Lecture 31: Buy & Sell Stocks with Transaction Fees

Same Concept of Stocks II, it's only that after A Complete TRANSACTION we have to pay a fees.
So while adding the price of stock during selling SUBTRACT ~~Fees~~ Transaction as well.

The Rest Remains the Same.

Lecture 32: Longest Increasing Subsequence

It is another modification of the subsequence problem where we use the pick/not pick algo.
It's just that Since it is INCREASING, we have to keep a prev_variable to keep track, that the elements should increasing.
The Recurrence / presenting in terms of index ⇒

f(ind, prev) ⇒ At the current 'ind' what is Longest Increasing Subs Achieved

The code below will work for element >=0 NOT for negative numbers.

Time Comp ⟶ O(N x N)
Space Comp ⟶ O(N x N) + O(N)

initialize it with INT_MIN.

```
int f(int ind, int prev, vector<int>& arr) {

    if (ind == arr.size())
        return 0;
    if (dp[ind][prev] != -1)
        return dp[ind][prev];

    int notpick = f(ind+1, prev, arr, dp);
    int pick = 0;
    if (arr[ind] > prev)
        pick = 1 + f(ind+1, arr[ind], arr, dp);

    return dp[ind][prev] = max(pick, notpick);
}
```

Tabulation:

The Same Rules will be applied..

1. Bas Case
   ↳ Since we will initialize the dp[][] by 0, no need to modify / change anything.
2. Traverse the Changing parameters in reverse fashion
   ind ⇒ n-1 ⟶ 0
   prev ⇒ ind-1 ⟶ -1
3. Copy recurrence & make sure to follow the right shift rule.

Code

```
int f(vector<int> &arr) {

    int n = arr.size();
    vector<vector<int>> dp(n+1, vector<int>(n+1, 0));

    for(int ind = n-1; ind >= 0; ind--) {
        for(int prev = ind-1; prev >= -1; prev--) {

            int len = 0 + dp[ind+1][prev+1];
            if(prev == -1 || arr[ind] > arr[prev])
                len = max(len, 1 + dp[ind+1][ind+1]);

            dp[ind][prev+1] = len;
        }
    }
    return dp[0][-1+1]
```

Now, Since there is 'ind' & 'ind+1' we can space optimize it using the same old prev & ~~curr~~ curr vector's concept.

Time → $O(N^2)$
Space → $O(N) \times 2$

Tabulation 2 (SC - O(N))

Now Try understand,
We declard a dp[] of size (n), where

dp[i] → Signifies the longest IS that ends at inden (i).

Eg:-

| 5 | 4 | 11 | 1 | 16 | 8 |

– Arr

See, initially we can can that individually every element is it own LIS. Thus initialise the dp[] by 1.

① At (5) → Since there is no prev inden, we move on

dp – | 1 | 1 | 1 | 1 | 1 | 1 |
i ↗

② At (4) → We have a prev as (5), which is greater Thus not a LIS, we move on

dp – | 1 | 1 | 1 | 1 | 1 | 1 |
i ↗

③ At (11) → We have two prev 5 & 4, and we can take any one of them into LIS, Thus changing it to 1+ dp[prev] i.e

dp – | 1 | 1 | 2 | 1 | 1 | 1 |
i ↗

④ At (1) → We can't anything.

| 1 | 1 | 2 | 1 | 1 | 1 |

(5) At (16) → We keep add dp[prev] values to dp[i], and we a check of max.
Notice after taking (11) & (16) into LIS
the dp becomes 3, because 11 ⇒ 2
Thus dp[i] = 1 + dp[prev].

dp — | 1 | 1 | 2 | 1 | 3 | 1 |

(6) At (8) → Only one out of 5, 4, 1 can be picked

∴ dp — | 1 | 1 | 2 | 1 | 3 | 2 |

Keeping a global maximum we get our length = 3

```
int f(vector<int>& arr){
    int n = arr.size();
    vector<int> dp(n+1, 1);
    int ans = -1;
    for(int i = 0, i<n; i++){
        for(int j = 0; j<i ; j++){
            dp[i] = max(dp[i], 1+dp[j]);
            if (dp[j] < dp[i])
                dp[i] = max(dp[i], 1+dp[j]);
        }
        maxi = max(maxi, dp[i]);
    }
    return maxi;
}.
```

M T W T F S S

Page No.:

Date: 07

YOUVA

Binary Search.

Refer the blog in bookmarks for better understanding.

Here we will take a temp[], to ~~CAL~~ CALCULATE LIS, NOTE: To Calculate, it's not the ans

* The logic is to keep on adding elements in back if curr > prev, (OR)

* replace the curr element with first Greater element in temp using Binary Search (OR) Lower Bound function

Eg

1	7	8	4	5	6	-1	9

~~S-1)~~ S-1) temp → 1

S-2) temp → 1, 7

S-3) temp → 1, 7, 8

S-4) temp → 1, ~~7~~, 8
↓
temp → 1, 4, 8
} replaced 7 by 4, since it's the first Greater element.

S-5) temp → 1, 4, 5

S-6) temp → 1, 4, ~~5~~, 6

S-7) temp → -1, 4, 5, 6

S-8) -1, 4, 5, 6, 9] temp

The Size of temp is our answer.

```
int f(vector<int> &arr) {

    vector<int> temp;
    temp.push_back(arr[0]);

    for(int i=1; i<arr.size(); i++) {
        if (arr[i] > temp.back())
            temp.push_back(arr[i]);
        else {
            int ind = lower_bound(temp.begin(), temp.end(),
                      arr[i]) - temp.begin();

            temp[ind] = arr[i];
        }
    }
    return temp.size();
}
```

TC → O(N log N)
SC → O(N).

Lecture 383 → Largest Divisble Subset.

Try Imagine, if we sort the array and instead of find Longest Increasing by dp[i] > dp[j], we replace our logic to longiest divisble by dp[i] % dp[j] == 0

And Since the Array is sorted all pair in front will be divisble with all in back.

The code will be exactly same As Printing LIS

```
~~int~~ vector<int> f(vector<int>& arr){
    int n=nums.size(), last=0, mani=1;
    vector<int> dp(n,1), hash(n); temp;
    sort(nums.bagin(), nums.end());
    for(int i=0; i<n; i++){ * hash[i]=i;
        for(int j=0; j<i; j++){
            if(nums[i]%nums[j]==0 && dp[i]<dp[j]+1){
                dp[i]=dp[j]+1;
                hash[i]=j;
            }
        }
        if(dp[i]>mani){
            mani=dp[i];
            last=i;
        }
    }
    temp.push_back(nums[last]);
    while(last != hash[last]){
        last=hash[last];
        temp.push_back(nums[last]);
    }
    return temp;
}
```

M T W T F S S
Page No.:
Date: 180
YOUVA

Lecture 34: Largest Chain String

This question also works on the previous logic, Simply Sord the String according to their size in decreasing and check if there is only one extra character between the current and it's next String. If yes add it to your answer otherwise don't.

```
int f(vector<string> & arr) {

    int n=arr.size(), maxi=-1;
    vector<int> dp(n,1);
    sort(arr.begin(), arr.end(), comp);

    for(int i=0; i<n; i++) {
        for(int j=0; j<i, j++) {
            if(check(arr[i], arr[j] && dp[i]<dp[j]+1)
                dp[i]=dp[j]+1;
        }
        maxi = max(maxi, dp[i]);
    }
    return maxi;
}

bool comp(string & a, String & b)
    return a.size() > b.size();
```

PTO

```
bool Check( string &a, string &b){
    if(a.size() != b.size()+1)
        return false;

    int i=0, j=0;
    while(i<a.size()){
        if(a[i]==b[j]){
            i++;
            j++;
        }
        else
            i++;
    }
    return i==a.size() && j==b.size();
}
```

We need to check that if after iterating over longer both the strings are completely exhausted if yes, then return true, Otherwise return false.

#Lecture 35 - Longest Bitonic Subsequence.

It says the sequence can be any of 4 forms

(1) First Increasing, Then Decreasing.
(2) Only Increasing.
(3) Only Decreasing.
(4) First Decreasing, then Increasing.

for This, Now try to Understand.

Arr = 1, 11, 2, 10, 4, 5, 2, 1

Ans ⇒ 6 (1, 2, 10, 5, 2, 1)

Now see, if we calculate LIS, dp we get

dp[] ⇒ | 1 | 2 | 2 | 3 | 3 | 4 | 2 | 1 |

Now if we reverse the arr (or) make dp[] from Last to ~~be~~ first. we can find Decreasing but with LIS logic.

Arr = 1, 2, 5, 4, 10, 2, 11, 1.

dp[] = | 1 | 2 | 3 | 3 | 4 | 2 | 5 | 1 |

Now if we Add both dp tables and Subtract (1) because of common element we will get, but reverse dp2 because the indexing got reversed.

ans = | 1 | 6 | 3 | 6 | 5 | 6 | 3 | 1 |

ans = 6

Lecture 36 :- Number Of Longest Increasing Subsequence.

We just need to tell how many LIS are present in an array.

arr = [1, 3, 5, 4, 7]

Ans ⇒ 2 ({1,3,5,7}, {1,3,4,7}).

for this keep a cnt[] initialised to 1, .

```
if (nums[i] > nums[j] && dp[i] < dp[j]+1)
        dp[i] = dp[j]+1;
        cnt[i] = cnt[j]  // make them same, as in
                         that since till here we probably have
                         only 1 or a variable 'x' ways so we
                         carry it forward.

else if (nums[i] > nums[j] && dp[i] == dp[j]+1)
        cnt[i] += cnt[j]  // in the Abv eg; this would
                          occur at nums[i] = 7; where we find
                          multiple value same and we encounter
                          2 paths which lead to LIS = 4, Thus we
                          add the Ans
```

** Code Remains exactly the same, with abv changes and then keep a global variable to store ans, whereever maxi == dp[i]; add it's cnt[i] value to ans.