43) Longest increasing subsequence

10 December 2024 02:28

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

From <https://leetcode.com/problems/longest-increasing-subsequence/description/>

```
    int helper(vector<int>& nums, int ind, int prev_ind, int n, vector<vector<int>>
&dp){
        if(ind == n) return 0;
        if(dp[ind][prev_ind+1] != -1) return dp[ind][prev_ind+1];
        int len = 0 + helper(nums, ind+1, prev_ind, n, dp);
        if(prev_ind == -1 || nums[ind] > nums[prev_ind]){
            len = max(len, 1 + helper(nums, ind+1, ind, n, dp));
        }
        return dp[ind][prev_ind+1] = len;
    }
    int lengthOfLIS(vector<int>& nums) {
        int n = nums.size();
        vector<vector<int>>dp(n, vector<int>(n+1, -1));
        return helper(nums, 0, -1, n, dp);
    }
```

//space optimization: very imp, as iss se related saari problem main yhi as a sample code use hoga

```
    int lengthOfLIS(vector<int>& arr) {
        int n = arr.size();
        vector<int> dp(n,1);

        int maxi = 1;

        for(int i=0; i<=n-1; i++){

            for(int prev_index = 0; prev_index <=i-1; prev_index ++){

                if(arr[i] > arr[prev_index] && 1 + dp[prev_index] > dp[i]){
                    dp[i] = 1 + dp[prev_index];
                }
            }

            if(dp[i] > maxi)
                maxi = dp[i];
        }
        return maxi;
    }
```

44) Print longest increasing subsequence

11 December 2024 01:56

Given an integer **n** and an array of integers **arr**, return the **Longest Increasing Subsequence** which is *Index-wise* lexicographically smallest.
Note - A subsequence S1 is **Index-wise lexicographically smaller** than a subsequence S2 if in the first position where S1 and S2 differ, subsequence S1 has an element that appears **earlier** in the array arr than the corresponding element in S2.

From <https://www.geeksforgeeks.org/problems/printing-longest-increasing-subsequence/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=printing-longest-increasing-subsequence>

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

    int len = solve(i + 1, prev, nums, dp);
    if (prev == -1 || nums[i] > nums[prev]) {
        len = max(len, 1 + solve(i + 1, i, nums, dp));
    }

    return dp[i][prev + 1] = len;
}

vector<int> longestIncreasingSubsequence(int n, vector<int>& nums) {
    vector<vector<int>> dp(n, vector<int>(n + 1, -1));
    int len = solve(0, -1, nums, dp); // Fill DP table

    // Reconstruct the LIS
    vector<int> lis;
    int i = 0, prev = -1;

    while (i < n && len > 0) {
        if (prev == -1 || nums[i] > nums[prev]) {
            int take = 1 + (i + 1 < n ? dp[i + 1][i + 1] : 0);
            if (take == len) {
                lis.push_back(nums[i]);
                prev = i;
                len--; // Reduce target length
            }
        }
        i++;
    }

    return lis;
}
```

45) *Longest Increasing Subsequence (special stl used)

11 December 2024 02:13

Given an array **arr[]** of integers, the task is to find the **length** of the **Longest Strictly Increasing Subsequence (LIS)**.

A subsequence is considered **strictly increasing** if each element in the subsequence is strictly less than the next element.

From <https://www.geeksforgeeks.org/problems/longest-increasing-subsequence-1587115620/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=longest-increasing-subsequence>

```
int longestSubsequence(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 ind = lower_bound(temp.begin(), temp.end(), arr[i]) - temp.begin();
            temp[ind] = arr[i];
        }
    }

    return len;
}
```

46) Largest Divisible subset

11 December 2024 02:48

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.

From <https://leetcode.com/problems/largest-divisible-subset/description/>

```
int solve(int i, int prev, vector<int>& nums, vector<vector<int>>& dp) {
    if (i == nums.size()) return 0;
    if (dp[i][prev + 1] != -1) return dp[i][prev + 1];
    int notTake = solve(i + 1, prev, nums, dp);
    int take = 0;
    if (prev == -1 || nums[i] % nums[prev] == 0) {
        take = 1 + solve(i + 1, i, nums, dp);
    }
    return dp[i][prev + 1] = max(take, notTake);
}
vector<int> largestDivisibleSubset(vector<int>& nums) {
    int n = nums.size();
    sort(nums.begin(), nums.end()); // Important for divisibility checks
    vector<vector<int>> dp(n, vector<int>(n + 1, -1));
    solve(0, -1, nums, dp);
    // Reconstruct subset
    vector<int> res;
    int i = 0, prev = -1;
    int len = dp[0][0]; // Max length
    while (i < n && len > 0) {
        if (prev == -1 || nums[i] % nums[prev] == 0) {
            int take = 1 + (i + 1 < n ? dp[i + 1][i + 1] : 0);
            if (take == len) {
                res.push_back(nums[i]);
                prev = i;
                len--;
            }
        }
        i++;
    }
    return res;
}
```

47) Longest Increasing Subsequence

11 December 2024 03:09

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.

From <https://leetcode.com/problems/longest-string-chain/description/>

```
    bool compare(string& s1, string& s2){
        if(s1.size() != s2.size() + 1) return false;

        int first = 0;
        int second = 0;

        while(first < s1.size()){
            if(second < s2.size() && s1[first] == s2[second]){
                first ++;
                second ++;
            }
            else first ++;
        }
        if(first == s1.size() && second == s2.size()) return true;
        else return false;
    }
    static bool comp(string& s1, string& s2){
        return s1.size() < s2.size();
    }

    int longestStrChain(vector<string>& arr){
        int n = arr.size();

        //sort the array

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

        int maxi = 1;

        for(int i=0; i<=n-1; i++){

            for(int prev_index = 0; prev_index <=i-1; prev_index ++){

                if( compare(arr[i], arr[prev_index]) && 1 + dp[prev_index] > dp[i]){
                    dp[i] = 1 + dp[prev_index];
                }
            }

            if(dp[i] > maxi)
                maxi = dp[i];
        }
        return maxi;
    }
```

48) Longest Bitonic Sequence

11 December 2024 03:36

A Bitonic Sequence is a sequence of numbers that is first strictly increasing and then strictly decreasing.

A strictly ascending order sequence is also considered bitonic, with the decreasing part as empty, and same for a strictly descending order sequence.

From <https://www.naukri.com/code360/problems/longest-bitonic-sequence_1062688?source=youtube&campaign=striver_dp_videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=striver_dp_videos&leftPanelTabValue=PROBLEM>

```
#include <bits/stdc++.h>
int longestBitonicSubsequence(vector<int>& arr, int n)
{
    vector<int> dp1(n, 1);
    vector<int> dp2(n, 1);
    for (int i = 0; i < n; i++) {
        for (int prev_index = 0; prev_index < i; prev_index++) {
            if (arr[prev_index] < arr[i]) {
                dp1[i] = max(dp1[i], 1 + dp1[prev_index]);
            }
        }
    }
    for (int i = n - 1; i >= 0; i--) {
        for (int prev_index = n - 1; prev_index > i; prev_index--) {
            if (arr[prev_index] < arr[i]) {
                dp2[i] = max(dp2[i], 1 + dp2[prev_index]);
            }
        }
    }
    int maxi = -1;
    for (int i = 0; i < n; i++) {
        maxi = max(maxi, dp1[i] + dp2[i] - 1);
    }
    return maxi;
}
```

49) No of longest increasing subsequence

11 December 2024 03:52

Given an integer array nums, return *the number of longest increasing subsequences.*
Notice that the sequence has to be **strictly** increasing.

From <https://leetcode.com/problems/number-of-longest-increasing-subsequence/description/>

```
    int findNumberOfLIS(vector<int>& arr) {
        int n = arr.size();
        vector<int> dp(n, 1);
        vector<int> ct(n, 1);
        int maxi = 1;
        for (int i = 0; i < n; i++) {
            for (int prev_index = 0; prev_index < i; prev_index++) {
                if (arr[prev_index] < arr[i] && dp[prev_index] + 1 > dp[i]) {
                    dp[i] = dp[prev_index] + 1;
                    ct[i] = ct[prev_index];
                } else if (arr[prev_index] < arr[i] && dp[prev_index] + 1 == dp[i])
{
                    ct[i] = ct[i] + ct[prev_index];
                }
            }
            maxi = max(maxi, dp[i]);
        }
        int numberOfLIS = 0;
        for (int i = 0; i < n; i++) {
            if (dp[i] == maxi) {
                numberOfLIS += ct[i];
            }
        }
        return numberOfLIS;
    }
```