Dynamic programming

February 26, 2016

Dynamic programming

It’s a very vast topic, with hundreds of problems on Codeforces and other websites.

The only way to become good at these types of problems is to solve as many as possible.

Definition

The definition of dynamic programming is that we solve a complex problem by splitting it into many smaller subproblems, each of which is solved exactly once then stored.

The subproblems we want to solve must have two main properties:

- They don’t lose information that we might need to find the best solution
- It’s *somewhat* easy to go from one subproblem to the other

(This definition is vague, so let’s look at some examples)

First problem

The first problem is actually one that we've done before: Maximum sequence sum.

We want to find the maximum sequence in an array.

Let's split into the following sub problem, best[i] = the sequence with the best sum that ends in i. The final solution will be the maximum of all of these.

What will best[i] be equal to? Well either it has length 1, so it's v[i], or we continue the best sequence that ends previously, so best[i-1]. This gives us:

best[i] = max(best[i-1] + v[i], v[i])

Maximum increasing sequence

You are given an array v, what's the maximum strictly increasing sequence?

(not necessarily continuous)

Example: [1,-1,7,4,5,3,7].

Solution

Let’s use the following subproblem, best[i], the longest possible sequence ending in the i position.

So what’s best[i] equal to ? Well it can either be alone in which case it’s 1, or for all previews positions with smaller values, we can continue that sequence.

best[i] = max(best[j] + 1) for j (1..i-1) and v[i] > v[j].

This will have a O(N^2) complexity, but it is worth mentioning that it can be solved in O(N*logN) with some more complex data structures.

Number of increasing sequences of length K

You are given an array v, how many strictly increasing sequences of length K are there? (not necessarily continuous)

For [1,-1,7,4,5,3,7] and K = 2, we have 17 sequences. (any pair (x,y) with x < y and x appears before y)

Solution

Let’s use the following subproblem, num[i][j], the number of sequences of length j that end in i. Our final solution will be sum(num[i][K]) for all i.

What will the state change look like ? Well, we fix the positions we’re counting for, so i and j. Now to get a sequence of length j that ends in i, we need a sequence of length j-1 that ends in l, with l < i, and v[l] < v[i] and we’ll continue it.

So num[i][j] = sum(num[l][j-1]) for all l < i and v[k] < v[i].

Which gives us a final complexity of O(N^2 * K)

Knapsack

You have N coins, each with its value v[i]. You want to find out if you can select a subset of those coins, so that their sum will be equal to S.

Solution

Let’s use the following subproblem, possible[i][j], if it’s possible to have the sum j by only using the first i coins. The final solution will be possible[N][S].

What will possible[i][j] equal to? We can extend the solution from step i-1, by either using or not using coin i, so we extend either sum j or sum j - v[i].

possible[i][j] = possible[i-1][j] OR possible[i-1][j-v[i]], for every j in 0...S

South East

You are given an N*M matrix with numbers in each cell, and you want to find the best path from [1,1] to [N,M] (path with maximum sum), by only moving south and east (from [i , j] to [i+1 , j] or [i , j+1]).

Solution

Let’s solve the following subproblem, best[i][j], the best path that starts in 1,1 and ends in i,j. The final solution will be best[N][M].

Here at i,j we have 2 options. either we continue the path to the north of us, or to the west of us, so we have best[i][j] = a[i][j] + max(best[i-1][j], best[i][j-1])

Coins on a line

You are given N coins on a line, each with value v[i]. There are two players, Alice and Bob that are playing a game with these coins. At each possible turn one of them will take a coin from one of the sides of the vector. This process is done until there are no more coins. If both play optimally and Alice starts, how much money will they have at the end?

Solution

Lets use the following subproblem best[i][j], how many many coins we can get if the remaining array is [i...j] and we're moving first.

What will best[i][j] be equal to ?

Well we can either take the i coin, which will give us v[i], and from the other player will be left in best[i+1][j]. This will give us a value of v[i] + sum(i+1 … j) - best[i+1][j]. Analogous for v[j].

This gives us a final recurrence of best[i][j] = max(v[i] + sum(i+1,j) - best[i+1][j], v[j] + sum(i,j-1) - best[i][j-1])

Matrix multiplication

You have N matrices you want to multiply them in the following order:

M1 * M2 * … * Mn (it is guaranteed that the sizes are compatible).

Since matrix multiplication is associative, any way you put parentheses in the multiplication, you'll still get the same result, but some multiplications take more computational time than others. Find the minimum possible total computation time.

Solution

The computational time required to multiply a matrix of size A*B and a matrix of size B*C is A*B*C.

Let’s make the following subproblem: best[i][j] is the minimum cost of multiplying matrices i, … j. Let k be the index corresponding to the last multiplication we made, meaning that we solve the subproblem (i,k) then the subproblem (k+1, j), then add a computational time of left[i] * right[k] * right[j]. We choose the k that minimizes best[i][k] + best[k+1][j] + the cost above.

The solution is given by best[1][N]. We can’t simply iterate through i and j like we did in previous problems, because [2,3] needs to be solved before [1,4]. In order to achieve this, we can solve the intervals of length 1 first, then the intervals of length 2 etc.