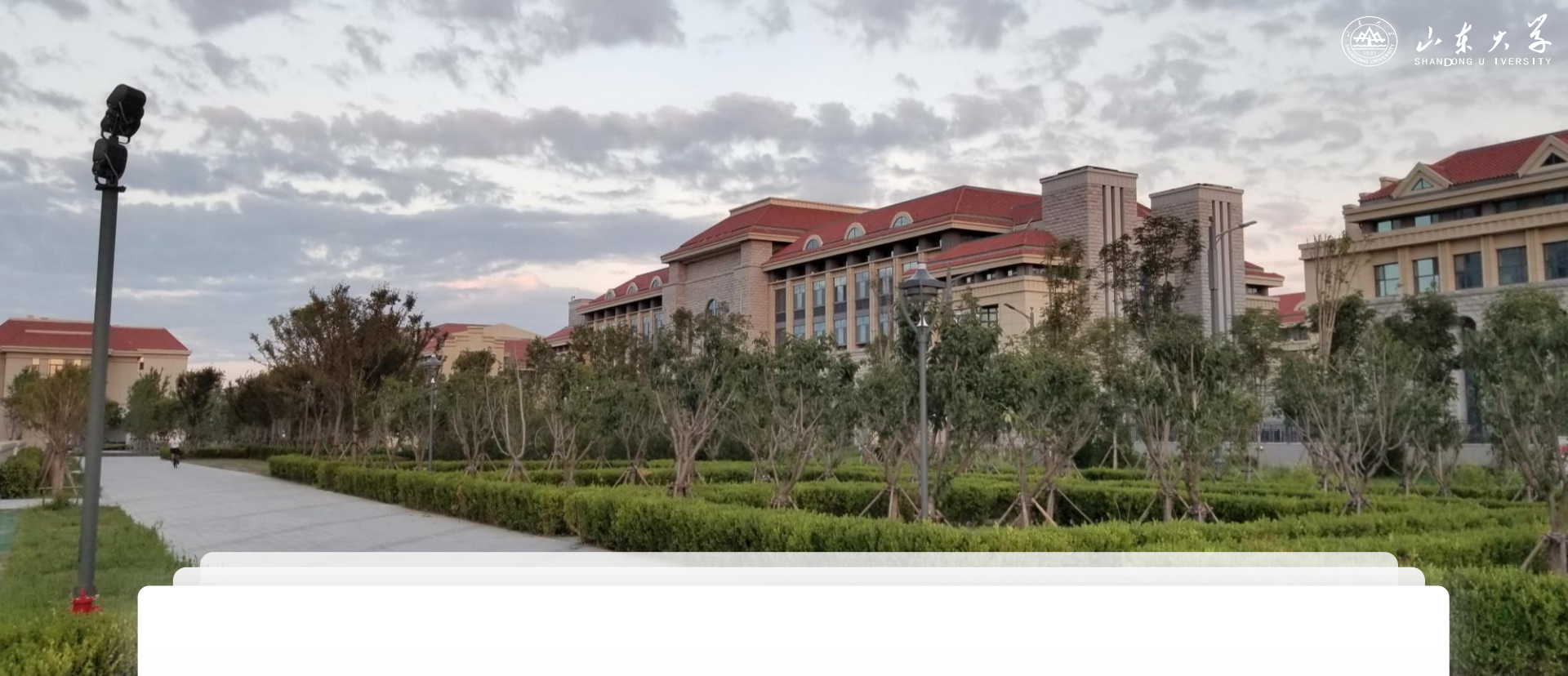




程序设计思维与实践

Thinking and Practice in Programming

矩阵快速幂 | 内容负责：苗顺源/尹浩飞

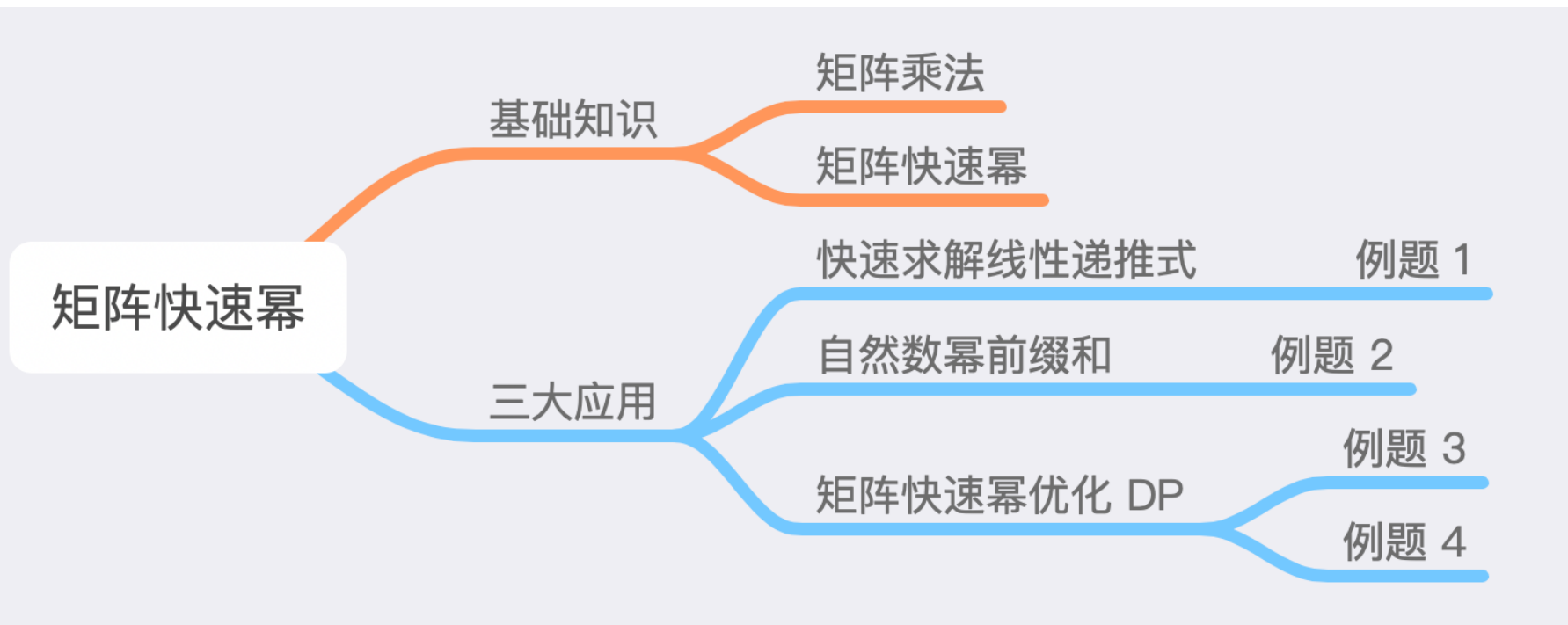


1

课程脉络

The outline of course

课程脉络





2

矩阵乘法

M a t r i x

矩阵乘法

1 预备知识

1.1 矩阵

矩阵可以看成是一个 $n \times m$ 的数表，其中 n, m 都是正整数。例如：

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix} \qquad \begin{bmatrix} 1 \end{bmatrix}$$

一般可以用 $\mathbf{A}[i, j]$ 表示第 i 行第 j 个数。

1.2 矩阵乘法

定义矩阵 \mathbf{A}, \mathbf{B} 。 \mathbf{A} 和 \mathbf{B} 可以作乘法操作当且仅当 \mathbf{A} 的大小是 $a \times b$ ， \mathbf{B} 的大小是 $b \times c$ ，其中 a, b, c 皆为正整数。设矩阵 $\mathbf{C} = \mathbf{AB}$ ，则 \mathbf{C} 的大小是 $a \times c$ ，且有

$$\mathbf{C}[i, j] = \sum_{k=1}^b \mathbf{A}[i, k] \mathbf{B}[k, j] \quad (1 \leq i \leq a, 1 \leq j \leq c) \quad (1)$$

而且矩阵乘法满足结合律，即 $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$ 。设 $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 的大小分别为 $a \times b, b \times c, c \times d$ ，证明如下：

矩阵乘法

$$\begin{aligned}
 ((\mathbf{AB}) \mathbf{C}) [i, j] &= \sum_{l=1}^c \left(\sum_{k=1}^b \mathbf{A} [i, k] \mathbf{B} [k, l] \right) \mathbf{C} [l, j] \\
 &= \sum_{k=1}^b \sum_{l=1}^c \mathbf{A} [i, k] \mathbf{B} [k, l] \mathbf{C} [l, j] \\
 &= \sum_{k=1}^b \mathbf{A} [i, k] \left(\sum_{l=1}^c \mathbf{B} [k, l] \mathbf{C} [l, j] \right) \\
 &= (\mathbf{A} (\mathbf{BC})) [i, j]
 \end{aligned} \tag{2}$$

而 $(\mathbf{AB}) \mathbf{C}$ 和 $\mathbf{A} (\mathbf{BC})$ 的大小都是 $a \times d$ ，得证。

直接按照定义进行矩阵乘法时间复杂度是 $O(abc)$ ，其中 a, b, c 分别表示两个矩阵大小是 $a \times b$ 以及 $b \times c$ 。将两个 $N \times N$ 的矩阵相乘，朴素算法的时间复杂度为 $O(N^3)$ 。其他有一些算法有更低的复杂度，例如 Strassen 算法时间复杂度约为 $O(N^{2.81})$ 。而现有的算法中理论复杂度最低的是 Coppersmith—Winograd 算法，时间复杂度约为 $O(N^{2.36})$ 。

矩阵乘法

- 矩阵 A , B 的大小分别为 $a \times b$ 和 $b \times c$
- 设 $C = AB$, 则 C 的大小为 $a \times c$

```
for(int i = 1; i <= a; i++) {  
    for(int j = 1; j <= c; j++) {  
        C[i][j] = 0;  
        for(int k = 1; k <= b; k++) {  
            C[i][j] += A[i][k] * B[k][j];  
        }  
    }  
}
```

- 一般我们只考虑方阵, 即 A 、 B 的大小都是 $n \times n$

矩阵乘法

- 推荐封装成一个结构体并且重载乘法运算符

```
const int N = 3;
struct Matrix {
    int x[N][N];
    Matrix operator*(const Matrix& t) const {
        Matrix ret;
        for(int i = 0; i < N; ++i) {
            for(int j = 0; j < N; ++j) {
                ret.x[i][j] = 0;
                for(int k = 0; k < N; ++k) {
                    ret.x[i][j] += x[i][k] * t.x[k][j];
                }
            }
        }
        return ret;
    }
};

// 强烈建议实现构造函数和复制构造!!!避免出现奇怪bug!!!
Matrix() { memset(x, 0, sizeof x); }
Matrix(const Matrix& t) { memcpy(x, t.x, sizeof x); }
};
```




矩阵快速幂

Fast Power of Matrix

矩阵快速幂

- 问题引入
 - 给定矩阵 A ，请快速计算出 A^n (n 个矩阵 A 相乘) 的结果，输出的每个数都 $\% p$

矩阵快速幂

- 简单回顾

- $2^{61} = 2^{32} \times 2^{16} \times 2^8 \times 2^4 \times 2^1$ (结合律)
- $2^{61} = (2^{32})^1 \times (2^{16})^1 \times (2^8)^1 \times (2^4)^1 \times (2^2)^0 \times (2^1)^1$

```
long long quick_pow(long long a, long long x) {  
    long long ans = 1;  
    while (x) {  
        if (x & 1) ans = ans * a;  
        a = a * a;  
        x >>= 1;  
    }  
    return ans;  
}
```

- 快速幂利用了运算满足结合律这一性质，只要某种运算满足结合律，都可以使用快速幂。
 - 例如：快龟速乘🐢

矩阵快速幂

- 矩阵乘法运算也满足结合律
- 矩阵快速幂！
 - 矩阵乘法中的单位元为单位矩阵 E
 - 快速幂中的乘法替换成矩阵乘法

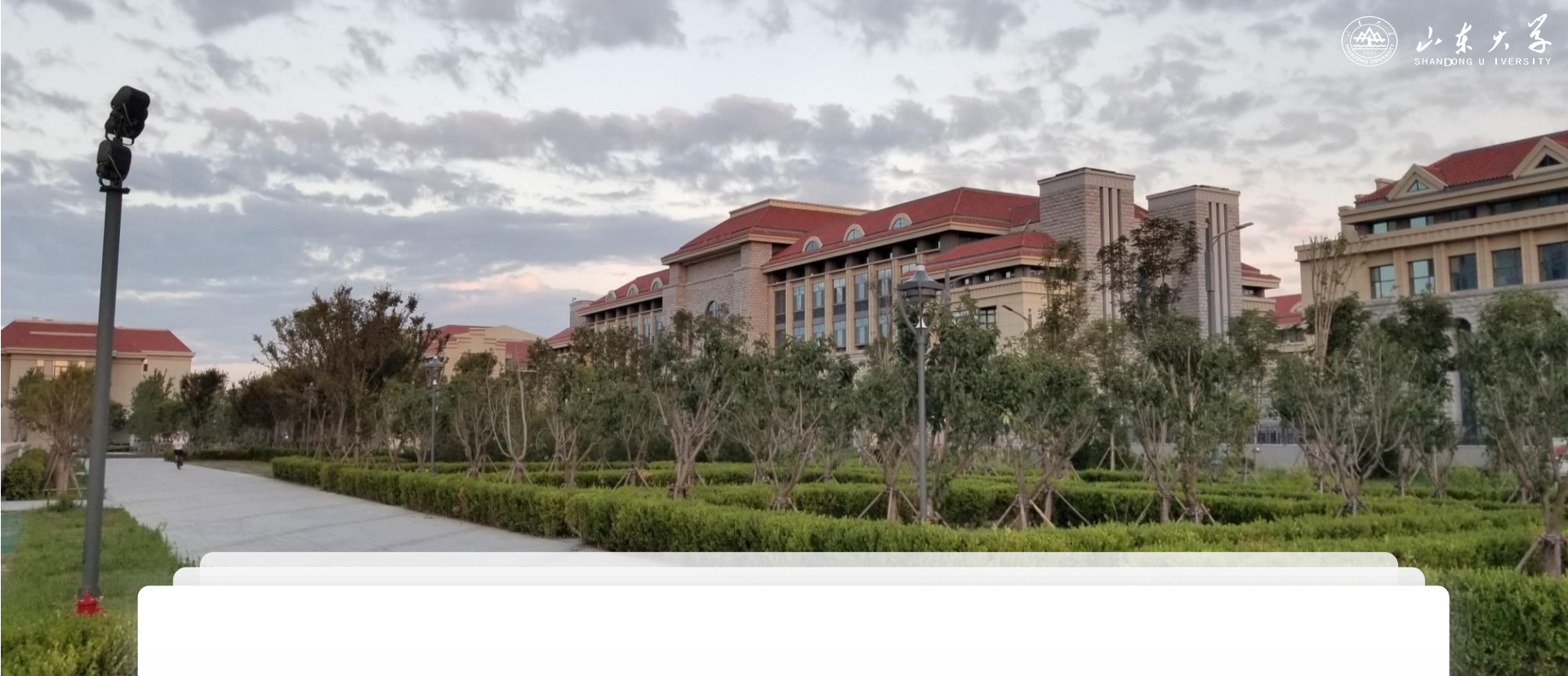
```
Matrix quick_pow(Matrix a, int x) {  
    Matrix ret;  
    /* 以2*2的矩阵为例  
       1 0  
       0 1  
    */  
    ret.x[0][1] = ret.x[1][0] = 0;  
    ret.x[0][0] = ret.x[1][1] = 1;  
    while (x) {  
        if(x & 1) ret = ret * a; // 重载*运算符  
        a = a * a;  
        x >>= 1;  
    }  
    return ret;  
}
```

矩阵快速幂

```
const int N = 2;
struct Matrix {
    int x[N][N];
    Matrix operator*(const Matrix& t) const {
        Matrix ret;
        for(int i = 0; i < N; ++i) {
            for(int j = 0; j < N; ++j) {
                ret.x[i][j] = 0;
                for(int k = 0; k < N; ++k) {
                    ret.x[i][j] += x[i][k] * t.x[k][j] % p;
                    ret.x[i][j] %= p;
                }
            }
        }
        return ret;
    }
};

// 强烈建议实现构造函数和复制构造!!!避免出现奇怪bug!!!
Matrix() { memset(x, 0, sizeof x); }
Matrix(const Matrix& t) { memcpy(x, t.x, sizeof x); }

Matrix quick_pow(Matrix a, int x) {
    Matrix ret;
    /* 以2*2的矩阵为例
        1 0
        0 1
    */
    ret.x[0][1] = ret.x[1][0] = 0;
    ret.x[0][0] = ret.x[1][1] = 1;
    while (x) {
        if(x & 1) ret = ret * a; // 重载*运算符
        a = a * a;
        x >>= 1;
    }
    return ret;
}
```

4

求解线性递推

Linear recursive

求解线性递推

- 再说斐波那契
 - 求解斐波那契第 N 项 $\% P$ 的余数
 - $N \leq 10^9, P \leq 10^8$
- 斐波那契数列

$$f(n) = \begin{cases} 1 & , n = 1, 2 \\ f(n-1) + f(n-2), & n \geq 3 \end{cases}$$

求解线性递推

- 斐波那契数列

$$f(n) = \begin{cases} 1 & , n = 1, 2 \\ f(n-1) + f(n-2), & n \geq 3 \end{cases}$$

- 矩阵快速幂

$$\begin{aligned} \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2 \cdot \begin{bmatrix} f(n-2) \\ f(n-3) \end{bmatrix} \\ &= \dots = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \cdot \begin{bmatrix} f(2) \\ f(1) \end{bmatrix} \end{aligned}$$

求解线性递推

- 如下式子应该如何求解？

- $f(n) = 2 * f(n - 1) + 4 * f(n - 2) + 3 * f(n - 3) + 5$

- 更一般的情况，又该如何求解？

- $f(n) = a_1 * f(n - 1) + a_2 * f(n - 2) + \dots + a_k * f(n - k) + b$

- 关键在于如何构造转移矩阵！



1

自然数幂和

Sum of the power

矩阵快速幂

- 问题引入
 - 给定 $n \leq 10^9$ 和 $k \leq 10$, 计算 $\sum_{i=1}^n i^k$
- 思路
 - n 很大, k 很小, 比较类似于矩阵快速幂的题
 - 可以考虑令 $S_n = \sum_{i=1}^n i^k$
 - 问题变成了如何构造矩阵, 使得可以从 S_{n-1} 推到 S_n

矩阵快速幂

- 令 $S_n = \sum_{i=1}^n i^k$, 推导如下:

- $S_n = S_{n-1} + n^k$

- $$\begin{aligned} n^k &= (n-1+1)^k \\ &= C_k^0 (n-1)^k + C_k^1 (n-1)^{k-1} + \dots + C_k^k (n-1)^0 \end{aligned}$$

- $$\begin{aligned} n^{k-1} &= (n-1+1)^{k-1} \\ &= C_{k-1}^0 (n-1)^{k-1} + C_{k-1}^1 (n-1)^{k-2} + \dots \end{aligned}$$

- ...

矩阵快速幂

- 利用二项式展开构造转移矩阵

$$\begin{bmatrix} S_n \\ n^k \\ n^{k-1} \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & C_k^0 & C_k^1 & C_k^2 & \dots & C_k^{k-1} & C_k^k \\ 0 & C_k^0 & C_k^1 & C_k^2 & \dots & C_k^{k-1} & C_k^k \\ 0 & 0 & C_{k-1}^0 & C_{k-1}^1 & \dots & C_{k-1}^{k-2} & C_{k-1}^{k-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{n-1} \\ (n-1)^k \\ (n-1)^{k-1} \\ \vdots \\ 1 \end{bmatrix}$$



矩阵快速幂优化DP

Fast Matrix Exponentiation

动态规划回顾

- 动态规划常见模型
 - 线性型
 - 坐标型
 - 背包型
 - 区间型
 - 状态压缩型
 - 树型
 - DP 的单调队列优化
 - **DP 的矩阵快速幂优化**

矩阵快速幂优化 DP

- 矩阵快速幂性质
 - 快速求解线性递推式的结果，如斐波那契递推
- 优化的 DP 需要满足的条件
 - 转移方程为线性递推式
 - 转移次数超级多 (n 很大)
- 接下来以例3、例4为例进一步讲解



例 3 讲 解

E x a m p l e 3

例3讲解

- 题意
 - 现需要用红、蓝、绿、黄四种颜色对 n 个连续格子染色
 - 询问满足红色、绿色格子数量同时为偶数的染色方案有多少种？（ $1 \leq n \leq 10^9$ ）
- 举例
 - Input: 2
 - Output: 6
 - [红红、绿绿、蓝蓝、黄黄、蓝黄、黄蓝]

例3讲解

- 算法判断?
 - 连续格子染色，很明显有子结构的性质，可以考虑 DP
 - 但是 n 很大，因此考虑矩阵快速幂优化 DP
- DP 思考
 - DP 状态
 - DP 转移方程（线性递推式）

例3讲解

- DP 状态思考?
 - 题目问 n 个格子，红绿均为偶数的染色方案数
 - 因此令 $A[i]$ 表示 i 个格子，红绿均为偶数的染色方案数
 - 只有 $A[i]$ 是否足够?
- DP 状态添加
 - 偶数需要从奇数转移而来，这一点在思考转移方程的时候可以发现，因此我们进行 DP 状态添加
 - $B[i]$ 表示 i 个格子，红绿均为奇数的染色方案数
 - $C[i]$ 表示 i 个格子，红绿有一个为偶数的染色方案数

例3讲解

- DP 转移方程
 - 根据 DP 状态，仔细思考转移情况即可得到
 - $A[i] = 2 * A[i-1] + C[i-1]$
 - $B[i] = 2 * B[i-1] + C[i-1]$
 - $C[i] = 2 * A[i-1] + 2 * B[i-1] + 2 * C[i-1]$
- 矩阵快速幂

$$ans[i] = \begin{bmatrix} A[i] \\ B[i] \\ C[i] \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix} * ans[i-1]$$

- 复杂度： $O(3^3 \log(n)) = O(\log(n))$



例 4 讲 解

E x a m p l e 4

例4讲解

- 题意
 - ZJM 有 M 件衬衫，现一共有 N 天
 - 如果 ZJM 昨天穿衬衫 A ，今天穿衬衫 B ，则他今天可以获得 $H[A][B]$ 快乐值
 - 询问 N 天过后，ZJM 最多可以获得多少快乐值？
 - ($2 \leq N \leq 10^5, 1 \leq M \leq 100$)
- 举例
 - Input: $N = 3, M = 2$, H 矩阵如下:
 - 0 1
 - 1 0
 - Output: 2

例4讲解

- 算法判断?
 - 天数连续，很明显有子结构的性质，可以考虑 DP
 - n 不是很大，因此不一定是矩阵快速幂优化，先列出 DP 状态和 DP 转移方程再进行判断
- DP 状态思考?
 - 题目问 n 天后，最多快乐值，且每天的快乐值由今日与昨日两天的衣服决定
 - 因此令 $f[i][j]$ 表示第 i 天，穿的衣服为 j 所获得的快乐值总和

例4讲解

- DP 转移方程
 - 根据 DP 状态，不难得到如下的转移方程
 - 枚举前一天穿的衣服为 k ，即
 - $f[i][j] = \max(f[i-1][k] + H[k][j]), 1 \leq k \leq M$
- 是否可行？
 - 直接求解复杂度： $O(N * M * M) = O(10^9) = O(\text{梦里有})$
- 如何优化？
 - 矩阵快速幂

例4讲解

- 矩阵快速幂转化方法

- $$f[i][j] = \max(f[i-1][k] + H[k][j]), 1 \leq k \leq M$$

$$ans[i] = \begin{bmatrix} f[i][1] \\ \vdots \\ f[i][M] \end{bmatrix} = \begin{bmatrix} H[1][1] & \cdots & H[M][1] \\ \vdots & \ddots & \vdots \\ H[1][M] & \cdots & H[M][M] \end{bmatrix} * \begin{bmatrix} f[i-1][1] \\ \vdots \\ f[i-1][M] \end{bmatrix}$$

- 转换过程

- $$C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$$

- 将累加换成了max，乘法换成了加法

例4讲解

- 为什么可行？
 - 矩阵快速幂要求矩阵乘法运算具有结合律
 - 因此新定义的矩阵乘法只要满足结合律，即正确
 - $C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$
- 如何证明？
 - $((AB)C)[i, j] = (A(BC))[i, j]$ ，具体数学证明在后一页
 - 写题时，可以用小例子验证
 - 三个 2×2 的矩阵，手动计算是否符合结合律

例4讲解

- 数学证明
 - $C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$
 - A、B、C 分别为 $a \times b$ 、 $b \times c$ 、 $c \times d$ 的矩阵

$$((AB)C)[i, j] = \max_{l=1}^c((AB)[i, l] + C[l, j])$$

例4讲解

- 数学证明
 - $C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$
 - A、B、C 分别为 $a \times b$ 、 $b \times c$ 、 $c \times d$ 的矩阵

$$\begin{aligned} ((AB)C)[i, j] &= \max_{l=1}^c ((AB)[i, l] + C[l, j]) \\ &= \max_{l=1}^c (\max_{k=1}^b (A[i, k] + B[k, l]) + C[l, j]) \end{aligned}$$

例4讲解

- 数学证明

- $C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$

- A、B、C 分别为 $a \times b$ 、 $b \times c$ 、 $c \times d$ 的矩阵

$$\begin{aligned} ((AB)C)[i, j] &= \max_{l=1}^c ((AB)[i, l] + C[l, j]) \\ &= \max_{l=1}^c (\max_{k=1}^b (A[i, k] + B[k, l]) + C[l, j]) \\ &= \max_{l=1}^c \max_{k=1}^b (A[i, k] + B[k, l] + C[l, j]) \end{aligned}$$

例4讲解

- 数学证明
 - $C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$
 - A、B、C 分别为 $a \times b$ 、 $b \times c$ 、 $c \times d$ 的矩阵

$$\begin{aligned}
 ((AB)C)[i, j] &= \max_{l=1}^c((AB)[i, l] + C[l, j]) \\
 &= \max_{l=1}^c(\max_{k=1}^b(A[i, k] + B[k, l]) + C[l, j]) \\
 &= \max_{l=1}^c \max_{k=1}^b(A[i, k] + B[k, l] + C[l, j]) \\
 &= \max_{k=1}^b \max_{l=1}^c(A[i, k] + B[k, l] + C[l, j])
 \end{aligned}$$

例4讲解

- 数学证明

- $$C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$$

- A、B、C 分别为 $a \times b$ 、 $b \times c$ 、 $c \times d$ 的矩阵

$$\begin{aligned}
 ((AB)C)[i, j] &= \max_{l=1}^c ((AB)[i, l] + C[l, j]) \\
 &= \max_{l=1}^c (\max_{k=1}^b (A[i, k] + B[k, l]) + C[l, j]) \\
 &= \max_{l=1}^c \max_{k=1}^b (A[i, k] + B[k, l] + C[l, j]) \\
 &= \max_{k=1}^b \max_{l=1}^c (A[i, k] + B[k, l] + C[l, j]) \\
 &= \max_{k=1}^b (A[i, k] + \max_{l=1}^c (B[k, l] + C[l, j]))
 \end{aligned}$$

例4讲解

- 数学证明
 - $C[i][j] = \Sigma(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$
 - A、B、C 分别为 $a \times b$ 、 $b \times c$ 、 $c \times d$ 的矩阵

$$\begin{aligned}
 ((AB)C)[i, j] &= \max_{l=1}^c ((AB)[i, l] + C[l, j]) \\
 &= \max_{l=1}^c (\max_{k=1}^b (A[i, k] + B[k, l]) + C[l, j]) \\
 &= \max_{l=1}^c \max_{k=1}^b (A[i, k] + B[k, l] + C[l, j]) \\
 &= \max_{k=1}^b \max_{l=1}^c (A[i, k] + B[k, l] + C[l, j]) \\
 &= \max_{k=1}^b (A[i, k] + \max_{l=1}^c (B[k, l] + C[l, j])) \\
 &= \max_{k=1}^b (A[i, k] + (BC)[k, j])
 \end{aligned}$$

例4讲解

- 数学证明
 - $C[i][j] = \sum(A[i][k] * B[k][j]) \Rightarrow \max(A[i][k] + B[k][j])$
 - A、B、C 分别为 $a \times b$ 、 $b \times c$ 、 $c \times d$ 的矩阵

$$\begin{aligned}
 ((AB)C)[i, j] &= \max_{l=1}^c((AB)[i, l] + C[l, j]) \\
 &= \max_{l=1}^c(\max_{k=1}^b(A[i, k] + B[k, l]) + C[l, j]) \\
 &= \max_{l=1}^c \max_{k=1}^b(A[i, k] + B[k, l] + C[l, j]) \\
 &= \max_{k=1}^b \max_{l=1}^c(A[i, k] + B[k, l] + C[l, j]) \\
 &= \max_{k=1}^b(A[i, k] + \max_{l=1}^c(B[k, l] + C[l, j])) \\
 &= \max_{k=1}^b(A[i, k] + (BC)[k, j]) \\
 &= (A(BC))[i, j]
 \end{aligned}$$

例4讲解

- 本质原因
 - \max 对于加法可分配, 即 $\max(a + b, a + c) = a + \max(b + c)$
 - \max 、加法均可结合, 可交换
 - 因此对应的矩阵乘法也满足结合律
- 总结
 - 矩阵快速幂可优化包含 \min 、 \max 、加减乘除的递推式



为天下储人才
为国家图富强

感谢收听

Thank You For Your Listening