21/8

Algorithm : Methods to solve a problem
└ Pseudo-code

T·H·Cormen : Algorithms

Assignment : 20
Mid-Sem : 30
End-Sem : 50

Algorithm : A sequence of logically related instructions to solve a computational problem

Computational problem : Input & Output
Ex. Array A, x ⟵ I/P
Does x ∈ A
Yes / No ⟵ O/P

Any Algo must satisfy :
(1) I/P, O/P
(2) Finiteness : Must terminate after finite amount of time
(3) Definiteness : Each instruction must be unambiguous
(4) Must Involve basic arithmetic instructions

Ex. Finding max. element in an array

① Loop, A[i] > max :
max = A[i];

② sort A
return A[n];

③ $a_1, a_2, \ldots a_{n/2}$ | $a_{n/2+1}, \ldots a_n$
max_1 | max_2
max

Many Algorithms

— Good Algo :
① Correctness : Approach desirable or not
② Time complexity / space complexity
③ Optimality

A set of computational problems

Solvable
∃ an algo
↳ How many diff algos
How to compare algos
How to pick best algo (efficiency)

Unsolvable
No algo

Correctness
Optimality (Time & Space)

30/8

$f(n)$, $g(n)$ — Step count (Non-negative)

$f(n) = O(g(n))$ iff $\forall n \geq n_0 \; \exists c > 0 \quad (f(n) \leq c \cdot g(n))$

$f(n) = O(g(n))$ iff $\exists n_0 > 0 \; \exists c > 0 \; \forall n \geq n_0 \; (f(n) \leq c \cdot g(n))$

$f(n) = 3n+6$ $\quad 3n+6 \leq 5n$, $c=5$, $\forall n \geq 3$, $n_0 = 3 \Rightarrow 3n+6 = O(n)$

$3n+6 \leq 15n$, $c=15$, $n_0 = 1$, $\forall n \geq 1 \implies 3n+6 = O(n)$

~ Big Omega (Ω).

$f(n) = \Omega(g(n))$ iff $\exists c > 0 \; \exists n_0 > 0 \; \forall n \geq n_0$
$(f(n) \geq c g(n))$

~ Tight Bound (Θ)

$f(n) = \Theta(g(n))$ iff $\exists c_1 > 0 \; \exists c_2 > 0 \; \exists n_0 > 0 \; \forall n \geq n_0$
$(c_1 g(n) \leq f(n) \leq c_2 g(n))$

$f(n) = \Theta(g(n))$ iff $f(n) = \Omega(g(n))$
&
$f(n) = O(g(n))$

~ Little oH:

$f(n) = o(g(n))$ iff $\forall c > 0 \; \exists n_0 > 0 \; \forall n \geq n_0$
$(f(n) \leq c \cdot g(n))$

~ Little Omega:

$f(n) = \omega(g(n))$ iff $\forall c > 0 \; \exists n_0 > 0 \; \forall n \geq n_0$
$(f(n) \geq c \cdot g(n))$

$n^3 + 4n^2 - n = \Omega(n^3)$
$= \Omega(n^{3-\varepsilon})$, $\varepsilon \geq 0$
$= \omega(n^{3-\varepsilon})$, $\varepsilon > 0$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c \Rightarrow f(n) = O(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c \Rightarrow f(n) = \Omega(g(n))$$

$\forall c$ ↓ [c : constant]

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \Rightarrow f(n) = \Theta(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) = \omega(g(n))$$

→ Reflexivity :

$f(n) = O(f(n))$ | $n = \Theta(n)$ | $n\log n = \Theta(n\log n)$

$f(n) = \Omega(f(n))$ | $n = O(n)$ | $O(n\log n)$

$f(n) = \Theta(f(n))$ | $n = \Omega(n)$ | $\Omega(n\log n)$

→ Symmetricity :

$f(n) = O(g(n))$ | $n = O(n^2)$

$\not\Rightarrow g(n) = O(f(n))$ | $\not\Rightarrow n^2 = O(n)$

$f(n) = \Omega(g(n))$ | $n = \Omega(1)$

$\not\Rightarrow g(n) = \Omega(f(n))$ | $\not\Rightarrow 1 = \Omega(n)$

Θ — symmetric

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

— Transitivity :

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

$\Rightarrow f(n) = O(h(n))$

$f(n) = n^2,\ g(n) = n^4,\ h(n) = 2^n$

$f(n) = \Omega(g(n)),\ g(n) = \Omega(h(n))$

$\Rightarrow f(n) = \Omega(h(n))$

$f(n) = n^2$

$g(n) = n\log n,\ h(n) = 1/n$

$f(n) = \theta(g(n))$, $g(n) = \theta(h(n))$

$\Rightarrow f(n) = \theta(h(n))$

→ Transitive Symmetry :

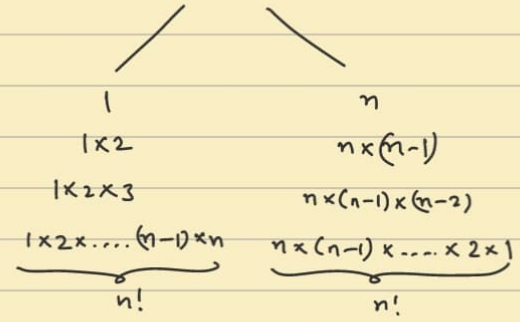
$f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$ $\quad n^3 = O(2^n) \Rightarrow 2^n = \Omega(n^3)$

$f(n) = \Omega(g(n)) \Rightarrow g(n) = O(f(n))$ $\quad n^3 = \Omega(n^2) \Rightarrow n^2 = O(n^3)$

→ Algorithm Design :

Iterative Algo (Top-down design) | Recursive Algo (Bottom-Up design)

Ex. Factorial (n)




```
fact = 1
for i=2 to n:
  fact *= i
```

```
if (n==1)
    return 1
else
    return n * fact(n-1)
```

Note : ∀ Iterative Algo , ∃ equivalent recursive algorithm ?
Which is more efficient

Ex. Computing n^{th} fibonacii number

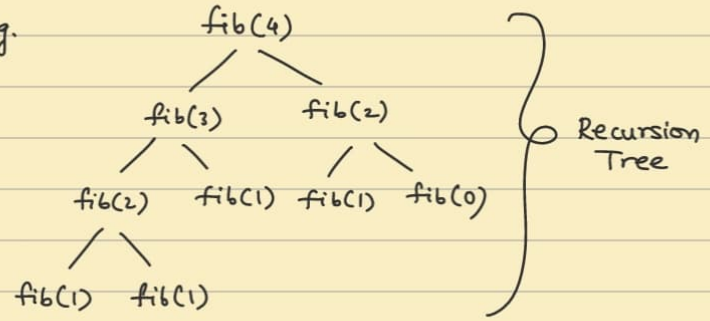
Iterative :

```
a=1
b=1
for i=2 to n:
    c = a+b
    a = b
    b = c
```

Recursive :

```
if (n==0) || (n==1)
    return 1
else
    return fib(n-1) + fib(n-2)
```

e.g.



Is Recursion doing extra work ?

Measure —— System Independent

Time Complexity involves step count

↳ Frequency of fundamental operations involved in the algo

→ Analysis :

Ex. Find max function :

		operations
(1)	max = a[1]	Assignment (1)
True (n-2+1) + False (1) = n	for i=2 to n	Assignment (n), comparison (n), increment (n)
(n-1)	if a[i] > max	Comparison (n-1)
(n-1)	max = a[i]	Assignment (n-1)
(1)	return max	

Worst case

↳ Step count analysis i.e. count of assignment operators

Ex. Fibonacci

1, 1, 2, 3, 5, 8

```
a = 1          ——→ (1)
b = 1          ——→ (1)
for i=2 to n   ——→ (n-2+1) + (1) = n  ⟹ 3n   (n) (n) (n)  (Assign + Compare + Inc)
c = a+b        ——→ (n-1)              ⟹ 2(n-1)  (Assign + Add)
print c        ——→ (n-1)
a = b          ——→ (n-1)
b = c          ——→ (n-1)
```

5n − 2 steps / instructions ∴ 8n − 3

Ex. Factorial

```
if (n==1)                    n=1   n≥2
  return 1                   (1)   (1)
                             (1)
else
  return n * fact(n-1)       ——    Multiply assign
                             (2)   (1) + (1)
                                   T_{n-1}
```

$T_n = 2$ if $n = 1$

$T_n = 3 + T(n-1)$ if $n \geq 2$

4/9

$T(n) = T(n-1) + n - 1$: Recursive Relations

Change of var:

$T(n) = T(\sqrt{n}) + 1$

$n = 2^m$

$T(2^m) = T(2^{m/2}) + 1$

$S(m) = T(2^m)$

$S(m) = S(m/2) + 1$

↳ Recursive Relation of Binary Search

$= \Theta(\log m)$

$\Rightarrow S(m) = \Theta(\log m)$

$T(n) = T(2^m) = \Theta(\log_2 m)$

$= \Theta(\log_2 \log_2 n)$

(9) $T(n) = T(\sqrt{n}) + n$ $T(2)$ — constant

$T(2^m) = T(\sqrt{2^m}) + 2^m$

$S(m) = S(m/2) + 2^m$

Sub:

$S(m/4) + 2^{m/2} + 2^m$

$= S(m/8) + 2^{m/4} + 2^{m/2} + 2^m$

$= S(m/2^k) + 2^{m/2^{k-1}} + \dots + 2^m$

$= S(1) + 2^{m/2^{k-1}} + \dots 2^m$ Assume $m = 2^k$

$S(1)$: const.

$$2^m \leq 2^m + 2^{m/2} + 2^{m/4} + \dots 2^2 + 2^2 \leq 2 \cdot 2^m$$

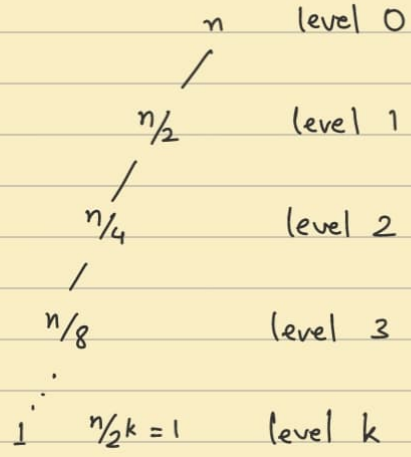
$$= \theta(2^m)$$

$$S(m) = \theta(2^m)$$

$$T(n) = T(2^m) + \theta(n)$$

→ Recurrence Tree method:

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

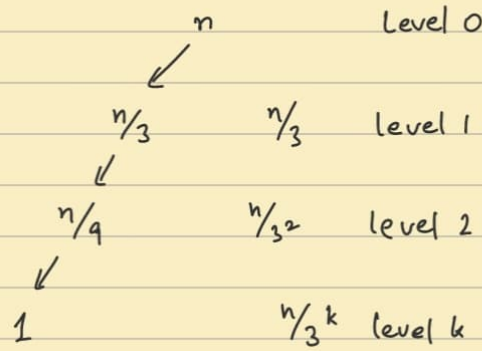


$$k = \log_2 n$$

$$(\log_2 n + 1) \times 1 = \log_2 n + 1$$

$$T(n) = 1 + \log_2 n$$

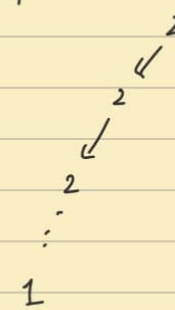
Ternary search:



Levels $= k+1$

$= \log_3 n + 1$

Computation time:



$\Rightarrow 2k+1$

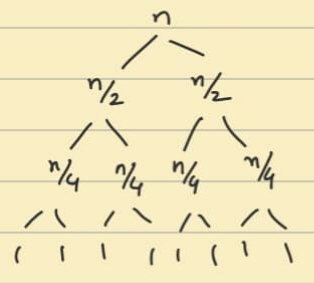
$\Rightarrow 2\log_3 n + 1$

$\theta(\log_3 n)$

$= \theta(\log_2 n)$

$= \theta(\log_k n)$ [fixed] $[k \in I]$

$T(n) = 2T(n/2) + 1$, $T(1) = 0$



level 0 → Input size Reduction tree

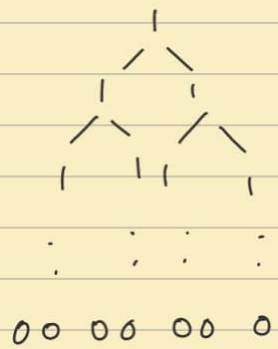
level 1

level k

$n/2^k = 1 \Rightarrow k = \log_2 n$

levels $= k+1 = \log_2 n + 1$

Computation tree:



$2^0 \times 1$

$2^1 \times 1$

$2^2 \times 1$

$2^{k-1} \times 1$

$2^k \times 0$

$2^0 + 2^1 + 2^2 + \ldots + 2^{k-1}$

$= 2^{\log_2 n - 1 + 1} = 2^{\log_2 n} - 1$

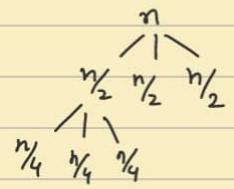
$= n - 1$

Divide and Conquer — 2 way Technique

6/9

$T(n) = 3T(n/2) + n^2$, $T(1) = 1$

I/p Reduction tree:



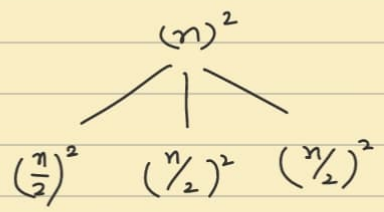
$3^k = 3^{\log_2 n}$ leafs

$n^{\log_2 3}$ leafs

$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

levels $= k+1 = \log_2 n + 1$

Computation tree :



$n^2 = (3/4)^0 n^2$

$\frac{3n^2}{4} = \left(\frac{3}{4}\right)^1 n^2$

$\left(\frac{3}{4}\right)^{\log_2 n - 1} \cdot n^2$

$\Rightarrow$ no. of levels $\times T(1)$

$\Rightarrow 3^{\log_2 n} \times T(1)$

$= n^{\log_2 3}$

$$n^2\left[\left(\frac{3}{4}\right)^0 + \left(\frac{3}{4}\right)^1 + \dots \left(\frac{3}{4}\right)^{\log_2 n - 1}\right] + n^{\log_2 3}$$

$$= n^2 \left[\frac{1 - \left(\frac{3}{4}\right)^{\log_2 n}}{1 - \frac{3}{4}}\right] + n^{\log_2 3}$$

Const.

$= n^2 \leq \qquad \leq 4n^2$

$= \Theta(n^2)$

$T(n) = a \cdot T(n/b) + f(n)$, $T(1) = 1$

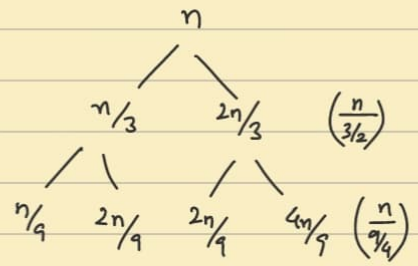
Cost at leaves $= n^{\log_b a} \cdot T(1)$

Q) $T(n) = 5T(n/2) + n^2$, $T(1) = 1$

$$\underbrace{\left(\frac{5}{4}\right)^0 n^2 + \left(\frac{5}{4}\right)^1 n^2 + \left(\frac{5}{4}\right)^2 n^2 + \ldots + \left(\frac{5}{4}\right)^{\log_2 n - 1} \cdot n^2}_{\text{cost from root to } \log_2 n - 1 \text{ levels}} + \underbrace{n^{\log_2 5} \cdot T(1)}_{= \Theta(n^{\log_2 5})}$$

cost @ $\log_2 n$ level

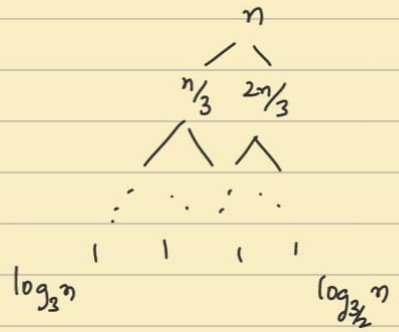
$T(n) = T(n/3) + T(2n/3) + n$, $T(1) = 1$



$\frac{n}{3^k} = 1 \Rightarrow k = \log_3 n$

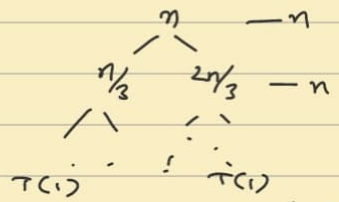
$\frac{n}{(3/2)^k} = 1 \Rightarrow k = \log_{\frac{3}{2}} n$

$\log_3 n < \log_{3/2} n$



leaves are distributed

Computation tree:



min height = $\log_3 n$

max height = $\log_{3/2} n$

$n \log_3 n \leq T(n) \leq n \cdot \log_{3/2} n$

$T(n) = \Omega(n \log_3 n)$

$T(n) = O(n \log_{3/2} n)$

$\log_3 n = \Theta(\log_{3/2} n)$

Subst. method ✗

Master's Th ✗

$T(n) = \Theta(n \log_3 n) = \Theta(n \log_{3/2} n)$

$= \Theta(n \log n)$

Master's Theorem :

$$T(n) = a \cdot T(n/b) + f(n), \quad T(1) = \text{constant}$$

a: Sub problem of n/b ; $f(n)$: Combining cost

$[b > 1]$

Case-I: If $f(n) = O(n^{\log_b a - \varepsilon})$, for some $\varepsilon > 0$

Then, $T(n) = \Theta(n^{\log_b a})$

Ex. $T(n) = 3T(n/2 + n) \Longrightarrow n = O(n^{\log_2 3 - \varepsilon}) \Rightarrow T(n) = \Theta(n^{\log_b a})$

$[\varepsilon > 0]$

Q2)

```
void hanoi (int n, char src, char dest, char aux){
    if (n == 1) {
        printf("Move disk 1 from %c to %c \n", srs, dest);
        return ;
    }
    hanoi (n-1, src, aux, dest) ;
    printf("Move disk %d from %c to %c \n", n, src, dest);
    hanoi(n-1, aux, dest, src) ;
```

Q1)

→

```
gcd (a, b) {
    if (a > b)
        a = a - b ;
    else if (a < b)
        b = b - a ;
    else
        return a ;
    return gcd (a, b) ;
}
gcd arr (arr, low, high) {
    if (low == high)
        return gcd (arr[low], arr[low])
    int left = gcd arr [arr, low, (low+high)/2]
    int right = gcd arr [arr, (low+high)/2, high]
    return gcd (left, right) ;
```

→

```
lcm(a,b){
    if (a == b)
        return a ;
    else if (a > b)
        if (a % b == 0)
            return a ;
    else {
        int temp = a ;
        int k = 2 ;
        while (temp % b != 0) {
            temp = a
            temp *= k ;
            k++ ;
        }
    return temp ; }
```

```
lcm arr (arr, low, high) {
    if (low == high)
        return lcm(arr[low], arr[low])
    int left = lcm arr [arr, low, (low+high)/2]
    int right = lcm arr [arr, (low+high)/2, high]
    return lcm (left, right) ;
```

Q3)

→

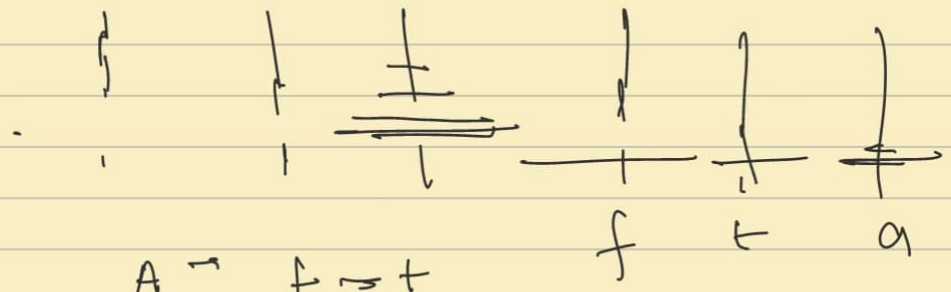
```
void merge(int arr[], int low, int m, int high) {
    int size1, size2;
    size1 = m - low + 1;
    size2 = high - m;
    int a[size1], b[size2];

    for (int i=0; i<size1; i++)
        a[i] = arr[low+i]
    for (int i=0; i<size2; i++)
        b[i] = arr[m+low+i]

    int i=0, j=0, k=low;
    while ((i<size1) && (j<size2)) {
        if (a[i] <= b[j]) {
            arr[k] = a[i];
            i++;
        }
        else {
            arr[k] = b[j];
            j++;
        }
        k++;
    }

    while (i<size1) {
        arr[k] = a[i];
        i++; k++;
    }
    while (j<size2) {
        arr[k] = b[j];
        j++; k++;
    }
}
```

```
void mergeSort(int arr[], int l, int h)
    if (l<h) {
        int m = (l+h)/2
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, h);
        merge(a, l, m, h);
    }
}
```

→ void merge(int arr[], int low, int m1, int m2, int high) {

```
int size1, size2, size3;
size1 = m1 - low + 1;
size2 = m2 - m1;
size3 = h - m2;
int a[size1], b[size2], c[size3]

for (int i=0; i<size1; i++)    a[i] = arr[low+i]
for (int i=0; i<size2; i++)    b[i] = arr[m1 + low + i]
for (int i=0; i<size3; i++)    c[i] = arr[m2 + low + i]

int i=0, j=0, k=0, m=low;
while ((i<size1) && (j<size2) && (k<size3)) {
    if ((a[i] <= b[j]) && (a[i] <= c[k]))
        arr[m++] = a[i++];
    else if ((b[j] <= a[i]) && (b[j] <= c[k]))
        arr[m++] = b[j++];
    else
        arr[m++] = c[k++]; }

while ((i<size1) && (j<size2)) {
  if (a[i] <= b[j])
     arr[m++] = a[i++];
  else
     arr[m++] = b[j++]; }
while ((j<size2) && (k<size3)) {
  if (b[j] <= c[k])
     arr[m++] = b[j++];
  else
     arr[m++] = c[k++]; }
while ((i<size1) && (k<size3)) {
  if (a[i] <= c[k])
     arr[m++] = a[i++];
  else
     arr[m++] = c[k++]; }

while (i<size1)    arr[k++] = a[i++];
while (j<size2)    arr[k++] = b[j++];
while (k<size3)    arr[k++] = c[k++];
}
```

```
void mergeSort(int arr[], int l, int h)
  if (l<h) {
     int m1 = (l+h)/3;
     int m2 = l + (2 * (h-l))/3
     mergeSort(arr, l, m1);
     mergeSort(arr, m1+1, m2);
     mergeSort(arr, m2+1, h);
     merge(a, l, m1, m2, h);
  }}
```



A → f → t
B → t → a
A → t → a
C → f → t
A → a → f
B → a → t

18/9

Counting Sort :

Time complexity : $\Theta(n+k)$

$\Theta(n+k)$

If $k = O(n)$,
Then $\Theta(n \log n)$
(OR) arr[2] = [1, 1000] — fails for $O(n)$

Radix Sort

Sort on least-significant digit first with auxillary stable sort.

$$T(n,b) = \Theta\left(\frac{b}{r}(n+2^r)\right)$$

- b: b bit word
- r: r-bit pieces
- n: n numbers

$\Theta(n+k)$

Counting sort $\rightarrow \Theta(n+2^r)$

No. of passes = b/r

Choose r to minimise $T(n,b)$:

$r \uparrow$ passes $\downarrow$

but $r \gg \log n$, $T(n,b) \uparrow$ exponentially

19/9 $T(n,b) = \frac{b}{r}(\Theta(n+2^r))$

if $r = \log n$:

$$T(n,b) = \frac{b}{\log(n)}(\Theta(n))$$

$$= \Theta\left(\frac{b \cdot n}{\log(n)}\right)$$

d: depth / no. of digits

$d = b/\log(n)$

$$= \Theta(b \cdot d)$$

Locality of Reference

→ BST :

Insertion : $O(n)$
Search : $O(n)$
Deletion : $O(n)$

Queue Implemented as array :

Insertion : $\theta(1)$

Deletion : $\theta(n)$

Heap — kind of tree but weakly ordered

$\theta(\log_2 n)$ — Insertion

$\theta(\log_2 n)$ — Deletion

→ node ≥ child

Stored and processed in RAM.

Can be Implemented by using arrays. (Priority Queue)

Removal : Deletion of root (heapArray[0])

└ Trickle Down — $\theta(2 \log 2)$

Insertion : Insertion of root

└ Trickle Up — $\theta(\log n)$

Search : $\theta(n)$

Asymptotically both are $\theta(\log n)$ but Insertion is faster

Building heap : $\theta(n \cdot \log n)$ — Insertion of n times

Heap sort : $\theta(n \log n)$

↳ Not stable ∴ Assign Key values and sort

Heap → Balanced tree

Algo	B.C	W.C	In-place	Stable
Bubble	$\theta(n^2)$	$\theta(n^2)$	✓	✓
Selection	$\theta(n^2)$	$\theta(n^2)$	✓	X
Insertion	$\theta(n)$	$\theta(n^2)$	✓	✓
Quick	$\theta(n \log n)$	$\theta(n^2)$	✓	X
Merge	$\theta(n \log n)$	$\theta(n \log n)$	X	✓
Heap	$\theta(n \log n)$	$\theta(n \log n)$	✓	X
Counting	$\theta(n+k)$	$\theta(n+k)$	X	✓
Radix	$\theta(dn)$	$\theta(dn)$	X	✓

Lower bound Theory - $\Omega(n \log n)$

Any comparison based sort.

Chapter - 2

Pruning : Ex.

Linear Search : $n \longrightarrow n-1 \longrightarrow n-2 \longrightarrow \ldots$

Binary Search : $n \longrightarrow n/2 \longrightarrow n/4 \longrightarrow \ldots$

↘ Excluding

Incremental design:

Ex. Insertion , Selection & bubble sort

Divide and Conquer :

Ex. Merge , Quick , Tower of Hanoi

Brute Force : All permutations

Greedy – Global optimum Along with proof of correctness

↳ min/max

Feasible ✓ → many optimal solutions

Optimal X

→ **Knapsack Problem :**

Given: $S = \{x_1, x_2, x_3, \ldots, x_n\}$

$\{w_1, w_2, w_3, \ldots w_n\}$

$P_1, P_2, \ldots \ldots P_n$

Objective: Find $S' \subseteq S$ at Profit (S') is maximum

Constraint: Weight $(S') \leq W$

Greedy : Pack as many objects as possible.

ROUGH

Q) $T(n) = n \cdot T(n-1) + n(n-1) \cdot T(n-2)$

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

$T(n-2) = (n-2) \cdot T(n-3) + (n-2)(n-3) \cdot T(n-4)$

$T(1) = T(0) = 1 \times T(0)$

$T(2) = 2T(1) + 2 \cdot 1 \cdot T(0) = 4T(0)$ $= 2 \times 2!$

$T(3) = 3T(2) + 3 \cdot 2 \cdot T(1) = 18T(0)$ $= 3 \times 6!$

$T(4) = 4T(3) + 4 \cdot 3 \cdot T(2) = 120\,T(0)$ $= 5 \times 4!$

$T(5) = 5T(4) + 5 \cdot 4 \cdot T(3) = 960T(0)$ $= 8 \times 5! \cdot T(0)$

$120 \times 5 + 5 \cdot 4 \cdot 18$

$600 + 360$ $= 13 \times 6!\ T(0)$

$T(6) = 6T(5) + 6 \cdot 5 \cdot T(4)$

$= 5760 + 3600$

$= 9360$

$T(n) = n! \times T(0) \times$

$T(n) + T(n-1) + T(n-2) = nT(n-1) + (n+1)(n-1) \cdot T(n-2)$ $\quad \frac{T(n-1)}{(n-1)!} + \frac{T(n-2)}{(n-2)!}$

$+ T(n-3) + \ldots \quad + n(n-2) \cdot T(n-3) + (n-1)(n-3) \cdot T(n-4)$

$+ \ldots$

$T(n) = (n-1) \cdot T(n-1) + (n^2-2)\,T(n-2) + (n^2-2n-1)\,T(n-3)$

$$T(n) = n! \times \left(\frac{T(n-1)}{(n-1)!} + \frac{T(n-2)}{(n-2)!} \right) \times T(0)$$

$O(n \times n!)$??

$O\left(\frac{\phi^n}{5!} \times n!\right)$??

10/10

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
$\{+1\}$	0	1	1	1	1	1	1	1	1	1	1	1
$\{+2\}$	0	1	6	7	7	7	7	7	7	7	7	7
$\{+3\}$	0	1	6	7	7	18	19	24	25	25	25	25
$\{+4\}$	0	1	6	7	7	18	22	24	28	29	29	40
$\{+5\}$	0	1	6	7	7	18	22	28	29	34	35	40

$$OPT(i, \omega) = \begin{cases} 0 & i = 0 \\ OPT(i-1, \omega) & \omega_i > \omega \\ \max\{OPT(i-1, \omega), v_i + OPT(i-1, \omega - \omega_i)\} & \text{otherwise} \end{cases}$$

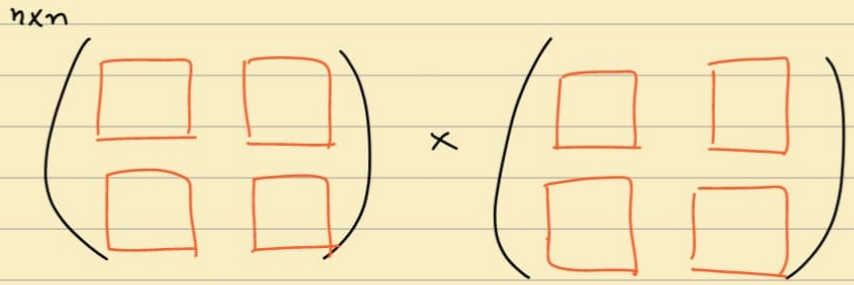
11/10

Matrix Multiplication:

$A_{n \times n}$, $B_{n \times n}$ $\longrightarrow$ $(AB)_{n \times n}$ $\longrightarrow$ $O(n^2 \times n) = O(n^3)$

$\hookrightarrow$ Until 1969

Strassen's Algo for matrix multiplication (1969)



$\frac{n}{2} \times \frac{n}{2}$

$\hookrightarrow$ Divide and Conquer

$\therefore T(n) = 8 \times T(n/2) + (< n^{\log_2 8})$

$\hookrightarrow f(n)$

$\therefore \theta(n^{\log_2 8}) \equiv \theta(n^3)$

Not in syllabus

Strassan Reduced to $T(n) = 7 \times T(n/2) + (< n^{\log_2 7})$

$\therefore \theta(n^{\log_2 7}) \approx \theta(n^{2.708})$

1970 : Coppersmith — $\theta(n^{2.3755})$

⋮

2014 : Le Gall — $\theta(n^{2.733})$

→ Matrix chain multiplication :

Input : $A_1, A_2, A_3, \ldots, A_n$

Output : $A_1 \times A_2 \times A_3 \times \ldots \times A_n$

min. no. of scalar multiplications $\quad [A_i \text{ cols} = A_{i+1} \text{ rows}]$

Ex. $A_{1(10\times100)}$, $A_{2(100\times5)}$, $A_{3(5\times50)}$

$\begin{pmatrix} p_0 = 10 \\ p_1 = 100 \\ p_2 = 5 \\ p_3 = 50 \end{pmatrix}$

$(A_1 \times A_2) \times A_3$

↳ 7500 multiplications

i.e. 5000 + 2500

$A_1 \times (A_2 \times A_3)$

↳ 75000 multiplications

i.e. 25000 + 50000

Brute Force ⟶ 2^n (Finding best paranthesization fashion)

DP:

$A_{ij} = A_i \times A_{i+1} \times \ldots \times A_j$

$\exists k,\ 1 \le k \le n-1$ such that $A_{1k} \times A_{(1+k)n}$ is the final multiplication

$$P(n) = \begin{cases} 1, & \text{if } n=1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k), & \text{if } n \ge 2 \end{cases}$$

Ex. A_1, A_2, A_3, A_4

$P(3) = 2$ ⟶ 2 different ways to multiply

$P(4) = 5$

$A_1 \times (A_2 \times A_3)$

$(A_1 \times A_2) \times A_3$

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

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

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

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

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

$P(5) = 14$

$\therefore 2^n$ different ways

↳ Upper bound

$P(2) \le 2^1$

$P(3) \le 2^2$

$P(4) \le 2^3$

$P(5) < 2^4$

↳ 2^{n-1}

$\Omega(2^n)$

$$A_{ij} = A_i \times A_{i+1} \times A_{i+2} \times \ldots\ldots \times A_j$$

$$m_{ij} = \begin{cases} 0 & , i=j \\ \min_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & , i<j \end{cases}$$

m_{ij} → No. of multiplications

Recursive Solution

Cost of Multiplying : $A_{ik} \times A_{(k+1)j}$ $(\exists k)$

$m[i,i] = 0$ ∵ Chain contains single matrix A_i

11/10

$n = 4$

$p = 3\ 7\ 6\ 2\ 9$

	1	2	3	4
1	0			
2	126	0		
3	126	84	0	
4	180	210	108	0

```
for l → 2 to n do
    for i = 1 to n-l+1 do
        for j = i+l-1 {right position}
            m[i,j] ← ∞
            for k = i to j-1 do
                m[i,j] = min{m[i,j], m[i,k] + m[k+1,j] + p_{i-1} p_j p_k}
```

Time complexity :

$n \times n = n^2$

$n + \frac{n^2-n}{2}$

$\frac{n^2+n}{2}(n-1)$

$= O(n^3)$ → D.P

Brute force : $\Omega(2^n)$

Result : $(A_1 \times (A_2 \times A_3)) \times A_4$

Note : Keep track of Optimal choices (k)

→ $B(n)$: no. of ways a Binary search trees can be constructed

$P(1) = 1$	$B(1) = 1$
$P(2) = 1$	$B(2) = 2$
$P(3) = 2$	$B(3) = 5$
$P(4) = 5$	$B(4) = 14$
$P(5) = 14$	

Catalan's number :

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

25/10

Theorem : A subpath of a shortest path is a subpath.

Theorem : Triangular Inequality

$$\forall u, v, x \in V \; ; \quad \delta(u,v) \leq \delta(u,x) + \delta(x,v)$$

→ Dijkstra's algorithm :

```
d[s] ← 0
∀ v ∈ V - {s}
    do d[v] ← ∞
S ← ∅
Q ← V
while Q ≠ ∅
  do u ← EXTRACT_MIN(Q)
     S ← S ⋃ {u}
     ∀ v ∈ Adj[u]
       do if d[v] > d[u] + w(u,v)
          then d[v] = d[u] + w(u,v)
```

Q → Priority Queue , maintaining V-S, keyed on d[v]

- → fibonacii heap
- → binomial heap
- → Array

Handshaking lemma :

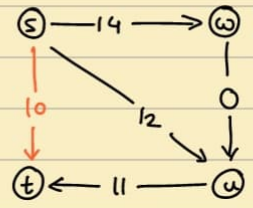
$$\Theta(V \cdot T_{extract\_min} + E \cdot T_{decrease\_key})$$

Q	$T_{extract\_min}$	$T_{decrease\_key}$	Total
Array	$O(V)$	$O(1)$	$O(V^2)$
Binary heap	$O(\log V)$	$O(\log V)$	$O(E \log V)$
fibonacii heap	$O(\log V)$	$O(1)$	$O(E + V \log V)$

Note :

(1) only works if edge weights are non-negative.

(2) Adding constant to every does not necessarily produce shortest path.



28/10

Hence, we use Bellman-Ford algorithm.

Someone once said

"Meow Meow Meow Meow"

Meanwhile in a parallel universe

Meow Meow Meow

"fuck you!"

As flies to wanton boys, we are for the gods, they kill us for the sport, soon the science will not only be able to slow down the aging of the cells, but soon the science will fix the cells to the State. We will become eternal, only accidents, crimes and wars will kill us. But unfortunately crimes and wars will multiply

I Love football. Thank you.

– Parth Pandey
I 6

meow

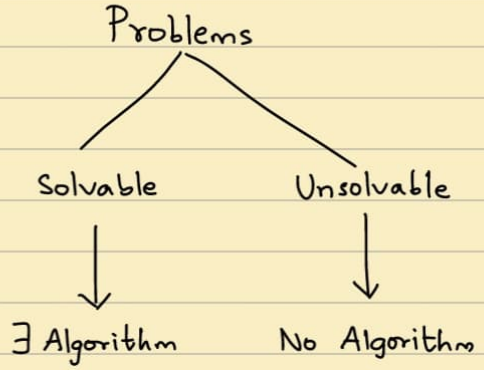
↳

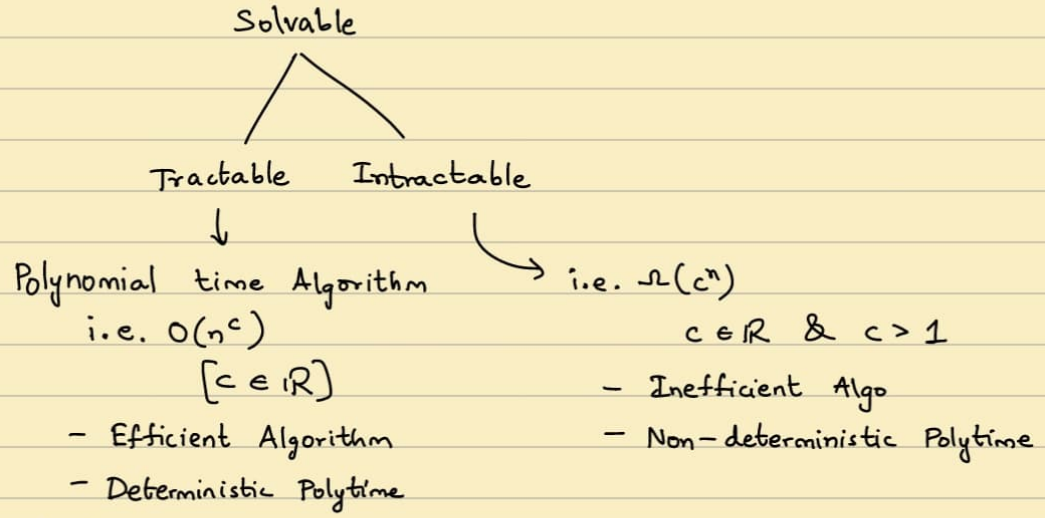
" oh my god you people
or god fly Hey
breathe fight &shan
drink none live
till you would "

29/10

Variable length encoding ⟶ Prefix property
↳ Huffmann code
↳ Greedy Algorithm

5/11





→ Cooke and Levin

P : A decision problem
Can be solvable in Deterministic polytime

NP - A decision problem
① Solvable in non-deterministic polytime
② Verifiable in deterministic polytime.
Ex. 2-SAT (2-CNF)

$x_1, x_2, x_3, \ldots\ldots, x_n$

$E = (x_1 \vee x_2) \wedge (x_1' \vee x_2') \wedge \ldots\ldots$

↳ $\Omega(2^n)$

Verifying - $\Theta(2n)$

6/11

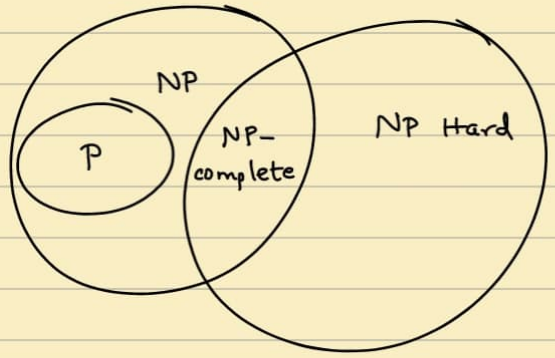
B is NP-complete if

(i) B is NP

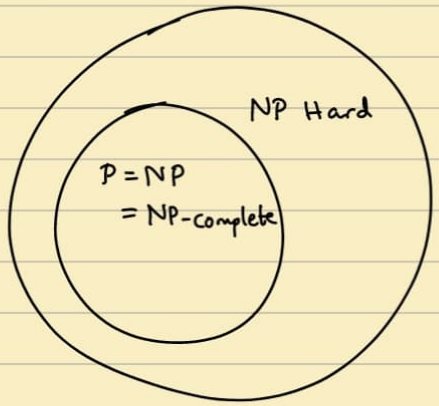
(ii) A is NP-complete

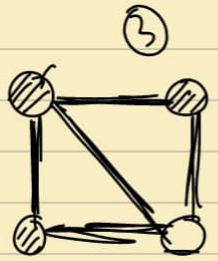
$A \leq_p B$

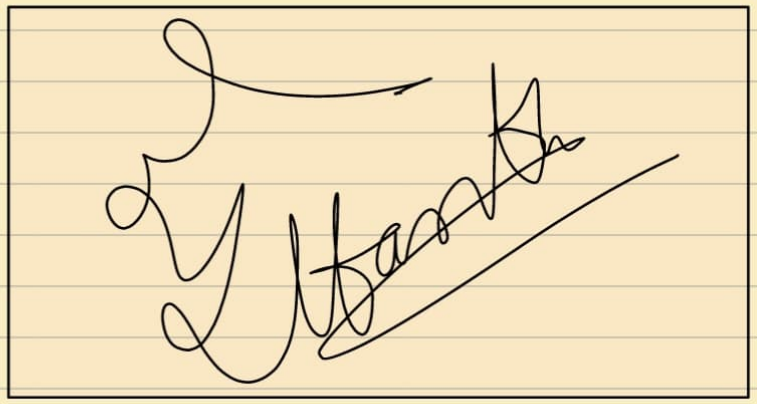
$P \neq NP$



$P = NP$







BACKTRACKING

↳ BASICALLY recursion OK
↳ BYE-

Knapsack using Branch & Bound

item	weight	value	$\frac{val}{weight}$
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

$W = 10$

brute force $= 2^n$

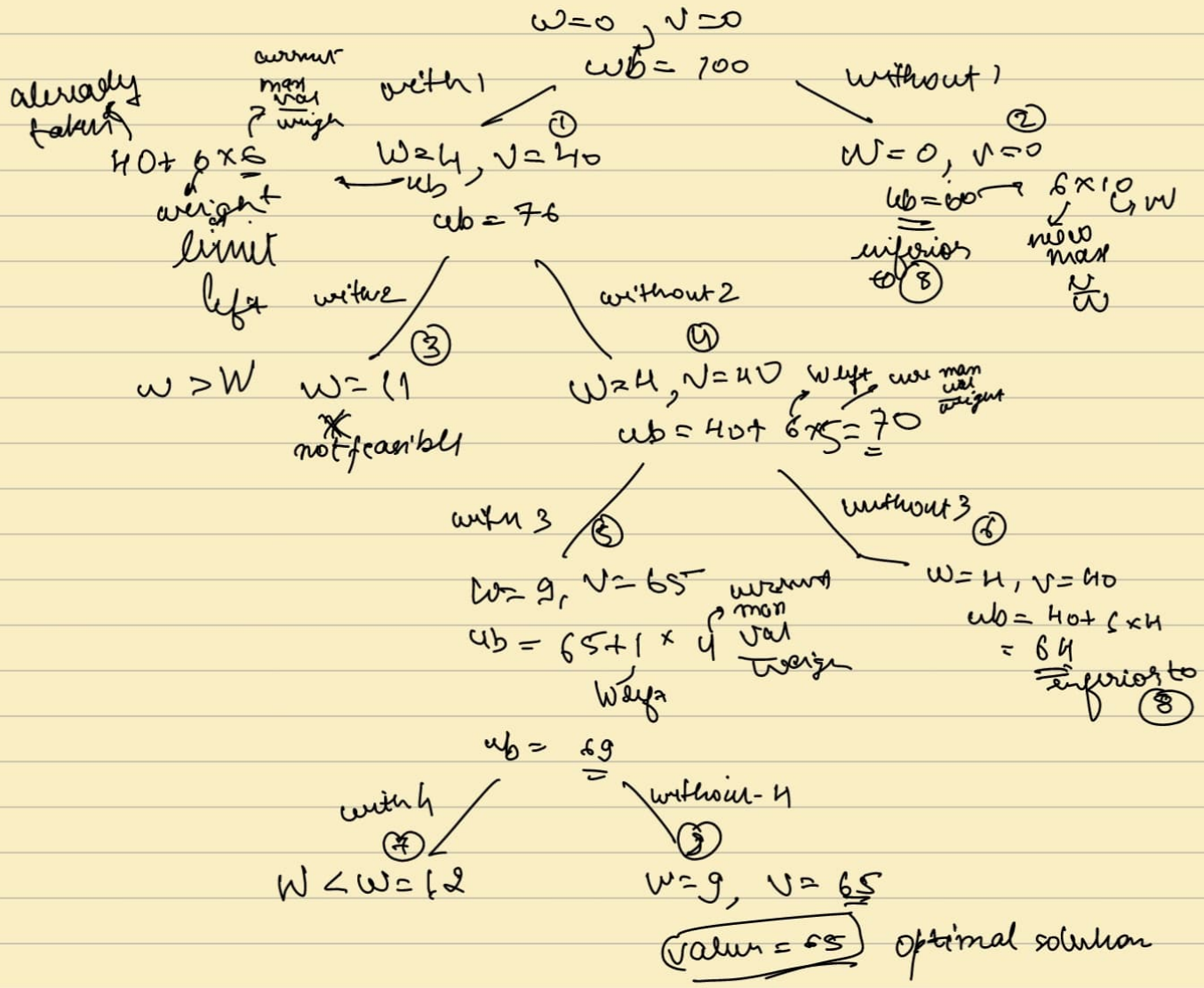
initial upper bound

max val/wei = 10

W = 10

maxprofit = 10 × 10 = 100

wei	val	val/wei
4	40	10
7	42	6
5	25	5
3	12	4



Dynamic Programming

L1

→ Greedy Algos – Fractional Knapsack
↓ – Coin change

Optimization
(Maximisation / Minimization)

Greedy : Local Optimum → Local Optimum → .... → Global Optimum

Charecteristics of any Greedy Algo :
(Greedy Strategy + Proof of correctness)

① Optimal solution to a problem lies with optimal solutions to its sub problems : Optimal Substructure

Ex. Coin change problem :

$(1, 3, 4)$, $x = 6$

Greedy : 4, 1, 1 #coins = 3

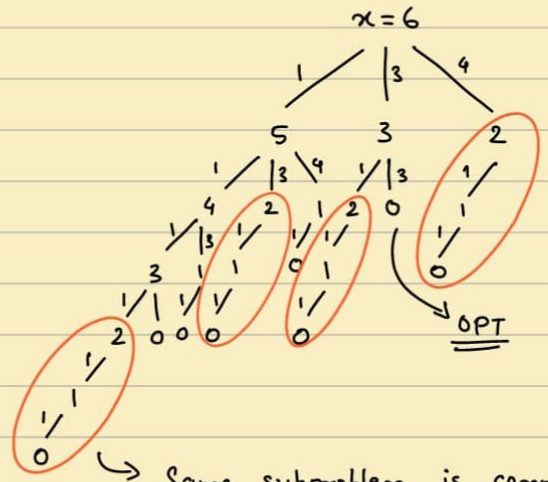
OPT : 3, 3 #coins = 2

↳ Explore all possible solutions before deciding OPT.
↳ This is not same as Brute force.
Difference : Overlapping Subproblems

② Overlapping Subproblems :

Ex. Coin change problem $(1, 3, 4)$, $x = 6$

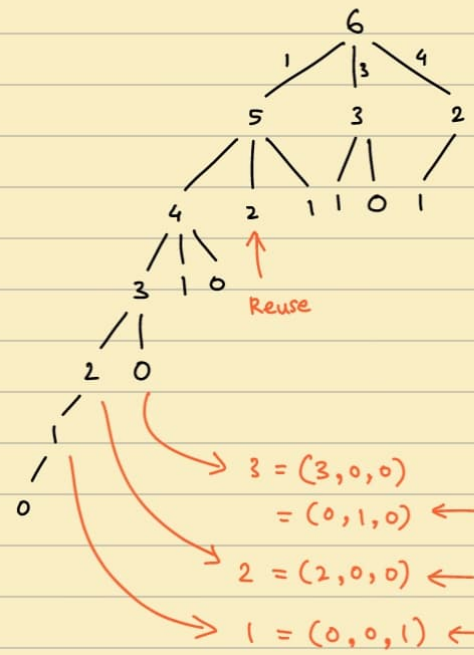
Brute Force :



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

↳ Same subproblem is computed more than once.
↳ Can we compute the overlapping subproblem exactly once ?
↳ An improvement over simple Brute Force.

L2



OPT :

$$c[6] = \min\{1 + c[5], 1 + c[3], 1 + c[2]\}$$

$$c[4] = \min\{1+c[3], 1+c[1], 1+c[0]\}$$
$$= 1$$

3 = (3,0,0)
= (0,1,0) ← OPT $\Rightarrow c[3] = 1$

2 = (2,0,0) ← OPT $\Rightarrow c[2] = 2$

1 = (0,0,1) ← OPT $\Rightarrow c[1] = 1$

Note : c[x] : Min no. of coins to give change for ₹$_x$.

$$c[6] = \text{Min}\{1+2, 1+1, 1+2\}$$
$$= 2$$

$1+1$ → (, 1,) + c[3]

2 → No. of coins

c[6] = (0, 2, 0) ← Solution

Solutions are being reused ⇒ Subproblems are calculated only once.

L3

$$\therefore c[x] = \min\{1 + c[x-1], 1 + c[x-3], 1 + c[x-4]\}$$

c[x] → min no. of coins

c[x-1] c[x-3] c[x-4]

Optimal substructure : Recursive subproblems in Bottom Up fashion

Base case :

$c[0] = 0$

$c[1] = 1$

$c[3] = 1$

$c[4] = 1$

$c[2] = 1 + c[1]$

$c[5] = \min\{1+c[4], 1+c[2], 1+c[1]\}$

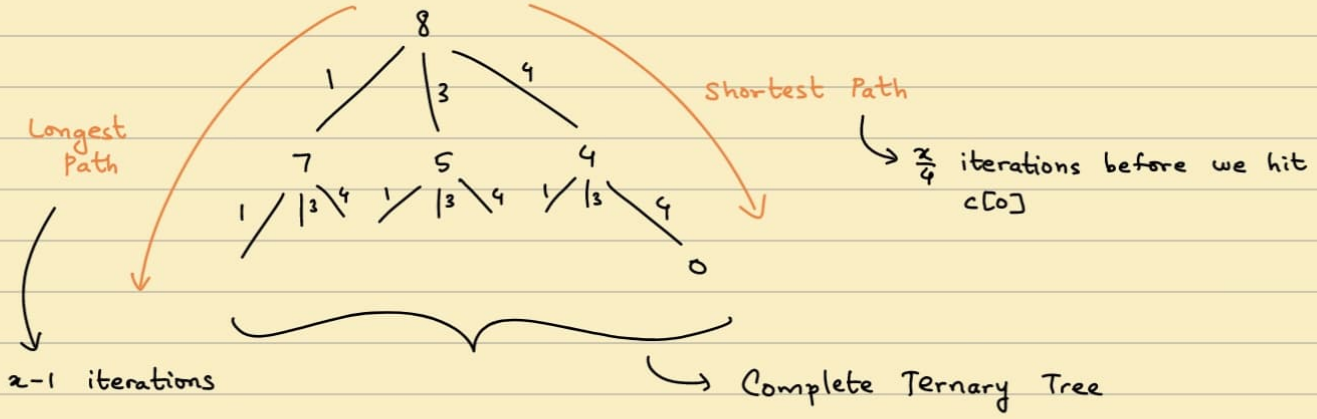
$d_1, d_2, \ldots\ldots, d_k$

$$c[x] = \min\{1 + c[x-d_1], 1 + c[x-d_2] + \ldots\ldots, 1 + c[x-d_k]\}$$

$c[d_1] = 1$
$c[d_2] = 1$
$\vdots \quad \vdots \quad \vdots$
$c[d_k] = 1$

Brute force : At what level the first 0 is seen.
At what level the last 0 is seen.

↳ Size of Recursion Tree — To find Time Complexity.



↳ $\frac{x}{4}$ iterations before we hit c[0]

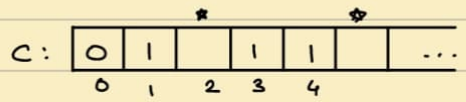
Size of Complete 'Sub' Ternary Tree lower bounds the no. of Recursive calls (Time complexity)

Size of Sub Ternary Tree $\longrightarrow \Omega(3^{x/4})$

↳ Exponential in I/P 'x'
(I/P value magnitude)

$\Rightarrow$ Brute force Algo runs in Expo time.

Dynamic Programming



Poly time, Poly in 'x'

General case:

$d_1, d_2, \ldots\ldots, d_k$ } $- x/D_{max}$

$D_{max} = \max\{d_1, d_2, \ldots\ldots, d_k\}$

Brute Force: $\Omega(k^{x/D_{max}})$

L4

$D = d_1, d_2, \ldots\ldots, d_k$

$D_{max} = Max(d_1, d_2, \ldots\ldots d_k)$

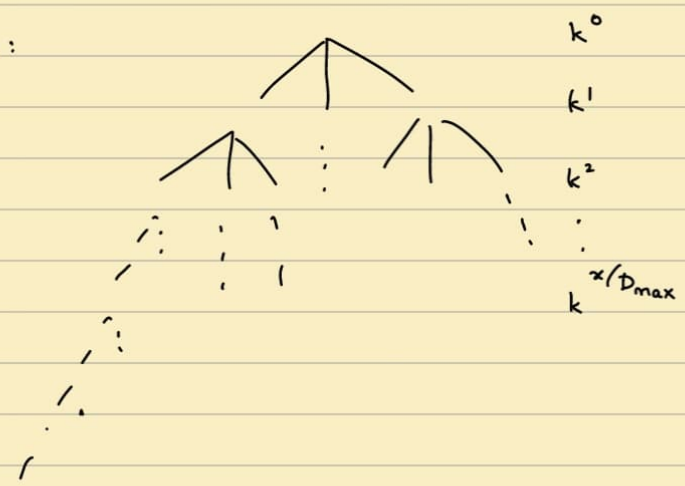
Brute force

↳ Size of the k-ary tree

$\Omega(k^{x/D_{max}})$

Time complexity of Brute Force:

$T(x) = T(x-d_1) + T(x-d_2) + \vdots + T(x-d_k) + O(1)$



$T(x) = T(x-1) + T(x-3) + T(x-4) + O(1)$

$C[x] = \min\{1 + c[x-1], 1 + c[x-3], 1 + c[x-4]\}$

↳ Solution

TC: $\Omega(k^{x/D_{max}})$

Atleast

$T(x) \leq 3 \cdot T(x-1) + O(1)$

$= O(3^x)$

$T(x) \geq 3 \cdot T(x-4) + O(1)$

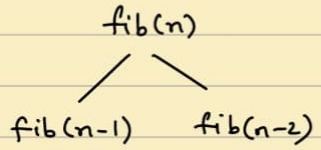
$= \Omega(3^{x/4})$

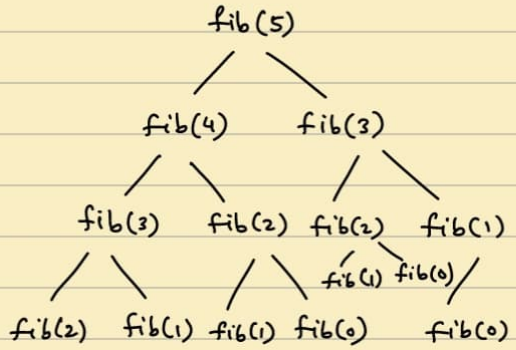
If Greedy fails $\Rightarrow$ Dynamic Programming

Provided overlapping subproblems exist.

Dynamic Programming is better than Brute Force.

Ex. Fibonacii :





Overlapping Subproblems ✓

Optimal Sub problem ✗

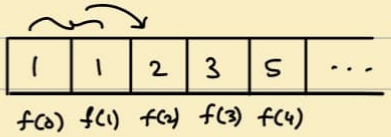
$fib(0) = fib(1) = 1$

$$T(n) = T(n-1) + T(n-2) + O(1)$$
$$\leq 2T(n-1)$$
$$\geq 2T(n-2)$$

$T(n) \leq 2T(n-1) \implies O(2^n)$

$T(n) \geq 2T(n-2) \implies \Omega(2^{n/2})$

Optimal Method :



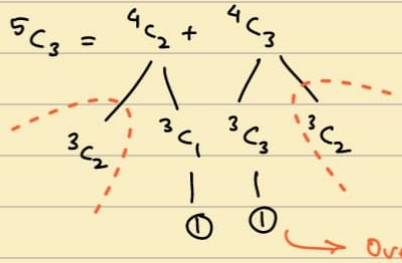
$\implies O(n)$

Computing Fib(n) with overlapping subproblem

[DP based solution]

Ex. ${}^nC_r$

$$C(n,r) = {}^nC_r$$
$${}^nC_r = {}^{n-1}C_{r-1} + {}^{n-1}C_r$$



Overlapping Subproblem

${}^nC_1$, ${}^nC_n$ $\implies$ Return 1

Computing ${}^nC_r$ + Overlapping Subproblem = Poly-time Algo

$$T(n,r) = T(n-1,r-1) + T(n-1,r) + O(1)$$

L5

Q) Sorting Variant

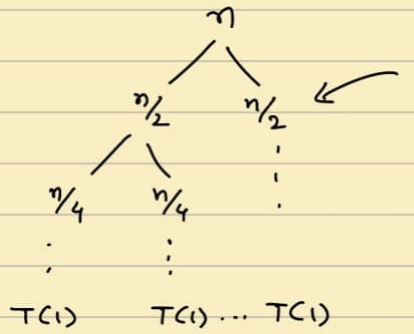
I/P : $A = \langle a_1, a_2, a_3, \ldots\ldots, a_n \rangle$

Q : Sort A

Obj : Sort A with min. no. of comparisons

Sol:

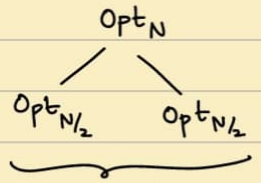
Method 1 — Divide & Conquer Algo [DP]



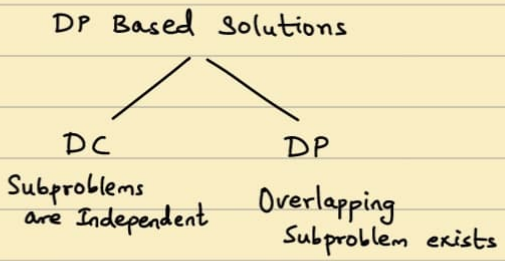
Sub problems are Independent.

⇒ No Overlapping Subproblems

But Optimal Substructure exists



Opt Substr exists



L6:

0-1 Knapsack:

Given : $x_1, x_2, \ldots\ldots, x_n$ objects

$W_1, W_2, \ldots\ldots, W_n$ weights - W

$P_1, P_2, \ldots\ldots, P_n$ profits

Find $S \subseteq \{x_1, x_2, \ldots\ldots, x_n\}$ S.T. $W(S) \leq W$

Profit (S) is maximum

Ex.

	x_1	x_2	x_3
W_i	2	4	6
P_i	10	5	3

$W = 10$

$S = \{x_1, x_2\}$

$W(S) = 2 + 4 \leq 10$

$P(S) = 10 + 5 = 15$

$S = \{x_1', x_2', \ldots\ldots, x_k'\}$

$S' \subset S$

Opt

Profit (S') must be maximum

Opt $[n, w]$ ⇐ Optimum solution with n-objects, capacity = w

opt $[8, 10]$

opt $[2, 10]$ opt $[2, 10-6]$

opt $[5, 10]$ ⟶ Optimum solution with 5 objects $\{x_1, x_2, \ldots, x_5\}$, $w = 10$

opt $[10, 200]$ ⟶ Optimum solution with 10 objects $\{x_1, x_2, \ldots, x_{10}\}$, $w = 200$

Opt $[n, w]$ ⟶ Optimum solution with $x_1, \ldots, x_n$, capacity = w

x_n is present in OPT = ①

Suppose $x_n \in$ OPT

⇒ OPT $[n, w]$ = OPT $[n-1, w-w_n]$ + P_n

$w_n \leq w$

If $w_n > w$ ⇒ $x_n \notin$ OPT

x_n is absent = ⓪

Suppose $x_n \notin$ OPT

⇒ OPT $[n, w]$ = OPT $[n-1, w]$

$$OPT[n, w] = \text{Max}\{OPT[n-1, w],\ P_n + OPT[n-1, w-w_n]\}$$

If $w_n > w$ ⇒ OPT $[n, w]$ = OPT $[n-1, w$

— Recursive Subproblem :

$$OPT[i, w] = MAX\{OPT[i-1, w],\ P_n + OPT[i-1, w-w_i]\}$$

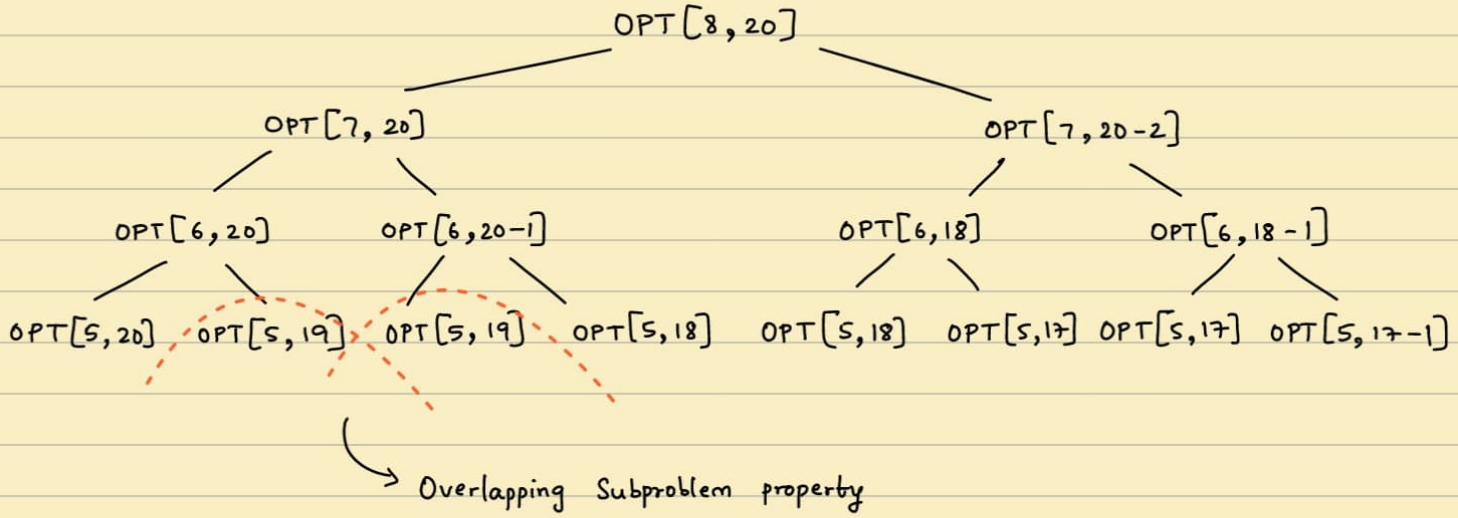
If $w_i > w$ ⇒ OPT $[i, w]$ = OPT $[i-1, w]$

L7

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
w_i	5	8	4	1	3	1	1	2
P_i	10	25	15	5	30	10	20	50

$W = 20$

OPT $[8, 20]$

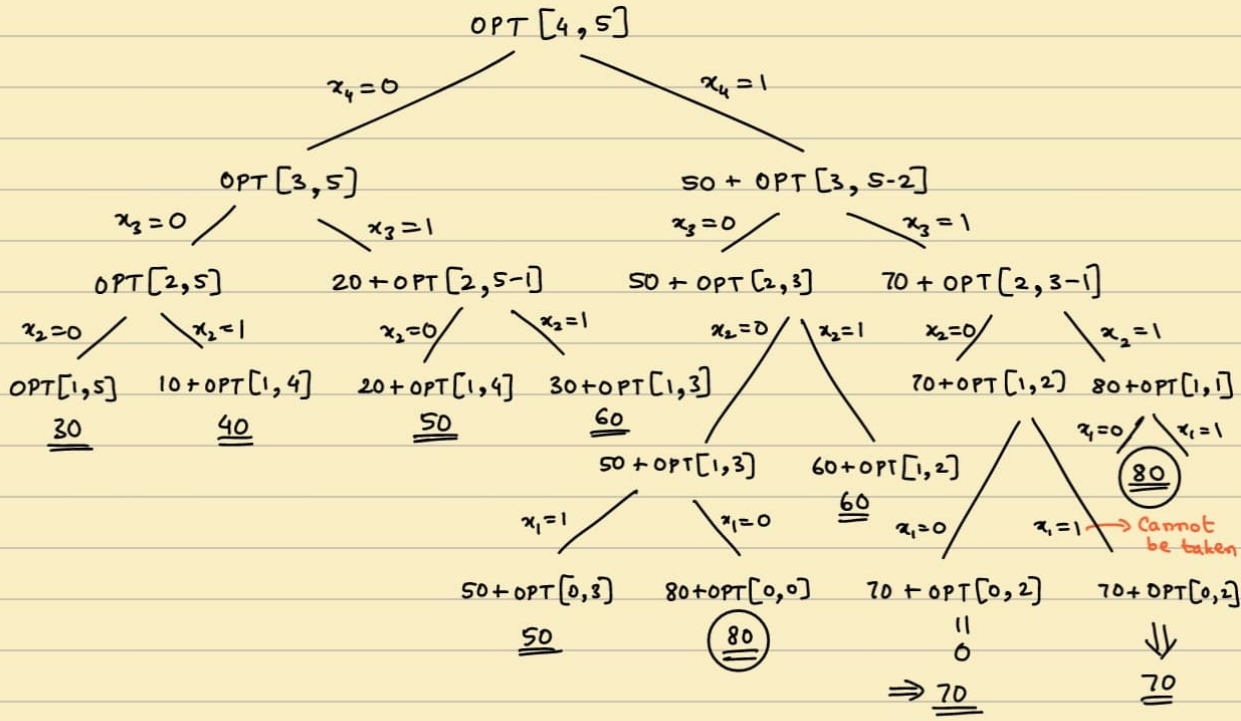


L8

Tracing Knapsack problem :

	x_1	x_2	x_3	x_4	
w_i	3	1	1	2	$W = 5$
P_i	30	10	20	50	
	0/1	0/1	0/1	0/1	

$2 \times 2 \times \dots \times 2 \Longrightarrow 2^n$ possibilities



$OPT[1, 2] = OPT[0, 2]$

$\because w_i > w \Rightarrow OPT[i, w] = OPT[i-1, w]$

$OPT[0, 1] = OPT[1, 1]$

$OPT[0, x] = 0$

Size of Recursion Tree : $\Omega(2^n)$ (or) $\theta(2^n)$

L9

$OPT[4,5] = Max\{ OPT[3,5] , P_4 + OPT[3,3] \}$

x_4	0	20	50	70	80	80
x_3	0	20	30	30	50	50
x_2	0	10	10	30	40	40
x_1	0	0	0	30	30	30
0	0	0	0	0	0	0
	0	1	2	3	4	5

x ↑ , W →

$OPT(0,w) = 0$

$OPT(x,0) = 0$

$OPT(1,1) = OPT(0,1)$

$OPT(2,1) = Max\{OPT(1,1) , 10 + OPT(1,0)\}$
$= 20$

$OPT(3,1) = Max\{OPT(2,1) , 20 + OPT(2,0)\}$
$= 20$

$OPT(4,1) = OPT(3,1)$ $[w_4 > w]$

$OPT(1,2) = OPT(0,2) = 0$ $[w_1 > 2]$

$OPT(2,2) = Max\{OPT(1,2) , 10 + OPT(1,1)\}$

$OPT(3,2) = Max\{OPT(2,2) , 20 + OPT(2,1)\}$

$OPT(4,2) = Max\{OPT(3,2) , 50 + OPT(3,0)\}$

$OPT(1,3) = Max\{OPT(0,3) , 30 + OPT(0,0)\} = 30$

$OPT(2,3) = Max\{OPT(1,3) , 10 + OPT(1,2)\} = 30$

$OPT(3,3) = Max\{OPT(2,3) , 20 + OPT(2,2)\} = 30$

$OPT(4,3) = Max\{OPT(3,3) , 50 + OPT(3,1)\} = 70$

$OPT(1,4) = Max\{OPT(0,4) , 30 + OPT(0,1)\} = 30$

$OPT(2,4) = Max\{OPT(1,4) , 10 + OPT(1,3)\} = 40$

$OPT(3,4) = Max\{OPT(2,4) , 20 + OPT(2,3)\} = 50$

$OPT(4,4) = Max\{OPT(3,4) , 50 + OPT(3,2)\} = 80$

$OPT(1,5) = Max\{OPT(0,5) , 30 + OPT(0,2)\} = 30$

$OPT(2,5) = Max\{OPT(1,5) , 10 + OPT(1,4)\} = 40$

$OPT(3,5) = Max\{OPT(2,5) , 20 + OPT(2,4)\} = 60$

$OPT(4,5) = Max\{OPT(3,5) , 50 + OPT(3,3)\} = 80$

x_1, x_4 OR x_2, x_3, x_4 ⟶ 80

L9

DP Table : $(n+1)$ rows, $(W+1)$ columns } size of DP Table : $(n+1)(W+1)$

DP Runtime : $\theta(nW)$

↳ Pseudo Polytime i.e. W can depend on n

I/P : $x_1, x_2, \ldots\ldots, x_n$, W

$W = O(1)$ ←

Brute force : $\theta(2^n)$

Exp in I/P size

DP Approach : $\theta(nw)$; Pseudo polytime

If $W = O(n^k)$, Then DP : Polytime

$[k \in I]$

Graph Algorithms

L1

→ Graph Traversals
- Breadth First Search (BFS)
- Depth First Search (DFS)

Tree Traversals
- Inorder
- Preorder
- Postorder
- Level order

→ Vertices : Nodes
Edges : Links

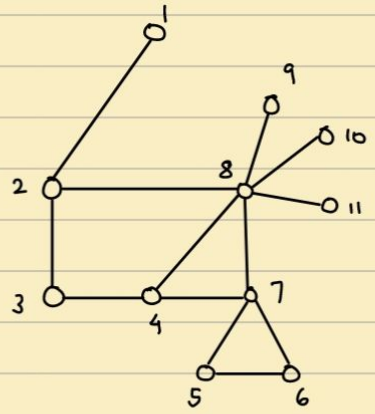
→ Traversals :

① BFS :

- Start from a node
- Explore all neighbouring nodes and mark them as visited.
- Level by level, perform this recursively

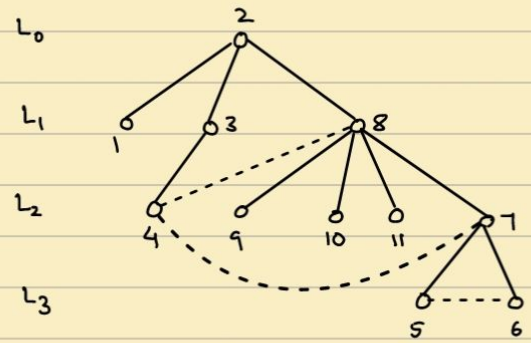
Output : Tree (BFS Tree)

Missing Edges : E(G) − E(T)
└ Edges



G

BFS (2)

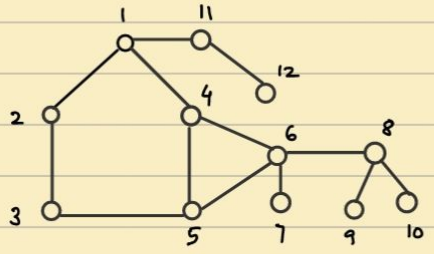


T

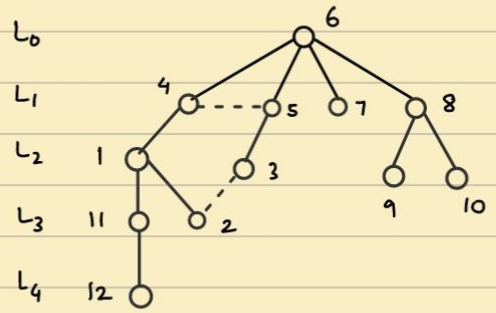
T + missing edge ⇒ A cycle in G

- C_1 − 2 3 4 8
- C_2 − 2 3 4 7 8
- C_3 − 7 5 6

Ex.



BFS(6):

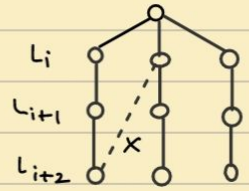


L2

Missing Edges → Cross Edges (Level No. of end points are same)

Missing Edges → Slanting Edges (Level No. of end points differ by 1)

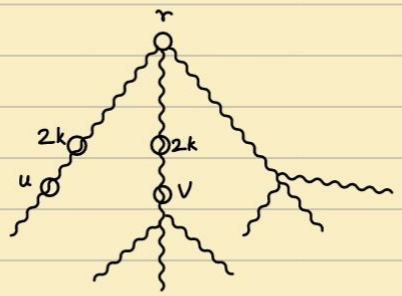
Note : Difference > 1 is not possible.



Presence of Slanted Edges : Even cycles

Presence of Cross Edges : Odd cycles

If missing edge exists $\Rightarrow \exists$ a cycle



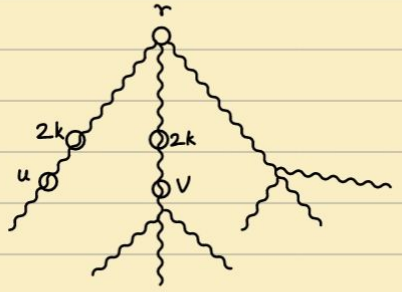
If path (r, u) is even :

$2k + 2k - 1$: ODD

If path (r, u) is odd :

$(2l+1) + (2l+1) - 1$: ODD

$\therefore$ If Cross Edge exists $\Longrightarrow \exists$ Odd cycle



If path (r, u) is even :

$2k + (2k+1) - 1$: EVEN

If path (r, u) is odd :

$(2l+1) + (2l+2) - 1$: EVEN

$\therefore$ If Slant Edge exists $\Longrightarrow \exists$ even cycle

→ Application of BFS:

① Test for cyclicity
   I/P : Graph G
   Does G have a cycle ?

② Test for odd cyclicity
   I/P : Graph G
   Does G have a odd cycle ?

③ Test for Tree
   I/P : Graph G
   is G is a Tree ?

④ Test for Connectedness

⑤ Test for Bipartiteness

⑥ Test for 2-Colourability

BFS Tree :

If Missing Edge set is non-empty $\Rightarrow$ $\exists$ cycle

If Cross Edge $\Rightarrow$ $\exists$ Odd cycle

Tree : Connected + Acyclic Graph

If BFS(G) yields a BFS Tree $\Rightarrow$ G is connected

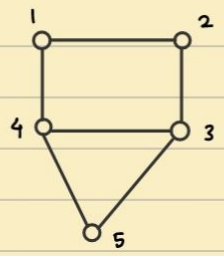
Else $\Rightarrow$ G is disconnected

↳ i.e. BFS(G) yields a BFS forest

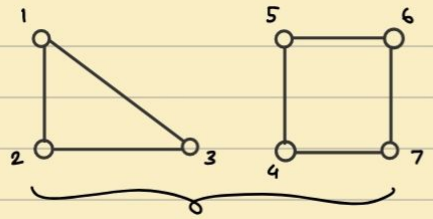
L3

G is connected $\Rightarrow$ If $\forall$ pairs of vertices (u, v), $\exists$ a path (u,v) in G

If $\exists$ a pair (u, v) S.T. No path (u,v) $\Rightarrow$ G is disconnected.

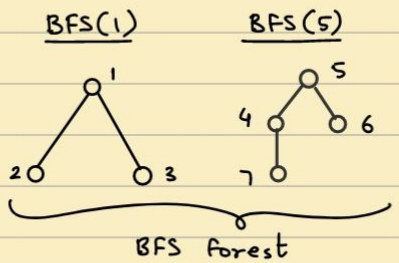


G_1



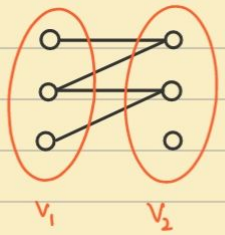
G_2

path(1,4) does not exist.

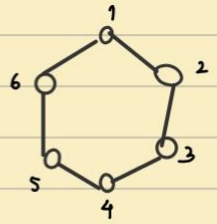


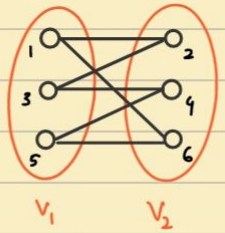
- **Bipartite Graph :**

  A Graph G is bipartite if G can be partitioned into two sets V_1, V_2 S.T. Each edge (u,v) is b/w V_1 & V_2.



Ex.





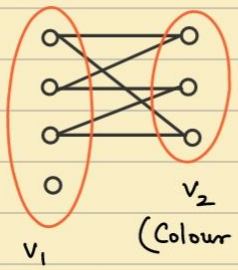
Not possible for 5 in the same format

Odd cycled Graphs are never bipartite.
i.e. G is bipartite iff G is odd cycle free.

(1) Run BFS
(2) Check for cross Edges in Missing Edges set
If yes $\Rightarrow$ Odd-cycle
$\Rightarrow$ Non-bipartite
If no $\Rightarrow$ Odd-cycle free
$\Rightarrow$ Bipartite

- **Proper Colouring :**

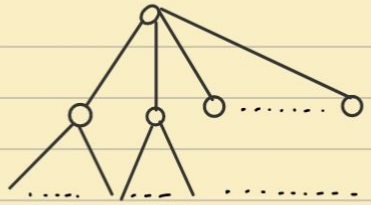
  Colour $V(G)$ S.T. adjacent vertices recieve diff. colour
  → Use minimum no. of colours



$\therefore$ G is 2 colourable iff G is a bipartite Graph

Note: Bipartite iff 2-colours
Tripartite iff 3-colours

- Time complexity:



$n = |V(G)|$

$m = |E(G)|$

BFS runs in $O(n+m)$ time

↳ Part of the I/P

Linear in I/P size

$m = O(n^2)$ $\left[\because m \leq \underbrace{\frac{n(n-1)}{2}}_{O(n^2)}\right]$

$\Rightarrow O(n+n^2) = O(n^2)$

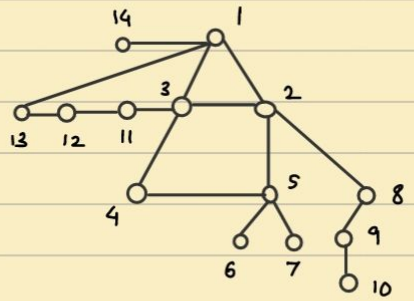
↳ Quadratic in I/P size

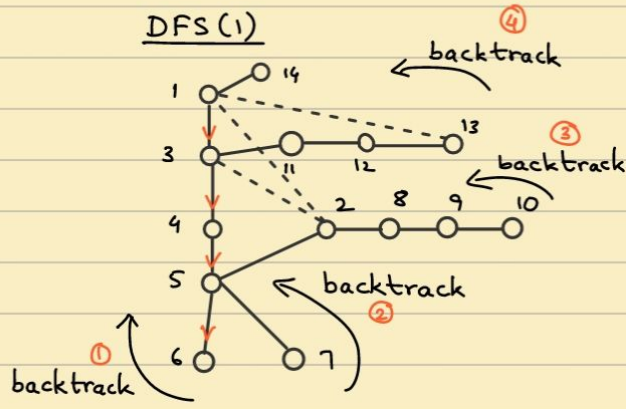
L4

② DFS:

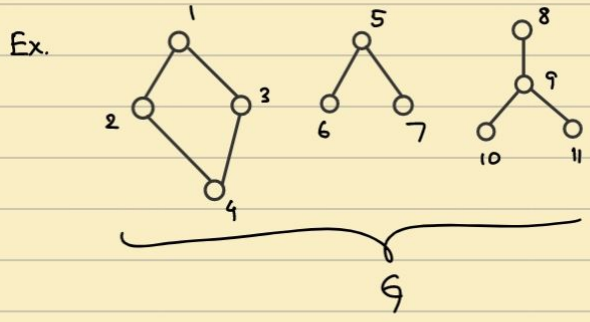
- Take any unvisited Node
- Choose a non-visited neighbour recursively until there are none left.
- Backtrack to immediate parent.
- Continue DFS

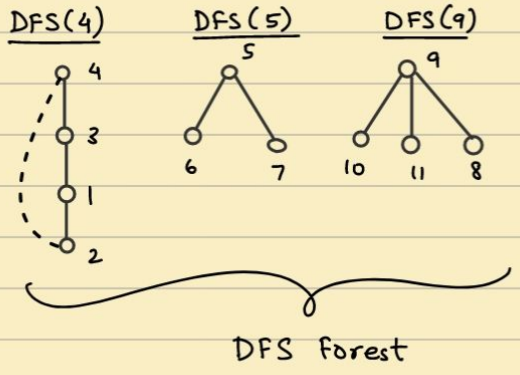
Ex.





Ex.





Each missing edge gives cycles

- Applications of DFS :

① Connectedness :

If DFS Tree $\Rightarrow$ G is connected

If DFS forest $\Rightarrow$ G is disconnected

② Cyclicity :

If $\exists$ Back Edge $\Rightarrow$ $\exists$ Cycle (Back Edge : Non Tree Edge)

③ Test for Odd/Even cyclicity

$\exists$ Cycle $\Rightarrow$ With the length of cycle

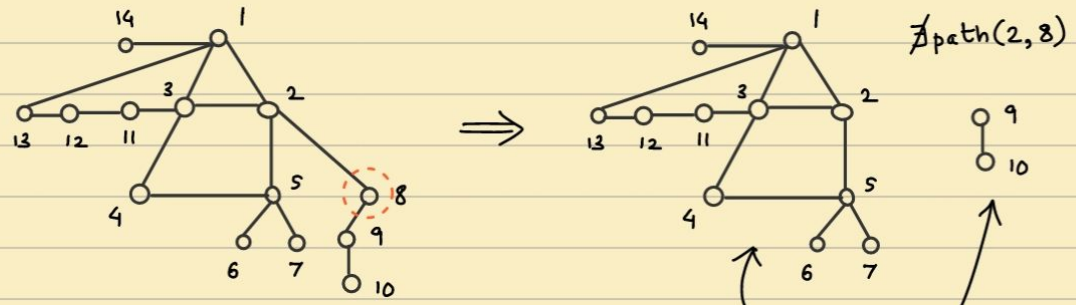
ODD EVEN

L5

④ Test for Articulation points (Cut vertices) and Bridges (Cut Edges)

Only defined for connected Graphs

Given a connected graph $[\forall u, v \ \exists path(u, v)]$
if $G - v$ is disconnected for some $u \in V(G)$,
then v is an Articulation point.
($G - V \equiv$ on Removal of v and its edges)

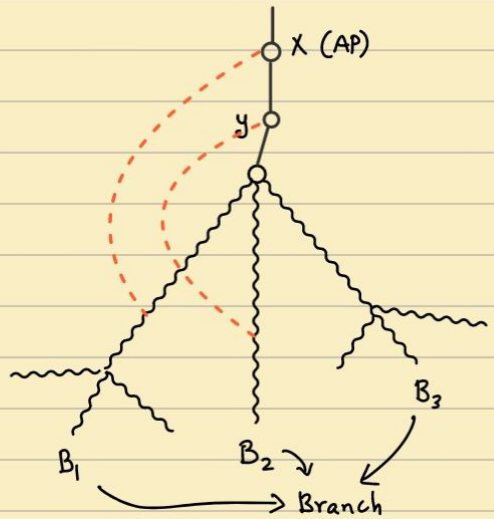


1, 5, 8, 9 $\longrightarrow$ Articulation points

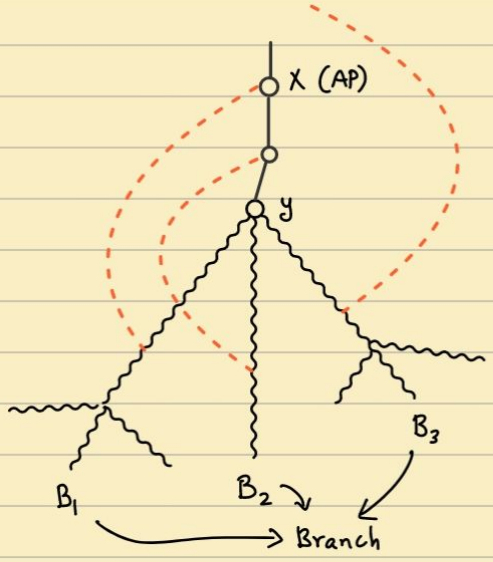
Maximal connected Subgraph

i.e. Cannot add any vertex & increase the connectedness

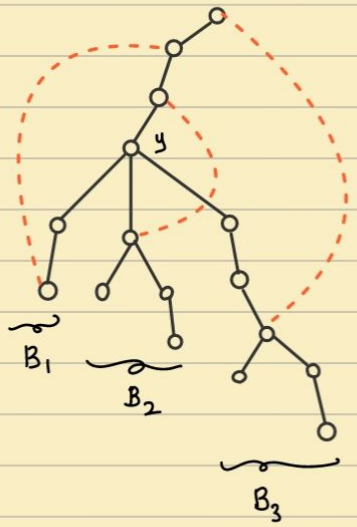
L6

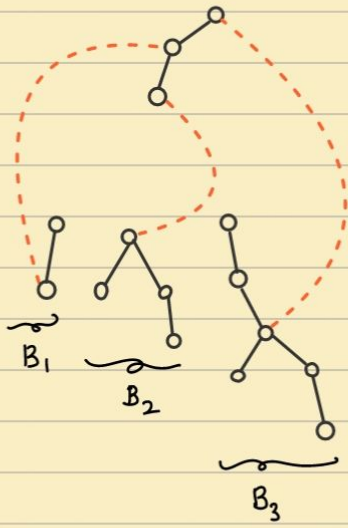


No back edge from any descendent (y) in B_3 to carry Ancestor (y)



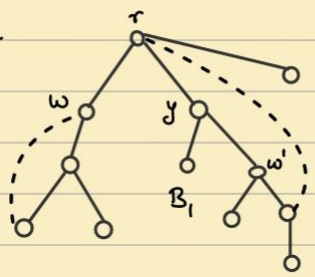
If $G-y$ connected ?





- Articulation Point (AP) : The root vertex is an articulation point if the degree of root in atleast two in DFS tree.
  - For a non-root vertex , say 'y', $(B_1, B_2, \ldots, B_k)$: Branches of 'y'. If $\exists B_i$ S.T. No Backedge from any desc(y) to any anc(y). Then , $y \in$ A.P.

Ex.

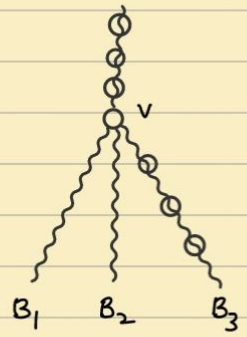


y is AP

w is AP

w' is AP

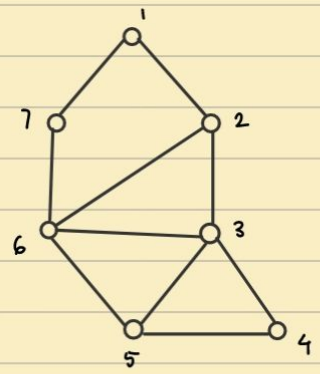
L7



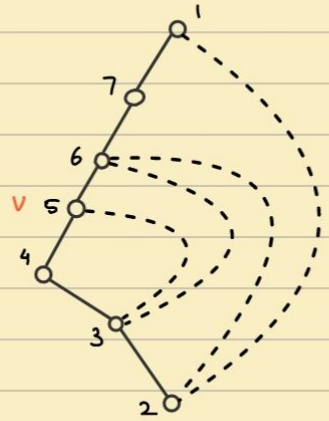
$\exists B_i$ S.T. No Backedge from any descendent (v) to any ancestor(v)

Note : The missing edges are within a branch and not between the branches.

Ex.



DFS (1)

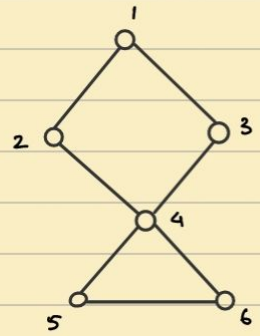


5 is not an AP

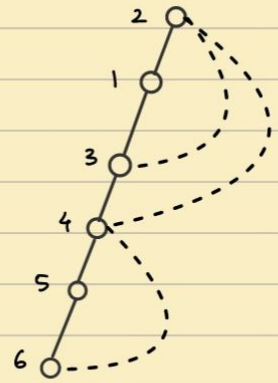
No point is an AP in this graph.

As a node below 5 i.e. desc(5) gives a backedge to a node above 5 i.e. asc(5)

Ex.



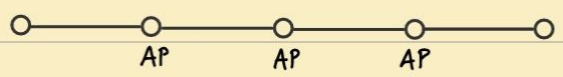
DFS (1) :



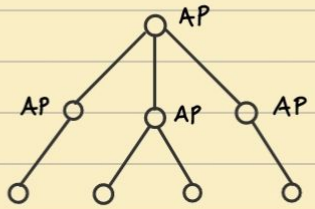
Only 4 is an AP

Rest all are not.

L8



Path Graph : All Internal Nodes are APs , i.e. n-2

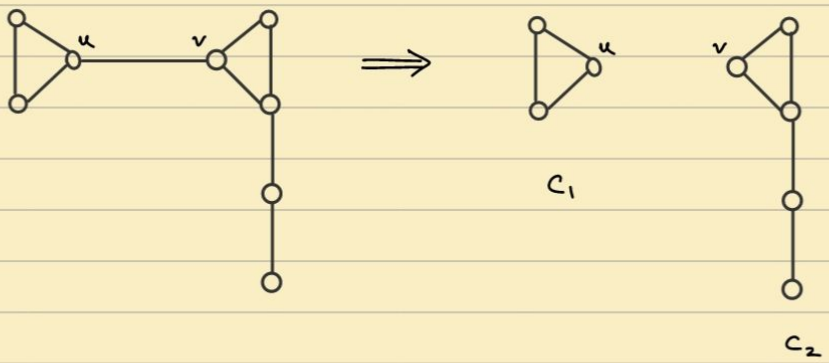


All internal Nodes are APs , Leaves are not.

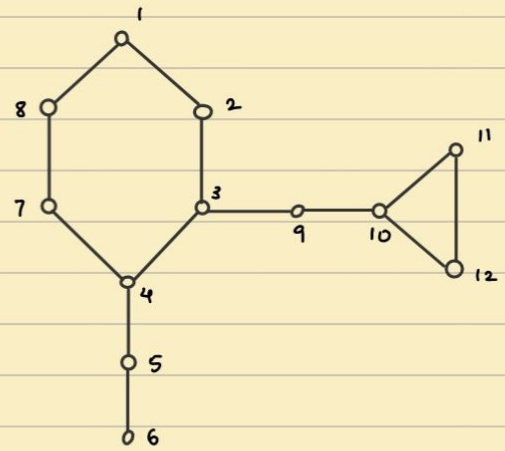
Real life Application of APs : Design of Reliable computer networks

L9

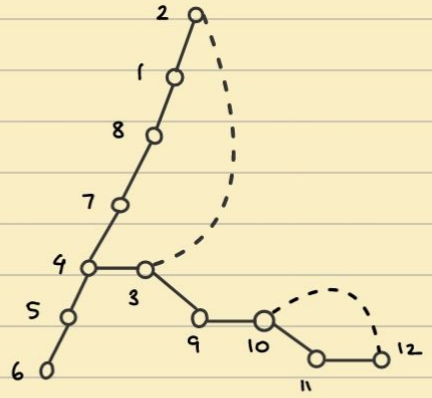
Bridge (Cut Edge) is an edge whose removal disconnects the graph into two connected components.



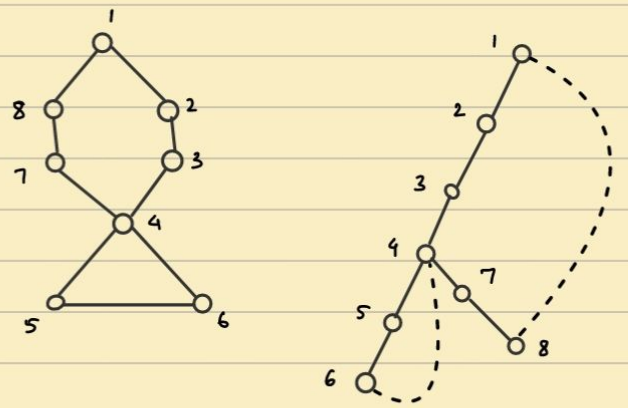
Using DFS :



DFS(2) :



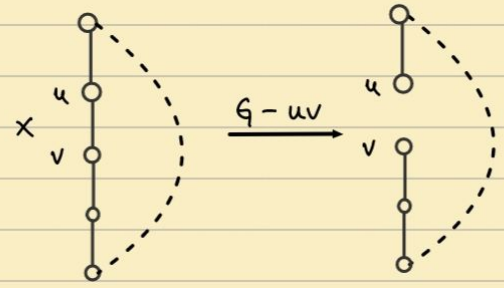
Ex.



No bridges

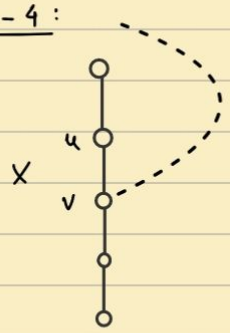
Case - I :

Case - II :

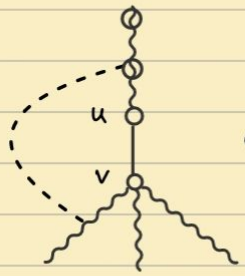


Case - 3:

Case - 4:



'uv' is not a bridge



'uv' is not a bridge

{u,v} is a bridge if all 3 below are true

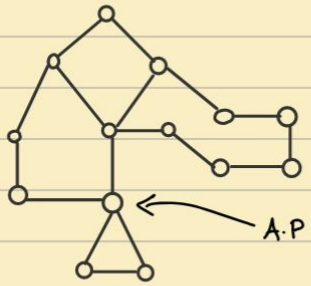
① No backedge from v to anc(u).

② No backedge from des(v) to u.

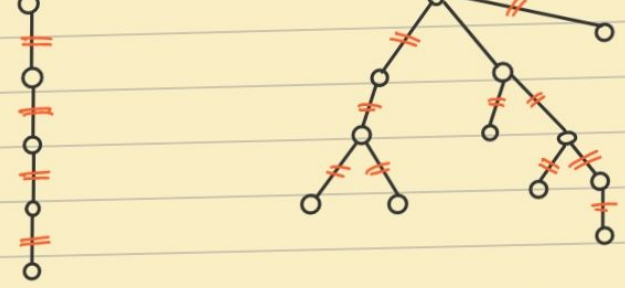
③ No backedge from des(v) to anc(u).

LIO

No. of bridges possible in a graph:



No bridges in this graph



A tree on 'n' vertices has 'n-1' edges and all edges are bridges

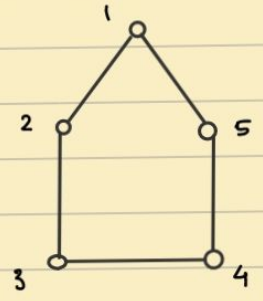
$\Rightarrow$ Max of (n-1) edges

$0 \leq$ no. of bridges in a graph $\leq (n-1)$

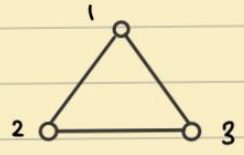
$0 \leq$ no. of articulation points in a graph $\leq (n-2)$

↳ Tight for P_n

- Biconnected Graph / Biconnected Components : (BCC)
  - A Graph is biconnected if it has no APs.
  - A connected component is a biconnected component if no APs.



No APs $\Rightarrow$ Biconnected



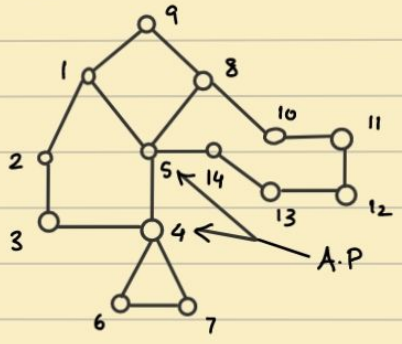
No APs $\Rightarrow$ Biconnected

Note :

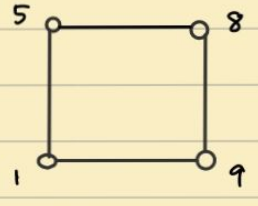
If G has no APs $\Rightarrow$ G is biconnected

If G has APs $\Rightarrow$ G can be decomposed into a collection of biconnected components.

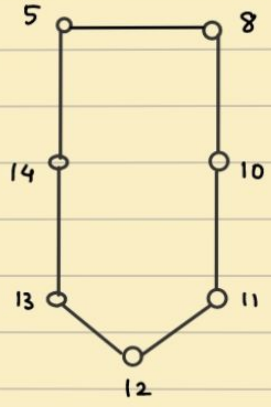
Ex.



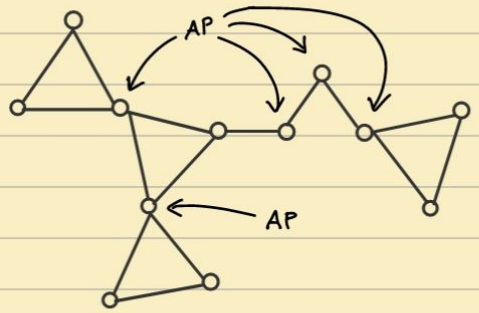
$\Rightarrow$



&

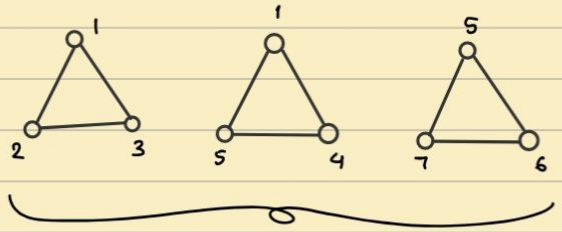


Ex.



G is not biconnected

Decomposition of G into BCC:

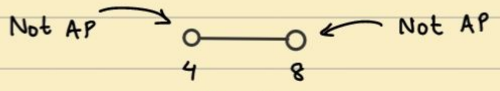


Each one is a Maximal BCC

$v \in AP$ if $G-v$ is disconnected

i.e. $G-v$ has ≥ 2 connected components

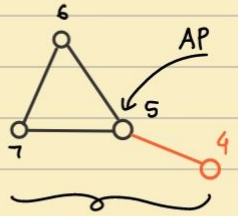
No other vertex along with edges can be added. ∵ The Induced graph won't be biconnected.



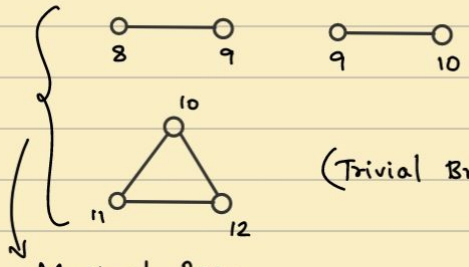
(BCC)

↳ Trivial BCC

i.e.



Not BCC

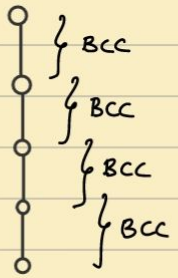


Maximal BCCs

(Trivial Bridges)

∴ Any Bridge is a trivial BCC

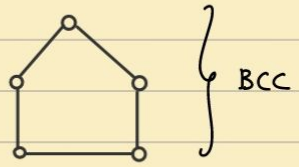
- Path Graph :



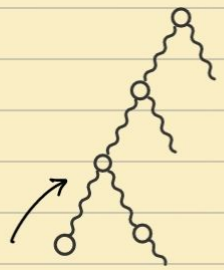
'n-1' BCC's

$1 \leq$ No. of BCCs $\leq (n-1)$

Ex.



- Time complexity of DFS :



Each vertex is visited atmost twice.

(Move down + Backtrack)

Time complexity : $n + m$

n ↳ $|V(G)|$, m ↳ $|E(G)|$

T.C. of DFS = $O(n+m)$

- List all backedges
  - ↳ cycle testing
  - ↳ APs, Bridges, BCCs

} Polynomial time, Poly in I/P size (n,m)
[$O(n+m)$ time]

L11

→ Minimum Spanning Tree Problem: (MST)

I/P : A connected graph G → Undirected
→ Weighted (Edge weighted Graph)

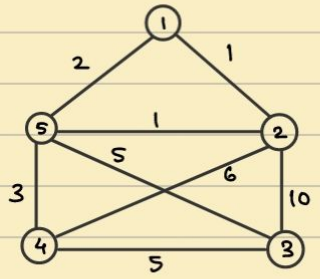
$e_1 \longrightarrow 10$
$e_2 \longrightarrow 100$
$e \longrightarrow w(e)$
↳ Integer

Q : Find a Min. weight spanning tree

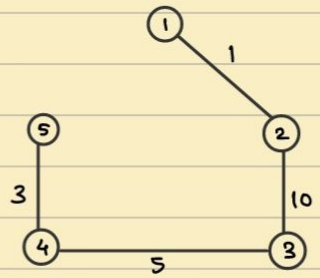
Tree : A connected + Acyclic graph

Spanning : $V(T) = V(G)$

Ex.

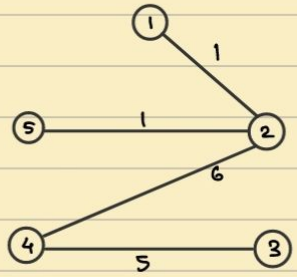


Spanning Trees :



Tree ✓

Weight (T) = 1+10+5+3
= 19



Tree ✓

Weight (T) = 1+1+5+5
= 12
↳ Better

.....

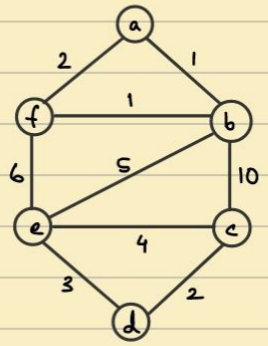
Algorithms used :

Krushkal's Algo
Prim's Algo

} Greedy Algo

↳ Invariant : A property preserved after each iteration

Ex.



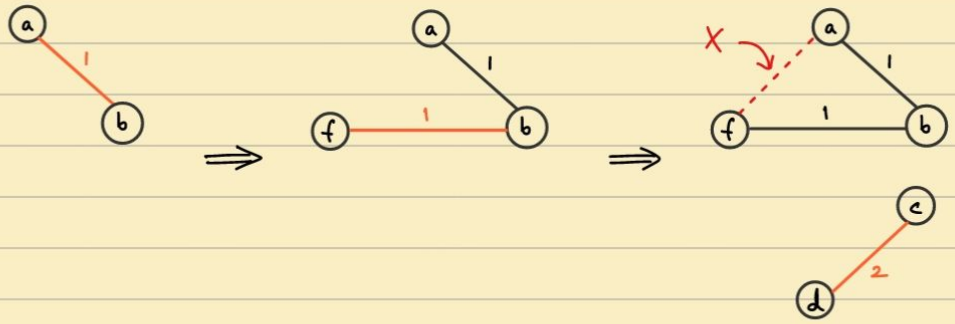
(1) Krushkal's Algo:

① Sort E(G), i.e. Edge set of G, in non-decreasing order, say order is σ.

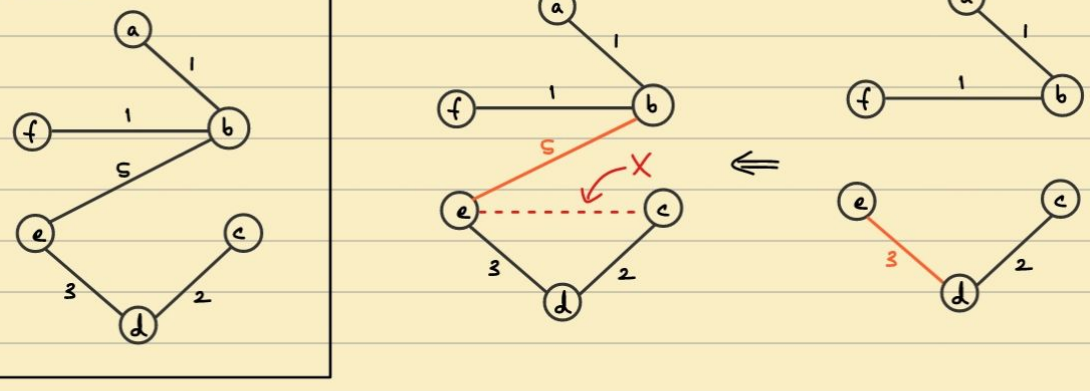
② W.r.t 'σ', include an edge T, if inclusion does not create a tree.

③ Run step ② for 'n-1' iterations (no. of vertices)
[Stop after including 'n-1' edges]

Method:



MST:

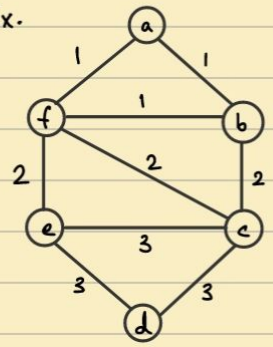


→ Weight(T) = 1 + 1 + 2 + 3 + 5
= 12

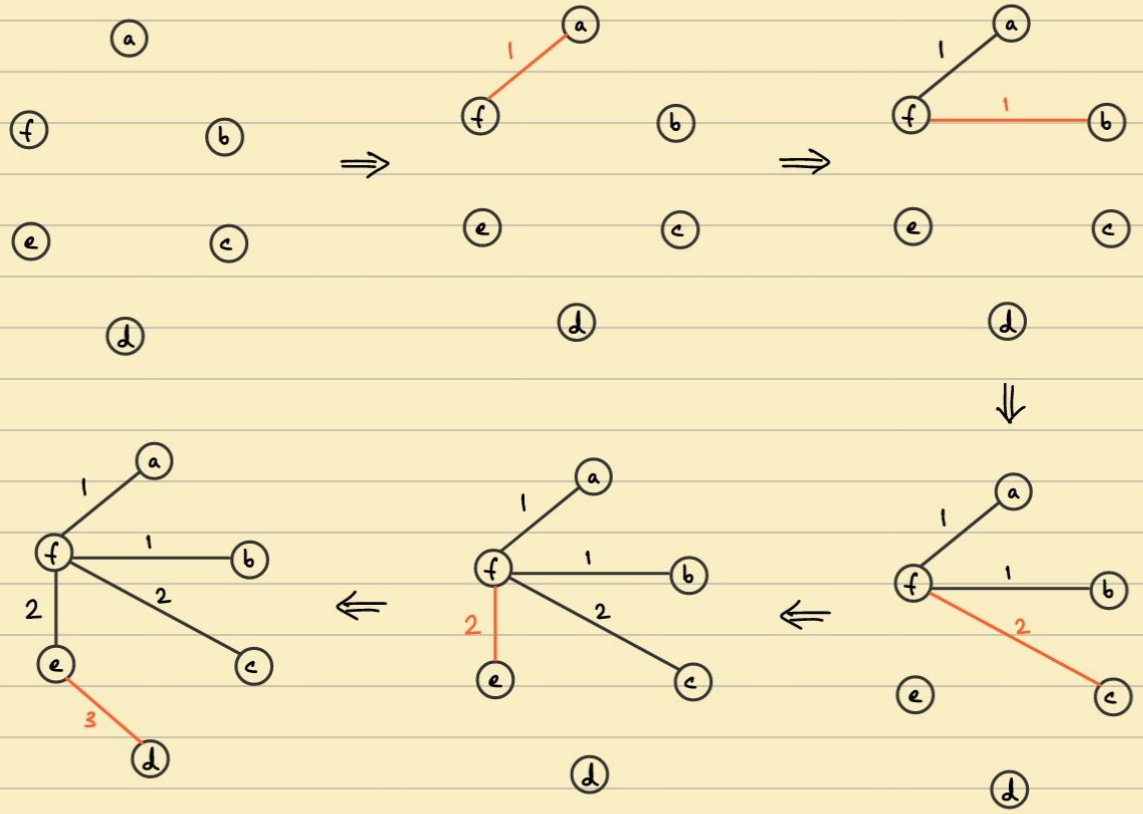
Note:
For a graph, there can exist multiple MSTs.

L12

Ex.



Applying Krushkal's Algo :

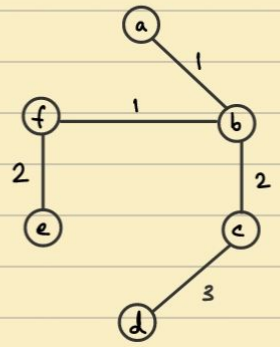


Final MST (T)

$W(T) = 1+1+2+2+3$

$= \underline{\underline{9}}$

There can be other spanning trees such as

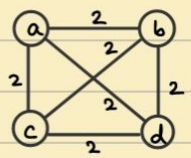


T', $W(T') = 1+1+2+2+3$

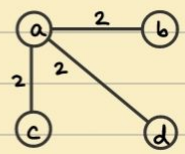
$= \underline{\underline{9}}$

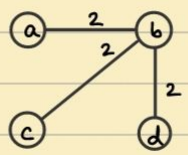
$\therefore$ MST is not unique.

Ex.



MSTs :

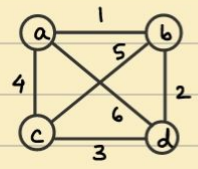




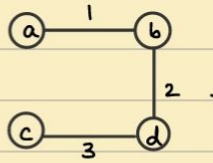
$\Rightarrow$ 16 different MSTs

& $\forall$ MST, $w = 6$

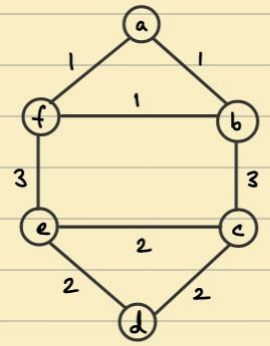
Ex.



Exactly one MST



Ex.



The max. no. of labelled spanning trees on n-vertices is n^{n-2}.

Time complexity of Krushkal's Algorithm:

I/P : n vertices, m edges
($\hookrightarrow |V(G)|$, $\hookrightarrow |E(G)|$)

Step 1 : $O(m \cdot \log(m))$ [Sorting]

Step 2 : $O(m+n)$ [Application of DFS - Cycle Detection]

$= O(n-1+n)$

$= O(n)$ per iteration

$\therefore (n-1) \times O(n)$

Overall T.C $= O(m \log m) + O(n^2)$

$= O(n^2 \cdot \log n^2) + O(n^2)$

$= O(n^2 \log n)$

$\left[O(n^2 \log n^2) = O(2n^2 \log n) = O(n^2 \log n) \right]$

↳ Poly time, Polynomial in I/P size

Invariant : Acyclicity [Not connectedness explicitly]

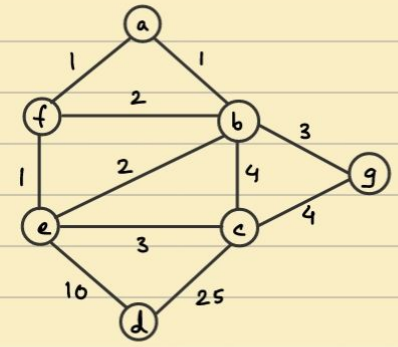
Note : Acyclicity + $(n-1)$ edges $\Rightarrow$ Connectedness

L13

I/P : A connected Edge weighted undirected Graph G

$\mathcal{P}$: Find a Min. weight spanning tree

Ex.



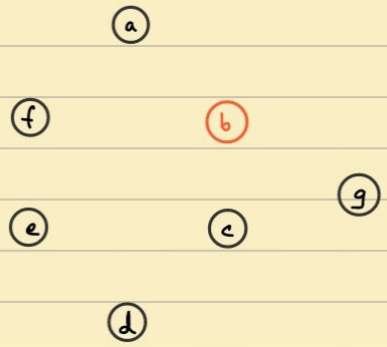
(2) Prim's Algorithm :

① Start with some arbitrary vertex, say b

② Choose a least weight weight edge incident on that vertex, S = {b, a}

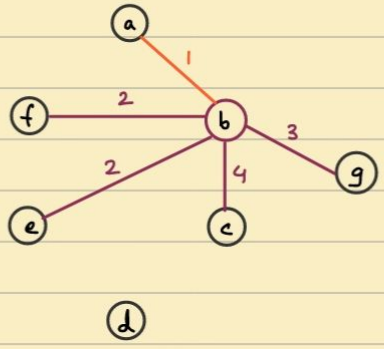
③ Choose a least weight edge incident on S.

④ Stop after 'n-1' edges



S = {b}

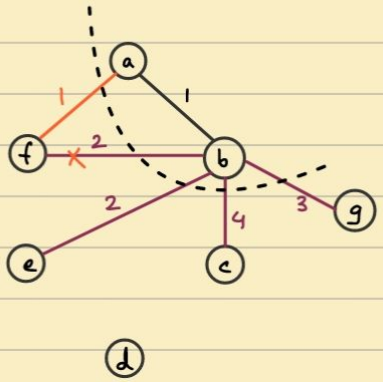
⇒



S = {b, a}

min {1, 2, 2, 4, 3} = 1 (a)

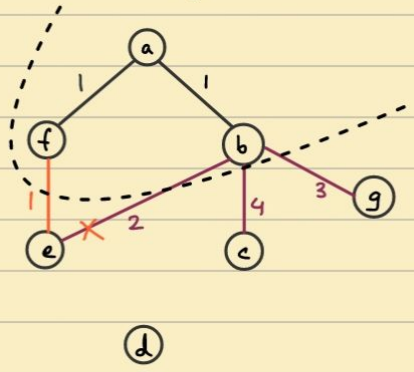
⇒



S = {b, a, f}

min {1, 2, 4, 3} = 1 (f)

⇓

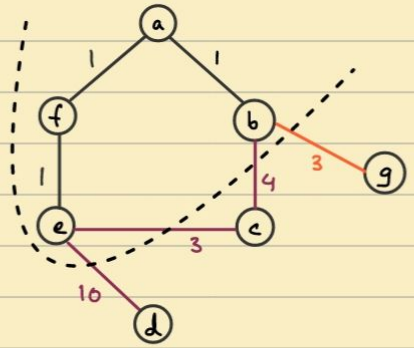


S = {b, a, f, e}

min {1, 4, 3} = 1 (e)

[No connection back to visited vertices, i.e. elements in S]

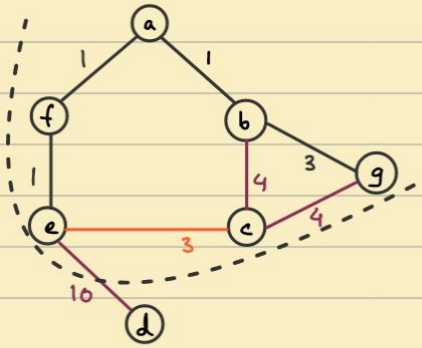
⇐



S = {b, a, f, e, g}

min {3, 3, 4, 10} = 3 (g)

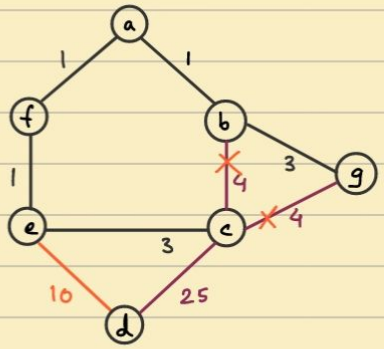
⇐



S = {b, a, f, e, g, c}

min {3, 4, 4, 10} = 3 (c)

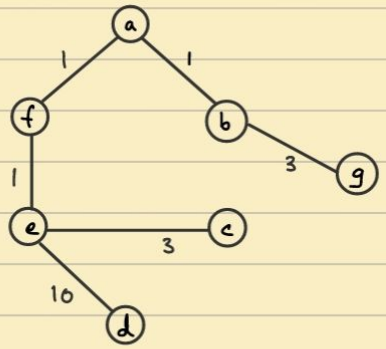
⇓



S = {b, a, f, e, g, c, d}

min {10, 25} = 10 (d)

⇒



Final MST (T)

Weight (T) = 1+1+1+3+3+10 = 19

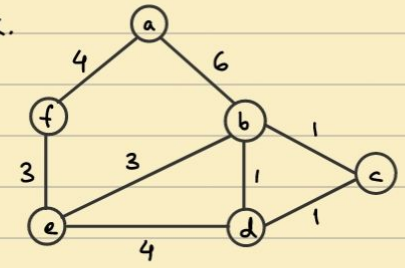
Invariant : (Greedy choice) [Not Acyclicity explicitly]

↳ Choose a least weighted edge while preserving connectivity

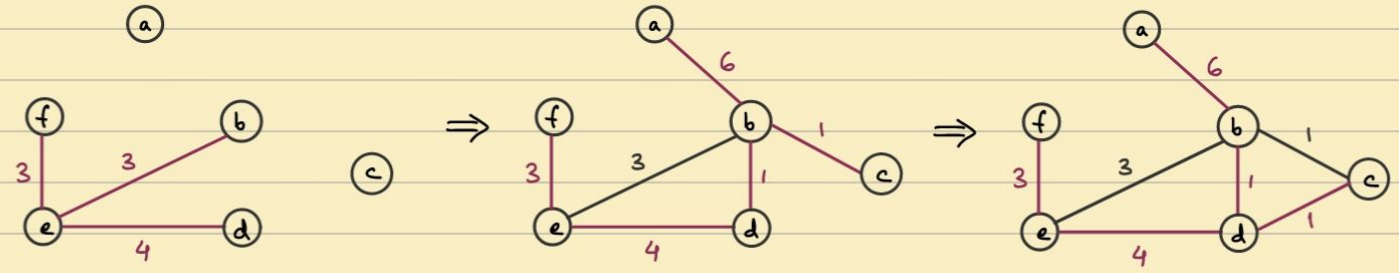
i.e. Connectivity + (n-1) edges ⇒ Acyclicity

L14

Ex.



Start with 'e' ⇒ S = {e}



min(3, 3, 4) = 3 → b

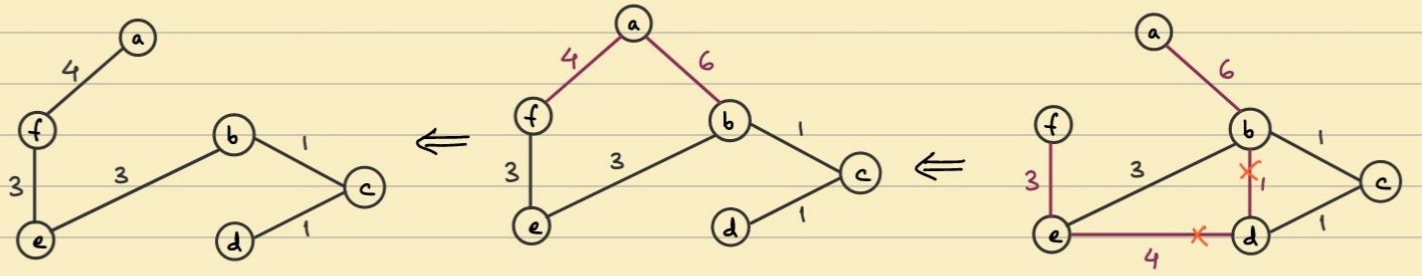
S = {e, b}

min(3, 4, 1, 1, 6) = 1 → c

S = {e, b, c}

min(3, 4, 1, 1, 6) = 1 → d

S = {e, b, c, d}



Final MST (T)

ω(T) = 1+1+3+3+4

= 12

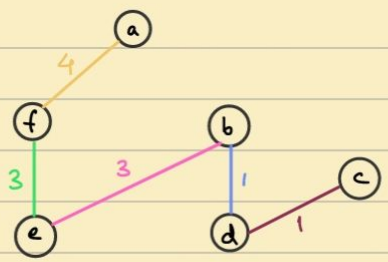
min(4, 6) = 4 → a

S = {e, b, c, d, f, a}

min(3, 6) = 3 → f

S = {e, b, c, d, f}

Same Ex, Krushkals Algo :



① ⇒ ② ⇒ ③ ⇒ ④ ⇒ ⑤

MST → T'

ω(T') = 1+1+3+3+4 = 12

For a given graph,

Both Algos give same minimum weight but the MSTs can be different.

Prims Algo — • Connectedness at each iteration
• Acyclicity is for free with each iteration

Krushkals Algo — • Acyclicity at each iteration
• Connectedness is for free which comes at the end.

Time complexity:

To implement Prims: Min Heap (Data Structure)

Extract-MIN : $O(\log n)$

For n elements : $O(n \log n)$

Build Min-Heap on Edge Set : $O(m)$

Extract Min : $O(\log m)$

For m edges : $O(m \log m)$

LIS

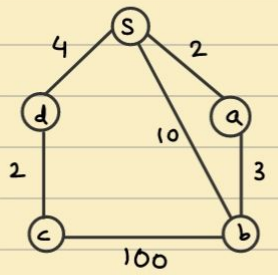
→ Shortest Path problem (SPATH)

I/P: A connected Graph, source vertex 's'.

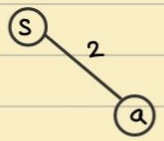
Q: Find SPATH(s, v)

$v \in V(G)$ — Single source spath problem

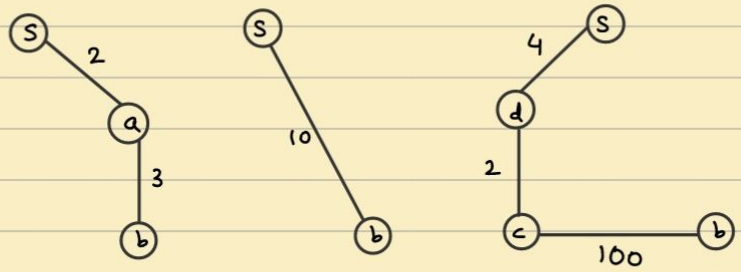
Ex.



Spath(s, a) = 2



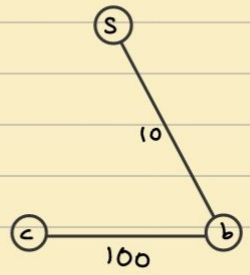
spath(s, b) = 5



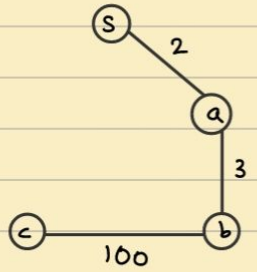
2+3 = 5 10 2+4+100 = 106

spath(s, c) = 6

4 + 2 = 6

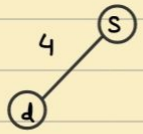


10 + 100 = 110



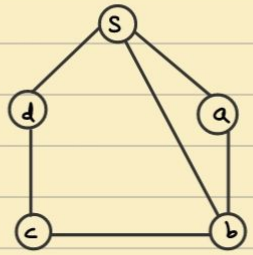
2 + 3 + 100 = 105

span(s, d) = 4

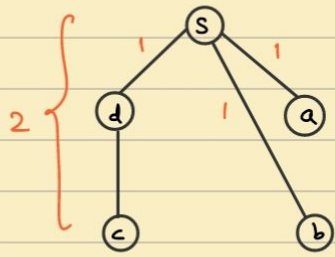


— Revisit BFS:

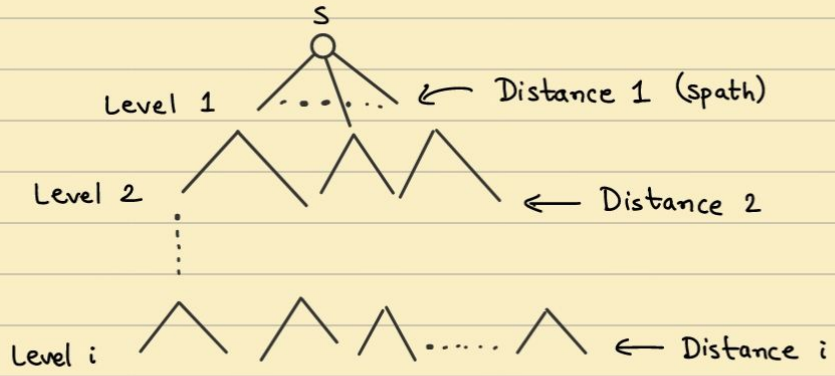
I/P: Undirected Unweighted

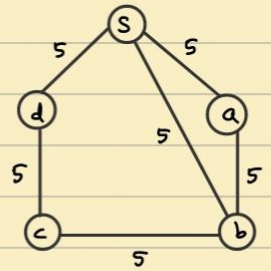


BFS:



Path length → no. of edges
(weight of a path)





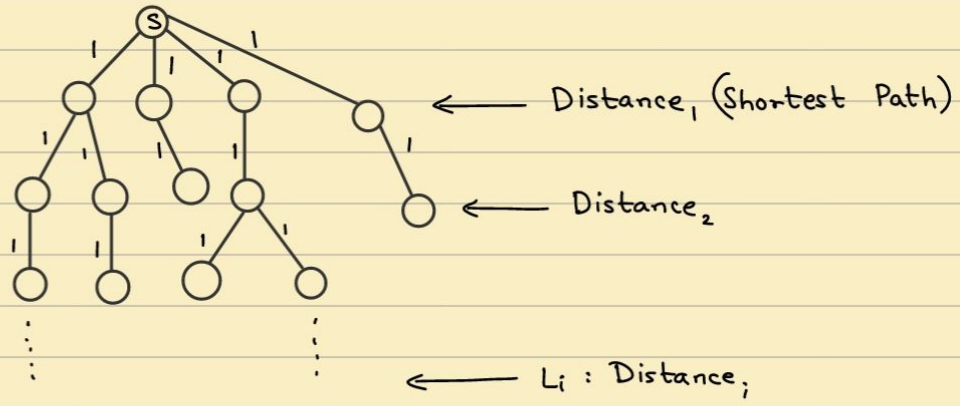
spath(s, v) = weight × spath(s, u)
o/p by BFS tree

i.e. spath(s, c) = 5 × 2 = 10

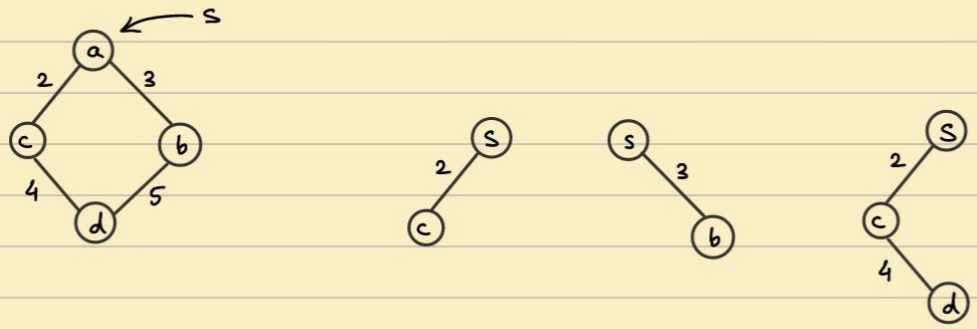
L16

I/p : Undirected Unweighted

Sol^n : BFS



Ex.



Use BFS as a blackbox :

Undirected Unweighted

↓

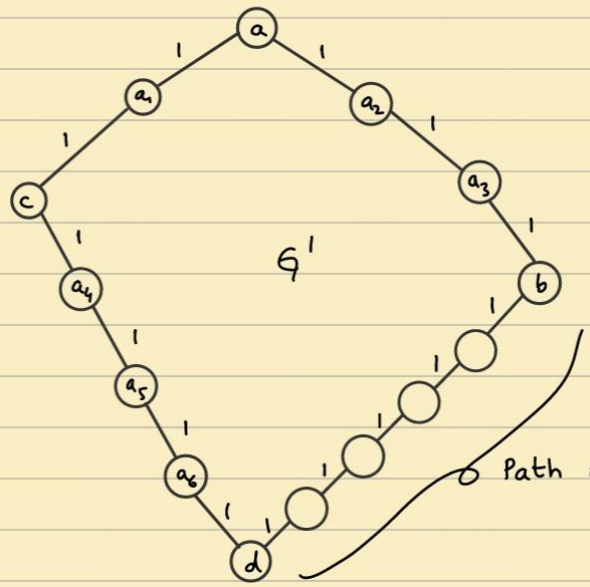
BFS

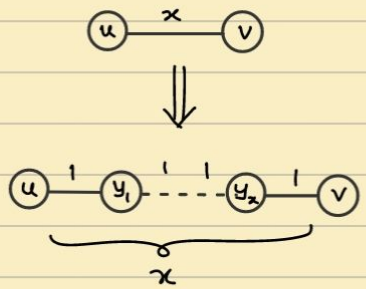
↓

BFS Tree

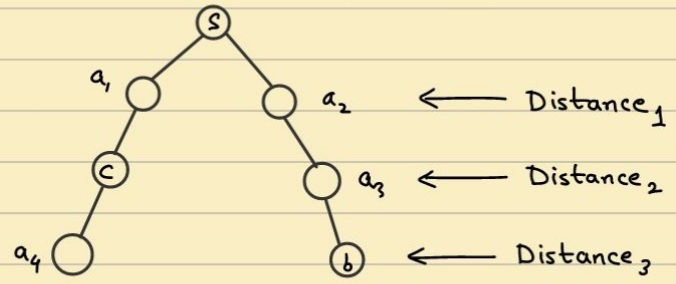
i.e. Weighted Graphs cannot be passed into BFS

Preprocessing :





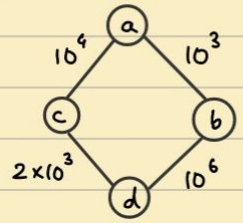
w.r.t G', Run BFS



Time complexity :

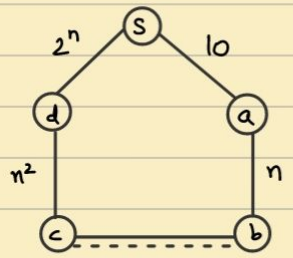
Size of G'

Ex.



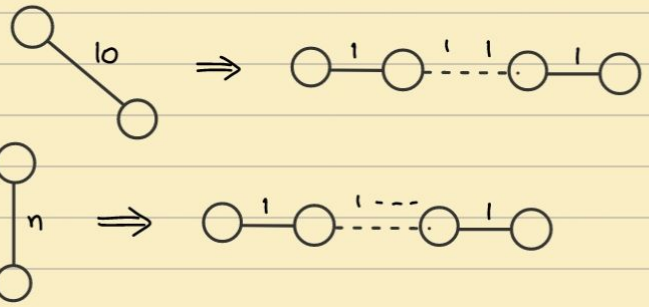
$\Rightarrow G'$: $999 + 10^6 - 1 + 2000 - 1 + 10^4 - 1 + 4$

$V(G')$: Exponential in 'n'.



I/P size (G) : n, m

I/P size (G') : $O(2^n)$



$G \rightarrow$ BFS : $O(n+m)$

$G' \rightarrow$ BFS : $O(2^n)$ [$\because$ m is exponential in n]

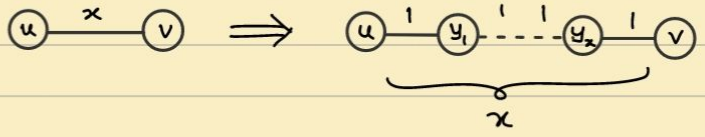
↳ Exponential time

↳ Exp in I/P size

L17 Undirected Unweighted – BFS

Undirected, Weighted with all weights equal – BFS

Undirected, Weighted – BFS [Exp. time for some I/Ps]

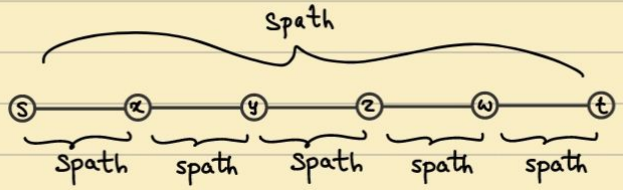


Better Algo : Dijkstra's Algorithm (Greedy Algo)

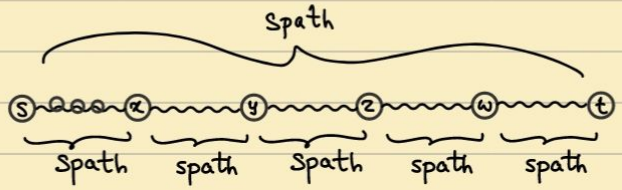
↳ spath Algo

↓

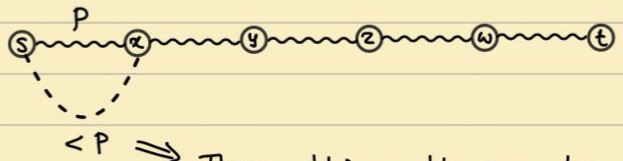
Principle of Optimality – Optimal Substructure Property



Need not be an edge, it can be a path as well.

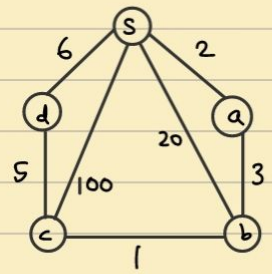


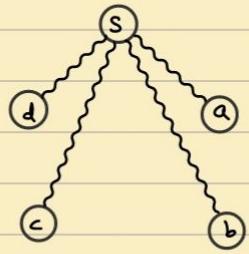
Suppose we have a path, P, from s to x, which is shorter than the spath



$\Rightarrow$ Then, this path can be used to reach t.
This path, from s to t is certainly smaller than s to t via P. Which is contradiction to the premise that s to t is the shortest path.
$\therefore$ i.e. each spath ($s \to x$, $x \to y$, $y \to z$, $z \to w$, $w \to t$) must be a shortest path.
$\Rightarrow$ If s to t is the shortest path, the intermediate paths are also shortest paths.

Principle of Optimality: The optimal solution to the problem lies with optimal solutions to the subproblems.
↳ Optimal Substructure Property





Say, Finding spath ($s \to d$)

Possibility 1

Possibility 2

Ex. Initially: 

Update: 

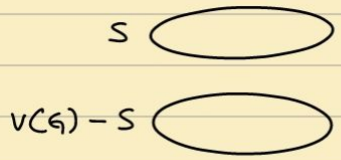
↳ if path length (

) < path length (

)

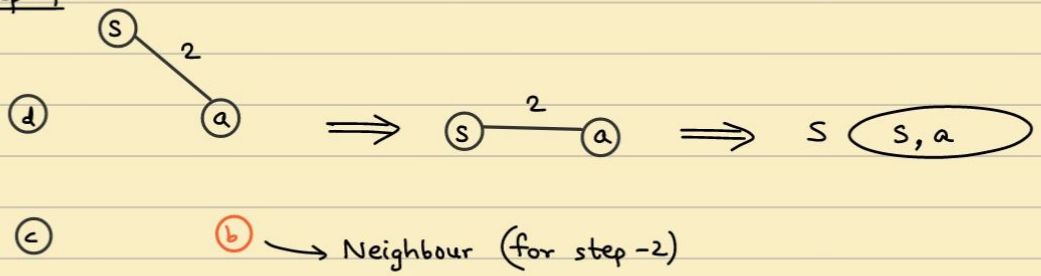
- i.e. At each iteration, when we look for something better than what we are, we update.

- Similar to Prim's Algo
- Maintain two sets

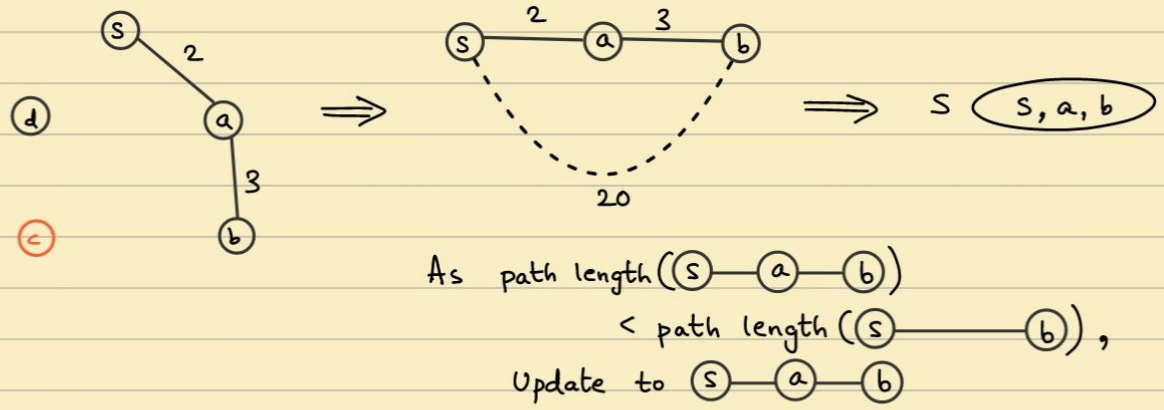


Then, we look at edges incident on 's' and we pick minimum.

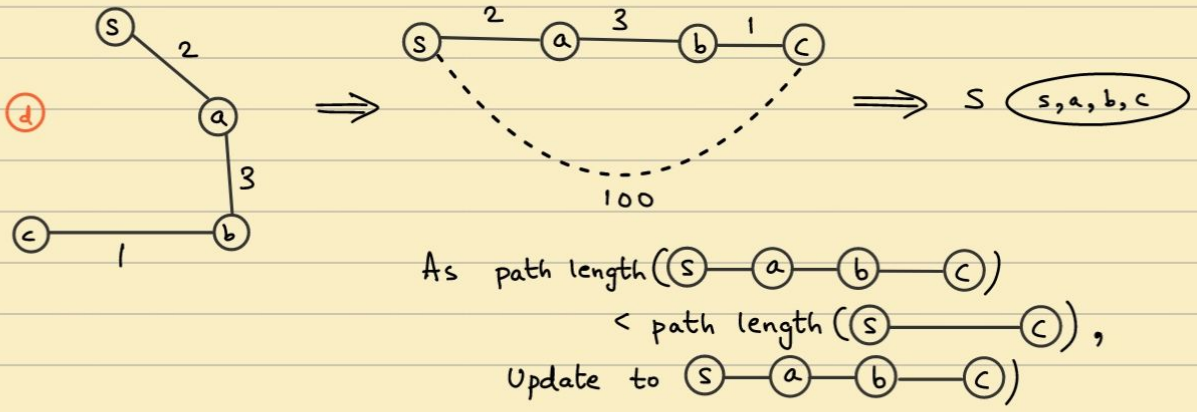
Step - 1:



Step - 2:



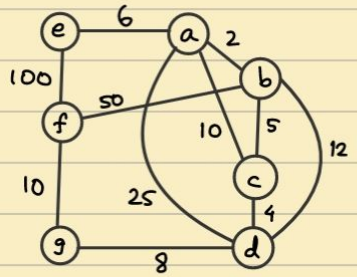
Step - 3:



As we keep unlocking new vertices, we start looking into possibilites of different paths involving them.

Let's trace a non-trivial example:

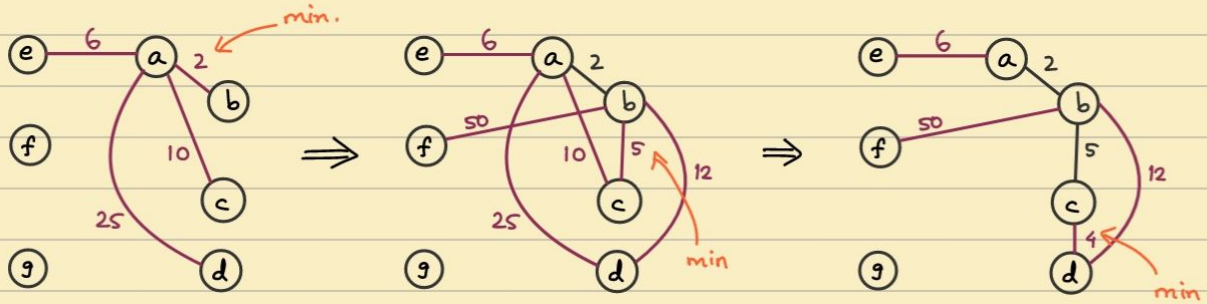
L18



Start with 'a'

i.e. source vertex = a

S {a}



S {a, b}

Explore all neighbours of 'b'

dist(a,c) = 10

wt(a,b) + wt(b,c) < d(a,c)

2 + 5 < 10

Update : d(a,c) = 7

wt(a,b) + wt(b,d) < d(a,d)

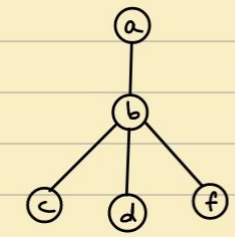
2 + 12 < 25

Update : d(a,d) = 14

wt(a,b) + wt(b,f) < d(a,f)

2 + 50 < ∞

Update : d(a,f) = 52



Step 1 :

dist(a,b) = 2

d(a,c) = 7 a-b-c

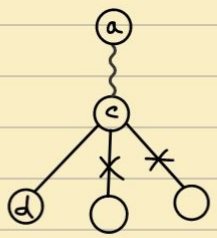
d(a,d) = 14 a-b-d

d(a,f) = 52 a-b-f

As min → 5 :

S {a, b, c}

Explore all neighbours of 'c'



We will not update already included in S.

dist(a,d) = 14

wt(a,c) + wt(c,d) < dist(a,d)

7 4 14

Update : d(a,d) = 11

As min → 4

S {a, b, c, d}

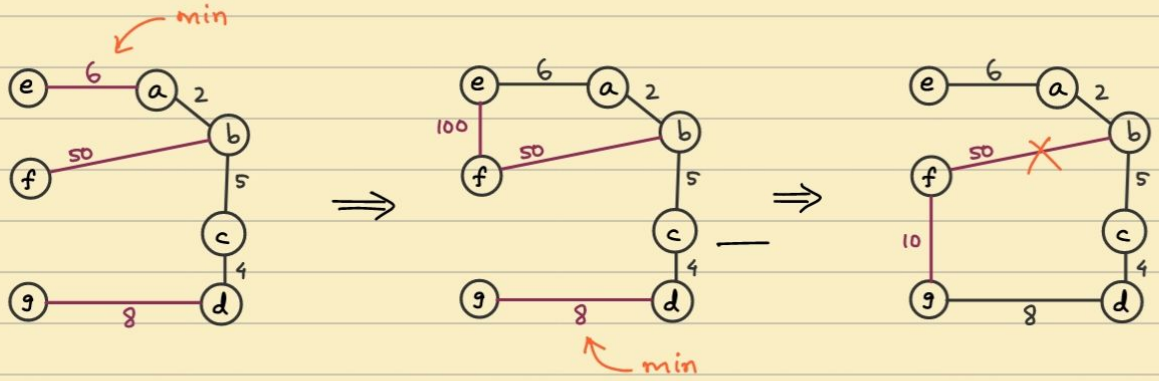
Explore all neighbours of 'd'

(Next page)

Step 2 :

dist(a,d) = 11

a-b-c-d



$wt(a,d) + wt(d,g) < d(a,g)$
11 8 ∞

Update dist(a,g) = 19

$wt(a,e) + wt(e,f) < d(a,f)$
6 100 52

No update

$wt(a,g) + wt(g,f) < d(a,f)$
19 10 52

Update dist(a,f) = 29

∴ Step 3 :
dist(a,g) = 19
a-b-c-d-g

As min → 6,
S (a,b,c,d,e)

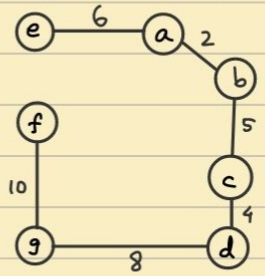
∴ Step 4 :
dist(a,f) = 52
a-b-f
(Coming from step-1)

As min → 8,
S (a,b,c,d,e,g)

∴ Step 5 :
dist(a,f) = 29
a-b-c-d-g-f

Now, we can stop
∵ 'n-1' vertices in set S.

We don't update weights b/w nodes which are already in set S.



End Sem Lab

Q1) I/P : No. of Integers – N
Array [N] – { }
Sum : S
To-do : Find all Subsets S.T. Sum of elements in subset = S, Using DP only
O/P : Yes
Subset of Integers x, y

(OR)

No

Q2) I/P : Number of Vertices (N)
Adjacency Matrix (N×N)
To-do : Check whether Graph is a tree or not
O/P : YES
(OR)
NO

Theory of NP-Completeness

L1

Computational Problem P

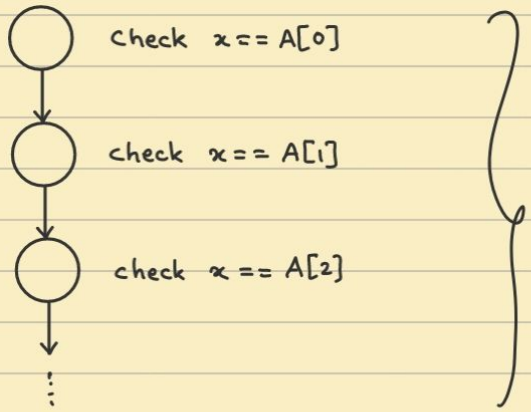
↓

∃ Solution (Solvable)

Known to have Poly-time solution (Poly-time Algo)	As of this date, Not known to have Poly-time Algo	No Poly-time Algo (Every Algo runs in Exp. time)
Poly-time solution : Ex. $O(n^2)$, $O(n^3 \log n)$, $O(n^{10})$ → Poly in 'n' (I/P size)	Ex. $O(2^n)$, $O(n!)$, $O(3^n \log n)$ Current Best : Exponential-time	Ex. $\Omega(2^n)$, $\Omega(3^n)$, $\Omega(n!)$
[EASY]	[DIFFICULT]	[DIFFICULT]

→ Computation :

- Search : A, x



Deterministic Computation – No guessing at each stage

Poly-time Algo ⟺ Deterministic Poly-time Algo

Expo-time Algo ⟺ Deterministic Expo-time Algo

- Not known to have Poly-time Algo (Deterministic Computation)
  ↳ Can we change 'Model of Computation'.

Determinism (No Guessing) | Non-Determinism (Guessing)

i.e. Can we think of designing Non-Deterministic Algorithms

Non-Deterministic Poly-time | Non-Deterministic Expo-time

L2

Non-deterministic Algos

- Clique :

I/P : Graph G , Integer k

Q : Does ∃clique of size 'k'

Clique : A completely connected subgraph of G.

i.e. $S \subseteq V(G)$, $\forall u,v \in S$, $\{u,v\} \in E(G)$

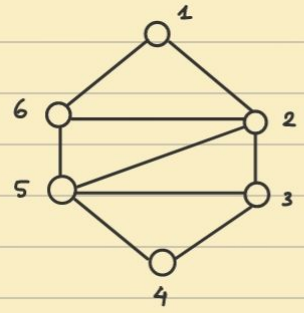
Clique of size $k \Rightarrow |S| = k$

Two versions :

Exact version $\Rightarrow$ |clique| = k

Atleast version $\Rightarrow$ |clique| $\geq k$

Ex.



Exact Version

I/P : G, $k=3$

O/P : Yes

I/P : G, $k=4$

O/P : No

Atleast version

I/P : G, $k=2$

i.e. ?∃|clique| ≥ 2

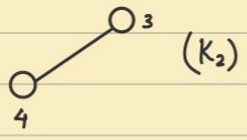
O/P : Yes

|Clicque| = 3 (K_3)

↳ {2, 5, 6}

No K_4 in G , No Clique of size ≥ 4

(K_1)



(K_2)

Does ∃ Poly-time solution ?

Solution :

(1) Try all Subsets of size k

$S_1, S_2, \ldots S_l$, $l = {}^nC_k$ [subsets] $\qquad |V(G)| = n$

(2) Verify S_i is a clique or not

i.e. $\forall u,v \in S_i$, $\{u,v\} \in E(G)$

$|S_i| = k$

no. of pairs in $S_i = {}^{k}C_2 = \theta(k^2)$

no. of edges in $S_i \leq \theta(k^2)$

Check whether S_i is a clique or not $\rightarrow \theta(n^2)$

Overall Time complexity : $\theta({}^{n}C_k \times k^2)$

If $k \rightarrow$ fixed, Algo is deterministic Polytime

i.e. $\theta(n^k)$

If $k \rightarrow$ variable, Algo is non-deterministic Polytime

i.e. $\theta(n^k \cdot k^2)$

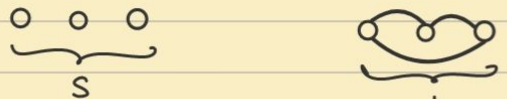
L3

- **Independent Set :**

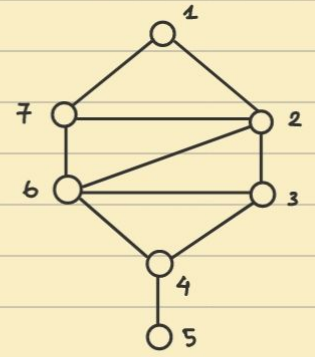
I/P : Graph G, Int K

φ : $\exists$ Independent set of size 'k' $\begin{cases} = k \\ \geq k \end{cases}$

$\hookrightarrow S \subseteq V(G)$, $\forall u,v \in S$, $\{u,v\} \notin E(G)$



$\hookrightarrow$ Independent Set $\quad$ $\hookrightarrow$ Clique



Independent Sets :

$S = \{1,4\}$

$S = \{1,3,5\}$

$S = \{3,5,7\}$

Deterministic Polytime Algo ?

(1) List all Subsets of size k

$S_1, S_2, \ldots\ldots, S_{\ell = {}^{n}C_k}$

(2) Check whether S_i is an independent set.

$\hookrightarrow \forall u,v \in S_i$ whether $A[u,v] = 0$ $\forall u,v \in S_i$

number of pairs $= \theta(k^2)$

Overall Time complexity : $\theta({}^{n}C_k \cdot k^2)$

$\hookrightarrow$ If k: fixed $\Rightarrow$ Deterministic Poly-time Algorithm

If k: Variable $\Rightarrow$ Non-deterministic Poly-time Algo

Can we think of $O(n^2)$ or $O(n^{10})$ Algo for Independent Set

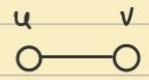
- **Vertex cover** :

I/P : Graph G , K

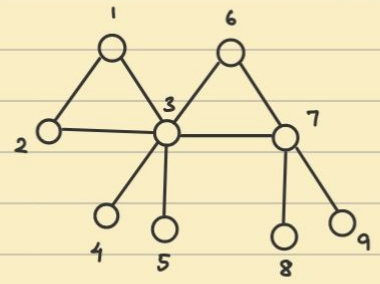
Φ : $\exists$ VC of size K — Exact : $|VC| = K$ / Atleast : $|VC| \leq K$

↳ $S \subseteq V(G)$, $\forall \{u,v\} \in E(G)$

$u \in S$ or $v \in S$



Ex.



$S = \{2,3,7,6\}$

↳ VC

$S = \{2,7,9\}$ ✗

∵ Neither 3 nor 6 is in S.

I/P : G , 3

$|VC| = 3$

$S = \{2,3,7\}$ ✓

Atmost version :

G, 3 Yes

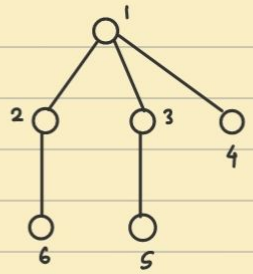
G, 2 No

G, 4 Yes

(i) Try all Subsets of size 'k'

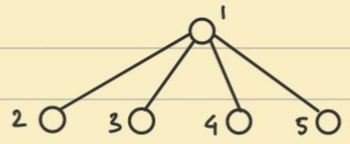
(ii) Check Subset is a VC

Ex.



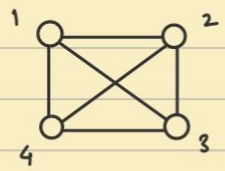
$S = \{1,2,3\}$ VC

$V(G) - S = \{4,5,6\}$ IS



$S = \{1\}$ V.C

$V(G) - S = \{2,3,4,5\}$ I.S



$S = \{1,2,3\}$ V.C

$V(G) - S = \{4\}$ I.S

Claim : If G has a V.C of size k,

Then G has a I.S of size $|V(G)| - k$

Inference :

Checking 'S' is V.C or not

≡ Checking 'V(G) - S' is an I.S or not

Try all subsets of size 'k'

Then, check 'subset of size k' is a V.C,

i.e. check V(G) - S is an I.S

∴ $\theta((n-k)^2)$

Overall Time Complexity : $\theta({}^nC_k \cdot (n-k)^2)$

↳ If k : fixed ⇒ Deterministic Poly-time

k : variable ⇒ Non-deterministic Poly-time

L4

Deterministic Model of Computation : → Hypothetical Machine

↳ i.e. Just to understand the limitations of Deterministic computation.

Read A

cmp (a_1, a_2) for comparing

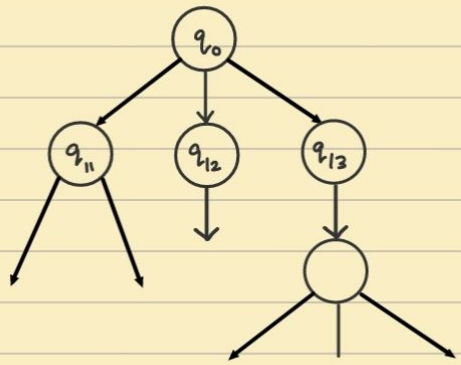
A[i] == x for search

A[n/2] == x for binary search

Computational Graph : Path

No Guessing Involved

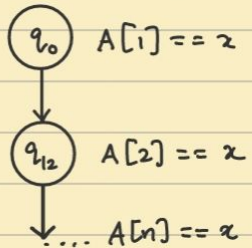
Non-Deterministic Model of Computation :



Computational Graph : Tree

Guessing is Involved (Choosing $q_{11}/q_{12}/q_{13}$)

Deterministic Search :



n iteration

O(n)

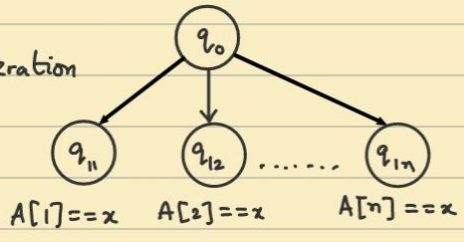
[Poly-time]

Non-deterministic Search :

Just One Iteration

O(1)

[Poly-time]



A Linear time Algorithm becomes a constant time algorithm.
(in deterministic model) (in non-deterministic model)

We can compare deterministic model with another deterministic model only & the same with non-deterministic model.

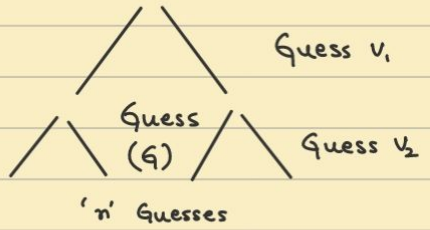
→ Non-Deterministic Clique:

I/P : Graph G, Int k

Q : |Clique| = k

$S \subseteq V(G)$

↳ Whether v_1 is present
v_2 is present
⋮ ⋮ ⋮
v_n is present



① If G is an Yes, then Machine stops & O/P one solution & size k

② Give Minimal Guessing Power to Machine

③ Can we guess the sol^n (clique = k) in just one iteration ?
Yes, but power of guessing should be limited.

For a non-deterministic Algo, Guessing & Verification

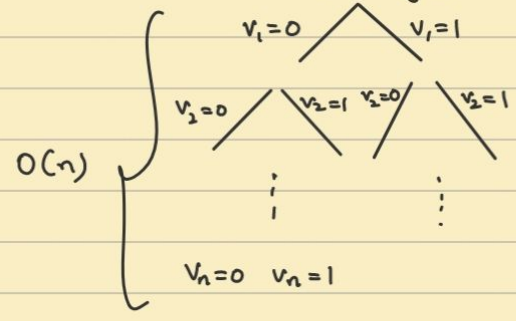
$S \subseteq V(G)$, Verify whether S is a clique ($O(n^2)$ Polytime in Deterministic nature)

i.e. Guessing ⟶ Non-determinism
Verification ⟶ Deterministic in nature

LS

→ Recap:

Clique Problem
- Deterministic – $O(n^k \cdot k^2)$ [Not known to have deterministic Poly-time]
- Non-Deterministic – Guessing, Verification



↳ $\forall u, v \in S$
$\{u,v\} \in E(G)$
$\Rightarrow O(k^2)$, $k=|S|$
$\Rightarrow O(n^2)$
↓
Deterministic Poly-time

Overall Computation : Non-deterministic Clique
Complexity : Non-deterministic Poly-time

Complexity Classes:

P : Deterministic Poly-time Algorithm
Ex. Searching, Sorting, Spath

NP : Non-deterministic Poly-time
Ex. Clique

} Just for Understanding
Actual defintion : Class of Problems

— **I.S Problem:**

$S \subseteq V(G)$

$\forall u,v \in S$, $\{u,v\} \notin E(G)$

Guess 'S'

v_1 v_2 v_3 ..... v_n $\Rightarrow$ n Guesses
0/1 0/1 0/1 0/1

Verify whether our guess 'S' is right
i.e. 'S' is an I.S $\longrightarrow$ $O(n^2)$ Deterministic Poly-time

Non-Deterministic Overall

$\boxed{I.S \in NP}$

— **Vertex Cover:**

$S \subseteq V(G)$, $\forall \{u,v\} \in E(G)$, $u \in S$ (or) $v \in S$

↳ Subset Problem → Guess 'Subset' (Non-Deterministic)
→ Verify 'Subset' (Deterministic)

$\therefore$ Overall : Non-deterministic in nature

$\boxed{\therefore VC \in NP}$

L6

① Are there difficult problems in NP ?
② Can we compare problems in NP ?

NP Hardness:

A problem 'P' is NP-Hard if every problem in NP polynomially reduces to 'P'
(Deterministic Poly-time Reduction)

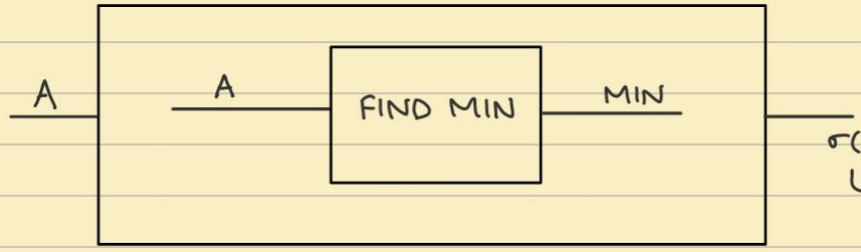
i.e. $\forall_{X \in NP}\ X \leq^{Poly} P \Rightarrow$ P is NP Hard

Inference : ① Every Problem polynomially reduces to P
i.e. P is the hardest Problem among NP
② If we solve 'P', then we can solve 'X'.

③ If we solve 'P' in Deterministic Poly-time, then X has a solution in Deterministic Poly-time.

$P \in$ Class P
$\Rightarrow X \in$ Class P
$\therefore$ Class P = Class NP

- Poly-time Reduction:
  Sorting using FIND-MIN as a blackbox



$\sigma(A)$
↳ Sorting sequence
(Some permutation of A)

Sorting

A[1]

$A - Min_1$	Min	A[2]
$A - Min_1 - Min_2$	Min	A[3]

i.e. O(n) calls to FIND-MIN

Cost of FIND-MIN : O(n)
No. of calls to FIND-MIN : O(n)
$\therefore$ Cost of Sorting : $O(n) \times O(n) = O(n^2)$

Sorting & FIND-MIN are polynomially related.
i.e. Poly-time reduction

- In the context of NP :
  - Can we compare two problems (P_1, P_2) ?
  - Can we solve P_1 using P_2 as a blackbox ?

L7

→ NP Hardness :
(Reducability b/w combinatorial problems)
Ex. Clique, Vertex Cover, Independent Set

- Poly-time Reduction:

$P \leq^{P}$ (Poly-time Reduction) φ

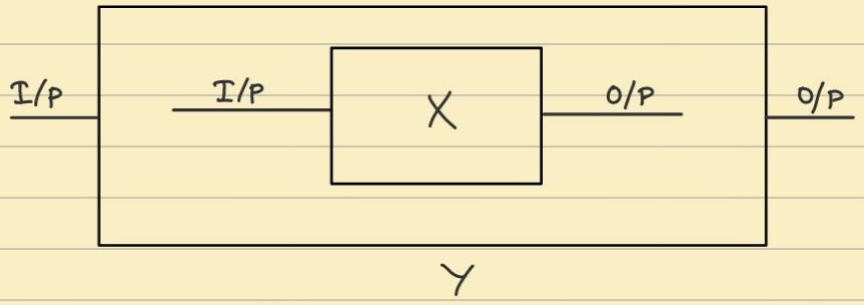
$\Rightarrow$ Each instance of 'P' has a corresponding instance in 'φ'.

$\Rightarrow$ Solution to 'φ' can be used to obtain solution to 'P'

Ex. Solve 'Sorting' using FIND-MIN.

If FIND-MIN is poly $\Rightarrow$ Sorting is Poly.

i.e. If Problem X is easy $\Rightarrow$ Then, Problem Y is easy.



If X is easy $\Rightarrow$ Y is easy [Solution]

Contrapositive: (NP Hard Concept)

If Y is Hard $\Rightarrow$ X is hard [Relative diff. b/w two Probs X, Y]

- Poly-Time reduction:

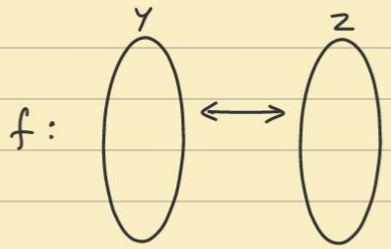
NP - Hardness:

$\forall_{y \in NP}\ Y \leq^{P} Z \implies Z$ is NP-Hard

Z is atleast as hard as every other problem in NP.

Problem Y $\xrightarrow{\text{Poly-time Red}^{n}}$ Problem Z, can be denoted as

$Y \leq^{P} Z$



Find f such that

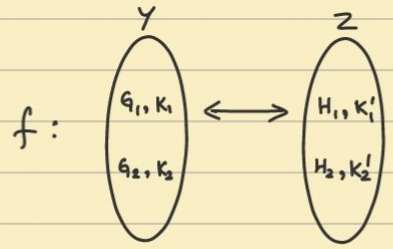
① Map instances of Y to the corresponding instances of Z.

② f is solution preserving reduction.

i.e. YES $\longleftrightarrow$ YES & NO $\longleftrightarrow$ NO

③ 'Y' has a solution iff 'Z' has a solution.

Ex. Graph Problem



- Case Study:

Clique $\leq^P$ Independent Set

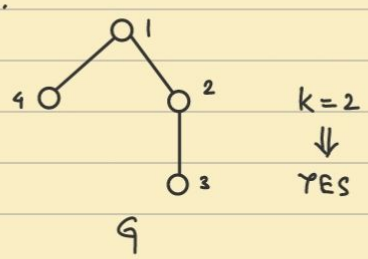
$(G, k) \longleftrightarrow (G', K')$

$G' = G^c$ [Complement of G]

$k' = k$, $k' = f(k)$

[where $f(x) = x$]

Ex.

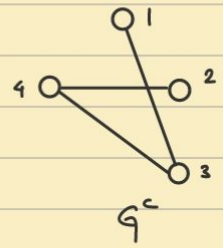


$k = 2$

$\Downarrow$

YES

$\leq^P$



$k' = 2$

G $\qquad\qquad$ G^c

$\{1, 2\} \rightarrow$ Clique

$\{2, 3\}$ ↗

$\{1, 2\} \rightarrow$ I·S

$\{2, 3\}$ ↗

$\therefore$ YES

$k = 3$

$\Downarrow$

NO

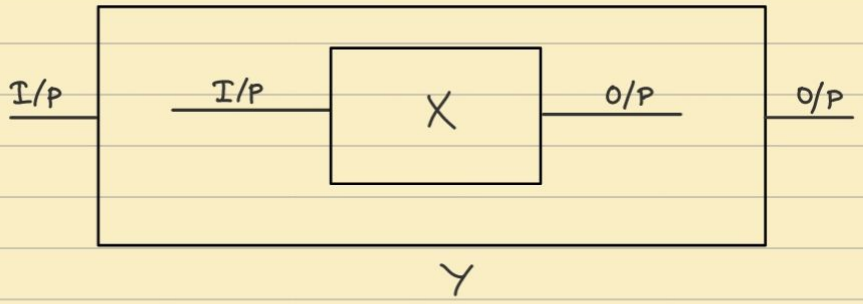
$k' = 3$

$\Downarrow$

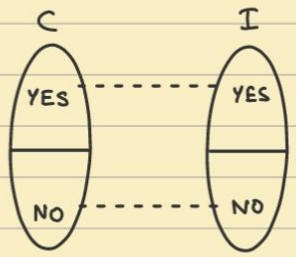
NO

Note : YES instances are mapped to YES instances & NO instances are mapped to NO instances

$\therefore$ Solution preserving Reduction



i.e.

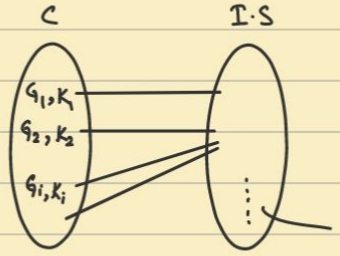


Observation:

① Clique iff I·S , i.e. $(G, K) \in$ (YES of Clique) iff $(G^c, K) \in$ (YES of I·S)

② Solution to I·S can be used to solve Clique.

Insights :



'Many to one' Reduction

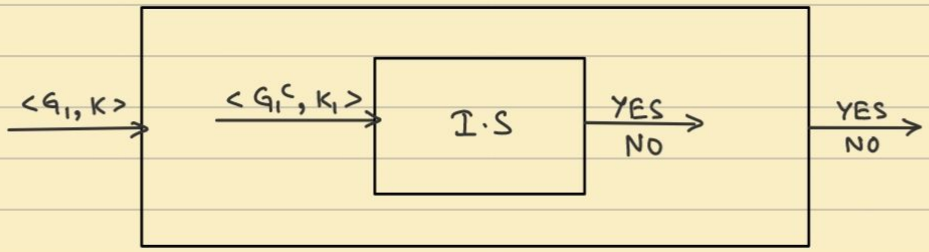
No Pre-Image, i.e. Do not have the corresponding Clique Instance but for each clique instance, ∃a corresponding instance on the I·S.

Clique $\leq^P$ I·S

G, K $\quad$ G', K'

∴ Clique iff I·S, i.e. Sol^n preserving Map

– Consequence :



Clique

If I·S is solvable efficiently, i.e. if I·S ∈ P, then Clique ∈ P, i.e. Clique will have an efficient solution.

∴ We demand Poly-time reduction

i.e. $O(n^2) + O(n^3) = O(n^3)$

↳ Time to obtain independent set.

↳ Reduction cost

Note : This is not the sol^n for Clique but just a framework brought by relating these problems.

Ex. Clique is atleast as hard as 3 SAT (Satisfiability) problem.

In the deterministic Polynomial world, we actually solve a problem & produce a solution. But, in Non-deterministic Polynomial world, we compare the relative hardness and establish a framework.

— Case Study :

$I.S_p \leq^P$ Vertex Cover

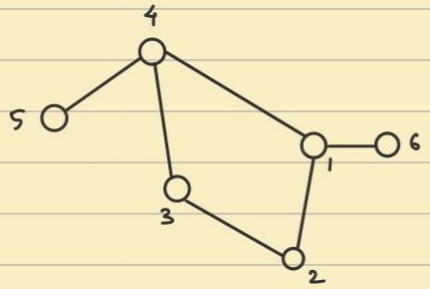
i.e. $\langle G, K \rangle \leq^P \langle G, k' = (n-k) \rangle$

Recall : Vertex Cover

I/P : G, K

Q : $|VC| = k$

↳ $\forall e = \{u, v\}$ in $E(G)$, $u \in VC$ or $v \in VC$



$\langle G, 3 \rangle$

$\{5, 3, 1\} \rightarrow$ I.S ✓

YES

$\langle G, 3 \rangle$

$\{4, 2, 6\} \rightarrow$ VC ✓

YES

$\langle G, 4 \rangle$

$\{5, 3, 1, \ldots\}$

↳ No more elements

$\{4, 2, 6, \ldots\}$

NO

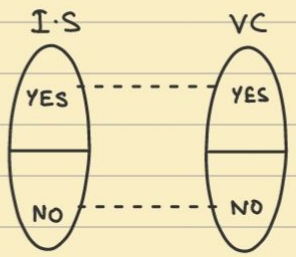
$\langle G, 2 \rangle$

$\{4, 1\} \rightarrow$ Edge $\{2, 3\}$ is uncovered

$\{4, 2\} \rightarrow$ Edge $\{1, 6\}$ is uncovered

NO

i.e.



Solution Preserving Reduction

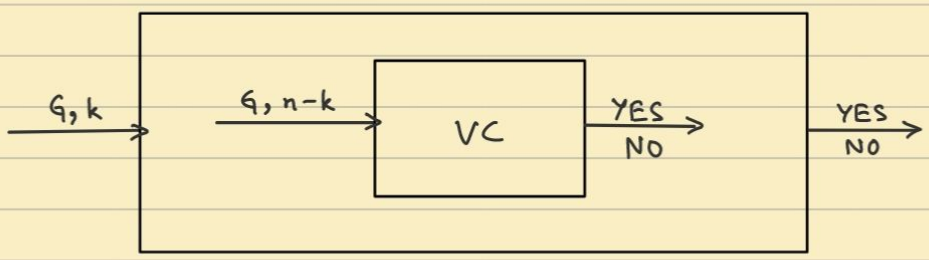
i.e. I.S iff VC

$\langle G, k \rangle$ iff $\langle G, n-k \rangle$

Now,

$I.S \leq^P VC$

$\Rightarrow$ If $VC \in P$, then $I.S \in P$



I.S

L9

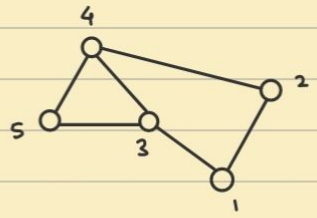
Clique $\leq^P$ VC

$\langle G, k \rangle$ $\quad \langle G = G^c, k' = n-k \rangle$

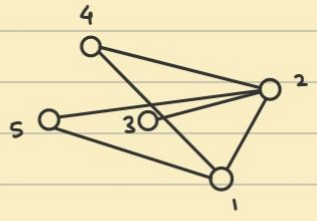
Clique $\leq^P$ I.S

I.S $\leq^P$ VC

Ex.



G



G^c

$G, k = 3$

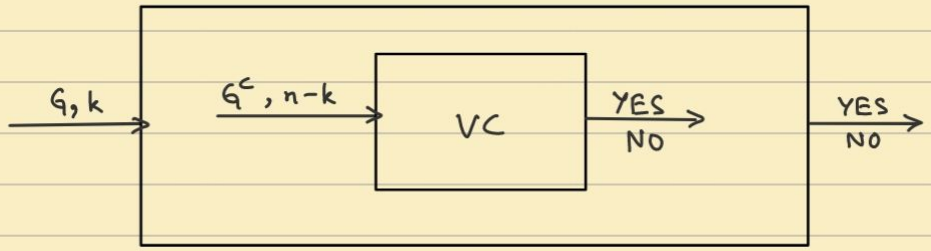
$\{4, 5, 3\}$ YES

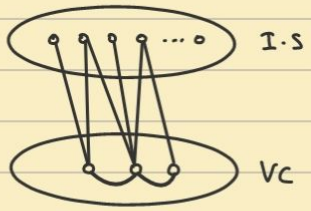
$G^c, k = 2$

$\{1, 2\}$ YES

$\{1,2\} \rightarrow$ VC

$\{3,4,5\} \rightarrow$ I.S





Clique

→ 3 − Satifiability :

I/P : $x_1, x_2, \ldots, x_n$

$C_1 \wedge C_2 \wedge \ldots \wedge C_m$

$C_i = (x_i \vee x_j \vee x_k)$

x_i (or) $\overline{x_i}$

Q : Does ∃ Truth Table values for $x_1, x_2, \ldots x_n$ S.T. $C_1 \wedge C_2 \wedge \ldots C_m$ is evaluated to TRUE.

Ex. I/P : x_1, x_2, x_3, x_4

$F = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_3)$

Q : Find TruthAssignment$(x_1, x_2, \ldots x_4)$ S.T. F is TRUE

Sol : Let $x_1 = 0, x_2 = 1, x_3 = 1$: True

$x_1 = 1, x_2 = 1, x_3 = 0$: True

∴ Generalising,

x_1	x_2	.....	x_n
0/1	0/1		0/1
0	0	.....	0
0	1	.....	1
0	0	.....	1 0
:	:		: :

} List all Binary Strings $\Rightarrow \theta(2^n)$

↓

Trivial Deterministic Algo

Claim : 3-SAT $\leq^P$ Clique

I/P : x_1, x_2, x_3

$(x_1 \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_3) \wedge (\bar{x_2} \vee \bar{x_3})$

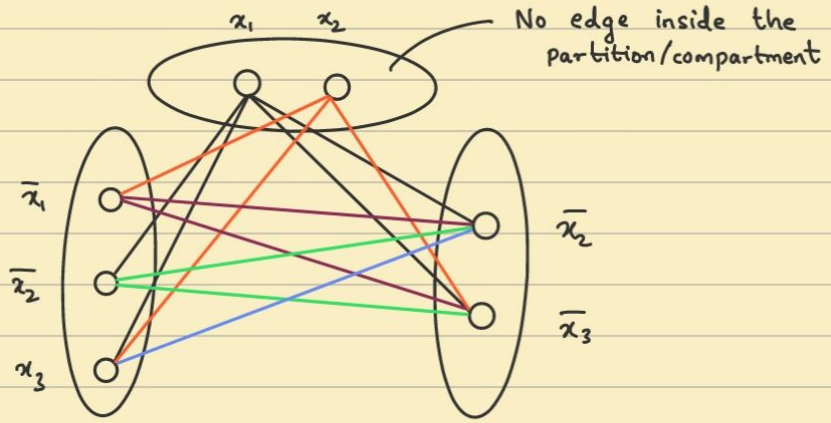
3 − SAT → Indicates $|C_i| \leq 3$

Graph ? k=? $\Rightarrow$ k=m

$x_3 = 1$
$x_2 = 0$
$x_1 = 1$
} YES

$\{x_1, \bar{x}_2, \bar{x}_2\}$ form a clique of size 3

YES



No edge b/w x_i and $\bar{x_i}$

i.e. $\forall x_i, x_j \ \{x_i, x_j\} \in E(G)$

$x_j \neq \bar{x_i}$

L10

$x_3 = 0$
$x_2 = 0$
$x_1 = 1$
} $\equiv$ $\bar{x_3} = 1$, $\bar{x_2} = 1$, $x_1 = 1$ } YES

$\{x_1, \bar{x}_2, \bar{x}_3\}$ forms a Clique , k=3

YES

$\therefore$ YES $\longrightarrow$ YES

i.e. Solution Preserving map

& NO $\longrightarrow$ NO

$x_1 = 1$
$x_2 = 1$
$x_3 = 1$
} NO

False,

No instance

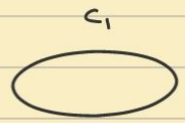
$\{x_1, x_2, x_3\}$ → No Clique , k=3

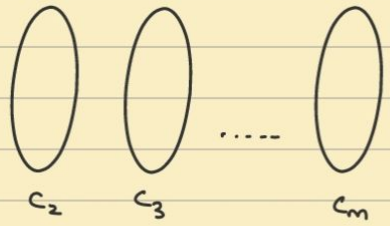
NO

$C_1 \wedge C_2 \wedge \ldots\ldots \wedge C_m \Rightarrow$ Guarantees Clicque of size 'm'

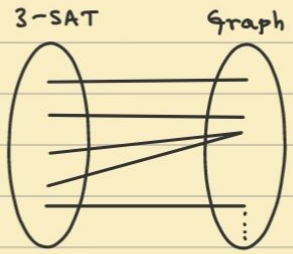
$\therefore$ 3-SAT $\leq^P$ Clique

$x_1, x_2, \ldots, x_m$
$c_1, c_2, \ldots, c_m$ $\leq^P$ $\langle G, k=m \rangle$



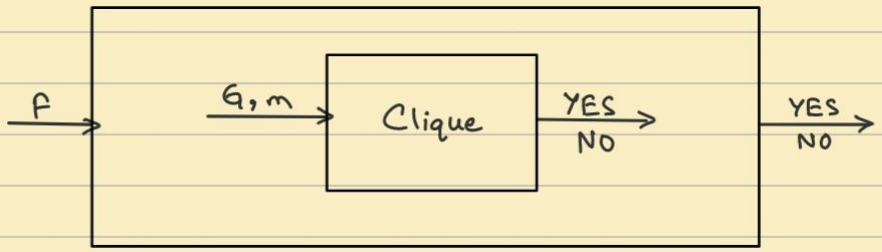


3-SAT is satisfiable iff
$(G, m) \in$ YES of Clique



No edges inside C_i $\forall i$
$\{x_i, x_j\} \in E(G)$ $\forall x_i, x_j$ s.t. $x_j \neq \overline{x_i}$

If Clique $\in$ P, then 3-SAT $\in$ P



3 - SAT

Complexity of 3-SAT = Complexity of Clique

L11

NP - Hardness:

A Problem X is NP-Hard if $\exists$ Poly-time Reduction from every problem in NP to X.

$\forall_{y \in NP}\ y \leq^P X \Rightarrow$ X is NP Hard.

- X is atleast as hard as every other problem in NP.
- X is the hardest Problem in NP.

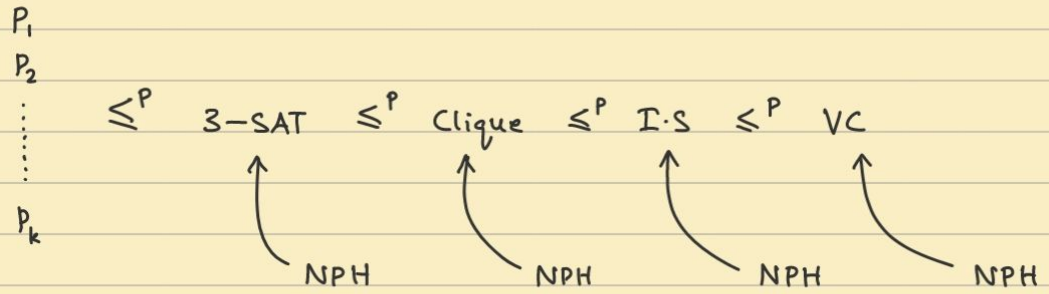
$\Rightarrow$ If the hardest problem $\in$ P, Then every Problem in NP has a deterministic Poly-time Algo (OR) belongs to class P.

$\Rightarrow$ P = NP

- **First 'NP-Hard' Problem:**

'Cook - Levin': Every problem in NP poly reduces to 3-SAT.

$\Rightarrow$ 3-SAT is NP-Hard -①



Every NP Problem $\leq^P$ Clique [Transitive Relation]

– To show problem X NP-Hard :

Choose known NP-Hard Problem Y

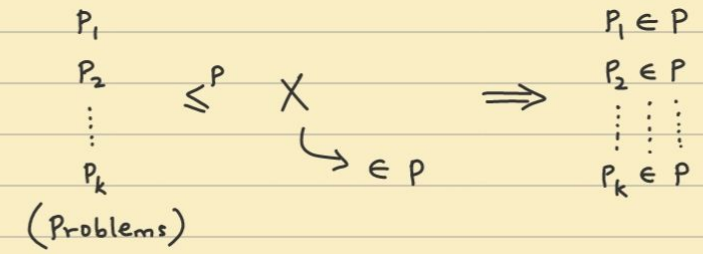
$Y \leq^P X \Rightarrow X$ is NP-Hard.

VC $\leq^P$ H.Cycle $\leq^P$ H.Path

If VC $\in$ P $\Rightarrow$ P = NP

→ NP - Hard :

If any NP-Hard Problem $\in$ class P , then P = NP



→ NP - Complete :

Problem X $\in$ NP-Complete if

① Problem X $\in$ NP (Non-deterministic Algo)

② Problem X is NP Hard.

(i) Choose any NP-Hard problem , say Y.

(ii) $Y \leq^P X$

↳ Poly-time Reduction

Ex. 3-SAT $\in$ NP-Complete

∴ Clique , I.S , VC , H.Cycle , H.Path : NP-Complete

i.e. Problem X $\in$ NP-Complete is in class P $\Rightarrow$ P = NP

———— X ————