2019-10-11 More Advanced DP

Patrick Zhang

October 2019

1 Review (Knapsack DP 2)

Last week, we discussed the following problem:
You have a bag (a "knapsack"), and three different types of coins with the values of 1, 3, and 5. How many ways are there to put the coins in the bag such that the coins sum to N ($0 \leq N \leq 10^5$)? Order doesn't matter, so adding a 1 coin then a 3 coin is the same as adding a 3 coin then a 1 coin.
We made a matrix of size [N+1][3] the stored the two following states: the current sum of the coins, and the value of the max coin.
Set dp[1][0], dp[3][1], and dp[5][2] to 1 as initial states.
We transitioned between states in the following way:
Loop k from $0- > N$ and j from $0- > 2$
Add a 1 coin if $j == 0$ (meaning the max coin that has been added has a value of 1): dp[k+1][0] += dp[k][j]
Add a 3 coin if $j <= 1$: dp[k+3][1] += dp[k][j]
Add a 5 coin if $j <= 2$: dp[k+5][2] += dp[k][j]
The answer is dp[N][0] + dp[N][1] + dp[N][2]. The complexity is $O(N)$.

2 The Method

1. **Make Big Array:** Identify which states are needed to solve the problem.
2. **Loop Through Array:** Figure out how to transition between the states.
3. **Get Answer**

3 Flower Problem

3.1 Problem

You have red flowers (R) and white flowers (W). How many ways are there to arrange N flowers ($1 <= N <= 10000$) such that there are never more than $M(1 <= M <= 1000)$ flowers of the same color in a row?

3.2 Solution

This solution will use only 2 states: The length of the arrangement of flowers and the last flower that was added. Thus, we will have a matrix, called dp, of size $[N][2]$. 0 correlates to a red flower being added last and 1 correlates to a white flower being added last.
Our initial states will be dp[0][0] = 1 and dp[0][1] = 1.
Loop k from 0 to N and j from 0 to 1. To transition between states, we will simulate adding from 1 to M flowers of the opposite color (if $j == 0$ add white flowers, if $j == 1$ add red flowers).
Looping through the matrix is O(N) (note that because second dimension is constant at 2, it doesn't contribute to the complexity). Transitioning between states is O(M). Thus, the complexity of this solution is $\mathbf{O}(NM)$. In general your overall complexity will be (complexity of looping through array) $\cdot$ (complexity of transitioning between states).
The answer will be dp[N][0] + dp[N][1].

3.3 Walkthrough

Let $N = 7$ and $M = 3$.
We will end up with a matrix that looks like this:

	0	1	2	3	4	5	6	7
0 (R)	1	1	2	4	7	13	24	44
1 (W)	1	1	2	4	7	13	24	44

Our answer is thus $44 + 44 = \boxed{88}$.

3.4 Extension

In addition to N and M, you are told that certain indices must be a certain flower. In this case, while looping from 1 to M, if you see that you are adding a different flower than what the restriction allows, immediately terminate the innermost loop.
For example, given $N = 7, M = 3$, and you know that the 3rd index must be a red flower, you will end up with the following matrix.

	0	1	2	3	4	5	6	7
0 (R)	1	1	2	4	3	6	11	24
1 (W)	1	1	2	0	4	7	13	20

Our answer is $24 + 20 + \boxed{44}$.

4 Flower Problem 2

4.1 Problem

Same as the first Flower Problem, only you have exactly N red flowers and N white flowers and need to arrange all of them in a line of length $2N$ and meet the same constraints as before.
Find $O(N^2M)$ solution.

4.2 Solution

This problem is more complicated because we need to store the number of red flowers and white flowers that we have already added. However, storing those values along with the index would give us $O(N^3)$ already, so it is too slow. Thus, we need to be clever in how we choose our states. We notice that given the number of red flowers and white flowers that have already been added, we can find the total number of flowers that has been added (using addition), and thus what index we are on.
We have a dp matrix of size $[N + 1][N + 1][2]$, which stores the number of red flowers, white flowers, and last flower that was added.
To help you visualize this solution better, this is how our matrix would represent an arrangement of WWWR-RWWRRR:

R	R	R	W			5
			W			4
			R	R	W	3
					W	2 W
					W	1
						0
5	4	3	2	1	0	
			R			

We can deal with restrictions from the first problem in a similar fashion.

4.3 Bonus

Find a solution with a lower complexity (no restrictions). O(NM^2) is considered lower. I'm not sure if this is possible. Even more bonus points if you solution can handle restrictions.

5 USACO December 2018 Gold Problem: Teamwork

5.1 Problem

http://www.usaco.org/index.php?page=viewproblem2&cpid=863
Farmer John's N cows ($1 \leq N \leq 10^4$) are all standing in a row, conveniently numbered $1 \ldots N$ in order. Cow i has skill level s_i at wrapping presents. These skill levels might vary quite a bit, so FJ decides to combine his cows into teams. A team can consist of any consecutive set of up to K cows ($1 \leq K \leq 10^3$), and no cow can be part of more than one team. Since cows learn from each-other, the skill level of each cow on a team can be replaced by the skill level of the most-skilled cow on that team.
Please help FJ determine the highest possible sum of skill levels he can achieve by optimally forming teams.

5.2 Solution

This problem requires an O(NK) solution. We will have a dp array of size $[N+1]$ and O(K) transitions.
Loop k from 1 to N and j from 1 to K. j represents the size of the team that you are adding. During the j loop, store and update variable called max. The transition is dp[k+j] = Math.max(dp[k+j],dp[k-1] + max*(j+1));
The answer is dp[N].

5.3 Code

```
BufferedReader f = new BufferedReader(new FileReader("teamwork.in"));
PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter("teamwork.out")));

StringTokenizer st = new StringTokenizer(f.readLine());

int n = Integer.parseInt(st.nextToken());
int m = Integer.parseInt(st.nextToken());

long[] array = new long[n+1];

for(int k = 1; k <= n; k++){
 array[k] = Long.parseLong(f.readLine());
}

long[] dp = new long[n+1];

for(int k = 1; k <= n; k++){
 long max = 0L;
 for(int j = 0; j < m; j++){
    if(k+j > n) continue;
    max = Math.max(max,array[k+j]);
    dp[k+j] = Math.max(dp[k+j],dp[k-1] + max*(j+1));
 }
}

out.println(dp[n]);

out.close();
```

6 USACO US Open 2019 Gold Probem: Snakes

6.1 Problem

`http://www.usaco.org/index.php?page=viewproblem2&cpid=945`
You have an array of integers of length $N(1 \leq N \leq 400$, which represent the number of snakes at the index. You want to capture all the snakes (in order). You have a net to capture the snakes. In order to catch the snakes at a certain index, the size of your net must be greater than the number of snakes. Every time you capture snakes, the waste is (size of net) - (number of snakes captured). You can decide the size of the net at the beginning and can change the size of the net $K(1 \leq k < N)$ times. What is the minimum waste after capturing all of the snakes?