# Algorithms Analysis and Design

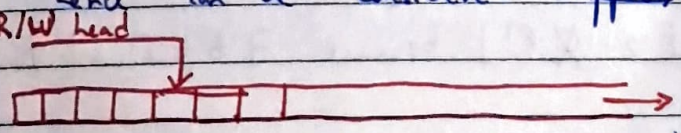
Algorithm :- No singular answer, operational definitions exist, like "Set of steps to reach a goal"

10th Hilbertian Question :- Give an algo to solve Random Diophantine Eqn. [1900]

The definition of an algorithm has answered multiple questions like "What is a machine?", "What is a computer?", etc.

Axioms

1) Information travels at finite speeds. [Spl. Theory of Relativity]
   → Random Access Memory is a myth. All memory is sequential, hence can be considered ~~tapes~~ as tapes.




Speed of the R/W head is 1 cell/per unit time.

2) Finite volume in space can be used to store or retrieve only finite amounts of information reliably [Quantum Mechanics]
   → This implies that we can not store ∞ precision of irrational numbers. Thus using (perfectly precise) pi or the like in an algorithm is not allowed.
   Thus each cell can only store a finite amount of data [Based on A1 & A2]

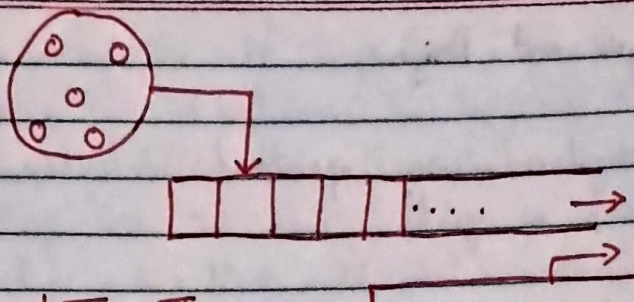
   Data in any cell ∈ Γ : Finite set of Tape alphabet

3) Only finitely many instructions can be packed a priori. [ISA is finite]
   → Computers are finite ; hence.

Algorithms book by Das Gupta

apsara

Date: ___




Church Turing Thesis

Algorithm:- Any algorithm that can be simulated using a Turing Machine

Turing Machine:- A Turing Machine is a 7-tuple

$\langle Q, \Gamma, \delta, q_{start}, q_{accept}, q_{reject}, \Sigma \rangle$

$Q \rightarrow$ Finite ~~state~~ set of states

$\Gamma \rightarrow$ Finite set of state alphabets (that can fill cells)

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

$\Sigma \subset \Gamma \quad [\Sigma \not\subset \Gamma]$ because $\exists\, b \in \Gamma$ s.t. $b \notin \Sigma$

A Turing Machine can have another Turing Machine as its input

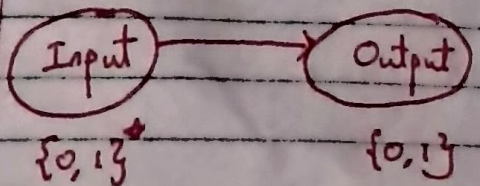
$U_{TM}(\langle M_{TM} \rangle, x) = M_{TM}(x)$

Where $U_{TM}$ & $M_{TM}$ are Turing Machines, $x$ is the input to M and $\langle\rangle$ is the string typecast.

$U_{TM}$ is the Universal Turing Machine

What is a computational problem?

Decision Problems




Test

Quick brown Fox Jumps over the lazy Dog

Problems having multiple bits can be considered as multiple problems having single bit outputs

All ~~problems~~ computational problems are thus ~~considered~~ defined as problems that ~~are~~ have a set of inputs which can be ~~bi-partitioned~~ bi-partitioned into a T/F inputs. Since the two partitions are ME & E, just being given the input set and one of its subsets is sufficient. All computational problems are hence decision problems.

Decision Problems = (Membership Queries in) Language.

A Language $L \subseteq \{0,1\}^*$

Given a graph G, is it connected or not? Decision problem

(or)

Does G belong to the set of all connected graphs Membership Query

$G \in \{G' \mid G' \text{ is a connected graph}\}$

Is a given number n prime?

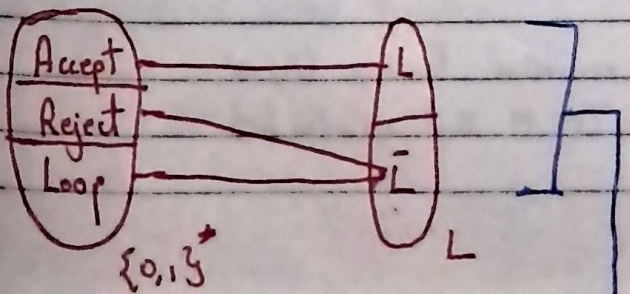
↓

$n \in \{i \mid i : \text{is prime}\}$

Problems are languages, solutions are Turing Machines

Turing Machines have three states, Accept, Reject and Loop

Languages have only two states, L & $\bar{L}$



apsara

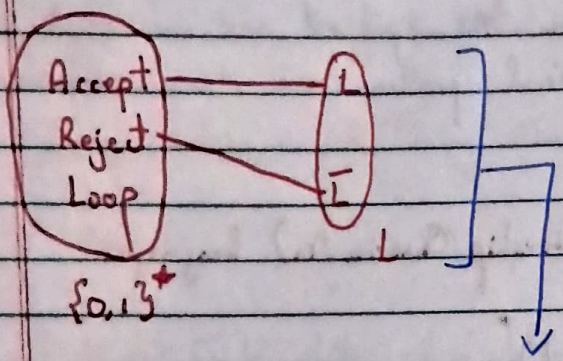
Date

**Definition:-** A language L is said to be recognized by a Turing Machine M if:-

$\forall x \in L$ ; M(x) accepts.

else ; M(x) rejects.

Hence the language L of M $[L(M)] = \{x \mid M(x) \text{ accepts}\}$




**Definition:-** A language L is said to be ~~recognized~~ decided by TM M is

$\forall x \in L$ ; M(x) accepts

else ; M(x) rejects

A TM that decides L also recognizes L, but vice versa does not hold.

**Theorem:-** There are languages that are unrecognizable

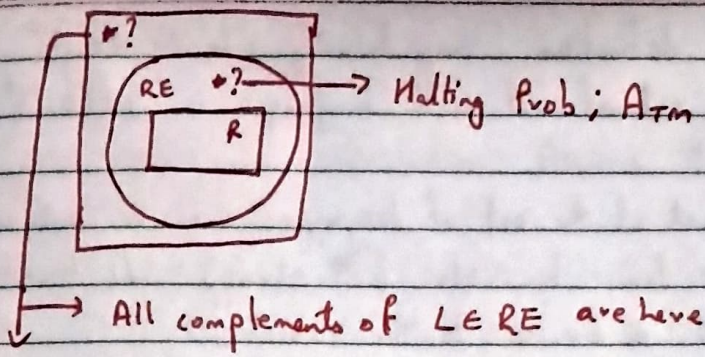
**Theorem:-** There are languages that are recognizable but undecidable

**Defn:-** The class of recursively ~~enumerated~~ enumerable (RE) languages is

$RE = \{L \mid \exists$ a TM M that recognizes $L\}$

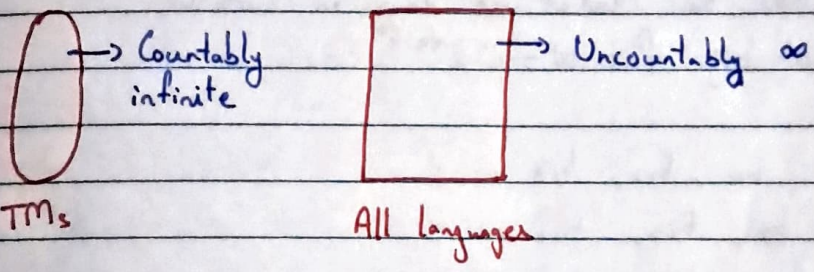
**Defn:-** The class of recursive languages (R) is

$R = \{L \mid \exists$ a TM M that decides $L\}$




How do you show that ~~there~~ $\exists$ languages here?

Take the set of all TMs and the set of all problems




- Any TM can be stored in a machine with finite bit strings, hence the no. of TMs $\leq$ no. of finite len bin strings
Thus TMs $\subseteq \{0,1\}^*$ → They can be enumerated
How?

**Theorem:-** $\{0,1\}^*$ is countable

**Proof:-** $f: N \to \{0,1\}^*$ must be a bijection
Consider the following enumeration of $\{0,1\}^*$
Shorter first, lexicographic ordering for eg less.

$\epsilon$, 0, 1, 00, 01, 10, 11, 000, ..., 0000, ...
(000, ... : 8 ; 0000, ... : 16)

This is a bijective map cause every string has a fixed index.
[Find a closed order formula]

**Theorem:-** The set of languages is uncountable.

**Proof:-** By definition, any language $L \subseteq \{0,1\}^*$.
So the set of all such languages is $P(\{0,1\}^*)$

Each subset of the set of languages can be uniquely identified by an $\infty$ len bin str $[i^{th}$ elem $\in$ or $\notin$ to the subset$]$ str$[i] = 0/1$ based on whether the

Any $L \equiv$ inf len bin str.

Assume that the set of all langs is countable.
Then $\exists$ a bij $f : N \to 2^{\{0,1\}^*}$

$f(1) = b_{11}, b_{12}, b_{13} \ldots$
$f(2) = b_{21}, b_{22}, b_{23} \ldots$
$f(3) = b_{31}, b_{32}, b_{33} \ldots$

Since its countable, all $N$ have been mapped to these $f(i)$
Now we diagonalization to create a new language
$L = \overline{b_{11}}, \overline{b_{22}}, \overline{b_{33}}, \overline{b_{44}}, \ldots$
This $L$ has no mapping and is distinct from all $f(i)$ : at least 1 pos. Hence its uncountably $\infty$.

The class RE are those programs which give yes, no or loop, but never returns a wrong answer.

Given a TM M and input x, will M accept x?

$A_{TM} = \{\langle M, x\rangle \mid M \text{ accepts } x\}$

This problem is recognizable, not decidable $\longrightarrow \in RE$

<M> → Program typecast to a str to be used as input!

Date

**Theorem:-** $A_{TM}$ is undecidable.

**Proof:-** We prove by contradiction. Assume that $A_{TM}$ was decidable
→ ∃ a TM H that decides $A_{TM}$.
→ ∀ $\langle M, x \rangle \in L$, $H(\langle M, x \rangle)$ accepts;
∀ $\langle M, x \rangle \notin L$, $H(\langle M, x \rangle)$ rejects; [not loop]

Consider a TM D which does the following
i) On input M
ⓐ Run ~~H (<M>, m)~~ H (M, <m>)
ⓑ Return ¬ H's result (accept on reject and vice versa)

Execute D (<D>)
On input <D>, Run H (D, <D>)
If H accepts, then rejects → (D, <D>) ∈ $A_{TM}$
This implies D accepts <D> then D rejects ::
Contradiction

If H rejects, D accepts → (D, <D>) ∉ $A_{TM}$
This implies D rejects <D> then D accepts.
Contradiction

(or)

| | $\langle M_1 \rangle, \langle M_2 \rangle \cdots \langle M_i \rangle$ |
|---|---|
| $M_1$ | $H(M_1, \langle M_2 \rangle)$ |
| $M_2$ | |
| ⋮ | |
| $M_i$ | |

Use diagonalization create a new D, but $H(D, \langle D \rangle)$ is a contradiction.
That's how diagonalization works!

**Theorem:-** If $L \in RE$ and $\bar{L} \in RE$, then $L \in R$

**Proof:-** ~~$L \in RE \Rightarrow \forall x \in L, \exists$ TM $M$ s.t $M$ accepts $x$~~

$\forall$

i) $L \in RE \Rightarrow \exists$ a TM $M$ s.t
$\forall x \in L$; $M$ accepts $x$
$\forall x \in \bar{L}$; $M$ rejects / loops on $x$

ii) $\bar{L} \in RE \Rightarrow \exists$ a TM $M'$ s.t
$\forall x \in \bar{L}$; $\bar{M}$ accepts $x$
$\forall x \in L$; $\bar{M}$ rejects / loops on $x$

Create a decider ~~the~~ by running $M$ and $M'$ in parallel.
$M$ accepts $x \in L$ and $M'$ accepts $x \in \bar{L}$.

**Theorem:-** ~~$A_{TM} \notin RE$~~ $\bar{A}_{TM} \notin RE$

**Proof:-** $A_{TM} \in RE$ ; $A_{TM} \notin R$
By contradiction and prev theorem, $\bar{A}_{TM} \notin RE$.

$TIME(n^k)$ → Problems solved in $O(n^k)$, where $k \in$ Rational
$TIME(n^{k+\epsilon})$ → Superset of $TIME(n^k)$

**Reduction:-** A language $L_1$ is mapping reduced to $L_2$ if $\exists$ a computable fn $f: \{0,1\}^* \to \{0,1\}^*$ s.t. $\forall x \in \{0,1\}^*$, $x \in L_1 \Leftrightarrow f(x) \in L_2$

↳ $L_1 \leq_m L_2$

~~**Defn:-** Language $A \leq_m B$~~

$f(\langle M, x \rangle) = \{ H(M,x) \wedge [M(x)] \}$

apsara

Date: __/__/____

Examples:-

Let $HALT_{TM} = \{ \langle M, x \rangle \mid M \text{ halts on input } x \}$

Lemma:- $A_{TM} \leq_m HALT_{TM}$

Proof:- Decider for $A_{TM}$ (using the one for $HALT_{TM}$)

Let H decide $HALT_{TM}$

On input $\langle M, x \rangle$ :-

→ Run $H(M,x)$

→ If H rejects, REJECT

→ If H accepts, ACCEPT or REJECT based off $M(x)$

Theorem:- If $A \leq_m B$ and $B \in R \rightarrow A \in R$

Theorem:- If $A \leq_m B$ and $A \notin R \rightarrow B \notin R$

Completeness Theory:-

Suppose $\forall A \in RE$, $A \leq_m HALT_{TM}$.

Then $HALT_{TM}$ is RE-complete

Theorem:- $HALT_{TM}$ is RE-complete

→ Anything in RE

Proof:- Let TM M recognize A & H decides $HALT_{TM}$

Construct a decider for A.

On input x :

Run $H(M,x)$. If

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

**Theorem:-** $EQUAL_{TM}$ is unrecognizable

**Proof:-** $\overline{A_{TM}}$ is unrecognizable ($\overline{A_{TM}} \notin RE$)

Let E recognize $EQUAL_{TM}$

ω

$M_1$ : On input [redacted], Reject

$M_2$ : On input ω, if $x \notin \omega$ Reject
If M accepts x, ACCEPT.
Else Reject

If $E(M_1, M_2)$ accepts $\Rightarrow$ M does not accept x.

$(M, x) \in A_{TM}$

Divide and Conquer.

Mergesort, Binary Search, Median of Medians.

Fourier Transforms:-
Input:- Two polynomials A(x) & B(x) of degree n-1
Output:- Their product $C(x) = A(x) \cdot B(x)$

A[n] → coefficients of A
B[n] → coefficients of B
C[2n-1] → coefficients of C

$$A(x) = \sum_{i=0}^{n-1} a_i x^i$$

$$B(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$C(x) = \sum_{i=0}^{2n-1} c_i x^i$$

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} \longmapsto$$ But this is an $O(n^2)$ algorithm

But Fast Fourier Transforms can do it in $O(n \log n)$

Algorithm :-

$O(n)$ ← i) ~~Evaluate~~ Select $2n-1$ or more points $x_1, \ldots x_m$ $[m \geq 2n-1]$

ii) Evaluate $A(x_i)$ and $B(x_i)$ $\longmapsto O(n^2)$ [n-pts with n-1 substitutions per pt]

iii) Multiply $A(x_i) \times B(x_i) \Rightarrow C(x_i)$ $\longmapsto O(n)$

iv) Interpolate $C(x_i)$'s to obtain $C(x)$ $\longmapsto O(n^2)$

Let $A(x)$ be the polynomial

$$A(x) = A_e(x^2) + x A_o(x^2)$$

$$A(x) = 5x^4 - 3x^3 + 7x^2 + x + 1$$
$$A_e(x) = 5x^2 + 7x + 1$$
$$A_o(x) = -3x + 2$$

For any $A(x_i) = A_e(x_i^2) + x_i A_o(x_i^2)$

$A(-x_i) = A_e(x_i^2) - x_i A_o(x_i^2)$

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

But ~~$A_e(x_i^2) = A_{ee}(x_i^4) + x_i^2 A_{eo}(x_i^4)$~~

~~$A_e(-x_i^2)$~~

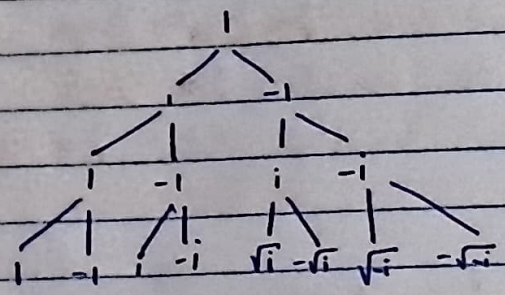
→ +/− space

A: $\{x_1 \quad -x_1 \quad x_2 \quad -x_2 \quad \cdots \quad x_d \quad -x_d$

$\{ x_1^2 \quad x_2^2 \quad x_d^2$

→ Not +/− space ⇒ Can't recurse further.

Hence, we pick complex nos to evaluate.
Let begin from the bottom.




4th roots of unity

8th roots of unity

$m^{th}$ roots of unity

$e^{i \frac{2\pi}{m} j}$ m is an exact power of $2 \geq 2n-1$

$j \rightarrow [1, m]$

Goal: To evaluate a polynomial A on the $m^{th}$ roots of unity
$1, \omega, \omega^2, \cdots, \omega^{m-1}$

Input: A[]
Output: $[A(1), A(\omega), \cdots, A(\omega^{m-1})]$

$$A = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} \qquad A(x) = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ & & \vdots & & \\ 1 & x_{d+1} & x_{d+1}^2 & \cdots & x_{d+1}^d \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix}$$

To evaluate our output, we multiply A with

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^d \\ \vdots & & & & \\ 1 & \omega^{m-1} & \omega^{(m-1)\cdot 2} & \cdots & \omega^{(m-1)^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{bmatrix} = \begin{bmatrix} A(1) \\ A(\omega) \\ \vdots \\ A(\omega^{m-1}) \end{bmatrix}$$

↳ $i,j^{th}$ entry $\rightarrow \omega^{ij}$ $\forall\, i,j \in [0, m-1]$

↳ Discrete Fourier Transform.

```
FFT FFT(A, ω):
    if ω == 1:
        return A(1)
    else:
        A_e = [A[0], A[2], ....]
        A_o = [A[1], A[3], ....]
        FFT(A_e
        E = FFT(A_e, ω^2)
        O = FFT(A_o, ω^2)
        for i in range(m-1):
            A[i] = E[i/2] + ω^2 O[i/2]
            # E and O are half the size of A, hence we
            # must recd at half the index
    for i in range((m•)/2):
        A[i] = E[i] + ω^i • O[i]
        A[m/2 + i] = E[i] - ω^i • O[i]
        # • Check penultimate page for the formula
```

Now, the second step in our algorithm (check antipenultimate page) runs in $O(n \log n)$

In step 1, we pick $m \geq 2n-1$ where $m$ is a power of 2, which are just the $m^{th}$ roots of unity.

I'm lost uwu [We've fucked]

# Interpolation

$$\begin{bmatrix} M \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} c(1) \\ c(\omega) \\ \vdots \\ c(\omega^{m-1}) \end{bmatrix}$$

$$\downarrow$$

$$M_{ij} = \omega^{ij}$$

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} = M^{-1} \begin{bmatrix} c(1) \\ c(\omega) \\ \vdots \\ c(\omega^{m-1}) \end{bmatrix}$$

$$M^{-1} = \frac{1}{m} M(\omega^{-1}) \quad [Y_{um}\ Y_{um}]$$

or

$$[Y_{um}]^{*}$$

$$[\omega^{ij}]_{m \times m} \times [\omega^{-ij}]_{m \times m}$$

$$= \begin{bmatrix} m & & & \\ & m & & \\ & & m & \\ & & & \ddots \end{bmatrix}$$

This (somehow! some fucking how!) makes the last step $O(\log n)$

## Karatsuba's Algorithm:-

Large integer multiplication [Two n-bit +ve integers]

Normal multiplication takes $O(n^2)$

Fastest way to multiply large integers is still FFT by converting the no. into a polynomial $f(x)$ where $f(2) = int$.

## Sorting Detour:-

Assuming all numbers are bounded.

Countsort :- Store them in an array at loc $x_i$

↓

count($x_i$)

Radix sort:- Apply countsort on every position, units, tens, etc. until the largest place.

## End detour:-

## Multiplying 2 complex numbers $(a+ib)(x+iy)$

$\Rightarrow ax - by + i(bx + ay)$

Looks like it needs 4 real products, but one can just use 3 :-

(i) $p_1 = (x_1 + y_1)(x_2 + y_2)$

(ii) $p_2 = x_1 \cdot x_2$

(iii) $p_3 = y_1 \cdot y_2$

Similarly, for n bit numbers a & b

$a = a_L \cdot 2^{n/2} + a_R$

$b = b_L \cdot 2^{n/2} + b_R$

$ab = a_L b_L 2^n + 2^{n/2}(a_L b_R + a_R b_L) + a_R b_R$

We can use this to establish a recurrence, but it would be $O(n^2)$. Instead we apply Gauss' Logic, reducing it to 3 multiplications:-

$$P_1 = (a_L + a_R)(b_L + b_R)$$
$$P_2 = a_L b_L$$
$$P_3 = a_R b_R$$

and $ab = P_2 2^n + 2^{n/2}[P_1 - P_2 - P_3] + P_3$

Recurrence on this results in

$$T(n) = 3T(n/2) + O(n)$$
$$= O(n^{\log \frac{3}{2}})$$

$$T(n) = 3T(n/2) + O(n)$$
$$= 3[3T(n/4) + O(n/2)] + O(n)$$
$$= 3^2(T(n/2^2)) + 3n/2 + n$$
$$= \ldots$$
$$= 3^3 T(n/2^3) + 3^2 n/2^2 + 3^1 n/2^1 + n$$
$$= 3^i T(n/2^i) + n \sum_{j=0}^{i-1} 3^j/2^j$$

Let $i = \log_2 n$

$$T(n) = 3^{\log_2 n} + n \sum_{j=0}^{\log n - 1} 1.5^j$$

$$\cong O(n^{\log \frac{3}{2}})$$

Strassen's Algorithm:-

$$[A]_{n \times n} \quad [B]_{n \times n} \rightarrow \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]_{n \times n} \quad \left[\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right]_{n \times n}$$

$$= \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}_{n \times n}$$

$T(n) = 8\,T(n/2) + \underbrace{O(n^2)}_{\text{Addition}}$

$= 8\left[8\,T(n/4) + O\left((n/2)^2\right)\right] + O(n^2)$

$= \ldots\ldots$

$\cong O(n^3)$

BAD! :<

Strassen's ~~technique~~ technique :- He does it in 7 mults

$T(n) = 7(T(n/2)) + O(n^2)$

$= \ldots$

$= O(n^{\log 7})$

The product would then be

$$\begin{bmatrix} P_5 + P_6 + P_4 - P_2 & P_1 + P_2 \\ P_3 + P_4 & P_5 - P_7 + P_1 - P_3 \end{bmatrix}$$

$P_1 = A(F-H)$

$P_2 = (A+B)H$

$P_3 = (C+D)E$

$P_4 = D(G-E)$

$P_5 = (A+D)(E+H)$

$P_6 = (B-D)(G+H)$

$P_7 = (A-C)(E+F)$

Fibonacci?

$F_n = F_{n-1} + F_{n-2}$

$F_0 = 0$

$F_1 = 1$

Recursion → $2^n$ → bad :<

Iteration → $n$ → okay

↳ but for large $n$, add becomes $O(n)$ — we have $O(n)$ additions.

↳ $n^2$ for large $n$

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^2 \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$$\vdots$$

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

↳ $\log n$ multiplications

## Greedy Algorithms

Used in optimization problems

Ex :- Shortest path - single source, Huffman encoding.

Optimum Substructure Property:- ~~Sup~~ Subparts of a problem must also be using the most efficient solution.

If $S \to V$ is the shortest path, then $S \to T \Rightarrow S \to V + V \to T$,

In such cases we usually recurse, given the first step to be taken.

Greedy Choice Property :- Picking the right first step.

We need both Optimum Substructure and Greedy Choice to have an efficient Greedy Algo

If only the Optimum Substructure holds, we try out all possibilities, but this is slow, unless we have :-

Overlapping Subproblems Property: If we are able to create a DAG with the subproblems. We toposort first and solve subproblems in toporder

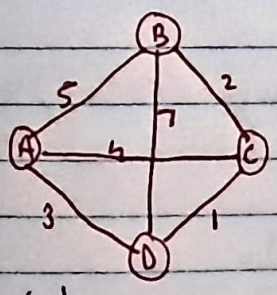
Greedy is a spl case of DP where the DAG is a path.

Minimum Spanning Trees:-

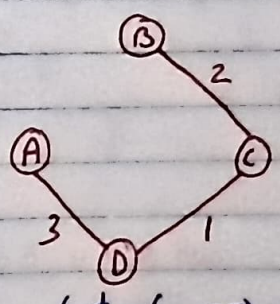
Input:- Edge Weighted Simple Undirected Path.

Subgraph of a tree on all its vertices

Cost of tree = $\Sigma$ edge wts.




Cost :- 21




Cost = 6 → MST

Output:- Least cost Spanning Tree T of G

The Cut Property:-

Let $G = (V, E)$

Suppose $X \subset E$ belongs to some MST of G.
Then let V be divided into S and V/S such that no edge spans from $S \to V/S$ or vice versa.

Then the least cost edge e b/w S & V/S is s.t $X \cup \{e\}$ is part of some MST of G.

Algo:- [Kruskal]

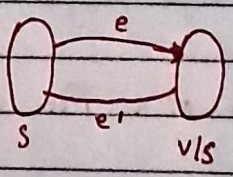
Sort edges by wt.
Add edges in that order ~~until~~ unless cycle is formed
the MST

Proof:- Suppose $X \cup \{e\}$ is part of T if X is a part of T.

$e \notin E(T)$

$e' \in T \to e' \in cut$

$wt(e) \leq wt(e')$




$(T \cup \{e\}) \setminus \{e'\}$ has to be connected cause $T \cup \{e\}$ formed a cycle.
↳ had n edges, so the graph $(T \cup \{e\}) \setminus \{e'\}$ has n-1 edges and is connected → is a tree T'

$cost(T') = cost(T) + wt(e) - wt(e')$.
$\leq cost(T)$

Since T is an MST, $cost(T') = cost(T)$
Hence T' is an MST

Prim's Algorithm:-

Let $X = \phi$

Repeat:

(i) Find a cut that doesn't contain X

(ii) Update $X = X \cup \{e\}$ where e is the min-cost edge in that cut.

Until $|V|-1$ edges are added.

Kruskals is preferred because of advanced data structures like union-find :- Find → $O(\log n)$, Union → $O(1)$ and tot complexity of $O(|E| \log |V|)$ without path compression.

With path compression ⇒ Find → $O(\log^* n)$

Activity Selection / Interval Scheduling Problems:

Input = $\{[s_1, f_1), [s_2, f_2), \cdots, [s_n, f_n)\}$

Non over lapping interval:

Output:- A Maximum sized subset of compatible activities

Hint:- Activity $A_i$ is part of some optimum soln.
Remove an activities compatible with $A_i$

Greedy Choice:- The activity with the least finish time is always a part of some optimum soln.

Algo:- Sort all to finish time. Initialize the first activity. Pick next activity that is compatible and continue

Knapsack Problem:-

Input:- Collection of items $\{(c_1, w_1), (c_2, w_2), \cdots, (c_n, w_n)\}$

Output:- Subset of items s.t total wt $\leq W$, maximize cost

Matroid Problem:-

Definition:- A matroid is a pair $\langle S, I \rangle$ where:

(a) S is a finite non-empty set

(b) I is a family of collection of subsets of S s.t. $[\phi \in I]$ (set of independent sets ↓)

(i) Hereditary Property:- If $A \subseteq S$ belongs to I, then $\forall B \subseteq A$, $B \in I$

(ii) Exchange Property:- If $A, B \in I$ and $|B| > |A|$, then $\exists x \in B \setminus A$ s.t $A \cup \{x\} \in I$

Question:- Given a Matroid $\langle S, I \rangle$ and given non-ve int wts to elements of S, find max wt element of I (or subset of S)

$w(A) = \sum_{x \in A} w(x)$ . $x \in S$ & $w(x)$ = wt of $x$.

Ans:- Universal Greedy Algo for weighted matroids.

→ Sort the elements of S in non-increasing order of wts
→ $A \leftarrow \phi$
→ In that order, let $x$ be the next element
if $A \cup \{x\} \in I$ then $A \leftarrow A \cup \{x\}$
Finally output A.

Proof of correctness / optimality

(i) Greedy Choice :- The element $x \in S$ with max wt belongs to some optimum solution.

Proof :- Let $A \in I$ be an optimum soln.

if $x \in A$ :

pass

else: $[x \notin A \;\&\; \{x\} \in I]$

Let $A = \{a_1, a_2, \ldots, a_k\}$

From exchange property, we know that $\exists$ a $B = (A \cup \{x\}) \setminus \{a_i\}$ & $B \in I$

$\omega(B) = \omega(A) + \omega(x) - \omega(a_i) \geq \omega(A)$

$a_i$ has ~~better~~ lesser wt than $x$ cause $x$ appeared before $a_i$ in sorted order

$M = \langle S, I \rangle$ contracts to $M' = \langle S', I' \rangle$, $A = A' \cup \{x\}$

$S' = S - \{x\}$ ; $I'$ : sets of $I$ with $x$ erased

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

$\omega(A) = \omega(A') + \omega(x)$

Given an edge weighted graph $G = (V, E)$

Let $S_G = E$

$I_G = \{ E' \subseteq E \mid E' \text{ does not have a cycle} \}$

Theorem :- $M_G = \langle S_G, I_G \rangle$ is a matroid.

$I_G$ is hereditary, cause if $A \in I_G$, all $s \subseteq A$ also $\in I_G$.

$I_4$ has the exchange property.

$|E_1| > |E_2|$ → forest with $|V| - |E_2|$ trees

↓

forest with exactly $|V| - |E_1|$ trees

$E_1$ has an edge $(u, v)$ s.t $u, v$ are in diff trees in $E_2$.
Add this edge to $E_2$, which will not form a cyclic graph.

$E_2 \in I \rightarrow E_2 \cup \{(u, v)\} \in I_4$

Hence $M_4$ is a matroid.