

# DYNAMIC PROGRAMMING

KAPIL YADAV



Kapilyadav22



## LECTURE - 29 MINIMUM INSERTIONS TO MAKE STRING PALINDROMIC

23 June 2022 12:41

Can we make a string palindromic by inserting some characters?

Ans:- Yes, str1 = "abcaq"

add aa cbq, str2 = "abc aa qa cbq" is palindromic. No. of operations = length of a string.

a b c a q



a a b c b a q

Q.) How do you approach it?

Keep the longest palindromic portion intact.

→ Take a q q.

a b c a c b q

→ Take a c q.

a b q c a b q

→ Take a b q

a a c b c q q

→ The number of insertions =

N - Longest palindromic subsequence

Tabulation:



```
1 int longestCommonSubsequence(string s1, string s2) {  
2     int m = s1.length();  
3     int n = s2.length();  
4  
5     vector<int> next(n+1, 0);  
6  
7     for(int ind1 = m-1; ind1 >= 0; ind1--)  
8     { vector<int> curr(n+1, 0);  
9         for(int ind2 = n-1; ind2 >= 0; ind2--)  
10         { if(s1[ind1] == s2[ind2])  
11             curr[ind2] = 1 + next[ind2+1];  
12         else  
13             curr[ind2] = max(next[ind2], curr[ind2+1]);  
14         }  
15         next = curr;  
16     }  
17     return next[0];  
18 }  
19  
20 int minInsertions(string s) {  
21     int N = s.size();  
22     string s2 = s;  
23     reverse(s2.begin(), s2.end());  
24     return N - longestCommonSubsequence(s, s2);  
25 }
```

TC: O(M\*N)  
SC: O(N)

## Space Optimization: Using one 1D array

```
1 int minInsertions(string s1) {
2     string s2 =s1;
3     reverse(s2.begin(),s2.end());
4     int m = s1.length();
5     int n = s2.length();
6
7     vector<int> next(n+1,0);
8
9     for(int ind1 = m-1;ind1>=0;ind1--)
10    { vector<int> curr(n+1,0);
11        for(int ind2 = n-1;ind2>=0;ind2--)
12        { if(s1[ind1]==s2[ind2])
13            curr[ind2]=1 + next[ind2+1];
14        else
15            curr[ind2]=max(next[ind2],curr[ind2+1]);
16        }
17        next = curr;
18    }
19    return m-next[0];
20 }
```

[LinkedIn/kapilyadav22](#)

TC:  $O(M \cdot N)$   
SC:  $O(N)$

# LECTURE - 30 Min Insertion/deletions to convert String A to String B

23 June 2022 12:42

## Minimum number of deletions and insertions. ↗

Easy Accuracy: 59.67% Submissions: 19571 Points: 2

Given two strings **str1** and **str2**. The task is to remove or insert the minimum number of characters from/in **str1** so as to transform it into **str2**. It could be possible that the same character needs to be removed/deleted from one point of **str1** and inserted to some another point.

**Example 1:**

**Input:** str1 = "heap", str2 = "pea"

**Output:** 3

str1 = heap      str2 = pea

remove h, p from str1.

Insert p in the beginning of str1.

- We needed 3 operations to make str1 = str2.
- But how we got there?

→ If we observe it carefully, we will find out that other than longest common subsequence, rest characters we need to remove from str1 and insert the characters from str2.



- No. of deletions = str1.length - LCS.
- No. of insertions = str2.length - LCS.

Total Min Operations = Insertion + Deletions

## Tabulation:

```
● ● ●

1 int longestCommonSubsequence(string s1, string s2) {
2     int m = s1.length();
3     int n = s2.length();
4
5     vector<int> next(n+1,0);
6
7     for(int ind1 = m-1;ind1>=0;ind1--)
8     { vector<int> curr(n+1,0);
9         for(int ind2 = n-1;ind2>=0;ind2--)
10        { if(s1[ind1]==s2[ind2])
11            curr[ind2]=1 + next[ind2+1];
12        else
13            curr[ind2]=max(next[ind2],curr[ind2+1]);
14        }
15        next = curr;
16    }
17    return next[0];
18 }
19 public:
20 int minOperations(string A, string B)
21 {
22     int m =A.size();
23     int n = B.size();
24     int deletions = m-longestCommonSubsequence(A,B);
25     int insertions = n-longestCommonSubsequence(A,B);
26     return insertions+deletions;
27 }
```

TC:  $O(M \cdot N)$   
SC:  $O(N)$

[LinkedIn](#)/kapilyadav22

# LECTURE - 31, SHORTEST COMMON SUPERSEQUENCE

23 June 2022 14:44

## 1092. Shortest Common Supersequence

Hard 2498 44 Add to List Share

Given two strings `str1` and `str2`, return the shortest string that has both `str1` and `str2` as subsequences. If there are multiple valid strings, return any of them.

A string `s` is a subsequence of string `t` if deleting some number of characters from `t` (possibly 0) results in the string `s`.

**Example 1:**

```
Input: str1 = "abac", str2 = "cab"
Output: "cabac"
Explanation:
str1 = "abac" is a subsequence of "cabac" because we can delete the first "c".
str2 = "cab" is a subsequence of "cabac" because we can delete the last "ac".
The answer provided is the shortest such string that satisfies these properties.
```

To find length of Shortest Common Supersequence :

Ex -  $\text{str1} = \underline{a}b\underline{a}c$ ,  $\text{str2} = \underline{c}ab$   
output  $\rightarrow c \underline{a} b \underline{a} c$

We need to add all the characters from `str1` and `str2`, but we have to remove the LCS one time, as it will occur twice

So, Length of Shortest Common Supersequence :  $\text{str1 length} + \text{str2 length} - \text{LCS}$ .

### To find Shortest Common Supersequence:

- Just print LCS
- If char is matching add it once.
- If not then add the char : from  $dp(i+1, j)$  or  $dp(i, j+1)$ , whichever index we will update.
- If i is updating, add from s1, if j is updating add from s2

Ex: `str1 = "abac"` `str2 = "cab"`

	c	a	b	
a	2	2	1	0
b	1	1	1	0
a	1	1	0	0
c	1	0	0	0
	0	0	0	0

LCS table.

	c	a	b	
a	2	2	1	0
b	1	1	1	0
a	1	1	0	0
c	1	0	0	0
	0	0	0	0

How LCS table is evaluated

## Finding Shortest Common Supersequence

- After Finding LCS, Take string ans to store our Shortest Common Supersequence
- Run the i, j loop from 0 index.
- Here a and c are not matching and we are getting maximum value from (i,j+1) so add **c** (str2[j]) in ans string and Increment j.
- Now we are at dp[0][1], a==a. So add **a** in ans and increment both i and j.
- Now we are dp[1][2]. b==b, so again add **b** in ans and increment both i and j.
- Now we are at dp[2][3], so it exceeds the length of str2, so add remaining elements of str1 in ans, i.e. "a" and "c".
- Our final string will be "**cabac**".

		c	q	b	
		a	2 → 2	1	0
		b	1	1	0
q		1	1	0	0
c		1	0	0	0
		0	0	0	0

add a,  
add c.

## Tabulation:

[LinkedIn](#)/kapilyadav22

```
1 string shortestCommonSupersequence(string s1, string s2) {
2     int m = s1.length();
3     int n = s2.length();
4     vector<vector<int>> dp(m+1, vector<int>(n+1, 0));
5
6     for(int ind1 = m-1; ind1>=0; ind1--)
7     { for(int ind2 = n-1; ind2>=0; ind2--)
8         { if(s1[ind1]==s2[ind2])
9             dp[ind1][ind2]=1 + dp[ind1+1][ind2+1];
10            else
11                dp[ind1][ind2]=max(dp[ind1+1][ind2],dp[ind1][ind2+1]);
12            }
13        }
14        string ans="";
15
16        int i =0;int j=0;
17
18        while(i<m && j<n)
19        {
20            if(s1[i]==s2[j])
21            { ans+=s1[i];
22                i++; j++;
23            }
24            else if(dp[i+1][j]>dp[i][j+1])
25            { ans+=s1[i];
26                i++;
27            }
28            else {
29                ans+=s2[j];
30                j++;
31            }
32        }
33        while(i<m)
34        { ans+=s1[i];
35            i++;    }
36
37        while(j<n)
38        { ans+=s2[j];
39            j++;   }
40
41        return ans;
42    }
```

## LECTURE - 32 DISTINCT SUBSEQUENCES (DP ON STRINGS)

Tuesday, 14 June 2022 2:45 PM

$$S = "babgbag" \quad S_2 = "bag"$$

b a b g b a g      |      b a b g b a g  
b a b g b a g      |      b a b g b a g  
b a b g b a g

Trying all ways - Recursion

→ Count the number of ways :-

Q.) How to write Recurrence?

1. Express everything in terms of  $(i, j)$ .
2. Explore all possibilities.
3. Return summation of all possibilities.
4. Base.

$\underset{n}{\rightarrow} S_1 = b a b g b a g$        $\underset{m}{\rightarrow} S_2 = b a g$

→ if  $(S_1[i] == S_2[j])$   
    return  $(f(i-1, j-1) + f(i-1, j))$ ;  
else  
    return  $f(i-1, j)$ ;

Base case :-

$i$   
 $\uparrow$        $b a b g b a g$        $\uparrow$        $b a g$   
 $-1$

- if  $S_1$  is exhausted, but  $S_2$  is not, so return 0.  
→ if  $S_2$  is exhausted, but  $S_1$  is not, return 1.

[Recursion; TC  $\rightarrow$  exponential  
SC  $\rightarrow$   $O(N+M)$ ]

### → Use Memoization

```
● ● ● ● ●

1 int countdistinct(string s, string t, int ssize, int tsize,
2   vector<vector<int>> &dp)
3   {
4     if(tsize<0) return 1;
5     if(ssize<0) return 0;
6     if(dp[ssize][tsize]!=-1)
7       return dp[ssize][tsize];
8     if(s[ssize]==t[tsize])
9       return countdistinct(s,t,ssize-1,tsize-1,dp) +
10      countdistinct(s,t,ssize-1,tsize,dp);
11    return countdistinct(s,t,ssize-1,tsize,dp);
12  }
13
14  int numDistinct(string s, string t) {
15    int n = s.size();
16    int m = t.size();
17    vector<vector<int>> dp(n, vector<int> (m, -1));
18    return countdistinct(s,t,n-1,m-1,dp);
19  }
```

→ It will give TLE.  
 $TC = O(N \times M)$ ,  $SC \rightarrow O(N \times M) + O(N+M)$

### → Tabulation :-

```
● ● ● ● ●

1 int numDistinct(string s, string t) {
2   int n = s.size();
3   int m = t.size();
4   vector<vector<int>> dp(n+1, vector<int> (m+1, 0));
5   for(int i=0; i<=n; i++)
6     dp[i][0] = 1;
7   for(int j=1; j<=m; j++)
8     dp[0][j] = 0;
9   for(int i=1; i<=n; i++)
10  { for(int j=1; j<=m; j++)
11    { if(s[i-1]==t[j-1])
12      dp[i][j] = dp[i-1][j-1] + dp[i-1][j];
13      else dp[i][j] = dp[i-1][j];
14    }
15  }
16  return dp[n][m];
17 }
```

→ It will give Runtime Error.      TC:  $O(N \times M)$   
 $SC: O(N \times M)$

- Try to change the datatype of vector dp



```
1 int numDistinct(string s, string t) {
2     int n = s.size();
3     int m = t.size();
4     //try to change the data type of vector dp
5     vector<vector<double>> dp(n+1, vector<double> (m+1,0));
6     for(int i=0;i<=n;i++)
7         dp[i][0] =1;
8     for(int j=1;j<=m;j++)
9         dp[0][j]=0;
10    for(int i=1;i<=n;i++)
11    { for(int j=1;j<=m;j++)
12        { if(s[i-1]==t[j-1])
13            dp[i][j]= dp[i-1][j-1] + dp[i-1][j];
14            else dp[i][j] = dp[i-1][j];
15        }
16    }
17    return (int)dp[n][m];
18 }
```

TC:O(N\*M)

SC: O(N\*M)

- Optimize it using two 1D arrays:



```
1 int numDistinct(string s, string t) {
2     int n = s.size();
3     int m = t.size();
4     vector<double> prev(m+1,0),curr(m+1,0);
5     prev[0]= curr[0]=1;
6
7     for(int i=1;i<=n;i++)
8     { for(int j=1;j<=m;j++)
9         { if(s[i-1]==t[j-1])
10             curr[j]= prev[j-1] + prev[j];
11             else
12                 curr[j] = prev[j];
13         }
14     prev = curr;
15     }
16     return (int)prev[m];
17 }
```

TC:O(N\*M)

SC: O(M)

- Use only one 1D dp



```
1 int numDistinct(string s, string t) {  
2     int n = s.size();  
3     int m = t.size();  
4     vector<double> prev(m+1, 0);  
5     prev[0] = 1;  
6  
7     for(int i=1; i<=n; i++)  
8     { for(int j=m; j>=1; j--)  
9         { if(s[i-1]==t[j-1])  
10             prev[j] = prev[j-1] + prev[j];  
11         }  
12     }  
13     return (int)prev[m];  
14 }
```

TC: O(N\*M)  
SC: O(M)

[LinkedIn/kapilyadav22](https://www.linkedin.com/in/kapilyadav22)

# LECTURE - 33 EDIT DISTANCE

23 June 2022 18:59

## 72. Edit Distance

Hard 8696 100 Add to List Share

Given two strings `word1` and `word2`, return the minimum number of operations required to convert `word1` to `word2`.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

→ Is it always possible?

→ Yes, Delete the word1 completely and insert the char same as word2.

$$TC = O(N+M)$$

String Matching :-

Try all possible ways - Recursion.

Q) How to write recurrence?

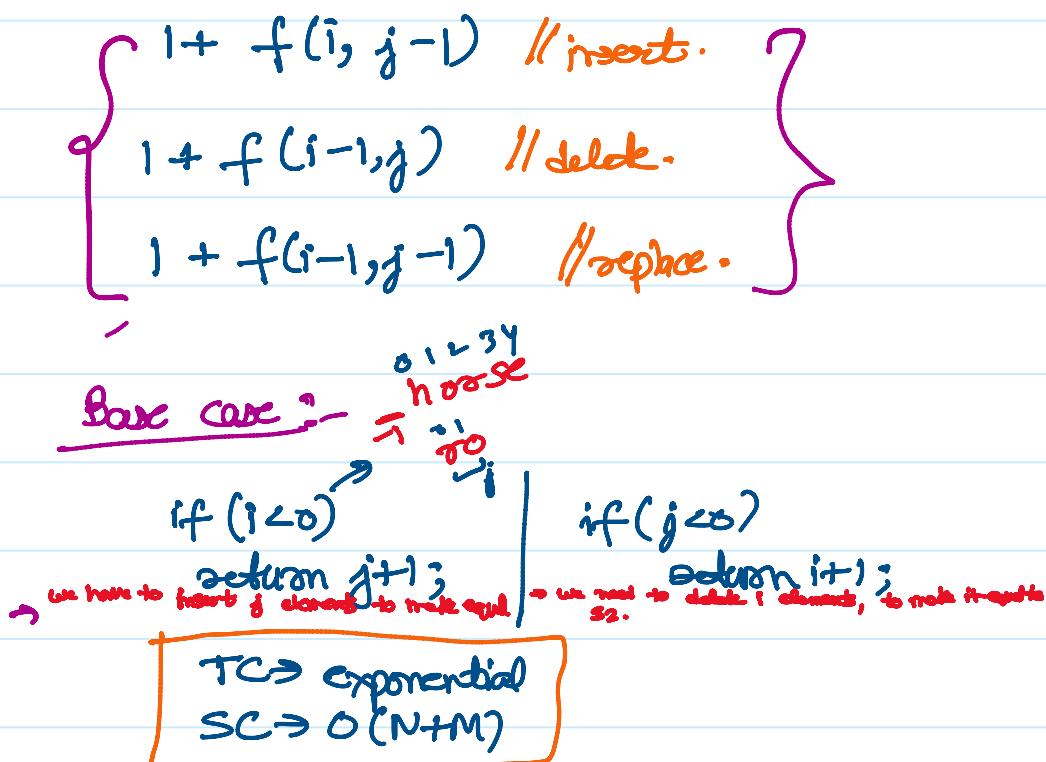
Ans) 1) Express in terms of  $(i, j)$ .  
2) Explore all paths of matching.  
3) Return the min of all paths.  
4) Base case.

$f(n-1, m-1) \rightarrow$  min operations to connect  $s_1[0 \dots i]$  to  $s_2[0 \dots j]$ .

→ if  $(s_1[i] == s_2[j])$   
action  $f(i-1, j-1)$

→ Insertion

{  $1 + f(i, j-1)$  //insert. ? }



## Recursion:

```

● ● ●

1 int findMinOp(int i, int j, string word1, string word2)
2 {
3     if(i<0)
4         return j+1;
5     if(j<0)
6         return i+1;
7     if(word1[i]==word2[j])
8         return findMinOp(i-1,j-1,word1,word2);
9     return 1+min(findMinOp(i-1,j,word1,word2),
10                  min(findMinOp(i,j-1,word1,word2),
11                      findMinOp(i-1,j-1,word1,word2)));
12
13 }
14
15 int minDistance(string word1, string word2) {
16     int n = word1.size();
17     int m = word2.size();
18     return findMinOp(n-1,m-1,word1,word2);
19
20 }

```

## MEMOIZATION:

```
1 int findMinOp(int i, int j, string word1, string word2, vector<vector<int>>& dp)
2 {
3     if(i<0)
4         return j+1;
5     if(j<0)
6         return i+1;
7     if(dp[i][j]!=-1)
8         return dp[i][j];
9
10    if(word1[i]==word2[j])
11        return findMinOp(i-1,j-1,word1,word2,dp);
12    int deletechar= findMinOp(i-1,j,word1,word2,dp);
13    int replacechar = findMinOp(i-1,j-1,word1,word2,dp);
14    int insertchar = findMinOp(i,j-1,word1,word2,dp);
15    return dp[i][j] = 1+min(insertchar, min(replacechar,deletechar));
16 }
17
18 int minDistance(string word1, string word2) {
19     int n = word1.size();
20     int m = word2.size();
21     vector<vector<int>> dp(n, vector<int>(m,-1));
22     return findMinOp(n-1,m-1,word1,word2,dp);
23 }
```

TC: O(N\*M) SC : O(N\*M) + O(N+M)

## Tabulation:

```
1 int minDistance(string word1, string word2) {
2     int n = word1.size();
3     int m = word2.size();
4     vector<vector<int>> dp(n+1, vector<int>(m+1,0));
5     for(int i=0;i<=n;i++)
6         dp[i][0]=i;
7     for(int j=0;j<=m;j++)
8         dp[0][j]=j;
9
10    for(int i=1;i<=n;i++)
11    { for(int j=1;j<=m;j++)
12        { if(word1[i-1]==word2[j-1])
13            dp[i][j]=dp[i-1][j-1];
14            else dp[i][j]= 1 + min(dp[i-1][j],min(dp[i][j-1],dp[i-1][j-1]));
15        }
16    } return dp[n][m];
17 }
```

TC: O(N\*M) SC : O(N\*M)

- Using 1D arrays,



```

1 int minDistance(string word1, string word2) {
2     //using 1d arrays
3     int n = word1.size();
4     int m = word2.size();
5     vector<int> prev(m+1, 0), cur(m+1, 0);
6
7     for(int j=0; j<=m; j++)
8         prev[j]=j;
9
10    for(int i=1; i<=n; i++)
11    { cur[0]=i;
12        for(int j=1; j<=m; j++)
13        { if(word1[i-1]==word2[j-1])
14            cur[j]=prev[j-1];
15            else cur[j]= 1 + min(prev[j],min(cur[j-1],prev[j-1]));
16        }
17        prev=cur;
18    }
19    return prev[m];
20 }
```

TC: O(N\*M) SC : O(M)

[LinkedIn/kapilyadav22](https://www.linkedin.com/in/kapilyadav22)

# LECTURE - 34 WILDCARD MATCHING

24 June 2022 20:09

[LinkedIn/kapilyadav22](#)

## 44. Wildcard Matching

Hard 5014 227 Add to List Share

Given an input string ( $s$ ) and a pattern ( $p$ ), implement wildcard pattern matching with support for `'?'` and `'*'` where:

- `'?'` Matches any single character.
- `'*'` Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

### Example 1:

```
Input: s = "aa", p = "a"  
Output: false  
Explanation: "a" does not match the entire string "aa".
```

### Example 2:

```
Input: s = "aa", p = "*"  
Output: true  
Explanation: '*' matches any sequence.
```

Ex -  $S_1 = "a ? a y"$  } True  
 $S_2 = "a a y"$  } True

$S_1 = "a b * c d"$  } True  
 $S_2 = "a b d e f c d"$  } True  
where  $*$  matches with def

$S_1 = "a * a b c d"$   
 $S_2 = "a b c d"$   
where  $*$  matches with length 0.

$S_1 = "a b ? *"$  } False  
 $S_2 = "a b c *"$  } False

- for ?, we only need to match one character
- But for \*, we need to match the pattern,

Ex - a b \* c d e.

a b Kap c d e.

- so pattern can be p, ap, Kap, bKap, abKap, abKapc, abKapcd or abKapcde, ....

- we need to check pattern for length 0, 1, 2, ...

- So, to find all possible ways, we need Recursion.

- If both the characters are matching, we will increment the index of both.

- If there is a '\*' of length 0, we can increment the index of  $S_2$ .

ex  $S_1 \rightarrow a^* \underline{C}$

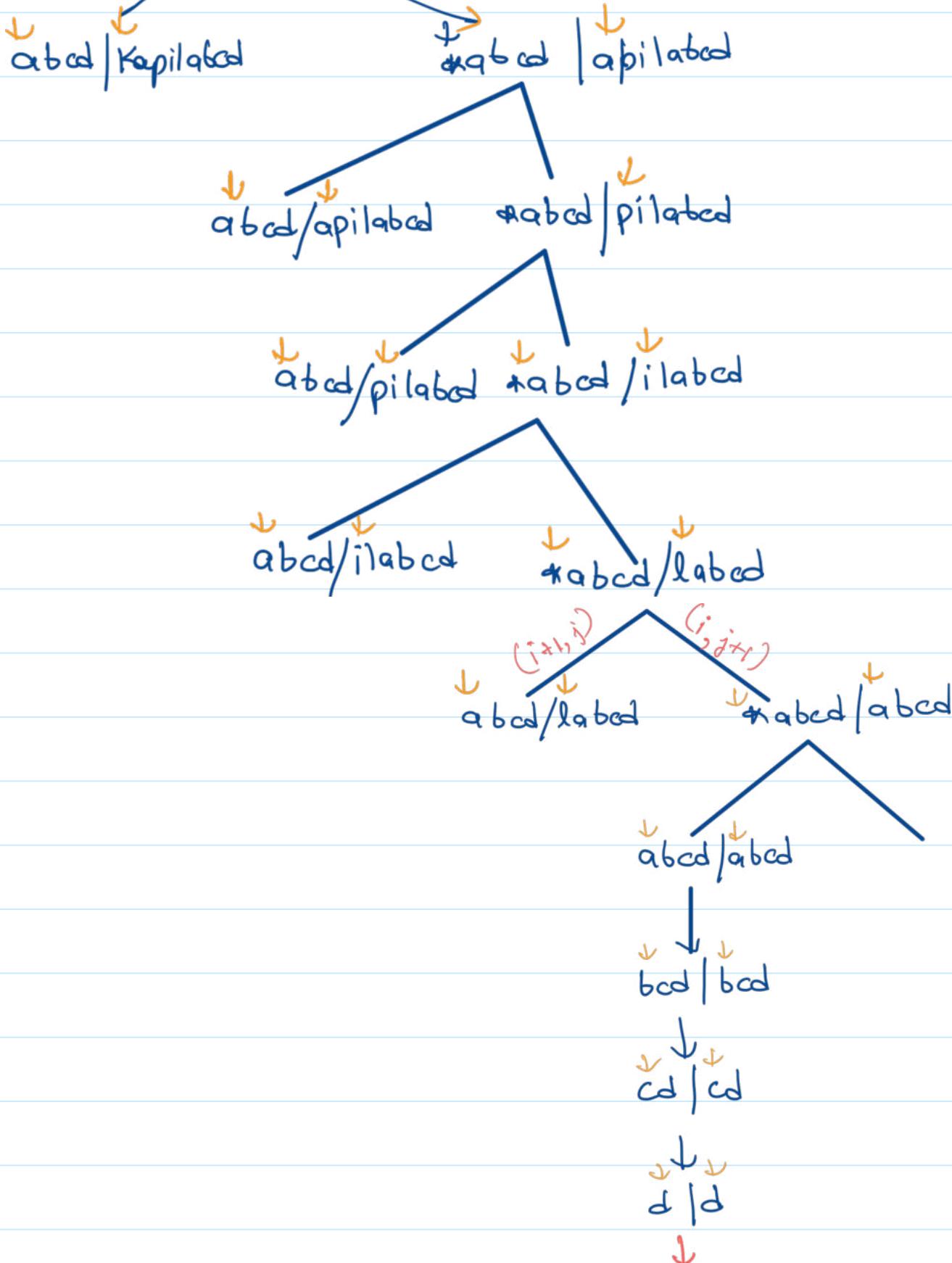
$S_2 \rightarrow \underline{a} C$

- for \* of length 1, 2, 3, ....

Ex →

$$S_1 = *abcd$$

$$S_2 = \text{Kapil}abcd$$



- If  $S_1 == \text{end}$  &  $S_2 == \text{end}$  return True.
- If  $S_1 == \text{end}$  &  $S_2 != \text{end}$  return False
- If  $S_1 != \text{end}$  &  $S_2 == \text{end}$   
 for( $i=0$  to  $S_1.\text{length}()$ )  
 if ( $S_1[i] != '*'$ )  
 return False.

It will return  
action false.

→ Recursion :-    TC → Exponential  
                    SC →  $O(N+M)$

→ Memoization :-    TC :  $O(N \times M)$   
                    SC :  $O(N \times M) + O(N+M)$

Memoization:



```
1 bool findwildcard(int Sind, int PInd, string& s, string& p,
2   vector<vector<int>>& dp)
3 {  int SEndInd = s.size() - 1;
4   int PEndInd = p.size() - 1;
5
6   if(Sind > SEndInd && PInd > PEndInd)
7     return true;
8   if(Sind <= SEndInd && PInd > PEndInd)
9     return false;
10  if(Sind > SEndInd && PInd <= PEndInd)
11  {  for(int i=PInd; i <= PEndInd; i++)
12    {  if(p[i] != '*')
13      return false;
14    }  return true;
15  }
16  if(dp[Sind][PInd] != -1)
17    return dp[Sind][PInd];
18  if(s[Sind] == p[PInd] || p[PInd] == '?')
19    return dp[Sind][PInd] = findwildcard(Sind + 1, PInd + 1, s, p, dp);
20  if(p[PInd] == '*')
21  {
22    return dp[Sind][PInd] = findwildcard(Sind + 1, PInd, s, p, dp) ||
23           findwildcard(Sind, PInd + 1, s, p, dp);
24  }
25  return dp[Sind][PInd] = false;
26
27
28  bool isMatch(string s, string p) {
29    int n = s.size();
30    int m = p.size();
31    vector<vector<int>> dp(n, vector<int> (m, -1));
32    return findwildcard(0, 0, s, p, dp);
33 }
```

## Tabulation:

```
1 bool isMatch(string s, string p) {
2     int n = s.size();
3     int m = p.size();
4     vector<vector<int>> dp(n+1, vector<int> (m+1, 0));
5     dp[n][m]=1;
6
7     for(int i=m-1;i>=0;i--)
8     { if(p[i]=='*')
9         dp[n][i]=1;
10        else
11            break;
12    }
13
14    for(int Sind=n-1;Sind>=0;Sind--)
15    {
16        for(int PInd=m-1;PInd>=0;PInd--)
17        {if(s[Sind]==p[PInd] || p[PInd]=='?')
18            dp[Sind][PInd] = dp[Sind+1][PInd+1];
19        else if(p[PInd]=='*')
20        {
21            dp[Sind][PInd]=dp[Sind+1][PInd] || dp[Sind][PInd+1];
22        }
23        else dp[Sind][PInd]=false;
24    }
25    }
26    return dp[0][0];
27 }
```

TC: O(N\*M)

SC: O(N\*M)

## \*LECTURE - 35 BEST TIME TO BUY AND SELL STOCK

24 June 2022 20:10

### 121. Best Time to Buy and Sell Stock

Easy 17280 564 Add to List Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

#### Example 1:

Input: `prices = [7,1,5,3,6,4]`

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit =  $6-1 = 5$ .

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

#### Example 2:

Input: `prices = [7,6,4,3,1]`

Output: 0

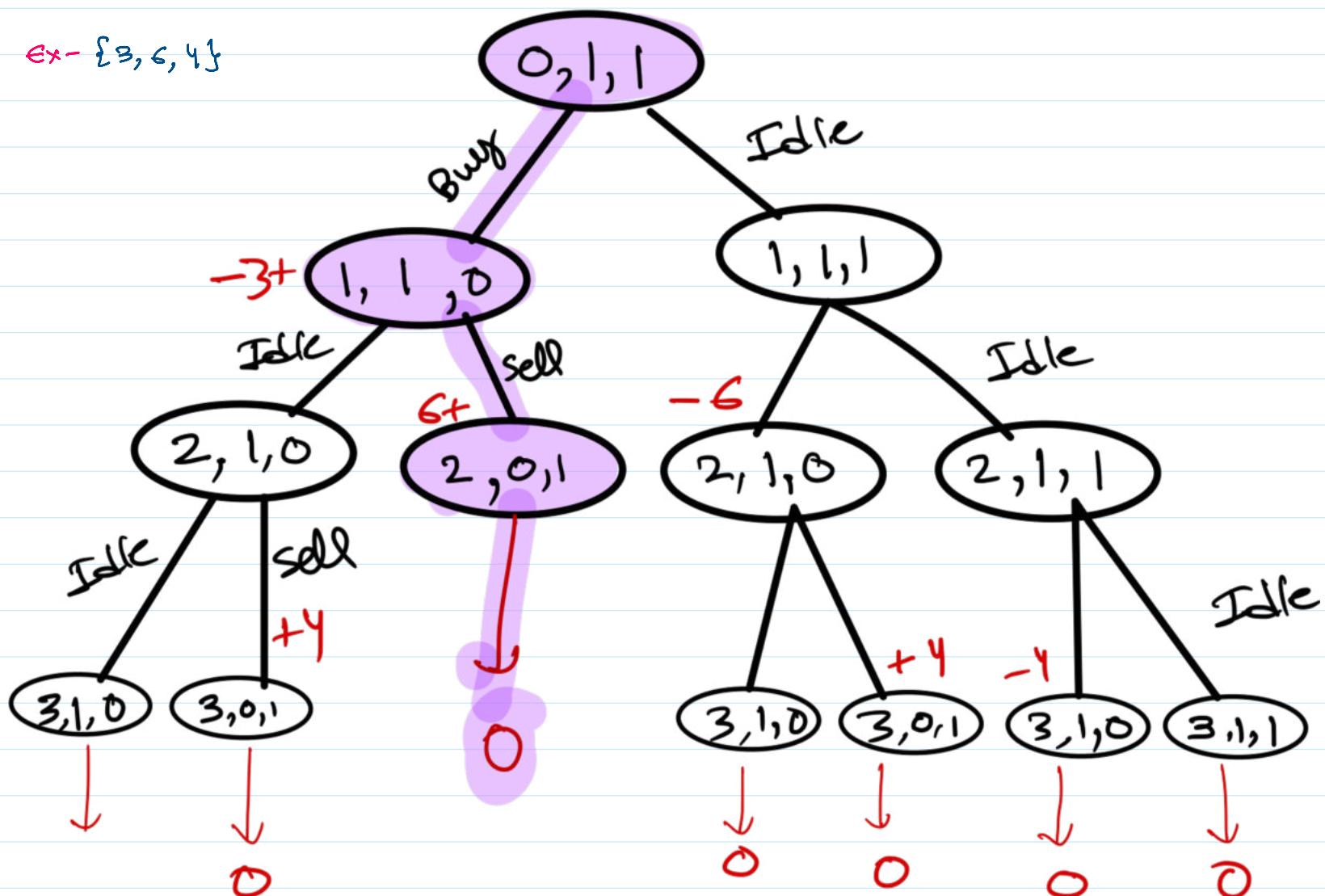
Explanation: In this case, no transactions are done and the max profit = 0.

There are 3 possibilities :

1. Idle (Neither buying Nor selling)
2. Buy the stock
3. Sell the stock (only if we have bought it previously)

So, There are 3 changing parameters:

1. CurrentDay
2. TransactionCount
3. Canbuy : whether we can buy on currentday or not.



## MEMOIZATION:

```
● ● ●
```

```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<vector<int>>> v(m, vector<vector<int>> (2, vector<int> (2,-1)));
4     return buyandsell(0,1,true,prices,v);
5 }
6
7 int buyandsell(int currentindex,int transactioncount,bool canbuy, vector<int>& prices,vector<vector<vector<int>>> & v)
8 {
9     if(currentindex>=prices.size() || transactioncount<=0)
10        return 0;
11
12    int key = v[currentindex][transactioncount][canbuy];
13    if(key!=-1)
14        return key;
15
16    int idle = buyandsell(currentindex+1,transactioncount,canbuy,prices,v);
17
18    if(canbuy)
19    {   int buyside = -prices[currentindex] + buyandsell(currentindex+1,transactioncount,0,prices,v);
20        return v[currentindex][transactioncount][canbuy] = max(buyside,idle);
21    }
22    else
23    {   int sellside = prices[currentindex] + buyandsell(currentindex+1,transactioncount-1,1,prices,v);
24        return v[currentindex][transactioncount][canbuy] = max(sellside,idle);
25    }
26 }
27 }
```

TC: O(M\*2\*2)  
SC : O(M\*2\*2) + O(M)

## Tabulation:

```
● ● ●
```

```
1 nt maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<vector<int>>> v(m+1, vector<vector<int>> (2, vector<int> (2,0)));
4     for(int currentindex=m-1;currentindex>=0;currentindex--)
5     {
6         for(int transactioncount=1;transactioncount>=1;transactioncount--)
7         {
8             for(int canbuy=0;canbuy<=1;canbuy++)
9             {
10                 int idle = v[currentindex+1][transactioncount][canbuy];
11
12                 if(canbuy)
13                 {   int buyside = -prices[currentindex] + v[currentindex+1][transactioncount][0];
14                     v[currentindex][transactioncount][canbuy] = max(buyside,idle);
15                 }
16                 else
17                 {   int sellside = prices[currentindex] + v[currentindex+1][transactioncount-1][1];
18                     v[currentindex][transactioncount][canbuy] = max(sellside,idle);
19                 }
20             }
21         }
22     }  return v[0][1][true];
23 }
24 }
```

TC: O(M\*2\*2)  
SC : O(M\*2\*2)

## Optimization:

```
● ● ●
```

```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<vector<int>>> v(m+1,vector<vector<int>> (2,vector<int> (2,0)));
4     for(int currentindex=m-1;currentindex>=0;currentindex--)
5     {   for(int canbuy=0;canbuy<=1;canbuy++)
6         {
7             int idle = v[currentindex+1][1][canbuy];
8
9             if(canbuy)
10                { int buyside = -prices[currentindex] + v[currentindex+1][1][0];
11                  v[currentindex][1][canbuy] = max(buyside,idle);
12                }
13            else
14                { int sellside = prices[currentindex] + v[currentindex+1][0][1];
15                  v[currentindex][1][canbuy] = max(sellside,idle);
16                }
17
18        }
19    } return v[0][1][true];
20 }
```

TC: O( $M^2$ )  
SC : O( $M^2 \cdot 2^2$ )

## Further Space Optimization:

```
● ● ●
```

```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<int>> next(2,vector<int> (2,0));
4     for(int currentindex=m-1;currentindex>=0;currentindex--)
5     {   vector<vector<int>> curr(2,vector<int> (2,0));
6         for(int canbuy=0;canbuy<=1;canbuy++)
7             { int idle = next[1][canbuy];
8
9                 if(canbuy)
10                    { int buyside = -prices[currentindex] + next[1][0];
11                      curr[1][canbuy] = max(buyside,idle);
12                    }
13                else
14                    { int sellside = prices[currentindex] + next[0][1];
15                      curr[1][canbuy] = max(sellside,idle);
16                    }
17
18            next = curr;
19        } return next[1][true];
20 }
```

TC: O( $M^2$ )  
SC : O( $M^2$ )

[LinkedIn/kapilyadav22](#)

## LECTURE - 36 BEST TIME TO BUY AND SELL STOCK -II

24 June 2022 20:10

### 122. Best Time to Buy and Sell Stock II

Medium 7978 2396 Add to List Share

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the maximum profit you can achieve*.

#### Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
Total profit is 4 + 3 = 7.
```

#### Example 2:

```
Input: prices = [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
Total profit is 4.
```

In This Question, We have no limitation on buying and selling, so it will be same as Previous question, We just need to eliminate the transactioncount part.

$$\text{ex} \rightarrow \{ \underline{7} \underline{1} \underline{5} \underline{3} \leq \underline{4} \}$$

Buy on day-2, sell on day-3.  $5-1=4$   
Buy on day-4, sell on day-5  $6-3=\underline{3}$   
7

## MEMOIZATION:



```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<int>> v(m, vector<int> (2,-1));
4     return buyandsell(0,true,prices,v);
5 }
6
7 int buyandsell(int currentindex,bool canbuy, vector<int>& prices,vector<vector<int>>& v)
8 {
9     if(currentindex>=prices.size() )
10         return 0;
11
12     int key = v[currentindex][canbuy];
13     if(key!=-1)
14         return key;
15
16     int idle = buyandsell(currentindex+1,canbuy,prices,v);
17
18     if(canbuy)
19     {   int buyside = -prices[currentindex] + buyandsell(currentindex+1,0,prices,v);
20         return v[currentindex][canbuy] = max(buyside,idle);
21     }
22     else
23     {   int sellside = prices[currentindex] + buyandsell(currentindex+1,1,prices,v);
24         return v[currentindex][canbuy] = max(sellside,idle);
25     }
26 }
```

TC: O(M\*2)

SC : O(M\*2) + O(M)

## Tabulation:

```
1 int maxProfit(vector<int>& prices) {  
2     int m = prices.size();  
3     vector<vector<int>> dp(m+1, vector<int> (2,0));  
4     dp[m][0]=0;  
5     dp[m][1]=0;  
6  
7     for(int currentindex=m-1;currentindex>=0;currentindex--)  
8     { for(int canbuy = 0;canbuy<=1;canbuy++)  
9         { int idle = dp[currentindex+1][canbuy];  
10  
11             if(canbuy)  
12             { int buysize = -prices[currentindex] +dp[currentindex+1][0];  
13                 dp[currentindex][canbuy] = max(buysize,idle);  
14             }  
15             else  
16             { int sellside = prices[currentindex] +dp[currentindex+1][1];  
17                 dp[currentindex][canbuy] = max(sellside,idle);  
18             }  
19         }  
20     }  
21     return dp[0][1];  
22 }
```

TC: O(M\*2)  
SC : O(M\*2)

## Space Optimization:

```
1 int maxProfit(vector<int>& prices) {  
2     int m = prices.size();  
3     vector<int> next(2,0);  
4  
5     for(int currentindex=m-1;currentindex>=0;currentindex--)  
6     { vector<int> curr(2,0);  
7         for(int canbuy = 0;canbuy<=1;canbuy++)  
8         { int idle = next[canbuy];  
9  
10            if(canbuy)  
11            { int buysize = -prices[currentindex] +next[0];  
12                curr[canbuy] = max(buysize,idle);  
13            }  
14            else  
15            { int sellside = prices[currentindex] + next[1];  
16                curr[canbuy] = max(sellside,idle);  
17            }  
18        }  
19        next =curr;  
20    }  
21    return next[1];  
22 }
```

TC: O(M)  
SC : O(2)



```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     int nextbuy=0,nextnotbuy=0;
4     int currbuy=0,currnotbuy=0;
5     //it is same as vector<int> next(2,0)
6     // we have just taken 4 variables instead of two 1d vectors
7     for(int currentindex=m-1;currentindex>=0;currentindex--)
8     {
9         //we are computing for for buy=0 and buy =1
10
11         //buy=1
12         int idle = nextbuy;//we are not buying it, taking the next bought profit
13         int buyside = -prices[currentindex] + nextnotbuy;
14         currbuy = max(buyside,idle);
15
16         //buy=0
17         idle =nextnotbuy; //we are not selling it, taking the next not bought profit
18         int sellside = prices[currentindex] + nextbuy;
19         currnotbuy = max(sellside,idle);
20         nextnotbuy = currnotbuy;
21         nextbuy =currbuy;
22     }
23     return nextbuy;
24 }
```

# LECTURE - 37 BEST TIME TO BUY AND SELL STOCK-III

24 June 2022 20:10

## 123. Best Time to Buy and Sell Stock III

Hard 6119 122 Add to List Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the `ith` day.

Find the maximum profit you can achieve. You may complete **at most two transactions**.

**Note:** You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

### Example 1:

```
Input: prices = [3,3,5,0,0,3,1,4]
Output: 6
Explanation: Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.
```

Same as best time to buy and sell stocks, here the transactioncount = 2, nothing else

### MEMOIZATION:

```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<vector<int>>> v(m, vector<vector<int>> (3, vector<int> (2,-1)));
4     return buyandsell(0,2,true,prices,v);
5 }
6
7 int buyandsell(int currentindex,int transactioncount,bool canbuy, vector<int>&
8     prices,vector<vector<vector<int>>>& v)
9 {
10     if(currentindex>=prices.size() || transactioncount<=0)
11         return 0;
12     int key = v[currentindex][transactioncount][canbuy];
13     if(key!=-1)
14         return key;
15
16     int idle = buyandsell(currentindex+1,transactioncount,canbuy,prices,v);
17
18     if(canbuy)
19     {   int buysize = -prices[currentindex] +
20         buyandsell(currentindex+1,transactioncount,0,prices,v);
21         return v[currentindex][transactioncount][canbuy] = max(buysize,idle);
22     }
23     else
24     {   int sellside = prices[currentindex] +
25         buyandsell(currentindex+1,transactioncount-1,1,prices,v);
26         return v[currentindex][transactioncount][canbuy] = max(sellside,idle);
27     }
28 }
```

TC: O(M\*3\*2)

SC :  $O(M^3 \cdot 2) + O(M)$

### Tabulation:

```
● ○ ●

1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<vector<int>>> v(m+1, vector<vector<int>> (3, vector<int> (2,0)));
4     for(int currentIndex=m-1;currentIndex>=0;currentIndex--)
5     {
6         for(int transactionCount=2;transactionCount>=1;transactionCount--)
7         {
8             for(int canBuy=0;canBuy<=1;canBuy++)
9             {
10                 int idle = v[currentIndex+1][transactionCount][canBuy];
11
12                 if(canBuy)
13                 { int buySide = -prices[currentIndex] + v[currentIndex+1][transactionCount][0];
14                     v[currentIndex][transactionCount][canBuy] = max(buySide,idle);
15                 }
16                 else
17                 { int sellSide = prices[currentIndex] + v[currentIndex+1][transactionCount-1][1];
18                     v[currentIndex][transactionCount][canBuy] = max(sellSide,idle);
19                 }
20             }
21         }
22     } return v[0][2][true];
23 }
```

TC:  $O(M^2 \cdot 2^2)$

SC :  $O(M^3 \cdot 2)$

### Space Optimization:

```
● ○ ●

1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<int>> next(3, vector<int> (2,0));
4     for(int currentIndex=m-1;currentIndex>=0;currentIndex--)
5     { vector<vector<int>> curr(3, vector<int> (2,0));
6         for(int transactionCount=2;transactionCount>=1;transactionCount--)
7         {
8             for(int canBuy=0;canBuy<=1;canBuy++)
9             {
10                 int idle = next[transactionCount][canBuy];
11
12                 if(canBuy)
13                 { int buySide = -prices[currentIndex] + next[transactionCount][0];
14                     curr[transactionCount][canBuy] = max(buySide,idle);
15                 }
16                 else
17                 { int sellSide = prices[currentIndex] + next[transactionCount-1][1];
18                     curr[transactionCount][canBuy] = max(sellSide,idle);
19                 }
20             }
21         }
22     next = curr;
23     }
24     return next[2][true];
25 }
```

TC: O(M\*2\*2)  
SC : O(3\*2)

**Another Way:** We can Run the recursive calls for M\*4 times, can find out whether to buy or not using the transactioncount index,  
If transactioncount%2==0 i.e. For even places buy,  
For odd indexes sell.



```
1 int maxProfit(vector<int>& prices) {
2     vector<vector<int>> v(prices.size(),vector<int> (4,-1));
3     return buyandsell(0,0,prices,v);
4 }
5
6     int buyandsell(int currentindex, int transactioncount, vector<int>& prices,
7                     vector<vector<int>>& v)
8     {
9         if(currentindex>=prices.size() || transactioncount>=4)
10             return 0;
11
12
13         if(v[currentindex][transactioncount]!=-1)
14             return v[currentindex][transactioncount];
15
16         int idle = buyandsell(currentindex+1,transactioncount,prices,v);
17         int buy,sell = 0;
18
19         if(transactioncount%2==0)
20         {   buy = -prices[currentindex] + buyandsell(currentindex+1,transactioncount+1,prices,v);
21             return v[currentindex][transactioncount] = max(buy,idle);
22         }
23         else
24         {   sell = prices[currentindex] + buyandsell(currentindex+1,transactioncount+1, prices,v);
25             return v[currentindex][transactioncount] = max(sell,idle);
26         }
27     }
```

[LinkedIn/kapilyadav22](https://www.linkedin.com/in/kapilyadav22)

TC: O(M\*4)  
SC : O(M\*4) +O(M)

## LECTURE - 38 BEST TIME TO BUY AND SELL STOCK-IV

24 June 2022 20:11

### 188. Best Time to Buy and Sell Stock IV

Hard    4135    155    Add to List    Share

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day, and an integer `k`.

Find the maximum profit you can achieve. You may complete at most `k` transactions.

**Note:** You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

#### Example 1:

Input: `k = 2, prices = [2,4,1]`

Output: 2

Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 = 2.

Same as best time to buy and sell stocks, here the transactioncount = K.

#### MEMOIZATION:

```
● ● ●
1 int maxProfit(int k, vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<vector<int>>> v(m, vector<vector<int>> (k+1, vector<int> (2, -1)));
4     return buyandsell(0, k, true, prices, v);
5 }
6
7 int buyandsell(int currentindex, int transactioncount, bool canbuy, vector<int>&
8     prices, vector<vector<vector<int>>>& v)
9 {
10     if(currentindex >= prices.size() || transactioncount <= 0)
11         return 0;
12
13     int key = v[currentindex][transactioncount][canbuy];
14     if(key != -1)
15         return key;
16
17     int idle = buyandsell(currentindex+1, transactioncount, canbuy, prices, v);
18
19     if(canbuy)
20     {   int buyside = -prices[currentindex] + buyandsell(currentindex+1, transactioncount, 0, prices, v);
21         return v[currentindex][transactioncount][canbuy] = max(buyside, idle);
22     }
23     else
24     {   int sellside = prices[currentindex] + buyandsell(currentindex+1, transactioncount-1, 1, prices, v);
25         return v[currentindex][transactioncount][canbuy] = max(sellside, idle);
26     }
27 }
```

TC:  $O(M \cdot k^2)$

SC :  $O(M \cdot k^2) + O(M)$

#### Tabulation:



```
1 int maxProfit(int k,vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<vector<int>>> v(m+1,vector<vector<int>> (k+1,vector<int> (2,0)));
4     for(int currentIndex=m-1;currentIndex>=0;currentIndex--)
5     {
6         for(int transactionCount=k;transactionCount>=1;transactionCount--)
7         {
8             for(int canBuy=0;canBuy<=1;canBuy++)
9             {
10                 int idle = v[currentIndex+1][transactionCount][canBuy];
11
12                 if(canBuy)
13                 {   int buySide = -prices[currentIndex] + v[currentIndex+1][transactionCount][0];
14                     v[currentIndex][transactionCount][canBuy] = max(buySide,idle);
15                 }
16                 else
17                 {   int sellSide = prices[currentIndex] + v[currentIndex+1][transactionCount-1][1];
18                     v[currentIndex][transactionCount][canBuy] = max(sellSide,idle);
19                 }
20             }
21         }
22     }  return v[0][k][true];
23 }
```

TC: O(M\*k\*2)  
SC : O(M\*k\*2)

### Space Optimization:



```
1 int maxProfit(int k,vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<int>> next(k+1,vector<int> (2,0));
4     for(int currentIndex=m-1;currentIndex>=0;currentIndex--)
5     {
6         vector<vector<int>> curr(k+1,vector<int> (2,0));
7         for(int transactionCount=k;transactionCount>=1;transactionCount--)
8         {
9             for(int canBuy=0;canBuy<=1;canBuy++)
10            {
11                 int idle = next[transactionCount][canBuy];
12
13                 if(canBuy)
14                 {   int buySide = -prices[currentIndex] + next[transactionCount][0];
15                     curr[transactionCount][canBuy] = max(buySide,idle);
16                 }
17                 else
18                 {   int sellSide = prices[currentIndex] + next[transactionCount-1][1];
19                     curr[transactionCount][canBuy] = max(sellSide,idle);
20                 }
21            }
22        }  next = curr;
23    }  return next[k][true];
24 }
```

TC: O(M\*k\*2)  
SC : O(k\*2)

### Another way:

We can Run the recursive calls for  $M*(2^k)$  times, can find out whether to buy or not using the transactioncount index,

If transactioncount%2==0 i.e. For even transactioncount indexes buy,

For odd transactioncount indexes sell.



```
1 int maxProfit(int k, vector<int>& prices) {
2     vector<vector<int>> v(prices.size(), vector<int> (2*k, -1));
3     return buyandsell(0, 0, k, prices, v);
4 }
5
6 int buyandsell(int currentindex, int transactioncount, int k, vector<int>& prices,
7                 vector<vector<int>>& v)
8 {
9     if(currentindex >= prices.size() || transactioncount >= 2*k)
10        return 0;
11
12    if(v[currentindex][transactioncount] != -1)
13        return v[currentindex][transactioncount];
14
15    int idle = buyandsell(currentindex+1, transactioncount, k, prices, v);
16    int buy, sell = 0;
17
18    if(transactioncount % 2 == 0)
19    {   buy = -prices[currentindex] + buyandsell(currentindex+1, transactioncount+1, k, prices, v);
20        return v[currentindex][transactioncount] = max(buy, idle);
21    }
22    else
23    {   sell = prices[currentindex] + buyandsell(currentindex+1, transactioncount+1, k, prices, v);
24        return v[currentindex][transactioncount] = max(sell, idle);
25    }
26
27 }
```

[LinkedIn](#)/kapilyadav22

TC:  $O(M^*k^2)$

SC :  $O(M^*k^2) + O(M)$

# LECTURE - 39 BEST TIME TO BUY AND SELL STOCK WITH COOLDOWN

24 June 2022 20:11

## 309. Best Time to Buy and Sell Stock with Cooldown

Medium 5967 213 Add to List

You are given an array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day.

Find the maximum profit you can achieve. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times) with the following restrictions:

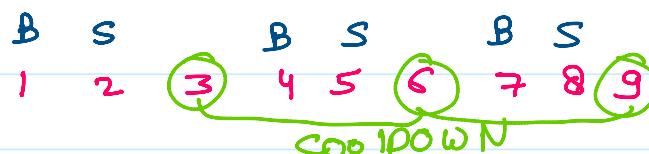
- After you sell your stock, you cannot buy stock on the next day (i.e., cooldown one day).

**Note:** You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

### Example 1:

```
Input: prices = [1,2,3,0,2]
Output: 3
Explanation: transactions = [buy, sell, cooldown, buy, sell]
```

Same as best time to buy and sell stocks 2, Here We Just need to cooldown for 1 day after selling the stock.  
Cooldown means we cannot buy any stock next day, just after selling it.



## MEMOIZATION:

```
● ● ●
```

```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<int>> v(m, vector<int> (2,-1));
4     return buyandsell(0,true,prices,v);
5 }
6
7 int buyandsell(int currentindex,bool canbuy, vector<int>& prices,vector<vector<int>>& v)
8 {
9     if(currentindex>=prices.size() )
10        return 0;
11
12
13     int key = v[currentindex][canbuy];
14     if(key!=-1)
15         return key;
16
17     int idle = buyandsell(currentindex+1,canbuy,prices,v);
18
19     if(canbuy)
20     {   int buyside = -prices[currentindex] + buyandsell(currentindex+1,0,prices,v);
21         return v[currentindex][canbuy] = max(buyside,idle);
22     }
23     else
24     {   int sellside = prices[currentindex] + buyandsell(currentindex+2,1,prices,v);
25         return v[currentindex][canbuy] = max(sellside,idle);
26     }
27 }
```

TC: O(M\*2)

SC : O(M\*2) + O(M)

**Tabulation:**



```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3     vector<vector<int>> dp(m+2, vector<int> (2,0));
4     dp[m][0]=0;
5     dp[m][1]=0;
6
7     for(int currentIndex=m-1;currentIndex>=0;currentIndex--)
8     { for(int canbuy = 0;canbuy<=1;canbuy++)
9     {   int idle = dp[currentIndex+1][canbuy];
10
11       if(canbuy)
12       {   int buyside = -prices[currentIndex] +dp[currentIndex+1][0];
13         dp[currentIndex][canbuy] = max(buyside,idle);
14       }
15     else
16     {   int sellside = prices[currentIndex] +dp[currentIndex+2][1];
17       dp[currentIndex][canbuy] = max(sellside,idle);
18     }
19 }
20 }
21 return dp[0][1];
22 }
```

TC: O(M\*2)  
SC : O(M\*2)

## Space Optimization:

```
1 int maxProfit(vector<int>& prices) {
2     int m = prices.size();
3
4     vector<int> next2(2,0);
5     vector<int> next1(2,0);
6     vector<int> curr(2,0);
7
8     for(int currentIndex=m-1;currentIndex>=0;currentIndex--)
9     {
10         int idle = next1[1];
11         int buyside = -prices[currentIndex] + next1[0];
12         curr[1] = max(buyside,idle);
13
14         idle = next1[0];
15         int sellside = prices[currentIndex] + next2[1];
16         curr[0] = max(sellside,idle);
17         next2 = next1;
18         next1 = curr;
19     }
20     return next1[1];
21 }
```

TC: O(M\*2)  
SC : O(2)

[LinkedIn](#)/kapilyadav22

# LECTURE - 40 BEST TIME TO BUY AND SELL STOCK WITH TRANSACTION FEE

24 June 2022 20:12

## 714. Best Time to Buy and Sell Stock with Transaction Fee

Medium 4127 106 Add to List Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the  $i^{\text{th}}$  day, and an integer `fee` representing a transaction fee.

Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

**Note:** You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

### Example 1:

```
Input: prices = [1,3,2,8,4,9], fee = 2
Output: 8
Explanation: The maximum profit can be achieved by:
- Buying at prices[0] = 1
- Selling at prices[3] = 8
- Buying at prices[4] = 4
- Selling at prices[5] = 9
The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.
```

Same as best time to buy and sell stocks 2, here we need to pay the transaction fee, while selling the stock, so overall profit on that day will be = selling price - buying price - transactionfee.

### MEMOIZATION:

```
1 int maxProfit(vector<int>& prices, int fee) {
2     int m = prices.size();
3     vector<vector<int>> v(m, vector<int> (2,-1));
4     return buyandsell(0,true,prices,v,fee);
5 }
6
7 int buyandsell(int currentindex,bool canbuy, vector<int>& prices,vector<vector<int>>& v,int fee)
8 {
9     if(currentindex>=prices.size() )
10         return 0;
11
12     int key = v[currentindex][canbuy];
13     if(key!=-1)
14         return key;
15
16     int idle = buyandsell(currentindex+1,canbuy,prices,v,fee);
17
18     if(canbuy)
19     {   int buyside = -prices[currentindex] + buyandsell(currentindex+1,0,prices,v,fee);
20         return v[currentindex][canbuy] = max(buyside,idle);
21     }
22     else
23     {   int sellside = prices[currentindex]-fee + buyandsell(currentindex+1,1,prices,v,fee);
24         return v[currentindex][canbuy] = max(sellside,idle);
25     }
26 }
```

TC: O(M\*2)  
SC : O(M\*2) + O(M)

### Tabulation:

TC: O(M\*2)  


```
1 int maxProfit(vector<int>& prices, int fee) {
2     int m = prices.size();
3     vector<vector<int>> dp(m+1, vector<int> (2,0));
4     dp[m][0]=0;
5     dp[m][1]=0;
6
7     for(int currentindex=m-1;currentindex>=0;currentindex--)
8     { for(int canbuy = 0;canbuy<=1;canbuy++)
9     { int idle = dp[currentindex+1][canbuy];
10
11     if(canbuy)
12     { int buysize = -prices[currentindex] +dp[currentindex+1][0];
13       dp[currentindex][canbuy] = max(buysize,idle);
14     }
15     else
16     { int sellside = prices[currentindex] +dp[currentindex+1][1]-fee;
17       dp[currentindex][canbuy] = max(sellside,idle);
18     }
19 }
20 }
21 return dp[0][1];
22 }
```

TC: O(M\*2)  
SC : O(M\*2)

## Space Optimization:



```
1 int maxProfit(vector<int>& prices, int fee) {
2     int m = prices.size();
3     vector<int> next(2, 0);
4
5     for(int currentIndex=m-1; currentIndex>=0; currentIndex--) {
6         vector<int> curr(2, 0);
7         for(int canbuy = 0; canbuy<=1; canbuy++) {
8             int idle = next[canbuy];
9
10            if(canbuy)
11            { int buysize = -prices[currentIndex] +next[0];
12              curr[canbuy] = max(buysize,idle);
13            }
14            else
15            { int sellside = prices[currentIndex] + next[1]-fee;
16              curr[canbuy] = max(sellside,idle);
17            }
18        }
19        next =curr;
20    }
21    return next[1];
22 }
```

[LinkedIn](#)/kapilyadav22

SC : O(2)

# \*LECTURE - 41 LONGEST INCREASING SUBSEQUENCE

24 June 2022 20:12

## 300. Longest Increasing Subsequence

Medium 12620 241 Add to List Share

Given an integer array `nums`, return the length of the longest strictly increasing subsequence.

A **subsequence** is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, `[3,6,2,7]` is a subsequence of the array `[0,3,1,6,2,2,7]`.

### Example 1:

Input: `nums = [10,9,2,5,3,7,101,18]`

Output: 4

Explanation: The longest increasing subsequence is `[2,3,7,101]`, therefore the length is 4.

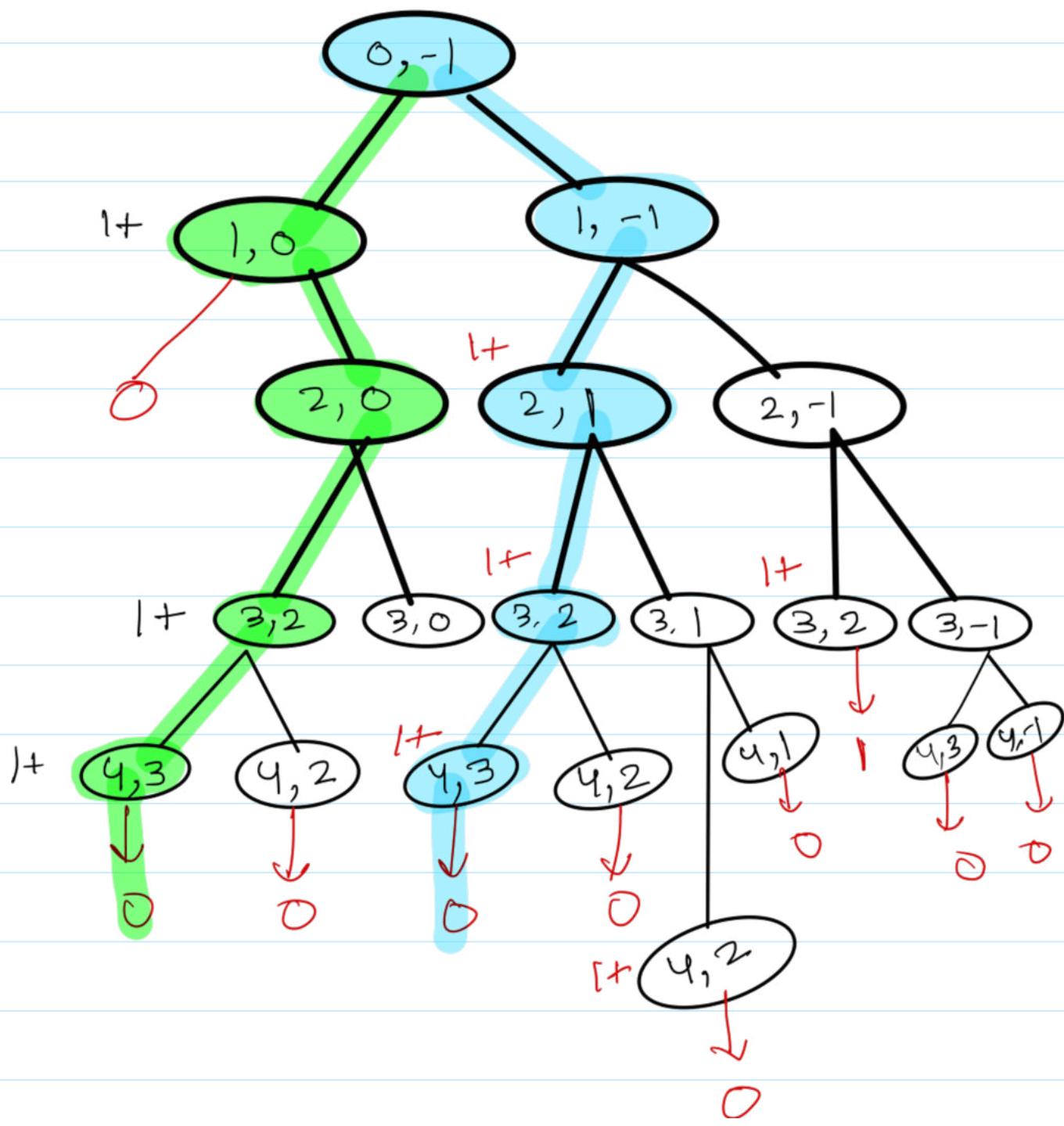
Ex-1 :  $\text{nums} = \{2,1,3,5\}$

There can be 2 LIS of Length 3.  $\{2,3,5\}$  and  $\{1,3,5\}$ .

At every index there are two possibilities:

1. Either we take it
2. Or We don't take it.

We can only take it, if it is greater than previous elements. For first element, since we don't have any previous element, so we have the option that we can either take it or not take it.



## MEMOIZATION:

```
1 int findlis(int currindex, int prev, int size, vector<int>& nums,
2             vector<vector<int>>& dp)
3             { if(currindex==size)
4                 return 0;
5                 if(dp[currindex][prev+1]!=-1)
6                     return dp[currindex][prev+1];
7
8                 int nottake = findlis(currindex+1,prev,size,nums,dp);
9                 int take = 0;
10                if(prev==-1 || nums[currindex]>nums[prev])
11                    take = 1 + findlis(currindex+1,currindex,size,nums,dp);
12                dp[currindex][prev+1] = max(take,nottake);
13            }
14
15        int lengthofLIS(vector<int>& nums) {
16            int size = nums.size();
17            vector<vector<int>> dp(size, vector<int> (size+1,-1));
18            int prev = -1;
19            return findlis(0,prev,size,nums,dp);
20        }
```

TC: O(N\*N)

SC: O(N\*N) +O(N)

## Tabulation:

```
1 int lengthofLIS(vector<int>& nums) {
2         int size = nums.size();
3         vector<vector<int>> dp(size+1, vector<int> (size+1,0));
4         for(int currindex=size-1;currindex>=0;currindex--)
5         {
6             for(int prev = currindex-1;prev>=-1;prev--)
7             {
8                 int nottake = dp[currindex+1][prev+1];
9                 int take = 0;
10                if(prev==-1 || nums[currindex]>nums[prev])
11                    take = 1 + dp[currindex+1][currindex+1];
12                dp[currindex][prev+1] = max(take,nottake);
13            }
14        }
15        return dp[0][0];
16    }
```

TC: O(N\*N)

SC: O(N\*N)

## Space Optimization:



```
1 int lengthOfLIS(vector<int>& nums) {
2     int size = nums.size();
3     vector<int> next(size+1,0);
4     for(int currindex=size-1;currindex>=0;currindex--) {
5         vector<int> curr(size+1,0);
6         for(int prev = currindex-1;prev>=-1;prev--) {
7             int nottake = next[prev+1];
8             int take = 0;
9             if(prev==-1 || nums[currindex]>nums[prev])
10                 take = 1 + next[currindex+1];
11             curr[prev+1] = max(take,nottake);
12         } next = curr;
13     }
14     return next[0];
15 }
```

TC: O(N\*N)

SC: O(N)

**Tabulation Method:** This method will be required, if we want to trace back the LIS.  
(for printing the LIS)



```
1 int lengthOfLIS(vector<int>& nums) {
2     int size = nums.size();
3     vector<int> dp(size,1);
4     int maxlen=1;
5     for(int i=0;i<size;i++)
6     { for(int j=0;j<i;j++)
7         { if(nums[i]>nums[j])
8             {//if we don't take maximum, it will fail for [0,1,0,3,2,3].
9             //dp[0]=1
10            //dp[1]=2 -> [max(1,1+1)=2] at j=0.
11            //dp[2]=1 -> [1]
12            //dp[3]=3 -> [max(2,1+2)=3] at j=1, if we dont take max, it
13             //will run till
14             //j=2, dp[3]=2 ->[1+dp[2]]=1+1=2, so to handle with
15             //decreasing element case, we are using max
16             dp[i]=max(dp[i],1+dp[j]);
17             maxlen=max(dp[i],maxlen);
18         }
19     }
20 }
```

TC: O(N\*N)

SC: O(N)

[LinkedIn/kapilyadav22](https://www.linkedin.com/in/kapilyadav22)

# LECTURE - 42 PRINTING LONGEST INCREASING SUBSEQUENCE

24 June 2022 20:13

Ex -

10	9	2	5	3	7	10	18
0	1	2	3	4	5	6	7

nums

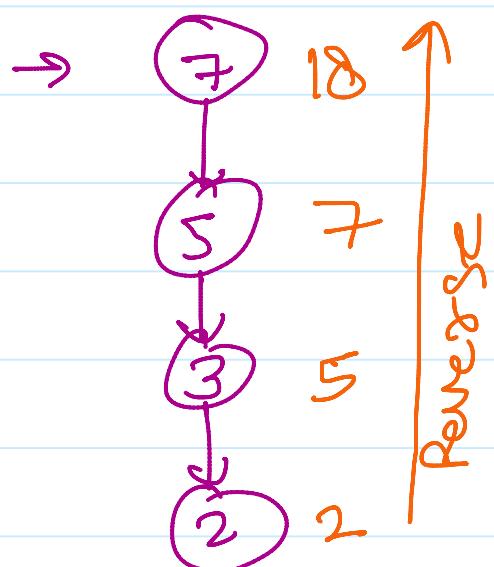
1	1	1	1	1	3	4	4
2	2	2	2	2	3	4	4

dp

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

hash

↑  
last index



→ It will point one of the LIS, to print all the LIS, we need to find all possibilities (Recursion)

**Tabulation:**

**TC : O(N\*N) + O(N)**

**SC: O(N+N+N)**



```
1 int lengthOfLIS(vector<int>& nums) {
2     int size = nums.size();
3     vector<int> dp(size,1), hash(size);
4     int maxlen=1;
5     int lastind=0;
6     for(int i=0;i<size;i++)
7     {   hash[i]=i;
8         for(int j=0;j<i;j++)
9         { if(nums[i]>nums[j] && dp[j]+1>dp[i])
10             {   dp[i]= 1+ dp[j];
11                 hash[i] = j;
12             }
13         }
14         if(dp[i]>maxlen)
15             { maxlen=dp[i];
16                 lastind = i;
17             }
18     }
19
20     //print any of the subsequence
21     vector<int> temp;
22     temp.push_back(nums[lastind]);
23     while(hash[lastind]!=lastind)
24     { lastind = hash[lastind];
25         temp.push_back(nums[lastind]);
26     }
27     reverse(temp.begin(),temp.end());
28     for(int i =0;i<temp.size();i++)
29         cout<<temp[i]<<" ";
30
31     return temp;
32 }
```

**OR**



```
1 int lengthOfLIS(vector<int>& nums) {
2     int size = nums.size();
3     vector<int> dp(size, 1);
4     int maxlen=1;
5     int lastind=0;
6     for(int i=0;i<size;i++)
7     {
8         for(int j=0;j<i;j++)
9         { if(nums[i]>nums[j] && dp[j]+1>dp[i])
10             { dp[i]= 1+ dp[j];
11             }
12         }
13         if(dp[i]>maxlen)
14         { maxlen=dp[i];
15             lastind = i;
16         }
17     }
18
19 //      print any of the subsequence
20     vector<int> temp;
21     temp.push_back(nums[lastind]);
22     for(int i =lastind;i>0;i--)
23     {
24         if(dp[i]==dp[i-1]+1)
25             temp.push_back(nums[i-1]);
26     }
27     reverse(temp.begin(),temp.end());
28     for(int i =0;i<temp.size();i++)
29         cout<<temp[i]<<" ";
30
31
32     return temp;
33 }
```

TC: O(N\*N) +ON  
SC: O(N+N)

[LinkedIn](#)/kapilyadav22

## LECTURE - 43 LONGEST INCREASING SUBSEQUENCE | BINARY SEARCH

24 June 2022 20:13

The TC of previous methods was  $O(N^2)$ .

But if the constraints are  $N = 10^5$ , then

$$N \times N = 10^5 \times 10^5 = 10^{10}, \text{ so}$$

Above methods will give TLE.

[Binary Search - LB, UB].

Ex - [1, 7, 8, 4, 5, 6, -1, 9]  
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

{1, 7, 8, 9} — len = 4.

{1, 4, 5, 6, 9} — len = 5.

{-1, 9} — len = 2.

→ If we find all the increasing subsequence then, we can take longest length subsequence from them.

→ This will take a lot of space, so we will not do it. But we can take intuition from it.

→ Instead of taking all subsequences, we can rewrite in one subsequence.

→ Ex - [1, 7, 8, 4, 5, 6, -1, 9]

Element	Subsequence
1	[1]
7	[1, 7]
8	[1, 7, 8]
• 4	[1, 4, 8]
5	[1, 4, 5]
6	[1, 4, 5, 6]

6	$[1, 4, 5, \textcolor{orange}{6}]$
-1	$[-1, 4, 5, \textcolor{red}{6}]$
9	$[-1, 4, 5, \textcolor{red}{6}, \textcolor{orange}{9}]$

This is not a correct subsequence,  
but the length it will give will be  
the length of LIS.

→ Because we are inserting elements at right place.

## WHY BINARY SEARCH?

when we were inserting 4, the subsequence was  $[1, 7, 8]$ , so we want to insert 4 in between 1 & 8 and array is sort (increasing subsequence).

So, we are using Binary Search.

Ex-2 : arr =  $[1 \downarrow 2 3 5 1 4 7]$

~~1 2 3 5 7~~  
1 4

→ We can use lower-bound.

→ We need either  $\text{arr}[i]$  or first index greater than  $\text{arr}[i]$ .



```
1 int lengthOfLIS(vector<int>& arr) {
2     int n = arr.size();
3     vector<int> temp;
4     temp.push_back(arr[0]);
5     int len=1;
6     for(int i=1;i<n;i++)
7     { if(arr[i]>temp.back())
8     { temp.push_back(arr[i]);
9         len++;
10    }
11    else
12    { int index =
13        lower_bound(temp.begin(), temp.end(),arr[i])-temp.begin();
14        temp[index] = arr[i];
15    }
16    }
17    return len;
18 }
```

# LECTURE - 44 LARGEST DIVISIBLE SUBSET || LIS

24 June 2022 20:14

## 368. Largest Divisible Subset

Medium    3365    155    Add to List    Share

Given a set of **distinct** positive integers `nums`, return the largest subset `answer` such that every pair `(answer[i], answer[j])` of elements in this subset satisfies:

- `answer[i] % answer[j] == 0`, or
- `answer[j] % answer[i] == 0`

If there are multiple solutions, return any of them.

### Example 1:

```
Input: nums = [1,2,3]
Output: [1,2]
Explanation: [1,3] is also accepted.
```

### Example 2:

```
Input: nums = [1,2,4,8]
Output: [1,2,4,8]
```

Ex: `nums: {1,2,4,8}`

$2 \% 1 = 0$ , so we can say  $\{1,2\}$  are divisible subset.

$4 \% 2 = 0$ , so we can say  $\{2,4\}$  are divisible subset.

NOTE: If  $x$  is divisible by  $y$ , and  $y$  is divisible by  $z$ , then  $x$  will also be divisible by  $z$ .

So, we can say  $\{1,2,4\}$  are divisible subset.

Similarly  $8 \% 4 = 0$ , so  $\{1,2,4,8\}$  can be divisible subset.

It is same as LIS, just use  $\text{nums}[i] \% \text{nums}[j] == 0$  in place  $\text{nums}[i] < \text{nums}[j]$ .

**NOTE: if array is not sorted, then sort the array.**

## Tabulation:

[LinkedIn/kapilyadav22](https://www.linkedin.com/in/kapilyadav22)

```
1 vector<int> largestDivisibleSubset(vector<int>& nums)
2     { vector<int> ans;
3
4         int size = nums.size();
5         sort(nums.begin(),nums.end());
6
7         vector <int> dp(size,1);
8         vector<int> hash(size);
9         int maxLength = 1, lastIndex = 0;
10
11        for(int i = 0; i<size; i++) {
12            hash[i]=i;
13            for(int j = 0; j<i; j++) {
14                if(nums[i]%nums[j] == 0 && dp[j]+1>dp[i]) {
15                    dp[i] = 1 + dp[j];
16                    hash[i] = j ;
17                }
18            }
19            if(maxLength<dp[i]) {
20                maxLength = dp[i];
21                lastIndex = i;    }
22        }
23
24        ans.push_back(nums[lastIndex]);
25        while(hash[lastIndex]!=lastIndex)
26        {
27            lastIndex = hash[lastIndex];
28            ans.push_back(nums[lastIndex]);
29        }
30        reverse(ans.begin(),ans.end());
31        return ans;
32    }
```

# LECTURE - 45 LONGEST STRING CHAIN || LIS

24 June 2022 20:15

## 1048. Longest String Chain

Medium 4836 201 Add to List Share

You are given an array of words where each word consists of lowercase English letters.

word<sub>A</sub> is a predecessor of word<sub>B</sub> if and only if we can insert exactly one letter anywhere in word<sub>A</sub> without changing the order of the other characters to make it equal to word<sub>B</sub>.

- For example, "abc" is a predecessor of "abac", while "cba" is not a predecessor of "bcad".

A word chain is a sequence of words [word<sub>1</sub>, word<sub>2</sub>, ..., word<sub>k</sub>] with k >= 1, where word<sub>1</sub> is a predecessor of word<sub>2</sub>, word<sub>2</sub> is a predecessor of word<sub>3</sub>, and so on. A single word is trivially a word chain with k == 1.

Return the length of the longest possible word chain with words chosen from the given list of words.

### Example 1:

```
Input: words = ["a", "b", "ba", "bca", "bda", "bdca"]
Output: 4
Explanation: One of the longest word chains is ["a", "ba", "bda", "bdca"].
```

### Example 2:

```
Input: words = ["xbc", "pcxbcf", "xb", "cxbc", "pcxbc"]
Output: 5
Explanation: All the words can be put in a word chain ["xb", "xb\u0303", "\u0303xbc", "pcxbc", "pcxbcf"].
```

ex-  $\left[ \begin{smallmatrix} "a", "b", "ba", "bca", "bda", "bdca \end{smallmatrix} \right]$

+1      +1      +1

=  $\left[ \begin{smallmatrix} a, b\u0303, b\u0303a, b\u0303c\u0303a, b\u0303d\u0303a, b\u0303d\u0303c\u0303a \end{smallmatrix} \right]$  can be one chain.

→ It is same as LIS, but here we need to do string matching as well, so for that create a function checkPossible to check, whether a particular string can be in the chain or not.

NOTE :- It can be possible that the strings may not be in the increasing order of their length. so Sort the array in increasing order.

## Tabulation:

```
● ○ ●

1 bool checkPossible(string &s1, string &s2)
2     { int size1 = s1.size();
3         int size2 = s2.size();
4         int first = 0;
5         int second =0;
6
7         if(size1!=(size2+1))
8             return false;
9
10        while(first < size1)
11        { if(s1[first]==s2[second])
12            { first++;
13                second++; }
14            else first++;
15        }
16
17        if(first==size1 && second==size2)
18            return true;
19        return false;
20    }
21
22    static bool comp(string &s1, string &s2)
23    { return (s1.size()<s2.size()); }
24
25
26    int longestStrChain(vector<string>& words) {
27        sort(words.begin(), words.end(),comp);
28        int size = words.size();
29        vector<int> dp(size,1);
30        int maxlen=1;
31        for(int i=0;i<size;i++)
32        { for(int j=0;j<i;j++)
33            { if(checkPossible(words[i],words[j]))
34                {dp[i]=max(dp[i],1+dp[j]);}
35            }
36        }
37        maxlen=max(dp[i],maxlen);
```

```
36         maxlen=max(dp[i],maxlen);
37     }
38     return maxlen;
39 }
```

[LinkedIn](#)/kapilyadav22

## LECTURE - 46 LONGEST BITONIC SUBSEQUENCE

24 June 2022 20:16

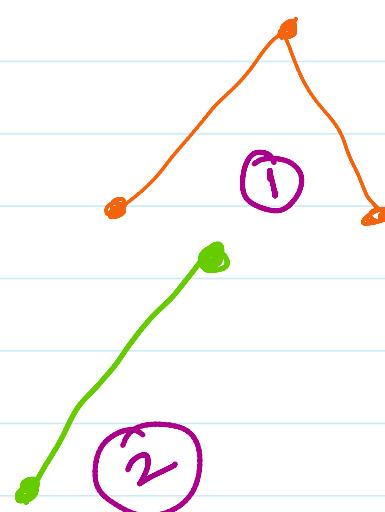
$$\textcircled{1} \text{ arr[]} = \{1, 11, 2, 10, 4, 5, 2, 1\}$$

$$\rightarrow \underline{\underline{1 \ 2 \ 10}} \quad \underline{\underline{4 \ 2 \ 1}}$$

$$\rightarrow \underline{\underline{1 \ 2 \ 10}} \quad \underline{\underline{5 \ 2 \ 1}}$$

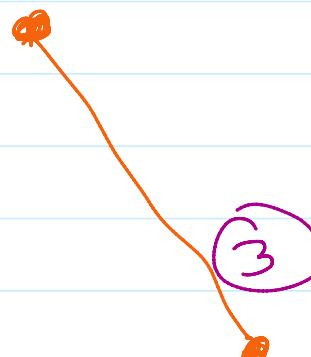
$$\rightarrow \underline{\underline{1 \ 11 \ 10}} \quad \underline{\underline{4 \ 2 \ 1}}$$

$$\rightarrow \underline{\underline{1 \ 11 \ 10}} \quad \underline{\underline{5 \ 2 \ 1}}$$



$$\textcircled{2} \text{ arr[]} = \{1, 2, 3, 4, 5, 6\}$$

$$\textcircled{3} \text{ arr[]} = \{6, 5, 4, 3, 2, 1\}$$



Ex - arr[] = {1, 11, 2, 10, 4, 5, 2, 1}

$$dp[] = \{ \underline{\underline{1 \ 2 \ 2}} \quad \textcircled{3} \ 3 \ 4 \ 2 \ 1 \}$$

$$dp2[] = \{ \underline{\underline{1 \ 5 \ 2}} \quad \textcircled{4} \ 3 \ 3 \ 2 \ 1 \}$$

← -1 →

$$\text{Bitonic} = \overbrace{\underline{\underline{1 \ 6 \ 3 \ 6 \ 5 \ 6 \ 3 \ 1}}}^{\text{↑}}$$

Use a variation of LIS? -



```
1 int longestBitonicSequence(vector<int>& nums, int n)
2 {
3     vector<int> dp1(n, 1);
4     for(int i=0; i<n; i++)
5     {   for(int j=0; j<i; j++)
6         { if(nums[i]>nums[j])
7             dp1[i]=max(dp1[i], 1+dp1[j]);
8         }
9     vector<int> dp2(n, 1);
10    for(int i=n-1; i>=0; i--)
11    {   for(int j=n-1; j>i; j--)
12        { if(nums[i]>nums[j])
13            dp2[i]=max(dp2[i], 1+dp2[j]);
14        }
15    }
16    int maxi=0;
17    for(int i =0; i<n; i++)
18    { maxi = max(maxi, dp1[i]+dp2[i]-1);
19    }
20    return maxi;
21 }
22
```

Variation of this question on Leetcode

## 1671. Minimum Number of Removals to Make Mountain Array

Hard    837    11    Add to List    Share

You may recall that an array `arr` is a **mountain array** if and only if:

- `arr.length >= 3`
- There exists some index `i` (**0-indexed**) with  $0 < i < \text{arr.length} - 1$  such that:
  - $\text{arr}[0] < \text{arr}[1] < \dots < \text{arr}[i - 1] < \text{arr}[i]$
  - $\text{arr}[i] > \text{arr}[i + 1] > \dots > \text{arr}[\text{arr.length} - 1]$

Given an integer array `nums`, return the **minimum number of elements to remove** to make `nums` a **mountain array**.

### Example 1:

Input: `nums = [1,3,1]`

Output: 0

Explanation: The array itself is a mountain array so we do not need to remove any elements.

In this question, we need to find the count of elements, which are not participating in the Longest Bitonic Subsequence, the only difference is , in this question, The mountain should be in strictly increasing and then strictly decreasing order.

NOTE: Only Strictly increasing or only strictly decreasing mountain will not work

EX: `arr[] = [9,8,1,7,6,5,4,3,2,1]`

In this question, we need to remove two elements which are 9 and 8, to make it a mountain. There will be one modification only.

$arr = \{9, 8, 1, 7, 6, 5, 4, 3, 2, 1\}$

$dp1 = \{1, 1, 1, 2, 2, 2, 2, 2, 2, 1\}$  →

$dp2 = \{9, 8, 1, 7, 6, 5, 4, 3, 2, 1\}$  ←

$maxi = \{1, 1, 1, \underline{\underline{8}}, 7, 6, 5, 4, 3, 1\}$

remove

$$N = 10$$

$$\text{so, } N - maxi = 10 - 8 = 2.$$



```
1 int minimumMountainRemovals(vector<int>& nums) {
2     int n = nums.size();
3     vector<int> dp1(n, 1);
4     for(int i=0; i<n; i++)
5     {   for(int j=0; j<i; j++)
6         { if(nums[i]>nums[j])
7             dp1[i]=max(dp1[i], 1+dp1[j]);
8         }
9     }
10    vector<int> dp2(n, 1);
11    for(int i=n-1; i>=0; i--)
12    {   for(int j=n-1; j>i; j--)
13        { if(nums[i]>nums[j])
14            dp2[i]=max(dp2[i], 1+dp2[j]);
15        }
16    }
17    int maxi=0;
18    for(int i=0; i<n; i++)
19    { //cout<<dp1[i]<<" "<<dp2[i]<<" "<<endl;
20        if(dp1[i] > 1 && dp2[i] > 1)
21            maxi = max(maxi, dp1[i]+dp2[i]-1);
22    }
23    return n-maxi;
24 }
```

[LinkedIn](#)/kapilyadav22

# LECTURE - 47 NUMBER OF LONGEST INCREASING SUBSEQUENCES || LIS

24 June 2022 20:16

$$\text{arr}[] \rightarrow \{1, 5, 4, 3, 2, 6, 7, 10, 8, 9\}$$

$$dp[] = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$$

$$cnt[] = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$$

initially :-

$$\text{arr}[] \rightarrow \{1, 5, 4, 3, 2, 6, 7, 10, 8, 9\}$$

$$dp[] = \{1, 1\}$$

$$cnt[] = \{1, 1\}$$

for i=1, j=0

$$\text{arr}[] \rightarrow \{1, 5, 4, 3, 2, 6, 7, 10, 8, 9\}$$

Since,  $\text{arr}[i] > \text{arr}[j]$  and  $dp[i] < 1 + dp[j]$

length of the lis will increase but not the count.

$$dp[] = \{1, 2\}$$

$$[1, 5]$$

for i=2, j=0

$\text{arr}[i] > \text{arr}[j]$ , and  $dp[2] < 1 + dp[0]$

$$dp[] = \{1, 2, 2\}$$

$$[1, 4]$$

$$cnt[] = \{1, 1, 1\}$$

$j=1$ ,  $\text{arr}[i] < \text{arr}[j]$ , do nothing.

$$dp[] = \{1, 2, 2, 2\}$$

$$[1, 4]$$

$$cnt[] = \{1, 1, 1, 1\}$$

for i=3, j=0

$\text{arr}[i] > \text{arr}[j]$ , and  $dp[3] < 1 + dp[0]$

$$dp[] = \{1, 2, 2, 2, 2\}$$

$$cnt[] = \{1, 1, 1, 1, 1\}$$

$j=1$ ,  $\text{arr}[i] < \text{arr}[j]$ , do nothing.

$j=2$ ,  $\times$

for i=4, j=0

$\text{arr}[i] > \text{arr}[j]$  and  $dp[4] < 1 + dp[0]$

$$dp[] = \{1, 2, 2, 2, 2, 2\}$$

$$1 < 1+1$$

400 L1 - 400 L2 - 100 L3 - 100 L4 - 100 L5

$$dp[] = \{1, 2, 2, 2, 2\}$$

$$cnt[] = \{1, 1, 1, 1\}$$

$j=1, x$

$j=2, x$

$j=3, x$

for  $i=5, j=0$

$l < i+j$

$$arr[i] > arr[j] \& dp[5] < l + dp[0]$$

$$dp[] = \{1, 2, 2, 2, 2\}$$

$$cnt[] = \{1, 1, 1, 1, 1\}$$

$j=1, arr[i] > arr[j] \& dp[5] < l + dp[1]$

$$dp[] = \{1, 2, 2, 2, 2\}$$

$$cnt[] = \{1, 1, 1, 1, 1\}$$

$j=2, arr[i] > arr[j] \& dp[5] == l + dp[2]$

$$dp[] = \{1, 2, 2, 2, 3\}$$

$$cnt[] = \{1, 1, 1, 1, 2\}$$

$j=3, arr[i] > arr[j] \& dp[5] == l + dp[3]$

$$dp[] = \{1, 2, 2, 2, 3\}$$

$$cnt[] = \{1, 1, 1, 1, 3\}$$

$j=4, arr[i] > arr[j] \& dp[5] == l + dp[4]$

$$dp[] = \{1, 2, 2, 2, 3\}$$

$$cnt[] = \{1, 1, 1, 1, 4\}$$

⋮

and so on.

1.) We are only updating  $dp[i]$  when  $arr[i] > arr[j]$ ,

$$\text{&} \quad dp[j]+1 < dp[i]$$

$$\textcircled{1} \quad dp[i] = 1 + dp[j];$$

$$\textcircled{2} \quad cnt[i] = cnt[j];$$

(2)  $arr[i] > arr[j] \Rightarrow dp[j]+1 = dp[i]$

update the  $cnt$ ;

$$cnt[i] += cnt[j];$$

(3) And size the maxlen;

(4) At last sum all the No of LIS at the positions having LIS.

$$dp = \{1, 2, 2, 3, 4, 4\}$$

$$cnt = \{1, 1, 1, 1, 5, 4\}$$

→ The maxlen = 4, and there are  $5+4=9$ , lis of length 4.



```
1 int findNumber0fLIS(vector<int>& nums) {
2     int size = nums.size();
3     int maxlen=1;
4     vector<int> dp(size,1);
5     vector<int> count(size,1);
6
7     for(int i=1;i<size;i++)
8     {   for(int j=0;j<i;j++)
9         { if(nums[i]>nums[j]  && dp[i]<dp[j]+1)
10             {   dp[i]=dp[j]+1;
11                 count[i]=count[j];
12             }
13             else if(nums[i]>nums[j]  && dp[i]==dp[j]+1)
14                 count[i]+=count[j];
15
16         }
17         maxlen = max(maxlen,dp[i]);
18     }
19
20     int nolis=0;
21     for(int i=0;i<size;i++)
22     { if(dp[i]==maxlen)
23         nolis+=count[i];
24     }
25
26     return nolis;
27 }
```

## LECTURE - 48 & 49 MATRIX CHAIN MULTIPLICATION

Thursday, 16 June 2022 9:13 PM

### Matrix Chain Multiplication

**Hard** Accuracy: 59.72% Submissions: 35076 Points: 8

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The efficient way is the one that involves the least number of multiplications.

The dimensions of the matrices are given in an array `arr[]` of size `N` (such that `N = number of matrices + 1`) where the  $i^{\text{th}}$  matrix has the dimensions (`arr[i-1] x arr[i]`).

**Example 1:**

**Input:**  $N = 5$

`arr = {40, 20, 30, 10, 30}`

**Output:** 26000

**Explanation:** There are 4 matrices of dimension  $40 \times 20$ ,  $20 \times 30$ ,  $30 \times 10$ ,  $10 \times 30$ . Say the matrices are named as A, B, C, D. Out of all possible combinations, the most efficient way is  $(A * (B * C)) * D$ .

The number of operations are -

$$20 * 30 * 10 + 40 * 20 * 10 + 40 * 10 * 30 = 26000.$$

$$\begin{aligned} & [(1+2+3) \times 5] \\ & [(1+2) \times (3 \times 5)] \end{aligned}$$

A B C

$$A = 10 \times 30$$

$$B = 30 \times 5$$

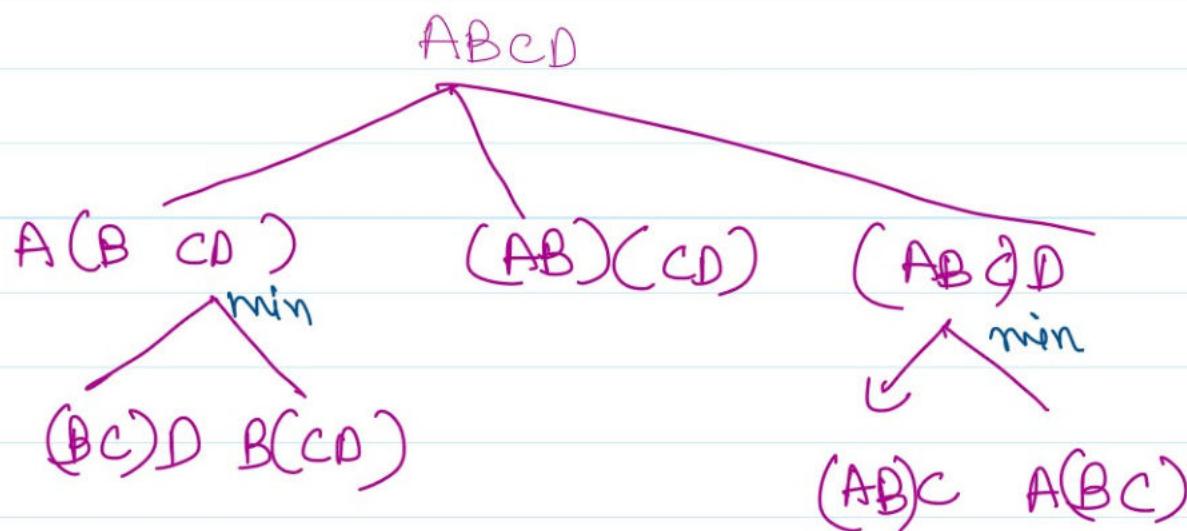
$$C = 5 \times 60$$

$$\rightarrow \text{operations} = 10 \times 30 \times 5 = \underline{\underline{1500}}$$

$$\begin{aligned} (A B) C &= (10 \times 30 \times 5) + (10 \times 5 \times 60) \\ &= 1500 + 3000 = \underline{\underline{4500}}. \underline{\underline{\text{Min operations}}} \end{aligned}$$

$$\begin{aligned} \underline{A(BC)} &= (10 \times 30 \times 60) + (30 \times 5 \times 60) \\ &= 18000 + 18000 = 27,000. \end{aligned}$$

Q.) Given the N matrix dimension, tell me the min cost to multiply them to a single one.

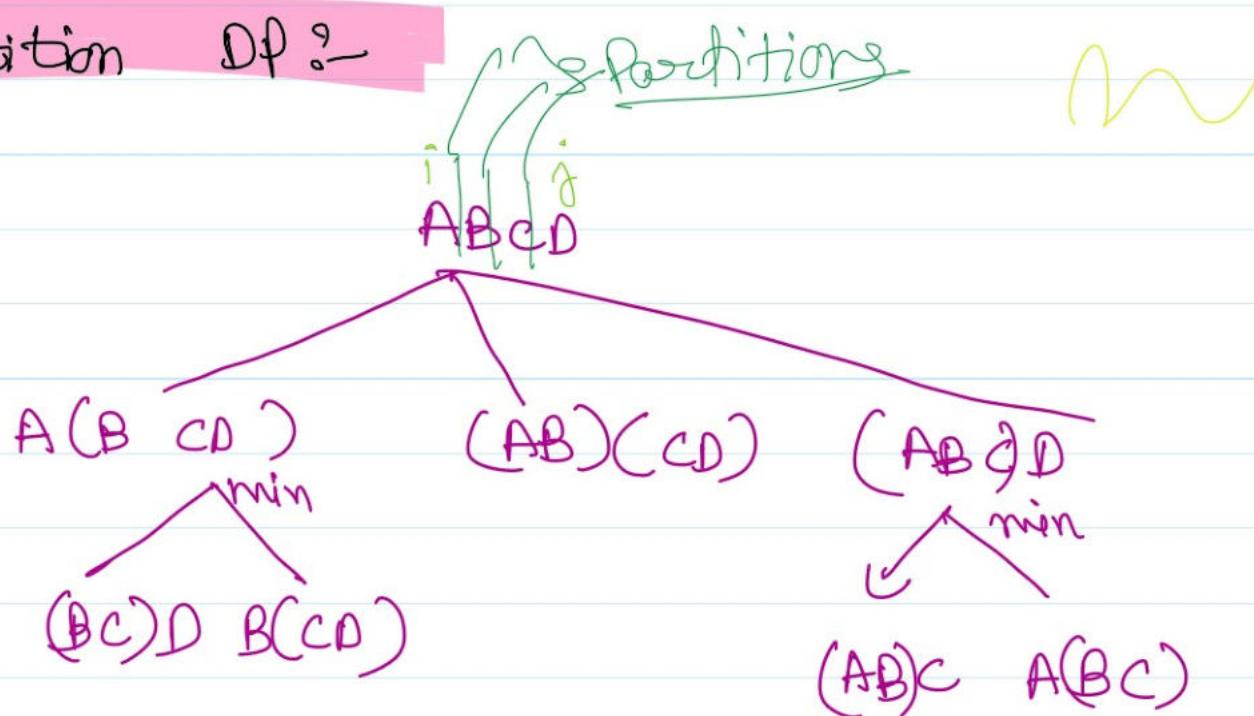


Given :- arr [] = {10, 20, 30, 40, 50}  
N=5      dimension of N-1 matrix

$$\begin{aligned} A &\rightarrow 10 \times 20 \\ B &\rightarrow 20 \times 30 \\ C &\rightarrow 30 \times 40 \\ D &\rightarrow 40 \times 50 \end{aligned}$$

$$\begin{aligned} 1^{\text{st}} &\Rightarrow A[0] \times A[1] \\ 2^{\text{nd}} &\Rightarrow A[1] \times A[2] \\ 3^{\text{rd}} &\Rightarrow A[2] \times A[3] \\ i^{\text{th}} &\Rightarrow A[i-1] \times A[i] \end{aligned}$$

Partition DP ?



→ start with entire block/array  $f(i, j)$

→ Try all partitions.

↪ run a loop to try all partitions.

→ Return the best possible two partitions

$$\underline{A} \underline{(BCD)}$$

1 2

$$\underline{(AB)} \underline{(CD)}$$

1 2

$$\underline{(ABC)} \underline{D}$$

1 2

$$arr[] = \{ \begin{matrix} 0 & 1 & 2 & 3 & 4 \\ 10, 20, 30, 40, 50 \end{matrix} \}$$

A B C D

i j  
↓  
n-1

$f(1, n-1)$  → return the min multiplications  
to multiply matrix 1 → matrix 4

$f(i, j)$

{ if ( $i == j$ ) action 0;

$$\begin{matrix} i \\ [10 & 20 & 30 & 40 & 50] \\ A & B & C & D \end{matrix}$$

$k = (i \rightarrow j - 1)$

↪  $f(i, k), f(k+1, j)$

$k=1 \quad f(1, 1), f(2, 4)$   
 $k=2 \quad f(1, 2), f(3, 4)$

$x=2 \quad f(1,2), f(3,4)$   
 $x=3 \quad f(1,3), f(4,4).$

OR

$k = (i+1) \rightarrow j$

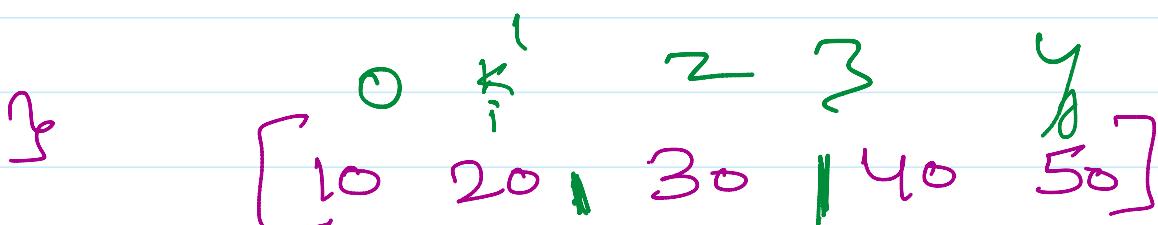
$f(i, k-1), f(k, j)$

$k=2 \quad f(1,1), f(2,4)$   
 $k=3 \quad f(1,2), f(3,4)$   
 $k=4 \quad f(1,3), f(4,4)$

$\min_i = \text{INT-MAX}$   
 $\text{for } (k = i \text{ to } k = j-1)$

$$\{ \underline{\text{steps}} = (\underbrace{A[i-1] \times \alpha[k] \times \alpha[j]}_{+ f(i, k) + f(k+1, j)})$$

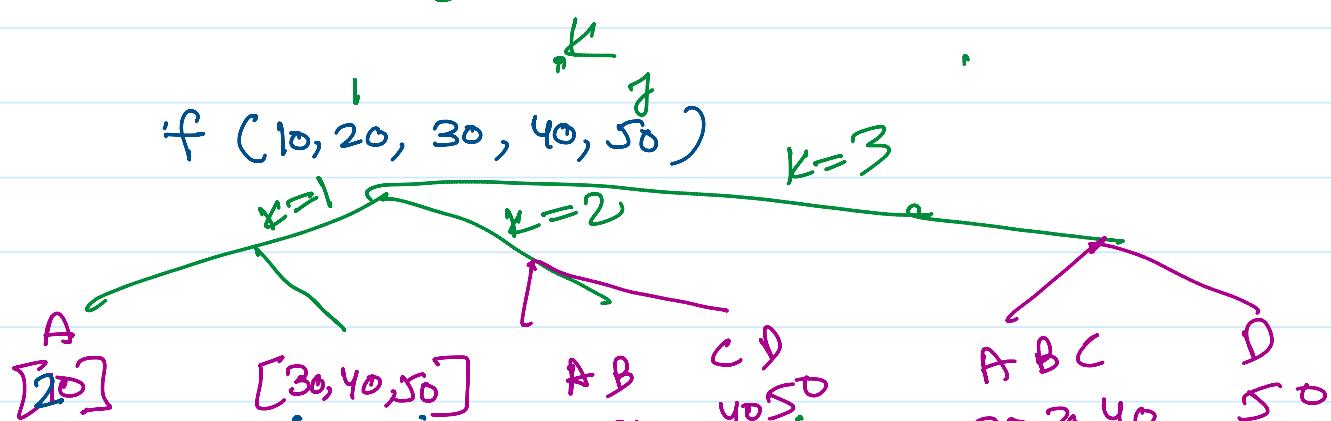
$$\min_i = \min(\min_i, \text{steps});$$

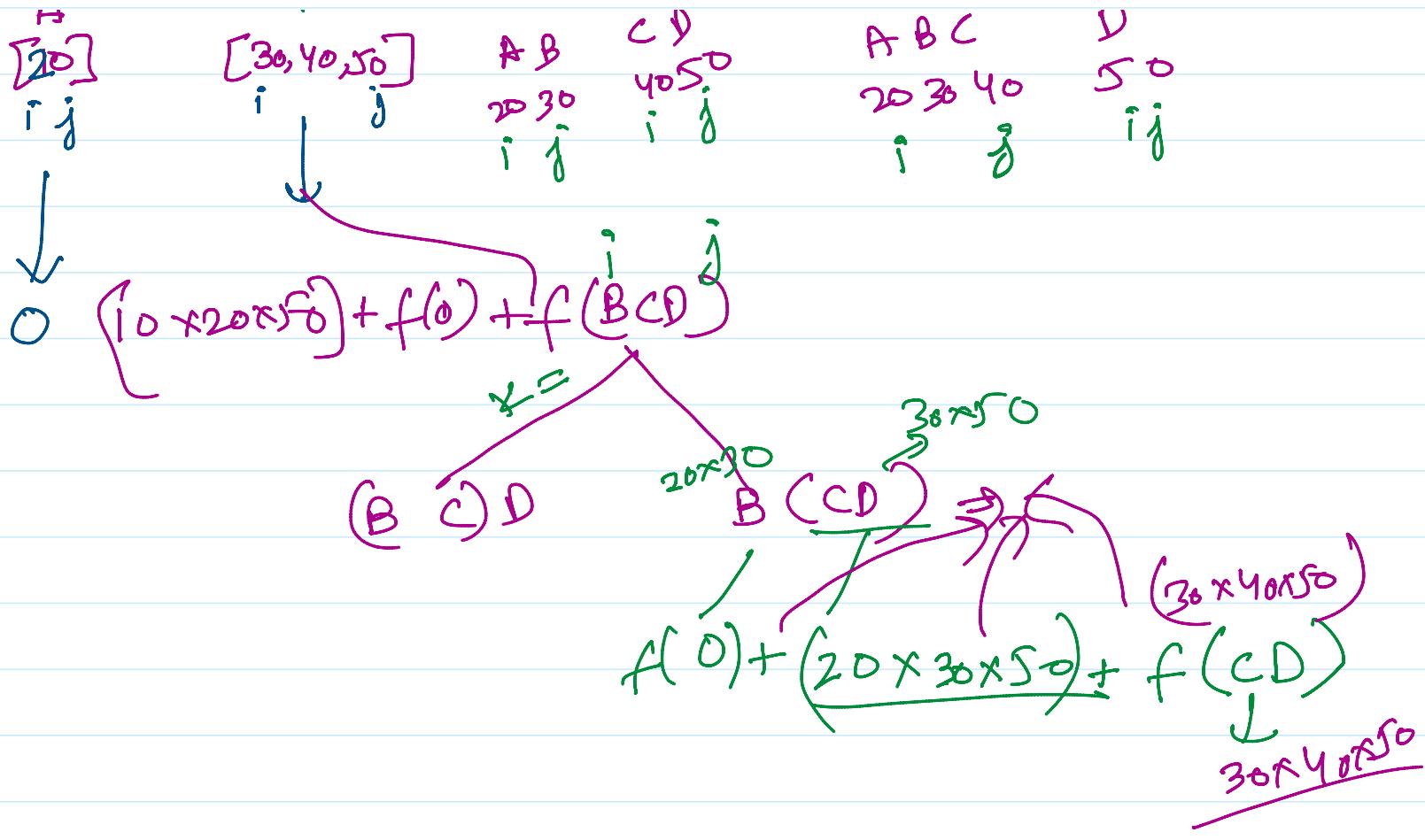


$\Rightarrow (A)(B \ C \ D)$   
 $10 \times 20 \rightarrow 20 \times 50$

$$\Rightarrow \underbrace{10 \times 20 \times 50} =$$

$A[i-1] \ A[k] \ A[j] + (\text{solving out})$





Tabulation :- (Bottom Up DP)

Rules :-

1.) Copy the Base Case.

dp[N][N]

for(int i=0 ; i<N ; i++)

dp[i][i] = 0.

2.) Write Down the changing parameters.

i = (n-1 → 1)

j = i+1 to n-1.



```
1 int f(int i,int j, int arr[],vector<vector<int>> &dp)
2 {
3     if(i==j)
4         return 0;
5     int mini = INT_MAX;
6     if(dp[i][j]!=-1)
7         return dp[i][j];
8     for(int k=i;k<j;k++)
9     {   int steps = arr[i-1]*arr[k]*arr[j] +
10        f(i,k,arr,dp) + f(k+1,j,arr,dp);
11         mini = min(mini,steps);
12     }
13 }
14 int matrixMultiplication(int N, int arr[])
15 { vector<vector<int>> dp(N, vector<int> (N,-1));
16     return f(1,N-1,arr,dp);
17 }
```

Tabulation:

[LinkedIn/kapilyadav22](https://www.linkedin.com/in/kapilyadav22)



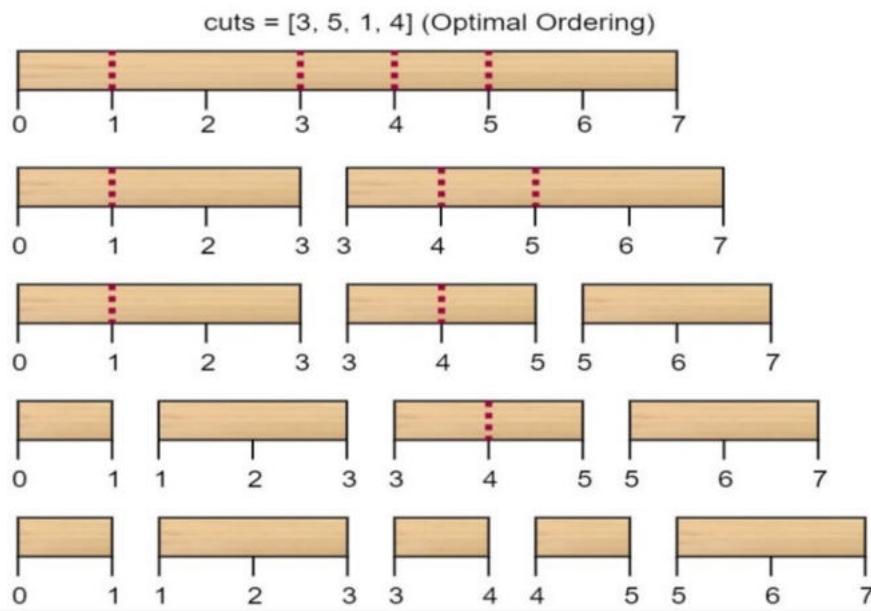
```
1 int matrixMultiplication(int N, int arr[])
2 { vector<vector<int>> dp(N, vector<int> (N));
3     for(int i =1;i<N;i++)
4         dp[i][i]=0;
5     for(int i=N-2;i>=1;i--)
6     { for(int j=i+1;j<N;j++)
7         { int mini = INT_MAX;
8             for(int k=i;k<j;k++)
9                 { int steps = arr[i-1]*arr[k]*arr[j] + dp[i][k] +dp[k+1][j];
10                    mini = min(mini,steps);
11                }
12             dp[i][j] = mini;
13         }
14     }
15     return dp[1][N-1];
16 }
```

# LECTURE - 50 MINIMUM COST TO CUT THE STICK

Friday, 17 June 2022 12:49 AM

(Pattern DP)

## Example 1:

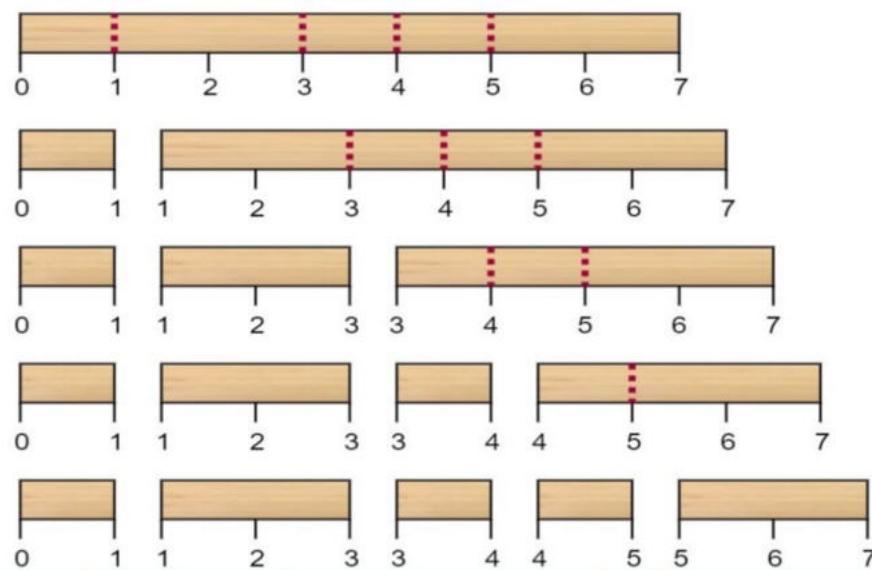


**Input:** n = 7, cuts = [1, 3, 4, 5]

**Output:** 16

**Explanation:** Using cuts order = [1, 3, 4, 5] as in the input leads to the following scenario:

cuts = [1, 3, 4, 5]



The first cut is done to a rod of length 7 so the cost is 7. The second cut is done to a rod of length 6 (i.e. the second part of the first cut), the third is done to a rod of length 4 and the last cut is to a rod of length 3. The total cost is  $7 + 6 + 4 + 3 = 20$ .

Rearranging the cuts to be [3, 5, 1, 4] for example will lead to a scenario with total cost = 16 (as shown in the example photo  $7 + 4 + 3 + 2 = 16$ ).

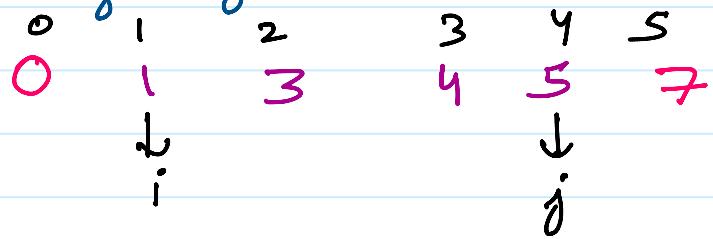
Cost = length of that stick in which  
you.

0 1 3 4 5 7  
i-1 i j j+1

⇒ cuts[j+1] - cuts[i-1]

Ex  $\rightarrow$  cuts = 1 3 4 5

$\rightarrow$  insert 0 at the beginning and length of the stick at the end.



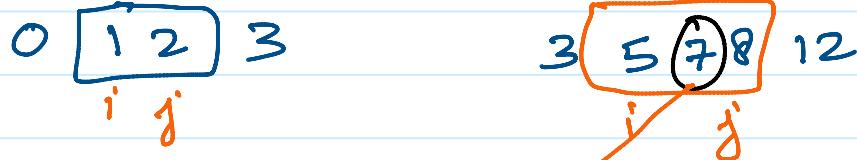
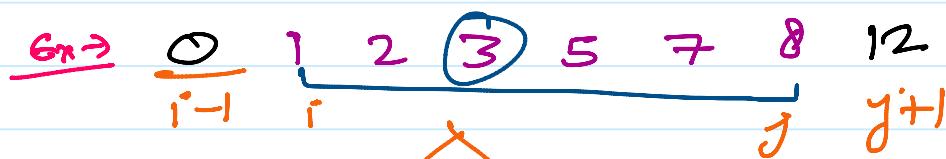
$\rightarrow f(i, j)$

{ if ( $i > j$ ) return 0;

$f(\text{ind} = i \rightarrow j)$

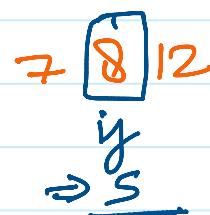
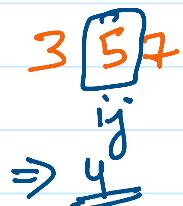
{ cost = cuts[j+1] - cuts[i-1]  
 $+ f(i, \text{ind}-1) + f(\text{ind}+1, j)$

}  $\min_i = \min(\min_i, \text{cost});$



$$3 - 0 = 3$$

$$12 - 3 = 9$$



Memoization:

```

1 int f(int i,int j,vector<int> & cuts,vector<vector<int>> &dp)
2     {   if(i>j) return 0;
3         int mini = INT_MAX;
4         if(dp[i][j]!=-1)
5             return dp[i][j];
6             for(int ind =i;ind<=j;ind++)
7             { int cost = cuts[j+1]-cuts[i-1] + f(i,ind-1,cuts,dp) +
8                 f(ind+1,j,cuts,dp);
9                 mini = min(mini,cost);
10            }
11        return dp[i][j]=mini;
12    }
12 int minCost(int n, vector<int>& cuts) {
13     int size = cuts.size();
14     vector<vector<int>> dp(size+1,vector<int> (size+1,-1));
15     vector<int> newcuts= cuts;
16     newcuts.push_back(n);
17     newcuts.insert(newcuts.begin(),0);
18     sort(newcuts.begin(),newcuts.end());
19     return f(1,size,newcuts,dp);
20 }
```

[LinkedIn/kapilyadav22](#)

### Tabulation:

```

1 int minCost(int n, vector<int>& cuts) {
2     int size = cuts.size();
3     vector<vector<int>> dp(size+2,vector<int> (size+2,0));
4     vector<int> newcuts= cuts;
5     newcuts.push_back(n);
6     newcuts.insert(newcuts.begin(),0);
7     sort(newcuts.begin(),newcuts.end());
8
9     for(int i =size;i>=1;i--)
10     { for(int j =1;j<=size;j++)
11     {   if(i>j)continue;
12         int mini = INT_MAX;
13         for(int ind =i;ind<=j;ind++)
14         { int cost = newcuts[j+1]-newcuts[i-1] + dp[i][ind-1] +
15             dp[ind+1][j];
16                 mini = min(mini,cost);
17             }
18             dp[i][j]=mini;
19         }
20     return dp[1][size];
21 }
```

# LECTURE - 51 BURST BALLOONS (Partition DP)

24 June 2022 20:22

[LinkedIn/kapilyadav22](#)

## 312. Burst Balloons

Hard ⚡ 6229 🗂 160 Add to List Share

You are given  $n$  balloons, indexed from  $0$  to  $n - 1$ . Each balloon is painted with a number on it represented by an array `nums`. You are asked to burst all the balloons.

If you burst the  $i^{\text{th}}$  balloon, you will get  $\text{nums}[i - 1] * \text{nums}[i] * \text{nums}[i + 1]$  coins. If  $i - 1$  or  $i + 1$  goes out of bounds of the array, then treat it as if there is a balloon with a  $1$  painted on it.

Return the maximum coins you can collect by bursting the balloons wisely.

### Example 1:

```
Input: nums = [3,1,5,8]
Output: 167
Explanation:
nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []
coins = 3*1*5 + 3*5*8 + 1*3*8 + 1*8*1 = 167
```

### Example 2:

```
Input: nums = [1,5]
Output: 10
```

$$\rightarrow \{3, 1, 5, 8\}$$

$\rightarrow$  If I burst 3, I will get  $1 \times 3 \times 1 = 3$

$$\{1, 5, 8\}$$

$\rightarrow$  If I burst 1, I will get  $1 \times 1 \times 5 = 5$

$$\{5, 8\}$$

$\rightarrow$  If I burst 5, I will get  $1 \times 5 \times 8 = 40$

$$\{8\}$$

$\rightarrow$  If I burst 8, I will get  $1 \times 8 \times 1 = 8$

$$\underline{56}$$

Another way:-

$$\{3, \cancel{1}, 5, 8\}$$

1

$$3 \times 1 \times 5 = 15$$

2, 3, 5, 8



$$3 \times 1 \times 5 = 15$$

{ 3, 5, 8 }



$$3 \times 5 \times 8 = 120$$

{ 5, 8 }



$$1 \times 3 \times 8 = 24$$

{ 8 }

$$1 \times 8 \times 1 = \frac{8}{167}$$

→ If we consider our array elements like

{ b<sub>1</sub> b<sub>2</sub> **b<sub>3</sub>** b<sub>4</sub> b<sub>5</sub> b<sub>6</sub> }

→ If we take b<sub>3</sub> first, can we solve for { b<sub>1</sub>, b<sub>2</sub> } & { b<sub>4</sub>, b<sub>5</sub>, b<sub>6</sub> } separately.

→ NO. WHY?

→ Let's say we solve for b<sub>1</sub>, b<sub>2</sub>, and we choose b<sub>2</sub>, the right of b<sub>2</sub> is b<sub>4</sub> after bursting b<sub>3</sub>. so b<sub>2</sub> is dependent on b<sub>4</sub> so while bursting b<sub>2</sub>, we will take

$$\underline{b_1 \times b_2 \times b_4} \text{ or}$$

→ if i burst out b<sub>4</sub> after b<sub>3</sub>, then, b<sub>4</sub> is dependent on b<sub>2</sub>.

$$\underline{\underline{b_2 \times b_4 \times b_5}}$$

→ We can run our conditions in reverse order and the Subproblems will be independent.

{ 3, 5, 8 }



$$3 \times 1 \times 5 = 15$$

{ 3, \*, 5, 8 }



$$3 \times 1 \times 5 = 15$$

{ 3, \*, 8 }



$$3 \times 5 \times 8 = 120$$

{ \*, 8 }

$$1 \times 3 \times 8 = 24$$



{ 8 }

$$1 \times 8 \times 1 =$$

$$\underline{167}$$

Gx3

{ b<sub>1</sub>, b<sub>2</sub>, ~~b<sub>3</sub>~~, b<sub>4</sub>, b<sub>5</sub>, b<sub>6</sub> }

i      |      ind      j

$$\Rightarrow a[i-1] \times a[ind] \times a[j+1] +$$

$$f(i, ind-1) + f(ind+1, j).$$


---

{ b<sub>1</sub>, b<sub>2</sub>, b<sub>3</sub>, ~~b<sub>4</sub>~~, b<sub>5</sub>, b<sub>6</sub> }



:

[ b<sub>1</sub>, b<sub>4</sub> ] or [ b<sub>2</sub>, b<sub>4</sub> ] or [ b<sub>3</sub>, b<sub>4</sub> ] or [ b<sub>4</sub>, b<sub>5</sub> ]  
or [ b<sub>4</sub>, b<sub>6</sub> ]

{ b<sub>4</sub> } → b<sub>4</sub> will be in  
second last burst

Recursion ? - TC = Exponential

## Memoization:



```
1 int findmaxcoins(int startind,int endind,vector<int>& nums,
      vector<vector<int>>& dp)
2     { if(startind>endind)
3         return 0;
4     if(dp[startind][endind]!=-1)
5         return dp[startind][endind];
6     int maxi = INT_MIN;
7     for(int ind = startind;ind<=endind;ind++)
8     {
9         int cost = nums[startind-1] * nums[ind]* nums[endind+1] +
10            findmaxcoins(startind, ind-1,nums,dp) +
11            findmaxcoins(ind+1, endind,nums,dp);
12         maxi = max(maxi,cost);
13     }
14     return dp[startind][endind]=maxi;
15
16
17     int maxCoins(vector<int>& nums) {
18         int n = nums.size();
19         nums.push_back(1);
20         vector<vector<int>> dp(n+1, vector<int> (n+1,-1));
21         nums.insert(nums.begin(),1);
22         return findmaxcoins(1,n,nums,dp);
23     }
```

TC: O(N<sup>3</sup>)

SC : O(N<sup>2</sup>) + O(N)

## Tabulation:



```
1 int maxCoins(vector<int>& nums) {
2     int n = nums.size();
3     nums.push_back(1);
4     nums.insert(nums.begin(),1);
5     vector<vector<int>> dp(n+2, vector<int> (n+2,0));
6
7     for(int startind=n;startind>=1;startind--)
8     { //endind==n
9         for(int endind=1;endind<=n;endind++)
10         { int maxi = INT_MIN;
11             if(startind>endind) continue;
12             for(int ind = startind;ind<=endind;ind++)
13             {
14                 int cost = nums[startind-1] * nums[ind]*
15                     nums[endind+1] + dp[startind][ind-1] + dp[ind+1][endind];
16                 maxi = max(maxi,cost);
17             }
18             dp[startind][endind]=maxi;
19         }
20     return dp[1][n];
21 }
```

**TC:**  $O(N^3)$   
**SC :**  $O(N^2)$

# LECTURE-52 EVALUATE BOOLEAN EXPRESSION TO TRUE

24 June 2022 20:24

22

## Boolean Parenthesization

Hard Accuracy: 49.75% Submissions: 37819 Points: 8

Given a boolean expression  $S$  of length  $N$  with following symbols.

Symbols

'T' ---> true

'F' ---> false

and following operators filled between symbols

Operators

& ---> boolean AND

| ---> boolean OR

^ ---> boolean XOR

Count the number of ways we can parenthesize the expression so that the value of expression evaluates to true.

Example 1:

**Input:**  $N = 7$

$S = T|T\&F^T$

**Output:** 4

**Explanation:** The expression evaluates to true in 4 ways  $((T|T)\&(F^T))$ ,  $(T|(T\&(F^T)))$ ,  $(((T|T)\&F)^T)$  and  $(T|((T\&F)^T))$ .

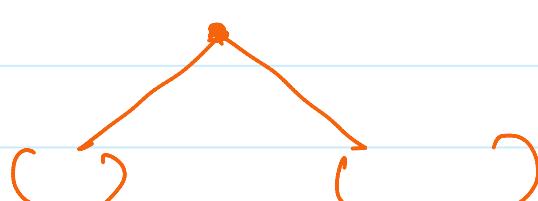
$\Rightarrow (T \wedge F \mid T \wedge F \wedge T \wedge F) \quad ?$

$(T \wedge F) \mid (T \wedge F \wedge T \wedge F) \quad \left. \begin{array}{l} \text{Partitions from} \\ \text{0th index.} \end{array} \right.$

$(T \wedge F \mid T) \wedge (F \wedge T \wedge F)$

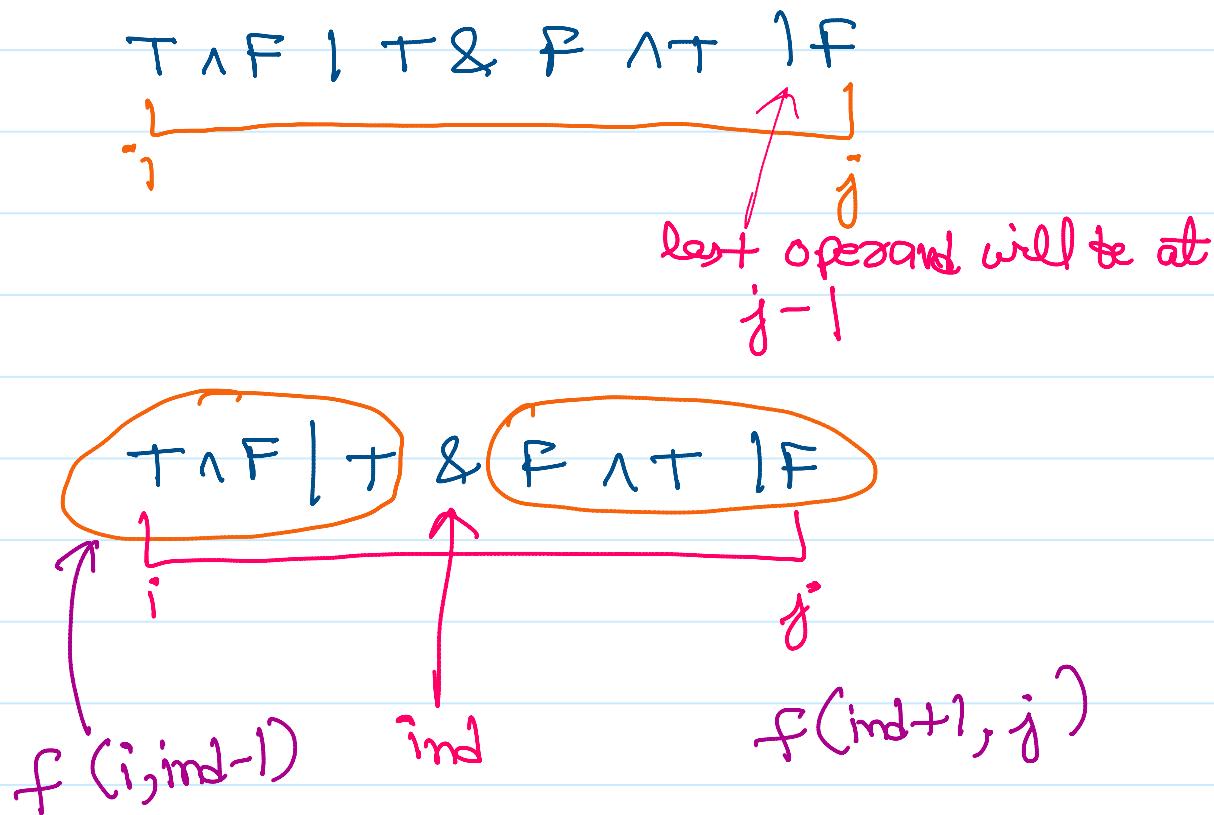
$(T \wedge F \mid T \wedge F) \wedge (T \wedge F)$

$(T \wedge F \mid T \wedge F \wedge T) \wedge (F)$



$\rightarrow$  At every index, we can break down into 2

→ At every index, we can break down into 2 partition in many ways.



→ If these are  $x$  ways when left is true, and  $y$  ways when right is true.

These will be  $x \times y$  total ways, in case of  $\&$  operator

→ In case of OR operator:-

$$\begin{array}{l} \text{left} \\ T = x_1 \\ F = x_4 \end{array} \quad | \quad \begin{array}{l} \text{right} \\ T = x_2 \\ F = x_3 \end{array}$$

Total ways =  $x_1 \times x_2 + x_1 \times x_3 + x_2 \times x_4$   
to make true

→ In case of XOR:-

$$\begin{array}{l} \text{left} \\ T = x_1 \\ F = x_4 \end{array} \quad | \quad \begin{array}{l} \text{right} \\ T = x_2 \\ F = x_3 \end{array}$$

Total ways =  $x_1 \times x_3 + x_2 \times x_4$ ,  
to make true

NOTE:- We have seen that in some cases we need false

NOTE:- we have seen that in some cases we need false to make output true (in xor).

In case of OR also, false is contributing to make output true. ( $T|F$  and  $F|T$ ).

→ so, we will take a variable isTrue to compare, whether our expression gives us true or not.

If we need True to make expression true,.isTrue will be 1, if we need false to make expression true, isTrue will be 0.

→ for partitioning, there can be four possibilities,

$$LT = f(i, \text{ind}-1, 1)$$

$$LF = f(i, \text{ind}-1, 0)$$

$$RT = f(\text{ind}+1, j, 1)$$

$$RF = f(\text{ind}+1, j, 0)$$

Recursion → exponential.

Memoization →  $(N \times N \times 2) \times N \cong (N^3)$

TOTAL WAYS TO MAKE IsTrue False

(i.) & Operator:-

$$\text{ways} = (RT \times LF) + (RF \times LT) + (RF \times LT)$$

(ii.) | Operator:-

$$\text{ways} = (RF \times LF)$$

(iii.) xor (^) operator

$$\text{ways} = (LT \times RT) + (LF \times RF)$$

Q.) But why we need to find  $\text{IsTrue} = \text{false}$ ?

Ans.) Ex →  $T \wedge T \& F$ .

→ we want to make this expression true. so if we

partition  $(T \wedge T) \& (F)$   
LS RS

→ Left side will be false as  $T \wedge T = F$ . and  
Right side is also F. so  
 $F \wedge F = F$ .

→ But, if we partition it like  $(T) \wedge (T \wedge F)$   
LS RS

then LS is true and right side will be  
 $T \wedge F = F$ , so  $T \wedge F = T$ .

→ so we have seen that to make the expression  
true, we need  $I_{\text{true}} = \text{False}$  in the Right  
Side i.e. we need the result of  $\wedge$  expression  
as false.

## Memoization:

```
1 int mod = 1003;
2 int findways(int startind, int endind,int isTrue, string S,vector<vector<vector<int>>& dp)
3 {
4     if(startind>endind) return 0;
5     if(startind==endind)
6     {   if(isTrue)
7         return S[startind]=='T';
8         else return S[startind]=='F';
9     }
10
11    if(dp[startind][endind][isTrue]!=-1)
12        return dp[startind][endind][isTrue];
13    int ways =0;
14    for(int ind = startind+1;ind<=endind-1;ind+=2)
15    {   int lT = findways(startind,ind-1,1,S,dp);
16        int lF = findways(startind,ind-1,0,S,dp);
17        int rT = findways(ind+1,endind,1,S,dp);
18        int rF = findways(ind+1,endind,0,S,dp);
19
20        if(S[ind]=='&')
21        {   if(isTrue)
22            ways = (ways + (lT*rT)%mod)%mod;
23            else
24            ways = (ways + (lT*rF)%mod + (lF*rT)%mod + (lF*rF)%mod)%mod;
25        }
26        else if(S[ind]=='|')
27        {   if(isTrue)
28            ways = (ways + (lT*rF)%mod + (lT*rT)%mod + (lF*rT)%mod)%mod;
29            else
30            ways = ways + ((lF*rF)%mod);
31        }
32        // T^T = F, T^F = T, F*F = F, F*T = T
33        else if(S[ind]=='^')
34        {
35            if(isTrue)
36                ways = (ways + (lT* rF)%mod + (lF * rT))%mod;
37            else
38                ways = (ways + (lT*rT)%mod +(lF*rF)%mod)%mod;
39        }
40
41    }
42    return dp[startind][endind][isTrue]= ways;
43 }
44 int countWays(int N, string S){
45     vector<vector<vector<int>> dp(N, vector<vector<int>> (N, vector<int> (2,-1)));
46     return findways(0,N-1,1,S,dp);
47 }
```

# LECTURE - 53 PALINDROME PARTITIONING-II (FRONT PARTITIONING)

24 June 2022 20:24

[LinkedIn/kapilyadav22](#)

## 132. Palindrome Partitioning II

Hard 3597 86 Add to List Share

Given a string  $s$ , partition  $s$  such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of  $s$ .

### Example 1:

Input:  $s = "aab"$

Output: 1

Explanation: The palindrome partitioning  $["aa", "b"]$  could be produced using 1 cut.

### Example 2:

Input:  $s = "a"$

Output: 0

### Example 3:

Input:  $s = "ab"$

Output: 1

Gx-1 a ab    |    a a b  
  | ab  
  | a (a|b)  
min(2,1) = 1

Gx-2 a b c d e

a | b c d e  
a | b | c d e  
a | b | c | d e  
a | b | c | d | e  
a | b | c | d | e

$\rightarrow$  in  $wc, N-1$  cuts will be these.

$\Rightarrow$  So we will cut at every index one by one and at every index, we will check for palindrome;  
 $\Rightarrow$  if start to end is palindrome. no need

To prevent,  
→ if start to end is palindrome, no need  
to cut, else run a loop till end,  
and cut accordingly.

### Memoization:

```
● ● ●

1 bool isPalindrome(string &s, int start, int end)
2 {
3     while(start<=end)
4     {
5         if(s[start]!=s[end])
6             return 0;
7         start++;
8         end--;
9     }
10    return 1;
11 }
12
13    int minCut(string s) {
14        vector<int> v(s.size()+1,-1);
15        return minimumcuts(s, 0,s.size()-1,v);
16    }
17
18    int minimumcuts(string &s, int start, int end, vector<int>&v)
19    {
20        if(start>end)
21            return 0;
22
23        if(isPalindrome(s,start,end)==1)
24            return 0;
25
26        if(v[start]==-1)
27            return v[start];
28
29        int ans = INT_MAX;
30        for(int currentcut=start;currentcut<end;currentcut++)
31        {
32            if(isPalindrome(s,start,currentcut))
33            {
34                int righthalf = minimumcuts(s,currentcut+1,end,v);
35                int tempans = 1 + righthalf;
36                ans = min(ans, tempans);
37            }
38        }
39        return v[start]=ans;
40    }
```

TC:  $O(N^2) * O(N)$  for checking palindrome

SC:  $O(N) + O(N)$  recursive space.

## Tabulation:

```
● ● ●
```

```
1 bool isPalindrome(string &s, int start, int end)
2 {
3     while(start<=end)
4     {
5         if(s[start]!=s[end])
6             return 0;
7         start++;
8         end--;
9     }
10    return 1;
11 }
12
13 int minCut(string s) {
14     int n =s.size();
15     vector<int> v(n+1,0);
16     v[n]=0;
17     for(int start = n-1; start>=0;start--)
18     { int ans =INT_MAX;
19     for(int currentcut=start;currentcut<n;currentcut++)
20     {
21         if(isPalindrome(s,start,currentcut))
22         {
23             int righthalf = v[currentcut+1];
24             int tempans = 1 + righthalf;
25             ans = min(ans, tempans);
26         }
27     }
28     v[start]=ans;
29     }
30     return v[0]-1;
31 }
```

TC: O(N\*N) \* O(N) for checking palindrome

SC: O(N)

# LECTURE - 54 PARTITION ARRAY FOR MAXIMUM SUM || FRONT PARTITION

24 June 2022 20:25

[LinkedIn/kapilyadav22](#)

## 1043. Partition Array for Maximum Sum

Medium 2341 203 Add to List Share

Given an integer array `arr`, partition the array into (contiguous) subarrays of length **at most** `k`. After partitioning, each subarray has their values changed to become the maximum value of that subarray.

Return the largest sum of the given array after partitioning. Test cases are generated so that the answer fits in a **32-bit** integer.

### Example 1:

```
Input: arr = [1,15,7,9,2,5,10], k = 3
Output: 84
Explanation: arr becomes [15,15,15,9,10,10,10]
```

### Example 2:

```
Input: arr = [1,4,1,5,7,3,6,1,9,9,3], k = 4
Output: 83
```

### Example 3:

```
Input: arr = [1], k = 1
Output: 1
```

Ex - {1, 15, 7, 9, 2, 5, 10} k=3

{1, 15 | 7 9 2 | 5 10}

{15, 15, 9, 9 | 5 10}

Sum = 77

OR

{1, 15, 7, 9, 2, 5, 10}

{1, 15, 7, 9, | 2, 5, 10}

{15, 15, 15, 9, 10, 10, 10}

Sum = 84

Rules to write Recurrence ?-

1. Express everything in terms of index.
2. Try every partition possible from that index.
3. Take the best partition.

**Memoization:**

```
 1 int findmaxsum(int ind,vector<int>& arr, int k, vector<int>& dp)
 2     { int n = arr.size();
 3         if(ind==n) return 0;
 4         if(dp[ind]!=-1)
 5             return dp[ind];
 6         int len = 0;
 7         int maxi = INT_MIN;
 8         int maxans = INT_MIN;
 9
10        for(int j = ind;j<min(ind+k,n);j++)
11        {
12            len++;
13            maxi = max(maxi,arr[j]);
14            int sum = (len* maxi) + findmaxsum(j+1,arr,k,dp);
15            maxans = max(maxans,sum);
16        }
17        return dp[ind] = maxans;
18    }
19
20    int maxSumAfterPartitioning(vector<int>& arr, int k) {
21        int n = arr.size();
22        vector<int> dp(n,-1);
23        return findmaxsum(0,arr,k,dp);
24    }
```

TC:  $O(N) * O(K)$

SC:  $O(N) + O(N)$

## Tabulation:

```
● ● ●

1 int maxSumAfterPartitioning(vector<int>& arr, int k) {
2     int n = arr.size();
3     vector<int> dp(n+1,0);
4     for(int ind = n-1; ind >= 0; ind--) {
5         int len = 0;
6         int maxi = INT_MIN;
7         int maxans = INT_MIN;
8
9         for(int j = ind; j < min(ind+k, n); j++) {
10            len++;
11            maxi = max(maxi, arr[j]);
12            int sum = (len * maxi) + dp[j+1];
13            maxans = max(maxans, sum);
14        }
15        dp[ind] = maxans;
16    }
17    return dp[0];
18}
```

TC: O(N) \* O(K)

SC: O(N)

# LECTURE - 55 MAXIMUM RECTANGLE AREAS WITH ALL 1'S || DP ON RECTANGLES

24 June 2022 20:25

[LinkedIn/kapilyadav22](#)

## 85. Maximal Rectangle

Hard 6915 110 Add to List Share

Given a `rows x cols` binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

### Example 1:

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

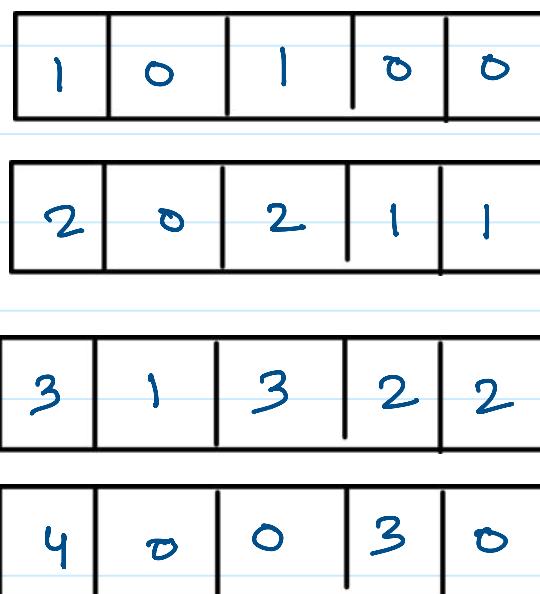
Input: matrix = `["1","0","1","0","0"], ["1","0","1","1","1"], ["1","1","1","1","1"], ["1","0","0","1","0"]`

Output: 6

Explanation: The maximal rectangle is shown in the above picture.

### Prerequisite: Largest rectangle in Histogram

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0



```

1 int largestRectangleArea(vector < int > & histo) {
2     stack < int > st;
3     int maxA = 0;
4     int n = histo.size();
5     for (int i = 0; i <= n; i++) {
6         while (!st.empty() && (i == n || histo[st.top()] >= histo[i])) {
7             int height = histo[st.top()];
8             st.pop();
9             int width;
10            if (st.empty())
11                width = i;
12            else
13                width = i - st.top() - 1;
14            maxA = max(maxA, width * height);
15        }
16        st.push(i);
17    }
18    return maxA;
19 }

20

21 int maximalRectangle(vector<vector<char>>& matrix) {
22     int maxarea = 0;
23     int n = matrix.size();
24     int m = matrix[0].size();
25     //TC : O(n*(m+n))
26     vector<int> height(m, 0);
27     for(int i = 0; i < n; i++)
28     {for(int j = 0; j < m; j++)
29      {if(matrix[i][j] == '1')
30       height[j]++;
31      else height[j] = 0;
32      }
33     //TC: O(n)
34     int area = largestRectangleArea(height);
35     maxarea = max(area, maxarea);
36     }
37     return maxarea;
38 }
```

TC:O( $n \cdot (m + n)$ )

SC: O(n)

# LECTURE - 56 COUNT SQUARE SUBMATRICES WITH ALL ONES || DP ON RECTANGLE

24 June 2022 20:26

[LinkedIn/kapilyadav22](#)

## 1277. Count Square Submatrices with All Ones

Medium 3463 61 Add to List Share

Given a  $m * n$  matrix of ones and zeros, return how many **square** submatrices have all ones.

### Example 1:

```
Input: matrix =  
[  
    [0,1,1,1],  
    [1,1,1,1],  
    [0,1,1,1]  
]  
Output: 15  
Explanation:  
There are 10 squares of side 1.  
There are 4 squares of side 2.  
There is 1 square of side 3.  
Total number of squares = 10 + 4 + 1 = 15.
```

Ex-1

1	0	1
1	1	0
1	1	0

- These are 6, size 1 squares.
- There are 1, size 2 squares.

Ex-2

0	1	1	1
1	1	1	1
0	1	1	1

- There are 10, size 1 squares.
- There are 4, Size 2 Squares.
- There are 1, Size 3 Squares.

Ex :-

0	1	2
0	1	1
1	1	2
2	1	2

→ *same as matrix*

- In Squares problem, tabulation is more intuitive. So take a dp of same size as square.
- Since the first row & first column will not make Right Bottom square, so these will same squares as in the matrix,

- If we sum up the values in dp then we will get the total count.  
But how to fill the dp?

1	1	1	1
1	1	1	1
1	1	1	1

1	1	1	1
1	2	2	2
1	2	3	3

→ To count the squares, check for  $(i-1, j)$ ,  $(i-1, j-1)$  and  $(i, j-1)$  and take the minimum of them and add 1.

```
● ● ●

1 int countSquares(vector<vector<int>>& matrix) {
2     int n = matrix.size();
3     int m = matrix[0].size();
4     vector<vector<int>> dp(n, vector<int>(m, 0));
5     for(int j=0; j<m; j++)
6         dp[0][j] = matrix[0][j];
7     for(int i=0; i<n; i++)
8         dp[i][0] = matrix[i][0];
9     for(int i = 1; i < n; i++)
10    {
11        for(int j=1; j < m; j++)
12        {
13            if(matrix[i][j] == 0)
14                dp[i][j] = 0;
15            else
16            {
17                dp[i][j] = 1 + min(dp[i-1][j], min(dp[i-1][j-1], dp[i][j-1]));
18            }
19        }
20    }
21    int sum = 0;
22    for(int i=0; i < n; i++)
23    {
24        for(int j=0; j < m; j++)
25        {
26            sum += dp[i][j];
27        }
28    }
29    return sum;
}
```

TC :  $O(N \cdot M) + O(N \cdot M)$   
SC :  $O(N \cdot M)$

Congrats,  
You Have  
studied DP



 Kapilyadav22