The remaining frames are stored in secondary Memory (Virtual).

#F	#P
1	2
2	5
3	
4	

3 is not present
So, 2 is sent to swap and 3 is given page 5
Then after 3 is completed 3 is sent to swap and 4 is given page 5.

Whenever page fault occurs needs to swap from swap space
- Since we are accessing the HD. it will be slow. So try to go with a cache in b/w.

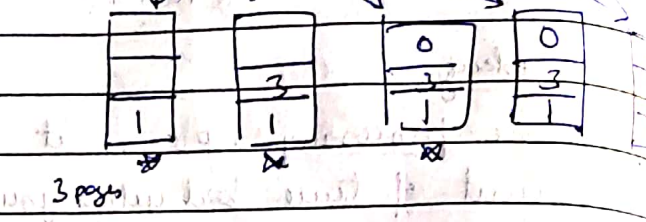
When cache is used it's called TLB When loop is broken down to frames.

say 1 and 2 then you will be constantly swapping 1 & 2.
- Why we are swapping and not fetching?
  - During execution there may be a case if the frame in RAM gets changed so if we would have only fetched then the updated frame would have been lost.

- Caches stores the Most Recently used frames and swaps out the LRU into the HD.

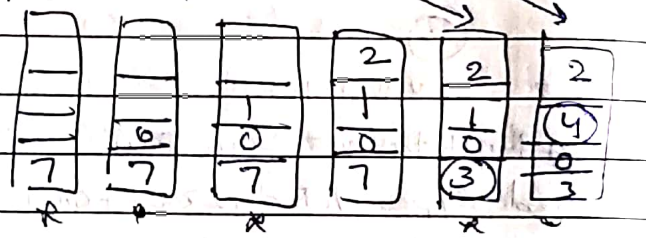
Page Replacement Algorithms

Order. 1 3 0 3 5 6 3



✗ → page fault
if FIFO used then 5 will replace 1.
if LRU used then 1 would be
For LRU

7 0 1 2 0 3 0 4 2 3 0 3 2



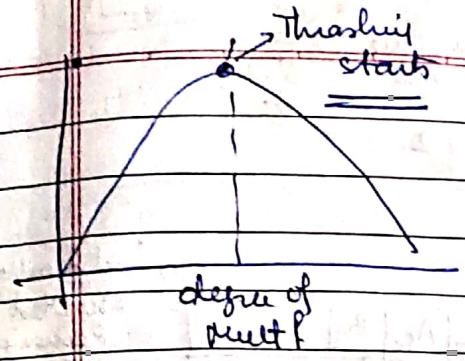
Replace the page that was used most time ago.

- It would seem that by increasing the frame size in FIFO → Faults will decrease. But this doesn't happen always
Belady Anomaly

Optimal Algorithm [predicts future pages]
↳ But not practical.
Second chance Algorithm.

Thrashing
when the no. of processes are very high → then needs continuous swapping with the HDD.
which further makes utilization of Resources weak

also high degree of Multiprogramming leads to Thrashing.



Test submit →

Name → Testname → Task 1 → Task 2

— — — —

LeetCode Weekly

0 0 0 0 0

10 -10

20 -20

25

10

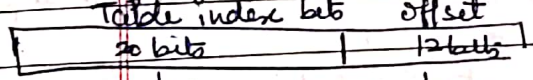
1

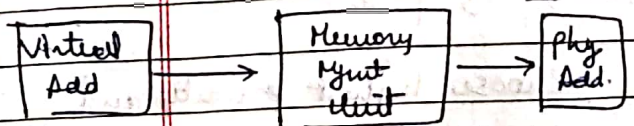
2 3

4 5 6 7

Operating Systems - 2

continue demand Paging.





MMU maps the Virtual to physical

Date
Page

Tue 20th

• Virtual address is created at Runtime ??
↳ where it is stored ??

Since Process Page Table require 4MB contiguous memory but if we don't find it. We can break it down into parts and do similar to how we broke into frames.

This is called 2-level demand Paging or Multi-level demand Paging ?
↳ Read about this ??

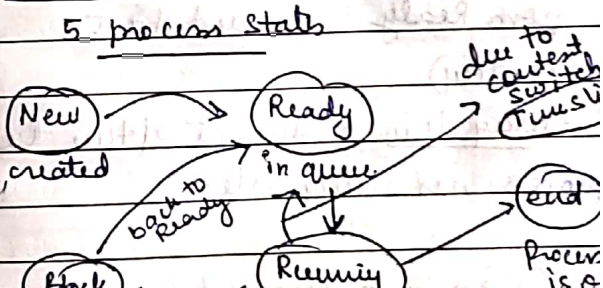
• Job Scheduling

① Non-preemptive → no interrupt is taking consideration

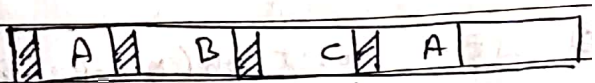
② Preemptive → interrupts are consi

Process States

5 process states



Types ① Long Scheduler
② Medium Scheduler
③ Short Scheduler
} look upon



↑ due to context switch
this time is taken by loading and deloading d process

1) CPU Utilisation
2) Throughput
3) Turn Around Time
4) Response Time
5) Waiting Time

① CPU util → if there is no context switch CPU util increases (Sitting Idle)

② Throughput → amount of process executed over a given interval of time (i.e that's why we give priority to shorter job so that throughput increases).

③ TAT → the moment it starts processing till the moment it's completely executed.

④ Response Time → The amount of time the process changes from ready to running (Time to take it out from Ready Queue and start executing)

⑤ Waiting Time → Amount of time the process was sitting idle.

① FCFS → First come First Serve.

A	B	C	D	E

A complete it no context switch (Non-preemptive). then B the C ...

- each process gets turn to execute
- Waiting time may be more.

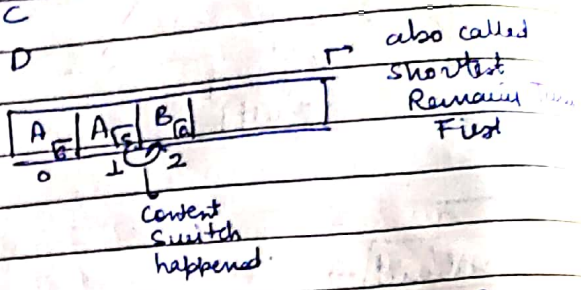
② SJF → Shortest Job (Non-preempt).

A	B	C	D
5	7	1	12

Sort by their burst time.
FCFS in sorted order.

Preemptive SJF

	Arr	Burst
A	0	7
B	2	
C		
D		



also called Shortest Remaining Time First

But practically it's hard to implement SJF because beforehand we can't predict the Burst Time of a Process.

- A may lead to starve because shorter process keep coming.

SJF is not used anywhere.
↳ gives Minimum Average Waiting Time

- Round Robin Scheduling.

give each process a quanta time

- Fair
- No starvation
- Better Throughput

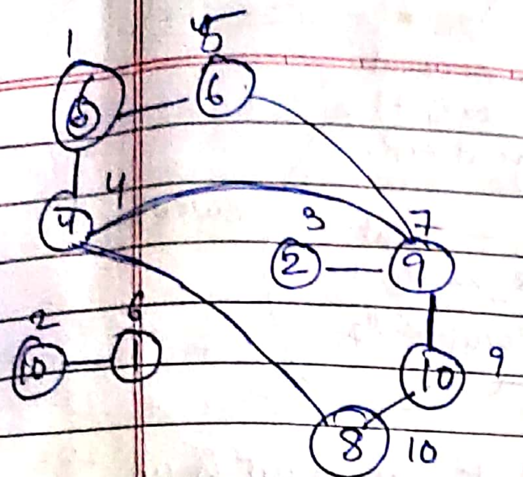
↳ choosing a better quanta is imp

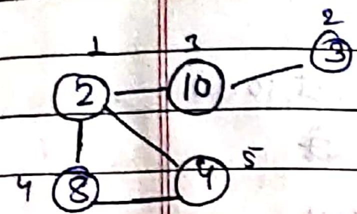
- Priority Scheduling :→

	Arr	Pro
A	5	3
B	2	1 →
C	3	2

- Choose highest priority and execute them
- starvation

$1+3+2+3+3+3$





$x_3 \rightarrow B$

no 2 cons → x_1

no 2 cons → x_2

$x_1\ x_2\ x_1\ x_2\ x$ . . .

$x_2\ x_1\ x_2$ . . . A → 2

$x_1\ x_3\ x_2$ — only $x_1\ x_2$ → 2 if x_3 cons.

$x_1\ x_2\ x_1$ 1

$x_2\ x_1\ x_2$ 1 x_3 — — — → 2, 4

$x_1\ x_2\ x_3$

$x_1\ x_3\ x_2$

$x_2\ x_1\ x_3$

$x_2\ x_3\ x_1$

$x_3\ x_2\ x_1$

$x_3\ x_2\ x_1$

1 → 4P 3PP

11 → 3 22 → 7

21 → 6 32 → 16 33 → 17

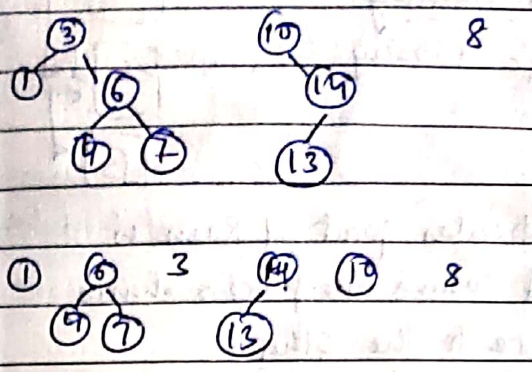
31 → 10 42 → 32 43 → 40

41 → 11 52 → 56 53 → 88

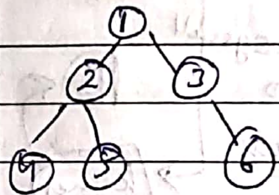
51 → 16 62 → 88 63 → 176

61 → 22 72 → 128 73 → 320

(Q) Max diff in Node & Ancestor.



1 4 7 6 3 13 14 10 8



123
456
579 /2 /2 /2 = 2+3+4

4 6 5
2 3 1
6 9 6
3 4 3

1 2 3
4 6 5

1 2 4
5 6 3
6 8 7
/2 /2 /2
3 4 3

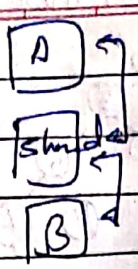
OS-2 continue

IPC → Interprocess Communication

- Convoy Effect → due to a large process other have to wait for a longer time ∴ SJF shows least convoy effect.

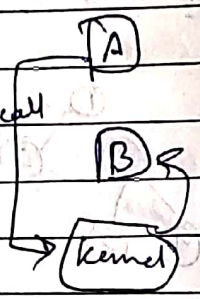
• If two process wants to interact with each other. It must be mutual from both processes.

1) Shared Memory
2) Message Passing

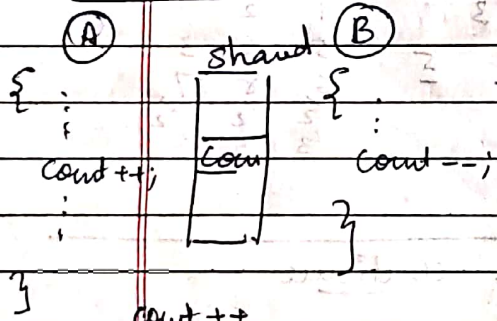


1. • a particular part of Ram is shared where one process shares the data to the other. Everything take place in RAM.

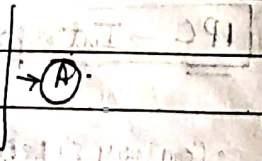
2. A sends data to the Kernel via a syscall then Kernel sends it to the desired process again by syscall. (Talking to the Kernel).



Synchronization.

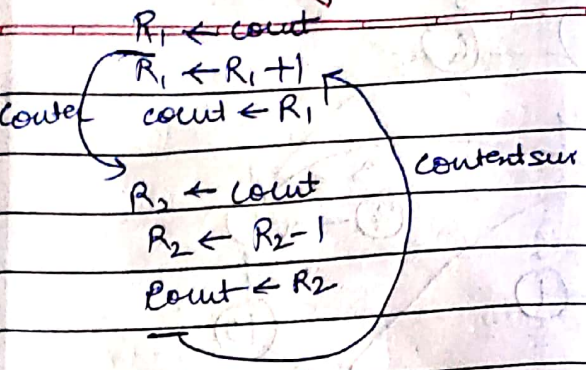


count++
⇒ R1 ← count
R1 ← R1(+)
count ← R1



if both A and B had been processed sequentially:
count = 0 (if only once)

• however say there was a context switch after a time slice and before completing A. B is completed then A.
then count can be random.

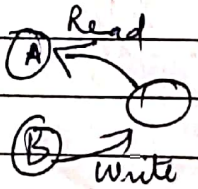


So what we want is no Context switch until A/B is finished →

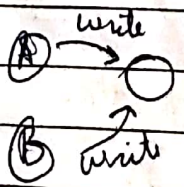
• you can't ask not to do context switch.

• Atomic operation
set of instructions which when interrupted should be redone from the beginning. (Either do it all or do again from start)

• Race conditions
if two processes are competing for same shared mem, whoever reaches 1st get to update.
WR Race WW Race.



1) A read first got 0 but B wrote 1 so, A thinks 0 but altered.



• A writes 1 then B writes 2. A thinks 1 but actually it's 2.

Take out Race conditions by applying concepts.

• locking concept.
lock the shared Memory then do your job.

one turn variable

turn =

Critical Section → The shared memory where every process is racing for.

* Critical Section →
① Mutual Exclusion
② Progress
③ Bounded Wait
} Must hold for CS to execute

① only 1 process can enter CS at a time.
② if there is no one inside CS, then the process which wants to enter CS must be allowed to do so.
③ any process which is waiting to enter inside CS must wait for a finite time. (It mustn't wait forever).

A
{
lock()
count++
unlock()
}

B
{
lock()
count--;
unlock();
}

→ CS

Context switch will happen, it can't be stopped. But when B starts the lock will not allow B to use count as A has already placed a lock.
Now ans = 0 everytime.

(A)
while(1){
while(turn==2)
{
}
//CS
turn=2
}

(B)
while(1){
while(turn==1)
{
}
//CS
turn=1
}

this inf loop does allow A to change turn because B is currently busy

this inf loop don't allow B to change turn because A is in CS

1. Mutual Excl ✓
2. Progress X →
A must complete itself to let B enter into CS. but what if A doesn't want to complete? so, Progress is not guaranteed.
A → B, B depends on A

How to remove the dependency?

a = false
b = false.

(A)
while(1){
while(b==true)
{
}
a=true
//CS
a=false
}

(B)
while(1){
while(a==true)
{
}
b=true
//CS
b=false
}

context.

① Mutual Excl X
both can execute CS together. Murphy's Law →
"if there's something wrong then it will go wrong sometime or the other".

want_a = false
want_b = false

Date
Page

(A)

```
while (1) {
want_a = true;
while (want_b == true)
{
}
}
// CS
want_a = false
}
```

(B)

```
while (1) {
want_b = true
while (want_a == true)
{
}
}
// CS
want_b = false
}
```

again no progress.
The above code can go into infinite loop
So both wants CS but no one is getting.

add a favor var which will fix all the problems.

(A)

```
want_a = true;
favor = 2
while (want_b == true
&& favor == 2)
{
}
}
// CS
want_a = false
```

(B)

```
want_b = true
favor = 1
while (want_a == true
&& favor == 1)
{
}
}
// CS
want_b = false
```

favor variable tells that we are favouring 1/2 process to enter.

Peterson Algorithm

MUTEX = Mutual Exclusion

mex → shared var.

(A)

```
lock (mex)
// CS
unlock (Mex)
```

(B)

```
lock (mex)
// CS
unlock (Mex).
```

```
lock () {
if mex == 0
mex = 1
else while (1) {
if (mex == 0)
break
}
}
```

If some p is using CS. Others will be in inf loop.

```
unlock () {
mex = 0;
}
```

optimization → instead of while loop use sleep()

```
lock () {
if (mex == 0)
mex = 1
else sleep ().
}
```

```
unlock
mex = 0
unlock ()
}
```

all the sleeping p^n are woken up and pushed to queue.

- Mutex can be unlocked by only that process which locked it. unlike semaphores.

* smallest K^{th} distance pair.

①

Leet → Dungeon Game →

l h mid
0 25 12

l h m
0 12 6 ✓

l h m
0 5 2 ✓

[-2 -3 3 2]
[-5 -10 1 5]
[10 30 -5 6]
-14 -24

4 → 1 → 4 → 5 →

l h m
0 4 2
3 4 3
4 4 4

[-3 5]

1 -3 3 1.3
3 0.6 -2 0.4 1 → -3
-3 10 -3 -3 4 1 →

90° in bottom up manner and remember never to drop below 1. (Dungeon Game)

(Q) Find minimum in Rotated array in case of Repeated elements → ??

in worst case O(lgn) methods will take O(N) time.
my simple O(N) leetcode soln beats 100%.

~~question~~

• question pallete

• Date Page

• ~~Inter & Subtract~~

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

sum = 13 6 d=3
sum = 5 7 d=3
sum = 5 15 d=3
sum = 12 8 6

P l r
0 0

sum = 0
1 → 0 1
3 → 0 2
6 → 0 3
d = 3
5 → 1 3
9 → 1 4
d = 3
7 → 2 4
d = 2
4 → 3 4
6 → 3 5
d = 2
2 → 4 5
12 → 4 6
d = 2
10 → 5 6
d = 1
8 6 66

OS-3

Operating System - 3

if lock is not used then race conditions would occur.

(A)
{
lock(mem);
count++;
unlock(mem);
}

(B)
{
lock(mem);
count--;
unlock(mem);
}

Atomic process / lock & unlock

lock(mem){
if(mem==0)
mem=1
else{while(mem==1)
{
}
mem=1

unlock(mem)
{ mem=0;
}

- Instead of infinite loop make the process sleep so the processor doesn't waste on infloop.
  Sleep → block state

```
lock(mem)
{ if (mem==0)
    mem=1
  else { sleep();
         mem=1;
       }
}
```

```
unlock(mem)
{ mem=0;
  wakeup();
}
```

all process that were slept are woken and sent to ready queue.

infloop → busy waiting
the processor is busy but doing an unproductive job

Mutex have ownership. The process which has locked can only unlock it.

Semaphores

1) Binary Sema
2) Counting Sema.

Sema is more powerful. → works on signalling
Sema has not ownership.

lock → mex 0→1
unlock → mex 1→0

- atomic operations are very small and fast. So that roll back time is very less.

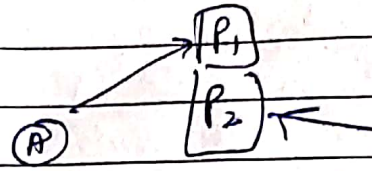
lock() → wait() → P()
unlock() → signal() → V()

```
wait(int *s)
{ while (*s<=0)
  { }
  *s--;
}
```

```
signal(int *s)
{ *s++;
}
```

If s is neg then don't do anything

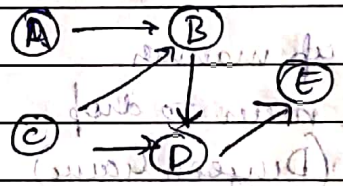
② Counting Semaphores
when there are multiple Shared Resources
Eg. Two Prin



S=2 at A
S=1 at B

wait
C → S=0 at C
so wait()

How can perform a simulation of Topological Sort using Semaphores?



context switch will keep taking place.

So we will do signal from each dependent process.
Total Sema used would be no. of edges → depends on

A	B	C	D	E
sig(S1)	wait(S1)	sig(S2)	wait(S2)	wait(S4)
	wait(S3)		wait(S3)	
	sig(S3)		sig(S4)	

wait infinite

if there would have been a cycle then it would lead to Deadlock.

Slight mistake ?? where write values of S_1, S_2 and S_3 you'll find S_3 will wait inf.

So to fix this issue, just signal all those semaphores which you were waiting.

Modified

A	B	C	D	E
sig(S_1)	wait(S_1) wait(S_2) : sig(S_3) sig(S_1) sig(S_2)	sig(S_2)	wait(S_3) wait(S_2) : sig(S_4) sig(S_3) sig(S_2)	wait(S_4) : sig(S_4)

Producer Consumer Problem

counting Semaphore to keep track of full Buffer and empty buffer

producer → wait

consumer → wait

count → shared

full, empty, mutex

```
produce(){
while(1){
item ← produce()
if(count==N)
  sleep(full)
lock(mutex)
count++;
unlock(mutex);
if(count==1)
  wakeup(empty);
}
}
```

```
consume(){
if(count==0)
  sleep(empty)
lock(mutex)
count--;
unlock(mutex);
if(count==N-1)
  wakeup(full)
}.
```

→ make these atomic

- if producer sleeps the full → consumer wakes it.

and vice versa.

can we face problem if ContextSwitch?

→ make the marked parts as atomic.

We can do this using Semaphores

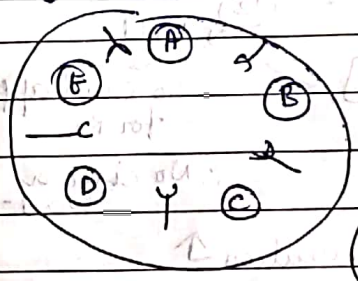
```
produce{
while(1)
item ← produce
wait(empty);
lock(mutex)
count++;
unlock(mutex);
signal(full);
}
```

```
consume(){
while(1){
wait(full)
lock(mutex)
count--;
unlock(mutex);
signal(empty);
}.
```

Deadlock

↳ Stuck
Nothing can be done

- Dining Philosophers

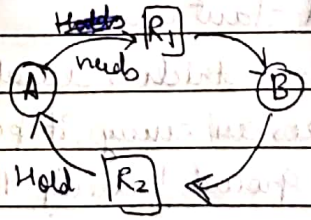


① Think
② Eat.

2 forks needed to eat

- in a greedy state → deadlock

- Resource Allocation graphs



→ deadlock, no one can progress

Conditions for Deadlock

all

① Mutual Exclusion → only 1 process can use a R_1

② Hold & Wait → P holding R_1 and needs R_2.

③ No-preemption
↳ You can't take away a Resource from a Process.

④ Circular Wait → P ⇄ Q

So to prevent deadlock avoid any one of the following →

① Can't Remove Mutual Exclusion because you can't allow multiple P to hold same R.

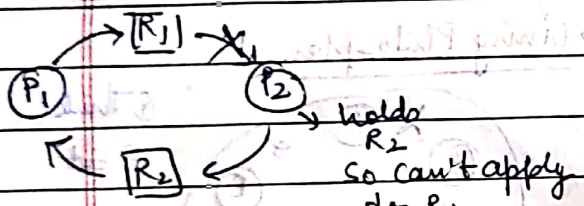
② Hold & wait → eg if a process needs R_1 then execute that P by first giving R to it.

③ · include preemption and forcefully give the holded Resource to another P.

④ to avoid circular wait do a hierarchial ordering.

P_1 holds R_1 → then it can only apply for $R_2, R_3 \ldots R_N$.

P_2 holds R_2 → then it can only apply for $R_3\ R_4 \ldots R_N$.



∴ No circular wait.

Deadlock Prevention ↑

* Deadlock avoidance

basically normal systems follows Ostrich Algo → Restarts the system and start again.

Why ostrich ? → Ostrich is dumb. whenever it sees an enemy it puts head in the ground believing that if it can't see the enemy, the enemy won't see it either. Haha!!

But important systems like Banking, Aeroplane etc. should never ever allow deadlock to occur.

Date
Page

• Detection of Deadlock

make a Resource Allocation graph and find whether there exists a cycle or not.

* Banker's Algorithm
↳ deadlock Avoidance Algo

try not to enter the unsafe one. Try to skip it. Read Mej

Deadlock Recovery

① Preemption → snatch the Resource
② Kill the process → as name say
③ Roll Back.
↳ keep saving the state of the process in HD and whenever a deadlock occurs, load that part where no deadlock and then do small changes so that same timeline doesn't repeat.

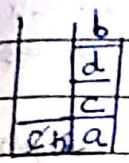
Reader & Writer Problem

only 1 writer can write
↳ use mutex.
and multiple Readers can Read → use Semaphore.
also, you can't write if someone is Reading.
Read Mej

• Remove duplicate letters – Hard Greedy.

c b a c d c b c.

c → 432
b → 21
a → 10
d → 1



7 2 5 10 8

10 32 mid
1 h 21

LINUX, GIT and CLOUD

Product Engineer

'Product Manager + Software Engg' = Product Engg

• Work with ambiguous problem statement • after applying the correct algorithms, measures the change in performance • Never blames the user instead finds what can be done better from product's side

• Backend Engineer

• works with well defined problems.
• works on optimising systems

1) Week 0 → Tools + Framework
• Git • Linux and Bash
• AWS • Flask • React

2) Week - 1 → Team
→ Project
→ API
→ Front End.

3) Week - 2 → Consume Frontend
→ Deploy

4) Week - 3 → Improvements

2:30 1.5 → 1
1 → 1/1.5

Date
Page 150 10 x 150/1.5 = 1:40

GUI Terminal

Services
OS
Hardware

• When you click a folder then GUI takes control and calls a service who (Show contents). Services do a specific job, service → OS → Hardware (Harddisk)
GUI

• You want to play songs before going to sleep and then after 30 min the PC should shutdown.

• Everything is File even dir's

• the commands does one thing well. cat → shows the content of file
, ls → list . ls -l (long list) ← flag

• touch 1.txt → creates the file.

• rm -rf ← flag
↳ force and everything inside

echo "____" > todo.txt

Task → segregate files on years they are created?

a basic simple
for i in 'ls'; do echo $i; done

var = "2018-05-01"
echo $var | cut -d '-' -f 1 ← flag
2
3

-d is delimiter, back ticks

years = $(for i in 'ls'; do echo $i | cut -d '-' -f 1; done);

for year in `echo $years`;
do
    mkdir $year;
done.

bash ⓪ —.sh

for img in `ls *.png`;
do
    year = `echo $img | cut -d '-' -f 1`
    mv $img $year/
done.

Task 2. sleep for after playing music.

Task 3 → download all wallpapers from a site.

1:33 resume
1:41

- git init
- git add.

git log -n 5 → who commited what.
with version

if multiple are working → clone to other's system and they can push.

Fork → copy to another remote
GitHub feature

How to pull?

- git reset --hard
  ↳ Version No.
  ↳ roll back to a previous version.

Advanced GIT and MVC

Date
Page

Task: Calculator in python
- add - subtract - mult - div
Two guys are implementing

How to merge these two files
① copy paste → humanaen
↳ resides in laptop
② Drive
③ docs → have version control
↳ collaborative

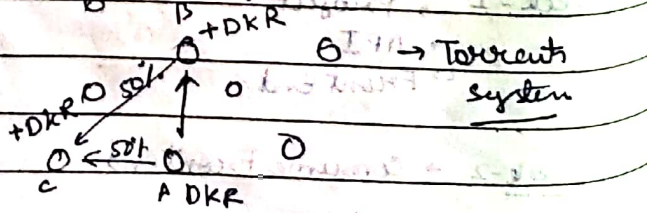
- if you are not sure about your code and needs review by other.
create a branch and push in that branch. In this case the master branch remains stable.

git checkout

merge conflict??
↳ collaborative tool
+ behavioral.
It encourages TeamWork

- Video + Flask
- Hosting
- 8 Questions (Med + Hard)
- create a to-do app in Flask

Peer to Peer Model
↳ many clients/server



View ← todo_view → gets a list → print in HTML
Model ← get_todos_by_name → returns todos for each
Controller ← todos → calls the above function accordingly

- Universal Id
- auto submit-
- server error after closing

- 6uvonths

- Coin change using dyp.

```
dp[n+1] = INT MAX
dp[0] = 0.
for (i=1; i<=V; i++)        → for each value till sum
    for (j=0; j< sq.size; j++)   → each denom.
        if (coin[j] <= i)
            dp[i] = min(dp[i-coin(j)]+1, dp(i))
```

using dp

using bfs

2, 4, 7 V=8

q → 8, 6, 4, 3, 4, 2, 2, 0, -3, 0, -2, -5

d = 1, 2

V=13

den = 1, 4, 9

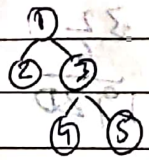
q → 13, 12, 9, 4, 11, 8, 3, 8, 5, 0, 3, 0

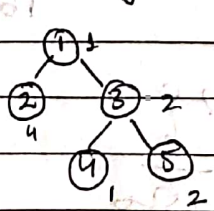
d → 1, 2

Model → db staff, business Logic

View → Representation

Controller → Request parsing & connecting View & Model.





2 → 1+1
3 → 1+1
4 → 1+1
4 → 2

bfs

1 → 2, 3
2
3 → 4, 5
4

2 → 1, 2
1 → 1
4 → 2
2 → 3

Date
Page

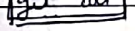
a) Create a folder → quiz-ielts

~~1) Heroku~~
1) git init → install git
2) heroku app create
3) add files
4) Cleardb MySQL.
5) add credit card.
6) ~~get DB url~~ add DB to ~~it~~ → it will be mine

cd

create app.

git init



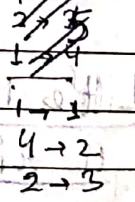
index.php
line 16 and line 17
check login
line 22, line 38, line 47,
line 71, line 110, line 132
line 146.
in test2.php
line 1184, line 46
logout.php
line 9, line 11

---

1 1 3 3

1 → 2, 3
2 →
3 → 4, 5
1 2 3 4 5
1 4 2 1 2

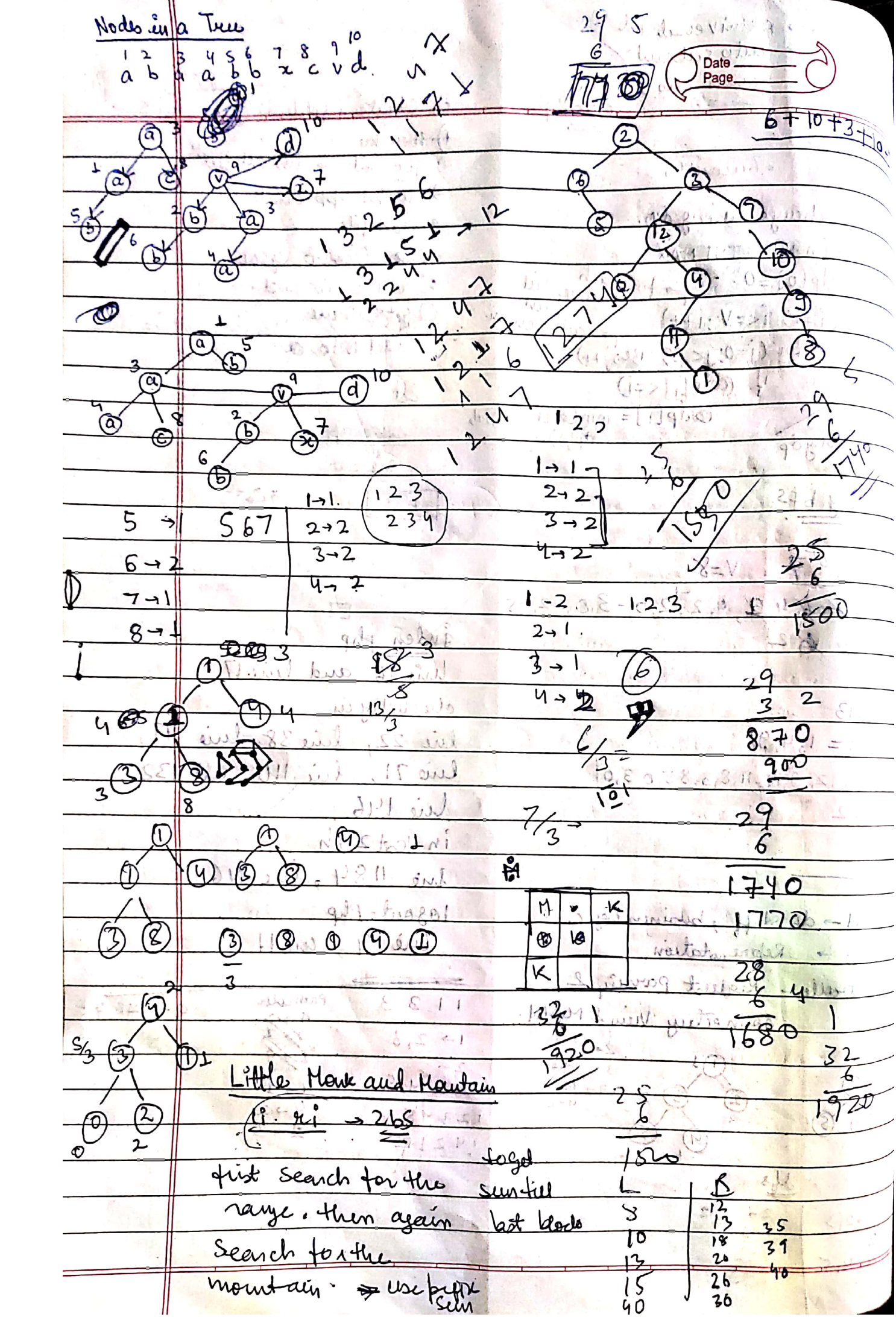
Path Color



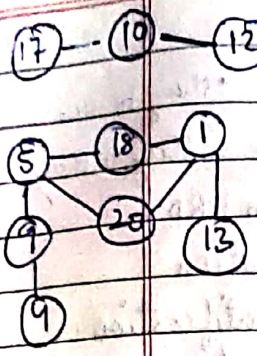
1 → 1
4 → 2
2 → 3

q → 1, 2, 3, 4, 5

Nodes in a Tree



1. Graph questions
2. DBMS-I, II
3. quiz



5 → 101
1 → 101
001 → 100
100

2 → 101
010 → 7
111 ✓

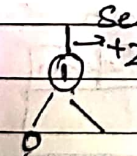
6 110
010
100

6 010
001 6+1=7
111 6^1=7

3 1 1 1 2

To count edges in an undirected graph → use Handshaking lemma.
- sum all the adj sizes and return sum/2.
  - ↳ because every edge is added twice.

- count self loops separately and size of adj list separately
  total edges in a connected comp = self + sum/2.



k=3

12
13
23
25

1 2 3 5 11 15 25
+1 +1 +1

11 5 2 15 25 1 3
k=3

1 2 3 5 11 15 25

(1,2) ✓ (1,2) → 2×3/2 = 3
(2,5) ✓
(2,5) → 3

12
13
23
25
35
+7

Date Page

x^2/M

M) x^2 (q
4 − Mq
R = N

$x - Mq = N$

1 → (1−1) + (1−7) → 6
2 → (1−2) + (2−7) → 6
} any

1 1 7 4 $x^2 = N + M\frac{x^2}{M}$

1 → (1−1) + (7−4) = 3
7 → (7−7) + (7−4) = 9 ✓

1 1 7 4

1 → (7−1) = 6 ✓
4 → 7−4 = 3

1 8 9

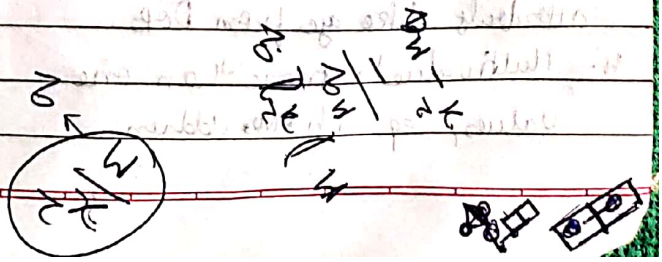
1 8 9

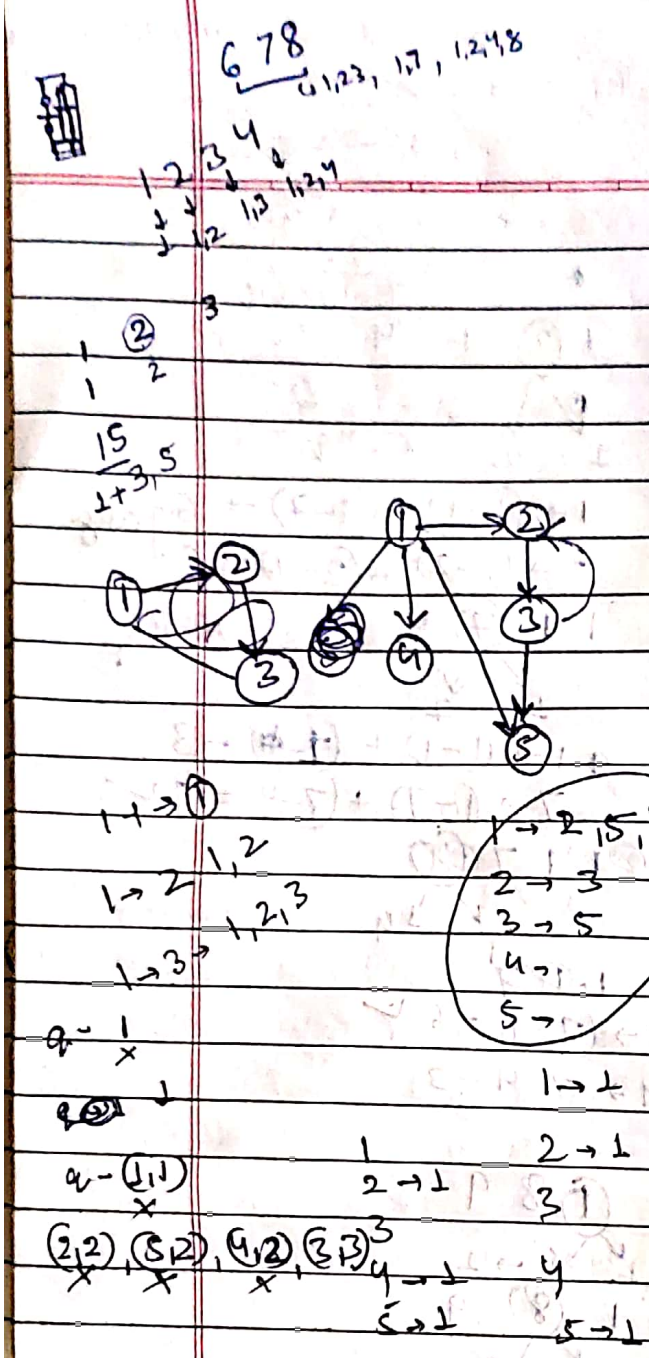
1 → (1−1) + (1−9) = 8
8 → (1−8) + (8−9) = 8
} any

1 1 9

1 → (1−1) = 0
9 → (1−9) = 8 ✓

1 1 9





Cardinality Relationships

- One to one
- Many to One
- One to many
- Many to Many

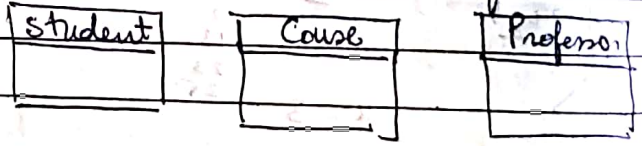
Steps to create an ERD →

- Entity identification
- Relationship identification
- Cardinality Identification
- Identify attributes
- Create ERD.

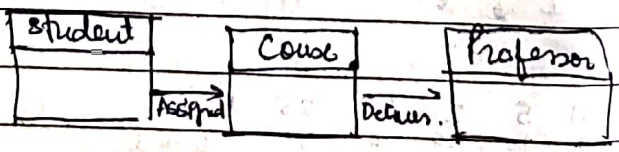
Ques - In a university, a student enrolls in courses. A student must be assigned to at least one or more courses. Each course is taught by a single Professor. To maintain instruction quality, Professor can deliver only 1 course.

① Entity identification

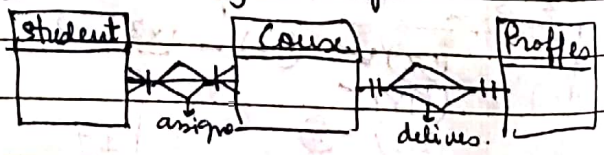
- Student
- Course
- Professor



② Relationship identification



③ Cardinality identification

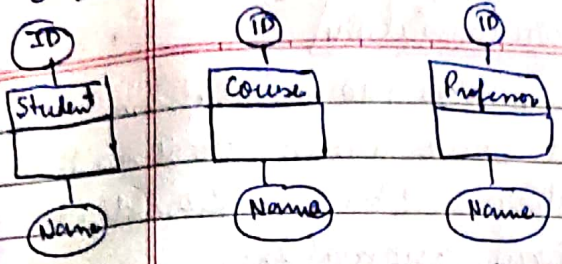


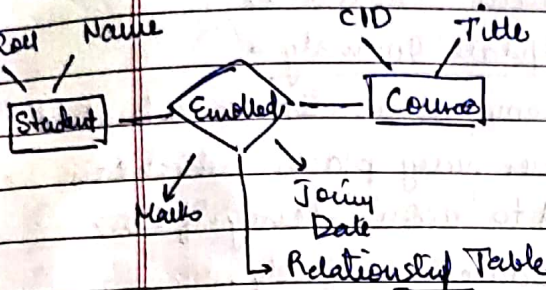
DBMS - I

Types of attributes

1. Simple attribute → which can't be divided further.
2. Composite → breakdown into simpler attributes.
3. Derived attributes → derived from a certain set of physical attributes like age from DOB.
4. Multivalued → more than one values, eg Ph → No, addres.

① Identify attributes.





Keys -

- Super key → an attribute or set of attributes that uniquely identifies a tuple within a relation.
- Candidate key → a super key such that no proper subset is a super key within the relation.
  - Gender + Name + Age → Super key but not candidate key.
  because there is a proper subset which is also a super key.
- The selected candidate key is called the 'Primary key' and rest are called 'Alternate key's.

- Foreign key → One or more attr in an entity type that represents a key, either primary or secondary (in any other table).

#FD → Functional dependency

$X \rightarrow Y$

So, if you have X then you can determine Y.

$X \rightarrow Y$ → dependent
↓
determinant

Emp_Id → Emp_Name

If you have Emp_Id then you can get emp_name.

A, B, C → D

It only say ABC together can determine D, but it doesn't mean that only A, B, C can determine D. (It can, it cannot)

- Closure F → all the dependencies that can be implied using the dependencies present in F.

Eg →

F	F^+
$X \rightarrow Y$	$X \rightarrow Y$
$X \rightarrow Z$	$X \rightarrow Z$
$Z \rightarrow A$	$Z \rightarrow A$
	$X \rightarrow Y, Z$
	$X \rightarrow Y, Z, A$

$A, B \rightarrow C$
$B, C \rightarrow D$ $\Rightarrow$ $A, B \rightarrow A, B, C, D$

(A will give C and C will give D.)

Rules of FD

1. Reflexive

If Y is a subset of X, then X determines Y.

$\therefore X \supseteq Y \rightarrow X \rightarrow Y$

$X = \{a, b, c, d, e\}$

$Y = \{a, b, c\}$

$X \rightarrow Y$

2. Augmentation Rule

The augmentation is also called as a partial dependency.

$X \to Y \quad XZ \to YZ$.

3. Transitive Rule

$X \to Y \quad Y \to Z$

$\Downarrow X \to Z$

4 Union

$X \to Y \;,\; X \to Z$

$X \to YZ$

5 Decomposition

$X \to YZ$.

↳ $X \to Y$

$X \to Z$

6 Pseudo Transitive Rule

$X \to Y$

$XZ \to W \quad YZ \to W$

Proof $X \to Y$

$WY \to Z$

• Trivial FD

① Trivial → $X \to Y$

if Y is a subset of X then this FD is trivial (It will always hold).

② Non-Trivial → $X \to Y$

if Y is not a sub-set of X. Then it's Non-Trivial.

③ Completely Non-Trivial →

$X \to Y$.

if $X \cap Y = \phi$ then completely non-Trivial.

Normalisation

There are many databases but some are better than the others. → because those don't contain anomalies.

• Update Anomaly →

some attributes are redundant over many places, which can lead to inconsistency if not updated carefully.

• We generally don't need to create big tables. Smaller chunks are more preferential.

one small reason→

Suppose a student is enrolled but he needs time to decide the subjects but since the table requires the subject, so unable to enroll him/her at the moment. ↑ this is called insertion anomaly.

• deletion anomaly → if a tuple is deleted then all the information will be deleted, however if want to retain some but can't

• updation anomaly →

Suppose some data gets updated so you need to update the entry at every column.

⇓

These were the common as major issues in non-normalized tables.

Forms of Normalization

Nasty pro's but I can as well.

- 1st NF

No column should contain multivalues → atomic valued.

Programming Lang Prog

Java, C++ ✗

C++ / Java ✓ 1NF

- 2 NF

Prime → attributes that are part of Primary Key.

Non-Prime → attributes that are not part of Primary key.

So, For 2NF to hold, every non-prime attribute should be functionally dependent upon any prime attribute.

If $X \to A$

then there shouldn't be any Y s.t. $Y \subset X$ and $Y \to A$

↳ then 2NF does not hold

Stu Id	Proj ID	Stu Name	Proj Name

Prime → Stu Id ; Pro ID → together or not at all.

but then Stu Id can identify Stu Name alone.

this is called Partial dependency.

Break the Table in Two.

Stu

Stu Id	Student Name	Proj ID

Prime

Proj

Proj ID	Proj Name

Prime

Now 2NF

- 3NF

① No non-prime attribute is transitively dependent on prime key attribute

② For any non-Trivial functional dependency $X \to A$ then either X is a super key or A is prime attribute.

Eg → Prime → Emp Id.

Non-prime → Emp Zip, Emp State, Emp City.

Now State and City are dependent on Zip and Zip is dependent on Id.

So, those are two are transitively dependent on Id.

Emp Id	Emp Name	Emp Zip

and

Emp Zip	Emp State	Emp City

④ BCNF

advance version of 3NF, stricter than 3NF.

(in BCNF if every FD is of form $X \to Y$, then X must be a super key.

3NF + LHS is Super key.

Dont keep on normalizing because at some point of time you need to back off since so many tables so many joins. So computation heavy.

#Transactions :→

• a set of logically related operation.

Date
Page

ACID.

A→ Atomicity → either commit all or none.

C→ Consistency → mostly from one consistent state to other means DB also change state } State and DB state must be consistent

I→ Isolation →

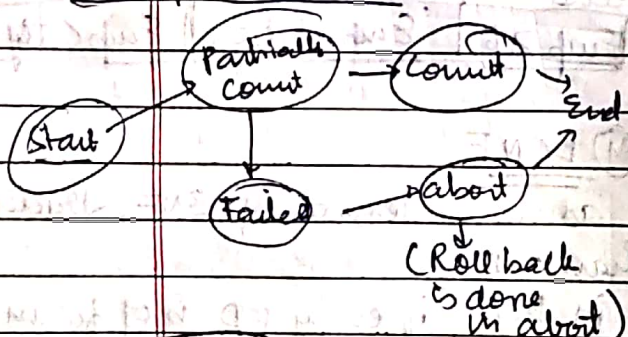
D→ Durability →

Isolation → each transaction must be isolate from other.
10,20 → but only 1 should receive not both.

durability → If a transaction is committed then under no circumstance it's undoable.

C→ Sum of the accounts related to the transaction would remain same before and after trans.

States of Transaction



Indexing

Indexing optimises the disk access whenever a query is performed.

↳ it's a kind of Meta data.

like index in a book → if want to read this, go to this Page No.

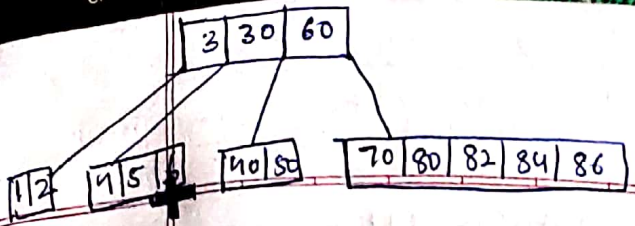
sometimes we need to index the index itself to reduce the space. This is called Multi-Level indexing.

Why B-Trees ?

Red-Black and AVL trees are assumed that everything is in Main Memory. But ok to handle the huge amount of data we need less mem access so B-Tree B-Tree is fat so disk access is lesser than O(h) times.

Prop of B-Tree.

- All leaves are at same level.
- A btree node is defined by term min degree t. The value of node t depends upon disk block size.
- Every node except root must contain at least t-1 key. Root may contain 1 key.
- All nodes including root may contain 2t-1 keys.
- No. of children of a node is equal to the no. of keys in it +1.
- All keys of a node are sorted in increasing order.



all operation of B-Tree need to revisit.

PBMS-2

SQL commands

- Alter table → add columns
- As → alias.
- select distinct col_name from Table

Select * from Person where name IN ("", "", "").
↓
Only among these (simplified Or).

- between _ and _ .
numeric, date and text.
- Case
Select movie_id
when cond then result1
when cond then result2
else result3
End
from table name.
very

- Select movie_id
case
when reviewer_stars >= 7 then num of ratings
else "Not a hit"
End
From Ratings

- Limit
limits the no. of row in the output

Date
Page

- Like → to search for a pattern.
Select actor from actors where first_name like "J%".
↓ all starting with J.

- Order by
select * from Ratings order by reviews_stars DESC.
limit 3
↳ Top 3 movies.

- Joins (INNER JOIN ON) table name.
inner join → return not null from both tables.
left join → null will be considered in the left table.
Right join → null will be considered in the right join
Full outer join → Null would be considered for both tables.
↳ if join condition fails then NULL would be there.

- group by used only be aggregate functions → group identical things
- How many movies done by an actor.

where is not used for aggregate function
use having if you want to handle aggregate functions

APIs FrontEnd

Date
Page

Q) what is Merge Conflict in Git?

Logic in HTML in Flask.

return ("View.html", todos = mytodo)

{% for todo in todos %}

{{ todo }}

{%. endfor %}

• Longest common Substring

1) Geekforgeeks

2) Geekquiz

		G	e	e	k	f	o	r	G	e	e	k
	•	0	0	0	1	0	0	0	1	1	1	1
G	1											
u	0											
e	1											
e	1											
k	1											
q	0											
u	0											
i	0											
z	0											

• Quiz Complete till Night
• 5 questions (Leetcode) + (CodeMonk)
• 2 Lectures IB

Count back

27 M
28
29
19
09
08
07

35
34
33
23
13

• Substring difference

• create a dp matrix which will store no. of mismatched char from i to j.

S1 → tabrina (0-6)

S2 → torinos

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

0123456 7

hellowordd yellowmarin

[4,7] → 2

owordd
owma → 2

l r mid

1 7 (3-0) 4 (1-1) →

i,j → (4,4), (4,5), (4,6),

(5,5), (5,6)

(6,6)

0	1	2	3	4	5
t	a	b	r	i	z

t	a	r	i	n	o
0	1	2	3	4	5

i,j → (0,3)

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

tabri
ori
tabr
orin

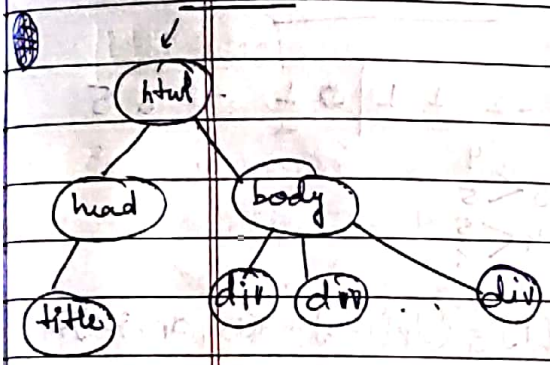
① Backend
↳ move view to diff. file
↳ connecting to dB
↳ CRUD & API's

Frontend

wget ——→ like → works like an api

• connect flask to DB.

HTML



↓ ↔ ??

DOM tree (DOC object Model)

↳ Browser transforms into Render tree

↓ ??

• DOM and Render → Imp stuff

• diff b/w head and body ??

↳ kind of index (nothing is displayed)

• search engine → web crawler reads the web pages. So optimize your head tag for SEO.

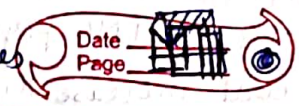
deeper meaning

• Semantic Web

↳ give deeper meaning to your html tags so that web crawlers of google (in gen) understands it properly like product, price ratio and display in results.

doctype specifies it's HTML-5

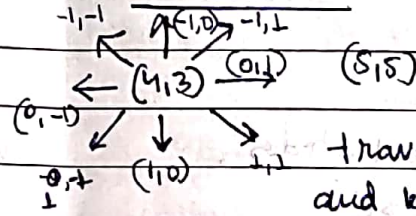
• one h1 → Primary heading
↳ multiple h1 confuses SEO.



• Grid concept of Bootstrap then always responsive.

• Build To-do list with flask and online DB on Heroku.

• Queen's Attack - II



• Manasa Slows

a,b → +a or +b.

and find all possible results

0 → 4 8 . 10,1100.

0, (0, 100), (20, 110), (110, 200),

(30, 120, 120, 210, 210, 300)

• Courses +1
2 vids
• 5 ques.

Consuming APIs

HTTP, HTTPS → Secure (encrypted)

Get, Put, Post, delete
Read update create del.

→ idempotent
(any req remains same over time)

• GET:

↳ get requests are cached but post req doesn't
↳ small in size that's why.

• Some parts of pages are generally common for all like the working screen (except the header (where name is))

So, compute only top part

and combine it with the cached HTML page and return. This will increase the performance.

→ powerful optimization

- by default Flask handles get requests so, you need to explicitely tell it to handle Post Requests

ajax in flask -

```
$.ajax({
    url : '/add-todos',
    type : 'POST'
    data : { todo: val, todo: todo_val }
});
```

JS-2

in JS everything is an object

↔ Mammal class.

```
var Mammal = function (obj) {
    this.name = obj.name
    this.dob = obj.dob
    this.eat = function
};
```

not a good practice

```
m1 = new Mammal ({...
});
```

this → new empty object.

m1.eat() →

Mammal.prototype.eat = function () {

};

- variable Hoisting

declarations go on top but assignment stays there.

var b; → goes to top

var b = 3 < b = 3 → doesn't Stay there.

INO1 → Calvin's Game

5 3 -2 1 1 | 0 1 -2 3 5

k=2 < 3 < 4 / 4 < 5 / 5

dp[i] = max(dp[i-2] + a(i), dp[i-1])

- TopCoder problems → DP

VIM → (Modal editor) ↳ have various modes.

:q → exit

:w → :wq → save & exit.

:q! → quit without save

() Command mode (.)

:set number → get line numbers.

i → to go into insert mode.

esc → pushes out of insert

dd → deletes the current line

3 dd → deletes 3 lines

u → undo last actions

Keep hitting u to undo

ctrl+r → redo actions

to search a string

/ pravi... → hit Enter to go to that point

go to next by 'n'

yy → to copy a line

p → to paste

1. CN+G
2. Revise
3. FlaskVideos

15L+

Date
Page

SYSTEM-DESIGN-I

- Ask good Question → what features, scalability
- Don't use Buzzwords
- clear & organized thinking
- Drive Discussions (You - 80%. Int - 20%.)

You need →

1) Features
2) Availability
3) Scalability
4) Class Diagram. ↳ OOPS
5) Cost Effective

6) Define API's. ↳ who are going to call the API's.
7) Latency Performance.
8) Durability ↳ data is not lost → data is not compromised
9) Security & Privacy

- Topics that one must know before attempting any System design Ques.

1) Vertical & Horizontal Scaling
   - Vertical: add more power, cpu, memory to the existing host (Faster) ↳ IPC → same machine ↳ bad idea → costly, limit to amount of memory, SPOF (Single Point of Fail.)
   - Horizontal: add more similar machines (Slower) • RPC (because Network calls)

2) CAP Theorem → (P is must trade b/w C and A)
   C → Consistency → Read the most recent write
   A → Availability → you'll always get back the result (may be most recent or not)
   P → Partition Tolerance → network packets are dropped b/w two points.
   (Redundant not fails by failure)
   Most DB's use C over A. → they might be unavailable for some time but will give you most recent write data.

NoSql DBs prefer A over C.
They guarantee availability over consistency.

ACID vs BASE. → NoSql database

A → Atomic → commit all or none
C → Consistent → state of DB remains same
I → Isolation → only 1 transaction at a time
D → Durability → after a commit, under no circumstance it should be undone.
↑ Relational DB's

BASE
Basically Available Soft State Eventually Consistent

Partitioning / Sharding Data
Storing a very large data to many nodes where each node is responsible for holding some amount
Key terms used in Sharding data → Consistent Hashing
↳ Advantages - disadv.

Optimistic and Pessimistic Locking

- Strong vs Eventual Consistency
  - Strong: the Reads will always see the Latest writes (Relational DB)
  - Eventual: ↳ the reads will see the writes which are not latest but eventually they will see.

Relational vs NoSQL
- Has high availability
- Follows Base

Types of NoSQL →
- Key value
- wide column DB
- document based
- Graph Based

- Architecture of Data center. Have Racks and Racks have Hosts
- Latency b/w Cross Racks
- Random vs Sequential read/write on Disk

Sequential Read/Write are faster??

- HTTP / HTTP2 / Web Sockets
  - Request response model (Entire Web runs on HTTP)

HTTP2 → can perform multiple requests on a single connection.
WebSockets

- TCP/IP Model → Know the 4 layers of this Model

- IPv4 vs IPv6
  - IPv4 → 32 bit address (will run out)
  - IPv6 → 128 bit address (will never die never)
  - IP Routing

- TCP vs UDP →
  - TCP → connection oriented reliable connection
  - UDP → unreliable connection

- DNS lookup →
  - ↳ gives back the IP → How??

- HTTPS and TLS
  - TLS → Transport Layer Security
    Maintains security b/w Client and server when used with HTTP

HTTP becomes HTTPS

- Symmetric and Asymmetric Enc.
  - Asymmetric → computationally heavy. So done on small amount of data. Eg. Public & Private keys
  - Symmetric → Eg AES

- Load Balancers
  How the load is scheduled over the nodes L4 and L7. (L7 more preferred)

#CDN's & Edge
- Data replicated around the globe so that to reach the client it looks into the nearest one (Netflix) Eg Akamai
- Edge → processing is done very close to the client.

#Bloom Filters
Count-Min Sketch
↳ You have million of events and you have to keep track of top k events (Frequency wise)
↳ with some inaccuracy.
} good DS's

Distributed Hosts → Leader Selection ??

- Design Patterns
  - ↳ Factory Pattern
  - Singleton

OOPs

- Virtual Machine ??
  Gives you an OS which gives you a feel that you have isolated system but its shared.
  Containers → Running your application and it's dependency in an isolated manner.

• Map Reduce
distributed & Parallel Processing of Big Data.

• Multithreading, Concurrency, locks, Synchronization, ??

• Technologies (Tools)
in Cassandra → wide column database, stores data in time etc.
Uses consistent Hashing to shard data
Can do both Eventual and strong consistency.

• MongoDB → stores document (JSON, XML)

• MySQL → depending on the use cases MySQL provides full adv. of ACID.

• Memcached, Redis → Distributed Cache
(Not the source of Truth.
• Holds limited amount of data)

• Zookeeper

• Kafka → Fault Tolerant highly available queue.
eg used in Streaming Application
Keeps the messages in order

• NGINX, HAProxy → Load Balancers (Highly advanced)

• Solr, Elastic Search → Provides Full Text search.

• Blob Store → S3 Bucket of Amazon storage

• Docker

• Kubernetes and Mesos are used to coordinate these containers

(Q) Design a service like Tiny URL.

• given a long URL → returns it's short URL
• given a short URL returns it's longer version.

Interviewer is looking for scalable and durable solution
Basic intuition is using a map (which is neither durable nor scalable).

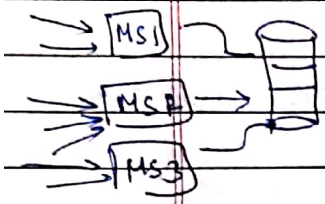
Two functions
① createTiny (longURL) → tinyurl
② getLong (tinyURL) → long URL.

• API
to how user will get back Req
• App. Layer
• Persistence Layer.

• Message Queues → queue + Load Balancer + Heartbeat
• It keeps the messages or tasks in order.
• It distributes the tasks fairly amongst the servers.
• It also checks for the liveness of server. If server doesn't respond then distribute tasks of that server to other Server.
Eg- Message Queues can be used in Pizza shops.

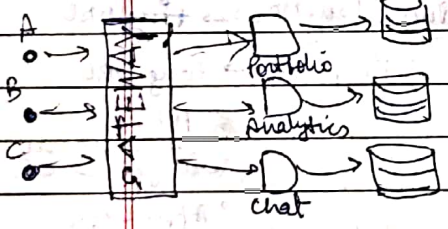
• Rabbit MQ
JMS (Java Messaging Service).

Monolith vs Microservices →
↓ tiny machines doing single jobs
one big Machine
all clients are connected to the single Machine

Monolith


↳ good for small teams (not req. to separate into Microservices).
• Faster because everything is in same place (so no RPC, only IPC)

• Dis → if a new member joins ↳ that member must go through the entire context
• Complicated Deployments
↳ Tight coupling → so too much dependencies

Microservices →


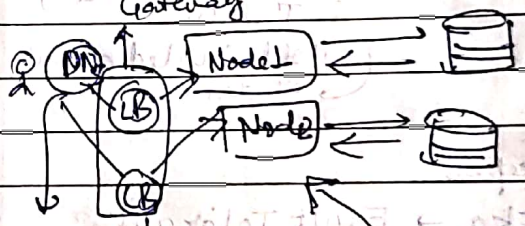
• easy for new members.
• Work in Parallel is easily possible because services are not dependent on each other. unlike the Monolith.
• good overview about each service (usage, data, latency etc)
• ** High skills are required to maintain this architecture

precisely Stack Overflow uses Monolith architecture

• Master Slave Architecture
• Master is the most updated node. Slaves update themselves by polling from master
• if Master fails then one of the slaves choose themselves as master by leader election

• DB Sharding
↳ generally done to maintain consistency. → Complicated to maintain shards.
↳ Joins are very heavy because tables are across shards.

• Avoid SPOF ↳ whose failure can take down entire system



Multiple IP's

multiple LB's
DNS will give a list of IP's to those LB's specific to a domain
Say fb.com

1. More nodes
2. Maintain Master Slave architecture
3. Multiple Regions ↳ if the whole Region is destroyed due to disaster.

• Netflix has something called Chaos Monkey which is sent into production which deliberately knocks down several nodes of Netflix to test their SPOF.

Distributed caching

- it saves Network calls. (store the commonly accessed data in the form of key-value)
- Avoid Recomputations (like say average age of all person)
- Reduce DB load.
  - Hit the cache instead of hitting DB everytime.

all these results in a speed up responses.

- Cache policy → ① LRU
  ② LFU
  - not frequently used.

B. size of cache is imp

- Thrashing → only loading and deleting but not using
- data consistency can be an issue.
  - use distributed cache → REDIS
    it's a global cache residing somewhere b/w server and DB. ☺ (slow but accurate)
    - if a server fails, the in memory cache also fails then everyone will now query the global cache (So, it's fault tolerant (Resilient)).

- Scale the Redis caches independently, without hurting anything
- Write Through
  - if there is an update
    - check in cache → if present then update it then update in DB.

⇒ 2 updates.

- Write Back.
  - if present in cache
    - delete it

and update the DB.

⇒ 1 update and if that this is requested the one DB query.

- in Write through, there can be many servers with entry X so you can't go on updating each of them.

- Keep Serving the requests and updating the cache (after say 20 sec) take a bulk data and update DB. (↳ this will save a lot of network calls).

- Caffeine → open source Java caching library. → yields high hit rates and excellent concurrency.

API Design

Application Programmable Interface

- How the external entities interact with your code, not how your code works.

- Where does the API belong
- Name of API
- Parameters
- define possible errors (don't throw same error on every error)

↓ What ↓ Where

POST gkcs.tech/chat/getAdmin

- no side effects (only one job)

What is NO-SQL and how is it used?

(SI) Smart Int

← DataTypes →

int, char, double

(4B) → signed → $(2^{-31} - 2^{31}-1)$

unsigned $(0 - 2^{32}-1)$

(Q) int setBits(int n)

```
{ int c=0;
  while(n)
  { if(n%2)
      c++
    n/=2;
  }
return c;
}
```

int f(int x, int y)

{ int sum = 1<<y;

sum2 = 1<<(x+y)

sum2 = ~sum2;

sum2 = sum1 || sum2

}

$(2^x) << y$

$2^y(2^x - 1)$

$2^{x+y} - 2^y$

① → 32 − 4 = 28

② → 64 − 2 = 62

$(2^x - 1) << y$

x → no. of 1's, y → 0's

(Q) ans = 0.

```
for(i=0; i<N; i++)
    for(j=0; j<N; j++)
        ans = ans ^ (a[i] ^ a[j])
```

2 3 5		10
2^2	2^3	2^5	101
3^3	3^2	3^5	111
5^5	5^2	5^3	11
			101
	7		110
1	6		10
			11
7	6		01

(Q) ⓪

→ 2 × (xor of all)

diagonal will be left.

(Q) a = 10, -5, 12, 8, -3

k = 19.

if sum

```
bool subset(int a[], int level, int n
            int s, int k)
    if (level == n)
    {   if(s==k)
            return 1
    }   else return 0
return subset(a, level+1,
        n, s+a[level], k)
    || subset(a, level+1,
        n, s, k)
```

```
bool subset(int a[], int n, int k,
            int s, level)
{   if (s==k)
        return 1;
    if (level==n && s!=k)
        return 0;
    return subset(a, n, k, s+a[level],
                  level+1)
        || subset(a, n, k, s, level+1);
}
```

```
size = pow(2, n);
for(i=0; i<size; i++)
    vector <int> sub;
{   for(j=0; j<n; j++)
    {   int set = (i & (1<<j));
        if (set)
            sub.push_back(a[j]);
    }
int sum=0;
for(int j=0; j<sub.size(); j++)
    sum += sub[j]
if (sum == k)
    return 1;
sub.clear()
}
return 0;
```

m

101

(Q) Generate all balanced parenthesis.

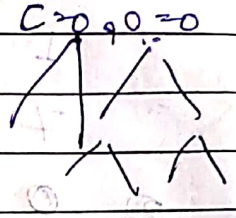
$^{12}C_6$

Date Page 7

str = "";

```
void valid(int n, int c, int o, string str)
{
    if (str.length() == n)
        Print str;
    else
    {
        if (o < n/2)
        {
            str += '{';
            valid(n, c, o+1, str);
        }
        if (c < o)
        {
            str += '}';
            valid(n, c+1, o, str);
        }
    }
}
```

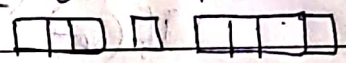
10



N = 15

(Q) colour → 0 1 2

box → 1 3 1 4

M → 

k=2 → atleast k distance apart

```
power(int x, int y)
{
    if (y == 0)
        return 1;
    int r = power(x, y/2)
    if (y%2)
        return r*r;
    else return r*r*x;
}
```

(Q) Word break

L : { smart, art, inter, view, s, mart, interview }

```
map<string, int> m;
int part(str, level, n, temp="")
{
    if (level == n)
    {
    }
    temp += str[level];
    for(int i=level; i<n; i++)
        temp += str[i];
        if (isValid(temp))
        {
            cnt++;
            temp="";
        }
    part(str, i+1, n, temp)
```

```
map<string, int> m;
int part(str)
for( int i=0;i
```

```
int part(str, N, L, idx) {
    if (idx == N)
        return 1;
    for( int i=idx; i<N; i++)
    {   if( belongs( substr(idx, i))
        {   ans += part(str, N, L, i+1)
        }
    }
    return ans.
```

```
    if m.find(str) != m.end()
        ans += m[str].
    else { c = part(str, N, L, i+1)
        ans += part(str, N, L, i+1)
        ans += c;
        m[str] = c;
    }
```

```
int part(str, N, L, idx, temp)
{   if(idx == N)
        return 1;
    for( int i=idx ; i<N ; i++)
    {   if (belongs(temp))
        {   if (m.find(temp))
                ans += m[temp]
            else
            {   c = part(str, N, L, temp+str[i],
                        idx+1)
                ans += c;
            }
        }
    }
}
```

(Q)

→ 10, 3, 18, 8, 5, 12, 15, 2, 7

count inversions

```
int merge(int a[], int l, int m, int r)
{   int i=l, j=m+1;
    vector<int> temp;
    while (i<=m && j<=r)
    {   if a(i) <=a(j)
        {   temp.push_back(a(i))
            i++;
        }
        else
        {   c += (m-i+1);
            temp.push_back(a(j))
            j++;
        }
    copy temp to a
```

```
int mergeSort(int a(), int l, int r)
    if (l<r)
{   m = (l+r)/2
    c = mergeSort(a, l, m)
    c += mergeSort(a, m+1, r)
    c += merge(a, l, m, r)
}
return c;
    int ct=0;
    for(int i=l; i<=r; i++)
        a[i] = temp[ct++];
return c;
```

(Q) $a(i) + a(j) = k$
$i \neq j$

Date
Page

• Using Hash Set

using HM, HS

```
unordered_map<int,int> M;
for(i=0;i<n;i++)
{   a[a(i)]
    x = a(i)
    y = k - a(i);
    if(m.find(y))
        return true
    m[x] = i
return false.
```

```
unordered_set<int> s
for(int i=0; i<n; i++)
{  int x = a[i].
   if(s.find(k-a(i))
       return true;
   s.insert(x);
}
return false.
```

a_1 a_2
10 9 8 7 6 .

10 19 28 35 41 ..
46 50 53 55

56

sort + BS

```
bool bs(int a[], int l, int r, int x)
  while(l<=r).
  {  int mid = (l+r)/2;
     if(a[mid]==x)
        return true;
     else if(a[mid]>x)
        r = mid-1
     else l = mid+1;
  }
  return false;
}
for(int i=0;i<n;i++)
{  int x = a[i];
   int y = k-a[i];
   if(bs(a, i+1, n-1, y))
       return true.
}
return false
```

$(x) \times 10$

$(a_1 + a_2 \ldots) + 10 = 140$

$a_1 + a_2 = 90$

$a_n = \frac{9 \times 10}{2} = 90$.

$a_n = 45$

$n = 5$

$x + (x-1)d = 100$

17 16 15 14 13
18 35 51 66 80

14 13 12 11 10
15 29 42 54 65

75 9 84 8 92 7 99

50 25
16 18 50

11 10 10 12 14 10 10 20

x + x-1, + x-2 . . . k = 100

$\frac{N(N+1)}{2} \geq 100 \rightarrow \frac{25 \times 26}{2}$

{5 2 1}

Date / Page

N floors, M eggs

(Q) arr_N 5, -1, 8, 12, 5, 7, -2.

Find all non-decreasing subseq.

```
size = pow(2, n);
for(int i=0 ; i<size ; i++)
{   vector <int> sub;
    for( int j=0; j<n; j++)
    {   int bit = (i & (1<<j));
        if(bit)
            sub.push_back( a[j] )
    }
    flag=0
    for(int j=1 ; j< sub.size() ; j++)
    {   if  sub[j] > sub[i]
        { flag=1
          break;
        }
    }
    if(!flag)
        c++
}
}.
```

```
vector<int> v
subset( int a[], int n, int idx )
{   if(v.size() == n)
        print arr
    if(idx==n)
    {   print(v)
    }
    v.push_back(a[idx])
    subset(a, n, idx+1, v)

    v.pop_back()
    subset (a, n, idx+1, v)
}
```

(Q) a: 10, 20, 13, 19, 16, 15, 18, 16, 17, 14, 18, 9, 12.

```
multiset <int> s;
map<int,int> m  multiset<int>
for(int i=0; i<n; i++)
    m[a[i]]++;  s.insert(a[i]);
max=0   c=1
auto it2 = s.begin(); it2++;
for( auto it= s.begin(); it != s.end; it++)
{   if ( it2-it == 0 || it2-it == 1)
    {   c++
        advance (it2)
    }
    else
    {   max = max(c, max)
        c=1
    }
}.
```

it it2

for (i → n)
set it = s.begin
for j → N it2 : it.

1) $N^3 \lg n$
2) N^3
3) $N^2 \rightarrow$ (M-m+1) = uniques.
Hash sets
maintain max & min
by checking new element

x+(n-1)

		5	-1	8	12	5	7	-2
	1	1	1	1	1	1	1	1
5	1							
-1	2							
8	1							
12	1							
5	1							
7	1							
-2	1							

dp[0]=1

```
dp
dp[n]
for(i=1; i<n; i++)
    dp[i]=0.
{ for(j=0; j<i; j++)
    { if(A[i] >= A[j])
        ans+=dp[i];
        dp[i]=ans;
}. }
```

```
P = -∞;
if( ar(idx) >= P)
    f(ar, N, idx+1, ar(idx))
    +
  f(ar, N, idx+1, P)
```

(Q) ar 0 0 1 0 1 0 0 0 0 1 1

(A) -1 -1 1 -1 1

set

X NVP

m = 0

```
for(int i=0; i<n; i++)
    sum += a(i).
    if (m.find(sum))
        len = i - m[sum]
        m = max(m, len)
    else
        m[sum] = i
}
```

(Q) str_N :

a b d c c d b d c c d y z z

- Find the no. of palindromic substrings
- Find Palidromic subsequences.

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

1) N^3, 1
2) N^2, 1
3) Rolling + PS + BS

(Q) smartdxyatzadydata.

1. N^3
2. N^2
3. BS
4. Two Pointer

```
sqrt(N)
l=1, r=N   ans=-1;
while(l<=r)
{ m=(l+r)/2;
  if(m*m<=n)
  { ans=m
    l=m+1;
  }
  else r=m-1
}
```

(Q) a_N: 10 5 7 1 8 7 2 10 k=4

```
l=max, r=sum;
while(l<=r)
{ m=(l+r)/2
  if isValid(mid)
    ans=mid
    r=m-1
  else l=m+1;
}return ans.
isValid(){ c=1 s=0
{ for(int i=0; i<n; i++)
  { if(s+a(i)<=mid)
      s=s+a(i)
    else
      c++
      s=a(i)
  if c<=k
    return 1
  return 0;
}.
```

Brute generate all partitions

$^{N-1}C_{k-1} \times N$

generate sum.

(Q) 3 5 8 10 17 20 23 28 40 42 47 50 55 k=5

```
min max
min=1   h=max-min
while(l<=r)
{ m=(l+r)/2;
  if(isValid(mid))
  { ans=mid
    l=mid+1
  }
  else r=mid-1
}
isValid last=A(0)
for(i=0; i<n; i++)
{ if(A[i]>=last)
    c++
    last=A[i]
  }
if(c<=k)
  return true
return false.
```

i n-i. n-j

(8) A = 3 5 8 10 15 25

B = 1 4 6 7 17 19 20

l = min, r = max

while (l <= r)

{ m = (l+r)/2;

int x = upper_bound (A.begin, A.end, m) - A.begin

int y = upper_bound (B.begin, B.end, m) - B.begin

total = n+m;

if (x+y == total/2) → ans = m, r = m-1

else x+y > total/2

r = m-1;

else l = m+1;

② Time → $\log_2 R(\log N + \log M)$

int solve (string a, string B)

{ int f1[26] = {0};

int n = a.length()

int m = b.length();

for (int i=0; i<m; i++)

f1[b[i]%97]++;

int low = m, high = n;

while (low <= high)

{ int mid = (low+high)/2

int f2[26] = {0};

for (int i=0; i<m; i++)

{ f2[a[i]%97]++;

int valid=0

{ if (check (f1, f2))

{ valid=1;

break;

}

f2[a[i-m]%97]--

}

if (valid)

{ ans = mid;

r = mid-1;

}

else l = mid+1;

}

return ans;

}

bool check (int f1[], int f2[])

{ for (int i=0; i<26; i++)

{ if f2[i] < f1[i]

return false

}

return true;

Smart 0 matrix subarray

~~s→1~~	a
~~m→1~~	n
a→1	t
n→1	d
t→1	x
x→1	y

s→1
m→1
a→1
n→1
t→1

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

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

1 2 3 4 5 6

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

0 1 2 6. 4 5 8 9

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

(Q) a_n: 0 1 2 3 4 5
0 0 0 0 0 0 ..

(Q) 1 i → make it (1 − a[i])
2 i to distance of nearest 1.

- insert all indexes → 1 to set
  do forward cover bound
  reverse lower bou
- erase the index if it's made 0

abcd
abcd code
abcd code

set<int>
>=
>

(Q) 8, 10, 12, 7, 15, 25
30, 7, 4, 2, 5, 45
70, 4, 1, 7, 6, 48
35, 7, 4, 10, 15, 28

8 10 12 12 15 25
30 30 30 30 30 45
70 70 70 70 70 70
35 35 35 35 35 35

⇒	8	10	12	12	15	25
✓	30	30	30	30	30	45
	70	70	70	70	70	70
I L→R T→B	70	70	70	70	~~70~~	~~70~~
	25	25	25	25	25	25
II R→L B→T	45	45	45	45	45	45
	70	48	48	48	48	48
	35	28	28	28	28	28

✓	70	48	48	48	48	48
	70	48	48	48	48	48
	70	48	48	48	48	48
	35	28	28	28	28	28

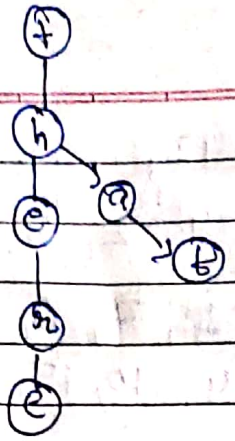
Spoj → Water

2)
W°

1000
101
001

set<int, greater<int>>
<=
<

• the tree that



(Q) given n words with q queries- find no. of words with given prefix.

1) $Q \times M \times N$, L
2) store in all prefixes in unordered set.
$NM^2 + Q \times M$ space $\rightarrow NM^2$

```
TrieNode {
    map<char, TrieNode*> child;
    map<char, int> f
    bool isWord;
};

struct Trie {
    int c;
    Trie a[26];
    bool isWord;
};
```

```
Main()
Trie root = new Trie();
for(int i=0; i<n; i++)
{   string s = s[i];
    int l = s.length();
    for(int j=0; j<l; j++)
    {   //if already
        int ind = s[j]-'a';
        if(root.a[ind]==null)
        {   root.a[ind] = new Trie();
            root = root.a[ind];
            root.c++;
        }
    }
    root.isWord = true;
}

query(x)
    int m = x.length();
    for(int i=0; i<m; i++)
    {   root =
    root =
    for(int i=0; i<m; i++)
    {   if root.a[ind] != null
            root = root.a[ind];
        else
            return 0;
    }
    return root->c;
```

Trie
map <char, TrieNode *> child
map <char, int> f
bool end;

TrieNode * getTrieNode()
{ TrieNode *temp = new TrieNode;
temp->end = false
return temp;

Insert
if (root == NULL)
root = getTrieNode()
TrieNode *cur = root;
int l = s.length
for (i=0; i<l; i++)
{ char x = s(i).
TrieNode *n = cur->child(x);
if (n==NULL)
{ n = getTrieNode.
cur->child(x) = n;
cur->f(x) = 1
}
else cur->f(x)++;
cur = n;
}
cur->end = True

getCount
TrieNode *cur = root;
int c = 0.
for (int i=0; i<l; i++)
{ char x = s(i);
TrieNode *n = cur->child(x)
if (!n)
return 0;
else
c = cur->f(x)
} return c;

(5) x, y

(a) Find the subarray with max Xor

| 2 | 5 | | 10 | 2 | | |

Prefix Xor

$O(N^3)$ → gen all and find Max

$O(N^2)$ → gen prefix Xor for all (i,j)
$x = p[j] \wedge p[i-1]$
find max(x).

$O(N)$ = trie of prefix Xor then find a pair max Xor

a[7] ^ a[2]

DSU

(Q) MAx Xor Pair .

101000
001000
001000

```
insert (n)
  trie *curr = root;
  for (i=31; i>=0; i--)
  {  int bit = (n)&(1<<i)
     if(bit==0)
     { while (curr->left)
          curr = curr->left
       if ( curr->left)
          cur = curr->left
       else
          curr->left = new trie
     }
     else
     { if (curr->right)
          curr = curr->right;
       else
          curr->right = new trie
  }.

getMax (a)
  for (int i=0; i<n; i++)
  {   val = a(i);
      for(j=31; j>=0; j--)
      {  bit = (val)&&(1<<j)
         x=0
         if (bit==0)
         { if (curr->right)
             {  x = x^pow(2,j)
             }  curr = curr->right)
           else  curr = curr->left
         }
         else
         { if (curr->left)
               x= pow(2,j);
               curr = curr->left;
           else
               curr = curr->right)
         }
      ans = max(ans, x);
  }
return ans.
```

(Q) Find no. of Subarrays with XOR < K

get prefix XOR

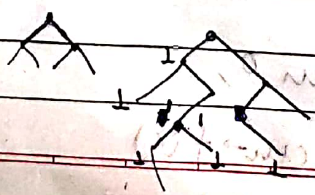
$O(N^2)$ = count all with < K

dp

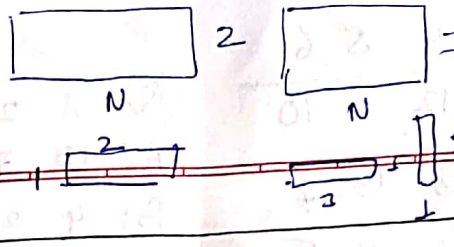
```
int fib( int N)
{   if (N==1 || N==2)
        return 1;

    return fib(n-1)+fib(n-2)
}

int fib( int n)
{   f(1) = 1
    f(2) = 1
    for (i=3; i<n; i++)
        dp(i)= dp(i-1)+dp(i-2)
    return f(n)
}
```



(Q) Tiling Problem



A 5xn floor
1x5 tiles
Find no. of ways to fill the floor.

5xN

$dp(i) = (dp(i-1)) \times 2 + dp(i-5) \times 2^5$

dp[1] = 1
dp[2] = 1

if coloured (b&w)

$dp(i) = dp(i-1) \times 2 + dp(i-5) \times 2^5$

Ex: 1 3 5 2 10

```
for (int i=1; i<=6; i++)
    for (int j=i; j<=sum; j++)
        if (a[i]<=5)
            dp[i][j] =
                dp[i-1][j-i]
```

P[0, N] =
max [a[0] + P[1, N], P(n-1) + P(0, n-2)]

dp[1]
dp[0]

dp[2][N]

1	4
10	8

```
for (i=0; i<=5; i++)
    for (j=0; j<n; j++)
```

$dp[i] = \min_{j=1}^{6} (dp(i-j) + c(j))$

if ((i-j)<=0)
    dp(i) = 0 + c(j)

i → 1 to N

max(a(l) + f(l+1, r), a(r) + f(l, r-1))

if (l>r) return 0.

(Q) Given a dice with 6 faces.

c(1) = 5 c(4) = 7
c(2) = 12 c(5) = 13
c(3) = 8 c(6) = 18

S = 10

d(i-1) * dp(i-2) +

1 2 3 5 8 13 21 34 55

(Q)

	1	2	3	4	5	6
R	10	4	8	12	6	10
B	4	7	10	3	15	7
G	9	2	8	1	7	5

Paint houses min cost such no 2 adj house of same

R	10	12				
B	4	6				
G	9	13				

dpR(i) → min cost to paint 1→i such that i^{th} house is Red.

dpB(i) →

dpG(i) →

dpR[i] = dpR[i-1] + a[i][0]

dpR[i] = min[dpB[i-1], dpG[i-1]) + R(i);

dpR[0] = R(0)
dpB[0] = B(0)
dpG[0] = G(0)
} → Base

dpB[i] = min(dpR[i-1], dpG[i-1]) + B(i);

dpG[i] = min(dpR[i-1], dpB[i-1]) + G(i);

R 10 8
B 4
G 9

ans = min(dpR(n), dpB(n), dpG(n))

Space → maintain 6 variables.

(Q)

	1	2	3	4	5	6
A:	10	7	8	12	18	5
B:	4	2	10	17	1	20

min time to produce all products.
if you switch from A-B or B-A there is an additional overhead of c. c = 2

dpf[i] = dp[i-1]

dp[i] → ~~produced pro~~ min cost till production of (i)th machine

dp(i) =

dpA(i) = min(dpA[i-1], c+dpB[i-1]) + A[i];

dpB[i] = min(dpB[i-1], c+dpA[i-1]) + B(i).

dpA[0] = A[0]
dpB[0] = B[0]

ans = Min(dpA(n-1), dpB(n-1))

1 2 3 1 2

Kadane