动态规划—矩阵优化DP 学习笔记

前置知识：矩阵、矩阵乘法。

矩阵乘法优化线性递推

斐波那契数列

在斐波那契数列当中，$f_1 = f_2 = 1$，$f_i = f_{i-1} + f_{i-2}$，求 f_n。

而分析式子可以知道，求 f_k 仅与 f_{k-1} 和 f_{k-2} 有关；
所以我们设矩阵 $F_i = \begin{bmatrix} f_{i-1} & f_{i-2} \end{bmatrix}$。

设矩阵 Base，使得 $F_{i-1} \times \mathrm{Base} = F_i$，接下来考虑 Base 是什么；
带入可得 $\begin{bmatrix} f_{i-2} & f_{i-3} \end{bmatrix} \times \mathrm{Base} = \begin{bmatrix} f_{i-1} & f_{i-2} \end{bmatrix}$。

即 $\begin{bmatrix} f_{i-2} & f_{i-3} \end{bmatrix} \times \mathrm{Base} = \begin{bmatrix} f_{i-2} + f_{i-3} & f_{i-2} \end{bmatrix}$；
根据矩阵乘法的规则可知 Base 的第 1 列应为 $\begin{bmatrix} 1 & 1 \end{bmatrix}^{\mathrm{T}}$，第 2 列应为 $\begin{bmatrix} 1 & 0 \end{bmatrix}^{\mathrm{T}}$。

所以求得 $\mathrm{Base} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$。

然后考虑 f_i 的值应该是多少；
根据前面的公式可以知道 $f_i = F_{n+1}$ 的第一个数，所以就是求这个数。

根据 $f_1 = f_2 = 1$，可以知道 $F_3 = \begin{bmatrix} f_2 & f_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}$，我们将这个作为边界值；
然后有 $F_4 = F_3 \times \mathrm{Base}$，$F_5 = F_4 \times \mathrm{Base} = F_3 \times \mathrm{Base} \times \mathrm{Base}$。

因为矩阵乘法有结合律，所以 $F_{n+1} = F_3 \times \mathrm{Base}^{n-2} = \begin{bmatrix} 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2}$。

因为矩阵没有交换律，所以 F_3（前）和 Base^{n-2}（后）一定不能写反了！

例题 1

$$\begin{cases} f_1 = f_2 = 0 \\ f_i = f_{i-1} + f_{i-2} + 1 \end{cases}$$

一点点看题解

▼ 点击查看题解

f_i 仅与 f_{i-1} 和 f_{i-2} 有关，同时还包括了常数 1，
所以我们设 $F_i = \begin{bmatrix} f_{i-1} & f_{i-2} & 1 \end{bmatrix}$，

然后设 Base 使得 $F_{i-1} \times \text{Base} = F_i$，
即 $\begin{bmatrix} f_{i-2} & f_{i-3} & 1 \end{bmatrix} \times \text{Base} = \begin{bmatrix} f_{i-1} & f_{i-2} & 1 \end{bmatrix}$。

因为 $f_{i-1} = f_{i-2} + f_{i-3} + 1$，所以易知：

$$\text{Base} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

边界条件为 $F_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$，
所以 $F_{n+1} = F_3 \times \text{Base}^{n-2}$。

即可求出 f_n.

例题２

$$\begin{cases} f_1 = 0，f_2 = 1 \\ f_i = f_{i-1} + f_{i-2} + i \end{cases}$$

▼ 点击查看题解

f_i 仅与 f_{i-1}、f_{i-2} 和 i 有关，为实现 i 的递增，还需设置常量 1；
所以我们设 $F_i = \begin{bmatrix} f_{i-1} & f_{i-2} & i & 1 \end{bmatrix}$，

由 $F_{i-1} \times \text{Base} = F_i$ 得 $\text{Base} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$.

边界条件为 $F_3 = \begin{bmatrix} 1 & 0 & 3 & 1 \end{bmatrix}$.

$F_{n+1} = F_3 \times \text{Base}^{n-2}$；即可求出 f_n。

例题３（来自 OI-Wiki）

$$\begin{cases} f_1 = f_2 = 0 \\ f_n = 7f_{n-1} + 6f_{n-2} + 5n + 4 \times 3^n \end{cases}$$

▼ 点击查看题解

我的解法与 OI-Wiki 上的有所不同：

设 $F_n = \begin{bmatrix} f_{n-1} & f_{n-2} & n & 3^n & 1 \end{bmatrix}$.

易知 $\text{Base} = \begin{bmatrix} 7 & 1 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 \\ 5 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$.

边界值 $F_3 = \begin{bmatrix} 0 & 0 & 3 & 27 & 1 \end{bmatrix}$.

则 $F_{n+1} = F_3 \times \text{Base}^{n-2}$.

例题 4

$$\begin{cases} f_1 = f_2 = 0，f_3 = 1 \\ f_i = 3f_{i-1} + 2f_{i-2} + f_{i-3} + 5i + 7 \end{cases}$$

▼ 点击查看题解

增加了 f_{i-3}，但是本质是一样的。

可以设 $F_i = \begin{bmatrix} f_{i-1} & f_{i-2} & f_{i-3} & i & 1 \end{bmatrix}$,

易得 $\text{Base} = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 1 \end{bmatrix}$.

而 $F_4 = \begin{bmatrix} 1 & 0 & 0 & 4 & 1 \end{bmatrix}$,

则 $F_{n+1} = F_4 \times \text{Base}^{n-3}$。

例题 5

洛谷 P1939 矩阵加速（数列）：https://www.luogu.com.cn/problem/P1939

考虑这道题 Base 该如何设置。

▼ 点击查看代码

```
const long long MOD = 1e9 + 7;

struct matrix
{
    long long a[4][4];
```

```
    matrix operator*(const matrix &t) const
    {
        matrix res;
        memset(res.a, 0, sizeof res.a);
        for (int i = 1; i <= 3; ++i)
            for (int j = 1; j <= 3; ++j)
                for (int k = 1; k <= 3; ++k)
                    res.a[i][j] = (res.a[i][j] + a[i][k] * t.a[k][j] %
MOD) % MOD;
        return res;
    }
};

int main()
{
    int T = rr;
    while (T--)
    {
        int n = rr;

        if (n <= 3)
        {
            printf("1\n");
            continue;
        }

        matrix Base = {{{0, 0, 0, 0},
                        {0, 1, 1, 0},
                        {0, 0, 0, 1},
                        {0, 1, 0, 0}}};
        matrix res = {{{0, 0, 0, 0},
                       {0, 1, 0, 0},
                       {0, 0, 1, 0},
                       {0, 0, 0, 1}}};

        int k = n - 3;
        while (k)
        {
            if (k & 1)
                res = res * Base;
            k >>= 1, Base = Base * Base;
        }

        printf("%lld\n", (res.a[1][1] + res.a[2][1] + res.a[3][1]) %
```

```
MOD);
    }

    return 0;
}
```

时间复杂度

矩阵乘法 $O(k^3)$ 其中 k 为矩阵的长（或宽）；
快速幂 $O(\log n)$；

所以［矩阵乘法优化线性递推］的时间复杂度为 $O(k^3 \log n)$。

矩阵乘法优化 DP

朴素矩阵乘法

有 $\mathrm{dp}[t][x][y] = \sum_{w=1}^{n} \mathrm{dp}[t][x][w] \times G[w][y]$,

则可以看为矩阵乘法的形式：$\mathrm{dp}_t = \mathrm{dp}_{t-1} \times G$，即 $\mathrm{dp}_t = Ans_0 \times G^t$。

广义矩阵乘法

对矩阵的乘法重载，即可用快速幂求解了。

具体的，可以看这篇文章：https://www.luogu.com.cn/blog/i207M/xie-ti-bao-gao-sp1716-gss3-can-you-answer-these-queries-iii。

多组询问的矩阵乘法优化 DP

例题：P6569 魔法值

我们要求一个 $\mathrm{Ans}_k = \mathrm{Ans}_0 \times \mathrm{Mp}^k$，其中 Ans_i 是一个长度为 n 的行向量。

那么，我们先预处理 Mp^k，即 Mp^{2^i}。

然后我们就是在求一个行向量和 $\log_2 k$ 个 $n \times n$ 的矩阵的乘积了。

在算答案的时候，我们先别算这 $\log_2 k$ 个方阵的乘积，先用 Ans_0 向量从左乘到右。

因为向量乘矩阵复杂度是 $O(n^2)$ 的！

这样复杂度就从 $O(q \times n^3 \log_2 t)$，变成了 $O(n^3 \log_2 t + q \times n^2 \log_2 t)$。

练习题

见：https://www.luogu.com.cn/training/385249

Reference

[1] https://oi-wiki.org/math/linear-algebra/matrix/

[2] https://www.cnblogs.com/ningago/p/17472070.html

[3] https://www.cnblogs.com/luckyblock/p/14430820.html

[4] http://blog.tsawke.com/Data/Blog/content/DDP.html

[5] https://blog.csdn.net/qq_41739081/article/details/128184363

合集：学习笔记

标签：算法 , 学习笔记