Bhavay Goyal

Convert any recursive dynamic programming code to iterative code?

NEXT ➡

Bhavay Goyal

- They are actually **same (same but not equal)**
- The only difference is the states and any loop inside the recursive code gets converted into nested loops
- One key thing to take care is the order in which we are running those loops and the base case
- By order I mean if dp[i][j] **depends upon dp[i+x][j+y]**, than dp[i+x][j+y] must be calculated before dp[i][j]

NEXT ➡

Bhavay Goyal

So follow these steps -:

- Try to identify the order in which we are going to run the loops
- Identify the base case
- Copy-paste the recursive code, and woo-whoo we're done
- (Swipe to see examples)

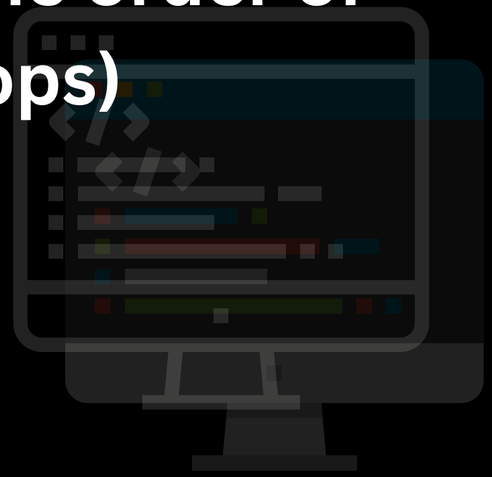
NEXT ➡

Bhavay Goyal

Factorial of a number (recursive code)

```
int fact(int i) {
    if (i == 0) return 1;
    if (dp[i] != -1) return dp[i];
    return dp[i] = fact(i-1)*i;
}
cout << fact(n) << endl;
```

Notice that here to calculate i, we need i-1 pre-calculated (to identify the order of loops)



NEXT ➡

Bhavay Goyal

Factorial of a number (iterative code)

```
dp[0] = 1;
for (int i = 1; i <= n; i++) {
    dp[i] = dp[i-1]*i;
}
cout << dp[n] << endl;
```

NEXT ➡

Bhavay Goyal

Longest common subsequence of 2 strings (recursive code)

```
int lcs(int i, int j) {
    if (i == a.size() || j == b.size()) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    int ans = max(lcs(i+1, j), lcs(i, j+1));
    if (a[i] == b[j]) ans = max(ans, 1+lcs(i+1, j+1));
    return dp[i][j] = ans;
}
cout << lcs(0, 0) << endl;
```

Notice that here, to calculate (i, j) we need (i+1, j), (i, j+1), (i+1, j+1) precalculated so how should we run the loops? (What about running both i and j from a.size() to 0 and b.size() to 0?)

NEXT ➡

Bhavay Goyal

Longest common subsequence of 2 strings (iterative code)

```
for (int i = a.size(); i >= 0; i--) {
    for (int j = b.size(); j >= 0; j--) {
        if (i == a.size() || j == b.size()) { dp[i][j] = 0; continue; }
        int ans = max(dp[i+1][j], dp[i][j+1]);
        if (a[i] == b[j]) ans = max(ans, 1+dp[i+1][j+1]);
        dp[i][j] = ans;
    }
}
cout << dp[0][0] << endl;
```

Everything other than the
order of loops is same right?

NEXT ➡

Bhavay Goyal

Count of pallindromic substrings in a string?

Basic idea -:

- Precompute for every sub-string if it is a palindrome or not (This can be done in N^2)
- Let getCount(i, j) store the number of palindromic substrings in range (i, j)
- Now we can get count of palindromic substrings in range (i+1, j), (i, j-1) from our recursive function and add them to get our answer (also add 1 if (i, j) is a palindrome), but note that here (i+1, j-1) will be double counted so let's subtract it
- Base case -> if (i == j) this is always a palindrome so return 1
- -> if (i > j) substring doesn't exist so return 0
- Overall time complexity is N^2

NEXT ➡

Bhavay Goyal

Count number of pallindromic substrings in a string (recursive code)

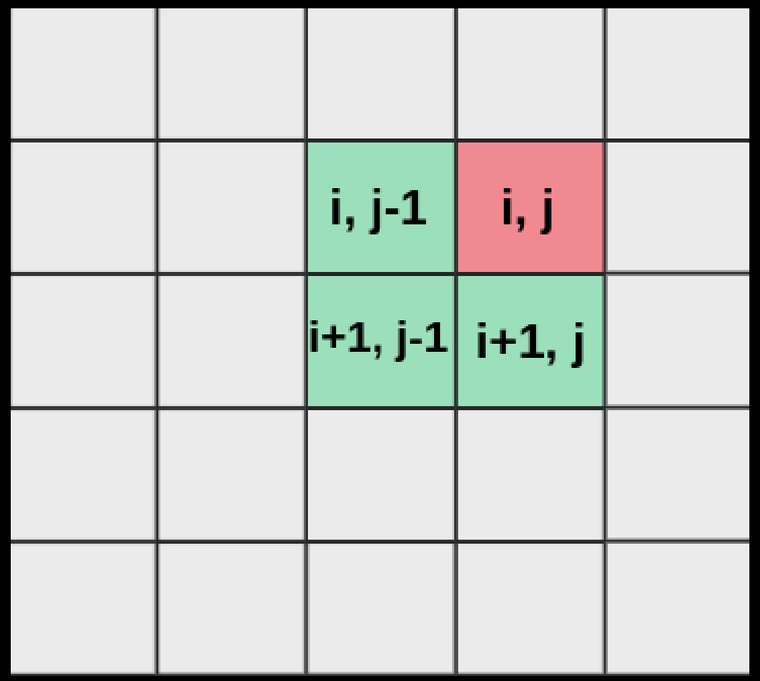
```
int getCount(int i, int j) {
    if (i == j) return 1;
    if (i > j) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    int x = getCount(i+1, j);
    int y = getCount(i, j-1);
    int repetition = getCount(i+1, j-1);
    return dp[i][j] = isPallindrome[i][j] + x + y - repetition;
}

cout << getCount(0, n-1) << endl;
```

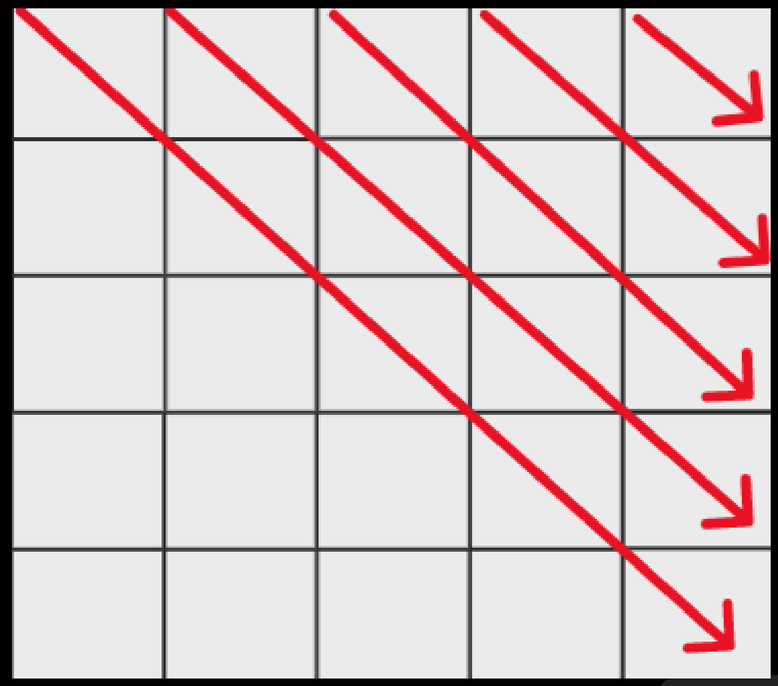
Here isPallindrome is a pre-computed 2D array which stores if substring of s from i to j is a pallindrome or not

NEXT ➡

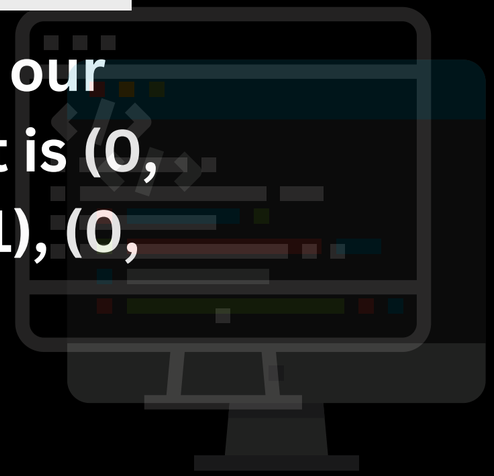
Bhavay Goyal



Now, here notice that to calculate (i, j) we need (i, j-1), (i+1, j-1), (i+1, j) precalculated beforehand



So we will be running our loops in this order, that is (0, 0), (1, 1), (2, 2), (n-1, n-1), (0, 1) (1, 2), ...



NEXT ➡

Bhavay Goyal

Count number of pallindromic substrings in a string (iterative code)

```
for (int g = 0; g < n; g++) {
    for (int i = 0, j = g; j < n; i++, j++) {
        if (i == j) { dp[i][j] = 1; continue; }
        int x = dp[i+1][j]; int y = dp[i][j-1];
        int repetition = dp[i+1][j-1];
        dp[i][j] = isPallindrome[i][j] + x + y - repetition;
    }
}
cout << dp[0][n-1] << endl;
```

- Try comparing this code with the previous recursive code, they're same (same but not equal) right?

- Here I didn't add if (i > j) condition cuz that was not needed because of the order in which we're running our loops

NEXT ➡

Bhavay Goyal

Bonus -:

- These kinds of questions in which we store the answer for a range in our dp are called range-dp questions.
- This technique/method of diagonally filling our dp table is called gap-method or gap-strategy.