TANISKA JHA
1107

Analysis and Design of Algorithm

→ Finite set of steps to solve a particular problem is called as 'Algorithm'

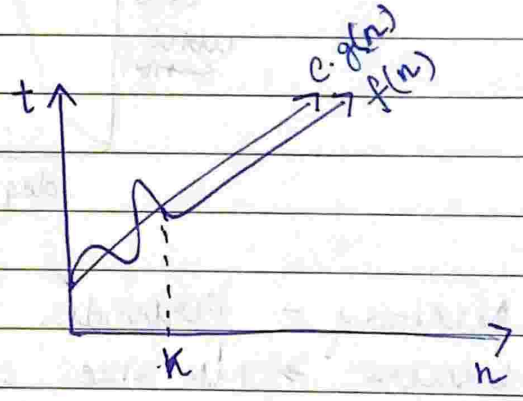
→ No. of instruction should be finite and also each instruction should take finite time

→ No ambiguity, should contain relevant signals.

→ Analysis of a process of comparing 2 algos w.r.t time, space etc.

Asymptotic Notation :-

① Big - Oh (O)



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

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

$c > 0$

$n \geq k$

$k \geq 0$

→ Worst Case

→ Upper Bound (At M

Eg :- if $f(n) = 2n^2 + n$

$2n^2 + n \leq c \cdot g(n^2)$

c cannot be 2 thus 3

$2n^2 + n \leq 3n^2$

$n \leq n^2$

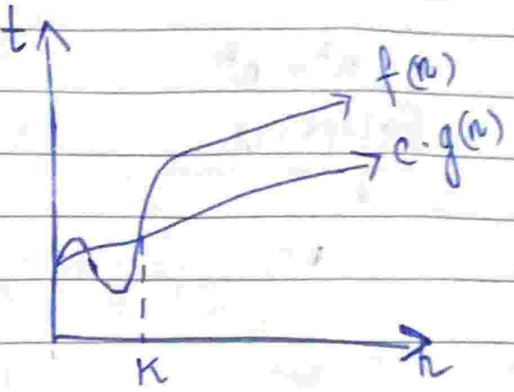
$n \geq 1$

② Big - Omega (Ω)

$f(n) = \Omega g(n)$
$f(n) \geq c \cdot g(n)$

If $f(n) = 2n^2 + n$

$2n^2 + n \geq c \cdot n^2$
$2n^2 + n \geq 2n^2$
$n \geq 0$



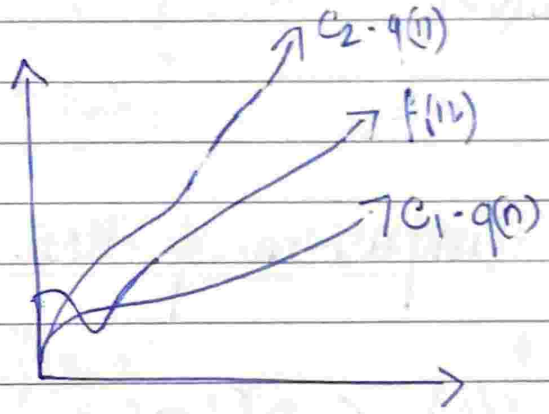
→ Best case
→ Lower bound

③ Theta (Θ)

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

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

$2n^2 \leq 2n^2 + n \leq 3n^2$



→ Average Case
→ Exact Time

→ Big(O)	$f(n) \leq c \cdot g(n)$	$a \leq b$	
→ Big(Ω)	$f(n) \geq c \cdot g(n)$	$a \geq b$	
→ Theta (Θ)	$c_1 g(n) \leq f(n) \leq c_2 g(n)$	$f(n) = g(n)$	$a = b$
→ Small (o)	$f(n) < c\, g(n)$	$a < b$	
→ Small (ω)	$f(n) > c \cdot g(n)$	$a > b$	

Properties

$f(n) = O(g(n))$; $g(n) = f(n) = O(h(n))$

	$n^2 = n^2$ Reflexive	$a<b$ $b<a$ Symmetric	Transitive
Big (O)	✓	×	✓
Big (Ω)	✓	×	✓
Theta (Θ)	✓	✓	✓
Small (o)	×	×	✓
Small (ω)	×	×	✓

Comparison of Time Complexity

$O(c) < O(\log\log n) < O(\log n) < O(n^{1/2}) < O(n)$
$< O(n\log n) < O(n^2) < O(n^3) < O(n^k) < O(2^n)$
$< O(n^n) < O(2^{2^n})$

Binary Search
Sequential Search
Quick Sort
Merge Sort
Insertion Sort
Bubble Sort
Heap Sort
Selection Sort
Height of CBT
Insertion in H
Construct Heap
Huffman
Prims
Kruskal
DFS, BFS
Dijkstra

Binary Search	$\log n$
Sequential Search	$O(n)$
Quick Sort	$O(n \log n)$
Merge Sort	$O(n \log n)$
Insertion Sort	$O(n^2)$
Bubble Sort	$O(n^2)$
Heap Sort	$O(n \log n)$
Selection Sort	$O(n^2)$
Height of CBT	$O(\log n)$
Insertion in Heap	$(\log n)$
Construct Heap	$(n \log n)$
Huffman	$(n \log n)$
Prims	$O(n^2)$, $O[(V+E) \log V]$
Kruskal	$O(E \log E)$
DFS , BFS	$O(V+E)$
Dijkstra	$O(V^2)$

Q1 Compare $f_1(n) = n^2 \log n$ $f_2(n) = n(\log n)^{10}$

keeping 10^9

$f_1(n) = (10^9)^2 \log_{10} 10^9$ $f_2(n) = 10^9 (\log_{10} 10^9)^{10}$

$= 9 \times 10^{18}$ $= 9^{10} \times 10^9$

If we compare :-

9×10^{18} & $9^{10} \times 10^9$

or 9×10^9 & 9^{10}

or 10^9 & 9^9

$10^9 > 9^9$

$\therefore f_1(n) > f_2(n)$

$\therefore f_1(n) = \Omega f_2(n)$

Q2 $f_1(n) = 2^n$, $f_2(n) = n^{3/2}$, $f_3(n) = n \log n$ $f_4(n) = n^{\log n}$

if $n = 4$

$\therefore 2^4, 2^3, 2 \times 2, 2^2$

since we got two terms same

if $n = 16$

$\therefore 2^{16}, (16)^{3/2} / 4^3 / 2^6, 16 \times 4, 16^4 / 2^{16}$

again same

if $n = 256$

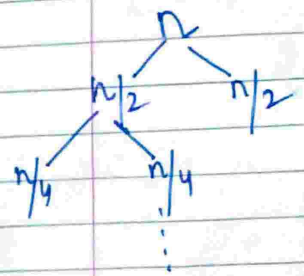
$\therefore 2^{256}, 16^3 / 2^{12}, 256 \times 8 / 2^{11}, 256^8 / 2^{64}$

$\therefore f_3 < f_2 < f_4 < f_1$

Recurrance Re

For Binary Se

Either fi
less



Solving the Rec

(1) Substitution M

Q T(n

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

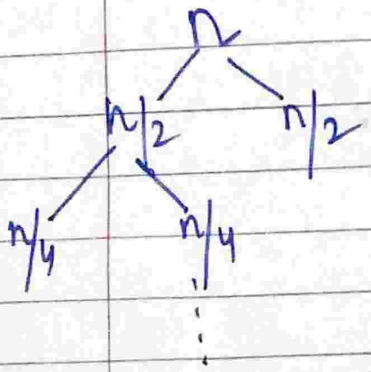
$\therefore$ T(n) =
T(n) =

T(n) =

Recurrance Relation

For Binary Search :-

Where find in b/w as we further go less ⩽ mid or greater ⩾ mid



$\therefore$ $$\boxed{T(n) = T(n/2) + c}$$

Recurrance Relation

Solving the Recurrance Relation

(1) Substitution Method

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

$$T(n) = T(n/2) + c \quad \text{— (1)}$$
$$T(n/2) = T(n/4) + c \quad \text{— (2)}$$
$$T(n/4) = T(n/8) + c \quad \text{— (3)}$$

$\therefore$ $$T(n) = T(n/4) + 2c$$
$$T(n) = T(n/8) + 3c$$
$$\vdots$$
$$T(n) = T(n/2^k) + kc \quad \text{— (4)}$$

If is given $T(1) = 1$

$\therefore$ for $T(1)$ to be 1

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

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

$$\therefore \quad n = 2^k$$

$$\log n = k \log 2$$

$$\log n = k$$

$$\therefore \quad T\left(\frac{n}{n}\right) + kc = T(n)$$

$$T(1) + \log n\, c = T(n)$$

$$1 + \log n\, c = T(n)$$

$\therefore$ all are constant thus complexity is $\log n$. i.e $O(\log_2 n)$

Q $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ n * T(n-1) & \text{if } n > 1 \end{cases}$

$T(n) = n * T(n-1)$ also $T(1) = 1$

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

$T(n-1) = (n-1) * T(n-2)$ —— (2)

$T(n-2) = (n-2) * T(n-3)$ —— (3)

$\therefore T(n) = n * T(n-1)$

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

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

⋮

taking to n-1 steps

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

$T(n) = n * (n-1) * (n-2) \; \ldots \; * 1$

$\Rightarrow n * (n-1) * (n-2) * \ldots * 3 * 2 * 1$

$\Rightarrow n * n\left(1 - \frac{1}{n}\right) * n\left(1 - \frac{2}{n}\right) \; \ldots \; n\left(\frac{3}{n}\right) * n\left(\frac{2}{n}\right) * \frac{n}{1}$

$\Rightarrow n * n * n \; \ldots$

$\Rightarrow n^n$

$\therefore O(n^n) \rightarrow$ factorial time complexity

Q $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + n & \text{otherwise} \end{cases}$

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

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

$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$ ——— ③

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

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

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

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

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

Now,

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

$T(n) = 8T\left(\frac{n}{8}\right) + 3n$ ——— ⑤

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

since $T(1) = 1$

$\therefore$ let $n = 2^k$

$\log n = k$

$\therefore T(n) = 2^k T(1) + kn$

substituting value of k ---

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

Now we will take the bigger term

i.e $n \log n$

Hence ans is $O(n \log n)$

Q $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + \log n & \text{if } n > 1 \end{cases}$

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

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

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

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

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

⋮

till $T(n-k)$

$\therefore T(n) = T(n-k) + \log(n-(k-1)) +$

$\log(n-(k-2)) \text{ ---- } \log n$

putting $n-k = 1$

$n = k$

$T(n) = T(1) + \log(1) + \log(2) \text{ --- } \log n$

$T(n) = 1 + \log(1 \times 2 \times 3 \text{ --- } n)$

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

$T(n) = 1 + \log(n^n)^0 \qquad (n! = n^n)$

$\therefore O(n \log n)$ — Ans

② Master T

$T(n) =$

Q

Soluti

Wh

(

K

② Master Method

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

$\boxed{a \geq 1}$; $\boxed{b > 1}$

Solution will come :-

$$T(n) = n^{\log_b a}\,[u(n)]$$

where $u(n)$ depends on $h(n)$

and $$\boxed{h(n) = \frac{f(n)}{n^{\log_b a}}}$$

Relation b/w $u(n)$ & $h(n)$ is :-

$h(n)$	$u(n)$
n^r, $r > 0$	$O(n^r)$
n^r, $r < 0$	$O(1)$
$(\log n)^i$, $i \geq 0$	$\frac{(\log_2 n)^{i+1}}{i+1}$

Q $T(n) = 8T(n/2) + n^2$

$a = 8 \quad b = 2 \quad f(n) = n^2$

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

$\Rightarrow T(n) = n^{\log_2 2^3} u(n)$

$\Rightarrow T(n) = n^3 u(n)$

now, $f(n) = n^2$

$\therefore h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^3} = n^{-1}$

$\therefore n^x$ where $x < 1$ satisfies

$\therefore h(n) = n^{-1}$

$\therefore u(n) = O(1)$

$T(n) = n^3 \cdot 1 = n^3$

Ans $O(n^3)$

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

$a = 1 \quad b = 2 \quad f(n) = c$

$\Rightarrow T(n) = n^{\log_2 1} u(n) = u(n)$

$h(n) =$

hence we need

$\therefore T(n)$

Q $T(n) =$

$f(n) = \log$

Thus we

$T(n) =$

→ let $n = 2^u$

$T(2^u)$

$T(2^m)$

→ let $T(2$

$$h(n) = \frac{c}{n^0} = c$$

hence we need to write it in third case form

$$\therefore \quad (\log_2 n)^0 \cdot c = h(n)$$

$$\therefore \quad u(n) = \frac{(\log_2 n)^{1+0}}{1+0} = \log n$$

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

$$O(\log n) \text{ — Ans}$$

Q $T(n) = \begin{cases} T(\sqrt{n}) + \log n & \text{if } n \geq 2 \\ O(1) & \text{else} \end{cases}$

$f(n) = \log n$ but in this we haven't got master form.
Thus we will convert

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

→ let $n = 2^m$

$$T(2^m) = T(n^{m/2}) + \log 2^m$$

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

→ let $T(2^m) = s(m)$

$g^m = 8$

$\Rightarrow S(M) = S\left(\frac{M}{2}\right) + M$

$a = 1 \quad b = 2 \quad f(n) = M$

$\Rightarrow S(M) = n^{\log_b^a} u(n) = u(n)$

$\Rightarrow h(n) = \frac{M}{n^{\log_b^a}} = M$

$\Rightarrow h(n) = (\log_2 n)^0 \cdot M$

$\Rightarrow u(n) = \frac{(\log_2 n)^{1+0}}{1+0} = \log n$

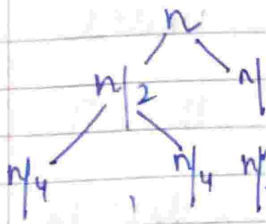
$\therefore S(M) = \log n$

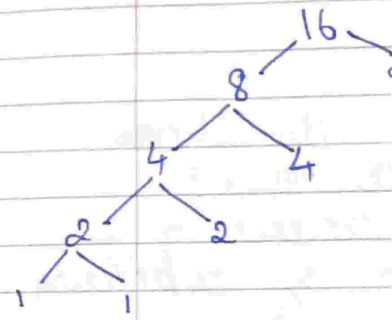
$T(2^M) = \log n$

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

$O(\log n)$ — Ans

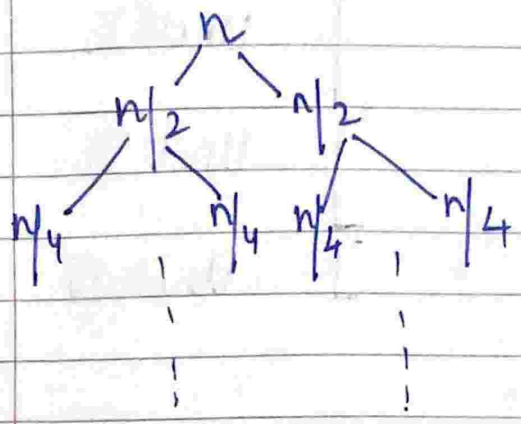
③ Recursive





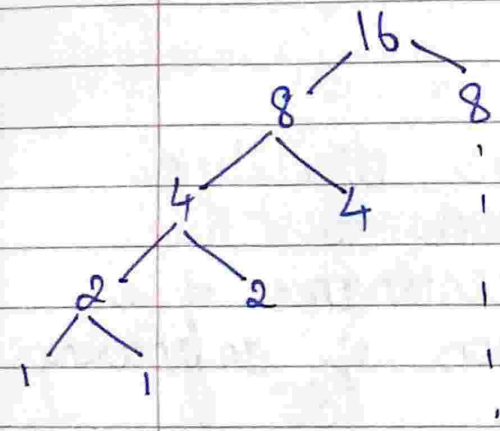
③ Recursive Tree Method

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



→ cn is the cost of each step

→ This would go until it becomes 1



→ Eg :- 16 has 4 steps or

→ we can say $\log_2 n$ steps

→ Thus total time cost $cn \cdot \log_2 n$

or $n \log n$ Ans

Analysis & Design of Algorithm

Strassens Matrix Multiplication

$$A\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times B\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = C\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$2 \times 2 \qquad 2 \times 2 \qquad 2 \times 2$

$$C_{ij} = \sum_{n=1}^{n} A_{ik} * B_{kj}$$

Writing piece of code for Matrix Multiplication

```
for ( i=0 , i<n ; i++ )
    for ( j=0 , j<n ; j++ )
        C [i, j] = 0
        for (k=0, k<n ; k++ )
            C[i,j] += A[i,k] * B[k,j];
```

analysing code the complexity is $O(n^3)$

$$C_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$C_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$C_{21} = a_{21}b_{11} + a_{22}b_{21}$$
$$C_{22} = a_{21}b_{12} + a_{22}b_{22}$$

} simple algo using loops

ADA 2

Taking th

$$A = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \end{bmatrix}$$

Dividing

C_{11} =
C_{12} =
C_{21} =
C_{22} =

Algorithm

{
if
{
C
}
else
{
}
}

f Algorithm

tic(1)

$= C \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}_{2\times 2}$

j]

trix Multiplication

-)

k++)

] * B[k, j] ;

is $O(n^3)$

simple algo using loops

ADA 2

Taking the case of a larger Matrix of 4x4

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

Dividing into four parts :-

$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$
$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$
$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$
$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

} using divide & conquer algo

Algorithm MM(A, B, n)

```
{
  if (n <= 2)
  {
    C = 4 formulas
  }
  else
  {
    mid = n/2
    MM(A11, B11, n/2) + MM(A12, B21, n/2)
    MM(A11, B12, n/2) + MM(A12, B22, n/2)
    MM(A21, B11, n/2) + MM(A22, B21, n/2)
    MM(A21, B12, n/2) (+) MM(A22, B22, n/2)
  }
}
```

(+) → Matrix addition

→ calling itself 8 time

deducing its recurrance relation

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

Using divide & conquer

complexity $= O(n^3)$

To reduce complexity Stressan's thought of an approach that by reducing no. of multiplication we might reduce complexity

Stressan's

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})\,B_{11}$$
$$R = A_{11}\,(B_{12} - B_{22})$$
$$S = [redacted]\,A_{22}\,(B_{21} - B_{11})$$
$$T = [redacted]\,B_{22}\,(A_{11} + A_{12})$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (B_{21} + B_{22})(A_{12} - A_{22})$$

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

$T(n) =$ {

$\log_2 7$

complexi

relation

	$n \leq 2$
n^2	$n > 2$

lexity = $O(n^3)$

ssan's thought of
reducing
re Night

22)

$A_{22}(B_{21} - B_{11})$
$B_{22}(A_{11} + A_{12})$
$+ B_{12})$
$- A_{22})$

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 7T(n/2) + n^2 & n > 2 \end{cases}$$

$\log_2 7 = 2.81$ $k = 2$

complexity = $O(n^{2.81})$

Dynamic Programming

Works by breaking down complex problems into simpler subproblem and then finding optimal solution to these subproblem.

Mathematical optimisation and compute methodology

Technique solves problems by breaking into smaller, overlapping subproblems. The results are then stored in array or in table using Memorization or tabularization to be reused so that they are not computed again

It has two approaches :-
1) Top - Down (Memorization)
2) Bottom - Up (tabularization)

DP solves recursion problems. A function is recursive if it calls itself during execution But not all recursion problems could be ~~used~~ solved using DP, unless solution to the subproblems overlap.

like merge sort, quick sort are not DP problem and hence are solved by Divide - and - conquer.

* 2 characteristic -
→ Principle of optimality Substructure
→ Sequence of overlapping subproblems

Drawbacks of

Uses memory
function call
all variable and
stack over
function req

Principal of Opti

A problem can
of decision.

Understanding using

$$fib(n) = \begin{cases} 0 \\ 1 \\ fib(n \end{cases}$$

```
int fib( int n
{
  if (n <= 1)
      return
  return fib(n
}
```

n complex problems
and then finding
ese subproblem.

and compute

y breaking into
oblems. The
n array are in
ore tabularization
f are not

ation)
ization)

ns. A function is
during execution
ursion problems
ut DP, unless
ns overlap.

e are not DP
solved by

uality Substructure
pping subproblems

Drawbacks of Recursion

Uses memory space less efficiently. Repeated function call creates entries entries for all variable and constants in memory stack
Stack overflow may occur if function req more memory.

Principal of Optimality

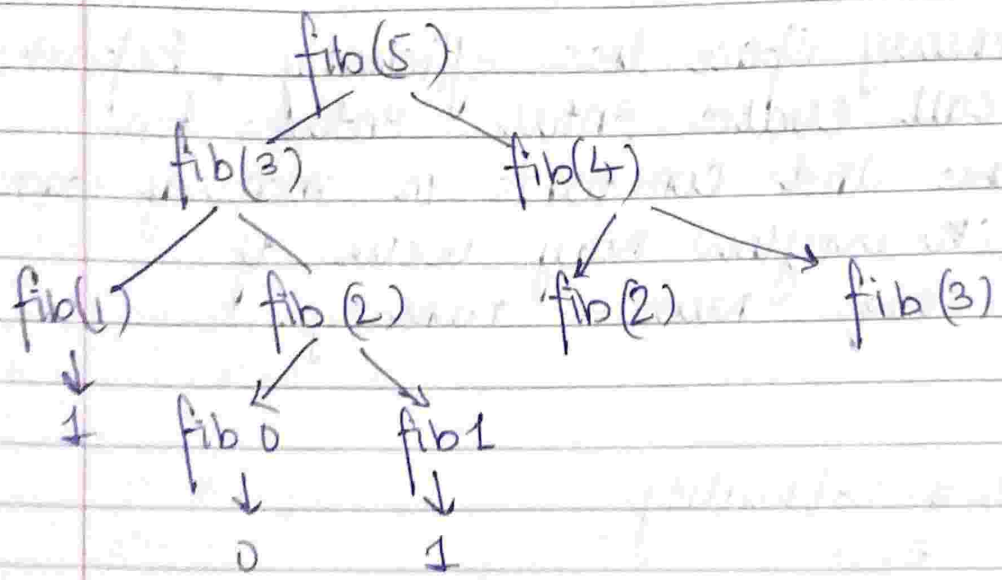
A problem can be solved by taking sequence of decision.

Understanding Memorization & Tabularisation using Example of Fibonacci Series

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-2) + fib(n-1) & \text{if } n > 1 \end{cases}$$

```
int fib ( int n)
{
  if (n ≤ 1)
      return n ;
  return fib(n-2) + fib(n-1) ;
}
```

Representation of the code :-

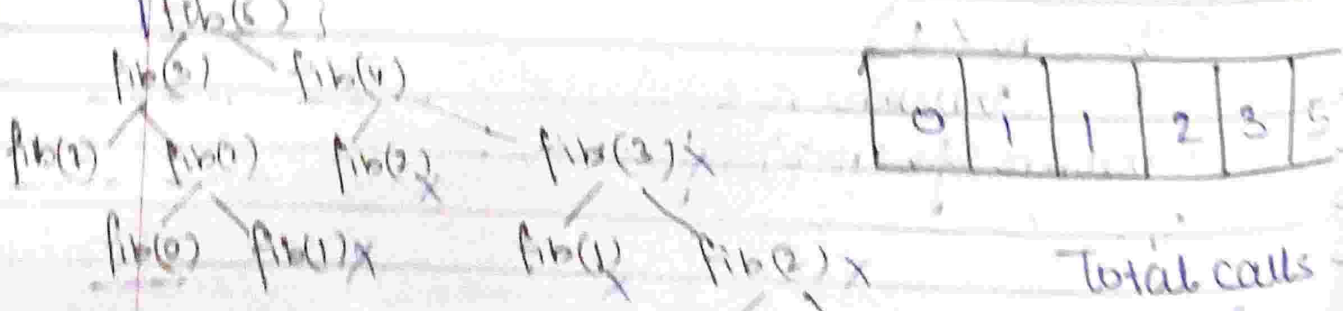


$T(n) = 2T(n-1) + 1$

$\rightarrow O(2^n)$

* Memoization

Reducing no. of repetitive calls.
We will take 1 one more global array



Total calls = 6

$fib(n) = n+1$ call

$\rightarrow O(n)$

* Tabularisation

In this we donot use recursion instead we use loops i.e iterative approach or iterative method

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-2) + fib(n-1) & \text{if } n > 1 \end{cases}$$

```
int fib (int n)
{
   if (n <= 1)
       return n;
F[0] = 0 ; F[1] = 1;
   for (int i = 2 , i <= n ; i++)
   {
      F[i] = F[i-2] + F[i-1];
   }
   return F[n];
}
```

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

(n^{th})

→ Memoisation involves solving a problem recursively and storing the result of subproblems in a data structure like array.

→ It has a top-down approach because it starts from main problem and breaks down to sub problem.

→ Tabulation is preferred when problem could be solved iteratively and there is no need for recursive calls.

→ It has bottoms up approach, starts with smallest sub-problem and builds up to main one. storing result in a table or array.

→ Divide-and-Conquer the problem is divided into subproblem which are treated independently and joined together at the end.

→ It is less example complex but time complexity is generally more than DP

→ Quick sort — $T(n-1)+n$ — $O(n^2)$

Merge sort — $2T(n/2)+n$ — $O(n\log n)$

Insertion sort — $T(n-1)+n$ — $O(n^2)$

Matrix Chain M

A X B

[ ] X [

5 X 4

Cost of Multipl

Now problem is matrices so t

$A_1 \cdot A_2$ ·

5 X 4 4 X 6

$A_1 = m[1,1] =$

$A_2 = m[2,2] =$

$A_3 = M[3,3] =$

$A_4 = M[4,4] =$

$A_1 \cdot A_2 = m[1,2]$

5 X 4 4 X 6 = 120

$M[2,3] = 48$

$M[3,4] = 84$

a problem
the result of
ata structure like

h because it
blem and breaks

when problem
ly and there
ive calls.

, starts with
l builds up to
result in a

problem is
which are
ed together at the

time complexity

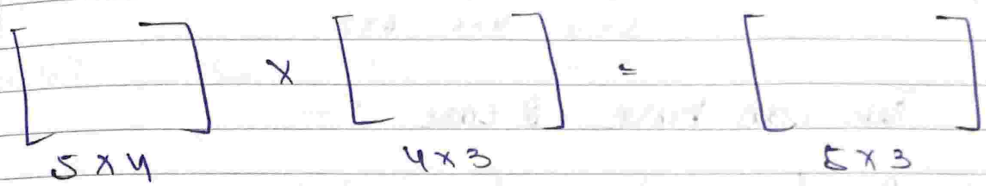
— $O(n^2)$

— $O(n \log n)$

— $O(n^2)$

Matrix Chain Multiplication

A x B = C



Cost of multiplying 2 Matrix = $5 \times 4 \times 3 = 60$

Now problem is how to multiply multiple matrices so that cost is minimum

$A_1 \cdot A_2 \cdot A_3 \cdot A_4$
$5 \times 4 \quad 4 \times 6 \quad 6 \times 2 \quad 2 \times 7$

$A_1 = m[1,1] = 0$
$A_2 = m[2,2] = 0$
$A_3 = M[3,3] = 0$
$A_4 = M[4,4] = 0$

$A_1 \cdot A_2 = m[1,2]$
$5 \times 4 \quad 4 \times 6 = 120$

$M[2,3] = 48$
$M[3,4] = 84$

M table

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

S table

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

Now selecting 3 matrix :-

① $M[1,3] = A_1 \cdot A_2 \cdot A_3$
$5\times4 \quad 4\times6 \quad 6\times2$

This can have 2 case :-

⇒ $A_1 \cdot (A_2 \cdot A_3) \qquad (A_1 \cdot A_2) \cdot A_3$

⇒ $M[1,1] + M[2,3] + \underset{5\times4\times2}{}$ $\qquad M[1,2] + M[3,3] + \underset{5\times6\times2}{}$

= 88 $\qquad$ = 180

Min is 88
and s table have A_1 i.e 1

② $M[2,4] \qquad A_2 \cdot A_3 \cdot A_4$
$4\times6 \quad 6\times2 \quad 2\times7$

$A_2 \cdot (A_3 \cdot A_4) \qquad (A_2 \cdot A_3) \cdot A_4$

$M[2,2] + M[3,4]$ $\qquad M[2,3] + M[4,4]$
$4\times6\times7 \qquad 4\times2\times7$

$0 + 84 + 168 \qquad 48 + 0 + 56$
$= 104$

Min is 104
s table will have A_3 i.e 3

Now finally selecting 4 matrix :-

$A_1 \cdot A_2 \cdot A_3 \cdot A_4$

$M[1,4] = \min \{ M[1,1] + M[2,4] + 5 \times 4 \times 7,$

$M[1,2] + M[3,4] + 5 \times 6 \times 7,$

$M[1,3] + M[4,4] + 5 \times 2 \times 7 \}$

Calculate Min out of these 3

$= 153$

5 table will have 3

$M[1,4] = A_1 (A_2 \cdot A_3 \cdot A_4)$ OR

$(A_1 \cdot A_2)(A_3 \cdot A_4)$ OR

$(A_1 \cdot A_2 \cdot A_3) \cdot A_4$

Analysis & Design of Algorithm

Unit 2 (part 2)

- Longest Common subsequence (LCS)
- Optimal Binary Search Tree Problem (OBST)
- 0-1 knapsack Problem
- Floydd Warshall Problem

* Longest Common Subsequence Problem

finding the longest subsequence

string 1 : a b c d e f g h i j

string 2 : e c d g i

⇒ lines should not cross

∴ egi and cdgi

hence longest subseq is cdgi

String 1 : a b d a c e

string 2 : b a b c e

or

a b d a c e

b a b c e

∴ bace and abce

sign of Algorithm

nce (LCS)
ree Problem (OBST)

nce Problem

ce
c d e f g h i j
d g i

d not cross
and cdgi

s cdgi

d a c e
b c e

or

b d a c e
a b c e

abce.

LCS can be solved using recursion and to avoid overlapping we can use Memoization

LCS using dynamic programming :-

Q String 1 2 : Longest
String 1 : stone

			l	o	n	g	e	s	t
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
s	1	0	0	0	0	0	0	1	1
t	2	0	0	0	0	0	0	1	2
o	3	0	0	1	1	1	1	1	2
n	4	0	0	1	2	2	2	2	2
e	5	0	0	1	2	2	3	3	3

(o) (n) (e)

Code :-

```
if ( A[i] = B[j])
    LCS[i,j] = 1 + LCS[i-1 , j-1]
else
    LCS [i,j] = max (LCS [i-1, j], LCS[i, j-1])
```

discription :-

first row/column is all 0
If letters dont match - max of right diagonal
If letters match - left diagonal element + 1

To find the substring we need to trace back diagonal relationships
∴ ans is one

* Optimal Binary Search Tree

This type of BT is useful for searching
Time taken to search is $\log n$

For eg :- If keys and frequencies are given then find cost of which binary tree is minimum

Q

	1	2	3	4	
	10	20	30	40	← key
	4	2	6	3	← freq

i \ j	0	1	2	3	4
0	0	4	8^1	20^3	26^4
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

Now we will fill this table
Considering i j

Taking $l = 0, 1, 2, 3$ and l is $j-i$

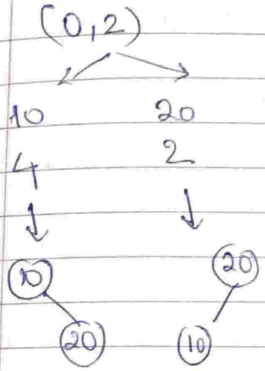
- $l = j-i = 0$
  - $0-0 = 0$ (0,0)
  - $1-1 = 0$ (1,1)
  - $2-2 = 0$ (2,2)
  - $3-3 = 0$ (3,3)
  - $4-4 = 0$ (4,4)

- $l = j-i = 1$
  - $1-0 = 1$ (0,1)
  - $2-1 = 1$ (1,2)
  - $3-2 = 1$ (2,3)
  - $4-3 = 1$ (3,4)

Now finding cost of these coordinates ignore first coordinate and take second one as the key.

(0,1) = 4 (1,2) = 2
10 20
4 2

- $l = j-i = 2$
  - $2-0 = 2$ (0,2)
  - $3-1 = 2$ (1,3)
  - $4-3 = 2$ (3,4)

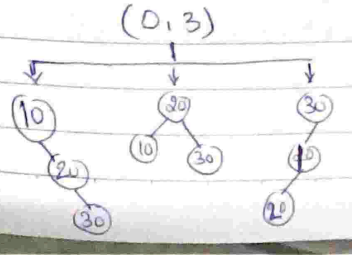


cost = 1×4 + 2×2 = 8

cost 2×1 + 2×4 = 10

Min cost = 8

- $l = j-i = 3$
  - $3-0 = 3$ (0,3)
  - $4-1 = 3$ (1,4)



ree

ve searching
log n

quencies are
cost of which
inimum

= key
= freq

ow we will fill this
table
ddressing i j

l is j-i

0,1)
(1,2)
(2,3)
(3,4)

cost of these coordinate
coordinate and take
as the key

(0,1) = 4 (1,2) = 2 (2,3) = 6 (3,4) = 3

10	20	30	40
4	2	6	3

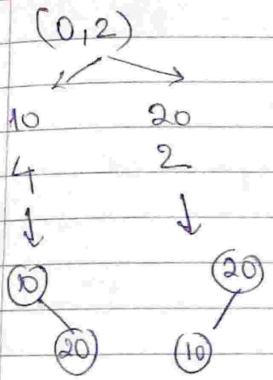
- l = j - i = 2

  2 - 0 = 2 (0,2)

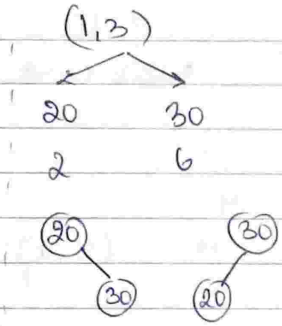
  3 - 1 = 2 (1,3)

  4 - 3 = 2 (3,4)

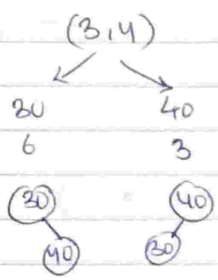
Finding cost by taking two keys



Cost = 1x4 + 2x2 = 8



Cost = 2x1 + 2x4 = 10



Cost = 1x2 + 2x6 = 14

Cost = 6x1 + 2x2 = 10

Cost = 1x6 + 3x2 = 12

Cost = 1x3 + 2x6 = 15

Min cost = 8

Min cost = 10

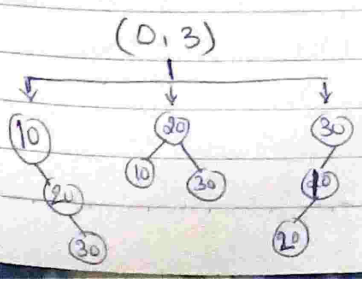
Min cost = 12

- l = j - i = 3

  3 - 0 = 3 (0,3)

  4 - 1 = 3 (1,4)

taking three keys



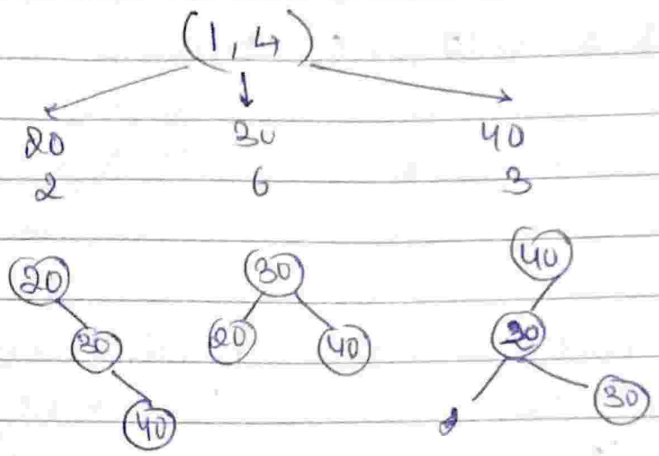
Calculating cost

1x4 + 2x2 + 3x6 = 26

1x2 + 2x4 + 2x6 = 22

1x6 + 2x4 + 3x2 = 20

Min = 20



$1\times2 + 2\times6 + 3\times3 = 22$

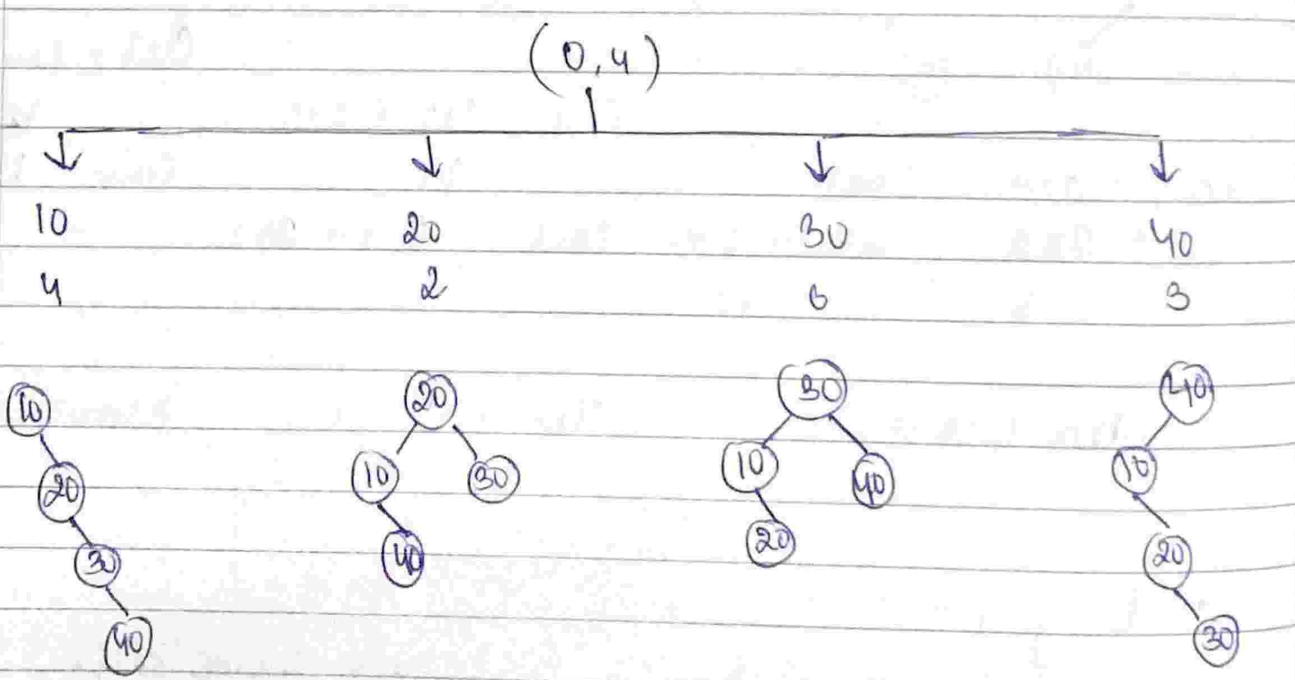
$1\times6 + 2\times2 + 2\times3 = 16$

$1\times3 + 2\times2 + 3\times6 = 25$

Min cost = 16

- $l = j - i = 4$

  $4 - 0 = 4 \quad (0,4)$



$1\times4 + 2\times2 + 3\times6 + 3\times4 = 38$

$1\times2 + 2\times4 + 2\times6 + 3\times3 = 31$

$1\times3 + 2\times4 + 2\times3 + 4\times6 = 26$

$1\times6 + 4\times2 + 3\times2 + 4\times2 = 28$

Min cost = 26

30

1×3 + 2×2 + 3×6
= 25

30 40
6 3

40 40 16 20 30

1×3 + 2×4 + 2×3 +
4×6 = 26

+ 3×2 + 4×2

M T W T F S S
Page No.:
Date:
YOUVA

* 0-1 knapsack Problem

Concept is :-

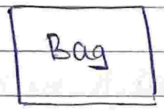
We have non fractionable objects with certain weight and then we have a bag to carry objects so we will either include the object or not

Include (1) Not Include (0)

Fractions not allowed

P - { 1, 2, 5, 6 }
W - { 2, 3, 4, 5 }

x_i = 0/1 x = { 1, 0, 0 }



M = 8

Condition :-

sum of profit should be Maximised

$$\max \sum p_i x_i$$

weight should be $\sum w_i x_i \leq m$

If we keep trying manually for n objects it will be 2^n solution thus time complexity too high $O(2^n)$ thus we opt DP approach.

Q Max weight of bag = 8
No. of object = 4

Profit {1, 2, 5, 6}
Weight {2, 3, 4, 5}

P_i	w_i		← wt → 0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5	6	6	7	8

Ans

→ Take first object and match with weight row fill the profit their

→ Previous to that fill same as above row
→ After that fill same number if other object dont exist

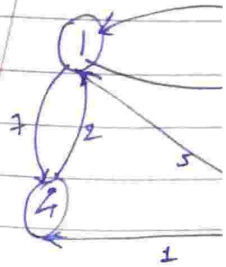
→ Taking 2 objects hence will consider two of them

Now to check sequence of object to be included in bag

→ starting from last row, does 8 comes in previous row ? No
Hence include 8

∴ profit left = 8 − 6 = 2

* Floyd



Now calculatin

wt ⟶

y	5	6	7	8
0	0	0	0	0
1	1	1	1	1
2	3	3	3	3
5	5	6	7	7
5	6	6	7	8

ch with weight crow
eis

as above row
number if other object

l consider two of them

bject to be

does 8 comes in previous

= 8 - 6 = 2

Checking third row, does 2 comes in prev row?
Yes
hence dont include third object

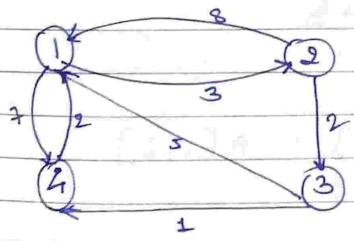
∴ Include second row
profit left = 2 - 2 = 0

Does 0 comes in 0th row, Yes
hence dont include first object

x_1 x_2 x_3 x_4 ← objects
0 1 0 1

Ans - include second and fourth object

* Floyd Warshall Method or All pairs shortest path



$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Self loop are = 0
Nodes w/o any connection = ∞

Now calculating A^1, A^2, A^3, A^4

A^1 = Keeping first column and row same and self loop always 0

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	5	∞	0

$A[2,3] = A[2,1] + A[1,3]$

$2 < 8 + \infty$

Choosing shortest path $\therefore = 2$

$A[2,4] = A[2,1] + A[1,4]$

$\infty > 8 + 7$

$\therefore 15$

$A[3,4] = A[3,1] + A[1,4]$

$1 < 5 + 7$

$\therefore 1$

$A[3,2] = A[3,1] + A[1,2]$

$\infty > 5 + 3$

$\therefore 8$

$A[4,2] = A[4,1] + A[1,2]$

$\infty > 2 + 3$

$\therefore 5$

$A[4,3] = A[4,1] + A[1,3]$

$\infty < 2 + \infty$

$\therefore \infty$

A^2 = Keeping

$A^2 =$ [0 3 / 8 0 / 5 / 2]

$A^3 =$ [0 / 7 / 5 / 2]

$A^4 =$ [0 /]

Formulae :-

$A[i,j] = N$

and row same and self loop are always 0

A^2 = Keeping second row and column of A^1 same and self loop = 0

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

finding all values same way

$$A^3 = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

$$A^4 = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

Formulae :-

$$A[i,j] = \min \left\{ A^{k-1}[i,j],\ A^{k-1}[i,k] + A^{k-1}[k,j] \right\}$$

Date
Page No.

Analysis and Design of Algorithm

GREEDY PROBLEM

We have a problem like travelling Delhi - Jaipur
Over this problem we have 'Constraints'
that we have to reach within 4 hours.

Considering the problem and its associated constrain we find out 'Feasible Solution'. It means a solution which satifying some constraints.
Eg - we can reach within 4 hrs only if we take flight or train

After this we go ahead to find that one optimal solution. 'Optimal solution' means that one feasible solution which satisfies the objective of a problem.

Objective was Min travel time
∴ Optimal solution is travel by flight.

Problems which require either Min / Max result such problems are called 'Optimization Problem'

Example of Greedy Method :- Taking selection Rounds when applying for a job.
First round filters some (who don't pass) then second and then third. Finally one best is selected

Alternatively, all participants must be given chance to participate all 3 rounds, but in greedy we want one best.
∴ Based on our approach we find

① 0/1 knapsack Pro

We have object
Profi
Weig

and a large

** Unlike d

→ Calculating Pro
We select objec

→ Now we will
Object t

→ Constraint

→ Objective

Solving :-

P
h
P/

Date
Page No.

lgorithm
EM

avelling Delhi - Jaipur
'Constraints'
within 4 hours.

, associated constrain
ution'. 4 means
some constraints.
reach within 4 hrs
train

find that one optimal
' means that one
sfies the objective of

ne
travel by flight.

Min / Max result
'Optimization Problem'

Taking selection Rounds
job.
dont pass) then second
one best is selected

be given chance to
greedy we want one best.

Date
Page No.

① 0/1 knapsack Problem (Greed Algo)

We have objects 1 2 3 4 5 6 7
profits 10 5 15 7 6 18 3
weights 2 3 5 7 1 4 1

and a large bag / container

** Unlike dynamic problem here objects are divisible like 5kg rice, 2 kg wheat

→ Calculating Profit / Weight of objects
We select objects base on this :-
P/W 5 1.3 3 1 6 4.5 3

→ Now we will calculate (x) → It is the fraction of object taken in bag

→ Constraint $\sum x_i w_i \leq 15$

→ Objective $\sum p_i x_i \geq \max$

Solving :-

P	10	5	15	7	6	18	3
W	2	3	5	7	1	4	1
P/W	5	1.3	3	1	6	4.5	3
X	1	2/3	1	0	1	1	1
	x_1	x_2	x_3	x_4	x_5	x_6	x_7

First selecting one with highest p/w
∴ Chose 5th object having
p/w = 6

hence full 5th object taken into the bag
Subtracting 5th object weight from
15kg so find amount of space left

15 - 1 = 14kg left.

Following same pattern we find x

15 - 1 = 14kg
14 - 2 = 12kg
12 - 4 = 8kg
8 - 5 = 3kg
3 - 1 = 2kg
2 - 2 = 0kg → taken 2kg out of 3kg as we did not have space left

∴ fraction x = 2/3

Calculating :-

$\sum x_i w_i = 15$ (Constraint satisfy)

$\sum p_i x_i = 54.6$ (objective met)

ecting one with highest p/w
object having
p/w = 6

5^{th} object taken into the bag
object weight from
so find amount of space left

14kg left.

find x

→ taken 2kg out of
3kg as we did
not have space left
∴ fraction x = 2/3

tisfiant satisfy)

ective met)

② TASK SCHEDULING PROBLEM

Jobs	J_1	J_2	J_3	J_4	J_5	J_6
Profit	35	30	25	20	15	12
deadline	3	4	4	2	3	1

Assuming deadline is the hours customer is willing to wait, where each job takes 1hr

Profit is what jobber will earn from doing job

∴ drawing Gantt Chart

0 (20) J_4 1 (J_3) 25 2 (35) J_1 3 (30) J_2 4

Choosing job with max profit first and hence descending

∴ Total profit = 20 + 25 + 35 + 30
= 110

Sequence = $J_4 \rightarrow J_3 \rightarrow J_1 \rightarrow J_2$

Q

Job	J_1	J_2	J_3	J_4	J_5
Profit	20	15	10	5	1
deadline	2	2	1	3	3

TABULAR

Job	Slot	Solution	Profit
J_1	[1,2]	J_1	20
J_2	[0,1][1,2]	$J_2 J_1$	35
J_3 X	[0,1][1,2]	$J_2 J_1$	35
J_4	[0,1][1,2][2,3]	$J_2 J_1 J_4$	40
J_5 X	[0,1][1,2][2,3]	$J_2 J_1 J_4$	40

③ MINIMUM COST SPANNING TREE

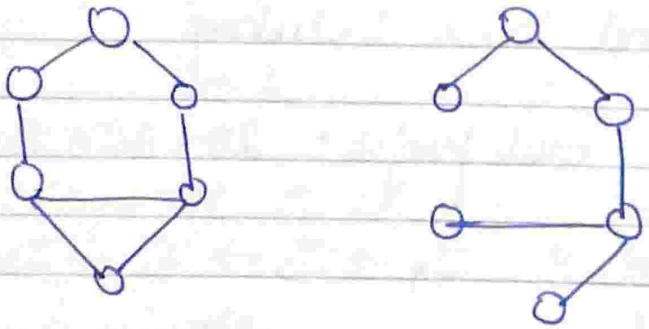
What is a spanning tree ?

Spanning tree is a subgraph which all vertices covered with minimum possible number of edges. It does not have a cycle

(graph) (ST)

No. of spanning tree possible = ${}^{E}C_{V-1}$ − no. of cycles

Example :-



E = 7 V = 6 Cycles = 2

∴ Spanning tree possible = ${}^{7}C_{6-1} - 2 = 5$

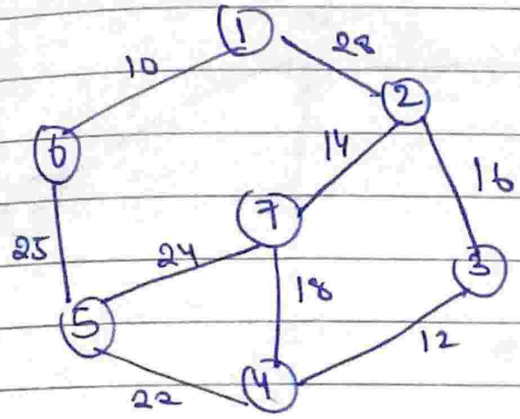
2 Algorithm

→ Prim's

→ Kruskal

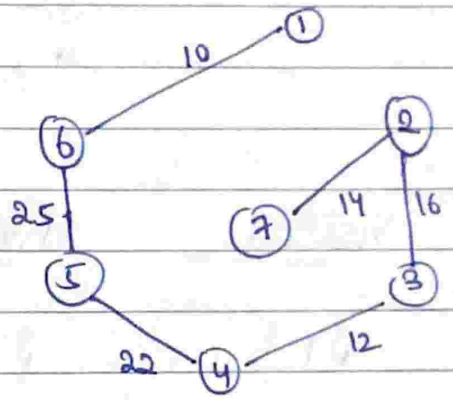
Prim's

Chose Min edge and Min edge joined to those 2 vertices



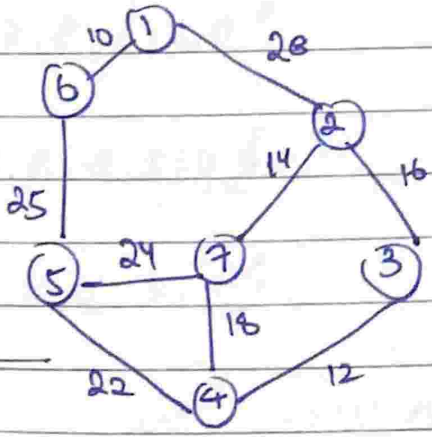
→ chosing edge 10 Then Min out of (1) as (6)

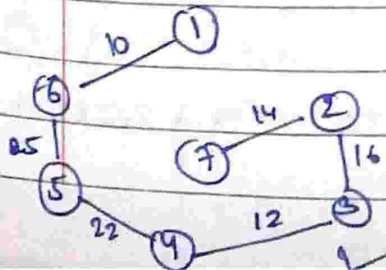
Prim's Result →



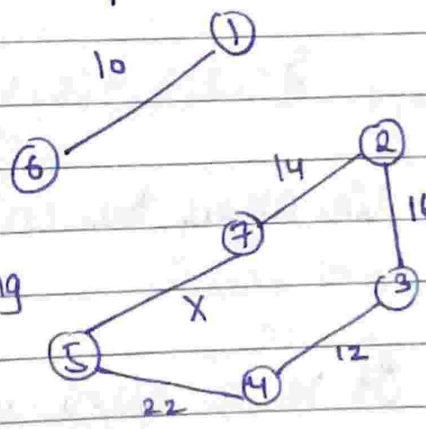
Kruskal

Chose Min edge and then the next Min edge





✓ correct

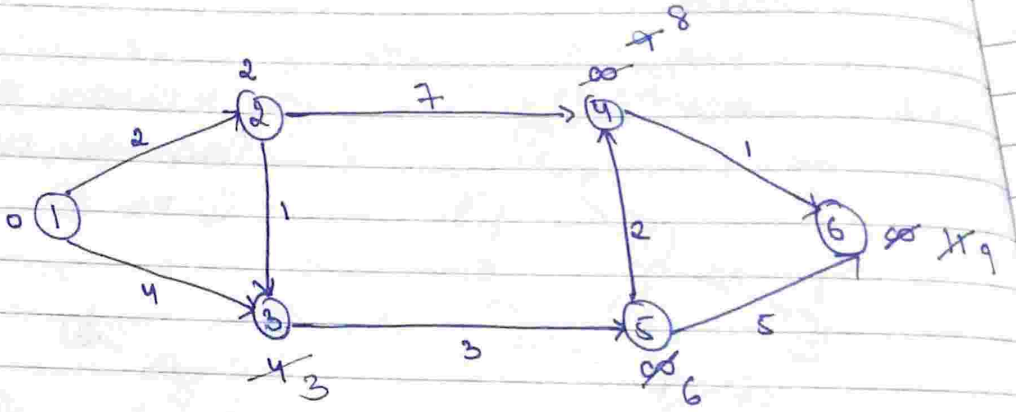


X wrong
You cannot form a cycle

④ Single Source Shortest Path

Dijkstra Algorithm

Eg: 1



Visited	Not Visited	Weights
{1}	{2,3,4,5,6}	{0 ②,4,∞,∞,∞}
{1,2}	{3,4,5,6}	{0,2,③,9,∞,∞}
{1,2,3}	{4,5,6}	{0,2,3,9,⑥,∞}
{1,2,3,5}	{4,6}	{0,2,3,8,6,11}
{1,2,3,4,5}	{6}	{0,2,3,8,6,9}
{1,2,3,4,5,6}	x	{0,2,3,8,6,9}

* We apply the concept of Relaxation here

$d[u] + c[v,v] < d[v]$

$d[u] + c[v,v] = d[v]$

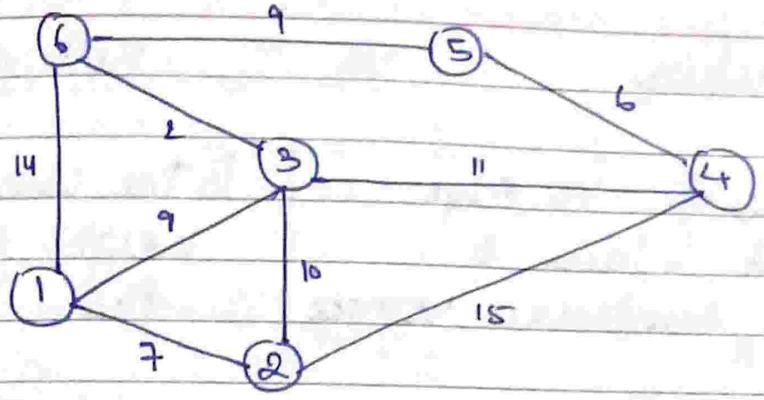
It is a minimisation problem hence using greedy Method

Eg: 2

Visited

{1
{1
{

Eg: 2



Visited	Not Visited	Weights
{1}	{2,3,4,5,6}	{0, ∞, ∞, ∞, ∞, ∞}
{1,2}	{3,4,5,6}	{0, ⑦, 9, ∞, ∞, 14}
{1,2,3}	{4,5,6}	{0, 7, ⑨, 22, ∞, 14}
{1,2,3,6}	{4,5}	{0, 7, 9, 20, ∞, ⑪}
{1,2,3,4,6}	{5}	{0, 7, 9, ⑳, 20, 11}
{1,2,3,4,5,6}	x	{0, 7, 9, 20, 20, 11}

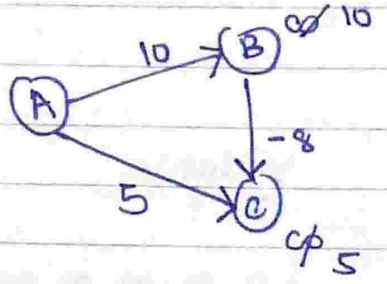
Bellman Ford

Dijkstra may or may not work properly for negitive edges. It follows dynamic strategy

We have to perform (n-1) times to achieve final answer. That is, we have to relax every edge (V-1) times.

Dijkstra vs Bellman

In the case of -ve edge weight it fails & does not minimizes vertices

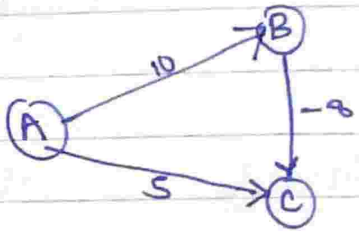


Chosing A and relaxing B and C
Then chosing CB as 5<10 but C is not relaxing any
Then chosing B but A, C are already relaxed.

∴ A - 0
B - 10
C - 5

but C should actually be 2

In the case of -ve weight of a cycle fails



Edge [(A,B) (B,C) (A,C)]

(A,B) - {0, 10, 5}

(B,C) - {0, ∞, ∞}

I st iteration

(A,B) - {0, 10, 5}

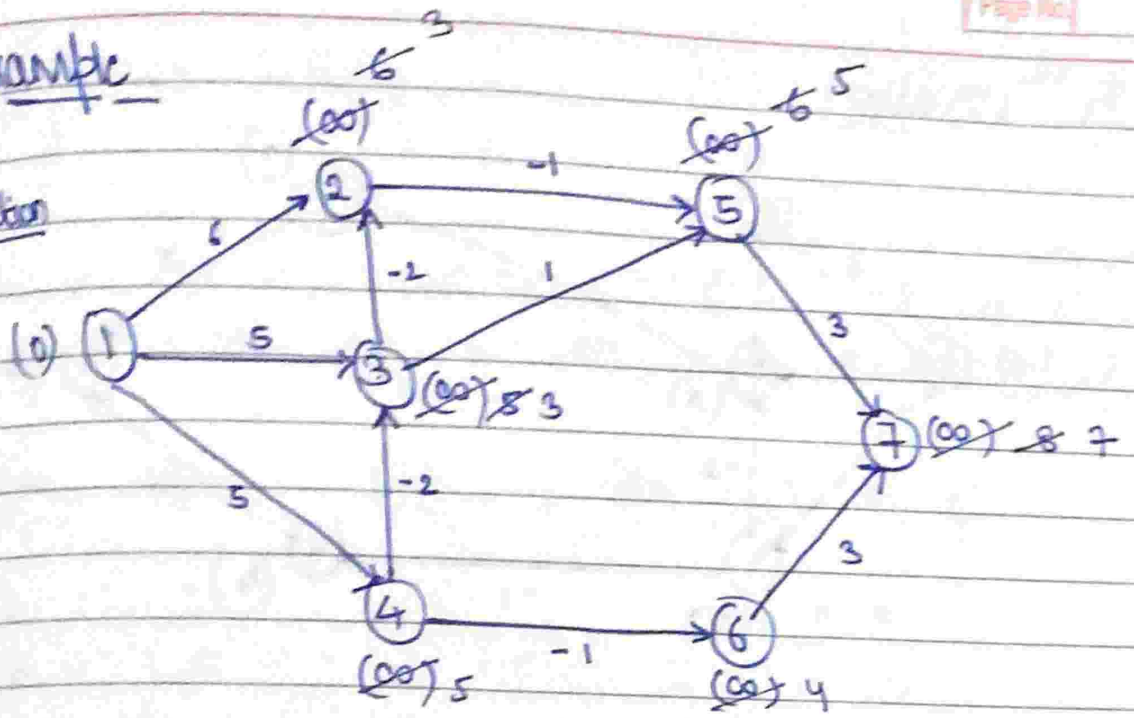
(B,C) - {0, 10, 2}

2nd iteration

Achieved Min value

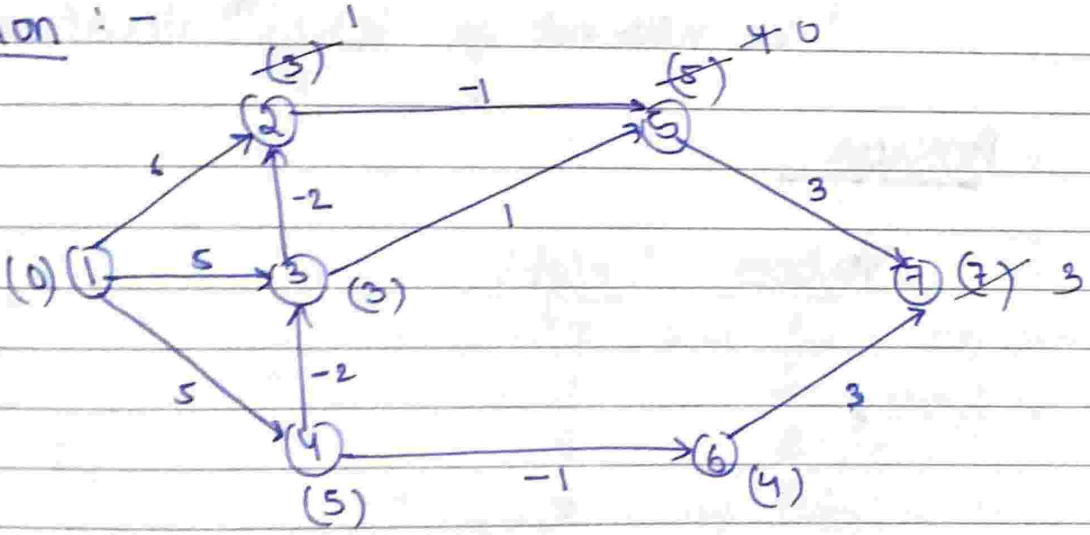
Example :-

I st iteration



Edges [(1,2), (1,3), (1,4) (3,5) (2,5) (3,2) (4,3) (4,6) (5,7) (6,7)]

II nd iteration :-



Again going through all edges :-

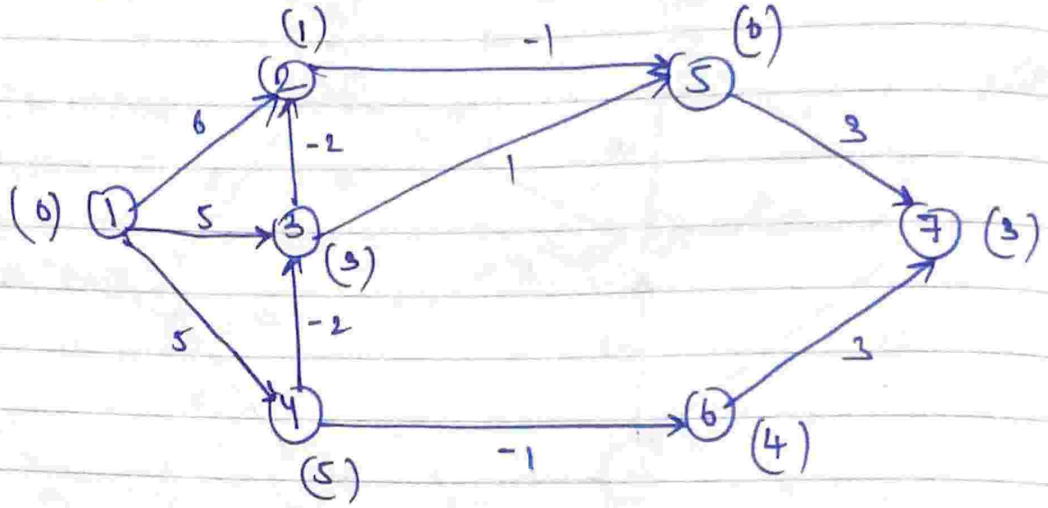
Edge (3,2) change weight of ② from 3 to 1

Edge (3,5) changed ⑤ from 5 to 4

Edge (2,5) changed ⑤ from 4 to 0

Edge (5,7) changed ⑦ from 7 to 3

III rd iteration :-



(1,2)	(1,3)	(1,4)	(3,5)	(2,5)	(3,2)	(4,3)	(4,6)	(5,7)	(6,7)
x	x	x	x	x	x	x	x	x	x

No change hence
we will not go till 6th iteration.

Answers

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

⑤ Huffman Coding

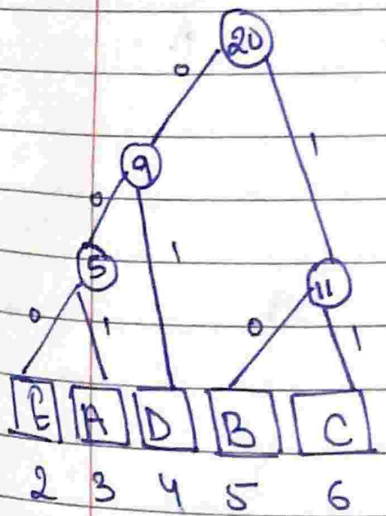
Message → BCCABBDDEACCBBAEDDCC
Find no. of bits req. to transmit this Message ?

# Char	Count	Bit	Bits Count x freq
A	3	001	3x3 = 9
B	5	10	2x5 = 10
C	6	11	2x6 = 12
D	4	01	2x4 = 8
E	2	000	3x2 = 6
= 5x8 bits		= 12 bits	= 45 bits
= 40 bits			

→ Arrange is ascending order lowest to highest bit order

2	3	4	5	6
E	A	D	B	C

→ Now take 2 min, add their bits and form a node and hence build a Tree



Tree

→ Appoint all left edges as 0 and and all right edges as 1

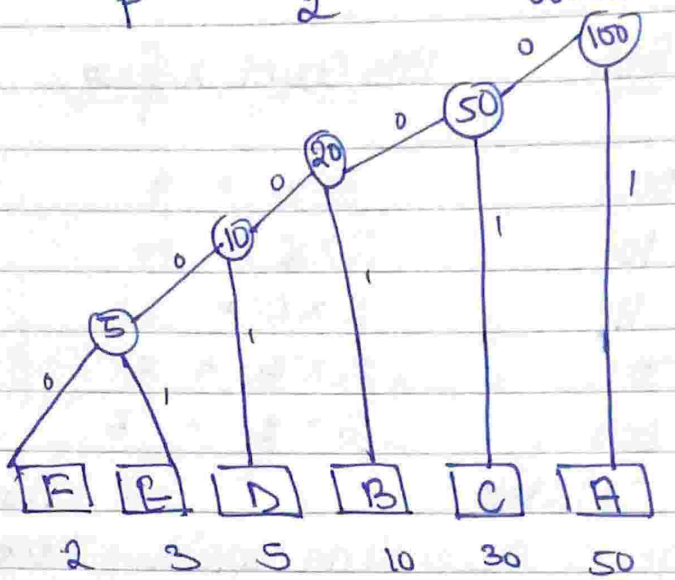
→ Now deduce the bits by checking the path from root node to the box

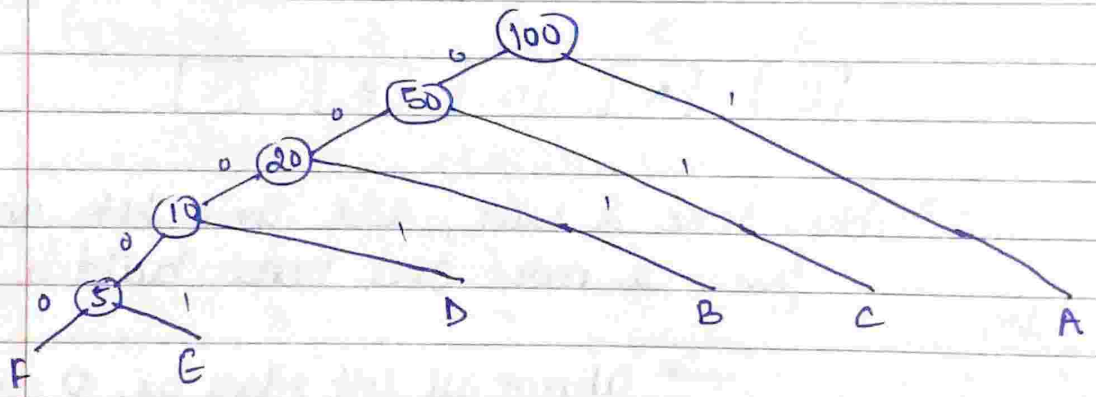
Message bits = 45
Table bits = 40+12
Total = 97 bits

Q

		bits
A	50	1
B	10	001
C	30	01
D	5	0001
E	3	00001
f	2	00000

$50 \times 1 = 50$
$10 \times 3 = 30$
$30 \times 2 = 60$
$5 \times 4 = 20$
$3 \times 5 = 15$
$2 \times 5 = 10$
185



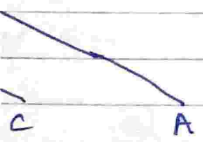


Message Bits = 185 bits
Table bits = 8×6 + 20 = 68
Total = 185 + 68

Date
Page No

$50 \times 1 = 50$
$10 \times 3 = 30$
$30 \times 2 = 60$
$5 \times 4 = 20$
$3 \times 5 = 15$
$2 \times 5 = 10$

185



= 68

Date
Page No

Analysis and Design of Algorithm
Unit IV

Topics

- Naive String Matching Algo
- Knuth-Morris-Pratt (KMP) Algo
- String Matching with finite automata
- Rabin Karp Algorithm

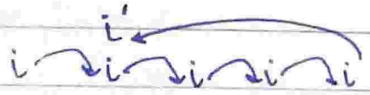
① Naive String Matching

String : a b c d a b c ab c d f

Pattern : a b c d f

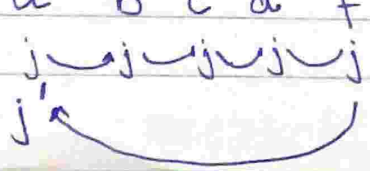
We have to check if the string contains the pattern or not
For that in naive matching

String will have an i pointer &
pattern will have a j
If i = j then i += 1 and j += 1
If i ≠ j then j will come to beginning and
i will back-track to second digit / alphabet



String : a b c d a b c a b c d f

Pattern : a b c d f



of Algorithm

lgo
utomata

b c d f

ng contains the

ointer &
e a j
j += 1
beginning and
second digit / alphabet

a b c d f

Worst Case :

(n) string : a a a a a a a b
1 2 3 4 5 6 7 8

(m) Pattern : a a a b
1 2 3 4

Naive / Basic = O(mn)

② KMP Algo

We have a pattern given for which we have to write a π table.

Eg 1 : a b c d a b e a b f
0 0 0 0 1 2 0 1 2 0

Eg 2 : a b c d e a b f a b c
0 0 0 0 0 1 2 0 1 2 3

Eg 3 : a a b c a d a b e
0 1 0 0 0 1 0 1 2 0

Eg 4 : a a a a b a a c d
0 1 2 3 0 1 2 0 0

In π table we identify if any set of letters are repeating itself

Time Complexity O(n+m)

Example :-

String : a b a b c a b c a b a b a b d
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

pattern : a b a b d

Drawing π table :

a	b	a	b	d
0	0	1	2	0

j	0	1	2	3	4	5
	a	b	a	b	d	
	0	0	1	2	0	

we will start iteration from 0

We will check if i and j+1 match and if yes then both will move forward

If dont, then j will jump to that position which is the no. of its π table's value.

Eg :- if j is at 4th position and j+1 ≠ i then j will jump to 2nd position

If j reaches last 0th position and cannot backtrack anymore then i will move one step forward

Then start comparing again if i = j+1 and when j reaches end and i jump out of string, answer is found.

Example :

string : a

pattern :

	0	1
	a	

i i i i i i

a b c a b

5th

j j i j j

0	1	2	3	4
a	b	a	b	
0	0	1	2	

① i = j+1 ✓

② i = j+1 ✓

③ i ≠ j+1 x h
back

④ i ≠ j+1 but
thus

⑤ i = j+1 ✓

⑥ i = j+1 ✓

⑦ i = j+1 ✓

⑧ i = j+1 ✓

∴ j reached la

c ab a b a b d
7 8 9 10 11 12 13 14 15

b a b d
0 0 1 2 0

5
d
0

tch and if yes then

that position which
ble's value.

and j+1 ≠ i then
nd position

and cannot backtrack
ve one step forward

if i = j+1
and i jump out
d.

Example:

string : a b c a b a b a

pattern :

0	1	2	3	4
a	b	a	b	
0	0	1	2	

i i i i i i i i i
1st a b c a b a b a

j j j j
0 1 2 3 4

a	b	a	b	
0	0	1	2	

→

i i j j j j
0 1 2 3 4

a	b	a	b	
0	0	1	2	

① i = j+1 ✓
② i = j+1 ✓

③ i ≠ j+1 × hence j jumping back to 0th

④ i ≠ j+1 but j cannot backtrack ×
thus i = i+1

⑤ i = j+1 ✓
⑥ i = j+1 ✓
⑦ i = j+1 ✓
⑧ i = j+1 ✓

∴ j reached last thus pattern is found

③ Rabin Karp Algorithm

String Matching with finite automata :-

Text - a b c d a b c e

(10, 6, 9, 7, 6, 8)

Pattern - b c e

2 + 3 + 5

= 10

Assigned Values

a - 1
b - 2
c - 3
d - 4
e - 5

∵ bce matches pattern's bce

= 10

hence pattern found

Case of Spurious hits :-

Text : c c a c c a e d b a

(3+3+1 = 7, 7, 7)

Pattern : d b a

4 + 2 + 1 = 7

Pattern don't match but automata does
Such case are spurious case
To avoid that Rabin Karp Algo got introduced

Task: c c a c c a a e d b a

Pattern: d b a

$4\times10^2 + 2\times10^1 + 1\times10^0$

$= 421$

In this simply dont add, we first multiply each with 10^{M-1} where M is the position it is placed on

for cca = $3\times10^2 + 3\times10^1 + 1\times10^0$

$= 331$

Since we use the 'Rolling Hash Function' hence, now we will take the next slot if this slot fails to match the automata of pattern

for cac =>

we have a formulae here

$$\left(\left\{\left[\underbrace{3\times10^2 + 3\times10^1 + 1\times10^0}_{(cca)}\right] - \underbrace{3\times10^2}_{c}\right\}\times10 + \underbrace{3\times10^0}_{c}\right)$$

$\Rightarrow (331 - 300)10 + 3$

$\Rightarrow 313$

$\Rightarrow 313 \neq 421$

$\therefore$ Similarly we will calculate till we find 421

ADA Revision

Greedy Approach

① Fractional Knapsack Problem –

Complexity O(N log N)

- Obj, profit, weight given.
- Calculate P/W
- Find x (fraction) by chosing max P/W
- Keep subtracting remaining weight
- Find constraint + objective

Pseudo Code :-

```
W = 0 ;
P = 0 ;
Sort items in descending order of P/W
for each item i in sorted list
    if ( W + Wi ≤ capacity )
        W = W + Wi
        P = P + Pi
    Else
        remaining = capacity - W
        profit = P + remaining * (P/W)
        break
```

② Job Scheduling Algo

- Given job, profit, deadl
- Assume deadline →
- Each job → takes 1 hr
- Either draw Gantt ch
- Take max profit job

③ Prims –

Complex
Data

- Select min edge
- Then, min edge fro

④ Kruskal –

Compl
Data
Appro

- Chose min edge the

⑤ Dijkstra's

Con
Dat

- We form a table o
- We perform Relaxati
- Minimisation Prob

on

-

Complexity O(N log N)

Max P/W
weight

ing order of P/W
sorted list
capacity)

capacity - W
+ remaining * (P/W)

② Job Scheduling Algo - Complexity $O(N^2)$

- Given job, profit, deadline.
- Assume deadline → hours customer willing to wait
- Each job → takes 1hr
- Either draw Gantt chart / table
- Take max profit job & start placing

③ Prims -

Complexity - $O(|E| \log |V|)$
Data structure - Binary Heap

- Select min edge
- Then, min edge from the two vertices

④ Kruskal -

Complexity - $O(E \log E)$
Data structure - disjoint set
Approach used - greedy

- Chose min edge then min edge

⑤ Dijkstra's

Complexity - $O(V + E \log V)$
Data structure - Min Priority Queue

- We form a table of visited & Non visited vertices
- We perform Relaxation for each vertices
- Minimisation Problem (Greedy)

⑥ Bellman Ford

Complexity - $O(V \cdot E)$
Data Structure - Array or list
Approach - Dynamic Programming

We write all the edges and perform Relaxation.
No. of Iteration = Vertices - 1

Relaxation - $d[u] + c[u,v] < d[v]$
$d[u] + c[u,v] = d[v]$

Pseudo Code for Prim's & Kruskal

Prim's

```
begin
    T = ϕ
    U = {1};
    while U ≠ V
    let (u,v) be least cost edge such that u∈U and v∈V-U;
    T = T U {(u,v)}
    U = U U {v}
End procedure
```

Kruskal's

```
begin
    A = ϕ
    For each vertex v ∈ G.V :
        MAKE-SET (v)
    For each edge (u,v) ∈ G.E ordered
      by weight in inc. order (u,v):
        if FIND-SET(u) ≠
              FIND-SET (v)
        A = A U {(u,v)}
        UNION (u,v)
    return A
    End kruskal
```

⑦ Huffman Coding

It is a data cor
data without lo

It is greedy beca
depends on freq

- Count freq of a
- Arrange in a
- Draw a tree
- Multiply freq

- Table bits =
- Total bits =

⑧ Naive String M

Backtracking

⑨ KMP Algo

For text, back
for pattern,

⑩ Rabin Karp Algo

$O(V \cdot E)$

e - Array or list

Dynamic Programming

perform Relaxation.

$< d[v]$

$] = d[v]$

uskal

such that $u \in U$ and $v \in V-U$;

if FIND-SET(u) ≠ FIND-SET (v)
A = A ∪ {(u,v)}
UNION (u,v)
return A
End kruskal

esed
:

⑦ Huffman Coding

Complexity - $O(N \log N)$
Approach - greedy

It is a data compression algo used to reduce size of data without loosing information

It is greedy because size of code assigned to a char depends on frequency of char

- Count freq of all characters
- Arrange in ascending order
- Draw a tree and find bits for each char
- Multiply freq of char x No. of bits character
  ↳ Message bits
- Table bits = No. of bits characters + 8 x No. of char
- Total bits = Msg + table Bits

⑧ Naive String Matching

Complexity - $O(mn)$

Backtracking of i - drawback

⑨ KMP Algo

Complexity - $O(M+n)$

For text, backtracking of i is avoided
For pattern, π table is drawn and j starts from 0

⑩ Rabin Karp Algo

Complexity - $O(M+n)$ Worst
$O(Mn)$ Avg or best

Dynamic Programming

① Matrix Multiplication –

Complexity – $O(n^3)$

it is bettern than naive which has $O(2^n)$

- We draw M table and S table
- $A_1\ A_2\ A_3\ A_4 = 0$
- $A_1 \cdot A_2$, $A_2 \cdot A_3$, $A_3 \cdot A_4$
- $A_1 A_2 A_3$, $A_2 A_3 A_4$

$A_1(A_2A_3)$ $(A_1A_2)A_3$ $A_2(A_3A_4)$ $(A_2A_3)A_4$

- $A_1A_2A_3A_4$

$A_1(A_2A_3A_4)$, $(A_1A_2)(A_3A_4)$, $(A_1A_2A_3)A_4$

② longest Common Sequence –

Complexity – $O(n * m)$

- Draw table with 0^{th} row and 0^{th} column
- Write string 1 vertically and string 2 horizontally

If $A[i] = B[j]$

$LCS[i,j] = LCS[i-1, j-1] - 1$

else

$LCS[i,j] = \max(LCS[i, j-1], LCS[i-1, j])$

Approach – dynamic programming

sity - $O(n^3)$
which has $O(2^n)$

ble

3) A_4

$A_2A_3)A_4$

$n*M)$

column
ing 2 horizontally

$LCS[i-1, j]$)

③ OBST

Complexity - $O(n^3)$
Approach - Dynamic Programming

Key → freq
Draw a table starting with 0 → cost table
Take L(j-i) = 0, 1, 2, 3
(if 4 keys given)
Then start searching values

④ 0/1 knapsack Problem

Complexity - $O(nW)$
Approach - Greedy

- Draw a table with 0th row and column
- Write profit and weight vertically and fill
- Find the objects to include

⑤ Floyd Warshall

Complexity = $O(n^3)$

$A[i,j] = \min(A[i,j], A[i,k] + A[k,j])$

LIST

Matrix Chain	$O(n^3)$	DP
OBST	$O(n^3)$	DP
Floyd Warshall	$O(n^3)$	DP
Longest Common (LCS)	$O(MN)$	DP
0/1 Knapsack	$O(nW)$	DP
Fractional Knapsack	$O(n \log n)$	Greedy
Huffman Coding	$O(n \log n)$	GDy
Job Scheduling	$O(n^3)$	GDy
Prim's	$O(E \log V)$	binary heap GDy
Kruskal	$O(E \log E)$	disjoint sets GDy
Dijkstras	$O(E \log V + V)$	priority Queue GDy
Bellman	$O(E * V)$	array / LL DP
Naive String	$O(MN)$	
KMP	$O(M+N)$	
Rabin Karp	$O(M+N)$	
Normal Matrix	$O(n^3)$	
Strassen's Matrix	$O(n^{2.81})$	
Fibonacci (Divide & Conquer)	$O(2^n)$	
Fibonacci (DP)	$O(n)$	

Binary search $T(n) = T(n/2) + 1$

Merge Sort $T(n) = 2T(\frac{n}{2}) + n$

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