- **Flow** of any algorithm could include :

1. Divide & Conquer : Break into sub problems.
2. Dynamic Programming : For any DAG topologically sorted problem solving the mini problems help solve the bigger
3. Greedy algorithms : quick success.
4. Approximation algorithm
5. Linear Programming

- **Adversarial** Strategy :-

eg quick sort. Worst case (n^2).
Adversary : Input given
Bigger adversary : Randomizes the pivot. } O(nlogn)
Adversaries interfere with each other.

→ START : digitalization
FLOW : divide & conquer } Methodology.
ADVERSARIAL : Randomization
STOP : Proactivization.

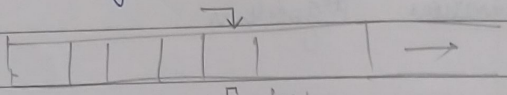
- Turing's Assumptions :

A1 : Finite memory can only ~~sto~~ reliably store finite information.
# of bits that can be

Super-polynomial = (const)^n

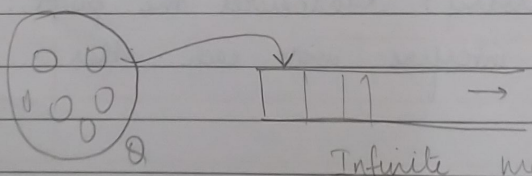
- Every information stored is a different mathematical curve. Space is of infinite precision.
- ∴, Turing argued that you can store this minute difference information but retrieval is difficult

(A2): Information travels at finite speed
- There's nothing like infinite size RAM
- memory is infinite but divided into finite chunks
  ✗ you should go to that location to access memory



(A3): A finite length program can have only finite control
- finite code ⇒ finite instructions ⇒ finite configurations
  ∞ many config.

Proposed model :-



∴, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

(δ: finite; Γ: finite)

- Turing stated that a machine takes in an input and outputs what the machine does to the input

$U(\underbrace{I, x}_{\text{input}}) = \underbrace{T(x)}_{\text{output}}$

S start A adversary F low E end

A turing machine is a 7-tuple

state input alphabet tape alphabet

$< Q, \Sigma, \Gamma, \delta, q_{start}, q_{acc}, q_{rej} >$

$\Sigma \subsetneq \Gamma$

blank char $b \in \Gamma$, $b \notin \Sigma$.

→ Church-Turing Hypothesis:

Algorithms ≡ T.M

• Graphs:

$G = (V, E)$

$E \subseteq V \times V$

Minimum vertex cover (C)

$C \subseteq V$ such that every edge of the graph has at least one end point in C

Conjecture: No efficient algorithm exists to find MVC.

– Instead we prove that output a VC that will be $<= 2 \cdot MVC$. End gracefully

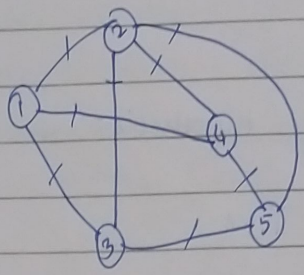
(If you do anything less than the bound it is predicted you'll solve it exactly).

• 2-approx algo for MVC:

Initially $C \leftarrow \phi$

- Choose an edge such that (u,v), $u \notin C$, $v \notin C$
- $C \leftarrow C \cup \{u,v\}$. (remove/mark all edges that end with either u or v)
- Until no edge left

For eg:



Choose: $\{1,2,3,5\}$ <= 2. M.V.C.

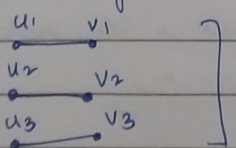
Theorem: The algo is 2-approximate
ie: $|C| <= 2|C^*|$

(Pf:) Let the algo choose k edges in all
then:

$|C| = 2k$ ... trivial

To show: $|C^*| >= k$.

Each pair of vertices chosen must be distinct.



G' graph is a subgraph of G

Now MVC of $G' <= |C^*|$
G' is such a simple graph ST MVC of G' is k

$\therefore$ $k <= |C^*|$

$\Rightarrow$ $|C| <= 2|C^*|$

→ Pseudo Randomness:

$$G(s) = \{0,1\}^{t(n)}$$
$$\uparrow \{0,1\}^n$$

No efficient test can distinguish whether random or not

→ Consider a virtual reality program where a coin is toesed. If this is deterministic then its not random

If $p_1, p_2 \ldots p_2$.

No: of bits $= \log\left(\frac{1}{p_i}\right)$.

Higher entropy
Lower Info.

$\therefore E\left(\log\left(\frac{1}{p_i}\right)\right) = \sum p_i \log\left(\frac{1}{p_i}\right) =$ Entropy $= H(n)$.

• Flow Well:

Consider integer multiplication of two n bit no:s
output = 2n bit integer

$O(n^2)$ naive

Big O notation:

$f(n) \in O(g(n))$ if $\exists$ a $c \in \mathbb{R}^+$ & $n_0 \in \mathbb{N}$ such that

$$f(n) \leq c\, g(n) \quad \forall \; n \geq n_0$$

Alternative: use divide & conquer.

Base = D

let $A = a_{n-1}\, a_{n-2} \ldots a_1\, a_0$

$B = b_{n-1}\, b_{n-2} \ldots b_1\, b_0$

$$A = \sum_{i=0}^{n} a_i D^i \qquad B = \sum_{i=0}^{n} b_i D^i$$

Consider splitting

$$A = A_1 | A_0$$
$$B = B_1 | B_0.$$

$\therefore$,
$$A = A_1 D^{n/2} + A_0$$
$$B = B_1 D^{n/2} + B_0$$

$\therefore$, 4 products will be obtained

$A_0 B_0$ $\quad$ $A_0 B_1$

$A_1 B_0$ $\quad$ $A_1 B_1$.

$\therefore$, A recursive relation will be

$$T(n) = \underbrace{4 \times T\left(\frac{n}{2}\right)}_{\text{4 products}} + \underbrace{O(n)}_{\text{adding digits}}$$
$$= O(n^2)$$

a = 4
b = 2
d = 1

$a > b^d$
$\log 4$
n^2

~ MASTERS TH:-

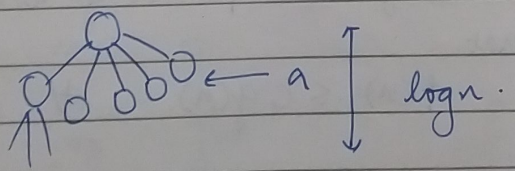
$$T(n) = a \cdot T(n/b) + O(n^d)$$

then

$$T(n) = \begin{cases} O(n^d) & d > \log a \\ O(n^d \log a) & d = \log a \approx a = b^d \\ O(n^{\log_b a}) & d < \log a \end{cases}$$

$a > b^d$

Pf:



$\therefore$, At each level work done

$$= a^k \left(\frac{n}{b^k}\right)^d$$

GP with ratio : $\left(\frac{a}{b^d}\right)$

Trick: By the mathematician Gauss:

Consider 2 complex no:s

$$(x+iy)(a+ib)$$
$$= (ax - by) + i(ay + bx)$$

With 3 real multiplications & some extra additions

Let:

$P_1 = ax$ (A0 B0)

$P_2 = by$ (A1 B1)

$P_3 = (a+b)(x+y)$ (A0+A1)(B0+B1)

$= ax + ay + bx + by$

$\therefore$ $P_1 - P_2 + i(P_3) - i(P_1 + P_2)$.

$\therefore$ Let $P_1 = A_0 B_0$

$P_2 = A_1 B_1$

$P_3 = (A_0 + A_1)(B_0 + B_1)$

Now, time complexity changes to

$$T(n) = 3T(n/2) + O(n).$$
$$O(n^{\log_2 3})$$

→ There is a big bargain between

- Representation &
- Operational Efficiency

∞ n^{th} fibo no:-

Let $f(n) = r^n$

$r^{2n} = r^{n-1} + r^{n-2}$

$r^2 = r + 1.$

$r^2 - r - 1 = 0$

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

$$\Rightarrow F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

Polynomial evaluation in Order(N)

Horner's Rule:- Eg: $4x^3 + 3x^2 + 7x + 5$

$x(4x^2 + 3x + 7) + 5$

$= x(x(4x + 3) + 7) + 5$

$= x(x(4x + 3) + 7) + 5$

Fast Fourier Transformation:

Works on polynomial multiplication.

$p(x) \rightarrow$ d degree

$q(x) \rightarrow$ d degree.

$r(x) = p(x) \times q(x)$.

If $p(x) = \sum_{i=0}^{d} a_i x^i$

$q(x) = \sum_{i=0}^{d} b_i x^i$

$r(x) = \sum_{i=0}^{2d} c_i x^i$

$C_k = \sum_{i=0}^{k} a_i b_{k-i}$ $O(d^2)$ – time.

Can we do better in $O(d \log d)$

→ Consider co-efficient representation

$[a_1, a_2 \ldots a_d]$.

Point Representation :- ↗ efficiently recover it.

No loss of info about the poly

$\begin{bmatrix} p(x_1), p(x_2), \ldots p(x_{d+1}) \ldots p(x_{2d+1}) \\ q(x_1), q(x_2), \ldots, q(x_{d+1}) \ldots q(x_{2d+1}) \end{bmatrix}$

Eg: $y = mx + c$

$r(x_i) = p(x_i) \times q(x_i)$

Step 1: Select $x_1, x_2, \ldots x_{2d+1}$ $(O(d))$

Step 2: Eval $p(x_1) \ldots p(x_{2d+1})$ $(O(d^2))$

$q(x_1) \ldots q(x_{2d+1})$ $(O(d))$

Step 3: Multiply $r(x_i) = p(x_i) * q(x_i)$

Step 4: Interpolate to obtain co-eff of $r(x_i)$

Plotting n points randomly gives O(n^2)

Date
Page

∴ Choose points efficiently.

Select n^{th} roots of unity

let the invariant be:- $p(x) = p_e(x^2) + x\, p_o(x^2)$

let x_2 be $-x_1$

$\therefore$, $p(x_1) = p_e(x_1^2) + x(p_o(x_1^2))$

$p(x_2) = p_o(x_1^2) - x_1 p_o(x^2)$.

$p(x) = x^6 - 2x^5 + 5x^4 + 7x^3 + x^2 - x + 1$ ← deg n

$= (x^6 + 5x^4 + x^2 + 1) + (-2x^5 + 7x^3 - x)$

→ deg n/2 → deg n/2

$= p_e(x^2) + x \cdot p_o(x^2)$

$p_e(y) = y^3 + 5y^2 + y + 1$.

$p_o(y) = -2y^2 + 7y - 1$.

Work: $E(n) = 2(E(n/2)) + O(n)$

n is a perfect pow 2

Assumption: n pts are paired in n/2 such that $\pm x_i$

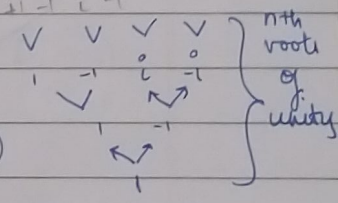
To do better to calculate $p(x), q(x)$ use n^{th} root of 1

To choose the values of $x_1, x_2 \ldots x_{d+1}$

Reforming Step 2:

Evaluate $p(1), p(\omega), p(\omega^2) \ldots, p(\omega^{n-1})$

$q(1), q(\omega), q(\omega^2) \ldots, p(\omega^{n-1})$



in $d \log d$.

[Splitting unit circle into n angle parts to obtain n pts on the circle hence $2\pi k/n$]

Consider A[n] → coefficient array of P(x).

Algo

```
FFT ( A[n] , ω)
{ If  n = 1   output  A[0]
  Else :
  S[] ← FFT( Ae[n/2] , ω^2)
  T[] ← FFT ( Ao[n/2] , ω^2)
  for i=0 to n-1
      P[ω^i] ← S[ω^2i] + ωi * T[ω^2i]
```

Half the terms take constant time

$\therefore (\omega_n^k)^2 = (\omega_n^{k+n/2})^2$

Let the matrix M be such that $M[i,j] = \omega^{ij}$

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & & \\ 1 & \omega^2 & & & \\ 1 & & & & \\ & & & \omega^{(n-1)(n-1)} & \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix} \begin{matrix} \rightarrow R(1) \\ \rightarrow (R(\omega)) \\ \\ \rightarrow (R(\omega^{n-1}) \end{matrix}$$

For a 3 degree poly^n $\begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega^4 \end{bmatrix}$

2 degree poly^n $\begin{bmatrix} 1 & 1 \\ 1 & \omega \end{bmatrix}$

Modifying Step 4: $r(n) = \left[\frac{1}{n} \text{FFT}(R[n], \omega^{-1})\right]$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega^4 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Pre multiplying:-

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = M^{-1} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- **PRIMALITY TESTING:** (MILLER-RABIN ALGORITHM)

Input : N > 2

Output : Yes if N is prime
No otherwise.

Naive algo:

Test divisibility upto $\sqrt{N}$

ie: check $N \% i \neq 0$ for all $i = 2$ to $\sqrt{N}$

! Practically too slow for 1000 bit prime

▲ Fermat's Little Theorem:

If n is a prime then

$\forall a \in [1, n-1]$

$$a^{n-1} \bmod n = 1.$$

Pf: Let $a \neq 0$, n be a prime no then,

$a \% n$, $2a \% n$, ... $(n-1)a \% n$ ~~are~~

~~none 0~~ all lie in $[1, n-1]$

→ To show that all $a \ldots (n-1)a$ are distinct then ⇒ that they'll be a permutation of $[1, n-1]$

Let us assume for some $i, j \in [1, n-1]$

$$ia \% n = ja \% n$$

$$\therefore (ia - ja) \% n = 0$$

$$a(i-j) \% n = 0$$

$$\Rightarrow n \mid a(i-j)$$

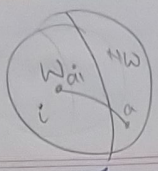
$1 \leq a \leq n-1$, $1 \leq (i-j) \leq n-1$

∴ 2 no:s less than a prime can never be divisible by a prime

Hence # contradiction.

→ Consider:- $a \% n$, $2a \% n$, $3a \% n$ .... $(n-1)a \% n$ $\xrightarrow{\text{mult}}$ $(n-1)! a^{n-1} \% n$

$1, 2, 3 \ldots (n-1) \xrightarrow{\text{mult}} (n-1)! \% n$



If $i^{n-1} \not\equiv 1 \pmod n$ working

$a^{n-1} \equiv 1 \pmod n$ not working

and $(ai)^{n-1} \not\equiv 1 \pmod n$

If this happens then ~~at least~~ no: of i's should be at least greater than or equal no: of as. ∵ a one-one mapping exists.

→ If n is prime then $x^2 \equiv 1 \bmod n$ has exactly 2 roots

Pf: $x^2 \equiv 1 \pmod n$

$n \mid (x^2-1) \quad = (x+1)(x-1)$

Algo: Input N; Let $N-1 = s2^r$ s is odd.

① Choose k random no:'s in $[2, N-1]$

say $a_1, a_2 \ldots a_k$.

② Create an array $M_{k \times (r+1)}$.

$M[i,j] = a_i^{s.2^j} \bmod N$.

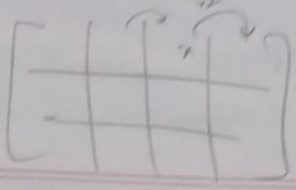
③ If ∃ a row of M such that the first non-1 no: from R-L is not n-1 then NO else YES

N=7 N-1=6 = 2.3 (S=2)

$a_0 = 2$

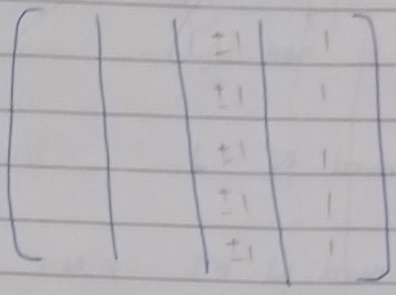
$a_1 = 3$

1	1
6	1



Thm: If N is prime algo always outputs YES.

Pf:

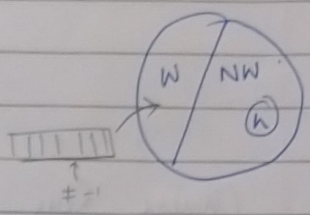


Thm: If N is composite, algo outputs NO with probability $\geq 1 - \frac{1}{2^k}$

Pf:

$f: NW \to W$

one-one map exists



Choose an a from $N-W$

$$h^{s2^{j*}} \equiv -1$$

$$h^{s2^{j*+1}} \equiv 1$$

$a \in NW$ ~~$a^{s2^{j*+1}} \equiv 1$~~

If N is composite $N = pq$ $\gcd(p,q) = 1$

From CRT: $\exists\ t$

$$t \equiv h \pmod p$$

$$t \equiv 1 \pmod p$$

To show $t \in W$ (why?)

$$t^{s2^{j*+1}} \equiv h^{s2^{j*+1}} \equiv 1 \pmod p$$

$$t^{s2^{j*}} \equiv h^{s2^{j*}} \equiv (p-1) \bmod p$$

- $t^{s \cdot 2^{j*+1}} \equiv 1 \mod q$

- $t^{s 2^{j*}} \equiv 1 \mod (q)$

- $t^{s 2^{j*+1}} \mod N = 1$

- $t^{s 2^{j*}} \mod N \neq 1 \quad \neq N-1$

$a \in NW$

$a^{s 2^{j*+1}} \equiv 1$

$a^{s 2^{j*}} \equiv \pm 1$

$t \in W.$

$t^{s_2 j*+1} \equiv 1$

$t^{s_2 j*} \neq \pm 1$

- **Chinese Rem Th:**

For a sequence of congruences:-

$$x \equiv a_1 \pmod{p_1}$$
$$x \equiv a_2 \pmod{p_2}$$
$$\vdots$$
$$x \equiv a_n \pmod{p_n}$$

where (p_i, p_j) $\forall i, j$ are co-prime there exists a unique $x \in [0, \pi p_i - 1]$ that satisfies the above cases.

$(x + m \pi p_j)$

Proof: If $a_1 = 0 = a_2 = \dots$

then $x \equiv 0 \pmod{p_1}$

$\vdots$

$x_n = 0 \pmod{p_n}$

$x < [\pi p_i]$

$\therefore$ Only x is $\underline{0}$.

▲ Now if:-

$$x \equiv 1 \pmod{p_1}$$
$$x \equiv 0 \pmod{p_2}$$
$$\vdots$$
$$x \equiv 0 \pmod{p_n}$$

$\therefore$ we can deduce that

$$x = \left(\prod_{i=2}^{n} p_i\right) \cdot k$$

Also, $k < p_1$ $\because$ ~~x~~ $x \in [0, \prod_{i=1}^{n} p_i - 1]$

ie: $x = \left(\prod_{i=2}^{n} p_j\right) k \equiv 1 \mod p_1$

That is:- Given p, ~~v~~ ST: $\gcd(v, p) = 1$

$$x \cdot v \equiv 1 \pmod{p_1}$$

$$x = v^{-1} \pmod{p_1}$$

~ GCD (p, v)

using Euclids GCD algo for p, q ST: $p \geq q$.

```
GCD ( p, q)
if (q==0) return p
else GCD ( q, p%q).
```

How:? GCD of $(p, 0) = p$ ⟶ base case TRUE ✓ (1)

$GCD(a,b) = ma + nb$: $m, n \in \mathbb{Z}$

To show:-

$GCD(p,q) = GCD(q, p \% q)$. — (2)

If this is proved then the algo is correct.

Let $g = GCD(p,q)$.

$g_2 = GCD(q, p \% q)$

then $gcd = g = \max\{y : y \mid p \text{ and } y \mid q\}$ — (3)

& $g_2 = \max\{y : y \mid q \text{ and } y \mid p \% q\}$ — (4)

To show (3) = (4)

way 1 If $y \mid p$ & $y \mid q \Rightarrow y \mid q$, $y \mid p \% q$

~~$y \mid q$~~ - trivial.

$y \mid (p \% q)$ - to show.

$p = m_1 y$ $\quad q = m_2 y$

Now, $p \% q = p - mq$.

$= m_1 y - m m_2 y$.

$\Rightarrow y(m_1 - m m_2)$

$\therefore y \mid p \% q$.

$\therefore$ (3) $\subseteq$ (4).

way 2: $q = m_2 y$ $\quad p \% q = m_1 y$

$p \% q = p - mq$.

$m_1 y = p - m m_2 y$

$p = y(m_1 - m m_2)$

$y \mid p$

$\therefore$ (4) $\subseteq$ (3)

$\Rightarrow \boxed{g = g_2}$ Algorithm correctness.

Bases here for O notation do not make a big difference ∵ only const changes

Thm: Euclidian GCD runs in $(O(\log_2 p))$
(Depth of recursion tree = $\log p$)

Proof: In 2 steps, the higher operand is halved
ie:- ↳ claim

(p, q)
↓
$(q, p \% q)$
$(p \% q, q \% (p \% q))$

∴ $2\log_2 p$ steps.

To show, $p \% q \leq \frac{p}{2}$

Case: $q > p/2$
$p \% q = p - q < p/2$

Case: $q <= p/2$
$p \% q < q$
$p \% q < p/2$

Coming Back: $g = GCD(p, q)$
$g = xp + y q \quad x, y \in \mathbb{Z}$.

$p = q m_1 + r_1 \Rightarrow r_1 = 1 \cdot p + (-m_1) q$
$q = r_1 m_2 + r_2 \Rightarrow r_2 = 1q + (-m_2)(1p - m_1 q)$
$= q - m_2 p + q(1 + m_1 m_2)$
$r_1 = r_2 m_3 + r_3$
⋮
$r_{n-2} = r_{n-1} m_n + r_n$ (0).

∴ finally g_{n-1} can be written as

→ Now for $\gcd(v, p) = 1$
$v^{-1} \pmod p = ?$
$1 = xp + yv$
$1 \% p = yv \% p$ ∴ $1 \equiv yv \pmod p$
∴ $v^{-1} = y \bmod p$

∴ Solution for the CRT prob :-

$$x = \sum_{i=} a_i \left\{ \left(\frac{\prod_{j=1}^{n} p_j}{p_j} \right) \left(\frac{\prod_{j=1}^{n} p_j}{p_i} \right)^{-1} \pmod{p_i} \right\} \bmod \prod_{i=1} p_i$$

Eg:

$x \equiv 2 \pmod 3$

$x \equiv 3 \pmod 5$

$x \equiv 2 \pmod 7$

$m = 105$

$b_1 = -1 \leftarrow \frac{m}{m_1} = \frac{105}{3} = (35 \times 2) \times 2$

1 $\frac{m}{m_2} = (21 \times 3) \times 1$

1 $\frac{m}{m_3} = (15 \times 1) \times 2$

$\sum = (140 + 63 + 30) \bmod 105$

$= 23$

35 = x3 + 2 ∴ 35-3

3 = 2x1 + 1 ∴ 3-2x

3 - 2x(35 - 8x1)

3 - 35x2 + 3x(x) -35x2 + 3(4x

35 = 3x11 + 2

3 = 2x1 + 1

1 = 1x1 + 0

1 = 3 - 2x1

= 3 - 1(35 - 3·11)

= 3 + 3·11 - 35

= 3(12) - 35

(21, 5)

21 = 5x4 + 1

5 = 1x5 + 0

1 = 21 - 5·4

15 = 7x2 + 1

TIP: Given primes 11, 7, 5, 3, 2

Rep :-

	11	7	5	3	2
23 =	1	2	3	2	1
11 =	6	3	2	2	1
Add.	7	5	0	1	0

Similarly for multiplication Mult 6 6 1 1 1

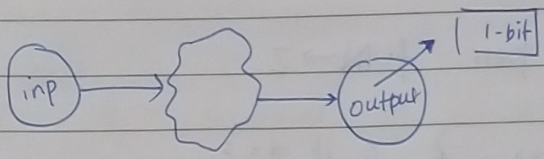
Solution is a turing machine

Set of solutions ≪ set of problems

- Are there computational prob for which no sol exists?

→ If the problem has infinite bit output → unsolvable computational prob.

→ Consider the model wherein



→ For an n-bit output consider it as k problems

→ Problem is a subset of input space

→ Characterize the problem by understanding whether the input is a part of the T set or F set.

Eg: Is graph G connected

- ↳ $G \in \{G \mid G \text{ is connected}\}$ ⟶ T or F

- ↳ $n \in \{n \mid n \text{ is prime}\}$ ⟶ T/F.

Now the subset of the input space (Σ^*) is essentially a language.

⚠ No. of problems >>> No. of solns.

Fact 1: If ∃ a bijection from A to B
$f: A \to B$ then $|A| = |B|$.

Fact 2: If ∃ a one-one fn $f: A \to B$ but no onto fn then $|B| > |A|$.

When dealing with infinities we use bijection fn.

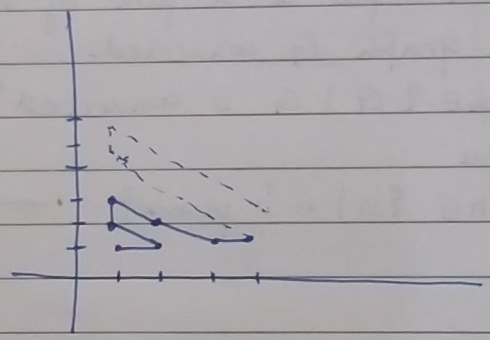
- A set is countable if ∃ a bijection $f: \mathbb{N} \to S$

→ Using diagonalization we can show no: of problems is uncountable but no: of C programs ~~are~~ countable

- Bijection from $f: \mathbb{N} \to \mathbb{Z}$.

$$f(x) = \begin{cases} 0 & \text{if } x = 1 \\ x/2 & \text{if } x \text{ is even} \\ -\frac{(x-1)}{2} & \text{if } x \text{ is odd} \end{cases}$$

- Bijection from $f: \mathbb{N} \to \mathbb{N} \times \mathbb{N}$.



Every pt. is visited exactly once.

Thm: $(0,1)$ is uncountable

Proof: Suppose the contrary.
Let $f: \mathbb{N} \to (0,1)$ is a bijection

∴, Let $f(1) = 0.d_{11}\,d_{12}\,d_{13}\ldots$
$f(2) = 0.\overline{d_{21}}\,d_{22}\,d_{23}\ldots$
$f(3) = 0.d_{31}\,\overline{d_{32}}\,d_{33}\ldots$

$\infty \times \infty = \infty$

$2^{\infty} \neq \infty \to \infty$ another ∞

Consider an element $x \in (0,1)$

ST:

$x = 0.\bar{d_{11}}\ \bar{d_{22}} \ldots \bar{d_{ii}} \ldots$

However this x has not occured on the RHS for any f(natural no.) $\therefore$, # contradiction

Thm: $2^{\{0,1\}^*}$ is uncountable

$\{0,1\}^*$:- strings

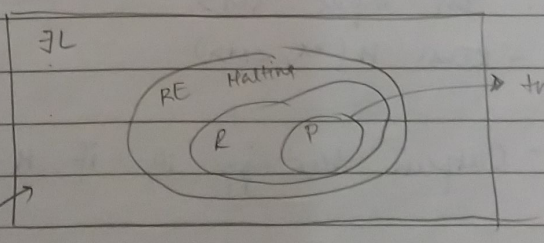
Pf: Let if possible
$f: N \to 2^{\{0,1\}^*}$ be a bijection

Let $f(1) = d_{11}\, d_{12}\, d_{13} \ldots$
$f(2) = d_{21}\, d_{22}\, d_{23} \ldots$

where each $d_{ij} \in \{0,1\}$
indicating whether an element $\in$ a subset or no

Create an $x = d_{11}\, d_{22}\, d_{33} \ldots$
by toggling the diagonal elements of each function values.
This x doesn't occur in the RHS
$\therefore$, # contradiction.

Therefore set of all programs $\subseteq \{0,1\}^*$
$\therefore$, C prog countable.



→ L - Turing Recognizable :- RE

If ∃ a machine M such that:

a) $w \in L$ $M(w)$ = accept

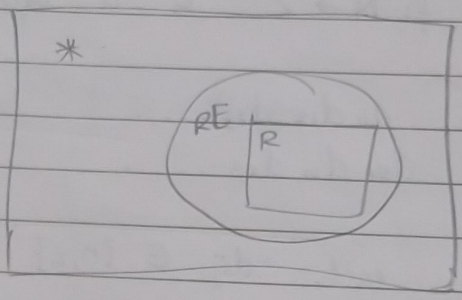
b) $w \notin L$ $M(w)$ = not accept. w.

→ L - Turing Decidable

if ∃ TM (M) → decides such that

a) $\forall\ w \in L$; M acc w

b) $\forall\ w \notin L$; M rejects w



Undecidability of languages

Consider an $A_{TM} = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$

Thm: A_{TM} is undecidable

Proof: Suppose A_{TM} were decidable say a decider H exists for A_{TM}.

$\forall\ \langle M, w \rangle \in A_{TM}$, $H(M, w)$ = accept

$\forall\ \langle M, w \rangle \notin A_{TM}$, $H(M, w)$ = reject

D: on input $\langle M \rangle$

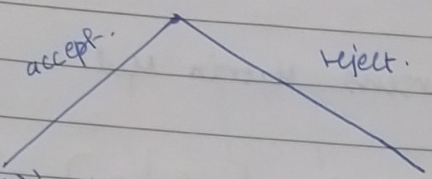
- Run $H(M, \langle M \rangle)$

- Output the opp if H accepts, rejects

Now see what is $D(\langle D \rangle)$?

If something is accepted by $D(\langle D \rangle)$ then $H(D, \langle D \rangle)$ rejects.

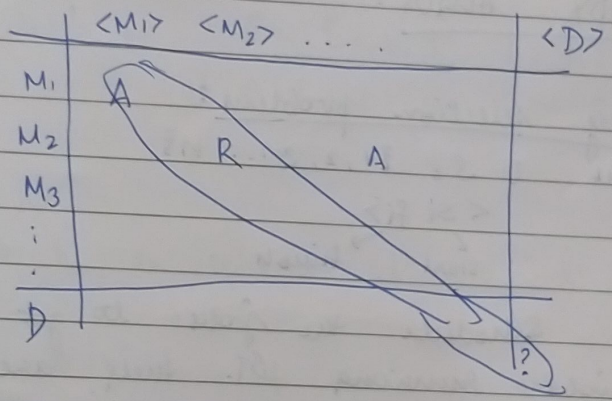
~~but then D~~



$H(D, \langle D \rangle)$ reject
$\Rightarrow D(\langle D \rangle) \notin A_{TM}$
D doesnt accept $\langle D \rangle$.

$H(D, \langle D \rangle)$ = accept
$\Rightarrow D(\langle D \rangle) \in A_{TM}$
$\Rightarrow$ D accepts $\langle D \rangle$.

Consider:



i^{th} entry : output of H when run on $H(M_i, \langle M_i \rangle)$

→ Now Consider a D s.t: it takes all diagonal entries & toggles

Thm: If $L \in RE$ & $\bar{L} \in RE$ then $L \in R$.

Now what about $\overline{A_{TM}}$?

$A_{TM} \in RE$ $A_{TM} \notin R$.

$\Rightarrow \overline{A_{TM}} \notin RE$

Review:

→ Divide & Conquer :-
- Merge sort
- Karatsuba
- FFT
- Starsen's Matrix Mul.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

8 mult
Strasen – 7
check the book

GREEDY ALGOS:

(1) Activity selection problem:

Input $S = \{1, 2, 3 \ldots n\}$

$\langle s_i \; f_i \rangle$ (start, finish)

Task: Schedule the jobs to get max no. of jobs running ST. they are non overlapping. [no implication to choose all jobs]

2 act i, j are non overlapping iff

$s_i \geq f_j$, $s_j \geq f_i$

Start by assuming:-

Sort f_is in non-decreasing order:

ie: $f_1 \leq f_2 \leq f_3 \ldots \leq f_n$

$A \leftarrow \{1\}$

$j \leftarrow 1$

for $i = 2$ to n

if $(s_i \geq f_j)$ or $(s_j \geq f_i)$ then

$A \leftarrow A \cup \{i\}$

$j \leftarrow i$

Proof of the correctness:- (Greedy choice Prop)

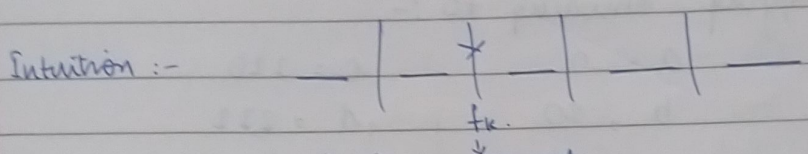
$\exists$ some opt. A^* ST $1 \in A^*$ ... (to show).

if $1 \notin A^*$ consider $B = A^* - \{k\} \cup \{1\}$

where $k \in A^*$

↑ smallest el.

Intuition :-



$f_1 \leq f_k$

∴, greedy choice prop holds

Thm: Optimum Sub Structure Property:-

Opt sol to $S = \{1, 2, 3, \ldots n\}$

$= 1 \cup$ {opt sol to $S' = \{i \mid s_i > f_1\}$}

Pf: suppose not.

Let B be an optimum sol^n to S'

then

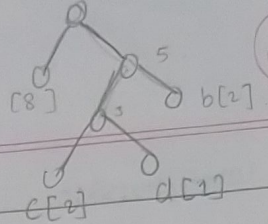
$B \cup \{1\}$ if this isn't optimum then there exists a bigger soln for A^*. Now excluding 1 from that, you get a bigger optimum sol for S' which is not B ∴, # contr.

(2) Huffman Codes:

Prefix free encodings avoid ambiguity.

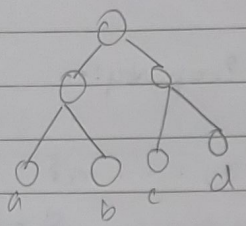
- Task: Given a string what is the optimum prefix free encoding number of bits required

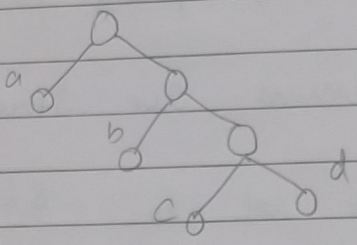
a = 8
b = 2
c = 2
d = 1



Consider
str = abbaaccabbaaad
with encoding as
a = 00 c = 10
b = 01 d = 11.

No: of bits = 30
Modifying encoding to :-
a = 0 c = 110
b = 10 d = 111
No: of bits = 25 ! :0 :0 :0

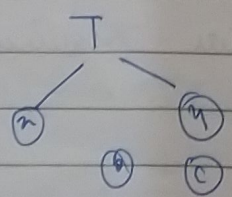




Going left = 0 Right = 1

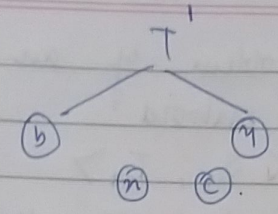
Algo:-
Compute frequencies of all letters
construct BT as follows:
~ Look at 2 least occuring freq & place them at the bottom (bottom up)
~ Create a parent for that st: parent freq = $c_1 + c_2$

Thm: Greedy Choice property proof.
Suppose x, y are the 2 least frequent occ alphabets st: $f(x) \leq f(y)$.



let $f(b) \leq f(c)$

Swap b & x



$\therefore$ cost (T) - cost (T') — ①

= (In T depth of x) * freq (x) + $d_T(B) \cdot f(B)$ - $d_{T'}(x) \cdot f(x)$ - $d_{T'}(B) \cdot f(B)$

$d_T(x) \cdot f(x) + d_T(b) \cdot f(b) - d_T(b) f(x) - d_T(x) \cdot f(b)$

① = $[f(x) - f(b)]\,[d_T(x) - d_T(b)] \geq 0$.

$\therefore$ cost of T'' $\leq$ cost T

Optimum sub structure :-

Opt : Σ, freq

x & y least freq.

$\Sigma' = (\Sigma \setminus \{x, y\}) \cup \{z\}$

$f(z) = f(x) + f(y)$.

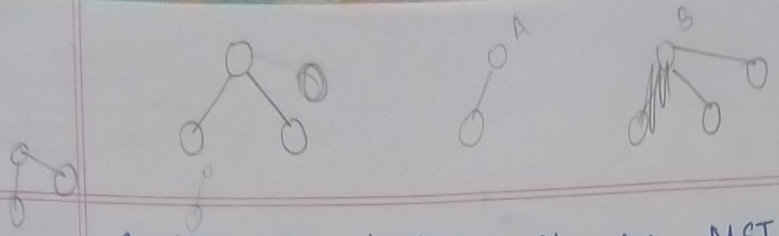
(2) MATROID PROBLEM:

M = <S, I> ... a matroid where

S : Non empty set

I : non empty collection of S

- Herditary property - If $A \in I$, $B \subseteq A$ then $B \in I$
- Exchange property - If $A \in I$, $B \in I$ & $|B| > |A|$ then $\exists\, x \in B \setminus A$ ST: $A \cup \{x\} \in I$.



Consider the problem of an MST:-

Let M be the matroid

$M = \langle S_G, I_G \rangle$

$S_G = E$ (set of all edges)

$I_G = \{ A \subseteq E \mid A \text{ is acyclic} \}$.

M is a matroid.

Thm: I_G has heridetary property.

Pf: $A \in I_G$ ... assume

then $\exists\ B \in I_G$ such that $B \subseteq A$

$\therefore$, B doesn't contain cycles either

$\therefore$, $B \in I_G$

Thm: I_G has exchange property.

Pf: let

$A \in I_G$ and $B \in I_G$ ST: $|B| > |A|$

$(n - |A|)$ trees $>$ $(n - |B|)$ trees

then

$\exists$ at least one edge in $n - |B|$ such that it connects 2 trees in $n - |A|$. If we add this edge to $|A|$ it should still be acyclic.

$\Rightarrow$ M is a matroid.

→ Weighted Matroids :

Positive weights for elements in S

– Problem:

Maximum weighted independent set in a matroid uses greedy approach.

Greedy algorithm :

$A \leftarrow \phi$

i) Sort all elements in S in non-decreasing order of weights & in order.

if $A \cup \{x\} \in I$:

$A \leftarrow A \cup \{x\}$.

~ Greedy choice property:

if $\{x\} \in I$ (max wt.)

$\Rightarrow \exists$ opt. A $x \in A$ (To show)

B : opt. soln ; $x \in B$

$\{x\} \in I$

$B - \{y\} \cup \{x\} \in I$.

$w(B) - w(y) + w(x) \geq w(B)$

$\therefore$ $\exists$ at least one optimal sl. $x \in B$

~ Optimal substructure property :-

$M' = (S', I')$

$S' = \{ y \mid \{x, y\} \in I \}$

$I' = \{ A \in I \mid A \cup \{x\} \in I \}$

→ ln(n) [ln|S|] approximate set-cover :-

Input - S, F

Ouput - min. no: of elements from F that covers S.

F is a set of subsets of S

$F \subseteq 2^S$

Greedy: Pick elements from F which covers max no: of elements not yet covered elements of S.

To prove: If greedy strategy picks t elements from $\mathcal{F}$ then $t \leq \ln(|S|) \cdot k^*$

Proof: $n = |S|$

n_i : no. of els to be covered after i^{th} iteration

$$n_0 = |S| \quad -(1)$$

$$n_1 \leq n_0 - \frac{n_0}{k^*} \Rightarrow n_1 \leq n_0\left(1 - \frac{1}{k^*}\right)$$

$$n_2 \leq n_1 - \frac{n_1}{k^*} \Rightarrow n_2 \leq n_0\left(1 - \frac{1}{k^*}\right)^2$$

$$\vdots$$

$$n_t \leq |S|\left(1 - \frac{1}{k^*}\right)^t \quad \text{from } (1).$$

$$n_t \leq |S|\left(e^{-\frac{1}{k^*}}\right)^t$$

$$e^{-t/k^*} < |S|$$

$$t \leq k^* \ln|S|.$$

REDUCTIONS: (In specific: mapping relations defined as $A \leq_m B$)

A is said to be mapping reduced to B if $\exists$ a computable function f

$$f: \Sigma^* \rightarrow \Sigma^*$$

$$x \in A \iff f(x) \in B$$

Apply $f(x)$ to x & check if $f(x) \in B$
If so, $x \in A$.

So basically convert input string to something for B

If A is red to B then B is at least as hard as A.

→ Halting Problem: Given a program & its input can you predict whether the program halts or not.

$HALT = \{ \langle M, w \rangle \mid M \text{ halts on input } w \}$

To show: $A_{TM} \leq_m HALT$. [We know A_{TM} is undecidable]

Proof: If a TM R decides HALT

Decides for A_{TM} :-

On input $\langle M, w \rangle$

- Run R ($\langle M, w \rangle$)
- If R rejects, REJECT.
- If R accepts execute M(w) & accepts if M accepts & REJECTS otherwise

Eg 2: $E_{TM} = \{ \langle M \rangle \mid L(M) = \phi \}$ Empty TM

E_{TM} is undecidable

$A_{TM} \leq_m E_{TM}$

TM R decides E_{TM}.

Decides for A_{TM} on input $\langle M, w \rangle$

- Create a new M'

On input x

If $x \neq w$ Reject

If $x = w$ run M on w and accept if M accepts.

Run R on ($\langle M' \rangle$)

if R accepts → Reject.

R rejects → Accept.

Eg 3: $EQUAL_{TM} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \}$

$EQUAL_{TM}$ is undecidable

↳ To find $A_{TM} \leq EQUAL_{TM}$.

Further use Eg2

$E_{TM} \leq EQUAL_{TM}$.

Let TM R decides $EQUAL_{TM}$

For $EQUAL_{TM}$ On input $\langle M \rangle$

- Create a new M'

On input x, REJECT.

Run R(M, M')

- if R accepts - ACCEPT
- if R rejects - REJECT.

Eg4: $REG_{TM} = \{ \langle M \rangle \mid L(M) \text{ is regular} \}$.

To check $A_{TM} \leq_M REG_{TM}$

Let R decide REG_M

Create a TM M'.

On input x if $x \in 0^n 1^n$ accept else

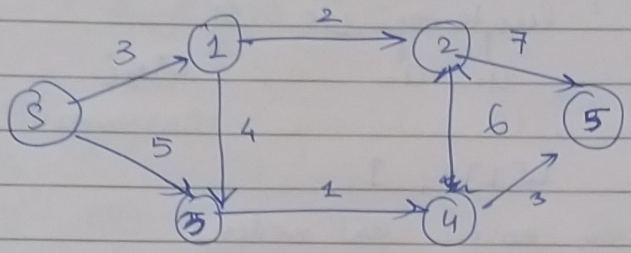
$x \neq 0^n 1^n$ accept if M accepts w

- Run R($\langle M' \rangle$)
- If R accepts ACCEPT
- If R rejects REJECT

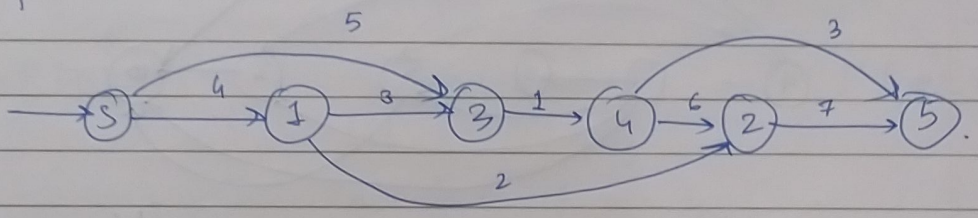
DYNAMIC PROGRAMMING: (a sledge hammer of algorithms).

Can we divide the problem into a DAG of sub-problems. So that no deadlock occurs.

Example #1: Shortest path in DAG:-



Topo sort:-



$$S[i] = \min_{j|(j,i) \in \text{Edges}} \{ S[j] + w(j,i) \}$$

$S[s] = 0$

$S[1] = 3$

$S[3] = \min(5, 3+4) = 5.$

$S[4] = 6$

$S[2] = \min(3+2, 5+1+6)$

$= 5.$

→ **Longest Increasing Sub Sequence:**

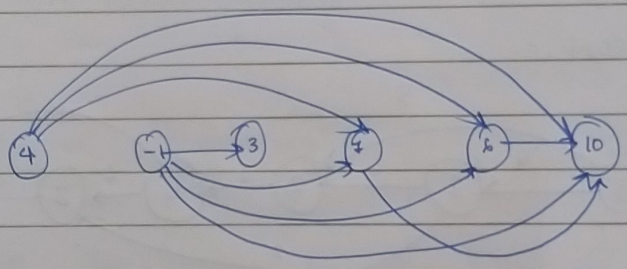
- Is a sequence of no:s where youire only allowed
- $a_1\ a_2\ a_3 \ldots a_9\ a_{10}$
  ↳ $a_1\ a_4\ a_7$.

Eg: 4, -1, 3, 7, 6, 10.

L[i] = length of LIS of $a_1, a_2 \ldots a_i$

We know: L[1] = 1.

Consider the no:s as nodes:-



L[i] → all LIS ending @ a_i

$$L[i] = \underset{\text{over all } i}{\text{Max}} \left\{ \max_{j(i,j) \in \text{edge}} \{ 1 + L[j] \} \right\}$$

- **Edit Distance:**

Inputs: 2 data strings.

Eg min no: of changes in terms of insert, delete, override.

SON
ONE

2 : delete S add E

Given strings :-

$x_1 \ldots x_n$

$y_1 \ldots y_m$

$E[i,j]$ = edit distance b/w

$x_1, x_2 \ldots x_i$

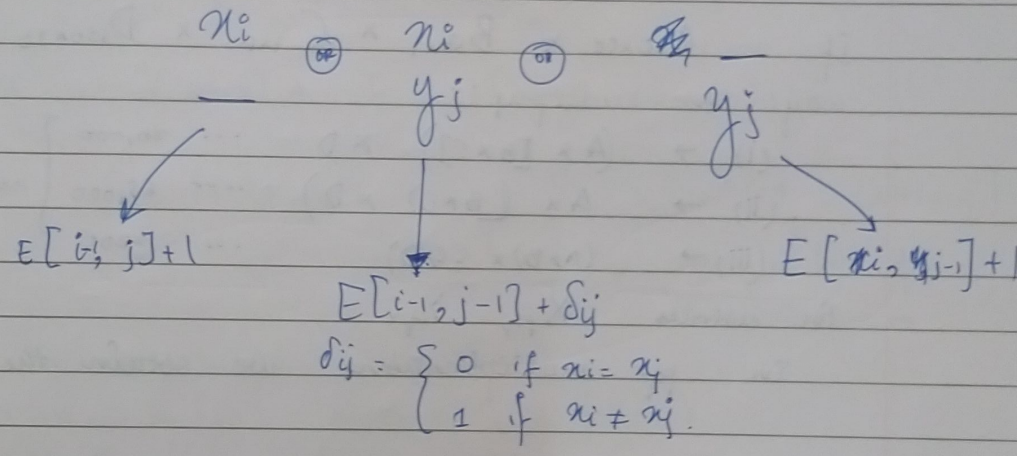
$y_1, y_2, \ldots y_j$

Goal → $E[n, m]$

Rewrite :-

delete (S O N — → write
— O N E)

∴, The 3 alignments possible are :-



$\delta_{ij} = \begin{cases} 0 & \text{if } x_i = x_j \\ 1 & \text{if } x_i \neq x_j \end{cases}$

∴,

$$E[i,j] = \begin{cases} i & \text{if } j == 0 \\ j & \text{if } i == 0 \\ \text{Min}\left[1 + E[i-1, j],\ 1 + E[i, j-1],\ E[i-1, j-1] + 1 \right] \end{cases}$$

		0	1 E	2 A	3 R	4 T	5 H
0	H	0	1	2	3	4	5
1	H	1	1	2	3	4	4
2	E	2	1	2	3	4	5
3	A	3	2	1	2	3	4
4	R	4	3	2	1	2	3
5	T	5	4	3	2	1 →	2

CHAIN - MATRIX MULTIPLICATION:

Input: n matrices $A_1, A_2, \ldots A_n$

Matrix A_i is of $p_{i-1} \times p_i$

∴, Output

$\prod_{i=1}^{n} A_i$ ie: $A_1 \times A_2 \times \ldots \times A_n$

$= (p_0 \times p_n)$ matrix

(Matrix multiplication is associative!)

If $A_{20\times 50} \times B_{50\times 1} \times C_{1\times 100} \times D_{100\times 10}$

ways to multiply:-

(i) → $(A \times [B \times C]) \times D$ ... 20,000

(ii) → $A \times ((B \times C) \times D)$ ... 65,000 } mult.

(iii) → $(A \times B) \times (C \times D)$ ... 2200

For matrices X_{pq} Y_{qr}

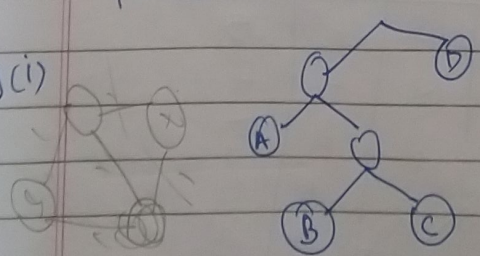
In pqr multiplications we obtain the answer

Aim: Minimize the no: of multiplications by tactfully parenthesising.

No: of ways in which you can parathesise grows exponentially: (Catalan numbers)

Parse trees can be drawn for the paranthesised formed:-

Eg (i)



→ No: of ways to parantheisise the expression is the total number of trees that can be constructed with given the no: of matrices.

Idea : [Leaves of the tree are your matrices.
: Each subtree must be an optimal subtree.]

∴, find optimum subtrees & build the complete subtree.

let C[i,j] = Ai ... Aj optimum cost of multiplying Ai to Aj

Base case : C[i,i] = 0 [Single matrix]

$$C[i,j] = \min_{i \le k \le j} \begin{cases} C[i,k] + C[k+1,j] \\ + p_{i-1} \times p_k \times p_j \end{cases}$$

Create a C[i,j] matrix as follows.

	1	2	3	4
1	0	1000	3000	22000
2	-	0	5000	1500
3	-	-	0	10000
4.	-	-	-	0

For C_{13} :-

k = 1 :- $C_{11} + C_{23} + 20 \times 50 \times 100$

$= 0 + 5000 + 100000 = 105000$

k = 2 :- $C_{12} + 0 + 2000$

$1000 + 2000 = 3000$

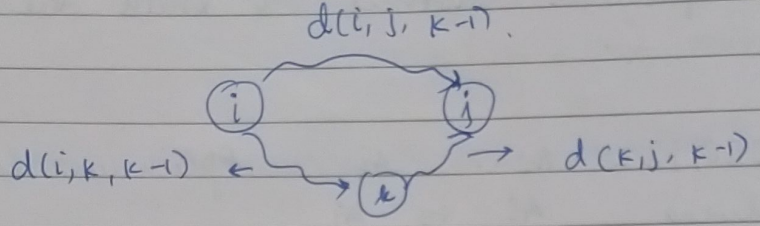
memoization

Floyd Warshal : APSP :

Define

$d(i, j, k)$: shortest distance from i to j using vertices in $[1, k]$

Now, $d(i, j, 0) = w_{ij}$



If node k is also used then it must be used exactly once ∵ loops won't exists. (neg. cycles).

$$d(i,j,k) = \min \begin{cases} d(i,j,k-1), \\ d(i,k,k-1) + d(k,j,k-1) \end{cases} \quad (*)$$

Algorithm : $\forall [i,j] \in [1,n]$

$$d(i,j,0) = \begin{cases} w_{ij} & \text{if } ij \in E \\ 0 & \text{otherwise.} \end{cases}$$

```
for (k = 1 to n)
    for (i = 1 to n)
        for (j = 1 to n)
            d(i, j, k) = Min of (*)
```

Knapsack

Wts : $w_1, w_2, \ldots w_n$

Costs : $c_1, c_2, \ldots c_n$.

→ Standard greedy approach: Be greedy on c_i/w_i. Doesn't always give optimum answer.

→ Fractional knapsack :- greedy works

Target wt: W (shouldn't exceed)

wt	val
6	30
3	14
4	16
2	9

Let if possible c_i belong to the optimum sol.
then:

$$K(W) = c_i + K(W - w_i)$$

Now to find for all i:-

$$K(W) = \max_i [c_i + K(W - w_i)]$$

Order: $O(nW)$

0	1	2	3	4	5	6	7	8	9	10
0	0	9	14	18	23					

FINDING MEDIAN in LINEAR TIME:

Order statistics : find the k^{th} smallest element.

Select $(A[n], k)$

Choose pivot x

L_x $\{y \mid y < x\}$.

R_x $\{y \mid y \geq x\}$

if $k \leq |L_x|$

select (L_x, k) → superlinear algo

else

select $(R_x, k - |L_x|)$

→ Select (A, k)

① Divide the n elements into $\lceil n/5 \rceil$ groups of 5 each

② Sort each group & pick $\lceil n/5 \rceil$ medians resp (each groups 3rd el. is its median).

③ Select $(M, \lfloor \frac{n}{10} \rfloor)$ recursively find median of medians

④ Pivot on x:

$L_x = \{y \mid y < x\}$

$R_x = \{y \mid y \geq x\}$.

⑤ if $k \leq |L_x|$

return Select (L_x, k)

else

return Select $(R_x, k - |L_x|)$

(Technical change:

we can include another partition namely

$E_x = \{y \mid y = x\}$

then step ⑤

↳ else if $(|L_x| < k \leq |L_x| + |E_x|)$

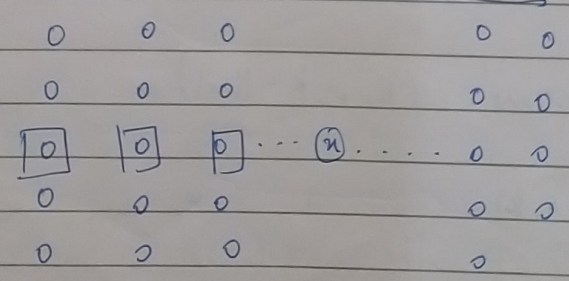
return x

return select $(R, k - |L_x| - |E_x|)$)

&; the recurrance rel is :-

$$T(n) = T\left(\frac{n}{5}\right) + \text{Max}\left(T(|L_x|) \to T(|R_x|)\right) + O(n)$$

Consider :- ← n/5 groups →



3 elements per group that are greater than x

$n/10$ groups whose median is $> x$

$\therefore$ $|L_x| \leq \frac{7n}{10}$

Similarly for each of the $n/10$ groups there are 3 elements lesser than x ~~(at least)~~

$\therefore$, $3n/10$ els do not belong to R_x

$$\therefore \quad |R_x| \leq \frac{7n}{10}.$$

Modifying the recurrance:-

$$\boxed{T(n) = T(n/5) + T\left(\frac{7n}{10}\right) + O(n)} \quad -(1)$$

Now, to verify whether this is ~~this~~ linear or not, if (1) is linear then

$$dn \leq \frac{dn}{5} + d\frac{7n}{10} + cn$$

ie: for every c I a d.

$$dn \leq n\left[d(0.2 + 0.7) + c\right]$$

For this inequality to hold. $d = 10.c$

Here the hidden constant is $10c$.

Hence yes, this is a linear-time algorithm.

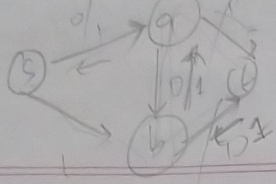
⚠ In case of splitting into groups of 5, if you choose groups of 3, the recurrance becomes

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{12n}{5}\right) + cn$$

Checking :-

$$dn \leq \frac{dn}{3} + \frac{2dn}{3} + cn = n[d + c] \quad \times \text{ not possible.}$$

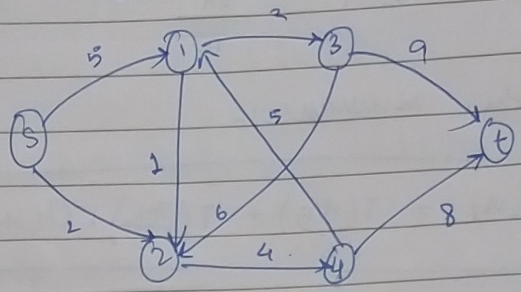
- However 7 can also be used.
- Randomly picking a pivot also works in E[linear time]



NETWORK FLOWS :

→ no self loop

Consider a weighted graph.



- Find the max flow network.
- Naive method : exponential.

Max_flow = Min_cut.

→ conservation of flow
→ non neg
→ capacity

Ford - Fulkerson Algo :

flow ≤ capacity
↳ local constraint

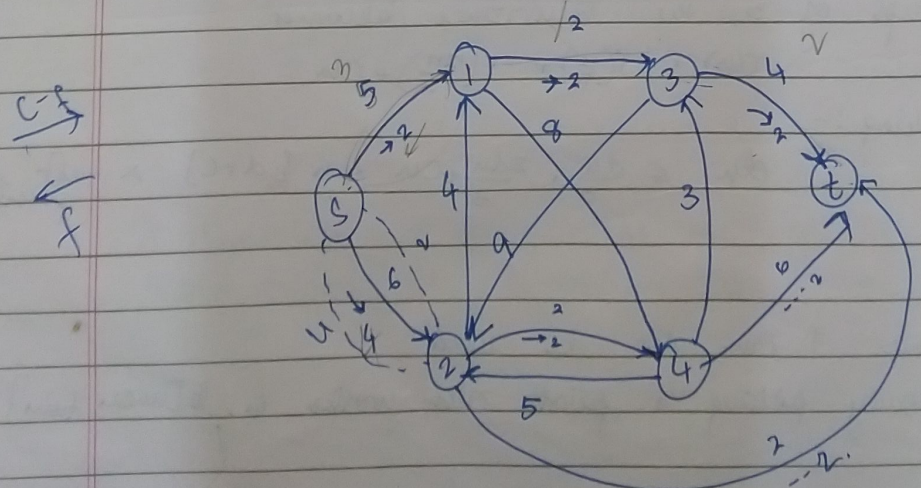
Thm. Max Flow = Min_Cut.

Max flow ≤ Min_cut

2^n cuts possible.

Take min (max value of flow for each cut).

Let Initial flow f = 0



Reverse edge :- decreasing the flow
necessary for augmentation

Is there an augmented path from s to t.

Yes :- $s \xrightarrow{3} 1 \xrightarrow{3} 4 \xrightarrow{3} t$

Now in the new graph

$s \xrightarrow{1} 2 \xrightarrow{1} 1 \xrightarrow{1} 4 \xrightarrow{1} t$

Bottleneck $4 \to t$ we have 1.

$|E_f| \leq 2|E|$ → Edges in flow $\leq$ 2 total edges

Keep creating the residual graphs accordingly.

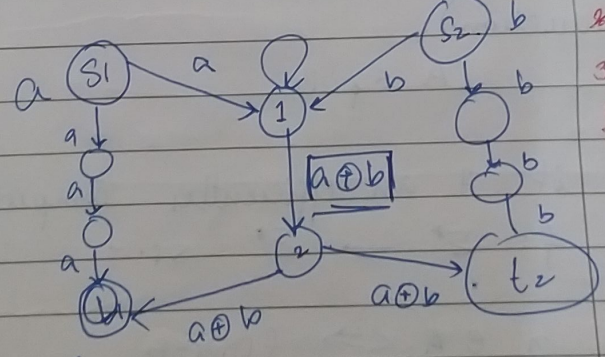
Repeat until no path from s to t.

To show that this actually gives us the max flow

[There must exist a partition such that $(S) \Rightarrow (V/S)$]

$c = f$

Define : $S = \{ u \mid \exists\ (s,u) - \text{path in } G_f \}$.



Can obtain b from a & a⊕b.

Max flow - min cut th.

1. f is max flow in G
2. Residual G_f contains no augmenting path
3. $|f| = c(S,T)$ for some cut (S,T) of G

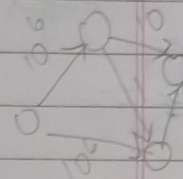
Complexity :
time to find a path in the graph = $O(E)$ → DFS, BFS
$\downarrow V \leq E$

FF :- $O(E * f^*)$ → f^* : max flow

→ Linear Programming :-
- can be used to solve max flow problem.

● QUIZ QUESTION: (2 parts)

Part 1 show that Ford Fulkerson Algo is not efficient in the worst case by giving an example network where the algo is slow ("super poly" in $|V|$).



Part 2 : Show that if the augmenting path in the residual graph is chosen via BFS (ie: the path with least no: of edges from s to t
then max flow algo runs in $O(|V| \times |E|^2)$ ← ??

Edmonds Karp Algorithm

min no. of edges

× ——— ×

→ Polynomial - time reductions :-
(Karp reduction).

$$A \leq_p B$$

$A \leq_p$ if ∃ an efficiently computable fn $f: \Sigma^* \rightarrow \Sigma^*$ ST:-

$$\forall \; x \in A \iff f(x) \in B$$

{ Max clique has greater than single bit output
∴ it isn't even a language }

• 3 SAT $\leq_p$ CLIQUE

→ 3-SAT :

Input : Boolean formula in 3 CNF (Product of Sums)

$$\phi = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (.. \vee .. \vee) \wedge \ldots$$

Output : Yes if ϕ is satisfiable else no.

⇒ 3-SAT = $\{ \langle \phi \rangle \mid \phi$ is satisfiable 3-CNF boolean formula $\}$

→ Clique :-

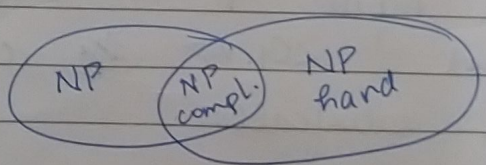
$= \{ \langle G, k \rangle \mid G$ has a complete subgraph of size $k \}$

A language L is NP-hard if :-
$\forall L' \in NP$

$L' \leq_p L$

L is NP-complete if
a) L is NP hard.
b) $L \in NP$

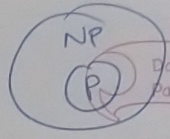
NP complete = NP $\cap$ NP-Hard.



~~NP: Set of languages~~

NP: Class of languages that have poly^n time verifiers

Every decider is a verifier



→ NP: Class of languages that have polynomial time verifiers.

ie: ∃ a TM V such that.

$$L = \{ x \mid \exists u, V(x,u) = \text{accept} \}$$

To Show: 3-SAT ∈ NP

Consider V on input (ϕ, u)

→ Checks if u is a satisfying assignment for ϕ accept if so else reject.

To Show: Clique ∈ NP:

V on input $\langle G, k \rangle$ and u

Where u is the clique given to you.

Again Check if u is satisfied.

Accept if so else reject.

Going Back to $3_{SAT} \leq$ clique

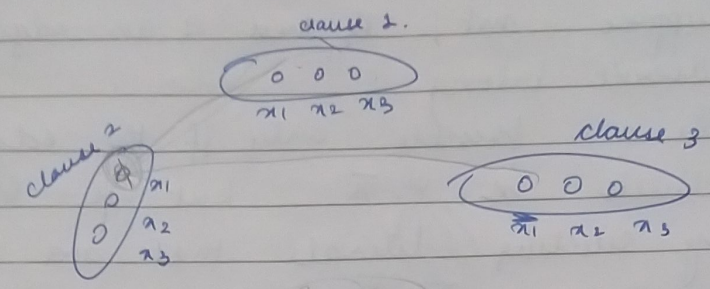
Given an input ϕ in 3-CNF

can I find $\langle G, k \rangle$ clique iff ϕ is satisfiable.

ϕ = n variables
l clauses.

Now, let G have 3l vertices

ie:



Step 1: Connect 2 vertices u, v iff they belong to different clauses and they are not related contradictarily

Step 2: Choose $k = l$

This G has a clique of size l iff ϕ is satisfiable.

- Part 1: If $\exists$ a clique of size l in G $\Rightarrow$ ϕ satisfiable

Max size clique possible = l sized clique

$\therefore$ If G has a clique of size l

$\exists$ exactly 1 node per clause in the clique

No contradictory labels.

Now, take every (literal) node in the clique and make it as true.

Now, each clause will be true $\Rightarrow$ ϕ satisfiable

- Part 2: If ϕ is satisfiable $\Rightarrow$ G has a clique of size l

Set an assignment:

every clause is true

at least 1 literal is true in every clause

Take the first true literal in every clause

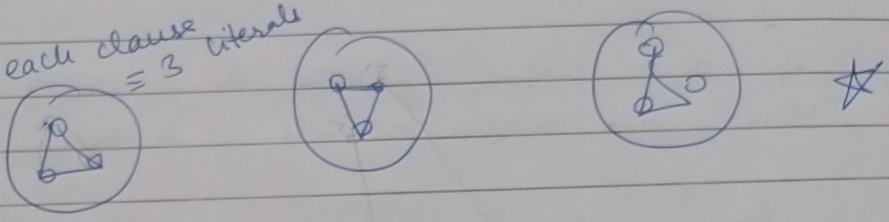
- Size l clique ... trivial
- Suppose not, then $\exists$ an edge which doesn't belong
  This is possible only if the edge is between x & $\bar{x}$ but we started by selecting literals that are 1
  $\therefore$, $x = 1 \quad \bar{x} = 1$ # Contradiction

$\therefore$, Clique exists

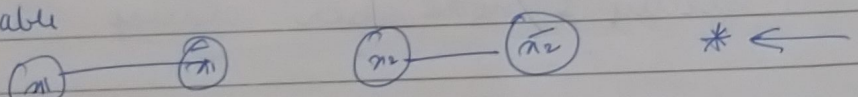
3 SAT $\leq_p$ VERTEX COVER.

$G(2n + 3l)$ vertices

each clause $\equiv$ 3 literals



each variable connect it to its complement



Connect every node from ★ to its corresponding literal in *

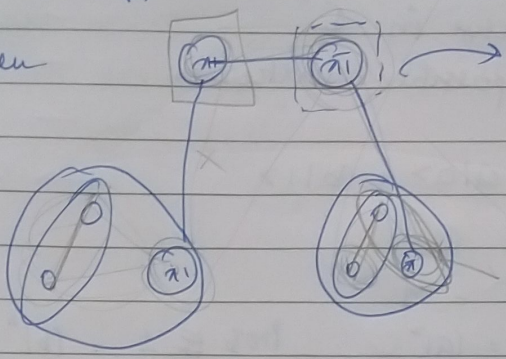
Min $k = 2l + n$

- $2l$: min 2 vertices per clause in ★
- n: one vertex per *

- Make all unchosen literals as true. To take care that unchosen ones are not x_i, $\bar{x}_i$ pairs

Suppose opp.

then



→ choosing this

⇒ $x_1 - x_1$ edge isn't covered :D

contradiction

Reverse :-

If ϕ is true :-

in Δ join two apart from the one you chose as true. And in * choose the one you left

Quantum Algorithms :

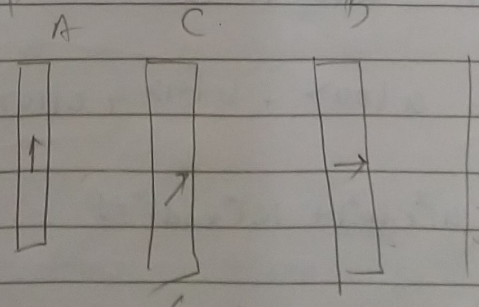
→ Intro to quantum mechanics

→ basic quantum gates & circuits.

→ Quantum teleportation.

→ Shor's Quantum Algo for integer factorization

→ Polarizers Exp



Introducing C :- to purify light

→ Qubit: → Hilbert space
- State vector in $\mathbb{C}^2$
- Only 2 possible outcomes.

- $|\psi\rangle = a|0\rangle + b|1\rangle$

$(a, b) \in \mathbb{C}^2$

$|a|^2 + |b|^2 = 1$

Prob of $0 = |a|^2$ Prob of $1 = |b|^2$

- superposition of all possible outcomes with varying degrees of prob.

- State collapses in-line with the observed/measured outcome

Any photon:-

$|\psi\rangle = a\,|\text{pass}\uparrow\rangle + b\,|\text{fail}\uparrow\rangle$

$|\nearrow\rangle = \frac{1}{\sqrt{2}}|\uparrow\rangle + \frac{1}{\sqrt{2}}|\rightarrow\rangle$

And $|\uparrow\rangle = \frac{1}{\sqrt{2}}|\nearrow\rangle + \frac{1}{\sqrt{2}}|\nwarrow\rangle$

→ 2-Qubits System:-
- State vector in $\mathbb{C}^4$

$|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$

and $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$

- Evolution postulates:

→ Nature allows these transformations to happen in reality unitary matrix U. (for UV to evolve)

$$U^{-1} = U^{+}$$

↓ conjugate transpose.

ToDo:

- Bizarness begets
- Profound bizzareness (4 famous eg:)
  1. Quantum cryptography
  2. Quantum Distributed Systems
  3. Quantum computers
  4. Is information real? Are we state vectors?

- Quantum teleportation: transmit objects as information & regenerate the object.

- Quantum entanglement:

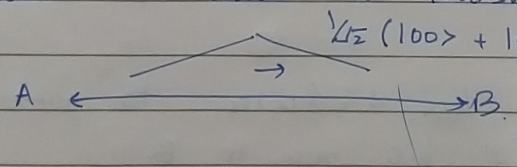
$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Both e spin up or both spin down.

spooky effects

2 single Q-bits systems cannot be represented as the above. It is ∴, an entanglement.

Consider A wants to transmit a qbit to B



$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\psi_{init}\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

$$= \frac{\alpha}{\sqrt{2}}|000\rangle + \frac{\alpha}{\sqrt{2}}|011\rangle + \frac{\beta}{\sqrt{2}}|100\rangle + \frac{\beta}{\sqrt{2}}|111\rangle$$

Protocol:

Step 1: A applies U_{CNOT} (control not).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$00 \rightarrow 00$
$01 \rightarrow 01$
$10 \rightarrow 11$
$11 \rightarrow 10$

- After step 1 (A apply CNOT B applies identity)

$$|\psi_1\rangle = \frac{\alpha}{\sqrt{2}}|000\rangle + \frac{\alpha}{\sqrt{2}}|011\rangle + \frac{\beta}{\sqrt{2}}|110\rangle + \frac{\beta}{\sqrt{2}}|101\rangle$$

Step 2: A applies U_H to the first qbit.

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

$|1\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

- After step 2:-

$$|\psi_2\rangle = \frac{\alpha}{2}|000\rangle + \frac{\alpha}{2}|100\rangle + \frac{\alpha}{2}|011\rangle + \frac{\alpha}{2}|111\rangle +$$

$$\frac{\beta}{2}|010\rangle - \frac{\beta}{2}|110\rangle + \frac{\beta}{2}|001\rangle - \frac{\beta}{2}|101\rangle$$

Step 3: A measures her 2 Qubits to obtain $|xy\rangle$. A sends the 2 bits x, y to B.

Renormalising $|xy\rangle$	Bob's Qubit	Step 4 Apply gates	Bob's Qubit
$\|00\rangle$	$\alpha\|0\rangle + \beta\|1\rangle$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$: Identity	$\|\psi\rangle$
$\|01\rangle$	$\alpha\|1\rangle + \beta\|0\rangle$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$:	$\|\psi\rangle$
$\|10\rangle$	$\alpha\|0\rangle - \beta\|1\rangle$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\|\psi\rangle$
$\|11\rangle$	$\alpha\|1\rangle - \beta\|0\rangle$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	

→ Applying H transform

$$H|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H^2|00\rangle \rightarrow \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

H^3 . . . .

Similarly for n bits

Quantum Parallelism

$$H^n \underbrace{|000\ldots0\rangle}_{n} = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$$

$U(H^n|000\ldots\rangle)$ .... apply algo on superposition of all inputs. In one run of algo you get the output superposition of all possible inputs.

● Linear Programming : (SIMPLEX Algorithm)

Solve : Maximize $C^T \cdot x$ → $[x_1, x_2 \ldots x_n]$

(C^T : constant vector)

Subject to $Ax \leq b$

$x \geq 0$

Eg: Max $x_1 + x_2 = k$

such that $x_1 + 2x_2 \leq 6$

$x_1 - x_2 \leq 3$

$x_1 \geq 0 \quad x_2 \geq 0$

When will $(0,0\cdots0)$ be the solution ?

@ $(0,0\cdots0)$

→ Stop if C_i's are all ≤ 0

→ else $(\exists\ i \quad C_i > 0)$

$\forall\ j \neq i \quad a_j = 0$

$x_i = \text{Min}\ (b_i / i^{th}\ \text{col of}\ A)$

Now change the co-ordinate axis such that your new point is the origin & now check whether it is optimum or not & repeat till optimum.

→ SIMPLEX ALGO :-

For the prev eg. slack variables z_1, z_2 can be added And the conditions become

③ $x_1 + x_2 = 6$

① $z_1 + x_1 + x_2 = 6$

② $z_2 + x_1 - x_2 = 3$

$x_1, x_2 \geq 0$

$z_1, z_2 \geq 0$.

Simplex tableau :-

		x_1	x_2	z_1	z_2	b
①	z_1	1	2	1	0	6
②	z_2	1	-1	0	1	3
③		1	1	0	0	0

Simplex doesn't handle unbounded cases classmate

Date
Page

→ Pseudo code:

Step 1: If all no.s in the last row are ≤ 0, stop & return (-1) [*]

Else choose some j s.t. the j^{th} value in last row > 0

Step 2: Find i^* such that $\text{Min}_i \frac{bi}{M[i,j]}$ [only positive cases]

Simplex tableau:-

	x_1	x_2	z_1	z_2	
z_1	0	3	1	-1	3 ←
x_1	1	-1	0	1	3
	0	2	0	-1	-3

× optimum

Step 3: Pivot @ $i^* j$

Scale i^{th} row by $\frac{1}{M[i^* j^{th} col]}$

Subtract appropriately.

∵ Still not optimum, repeat

	x_1	x_2	z_1	z_2	b
x_2	0	3	1	-1	1
x_1	1	0	1/3	2/3	4
	0	0	-2/3	-1/3	-5

OVERALL REVIEW:

4 Dimensions :-

1) Start :

i) Church Turing Hypothesis

ii) Diagonalisation

iii) Computability problems $A_{TM} \in RE$ $\overline{A_{TM}} \notin RE$ A_{TM} &

iv) Reductions $L_1 \leq L_2$

L_1 not solvable $\Rightarrow$ L_2 not solvable

L_2 solvable $\Rightarrow$ L_1 solvable

L_2 is at least as hard as L_1

vi) Karp Reductions

vii) Trans disciplinary reductions.

viii) Polynomial time verifier $\Leftrightarrow$ lang $\in$ NP

ix) P vs NP

2) End

10. 2-Approx MVC

11. Pseudo solutions.

12. Proactivisation

3) Flow:

13. Strassens

Fast Fourier

Karatsuba

14. Greedy Algorithms

Activity selection

Matroid theory

Hoffman Code

Greedy set cover

15. Matroid Theory
    Min/Max spanning tree
    Max ind set in any weighed matroid
    Greedy choice property
    Optimum substructure

16. Dynamic Programming
    Edit distance
    longest path in a DAG
    Chain matrix multiplication

17. Linear Programming
    → Simplex algorithm

18. Network Flows
    Ford Fulkerson

19. Median of Median

20. Randomization