* Longest common substring

I/p - 2 strings

1 → a b c d e — m (length)

2 → a b f c e — n (length)

o/p → length of longest common substring

t[m+1][n+1]

	0	1	2	. . .	m+1
0	0	0	0	0	0
1	0				
2	0				
.	0				
n+1					

```
if (a[i-1] == b[j-1])
    t[i][j] = t[i-1][j-1] + 1
else
    t[i][j] = 0
```

change this in LCS code before. we discontinue if no match, ie- set to 0.

* Print longest common subsequence

I/p - a, b → 2 strings m, n - len(a), len(b)

o/p → LCS (string)

a = a b c d a f

b = a c b c f

	0	a	b	c	d	a	f
0	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

c != f, so this is max of ((i-1, j), (i, j-1))

f = f so this comes from (i-1, j-1) add to string s

S = "f c b a"

Iterate backwards in the matrix, and check from where the answer came from and add accordingly to our subsequence.

Step 1 → LCS (a, b, m, n) → we get t[m+1][n+1]

Step 2 → int i = m, j = n string s = " "

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

3 3

Step 3. Reverse s.

* **Shortest Common Supersequence.**

a = "geek"
b = "eke"

Supersequence of a & b → which contains both a and b as subsequence

eg. a A G G T A B
b G X T X A Y B

Supersequence: A G G T G X A B T X A Y B
(order should be same GX not XG)

A G G T A B
G X T X A Y B
↓
A G G X T X A Y B → shortest supersequence
9

Find Longest common subsequence (LCS) of a & b and remove that many chars from sum of len of a & len b.

⇒ $\boxed{m + n - LCS}$ ⇒ length of shortest supersequence

⇒ m + n − LCS(a, b, m, n)
⇒ 7 + 6 − 4 = 9

Minimum number of insertions and deletions to convert a to b

I/p - String a, String b
a : ~~h~~ e a ~~p~~
b : ~~p~~ e a
O/p - Insert - 1 , Delete - 2

no. of Insert = b.length() − LCS(a, b, m, n)
no. of delete = a.length() − LCS(a, b, m, n)

Longest palindromic subsequence

I/p - s : agbcba
o/p : 5. (abcba)

Reverse s

S1 = s
S2 = Reverse(s)

a g b c b a
a b c b g a

→ LCS(S1, S2, m, n) → answer

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

* Basically give largest palindromic string after the deletions

$$\text{length of LPS} \propto \frac{1}{\#\text{ of deletion}}$$

I/p s — agbcba

O/p => s-length − LPS(s).

= 6 − 5

= 1 => min no. of deletions.

Print shortest common supersequence (SCS)

I/p a : a c b c f len - m

b : a b c d a f len n

o/p − "acbcdaf"

SCS → m+n − LCS(a,b,m,n)

LCS → abcf → Print only once

a \ b	0	a	b	c	d	a	f
0	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

If a(i) == b(j)
  push it
else
  if t[i-1][j] > t[i][j-1]
    push b(j-1)
    i--
  else
    push a(j-1)
    j--

(similar to print lcs)

```
while (i>0 && j>0)
 {
   if ( a[i-1] == b[j-1])
      { s.push (a[i-1])
        i--
        j--
      }
  else
   {
     if ( t[i][j-1] > t[i-1][j]
        {
          s.push(b[j-1])
          j--
        }
     elseif  t[i-1][j] > t[i][j-1]
        {  s.push(a[i-1])
           i--
        }
   }

while (i>0)
{
```

```
    S.push( a[i-1])
  }
while(j > 0)
 {
   S.push (b[j-1])
  }
```

add remaining (eg if one string is empty)

Longest Repeating Subsequence.

Ip- Str: "A A B E B C D D"

subseq. A B C → 2x len 3

A B → 2x 2

B D → 2x 2

Op → 3. length of longest repeating subsequence

A A B E B C D D — should be diff indexes

	0	1	2	3	4	5	6	7
S1	A	A	B	E	B	C	D	D
S2	A	A	B	E	B	C	D	D
	0	1	2	3	4	5	6	7

S1 A → 0, 1 S1 E → 3

S2 A → 0, 1 S2 E → 3

should be on diff indexes.

Ip.. a = S len m

b = S len n

LCS (a, b, m, n) with 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]).
```

only this change in LCS.

Sequence Pattern Matching.

Is a a subsequence of b?

Ip- a: A X Y

b: A D X C P Y

Op - True/False

```
if (LCS(a,b,m,n) == a.length())
        return True
else return false.
```

Minimum no. of insertions in a string to make it a palindrome

i/p : s : "a e b c b d a"

Longest Palindromic Subsequence (LPS) $\propto \frac{1}{\text{Deletion}}$

no. of insertions = no. of deletions
= s.length() − LPS.

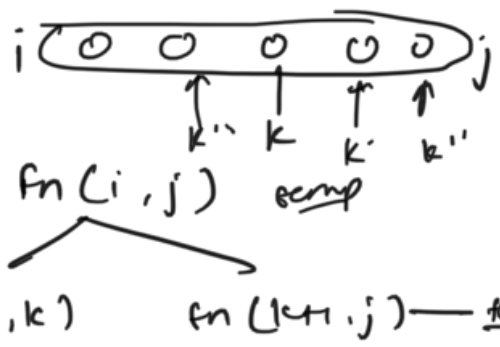
★ Matrix Chain Multiplication (MCM)

Related Problems

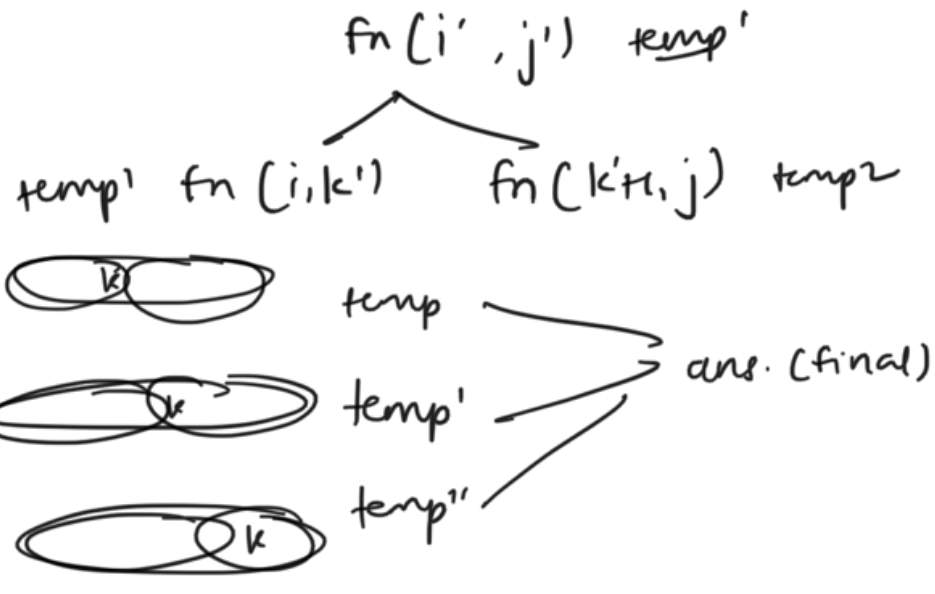
1) MCM
2) Print MCM
3) Evaluate expr to T/F Parenthesization
4) min/max value of expr
5) Palindrome partitioning
6) Scramble string
7) Egg Dropping Problem

MCM - Matrix Chain Multiplication

Identify-



Move from i to j, take each value as k and calculate temporary ans



Format-

left most valid idx

right most valid idx

```
int solve( int[] arr , int i, int j )  → int[] /String
{
    if (i > j) return 0;

    for(int k = i ; k < j ; k++)
    {
        //calculate temp ans
```

conditions might differ but overall format is same.

k++
k+2 -- etc

```
        tempans = solve(arr,i,k) + solve(arr, k+1, j)

            ans ← fn(tempans)
        }
    return ans
}
```

I/P · arr[] = {40 20 30 10 30}

Given diff matrix, minimize cost of their multiplication

$$\overset{A_1}{[\ ]_{2\times5}}\quad \overset{A_2}{[\ ]_{10\times20}}\quad \overset{A_3}{[\ ]_{30\times10}}\quad \overset{A_4}{[\ ]_{10\times5}}$$

$A_1 \times A_2 \times A_3 \times A_4$ — min cost

* Cost of matrix multiplication

$$[\ ]_{2\times\boxed{3}} \times [\ ]_{\boxed{3}\times6} = 2\times3\times6 = \underline{36}$$

Should be same

$((A_1\ (A_2\ A_3))A_4)$ — C_1

$(A_1\ A_2)(A_3, A_4)$ — C_2 — min cost

$A_1(A_2(A_3, A_4))$ — C_3

eg – A – 10×30

B — 30×5

C — 5×60

$(A\ B)\ C$

$10\times30\times5 +$

$10\times5\times60 = \underline{4500}$ (temp)

$A(B\ C)$

$30\times5\times60 +$

$10\times30\times60 = \underline{2700}$ (temp)

min(4500, 2700)

$\underline{2700}$

arr = [40 20 30 10 30] — dimensions of matrices

no of matrix = array size − 1

$= 5-1 = 4$

$A_i = arr[i-1] * arr[i]$

$A_1 = 40\times20$

$A_2 = 20\times30$

$A_3 = 30\times10$

$A_4 = 10\times30$

1 n−1

```
int solve ( int[] arr, int i , int j)
{
    if (i >= j)          ---> check for valid/invalid i/p to find
        return 0;                this condition

  // Scheme 1 - k = i to j-1.        (i,k) (k+1,j)
  // Scheme 2 - k = i+1.             (i to k-1) (k to j)
    int min = Int-max.
    for (int k = i ; k <= j-1; k++)
     {
    tempans =    solve (arr, i, k) +
                 solve (arr, k+1, j) +
                 arr[i-1] x arr[k] x arr[j]

      if (tempans < min)
            min = tempans          ---> Recursive
     }
  return min
}
```

Step 1 Find i, j
2. Find base case
3. Find k loop scheme
4. calculate ans from temp ans

★ **Bottom up from Recursive**

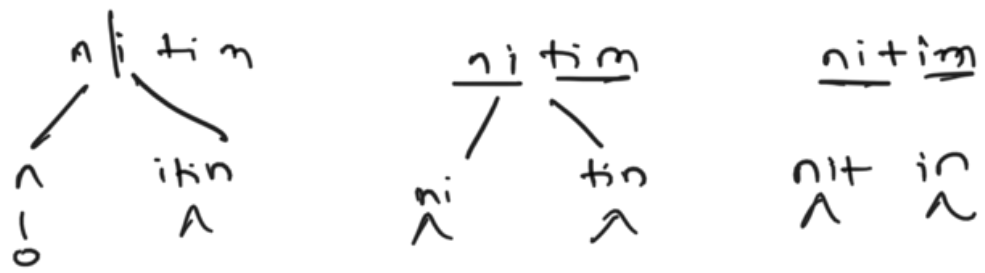
```
static int t[1001][1001]    - use i,j for t dimensions.
  Arrays.fill(t, -1)                // init with -1.
int solve ( int[] arr, int i, int j)
{
    if (i >= j) return 0;
    if (t[i][j] != -1) return   t[i][j];

    int min = int-max.
    for (int k = i; k < j ; k++)
     {
      int temp = solve(arr, i, k) + solve(arr, k+1, j)
                 + arr[i-1] x arr[k] x arr[j]

      if (temp < min)
            min = temp.
     }
  return t[i][j] = min;
```

Palindrome Partitioning

Q: I/p - s : n|iti|n

Minimum no. of partitions to convert each string in palindrome

O/p - 2



--and so on.

```
                       0        size-1
int solve(String s, int i, int j)
{
      if(i>=j) return 0;

      // check if palindrome already
       if(ispalindrome(s,i,j) return 0;

       //scheme 1   k=i to j-1     (i,k) (k+1,j)
       //scheme 2   k=i+1 to j     (i,k-1) (k to j)
       int min= Int-MAX
        for(int k=i; k<=j-1; k++)
      {
      int tempans=  solve(s,i,k) + solve(s,k+1,j)+1

        if(tempans<min)
                 min=tempans
      }
      return   min;
}
```

* Palindrome Partitioning Bottomup(Memoization)

```
                   1001 1000
int[] t = new int[M][j]            0<i,j<=1000
Arrays.fill(t,-1)
int solve(int[] arr, int i, int j)
{
      if(i>=j) return 0;
      if(ispalindrome(s,i,j) return 0;
       if (t[i][j]!=-1) return t[i][j];
       int min = Int-MAX
       for(int k=i , k<=j-1  ; k++)
       {
         int temp = 1 + solve(s,i,k)+ solve(s, k+1,j)
         If (temp<min) min=temp;
```

```
    }

    return t[i][j]=min
}
```

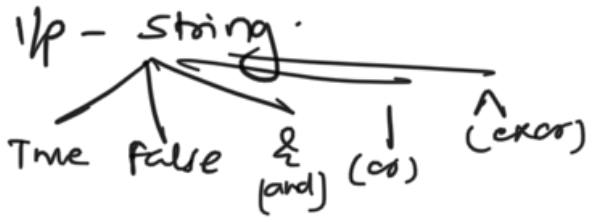
* palindrome partition optimized

change for loop

```
for( int k=i, k<=j-1 ; k++)
  {
    if (t[i][k] != -1)
            left = t[i][k]
      else  left = solve(S,i,k)

    if ( t[k+1][j] != -1)
            right = t[k+1][j]
      else  right = solve(S, k+1,j)

  int temp = 1 + left + right .

  }
```

* Evaluate expression to True Boolean Parenthesization (Recursive)

I/p - string.



find no. of ways to put parenthesis so that the string evaluates to true

S = " T | F & T ^ F " → True (no of ways)

→ k+=2 → why? you need 2 operators to evaluate an expr

solve (S, i, j, isTrue) isTrue :- True/False as answer

Step 1: find i & j

i=0 j = strlen -1

Step 2 : find Base condn.

```
if (i>j) return False.
if (i==j)
  {
   if (isTrue == True)
          return s[i] == 'T'
   else
          return s[i] == 'F'
  }
```

Step 3 : k loop

T or F and T exor F $(i, k-1)\ (k+1, j-1)$

k-1 k k+1

lt rt left true = lf . and so on

lf rf

```
int ans = 0
 for ( int k = i+1 ; k ≤ j-1 ; k=k+2 )
 {
    int lt = solve(S, i, k-1, T)
    int lf = solve(S, i, k-1, F)
    int rt = " (S, k+1, j, T)
    int rf = " (S, k+1, j, F)

    if (S[k] == '&')
 {
     if (isTrue == true)
        ans = ans + lt * rt
     else
        ans = ans + lf x rt + lf x rf + lt x rf
   }
  else if (S[k] == '|')
   {
      if (isTrue)
        ans = ans + lt * rf + lt x rf + lf x rt
      else
        ans = ans + lf x rf
    }
   else if (S[k] == '^')
    {
      if (isTrue)
        ans = ans + lf x rt + lt x rf
      else
        ans = ans + lt x rt + lf x rf
    }
 }
 return ans;
```

XOR

T	T	F
T	F	T
F	T	T
F	F	F

★ Bottom up (memoization) of above problem

changing variables i, j, k = 3

⇒ 3D matrix for a map i,j,k → val.
key — i+" "+j+" "+k

t[100][100][2]
i j isTrue
range of i,j

code

```
if (i > j) return True
if (i == j)
{
    if (isTrue)
        return s[i] == 'T'
    else
        return s[i] == 'F'
}
String key = i + " " + j + " " + k
if (map.get(key))
        return map.get(key)

else
 step3 code
 {
 }
return map.put(key, ans);
```

Scrambled string

a = "great"
b = "rgeat"
o/p - True/False.

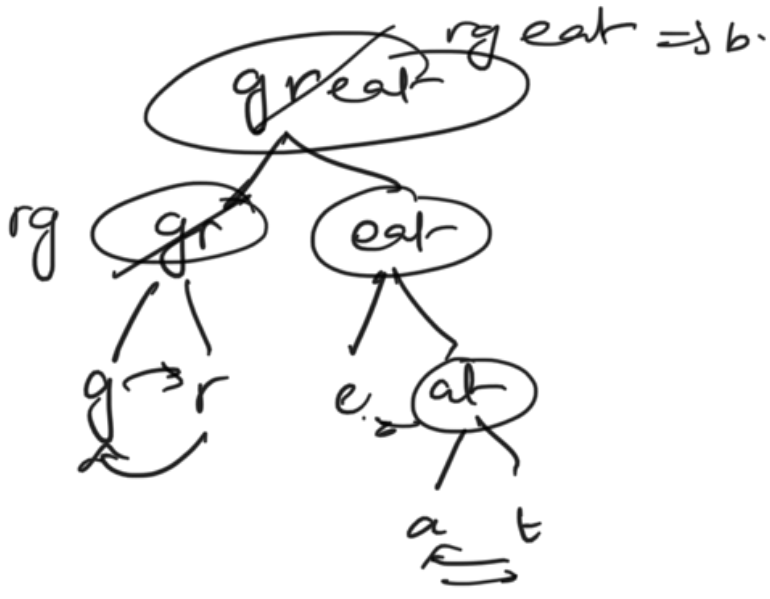
Q. Are a and b scrambled strings?

Scrambled string

- Represent as binary tree
- no node can be empty string

We can swap order of leaf nodes zero or more times.

Can we replace non leaf node in a to get b? → yes → scrambled



partition i = 1 to n-1

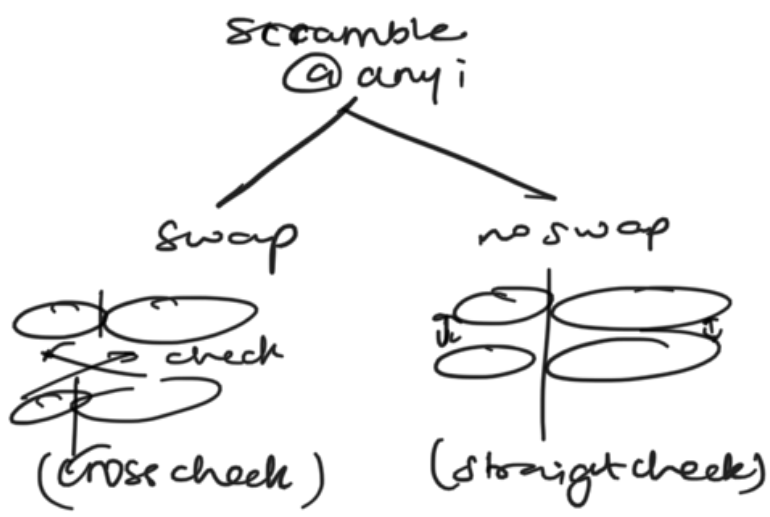
n = length

∴ MCM k from 1 to n-1

great great → scrambled

grkat eatgr → —"—

swap i=2




```
bool solve (string a, string b)
{
   if (a.compare(b) == 0)
            return True
   if (a.length <= 1)
            return false

   int n = a.length()
   boolean flag = false;

   for (int i = 1 ; i <= n-1 ; i++)
   {
    if (condition1 || condition2)
      {
         flage = true
            break;

      }
   }
```

return flag

}

condition 1.

length

solve(a.substr(0, i), b.substr(n-i, i)) == True

&&

solve(a.substr(i, n-i), b.substr(0, n-i)) == True.

condition 2

(solve(a.substr(0, i), b.substr(0, i)) == True

&& (solve(a.substr(i, n-i), b.substr(i, n-i)) == True

Scrambled string Bottom Up.

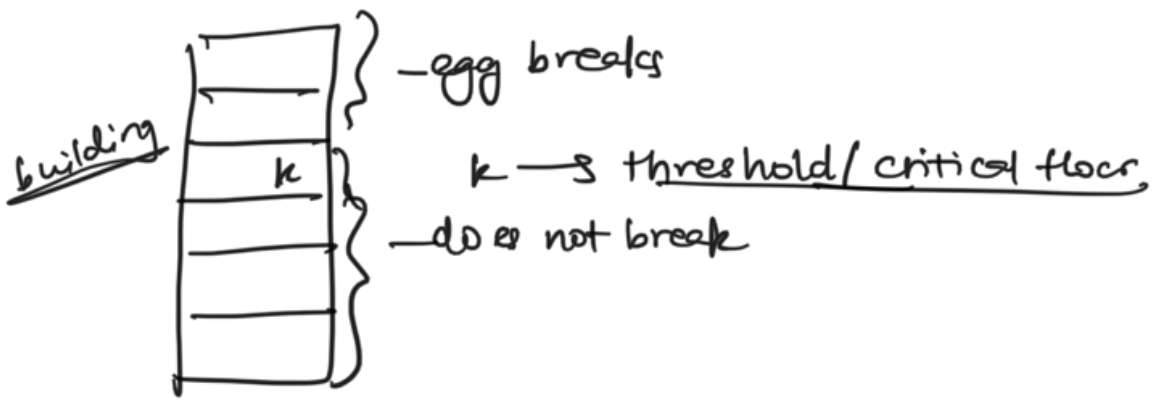
```
boolean solve( string a , string b)
{
  if(a.compareto(b) == 0)
          return True
  if (a.length() <= 1)
          return false

  string key = a + " " + b;
  If (map.get(key))
          return map.get(key)      ─ extra

  int n = a.length
   boolean flag = false

  for(i = 1 to n)
   {
    if (condition1 || condition 2)
     {
       flag = true
        break;
     }
   }
  map.put(key, flag)   ─ extra.
   return flag
```

Egg Dropping Problem

I/P: e = 3 (eggs)

f = 5 (floor)

op — s (int) Minimum no. of attempts.



Q. Minimum no. of attempts to find critical floor.

Use eggs wisely.

Best - start from bottom until egg breaks.

worst case — k is top floor.

1. Find i, j
   0 to n-1.

2. Find Bc ① e=1 op-f ② f=0/1 -return f.
3. k loop (1 to f)
4. ans ← temp ans.

solve(e,f)

break ✓ solve(e-1, k-1)

does not break ✗ solve(e, f-k)

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

  if(e==1)
        return f

  int min = INT.MAX
  for(int k=1; k<=f; k++)
```

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

    min = Math.min(min, temp)
}

return min;
```

Egg Dropping Bottom up

```
static int t[e+1][f+1]
Arrays.fill(t, -1)
int solve(int e, int f)
{
  if(f==1 || f==0) return f
  if(e==1) return f

  if(t[e][f] != -1) return t[e][f]

  int min = ∞
  for(int k=1, k≤f; k++)
  {
    temp = 1 + max(solve(e-1, k-1), solve(e, f-k))

    min = Math.min(min, temp)
  }
  t[e][f] = min
return t[e][f].
```

Optimized

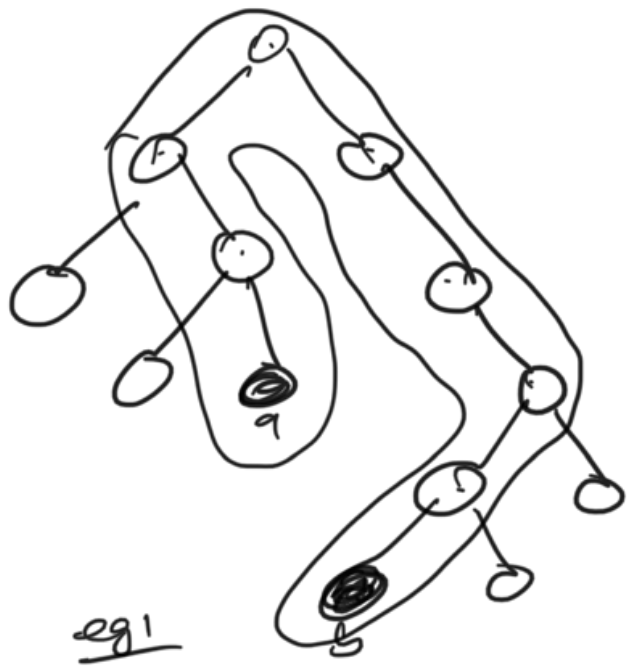
```
if(t[e-1][k-1] != -1)
    low = t[e-1][k-1]
else { low = solve(e-1, k-1)
       t[e-1][k-1] = low }
if(t[e][f-k] != -1)
    high = t[e][f-k]
else
    high = solve(e, f-k)
    t[e][f-k] = high
tmp = 1 + max(low, high)
min = min(tmp, min)
```

Dynamic Programming on Trees.

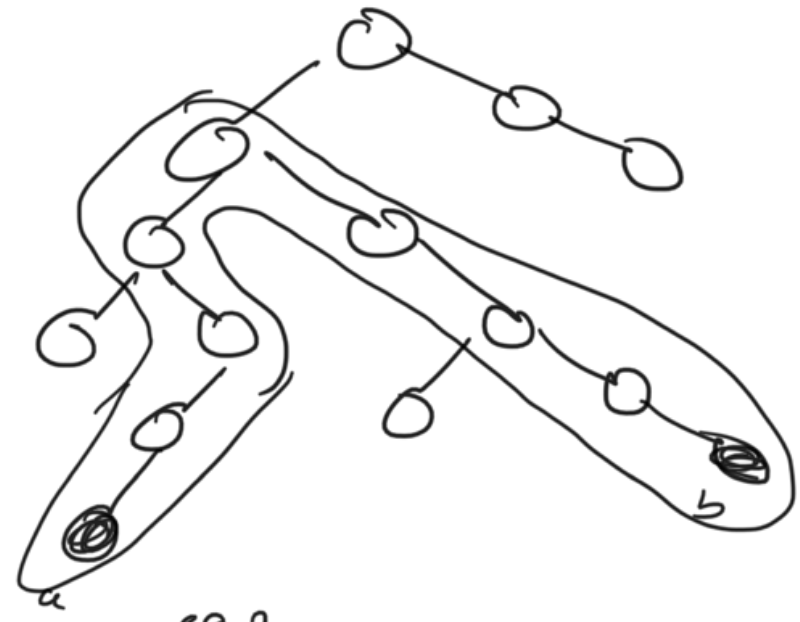
- General Syntax
- How DP can be applied on Trees (Identification)
- Diameter of a Binary Tree ——— (1)
- Max. Path Sum from any node to any ——— (2)
- max path sum from leaf to leaf ——— (3)
- Diameter of n-ary tree ——— (4)

Identification

diameter of tree → longest path b/w two leafs.



eg1



eg. 2

Longest path will not always pass through root node as in eg. 2

left tree height + right tree height +1 → in case it pass through root

$O(n)$.

else calculate with that node as root.

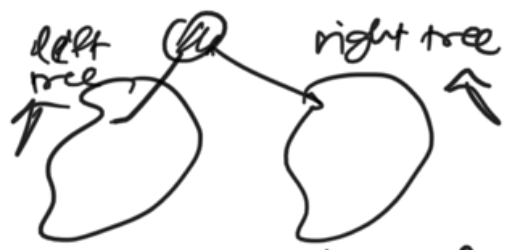
$O(n) \times n \simeq O(n^2)$

IMP

** If you want to traverse every node in tree and each node has some operation in $O(n)$ on its own (ie- $O(n^2)$) then u can use DP.

General syntax

```
int fnname (-i/p-)
{
  Base case
  HYPOTHESIS ——— ?
                 Don't question how it comes
  INDUCTION ——— combine left & right answer to get ans for root.
}
```




```
int fnname (i/p)
{
  BC                  →  if (root == null) return 0;   (Problem dependent).
  Hypothesis          →  int l = solve(root.left, res)
                         int r = solve(root.right, res)
  induction           →  int temp = calculate temp ans
                         (pass up)        (1 + max(l, r))
                         int ans = max(temp, relation)
                         (return)                 (1 + l + r)
                         res = max(res, ans)
}

res = Int.min
int solve(Node * root, int & res)   // addr
{
  ...
  return temp;
```

(either you have answer, or you pass it on to upper nodes).

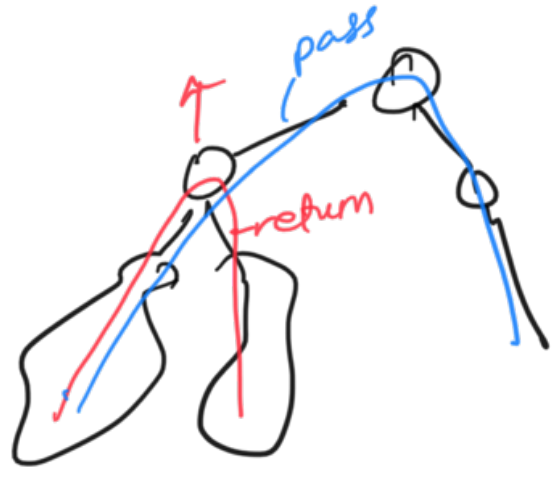
Diameter of Binary Tree.

- longest dist b/w two leaf nodes

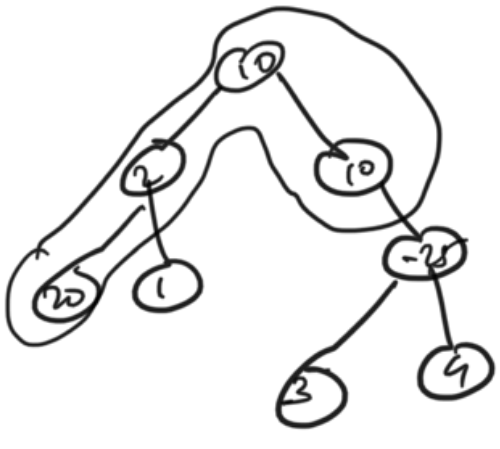
I/p - tree

O/p - no. of nodes in longest path - Integer.

```
int solve (node * root, int & res)
{
    if(root == null) return 0;

    int l = solve( root.left, res)
    int r = solve(root.right, res)

    int temp = max(l, r)+1
    int ans = max(temp, (l+r+1))

    res = max (temp, ans)
    return temp;
}

int main()
{
    int res = int_min
    solve (root, res).
    return res;
}
```



Maximum Path Sum → from any node to any node

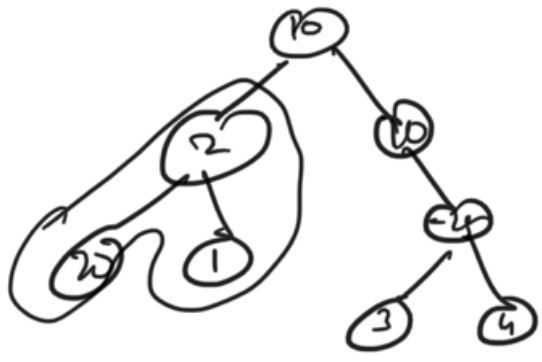


$20 + 2 + 10 + 10 = 42$

only change 2 lines in above code

```
int temp = max (max(l,r) + root.val , root.val)
int ans = max(temp, l + r + root.val)
res = max(res,ans)
```

Maximum path sum from leaf to leaf



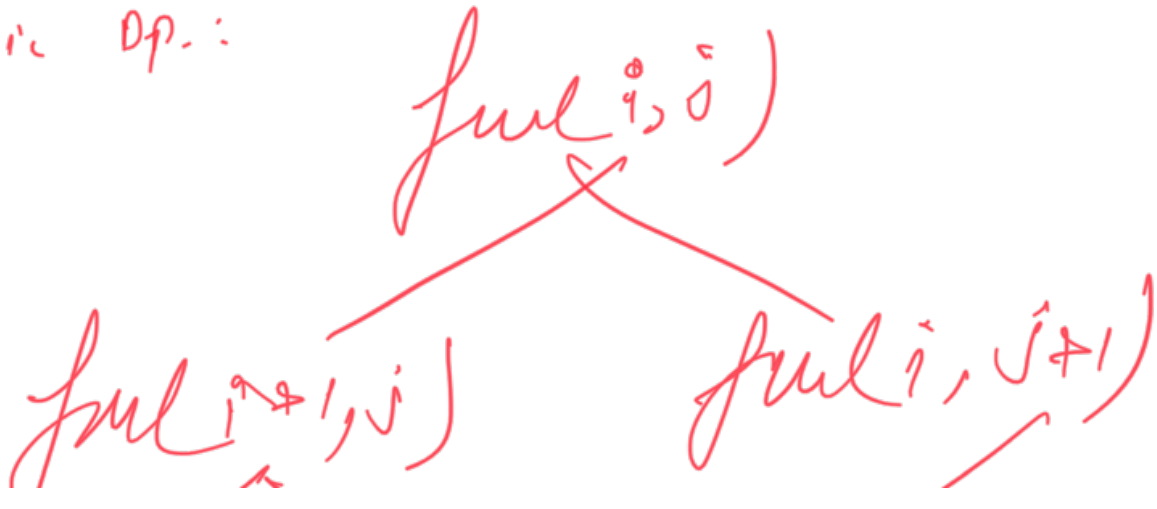
$20+2+1 = 23$

```
int solve(Node *root, int *res)
{
    if(root==null) return 0
    int l = solve(root.left, res)
    int r = solve(root.right, res)

    int temp = max(l, r) + root.val.

      if (root.left == null & root.right == null) //leaf node
            temp = max(temp, root.val)

    ans = max(temp, l+r+root.val)
    res = max(res, ans)

    return ans
}
```

Dynamic Programming Bottom Up

Top Down

Topic DP:




fun(i+2)
fun(i+1, j+1)