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 = "abcaa"

add aacba, str2 = "abcaaaacba" is palindromic. No. of operations = length of a string.

a b c a a

↓

a a b c b a a

Q.) How do you approach it?

Keep the longest palindromic portion intact.

→ Take aaa.

a b c a c b a

→ Take aca.

a b a c a b a

→ Take aba

a a c b c a a

→ The number of insertions =

N − Longest palindromic subsequence

Tabulation:

```cpp
int longestCommonSubsequence(string s1, string s2) {
        int m = s1.length();
        int n = s2.length();

        vector<int> next(n+1,0);

        for(int ind1 = m-1;ind1>=0;ind1--)
        { vector<int> curr(n+1,0);
            for(int ind2 = n-1;ind2>=0;ind2--)
            { if(s1[ind1]==s2[ind2])
             curr[ind2]=1 + next[ind2+1];
            else
            curr[ind2]=max(next[ind2],curr[ind2+1]);
            }
         next = curr;
        }
        return next[0];
    }

    int minInsertions(string s) {
       int N = s.size();
         string s2 =s;
        reverse(s2.begin(),s2.end());
        return N-longestCommonSubsequence(s,s2);
        }
```

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

Space Optimization: Using one 1D array

```cpp
int minInsertions(string s1) {
        string s2 =s1;
         reverse(s2.begin(),s2.end());
         int m = s1.length();
         int n = s2.length();

         vector<int> next(n+1,0);

         for(int ind1 = m-1;ind1>=0;ind1--)
         { vector<int> curr(n+1,0);
             for(int ind2 = n-1;ind2>=0;ind2--)
             { if(s1[ind1]==s2[ind2])
              curr[ind2]=1 + next[ind2+1];
             else
             curr[ind2]=max(next[ind2],curr[ind2+1]);
             }
          next = curr;
        }
        return m-next[0];
        }
```


TC:O(M*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

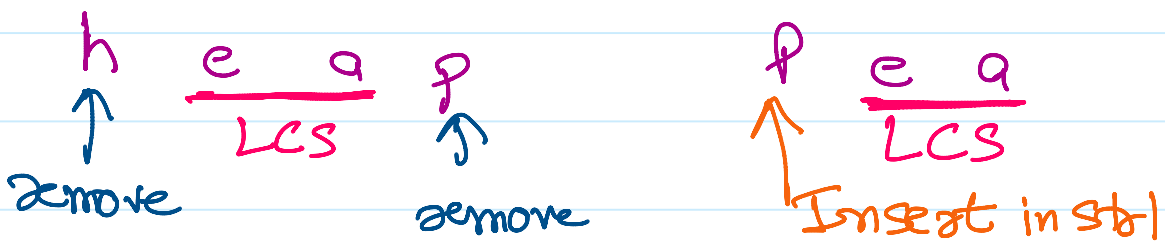
remove h, p from str1.

Insert p in the begining 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 Commonsubsequence, 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:

```cpp
int longestCommonSubsequence(string s1, string s2) {
        int m = s1.length();
        int n = s2.length();

        vector<int> next(n+1,0);

        for(int ind1 = m-1;ind1>=0;ind1--)
        { vector<int> curr(n+1,0);
            for(int ind2 = n-1;ind2>=0;ind2--)
            { if(s1[ind1]==s2[ind2])
             curr[ind2]=1 + next[ind2+1];
            else
            curr[ind2]=max(next[ind2],curr[ind2+1]);
            }
         next = curr;
        }
        return next[0];
    }
    public:
    int minOperations(string A, string B)
    {
        int m =A.size();
        int n = B.size();
        int deletions = m-longestCommonSubsequence(A,B);
        int insertions = n-longestCommonSubsequence(A,B);
        return insertions+deletions;
    }
```

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

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 – str1 = abac , str2 = cab

output → c a b a c

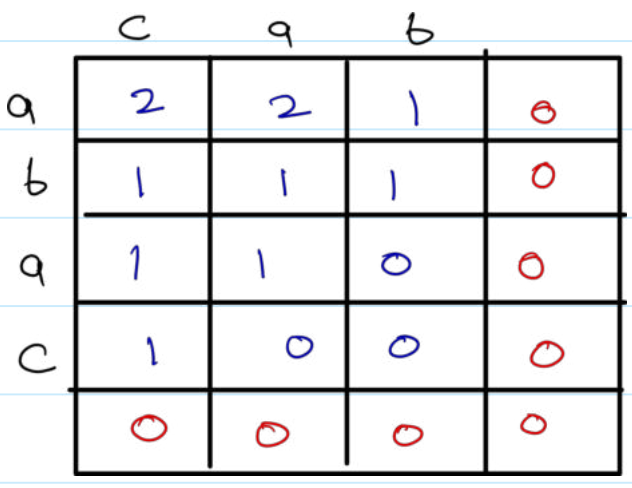
We need to add all the characters from str1 and str2, but we have to remove the LCS one time, as its will occur twice

So, Length of Shortest Common Supersequence : str1 length +str2 length -LCS.

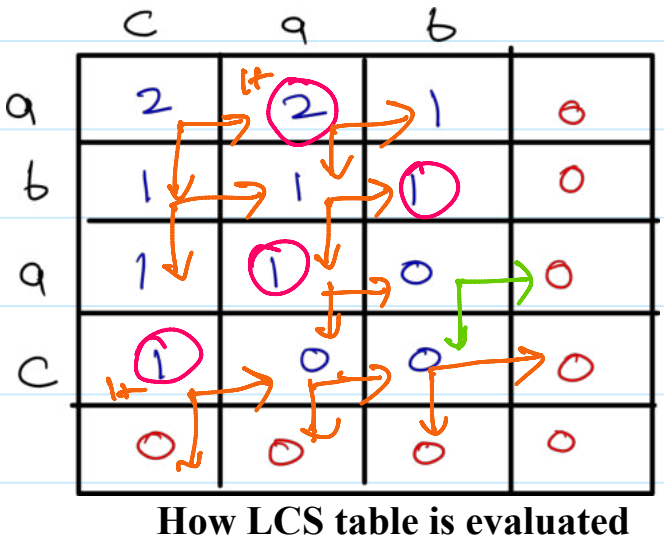
To find Shortest Common Supersequence:

- 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"



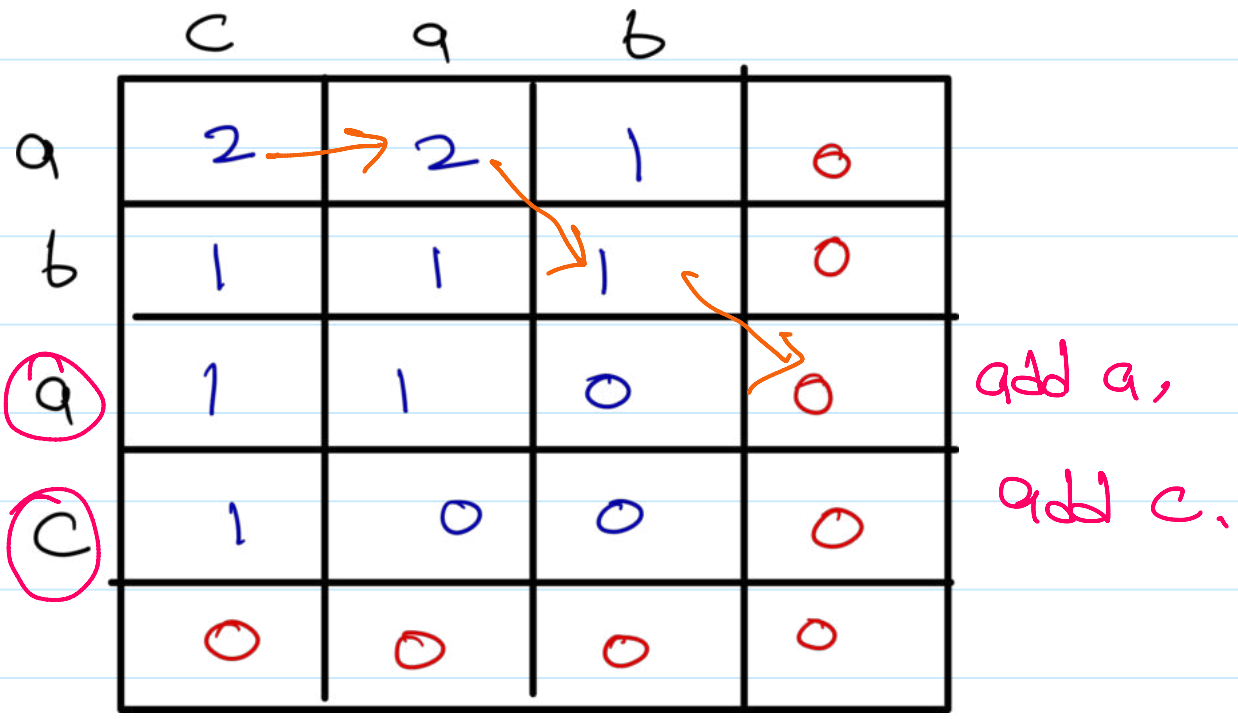
LCS table.



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 Increament j.
- Now we are at dp[0][1], a== a. So add **a** in ans and increament both i and j.
- Now we are dp[1][2]. b==b, so again add **b** in ans and increament 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".**



Tabulation:


```cpp
string shortestCommonSupersequence(string s1, string s2) {
        int m = s1.length();
        int n = s2.length();
        vector<vector<int>> dp(m+1,vector<int>(n+1,0));

        for(int ind1 = m-1;ind1>=0;ind1--)
        { for(int ind2 = n-1;ind2>=0;ind2--)
            { if(s1[ind1]==s2[ind2])
             dp[ind1][ind2]=1 + dp[ind1+1][ind2+1];
            else
            dp[ind1][ind2]=max(dp[ind1+1][ind2],dp[ind1][ind2+1]);
            }
        }
        string ans="";

    int i =0;int j=0;

      while(i<m && j<n)
        {
            if(s1[i]==s2[j])
            {  ans+=s1[i];
             i++; j++;
            }
            else if(dp[i+1][j]>dp[i][j+1])
            {   ans+=s1[i];
                i++;
            }
          else {
              ans+=s2[j];
                j++;
          }
      }
       while(i<m)
       { ans+=s1[i];
           i++;    }

       while(j<n)
       { ans+=s2[j];
           j++;  }

       return ans;
            }
```

LECTURE - 32 DISTINCT SUBSEQUENCES (DP ON STRINGS)

Tuesday, 14 June 2022 2:45 PM

$s = "babgbag"$ $s_2 = "bag"$

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

n → s_1 = b a b g b a g (i) (n-1)
m → s_2 = b a g (m-1)

```
if (s1[i] == s2[j])
    return (f(i-1, j-1) + f(i-1, j));
else
    return f(i-1, j);
```

Base case :—

i
b a b g b a g
↑ 0
-1

b a g
↑

→ if str1 is exhausted, but str2 is not, so return 0.
→ if str2 is exhausted, but str1 is not, return 1.

Recursion; TC → exponential
SC → O(N+M)

→ Use Memoization

```cpp
int countdistinct(string s,string t,int ssize, int tsize,
vector<vector<int>> &dp)
    {
         if(tsize<0) return 1;

        if(ssize<0)  return 0;

        if(dp[ssize][tsize]!=-1)
            return dp[ssize][tsize];
        if(s[ssize]==t[tsize])
            return countdistinct(s,t,ssize-1,tsize-1,dp) +
          countdistinct(s,t,ssize-1,tsize,dp);
      return countdistinct(s,t,ssize-1,tsize,dp);
    }

    int numDistinct(string s, string t) {
     int n = s.size();
     int m = t.size();
    vector<vector<int>> dp(n,vector<int> (m,-1));
        return countdistinct(s,t,n-1,m-1,dp);
    }
```

→ It will give TLE. TC = O(N×M), SC → O(N×M) + O(N+M)

→ Tabulation :-

```cpp
int numDistinct(string s, string t) {
     int n = s.size();
     int m = t.size();
     vector<vector<int>> dp(n+1,vector<int> (m+1,0));
        for(int i=0;i<=n;i++)
            dp[i][0] =1;
        for(int j=1;j<=m;j++)
            dp[0][j]=0;
        for(int i=1;i<=n;i++)
        { for(int j=1;j<=m;j++)
            {  if(s[i-1]==t[j-1])
                dp[i][j]= dp[i-1][j-1] + dp[i-1][j];
               else dp[i][j] = dp[i-1][j];
            }
        }
       return dp[n][m];
    }
```

→ It will give Runtime error.

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

- **Try to change the datatype of vector dp**

```cpp
int numDistinct(string s, string t) {
      int n = s.size();
      int m = t.size();
         //try to change the data type of vector dp
      vector<vector<double>> dp(n+1,vector<double> (m+1,0));
         for(int i=0;i<=n;i++)
             dp[i][0] =1;
         for(int j=1;j<=m;j++)
             dp[0][j]=0;
         for(int i=1;i<=n;i++)
         { for(int j=1;j<=m;j++)
             {  if(s[i-1]==t[j-1])
                 dp[i][j]= dp[i-1][j-1] + dp[i-1][j];
                else dp[i][j] = dp[i-1][j];
             }
         }
        return (int)dp[n][m];
    }
```

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

- **Optimize it using two 1D arrays:**

```cpp
int numDistinct(string s, string t) {
      int n = s.size();
      int m = t.size();
         vector<double> prev(m+1,0),curr(m+1,0);
         prev[0]= curr[0]=1;

         for(int i=1;i<=n;i++)
         { for(int j=1;j<=m;j++)
             {  if(s[i-1]==t[j-1])
                 curr[j]= prev[j-1] + prev[j];
                else
                 curr[j] = prev[j];
             }
         prev = curr;
         }
        return (int)prev[m];
    }
```

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

- **Use only one 1D dp**

```cpp
int numDistinct(string s, string t) {
    int n = s.size();
    int m = t.size();
       vector<double> prev(m+1,0);
       prev[0]= 1;

       for(int i=1;i<=n;i++)
       { for(int j=m;j>=1;j--)
           {  if(s[i-1]==t[j-1])
                prev[j]= prev[j-1] + prev[j];
           }
       }
      return (int)prev[m];
   }
```

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

LinkedIn/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?

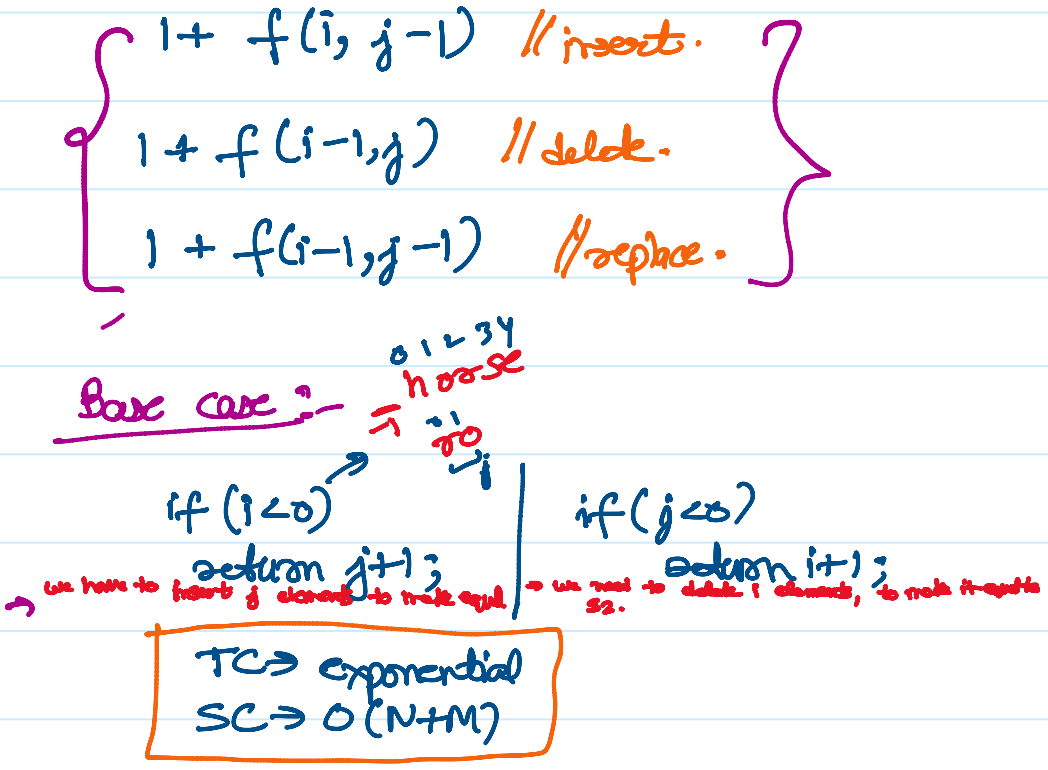
Ans) 1) Express in terms of (i, j).
2) Explore all paths of matching.
3) Return the min of all paths.
4) Base case.

f(n-1, m-1) → min operations to convert s1[0 .... i] to s2[0 .... j].

→ if (s[i] == s2[j]
return f(i-1, j-1)

→ **Insertion**

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



Recursion:

```cpp
int findMinOp(int i, int j, string word1, string word2)
    {
        if(i<0)
            return j+1;
        if(j<0)
            return i+1;
        if(word1[i]==word2[j])
            return findMinOp(i-1,j-1,word1,word2);
        return 1+min(findMinOp(i-1,j,word1,word2),
                     min(findMinOp(i,j-1,word1,word2),
                     findMinOp(i-1,j-1,word1,word2)));

    }

    int minDistance(string word1, string word2) {
        int n = word1.size();
        int m = word2.size();
        return findMinOp(n-1,m-1,word1,word2);

    }
```

MEMOIZATION:

```cpp
int findMinOp(int i, int j, string word1, string word2,vector<vector<int>>& dp)
    {
        if(i<0)
            return j+1;
        if(j<0)
            return i+1;
        if(dp[i][j]!=-1)
            return dp[i][j];

        if(word1[i]==word2[j])
            return findMinOp(i-1,j-1,word1,word2,dp);
        int deletechar= findMinOp(i-1,j,word1,word2,dp);
        int replacechar = findMinOp(i-1,j-1,word1,word2,dp);
        int insertchar = findMinOp(i,j-1,word1,word2,dp);
        return dp[i][j] = 1+min(insertchar, min(replacechar,deletechar));
    }

    int minDistance(string word1, string word2) {
        int n = word1.size();
        int m = word2.size();
        vector<vector<int>> dp(n, vector<int>(m,-1));
        return findMinOp(n-1,m-1,word1,word2,dp);
    }
```

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

Tabulation:

```cpp
int minDistance(string word1, string word2) {
        int n = word1.size();
        int m = word2.size();
        vector<vector<int>> dp(n+1, vector<int>(m+1,0));
        for(int i=0;i<=n;i++)
            dp[i][0]=i;
        for(int j=0;j<=m;j++)
            dp[0][j]=j;

        for(int i=1;i<=n;i++)
        { for(int j=1;j<=m;j++)
            { if(word1[i-1]==word2[j-1])
                dp[i][j]=dp[i-1][j-1];
               else dp[i][j]= 1 + min(dp[i-1][j],min(dp[i][j-1],dp[i-1][j-1]));
            }
        } return dp[n][m];
    }
```

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

- Using 1D arrays,

```cpp
int minDistance(string word1, string word2) {
        //using 1d arrays
        int n = word1.size();
        int m = word2.size();
        vector<int> prev(m+1,0), cur(m+1,0);

        for(int j=0;j<=m;j++)
            prev[j]=j;

        for(int i=1;i<=n;i++)
        { cur[0]=i;
            for(int j=1;j<=m;j++)
            { if(word1[i-1]==word2[j-1])
                cur[j]=prev[j-1];
               else cur[j]= 1 + min(prev[j],min(cur[j-1],prev[j-1]));
            }
        prev=cur;
        }
        return prev[m];
    }
```

TC: O(N*M) **SC :** O(M) LinkedIn/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 = "?ay" } True
S_2 = "xay"

S_1 = "ab*cd" } True
S_2 = "abdefcd"

where * matches with def

S_1 = "**abcd"
S_2 = "abcd"

where * matches with length 0.

S_1 = "ab?*" } False
S_2 = "abcx"

→ 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 increament the index of both.

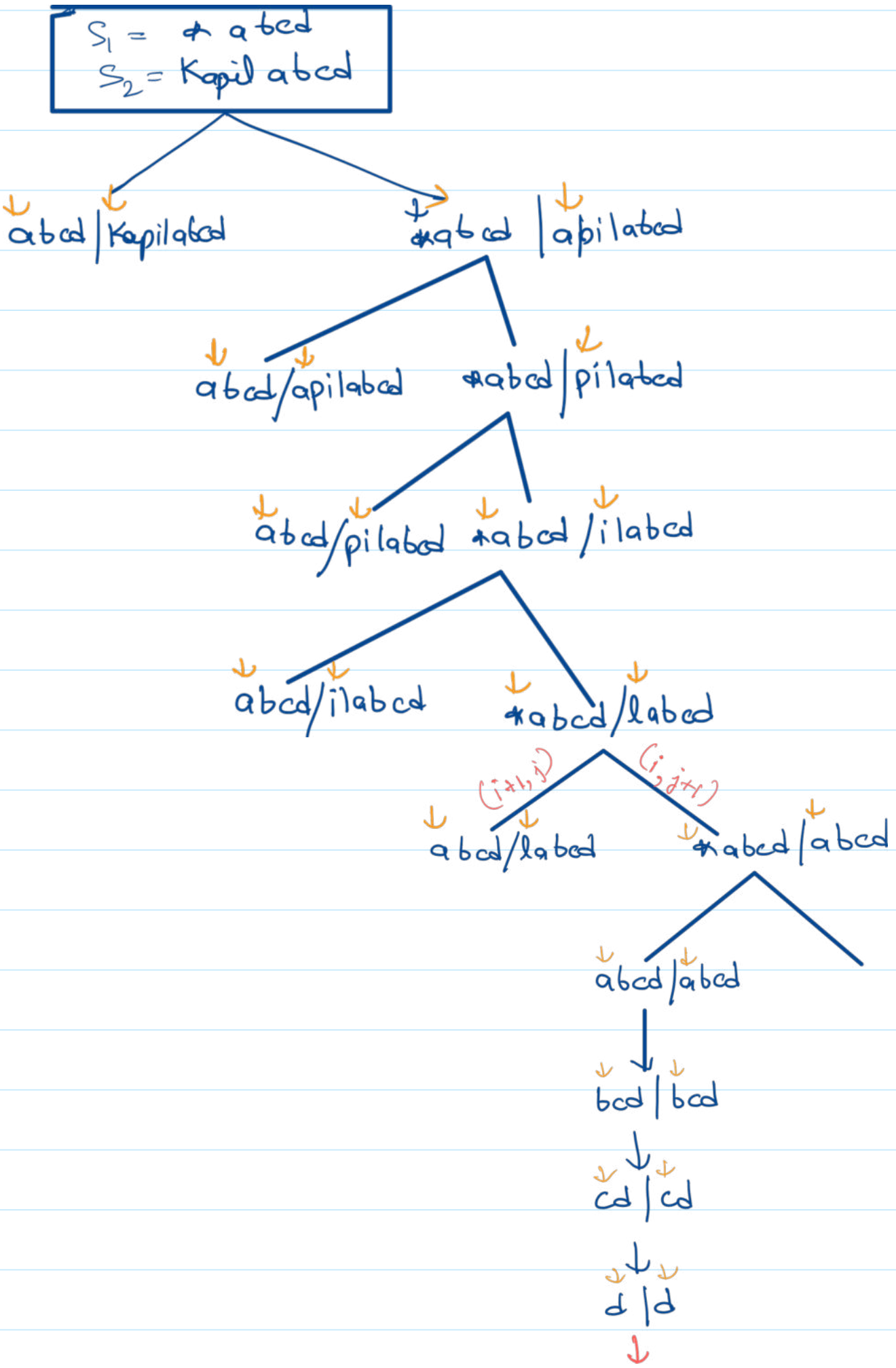
→ If there is a '*' of length 0, we can increament the index of S_2.

ex S_1 → a * c

S_2 → a c

→ for * of length 1, 2, 3, ...

Ex →



→ If $s_1 == end$ & $s_2 == end$ return True.

→ If $s_1 == end$ & $s_2 != end$ return False

→ If $s_1 != end$ & $s_2 == end$

```
for(i=0 to s1.length())
    if (s1[i] != '*')
        return false.
```

return false.

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

→ Memoization :- TC : O(N×M)
SC : O(N×M) + O(N+M)

Memoization:

```cpp
bool findwildcard(int SInd, int PInd,string& s, string& p,
vector<vector<int>>& dp)
{   int SEndInd = s.size()-1;
    int PEndInd = p.size()-1;

    if(SInd>SEndInd && PInd>PEndInd)
     return true;
    if(SInd<=SEndInd && PInd>PEndInd)
     return false;
    if(SInd>SEndInd && PInd<=PEndInd)
    {  for(int i=PInd;i<=PEndInd;i++)
        { if(p[i]!='*')
                return false;
        } return true;
    }
  if(dp[SInd][PInd]!=-1)
      return dp[SInd][PInd];

  if(s[SInd]==p[PInd] || p[PInd]=='?')
      return dp[SInd][PInd] = findwildcard(SInd+1,PInd+1,s,p,dp);
  if(p[PInd]=='*')
  {
      return dp[SInd][PInd]=findwildcard(SInd+1,PInd,s,p,dp) ||
                            findwildcard(SInd,PInd+1,s,p,dp);
  }
  return dp[SInd][PInd]=false;
}


    bool isMatch(string s, string p) {
        int n = s.size();
        int m = p.size();
        vector<vector<int>> dp(n, vector<int> (m,-1));
        return findwildcard(0,0,s,p,dp);
    }
```

Tabulation:

```cpp
bool isMatch(string s, string p) {
        int n = s.size();
        int m = p.size();
        vector<vector<int>> dp(n+1, vector<int> (m+1,0));
        dp[n][m]=1;

       for(int i=m-1;i>=0;i--)
        { if(p[i]=='*')
                dp[n][i]=1;
            else
                break;
        }

        for(int Sind=n-1;Sind>=0;Sind--)
        {
            for(int PInd=m-1;PInd>=0;PInd--)
            {if(s[Sind]==p[PInd] || p[PInd]=='?')
                  dp[Sind][PInd] = dp[Sind+1][PInd+1];
              else if(p[PInd]=='*')
              {
                  dp[Sind][PInd]=dp[Sind+1][PInd] || dp[Sind][PInd+1];
              }
              else dp[Sind][PInd]=false;
            }
        }
        return dp[0][0];
    }
```

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^{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:

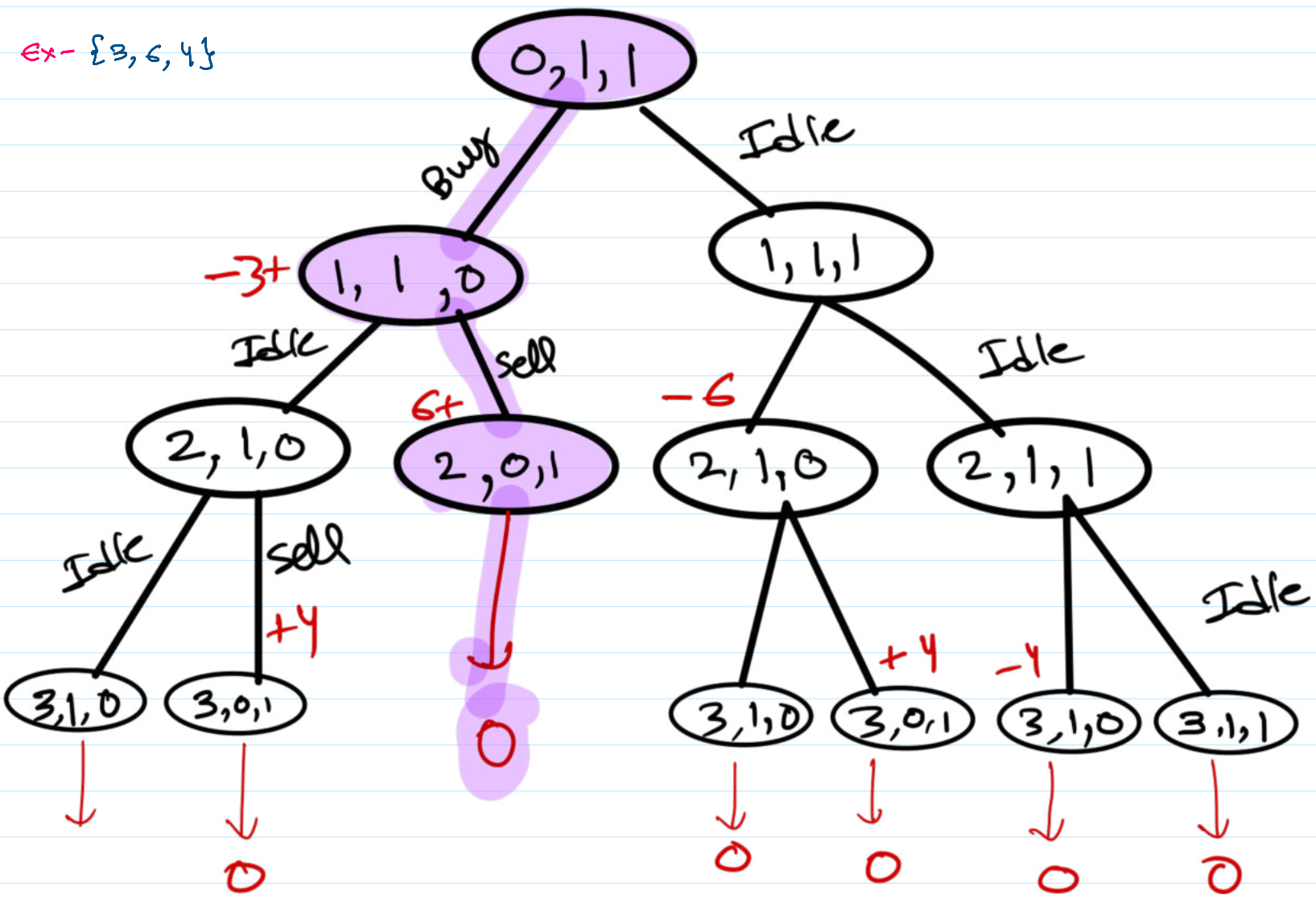
```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

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:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<vector<int>>> v(m,vector<vector<int>> (2,vector<int> (2,-1)));
        return buyandsell(0,1,true,prices,v);
    }

int buyandsell(int currentindex,int transactioncount,bool canbuy, vector<int>& prices,vector<vector<vector<int>>>& v)
{
    if(currentindex>=prices.size() || transactioncount<=0)
        return 0;


    int key = v[currentindex][transactioncount][canbuy];
    if(key!=-1)
        return key;

   int idle = buyandsell(currentindex+1,transactioncount,canbuy,prices,v);

    if(canbuy)
    {  int buyside = -prices[currentindex] + buyandsell(currentindex+1,transactioncount,0,prices,v);
     return v[currentindex][transactioncount][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] + buyandsell(currentindex+1,transactioncount-1,1,prices,v);
     return v[currentindex][transactioncount][canbuy] = max(sellside,idle);
    }
    }
```

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

Tabulation:

```cpp
nt maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<vector<int>>> v(m+1,vector<vector<int>> (2,vector<int> (2,0)));
        for(int currentindex=m-1;currentindex>=0;currentindex--)
        {
            for(int transactioncount=1;transactioncount>=1;transactioncount--)
            {
                for(int canbuy=0;canbuy<=1;canbuy++)
                {
                    int idle = v[currentindex+1][transactioncount][canbuy];

                    if(canbuy)
                    {  int buyside = -prices[currentindex] + v[currentindex+1][transactioncount][0];
                        v[currentindex][transactioncount][canbuy] = max(buyside,idle);
                    }
                    else
                    {    int sellside = prices[currentindex] + v[currentindex+1][transactioncount-1][1];
                        v[currentindex][transactioncount][canbuy] = max(sellside,idle);
                    }
                }
            }
        }  return v[0][1][true];
    }

```

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

Optimization:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<vector<int>>> v(m+1,vector<vector<int>> (2,vector<int> (2,0)));
        for(int currentindex=m-1;currentindex>=0;currentindex--)
        {    for(int canbuy=0;canbuy<=1;canbuy++)
                {
                    int idle = v[currentindex+1][1][canbuy];

                    if(canbuy)
                    {  int buyside = -prices[currentindex] + v[currentindex+1][1][0];
                        v[currentindex][1][canbuy] = max(buyside,idle);
                    }
                    else
                    {    int sellside = prices[currentindex] + v[currentindex+1][0][1];
                        v[currentindex][1][canbuy] = max(sellside,idle);
                    }

            }
        }  return v[0][1][true];
    }
```

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

Further Space Optimization:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<int>> next(2,vector<int> (2,0));
        for(int currentindex=m-1;currentindex>=0;currentindex--)
        {       vector<vector<int>> curr(2,vector<int> (2,0));
            for(int canbuy=0;canbuy<=1;canbuy++)
                { int idle = next[1][canbuy];

                    if(canbuy)
                    {  int buyside = -prices[currentindex] + next[1][0];
                        curr[1][canbuy] = max(buyside,idle);
                    }
                    else
                    {    int sellside = prices[currentindex] + next[0][1];
                        curr[1][canbuy] = max(sellside,idle);
                    }
            }
         next = curr;
        }  return next[1][true];
    }
```


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

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^{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 limitaion on buying and selling, so it will be same as Previous question, We just need to eliminate the transactioncount part.

ex → {7 1 5 3 6 4}

Buy on day-2, sell on day-3. 5-1 = 4
Buy on day-4, sell on day-5 6-3 = 3
7

MEMOIZATION:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<int>> v(m,vector<int> (2,-1));
        return buyandsell(0,true,prices,v);
    }

int buyandsell(int currentindex,bool canbuy, vector<int>& prices,vector<vector<int>>& v)
{
    if(currentindex>=prices.size() )
        return 0;

    int key = v[currentindex][canbuy];
    if(key!=-1)
        return key;

   int idle = buyandsell(currentindex+1,canbuy,prices,v);

    if(canbuy)
    {  int buyside = -prices[currentindex] + buyandsell(currentindex+1,0,prices,v);
     return v[currentindex][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] + buyandsell(currentindex+1,1,prices,v);
     return v[currentindex][canbuy] = max(sellside,idle);
    }
    }
```

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

Tabulation:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
       vector<vector<int>> dp(m+1,vector<int> (2,0));
      dp[m][0]=0;
      dp[m][1]=0;

   for(int currentindex=m-1;currentindex>=0;currentindex--)
    { for(int canbuy = 0;canbuy<=1;canbuy++)
      {  int idle = dp[currentindex+1][canbuy];

      if(canbuy)
      {  int buyside = -prices[currentindex] +dp[currentindex+1][0];
       dp[currentindex][canbuy] = max(buyside,idle);
      }
      else
      {    int sellside = prices[currentindex] +dp[currentindex+1][1];
         dp[currentindex][canbuy] = max(sellside,idle);
      }
        }
    }
        return dp[0][1];
    }
```

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

Space Optimization:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
       vector<int> next(2,0);

      for(int currentindex=m-1;currentindex>=0;currentindex--)
      { vector<int> curr(2,0);
          for(int canbuy = 0;canbuy<=1;canbuy++)
      {  int idle = next[canbuy];

    if(canbuy)
    {  int buyside = -prices[currentindex] +next[0];
     curr[canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] + next[1];
       curr[canbuy] = max(sellside,idle);
    }
}
        next =curr;
    }
        return next[1];
    }
```

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

Same as Previous, No optimization, we are just using 4 variables instead of taking vector.

LinkedIn/kapilyadav22

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
       int nextbuy=0,nextnotbuy=0;
      int currbuy=0,currnotbuy=0;
      //it is same as vector<int> next(2,0)
      // we have just taken 4 variables instead of two 1d vectors
     for(int currentindex=m-1;currentindex>=0;currentindex--)
     {
         //we are computing for for buy=0 and buy =1

         //buy=1
      int idle = nextbuy;//we are not buying it, taking the next bought profit
      int buyside = -prices[currentindex] + nextnotbuy;
      currbuy = max(buyside,idle);

         //buy=0
      idle =nextnotbuy;  //we are not selling it, taking the next not bought profit
      int sellside = prices[currentindex] + nextbuy;
     currnotbuy = max(sellside,idle);
     nextnotbuy = currnotbuy;
      nextbuy =currbuy;
}
      return nextbuy;
    }
```

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 i^{th} 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:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<vector<int>>> v(m,vector<vector<int>> (3,vector<int> (2,-1)));
        return buyandsell(0,2,true,prices,v);
    }

int buyandsell(int currentindex,int transactioncount,bool canbuy, vector<int>& prices,vector<vector<vector<int>>>& v)
{
    if(currentindex>=prices.size() || transactioncount<=0)
        return 0;

    int key = v[currentindex][transactioncount][canbuy];
    if(key!=-1)
        return key;

   int idle = buyandsell(currentindex+1,transactioncount,canbuy,prices,v);

    if(canbuy)
    {  int buyside = -prices[currentindex] + buyandsell(currentindex+1,transactioncount,0,prices,v);
     return v[currentindex][transactioncount][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] + buyandsell(currentindex+1,transactioncount-1,1,prices,v);
     return v[currentindex][transactioncount][canbuy] = max(sellside,idle);
    }
    }
```

TC: O(M*3*2)

SC : O(M*3*2) + O(M)

Tabulation:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<vector<int>>> v(m+1,vector<vector<int>> (3,vector<int> (2,0)));
        for(int currentindex=m-1;currentindex>=0;currentindex--)
        {
            for(int transactioncount=2;transactioncount>=1;transactioncount--)
            {
                for(int canbuy=0;canbuy<=1;canbuy++)
                {
                    int idle = v[currentindex+1][transactioncount][canbuy];

                    if(canbuy)
                    {  int buyside = -prices[currentindex] + v[currentindex+1][transactioncount][0];
                        v[currentindex][transactioncount][canbuy] = max(buyside,idle);
                    }
                    else
                    {    int sellside = prices[currentindex] + v[currentindex+1][transactioncount-1][1];
                        v[currentindex][transactioncount][canbuy] = max(sellside,idle);
                    }
                }
            }
        }  return v[0][2][true];
    }
```

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

Space Optimization:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
       vector<vector<int>> next(3,vector<int> (2,0));
        for(int currentindex=m-1;currentindex>=0;currentindex--)
        { vector<vector<int>> curr(3,vector<int> (2,0));
            for(int transactioncount=2;transactioncount>=1;transactioncount--)
            {
                for(int canbuy=0;canbuy<=1;canbuy++)
                {
                    int idle = next[transactioncount][canbuy];

                    if(canbuy)
                    {  int buyside = -prices[currentindex] + next[transactioncount][0];
                        curr[transactioncount][canbuy] = max(buyside,idle);
                    }
                    else
                    {    int sellside = prices[currentindex] + next[transactioncount-1][1];
                        curr[transactioncount][canbuy] = max(sellside,idle);
                    }
                }
            }
     next = curr;
        }
    return next[2][true];
    }
```

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.

```cpp
int maxProfit(vector<int>& prices) {
        vector<vector<int>> v(prices.size(),vector<int> (4,-1));
    return buyandsell(0,0,prices,v);
    }

    int buyandsell(int currentindex, int transactioncount, vector<int>& prices,
                   vector<vector<int>>& v)
    {
        if(currentindex>=prices.size() || transactioncount>=4)
            return 0;


        if(v[currentindex][transactioncount]!=-1)
            return v[currentindex][transactioncount];

        int idle = buyandsell(currentindex+1,transactioncount,prices,v);
        int buy,sell = 0;

        if(transactioncount%2==0)
        {   buy = -prices[currentindex] + buyandsell(currentindex+1,transactioncount+1,prices,v);
          return   v[currentindex][transactioncount] = max(buy,idle);
        }
         else
         {   sell = prices[currentindex] + buyandsell(currentindex+1,transactioncount+1, prices,v);
          return   v[currentindex][transactioncount] = max(sell,idle);
         }
         }
```


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^{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:

```
int maxProfit(int k,vector<int>& prices) {
        int m = prices.size();
        vector<vector<vector<int>>> v(m,vector<vector<int>> (k+1,vector<int> (2,-1)));
        return buyandsell(0,k,true,prices,v);
    }

int buyandsell(int currentindex,int transactioncount,bool canbuy, vector<int>&
prices,vector<vector<vector<int>>>& v)
{
    if(currentindex>=prices.size() || transactioncount<=0)
        return 0;


    int key = v[currentindex][transactioncount][canbuy];
    if(key!=-1)
        return key;

   int idle = buyandsell(currentindex+1,transactioncount,canbuy,prices,v);

    if(canbuy)
    {  int buyside = -prices[currentindex] + buyandsell(currentindex+1,transactioncount,0,prices,v);
     return v[currentindex][transactioncount][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] + buyandsell(currentindex+1,transactioncount-1,1,prices,v);
     return v[currentindex][transactioncount][canbuy] = max(sellside,idle);
    }
    }
```

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

Tabulation:

```cpp
int maxProfit(int k,vector<int>& prices) {
        int m = prices.size();
        vector<vector<vector<int>>> v(m+1,vector<vector<int>> (k+1,vector<int> (2,0)));
        for(int currentindex=m-1;currentindex>=0;currentindex--)
        {
            for(int transactioncount=k;transactioncount>=1;transactioncount--)
            {
                for(int canbuy=0;canbuy<=1;canbuy++)
                {
                    int idle = v[currentindex+1][transactioncount][canbuy];

                    if(canbuy)
                    {  int buyside = -prices[currentindex] + v[currentindex+1][transactioncount][0];
                        v[currentindex][transactioncount][canbuy] = max(buyside,idle);
                    }
                    else
                    {    int sellside = prices[currentindex] + v[currentindex+1][transactioncount-1][1];
                        v[currentindex][transactioncount][canbuy] = max(sellside,idle);
                    }
                }
            }
        }  return v[0][k][true];
    }
```

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

Space Optimization:

```cpp
int maxProfit(int k,vector<int>& prices) {
        int m = prices.size();
       vector<vector<int>> next(k+1,vector<int> (2,0));
        for(int currentindex=m-1;currentindex>=0;currentindex--)
        {
       vector<vector<int>> curr(k+1,vector<int> (2,0));
            for(int transactioncount=k;transactioncount>=1;transactioncount--)
            {
                for(int canbuy=0;canbuy<=1;canbuy++)
                {
                    int idle = next[transactioncount][canbuy];

                    if(canbuy)
                    {  int buyside = -prices[currentindex] + next[transactioncount][0];
                        curr[transactioncount][canbuy] = max(buyside,idle);
                    }
                    else
                    {    int sellside = prices[currentindex] + next[transactioncount-1][1];
                        curr[transactioncount][canbuy] = max(sellside,idle);
                    }
                }
            }  next = curr;
        }  return next[k][true];
    }
```

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.

```cpp
int maxProfit(int k,vector<int>& prices) {
        vector<vector<int>> v(prices.size(),vector<int> (2*k,-1));
    return buyandsell(0,0,k,prices,v);
    }

    int buyandsell(int currentindex, int transactioncount, int k,vector<int>& prices,
                   vector<vector<int>>& v)
    {
        if(currentindex>=prices.size() || transactioncount>=2*k)
            return 0;


        if(v[currentindex][transactioncount]!=-1)
            return v[currentindex][transactioncount];

        int idle = buyandsell(currentindex+1,transactioncount,k,prices,v);
        int buy,sell = 0;

        if(transactioncount%2==0)
        {   buy = -prices[currentindex] + buyandsell(currentindex+1,transactioncount+1,k,prices,v);
          return   v[currentindex][transactioncount] = max(buy,idle);
        }
         else
        {   sell = prices[currentindex] + buyandsell(currentindex+1,transactioncount+1,k, prices,v);
         return   v[currentindex][transactioncount] = max(sell,idle);
        }
        }
```


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 Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{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:

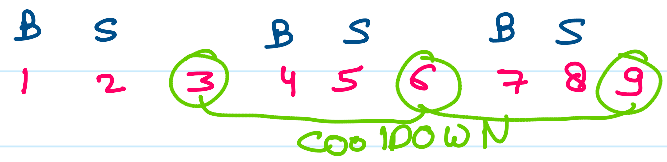
- After you sell your stock, you cannot buy stock on the next day (i.e., cooldown one day).

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:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<int>> v(m,vector<int> (2,-1));
        return buyandsell(0,true,prices,v);
    }

int buyandsell(int currentindex,bool canbuy, vector<int>& prices,vector<vector<int>>& v)
{
    if(currentindex>=prices.size() )
        return 0;


    int key = v[currentindex][canbuy];
    if(key!=-1)
        return key;

   int idle = buyandsell(currentindex+1,canbuy,prices,v);

    if(canbuy)
    {  int buyside = -prices[currentindex] + buyandsell(currentindex+1,0,prices,v);
     return v[currentindex][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] + buyandsell(currentindex+2,1,prices,v);
     return v[currentindex][canbuy] = max(sellside,idle);
    }
    }
```

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

Tabulation:

```
int maxProfit(vector<int>& prices) {
        int m = prices.size();
        vector<vector<int>> dp(m+2,vector<int> (2,0));
       dp[m][0]=0;
       dp[m][1]=0;

      for(int currentindex=m-1;currentindex>=0;currentindex--)
      { for(int canbuy = 0;canbuy<=1;canbuy++)
      {  int idle = dp[currentindex+1][canbuy];

    if(canbuy)
    {  int buyside = -prices[currentindex] +dp[currentindex+1][0];
     dp[currentindex][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] +dp[currentindex+2][1];
       dp[currentindex][canbuy] = max(sellside,idle);
    }
}
    }
       return dp[0][1];
      }
```

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

Space Optimization:

```cpp
int maxProfit(vector<int>& prices) {
        int m = prices.size();

        vector<int> next2(2,0);
        vector<int> next1(2,0);
        vector<int> curr(2,0);

      for(int currentindex=m-1;currentindex>=0;currentindex--)
      {
       int idle = next1[1];
       int buyside = -prices[currentindex] + next1[0];
       curr[1] = max(buyside,idle);

     idle = next1[0];
     int sellside = prices[currentindex] + next2[1];
     curr[0] = max(sellside,idle);
          next2 = next1;
          next1 = curr;
    }
       return next1[1];
     }
```

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

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^{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:

```cpp
int maxProfit(vector<int>& prices,int fee) {
        int m = prices.size();
        vector<vector<int>> v(m,vector<int> (2,-1));
        return buyandsell(0,true,prices,v,fee);
    }

int buyandsell(int currentindex,bool canbuy, vector<int>& prices,vector<vector<int>>& v,int fee)
{
    if(currentindex>=prices.size() )
        return 0;

    int key = v[currentindex][canbuy];
    if(key!=-1)
        return key;

   int idle = buyandsell(currentindex+1,canbuy,prices,v,fee);

    if(canbuy)
    {  int buyside = -prices[currentindex] + buyandsell(currentindex+1,0,prices,v,fee);
     return v[currentindex][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex]-fee + buyandsell(currentindex+1,1,prices,v,fee);
     return v[currentindex][canbuy] = max(sellside,idle);
    }
    }
```

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

Tabulation:

TC: O(M*2)

```cpp
int maxProfit(vector<int>& prices,int fee) {
        int m = prices.size();
       vector<vector<int>> dp(m+1,vector<int> (2,0));
      dp[m][0]=0;
      dp[m][1]=0;

     for(int currentindex=m-1;currentindex>=0;currentindex--)
     { for(int canbuy = 0;canbuy<=1;canbuy++)
     {  int idle = dp[currentindex+1][canbuy];

    if(canbuy)
    {  int buyside = -prices[currentindex] +dp[currentindex+1][0];
     dp[currentindex][canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] +dp[currentindex+1][1]-fee;
       dp[currentindex][canbuy] = max(sellside,idle);
    }
}
    }
       return dp[0][1];
    }
```

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

Space Optimization:

```cpp
int maxProfit(vector<int>& prices,int fee) {
        int m = prices.size();
       vector<int> next(2,0);

      for(int currentindex=m-1;currentindex>=0;currentindex--)
      { vector<int> curr(2,0);
          for(int canbuy = 0;canbuy<=1;canbuy++)
      {  int idle = next[canbuy];

    if(canbuy)
    {  int buyside = -prices[currentindex] +next[0];
     curr[canbuy] = max(buyside,idle);
    }
    else
    {    int sellside = prices[currentindex] + next[1]-fee;
       curr[canbuy] = max(sellside,idle);
    }
}
        next =curr;
    }
        return next[1];
    }
```


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:

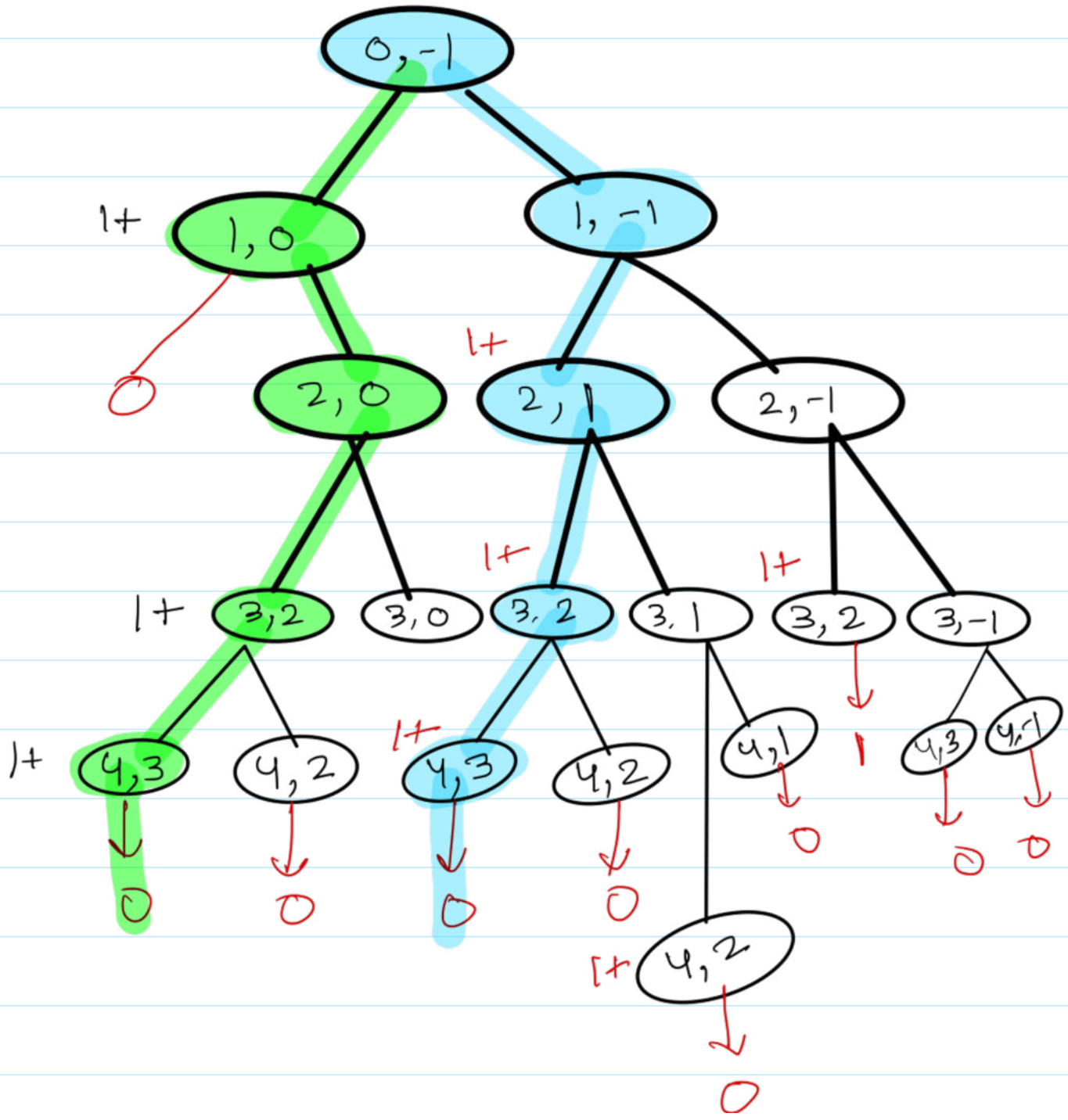
```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Ex-1 : 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.


0, -1
1+
1, 0
1, -1
2, 0
1+
2, 1
2, -1
1+
1+
1+
3, 2
3, 0
3, 2
3, 1
3, 2
3, -1
1+
1+
4, 3
4, 2
4, 3
4, 2
4, 1
4, 3
4, -1
1+
0
0
0
0
0
1
0
0
0
1+
4, 2
2
0

MEMOIZATION:

```cpp
int findlis(int currindex, int prev, int size, vector<int>& nums,
vector<vector<int>>& dp)
    { if(currindex==size)
        return 0;
       if(dp[currindex][prev+1]!=-1)
           return dp[currindex][prev+1];

     int nottake = findlis(currindex+1,prev,size,nums,dp);
     int take = 0;
     if(prev==-1 || nums[currindex]>nums[prev])
        take = 1 + findlis(currindex+1,currindex,size,nums,dp);
      return dp[currindex][prev+1] = max(take,nottake);
    }

    int lengthOfLIS(vector<int>& nums) {
        int size = nums.size();
        vector<vector<int>> dp(size,vector<int> (size+1,-1));
        int prev = -1;
        return findlis(0,prev,size,nums,dp);
    }
```

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

Tabulation:

```cpp
int lengthOfLIS(vector<int>& nums) {
        int size = nums.size();
        vector<vector<int>> dp(size+1,vector<int> (size+1,0));
        for(int currindex=size-1;currindex>=0;currindex--)
        {
            for(int prev = currindex-1;prev>=-1;prev--)
            {
                 int nottake = dp[currindex+1][prev+1];
                 int take = 0;
                 if(prev==-1 || nums[currindex]>nums[prev])
                    take = 1 + dp[currindex+1][currindex+1];
                dp[currindex][prev+1] = max(take,nottake);
            }
        }
      return dp[0][0];
    }
```

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

Space Optimization:

```cpp
int lengthOfLIS(vector<int>& nums) {
        int size = nums.size();
      vector<int> next(size+1,0);
        for(int currindex=size-1;currindex>=0;currindex--)
        {  vector<int> curr(size+1,0);
            for(int prev = currindex-1;prev>=-1;prev--)
            {
                int nottake = next[prev+1];
                int take = 0;
                if(prev==-1 || nums[currindex]>nums[prev])
                   take = 1 + next[currindex+1];
               curr[prev+1] = max(take,nottake);
            } next = curr;
        }
      return next[0];
    }
```

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)

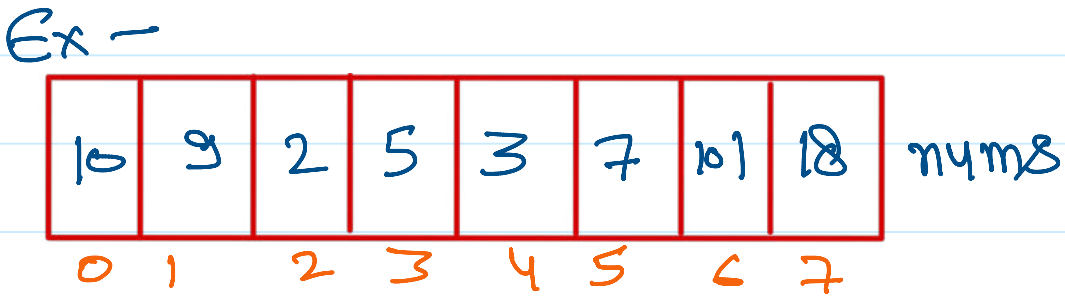
```cpp
int lengthOfLIS(vector<int>& nums) {
        int size = nums.size();
        vector<int> dp(size,1);
        int maxlen=1;
        for(int i=0;i<size;i++)
        {  for(int j=0;j<i;j++)
            { if(nums[i]>nums[j])
                {//if we don't take maximum, it will fail for [0,1,0,3,2,3].
                //dp[0]=1
                //dp[1]=2 ->  [max(1,1+1)=2] at j=0.
                //dp[2]=1 -> [1]
                //dp[3]=3 ->  [max(2,1+2)=3] at j=1, if we dont take max, it
                   will run till
                //j=2, dp[3]=2 ->[1+dp[2]]=1+1=2, so to handle with
                decreasing element case, we are using max
                dp[i]=max(dp[i],1+dp[j]);
                  maxlen=max(dp[i],maxlen);
                }
            }
        }
        return maxlen;
    }
```

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

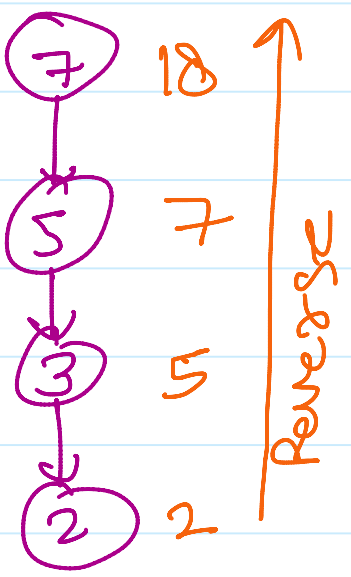
LECTURE - 42 PRINTING LONGEST INCREASING SUSBSEQUENCE

24 June 2022 20:13

Ex –



→



→ It will print 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)

```cpp
int lengthOfLIS(vector<int>& nums) {
        int size = nums.size();
        vector<int> dp(size,1), hash(size);
        int maxlen=1;
        int lastind=0;
        for(int i=0;i<size;i++)
        {  hash[i]=i;
            for(int j=0;j<i;j++)
            { if(nums[i]>nums[j] && dp[j]+1>dp[i])
                {  dp[i]= 1+ dp[j];
                   hash[i] = j;
                }
            }
         if(dp[i]>maxlen)
          { maxlen=dp[i];
             lastind = i;
          }
        }

      //print any of the subsequence
        vector<int> temp;
        temp.push_back(nums[lastind]);
       while(hash[lastind]!=lastind)
       { lastind = hash[lastind];
          temp.push_back(nums[lastind]);
       }
        reverse(temp.begin(),temp.end());
        for(int i =0;i<temp.size();i++)
            cout<<temp[i]<<" ";

        return temp;
    }
```

OR

```cpp
int lengthOfLIS(vector<int>& nums) {
        int size = nums.size();
        vector<int> dp(size,1);
        int maxlen=1;
        int lastind=0;
        for(int i=0;i<size;i++)
        {
            for(int j=0;j<i;j++)
            { if(nums[i]>nums[j] && dp[j]+1>dp[i])
                {  dp[i]= 1+ dp[j];
                }
            }
         if(dp[i]>maxlen)
          { maxlen=dp[i];
             lastind = i;
          }
       }

//         print any of the subsequence
      vector<int> temp;
        temp.push_back(nums[lastind]);
        for(int i =lastind;i>0;i--)
        {
            if(dp[i]==dp[i-1]+1)
                temp.push_back(nums[i-1]);
        }
        reverse(temp.begin(),temp.end());
        for(int i =0;i<temp.size();i++)
            cout<<temp[i]<<" ";


        return temp;
    }
```

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

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

Above methods will give TLE.

[Binary Search - 2B, 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 intution 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, 6]
-1	[-1, 4, 5, 6]
9	[-1, 4, 5, 6, 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 2 3 5 1 4 7]

↓

~~1~~ 2 3 ~~5~~ 7
1 4

→ We can use lower_bound.
→ We need either arr[i] or first index greater than arr[i].

```cpp
int lengthOfLIS(vector<int>& arr) {
        int n = arr.size();
        vector<int> temp;
        temp.push_back(arr[0]);
        int len=1;
        for(int i=1;i<n;i++)
        { if(arr[i]>temp.back())
        {  temp.push_back(arr[i]);
            len++;
        }
            else
        { int index =
            lower_bound(temp.begin(), temp.end(),arr[i])-temp.begin();
            temp[index] = arr[i];
        }
    }
        return len;
       }
```

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 be 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 nums[i]% nums[j] == 0 in place nums[i]<nums[j].

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

Tabulation:

LinkedIn/kapilyadav22

```cpp
vector<int> largestDivisibleSubset(vector<int>& nums)
    {  vector<int> ans;

       int size = nums.size();
       sort(nums.begin(),nums.end());

       vector <int> dp(size,1);
        vector<int> hash(size);
       int maxLength = 1, lastIndex = 0;

       for(int i = 0; i<size; i++) {
           hash[i]=i;
            for(int j = 0; j<i; j++) {
                if(nums[i]%nums[j] == 0  && dp[j]+1>dp[i]) {
                    dp[i] = 1 + dp[j];
                    hash[i] = j ;
                }
            }
           if(maxLength<dp[i]) {
           maxLength = dp[i];
           lastIndex = i;    }
      }

    ans.push_back(nums[lastIndex]);
    while(hash[lastIndex]!=lastIndex)
    {
        lastIndex = hash[lastIndex];
        ans.push_back(nums[lastIndex]);
    }
         reverse(ans.begin(),ans.end());
       return ans;
    }
```

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.

$word_A$ is a **predecessor** of $word_B$ if and only if we can insert **exactly one** letter anywhere in $word_A$ **without changing the order of the other characters** to make it equal to $word_B$.

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

A **word chain** is a sequence of words $[word_1, word_2, ..., word_k]$ with $k >= 1$, where $word_1$ is a **predecessor** of $word_2$, $word_2$ is a **predecessor** of $word_3$, 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", "xbc", "cxbc", "pcxbc", "pcxbcf"].
```

ex- ["a", "b", "ba", "bca", "bda", "bdca"]

+1 +1 +1

= [a, ba, bca, bdca] 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:

```cpp
bool checkPossible(string &s1, string &s2)
    {   int size1 = s1.size();
        int size2 = s2.size();
        int first = 0;
        int second =0;

     if(size1!=(size2+1))
            return false;

        while(first < size1)
        {   if(s1[first]==s2[second])
                { first++;
                  second++; }
            else first++;
        }

       if(first==size1 && second==size2)
            return true;
     return false;
    }

   static bool comp(string &s1, string &s2)
   {  return (s1.size()<s2.size());
   }

    int longestStrChain(vector<string>& words) {
        sort(words.begin(), words.end(),comp);
        int size = words.size();
        vector<int> dp(size,1);
        int maxlen=1;
        for(int i=0;i<size;i++)
        {  for(int j=0;j<i;j++)
            { if(checkPossible(words[i],words[j]))
                {dp[i]=max(dp[i],1+dp[j]);
                }
            }
          maxlen=max(dp[i],maxlen);
```

```
37              maxlen=max(dp[i],maxlen);
38          }
39          return maxlen;
40      }
```

LinkedIn/kapilyadav22

LECTURE - 46 LONGEST BITONIC SUBSEQUENCE

24 June 2022 20:16

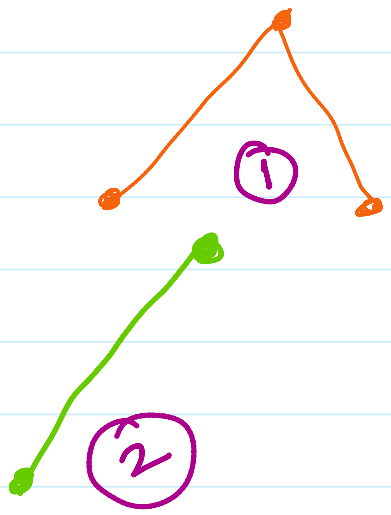
1. arr[] = { 1, 11, 2, 10, 4, 5, 2, 1 }

→ 1 2 10 4 2 1

→ 1 2 10 5 2 1

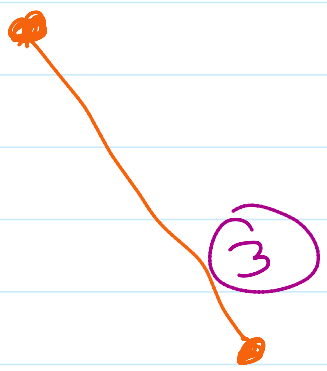
→ 1 11 10 4 2 1

→ 1 11 10 5 2 1



2. arr[] = { 1, 2, 3, 4, 5, 6 }

3. arr[] = { 6, 5, 4, 3, 2, 1 }



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

dp[] = { 1 2 2 3 3 4 2 1 }

dp2[] = { 1 5 2 4 3 3 2 1 }

-1

Bitonic = 1 6 3 6 5 6 3 1

Use a variation of LIS :-

```cpp
int longestBitonicSequence(vector<int>& nums, int n)
{
        vector<int> dp1(n,1);
        for(int i=0;i<n;i++)
        {  for(int j=0;j<i;j++)
            { if(nums[i]>nums[j])
                dp1[i]=max(dp1[i],1+dp1[j]);
            }
        }
        vector<int> dp2(n,1);
        for(int i=n-1;i>=0;i--)
        {  for(int j=n-1;j>i;j--)
            { if(nums[i]>nums[j])
                dp2[i]=max(dp2[i],1+dp2[j]);
            }
        }
    int maxi=0;
     for(int i =0;i<n;i++)
     { maxi = max(maxi,dp1[i]+dp2[i]-1);
     }
    return maxi;
    }

```

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 < arr.length - 1` such that:
  - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
  - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`

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

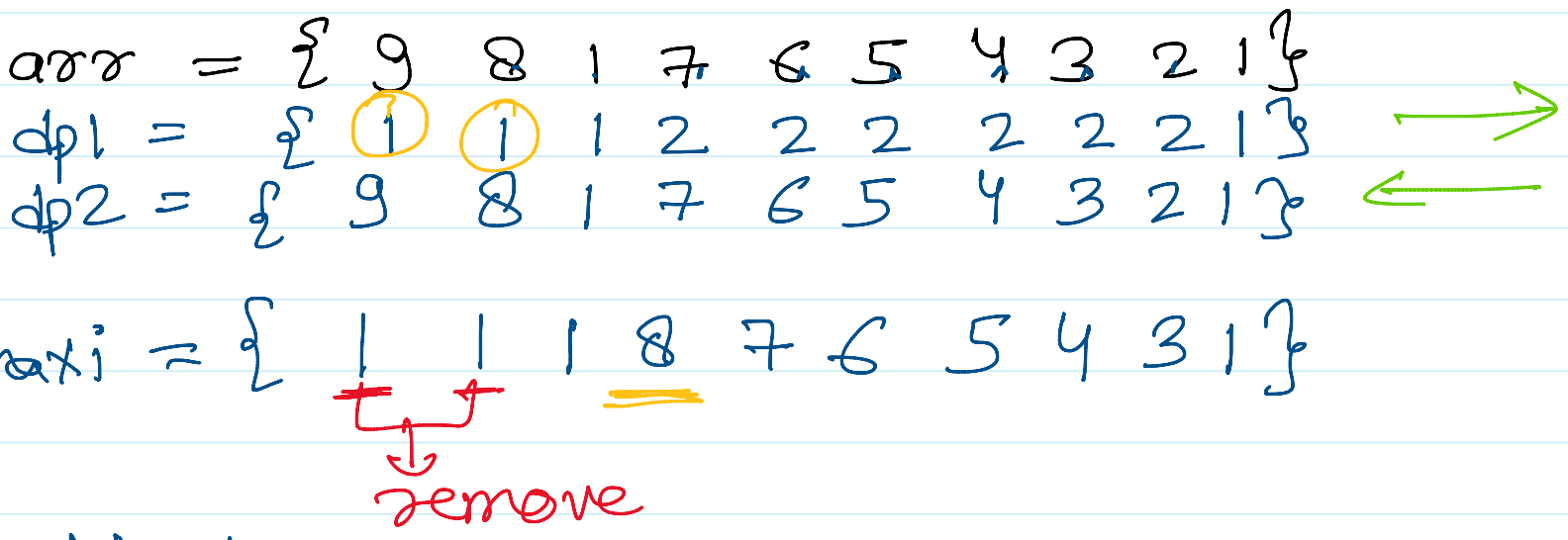
Example 1:

```
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.



```cpp
int minimumMountainRemovals(vector<int>& nums) {
        int n = nums.size();
        vector<int> dp1(n,1);
        for(int i=0;i<n;i++)
        {  for(int j=0;j<i;j++)
            { if(nums[i]>nums[j])
                dp1[i]=max(dp1[i],1+dp1[j]);
            }
        }
        vector<int> dp2(n,1);
        for(int i=n-1;i>=0;i--)
        {  for(int j=n-1;j>i;j--)
            { if(nums[i]>nums[j])
                dp2[i]=max(dp2[i],1+dp2[j]);
            }
        }
    int maxi=0;
     for(int i =0;i<n;i++)
     {  //cout<<dp1[i]<<"   "<<dp2[i]<<" "<<endl;
         if(dp1[i] > 1 && dp2[i] > 1)
         maxi = max(maxi,dp1[i]+dp2[i]-1);
     }
    return n-maxi;
    }
```

LinkedIn/kapilyadav22

LECTURE - 47 NUMBER OF LONGEST INCREASING SUBSEQUENCES || LIS

24 June 2022 20:16

arr[] → {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 :-

arr[] → {1, 5, 4, 3, 2, 6, 7, 10, 8, 9}
dp[] = {1,}
cnt[] = {1,}

for i=1, j=0
arr[] → {1, 5, 4, 3, 2, 6, 7, 10, 8, 9}
Since, arr[i] > arr[j] and dp[i] < 1+dp[j]
length of the lis will increase but not the counts.
dp[] = {1, 2}
cnt[] = {1, 1}

[1,5]

for i=2, j=0
arr[i] > arr[j], and dp[2] < 1+dp[0]
dp[] = {1, 2, 2}
cnt[] = {1, 1, 1}

[1,4]

j=1, arr[i] < arr[j], do nothing.
dp[] = {1, 2, 2}
cnt[] = {1, 1, 1}

[1,4]

for i=3, j=0
arr[i] > arr[j], and dp[3] < 1+dp[0]
dp[] = {1, 2, 2, 2}
cnt[] = {1, 1, 1, 1}
j=1, arr[i] < arr[j], do nothing.
j=2, X

for i=4, j=0
arr[i] > arr[j] and dp[4] < 1+dp[0]
dp[] = {1, 2, 2, 2, 2}

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

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

j=1, ✗

j=2, ✗

j=3, ✗

for i=5, j=0

arr[i] > arr[j] & dp[5] < 1+dp[0] (1 < 1+1)

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

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

j=1, arr[i] > arr[j] & dp[5] < 1+dp[1] (2 < 1+2)

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

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

j=2, arr[i] > arr[j] & dp[5] == 1+dp[2] (3 == 1+2)

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

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

j=3, arr[i] > arr[j] & dp[5] == 1+dp[3] (3 == 1+2)

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

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

j=4, arr a[i] > arr[j] & dp[5] == 1+dp[4] (3 == 1+2)

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],

&& dp[j]+1 < dp[i]

① dp[i] = 1 + dp[j];

② cnt[i] = cnt[j];

② arr[i] > arr[j] && dp[j]+1 == dp[i]

update the cnt;

cnt[i] += cnt[j];

③ And store 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.

```cpp
int findNumberOfLIS(vector<int>& nums) {
        int size = nums.size();
         int maxlen=1;
        vector<int> dp(size,1);
        vector<int> count(size,1);

        for(int i=1;i<size;i++)
        {  for(int j=0;j<i;j++)
            { if(nums[i]>nums[j]  && dp[i]<dp[j]+1)
                {  dp[i]=dp[j]+1;
                    count[i]=count[j];
                }
        else if(nums[i]>nums[j]  && dp[i]==dp[j]+1)
                    count[i]+=count[j];

            }
          maxlen = max(maxlen,dp[i]);
        }

        int nolis=0;
        for(int i=0;i<size;i++)
        { if(dp[i]==maxlen)
           nolis+=count[i];
        }

       return nolis;
    }
```

LECTURE - 48 & 49 MATRIX CHAIN MULTIPLICATION (Recursion DP)

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^{th} matrix has the dimensions **(arr[i-1] x arr[i])**.

Example 1:

```
Input: N = 5
arr = {40, 20, 30, 10, 30}
Output: 26000
Explaination: There are 4 matrices of dimension
40x20, 20x30, 30x10, 10x30. 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.
```

$(1+2+3) \times 5$

$(1+2) \times (3 \times 5)$

ABC

A = 10 x 30

B = 30 x 5

C = 5 x 60

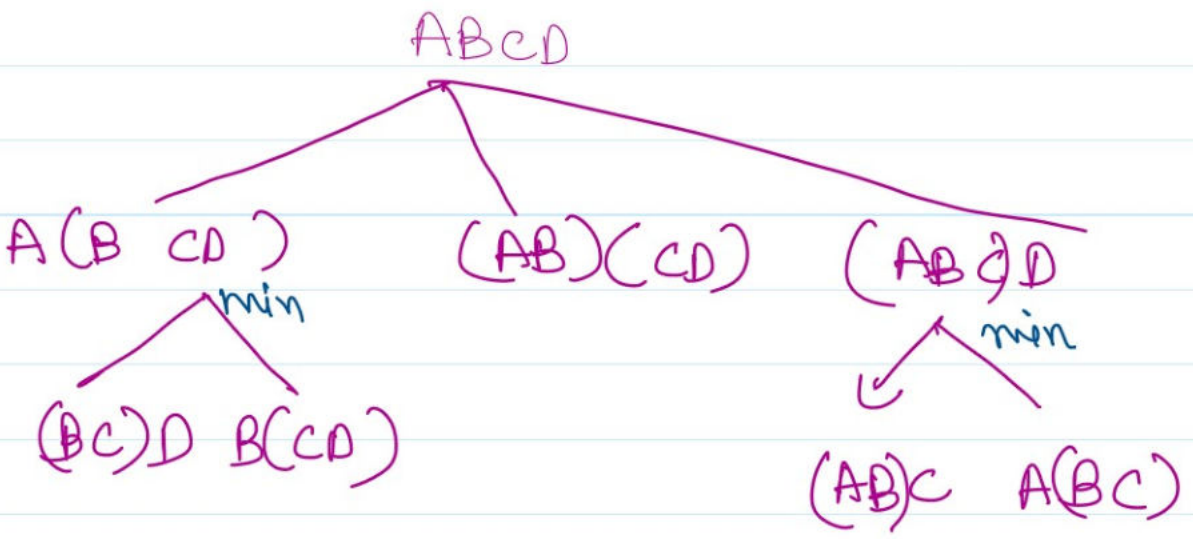
→ operations = $10 \times 30 \times 5 = 1500$

$(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60)$

$= 1500 + 3000 = 4500$. Min operations

$A(BC) = (10 \times 30 \times 60) + (30 \times 5 \times 60)$
$= 18000 + 18000 = 27,000.$

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}

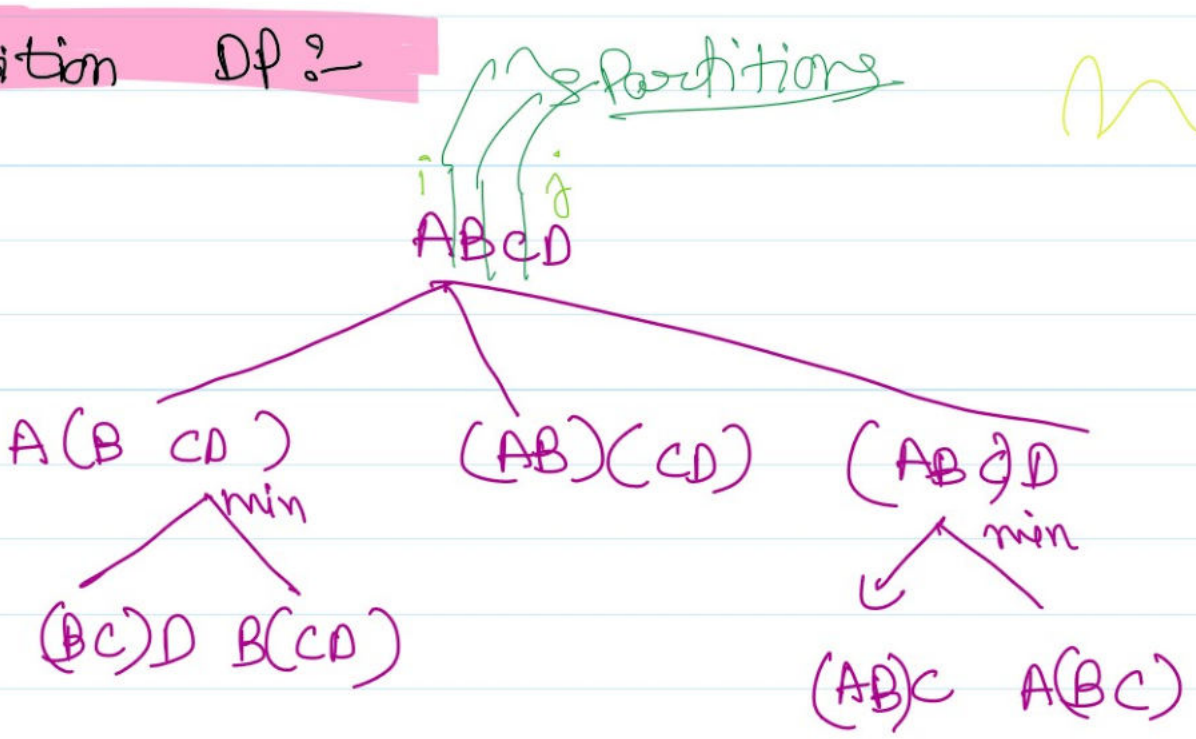
dimension of N-1 matrix

N = 5

A → 10×20
B → 20×30
C → 30×40
D → 40×50

1st ⇒ A[0] × A[1]
2nd → A[1] × A[2]
3rd → A[2] × A[3]
ith → A[i-1] × A[i]

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.

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

1 2 1 2 1 2

arr[] = { 10, 20, 30, 40, 50 }
0 1 2 3 4
A B C D

i → 1

j → n-1

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

```
f(i, j)
{   if (i == j) return 0;
```

[10 20 30 40 50]
i j
A B C D

k = (i to j - 1)

→ $f(i,k) + f(k+1, j)$

k = 1 f(1, 1) + f(2, 4)
k = 2 f(1, 2) + f(3, 4)

$k=2$ f(1,2) , f(3,4)
$k=3$ f(1,3) , f(4,4).

OR

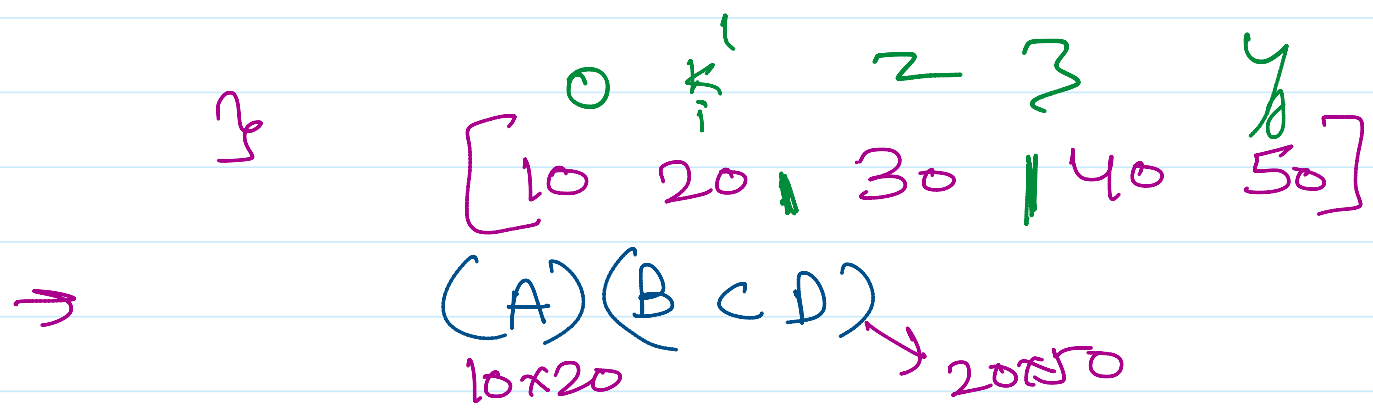
$k = (i+1 \rightarrow j)$

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

OR

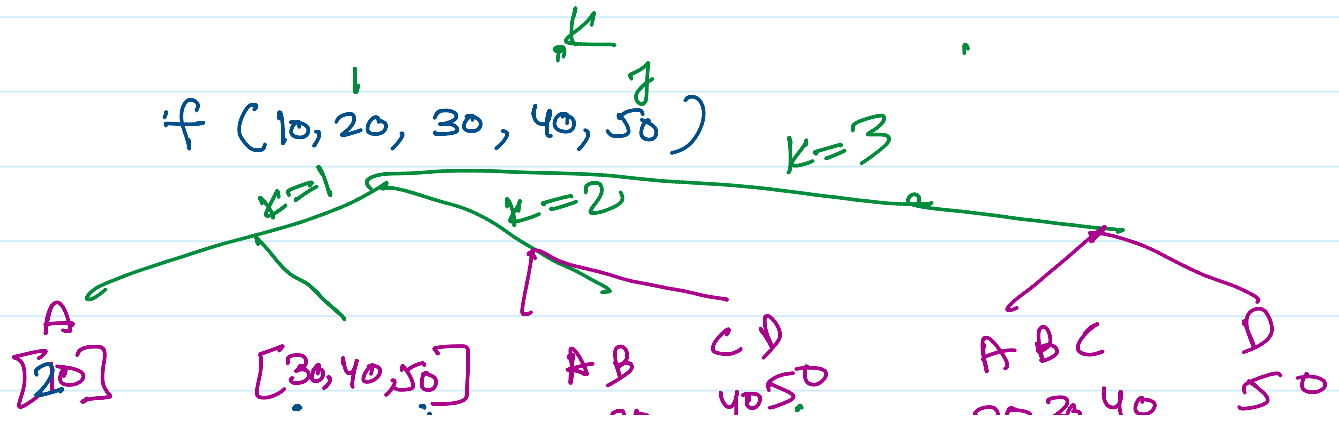
$k=2$ f(1,1), f(2,4)
$k=3$ f(1,2) f(3,4)
$k=4$ f(1,3) f(4,4)

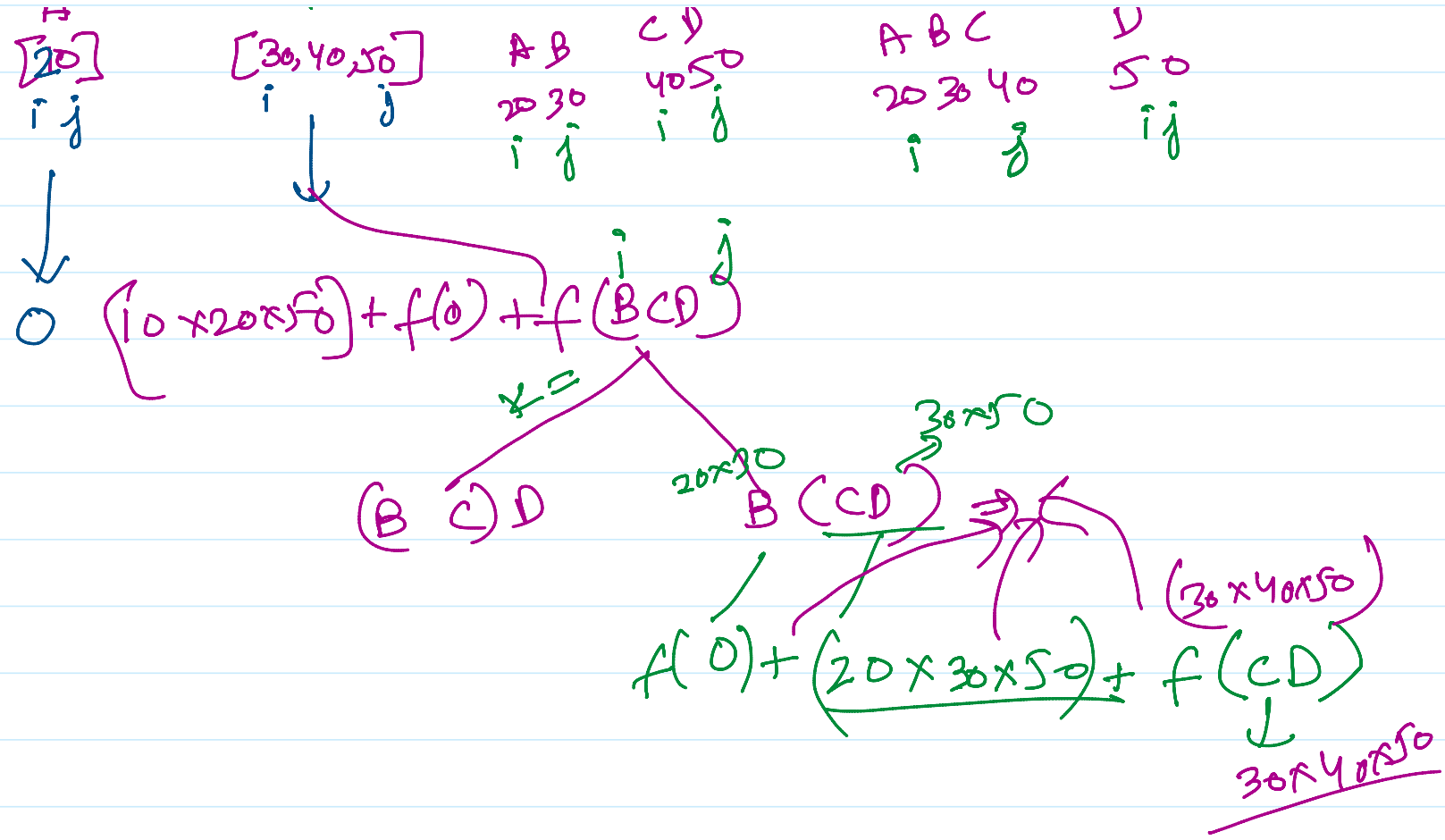
mini = INT_MAX
for(k = i to k = j-1)
{ steps = (A[i-1] x arr[k] x arr[j]) + f(i,k) + f(k+1, j)
mini = min(mini, steps);
}



$\Rightarrow 10 \times 20 \times 50 =$

A[i-1] A[k] A[j] + (solving cost)





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.

```cpp
int f(int i,int j, int arr[],vector<vector<int>> &dp)
{
    if(i==j)
    return 0;
    int mini = INT_MAX;
    if(dp[i][j]!=-1)
      return dp[i][j];
    for(int k=i;k<j;k++)
    {  int steps = arr[i-1]*arr[k]*arr[j] + f(i,k,arr,dp) + f(k+1,j,arr,dp);
        mini = min(mini,steps);
    }
    return dp[i][j]=mini;
}
    int matrixMultiplication(int N, int arr[])
    { vector<vector<int>> dp(N, vector<int> (N,-1));
        return f(1,N-1,arr,dp);
    }
```

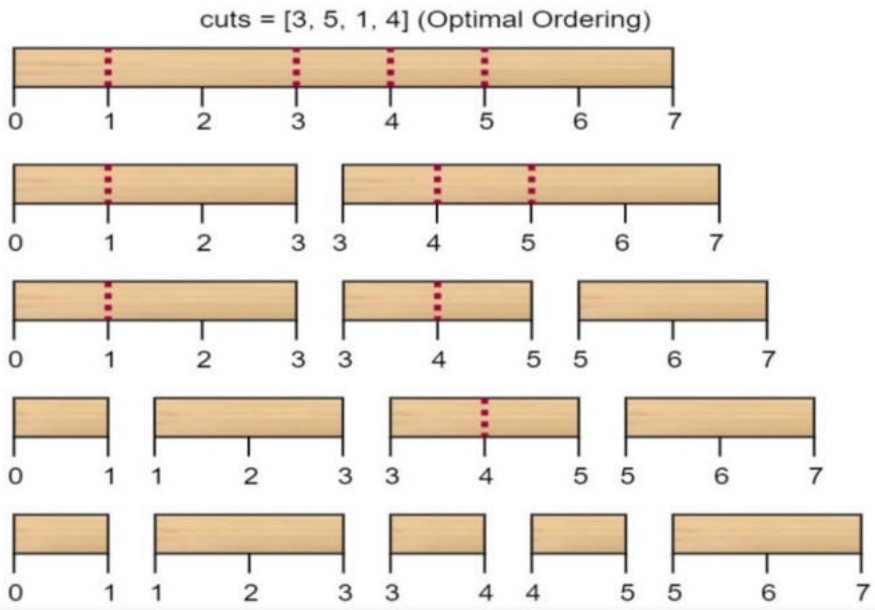
Tabulation:


```cpp
int matrixMultiplication(int N, int arr[])
    { vector<vector<int>> dp(N, vector<int> (N));
    for(int i =1;i<N;i++)
    dp[i][i]=0;
    for(int i=N-2;i>=1;i--)
    { for(int j=i+1;j<N;j++)
    {  int mini = INT_MAX;
    for(int k=i;k<j;k++)
    {  int steps = arr[i-1]*arr[k]*arr[j] + dp[i][k] +dp[k+1][j];
        mini = min(mini,steps);
    }
    dp[i][j] = mini;
    }
    }
    return dp[1][N-1];
    }
```

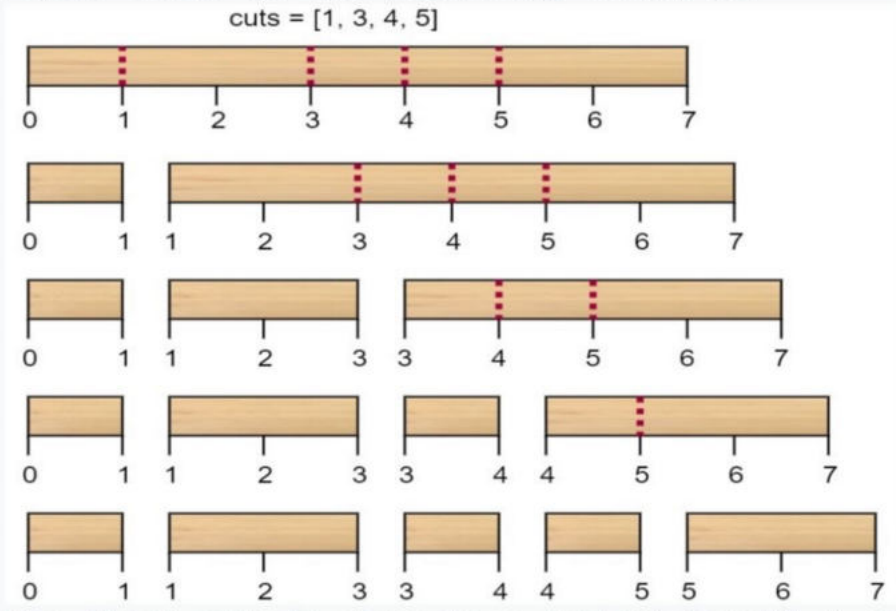
LECTURE - 50 MINIMUM COST TO CUT THE STICK (Pattern DP)

Friday, 17 June 2022 12:49 AM

Example 1:

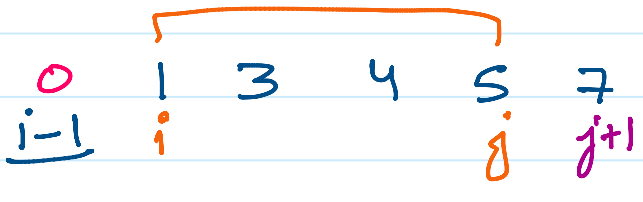


Input: n = 7, cuts = [1,3,4,5]
Output: 16
Explanation: Using cuts order = [1, 3, 4, 5] as in the input leads to the following scenario:



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 4 is.



⇒ cuts[j+1] - cuts[i-1]

Ex → cuts = 1 3 4 5

→ insert 0 at the beginning and length of the stick at the end.

0	1	2	3	4	5
0	1	3	4	5	7
	↓ i			↓ j	

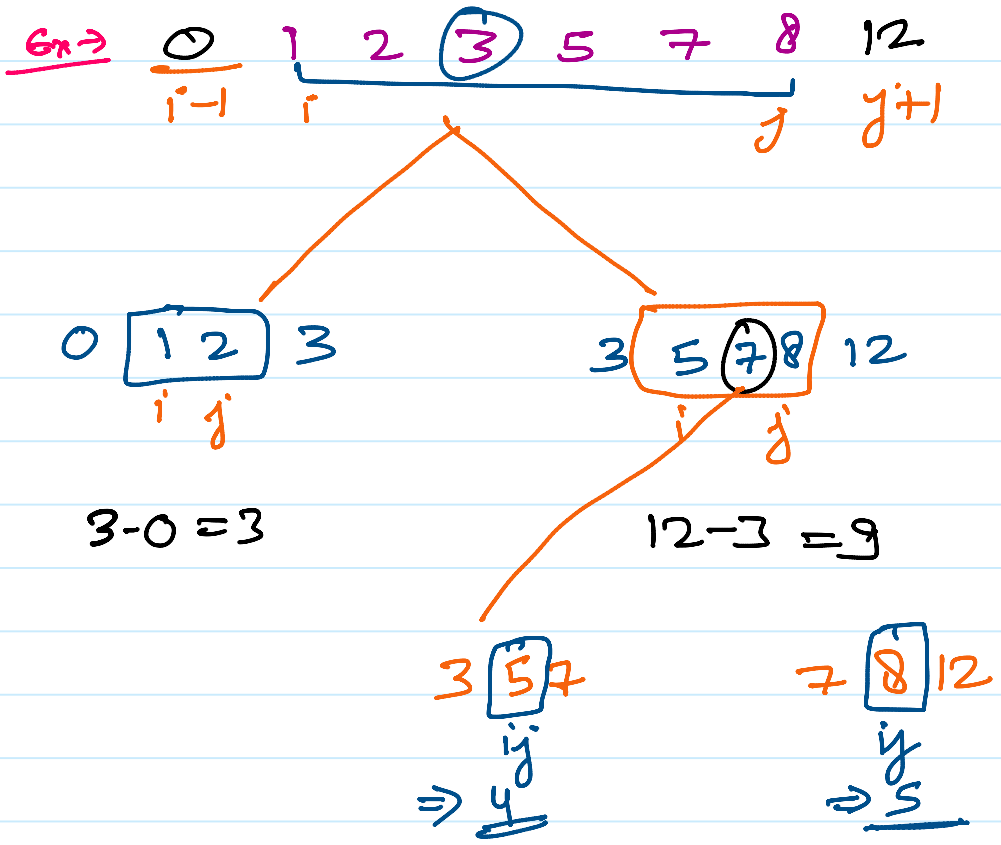
→ f(i,j)

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

  f(ind = i to j)

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

  mini = min(mini, cost);
}
```



Memoization:

```cpp
int f(int i,int j,vector<int> & cuts,vector<vector<int>> &dp)
    {   if(i>j) return 0;
        int mini = INT_MAX;
     if(dp[i][j]!=-1)
        return dp[i][j];
         for(int ind =i;ind<=j;ind++)
        { int cost = cuts[j+1]-cuts[i-1] + f(i,ind-1,cuts,dp) +
                                                f(ind+1,j,cuts,dp);
            mini = min(mini,cost);
        }
     return dp[i][j]=mini;
    }
    int minCost(int n, vector<int>& cuts) {
        int size = cuts.size();
        vector<vector<int>> dp(size+1,vector<int> (size+1,-1));
        vector<int> newcuts= cuts;
        newcuts.push_back(n);
        newcuts.insert(newcuts.begin(),0);
        sort(newcuts.begin(),newcuts.end());
        return f(1,size,newcuts,dp);
     }
```


Tabulation:

```cpp
int minCost(int n, vector<int>& cuts) {
        int size = cuts.size();
        vector<vector<int>> dp(size+2,vector<int> (size+2,0));
        vector<int> newcuts= cuts;
        newcuts.push_back(n);
        newcuts.insert(newcuts.begin(),0);
        sort(newcuts.begin(),newcuts.end());

        for(int i =size;i>=1;i--)
        { for(int j =1;j<=size;j++)
        {   if(i>j)continue;
          int mini = INT_MAX;
        for(int ind =i;ind<=j;ind++)
        { int cost = newcuts[j+1]-newcuts[i-1] + dp[i][ind-1] +
                                                dp[ind+1][j];
            mini = min(mini,cost);
        }
         dp[i][j]=mini;
        }
        }
  return dp[1][size];
     }
```

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^{th} balloon, you will get `nums[i - 1] * nums[i] * 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
```

Ex → {3 1 5 8}

→ If I burst 3, i will get 1×3×1=3

{1, 5, 8}

→ If I burst 1, i will get 1×1×5=5

{5, 8}

→ If I burst 5, i will get 1×5×8 = 40

{8}

→ If I burst 8, i will get 1×8×1=8

56

Another way:-

{3, ×, 5, 8}

3×1×5=15

[3, x, 5, 8]

↓ $3 \times 1 \times 5 = 15$

[3, ~~5~~, 8] $3 \times 5 \times 8 = 120$

↓

{ ~~3~~, 8 } $1 \times 3 \times 8 = 24$

↓

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

167

→ If we consider our array elements like

$[b_1 \quad b_2 \quad \boxed{b_3} \quad b_4 \quad b_5 \quad b_6]$

→ If we take b_3 first, can we solve for $\{b_1, b_2\}$ & $\{b_4, b_5, b_8\}$ separately.

→ NO. WHY?

→ Let's say we solve for b_1 b_2, and we choose b_2, the right of b_2 is b_4 after bursting b_3. so b_2 is dependent on b_4 so while bursting b_2, we will take

$b_1 \times b_2 \times b_4$ or

→ if i burst out b_4 after b_3, then, b_4 is dependent on b_2.

$b_2 \times b_4 \times b_5$

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

[3, x, 5, 8]

↑ $3 \times 1 \times 5 = 15$

{ 3, ~~x~~, 5, 8 }

3 × 1 × 5 = 15

{ 3, ~~5~~, 8]

3 × 5 × 8 = 120

{ ~~x~~ 8 }

1 × 3 × 8 = 24

{ 8 }

1 × 8 × 1 = 8

167

Ex → { b_1 b_2 ~~b_3~~ b_4 b_5 b_6 }

i — ind — j

→ a[i-1] × a[ind] × a[j+1] + f(i, ind-1) + f(ind+1, j).

{ b_1 b_2 b_3 ~~b_4~~ b_5 b_6 }

[b_1 b_4] or [b_2 b_4] or [b_3 b_4] or [b_4 b_5] or [b_4 b_6]

{ b_4 } → b_4 will be in second last level

Recursion :- TC = Exponential

Memoization:

```cpp
int findmaxcoins(int startind,int endind,vector<int>& nums,
vector<vector<int>>& dp)
    { if(startind>endind)
        return 0;
     if(dp[startind][endind]!=-1)
         return dp[startind][endind];
     int maxi = INT_MIN;
     for(int ind = startind;ind<=endind;ind++)
     {
         int cost = nums[startind-1] * nums[ind]* nums[endind+1] +
                    findmaxcoins(startind, ind-1,nums,dp) +
                      findmaxcoins(ind+1, endind,nums,dp);
       maxi = max(maxi,cost);
     }
        return dp[startind][endind]=maxi;
    }


    int maxCoins(vector<int>& nums) {
        int n = nums.size();
        nums.push_back(1);
        vector<vector<int>> dp(n+1, vector<int> (n+1,-1));
        nums.insert(nums.begin(),1);
        return findmaxcoins(1,n,nums,dp);
    }
```

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

Tabulation:

```cpp
int maxCoins(vector<int>& nums) {
        int n = nums.size();
        nums.push_back(1);
        nums.insert(nums.begin(),1);
        vector<vector<int>> dp(n+2, vector<int> (n+2,0));

        for(int  startind=n;startind>=1;startind--)
        {  //endind==n
        for(int endind=1;endind<=n;endind++)
            {   int maxi = INT_MIN;
                if(startind>endind) continue;
            for(int ind = startind;ind<=endind;ind++)
             {
                 int cost = nums[startind-1] * nums[ind]*
                   nums[endind+1] + dp[startind][ind-1] + dp[ind+1][endind];
               maxi = max(maxi,cost);
             }
       dp[startind][endind]=maxi;
            }
        }
        return dp[1][n];
    }
```

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

LECTURE-52 EVALUATE BOOLEAN EXPRESSION TO TRUE

24 June 2022 20:24

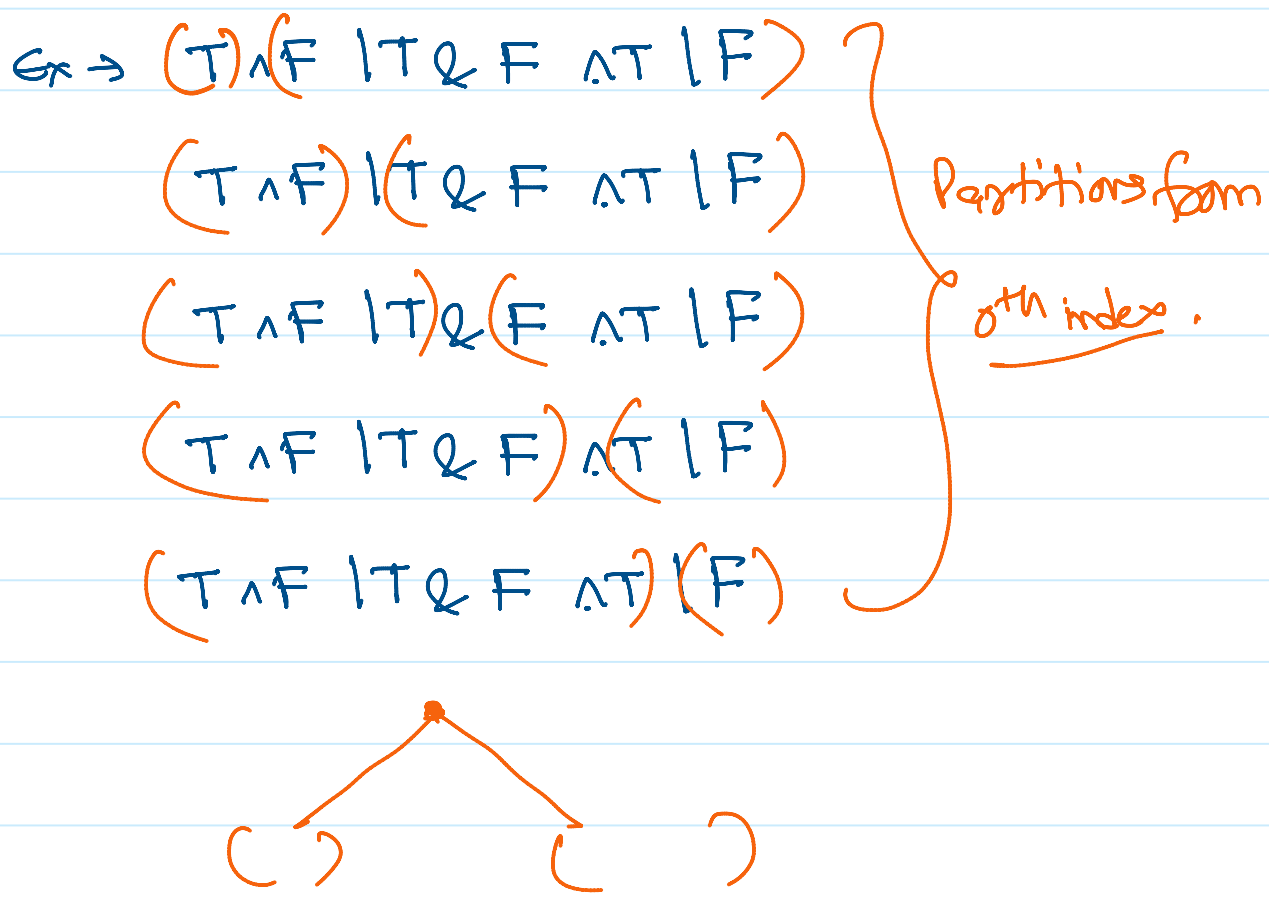
Boolean Parenthesization

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.

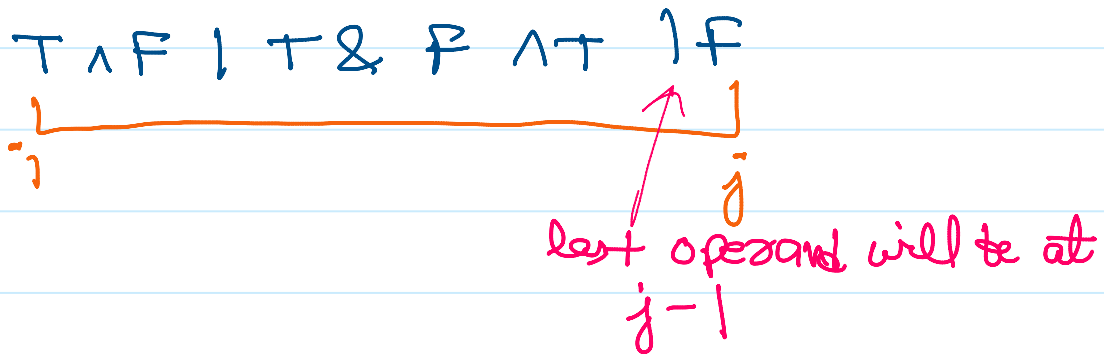
Example 1:

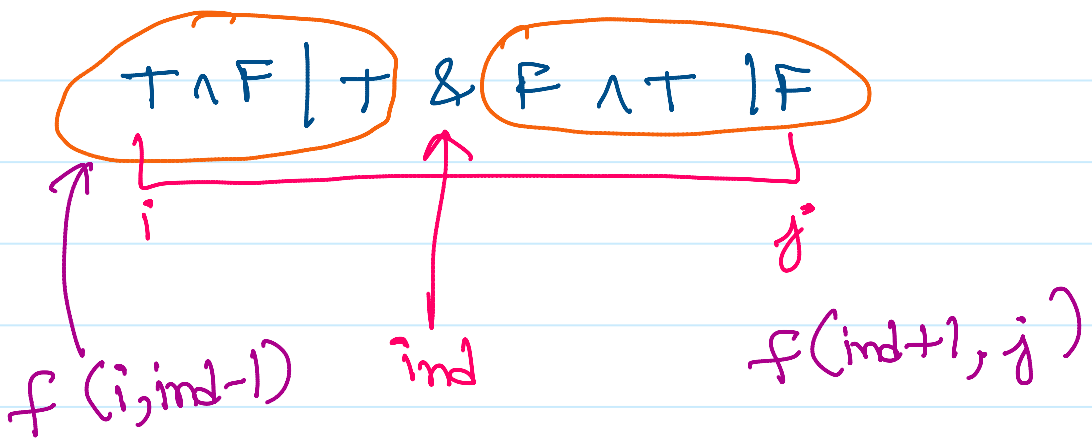
```
Input: N = 7
S = T|T&F^T
Output: 4
Explaination: The expression evaluates
to true in 4 ways ((T|T)&(F^T)),
(T|(T&(F^T))), (((T|T)&F)^T) and (T|((T&F)^T)).
```



→ 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 × y total ways in case of & operator

→ In case of OR :-

left	right
T = x_1	T = x_2
F = x_4	F = x_3

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

→ In case of XOR :-

left	right
T = x_1	T = x_2
F = x_4	F = x_3

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

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 expesion gives us true or not.
If we need True to make expesion true, isTrue will be 1, if we need false to make expression true, isTrue will be 0.

→ for partioning, there can be four possibilities,

lT = f(i, ind-1, 1)
lF = f(i, ind-1 0)
rT = f(ind+1, j, 1)
rF = f(ind+1, j, 0

Recursion → exponential.

Memoization → (N x N x 2) x N ≅ (N^3)

TOTAL WAYS TO MAKE IsTrue False

(i.) & Operator :-
ways = (rT x lF) + (rF x lF) + (rF x lT)

(ii.) | operator:-
ways = (rF x lF)

(iii.) xor (^) operator
ways = (lT x rT) + (lF x rF)

Q.) But Why we need to find Istrue = false?
Ans) Ex → T ^ T & F.
→ we want to make this expesion true. so if we partition (T ^ T) & (F)
LS RS

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

$F \& F = F$.

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

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

→ so we have seen that to make the expression true, we need Istrue = False in the Right side. i.e. we need the result of & expression as false.

Memoization:

```cpp
int mod = 1003;
int findways(int startind, int endind,int isTrue, string S,vector<vector<vector<int>>>& dp)
{
    if(startind>endind) return 0;
    if(startind==endind)
    {   if(isTrue)
            return S[startind]=='T';
        else return S[startind]=='F';
    }

    if(dp[startind][endind][isTrue]!=-1)
        return dp[startind][endind][isTrue];
    int ways =0;
    for(int ind = startind+1;ind<=endind-1;ind+=2)
    {       int lT = findways(startind,ind-1,1,S,dp);
            int lF = findways(startind,ind-1,0,S,dp);
            int rT = findways(ind+1,endind,1,S,dp);
            int rF = findways(ind+1,endind,0,S,dp);

            if(S[ind]=='&')
            {  if(isTrue)
                    ways = (ways + (lT*rT)%mod)%mod;
               else
               ways = (ways + (lT*rF)%mod + (lF*rT)%mod + (lF*rF)%mod)%mod;
            }
            else if(S[ind]=='|')
            {  if(isTrue)
                    ways = (ways + (lT*rF)%mod + (lT*rT)%mod + (lF*rT)%mod)%mod;
                else
                     ways = ways + ((lF*rF)%mod);
            }
            // T^T = F, T^F = T, F*F = F, F*T = T
            else if(S[ind]=='^')
            {
                if(isTrue)
                    ways = (ways + (lT* rF)%mod + (lF * rT))%mod;
                 else
                    ways = (ways + (lT*rT)%mod +(lF*rF)%mod)%mod;
            }

    }
     return dp[startind][endind][isTrue]= ways;
}
    int countWays(int N, string S){
        vector<vector<vector<int>>> dp(N,vector<vector<int>> (N, vector<int> (2,-1)));
        return findways(0,N-1,1,S,dp);
    }
```

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
```

Ex a a b

a|a b

a|(a|b)

a a|b

min(2,1) = 1

Ex-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 → in wc, N-1 cuts will be there.

→ So we will cut at every index one by one and at every index, we will check for palindrome ;-

→ if start to end is palindrome, m need

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

Memoization:

```cpp
bool isPalindrome(string &s, int start, int end)
{
  while(start<=end)
  {
    if(s[start]!=s[end])
      return 0;
    start++;
    end--;
  }
  return 1;
}

 int minCut(string s) {
 vector<int> v(s.size()+1,-1);
   return minimumcuts(s, 0,s.size()-1,v);
 }

 int  minimumcuts(string &s, int start, int end, vector<int>&v)
{
     if(start>end)
     return 0;

  if(isPalindrome(s,start,end)==1)
    return 0;

if(v[start]!=-1)
  return v[start];

   int ans =INT_MAX;
  for(int currentcut=start;currentcut<end;currentcut++)
  {
     if(isPalindrome(s,start,currentcut))
     {
       int righthalf = minimumcuts(s,currentcut+1,end,v);
        int tempans = 1 + righthalf;
        ans = min(ans, tempans);
     }
  }
  return v[start]=ans;
}
```

TC: O(N*N) * O(N) for checking palindrome
SC: O(N) + O(N) recursive space.

Tabulation:

```cpp
bool isPalindrome(string &s, int start, int end)
{
  while(start<=end)
  {
    if(s[start]!=s[end])
      return 0;
    start++;
    end--;
  }
  return 1;
}

 int minCut(string s) {
      int n =s.size();
      vector<int> v(n+1,0);
      v[n]=0;
      for(int start = n-1; start>=0;start--)
      {  int ans =INT_MAX;
    for(int currentcut=start;currentcut<n;currentcut++)
    {
       if(isPalindrome(s,start,currentcut))
       {
         int righthalf = v[currentcut+1];
          int tempans = 1 + righthalf;
          ans = min(ans, tempans);
       }
    }
      v[start]=ans;
       }
      return v[0]-1;
    }
```

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 9 | 10 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.

Recursion : TC - exponential

Memoization:

```cpp
int findmaxsum(int ind,vector<int>& arr, int k, vector<int>& dp)
    {  int n = arr.size();
       if(ind==n) return 0;
     if(dp[ind]!=-1)
         return dp[ind];
     int len = 0;
     int maxi = INT_MIN;
     int maxans = INT_MIN;

     for(int j = ind;j<min(ind+k,n);j++)
     {
         len++;
         maxi = max(maxi,arr[j]);
         int sum = (len* maxi) + findmaxsum(j+1,arr,k,dp);
         maxans = max(maxans,sum);
     }
     return dp[ind] = maxans;
    }

    int maxSumAfterPartitioning(vector<int>& arr, int k) {
     int n = arr.size();
    vector<int> dp(n,-1);
        return findmaxsum(0,arr,k,dp);
    }
```

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

Tabulation:

```cpp
int maxSumAfterPartitioning(vector<int>& arr, int k) {
     int n = arr.size();
     vector<int> dp(n+1,0);
    for(int ind =n-1;ind>=0;ind--)
    {  int len = 0;
     int maxi = INT_MIN;
     int maxans = INT_MIN;

     for(int j = ind;j<min(ind+k,n);j++)
     {
         len++;
         maxi = max(maxi,arr[j]);
         int sum = (len* maxi) + dp[j+1];
         maxans = max(maxans,sum);
     }
        dp[ind] = maxans;
    }
         return  dp[0];
    }
```

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

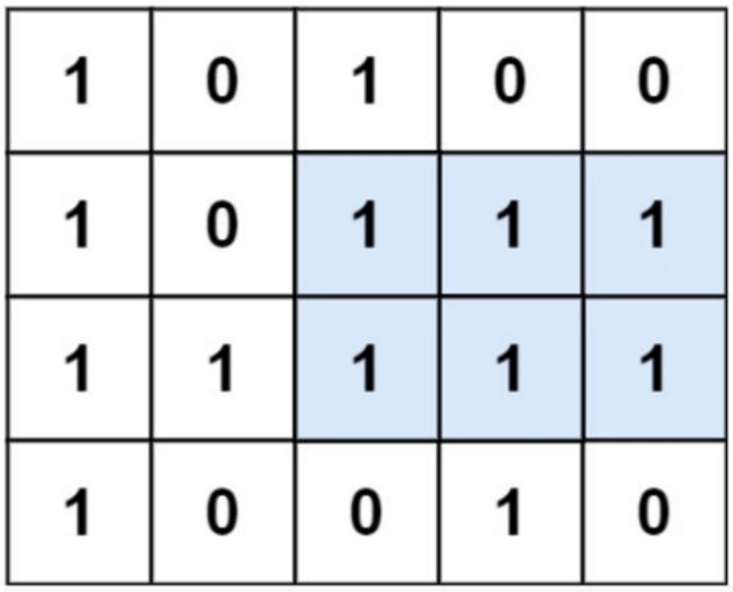
LinkedIn/kapilyadav22

85. Maximal Rectangle

Hard 6915 110 Add to List Share

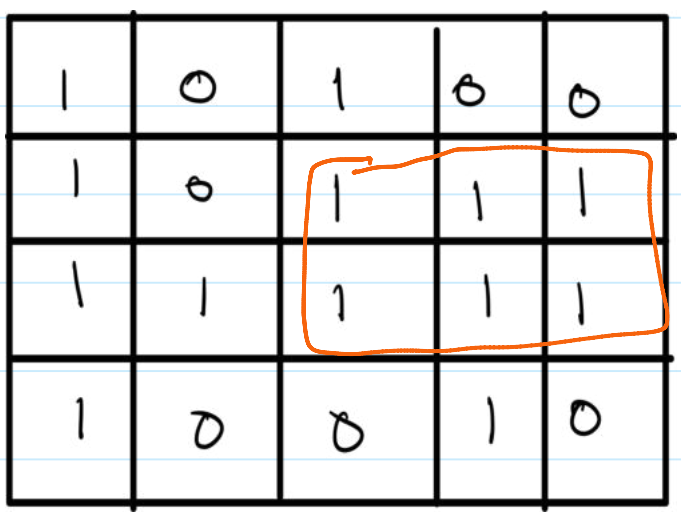
Given a `rows x cols` binary `matrix` filled with `0`'s and `1`'s, find the largest rectangle containing only `1`'s and return *its area*.

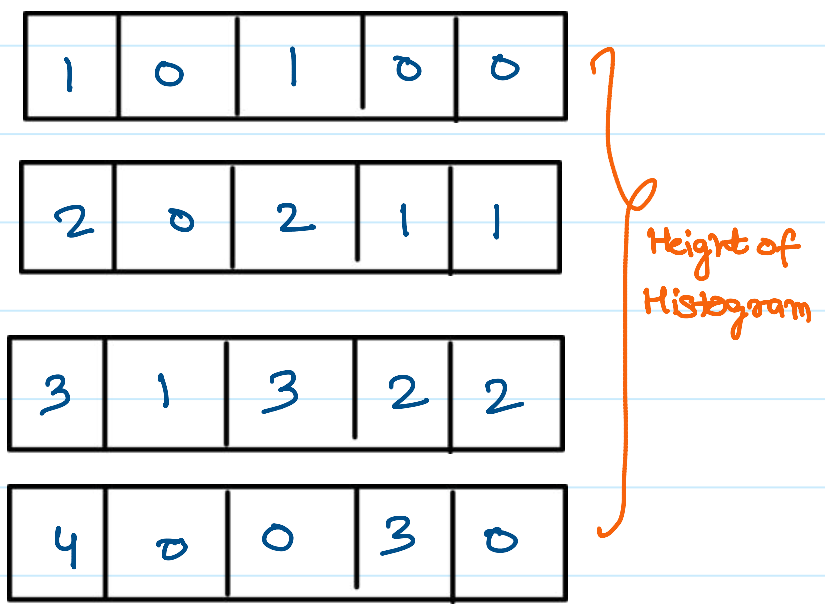
Example 1:




```
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





```cpp
int largestRectangleArea(vector < int > & histo) {
      stack < int > st;
      int maxA = 0;
      int n = histo.size();
      for (int i = 0; i <= n; i++) {
        while (!st.empty() && (i == n || histo[st.top()] >= histo[i])) {
          int height = histo[st.top()];
          st.pop();
          int width;
          if (st.empty())
            width = i;
          else
            width = i - st.top() - 1;
          maxA = max(maxA, width * height);
        }
        st.push(i);
      }
      return maxA;
    }

    int maximalRectangle(vector<vector<char>>& matrix) {
        int maxarea =0;
        int n = matrix.size();
        int m = matrix[0].size();
        //TC : O(n*(m+n))
        vector<int> height(m,0);
        for(int i =0;i<n;i++)
        {for(int j= 0;j<m;j++)
            {  if(matrix[i][j]=='1')
                height[j]++;
             else height[j]=0;
            }
         //TC: O(n)
         int area = largestRectangleArea(height);
         maxarea = max(area,maxarea);
            }
        return maxarea;
        }
```

TC:O(n*(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

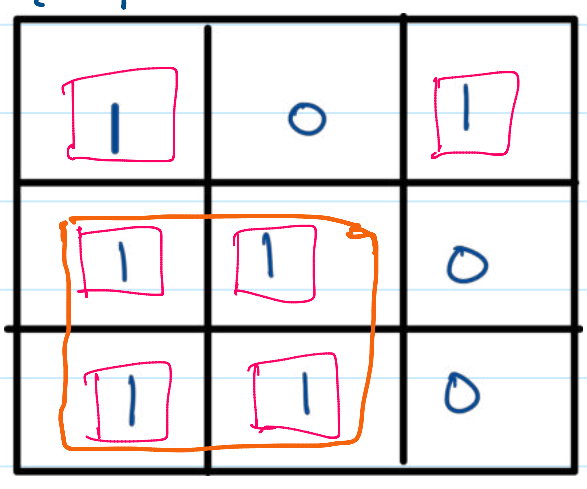
Medium 3463 61 Add to List Share

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.
```

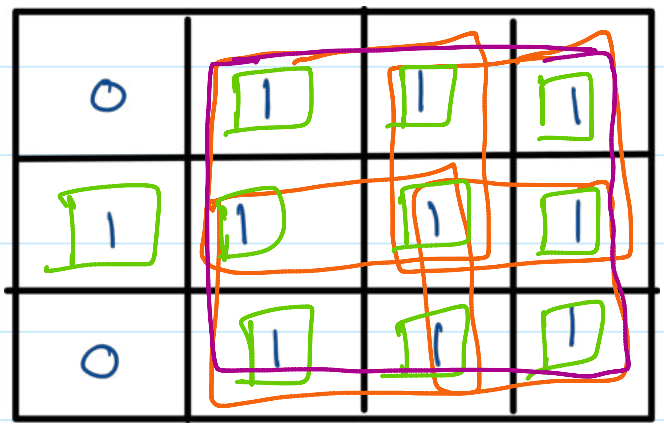
Ex-1



→ There are 6, size 1 squares.

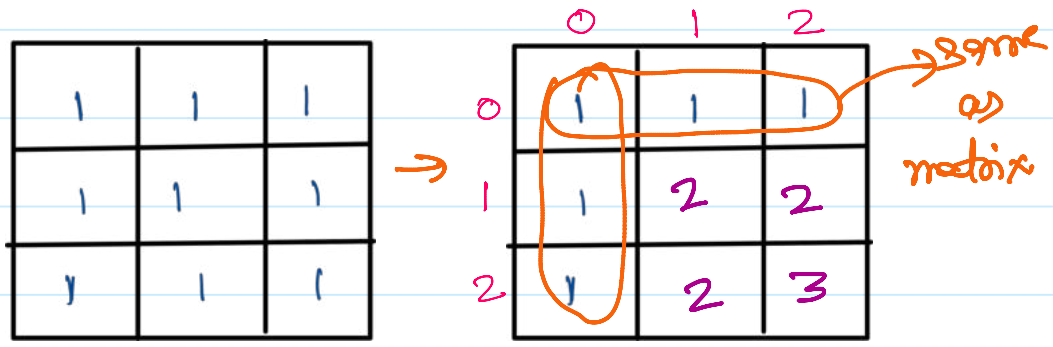
→ There are 1, size 2 squares.

Ex-2



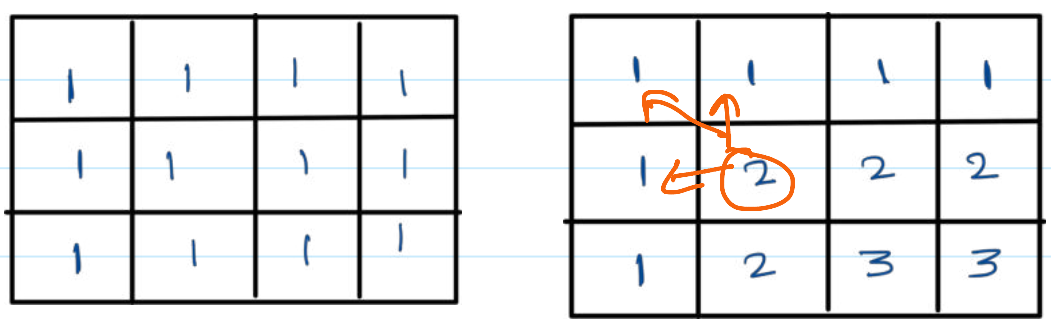
→ There are 10, size 1 squares.
→ There are 4, size 2 squares.
→ There are 1, size 3 squares.

Ex :-



→ 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 ?



→ 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.

```cpp
int countSquares(vector<vector<int>>& matrix) {
        int n = matrix.size();
        int m = matrix[0].size();
        vector<vector<int>> dp(n,vector<int>(m,0));
        for(int j=0;j<m;j++)
            dp[0][j]=matrix[0][j];
        for(int i=0;i<n;i++)
            dp[i][0]=matrix[i][0];
        for(int i =1;i<n;i++)
        {
            for(int j=1;j<m;j++)
            {   if(matrix[i][j]==0)
                    dp[i][j]=0;
               else
               {
                   dp[i][j]= 1 + min(dp[i-1][j],min(dp[i-1][j-1],dp[i][j-1]));
               }

            }
        }
        int sum =0;
        for(int i=0;i<n;i++)
        { for(int j=0;j<m;j++)
        {
            sum+=dp[i][j];
        } }

        return sum;
    }
```

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

Congrats,

You Have studied DP

Kapilyadav22