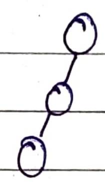
Dynamic Playlist (Aditya Verma)

DP = enhanced recursion

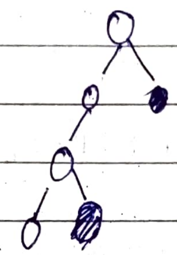
How to identify DP problem

1) Where there is recursion $\xrightarrow{\text{(2 calls)}}$, DP is used (for overlapping problem)

   a) Choice



1 function call
X (NO DP)



2 function call
(DP)

   b) Optimal – Min, max, largest

How to write DP code?

Recursive solution → memoization → Top down approach

Questions on DP.

1) 0-1 Knapsack (6)
2) Unbounded Knapsack (5)
3) Fibonacci (7)
4) LCS (15) (longest common subsequence)
5) LIS (10) (longest increasing subsequence)
6) Kadane's Algorithm (6)
7) Matrix Chain Multiplication (7)
8) DP on trees (4)
9) DP on grid (14)
10) Others (5)

Types of Knapsack

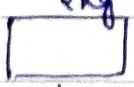
1. Subset sum
2. Equal sum partition
3. Count of subset sum.
4. Minimum subset diff
5. Target sum.
6.

0-1 Knapsack Problem

- Fractional Knapsack (Greedy)
- 0-1 Knapsack
- Unbounded Knapsack

• 0-1 Knapsack → We are given some weight & some value. Then a max weight W. pick items so that the profit is maximum. And the weight has a given bound W.

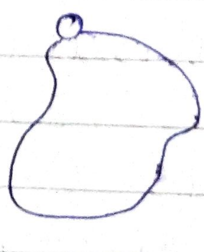
2Kg



→ ₹10

Brick

	I_1	I_2	I_3	I_4
wt[] =	1	3	4	5
val[] =	1	4	5	7

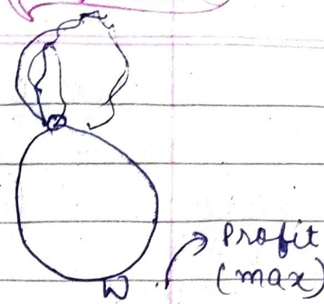


W = 7kg.

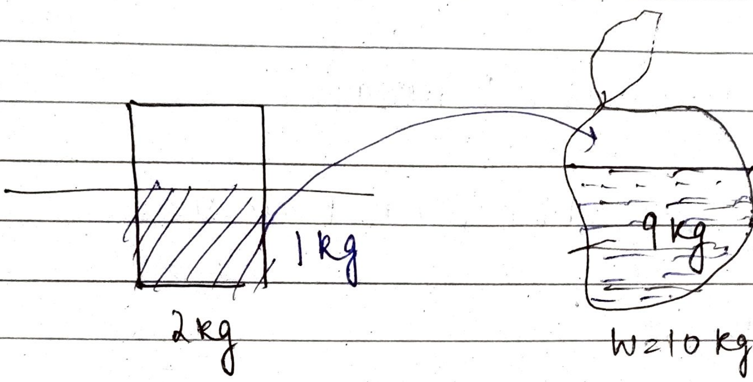
Max Profit = ?

P_1 P_2 P_3 P_4

W_1 W_2 W_3 W_4



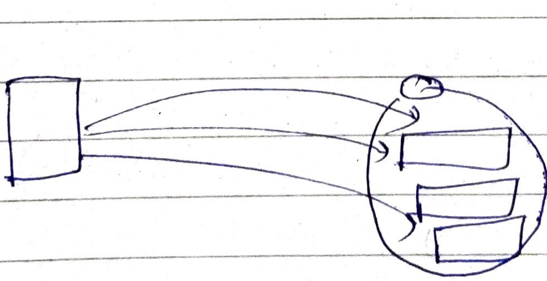
Fractional knapsack



Greedy approach

Unbounded knapsack

Unlimited supply of every item



0-1 knapsack

1) How to identify

wt[] : 1 3 4 5 W : 7kg

val[] : 1 4 5 7

O/p : max profit

1) Choice
↓
(either put in knapsack) (either don't put in knapsack)

2) Optimal (max profit)

DP: Recursive → Memoization (DP) → Top down (DP)

DP → recursion + storage

0-1 knapsack Recursive

Identify

DP → Recursive → DP(top down), DP (memoization)

I/P wt[] = | 1 | 3 | 4 | 5 |

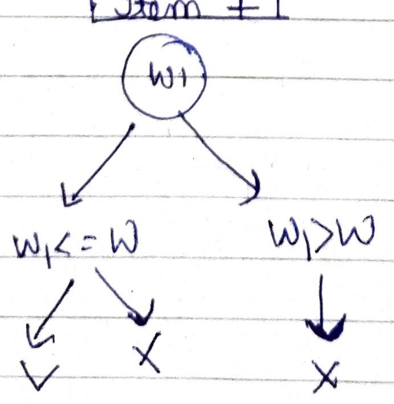
val[] = | 1 | 4 | 5 | 7 |

O/P → Max Profit

(capacity of knapsack) W = 7 kg

Choice diagram

Item 1



We have to return the max profit so return type would be int.

Base cond^n → think of the smallest valid i/p.

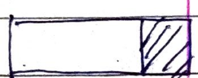
I/P wt [ ] : ☐☐☐] → (n) → 0.
val[ ] : ☐☐☐
W : 10kg ——→ 0 kg

max profit ←

```
int Knapsack ( int wt[ ], int val[ ], int W, int n) {
   // base condition
     if (n==0 || W==0)
         return 0;

   // choice diagram

     if (wt[n-1] <= W) {
        return max(val[n-1] + Knapsack (wt, val, W-wt[n-1], n-1),
                   Knapsack (wt, val, W, n-1);
     }
     else if (wt[n-1]>W) {
        return Knapsack (wt, val, W, n-1);
     }
```

→ weight less hain & including

↓ weight less hai, but dont include



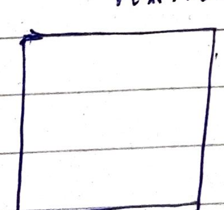
n-1

01 Knapsack Memoization

Memoization = Recursive + 2 lines

t

n*m



func (IP/doesnt change , IP/change)

n-1

W-wt[n-1]

Val of n, m &

Knap Sack (wt[ ], val[ ], n , W)

t [n+1] [W+1]

n+1

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	2	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

w+1

if (-1) is not present then val exists so, return

initialise this matrix with -1.

```
int t [n+1] [w+1]
memset (t, -1, size of(t))
```

Changes

New declare the matrix globally.

```
int static t [102][1002];   constraint
                              n<=100
memset (t, -1, sizeof t)      w<=1000

int Knapsack (int wt [], int val[], int w, int n)
{
    if (n==0 || w==0)
        return 0;
    if (t [n][w] != -1)
        return t[n] [w];

if (wt[n-1] <= w)
return t[n][w] = max (val [n-1] + knapsack (wt, val, w-wt[n-1], n-1)
    , knapsack (wt, val, w, n-1) );
```

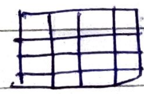
```
else if ( wt[n-1] > W)
return t[n][W]= Knapsack ( wt, val, W, n-1);

}
```

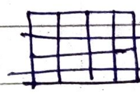
The complexity of top down & Memoization remains same but the problem with memoization is the stack gets full due to repeated funcn calls.

(0-1 Top Down Knapsack) → Real DP

Recursive → Memoize → Top down → 6 Problems

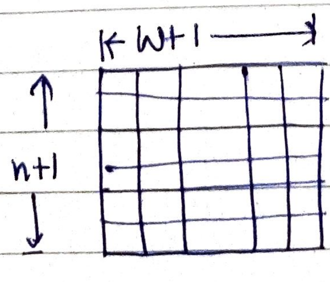
- Recursive ↓ BC + recursive calls
- Memoize ↓ R C + table
- Top down ↓ Only Table





Recursive → Memoization — Initialising matrix with -1

Recursive → Top-down (totally omit the recursive call & use the table only)



We only make table for the i/p which is changing.

2 steps to make table for top down

Step1: Initialization

Step 2: Recursive code $\xrightarrow{\text{changes}}$ Iterative code

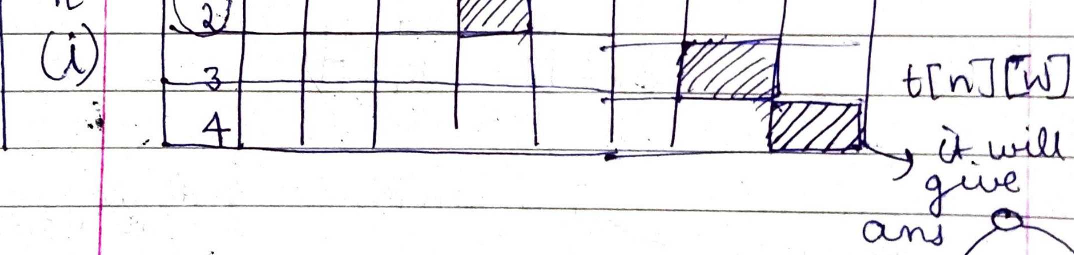
Step1: Initialize

W = 7

n = 4

	0	1	2	3
wt[] =	1	3	4	5
val[] =	1	4	5	7

val	wt
1	1
4	3
5	4
7	5



wt[] = [1 3 4 5]
val[] = [1 4 5 7]

wt[] = [1 3]
val[] = [1 4]
W = 3

W = 3

wt[] = [1 3 4]
val[] = [1 4 5]
W = 6

RC + table $\longrightarrow$ table

Base cond^n $\longrightarrow$ Initialization

	w=0	1	2	. . .
1	0	0	0	0
2	0			
3	0			
4	0			

Base cond^n

RC

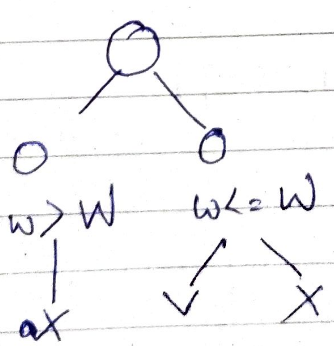
```
if (n==0 || W==0)
    return 0;
```

↓

table

```
for (int i=0; i<n+1; i++) {
    for (int j=0; j<w+1; j++)
    {
        if (i==0 || j==0)
            t[i][j] = 0;
    }
}
```

Choice diagram



n, w $\longrightarrow$ i, j

7+

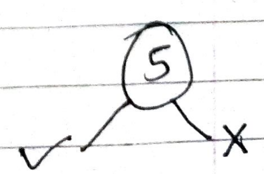
dp[i][j]

RC.

wt = 1 3 4 5

val = 1 4 5 7

W=7



max(val[n-1] + dp[i-1][j - wt[i-1]],
dp[i-1][j])

Recursive	Top down
if(wt[n-1] <= W)	if (wt[n-1] <= W)
return max(val[n-1] + Knapsack(wt, val, W - wt[n-1], n-1),	t[n][W] = max(val[n-1] + t[W - wt[n-1]][n-1],
Knapsack(wt, val, W, n-1))	t[n-1][W])
else if (wt[n-1] > W)	else
return Knapsack(wt, val, W, n-1)	t[n][W] = t[n-1][W]

```
       0 1 2 3
wt  =  1 3 4 5      W = 7
val =  1 4 5 7      n = 4

W = 0 1 2 3 4 5 6 7
    0 1 2 3 4 5 6 7
```

pseudo code

Top - down approach

```
int t[n+1][w+1];
for (int i=1; i<n+1; i++)
    for (int j=1; j<w+1; j++)
    {
        if (wt[i-1] <= j)
            t[i][j] = max( val[i-1] + t[i-1][j - wt[i-1]],
                           t[i-1][j] )
        else
            t[i][j] = t[i-1][j]
    }
return t[n][w]
```

i - 1 = 0

1 → 4

1 → 7

1 → 1 <= 1 → True

i=1 wt[i-1] <= 1

1 <= 1

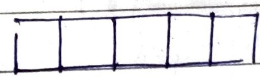
t[i][j] = max(1 + t[0][0],
t[0][1])

= max(1, 0)

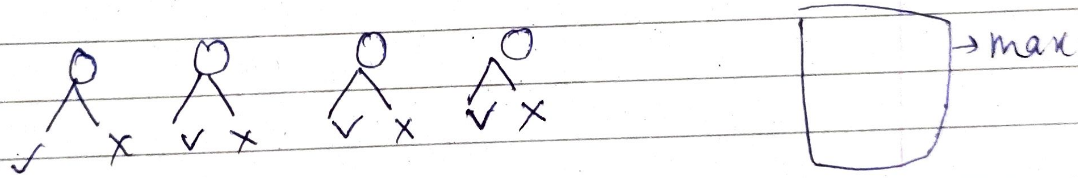
= 1

Identification of Knapsack Problem

1) Subset sum problem
2) Equal sum partition
3) Count of subset sum.
4) Minimum subset sum diff
5) Target sum
6) No of subset with a given diff.

Ip: Item array : 

W : Capacity



1. Subset Sum problem

arr[] : 2 3 7 8 10

sum = ...

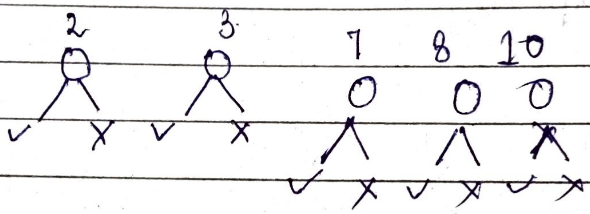
a) Problem statement
b) Similarity with knapsack
c) Code variation

1) Problem statement : find if there is a subset present in an array with given sum.

arr [] : 2 3 7 8 10
Sum : 11

2) Similarity

item array [] : → 2 3 7 8 10
weight : capacity → 11



3) Code variation

t[n+1] [w+1] t[5+1] [11+1]

sum → t[6] [12]

Initialization :

	0	1	2	3	4	5	6	7	8	9	10	11
0	T	F	F	F	F	F	F	F	F	F	F	F
1	T											
2	T											
3	T											
4	T											
5	T											

g → True/false

↓ n

arr [] :
sum : 0

arr[ ] : 1
sum = 0 '{}'

arr[] : 2
sum = 0 '{}'

arr[] :
Sum = 2

when array is empty sum can't be anything

arr[ ]: no elements

Sum: 1 → not possible

t[n+1][sum+1]

Initialisation: for(int i = 1 . . . n+1)

for(int j = 1 . . . W+1)

if (i == 0)

t[i][j] = false.

2 3 7 8 10

if (j == 0)

t[i][j] = True

10

× ✓

knapsack → arr.	Subset sum
if (wt[i-1] <= j)	if (arr[i-1] <= j)
t[i][j] = max(val[i-1] + t[i-1][j - wt[i-1]], t[i-1][j])	t[i][j] = t[i][j - arr[i-1]] || t[i-1][j]
↓ there is no max in true or false.	
else	else
t[i][j] = t[i-1][j]	t[i][j] = t[i-1][j]
	return t[n][sum];

i j

t[1][0] = t[1][0 - arr[0]] || t[0][0]

T

= t[1][-2] || t[0][0]

|| T

2, 3, 8

t[i][j] → {3, 8} → True ✓

→ {3} → False ×

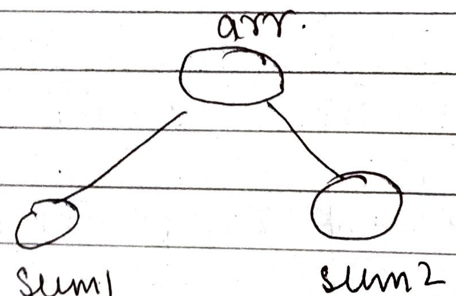
2. Equal Sum Partition Problem

1) Problem Statement
2) Subset Sum Similarity
3) Odd / Even significance
4) Code variation

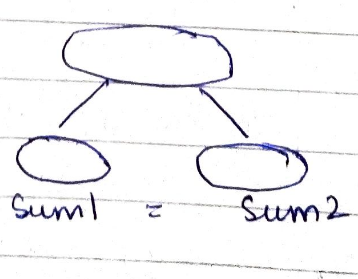
1) Problem Statement

arr[] = {1, 5, 11, 5}
O/p : T/F

Is it possible to divide the array such that both the subset gives an equal sum



sum 1 = sum2 → True / False



sum 1 = sum 2 → It is equal when the no is even. Sum of all the array elements should be even to change it into equal parts.

22

11 11

sum1 = sum2

23

11 12 X

Sum1 ≠ sum2

I/p
arr[]

→ True O/P

→ False

```
for (int i = 0; i<size; i++) {
    sum = sum + arr[i];
}
```

{1, 5, 11, 5}

Subset even

Subset odd

sum = 22

(false)

Subset = 11

Subset 11

```
if (sum % 2 != 0)        (sum is odd)
    return false
```

```
else if (sum % 2 == 0)
```

→ we need to find one subset with sum 11 the next subset would automatically be 11.

```
return subsetsum(arr, sum/2);
```

3. Code:

i/p → arr[], n.

```
int sum = 0;
for (int i = 0; i<n; i++)
    sum += arr[i];
if (sum % 2 != 0)
    return false
else
    return subsetsum(arr, sum/2);
```

3. Count of Subsets sum with a given sum.

I/p:

arr[ ] = 2 3 5 6 8 10

sum = 10.

Flow

1) Problem statement
2) Similarity to subset sum
3) Code variation.

Initialisation   Code

4) Return Type.

1) Problem statement

arr[ ] 2 3 5 6 8 10

sum : 10

O/p = 3.

{2,8}. → Yes/True (in subset sum)

{2, 8} = 10
{5, 2, 3} = 10
{10} = 10

count = 3

2) Similarity

2, 8

Yes NO

return count

3. Code variation

Subset sum

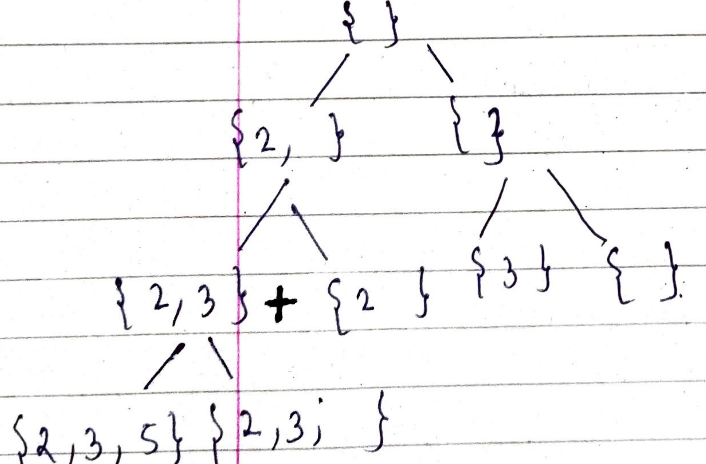
due to T/F ← bool

False → 0 (no of subset 0)

True → null subset (no of subset 1)

if (arr[i-1] <= j)
dp[i][j] = dp[i-1][j] ~~||~~ + t[i-1][j-arr[i-1]]

else
dp[i][j] = dp[i-1][j]



Count
↓
int

~~0~~ 1	0	0	0	0	0
1					
1					

we will add all the subset so or would be changed to + . True & false case we can use ~~bool~~ or.

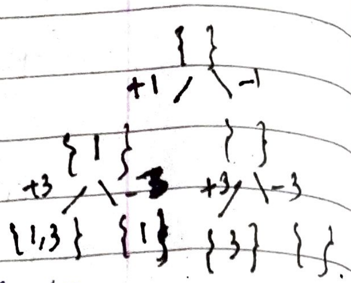
if (arr[i-1] <= j)
dp[i][j] = dp[i-1][j] + dp[i-1][j-arr[i-1]]

else
dp[i][j] = dp[i-1][j]

if (arr[i-1] <= j)
dp[i][j] = dp[i-1][j-arr[i-1]] + dp[i-1][j]
else
dp[i][j] = dp[i-1][j].

arr[] = 2 3 5 6 8 10
Sum = 10
O/p = 3

1, 3, 2, 5   5
O/p → {3, 2}
{5}



arr[0] <= 1

dp [N+1][Sum+1] = dp[6+1][10+1]
= dp[7][11]

i=1 j=1 2-2=0
2 <= 1
i=1 j=2
arr[0] <= 2
2 <= 2
dp[0][0]
dp[0][2]

Sum →

	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1								
2	1										
3	1										
4	1										
5	1										
6	1										

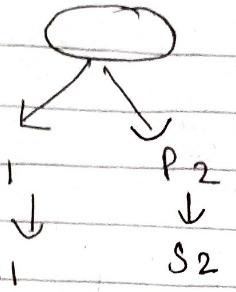
Minimum subset sum difference

1) Problem statement
2) Similarity
3) Solve using its previous concept

1) Problem statement

arr[] : | 1 | 6 | 11 | 5 |

OP : 1



Equal sum : $S1 - S2 = 0$

Minm' subset : $S1 - S2 = \min$

$abs(S1 - S2) = \min$ (should be min)

1, 6, 11, 5

{1, 6} → 7

{11, 5} → 16

$16 - 7 = 9$ → minimize of find could it be done in a better way.

{1, 6, 5} → 12

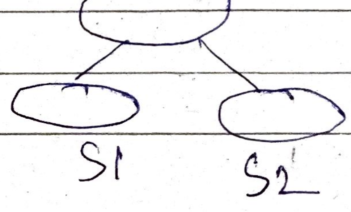
{11} → 11

$12 - 11 = 1$ → Can't be minimized further

→ O/P.

2) Similarity

It is similar to equal sum partition.



Date
Page

We need to find S1 & S2.

arr[] | 1 | 6 | 11 | 5 |

{ } { }

S1 S2

we can find the range of S1 & S2.

$\to S1 = 0$ (0+0+0+0)

{ }

{ arr }

extreme value of subset

$\to S_2 = 23$ (1+6+11+5)

S_1 S_2

0 23

arr[] : {1, 2, 7}

0 1 2 3 4 5 6 7 8 9 . . . 23

Sum line

127

12 7

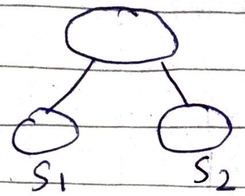
0 1 2 3 4 5 6 7 8 9 . 10

x x x

$\Sigma arr[i]$

S_1 / S_2 can't be 5

S1/S2 = {0, 1, 2, 3, 7, 8, 9}



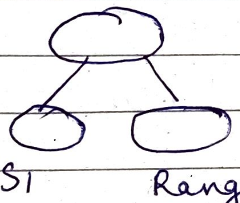
$S1 + S2 = \Sigma arr[i]$

$S1/S2 = \{0, 1, 2, 3, \mid 7, 8, 9, 10\}$

S1 (S1) | Range - S1 (S2)

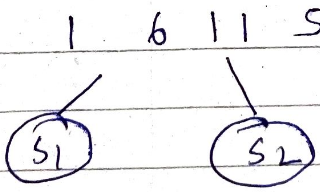
$S1 - S2 =$ minimize

abs(S2 - S1) or
abs(S1 - S2)



abs(S2 - S) = minimize
abs(Range - S1 - S1) = minimize. abs(Range - 2S1) → minimize

Sum up



$\left.\begin{array}{l} S1 - S2 \\ S2 - S1 \end{array}\right\}$ minimize

↓

((Range - S1) - S1)

(Range - 2S1)

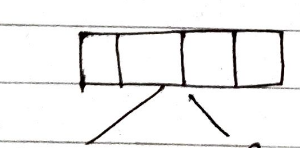
	0	1	2	3	4	5	6	7	8	9	10
0											
1											
2											
3	T	T	T	T	F	F	F	T	T	T	T

→ we will push the last row in vector till half way.

0	1	2	3.

```
int min = INT_MAX;
  for (int i = 0; i < v.size(); i++)
  {
     mn = min (mn, Range - 2v[i])
  }
  return mn;
}
```

Concept



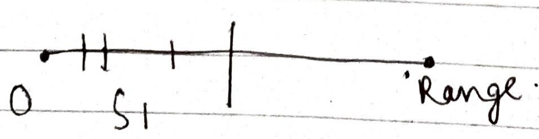
$(S_2 - S_1) = \min$

↓

smaller

(S_1) $(Range - S1)$

$Range - 2S1 \rightarrow Min^m$



Code

Subset sum(int arr[ ], int range)

{

last row filled in vec till halfway

0	0	0	

}

```
int mindiff (int arr[], int n) {
  int Sum = 0;
   for (int i = 0; i<n; i++) {
       Sum += arr[i];
   }
   bool dp[n+1][sum+1];
   for (int i=0; i< n+1; i++){
      for (int j=0; j< Sum+1; j++) {
      if(i==0)    dp[i][j]= false;
      if(j==0) dp[i][j]=true;
      }
   }
  for (int i=1;  i<n+1; i++) {
      for (int j=1; j<sum+1; j++) {
         if (arr[i-1]<=j)
            dp[i][j]= dp[i-1][j-arr[i-1]] || dp[i-1][j];
         else
            dp[i][j]= dp[i-1][j];
      }
  }
  int diff = INT_MAX;
   for (int j= sum/2; j>=0; j--) {
      if(dp[n][j] == true) {
         diff = min(sum- 2*j, diff)
      }
   return diff;
}
```

Date
Page

Count the number of subset with a given diff

1) Problem statement
2) Will try to reduce the actual statement
3) Solve it using already solved problem.

1) Problem statement

arr[ ] : | 1 | 1 | 2 | 3 |

Diff : 1

| 1 | 1 | 2 | 3 |

S1   S2

S1 − S2 = Diff → return the count of Subset with diffⁿ.

| 1 | 1 | 2 | 3 |

1+3   1+2

4   3 = 1

1

| 1 | 1 | 2 | 3 |

1+3   1+2

4   3

= 1

| 1 | 1 | 2 | 3 |

1+1+2   3

4 − 3 = 1

1	1	2	3

S_1 $\qquad$ S_2

(diff) S = sum of arr

$$\text{sum}(S1) - \text{sum}(S2) = \text{diff}$$
$$\text{sum}(S1) + \text{sum}(S2) = S$$

$$2S_1 = \text{diff} + \text{Sum}(\text{arr})$$

$$S_1 = \left(\frac{\text{diff} + \text{sum}(\text{arr})}{2}\right)$$

$$= \frac{1+7}{2} = \frac{8}{2} = 4$$

$S_1 = 4$ $\qquad$ $S_2 = S_1 - \text{diff}$

$= 4 - 1$

Count = ? $\qquad$ $= 3$.

- Find count when sum of $S1 = 4$

no of count of Subset with given diff $\longrightarrow$ count of Subset sum.

Sum up.

S_1 $\qquad$ S_2

$S_1 - S2 = \text{diff}$

$S_1 + S2 = \text{sum}(\text{arr})$

$2S1 = \text{diff} + \text{sum}(\text{arr})$

$$S_1 = \frac{\text{diff} + \text{sum}(\text{arr})}{2}.$$

Date
Page

int sum = $\frac{diff + sum(arr)}{2}$.

return countofsubset sum(arr, sum);

```
int countofSubsetwithdiff (int arr[], int n) {
                                  int diff
    int arrsum = 0
    for (int i=0; i<n; i++){
        arrsum += arr[i];
    }

    int sum = 0;
    Sum = (diff + arrsum) / 2 .

    return countofsubsetsum( arr, sum, n);
}

int countofsubsetsum(int arr[], int sum, int n){
    int dp [n+1][sum+1];
    for (int i=0; i<n+1 ; i++){
        for (int j=0; j<sum+1; j++) {

            if (i==0)   dp[i][j] = 0;
            if (j==0)   dp [i][j] = 1;
        }
    }
```

```
for (int i = 1; i < n+1; i++){
        for(int j = 1; j < sum+1; j++) {

            if (arr[i-1] <= j) {
            dp[i][j] = dp[i-1][j - arr[i-1]] + dp[i-1][j];}
            else {
            dp[i][j] = dp[i-1][j];
            }
        }
}
return dp[n][sum];
}
```

Target Sum.

arr : | 1 | 1 | 2 | 3 | → (+/-)

sum : 1

tc : [0,0,0,0,0,0,1]
target = 1

O/p : 3

Target value = 2

+1 -1 -2 + 3 = 1
-1 +1 - 2 + 3 = 1
+1 +1 + 2 - 3 = 1

{0,2}

(+0, 2) (-0, 2)

count = 2

so your answer is increased by a power of (2^ no of zeroes)

pow(2, no of zeroes)

arr [ ] → | + | - | + | + |

{0, 0, 2}

{+0, +0, 2} {+0, -0, 2} {-0, -0, 2}

{-0, 0, 2}

= 4 (count

| | | | |

S1 S2

S1 - S2 = diff

Date
Page

+ - - +

1 1 2 3

+1+3 -1-2

(1+3) - (1+2)

S1 - S2 → count of subset with given diff^n.

~~Unbounded Knapsack~~

```
int findTargetSumWays(vector<int>& nums, int S) {
    int cnt = 0, sum = 0;
    int n = nums.size();
    for (int i = 0; i < nums.size(); i++) {
        Sum = sum + nums[i];
        if (nums[i] == 0)
            Cnt = cnt + 1;
    }
    S = abs(S);
    if (S > sum || (S + sum) % 2 != 0)
        return 0;
    int s = (S + sum)/2;
    int dp[n+1][s+1];

    for (int i = 0; i < n+1; i++)
        for (int j = 0; j < sum+1; j++)
            if (i == 0) dp[i][j] = 0;
            if (j == 0) dp[i][j] = 1;
```

Cnt the no of zeroes in subset

```
for (int i=1; i<n+1; i++) {
    for (int j=1; j<s+1; j++) {

        if(nums [i-1] == 0)
            dp[i][j] = dp[i-1][j];

        else if (num[i-1]>j)
            dp [i][j] = dp[i-1][j];

        else
            dp[i][j] = dp[i-1][j-nums[i-1]] +
                        dp[i-1][j]
    }.
  }.
  return pow(2,cnt) * dp[n][s];
}.
```

if 0 is there in the array dont take it

13. Unbounded Knapsack

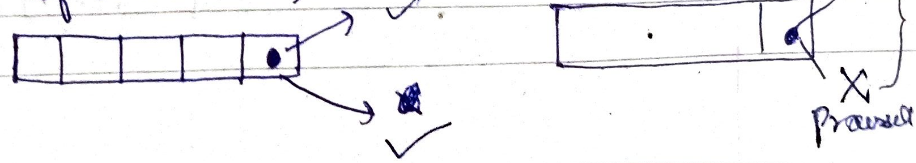
Related Problems

1) Road cutting
2) Coin change I (Max no of ways)
3) Coin change II (Min no of ways)
4) Maxm: Ribbon cut

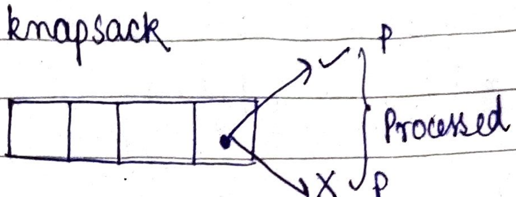
(Variations of unbounded knapsack)

Unbounded (multiple occurrences of same item)

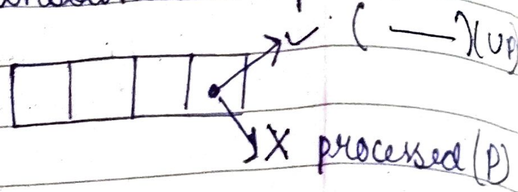
Knapsack (only one occurrence)



knapsack



Unbounded knapsack



Multiple occurrences
- ✓ (Can revisit many times)
- X

Comparision bet^{n}.

Knapsack

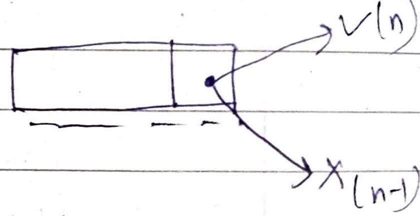
	0	1	2	3	4
0	0	0	0	0	0
1	0				
2	0				

t[n+1][w+1]

```
if(wt[i-1]<=j){
t[i][j] = max( val[i-1]+t[i-1][j-wt[i-1]],
               t[i-1][j]); }
else {
t[i][j] = t[i-1][j];
}.
```

Unbounded Knapsack

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




```
if (wt[i-1] <= j)
t[i][j] = max(val[i-1] + t[i][j-wt[i-1]],
              t[i-1][j] );
else
t[i][j] = t[i-1][j]
```

14. Road Cutting Problem

length [ ] :

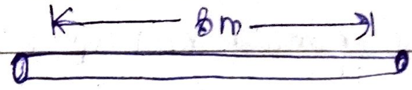
1	2	3	4	5	6	7	8

price [ ] :

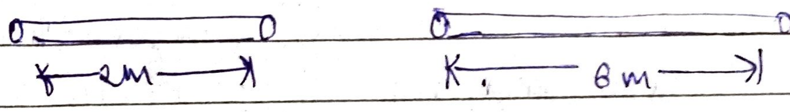
1	5	8	9	10	17	17	20

N : 8

1) Problem statement



length of rod is given. we need to cut it in such a way the profit is max^m.



Rs 5   Rs 17

$17 + 5 = 22$

Total = 5+17
= Rs 22

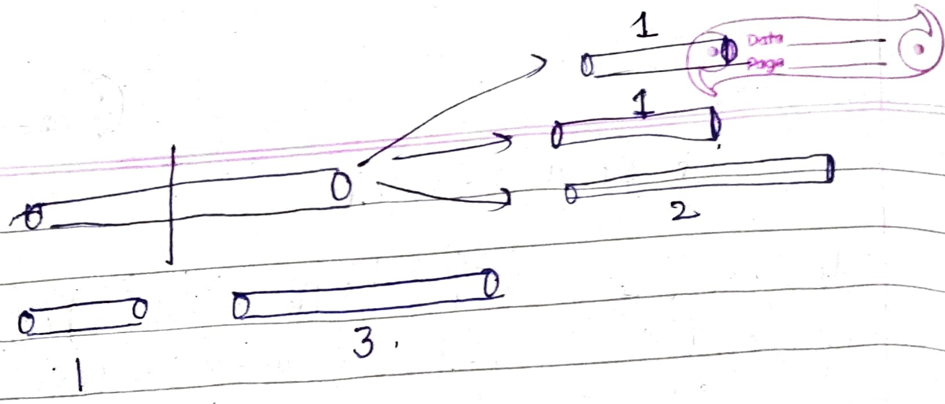
knapsack

wt
val
W.

length = 1 to N
Price = [| | | |]
N = 8

if length array is not give make it & fill it will 1toN.

knapsack	unbounded knapsack
val[ ]	price [ ]
wt[ ]	length [ ]
W	N



Code variation

t[N+1] [N+1]

```
if (length[i-1] <= j)
    dp[i][j] = max(price[i-1] + dp[i][j-length[i-1]],
                   dp[i-1][j])
else
    dp[i][j] = dp[i-1][j];
```

sometimes arr size changes therefore we need to find the size of array.

dp[size+1][N+1].

length →

↓ size

0	0	0	0	0	0
0 0					

Coinchange Problem

Max^m no of ways

Min^m no of coins

Max^{m} no of ways.

Problem statement

coin [ ] : | 1 | 2 | 3 | → (unlimited)

sum : 5.

Use the coin array unlimited time & get the sum 5. Now find the max^{m} no of ways in which we can get 5.

	O/P
2 + 3	5.
1 + 2 + 2	5.
1 + 1 + 3	5.
1 + 1 + 1 + 1 + 1	5.
1 + 1 + 1 + 2	5

5 ways.

Q Why knapsack?

→ Every coin has a choice to include or not

coin : | 1 | 2 | 3 |

× ✓ × ✓ × ✓

1 2 3

× ✓ ✓

item

- wt [ ] ✓
- val [ ] × (when one array is given)

Matching

wt[] → coin[]
W → sum

1	2	3.

Sum 5 : 1 + 1 + 3

one item used
many times

If taking one item many times does the sum allows then it is unbounded knapsack.

Subset sum

1	2	3	5

→ T
→ F.

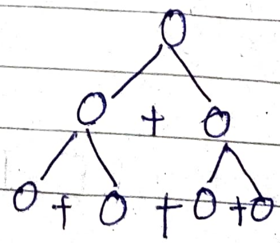
Sum : 8

Subset sum

```
if (arr[i-1]<=j){
t[i][j] = t[i-1][j] ~~||~~ t[i-1][j-arr[i-1]]
} else {
t[i][j] = t[i-1][j];
}
```

for count
we remove
|| by +.

count / no of ways



We need to find maxim. no of ways.

Matching → Knapsack → 0-1, → Unbounded

wt → coin

W → sum.

```
if ( coin[i-1]<=j )
  t[i][j] =  t[i-1][j] + t[i][j - coin[i-1]].
else
  t[i][j] = t[i-1][j]
```

sum →

size ↓	0	1	2	3	..	..
0	1	0	0	0	0	0.
1	1					
2	1					
3	1					

Coin change-II (Min^m no of coins)

coin[ ] = | 1 | 2 | 3 |

sum = 5

2+3 → 5 → 2 coins

1+2+2 → 5 → 3 coins

1+1+1+2 → 5 → ~~3~~ 4 coins

1+1+3 → 5 → 3 coins

1+1+1+1+1 → 5. → 5 coins

find min^m. no. of coins to make sum as 5.

coin[ ] = [1 | 2 | 3]
sum = 5
O/P : 2 (2+3 = 5)

Initialisation : t[n+1][w+1] → t[n+1][sum+1]

wt → coin[ ] = n
Val → X
W → sum.

n=3
sum=5

	0	1	2	3	4	5	6
0		INT_MAX -1	INT_MAX -1	INT_MAX -1	INT_MAX -1	INT_MAX -1	INT_MAX -1
1	0						
2	0						
3	0						

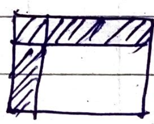
sum (5) →

size (n)

→ 2nd Row initialize

↓ code

sum = 3
coin[ ] = 1

Initialisation  + Twist

coin[ ] : empty
sum : 1 } → INT_MAX (∞ coins)

coin[ ] : empty
sum : 0 } → INT_MAX - 1

[redacted]

coin[ ] : 1
sum : 0 } 0 coins

coin[ ] : 1 2
sum : 0 } 0 coins

For ques — sum = 5.
arr = [3, 5, 2]

when arr [3]
sum = 4

$\frac{3}{3} = 1$ $\qquad$ $\frac{4}{3}$ = INT_MAX

i \ j		0	1	2	3	4	5
0		INT_MAX − 1					
1			1				
2	0						
3							
4							

sum = 1
coin = 1

sum = 2
coin = 1 (1)

$\frac{4}{3}$ = INT_MAX

$\frac{j}{arr[0]} = \frac{3}{3} = 1$

size will always remains 1 so we will use arr[0] as it is the second row

For second row

```
for(int i=1; j< sum+1; j++) {
    if (j % arr[0] == 0)
        t[i][j] = j/ arr[0]
    else
        t[i][j] = INT_MAX-1
```

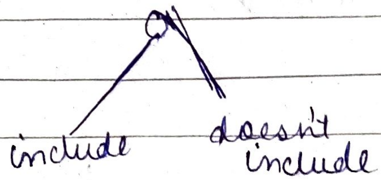
$\frac{j}{arr[0]}$ — $\frac{3}{3} = 1$; $\frac{4}{3}$ = INT_MAX

Code variation

1st row → INT_MAX   1st col → ~~start~~ 0

(INT_MAX − 1)

```
     coin
if ( ~~coi~~ [i-1] <= j)

t[i][j] = min( t[i-1][j] , t[i][j - coin[i-1]] )
                                 +1
else
    t[i][j] = t[i-1][j]
```




```
int minCoins( int coins[], int M, int V) {
    // m = arrsize   v = sum.
    int dp [M+1][ V+1 ];
    for (int i=0; i<M+1; i++){
        for (int j = 0 ; j<V+1; j++) {
            if (j==0 )  dp[i][j] = 0;
            if (i==0 )  dp[i][j] = INT_MAX-1;
    } }

    for(int i=1, j=1 ; j<V+1 ; j++) {
        if(j % coins[0] ==0)  dp[i][j] = j/coins[0];
        else
                              dp[i][j] = INT_MAX-1;

    for (int i = 2; i<M+1; i++) {
        for(int j = 1; j< V+1; j++) {
        if (coins [i-1]<= j)
            dp[i][j] = min (dp[i-1][j], 1+dp[i][j-coins[i-1]]
        else  dp [i][j] = dp [i-1][j];
    }}

    if(dp [M][V] == INT_MAX-1) return -1;
    return dp[M][V];
}.
```

Longest Common Subsequence

1) Longest common substring
2) Print LCS
3) Shortest common supersequence
4) Print SCS
5) Min^m no of insertion and deletion a→b
6) Largest repeating subsequence
7) Length of largest subsequence of a which is a substring in b.
8) Subsequence pattern matching
9) Count how many times a appear subsequence in b
10) Largest Palindromic subsequence
11) Largest palindromic substring
12) Count of palindromic substring
13) Minimum no of deletion in a string to make it a palindrome
14) Minimum no of insertion in a string to make it a palindrome

Longest Common Subsequence (Recursive)

Problem Statement

X : a b c d g h
Y : a b e d f h r.

String X = abcdgh
String Y = abedfhr

O/P → abdh
↘ 4 (length of string)

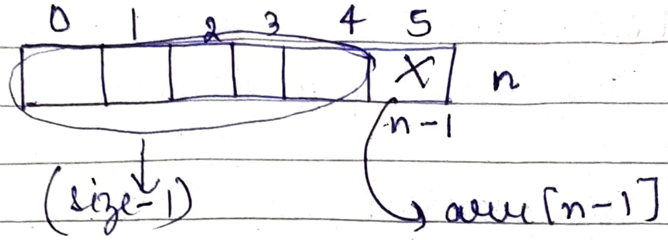
discontn. → longest common subsequence - abdh
contn. → longest common substring - ab

Recursive approach : Base condn + Choice diagram + i/p small

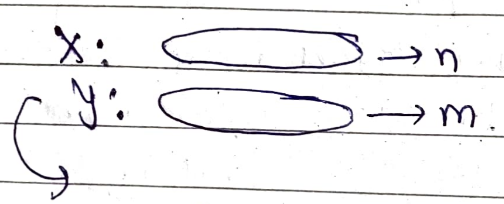
↓

(x , y)

func(x, y)
{
func(x, y)
↓ smaller x ↓ smaller y
}



Base condn : Think of the smallest valid input.



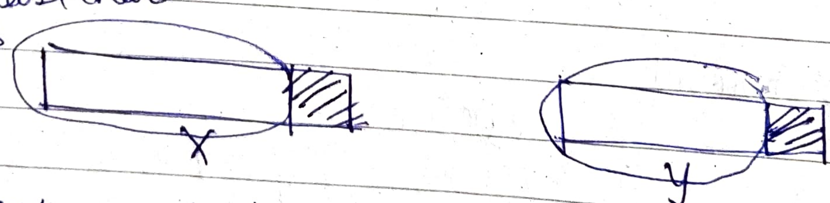
n=0 m=0 LCS = 0
(empty string)

if (n==0 || m==0)
return 0

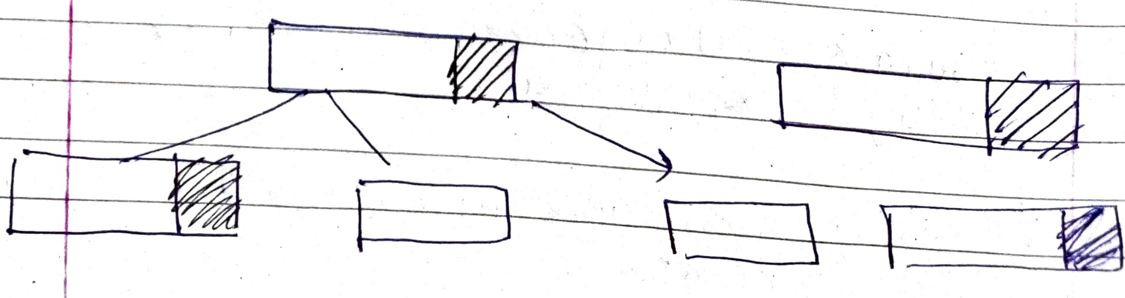
Choice diagram :

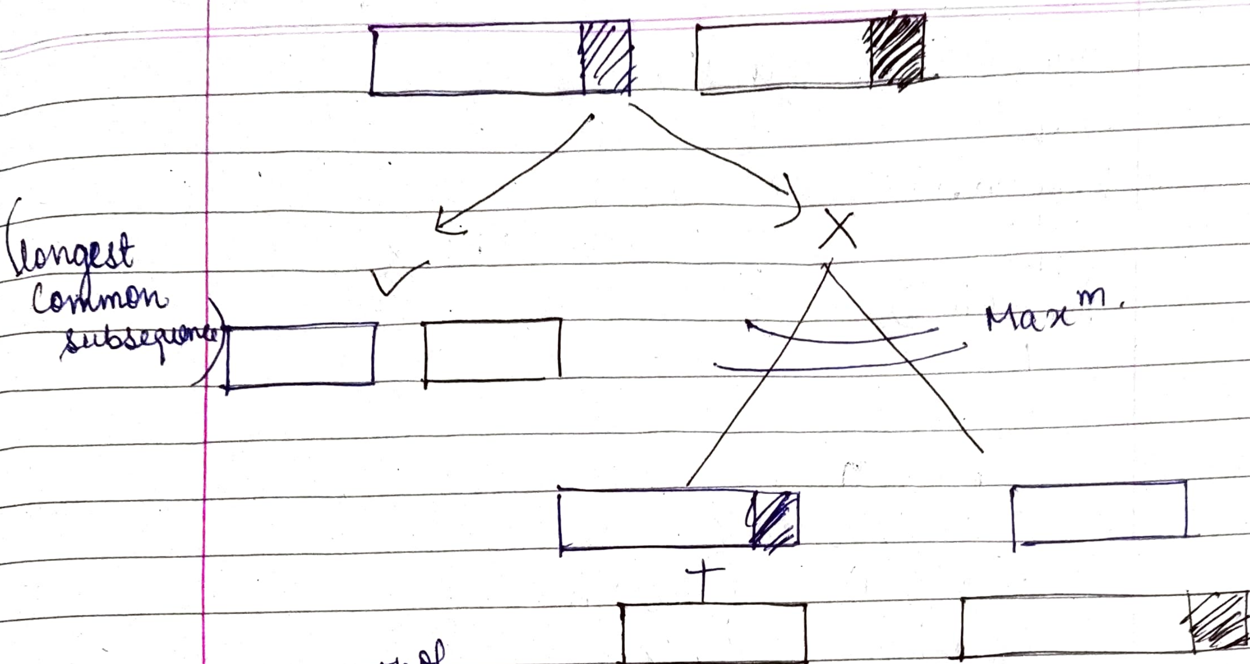
X : abcdgh
Y : abedfhr

① when last char matches



② when last char don't match





length of LCS

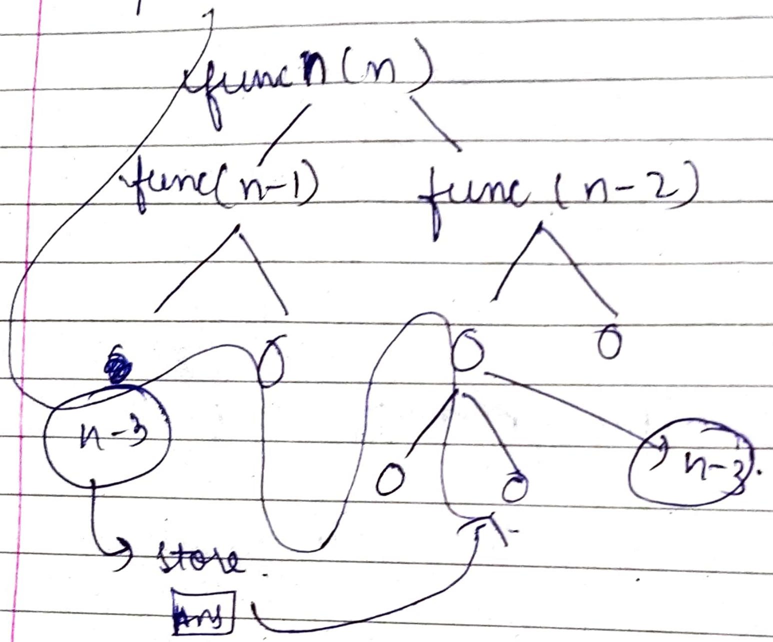
```
int LCS (string X, string Y, int n, int m) {

    if (n==0 || m==0) return 0;

    if ( X[n-1] == Y[m-1])
        return 1+LCS(X, Y, n-1, m-1);

    else
        return max (LCS(X, Y, n, m-1),
                    LCS( X, Y, n-1, m));

}
```

LCS memoization (Bottom up approach)

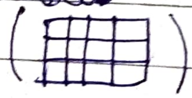
func(n)
↓
func(n−1)
|
func(n−2)
|

func( )
{
func( )
}

→ DP ✗



R.C + table



"AXYT" "AYZX"

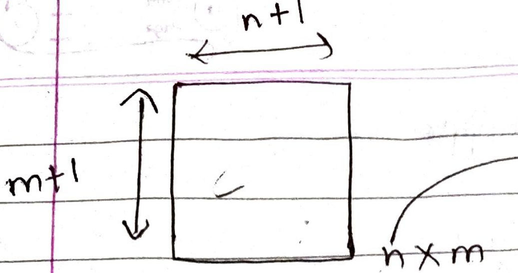
LCS(AXY, AYZX)

LCS(AXYT, AYZ)

LCS(AX AYZ) LCS(AXY, AYZ) LCS(AXY, AYZ)

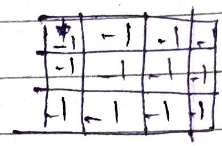
LCS(AXYT, AY)

↓
Sub − Problem
repeating twice



R C (change)
(n+1, m+1).

```
int t[1001][1001];

int main(){
    memset(t, -1, sizeof(t));
    LCS(    )
}

int LCS(string X, string Y, int m, int n){

    if (n==0 || m==0)
        return 0;
    if (t[m][n] != -1)
        return t[m][n];

    if (X[m-1] == Y[n-1])
    return t[m][n] = 1+LCS(X, Y, m-1, n-1)
    else
    return t[m][n] = max( LCS(X, Y, m, n-1),
                          LCS(X, Y, m-1, n))

}
```



exponential → O(n²)

LCS Top-down Approach

x : a b c d a f
y : a c b c f

o/p → 4 (abcf)

→ length y

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

↓ length x

Base ~~cond~~ condn → initialization
(RS) (Top down)

```
int LCS( String X, String Y, int n, int m) {
    int dp[m+1][n+1];

    for (int i=0; i<m+1; i++)
        for (int j=0; j<n+1; j++)
            if (i==0 || j==0)
                dp[i][j]=0

    for (int i=1; i<m+1; i++)
        for (int j=1; j<n+1; j++).
            if( X[i-1]== Y[j-1])
                dp[i][j]= 1+ dp[i-1][j]
    else
    dp[i][j]= max(dp[i][j-1], dp[i-1][j]);
```

Date
Page

return dp[m][n];
}

Longest common Substring

I/p a :→ abc de
b :→ abfce

O/p → 2 (ab)   ab, c, e

longest = ab

a b c d e
a b f c e

~~1~~ 2   ~~0~~ ~~1~~ ~~0~~ 1 → length = 0

m →

n ↓

0	0	0	0
0			
0			
0			

dis continuity = 0

```
if ( a[i-1] == b[j-1] )
    dp[i][j] = dp[i-1][j-1] + 1
else
    dp[i][j] = 0
// for max^m val.
    int maxi = 0;                 return maxi;
    for ( i = 0 → n+1 )
      for ( j = 0 → m+1 )  max(maxi, dp[i][j])
        maxi = max(maxi, dp[i][j])
```

Print LCS btw^n 2 string

a : a c b c f
b : a b c d a f

O/P → a b c f

~~φ a b c d a f~~

~~a c b c f~~

	φ	a	b	c	d	a	f
φ	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
f	0	1	2	3	3	3	4

Now, we need to print LCS

"f c b a"

abcf ← reverse

if equal i, j → (i--, j--)

if not equal i, j → max ((i-1, j), (i, j-1))

when equal
(i--, j--)

4

when not equal
(max betn a & b)

a

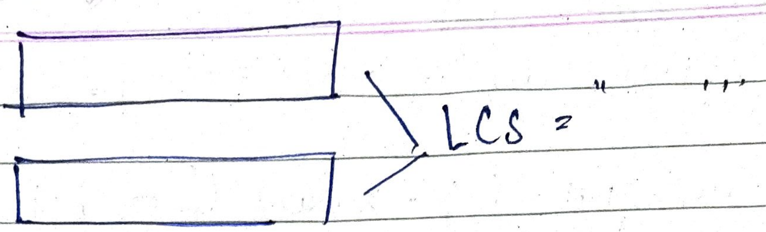
4

b

add in string

nothing to be added

```
int i = m, j = n;
String s = "";

while (i>0 && j>0)
{   a                b
  if (a[i-1] == b[j-1])
   {
     s.push_back(t[i-1])
       i--;
       j--;
   }
  else {
       if(t[i][j-1] > t[i-1][j])
          j--;
       else
          i--;
  }
reverse (s.begin(), s.end());
```



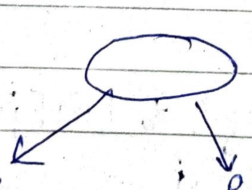

```
if (a[i-1] == b[j-1])
    s.push_back(a[i-1])
    i--;
    j--;
else
    move in the direcn of max'm
reverse(s.begin(), s.end())
```

24. Shortest Common Supersequence

a: "geek"
b: "eke"

1) Problem statement

→ 2 string are given. We need to merge the string such that we get both the subsequence.
Mix 2 sequence to make it a super sequence



geek + eke

O/P — geekeke , geeke

find shortest

Subsequence → Sequence of events

S_1 S_2 S_3

Order should be maintained but don't need to be continuous always

a : A G G T A B

b : G X T X A Y B

Super sequence : A G G T G X A B T X A Y B

A G G X T X A Y B → shortest supersequence

A G G T A B   G X T X A Y B

Give the length in O/P.

The one letter which is common write once.

A G G T A B

G X T X A Y B

↓

G T A B is common in both

i.e LCS.

Now thinking the brute force approach we can merge both the string & then subtract GTAB.

a: AGGTAB   b: GXTXAYB

Worst case: a+b = AGGTAB GXTXAYB   length of SS a+b

↓

AGGTABGXTXAYB − GTAB

↓

AGGXTXAYB

Shortest length = (m+n) − LCS

25. Minimum no of insertion & deletion to convert a string a to string b.

a : heap   O/p = 1 (Insertion)
b : pea   2 (deletion)

a ⟶ b

p(heap) ⟶ pea

insertion   deletion

When to use LCS.

	I/p	Q	O/p
LCS	a : ⬭ b : ⬭	LCS	int
Given Ques	a : ⬭ b : ⬭	insertion/deletion	int

→ Pattern matching algorithm

Convert heap to pea.

p ~~h~~ e a ~~p~~ ⟶ pea

"ea" is the LCS of both string

heap → pea

Insertion/deletion

$$a \longrightarrow b$$
$$a \searrow \text{LCS (via)} \nearrow b$$

$$\text{heap} \xrightarrow[\text{(hp)}]{\text{2 deletion}} \underset{\text{(LCS)}}{\text{ea}} \xrightarrow[\text{(p)}]{\text{1 insertion}} \text{pea}$$

No of deletion = a length − LCS
No of insertion = b length − LCS

26. Longest Palindromic Subsequence

Problem Statement : S : agbcba
O/P = 5

5 ← longest
- abcba ✓
- bcb ✓
- b ✓

O/p → 5 (abcba)

2) Is it LCS?

		I/p	Q	O/p
LCS		a: b:	LCS	int
Q		a:	LPS	int

2/3   1/3

LCS ?

LCS   I/p   a, b

LPS   I/p   a   b = func(a)
(hidden
string / redundant)

"agbcba"

↓

LPS

↙

a → agbcba

b → reverse(a) abcbga

↓

LCS

abcbga

agbcba

> abcba

LPS(a) ≡ LCS (a, reverse(a))

LPS(agbcba) → return (agbcba,

28. Minimum no. of deletion in a string to make it palindrome

a : "agbcba"

O/P = 1

S : agbcba

a	g	b	c	b	a

↓ No of deletions (Minimize)

→ New string (palindrome)

~~ag~~bcb~~a~~

bcb (palindrome) 3

agcba (not palindrome)

agbcba

~~ag~~bcb~~a~~ — bcb (3) (LPS)

~~agb~~c~~ba~~ — c (5) (LPS)

a~~g~~bcba — abcba (1) LPS

Minm.

$\uparrow$ length of LPS $\propto \frac{1}{\text{no of deletions}} \downarrow$

LPS

$\downarrow$ minm no of deletions.

$\longrightarrow$ Palindromic subsequence.

length of LPS $\uparrow$ $\propto \frac{1}{\text{no of deletions} \downarrow}$

(Longest palindromic subsequence)

agbcba

$\downarrow$

LCS(s, reverse(s))

$\downarrow$

abcba (LPS)

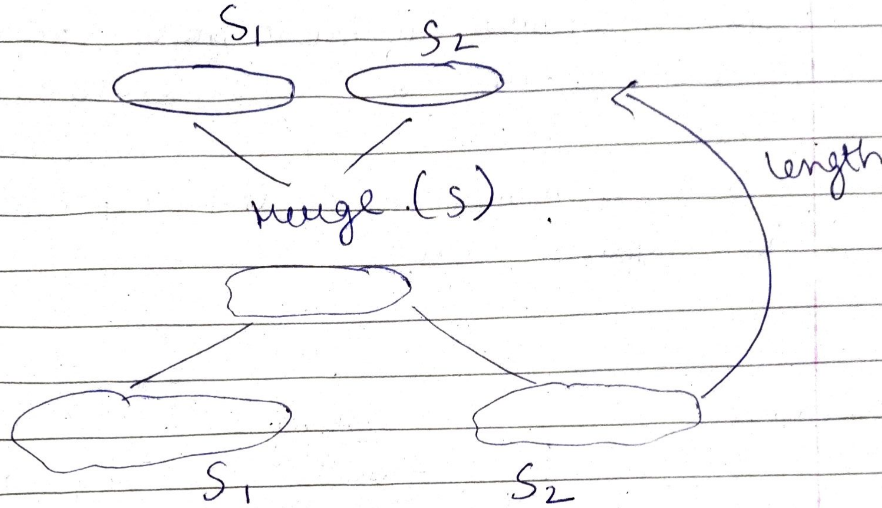
Min no of deletion = $\underset{\text{(length)}}{S}$ − LPS(length)

29. Print Shortest Common Subsequence

i/p a: acbcf
b: abcdaf

O/p: acbcdaf

SCS



acbcf   abcdaf

worst case

acbcdaf → print the whole string

m+n

acbcf abcdaf

↓

Now

m+n − LCS.

LCS $\xrightarrow{\text{(similar)}}$ SCS

Print LCS   Print SCS

	φ	a	b	c	d	a	f
φ	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
f	0	1	2	3	3	3	4

→ LCS

LCS → common ni hai to move kar jao, common hai to print karlo.

SCS → common hai to ek bar print kro else print karo.

LCS.

```
string s = " ".
while (i>0 && j>0){
  if (a[i-1] == b[j-1]){
    s.push_back(a[i])
    i--;
    j--;
  }
else{
  if(t[i][j-1] > t[i-1][j])
    j--;
  else if (t[i-1][j] > t[i][j-1])
    i--;
}
```

SCS

```
string s = " ";
while (i>0 && j>0) {
  if (a[i-1] == b[j-1]) {
    s.push_back (a[i])
    i--, j--; }
  else {
    if (t[i][j-1] > t[i-1][j])
      s.pushback (b[j-1]);
      j--;
    else if( t[i-1][j] > t[i][j-1])
      s.push_back (a[i-1]);
      i--; }
while (i>0)
  s.push_back (a[i-1]); i--;
while (j>0)
  s.push_back (b[j-1])
  j--;
```

1) Now in SCS code we include the letter on moving at i-1 or j-1.

2) Now in (i>0 && j>0) we need to change because in case of LCS we don't need to stop at the topmost but in SCS we need to.

In case of LCS

	φ	a	b	f
φ	0	0	0	0
a	0			
c	0			
b	0			
d	0			

	φ	a	b	f
φ	0	0	0	0
d	0			
f	0			
b	0			
d	0			

Some can't stop here

ac
" "] LCS (" ")

ac
" "] SCS ≠ (ab

Longest Repeating Subsequence

Str = "AABEBCDD"
O/p = 3

Problem Statement - Find a subsequence which is repeating

A A B E B C D D → order ✓
→ cont X

ABD

ABD

ABD (2X)
AB (2X)] longest → "ABD"
O/p = 3

S = A A B E B C D D

E → 3
C → 5

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

LCS = A A B E B C D D.

A A B B D D
once → ans

E & C are occuring once.
letter at the same index don't take

E → 3 → i, j $i == j$ ✗
C → 5.

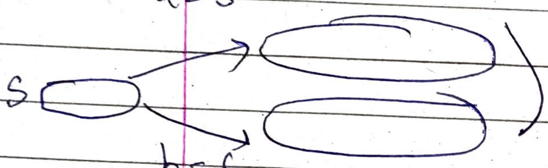
A → 0, 1   B → 2, 4
A → 0, 1   B → 2, 4

same index   diff index

So don't take A of same index take from other index.

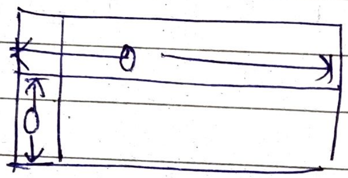
Sum up:

a = S



b = S.

where LCS ($i == j$) ✗



We can't take same letter for both the string during LCS i.e. ($i == j$)

```
if (a[i-1] == b[j-1] && i != j)
    t[i][j] = t[i-1][j-1] + 1
else
    t[i][j] = max(t[i][j-1],
                  t[i-1][j])
```

31. Sequence Pattern Matching

a = "AXY"   O/P → T/F
b = "ADXCPY"

Problem statement: Is string "A" a subsequence of "B"

a : AXY   b : ADXCPY

b : ADXCPY
a : AXY

O/p → True.

LCS : AXY

I/p

LCS: $\boxed{\begin{matrix} a \\ b \end{matrix}}$ → $\boxed{\text{sequence}}$ → $\boxed{\text{int}}$

Q: $\boxed{\begin{matrix} a \\ b \end{matrix}}$ → $\boxed{\text{sequence}}$ → $\boxed{\text{T/F}}$

I/p

b : ADXCPY   b : ADXCPY
a : AXY   a : AXYZ
LCS : AXY (LCS = = a)   LCS : AXY (LCS != a)

if (LCS = = A) return true
else
    return false

Can it be done using length. ?

a, b → LCS

a : AXY (3)
b: ADXCPZ (6)

m n

LCS = 0 to min(m,n)

LCS = 2

if (LCS == a.length())
return true
else
return false.

32. Min^m no of insertions in a string to make it palindrome

I/p: S : "aebcbda"
O/p → 2

x a e b c b d a x
d c

adebcbeda. x adebcbedax
(2) (4)

Minimum = 2

Min^m no of deletion to make a string palindrom.

a e b c b d a

aea abcba
(4) (2)

Date
Page

S.length() → LPS. (deletion)

S : 'aebcbda'

→ Palindromic string

$\boxed{a}$ e $\boxed{bcb}$ d $\boxed{a}$

e × { delete
d × { e & d

e ✓ { insert e & d }
d ✓ { & make their pair

no of insertions = no of deletions
||
S.length − LPS.

Deletion → single (Pair) → ×
Insertion → Single (Pair) → ✓

33. MCM (Matrix Chain Multiplication)

1) MCM
2) Printing MCM
3) Evaluate Exp. to True / Boolean Parenthesization
4) Min / Max value of an Expr.
5) Palindrome partitioning
6) Scramble string
7) Egg Dropping problem

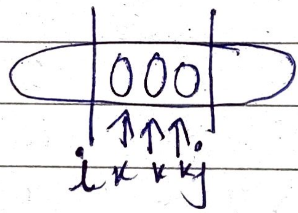
MCM format

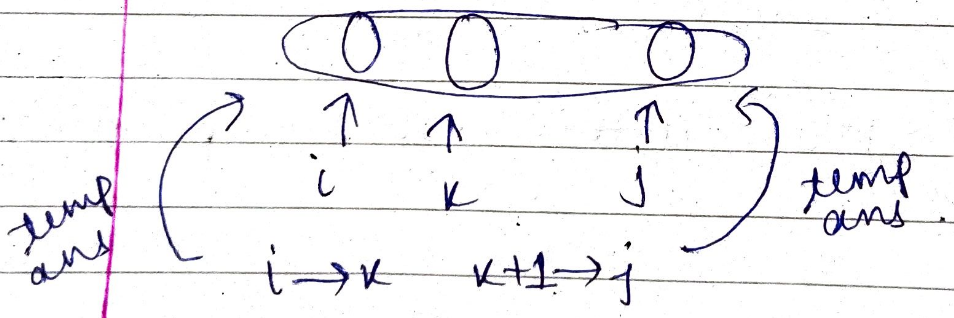
1) MCM
2) Printing MCM
3) Evaluate Expr to True / Boolean parenthesization
4) Min / Max value of an Expr
5) Palindrome partitioning
6) Scramble string
7) Egg dropping problem

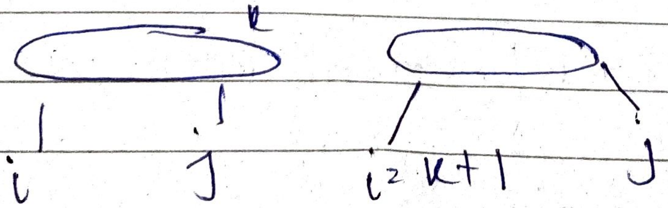
Identify → MCM → basic format

Identification + Format

MCM → string or → array



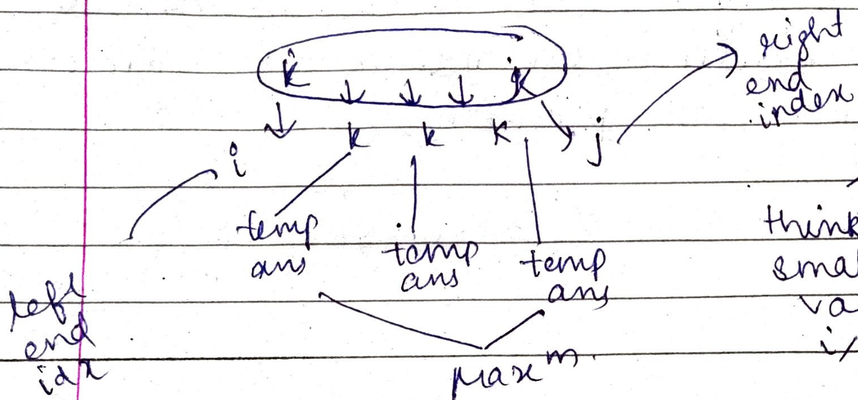




for (i, j)

temp — for (i, k)   for (k+1, j) — temp

ans ← f (temp ans)



Base cond^n

think of smallest valid i/p   think of invalid i/p

```
int solve ( int arr[], int i, int j)
{
    if (i>j)          →   this may be diff acc^n to ques
    return 0;

  for (int k=i ; k<j ; k++)
   {
     // Calc. temp ans
       temp ans = solve(arr, i, k) +
                        Solve (arr, k+1, j);

       ans =  func (temp ans)
   }
```

MCM (Recursive)

Problem Statement : arr[] = {40, 20, 30, 10, 30}.

$m \times n$ $n \times m$ $n \times n$

A_1, A_2, A_3, A_4

Some matrix are given like A_1, A_2, A_3, A_4 we have to multiply the matrix to reduce the cost (no of multiplications).

$b = c$ (then only we can multiply)

$[\ ]_{2 \times 3}$ $[\ ]_{3 \times 6}$

$a \times b$ $c \times d$

$$\begin{array}{ccc} 2 \times & 3 & 3 \times 6 \\ \downarrow & \downarrow & \downarrow \\ 2 & 3 & 6 \end{array} = 36 \text{ cost (no of multiplications)}$$

$A_1\ A_2\ A_3\ A_4$

dimension [] = {x, y, z, w}.

Date ______
Page ______

$A_1 \quad A_2 \quad A_3 \quad A_4$

$(A_1(A_2A_3))A_4 \qquad ((A_1A_2)(A_3A_4)) \qquad A_1(A_2(A_3A_4))$

$C_1 \qquad\qquad C_2 \qquad\qquad C_3$

$A \rightarrow 10 * 30$
$B \rightarrow 30 * 5$
$C \rightarrow 5 * 60$

$(AB)C = 10 * 30 * 30 * 5$

$10 * 5 \quad\quad 60$

$(AB)C = (10 * 30 * 30 * 5)\ 5 * 60$

$= 10 * 5 * 30 * 5 * 60$

$=$

$10 * 30 * 5 * 60 * 10 * 5$

$= 4500$

$A_1A_2A_3 \quad A4$

do parenthesiation in such a way cost is the least

$A_1(A_2A_3)A_4$ C_1
$(A_1A_2)(A_3A_4)$ C_2
$A_1(A_2(A_3A_4))$
C_3

$A(B\ C) \rightarrow$ min cost

$30 * 5 \quad 5 * 60$

$BC \rightarrow 30 * 5 * 60$

$A * BC \rightarrow 10 * 30 \quad 30 * 60$ +

$10 * 30 * 60$

cost

Bracket lagane par alag alag cost aa rahi hai.

0 1 2 3 4

arr[ ] : {40, 20, 30, 10, 30}

arr size n — n-1 matrix

$A_1 \rightarrow 40 * 20$
$A_2 \rightarrow 20 * 30$
$A_3 \rightarrow 30 * 10$
$A_4 \rightarrow 10 * 30$

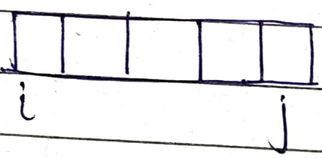
$A_1 * A_2 * A_3 * A_4$

Cost (minm)
⇓
(no of multiplication should be less)

$A_i \rightarrow arr[i-1] * arr[i]$

$A[i] = arr[0] + arr[1]$

Next step → Format



$A_1\ A_2\ A_3\ A_4$

(dimension)
arr[ ]: |40|20|30|10|30|
0 1 2 3 4

↓

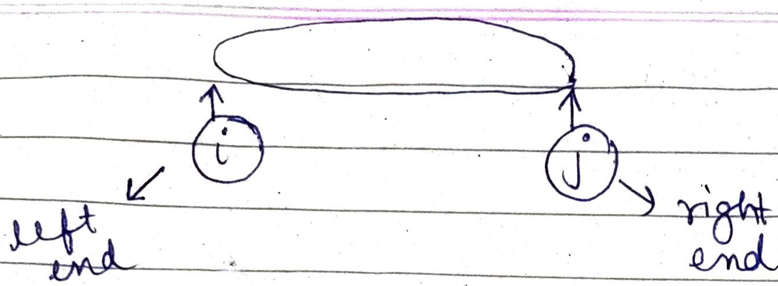
$(A_1)\ (A_2\ A_3\ A_4)$
↑
k

↓

Min cost + Min cost → temp ans

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

put brackets on different places to get the answer
Slide k at different positions.



40	20	30	10	30

i i j

Ai → arr[i-1] * arr[i]

when i=0 Ai → arr[-1] * arr[0] ✗

when i=1 Ai → arr[0] * arr[1] ✓

Aj → arr[j-1] * arr[j]

arr[3] * arr[4]

i>j → size=0

i==j → size=1

[(i-1) ni milega]

i = 1
j = n-1 } return cost

```
                       →1            n-1
int solve ( int arr[], int i, int j)
{
  if (i>=j)
    return 0;
    int mn = INT_MAX;
  for (int k=i ; k <=j-1 ; k++){
    int temp ans = solve (arr, i, k) +
                   solve (arr, k+1, j)
                 + arr[i-1] * arr[k] * arr[j]

  if(tempans)
    {
      mn= temp ans;
  }
```

return mn.
}

A B CD

AB → A_1 CD → B_1

40×20 20*30 30*10 10*30

A*B 40*20*30 30*10*30 C*D

A_1 * B_1

40*30 30*30

+ +

$\boxed{40*30*30}$

AB * CD

minimise

Steps for MCM

1) find i & j value
2) Find right Base cond^n.
3) Move K → i to j. (find K loop scheme)
4) Calculate K cost for P temp ans.

40 20 30 10 30

↑

K

i to K K+1 to j

i k j

40 20 30 10 30

↑

K

i to K (K+1) j → empty

Date
Page

i   j

40   20   30   10   30

k

i to k   k+1 to j

40 * 20
20 * 30   10 * 30.
30 * 10.

k = i   k = j - 1   (i → k   k+1 → j)

i   j

40   20   30   10   30

k

i to k-1   (k to j)

2 schemes.

k = i   to   k = j-1   i to k   k+1 to j
k = i+1   to   k = j   i to k-1   k to j

cost (extra)

0	1	2	3	4
40	20	30	10	30
i-1	i	k	k+1	j

for (i to k)   for (k+1 to j)

~~40 * 20 * 20 * 30~~   30 * 10   10 * 30

↓   ↓

solve (i to k)   40 * 20 * 30.   30 * 10 * 30
solve (k+1 to j)

40 * 20 * 30 * 30
↓
arr[i-1] ← 40 * 30 * 30 → arr[j]
→ arr[k]

dimension arr: [ ]

MCM Bottom Up (DP)

(dimension) arr[ ] : [ ]

```
int dp[1001][1001]

memset( dp, -1 , sizeof(dp))

int solve( int arr[], int i, int j)

    if (i <= j)
        return 0;

    if (dp[i][j] != -1)
        return dp[i][j];

    int mn = INT_MAX;
    for (int k = i; k<j; k++) {

    int temp_ans = solve(arr, i, k) +
                   solve (arr, k+1, j) +
                   arr[i-1] * arr[k] * arr[j];

    if (temp_ans < mn)
        mn = temp_ans;
    }

    dp[i][j] = mn;
}   return mn;
```

~~Palindrome Partitioning~~

```
int dp[1001][1001];

class solution {
public:
    int solve(int arr[], int i, int j) {
        if(i>=j)
            return 0;

        if(dp[i][j] != -1)
            return dp[i][j];

        int mn = INT_MAX;
        for(int k = i; k<= j-1; k++){
            int temp = solve(arr, i, k) + solve(arr, k+1, j)
                        + arr[i-1]* arr[k] * arr[j];
            mn = min(temp, mn);
        }
        dp[i][j]= mn;
        return dp[i][j];
    }

    int matrixmult(int N, int arr[]) {
        memset(dp, -1, sizeof(dp));
        int ans = solve(arr, 1, N-1);
        return ans;
    }
};
```

Palindrome partitioning

1) Problem statement

S : nitik → it is not as a whole is a palindrome

o/p : 2

divide string in such a way all the strings are palindrome.

① ②
nitik

n iti n

minimise the partition (2 partitions)

n i t i k

4 partitions

↳ minm i.e = 2

nitik

X

0

not (palindrome)

worst case partition : n-1

nitik

i j

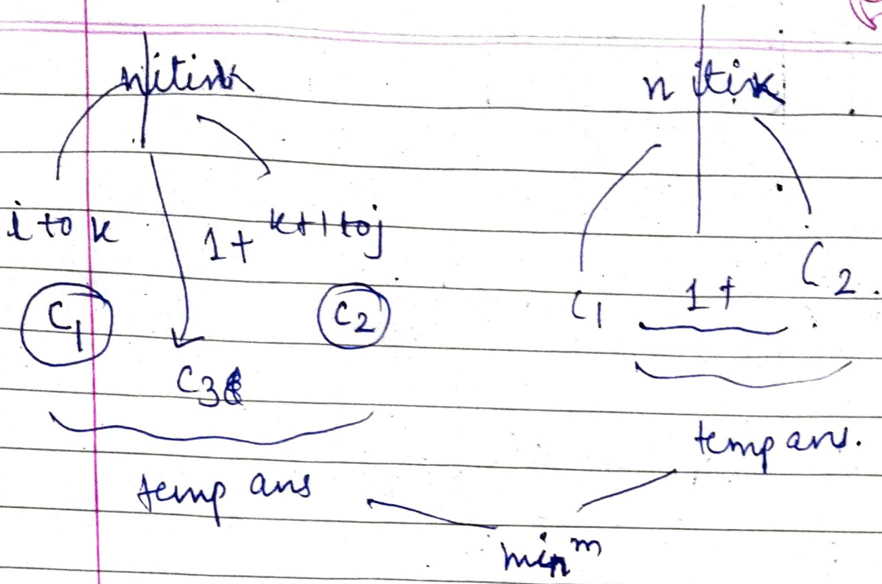
K

n (no further partition needed as we got a palindromic string)

itin

iti

n



Format

1. Find i & j
2. Find B C.
3. Find K loop (scheme)

4. temp ans → min^m

Recursive code

n i t i K

i j

i = 0 j = n-1 (no need to start i from 1 since it doesn't require i-1)

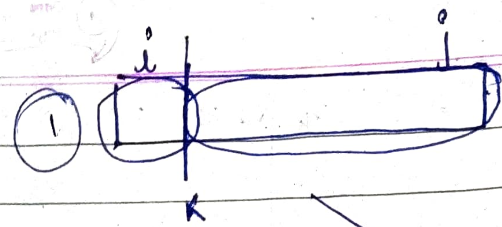
B.C

i == j size = 1

i > j size = 0 (if size is not given or 0 or equal no of partition would be 0 as string is empty) & if palindrome partition should be 0.

ispalindrome (s, i, j)

loop

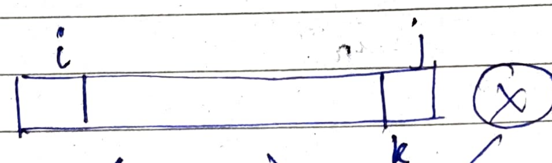


i to k   k+1 to j

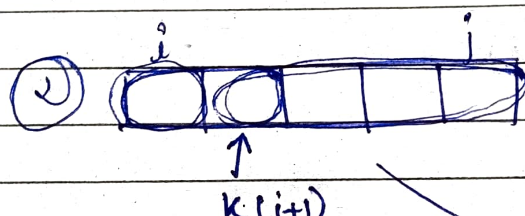
k = i   k = j-1

i to k   k+1 to j

$k \to i$, $k \to i+1$ $k \to j$, $k \to j-1$



i to k   k+1 to j ✗



~~i to k~~

i to k-1   k to j

k = i+1   k = j

1) k = i	k = j-1	i to k	k+1 to j
2) k = i+1	k = j	i to k-1	k to j

```
for (int k = i ; k <= j-1 ; k++)
{
    int temp = solve(s, i, k) + solve(s, k+1, j)
                + 1
```

i | j

n | i t i n

k

i to k | k+1 to j

"n" | "itin"

c_1 + 1 + c_2

```
    ans = min (ans, temp)
}
return ans;
}
```

Final code

```
int solve (string s, int i, int j) {
    if (i>=j)
        return 0;

    if( is palindrome (s, i, j) == True)
        return 0;
    int mn = INT_MAX ;

    for(int k=i ; k<=j-1 ; k++) {
        int temp = 1+ solve (s, i, k) + solve(s, k+1, j);

        mn = min (mn, temp)
    }
    return mn;
}
```

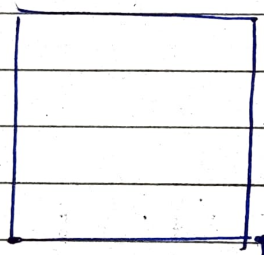
memset $\left\langle \begin{array}{l}-1\\0\end{array} \right\rangle$ only allow 2 values

Date
Page

Palindrome partitioning (bottom up)(memoization)

i/p = nitin nitik
o/p = 0 2.

1) Initialise the matrix with -1.
2) Check if the value is $\begin{cases} -1 & \text{(value isn't evaluate} \\ \neq -1 & \text{value is evaluated so we store \& return the value.} \end{cases}$



nxm

i & j changes.

variables which changes basically

```
int dp [1001][1001];
memset (dp, -1, size of(dp));
int solve ( string s, int i, int j) {
    if( i>=j)
        return 0;

    if (is palindrome(s, i, j))
        return 0;
```

```
if (dp[i][j] != -1)
    return dp[i][j];
int mn = INT_MAX;
for (int k = i ; k <= j-1 ; k++ ) {
int temp = solve(s, i , k) + solve(s, k+1, j) + 1
mn = min (mn, temp);
}
dp[i][j] = mn;
return mn;
}

int palindromepart( string s) {
  int n = s.length() - 1;
  int ans = solve( s, 0, n);
  return ans;
}

bool ispalindrome (string s, int i, int j) {

if (i == j)
    return True;

if (i > j)
    return True;

while (i < j)
if (s[i] != s[j])
    return false
else
    i++;
    j--;
```

Geeksforgeeks = ✓
Interviewbit = X

→ Further optimization

Why the above code is not most optimized?

Since we are calling

int temp = 1+ solve(s, i, k) + solve(s, k+1, j)

RC (left)   RC (right)

There is a possibility, might be one of the RC is solved or called.

The code will remain same but the diffnce is

int temp = 1+ solve(s, i, k) + solve(s, k+1, j)

```
if (t[i][k] != -1)
    left = t[i][k]
else
    left = slove(s, i, k)
    t[i][k] = left

if (t[k+1][j] != -1)
    right = t[k+1][j]
```

```
else
    right = solve(s, k+1, j)
        t[k+1][j] = right.

int temp = 1 + left + right;
```

Evaluate Expression to True
Boolean parenthesization

String : T or F and T.
O/p : 2

Problem statement : A string is given. String might have some characters like T → true F → false

String →
- T (True)
- F (False)

char +

- & (AND)
- | (OR)
- ^ (XOR)

symbols

How to put bracket such that the expression evaluates to true.

Find the no of ways in which when bracket is put it evaluates to True.

i/p : "T|F&T^F"

no of ways

(( )) ( ) → T
( ) ( ) → F
( ) ( ) → T
( ) ( ) → T

3 ways.

In MCM we put brackets for min^m cost & in this also we do the same.

(T|F & T^F)

k-1 k k+1

We need to break bracket on operator

Expr^n operator Expr^n

4 steps:

1) find i & j
2) find base cond^n
3) Find k loop
4) temp ans → func^n

↓

Main ans.

i j

T | F & T ^ F

1) i = 0
j = str.length() - 1

2) BC

i j

(T or F and T) xor (F)

i to k-1 k k+1 to j

(left) Exprⁿ1 XOR Exprⁿ2 (right)

i to k-1 k k+1 to j

T^T = False
F^F = False
T^F = True
F^T = True

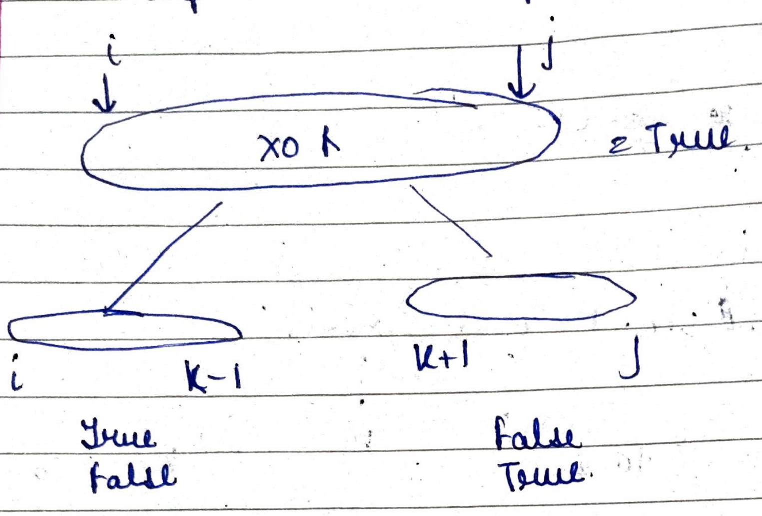
no of true ways = lT * rF + lF * rT

T^F = T
F^T = T

exprⁿ XOR exprⁿ

no of ways (True) → 2 * 4 ← no of ways false

Expr^n XOR Expr^n




```
int solve ( string S, int i, int j, boolean isTrue
                                            or
                                            isFalse )
```

Base cond^n

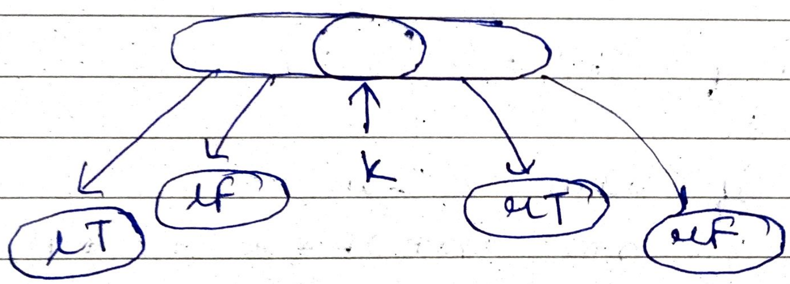
```
i                          j
T or F and T xor F
  T | F & T ∧ F

boolean isTrue

if (i > j) return false
if (i == j)
   if (isTrue == True)
        return S[i] == 'T'
   else
        return S[i] == 'F'
```

loop

```
i             j
T | F & T ^ F
  ↑   ↑   ↑
  k  k=k+2  k=j-1
 i+1
```

for (int k = i+1 ; k<= j-1 ; k= k+2)
int ans = 0;
int lT = (S , i , k-1, T)
int lF = (S , i , k-1 , F)
int rT = (S , k+1 , j , T)
int rf = (S , k+1 , j , T)

$\Big\}$ temp ans .



$\underbrace{(\quad)}_{\text{left}}$ XOR $\underbrace{(\quad)}_{\text{right}}$

left → lT, lf

right → rT, rf

ans + = lT * rf + lf * rT

```
if (s[k] == '&')
{
if (isTrue == True)
  ans = ans + lT * rT
else {
ans = ans + lF * rT + lT * rF + lF * rF;
}
else if (s[k] == '|') {
  if (isTrue == True)
    ans = ans + lT * rT + lT * rF +
                      lF * rT.
  else
     ans = ans + lF * rF.
}

else if (s[k] == '^')
{
if (isTrue == True)
   ans = ans + lF * rT + lT * rF
  else
   ans = ans + lT * rT + lF * rF
  }
return ans;
}
```

whole code

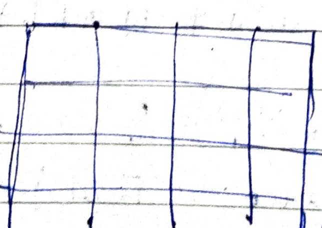
```
int solve (string s ,int i, int j , bool isTrue ){

    if(i>j) {
        return False; }
    if(i==j) {
        if(isTrue == True)
            return s[i] == 'T';
        else
            return s[i]=='F';
    }
    for( int k= i+1 ; k<=j-1 ; k+=2) {
        int ans =0;
        int LT  =  solve (s, i, k-1, T);
        int LF  =  Solve(s, i, k-1, F);
        int RT  =  Solve(s, k+1, j, T);
        int RF  =  solve(s, k+1, j, T);

        if (s[k] == '&') {
            if (isTrue == True)
                ans = ans + LT* RT ;
            else
                ans = ans + LF*RT+ LT*RF + LF*RF;
        }
        else if ( s[k] =='|') {
            if (isTrue == True)
                ans += LT* RT + LT* RF+ LF*RT;
            else
                ans += LF*RF;
        }
```

```
else if ( S[k] == '^' ) {
        if (isTrue == True)
            ans += LF * RT + LT * RF;
        else
            ans += LT * RT + LF * RF;
    }
}
return ans;
}
```

Evaluate Expression to True Boolean Parenthesization - (memoization)
(Bottom Up - DP)
3D dp

Recursive → Recursive call (RC)
Top down → table
Bottom up → RC + table

PS → put brackets in such a way that the expression evaluates to true.



i*j* isTrue
(3d)

```
int t [1001] [1001] [2]
```

isTrue

True   False

More better way: we can use map

map	value
i, j, isTrue	
i+" "+j+" "+ isTrue	

```
unordered_map < string, int > mp;

int main(){
    mp.clear()
    Solve( )
}
```

After Base condn:-

```
string temp = to_string(i);
    temp.push_back(' ');
  temp.append(to_string(j));
   temp.push_back(' ');
   temp.append(to_string(isTrue));

if(mp.find(temp)!= mp.end()){
        return mp[temp];
    }
```

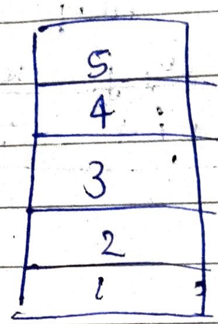
```
return mp[temp]= ans;
}
```

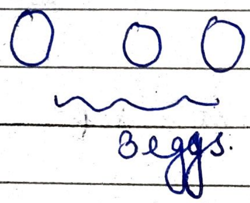
Egg Dropping Problem

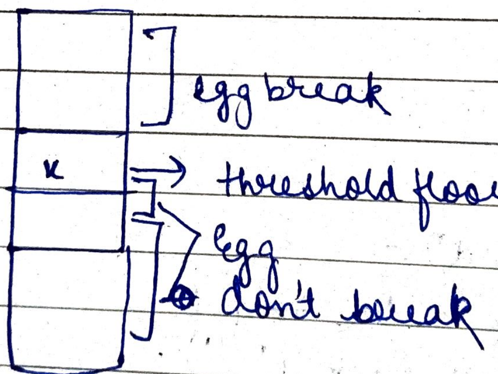
I/p : e = 3
f = 5

O/p : 3 → minimize no of attempts in worst case.

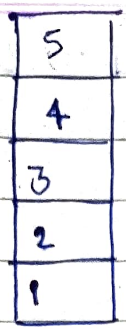
Problem statement :

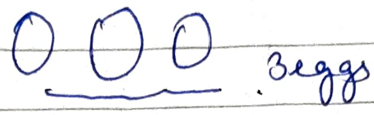




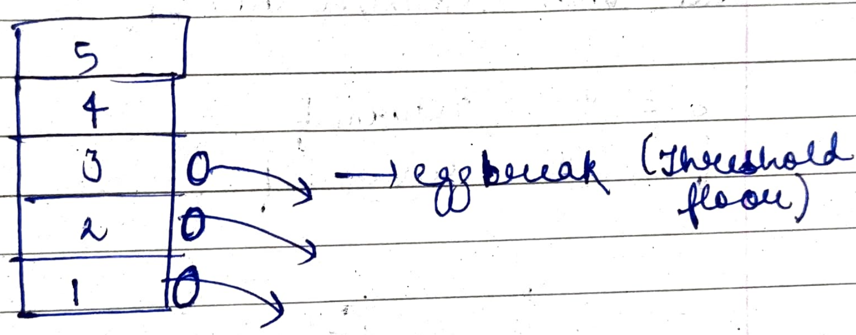


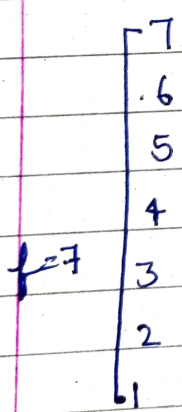
We are in a building, we need to find the minm no of attempts to find the critical floor.



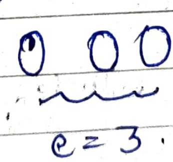


Safe strategy → worst case (1 egg) so drop from the last it won't break untill threshold.



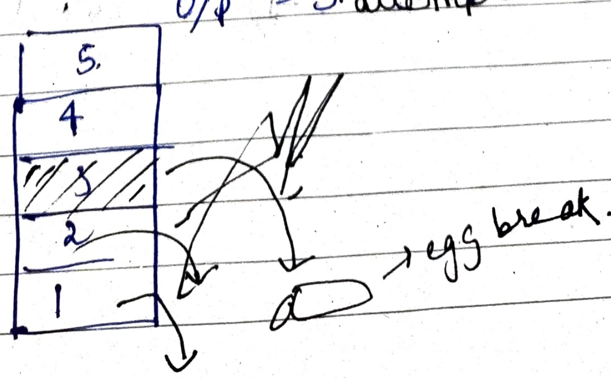


Worst-case → minimize no of attempts to find threshold floor.



I/p e=3
f=5

O/p = 3 attempts



(floor) f | 1 | 2 | 3 | 4 | 5 | 6 | 7 | → Building look like an array

i ↑ ↑ j

from where to take K?

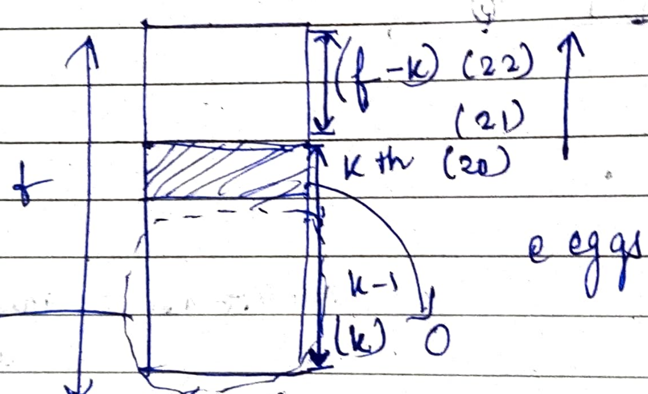
→ We will check for all K values

for K = 1 K = f K++

Base condⁿ (smallest valid i/p)

e = 1 return f

f = 0/1 return f



threshold floor lies in (K-1) floor

solve(e,f)

egg break

solve(e-1, K-1)

egg don't break

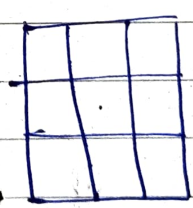
solve(e, f-K)

```
int solve (int e, int f)
{
    if (f == 0 || f == 1)
        return f;

    if (e == 1)
        return f;

    int mn = INT_MAX;

    for (int k = 1;   k <= f; k++)
    {                              for the worst case
        int temp = 1 +  max( solve (e-1, k-1),
                             solve (e, f-k));
        mn = min (mn, temp);
    }
    return mn;
}
```

Egg Dropping Memoization

$1 <= T <= 30$
$1 <= e <= 1000$
$1 <= f <= 1000$

```
// globally defined
int dp[1001][1001];
memset (dp, -1, sizeof(dp));
```



ex)

```
int solve ( int e, int f){
    if (e == 1)
        return f;
    if (f == 0 || f == 1)
        return f;
```

```
if (t[e][f] != -1)
    return t[e][f];
int mn = INT_MAX;
for (int k = 1 ; k <= f ; k++) {

    int temp = max(solve(e-1, k-1),
                   solve(e, f-k)) + 1 ;

    mn = min(mn, temp);
}
return t[e][f] = mn;
}
```

Further Optimization

Inside the loop.

```
if (t[e-1][k-1] != -1)
    int low = t[e-1][k-1]
else
    low = solve(e-1, k-1)
    t[e-1][k-1] = low;

if (t[e][f-k] != -1)
    int high = t[e][f-k];
else
    high = solve(e, f-k);
    t[e][f-k] = high;

int temp = 1 + max(low, high);
```