Dynamic Programming
(Programming Club 2)

Tomasz Kosciuszko

29.01.2018

1 Introduction

Dynamic Programming is a very powerful algorithmic technique. Often problems which appear difficult and unsolvable in a reasonable complexity can be easily solved using **Dynamic Programming**.
The concept is closely connected to **recusion**, **induction** and the **divide and conquer** principle. We try to split our problem into smaller, trivial subproblems. They can be easily solved and then combined into the full solution.

2 Problem example - "Fibonacci numbers"

Everybody knows what the Fibonacci sequence is:

$$1, 1, 2, 3, 5, 8, 13, 21, 34....$$

If you have not heard about it, it is defined as

$$f_{i+2} = f_i + f_{i+1}$$

But how to compute it? Look what is happening when we try to compute f_6 using the definition:

$$f_6 = (f_4+f_5) = (f_2+f_3)+(f_3+f_4) = (f_0+f_1)+(f_1+f_2)+(f_1+f_2)+(f_2+f_3) = ...$$

and we see that the computation will take too much time even for medium-size inputs.

But look how much more effective we are if we proceed in the other way and remember the values by the way:

$$f_0 = 1$$

$$f_1 = 1$$

$$f_2 = 1 + 1 = 2$$

$$f_3 = 1 + 2 = 3$$

$$f_4 = 2 + 3 = 5$$

$$f_5 = 3 + 5 = 8$$

$$f_6 = 5 + 8 = 13$$

Have a look at the code below, that implements the computation of f_n using the **DP** approach (Python):

```
n = int(input())
fib = [1, 1]
for i in range(0, n - 1):
    fib.append(fib[i] + fib[i+1])
print(fib[n])
```

3 Problem example - "Flowers"

Let's have a look at this problem's statement:

`http://codeforces.com/problemset/problem/474/D`

The problem is basically asking the following question: How many strings of length n are there, that 'W' appears only in series divisible by k.

It is not easy to think about this problem, is it? At least if you have not had much experience with **DP**. But let's pretend that we know how to solve the problem and create the array $ans[]$ to store the numbers of strings for each n. Notice that it is obvious that:

$$ans[0] = ans[1] = ... = ans[k-1] = 1.$$

That is good enough for a start. Now let's think about $ans[k]$, we can either put k white flowers at the end and then deal with the remaining 0 slots. We can also put a red flower at the end and then deal with the remaining $k-1$ slots. But $ans[0]$ and $ans[k-1]$ have been already computed. Therefore:

$$ans[k] = ans[0] + ans[k-1]$$

With the same argument we can compute:

$$ans[i] = ans[i-k] + ans[i-1]$$

Providing that we know the answers for $0...(i-1)$. Let's now run a loop and get all the answers at once (C++):

```
const int N = 1000001, mod = 1000000007;
int ans[N], sumTo[N], t, k;
int main(){
    cin >> t >> k;
    for (int i = 0; i < k; i++)
        ans[i] = 1;
    for (int i = k; i < N; i++)
        ans[i] = (ans[i - 1] + ans [i - k]) \% mod;
    for (int i = 1; i < N; i++)
        sumTo[i] = (sumTo[i - 1] + ans[i]) \% mod;
    for (int i = 0; i < t; i++) {
        int a, b;
        cin >> a >> b;
        int res = (sumTo[b] - sumTo[a - 1] + mod) \% mod;
        cout << res << "\n";
    }
}
```

Notice that I am using one additional array *sumTo*[], in order to answer the queries quicker.
Now try to put the example code together and compile it. Or, write your own if you prefer. All popular programming languages are supported on codeforces! Once you are convinced it works submit it via codeforces to check if it passes all the tests.

4 Important remarks

- Why is the presented solution **DP**? Because we are reducing the problem to a smaller one recursively.
- Often when you implement your **DP**, you need to use an array to compute answers for smaller sub-problems. You need to decide in respect to what feature you are going to reduce the problem. Sometimes you will need more than one dimention, as with the Levenshtein Distance problem `http://en.wikipedia.org/wiki/Levenshtein_distance`.
- Is the solution correct? Usually it is quite straightforward to prove that your **DP** solution is correct, because you would proceed with an induction proof exactly in the same way as you compute values. You only need to pay attention that your induction step is correct.
- If you are not sure how to deal with **standard input and output** in your favourite language, have a look at some accepted solutions by other people: `http://codeforces.com/problemset/status`.

5 Practice problems

- Fence (easy):
  `http://codeforces.com/problemset/problem/363/B`
- Ilya and Queries (easy):
  `http://codeforces.com/problemset/problem/313/B`
- Hard Problem (medium):
  `http://codeforces.com/problemset/problem/706/C`
- K-Tree (medium):
  `http://codeforces.com/problemset/problem/431/C`
- Greg and Graph (hard):
  `http://codeforces.com/contest/295/problem/B`

6 Hints

- Fence: How can you quickly compute the sum of consecutive numbers? Have a look at the solution of the example problem!
- Ilya and Queries: If you knew the number of hashes and dots in each interval, what the answer would be then?
- Hard Problem: Try to use a two-dimensional DP array. The dimensions will be number of words, and whether the last word is reverted.
- K-Tree: Try to use the array $dp[n][d][2]$.
- Greg and Graph: Try to generate the answer starting from the end.