October 25, 2025

DP FORM 2 & 3 DEEP DIVE

* Form 2

deals with subsequences

[_ _ _ _ (i)]

↳ Best you can build ending over here.

DP(i, ...) → Best __ ending at i

↑ restrictions

Ques 1. A. Find the no. of Bitonic subsequences of an array. Bitonic means something that strictly increases & then decreases. ↗ ↘

Example : 1 3 2 5 1

Bitonic Sequences :

(1) 3 (2) (5) (1) → 1 2 5 1

(1) (3) 2 5 (1) → 1 3 1

(1) (3) (2) 5 (1) → 1 3 2 1

So this array has a lot of bitonic sequences and we have to find total such no. of bitonic seq.

B. Find the longest bitonic sequence

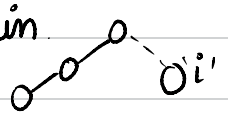
↳ length

C. Find the count of no. of longest bitonic sequence that can be made.

Solution A.

Step 1 : Form : Form 2

This problem deals with subsequence where we are building a bitonic sequence with the ques : "Can we append the element 'i'th element with the last element or not" This is what we are keeping track of. Its like building a chain.



Subsequence problems are generally form 2.

② Now we have to define our DP fxn with its parameters & its return value but before that let's see what can be the restrictions which will then define our parameters

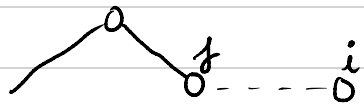
→ Peak : It is important for us to know if we have surpassed peak or are we yet reach peak as it will determine the nature i. You see we are simply extending our bitonic array. let's say the last element is j of this array. If we have not seen peak yet, then we can only accept i>j but if we have seen the peak then i must be less than j. (i<j)

Seen : i (<j)

Not seen : i (>j)

DP(i, seen_peak) → # of bitonic subseq that ends at i.

AlgoZenith



→ Can we extend this solution or not that ends at i, it only depends on 'j' and if we have seen the peak or not.

→ $DP(j, 1)$ which will tell us how many sequences ends at j with the peak already being seen. 1 denotes peak seen. 0 denotes not seen.

So,

for $j = 0 \to i-1$)

$DP(i, 1) \mathrel{+}= DP(j, 1)$

③ Transition

In Form 1, transition were the choices.

In Form 2, the transition is generally on what to extend.

So, whatever we are extending is what we loop on. So, we will loop on j to build i.

Let's say we want to find the val of $DP(i, 0)$ (no of bitonic sequences that ends at i and you have not seen any peak) So, the relation b/w i & j

$$DP(i, 0) = \sum_{\substack{j=1 \\ (arr[j] < arr[i])}}^{i-1} DP(j, 0) + 1$$

$$DP(i, 1) = \sum_{\substack{j=1 \\ (arr[j] > arr[i])}}^{i-1} DP(j, 0) + DP(j, 1)$$

	0	1	2	3	4
	1	3	2	5	1
DP(0)	{1}	{3} {1,3}	{2} {1,2}	{5} {2,5} {1,2,5} {3,5} {1,3,5} {3,1,5}	{1}
DP(1)			{3,2} {1,3,2}	X	{5,1} {2,5,1} {1,2,5,1} {3,5,1} {1,3,5,1} {2,1} {1,2,1} {3,2,1} {1,3,2,1} {3,1} {1,3,1}

$$Ans = \sum_{i=0}^{N-1} DP(i, 0) + DP(i, 1)$$

④ Time Complexity

$$\#\text{states} = DP(\underset{N}{i}, \underset{2}{0/1}) = 2N$$

Avg $\#\text{Transitions} = \frac{N}{2}$

$$\Rightarrow 2N\left(1 + \frac{N}{2}\right) = O(N^2)$$

Code for 1. A.

```
int n;
int arr[1001];

int rec(int i,int dec){
    // pruning
    // basecase
    if(i<0)return 0;
    // cache check
    // transition
    int ans = 0;
    if(dec==0){
        ans = 1;
        for(int j=0;j<i;j++){
            if(arr[j]<arr[i]){
                ans += rec(j,0);
            }
        }
    }else{
        for(int j=0;j<i;j++){
            if(arr[j]>arr[i]){
                ans += rec(j,0) + rec(j,1);
            }
        }
    }
    // save and return
    return ans;
}
void solve(){
    cin>>n;
    for(int i=0;i<n;i++)cin>>arr[i];

    int ans = 0;
    for(int i=0;i<n;i++){
        ans += rec(i,0) + rec(i,1);
    }
    cout<<ans<<endl;
}
```

$$DP(i,0) = \max_{Arr[j]<Arr[i]} (DP(j,0)) + 1 \quad / \quad 1$$

$$DP(i,1) = \max_{Arr[j]>Arr[i]} \begin{pmatrix} DP(j,0) \\ DP(j,i) \end{pmatrix} + 1$$

```
int rec(int i,int dec){
    // pruning
    // basecase
    if(i<0)return 0;
    // cache check
    // transition
    int ans = 0;
    if(dec==0){
        ans = 1;
        for(int j=0;j<i;j++){
            if(arr[j]<arr[i]){
                ans =max(ans, rec(j,0)+1);
            }
        }
    }else{
        for(int j=0;j<i;j++){
            if(arr[j]>arr[i]){
                ans = max({ans,rec(j,0)+1, rec(j,1)+1});
            }
        }
    }
    // save and return
    return ans;
}
void solve(){
    cin>>n;
    for(int i=0;i<n;i++)cin>>arr[i];

    int ans = 0;
    for(int i=0;i<n;i++){
        ans = max({ans,rec(i,0),rec(i,1)});
    }
    cout<<ans<<endl;
}
```

This is not a DP solution, its just using the normal recurrence relation

Solution 1. B. Longest Bitonic Sequence

① Step 1 : Form 2

② Since the restriction is same as prev. state hence the state will also remain same.

DP(i, dec) → longest bitonic seq ending at i

③ The transition will also be the same

DP solution :

```
int dp[1001][2];
int rec(int i,int dec){
    // pruning
    // basecase
    if(i<0)return 0;
    // cache check
    if(dp[i][dec]!=-1)return dp[i][dec];
    // transition
    int ans = 0;
    if(dec==0){
        ans = 1;
        for(int j=0;j<i;j++){
            if(arr[j]<arr[i]){
                ans =max(ans, rec(j,0)+1);
            }
        }
    }else{
        for(int j=0;j<i;j++){
            if(arr[j]>arr[i]){
                ans = max({ans, rec(j,0)+1, rec(j,1)+1});
            }
        }
    }
    // save and return
    return dp[i][dec] = ans;
}
```

```
void solve(){
    cin>>n;
    for(int i=0;i<n;i++)cin>>arr[i];
    memset(dp,-1,sizeof(dp));

    int ans = 0;
    for(int i=0;i<n;i++){
        ans = max({ans,rec(i,0),rec(i,1)});
    }
    cout<<ans<<endl;
}
```

```
void solve(){
    cin>>s>>t;
    n = s.length(), m = t.length();

    int ans = 0;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            ans = max(ans, rec(i,j));
        }
    }

    cout<<ans<<endl;
}
```

* Form 3

talks about multisequence

Ques 2. Given two strings, DNA A and DNA B, find longest common substring.

Brute force :

$O(N^3)$

```
for(i=0; i<N ; i++)  //DNA A
    for(j=0; j<M ; j++) //DNA B
        ans = max(ans, longest(i,j))
```

longest(i,j) ← loop

Have to optimise on the fxn longest(i,j)

longest(i,j) = 1 + longest(i,j)

(A[i] == B[i]) & 0 $\Rightarrow O(N^2)$

```
int n,m;
string s,t;

int dp[2002][2002];
int rec(int i,int j){
    if(i==n||j==m)return 0;

    if(dp[i][j]!=-1)return dp[i][j];

    int ans = 0;
    if(s[i]==t[j]) ans = 1+rec(i+1,j+1);

    return dp[i][j] = ans;
}
```