Solving Linear Recurrence for Programming Contest – Examples

August 24, 2011 · by Ashar Fuadi · 7 Comments

After grasping the basic method of solving linear recurrence, let's apply it in real problems.

SPOJ Recursive Sequence

This is a classical problem that is directly solvable using the basic method I explained in the previous post. It is expected that you solve this problem and master the basic method before solving the next problems.

SPOJ Recursive Sequence (Version II)

A slightly harder version of the previous problem. This time, instead of determining a_n, we have to determine $a_m + a_{m+1} + \cdots + a_n$.

Let's simplify the problem first, using a well-known technique called partial sum. Define a series S, where $S_i = a_1 + a_2 + \cdots + a_i$, with $S_0 = 0$. Then, the result we are looking for is exactly $S_n - S_{m-1}$ (prove that). So, instead of determining (n – m + 1) terms of sequence a, it is sufficient to determine only two terms of series S.

How to compute S? The crucial observation is that $S_i = S_{i-1} + a_i$, i.e., S itself is a linear recurrence! So, we can adopt the matrix equation of a to incorporate S. The final matrix equation of would look like this:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & c_k & c_{k-1} & c_{k-2} & \cdots & c_1 \end{bmatrix} \begin{bmatrix} S_{i-1} \\ a_i \\ a_{i+1} \\ a_{i+2} \\ \vdots \\ a_{i+k-1} \end{bmatrix} = \begin{bmatrix} S_i \\ a_{i+1} \\ a_{i+2} \\ a_{i+3} \\ \vdots \\ a_{i+k} \end{bmatrix},$$

with initial values:

$$\begin{bmatrix} S_0 = 0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_k \end{bmatrix}.$$

After finding the matrix equation, S_n and S_{m-1} can be computed easily: S_n is the first row of $T^n F_1$, and S_{m-1} is the first row of $T^{m-1} F_1$.

You may also want to solve SPOJ Fibonacci Sum. It uses exactly the same idea.

SPOJ Summing Sums

As an example, assume that N = 4. Let

- A_i = the current number of cow i,
- B_i = the number of cow i in the next iteration.

From the story, we know that the next number of a cow is the sum of all other cows' current numbers. In other words,

$B_1 = A_2 + A_3 + A_4$.

$B_2 = A_1 + A_3 + A_4$.

$B_3 = A_1 + A_2 + A_4$.

$B_4 = A_1 + A_2 + A_3$.

We have N = 4 recurrence relation here, and all are related. This is essentially not harder than solving with basic method. To show this, rewrite the equations in matrix form:

$$\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix},$$

If we define $A_{i,j}$ as the number of cow i in the j-th iteration, with $A_{i,0} = C_i$ (the starting number of cow i), then the above equation can be rewritten as:

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{1,j} \\ A_{2,j} \\ A_{3,j} \\ A_{4,j} \end{bmatrix} = \begin{bmatrix} A_{1,j+1} \\ A_{2,j+1} \\ A_{3,j+1} \\ A_{4,j+1} \end{bmatrix}.$$

The equation looks like what we did with the basic method, right? So, the problem is solved. Let X = the number of iterations the process is performed. The number of all cows can be found by computing:

$$\begin{bmatrix} A_{1,X} \\ A_{2,X} \\ A_{3,X} \\ A_{4,X} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}^X \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix}.$$

Also solve SPOJ Life Game.

SPOJ Number of Quite Different Words

Let's try to attack a problem where the recurrence relation is not so obvious.

In this problem, we would like to count the number of strings of length N that are quite different from W, i.e. have no common subsequence of length greater than 1 with W. The strings only use the first C letters of the alphabet. Consider one such string. The string will not have any subsequence of length two, or else it will not be quite different (as 2 > 1). This condition is sufficient. So, to construct a string that is quite different, we can place the letters one-by-one from left to right, each time ensuring that the current letter doesn't form a pair with any previous letter that occurs as a subsequence of length two in W.

For example, let W = "ABC". The forbidden pairs for this case are AB, AC, and BC.

We will first solve this problem with dynamic programming. Let dp[i][mask] be the number of strings of length **i** that are quite different with W, and all letters that occur in the strings are contained in **mask**. The set of "used" letters are represented with a bitmask that ranges from 0 to 2^K-1.

The base cases are, obviously, dp[0][0] = 1 and dp[0][x] = 0 for **x** greater than 0.

Now we will define the recurrence. How many strings of length **i** that are quite different are there, and only use the letters in **mask**? The strings are built from strings of length (**i**-1) that are quite different, using only letters in **prev** (the number of such strings is dp[i-1][prev]), plus a new character **a**, where **prev** + {**a**} = **mask**. In addition, **a** must not form a "forbidden pair" with any letter in **prev**.

In other words,

$$dp[i][mask] = \sum_{prev=0}^{2^K-1} a_{prev} \times dp[i-1][prev],$$

where $\mathbf{a}_{\mathbf{prev}}$ is the number of "valid" letters, i.e. number of letters **a** such that **a** does not form a forbidden pair with any letter in **prev**, and **prev** + {**a**} = **mask**.

The recurrence above is linear, so we can transform it into the method that we have learned before, exactly the same way as SPOJ Summing Sums. We have to determine matrix T such that

$$T \begin{bmatrix} dp[i][j] \\ dp[i][j+1] \\ \vdots \\ dp[i][2^K-1] \end{bmatrix} = \begin{bmatrix} dp[i+1][j] \\ dp[i+1][j+1] \\ \vdots \\ dp[i+1][2^K-1] \end{bmatrix},$$

and proceed the solution as usual. The answer is then $\sum_{mask=0}^{2^K-1} dp[N][mask]$.

–

Have another problems that can be solved using linear recurrence? Please, share with us in the comment section.

Follow Like 4 g+1 0

- Stu

Tweet 0

Related posts:

- Solving Linear Recurrence for Programming Contest
- CodeSprint 2 Interview Street Problems Analysis
- Vim as a Programming Editor in Ubuntu

filed under: tips · tagged: c++, linear recurrence

About Ashar Fuadi
Ashar Fuadi is a competitive programmer from University of Indonesia. He loves to code, especially for TopCoder SRM, Codeforces, and ICPC.
Follow Ashar on Google+ and Twitter.

Comments

3haboys says:
August 25, 2011 at 21:11

hi fushar,
can problems like http://acm.hdu.edu.cn/showproblem.php?pid=2971 solved by this method?

fushar says:
August 26, 2011 at 17:42

Hi, 3haboys. I'll think about this problem. If there are readers that can solve this problem, please share

imiro says:
September 8, 2011 at 00:38

Wow! What a nice writing you have here. Two thumbs up!

Google Code Jam 2008 round 1A, problem C, (Numbers, http://code.google.com/codejam/contest/dashboard?c=32016#s=p2) could also be solved using this method.

Natsu says:
February 27, 2013 at 16:14

Can you provide any hints for solving the following spoj problem?
http://www.spoj.com/problems/ASUMHARD/

Thanks

savan says:
June 22, 2013 at 20:26

Thank you very much man,
ur blog gave me confidence in solving recurrence problem
Thank u very very much

anon says:
July 22, 2013 at 11:58

I think your solution for problem "Summing Sums" will get TLE verdict, for N can be quite large (N<=50000) CMIIW~

Divij says:
February 18, 2014 at 14:28

Your solution for "Summing Sums" has complexity O(N^3*logT), which will give TLE as N<=50000

Speak Your Mind

Name *

Email *

Website

Post Comment

Return to top of page