Chapter 1

Advance Algorithm Analysis and Design Techniques:

Amortized cost Analysis :-

⇒ It is also known as average cost Analysis. It is the method of analyzing the costs associated with the data structure that averages the worst operations out over time. This is required because one worst case cannot signify the overall performance of an algorithm. So, amortized analysis is used to ~~always~~ average out the costly operations in the worst case.

There are three main types of amortized analysis:-

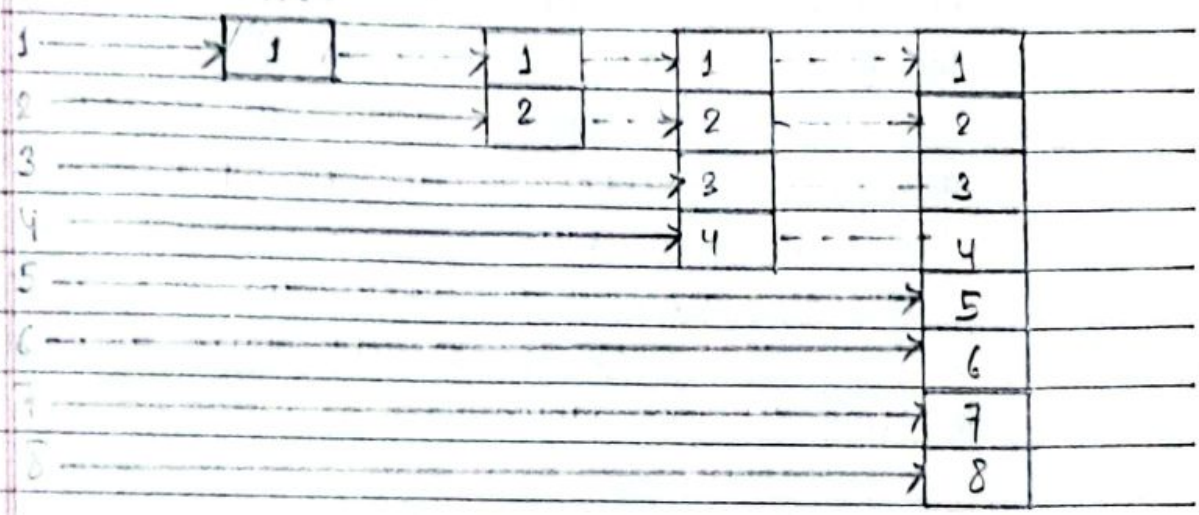
1. Aggregate Analysis.
2. Accounting Analysis method.
3. Potential Analysis method.

1. Aggregate Analysis :-

In Aggregate analysis, there are two steps, first, we must show that the sequence of n operations takes T(n) time in the worst case. Then, we show that each operation takes $\frac{T(n)}{n}$ time on average. Therefore, in aggregate analysis, each

operation has some cost.

Consider an example of Hash Table Insert.



Insert i	1	2	3	4	5	6	7	8	9	10
size i	1	2	4	4	8	8	8	8	16	16
Cost Ci	1	2	3	1	5	1	1	1	1	1

Here, Ci is the cost involved. which is the sum of copy + insert.

Now,

$$C_i = \begin{cases} 1 & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \text{ (Insert)} \\ & 1\ 2\ \ \ 4\ \ \ \ \ \ \ 8 \text{ (Copy)} \end{cases}$$

$$C_i = \begin{cases} i & \text{if } (i-1) \text{ is the exact power of 2} \\ 1 & \text{otherwise.} \end{cases}$$

Again,

$$C_i = n + \sum_{j=0}^{\lfloor \log_2 i \rfloor} 2^j$$

As this is the geometric series, it can be generalized as

$$C_i = n + 2n - 3$$
$$= 3n - 3$$

$O(n) = n$

Average cost $\Theta(n) = \frac{O(n)}{n} = 1$

Amortized cost analysis of hash table if the table doubling is also included.

⇒ While using dynamic table as array, following are the step needed to be followed, when table become full.

i) Allocate memory for twice the old table size.
ii) Copy the contents of old table to new table
iii) free the old table.

If the table doubling cost is also included then,

Item	size	Operation Cost	Copying cost	Doubling cost	Total cost
1	1	1	0	0	1
2	2	1	1	1	3
3	4	1	2	1	4
4	8	1	0	0	1
5	8	1	4	1	6
6	8	1	0	0	1
7	8	1	0	0	1
8	8	1	0	0	1
9	16	1	8	1	10
10	16	1	0	0	1

Here, C_i = operation cost + copying cost + Doubling

i.e. $C_i = n + \sum_{i=0} 2^i + \sum_{j=0} 1$

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

$= 2n + 2n - 3$

$= 4n - 3$

$= O(n)$

$\therefore \hat{C_i} = \Theta(n) = On/n = 1$

DATE
PAGE No.

② Accounting Analysis Method:-

The accounting method is aptly named because it burrows ideas and terms from accounting. This method is burrowed from finance where there is the concept of storing excess amount in the bank for future use. Here, each operation is assigned a charge called the amortized cost. Some operations can be charged more or less than they actually cost. If amortized cost exceeds its actual cost, we assign the difference called credit. Credit can never be negative in any sequence of operations. This method requires setting the cost of C_i (Amortized Cost), which should be equal for all inserts such that. Bank ≥ 0 (Cost)
which is turn will satisfy the inserts of all the inputs. $\hat{C}_i = 3$

Case 1

$\hat{C}_i$	3	3	3	3	3	3	3	3	3	3
Bank$_i$	2	3	3	5	3	5	7	9	3	1
C_i	1	2	3	1	5	1	1	1	9	1

$\hat{C}_i = 3n$

Case 2

i	1	2	3	4	5	6	7	8	9	10
Size$_i$	1	2	4	4	8	8	8	8	16	16
C_i	1	2	3	1	5	1	1	1	9	1

$\hat{C}_i$	3	3	3	3	3	3	3	3	3	3
Bank	2	2	3	5	3	5	7	3	3	5

if guess = 2

c_i	2	2	2	2	2	2	2 2	2	2
Bank	1	1	0	1	-1				

which is invalid.

So,

$\hat{c_i} = 3$ ← limit is correct guess.

if every 1 operation cost is 3 then,

n operation cost is 3n.

$\therefore \theta(n) = \frac{3n}{n} = 3 = 1$ (constant).

(c) Potential Method:-

Here, we have to define potential function (ϕ)

$\phi(D_0)$

$\phi(D_i)$

$\phi(D_{i-1})$

Potential difference = $\phi(D_i) - \phi(D_{i-1})$

lets assume $\phi(D_i) = 2i - 2^{\lceil \log i \rceil}$

$\phi(D_{i-1}) = 2(i-1) - 2^{\lceil \log (i-1) \rceil}$

$\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})$

Here,

$c_i = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}$

$\therefore \hat{c_i} = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases} + 2i - 2^{\lceil \log i \rceil} - [2(i-1) - 2^{\lceil \log (i-1) \rceil}]$

Case I :-

$\hat{C_i} = i - 2i - 2^{\lceil \log i \rceil} \ldots - 2i - 2 - 2^{\lceil \log (i-1) \rceil}$

$= i - 2(i-1) + 2 + (i-1)$ generalization

$= i - 2i + 2 + 2 + i - 1$

$= 4 - 1$

$= 3$

Case II :-

$\hat{C_i} = 1 - 2^{\lceil \log i \rceil} + 2 + 2^{\lceil \log (i-1) \rceil}$

$\Rightarrow$ generalization

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

$= 1 - i + 1 + 2 + i - 1$

$= 3$

Probabilistic Analysis :-

- Las vegas
- Monte carlo

> come from casino gambling concept.

- Las vegas:

Las vegas answer is always correct.
{It's output is always correct}
eg: Randomized Quick sort.

- Monte carlo:-

Its output may or maynot be always correct. Eg: primarility test.

$a^n \,\%\, n = 1$; $a < n$

where,

DATE
PAGE No.

n is the number to be tested (input)
If n=3.
then, q=1,2

q=1 $1^3 \% 3 = 1$

q=2 $2^3 \% 3 = 8 \% 3 = 2$

Monte carlo = O(n)=1 " Brute force O(n)=n.
(primarility)

DATE 2078-08-17
PAGE No.

Greedy Approach:-

a. Job scheduling
b. Tree vertex splitting

a. Job scheduling:-

Assumptions:-

1. Only one Machine
2. J_i (~~Any~~ Each Job) requires 1 unit of time.
3. J_i is associated with d_i and P_i where:
   d_i: deadline
   P_i: Profit
4. J_i is associated with P_i iff it meets its deadline
5. find the subset of Jobs and its sequence, ~~and~~ so that its profit is maximum.

Algorithm:-

1. Arrange J_i in decending order with respect to P_i.
2. Allocate Time slot with respect to maximum deadline.
3. $k = \min(\max(\text{deadline}), d_i)$
4. Allocate the time slot for J_i with respect to value of k.
5. If the time slot is already occupied then search for the positions before it.
6. If all the timeslot is occupied publish the set and sequence and max profit.

Example:-

①

Jobs :	J_1	J_2	J_3
deadline :	1	1	1
Profit :	10	15	30

Soln Here,

Ji in decending order with respect to Pi,

Ji	J_3	J_2	J_1
di	1	1	1
Pi	30	15	10
i=	1	2	3

∴ Time slot = 1

Then, i = 1

k = min (max (deadline), di)

= min (1, 1)

= 1

| J_3 |

set: {J_3}

Sequence : J_3

Profit : 30 ✱

Q. No. 2

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

Soln Here,

Ji in decending order with respect to Pi,

Ji:	J_1	J_4	J_3	J_2
Pi:	100	27	15	10
Di:	2	1	2	1
i	1	2	3	4

DATE
PAGE No.

∴ Time slot = 2

1	2

Then, i = 1

k = min (max (deadline), di)

= min (2, 2)

= 2

1	2
	J_1

J_1	

set:

Sequence:

Profit:

Then, i = 2

k = min (max (deadline), di)

= min (2, 1)

= 1

1	2
J_4	J_1

set(J): { J_1, J_4 }

Sequence : J_4, J_1

Max profit : 127 ✱

Q. No. 3

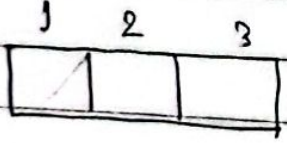
Ji:	J1	J2	J3	J4
Pi:	30	15	75	80
di:	1	2	3	2

Sol^n

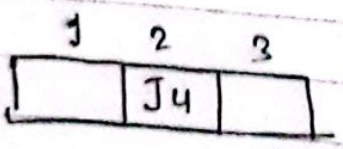
Here,
Ji in decending order w.r.t Pi,

Ji:	J4	J3	J1	J2
Pi:	80	75	30	15
di:	2	3	1	2
i	1	2	3	4

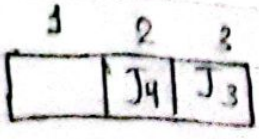
Then,
Time slot = 3



Then, i = 1
K = min (max (deadline), di)
= min (3, 2)
= 2



for, i = 2,
K = min. (max (deadline), di)
= min. (3, 3)
= 3



for, i = 3,

k = min (max (deadline), di)
= min (3, 1)
= 1

	1	2	3
∴	J1	J4	J3

Then,

J1	J4	J3

Set : {J1, J3, J4}
Sequence :- J1, J4, J3
Max profit : 185 ✗✗

∴ Time complexity = O(n)
= n

P.No.4

Ji	J1	J2	J3	J4
Pi	30	15	75	80
di	1	2	2	2

S.n

In decending order w.r.t. Pi,

Ji:	J4	J3	J1	J2
Pi:	80	75	30	15
di:	2	2	1	2
i:	1	2	3	4

No.4

Ji : J1 J2 J3 J4
Pi : 30
di

∴ Time slot = 2

	1	2
	J3	J4

for, i = 1,

k = min (max (deadline), di)
= min (2, 2)
= 2

for, i = 2,

k = min (max (deadline), di)

= min (2, 2)

= 2

1	2
J3	J4

set[J] = {J3, J4}

Sequence = J3, J4

Max profit = 155 ✱

∴ Time complexity = $O(n^2) = n^2$

- Analysis :-

Case I :

Each time slot calculated is vacant. So just insert is required; which is required to the no. of iterations. ∴ $O(n) = n$

Case II :

Time slot calculated is not vacant So, requires another iteration to find the vacant slot.

$O(n) = n^2$

- Greedy Job(d, J, n)

{ J = {1}

for i = 2 to n do

{if all deadlines in J ∪ {i} are feasible then

J = J ∪ {i}

}

```
    return J;
}
```

b. # Tree vertex splitting:-

Let, $T=(V, E, W)$ & δ as tolerance, He

V = Vertex

E = Edge

W: Weight

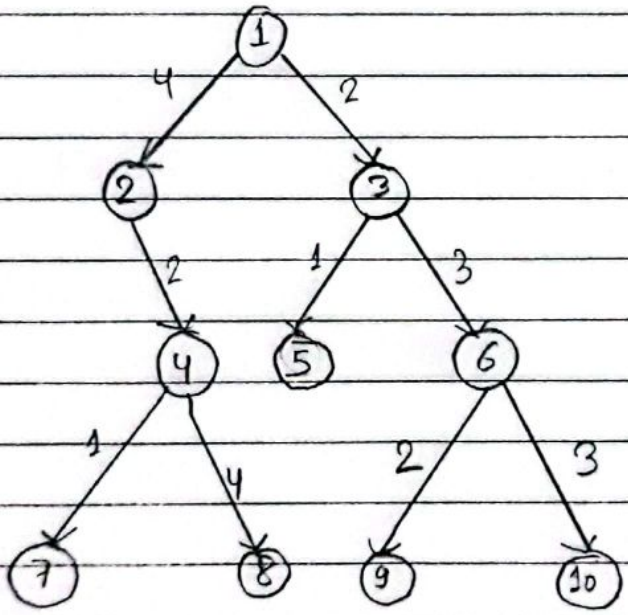


fig: Directed graph.

⇒ Here, All leaf nodes delay = 0 and leaf nodes are, 5, 7, 8, 9, 10.

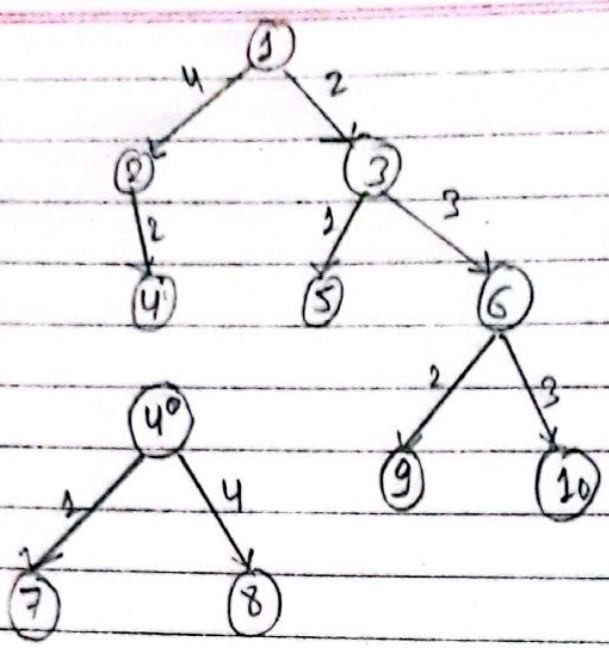
$d[5], d[7], d[8], d[9], d[10] = 0$

We have, $d[v]$ & Suppose $\delta = 5$, then,

$$d[4] = \max \begin{cases} d[4] + W(2,4) & [d[4] = d(4,7) \\ d[4] + W(2,4) & [d[4] = d(4,8) \end{cases}$$

$$= \max \begin{cases} 1+2=3 \\ 4+2=6 \end{cases}$$

$$= 6$$



Here, i= inbound and 0 = outbound.

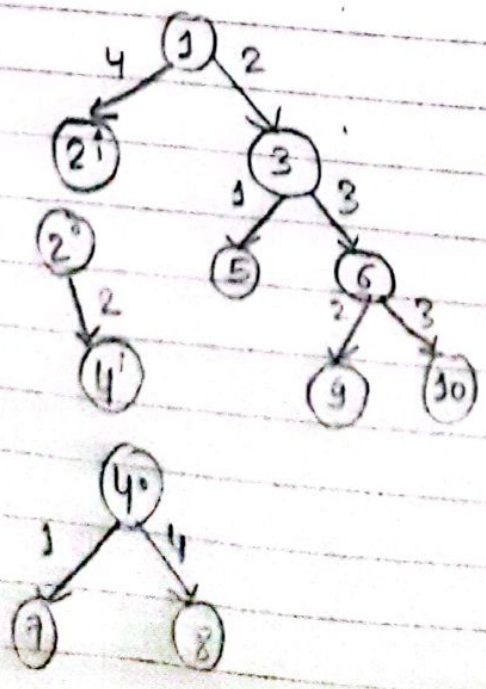
$d[2] = \max \begin{cases} d[2] + w(4) & [d[2] = d(2,4)] \\ \cancel{d[2] + w} \end{cases}$

$= \max [2+4]$

$= \max [6]$

$= 6$

Then,

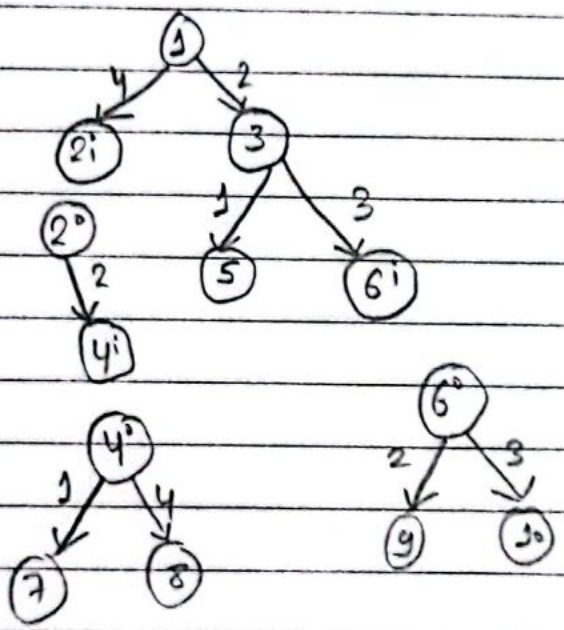


$d[6] = \max \{ d(6) = (5,9),\ d[6] = (8,10) \}$

$= \max \{ 2+3 = 5,\ 3+3 = 6 \}$

$= 6$

Then,



Then,

$d[3] = \max \{ d(3) = d(3,5),\ d[3] = d[3,6) \}$

$= \max \{ 1+2 = 3,\ 3+2 = 5 \}$

$= \max \{ 3, 5 \}$

$= 5$

✻

Analysis,

- Visit each vertex = n
- In each, vertex there is constant time required for calculating and comparing is with 5.

  ∴ O(n) = n.

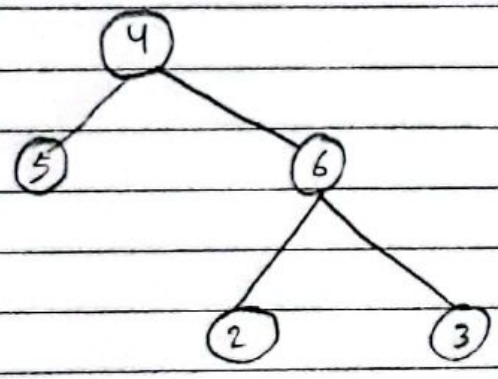
Dynamic Programming

DATE 2078-08-27
PAGE No.

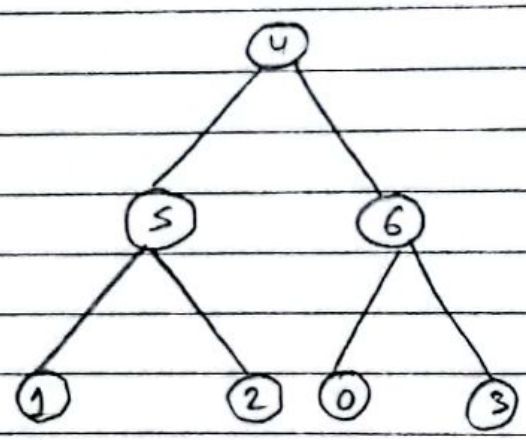
⇒ OBST (Optimal Binary Search Tree)

⇒ Difference between BT and BST.

⇒ Binary Tree (BT)

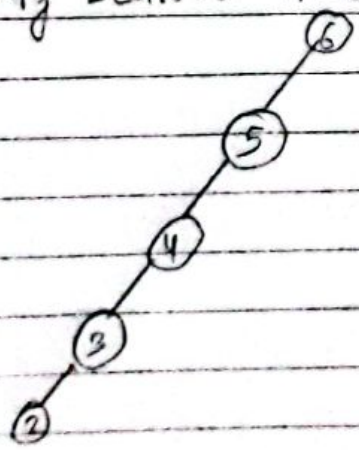


(a) Incomplete Binary Tree

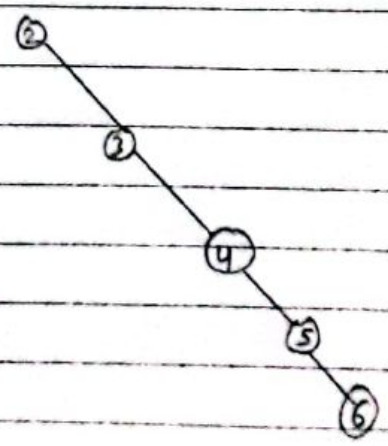


(b) Complete BT

⇒ Binary Search Tree (BST)



(c) Left-skewed Binary Search Tree



(d) Right-skewed BST

	BT	BST
Eg	No constraints on placement of values within child nodes	(e) Strict constraints have to be followed for placement of values within child nodes. less ← → greater

Let us suppose 'n' is the no. of keys. Total no. of structures (BST) that can be formed is

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

Suppose, n = 4 keys

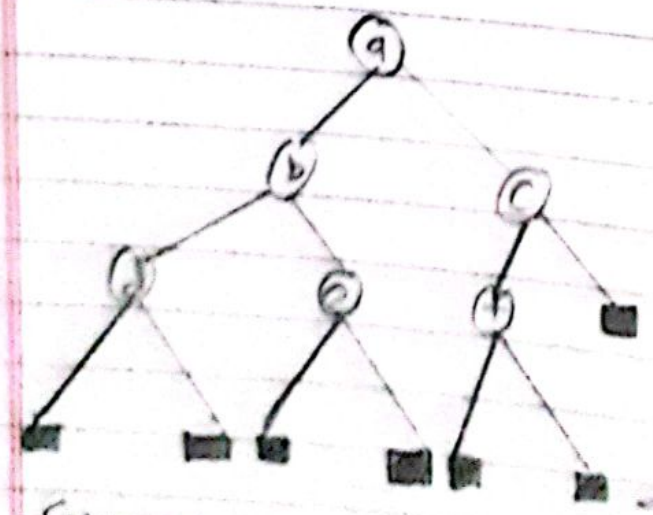
Then,

Total structure = $\frac{8!}{4! \times 5!}$

$= \frac{8 \times 7 \times 6 \times 5!}{4! \times 5!}$

$= \frac{2\cancel{8} \times 7 \times \cancel{6}^{3}}{\cancel{4} \times 3 \times 2 \times 1}$

$= 14$



Suppose — dummy keys

p_i = success probability

q_i = failure probability

$\therefore \sum p_i + \sum q_i = 1$

DATE
PAGE No.

Q.1. Let k_1, k_2, k_3, k_4, k_5 be the keys provided $k_1 < k_2 < k_3 < k_4 < k_5$. Place the dummy keys yourself. Given that, success and failure probabilities are as follows:-

i	0	1	2	3	4	5
Pi		0.15	0.10	0.05	0.10	0.20
qi	0.05	0.10	0.05	0.05	0.05	0.10

Ans

To solve this type of problem:
Let $e[i,j]$ be the expected search cost of OBST.

Table 1:-

$e[i,j] = e[i,r-1] + e[r+1,j] + w[i,j]$ for $j \geq i$.

$= q_{i-1}$ for $j < i$.

$\therefore e[i,j] = q_{i-1}$ for $j < i$ — 2nd process

Table 2:-

$w[i,j] = w[i,j-1] + p_j + q_j$ for $j \geq i$

$= q_{i-1}$ for $j = i-1$

$\therefore w[i,j] = q_{i-1}$ for $j = i-1$ — first process

Table 3:-

Root Table = r — 3rd process

Q.1. Ans

Solution:-

for Table 1; [w [i,j]),

W	0 (j →)	1	2	3	4	5
1 (i ↓)	q_0 0.05	0.30	0.45	0.55	0.7	1.0
2		q_1 0.10	0.25	0.35	0.5	0.8
3			q_2 0.05	0.15	0.3	0.6
4				q_3 0.05	0.2	0.5
5					q_4 0.05	0.35
6						q_5 0.10

Then,
for, i
W (1, 0),
Then,
Here,

Then, i j
⇒ for W (1, 0)
Here,
i = 1
j = 0
i.e, j = i - 1
= 1 - 1
= 0

$\therefore w[i,j] = q_{i-1}$

$= q_{1-1}$

$= q_0$

$= 0.05$

$\Rightarrow$ for $w[2,1]$

$i = 2$

$j = 1$

i.e, $j = i-1$

$= 2-1$

$= 1$

$\therefore w[i,j] = q_{i-1}$

$= q_{2-1}$

$= q_1$

$= 0.10$

~~$\Rightarrow$ for~~

$\Rightarrow$ for $w[1,1]$,

$i = 1$

$j = 1$

~~i.e, $j = i-1$~~

~~$= 1-1$~~

~~$= 0$~~

~~$\therefore w[1,1] = q_{i-1}$~~

~~$= q_{1-1}$~~

~~$= q_0$~~

Here, $i = j = 1$

So,

$w[1,1] = w[1, 1-1] + p_1 + q_1$

$= w[1,0] + p_1 + q_1$

$= 0.05 + 0.15 + 0.10$

$= 0.30$

for, $w[2,2]$,

So,

$w[2,2] = w[2, 2-1] + p_2 + q_2$

$= 0.25$.

for Table 2 $e[i,j]$

e	j→ 0	1	2	3	4	5
i ↓		(r=1)	(r=1)			
1	0.05	0.45	0.9	1.25	1.75	2.75
			(r=2)			
2		0.10	0.4	0.7	1.2	2
				(r=3)		
3			0.05	0.25	0.6	1.3
					(r=4)	
4				0.05	0.3	0.9
						(r=5)
5					0.05	0.5
6						0.10

for, $e[1,0]$, $i=1$, $j=0$ | $j<i$

$\therefore e[1,0] = q_{j-1}$

$= q_{1-1}$

$= q_0$

$= 0.05$

for, $e[1,1]$, $i=1$, $j=1$ | $j=i=1$ so, $r=1$

$\therefore e[1,1] = e[i, r-1] + e[r+1, j] + w[i,j]$

$= e[1, 1-1] + e[1+1, 1] + w[1,1]$

$= e[1,0] + e[2,1] + 0.30$

$= 0.05 + 0.10 + 0.30$

$= 0.45$

[$e[2,1] = q_{2-1}$ $= q_1 = 0.10$]

for, $e[1,2]$, $i=1$, $j=2$, $r=1$ & $r=2$ Here, $r=1$ is minimum.

e[1,5] root, r = 1, 2, 3, 4, 5 (so we)

e :

DATE
PAGE No.

for, r = 1

$e[1,2] = e[1,0] + e[2,2] + w[1,2]$ {r=1}

$= 0.05 + 0.4 + 0.45$

$= 0.9$

for, r = 2,

$e[1,2] = e[1,1] + e[3,2] + w[1,2]$ {r=2}

$= 0.45 + 0.05 + 0.45$

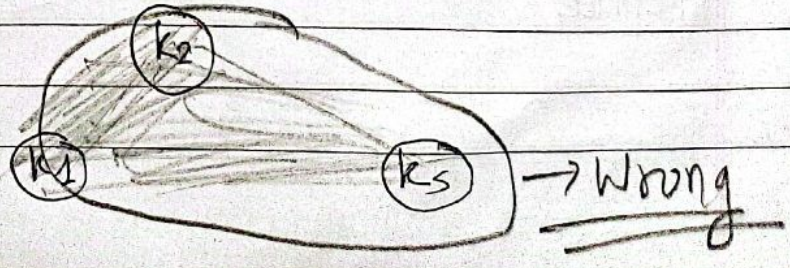
$= 0.95$

$\therefore e[1,2] = \min\{0.9, 0.95\}$

$= 0.9$ (r=1)

for Table 3, Root table,

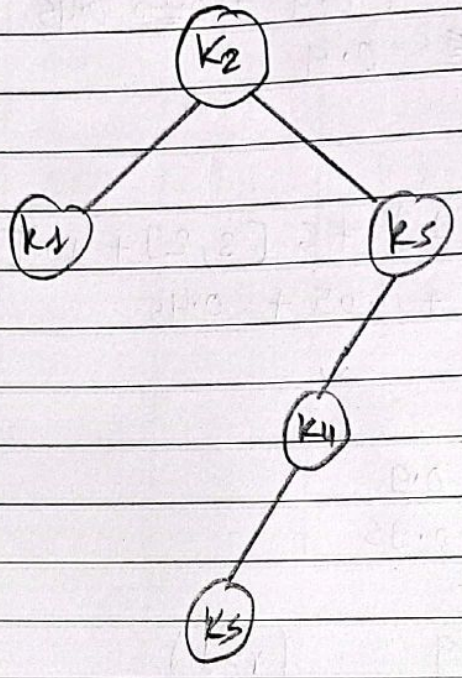
i ↓ \ e →	1	2	3	4	5
1	1	1	2	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5



→ Wrong

DATE
PAGE No.

Optimal Binary Search Tree that can be formed B:-



2078-09-12

String Editing :-

String 1 : aabab (source)

↓ edit

string 2 : babb (Destination)

Termindogies :-

Copy Ci
Remove (Delete)
Insert Ji
Replace

Technique 1:-

Cost

Remove all the characters within string 1. (5)
Insert the required strings (4)

9

DATE
PAGE No.

Teching

Technique 2:-

Remove Remove ↓ Insert

a a b a b b

(1) (1) (1)

Cost (3)

Technique 3:-

Replace Remove

a a b a b

b (1)

(2) Cost (3)

Technique 4:-

b Replace b Replace Remove

a a b a

2 2 1 cost (5)

j →		Null	b	a	b	b
	Null	0	1	2	3	4
i ↓	a	1	1	1 →	2 →	3
	a	2	2	1 →	2 →	3
	b	3	2 →	2 →	1 →	2
	a	4	3	2 →	2 →	2
	b	5	4 →	3	2 →	2

Character equal

Copy

Character unequal

Replace Remove

Insert

min { three +1

S1 S2 S1 S2

NULL → 4 → babb aabab → 5 → NULL

DATE
PAGE No.

$$\text{Cost}(i,j) = \begin{cases} 0 \\ \text{Cost}(i-1, 0) + D(x_i), & i>0, j=0 \\ \text{Cost}(0, j-1) + I(y_j), & i=0, j>0 \end{cases}$$

$$= \min \begin{cases} \text{cost}(i-1, j) + D(x_i) \\ \text{cost}(i, j-1) + I(y_j) \\ \text{cost}(i-1, j-1) + c(x_i, y_j) \end{cases} \quad i>0, j>0.$$

Page No.
Date

String Editing

We are given two string,

$X = x_1, x_2, \dots x_n$ & $Y = y_1, y_2, \dots y_m$,

where, x_i, $1 \le i \le n$ & y_j, $1 \le j \le m$ are members of a finite set of symbols known as alphabe

We want to transform X into Y using a sequence of edit operations on X.

The permissible operations are :-

a) Insert.
b) Delete
c) change/ convert
d) copy

There is cost associated with performing each.

The cost of the sequence operations is the sum of costs of the individual operations in the sequence. The problem of string editing is to identify the minimum cost sequence of edit operations that will transform X into Y.

$$\text{cost}(i,j) = \begin{cases} 0 & i=j=0 \\ \text{cost}(i-1,0) + D(x_i) & i>0 ;\ j=0 \\ \text{cost}(0,j-1) + I(y_j) & i=0,\ j>0 \end{cases}$$

OR

$$\text{cost}(i,j) = \min \begin{cases} \text{cost}(i-1,j) + D(x_i) & \\ \text{cost}(i,j-1) + I(y_j) & \\ \text{cost}(i-1,j-1) + C(x_i,y_i) & i>0,\ j>0 \end{cases}$$

String 1 :- a a b a b (source) 5 cost = 9.
String 2 :- b a b b (destination). 4

~~a a b a b~~

Technique 1 :- Remove all the characters within string 1
Insert the required string

a a b a b b a b b cost = 9
D D D D D I I I I

Technique 2 :-

First insert

a a b a b b cost → 3
D D Insert

Technique 3 :-

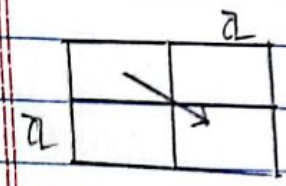
b 1
~~a~~ a b ~~a~~ b cost → 3
R D

Technique 4 :- b b D
~~a~~ a b ~~a~~ ~~b~~ cost → ~~5~~
R R

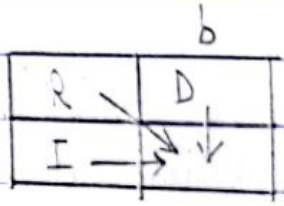
Technique 5 :-

a
~~a~~ a b a b cost → 2
b
D

Character equal → take diagonal element



$x_i = y_j$



copy (no cost)

		Null	b	a	b	b
	Null	0	1	2	3	4
	a	1	I → R 1 (0+1)	1	2 (1+1)	3 (2+1)
x° ↓	a	2	1 (1+1)	C 1	2 (1+1)	3 (2+1)
	b	3	2	2 (1+1)	C 1	2
	a	4	3 (2+1)	2	2 (1+1)	2 (1+1)
	b	5	4	3 (2+1)	2	C 2

Therefore cost = 2

copy = 3 , no cost

Insert = 1, cost 1

Delete = 1 cost 1

cost = 0

Unit-4: Mesh and Hypercube Algorithms

Page No.
Date

4.1. Mesh Algorithm

A mesh is an $a \times b$ grid in which there is a processor at each grid point. The edges correspond to communication links and are bidirectional. Each processor of the mesh can be labeled with a tuple (i, j) where $1 \leq i \leq a$ and $1 \leq j \leq b$.

Every processor of the mesh is a RAM with some local memory. Hence, each processor can perform any of the basic operations such as substraction, addition, multiplication, comparision, local memory access and so on. The computation is assumed to be synchronous.

We consider square ~~matrix~~ meshes, that is meshes for which $a = b$. A $\sqrt{p} \times \sqrt{p}$ ~~matr~~ mesh is shown in figure (a).

A closely related model is the linear array. A linear array consists of p processors ~~p and~~ named $(1, 2, \ldots, p)$ connected as follows :-

a) processor i is connected to the processors $i-1$ & $i+1$ for $2 \leq i \leq p-1$.

b) processor 1 is connected to processor p_2 and processor p is connected to processor $p-1$.

processor 1 and p are known as boundary processors.
procesor $i-1$ and $i-j$ are known as left neighbour (right neighaur) of i.

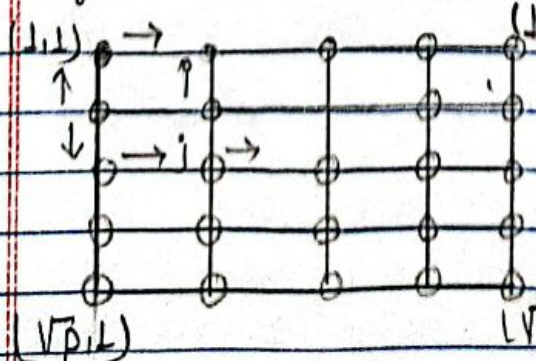


fig (a) Mesh

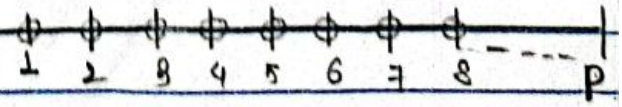
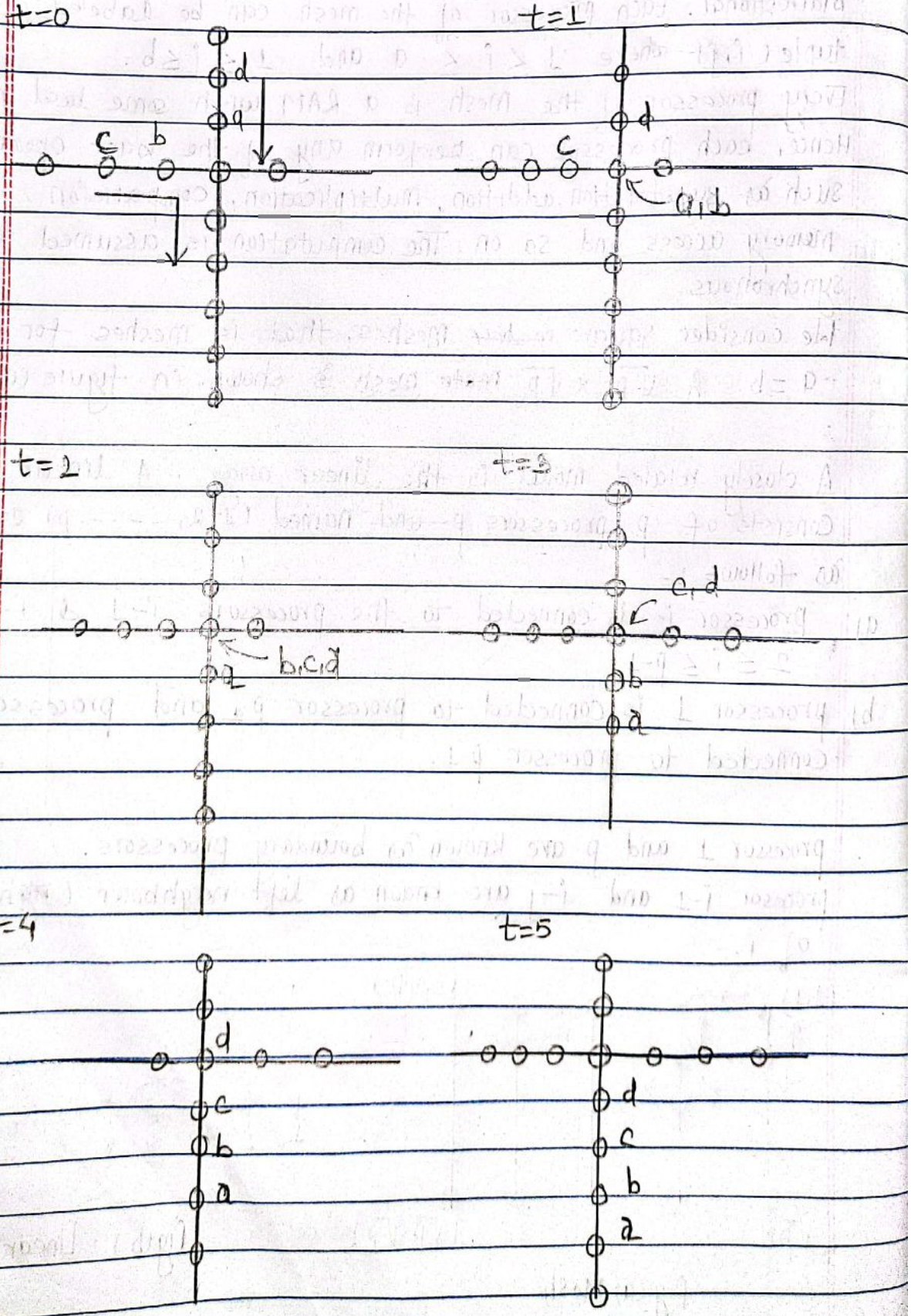


fig (b) : Linear array

Page No.
Date

Consider 4 packets a, b, c and d. and their final destination are shown in figure below :-



Page No.
Date

Since it needs p-1 steps to complete the linear array., the worst case of linear array is $O(p-1)$.

Broadcasting in mesh

In the case of a $\sqrt{p} \times \sqrt{p}$ mesh broadcasting can be done in two phases :-

1. If (i, j) is the processor of message origin in phase 1, M could be broadcast to all the processors in row i.
2. In phase 2, broadcasting of m is done in each column.

This algorithm takes $\leq 2(\sqrt{p}-1$ steps) which is expressed as :-
The broadcasting on a p= processor linear array can be completed in p steps or less. On a $\sqrt{p} \times \sqrt{p}$ mesh, the same can be performed in $\leq 2(\sqrt{p}-1) \leq O(\sqrt{p})$ time.

Broadcasting can be done in the following ways :-

a) Broadcasting without ~~redundancy~~ constraints (Redundancy)
b) Broadcasting with constraints (No redundancy)

Prefix computation :-

Consider a $\sqrt{p} \times \sqrt{p}$ mesh. The problem of computing prefix sums on the mesh can be reduced to three phases in ~~which~~ each of ~~the~~ which computation is local to the individual rows. and columns. This algorithm assumes the row major indexing scheme.
The prefix computation on mesh can be done in following steps :-

1. Step 1 :- Prefix computation Row-wise.
   Row i (for $i = 1, 2, \ldots \sqrt{p}$) computes the prefixes of its p elements.

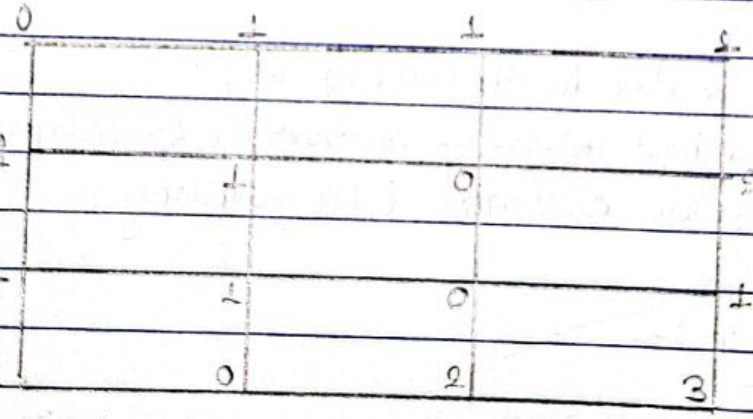
2. Step 2 :- prefix computation (column-wise), last column only.
only $\sqrt{p}$ column computes the prefix of sums computed in step 1.

3. Step 3 :- Shift 1 down to column wise, last column only & broadcast in its respective row (column-wise) & perform prefix computation.

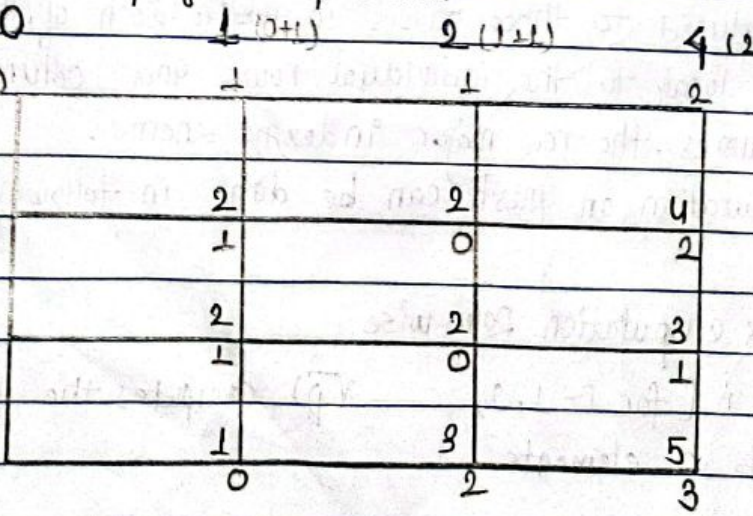
Shifting :- $(i, \sqrt{p})$ Shifting $(i+1, \sqrt{p})$

Broadcast :- Broadcast (i, p) in row $i+1$ (for $i = 1, 2, \ldots \sqrt{p}-1$)
Node j and in row i updates final results.

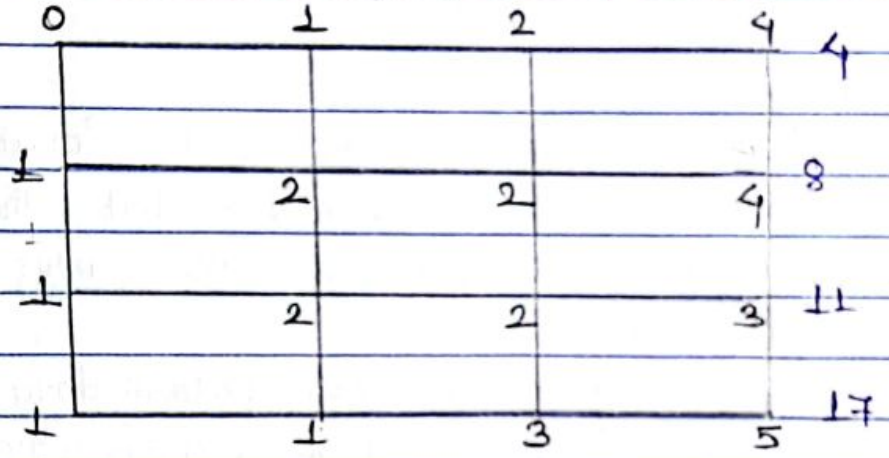
Consider the data on the 3x3 mesh and the problem of prefix sums under the row major indexing scheme.



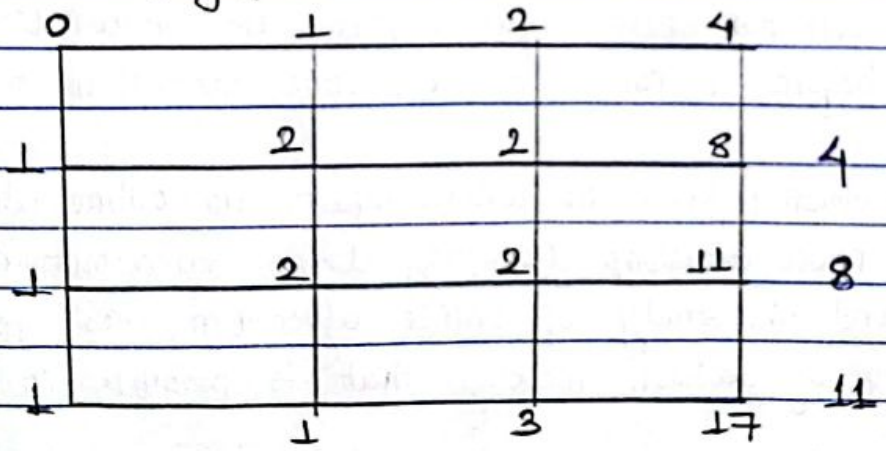
Step 1 :- Row wise prefix computation



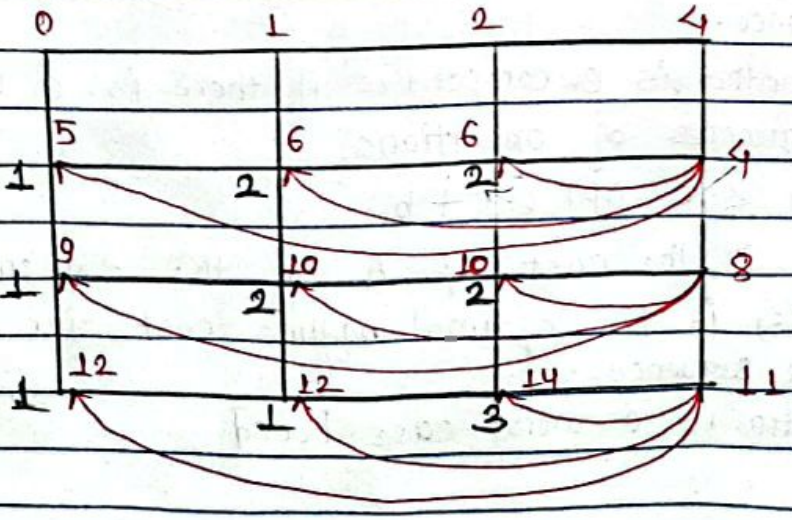
Step 2 :- column wise prefix computation. last column only



Step 3 :- shifting 1 down column wise . last column only.



After broadcasting in respective row,



Hence, This is the final prefix computation.

The worst case complexity analysis for prefix computation is $O(2\sqrt{p}-1)$

Page No.
Date

Unit 3 :- Online and PRAM Algorithms

3.1. Online Algorithms

Online algorithm is one that can process its input piece by piece in a serial fashion. ie. in the order that the input is fed to the algorithm, without having the entire input available from the start.

An offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand.

Example :- Selection short requires that the entire list be given before it can sort it, while insertion sort doesn't.

Because it doesnot know the whole input, an online algorithm is forced to make decisions that may later turn out not to be optimal, and the study of online algorithm has focused on the quality of decision making that is possible in this setting.

Competitive analysis formalizes this idea by comparing the relat performance of an online & offline algorithm for the same problem instance.

An online algorithm is c-competitive if there is a constant b for all sequences of operations.

$$A(s) < c\ OPT(s) + b$$

where A(s) is the cost of A on the sequence s.

OPT(s) is the optimal offline cost for the same sequence.

Competitive ratio is a worst case bound.

Page No.
Date

Ski-rental problem

Assume that you are taking ski. lessons. After each lesson you decide (depending on how much you enjoy it, and what is your bones status) whether to continue to ski or stop totally. You have the choice of either renting skis for 1 $ a time or buying skis for y$. will you buy or rent?

If you know in advance how many times you would ski in your life then the choice of whether to rent or buy is simple. If you will ski more than y times then buy before you start, otherwise, always rent.

The cost of this algorithm is min (t, y).

This type of strategy with perfect knowledge of the future is known as offline strategy.

In practice, you don't know how many times you will ski, what should you do?

An online algorithm will be a number K such that after renting K-1 times you will buy skis (just before your K^{th} visit)

claim :- Setting K = y guarantees that you never pay more than twice the cost of the offline strategy.

Example :-

Assume, y = 7 $. Thus after 6 rents, you buy. your total payment is, 6+7 = 13 $

Theorem :-

Setting K = y guarantees that you never pay more than twice twice the cost of the offline strategy.

Proof :- When you buy ski's in your K^{th} visit, even if you quit right after this time, t >= y

→ your total payment is $k-1+y = y-1+y = 2y-1$

→ The offline cost is $\min(t, y) = y$

→ The ratio is $(2y-1)/y = 2-(1/y)$

Therefore, we say that this strategy is $(2-1/y)$ competitive.

Conclusion or General rule.

When balancing small incremental costs a big one time cost, you want to delay spending the big cost until you have accumulated roughly the same amount in small costs.

Page No.
Date

Prefix computation Algorithm :-

Let S be the domain in which binary associative operator $\oplus$ is defined.

An operator is binary associative if for any of three elements x, y, z from S, $((x \oplus y) \oplus z) = (x \oplus (y \oplus z))$ i.e. the order in which the operation $\oplus$ performed doesn't matter. [$\oplus$ can be $+, -, /, \max, \min, \text{avg}$]

The prefix computation problem on S has an input n elements from S, say $x_1, x_2, x_3, \ldots\ldots x_n$.

The problem is to compute the n elements

$$x_1, x_1 \oplus x_2, \ldots\ldots x_1 \oplus x_2 \oplus x_3 \cdot \oplus, \ldots\ldots \oplus x_n.$$

The output elements are referred to as prefixes.

Example :- Let S be the set of integers and $\oplus$ be the usual addition operation.

Input = 3, -5, 8, 2, 5, 4.

The output is 3, -2, 6, 8, 13, 17.

Example

Let $n = 8$ and $p = 8$.

Let the input to the prefix computation problem be, 12, 3, 6, 8, 11, 4, 5, 7 & let $\oplus$ be addition.

Step 1 :- processor P_1 to P_4 computes the prefix sum of 12, 3, 6, 8 to arrive at 12, 15, 21, 29.

And,

processor P_5 to P_8 computes the prefix sum of 11, 4, 5, 7 to arrive at 11, 15, 20, 27.

Step 2 :- processor P_1 to P_4 sits idle
processor P_5 to P_8 will update their results by adding 29 to prefix to obtain 40, 44, 49, 56.

Step 1 :-

12	12,3	12,3,6	12,3,6,8	11	11,4	11,4,5	11,4,57

12	12,15	12,15,21	12,15,21,29	11	11,15	11,15,20

11,15,20,27

12,15,21,29	11,15,20,27

Step 2 :-

12,15,21,29,	40,44,49,56

Analysis

Step 1 takes $T(n/2)$

Step 2 takes $O(1)$.

Recurrance relation is,

$$T(n) = T(n/2) + O(1), \quad T(1) = 1$$

This solves to,

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