307-09-10矩阵乘法与快速幂

矩阵乘法

定义矩阵A，B，其中A的大小为$a \times b$，B的大小为$b \times c$，对于矩阵$C = AB$中的每一个元素$C(i.j)$，$i \in [1, a]$，$j \in [1, c]$，存在以下：

$$C(i, j) = \sum_{k=1}^{b} A(i, k) \times B(k, j) \tag{1}$$

矩阵乘法的结合律

矩阵乘法存在结合律，首先定义矩阵$A_{a \times b}$，$B_{b \times c}$，$C_{c \times d}$，存在$(AB)C = A(BC)$。证明：

对于$i \in [1, a]$，$j \in [1, d]$，有：

$$\begin{aligned}
((AB)C)(i, j) &= \sum_{l=1}^{c} (\sum_{k=1}^{b} A(i, k) \times B(k, l)) \times C(l, j) && (2)\\
&= \sum_{k=1}^{b} \sum_{l=1}^{c} A(i, k) \times B(k, l) \times C(l, k) && (3)\\
&= \sum_{k=1}^{b} A(i, k) \times (\sum_{l=1}^{c} B(k, l) \times C(l, k)) && (4)\\
&= (A(BC))(i, j) && (5)
\end{aligned}$$

由此，可以证明，矩阵的乘法具有结合律。

Floyd 算法

Floyd算法的主要目的是在一张图上求任意两点之间的最短路，而最短路的核心思想其实可以由一个方程来表达：

$$dis[i][j] = \min_{j \in [1,n]} \{dis[i][j],\ dis[i][k] + dis[k][j]\} \tag{6}$$

其表示$i \to j$的最短路。

但是它的初始化值得注意，其应该初始化为一个图论的单位矩阵，即主对角线的值为0，其余值均为∞的一个“单位矩阵”，可以如下定义：

$$dis[i][j] = \begin{cases} \infty & i \neq j \\ 0 & i = j \end{cases} \tag{7}$$

为了实现上述思想，以下为代码实现：

```
for (int k = 1; k <= n; k ++)
    for (int i = 1; i <= n; i ++)
        for (int j = 1; j <= n; j ++)
            if (m[i][k] + m[k][j] < m[i][j])
                m[i][j] = m[i][k] + m[k][j];
```

注：Floyd算法的k的那层循环必须在最外面，否则有些值无法被更新。

快速幂

如果我们要计算a^n，则会将a乘n次，时间复杂度为$O(n)$，而这太浪费时间了，为此，有了快速幂之中算法，快速幂可以将时间复杂度降为$O(\log n)$，其本质就是将指数化为一个2进制数来进行记忆化的乘法，现在假设有一个需求要求a^{13}，我们可以试着将13化为二进制：$13 = (1101)_2$，也就是说：

$$a^{13} = a^{2^3+2^2+2^0} \tag{8}$$

换言之，如果我们要计算a^k，我们只需要找到：$2^{k_1} + 2^{k_2} + \ldots + 2^{k_m} = n$，这样可以进行递推，很快地完成整个运算过程：

```cpp
ll quick_pow(int a, int n)  {
    ll ans = 1;
    while (n > 0) {
        // n的最后一位是否为1，若为1就进行
        if (n & 1)
            ans = ans * a;
        n >>= 1;
        a = a * a;
    }
    return ans;
}
```

矩阵乘法结合律的应用

前文已经证明了矩阵的乘法具有结合律，而既然有结合律那么自然可以使用快速幂求一个矩阵的幂运算，对于基于乘法的快速幂算法而言，需要修改的是运算符和初始化值。

在矩阵乘法中的初始化值也比较重要，当然这一步也可以忽略，因为执行$n-1$次其实也可以达到目的。但是还有一种更为优美的写法，就是初始化一个单位矩阵A_0：

$$A_0(i,j)=\begin{cases}0 & i\neq j\\ 1 & i=j\end{cases} \tag{9}$$

这样做的原因是对于任意一个矩阵A而言，存在$AA_0=A$。

代码（模板P3390）：

```cpp
#include <iostream>
#include <stdio.h>

using namespace std;

typedef long long ll;

const int maxn = 105;
const int P = 1000000007;

struct matrix {
    ll m[maxn][maxn];
};

ll n, k;

matrix matrix_multi(matrix a, matrix b) {
    matrix ans;
    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= n; j ++) {
            ans.m[i][j] = 0;
```

```
            for (int k = 1; k <= n; k ++) {
                ans.m[i][j] = (ans.m[i][j] % P + (a.m[i][k] * b.m[k][j]) % P) % P;
            }
        }
    }
    return ans;
}

matrix quick_matrix_pow(matrix a, ll t) {
    matrix ans;
    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= n; j ++) {
            if (i == j) ans.m[i][j] = 1;
            else        ans.m[i][j] = 0;
        }
    }
    while (t > 0) {
        if (t & 1)
            ans = matrix_multi(a, ans);
        a = matrix_multi(a, a);
        t >>= 1;
    }
    return ans;
}

int main() {
    cin >> n >> k;
    matrix a;
    for (int i = 1; i <= n; i ++)
        for (int j = 1; j <= n; j ++)
            scanf("%lld", &a.m[i][j]);
    matrix ans = quick_matrix_pow(a, k);
    for (int i = 1; i <= n; i ++) {
        for (int j = 1; j <= n; j ++)
            cout << ans.m[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

使用矩阵乘法（快速幂）求斐波那契数列

斐波那契数列：

$$f(n) = f(n-1) + f(n-2),\ f(1) = f(2) = 1 \tag{10}$$

如果使用递归，dp，记忆化，等的方式，时间复杂度仍然至少为$O(n)$。

现在购下一个矩阵A，使得如下等式成立：

$$A\begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix} = \begin{pmatrix} f(n+1) \\ f(n) \end{pmatrix} \tag{11}$$

由于$f(n+1) = f(n) + f(n-1)$，所以设$A = \begin{pmatrix} x & y \\ z & w \end{pmatrix}$，则：

$$\begin{pmatrix} x & y \\ z & w \end{pmatrix}\begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix} = \begin{pmatrix} f(n) + f(n-1) \\ f(n) \end{pmatrix} = \begin{pmatrix} xf(n) + yf(n-1) \\ zf(n) + wf(n-1) \end{pmatrix} \tag{12}$$

所以可以得到结论：

$$\begin{cases} x = 1 \\ y = 1 \\ z = 1 \\ w = 0 \end{cases} \tag{13}$$

所以我们可以得到结论：有斐波那契数列前后连续的两项组成的一个列向量的矩阵乘上一个矩阵$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$便可以得到由斐波那契数列的后一项和当前项组成的另外一个列向量。

可以得到以下的推论：

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} f(n) \\ f(n-1) \end{pmatrix} = \begin{pmatrix} f(n+1) \\ f(n) \end{pmatrix} \tag{14}$$

由于前文证明的矩阵乘法的结合律，可以得到结论：

$$\begin{pmatrix} f(n+1) \\ f(n) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}\begin{pmatrix} f(2) \\ f(1) \end{pmatrix} \tag{15}$$

由此可以通过矩阵乘法的快速幂解决斐波那契数列的第n项求解，且时间复杂度为$O(\log n)$。

```cpp
#include <iostream>

using namespace std;
typedef long long ll;

const int P = 1000000007;

ll n;
struct matrix {
    ll a11, a12, a21, a22;
};

matrix matrix_multi(matrix a, matrix b) {
    matrix ans;
```

```
    ans.a11 = ((a.a11 * b.a11) % P + (a.a12 * b.a21) % P) % P;
    ans.a12 = ((a.a11 * b.a12) % P + (a.a12 * b.a22) % P) % P;
    ans.a21 = ((a.a21 * b.a11) % P + (a.a22 * b.a21) % P) % P;
    ans.a22 = ((a.a21 * b.a12) % P + (a.a22 * b.a22) % P) % P;
    return ans;
}

matrix matrix_quick_pow(matrix a, ll k) {
    matrix ans = (matrix) {1, 0, 0, 1};
    while (k > 0) {
        if (k & 1) ans = matrix_multi(ans, a);
        a = matrix_multi(a, a);
        k >>= 1;
    }
    return ans;
}

int main() {
    cin >> n;
    matrix e_0 = (matrix) {1, 1, 1, 0};
    matrix e_t;
    e_t = matrix_quick_pow(e_0, n - 2);
    ll ans = (e_t.a11 % P + e_t.a12 % P) % P;
    cout << ans << endl;
    return 0;
}
```

P2886 [USACO07NOV]牛继电器Cow Relays

For their physical fitness program, N ($2 \leqslant N \leqslant 1{,}000{,}000$) cows have decided to run a relay race using the T ($2 \leqslant T \leqslant 100$) cow trails throughout the pasture.

Each trail connects two different intersections ($1 \leqslant I1i \leqslant 1{,}000$; $1 \leqslant I2i \leqslant 1{,}000$), each of which is the termination for at least two trails. The cows know the lengthi of each trail ($1 \leqslant lengthi \leqslant 1{,}000$), the two intersections the trail connects, and they know that no two intersections are directly connected by two different trails. The trails form a structure known mathematically as a graph.

To run the relay, the N cows position themselves at various intersections (some intersections might have more than one cow). They must position themselves properly so that they can hand off the baton cow-by-cow and end up at the proper finishing place.

Write a program to help position the cows. Find the shortest path that connects the starting intersection (S) and the ending intersection (E) and traverses exactly N cow trails.

给出一张无向连通图，求S到E经过k条边的最短路。

输入样例#1： 复制

```
2 6 6 4
11 4 6
4 4 8
8 4 9
6 6 8
2 6 9
3 8 9
```

输出样例#1： 复制

```
10
```

题解：

对一个图的邻接矩阵进行平方，得到的每个元素(a,b)代表的就是由a到b经过两条道路有多少种走法。立方就是通过三条道路有多少种走法，以此类推：

$$G^2(a,b)=\sum_{i=1}^{n}G(a,i)G(i,b) \tag{16}$$

$$G^3(a,b)=\sum_{i=1}^{n}\sum_{j=1}^{n}G(a,i)G(i,j)G(j,b) \tag{17}$$

$$\cdots \tag{18}$$

$$G^k(a,b)=\sum_{i=1}^{n}G^{k-1}(a,i)G(i,b) \tag{19}$$

在这道题中要求经过i条边走到j的最短路，可以列出以下方程：

设恰好通过i条边，走到j点的最短距离为$f(i,j)$

$$f(i,j)=\min\{f(i-1,k)+G[k][j].w\} \tag{20}$$

我们可以对比一下类似地方程：设通过i条边，走到j点的方案数为$g(i,j)$

$$g(i,j)=\sum_{k=1}^{n}g(i-1,k)\times G[k][j].\text{side\_count} \tag{21}$$

我们可以简化一下方程：

对于一组点(i,j)设$C(i,j)$为$i\to j$通过两条边达到，将C考虑为一个数集，集矩阵，则：

$$C(i,j)=\min_{k=1}^{n}\{C(i,j),\ G(i,k).w+G(k,j).w\} \tag{22}$$

值得注意的是这其实与Floyd算法实现有所不同，因为k为参与min之中的条件，应该为最内层循环。

通过上面的这个式子，不难想到：如果要进过k条路，则要枚举出所有的中间节点，然而这个时候我们发现时间复杂度太大了，要想办法解决这个问题，同时考虑到这都是重复的运算，由此想到对取min这个操作进行快速幂运算思想的优化。

首先，证明结合律的正确性：

定义新运算：矩阵$A_{a\times b}, B_{b\times c}$：$(AB)(i,j) = \min_{k=1}^{b}\{A(i,k)+B(k,j)\}$

$$
\begin{aligned}
((AB)C)(i,j) &= \min_{l=1}^{c}\{\min_{k=1}^{b}\{A(i,k)+B(k,l)\}+C(l,j)\} && (23)\\
&= \min_{k=1}^{b}\min_{l=1}^{c}\{A(i,k)+B(k,l)+C(l,j)\} && (24)\\
&= \min_{k=1}^{b}\{A(i,k)+\min_{l=1}^{c}\{B(k,l)+C(l,j)\}\} && (25)\\
&= (A(BC))(i,j) && (26)
\end{aligned}
$$

综上所述，这是一道通过快速幂来进行矩阵的快速变换的最短路问题。

过程：设单位矩阵为A_0，图矩阵为A：

$$
\begin{aligned}
&\text{via 0 side: } A_0\\
&\quad\Downarrow \times A\\
&\text{via 1 side: } A_1 (= A)\\
&\quad\Downarrow \times A\\
&\text{via 2 sides: } A_2\\
&\quad\Downarrow \times A\\
&\text{via 3 sides: } A_3\\
&\quad\Downarrow \times A\\
&\quad\cdots\\
&\quad\Downarrow \times A\\
&\text{via k sides: } A_k
\end{aligned}
$$

对于每组点(i,j)，经过k条边$i\to j$的最短路为$A_k(i,j)$。

代码：

```cpp
#include <iostream>
#include <cstring>

using namespace std;

const int maxn = 205;

int n = 0, k, m, s, t, d[maxn][maxn], a[maxn][maxn], map[1005];

```

```cpp
void mul(int a[maxn][maxn], int b[maxn][maxn]) {
    int t[maxn][maxn];
    for (int i = 0; i < n; i ++) {
        for (int j = 0; j < n;j ++) {
            t[i][j] = 1e9;
            for (int k = 0; k < n; k ++) {
                if (t[i][j] > a[i][k] + b[k][j])
                    t[i][j] = a[i][k] + b[k][j];
            }
        }
    }
    for (int i = 0; i < n; i ++)
        for (int j = 0; j < n; j ++)
            a[i][j] = t[i][j];
}

int main() {
    cin >> k >> m >> s >> t;
    memset(d, 0x3f, sizeof(d));
    memset(a, 0x3f, sizeof(a));
    memset(map, 0xff, sizeof(map)); // -1


    // 由于数据关系，需要对数据进行过映射
    while (m --> 0) {
        int l, u, v; cin >> l >> u >> v;
        if (map[u] == -1) map[u] = n ++;
        if (map[v] == -1) map[v] = n ++;
        d[map[u]][map[v]] = d[map[v]][map[u]] = l;
    }

    // a为图的单位矩阵，即A_0，只有对角线是0，其他都是无穷大
    for (int i = 0; i < n; i ++) a[i][i] = 0;

    // 快速幂
    while (k > 0) {
        if (k & 1) mul(a, d);
        mul(d, d);
        k /= 2;
    }
    cout << a[map[s]][map[t]] << endl;
    return 0;
}
```