Dynamic Programming

(Easy to Medium)

Prerequisites: Recursion + Time Complexity Calculation

Why DP? Let us understand that from a problem

Problem: Given an N * M grid (N rows and M columns) with numbers written in each cell. Find the minimum possible sum of values on the path from (0, 0) to (N - 1, M - 1)

$$\begin{tikzcd}
1 \arrow[r] & 5 \arrow[d] & 8 \\
6 & 2 \arrow[d] & 7 \\
9 & 3 \arrow[r] & 4
\end{tikzcd}$$

Ans = 1 + 5 + 2 + 3 + 4 = 15 (This is the minimum possible)

Naive Solution: Recursion

f(i, j) = sum of values on the best path from (i, j) to (n-1, m-1)

$f(i, j) = \min(f(i + 1, j), f(i, j + 1)) + a(i, j)$

Time Complexity: $O(C(n + m, n))$

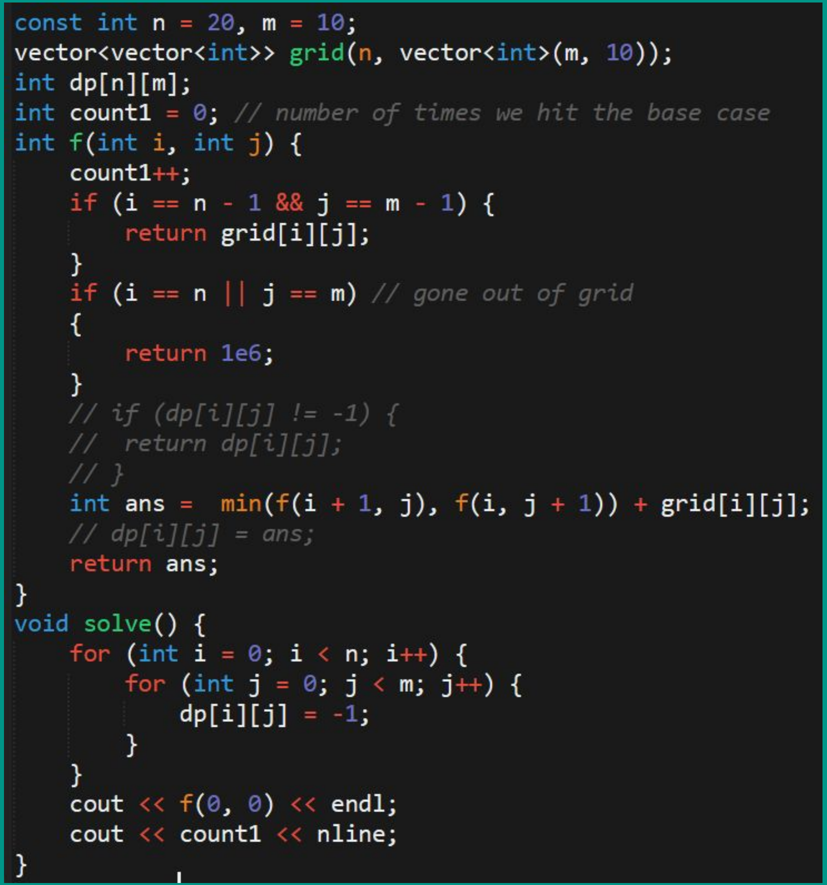
As there are $C(n + m, n)$ possible paths from (0, 0) to (n - 1, m - 1)

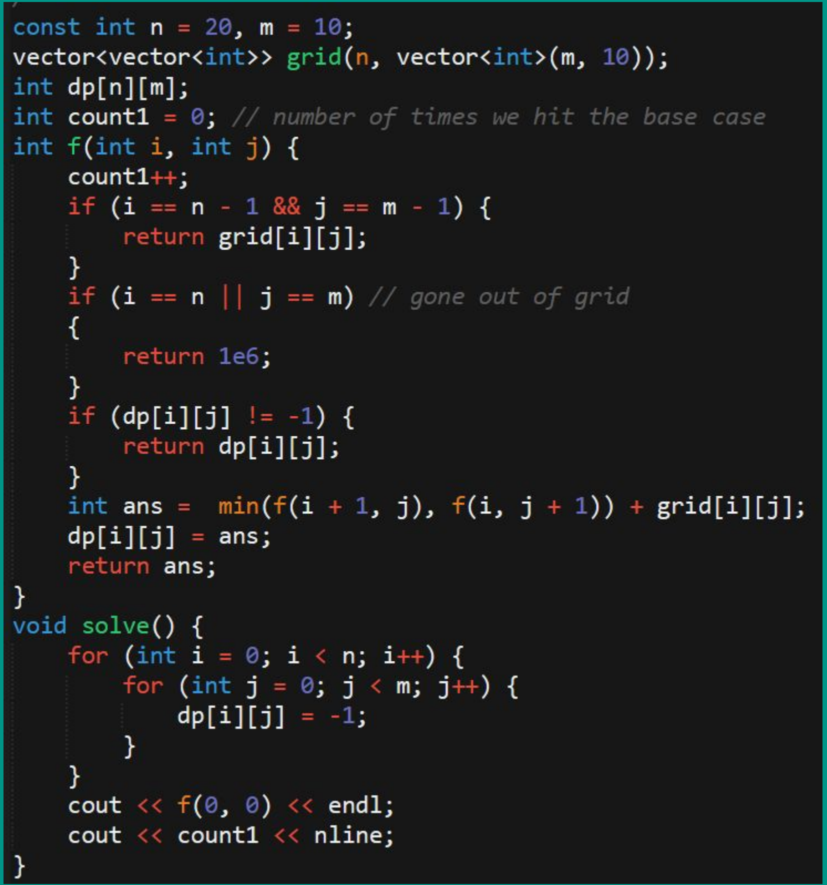
Efficient Solution: DP

Why not store all the values of f(i, j) for all possible (i, j) when we have calculated the answer for that. Next time, when we need f(i, j) it can be returned in $O(1)$

Time Complexity: $O(n*m)$

Each state requires constant time to get evaluated and there are m*n states


```
const int n = 20, m = 10;
vector<vector<int>> grid(n, vector<int>(m, 10));
int dp[n][m];
int count1 = 0; // number of times we hit the base case
int f(int i, int j) {
    count1++;
    if (i == n - 1 && j == m - 1) {
        return grid[i][j];
    }
    if (i == n || j == m) // gone out of grid
    {
        return 1e6;
    }
    // if (dp[i][j] != -1) {
    //  return dp[i][j];
    // }
    int ans =  min(f(i + 1, j), f(i, j + 1)) + grid[i][j];
    // dp[i][j] = ans;
    return ans;
}
void solve() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            dp[i][j] = -1;
        }
    }
    cout << f(0, 0) << endl;
    cout << count1 << nline;
}
```


```
const int n = 20, m = 10;
vector<vector<int>> grid(n, vector<int>(m, 10));
int dp[n][m];
int count1 = 0; // number of times we hit the base case
int f(int i, int j) {
    count1++;
    if (i == n - 1 && j == m - 1) {
        return grid[i][j];
    }
    if (i == n || j == m) // gone out of grid
    {
        return 1e6;
    }
    if (dp[i][j] != -1) {
        return dp[i][j];
    }
    int ans =  min(f(i + 1, j), f(i, j + 1)) + grid[i][j];
    dp[i][j] = ans;
    return ans;
}
void solve() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            dp[i][j] = -1;
        }
    }
    cout << f(0, 0) << endl;
    cout << count1 << nline;
}
```

Without DP: Count = 46276229

With DP: Count = 399

States and Transitions

State: A subproblem that we want to solve. The subproblem may be complex or easy to solve but the final aim is to solve the final problem which may be defined by a relation between the smaller subproblems. In the previous problem a state is simply (i, j) and dp[i][j] tells us the path sum of the optimal path from (i, j) to (n - 1,m - 1)

Transition: Calculating the answer for a state (subproblem) by using the answers of other smaller states (subproblems). In the previous problem, a transition is nothing but saying that dp[i][j] = min(dp[i+1][j], dp[i][j+1]) + a[i][j]

How to identify a DP problem?

Repeating subtasks: If I have the answer of state, then why should I calculate it again and waste time

Pro Tips for contests:

- Look for small constraints in the problem. (Most probably it would be dp and not greedy)
- Identify states and transition time for each state.
- Calculate time complexity as (number of states * transition time for each state).
- If this number fits into your Time limit (Great), if not, try to see if you can skip some states and still get the right answer.
- Try to reduce the transition time by using some Data Structure if transition time is the bottleneck
- Never try to over optimize. If your current states and transition time fit into your Time Limit, just code it and do not optimize it further.

Let us solve some problems now

(Easy to Medium)

Array Description (CSES) Link

There is an array ‘a’ of length ‘n’ with elements in the range from 1 to ‘m’ with the condition that adjacent elements differ by no more than 1. Due to an accident, some of the array elements were lost and replaced by 0. Find the number of arrays that could have been the original array modulo $10^9 + 7$

$$1 <= n <= 10^5$$

$$1 <= m <= 100$$

Example:

n = 3, m = 5

Given Array = {2, 0, 2}

Possible arrays that satisfy this array = 3 [{2, 1, 2}, {2, 2, 2}, {2, 3, 2}]

Two Sets 2 (CSES) Link

Given an permutation ‘a’ of ‘n’ length, find the number of ways to split it into 2 subsets of equal sum. Print the number of ways modulo $10^9 + 7$

$1 <= n <= 100$

Examples:

n = 7 Answer = 4

Possible partitions:

{1, 3, 4, 6}, {2, 5, 7}

{1, 2, 5, 6}, {3, 4, 7}

{1, 2, 4, 7}, {3, 5, 6}

{1, 6, 7}, {2, 3, 4, 5}

Maximum Length Consecutive Subsequence Link

A consecutive subsequence is defined as [x, x + 1, x + 2, …]. Find the maximum possible length subsequence in the given array ‘a’.

$$1 <= a[i] <= 10^9$$

$$1 <= |a| <= 10^5$$

Examples:

6, 7, 8, 3, 4, 5, 9, 10, 11 - Answer = 6 [6, 7, 8, 9, 10, 11]

1, 9, 10, 11, 2, 3, 12 - Answer = 4 [9, 10, 11, 12]

Alice and Bob play a game of Primes

The game starts with a number ‘n’ and Alice and Bob make moves alternatively with Alice going first. If the current number is ‘k’, the player, whose turn it is, can convert k to {k-1, k-2, k-3}. If the current number is a Prime, the player cannot make a move and thus loses. Determine who will win if they both play optimally.

$$2 <= n <= 10^6$$

Examples:

N = 6 - Alice wins as she can subtract 1 from 6 and convert it 5 (a Prime)

N = 27 - No, matter what Alice converts 27 to {26, 25, 24}, Bob can convert the next number to 23 (a Prime)and win.

Strange Beauty Link

Given an array ‘a’ of ‘n’ integers, find the maximum length subsequence from this array such that for every 2 elements in the chosen subsequence, the following condition holds:

Either a_i divides a_j

Or a_j divides a_i

$$1 <= n <= 10^5$$

$$1 <= A[i] <= 10^5$$

Example:

7, 9, 3, 14, 63 - Answer = 3 [9, 3, 63]

1, 2, 3, 5, 7 - Answer = 2 [1, 2] ([1, 3], [1, 5], [1, 7] are also valid)

DP with Bitmasking (Hard Topic)

Nothing special, here you just express the state as a bitmask rather than a single integer. A great way to reduce the time complexity from n! to 2^n

Let us look at a problem from leetcode to understand the importance of this concept. Link

More about it in some other class :)

DP on Trees (Only useful if you know Trees)

If you know about Trees, you might be already using it unknowingly. Again, nothing special about this. It is just that you find the answer to your current state using states from the children of the current node or the parent of the current node.

Problem: Given a tree, find the subtree sum of each node in the tree.

More about it in some other class :)