0/1 Knapsack Problem

Ex- $n=3$ $m=10$

obj	obj1	obj2	obj3
Profit	35	40	70
weight	5	5	7
P/W	7	8	10

$m = 10-7 = 3$

$X = \langle 0 \quad 0 \quad 1 \rangle$

Profit = 70

Without using greedy, Profit = 75

Hence, Greedy does not always give an optimum solution

Two approaches Top-down approach and bottom-up approach

In Top-down we use recursion, also known as memoization and in bottom-up we use tabulation.

memoization (Demerit - stack memory)

```
Ex: fib(n)
{ if(n==0||n==1)
    return n
  else
  { if (Table[n-1]== NULL)
      Table[n-1] = = fib(n-1);
    if (Table[n-2] = = NULL)
      Table[n-2] = fib(n-2);
    Table[n] = Table[n-1] + Table[n-2];
  }
  return (Table[n]);
}
```

Tabulation (Frequently used)

```
fib( fib[], n)
{
  fib[0] = 0, fib[1] = 1;
for(i = 2, i<=n, i++)
{ fib[i] = fib[i-1] + fib[i-2];
int fib(int n)
{ int f[n+1];
  int i;
  f[0] = 0, f[1] = 1;
for (i=2, i<=n; i++)
{ f[i] = f[i-1] + f[i-2]
}
  return f[n];
}
```

Q) Apply dynamic programming technique to following instance of 0/1 knapsack problem with the capacity m = 5

item	1	2	3	4
weight	2	1	3	2
Profit	12	10	20	15

no. of columns = capacity of knapsack + 1

item ↓ / weight → j	0	1	2	3	4	5
i = 0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	32	32
4	0	10	15	25	30	**37**

means maximum value includes obj

$X = \langle 1 \; 1 \; 0 \; 1 \rangle$

$m = 5$

$m = 5 - 2 = 3$

$m = 3 - 1 = 2$

$m = 2 - 2 = 0$

Q)

Item	weight	Profit
1	2	4
2	1	8
3	3	7
4	5	12
5	4	24

0 1 2 3 4

$\langle 1 \; 1 \; 0 \; 1 \; 0 \rangle$

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	4	4	4	4	4	4	4
2	0	8	8	12	12	12	12	12	12
3	0	8	8	12	15	15	19	19	19
4	0	8	8	12	15	15	20	20	24
5	0	8	8	12	15	15	20	20	**24**

Pseudo code

```
main( )
{ int n, m;
  int P[n];
  int wt[n];
  int K[n+1][m+1];
  for( i=0; i<=n; i++)
  {
    for( w=0, w<=m; w++)
    {
      if ( i==0 || w==0)
        K[i][w] = 0;
      if (wt[i] <= w)
        K[i][w] = max (P[i]+K[i-1][w-wt[i]],
                       K[i-1][w])
      else
        K[i][w] = K[i-1][w];
    }
  }
  return K[n][m];
}
```

items	wt.	Profit
1	2	1
2	3	2
3	4	5
4	5	6

<0 1 0 1>
1 2 3 4

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

Matrix Chain multiplication

A B C
3×2 2×2 2×1

(AB)×C A×(BC)
3×2 2×1 3×2 2×1

12+6 6+4
=18 =10

3×2 (A) 2×1 (B)

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \times \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}\times b_{11} + a_{12}\times b_{21} \\ a_{21}\times b_{11} + a_{22}\times b_{21} \\ a_{31}\times b_{11} + a_{32}\times b_{21} \end{bmatrix}$$

A B C D
3×2

A 3×2 B 2×4 C 4×5 D 5×1

A(BCD) (AB)(CD) (ABC)D

A((BC)D) A(B(CD)) ((AB)C)D (A(BC))D

no. of possible ways to find = $\frac{C_{n-1}^{2(n-1)}}{n}$

0/1 Knapsack Time complexity

Without Dynamic :- $O(2^n)$
Memorization :- $O(N*W)$
Tabulation (Bottom-up) :- $O(N*W)$
N is the no. of items and W is the max capacity

$$M[i,j] = \min_{i \le k < j} \{ M[i,k] + M[k+i, j] + d_{i-1} * d_k * d_j$$

A_1 A_2 A_3 ⇒ Two possibilities

3×4 4×2 2×5 $(A_1 A_2) A_3$ $A_1 (A_2 A_3)$

d_0 d_1 d_2

Three matrices so size of matrix is $n \times n = 3 \times 3$

M	1	2	3
1	0	24	54
2		0	40
3			0

K	1	2	3
1		1	2
2			2
3			

$A_1 \times A_2 = M[1,1] + M[2,2] + d_0 * d_1 * d_2$

$3 \times 4 \quad 4 \times 2 \qquad 3 \times 4 \times 2$

$M[2,3] = M[2,2] + M[3,3] + d_1 * d_2 * d_3$

$= 40 \qquad 4 \times 2 \times 5$

~~$A_1 \times A_3 = M[1,2] + M[2,3]$~~

~~$A_1 \times A_3 = M[1,3] = M[1,1] + M[$~~

$$A_1 \times A_3 = M[1,3] = \min_{1 \le k < 3} \begin{cases} k=1 \\ M[1,1] + M[2,3] + d_0 \times d_1 \times d_3 \\ 0 \qquad 40 + 3 \times 4 \times 5 \\ k_2 = 2 \\ M[1,2] + M[3,3] + d_0 \times d_2 \times d_3 \\ 24 + 0 + 3 \times 2 \times 5 = 54 \end{cases}$$

✓ $(A_1 A_2) A_3$

$$M[i,j] = \min_{i \le k < j} \{ M[i,k] + M[k+i, j] + d_{i-1} * d_k * d_j$$

A_1 A_2 A_3 ⇒ Two possibilities

3×4 4×2 2×5 $(A_1 A_2) A_3$ $A_1 (A_2 A_3)$

d_0 d_1 d_2

Three matrices so size of matrix is ~~$n \times n$~~ $n \times m = 3 \times 3$

M	1	2	3
1	0	24	54
2		0	40
3			0

K	1	2	3
1		1	2
2			2
3			

$A_1 \times A_2 = M[1,1] + M[2,2] + d_0 * d_1 * d_2$

$3 \times 4 \quad 4 \times 2 \qquad 3 \times 4 \times 2$

$M[2,3] = M[2,2] + M[3,3] + d_1 * d_2 * d_3$

$= 40 \qquad 4 \times 2 \times 5$

~~$A_1 \times A_3 = M[1,2] + M[2,3]$~~

~~$A_1 \times A_3 = M[1,3] = M[1,1] + M[$~~

$$A_1 \times A_3 = M[1,3] = \min_{1 \le K < 3} \begin{cases} K=1 \\ M[1,1] + M[2,3] + d_0 \times d_1 \times d_j \\ 0 \qquad 40 + 3 \times 4 \times 5 \\ K_2 = 2 \\ M[1,2] + M[3,3] + d_0 \times d_2 \times d_3 \\ 24 + 0 + 3 \times 2 \times 5 = 54 \end{cases}$$

✓ $(A_1 A_2) A_3$

Q)

	A_1	A_2	A_3	A_4
	5×4	4×6	6×2	2×7
	d_0	d_1	d_2	d_3

M	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

K	1	2	3	4
	1	1	3	
		2	3	
			3	

d_4

$$M[1,3] = \min_{1 \le K < 3} \begin{cases} K=1: M[1,1] + M[2,3] + 5 \times 4 \times 2 \\ \quad = 0 + 48 + 40 = 88 \\ K=2: M[1,2] + M[3,3] + 5 \times 6 \times 2 \end{cases}$$

$$= 120 + 60 = 180$$

$$M[2,4] = \min_{2 \le K < 4} \begin{cases} K=2: M[2,2] + M[3,4] + 4 \times 6 \times 7 \\ \quad = 0 + 84 + 168 = 252 \\ M[2,3] + M[4,4] + 4 \times 2 \times 7 \end{cases}$$

$$= 48 + 0 + 56$$

$$= 104$$

$$M[1,4] = \min_{1 \le K < 4} \begin{cases} K=1: M[1,1] + M[2,4] + 5 \times 4 \times 7 \quad (d_0 \times d_1 \times d_4) \\ \quad 104 + 140 = 244 \\ K=2: M[1,2] + M[3,4] + 5 \times 6 \times 7 \\ \quad = 120 + 84 + 210 = 414 \\ K=3: M[1,3] + M[4,4] + 5 \times 2 \times 7 \end{cases}$$

$$= 88 + 0 + 70$$

$$= 158$$

$(A_1 (A_2 A_3))(A_4)$

Longest Common Subsequence

A sub-sequence is a sequence that appears in the same relative order but contiguous.

Problem statement:- Given two sequences, find the length of longest subsequence present in both of them.

$X = \langle A, B, B, A, B, B \rangle$

$Y = \langle B, A, A, B, A \rangle$

Q) X = A G G T A B

Y = G T T A A B

no. of rows = size of Y + 1

no. of columns = size of X + 1

		A 1	G 2	G 3	T 4	A 5	B 6
	0	0	0	0	0	0	0
G 1	0	0↑	↖1	↖1	←1	←1	←1
T 2	0	0↑	1↑	1↑	↖2	←2	←2
T 3	0	0↑	1↑	1↑	↖2	←2	←2
A 4	0	↖1	1↑	1↑	2↑	↖3	←3
A 5	0	↖1	1↑	1↑	2↑	↖3	←3
A 6	0	↖1	1↑	1↑	2↑	↖3	←3
B 7	0	↖1↑	1↑	1↑	2↑	3↑	↖4

GTAB

q) X = (B, A, B, A, B, A)

Y = (A B A A B A)

		B	A	B	B	A
	0	0	0	0	0	0
A	0	0↑	↖1	←1	←1	↖1
B	0	↖1	1↑	↖2	2	←2
A	0	1↑	↖2	2↑	2↑	↖3
A	0	1↑	↖2	2↑	2↑	↖3
B	0	↖1	2↑	↖3	↖3	3↑
A	0	1↑	↖2	3↑	3↑	↖4

ABBA

		B	A	B	A	B	A
	0	0	0	0	0	0	0
A	0	0↑	↖1	←1	↖1	←1	↖1
B	0	↖1	←1	↖2	←2	↖2	←2
A	0	1↑	↖2	2↑	↖3	←3	↖3
A	0	1↑	↖2	2↑	↖3	←3	↖4
B	0	↖1	2↑	↖3	3↑	↖4	4↑
A	0	1↑	↖2	3↑	↖4	4↑	↖5

ABABA

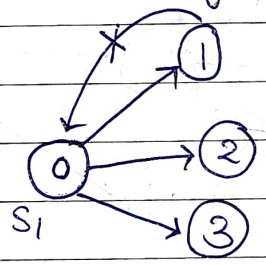
Pseudo code

```
LCS ( x, y, m, n)
{   int L[m+1][n+1];
    int i, j;
    for (i=0; i<=m; i++)
    {
      for (j=0; j<=n; j++)
        if (i==0 || j==0)
          L[i][j]=0
        else if (X[i]==Y[j])
```

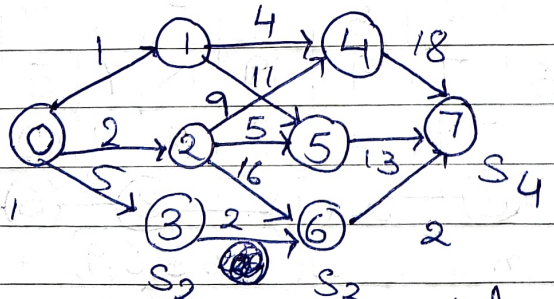
```
        L[i][j] = L[i-1][j-1] + 1;
      else
        L[i][j] = max(L[i-1][j], L[i][j-1]);
    }
}
```

Multistage Graph

Multi-stage graph is a directed weighted graph in which the nodes can be divided into a set of stages such that all edges ~~stages~~ edges are from a stage to next stage only.



At any stage, nodes in the same stage would not be connected to each other.



Using Greedy method, path from 0 to 7 = 23
But, the shortest path cost is 9

we use tabulation method for this,

Cost(3, 4)
↑ stage no, ↘ vertex

V	0	1	2	3	4	5	6	7
cost(s,v)	9	22	18	4	18	13	2	0
d	3	4	5/6	6	7	7	7	7

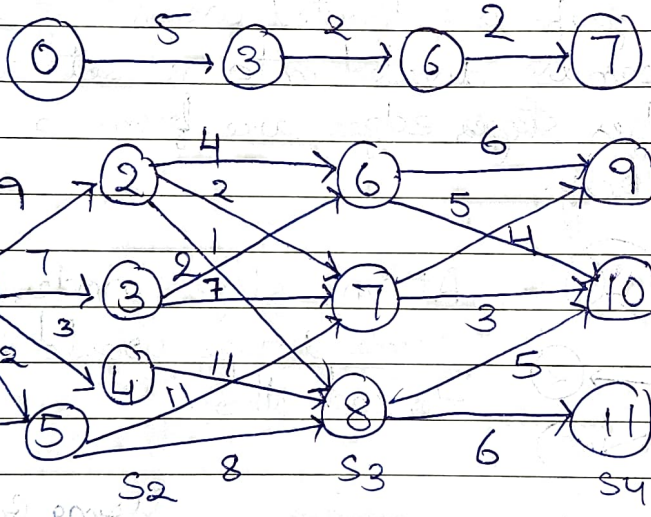
$$Cost(2,1) = \min \begin{cases} w(1,4) + cost(3,4) = 22 \\ w(1,5) + Cost(3,5) = 24 \end{cases}$$

cost.

$$cost(2,2) = \min \begin{cases} w(2,4) + cost(3,4) = 27 \\ w(2,5) + cost(3,5) = 18 \\ w(2,6) + cost(3,6) = 18 \end{cases}$$

$$cost(2,3) = w(2,6) + cost(3,6) = 4$$

$$cost(1,0) = \min \begin{cases} w(0,1) + cost(2,1) = 23 \\ w(0,2) + cost(2,2) = 20 \\ w(0,3) + cost(2,3) = 9 \end{cases}$$



V	0	1	2	3	4	5	6	7	8	9	10	11	12
cost(s,v)		16	7	9	19	15	7	5	7	4	2	5	0
d		2/3	7	6	8	8	10	10	10	12	12	12	12

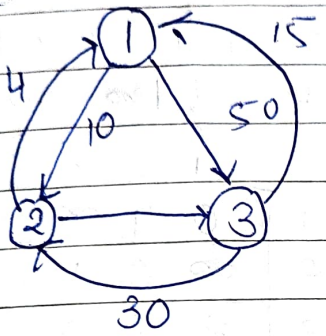
1 —9— 2 —2— 7 —3— 10 —2— 12 —0— D or 1 —7— 3 —2— 6 —5— 10 —2— 12

$$cost(3,6) = \min \begin{cases} w(6,9) + cost(4,9) = 10 \\ w(6,10) + cost(4,10) = 7 \end{cases}$$

$$cost(2,2) = \min \begin{cases} w(2,6) + cost(3,6) = 11 \\ w(2,7) + cost(3,7) = 7 \\ w(2,8) + cost(3,8) = 8 \end{cases}$$

$$cost(1,1) = \min \begin{cases} w(1,2) + cost(2,2) = 16 \\ w(1,3) + cost(2,3) = 16 \\ w(1,4) + cost(2,4) = 22 \\ w(1,5) + cost(2,5) = 17 \end{cases}$$

Floyd Warshall's algorithm



$1 \to 2$
$1 \to 3$
$2 \to 1$
$2 \to 3$
$3 \to 1$
$3 \to 2$

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} 0 & 10 & 50 \\ 4 & 0 & 15 \\ 15 & 30 & 0 \end{bmatrix}$$

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} 0 & 10 & 50 \\ 4 & 0 & 15 \\ 15 & 25 & 0 \end{bmatrix}$$

$A^1[1,2] = A^0[1,1] + A^0[1,2]$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} 0 & 10 & 25 \\ 4 & 0 & 15 \\ 15 & 25 & 0 \end{bmatrix}$$

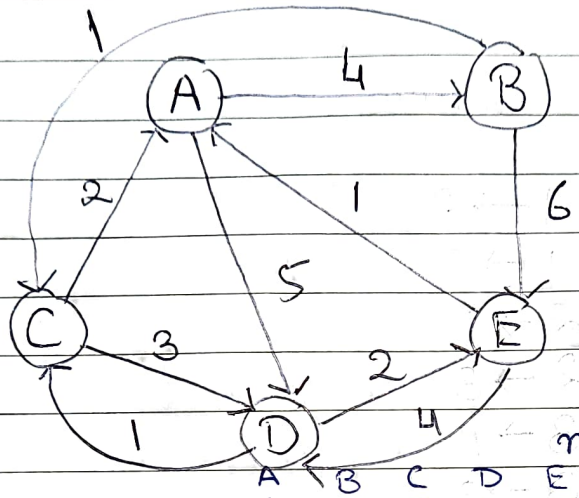
$A^2[1,3] = A^1[1,2] + A^1[2,3] = 10+15 = 25$

$A^2[3,1] = A^1[3,2] + A^1[2,1] = 25+4 = 29$

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 10 & 25 \\ 4 & 0 & 15 \\ 15 & 25 & 0 \end{bmatrix} \end{array}$$

$$A^k[i,j] = \begin{cases} 0 & \text{if } i=j \\ \min\left(A^{k-1}[i,k] + A^{k-1}[k,j],\ A^{k-1}[i,j]\right) \end{cases}$$

Q)



m^0 =

	1	2	3	4	5
A 1	0	4	∞	5	∞
B 2	∞	0	1	∞	6
C 3	2	∞	0	3	∞
D 4	∞	∞	1	0	2
E 5	1	∞	∞	4	0

m^1 =

	A 1	B 2	C 3	D 4	E 5
A 1	0	4	∞	5	∞
B 2	∞	0	1	∞	6
C 3	2	6	0	3	∞
D 4	∞	∞	1	0	2
E 5	1	5	∞	4	0

m^2 =

	A 1	B 2	C 3	D 4	E 5
A 1	0	4	5	5	10
B 2	∞	0	1	∞	6
C 3	2	6	0	3	6
D 4	∞	∞	1	0	2
E 5	1	5	0	4	0

m^3 =

	A 1	B 2	C 3	D 4	E 5
A 1	0	4	5	5	10
B 2	3	0	1	4	6
C 3	2	6	0	3	6
D 4	3	∞	1	0	2
E 5	1	5	0	4	0

m^4 =

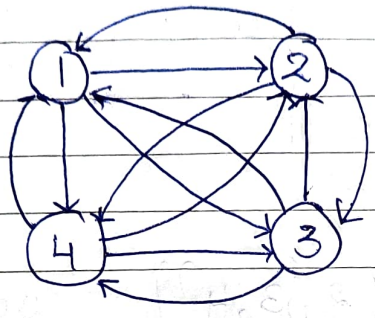
	A 1	B 2	C 3	D 4	E 5
A 1	0	4	5	5	7
B 2	3	0	1	4	6
C 3	2	6	0	3	5
D 4	3	∞	1	0	2
E 5	∞	5	5	4	0

m^5 =

	A 1	B 2	C 3	D 4	E 5
A 1	0	4	5	5	7
B 2	3	0	1	4	6
C 3	2	6	0	3	5
D 4	3	7	1	0	2
E 5	1	5	5	4	0

Travelling Salesman problem

note: Distance from 1 to 2 is not same as 2 to 1 etc.

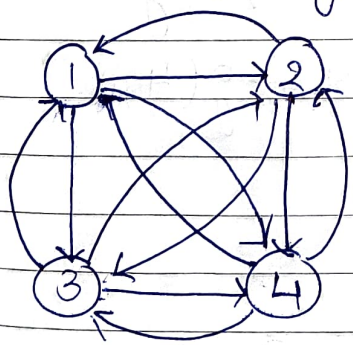


$1 \to 2 \to 3 \to 4 \to 1$
$1 \to 3 \to 2 \to 4 \to 1$
$1 \to 4 \to 2 \to 3 \to 1$
$1 \to 4 \to 3 \to 2 \to 1$
$1 \to 3 \to 4 \to 2 \to 1$
$1 \to 2 \to 4 \to 3 \to 1$

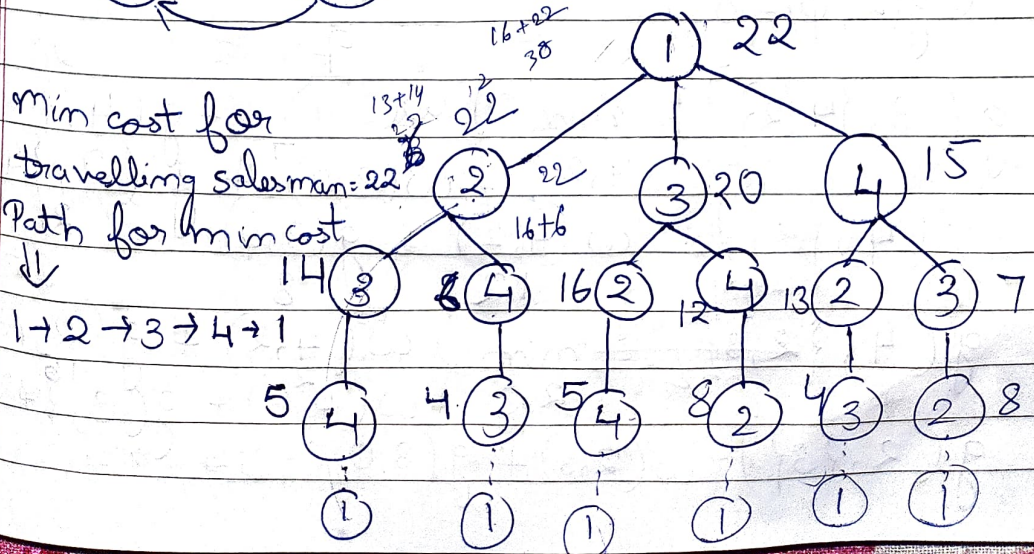
NP hard problem
non-deterministic time complexity polynomial hard

If there are n cities to visit, no. of possibilities
$(n-1)! \Rightarrow n!$
∴ Time complexity = $O(n^n)$

When we use Dynamic programming for this,
Time complexity = $O(2^n)$



	1	2	3	4
1	0	16	11	6
2	8	0	13	16
3	4	7	0	9
4	5	12	2	0



min cost for travelling salesman: 22
Path for min cost
⇓
$1 \to 2 \to 3 \to 4 \to 1$

Formula :-

$$g(i,S) = \min_{j \in S} \{ w(i,j) + g(j, S-j) \}$$

$$g(1,\{2,3,4\})$$

$$= \min_{2,3,4} \begin{cases} j=2 \quad 16 \quad 22 \\ w(1,2) + g(2,\{3,4\}) = 38 \\ j=3 \quad 11 \quad 28 \\ w(1,3) + g(3,\{2,4\}) = 39 \\ j=4 \quad 6 \quad 17 \\ w(1,4) + g(4,\{2,3\}) = \boxed{23} \end{cases}$$

$$g(2,\{3,4\}) = \min_{3,4} \begin{cases} w(2,3) + g(3,\{4\}) \quad (13 \quad 14) = 27 \\ w(2,4) + g(4,\{3\}) \quad (16 \quad 6) = 22 \end{cases}$$

$$g(3,\{4\}) = \min_{4} \{ w(3,4) + g(4,\{\phi\}) \} = 14$$

$$g(4,\phi) = 5$$

$$g(3,\phi) = 4$$

$$g(2,\phi) = 8$$

$$g(3,\{4,2\}) = \min_{4,2} \begin{cases} w(3,4) + g(4,\{2\}) \quad (9 \quad 20) = 29 \\ w(3,2) + g(2,\{4\}) \quad (7 \quad 21) = 28 \end{cases}$$

$$g(4,\{2\}) = w(4,2) + g(2,\phi) \quad (12 \quad 8)$$

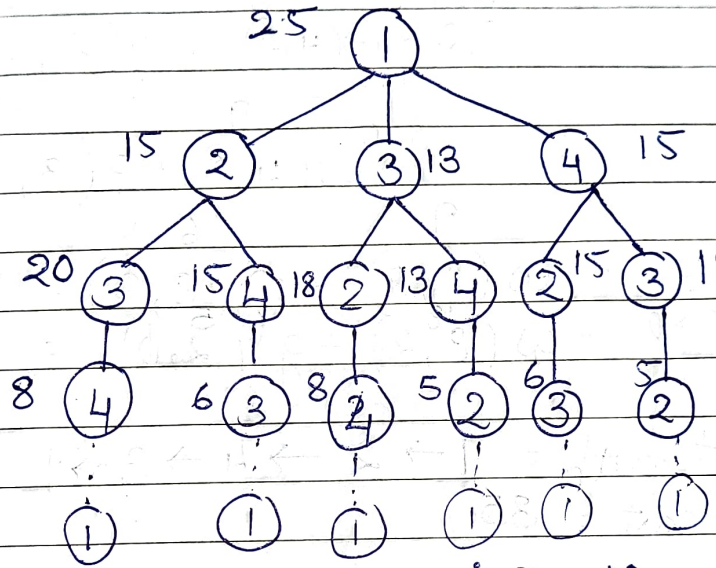
$$g(4,\{2,3\}) = \min_{2,3} \begin{cases} w(4,2) + g(2,\{3\}) \quad (12 \quad 17) = 29 \\ w(4,3) + g(3,\{2\}) \quad (2 \quad 15) = 17 \end{cases}$$

$$g(2,\{3\}) = w(2,3) + g(3,\phi) = 13 + 4$$

22/10/24

q)

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



$g(2,\phi) = 5$

$g(3,\phi) = 6$

$g(4,\phi) = 8$

$$g(1,\{2,3,4\}) = \min_{2,3,4} \begin{cases} j=2 \;\; 10 \;\; 25 \\ \omega(1,2) + g(2,\{3,4\}) = 35 \\ j=3 \;\; 15 \;\; 25 \\ \omega(1,3) + g(3,\{2,4\}) = 40 \\ j=4 \;\; 20 \;\; 23 \\ \omega(1,4) + g(4,\{2,3\}) = 43 \end{cases}$$

$$g(2,\{3,4\}) = \min_{3,4} \begin{cases} j=3 \;\; 9 \;\; 20 \\ \omega(2,3) + g(3,\{4\}) = 29 \\ j=4 \;\; 10 \;\; 15 \\ \omega(2,4) + g(4,\{3\}) = 25 \end{cases}$$

$$g(3,\{4\}) = \omega(3,4) + g(4,\phi) = 20$$

$$g(4,\{3\}) = \omega(4,3) + g(3,\phi) = 15$$

$$g(3,\{2,4\}) = \min_{2,4} \begin{cases} j=2 \;\; 13 \;\; 18 \\ \omega(3,2) + g(2,\{4\}) = 31 \\ j=3 \;\; 12 \;\; 13 \\ \omega(3,4) + g(4,\{2\}) = 25 \end{cases}$$

22/10/24

$g(2,\{4\}) = \overset{10}{w(2,4)} + g(\overset{8}{4},\phi) = 18$

$g(4,\{2\}) = \overset{8}{w(4,2)} + g(\overset{5}{2},\phi) = 13$

$g(4,\{2,3\}) = \min_{2,3} \begin{cases} j=2 & \overset{8}{w(4,2)} + g(\overset{15}{2},\{3\}) = 23 \\ j=3 & \overset{9}{w(4,3)} + g(\overset{18}{3},\{2\}) = 27 \end{cases}$

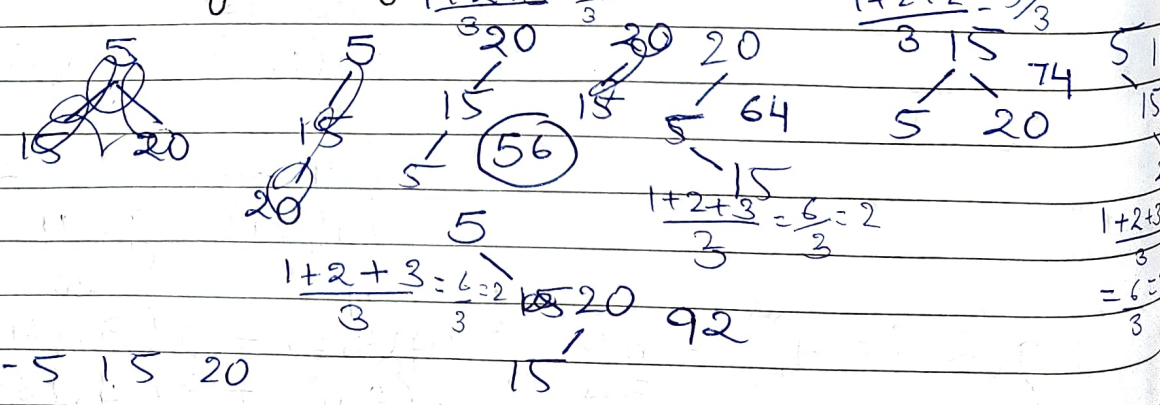
$g(2,\{3\}) = \overset{9}{w(2,3)} + g(\overset{6}{3},\phi) = 15$

$g(3,\{2\}) = \overset{13}{w(3,2)} + g(\overset{5}{2},\phi) = 18$

Shortest path = $1 \to 2 \to 4 \to 3 \to 1$

Path cost = 35

Q) Suppose you are having three keys 5, 15 and 20. How many binary search trees are possible?



Key	5	15	20
Freq	2	10	30

For n no. of keys, possible no. of binary trees are

$$\frac{2n C_n}{n+1}$$

Problem statement: Out of all binary search trees find the optimal BST with given frequency.

Q)

	0	1	2	3	4
Keys		10	20	30	40
Frequency		4	2	6	3

C:

i \ j	0	1	2	3	4
0	0	4	8	20	26
1		0	2	10	16
2			0	6	12
3				0	3
4					0

R:

i \ j	0	1	2	3	4
0		1	1	3	3
1			2	3	3
2				3	3
3					4
4					

$j-i=0$

$C[0,1]$

$j-i=1$

$C[0,2] = 4$ (10)

$C[1,2] = 2$ (20)

$C[2,3] = 6$

$C[3,4] = 3$

$j-i=2$

$$C[0,2] = \min_{K=1,2} \begin{cases} C[0,K-1] + C[K,j] \\ K=1 \quad C[0,0] + C[1,2] \\ K=2 \quad C[0,1] + C[2,2] \end{cases} + w(i,j) = 8$$

(i = 0, j = 2; 0 + 0 = 0; 4 + 0 = 4; w(i,j) = 8)

$$w(i,j) = \sum_{K=i+1}^{j} \text{frequency}(K)$$

$w(0,2) = 4+2 = 6$

$$C[1,3] = \min_{2,3} \begin{cases} C[1,1] + C[2,3] \\ C[1,2] + C[3,3] \end{cases} + w(1,3) = 10$$

(0 + 6; 2 + 0; w(1,3) = 10)

$$C[2,4] = \min_{3,4} \begin{cases} K=3 \; C[2,2] + C[3,4] \\ K=4 \; C[2,3] + C[4,4] \end{cases} + w(2,4)$$

$j - i = 3$

$$C[0,3] = \min_{K=1,2,3} \{$$

$$C[1,4] = \min_{K=2,3,4} \{$$

$$C[0,4] = \min_{K=1,2,3,4} \begin{cases} K=1 \; C[0,0] + C[1,4] \\ K=2 \; C[0,1] + C[2,4] \\ K=3 \; C[0,2] + C[3,4] \\ K=4 \; C[0,3] + C[4,4] \end{cases}$$

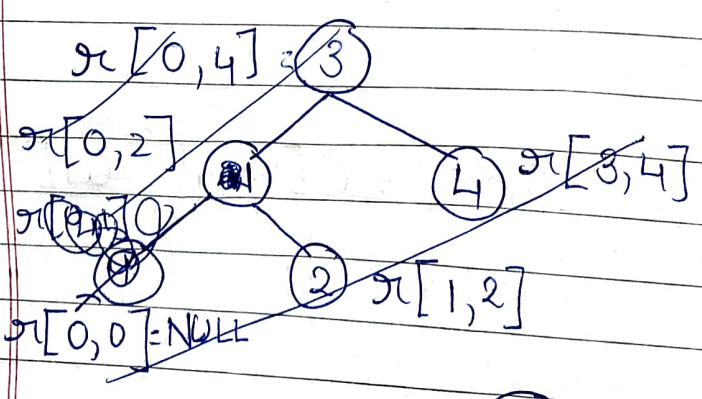
$0 + 16 = 16$

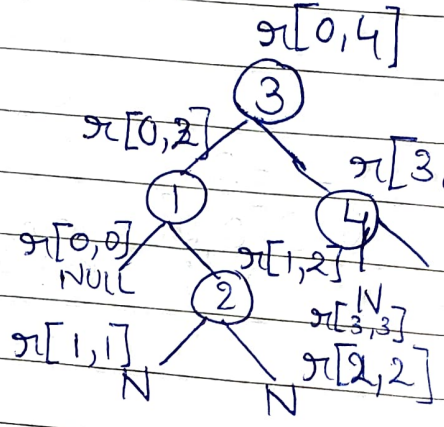
$4 + 12 = 16$

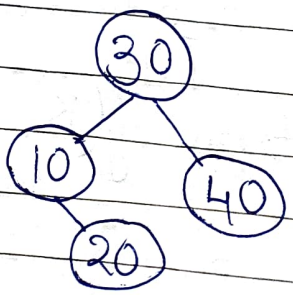
$8 + 3 = 11$

$20 + 0 = 20$

$w[0,4] = 26$







26/10/24

Amortized Analysis

If there are chances that only a few cases have a higher time complexity while others are constant, then only use Amortized analysis.

$$T(n) = \frac{n}{2} + \frac{n}{2} + \frac{n}{2} + \underbrace{1 + 1 + 1 \cdots}_{n-3}$$

$$= \frac{3n}{2} + n - 3$$

$$= O(n)$$

1 insertion $= O(n) = O(1)$

Amortized algorithm analysis is used for algorithms where an ocassional operation is very slow but most of the operations are faster.

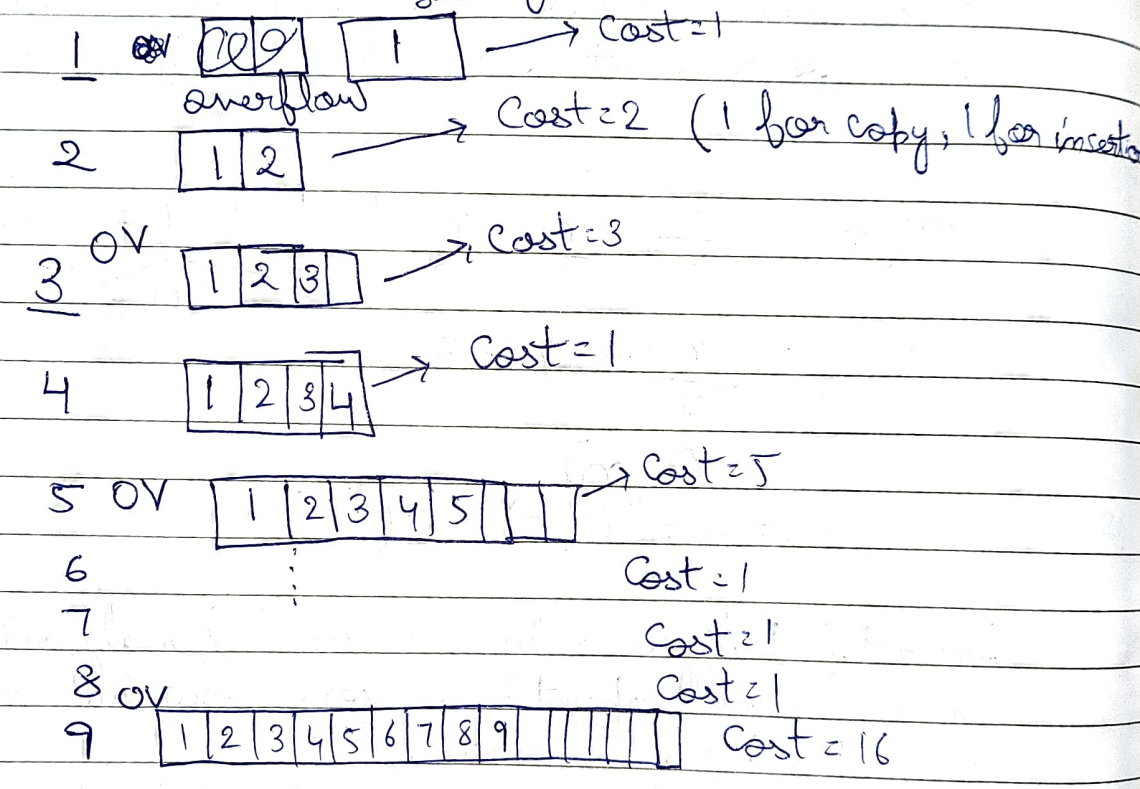
- The example data structures whose operations are analyzed using amortized are Hash table, Disjoint sets and Splay trees.
- It is a technique used to analyze the average case time complexity of algorithms that perform a sequence of operations where some operation maybe more expensive than others.

Three methods are used for amortized analysis:-

i) Aggregate method
ii) Accounting method
iii) Potential method

26/10/24

Q) Dynamic array where size doubles whenever there's an overflow and all the elements are copied to the new array. Find T.C.



items	1	2	3	4	5	6	7	8	9	10	11	...	16	17	...
Size	1	2	4	4	8	8	8	8	16	16	16		16	32	
cost	1	2	3	1	5	1	1	1	9	1	1		1	17	

$(2+3+5+9+17) + (1+1+\dots 1)$

$1+1 \quad 2+1 \quad 4+1 \quad 8+1 \quad 16+1$ — 27

$1+2+4+8+16 + (1+1+\dots 1)$ — 32

$2^0+2^1+2^2+2^3+2^4$

Out of 32 insertions only 5 have more cost,

$2^5 = 32$

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

26/10/24

For n insertions,

$2^0 + 2^1 + 2^2 + 2^3 + 2^4 \ldots 2^{\log n - 1} + (n \times O(1))$

$\Rightarrow \frac{2^{\log_2 n} - 1 + n}{n - 1 + n}$

$\Rightarrow O(2n)$

$\Rightarrow O(n)$ (using amortized analysis & using aggregate method)

$S_n = \frac{a(r^n - 1)}{r - 1}$

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

Accounting method

Generally,

Amortized cost => > Actual cost

Credit = Amortized cost - Actual cost

i

2 | 1 | 2 | - Cost - 2

3 | 1 | 2 | 3 | | Cost - 3

4 | 1 | 2 | 3 | 4 | Cost - 1

5 | 1 | 2 | 3 | 4 | 5 | | | | Cost - 5

For 1 insertion, 3 units of time

For n insertions, $3 \times n$

1 insertion = $O(3) = O(1)$

n insertions = $O(3n) = O(n)$

Amortized cost = 3 unit

Insertion	Am	Ac	Credit
1	3	1	3-1 = 2
2	3	2	(3-2)+2 = 3
3	3	3	0+3 = 3
4	3	1	2+3 = 5
5	3	5	(3-5)+5 = 3
6	3	1	5
7	3	1	7
8	3	1	9
9	3	9	(3-9)+9 = 3
:			
16	3	1	17
17	3	17	(3-17)+17 = 3

Incrementing a Binary number

K-bit binary number

K=2 , 4 numbers

0 → 00
1 → 01
2 → 10
3 → 11

K=4 , 16

0 → 0000
1 → 0001
2 → 0010
3 → 0011
⋮
7 → 0111
8 → 1000
⋮
15 → 1111

Asymptotic time complexity :)

$O(n \times k)$

```
increment(A)
{
  i=0;
  while (i < A.length && A[i] == 1)
  {
    A[i] = 0;
    i++;
  }
  if (i < A.length)
    A[i] = 1
}
```

	A[3]	A[2]	A[1]	A[0]
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

Aggregate method

Cost

A[3] A[2] A[1] A[0]

$1 + 2 + 4 + 8$

$\frac{n}{8} + \frac{n}{4} + \frac{n}{2} + n$

$$T.C. = \sum_{i=0}^{K-1} \frac{n}{2^i} < \sum_{i=0}^{\infty} \frac{n}{2^i}$$

Sum of infinite GP $= a\left(\frac{1}{1-r}\right)$

Amortized T.C. for 1 increment $= \frac{O(n)}{n} = 1$

Accounting method

unit of time

2 units for flipping 0 to 1

0 units of time for flipping 1 to 0

		Amortized cost	Actual cost	Credit
0	0 0 0 0	0	0	0
1	0 0 0 1	2	1	1
2	0 0 1 0	2	2	1
3	0 0 1 1	2	1	2
4	0 1 0 0	2	3	1
5	0 1 0 1	2	1	2
6	0 1 1 0	2	2	2
7	0 1 1 1	2	1	3
8	1 0 0 0	2	4	1

Amortized cost for 8 operations $= 2 \times 8 = 16$

For n operations $= 2 \times n = O(2n) = O(n)$

For 1 operation $= O\left(\frac{n}{n}\right) = O(1)$

Amortized T.C. for 1 operation is constant

30/10/24

Potential method

Item	Size	Actual cost (c)	$\phi(D_i)$	$\phi(D_{i-1})$	$\hat{C}$
1	1	1	1	0	2
2	2	2	2	1	3
3	4	3	2	2	3
4	4	1	4	2	3
5	8	5	2	4	3
6	8	1	4	2	3
7	8	1	6	4	3
8	8	1	8	6	3
9	16	9	2	8	3
16	16	1	16		

ϕ = (2 × no. of items present in the array) - capacity of the

$\hat{C}_i = C_i + \phi(D_i) - \phi(D_{i-1})$

Amortized time complexity for n insertions : $O(3n)$
$\Rightarrow O(n)$

" For 1 insertion : $O(1)$

~~Binary counter (Potential method)~~

$$C_i = \begin{cases} i & \text{, if } i-1 \text{ is power of 2} \\ 1 & \text{, otherwise} \end{cases}$$

$\phi(i)$ = 2 × no. of items present in the array - capacity

30/10/24

Binary counter (Potential method)

ϕ = no. of 1's
4-bit

		Actual cost	$\phi(D_i)$	$\phi(D_{i-1})$	$\hat{C}$
0	0 0 0 0	0	0	0	0
1	0 0 0 1	1	1	0	2
2	0 0 1 0	2	1	1	2
3	0 0 1 1	1	2	1	2
4	0 1 0 0	3	1	2	2
5	0 1 0 1	1	2	1	2
6	0 1 1 0	2	2	2	2
7	0 1 1 1	1	3	2	2
8	1 0 0 0	4	1	3	2

Amortized time : $O(2n)$
for n insertion $\Rightarrow O(n)$
for 1 insertion $\Rightarrow O(1)$

30/10/24

Randomized algorithms

Randomized algorithms are those that use randomness in their computeness to achieve their desired output.

They are different from Deterministic algorithms as Deterministic algos perform a definite procedure to get the same output everytime an input is passed whereas randomized algorithms produce a different output everytime they are executed.

Randomized algorithms are classified based on whether they have time constraints as the random variable or deterministic.

They are designed in the two common forms;

i) Las Vegas
ii) Monte Carlo

i) Las Vegas algorithm is a randomized algorithm that always gives correct output but the time in which it does so is a random variable. Ex - Randomized quicksort result Time taking but assures of finding the

ii) Monte Carlo algorithms - is a randomized algorithm with deterministic run time but some probability for giving the incorrect output.

Example - Randomized algorithm for approximate median

0 1 2 3 4 5

Ex- A = { 5, 1, 3, 2, 4, 6 }

Las Vegas

```
repeat
    K = RGF(0,5)          (RGF:- Random generator value)
    if (A[K] == 2)
        return K
```

Ex - A: { 5, 1, 3, 2, 7, 6 }

```
repeat for 10 :
{
  K = RGF(0,5)
  if (A[K] == 2)
  return K;
}
```

Randomized Quicksort

```
RQS(a, p, q)
{
  if (p == q)
  return (a[p])
  else if (p < q)
  {
  r = RGF(p, q)
  swap( A[p], A[r])
  m = partition (a, p, q)
  RQS ( a, p, m-1);
  RQS (a, m+1, q);
  }
}
```

Time complexity : $\Theta(n \log n)$

$= O(n^2)$ worst case but possibility greatly reduced

05/11/24

COMPLEXITY CLASSES

Based on the time complexity, problems are devided into two classes:-

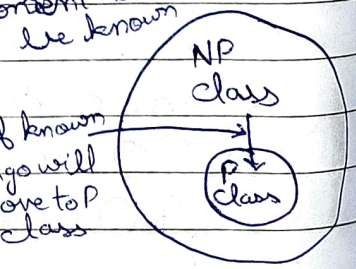
- **P-class problems** - are the problems with polynomial time complexity and can be solved using deterministic algorithms. P-class problems are tractable (possible to implement in theory as well as practical)
- **NP-class problems** - are those problems which can be solved using non-deterministic algorithms in polynomial time complexity.
  NP-class problems are intractable (theoretically possible but practically not possible)

Non-Deterministic search algorithm

Goal:- To find the search element in constant time.

```
NPsearch(A, X)
{
    i = choice(x);   ⇒ O(1)
    if(A[i] == x)
      success( );
    else
      failure( );
}
```

→ not known at the moment but in the future it may be known



P ⊂ NP

P-class	NP-class
• The P-problems are set of problems that can be solved in polynomial time using deterministic algo.	• NP problems are problems that can be solved in non-deterministic Polynomial time.
• P-problems can be solved and verified in polynomial time.	• The solution to NP problems cannot be obtained in polynomial time but if a solution is given it can be verified in polynomial time.

25/11/24

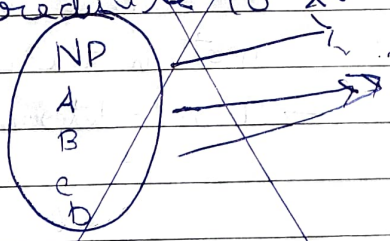
• All P - problems are deterministic in nature. • Eg - Linear search, Binary search, merge sort, matrix chain multiplication	• All NP problems are non-deterministic in nature. • Eg - Travelling salesman, 0/1 Knapsack

$A \xrightarrow{\text{polynomial}} B$

Let A and B are two problems then problem A reduces to problem B if and if there is a way

NP-hard

A problem X is NP-hard, if all the problems in NP are Polynial time reducible to X.



NP complete - A language B is NP complete if it satisfies two conditions. Eg - B is in NP.

Every NP problems is possible

05/11/24
Rework
Ft. Sai

Complexity classes

P-class problems are the problems with polynomial time complexity and they can be solved using deterministic algos.

- P-class problems are tractable (In theory as well as practical, ~~use case~~ implementation can be done)

NP-class problems Non-deterministic algorithms are used for solving these in polynomial time complexity.

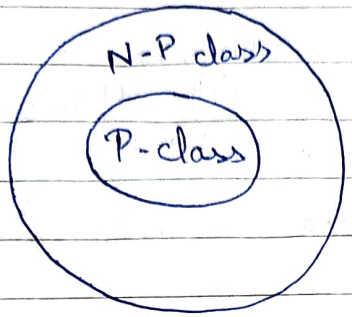
(Most of the statements are deterministic in nature where some of the statements are non-deterministic. We know what it will do, but how it will do, we don't know.)

→ Intractable (Theoretically possible but practically it can't be implemented as not in the present)

→ Non-deterministic algorithms are nothing but a framework

```
NPsearch (A, x)   // A is the array and x is the element
{                    that needs to be searched
  i = choice(x), → O(1)
  if (A[i] == x)   ///choice is a function which will return
    success();        the position of x
  else             // i is the index value where x is
    failure();       stored
}
Time complexity = O(1)
```

Choice function is not known at the moment but it may be known in the future



$P \subset NP$

million dollar baby - Tommy Richman

Is $P = NP$?

Solutions of NP class problems can't be solved in polynomial time complexity but can be checked in polynomial time complexity.

When an NP-class problem can be solved in polynomial time and is known then it is moved to P-class.

Reduction:-

A — Soln. of A is not known

B — Soln. of B is known

I_1 (reduction) I_2 (Reduction must be in Polynomial time)

(I_1 can be solved with the help of B)

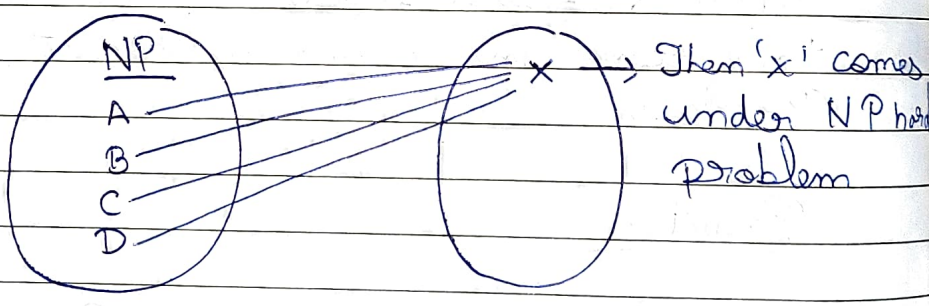
→ $A \propto B$ (A is reducable to B)

If we can solve A using the solution of B and the reduction must be in polynomial time, then $A \propto B$.

[vice versa is also correct]

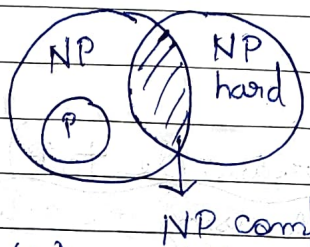
Let A & B be two problems then problem A reduces to problem B iff there is a way to solve problem A by the deterministic algorithm that solve problem B in polynomial time.

☑ NP hard :- (A problem is NP hard when an algorithm to solve it can be translated into an algorithm to solve any NP problem.) A problem 'x' is NP hard if all the problems in NP are polynomial and reducable to 'x'.



☑ NP complete problem :-

When a problem 'Y' is NP as well as NP hard, it is called as NP complete.



A language 'B' is NP complete if it satisfies two conditions -

i) 'B' is in NP

ii) Every 'A' problems in NP is polynomial time reducable to 'B'

If a Polynomial time algorithm exists in the any of these problems, all problems in NP would be polynomial time solvable. These problems are called NP-complete problem.

If a problem is NP and all other problems in NP is polynomial time reducable to it, it is known as NP complete problem.

6/11/24

Decision problem

- Any problem for which the answer is yes or no is called a decision problem. An algorithm for a decision problem is termed as decision algorithm. Ex- In 0/1 Knap Sack finding the maximum profit for the given 0/1 Knap Sack is optimization problem.
- Is it possible to get a profit of atleast K?

Optimization problem

Optimization problem - Any problem that involves the optimization of a value of a given cost function is an optimization problem. A optimization algorithm is used for solving an optimization problem.

- Example - In 0/1 Knapsack finding the maximum profit for the given 0/1 Knapsack is Optimization problem.

6/11/24

Satisfiability problem or boolean satisfiability problem is a problem of determining if there exists an interpretation that satisfies a given boolean formula.

Literals $x, \bar{x}$

Clauses $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) = 1$

C_1 C_2

= indicates that it is satisfiable

Satisfiability problem is NP Hard problem.

3 CNF Problem is one of the satisfiability problem

Cook's theorem states that if a satisfiability problem can be solved in polynomial time, then P = NP.

3 CNF problem

To prove 3CNF problem is NP complete

$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

3CNF $\subseteq$ SAT

↓

3CNF is NP

Reducing SAT to 3CNF problem

i) $C_3 = (x_1 \vee x_2 \vee x_3 \vee x_4)$ SAT

$C_4 = (x_1 \vee x_2 \vee x_A) \wedge (\bar{x}_A \vee x_3 \vee x_4)$

11) $C_3 = (x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5)$

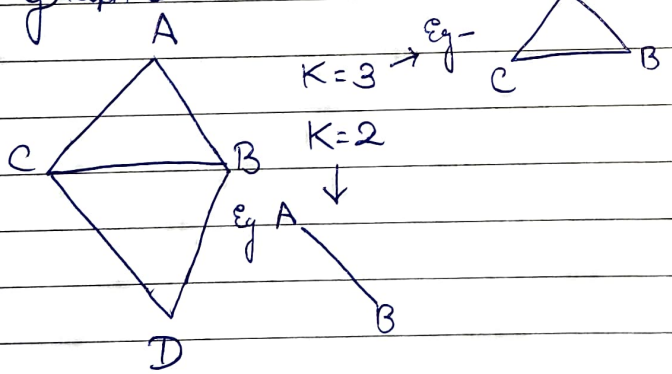
$C_5 = (x_1 \vee x_2 \vee \bar{x}_A) \wedge (\bar{x}_A \vee x_3 \vee x_B) \wedge (\bar{x}_B \vee x_4 \vee x_5)$

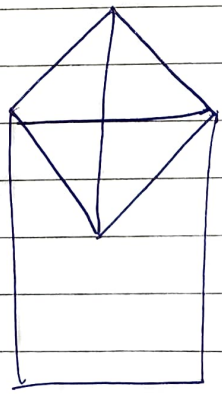
∵ 3 SAT is NP hard and it can be reduced to 3CNF
3CNF is also NP hard ; ∴ 3CNF is NP complete

Maximal ~~Click~~ Clique Problem

~~Click~~ Clique is a part of a graph which is complete.
Objective :- To find the maximal size of the ~~click~~ Clique

To check whether a ~~click~~ clique of size K is present in the graph or not.





maximal size of Clique = 4