扱うこと

$K+1$項間の線型漸化式を持つ数列のN項目は行列累乗で求めたら$O(K^3 \log N)$かかるが、このスライドの手法でやると、もうちょっと速くなる。

対象とする人

- 行列の積を知っている
- 行列式の定義を知っている
- 余因子展開(ラプラス展開)を知っていると嬉しいかも
- 競技プログラミングの知識を別に問うわけではないけど、実装を理解するにはPythonの知識が必要かも

考える問題

$$a_{n+K} = c_1 a_{n+K-1} + c_2 a_{n+K-2} + \cdots + c_{K-1} a_{n+1} + c_K a_n$$

という$K+1$項間線形漸化式があるとする。

$c_1, c_2, \ldots, c_K$と、$a_0, a_1, \ldots, a_{K-1}$ が与えられる。 a_Nを求めよ。

制約

- $1 \leq N \leq 10^{18}$
- $K \leq N$
- $K \leq 10^5$(だいたい)
- だいたい2秒程度で答えたい

行列累乗で解く

線型漸化式で次の項を求める操作は行列の積と考えることができる

$$
\begin{bmatrix}
a_{n+K} \\
a_{n+K-1} \\
a_{n+K-2} \\
\vdots \\
a_{n+2} \\
a_{n+1}
\end{bmatrix}
=
\begin{bmatrix}
c_1 & c_2 & c_3 & c_4 & \dots & c_K \\
1 & 0 & 0 & 0 & \dots & 0 \\
0 & 1 & 0 & 0 & \dots & 0 \\
0 & 0 & 1 & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \dots & 0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
a_{n+K-1} \\
a_{n+K-2} \\
a_{n+K-3} \\
\vdots \\
a_{n+1} \\
a_n
\end{bmatrix}
$$

すると、先ほどの式を繰り返し適用すると

$$\begin{bmatrix} a_{N+K-1} \\ a_{N+K-2} \\ a_{N+K-3} \\ \vdots \\ a_{N+1} \\ a_N \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 & c_4 & \dots & c_K \\ 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 1 & 0 \end{bmatrix}^N \begin{bmatrix} a_{K-1} \\ a_{K-2} \\ a_{K-3} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}$$

となる。
数列のN項目はA^Nを計算することで求められる。行列の積は1回あたり$O(K^3)$かかるので、A^Nを求めるのに繰り返し二乗法を使うと、a_Nは全体で$O(K^3 \log N)$ の計算量で求められる。

繰り返し二乗法: $A^{63} = A^{32} \times A^{16} \times A^8 \times A^4 \times A^2 \times A^1$ みたいに指数の肩を2べきの和で表すと N 乗が $O(\log N)$ 回の掛け算で表せることを利用したテク

さっきの制約ではKが10^5になることもあるので、$K^3 \log N$ はだいたい
$10^{15} * 60 = 6 \times 10^{16}$となって計算に1000万秒くらいかかり、だいぶキツい。

行列累乗で考えているとどうしてもK^3が重い。(Strassenのアルゴリズムなど$O(K^3)$
より速い行列積アルゴリズムも存在はするが、競プロ文脈だと速くないらしい)

もっと速くできます！！！！

ケイリーハミルトンの定理

緑の線形代数の教科書(線形代数講義と演習 改訂版 小林正典, 寺尾宏明) P98 定理17.1 を見ると、

> 定理17.1 (ケイリーハミルトンの定理)
> n次正方行列Aの固有多項式$\varphi_A(x)$にAを代入したものは零行列に等しい

つまり、$\varphi_A(A) = \mathbf{0}$。

ちなみに(1): 多項式に行列を代入するとは

ここで、多項式に行列を代入したものは、

$$x^3 + 2x^2 + 5x + 3\Big|_{x=A} = A^3 + 2A^2 + 5A + 3E$$

のように、行列の累乗とその和で定義されている。(定数項に注意！)

ちなみに(2): 固有多項式とは

固有多項式というのは、正方行列Aに対して、

$$\det(x - AE)$$

で定義される。

だから、

$$\varphi_A(x) = \det \begin{bmatrix} x - c_1 & -c_2 & -c_3 & -c_4 & \dots & -c_K \\ -1 & x & 0 & 0 & \dots & 0 \\ 0 & -1 & x & 0 & \dots & 0 \\ 0 & 0 & -1 & x & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & -1 & x \end{bmatrix}$$

ちなみに終わり

> 定理17.1 (ケイリーハミルトンの定理)(再掲)
> n次正方行列Aの固有多項式$\varphi_A(x)$にAを代入したものは零行列に等しい

ここで x^Nを $\varphi_A(x)$ で割ったあまりを$r(x)$、商を$q(x)$とする。つまり、
$x^N = q(x)\varphi_A(x) + r(x)$。ただし$\deg(r(x)) < \deg(\varphi_A(x))$

すると、$A^N = q(A)\varphi_A(A) + r(A) = \mathbf{0} + r(A) = r(A)$となる。

$r(x) = r_0 + r_1 x + \cdots + r_{K-1}x^{K-1}$　とおくと(行列Aは$K \times K$行列だから、$\varphi_A(x)$はK次多項式で、そのあまりである$r(x)$はたかだか$K-1$次)
$A^N = r(A) = r_0 E + r_1 A + \cdots + r_{K-1}A^{K-1}$とわかる。

驚きの事実

求めたいa_{N+1}は、A^Nの一番下の行と、$[a_{K-1}, a_{K-2}, \ldots, a_0]$の内積だった。
だから、$E, A^1, A^2, \ldots A^{K-1}$の一番下の行のみわかればよい。

ここで、驚きの事実として、A^i ($0 \leq i < K$) の一番下の行は、$K - i$列目だけ1で、他は0となっている。

なので、

$$r_i A^i \begin{bmatrix} a_{K-1} \\ a_{K-2} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & * & * \\ 0 & \ldots & r_i & \ldots & 0 \end{bmatrix}$$

のようになる。(iがあるのは$K - i$列目)

驚きの事実の説明

横ベクトル$\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots \boldsymbol{p_K}$を縦に並べた行列に左からAをかけることを考える。

$$\begin{bmatrix} c_1 & c_2 & c_3 & c_4 & \ldots & c_K \\ 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p_1} \\ \boldsymbol{p_2} \\ \boldsymbol{p_3} \\ \boldsymbol{p_4} \\ \vdots \\ \boldsymbol{p_K} \end{bmatrix} = \begin{bmatrix} c_1\boldsymbol{p_1} + c_2\boldsymbol{p_2} + \cdots + c_K\boldsymbol{p_K} \\ \boldsymbol{p_1} \\ \boldsymbol{p_2} \\ \boldsymbol{p_3} \\ \vdots \\ \boldsymbol{p_{K-1}} \end{bmatrix}$$

このように、行列に対してAを左からかけると、行が一行下にずれて、一番上の行は行の線形結合になる。

ここで、

> 驚きの事実
> A^i ($0 \leq i < K$) の一番下の行は、$K-i$列目だけ1で、他は0となっている。

を拡張した、

> 驚きの事実2
> A^i ($0 \leq i < K$)のj行目 ($i+1 \leq j \leq K$)は、$j-i$列目だけ1で、他は0となっている。

を考える。 A^0やA^1について成立するのは明らか。

$$A = \begin{bmatrix} c_1 & c_2 & c_3 & c_4 & \dots & c_K \\ 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 1 & 0 \end{bmatrix}$$

> 驚きの事実2
> A^i $(0 \leq i < K)$のj行目 $(i+1 \leq j \leq K)$は、$j-i$列目だけ1で、他は0となっている。

帰納法で証明する。$i = 0, 1$で成立するのは明らか。0以上K未満のある整数iについて驚きの事実2が成立すると仮定する。

仮定から、A^iの j行目 $(i+1 \leq j \leq K)$は、$j-i$列目だけ1で、他は0。

$AA^i = A^{i+1}$だから、A^{i+1}のj行目 $(2 \leq j \leq K)$は A^iの$j-1$行目に一致する。

つまり、$(i+1)+1 \leq j \leq K$ なるjについて、A^{i+1}のj行目はA^iの$j-1$行目と同じく$j-1-i = j-(i+1)$列目だけ1で、他は0であるとわかる。

これは、

驚きの事実2が$i+1$でも成立していることに他ならない。

帰納的に、$0 \leq i < K$についても驚きの事実2が成立する。(説明終わり)

驚きの事実などを使うと、

$$r_0E + r_1A + \cdots + r_{K-1}A^{K-1} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ * & * & * & * & * \\ r_{K-1} & r_{K-2} & \cdots & r_1 & r_0 \end{bmatrix}$$

となることがわかるから、$a_N = r_{K-1}a_{K-1} + r_{K-2}a_{K-2} + \ldots r_1a_1 + r_0a_0$である。

これまでのことをまとめると、

1. 線型漸化式の遷移を表す行列の固有多項式$\varphi_A(x)$を求める。
2. x^Nを$\varphi_A(x)$で割った余りを求めて、それを$r_0 + r_1x + \cdots + r_{K-1}x^{K-1}$ とする。
3. $a_N = r_{K-1}a_{K-1} + \cdots + r_0a_0$である。

という3ステップでa_Nが計算できることがわかった。

ステップ1: $\varphi_A(x)$を求める

実は、固有多項式は線型漸化式によって決められるので、入力で線型漸化式を受け取ってから、掃き出し法をして ... みたいな計算は必要なく、($c_1, c_2, \ldots c_K$を使った式で)事前に求められる。。(あたりまえかも)

$$\varphi_A(x) = |xE - A| = \begin{vmatrix} x - c_1 & -c_2 & -c_3 & -c_4 & \ldots & -c_K \\ -1 & x & 0 & 0 & \ldots & 0 \\ 0 & -1 & x & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & 0 & -1 & x \end{vmatrix}$$

(結論を先に書くと、$\varphi_A(x) = x^K - c_1 x^{K-1} - \cdots - c_{K-1} x^1 - c_K$)

()

ステップ2: x^Nを $\varphi_A(x)$で割った余りを求める

mod $\varphi_A(x)$ 上でx^Nを計算すればよい。FFTを活用して頑張ると割り算に$O(K \log K)$かかり、繰り返し二乗法で掛け算を$O(\log N)$回やるので $O(K \log K \log N)$ で求められる。

(割り算を筆算で実装した場合は 割り算で $O(K^2)$ かかるので全体では $O(K^2 \log N)$ になる)

ステップ3 $a_N = r_K a_K + r_{K-1} aK - 1 + \cdots + r_0 a_0$を計算する

はい。やるだけ。

実装

次の２つの実装方針で、出来ることと必要な労力がだいぶ変わる

1. $O(K \log K)$ での割り算を頑張って実装する場合

- メリット: 速い。ライブラリとして殴れる幅がかなり広がる。
- デメリット: 実装がかなり重すぎる(後述)。任意modでやろうとするとなおさら。

2. $O(K^2)$ での割り算を筆算で実装する場合

- メリット: 遅い
- デメリット: 実装が(比較的)軽い

ということで、このスライドでは方針2: $O(K^2)$の筆算で妥協することにする。

(ちなみに) $O(K \log K)$ での割り算を頑張って実装する場合

前提として、FFT(高速フーリエ変換)ができる必要がある(`double` での複素数計算の誤差が無視できる or mod998244353のような、$p-1$の約数に2べきがいっぱいある法で考える場合)

1. FFTを使った畳み込みを実装する
2. K 項の形式的べき級数の逆元をニュートン法と畳み込みを使って $O(K \log K)$ で求める関数を実装する
3. reverseしたりinvをうまく使うと割り算が $O(K \log K)$ でできる

(この方針で実装したライブラリ in C++: https://harui-i.github.io/library/formal-power-series/fiduccia.hpp)

理論を含めて詳しく知りたい人へ: コンピュータ代数ハンドブックを読みましょう。都立大の南大沢の図書館にあります(日野にもあるかも。

実際に実装してみる with Python

```
print("間に合いませんでした")
```

Verify用問題など

- https://atcoder.jp/contests/tdpc/tasks/tdpc_fibonacci (AtCoder, mod $10^9 + 7$) で解く問題で、 $O(K^2 \log N)$ が通る制約になっている。)
- https://judge.yosupo.jp/problem/kth_term_of_linearly_recurrent_sequence (Library Checker, まさにこのスライドで扱ったような問題。 mod 998244353で求める問題で、 $O(K \log K \log N)$ が想定で、$O(K^2 \log N)$ は通らない。)

参考文献

https://blog.miz-ar.info/2019/02/typical-dp-contest-t/ :
https://qiita.com/ryuhe1/items/da5acbcce4ac1911f47a : Bostan-MoriのMoriさんです