DP Form 2 & 3 Deep Dive

* ~~Form 2~~ Form 2

deal with subsequences

$DP(i, \ldots) \rightarrow$ Best _ ending at i

↑ restrictions

[- - - - $\overset{i}{\ominus}$
↳ best you can build ending at index i.

Q1. (a) Find the no. of Bitonic subsequences of an array. Bitonic means something that strictly increases & then decreases.

Exa: 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

We have to find total such no of bitonic sequences.

(b) Find the longest bitonic sequence

(c) Find the count of no. of longest bitonic ~~subsequence~~ that can be made.

Sol A.

Step 1 : Form 2

This problem deals with subsequence

Step2 : The DP Func^n needs to track restrictions

→ Peak : It is important to know if we have surpassed the peak or are yet to reach the peak. This determines the nature of the array extension.

- Let j be the last element and i be the current element.
- Not seen peak (increasing) : i ($>j$)
  seen (decreasing) : i ($<j$)

$DP(i, \text{seen\_peak}) \rightarrow$ # of bitonic subsequence that ends at i.

→ Can we extend this solution or not that ends at i, it only depends on j.

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

so, for $j = 0 \rightarrow i-1$

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

step 3 : Transition

Form 1 : Transition were the choices.

Form 2 : Transition is generally on what to extend. so, we loop on a previous index j to build the sequence ending at i.

let say we want to find value of DP(i,0) (no. of bitonic sequences that ends at i and you have not ~~seen~~ seen any peak). so, the relation b/w i & j.

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

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

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

Step 4 : Time complexity

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

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

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

```
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;
}
```

Sol. 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.

$DP(i, dec) \rightarrow$ longest bitonic sequence ending at i

3 The transition will also be the same

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

$$DP(i,1) = \max_{(arr[j] > arr[i])} \cancel{(DP(j,0} \left(\begin{matrix} DP(j,0) \\ DP(j,i) \end{matrix} + 1 \right)$$

```
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;
}
```

```cpp
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;
}
```

```
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;
}
```

Q2. Given two strings, DNA A and DNA B, find longest common substring.

Brute Force :

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

$\underbrace{\text{longest}(i,j)}_{\text{loop}}$

$$\text{longest}(i,j) = \begin{cases} 1 + \text{longest}(i,j) & (A[i] == B[i]) \\ 0 & \end{cases} \Rightarrow O(N^2)$$

```
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;
}
```