T(n) = 2T(n/2) + 2$.

$n = 2^K$.

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

$= 2[2.T(2^{K-2}) + 2] + 2$.

$= 2^2. T(2^{K-2}) + 2^2 + 2$.

$= 2^3 T(2^{K-3}) + 2^3 + 2^2 + 2$.

$\vdots$

upto $n = 1$ $\quad 2^K T(\cancel{2^{K-K}}) + \underbrace{2^K + 2^{K-1} + \dots + 2}$

Geometric progression

but we are missin 2^0

$= 2(2^{K-1} + 2$

$2(2^{K-1} + 2^{K-2} + \dots 2 + 2^0)$

$S_n = \frac{ar^n - 1}{r - 1}$

$= 2.\left[\frac{1 . 2^K - 1}{2 - 1}\right]$

$= 2[2^K - 1]$

$= 2(n - 1)$

for a very large value of n.

$T(n) \approx 2n$.

$\boxed{T(n) \in \Theta(n).}$

II. <u>using Masters theorem</u> $T(n) = a \cdot T(n/b) + f(n)$

Algorithm : recurrance relation

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

$$a = 2, \quad f(n) = 0, \quad b = 2, \quad d = 0.$$

$$a \cdot \quad b^d$$

$$2 > 2^0$$

$$2 > 1$$

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

$$\in \Theta(n^{\log_2 2})$$

$$\boxed{T(n) \in \Theta(n)}$$

<u>Matrix multiplication</u> :-

$$\left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 0 & 1 & 2 \\ 3 & 4 & 5 & 6 \end{array}\right]$$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} A_{11} * B_{11} + A_{12} * B_{21} & A_{11} * B_{12} + A_{12} * B_{22} \\ A_{21} * B_{11} + A_{22} * B_{21} & A_{21} * B_{12} + A_{22} * B_{22} \end{bmatrix}$$

$$T(n) = \begin{cases} 1 & n = 1 \\ 8T(n/2) & n > 1 \end{cases}$$

$$T(n) = 8T(n/2) + 0 \qquad \text{until } T(1) = 1$$

$$a = 8, \; b = 2, \; d = 0, \; f(n) = 0.$$

$$a \quad b^d$$

$$8 > 2^0$$

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

$$\in \Theta(n^{\log_2 8})$$

$$T(n) \in \Theta(n^3)$$

Strassen's matrix multiplication.

$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$. $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$P = (A_{11} + A_{22}) * (B_{11} + B_{22})$.
$Q = (A_{21} + A_{22}) * B_{11}$.
$R = A_{11} * (B_{12} - B_{22})$
$S = A_{22} * (B_{21} - B_{11})$.
$T = (A_{11} + A_{12}) * B_{22}$.
$U = (A_{21} - A_{11}) * (B_{11} + B_{12})$.
$V = (A_{12} - A_{22}) * (B_{21} + B_{22})$

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

$C_{11} = P + S - T + V$
$C_{12} = R + T$
$C_{21} = Q + S$.
$C_{22} = P + R - Q + U$.

multiplication = 7.
Addition / subtraction = 18

21/3/17

Ex :-

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

P = 30.   T = 20
Q = 5   U = -3.
R = -4   V = -7
S = 8

$$C = \begin{bmatrix} \overset{11}{P+S-T+V} & \overset{16}{R+T} \\ \overset{13}{Q+S} & \overset{18}{P+R-Q+U} \end{bmatrix}.$$

$V \cancel{\cdot (A_{12}-A_{22}}$)

Analysis

1) Input parameter 'n' . order of matrix $n \times n$.
2) B.O. – multiplication.
3) Depends only on 'n'
4)

$$T(n) = \begin{cases} 1 & n=1 \\ 7T(n/2), & n \geq 2 \end{cases}$$

5) $T(n) = 7T(n/2)$ until $T(1) = 1$

$a=7,\ b=2,\ f(n)=0\ d=0.$

$a \cdot b^d.$

$7 \quad 2^0$

$7 > 1$

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

$T(n) \in \Theta(n^{\log_2 7})$

$$\boxed{T(n) \in \Theta(n^{2.81})}$$

Decrease and Conquer

The problem of i/p size n is decreased by .

→ Constant value . (Ex . $a^n \to a^{n-1}$ ~~n → n~~ -1)

→ constant factors (ex $a^n \to a^{n/2} \cdot a^{n/2}$.

→ Decrease by variable size (Ex GCM (m, n % m)

→ constant value → BFS, DFS, Insertion sort, Topological sorting

Topological ordering :- / sorting

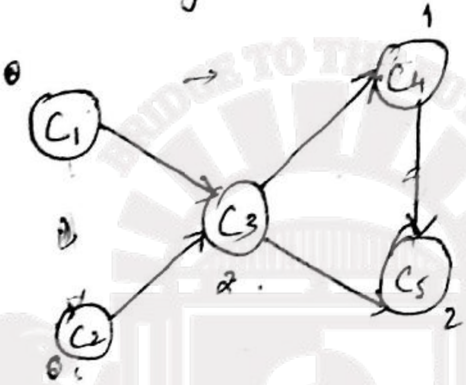
Its a graph traversal technique exclusively on DAG → Directed Acyclic technique.

C_1, C_2, C_3, C_4, C_5.

→ To attempt to C_3, either C_1 or C_2 should be complete

→ To attempt C_4, C_3 should be complete.

→ To attempt C_5, either C_4 or C_3 should be complete



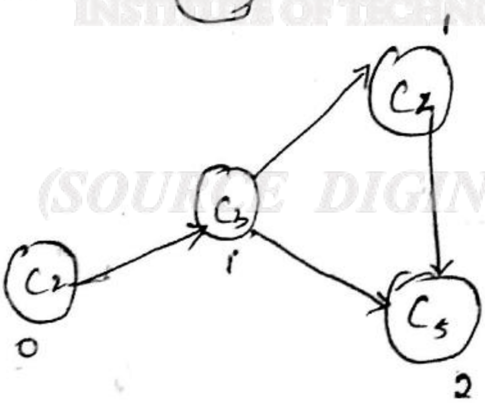
To solve Topological ordering

1) DFS method
2) Source Removal method.

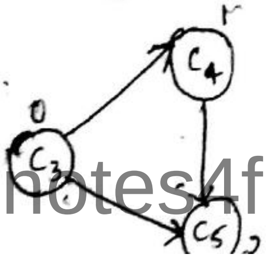
1. Source Removal method

source → vertex with indegree - zero.

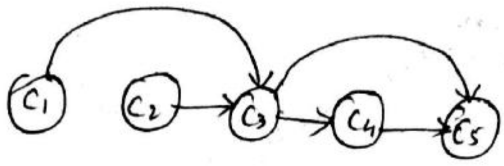
Remove C_1



Remove C_2



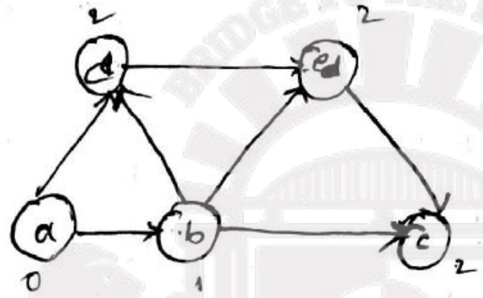
Remove C_3



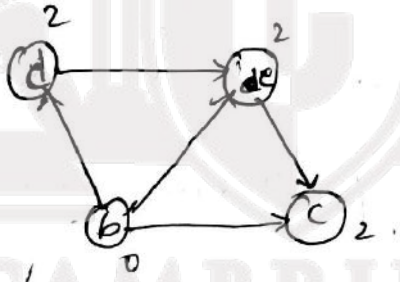
Remove C_4

Remove C_5

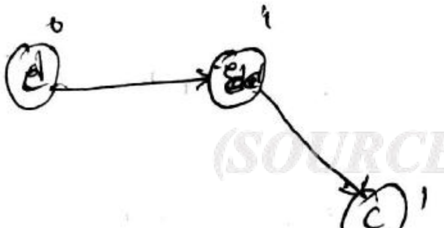
Ex 2 :-



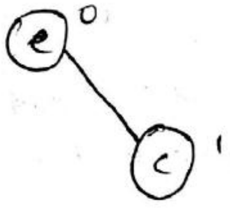
Remove a



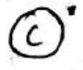
Remove b



Remove d



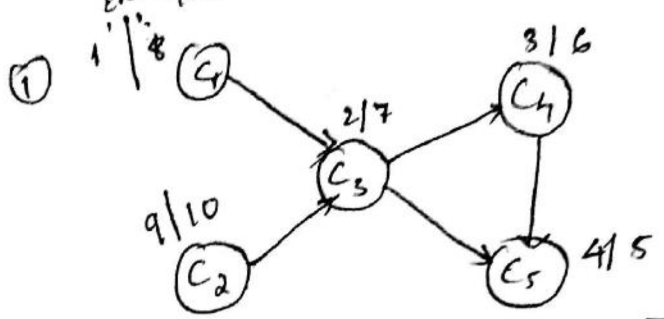
Remove e.

e 

Remove c.

a b d e c topological order

Topological ordering using DFS method

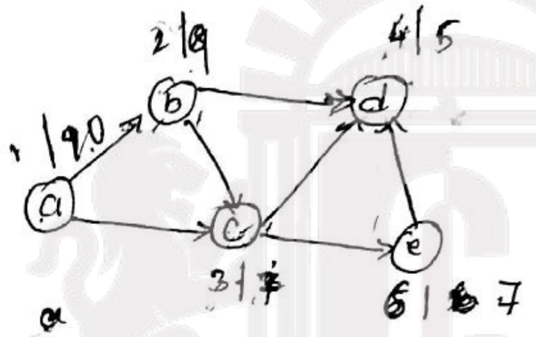


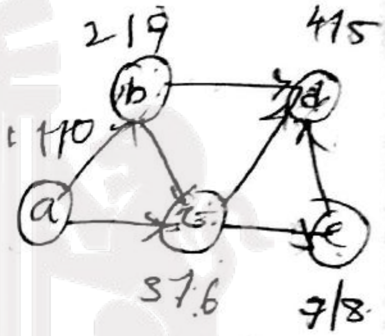
source clearance method

Entry time | exit time

$\boxed{C_2\ C_1\ C_3\ C_4\ C_5}$

topological order.





a b c e d.   a b c e d

Control abstract for Divide and conquer

Type D and c (P)

{

→ Smallest enough to solve p by Staight forward method

if Small(P) then .

return S(P) ⟶ solution of P

else

Divide problem ~~in steps~~ instance

. p into $P_1\ P_2\ P_3 \ldots P_k$ where $k \geq 1$

solve each subproblem by ~~Divide~~ D and C (P_1),

~~Divide~~ and C (P_2) ------

return Combine(D and C(P_1), ~~and~~

D and C(P_2) ... D and C(P_k)

} end if .

30/3/17

MODULE - 3

Greedy Technique.

A problem is solved through sequence of subproblem, each sub-problem is solved by greedy technique.

-> feasibility -> limit, budget
-> locally optimal.
-> Irrevocable. -> no replacement.

Optimising problem -> greedy technique.

Defination :-

Constructing a solution through sequence of steps, expanding partially construction sol^n obtained so far, until the complete sol^n for the problem is reached.

Each step should have 3 properties.

1. **Feasible** :- The subproblem should satisfy the imposed constraints.

2. **Locally optimal** :- Among the Feasible solution for the subproblem, it should choose the best sol^n called as optimal sol^n.

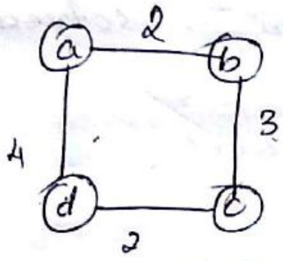
Irrevocable :-
Once the subproblem is solved, it should Remain unchanged

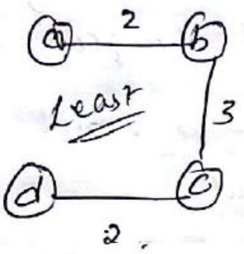
Problems

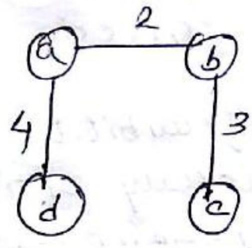
1) minimum spanning tree
2) knapsack problem
3) coin change problem
4) single source - shortest path

1. Spanning tree [minimum]

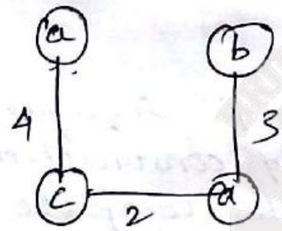
Acyclic graph which has span at all nodes

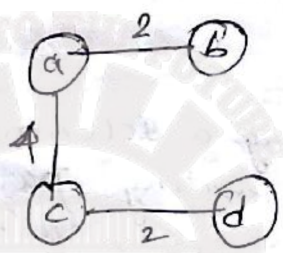






cost = 2+3+2=7   4+2+3 = 9.



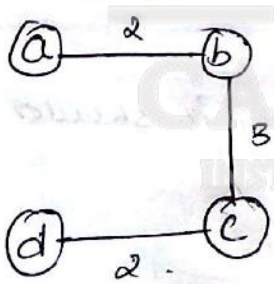


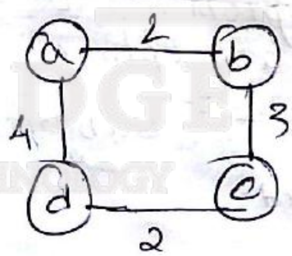
4+2+3 = 9   4+2+2 = 8.

To solve minimum spanning tree we've two algorithms.

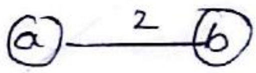
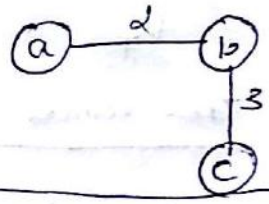
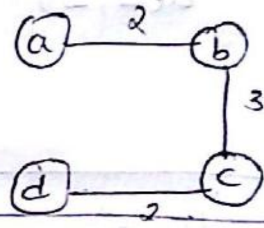
1. Prim's Algorithm :- Nearest neighbour first

Starts with refrence node (Arbitrarily selected)





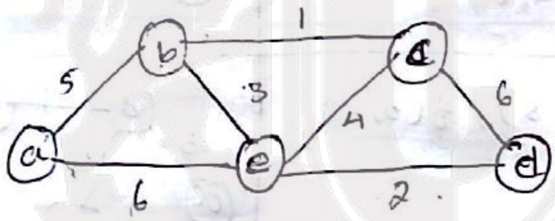
minimum Spanning tree   normal tree

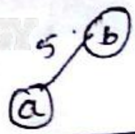
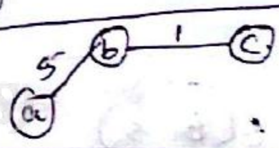
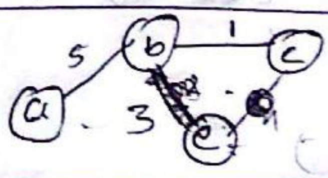
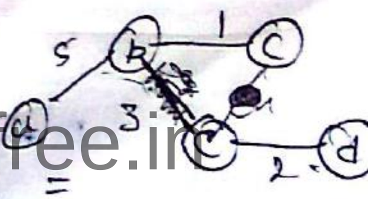
Tree vertices	Remaining vertices	Spanning tree
Initial a(–, –)	b(a, 2), c(–, ∞), d(a, 4)	
b(a, 2)	c(b, 3), d(a, 4)	
c(b, 3)	d(c, 2) .	
d(c, 2)	NIL	

→ Repeat these ~~spe~~ steps until there is no extra vertices.

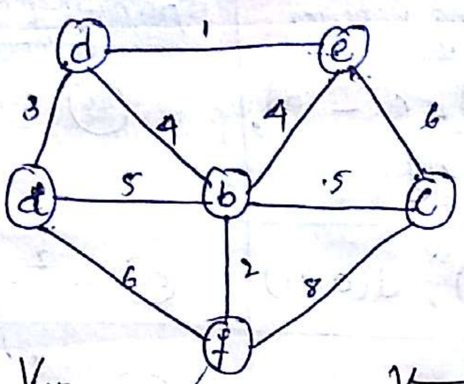
→ The cost of the minimum edges = 2+3+2 = 7.

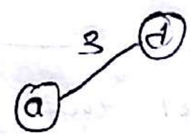
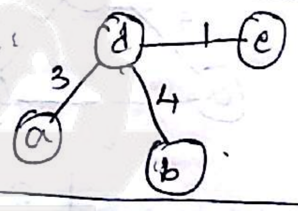
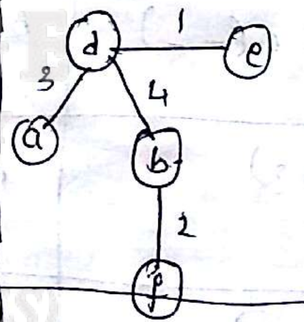
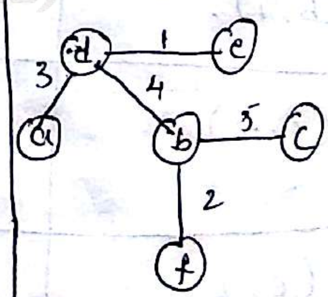
(2)



Tree vertices	Remaining vertices	Spanning tree
Initial a(–, –)	b(a, 5), c(–, ∞) d(–, ∞) e(a, 6)	
b(a, 5)	c(b, 1), e(b, 3). d(–, ∞).	
c(b, 1).	d(c, 6), e(c, 4)	
e(c, 4)	d(e, 2)	
d(e, 2)	~~Nil~~ . Nil. minimum cost = 10.	

31/3/17



V_T Tree vertices	~~$V-\Phi$~~ $V-V_T$ Remaining vertices	E_T Spanning tree
a(-,-)	b(a,5) c(-,∞) d(a,3) e(-,∞) f(a,6)	
d(a,3)	b(d,4), c(-,∞) e(d,1), f(a,b)	
e(d,1)	c(e,6), b(d,4) f(a,6).	
b(d,4)	c(b,5), f(b,2)	
f(b,2)	c(b,5)	
c(b,5)	NIL	

$3+1+4+5+2 = 15$

Prim
Algorithm Prim (G)
//Input : A weighted connected graph $G = \{V, E\}$
//Output : E_T set of edges constructing minimum spanning tree

$V_T \leftarrow \{v_0\}$

$E_T \leftarrow \phi$

for $i \leftarrow 1$ to $|V|-1$ do

Find the minimum weighted edge $e^* = (u^*, v^*)$ among all the edges (u, v) such that u is in V_T and v is in $V - V_T$

$V_T \leftarrow V_T \cup \{v^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

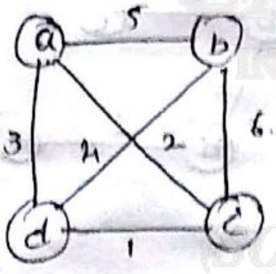
end for

return E_T.

__Efficiency of Prims algorithm__ [out of syllabus]

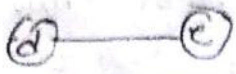
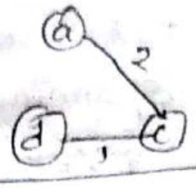
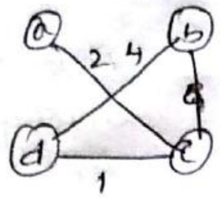
$O(|E| \log |V|)$ $E \rightarrow$ is the edges.
$V \rightarrow$ is the vertices.

__KRUSKAL'S Algorithm__

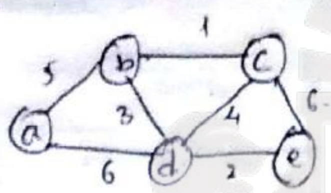


dc, ~~db~~ ac, ad, db, ab, bc.

P.T.O.

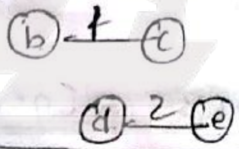
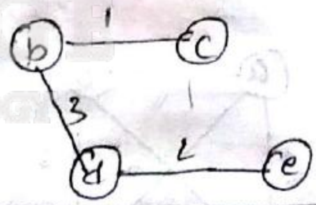
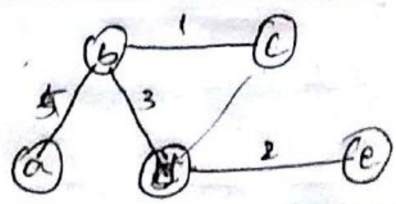
Tree edge	Acyclic / cyclic	Spanning tree
dc	Acyclic	
ac	Acyclic	
ad	cyclic	X
bd	Acyclic	

2.



$\overset{1}{bc}$ $\overset{2}{de}$ $\overset{3}{bd}$ $\overset{4}{dc}$ $\overset{5}{ab}$

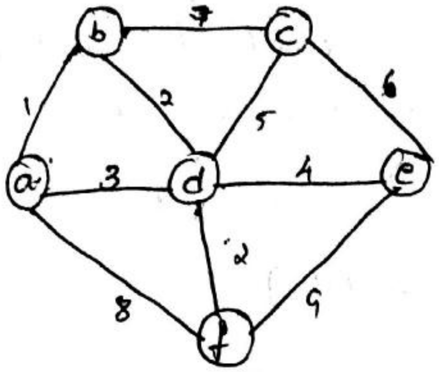
~~cd~~ $\overset{6}{ad}$ or $\overset{6}{ce}$

Tree edge	Acyclic / cyclic	Spanning tree
bc	Acyclic	
de	Acyclic	
bd	Acyclic	
dc	cyclic	X
ab	Acyclic	
ad	cyclic	X
ce	cyclic	X

$5+3+2+1 = 11$

4/4/17

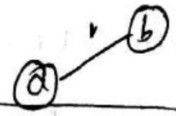
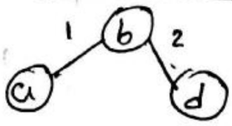
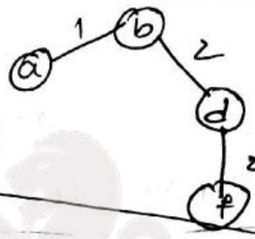
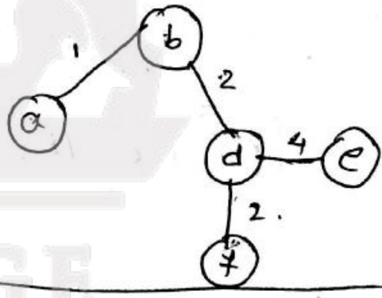
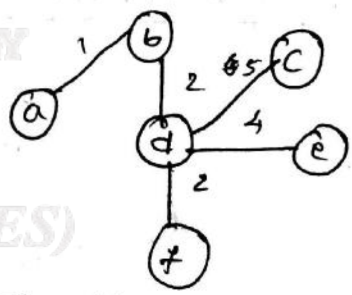
3.



e_k

$\overset{1}{ab}$ $\overset{2}{bd}$ $\overset{2}{df}$ $\overset{3}{ad}$ $\overset{4}{de}$ $\overset{5}{cd}$ $\overset{6}{ce}$ $\overset{7}{bc}$ $\overset{8}{af}$ $\overset{9}{ef}$

E_T

Tree edges	Acyclic/cyclic	Min.cost spanning tree
ab	Acyclic	
bd	Acyclic	
df	Acyclic	
ad	cyclic	✗
dc	Acyclic	
cd	Acyclic	

$= \underline{14}$

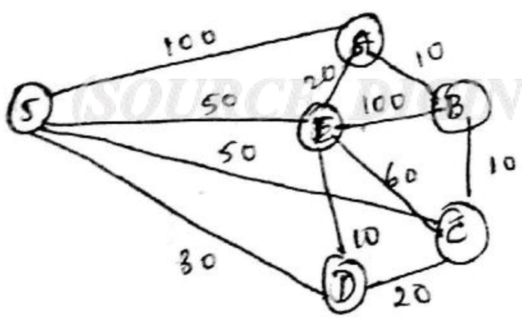
P.T.O.

Algorithm :- Kruskal's
//Input : A weighted -connected graph $G = \{V, E\}$
//output : E_T . set of edges constructing MST
Sort all edges of E in non-decreasing order ie.
$w(e_1) \leq w(e_2) \leq \ldots \leq w(e_n)$
$E_T \leftarrow \phi$
ecounter $\leftarrow$ 0
$K \leftarrow 0$ // Index for choosing edges
while (ecounter < |V| -1) do
    $K \leftarrow K+1$
    if ($E_T \cup \{e_k\}$ is acyclic) then
        $E_T \leftarrow E_T \cup \{e_k\}$
        ecount $\leftarrow$ ecount +1
    endif
endwhile
Return E_T

Time efficiency

$\boxed{O(|E| \log |E|)}$

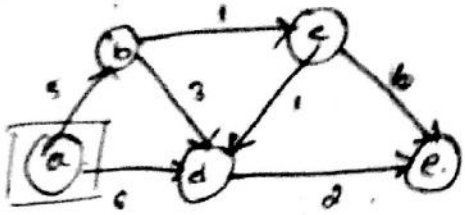
Single source shortest path (SSSP)



1. Dijkstra's algorithm [Dia - k - stra]
2. used in Link - stats routing algorithm

Dijkstra's algorithm is based on nearest neighbour ...

4/4/17



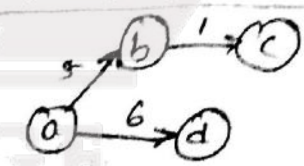
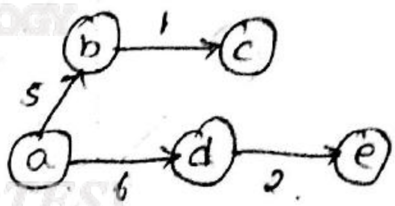
$a \to b = 5$

$a \to c = \cancel{\infty} \quad 5+1$

$a \to d = 6, \ \cancel{5+3}, \ \cancel{6+1}$

$a \to e = \infty \ \cancel{12}, \ 8$

a(— , —)
Penultimate vertex
cost

Tree vertices	Remaining vertices	Initial path
Initial a(–,0)	b(a,5), c(–,∞) d(a,6), e(–,∞)	a → a = 0.
b(a,5)	c(b,5+1), d(a,6) e(–,∞)	a → b = 5
c(b,6)	d(a,6), e(c,6+6)	a → b → c = 6
d(a,6)	e(d,8)	
e(d,8)	Nil	

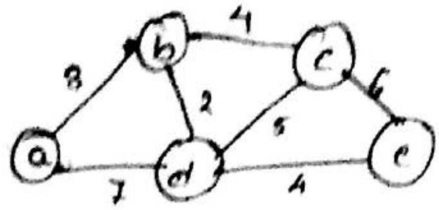
$a \to b = 5$

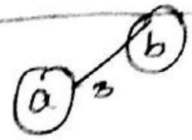
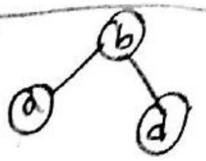
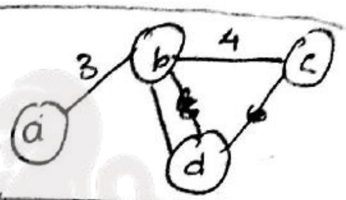
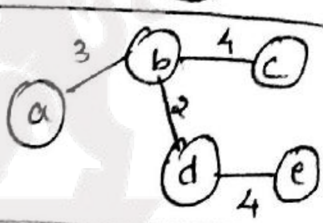
$a \to b \to c = 6$.

$a \to d = 6$.

$a \to d \to e = 8$

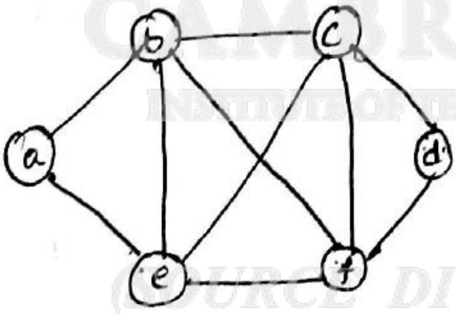
2.



Tree vertices	Remaining vertices	Shortest path
Initial a(-,0)	b(a,3) c(-,∞) d(a,7) e(-,∞)	a
b(a,3)	c(b,3+4) d(b,3+2), e(-,∞)	
~~c(b,7)~~ d(b,5)	~~d(a,7)~~ ~~e(~~ c(b,7) e(d,11)	
c(b,7)	e(d,11)	
e(d,11)	Nil	

Shortest path = 3+4+2+4 = 13.

3.



Algorithm :- Dijkstras (G, s).

// Input : A weighted converted graph $G = \{V, E\}$
and source vertex s

//output : d_v - shortest path from source vertex
to $v \in V - s$.

P_v - Penultinate vertex of v.

Initialize (Q) // priority queue.

for every vertex v in V

$d_v \leftarrow \infty$

$P_v \leftarrow null$

Insert(Q, v, d_v)

end for

$d_s \leftarrow 0$

Decrease (Q, s, d_s)

$V_T \leftarrow \phi$.

for $i \leftarrow 0$ to $|V| - 1$ do

$u^* \leftarrow$ Delete_min (Q).

$V_T \leftarrow V_T \cup \{u^*\}$

for every vertex u adjacent to u^* and belongs to $V - V_T$ do

~~if ((du*~~

if $((d_{u^*} + w(u^*, u)) < du)$

$du \leftarrow du^* + w(u^*, u)$

$P_u \leftarrow u^*$

Decrease (Q, u, du)

end if.

end for

end for.

6/4/17

Time ~~analysis~~ complexity of Dijkstra's algo

$$\boxed{O(\cancel{\log}\ |E| \log M)}$$

Knapsack Problem

n - items

weight = $\{w_1, w_2, w_3 \ldots w_n\}$

Price / profit / value : $\{P_1, P_2 \ldots P_n\}$

W = capacity

Feasibility condition : $w_1 + w_2 + \ldots \leq W$.

~~Out~~
Optimal solⁿ :- maximum value in Knapsack

Example

$n = 3$

Profit = $\{25, 15, 24\}$

weight = $\{18, 10, 14\}$

$W = 25$

Brute force

Exahustives search method {time consuming but good}

	condition	profit
{0} =	0 < 25	= 0
{1} =	18 < 25	= 25
{2} =	10 < 25	= 15
{3} =	14 < 25	= 24
{1, 2} =	28 ≮ 25	= not feasible
{1, 3} =	32 ≮ 25	= not feasible
$\boxed{\{2, 3\} = 24 < 25 = 39}$		
{1, 2, 3} =	42 > 25	= Not feasible.

knapsack problem

- .0/1 knapsack [completely or dont take]
- Fractional knapsack → greedy technique

Fractional knapsack → greedy technique

1) Find the ratio of Profit/weight

2) Reorder the items base P_i/w_i such that $P_1/w_1 \geq P_2/w_2 \geq P_3/w_3 \geq \ldots$

items	Profit	weight	P_i/w_i	W Remaining capacity	Fraction of items	Profit of the knap sack
3	24	14	1.71	25 − 14 = 11	1	0 + 1×24 = 24
2	15	10	1.5	11 − 10 = 1	1	24 + 1×15 = 39
1	25	18	1.38	1 − 1 = 0	1/18	39 + (1/18 × 25) = 40.38

Ex1 :- $n=3$

Profit $= \{25, 15, 24\}$

weight $\{18, 10, 14\}$

$W = 25$

items

Item	P	W	Pi/wi
1	25	18	1.38 III
2	15	10	1.5 II
3	24	14	1.71 I higher value first

2. $n=4$

weight $= \{7, 3, 4, 5\}$

profit $= \{42, 12, 40, 25\}$

$W = 12$

Item	P	W	Pi/wi
1	42	7	6
2	12	3	4
3	40	4	10
4	25	5	5

items	Profit	weight	Pi/wi	W Remaining capacity	x[i] fraction of item	x[i] Profit of knapsack
3	40	4	10	12-4=8	1	40
1	42	7	6	8-7=1	1	42+40=82
4	25	5	5	1-1=0	1/5	$\boxed{87}$
2	12	3	4	0	0	87+0 = 87

Profit = 87

3. $n = 7$

weight = {2, 3, 5, 7, 1, 4, 1}

profit = {10, 5, 15, 7, 6, 18, 3}

$w = 15$

Algorithm Greedy-Knapsack (W, n)

//Input : P[1...n] Price w[1....n] weight of items sorted in non-decreasing order of P_i/w_i ratio .. W - Capacity of Knapsack.

//output : x[1...n] Solution vectors

U → ~~capacity~~ or accomodated in knapsack (capacity of knapsack) → left out capacity

```
for i ← 1 to n do
        x[i] ← 0
end for
U ← W
for i ← 1 to n do
      if (w[i] > U) then
          break.
      end if
      x[i] ← 1.0
      U ← U - w[i]
end for
if (i ≤ n)
    x[i] ← U/w[i]
end if
return (x)
```

6. $n = 3$

$P\{25, 24, 15\}$

$W\{18, 15, 10\}$

$W = 20$

Item	P	W	P_i/W_i
1	25	18	1.36
2	24	15	1.7
3	15	10	1.5

capacity of knapsack

remaining weight

items	P	W	U = 20	x
2	24	15	20 − 15 = 5	ϕ 1.0
3	25	10	5 − 5 = 0	$\phi = \frac{7}{10}$ = 0
1				

return
= 8
− 24
= 7.5
= 0
31.5

8/4/17

Job-sequencing /sheduling with deadline.

Set of jobs − n
each job associated with deadline
(time to complete).
each job is associated with price
Feasibility condition : sequenced jobs have to be completed with that deadline.

Optimal solution : Sequence with maximum profit/pi

Ex : $n = 5$ jobs

Job no :-	1	2	3	4	5
deadline	3	3.	2	1	2
Price	5	1	20	10	15

Reorder

based price
Descending order

Job considered	Price	deadline	Action Taken	Total profit
3	~~20~~ 20	2	[, 3,] (3, 2)	0+20=20
5	15	2	[5, 3,] (5, 1)	20+15=35
4	10	1	not scheduled	35
1	5	3	[5, 3, 1] (1, 3)	40
2	1	3	Not sequenced	40

Profit = 40

Ex: 2 n = 7

Job no:-	1	2	3	4	5	6	7
deadline	1	3	4	3	2	1	2
Price	3	5	20	18	1	6	30
	6	5	2	3	7	4	1

Job considered	Price	deadline	Action Taken	Total Profit
7	30	2	[, 7, ,] (7, 2)	30
3	20	4	[, 7, , 3] (3, 4)	30+20=50
4	18	3	[, 7, 4, 3] (4, 3)	50+18=68
6	6	1	[6, 7, 4, 3] (6, 1)	68+6=74
2	5	3	not scheduled	74
1	3	1	not scheduled	74
5	1	2	Not scheduled	74

Algorithm :- Job-sequencing (d, P, n).

// Input :- Jobs ordered based on price in non-increasing order d-deadline, P-Price, n-no of jobs.

// output :- sequenced Job - J.

J ← {1}

for i ← 2 to n do

if (all the jobs in J ∪ {i} can be completed with deadline) then

J ← J ∪ {i}.

end if

end for

return J

General method of Greedy Technique

Note: by default need to be included in greedy technique explaination.

Algorithm Greedy (a, n)

// Input :- An array a[1...n] of subproblems

// output :- Feasible and optimal solution.

solution ← φ

for i ← 1 to n do

x ← select (a) // sol^n of subproblem i to x.

if (feasible (solution, x)) then

solution ← Union (solution, x)

end if

end for

return (solution)

15/4/17

ASCII / EBCDIC.
Fixed length coding.

Huffmann coding

coding Technique to represent non-numeric elements with binary value.

-> variable length coding.

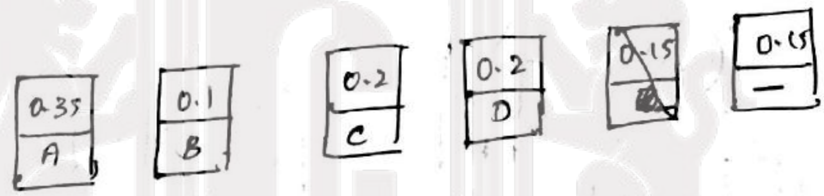
-> Based on the frequency of occurence of the character, the no of bits to represent varies. ie frequently occuring characters will be assigned with less no of bits. and rarely occuring characters with maximum no of bits

-> Huffmann coding is also called as one of the loss less compression technique.

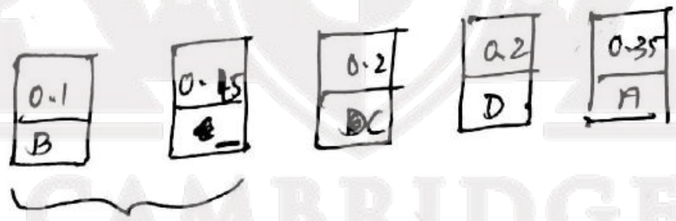
Ex:-

Character	A	B	C	D	E
frequency Probability	0.35	0.1	0.2	0.2	0.15

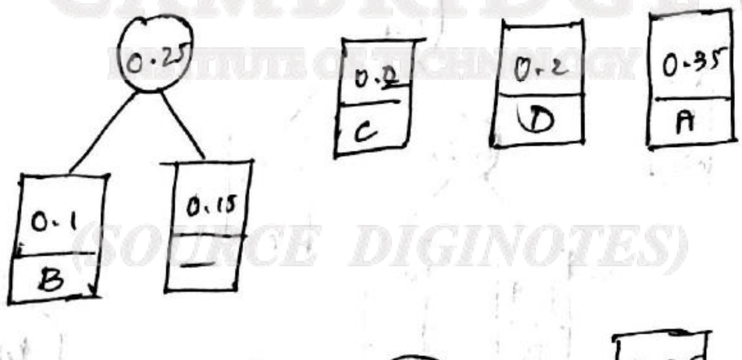
construct a tree for every character



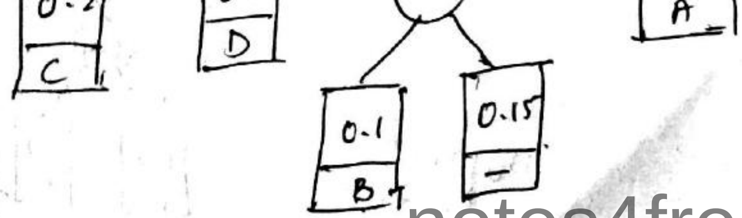
Reorder (ascending)



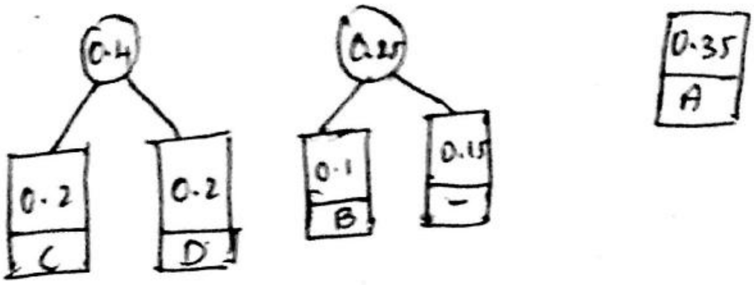
merge



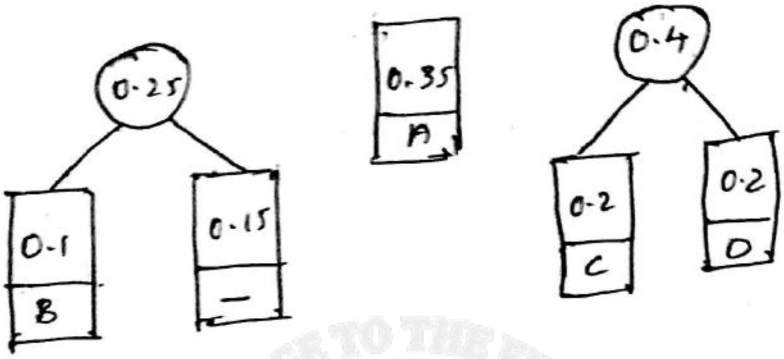
Reorder



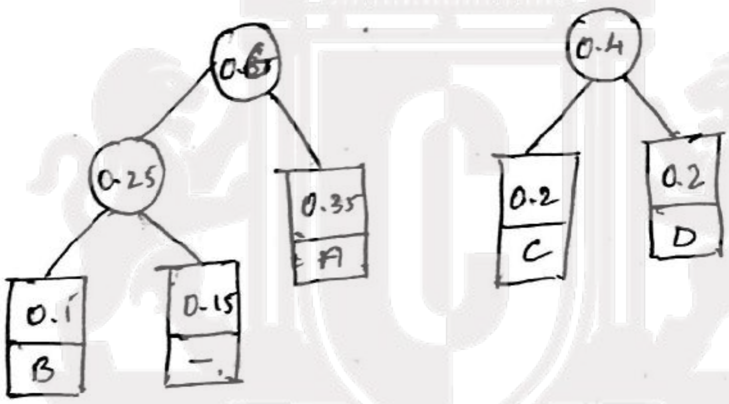
merge.



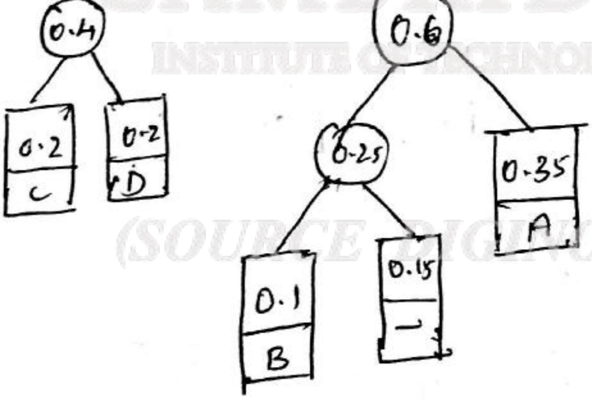
Reorder



merge

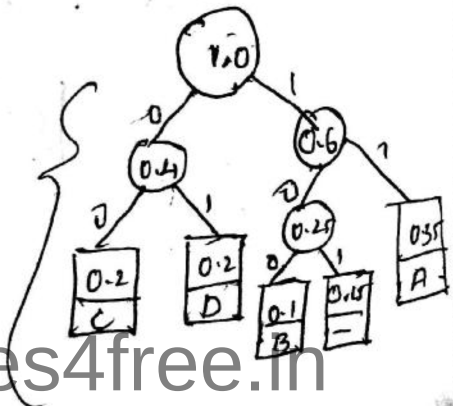


Reorder



merge

Huffman tree

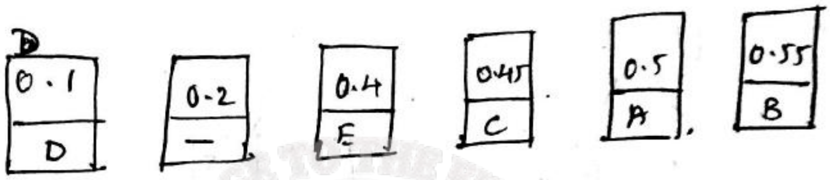


A	B	C	D	—
11	100	00	01	101

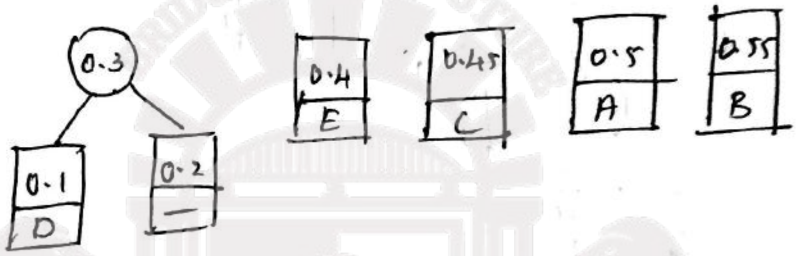
Example 2

character	A	B	C	D	E	—
Probability	0.5	0.55	0.45	0.1	0.4	0.2

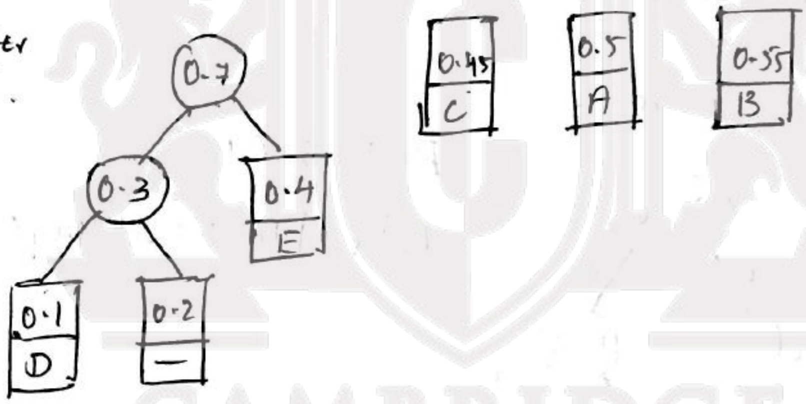
Reorder



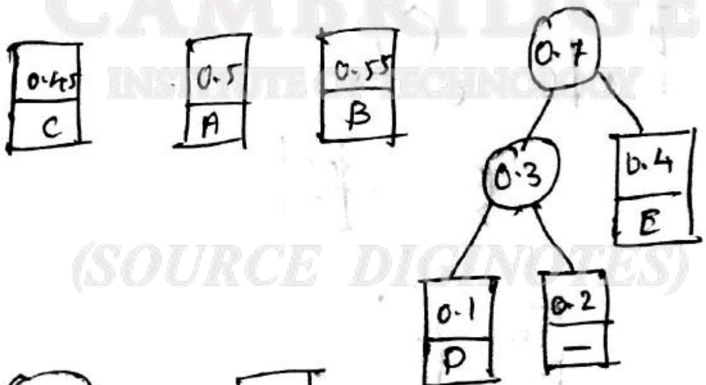
merge.



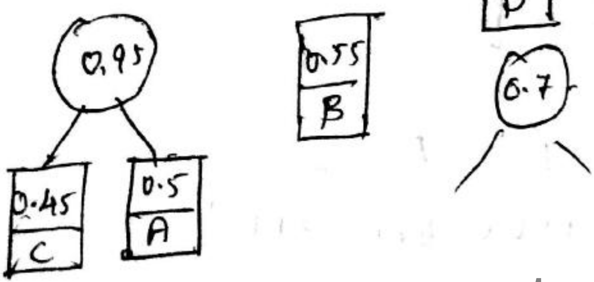
~~Reorder~~
merge.



Reorder

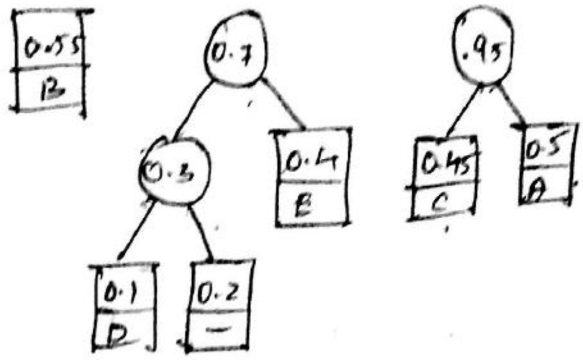


Merge

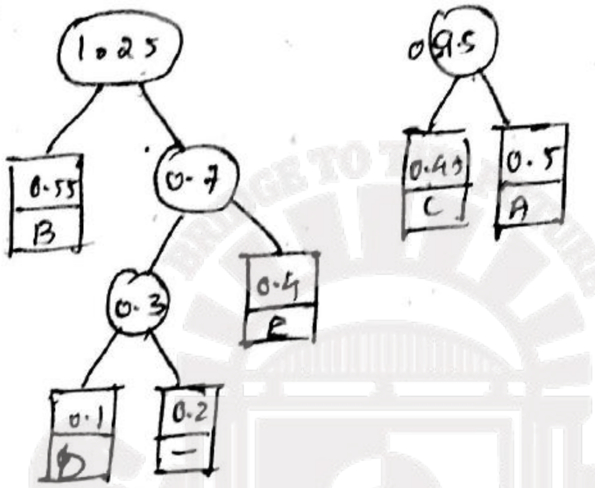


Vector → array

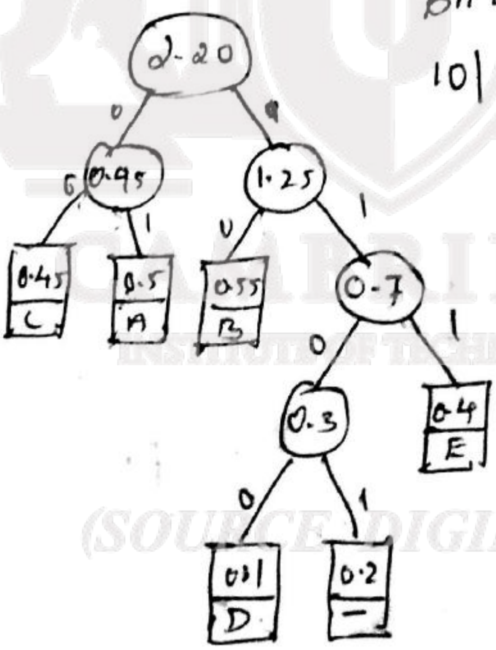
Reorder



Merge.



Reorder and merge.



BAD DAA.

10|01|1100|1101|1100|01|01

A	B	C	D	E	_
01	10	00	1100	111	1101

Algorithm :- Huffmantree .
//Input : A ~~V~~ (array) vector of characters and their frequency.
//output : Huffman tree .

step 1 : Initialize 'n' node trees and label them with the character of the alphabet. Record the frequency of ~~the~~ each character in its tree root to indicate the tree's weight

Step 2 : Repeat the following operation until a single tree is obtained.

a) Find the two trees with smallest weight .

b) make them as left and Right subtree of a new tree .

c) Record the sum of their weights in the root of the new tree as its weight .

step 3 :- Mark the left edge with '0' and right edge with '1'.

28/4/17.

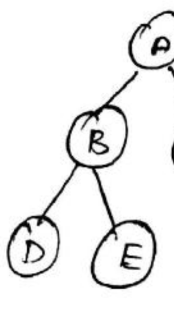
Heap sort

uses Transform and conquer design technique .

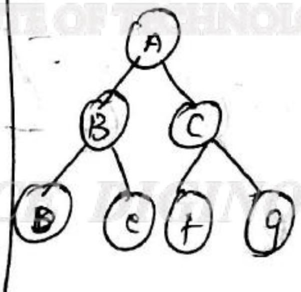
Heap tree

* Binary tree .
* Essentially complete Binary tree .

Strictly B.T



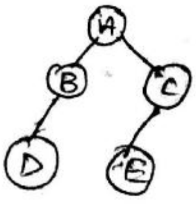
complete B.T .



→ 2^d .
→ children same level
→ level completely fill
→ $2^{d+1} - 1$ → total no of node .

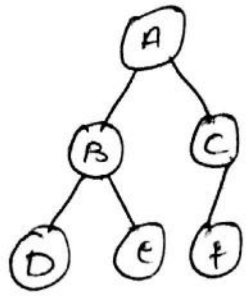
height = 1+depth .

Almost complete B.T



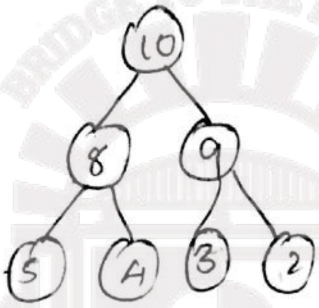
$2*k$
$2*k+1$

* Essentially complete Binary tree.
  is an almost complete B.T.
  missing nodes are from Right to left.



* Parental Dominance.

Parent node value should be larger than Childern (Descending heap tree



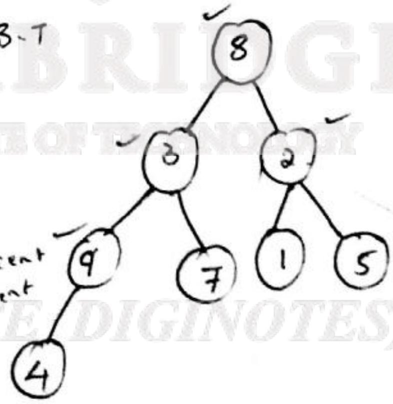
18/4/17

<u>Construction of Heap tree</u> (& Heapification)

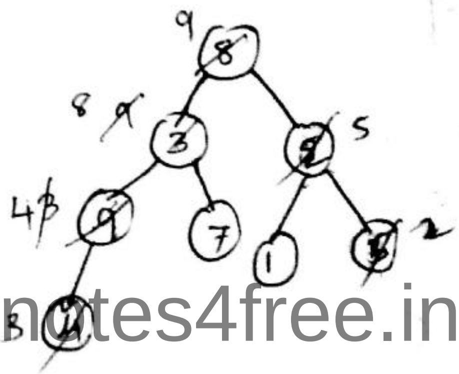
	1	2	3	4	5	6	7	8	
H[1....n]	8	3	2	9	7	1	5	4	\ \ \ \

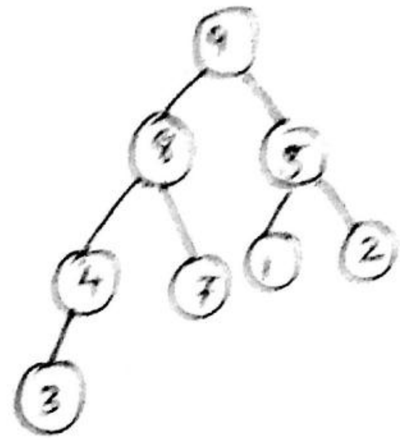
Represent Array
into essentially C.B.T

Recent parent



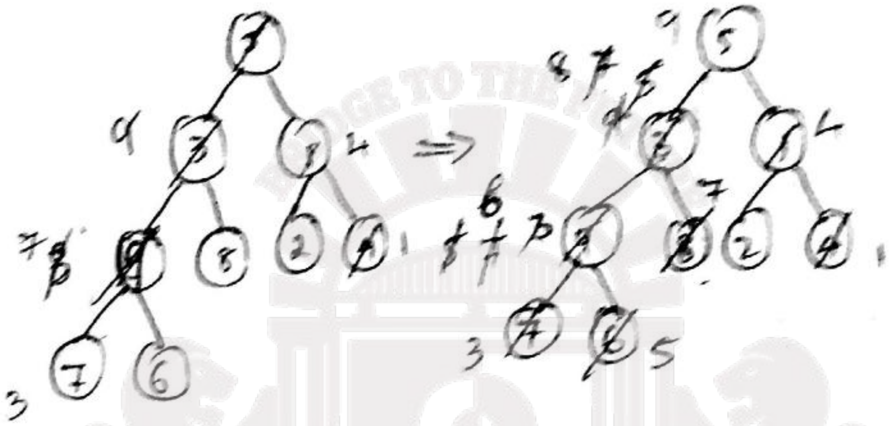
From recent Parent
try to satisfy
parental dominance.

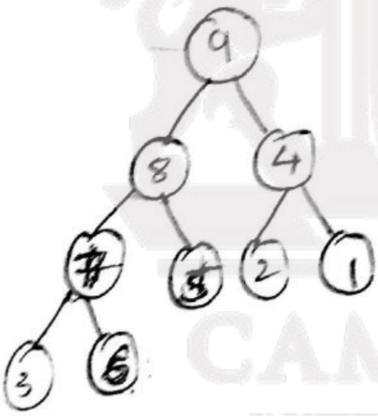




2.

5	3	1	9	8	2	4	7	6





<u>Algorithm</u> :- Heap_Bottom up (H[1....n])

// Input : An array H[1....n] of Orderable elements.

// Output ; A ~~map~~ heap tree .H[1....n]

$n/2$ → recent parent

for i ← ⌊n/2⌋ downto 1 do.

k ← i

v ← H[k]

Heap ← false.

while (not heap and 2*k ≤ n) do.

j ← 2*k

```
if (j<n) then  // there are 2 children
   if (H[j] < H[j+1]) then
          j ← j+1
   end if
end if
if (v ≥ H[j])
      heap ← true
else
   H[k] ← H[j]
      k ← j
   end if
end while
   H[k] ← v
end for
```

20/4/17

MODULE – 4

DYNAMIC PROGRAMMING (Planning)

Richard Bellman

It is a general method for optimizing multistage decision process.

Dynamic programming

It is a technique for solving problem through overlapping subproblems. Each subproblem is generated through recurrence relation solving them only once and recording the result for future use.

Ex:- Fib(5)

5 ≠ 0/1   Fib(4) + Fib(3) = 5

3   2

4 ≠ 0/1   Fib(3) + Fib(2)

2   1

3 ≠ 0/1   Fib(2) + Fib(1)

1

2 ≠ 0/1   Fib(1) + Fib(0)

1   0

Fib(0) = 0
Fib(1) = 1
Fib(2) = 1
Fib(3) = 2
Fib(4) = 3
Fib(5) = 5.

$^1C_1$ $a+b = a+b$ (1, 1)

$^2C_0 + {^2C_1} + {^2C_2}$. $(a+b)^2 = a^2 + 2ab + b^2$.   (1, 2, 1)

$^3C_0 + {^3C_1} + {^3C_2} + {^3C_3}$ $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$ (1, 3, 3, 1)

$^4C_0 + {^4C_1} + {^4C_2} + {^4C_3} + {^4C_4}$ $(a+b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$ (1, 4, 6, 4, 1)

$$C(n,k) = \begin{cases} 1 & k=0 \text{ or } n=k \\ C(n-1, k-1) + C(n-1, k), & \text{otherwise} \end{cases}$$

$C(5,3)$.

	0	1	2	3
0	1			
1	1	1		
2	1	2	1	
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10

$C(6,4)$

	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1
5	1	5	10	10	5
6	1	6	15	20	15

Algorithm :- Binomial- coefficient (n,k)

// Input :- Positive integers n, k where $n \geq k \geq 0$.

// output :- $C(n,k)$ coefficient.

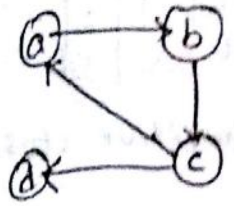
```
for i ← 0 to n do
    for j ← 0 to min (i, k) do
        if (j = 0 or i = j) then
            C[i, j] ← 1
        else
            C[i, j] ← C[i-1, j-1] + C[i-1, j]
        endif
    end for
end for
return (C(n, k))
```

20/11/13

Warshall's Algorithm

AND operation

Transitive closure of adjacency matrix



$R^{(0)} =$

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

(4)

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

Transitive closure

→ warshall Algorithm estimates transitive closure for the given adjacency matrix

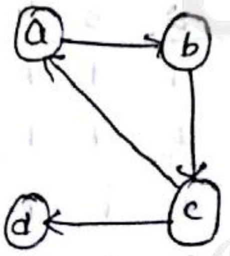
→ For any two vertices u and v of a given graph, if there is a direct path or via intermediate vertices then $R(u,v) = 1$ else zero

→ Initially $R^{(0)}$ is same as adjacency matrix

→ Estimate $R^{(1)}$ (vertex 1 as intermediate node), then $R^{(2)} \ldots R^{(n)}$

→ Declare $R^{(n)}$ as transitive closure

ex



$R^{(0)} =$

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

$R^{(a)} =$

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

$R^{(b)} =$

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

$R^{(c)}$

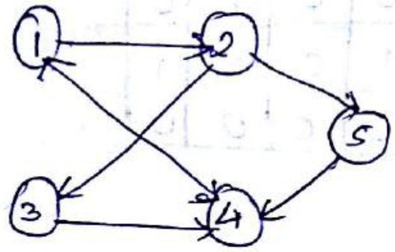
	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	1	1	1	1
d	0	0	0	0

$R^{(d)} =$

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

Transitive closure.

(2).



$R^{(0)} =$

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

$R^{(1)} =$

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

$R^{(2)} =$

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

$R^{(3)} =$

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

$R^{(4)} =$

	1	2	3	4	5
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1

$R^{(5)} =$

	1	2	3	4	5
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1

Warshall (A[1...n, 1...n])

Algorithm

//Input : Adjacency matrix A[1...n, 1...n] of graph

//Output : Transitive closure $R^{(n)}$

$R^{(0)} \leftarrow A$

for $k \leftarrow 1$ to n do

  for $i \leftarrow 1$ to n do

    for $j \leftarrow 1$ to n do

      $R^{(k)}[i, j] \leftarrow R^{(k-1)}[i][j]$ OR

        $(R^{(k-1)}[i, k]$ AND $R^{(k-1)}[k, j])$

    end for

  end for

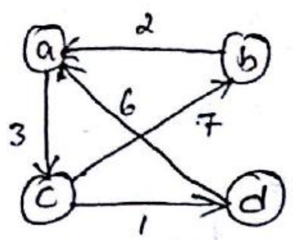
end for

Return $R^{(n)}$

27/4/17

Floyd's Algorithm [All pair Shortest Pair]

(1)



we've to find the shortest path

$$D^{(0)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \end{array} \Rightarrow I/P$$

shortest distance.

.OR operation

$$D^{(1)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

→ fix diagonal element

→ add row and column and replace ∞ with smaller no of addition

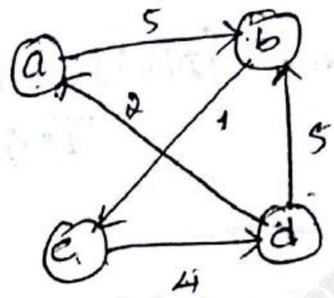
$$D^{(2)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & 9 & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

$$D^{(3)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 9 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$$

$D^{(4)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	7	0	1
d	6	16	9	0

②



$= D^0 =$

	a	b	c	d
a	0	5	∞	∞
b	∞	0	1	∞
c	∞	∞	0	4
d	2	5	∞	0

$D^{(1)} =$

	a	b	c	d
a	0	5	∞	∞
b	∞	0	1	∞
c	∞	∞	0	4
d	2	5	∞	0

$D^{(2)} =$

	a	b	c	d
a	0	5	6	∞
b	∞	0	1	∞
c	∞	∞	0	4
d	2	5	6	0

$D^{(3)} =$

	a	b	c	d
a	0	5	6	10
b	∞	0	1	5
c	∞	∞	0	4
d	2	5	6	0

$D^{(4)} =$

	a	b	c	d
a	0	5	6	10
b	7	0	1	5
c	6	9	0	4
d	2	5	6	0

P. T. O

Algorithm Floyd (w[1....n]).

// Input : weight matrix w[1....n]

// output : D - shortest path.

D ← w.

for (k ← 1 to n do //vertex

for i ← 1 to n do //row

for j ← 1 to n do //column

$D[i,j] \leftarrow \min\{D[i,j], D[i,k] + D[k,j]\}$

end for

end for

end for

return D

Time efficiency

$$T(n) = \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} 1$$

$$\boxed{T(n) \in \Theta(n^3)}$$

0/1 Knapsack ~~using~~ using Dynamic Programming

n = no of items, m = capacity of knapsack.

$P[P_1, P_2, P_3 \ldots P_n]$ $w[w_1, w_2, \ldots w_n]$

Value table.

j →

V	0	1	2	...	m
0	0	0	0 0 ...		0
1	0				
2	0				
3	0				
⋮	⋮				
n	⋮				

i ↓

~~i=0 & j=0~~

$$v[i,j] = \begin{cases} 0 & i=0 \;\&\; j=0 \\ v[i-1,j] \text{ (previous)} & j< w_i \\ \max\{v[i-1,j],\; p[i]+v[i-1,\; j-w[i]]\} & j \geq w_i \end{cases}$$

doubt

28/4/17

Ex

$n=4 \quad m=5$

$p[12, 10, 20, 15]$, $w[2, 1, 3, 2]$

→ capacity

v	0	1	2	3	4	5
i						
0	0	0	0	0	0	0
w=2, p=12 1	0	0				
w=1, p=10 2	0					
w=3, p=20 3	0					
w=2, p=15 4	0					

j →

i ↓	v	0	1	2	3	4	5
	0	$\boxed{0}$	0	$\boxed{0}$	0	0	0
w=2, p=12	1	0	0	max[0, 12+0] $\boxed{12}$	max(0, 12+0) $\boxed{12}$	max(0, 12+0) 12	max (0, 12+0) 12
w=1, p=10	2	0	max(0, 10+0) 10	max(12, 10+0) 12	max(12, 10+12) $\boxed{22}$	max(12, 10+12) 22	max (12, 10+12) 22
w=3, p=20	3	0	10	12	max(22, 20+0) $\boxed{22}$	max(22, 20+10) 30	max (12, 20+12) $\boxed{32}$
w=2, p=15	4	0	10	max(12, 15+0) 15	max(22, 15+10) 25	max(30, 15+12) 30	max (32, 15+22) $\boxed{37}$

optimal solution is 37.

$$\begin{bmatrix} \cancel{0} & \cancel{0} & 0 & \cancel{0} \end{bmatrix} = 1, 2, 4$$

1 1 0 1 = 37

Example 2

$n = 5$, $m = 7$

$P[25, 20, 15, 40, 50]$, $w[3, 2, 1, 4, 3]$.

j →

V	0	1	2	3	4	5	6	7
0	$\boxed{0}$	$\boxed{0}$	0	$\boxed{0}$	0	0	0	0
1 ($w=3$, $p=25$)	0	6	0	max(0, 25+0) $\boxed{25}$	max(0, 25+0) $\boxed{25}$	max(0, 25+0) 25	max(0, 25+0) 25	max(0, 25… 25
2 ($w=2$, $p=20$)	0	0	(0, 20+0) 20	(25, 20+0) $\boxed{25}$	(25, 20+0) $\boxed{25}$	(25, 20+25) 45	(25, 20+25) 45	(25, 20+25) 45
3 ($w=1$, $p=15$)	0	(0, 15+0) 15	(20, 15+0) 20 ~~15~~	(25, 15+20) 35	(25, 15+25) $\boxed{40}$	(45, 15+25) 45	(45, 15+45) 60	60
4 ($w=4$, $p=40$)	0	15	20	35	(40, 40+0) $\boxed{40}$	(45, 40+15) 55	(60, 40+20) 60	(60, 40+35) $\boxed{75}$
5 ($w=3$, $p=50$)	0	15	20	(35, 50+0) 50	(40, 50+15) 65	(55, 50+20) 70	(60, 50+35) ~~90~~ 85	$\boxed{90}$

i ↓

Optimal $sol^n = 90$

[0 0 0 0 0]

02/5/17

Algorithm: DP_Knapsack (n, m, P[ ], w[ ]).

//input : no. of items – n , knapsack capacity = m

Prices of all items – P[1...n], weight of all items – w[1...n]

//output : optimal feasible solution v[n, m]

for i ← 0 to n do

  for j ← 0 to m do

    if (i = 0 or j = 0) then

      v[i, j] ← 0

    else if (j < w[i]) then

      v[i, j] ← v[i−1, j]

    else

$v[i,j] \leftarrow max(v[i-1,j],\ P[i] + v[i-1, j - w[i]])$

end if

end for

end for.

return $v[n,m]$

Time efficiency:

$T(n) \in \Theta(n,m)$

Memory Function Knapsack

$n=4$, $m=5$.

$P=\{12,10,20,15\}$ $w=\{2,1,3,2\}$.

It is an optimised DP-Knapsack problem. It solves only those set of sub-problems which yields optimal selection $v[n]$. And rest of the other subproblems are not been solved.

Algorithm MF_Knapsack (i,j)

//input : 2 non-negative integer i, j

where i denotes item no and j denotes capacity of knapsack

//output : optimal feasible solution ie $v[n,m]$

//note : initialize $v[][]$ with -1 for its cells except 0^{th} row and 0^{th} column

0	0	0	0	0
0	-1	-1	-1	-1
0	-1	-1	-1	-1
0	-1	-1	-1	-1

if $(v[i,j] < 0)$ then

if $(j < w_i)$ then

else

~~return (max (MF-knapsack (~~

return (max (MF_knapsack (i-1, j), P_i + MF_knapsack (i-1, j - w_i)))

end if

end if

~~Ex~~

Example $n=4$, $m=5$

$P = \{12, 10, 20, 15\}$ $w = \{2, 1, 3, 2\}$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	~~-1~~ ~~0~~	~~-1~~ 12	~~-1~~ 12	~~-1~~ 12	~~-1~~ 12
2	0	~~0~~ -1	-1 12	-1	-1	~~-1~~ 22
3	0	-1	-1	~~-1~~ 22	-1	~~-1~~ 32
4	0	-1	-1	-1	-1	~~-1~~ 37

MFK (4, 5)

~~P.T.O.~~

$MFK(4,5)$ 37

82, 1 37, 1 22

$5<2$ ✗ $\max(MFK(3,5), 15+MFK(3,3))$ 32

$5<3$ ✗ $\max\{MFK(2,5), 20+MFK(2,2)\}$ 22, 20+12

$5<1$ ✗ $\max\{MFK(1,5), 10+MFK(1,4)\}$ 12, 22

$4<2$ $\max\{MFK(0,4), 12+MFK(0,1)\}$ 0, 12

$5<2$ ✗ $\max\{MFK(0,5), 12+MFK(0,3)\}$ 0, 12+0

$2<1$ ✗ $\max(MFK(1,2), 10+MFK(1,1))$ 12

$2<2$ ✗ $\max(MFK(0,2), 12+MFK(0,0))$ 0, 12, 0

22

$2<3$ ✗ $\max\{MFK(2,3), 20+MFK(2,0)\}$ 22

$3<1$ ✗ $\max\{MFK(1,3), 10+MFK(1,2)\}$ 12, 10+12, 22

3/5/17

solve the given knapsack problem using Dynamic programming technique where

$n=4$, $m=100$ = capacity

$P(40, 42, 25, 12]$ $w[40, 70, 50, 30]$

65

$i = 4, \; j = 100$

if ($j < w_i$)

$v[i][j] \leftarrow MFK(i-1, j)$

else

$v[i][j] \leftarrow \max(MFK(i-1, j), \; p_i + MFK(i-1, j-w_i))$

$P(40, 42, 25, 12)$ $\quad w[\overset{1}{40}, \overset{2}{70}, \overset{3}{50}, \overset{4}{30}]$

j

MFK(4,100) ~~65~~ 65

100 < ~~30~~ ✗

65 / 12 + ~~65~~ 42

$\max\{MFK(3,100), \; 12 + MFK(3,70)\}$.

100 < 50. ✗

65

42 / 25+40 = 65

$\max\{MFK(2,100), \; 25 + MFK(2,50)\}$

100 < 70 ✗

42

40 / 42+0

$\max\{MFK(1,100), \; 42 + MFK(1,30)\}$

100 < 40 ✗

40

$\max\{MFK(0,100), \; 40 + MFK(0,60)\}$

ϕ $\quad \phi$

MFK(4,100)

MFK(3,100) = ~~65~~ 65

MFK(3,70) = ~~65~~ 42

MFK(2,100) = 42

MFK(2,50) = 40

MFK(1,100) = 40

MFK(1,30) = 0.

MFK(1,50) = 40

MFK(2,70) = 42 → MFK(1,30)

MFK(2,45) = 40

30 < 40 $\quad$ MFK(0,30) → 0

0.

→ MFK(2,50).

40

50 < 70 $\quad$ MFK(1,50).

40

40

50 < 40 ✗ $\quad \max\{MFK(0,50), \; 40 + MFK(0,10)\}$

0 $\quad$ 40 + 0

→ MFK(3,70) ~~65~~ 42

70 < 50 ✗

42 / 25 + ~~00~~ 0

$\max\{MFK(2,70), \; 25 + MFK(2,20)\}$

42

70 < 70 ✗

40 / 42 + 0

$\max\{(1,70), \; 42 + MFK(1,0)\}$

0

MFK(1,70) = 40

MFK(1,0) = 0

→ MFK(1,70) ← 40

70<40 ✗   MAX{MFK(0,70), 40 + (MFK(0,30)}

40 + 0

→ MFK(1,28) ← 0

~~28<40 MFK(0,28)~~

0

→ ~~MFK(2,45)~~ ← 40

45<70 MFK(1,45) ← 40

~~45<40 MAX{MFK(0,45), 40+MFK(0,5)}~~

0   40+0   40   0

→ MFK(2,20)

20<70 MFK(1,20) ← 0

20<40 MFK(0,20)

0

V	0	20	30	50	70	100
0	0	0	0	0	0	0
1	0	0	0	40	40	40
2	0	0	-1	40	42	42
3	0	-1	-1	-1	42	65
4	0	-1	-1	-1	-1	65

ex 3

$i = 3, j = 20.$

$n = 3, m = 20;$

$P[25, 24, 15]$ $w[18, 10, 15]$

$MFK(3, 20) \leftarrow 25$

$20 < 15$ ✗ $\quad MAX\{MFK(2, 20) \leftarrow 25, \; 15 + MFK(2, 5) \leftarrow 0\}$

$20 < 10$ ✗ $\quad MAX\{MFK(1, 20) \leftarrow 25, \; 24 + MFK(1, 10) \leftarrow 0\}$

$20 < 18$ ✗ $\quad MAX\{MFK(0, 20), \; 25 + MFK(0, \ldots)\}$

0 0

$MFK(3, 20) = 25$
$MFK(1, 20) = 25$
$MFK(1, 10) = 0.$
$MFK(2, 20) = 25$
$MFK(2, 5) = 0$
$MFK(1, 5) = 0$

$\rightarrow MFK(1, 10) \leftarrow 0$

$10 < 18$ $\quad MFK(0, 10)$

0

$\rightarrow MFK(2, 5) \leftarrow 0$

$5 < 10$ $\quad MFK(1, 5) \leftarrow 0$

$5 < 18$ $\quad 0, MFK(0, 5)$

0

	0	2	5	10	20
0	0	0	0	0	0
1	0	−1	0	0	25
2	0	−1	0	−1	25
3	0	−1	−1	−1	25

9/5/17

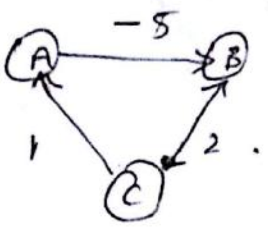
Bellman Ford Algorithm

→ single source shortest path.

→ used in routing ~~Proble~~ protocol - Distance vector

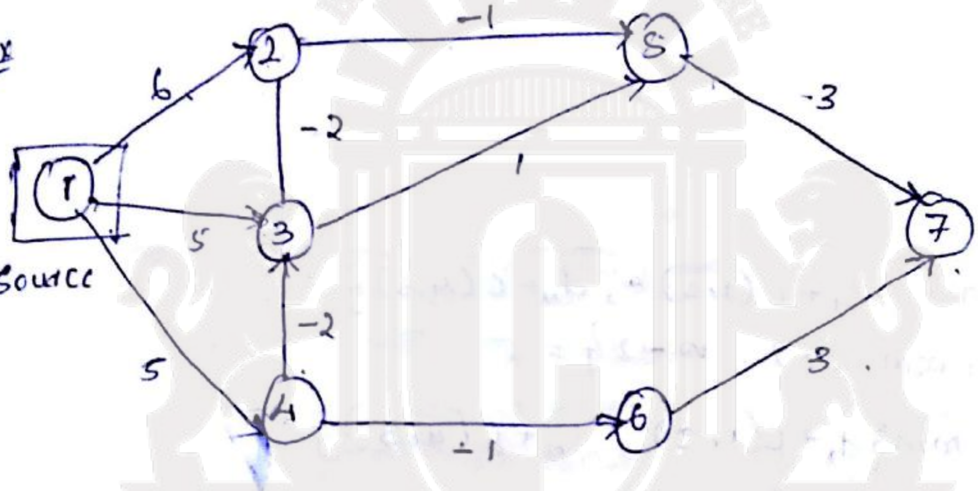
→ works on the negative weights.

[cannot work on −ve cycle].



Estimate ~~shortest~~ shortest path based on neighbour shortest path from source.

Ex



Iterations	d_1	d_2	d_3	d_4	d_5	d_6	d_7
Initial	0	∞	∞	∞	∞	∞	∞
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3
negative cycle check	0	1	3	5	0	4	3

Same value
no negative sign

(1) $d_2 = \min\{d_1 + c(1,2),\ d_3 + c(3,2)\}$

$= \min\{0+6,\ \infty - 2\} = 6.$

(2) $d_2 = \min\{d_1 + c(1,2),\ d_3 + c(3,2)\}.$

$\min\{0+6,\ 5+(-2)\} = 3$

(3) $d_2 = \min\{0+6,\ 3-2\} = 1$

(4) $d_2 = \min\{0+6,\ 3-2\} = 1$

(5) $d_2 = \min\{0+6, \quad = 1.$

(6) $d_2 = 1$

---

(1) $d_3 = \min\{d_1 + c(1,3),\ d_4 + c(4,3)\}$

$= \min\{0+5,\ \infty - 2\} = 5$

(2) $d_3 = \min\{d_1 + c(1,3),\ d_4 + c(4,3)\}$

$\min\{0+5,\ 5-2\} = 3$

(3) $d_3 = \min\{0+5,\ 5-2\} = 3$

(4) $d_3 = \min\{0+5,\ 5-2\} = 3$

(5) $d_3 = 3$

(6) $d_3 = 3$

$d_4 = \min\{ \cancel{d_1}\, d_1 + c(1,4) \}$

$= 0 + 5 = 5$

$d_4 = d_1 + c(1,4)$

$0 + 5 = 5$

$d_4 = 0 + 5 = 5$

$d_4 = 0 + 5 = 5$

$d_4 = 0 + 5 = 5$

$d_4 = 0 + 5 = 5.$

$d_5 = \min\{ d_2 + c(2,5),\ d_3 + c(3,5) \}$

$\infty + (-1),\ \infty + 1 \quad = \infty.$

$d_5 = \min\{ d_2 + c(2,5),\ d_3 + c(3,5) \}$

$\min\{ 6 - 1,\ 5 + 1 \} = 5$

$d_5 = \min\{ 3 - 1,\ 3 + 4 \} = 2$

$d_5 = \min\{ 1 - 1,\ 3 + 4 \} = 0$

$0 \quad 1$

$d_5 = 0$

$d_5 = 0$

$d_6 = d_4 + c(4,6) =$

$\infty + (-1) = \infty$

$d_6 = d_4 + c(4,6)$

$= 5 - 1 = 4$

$d_6 \quad 5 - 1 = 4$

$d_7 = \min\{d_5 + c(5,7),\ d_6 + c(6,7)\}.$

$\{\infty + 3,\ \infty + 3\} = \infty$

$d_7 = \min\{d_5 + c(5,7),\ d_6 + c(6,7)\}$

$\min\{\infty + 3,\ \infty + 3\} = \infty$

$d_7 = \min\{5+3,\ 4+3\} = 7$

$d_7 = \min\{2+3,\ 4+3\} = \cancel{7}\ 5$

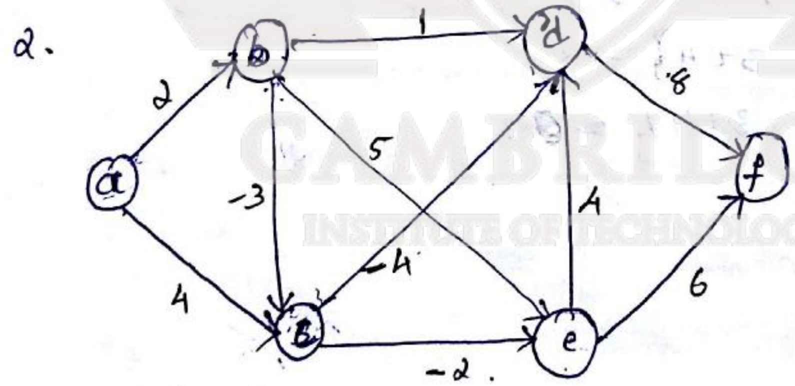
$d_7 = \min\{0+3,\ 4+3\} = 3$

$d_7 = 3$

Shortest path

1. $1 \to 1 = 0$
2. $1 \to 4 \to 3 \to 2 = -1$
3. $1 \to 4 \to 3 = 3$
4. $1 \to 4 = 5$
5. $1 \to 4 \to 3 \to 2 \to 5 = 0$
6. $1 \to 4 \to 6 = 4$
7. $1 \to 4 \to 3 \to 2 \to 5 \to 7 = 3$

2.



Shortest path

a) $a \to a = 0$
b) $a \to b = 2$
c) $a \to b \to c = -1$
d) $a \to b \to c \to d = -5$
e) $a \to b \to c \to e = -3$
f) $a \to b \to c \to e \to f = 3$

Iterations	d_a	d_b	d_c	d_d	d_e	d_f	
Initial	0	∞	∞	∞	∞	∞	
1	0	2.	4.	∞	∞.	∞	
2	0	2	−1	0	2	∞	
3	0	2	−1	−5	−3	8	
4	0	2	−1	−5	−3	3	
5	0	2.	−1	−5	−3	3	
Neg cycle check	0	2	−1	−5	−3	3	

$d_B = \cancel{min}\ d_a + c(a,b)$

$= 0 + 2 = 2.$ } It is set.

$d_c = \cancel{d}\ \min\{d_a + c(a,c),\ d_b + c(b,c)\}.$

$\{0+4,\ \infty + (-3)\} = 4.$ It is set

$d_c = \min\{0+4,\ 2-3\} = -1$

$d_d = \min\{d_B + c(b,d),\ d_c + c(c,d),\ d_e + c(e,d)\}$

$\{\infty,\ \infty,\ \infty\} = \infty.$

$d_d = \min\{2+1,\ 4-4,\ \infty\} = 0$

$d_d = \min\{3,\ -1-4,\ 2+4\} = -5$ { set

$d_e = \cancel{d}\min\{\cancel{d}_b + c(b,e),\ d_c + c(c,e)\}.$

$\{\infty,\ \infty\} = \infty$

$d_e = \min\{d_b + c(b,e),\ d_c + c(c,e)\}$

$\{2+5,\ 4-2\} = 2.$

$d_e = \min\{2+5,\ -1-2\} = -3$ set

$d_f = \min\{ d_d + c(d, f),\ d_e + c(e, f) \}$

$\qquad \infty \quad , \quad \infty \qquad = \infty$

$d_f = \min\{ \infty , \infty \} = \infty$

$d_f = \min\{ 0+8 , 2+6 \} = 8$

$d_f = \min\{ -5+8 ; -3+6 \} = 3$. <u>set</u>

Algorithm : Bellman Ford (v, cost[i, j], dist[ ], n)

//input : source vector -v, graph in cost matrix ~~cost[1..n, 1..n]~~.

-cost[1....n, 1...n] ,shortest distance dist[] ,

no of vertices -'n'

//output : shortest distance from source vertices -v ie dist[]

for i ← 1 to n do // first iteration

dist[i] ← cost[v, i] .

end for

for k ← 1 to n-2 do

for each vertex u such that u ≠ v and

u has atleast one incoming edge do

~~if~~

~~for each~~

for each {i, u} in graph do

if dist[u] > dist[i] + cost[i,u] do

dist[u] ← dist[i] + cost[i,u]

end if

end for

end for

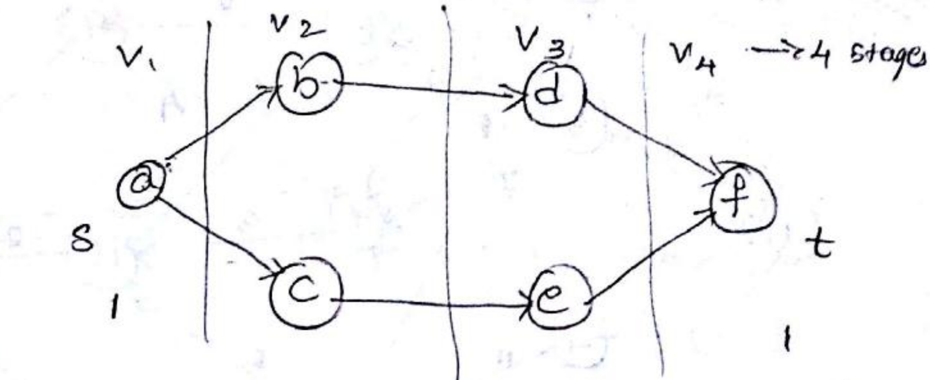
end for

return dist[]

{i, u} → is an edge coming to an vertex.

Multistage graph

~~A graph~~ :



→ A multi-stage graph G is a directed graph in which the vertices are partitioned into K-disjoint sets. as.

$G = \{V, E\}$ as V_i where $1 \le i \le x$.
& $K \geq 2$.

→ And if $\{u, v\}$ is an edge in 'E' then, $u \in V_i$ and $v \in V_{i+1}$, $|V_1| = |V_k| = 1$

→ Let $s \in V_1$ & $t \in V_k$.
source vertex — sink vertex

→ The cost of path from s to t is the sum of the cost of the edges on the path. The MSG problem is to find minimum cost path from s to t.

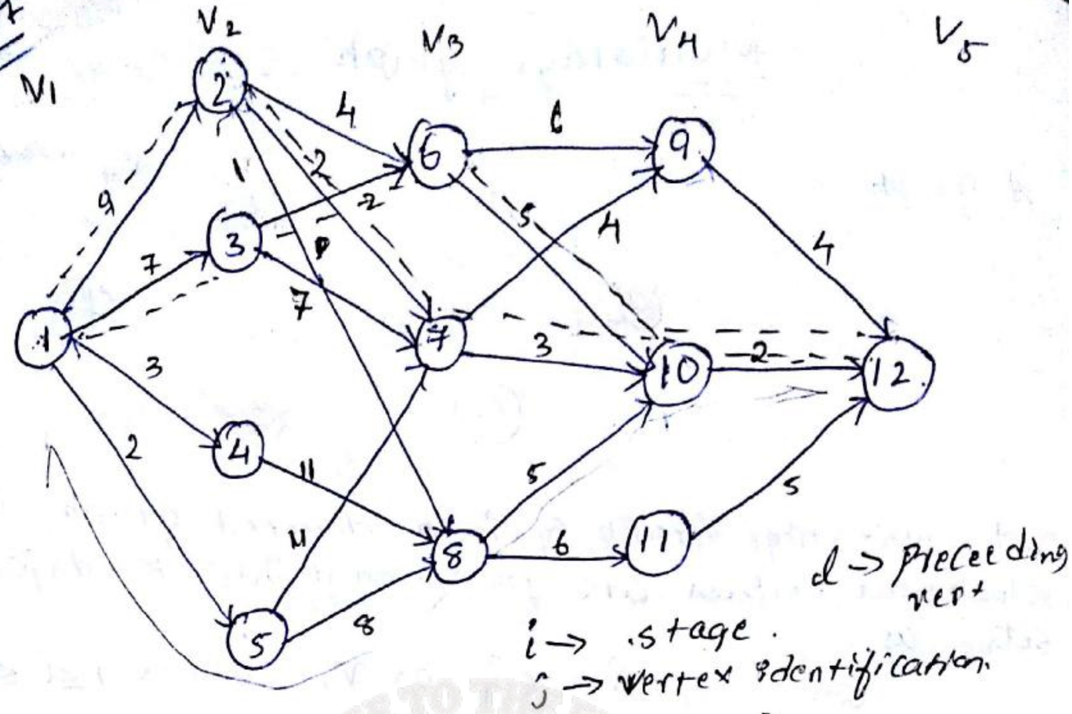
Every V_i denotes a stage

V_1 — stage 1
V_2 — stage 2
⋮
V_k — stage k.

∴ It is called as multistage graph.

Cost → ~~cons~~ shortest path
C → connectivity cost.

11/5/17



d → preceding vert

i → stage

j → vertex identification

cost(i, j) = min { c(j, d) + cost(i+1, d))

$d \in V_{i+1}$

~~(j, d)~~ e (j, d) ∈ E

Stage 4

cost(4, 9) = c(9, 12) = 4 — d[9] = 12

cost(4, 10) = c(10, 12) = 2 — d[10] = 12

cost(4, 11) = c(11, 12) = 5 — d[12] = 12.

Stage 3

cost(3, 6) = min { c(6, 9) + cost(4, 9),
c(6, 10) + cos(4, 10) }.

= min { 6+4 , 5+2 }

= 7. — d[6] = 10

cost(3, 7) = min { c(7, 9) + cost(4, 9),
c(7, 10) + cost(4, 10) } — d[7] = 10

min { 4+4 , 3+2 }

= 5

cost(3, 8) = min { c(8, 10) + cost(4, 10),
c(8, 11) + cost(4, 11) }

min { 5+2 , 6+5 } = 7 — d[8] = 10.

Stage 2

$cost(2,2) = \{ c(2,6) + cost(3,6),\ c(2,7) + cost(3,7),$
$c(2,8) + cost(3,8) \}$.

$4+7,\ 2+5,\ 1+7 \}$

$= 7$. $\Rightarrow d[2] = 7$.

$cost(2,3) = \min\{ c(3,6) + cost(3,6),$
$c(3,7) + cost(3,7) \}$.

$\min\{2+7,\ 7+5\}$

$= 9$. $d[3] = \cancel{4}.6$.

$cost(2,4) = c(4,8) + cost(3,8)$.
$11 + 7 = 18$ $d[4] = 18$.

$cost[2,5) = \min\{ c(5,7) + cost(3,7),\ c(5,8) + cost(3,8) \}$
$\min\{ 11+5,\ \underline{8+7} \}$ $d[5] = \cancel{15}\ 8$
15

Stage 1

$cost(1,1) = \min\{ c(1,2) + cost(2,2),\ c(1,3) + cost(2,3),$
$c(1,4) + cost(2,4),\ c(1,5) + cost(2,5) \}$

$d[1] = 2/3$

$= \cancel{16}$

$= \min\{ \underset{16}{\underset{2}{\underline{9+7}}},\ \underset{16}{\underset{3}{\underline{7+9}}},\ 3+18,\ 2+15 \}$

$= 16$.

Conclusion

	1	2	3	4	5	6	7	8	9	10	11	12
d[	2/3	7	6	8	8	10	10	10	12	12	12	12

Path ① $1 \rightarrow 2 \rightarrow 7 \rightarrow 10 \rightarrow 12$.

Path ② $1 \rightarrow 3 \rightarrow 6 \rightarrow 10 \rightarrow 12$.

Algorithm Fgraph (G, k, n, P[1....k])

// Input : Graph G = {V, E}, k → no of stages,
n – no of vertices, ~~min-cost~~
min-cost Path – P[1...k]

output : minimum-cost path P[1...k].

cost[n] ← 0.0 ;

for j ← n-1 downto 1 do.

minimum // Let r be a vertex such that <j, r> is edge of G and c[j, r] + cost[r] is minimum, then.

cost[j] ← c[j, r] + cost [r]

d[j] ← r

end for

P[1] ← 1 , P[k] ← n.

for j ← 2 to k-1 do .

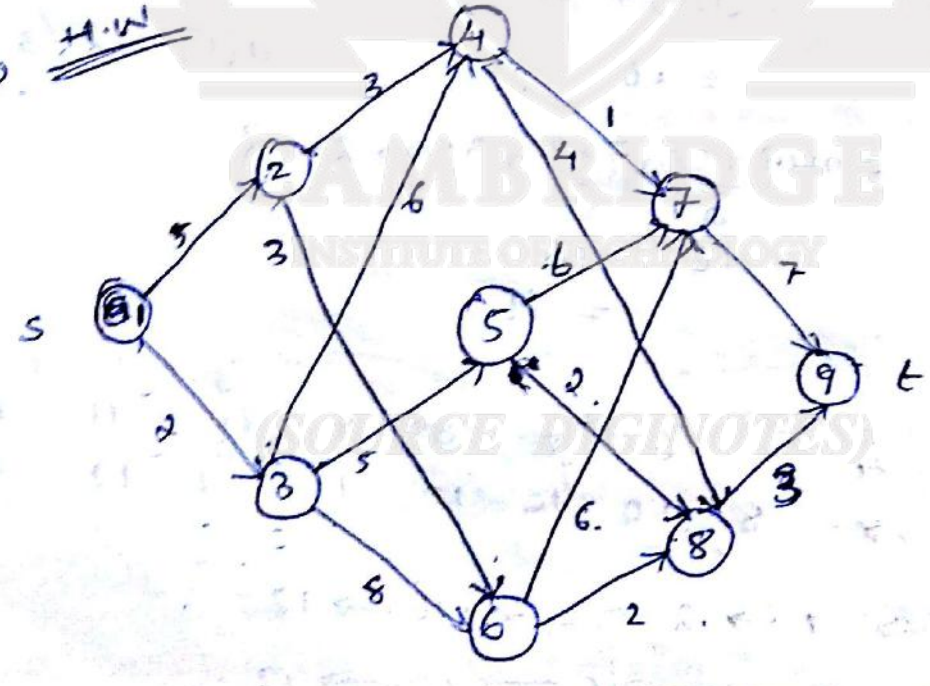
P[j] ← d[p[j-1]]

end for

return P.

H.W

Q.

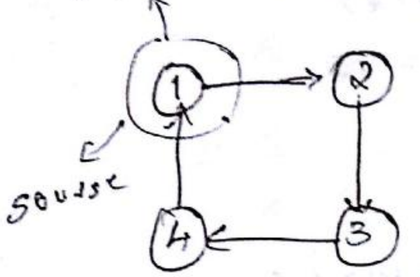


notes4free.in

Travelling Salesperson problem.

$G = \{V, E\}$

$$g(1, V-\{1\}) = \min_{2 \le k \le n} \{ c_{1k} + g(k, V-\{1,k\}) \}$$



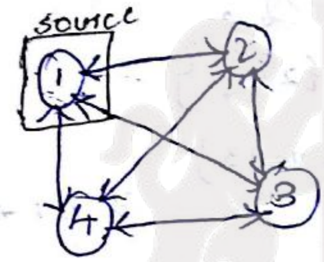
$i \rightarrow$ source

generalized eq^n

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

$$g(i, \phi) = c_{i1}$$

Ex



cost matrix

$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

Zero - intermediate ~~vertices to~~ source.

$c(2, \phi) = c_{21} = 5$

$c(3, \phi) = c_{31} = 6.$

$g(4, \phi) = c_{11} = 8.$

1 Intermediate vertex to source

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

$= 9 + 6 = 15$

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

$= 10 + 8 = 18$

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

$= 13 + 5 = 18$

$g(3,\{4\}) = C_{34} + g(4,\phi)$
$= 12 + 8 = 20$

$g(4,\{2\}) = C_{42} + g(2,\phi)$
$= 8 + 5 = 13$

$g(4,\{3\}) = C_{43} + g(3,\phi)$
$9 + 6 = 15$

<u>2 - intermediate vertex</u>

$g(2,\{3,4\}) = \min\{ C_{23} + g(3,\{4\}),$
$C_{24} + g(4,\{3\}) \}$
$= \min\{ 9+20 , \underline{10+15} \}$
$= 25$ $\quad 2 \to 4 \to 1$

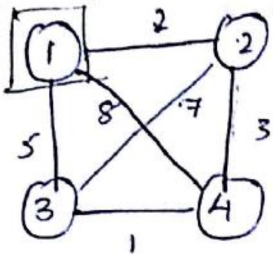
$g(3,\{2,4\}) = \min\{ C_{32} + g\{2,\{4\}\},$
$C_{34} + g\{4,\{2\}\} \}$
$\min\{13 + 18 , \underline{12 + 13}\}$ $\quad 3 \to 4 \to 1$
$= 25$

$g(4,\{2,3\}) = \min\{ C_{42} + g(2,\{3\}) , C_{43} + g(3,2)\}$
$= \min\{\underline{8+15} , 9+18\}$
23 $\quad 4 \to 2$

<u>3-intermediate Vertex</u>

$g(1,\{2,3,4\}) = \min\{ C_{12} + g\{2,\{3,4\}\}, C_{13} + g(3,\{2,4\})$
$C_{14} + g\{4,\{2,3\}\} \}$
$= \min\{ 10+25, 15+25, 20+23\}$
$= \min\{\underline{35}, 40, 43\}$
$= 1 \to 2 \to 4 \to 3 \to 1$
$10 + 10 + 9 + 6 = 35$

Ex 2 Source



It is a symmetric matrix

0 - intermediate vertex.

$c(2,\phi) = 2$.

$c(3,\phi) = 5$

$c(4,\phi) = 8$.

1 - intermediate vertex.

$g(2,\{4\}) = c_{24} + g(4,\phi)$
$= 3 + 8 = 11$.

$g(2,\{3\}) = c_{23} + g(3,\phi)$
$= 7 + 5 = 12$.

$g(3,\{2\}) = c_{32} + g(2,\phi)$.
$= 7 + 2 = 9$.

$g(3,\{4\}) = c_{34} + g(4,\phi)$
$= 1 + 8 = 9$.

$g(4,\{2\}) = c_{42} + g(2,\phi)$
$= 3 + 2 = 5$

15/5/17

Optimal Binary Search Tree

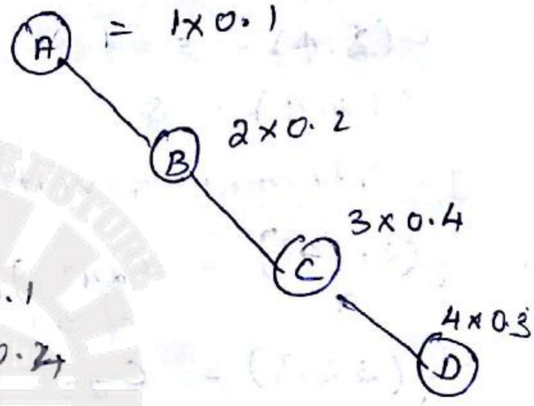
Dictionary - Unique elements (key)

$a_1, a_2 \ldots a_n$ (Ascending order).

$P_1, P_2 \ldots P_n$ (Probability search)

To build BST - minimum no of Average comparision.

	A	B	C	D
$P_i =$	0.1	0.2	0.4	0.3



A = 1×0.1 = 0.1

B = 2×0.2 = 0.4

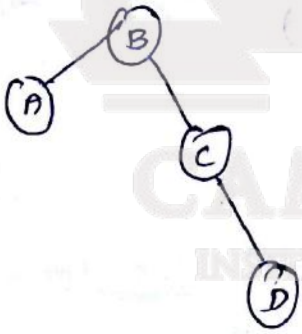
C = 3×0.4 = 1.2

D = 4×0.3 = 1.2

$\overline{2.9}$

minimumum no of Av.

(2)



A = 2×0.1 = 0.2

B = 1×0.2 = 0.2

C = 2×0.4 = 0.8

D = 3×0.3 = 0.9

$\overline{2.1}$

~~n+1 →~~

Average value table

$n+1 \rightarrow$ rows $[1 \ldots n+1]$

$n+1 \rightarrow$ column $[0 \ldots n]$

C	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0.

Root table

R	0	1	2	3	4
1		1	2	3	3
2			2.	3	3
3				3	3
4					4
5					

Initially $C[i, i-1] \leftarrow 0$.

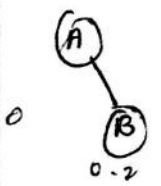
$R[i, i] \leftarrow i$

$$C[i,j] = \min_{i \le k \le j} \{ \underbrace{C(i, k-1)}_{\text{left sub tree}} + \underbrace{C(k+1, j)}_{\text{Right sub tree}} \} + \sum_{s=i}^{j} P_i$$

$$C[1,2] = \min \{ \overset{k=1}{C[1,0] + C[2,2]}, \overset{k=2}{C[1,1] + C[3,2]} \} + P_1 + P_2$$

$$= \min \{ 0 + 0.2, \underline{0.1 + 0} \} + 0.1 + 0.2$$

$$= 0.4.$$



$$C[2,3] = \min \{ \overset{k=2}{C[2,1] + C[3,3]}, \overset{k=3}{C[2,2] + C[4,3]} \} + P_2 + P_3$$

$$= \min \{ 0 + 0.4, \underline{0.2 + 0} \} + 0.2 + 0.4$$

$$= 0.8$$

$$C[3,4] = \min \{ \overset{k=3}{C[3,2] + C[4,4]}, \overset{k=4}{C[3,3] + C[5,4]} \} + P_3 + P_4$$

$$\min \{ \underline{0 + 0.3}, 0.4 + 0 \} + 0.4 + 0.3$$

$$1.0$$

Next Diagonal.

$$C[1,3] = \min \{ \overset{k=1}{C[1,0] + C[2,3]}, \overset{k=2}{C[1,1] + C[3,3]}, \underset{k=3}{C[1,2] + C[4,3]} \} + P_1 + P_2 + P_3$$

$$= \min \{ 0 + 0.8, 0.1 + 0.4, \underline{0.4 + 0} \} + 0.1 + 0.2 + 0.4$$

$$= 1.1$$

$$c[2,4] = \min\{\underset{k=2}{c[2,1]+c[3,4]},\ \underset{k=3}{c[2,2]+c[4,4]},\ \underset{k=4}{c[2,3]+c[5,4]}\} + P_2+P_3+P_4$$

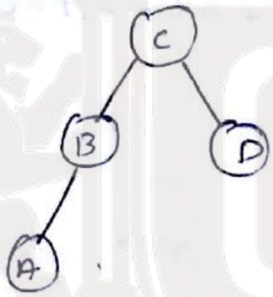
$$= \min\{0+1.0,\ \underline{0.2+0.3},\ 0.8+0\} + 0.2+0.4+0.3$$

$$= 1.4$$

$$c[1,4] = \min\{\underset{k=1}{c[1,0]+c[2,4]};\ \underset{k=2}{c[1,1]+c[3,4]},\ \underset{k=3}{c[1,2]+c[4,4]},\ \underset{k=4}{c[1,3]+c[5,4]}\} + P_1+P_2+P_3+P_4.$$

$$= \min\{0+1.4,\ 0.1+1.0,\ \underline{0.4+0.3},\ 1.1+0\} + 1.0$$

$$\underline{\underline{1.7}}$$



<u>26/5/17</u>

2.

	A	B	C	D	E
P	0.1	0.3	0.2	0.1	0.3

Average value	0	1	2	3	4	5
1	0	0.1	0.5	0.9	1.2	2.0
2		0	0.3	0.7	1.0	1.7
3			0	0.2	0.4	1.0
4				0	0.1	0.5
5					0	0.3
6						0

	0	1	2	3	4	5
1		1	2	2	2	3
2			2	2	2/3	3
3				3	3	5
4					4	5
5						5
6						

$$c[1,2] = \min\{\overset{k=1}{c[1,0]+c[2,1]},\ \overset{k=2}{c[1,1]+c[3,2]}\} + P_1 + P_2 .$$

$$= \min\{0+0.3,\ \underline{0.1+0}\} + 0.1 + 0.3 = 0.5$$

$$c[2,3] = \min\{\underline{\overset{k=2}{c[2,1]+c[3,3]}},\ \overset{k=3}{c[2,2]+c[4,3]}\} + P_2 + P_3 .$$

$$= \cancel{0.4}$$

$$\min\{\underline{0+0.2},\ 0.3+0\} + 0.3 + 0.2 .$$

$$= 0.7 .$$

$$c[3,4] = \min\{\overset{k=3}{c[3,2]+c[4,4]},\ \overset{k=4}{c[3,3]+c[5,4]}\} + P_3 + P_4 .$$

$$= \min\{\underline{0+0.1},\ 0.2+0\} + 0.2 + 0.1 .$$

$$= 0.4 .$$

$$c[4,5] = \min\{\overset{k=4}{c[4,3]+c[5,5]},\ \overset{k=5}{c[4,4]+c[6,5]}\} + P_4 + P_5$$

$$= \min\{0+0.5,\ \underline{0.1+0}\} + 0.1 + 0.3 .$$

$$0.5$$

$$c[1,3] = \min\{\overset{k=1}{c[1,0]+c[2,3]},\ \overset{k=2}{c[1,1]+c[3,3]},\ \overset{k=3}{c[1,2]+c[4,3]}\} + P_1 + P_2 + P_3 .$$

$$c[\cancel{2,4}] = \min\{c[2,1]$$

$$= \min\{\overset{k=1}{0+0.7},\ \underline{0.1+0.2},\ 0.5+0\} + 0.1 + 0.3 + 0.2$$

$$= 0.9 .$$

lly $c[2,4] = 1.0 .$

" $c[3,5] = 4.0 .$

$$c[1,4] = \min\{ \overset{k=1}{c[1,0] + c[2,4]},\ c[1,1] + c[3,4],\ \overset{k=3}{c[1,2] + c[4,4]},\ \overset{k=4}{c[1,3] + c[5,4]} \} + P_1 + P_2 + P_3 + P_4$$

$$= \min\{0 + 1.0,\ \underline{0.1 + 0.4},\ 0.5 + 0.1,\ 0.9 + 0\} + 0.7$$

$$= 1.2$$

$$c[2,5] = \min\{ c[2,1] + \overset{k=2}{c[3,5]},\ \overset{k=3}{c[2,2] + c[4,5]},\ \overset{k=4}{c[2,3] + c[6,5]},\ \overset{k=5}{c[2,4] + c[3,5]} \}$$

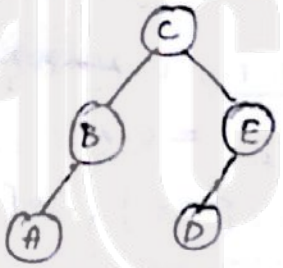
$$= \min\{0 + 1.0,\ \underline{0.3 + 0.5},\ 0.7 + 0.3,\ 1.0 + 0\} + 0.9$$

$$= 1.7.$$

$$c[1,5] = \min\{ \overset{k=1}{c[1,0] + c[2,5]},\ \overset{k=2}{c[1,1] + c[3,5]},\ \overset{k=3}{c[1,2] + c[4,5]},\ \overset{k=4}{c[1,5] + c[5,5]},\ \overset{k=5}{c[1,4] + c[6,5]} \}.$$
$$+ P_1 + P_2 + P_3 + P_4 + P_5$$

$$= \{0 + 1.7,\ 0.1 + 1.0,\ \underline{0.5 + 0.5},\ 0.9 + 0.3,\ 1.2 + 0\} + 1$$

$$= 2$$



Algorithm : Optimal_BST (P[1...n]).

// Input : An array P[1...n] of search probability for sorted ~~using~~ a list of n keys.

// output : An ~~array~~ average no of comparision in succesful search in the optimal BST and Table 'R' of subtree roots in optimal BST.

```
for i ← 1 to n do
    c[i, i-1] = 0   // diagonal
    c[i, i] ← Pi
    R[i, i] ← i
end for
c[n+1, n] ← 0
```

```
for d ← 1 to n-1 do.
  for i ← 1 to n-d do
        j ← i + d                          // starting with 2.
        minval ← ∞
        for k ← i to j do.
            if((c[i, k-1] + c[k+1, j] < minval)
                                                then
                minval ← c[i, k-1] + c[k+1, j]
                kmin ← k .

            end if
        end for
        R[i, j] ← kmin
        sum ← P[i]
        for s ← i+1 to j do.
            sum ← sum + P[i]
        end for.
        c[i, j] ← minval + sum.

      en
  end for
end for
return(R, c[1, n])
```

6/5/17

MODULE :- 5

BACK TRACKING

The principal idea is to construct solutions through one component at a time and evaluate such partially constructed candidate solⁿ.

If the partially constructed solⁿ can be developed further without violating the problem constraints, it then done through remaining legitimate option for the next component.

If there is no legitimate option for the next component, no alternative for any relative for any remaining component, then algorithm backtracks to ~~reach~~ replace the last component solution with the next available option.

It is generally represent by ~~space-~~ State - space - tree.

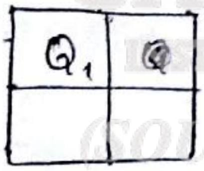
n- Queens problem

nxn - chess board / cross - board

n - queens.

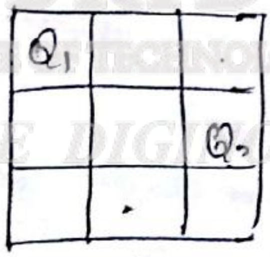
Objective :- place all queens in non-attacking position.

1x1



no - solution

2 x 2

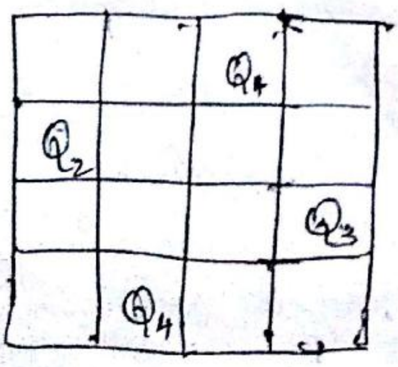


x x x

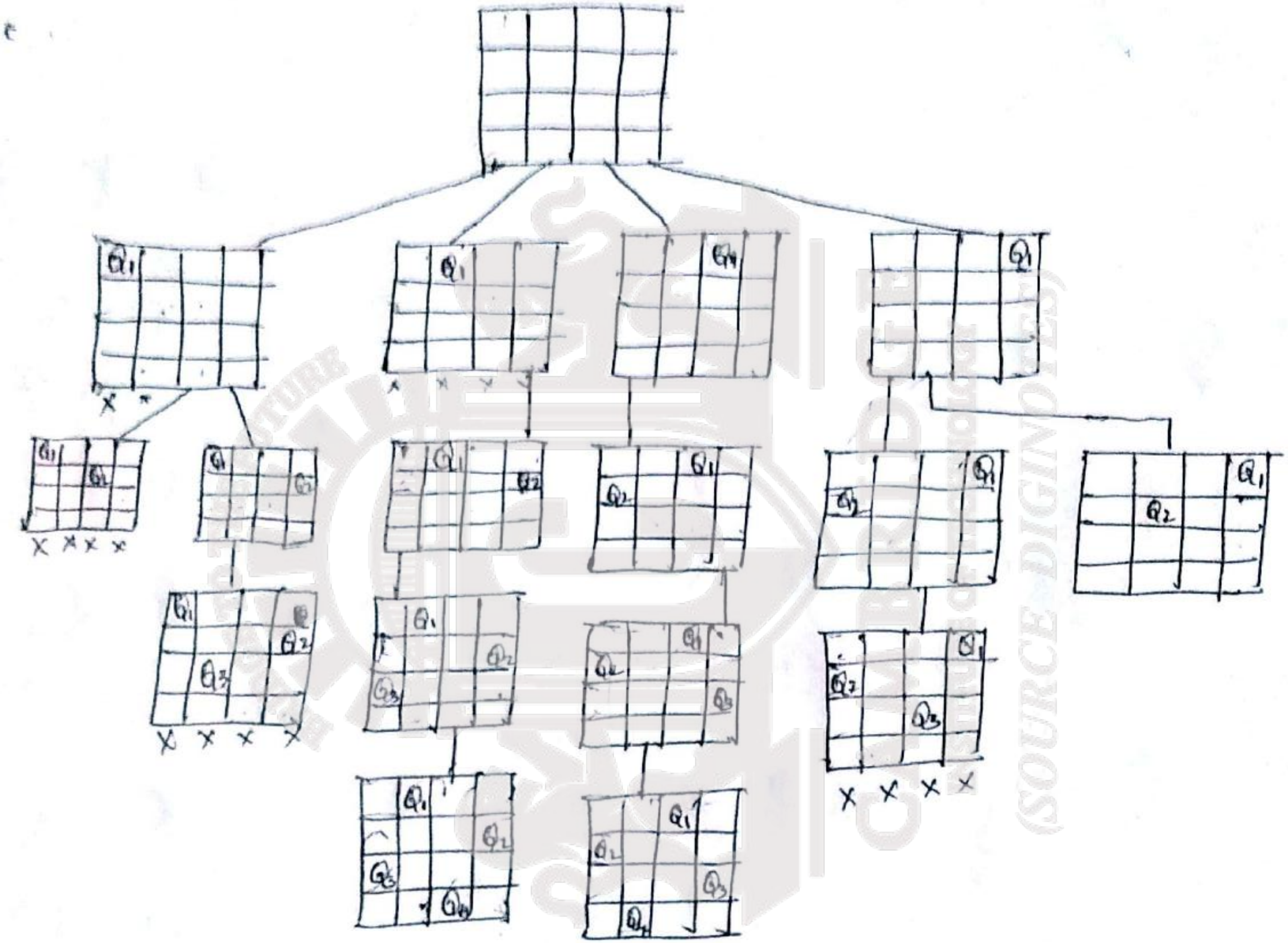
No - solution for 3x3

x x x x

~~No - solution for~~ 4x4



State space tree



Sub-set sum problem

set $S = \{s_1, s_2, \dots s_n\}$ where $s_1 < s_2 < \dots < s_n$

d = sum of sub set

Ex :- $S = \{1, 2, 3, 5, 7\}$

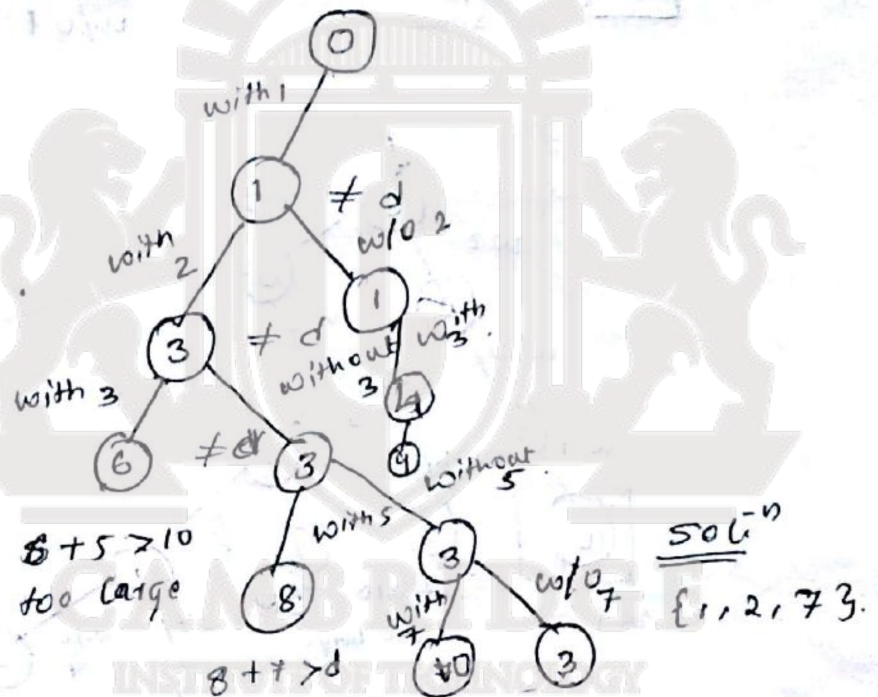
d = 10 : $\{2, 3, 5\}, \{1, 2, 7\}, \{3, 7\}$

Branch Terminating condition.

$s' + s_{i+1} > d \Rightarrow$ too large.

$s' + \sum_{j=i}^{n} s_j < d \Rightarrow$ too small

$s' \rightarrow$ Temporary sum



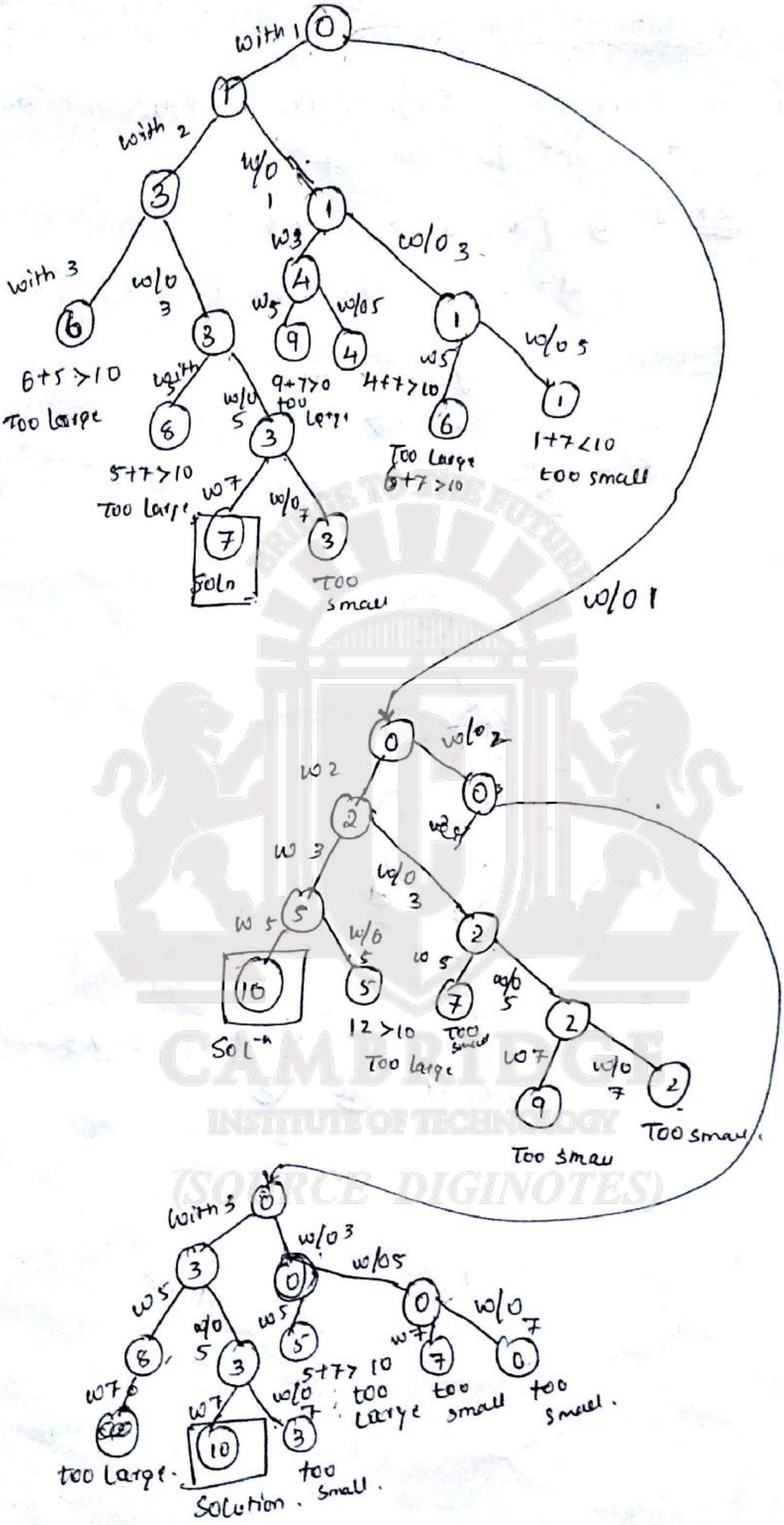
$6 + 5 > 10$ too large

$8 + 7 > d$ too large

$3 + 0 > 10$ too small.

soln $\{1, 2, 7\}$.

P.T.O.

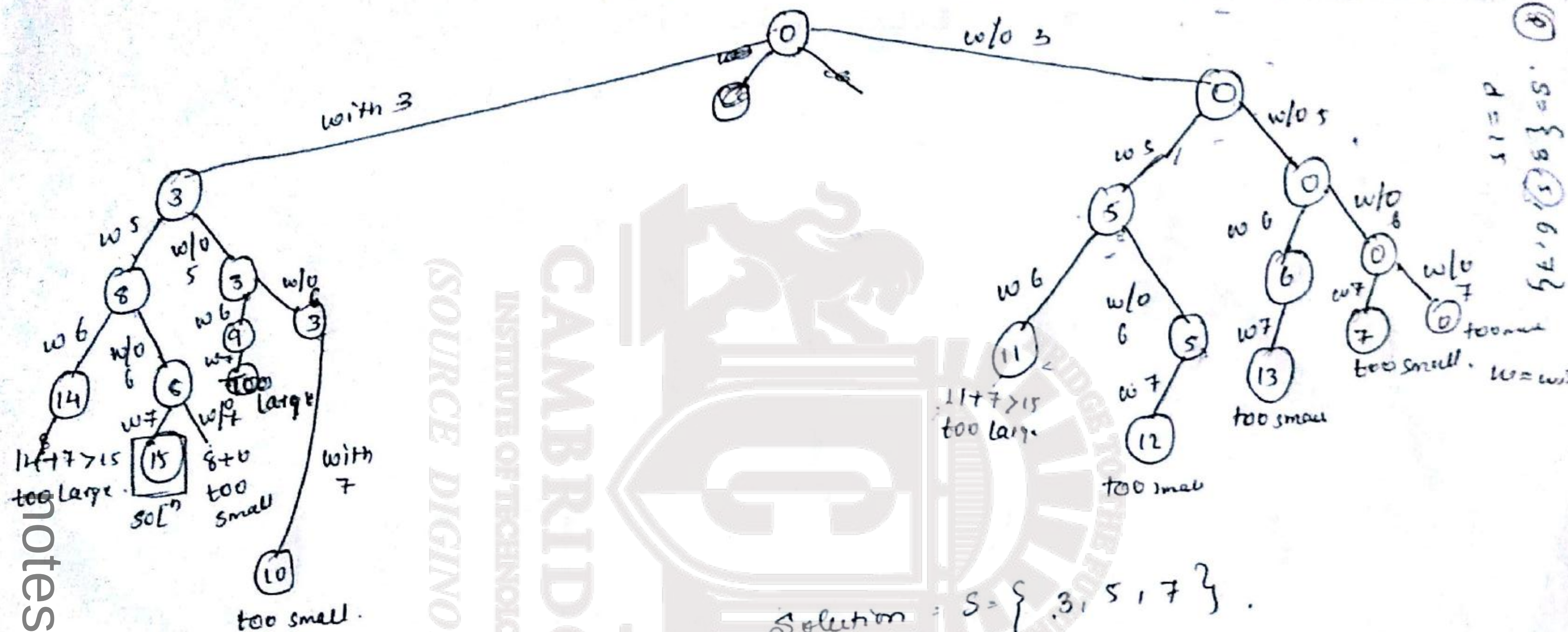


notes4free.in

Scanned by CamScanner

(8) $S = \{3, 5, 6, 7\}$

$d = 15$



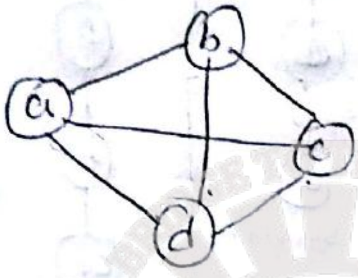
Solution: $S = \{3, 5, 7\}$.

19/6/17

Hamiltanian Circuit Problem.

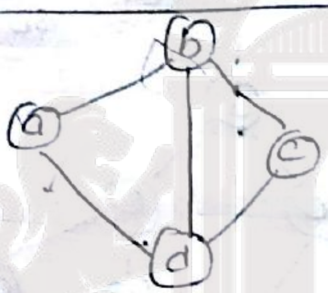
$G\{V, E\}$

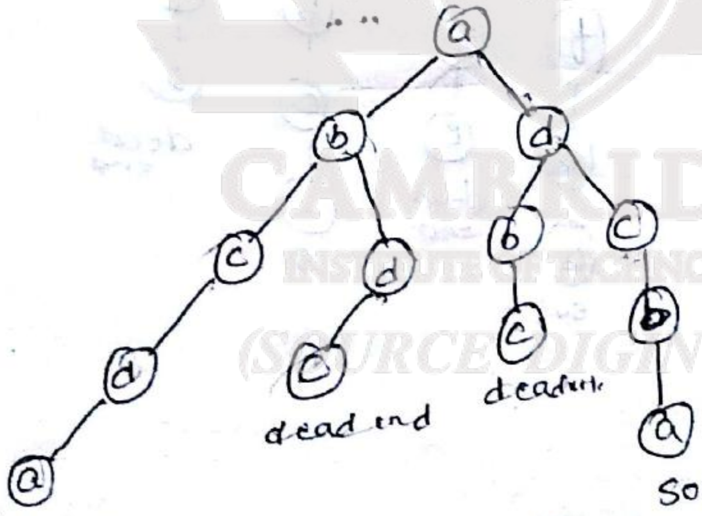
$u \to v_1 \to v_2 \cdots \to u$ → cyclic path. reaching all the vertices once and back to source vertix (u).



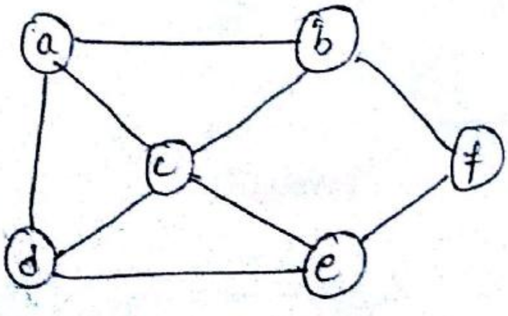
$a \to b \to c \to d \to a$.
$a \to d \to c \to b \to a$
$a \to b \to d \to c \to a$.
$a \to d \to b \to c \to a$.
$a \to c \to b \to d \to a$,
$a \to c \to d \to b \to a$.

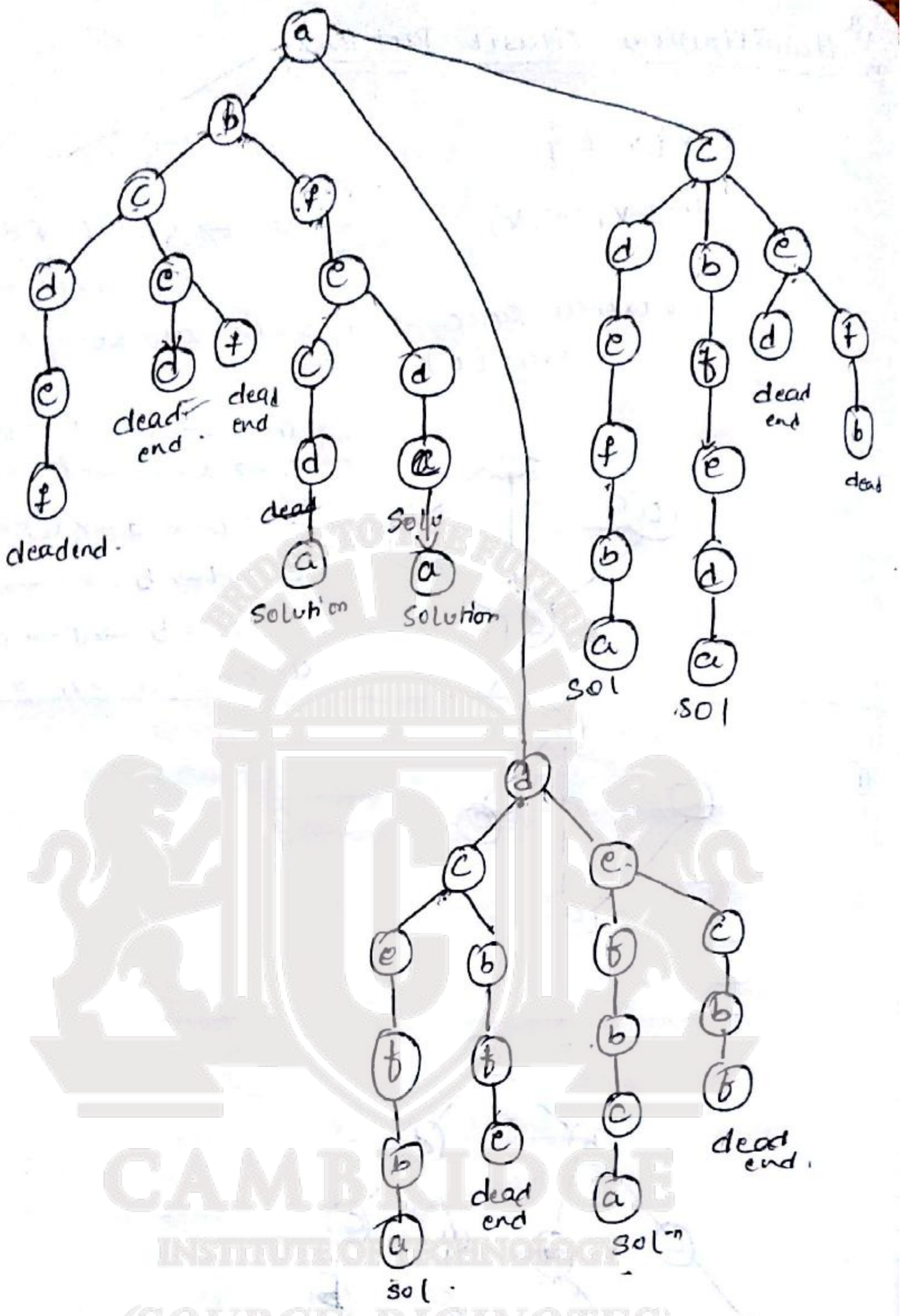
1.





3




deadend
dead end
dead end
dead
solution
Solu
solution
sol
sol
dead end
dead
sol
dead end
sol^n
dead end

Branch and Bound

→ Solution for limitation of Algorithm
→ Applied to optimization problems.
→ By calculating upper band (maximization) / Lower band (minimization)
→ Branch and Bound provides best solⁿ found so far
→ It bounds on the best value of the objective function.

→ Job Assignment problem
→ Knapsack problem.
→ TSP

① Job Assignment Problem :-

→ n - Jobs and n - people.
→ Every person can perform all the jobs with different cost.

feasible Constraint :- Assign one job to one person.

Optimal solution :- Least cost Job assignment.

Ex

	J_1	J_2	J_3	J_4
P_1	9	$\boxed{2}$	7	8
P_2	$\boxed{6}$	4	3	7
P_3	5	8	$\boxed{1}$	8
P_4	7	6	9	$\boxed{4}$

lower bound = $\min P_1 + \min P_2 + \min P_3 + \min P_4$

$\boxed{lb = 2 + 3 + 1 + 4 = 10.}$

$P_1 - J_1$: $\boxed{lb = 9 + 3 + 1 + 4}$ 17 ×

$P_1 - J_2$: $\boxed{lb = 2 + 3 + 1 + 4}$ = 10

$P_1 - J_3$: $\boxed{lb = 7 + 4 + 5 + 4}$ = 20 ×

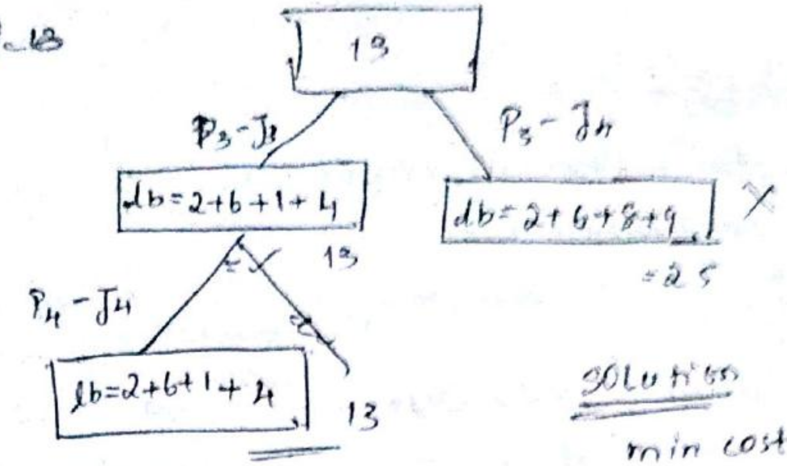
$P_1 - J_4$: $\boxed{lb = 8 + 3 + 1 + 6}$ 18 ×

$P_2 - J_1$: $\boxed{lb = 2 + 6 + 1 + 4 = 13}$ ✓

$P_2 - J_3$: $\boxed{lb = 2 + 3 + 5 + 4 = 14}$ ×

$P_2 - J_4$: $\boxed{lb = 2 + 7 + 1 + 7 = 17}$ ×

l-13



<u>Solution</u>

min cost = 13

$P_1 - J_2$

$P_2 - J_1$

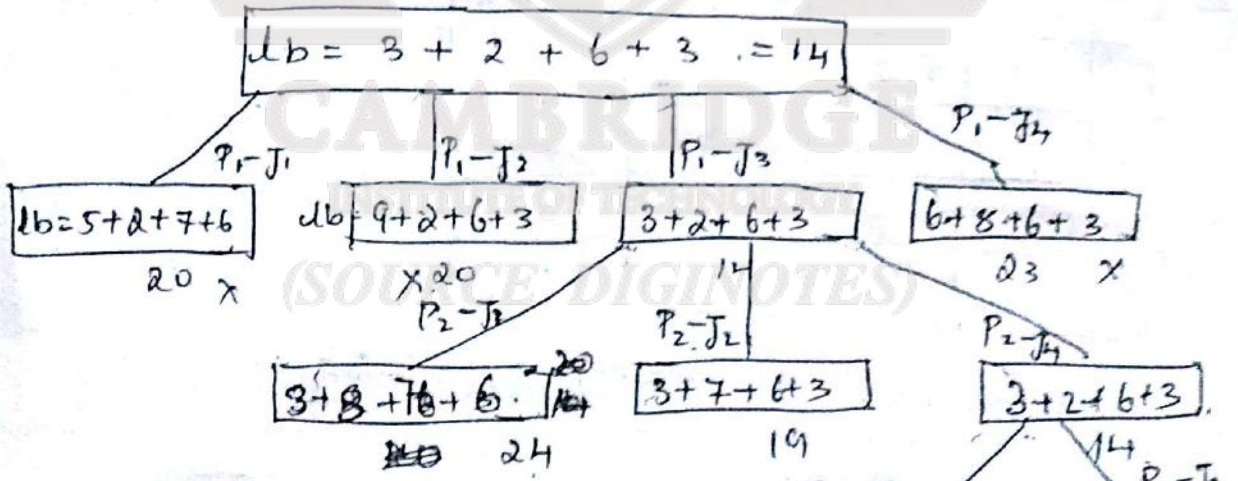
$P_3 - J_3$

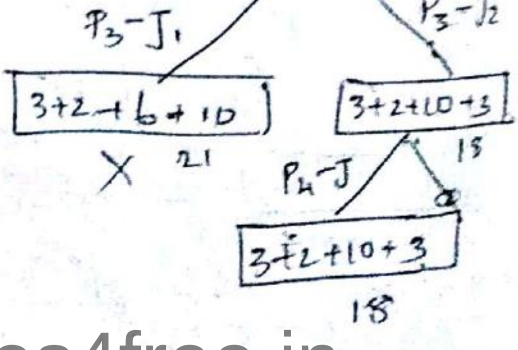
$P_4 - J_4$

<u>Example 2</u>

$$\begin{array}{c|cccc} & J_1 & J_2 & J_3 & J_4 \\ \hline P_1 & 5 & 9 & \boxed{3} & 6 \\ P_2 & 8 & 7 & 8 & \boxed{2} \\ P_3 & 6 & \boxed{10} & 12 & 7 \\ P_4 & \boxed{3} & 10 & 8 & 6 \end{array}$$

lower bound = $\min P_1 + \min P_2 + \min P_3 + \min P_4$





sol

min-cost = 18.

$P_1 - J_3$

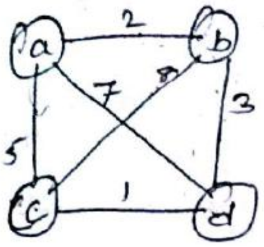
$P_2 - J_4$

$P_3 - J_2$

$P_4 - J_1$

23/05/17

Travelling salesperson problem [using branch & bound]



$$lb = \frac{\sum_{i=1}^{n} (\text{incoming edge of } v_i + \text{outgoing edge of } v_i)}{2}$$

divide by 2 because 2 vertices share the same edge.

$$lb = \frac{(2+5)+(2+3)+(1+3)+(1+5)}{2} = 11$$

a-b: $lb = \frac{(2+5)+(2+3)+(5+1)+(1+3)}{2} = 11$

a-c: $lb = \frac{(5+2)+(2+3)+(5+1)+(1+3)}{2}$ = 11

a-d: $lb = \frac{(7+2)+(2+3)+(5+1)+(7+1)}{2} = 14$

b-c: $lb = \frac{(2+5)+(2+8)+(1+8)+(3+1)}{2}$ 15

$lb = (2+5)+(2+3)+(5+1)+(3+1)/2 = 11$

$lb = \frac{(2+5)+(8+2)+(5+8)+(1+3)}{2} = 17$

$lb = \frac{(5+2)+(2+3)+(5+1)+(1+3)}{2} = 11$

d-c & c-a

$lb = \frac{(2+5)+(2+3)+(1+5)+(3+1)}{2}$ 11

Soln

$lb = (2+5)+(3+2)+(5+1)+(1+3) = 11$

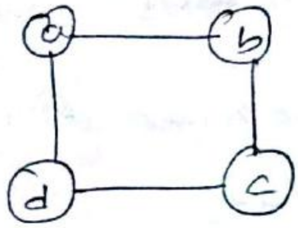
SOL.

Graph coloring problem (Back tracking method)

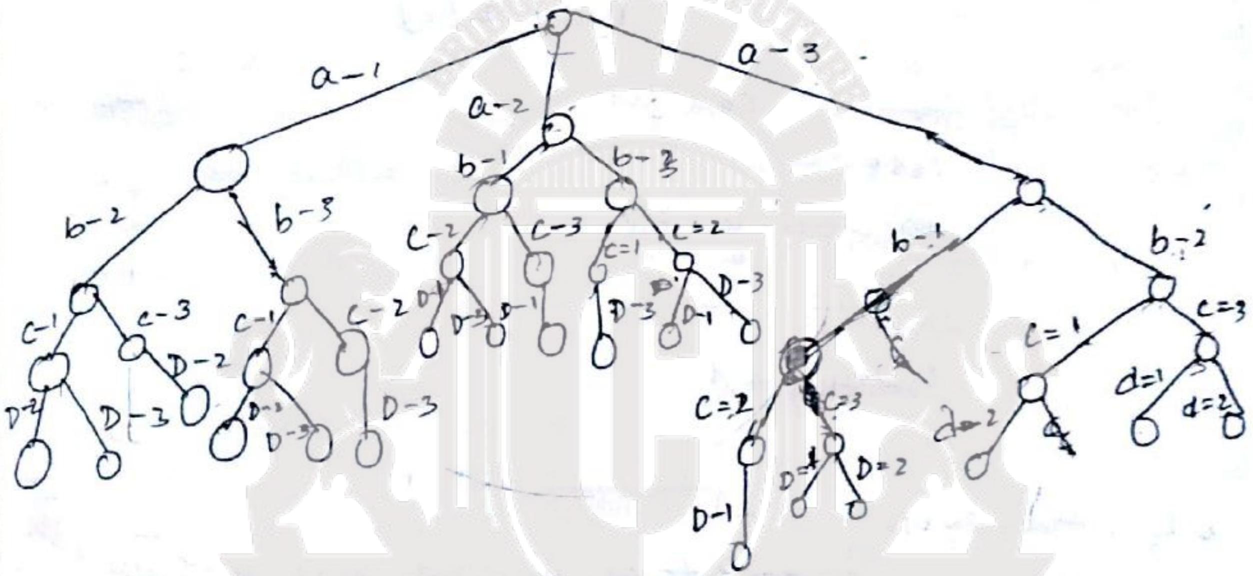
m - colours.

Graph $G = \{V, E\}$

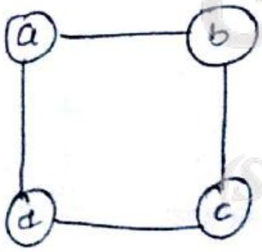
coloring vertices → no two adjacent vertices should have same color.



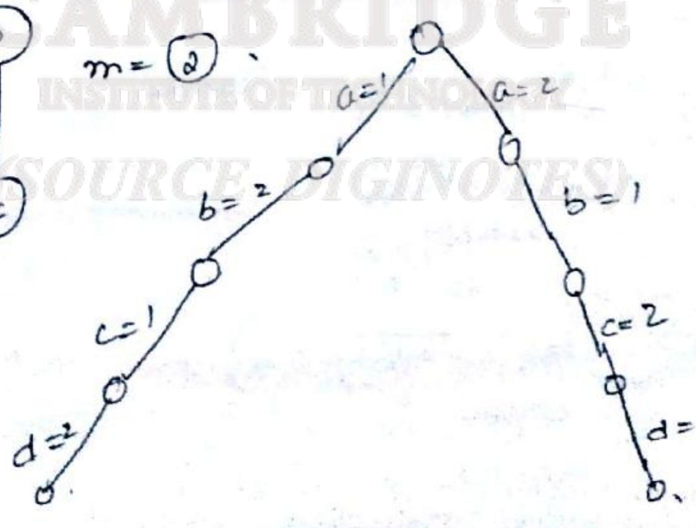
$m = 3$ ie 1, 2, 3.



Ex 2



$m = 2$.



```
Algorithm mcoloring(k)
//Input : vertex k - need to be coloured which
        varies from coloured[1...n]
//output : colour assigned to vertex k in x[1...n]
Repeat forever
     next value (k)
     if (x[k] = 0) then      //no new color is possible
          break.
     if (k = n) then   // All the nodes are coloured
          for i ← to n do
               print x[i]
          end for
     else
     mcoloring (k+1)
     end if
     endrepeat

Algorithm nextvalue (k)
// Input : vertex k need to be assigned with a color
//output : Assigned colour for k in x[k]
                                   m = number of colours.
Repeat
     x[k] ← (x[k]+1) % (m+1)
       if(x[k] = 0) then
          return
     for j ← 1 to n
          if (G[k][j] = 1} and x[k] = x[j]) then
               break.
          end if
     end for
     if j = n+1 then
          return
     end if
```

Knapsack problem [Branch-and-Bound]

$$ub = v + (m-w) * \frac{v_{i+1}}{w_{i+1}}$$ } → value to weight ratio of next item.

v → current value of knapsack

m → Capacity of knapsack.

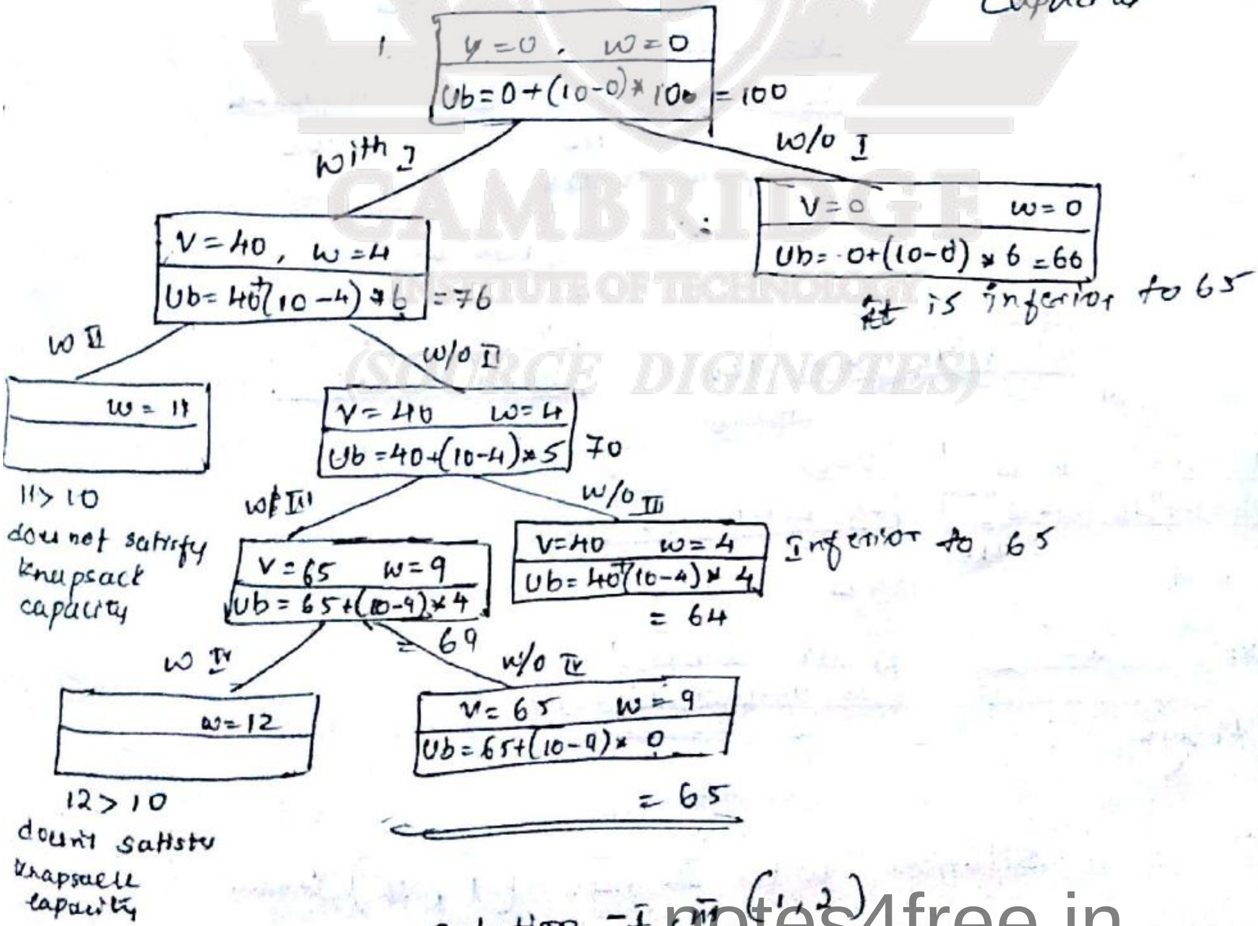
w → occupied capacity in knapsack.

1. Reorder items based value/weight ratio (Descending)
2. Calculate ub based on equation.
3. Proceed with the branch which has larger upper bound.

Example: $m = 10$

item	weight (W)	value (V)	v_i/w_i	
1	4	40	10	I
2	5	25	5	III
3	7	42	6	II
4	3	12	4	IV

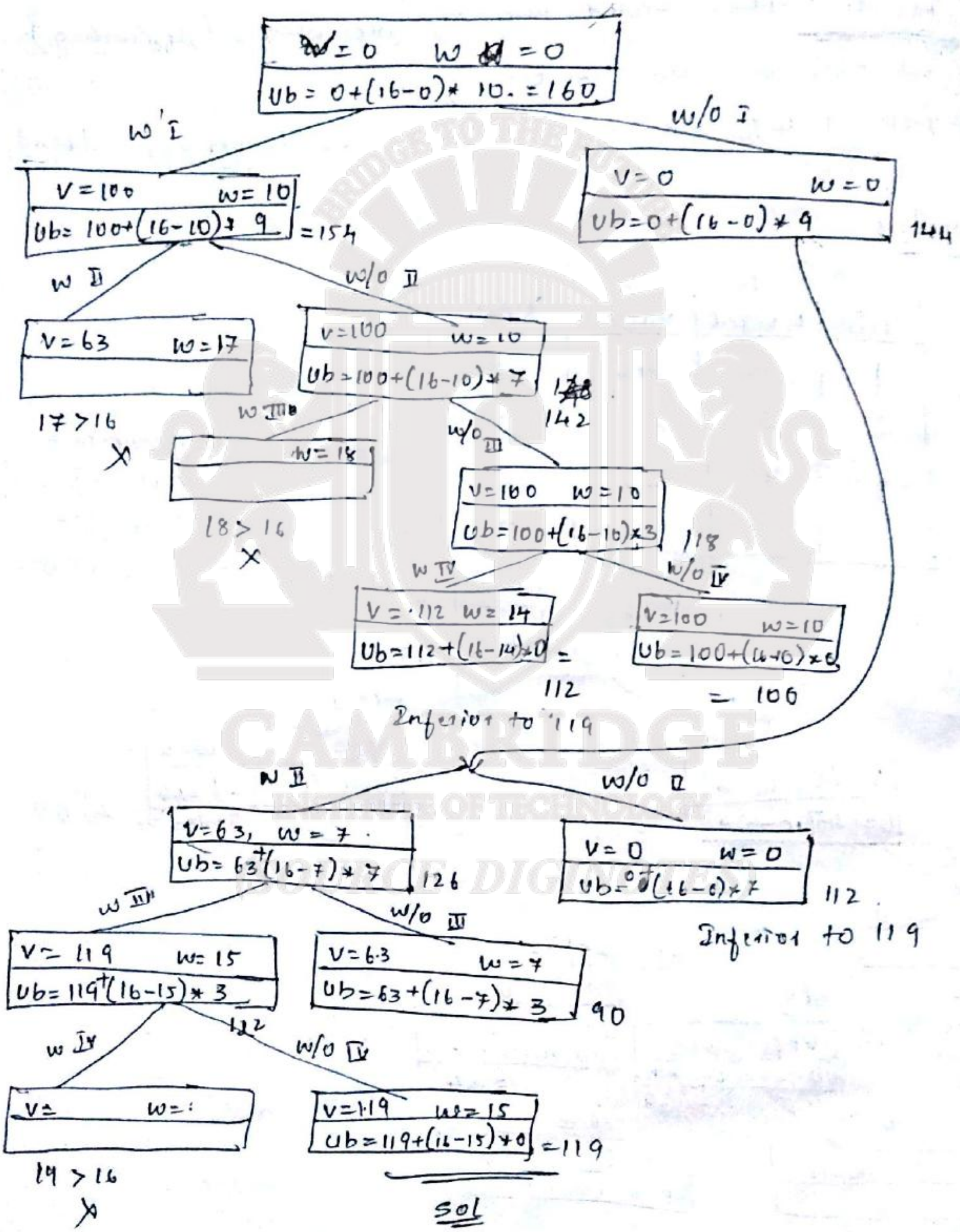
V → current value of knapsack.
Capacity → m.



Solution – I III (1,2)

Example 2. $m = 16$

item	weight	value	v_i/w_i	
1	8	56	7	III
2	10	100	10	I
3	4	12	3.	IV
4	7	63	9	II



solution with III, II (1, 4) Items.