
ACM 代码册

EDITED BY

PUFANYI

目录

第一章 搜索	1
1.1 Dancing Links	1
1.2 $\alpha - \beta$ 剪枝	3
第二章 字符串	5
2.1 KMP	5
2.2 Z-function	5
2.3 AC 自动机	6
第三章 数学	9
3.1 快速幂	9
3.2 位运算	9
3.2.1 Gray 码	9
3.3 数论	9
3.3.1 最大公约数	9
3.3.2 欧几里得算法	9
3.3.3 筛法	10
3.3.4 EXCRT	10
3.3.5 Lucas	11
3.3.6 Pollard-Rho	11
第四章 数据结构	15
4.1 动态树	15
4.1.1 Link-Cut Tree	15
第五章 图论	19
5.1 生成树	19
5.1.1 矩阵树	19
5.1.2 最小生成树计数	20
5.2 网络流	24
5.2.1 最大流	24

第六章 其他	31
6.1 读入输出优化	31

第一章 搜索

1.1 Dancing Links

```
1  const int maxn = 505;
2  const int maxm = 6005;
3
4  struct Dancing_Links {
5      int n, m, total, ans;
6
7      struct Node {
8          int up, down, left, right, row, column;
9      } no[maxm];
10
11     int siz[maxn];
12     int first[maxn];
13     int stk[maxn];
14
15     void init(int n, int m) {
16         ans = 0;
17         this->n = n, this->m = m;
18         memset(first, 0, sizeof(first));
19         memset(siz, 0, sizeof(siz));
20         for (int i = 0; i <= m; ++i) {
21             no[i].left = i - 1, no[i].right = i + 1;
22             no[i].up = no[i].down = i;
23         }
24         no[0].left = m, no[m].right = 0, total = m;
25     }
26
27     void insert(int row, int col) {
28         total++, siz[col]++;
29         no[total].row = row, no[total].column = col;
30         no[total].down = col, no[total].up = no[col].up;
31         no[col].up = total, no[no[total].up].down = total;
32         if (!first[row]) {
33             first[row] = no[total].left = no[total].right = total;
34         } else {
35             no[total].right = first[row], no[total].left = no[first[row]].left;
```

```

36     no[no[total].left].right = no[first[row]].left = total;
37 }
38 }
39
40 void remove(int col) {
41     no[no[col].left].right = no[col].right;
42     no[no[col].right].left = no[col].left;
43     for (int i = no[col].down; i != col; i = no[i].down) {
44         for (int j = no[i].right; j != i; j = no[j].right) {
45             no[no[j].up].down = no[j].down;
46             no[no[j].down].up = no[j].up;
47             siz[no[j].column]--;
48         }
49     }
50 }
51
52 void recover(int col) {
53     for (int i = no[col].up; i != col; i = no[i].up) {
54         for (int j = no[i].left; j != i; j = no[j].left) {
55             no[no[j].up].down = no[no[j].down].up = j;
56             siz[no[j].column]++;
57         }
58     }
59     no[no[col].left].right = no[no[col].right].left = col;
60 }
61
62 bool dance(int dep) {
63     if (!no[0].right) {
64         ans = dep - 1;
65         return true;
66     }
67     int col = no[0].right;
68     for (int i = no[0].right; i; i = no[i].right) {
69         if (siz[i] < siz[col]) {
70             col = i;
71         }
72     }
73     remove(col);
74     for (int i = no[col].down; i != col; i = no[i].down) {
75         stk[dep] = no[i].row;
76         for (int j = no[i].right; j != i; j = no[j].right) {
77             remove(no[j].column);
78         }
79         if (dance(dep + 1)) {
80             return true;
81         }
82         for (int j = no[i].left; j != i; j = no[j].left) {

```

```
83         recover(no[j].column);
84     }
85 }
86     recover(col);
87     return false;
88 }
89 } dlx;
90
91 int main() {
92     int n, m, x;
93     read(n), read(m);
94     dlx.init(n, m);
95     for (int i = 1; i <= n; ++i) {
96         for (int j = 1; j <= m; ++j) {
97             if (read(x) && x) {
98                 dlx.insert(i, j);
99             }
100         }
101     }
102     if (dlx.dance(1)) {
103         for (int i = 1; i <= dlx.ans; ++i) {
104             writesp(dlx.stk[i]);
105         }
106         puts("");
107     } else {
108         puts("No Solution!");
109     }
110     return 0;
111 }
```

1.2 $\alpha - \beta$ 剪枝

第二章 字符串

2.1 KMP

```
1 std::vector<int> kmp(std::string s) {
2     int n = s.length();
3     std::vector<int> pi(n);
4     for (int i = 1; i < n; ++i) {
5         int j = pi[i - 1];
6         while (j && s[i] != s[j]) {
7             j = pi[j - 1];
8         }
9         if (s[i] == s[j]) {
10             j++;
11         }
12         pi[i] = j;
13     }
14     return pi;
15 }
```

2.2 Z-function

```
1 std::vector<int> z_function(std::string s) {
2     int n = s.length();
3     std::vector<int> z(n);
4     z[0] = n;
5     for (int i = 1, l = 0, r = 0; i < n; ++i) {
6         if (i <= r && z[i - l] < r - i + 1) {
7             z[i] = z[i - l];
8         } else {
9             z[i] = std::max(0, r - i + 1);
10            while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
11                z[i]++;
12            }
13        }
14        if (i + z[i] - 1 > r) {
15            l = i, r = i + z[i] - 1;
16        }
17    }
18 }
```

```
17 | }
18 |     return z;
19 | }
```

2.3 AC 自动机

```
1 | const int maxn = 200005;
2 |
3 | int ans[maxn];
4 |
5 | struct Aho_Corasick {
6 |     std::vector<int> id[maxn];
7 |     int son[maxn][26];
8 |     int fail[maxn];
9 |     int val[maxn];
10 |    int cnt;
11 |
12 |    Aho_Corasick() {
13 |        cnt = 0;
14 |        memset(son, 0, sizeof(son));
15 |        memset(fail, 0, sizeof(fail));
16 |        memset(val, 0, sizeof(val));
17 |    }
18 |
19 |    void insert(std::string s, int _id) {
20 |        int now = 0;
21 |        for (auto c : s) {
22 |            const int x = c - 'a';
23 |            if (!son[now][x]) {
24 |                son[now][x] = ++cnt;
25 |            }
26 |            now = son[now][x];
27 |        }
28 |        id[now].push_back(_id);
29 |    }
30 |
31 |    std::vector<int> fas[maxn];
32 |
33 |    void build() {
34 |        std::queue<int> q;
35 |        for (int i = 0; i < 26; ++i) {
36 |            if (son[0][i]) {
37 |                q.push(son[0][i]);
38 |            }
39 |        }
40 |        while (!q.empty()) {
```

```
41     int now = q.front();
42     q.pop();
43     for (int i = 0; i < 26; ++i) {
44         if (son[now][i]) {
45             fail[son[now][i]] = son[fail[now]][i];
46             q.push(son[now][i]);
47         } else {
48             son[now][i] = son[fail[now]][i];
49         }
50     }
51 }
52 }
53
54 void getval(std::string s) {
55     int now = 0;
56     for (auto c : s) {
57         now = son[now][c - 'a'];
58         val[now]++;
59     }
60 }
61
62 void build_fail_tree() {
63     for (int i = 1; i <= cnt; ++i) {
64         fas[fail[i]].push_back(i);
65     }
66 }
67
68 void dfs(int now = 0) {
69     for (auto x : fas[now]) {
70         dfs(x);
71         val[now] += val[x];
72     }
73     if (!id[now].empty()) {
74         for (auto x : id[now]) {
75             ans[x] = val[now];
76         }
77     }
78 }
79 };
80
81 Aho_Corasick ac;
82
83 int n;
84
85 int main() {
86     std::cin >> n;
87     for (int i = 1; i <= n; ++i) {
```

```
88     std::string s;
89     std::cin >> s;
90     ac.insert(s, i);
91 }
92 ac.build();
93 std::string s;
94 std::cin >> s;
95 ac.getval(s);
96 ac.build_fail_tree();
97 ac.dfs();
98 for (int i = 1; i <= n; ++i) {
99     std::cout << ans[i] << std::endl;
100 }
101 return 0;
102 }
```

第三章 数学

3.1 快速幂

```
1 template <class T>
2 T ksm(T a, T b, T mod) {
3     T ans = 1;
4     for (; b >>= 1, a = (LL) a * a % mod) {
5         if (b & 1) {
6             ans = (LL) ans * a % mod;
7         }
8     }
9     return ans;
10 }
```

3.2 位运算

3.2.1 Gray 码

```
1 int g(int n) {
2     return n ^ (n >> 1);
3 }
4
5 int rev_g(int g) {
6     int n = 0;
7     for (; g; g >>= 1) {
8         n ^= g;
9     }
10    return n;
11 }
```

3.3 数论

3.3.1 最大公约数

3.3.2 欧几里得算法

```
1 template <class T>
2 T gcd(T a, T b) {
3     while (b) {
4         int t = a % b;
5         a = b;
6         b = t;
7     }
8     return a;
9 }
```

3.3.3 筛法

Eratosthenes 筛法

Euler 筛法

```
1 void Euler(const int n = 100000) {
2     np[1] = true;
3     int cnt = 0;
4     for (int i = 2; i <= n; ++i) {
5         if (!np[i]) {
6             prime[++cnt] = i;
7         }
8         for (int j = 1; j <= cnt && (LL) i * prime[j] <= n; ++j) {
9             np[i * prime[j]] = true;
10            if (!(i % prime[j])) {
11                break;
12            }
13        }
14    }
15 }
```

3.3.4 EXCRT

```
1 LL CRT(int k, LL* a, LL* r) {
2     LL n = 1, ans = 0;
3     for (int i = 1; i <= k; i++) n = n * r[i];
4     for (int i = 1; i <= k; i++) {
5         LL m = n / r[i], b, y;
6         exgcd(m, r[i], b, y); //  $b * m \bmod r[i] = 1$ 
7         ans = (ans + a[i] * m * b % mod) % mod;
8     }
9     return (ans % mod + mod) % mod;
10 }
```

3.3.5 Lucas

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

3.3.6 Pollard-Rho

```

1  typedef unsigned long long ULL;
2  typedef long long LL;
3
4  std::set<int> ans;
5
6  inline ULL rnd() {
7      static ULL seed = 2333;
8      seed ^= seed << 40;
9      seed ^= seed >> 23;
10     seed ^= seed << 7;
11     return seed;
12 }
13
14 template <typename T>
15 inline T gcd(T a, T b) {
16     while (b) {
17         T t = a % b;
18         a = b;
19         b = t;
20     }
21     return a < 0 ? -a : a;
22 }
23
24 template <typename T>
25 inline void add(T& x, T y, T mod) {
26     x += y;
27     if (x >= mod) {
28         x -= mod;
29     } else if (x < 0) {
30         x += mod;
31     }
32 }
33
34 inline LL cheng(LL a, LL b, LL mod) {
35     LL tmp = ((long double) a * b + .5) / mod;
36     return ((a * b - tmp * mod) % mod + mod) % mod;
37 }
38
39 inline LL ksm(LL a, LL b, LL mod) {
40     LL ans = 1;

```

```
41     for (; b; b >>= 1, a = cheng(a, a, mod)) {
42         if (b & 1) {
43             ans = cheng(ans, a, mod);
44         }
45     }
46     return ans;
47 }
48
49 inline bool witness(LL a, LL n) {
50     LL u = n - 1;
51     int t = 0;
52     while (!(u & 1)) {
53         u >>= 1;
54         t++;
55     }
56     LL x = ksm(a, u, n);
57     for (int i = 1; i <= t; ++i) {
58         LL lstx = x;
59         x = cheng(x, x, n);
60         if (x == 1 && lstx != 1 && lstx != n - 1) {
61             return false;
62         }
63     }
64     if (x != 1) {
65         return false;
66     }
67     return true;
68 }
69
70 inline bool MR(LL n) {
71     if (n == 2) {
72         return true;
73     }
74     static const int s = 5;
75     for (int i = 1; i <= s; ++i) {
76         if (!witness(rnd() % (n - 1) + 1, n)) {
77             return false;
78         }
79     }
80     return true;
81 }
82
83 inline LL rho(LL n) {
84     if (MR(n)) {
85         return n;
86     }
87     LL x = rnd() % n;
```



```
88     LL y = x;
89     LL p = (n & 1) ? 1 : 2;
90     while (p == 1) {
91         LL cc = rnd() % n;
92         while (true) {
93             int bitt = 127;
94             LL xx = 1;
95             while (bitt--) {
96                 x = cheng(x, x, n);
97                 add(x, cc, n);
98                 y = cheng(y, y, n);
99                 add(y, cc, n);
100                y = cheng(y, y, n);
101                add(y, cc, n);
102                if (x == y) {
103                    break;
104                }
105                LL tx = (__int128) xx * (y - x) % n;
106                if (tx) {
107                    xx = tx;
108                } else {
109                    break;
110                }
111            }
112            LL d = gcd((LL) xx, n);
113            if (d != 1 && d != n) {
114                p = d;
115                break;
116            }
117            if (x == y) {
118                break;
119            }
120        }
121    }
122    return std::max(rho(p), rho(n / p));
123 }
124
125 inline void solve() {
126     LL n;
127     read(n);
128     if (MR(n)) {
129         puts("Prime");
130     } else {
131         writeln(rho(n));
132     }
133 }
```


第四章 数据结构

4.1 动态树

4.1.1 Link-Cut Tree

```
1  #include <cstdio>
2  #include <iostream>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int maxn = 300005;
8
9  class LCT {
10     // node
11
12     public:
13         int sum[maxn], val[maxn];
14         int s[maxn][2], fa[maxn];
15
16     private:
17         bool lzy_fan[maxn];
18
19         void push_up(int x) {
20             sum[x] = val[x] ^ sum[s[x][0]] ^ sum[s[x][1]];
21         }
22
23         bool nrt(int x) {
24             return s[fa[x]][0] == x || s[fa[x]][1] == x;
25         }
26
27         void fan(int x) {
28             swap(s[x][0], s[x][1]);
29             lzy_fan[x] ^= 1;
30         }
31
32         void push_down(int x) {
33             if (lzy_fan[x]) {
```

```

34     if (s[x][0]) {
35         fan(s[x][0]);
36     }
37     if (s[x][1]) {
38         fan(s[x][1]);
39     }
40     lzy_fan[x] = 0;
41 }
42 }
43
44 // splay
45 private:
46 void rotate(int x) {
47     int y = fa[x], z = fa[y];
48     int k = (s[y][1] == x), ss = s[x][!k];
49     if (nrt(y)) {
50         s[z][s[z][1] == y] = x;
51     }
52     fa[x] = z;
53     s[x][!k] = y;
54     fa[y] = x;
55     s[y][k] = ss;
56     if (ss) {
57         fa[ss] = y;
58     }
59     push_up(y);
60     push_up(x);
61 }
62
63 int sta[maxn];
64 void splay(int x) {
65     int K = x, top = 0;
66     sta[++top] = K;
67     while (nrt(K)) {
68         sta[++top] = K = fa[K];
69     }
70     while (top) {
71         push_down(sta[top--]);
72     }
73     while (nrt(x)) {
74         int y = fa[x], z = fa[y];
75         if (nrt(y)) {
76             rotate(((s[y][0] == x) ^ (s[z][0] == y)) ? x : y);
77         }
78         rotate(x);
79     }
80 }

```

```
81
82 // LCT
83 private:
84 void access(int x) {
85     for (int y = 0; x; x = fa[y = x]) {
86         splay(x);
87         s[x][1] = y;
88         push_up(x);
89     }
90 }
91
92 void make_root(int x) {
93     access(x);
94     splay(x);
95     fan(x);
96 }
97
98 int find_root(int x) {
99     access(x);
100    splay(x);
101    while (s[x][0]) {
102        push_down(x);
103        x = s[x][0];
104    }
105    splay(x);
106    return x;
107 }
108
109 void split(int x, int y) {
110     make_root(x);
111     access(y);
112     splay(y);
113 }
114
115 public:
116 void link(int x, int y) {
117     make_root(x);
118     if (find_root(y) != x) {
119         fa[x] = y;
120     }
121 }
122
123 void cut(int x, int y) {
124     make_root(x);
125     if (find_root(y) == x && fa[y] == x && !s[y][0]) {
126         fa[y] = s[x][1] = 0;
127         push_up(x);
```

```
128     }
129 }
130
131 void change(int x, int y) {
132     splay(x);
133     val[x] = y;
134     push_up(x);
135 }
136
137 int ask(int x, int y) {
138     split(x, y);
139     return sum[y];
140 }
141 } tr;
142
143 int main() {
144     int n, m;
145     scanf("%d%d", &n, &m);
146     for (int i = 1; i <= n; ++i) {
147         scanf("%d", &tr.val[i]);
148         tr.sum[i] = tr.val[i];
149     }
150     while (m--) {
151         int cmd, x, y;
152         scanf("%d%d%d", &cmd, &x, &y);
153         switch (cmd) {
154             case 0:
155                 printf("%d\n", tr.ask(x, y));
156                 break;
157             case 1:
158                 tr.link(x, y);
159                 break;
160             case 2:
161                 tr.cut(x, y);
162                 break;
163             case 3:
164                 tr.change(x, y);
165         }
166     }
167     return 0;
168 }
```

第五章 图论

5.1 生成树

5.1.1 矩阵树

假设给出图为 G ，定义一个 $n \times n$ 的矩阵 $D(G)$ 表示 G 个点的度数，当 $i \neq j$ 时， $d_{i,j} = 0$ ，当 $i = j$ 时， $d_{i,j}$ 等于节点 i 的度数。再定义一个 $n \times n$ 的矩阵 A_G 表示 G 的邻接矩阵， $A_{i,j}$ 表示 i 到 j 的边数。然后我们定义基尔霍夫矩阵 $C(G) = D(G) - A(G)$ 。则 G 中生成树个数等于 $C(G)$ 中任意一个 $n-1$ 阶主子式的行列式的绝对值。所谓一个矩阵 M 的 $n-1$ 阶主子式就是对于两个整数 $r (1 \leq r \leq n)$ ，将 M 去掉第 r 行和第 r 列后形成的 $n-1$ 阶的矩阵，记作 M_r 。

```
1  const int maxn = 13;
2
3  int n, m;
4
5  struct Matrix {
6      double mt[maxn][maxn];
7
8      inline double* operator [] (int x) {
9          return mt[x];
10     }
11
12     inline void clear() {
13         for (int i = 1; i <= n; ++i) {
14             for (int j = 1; j <= n; ++j) {
15                 mt[i][j] = 0;
16             }
17         }
18     }
19
20     inline double getans() {
21         int nn = n - 1;
22         double ans = 1.;
23         for (int i = 1; i <= nn; ++i) {
24             int mx = i;
25             for (int j = i + 1; j <= nn; ++j) {
26                 if (mt[mx][i] < mt[j][i]) {
27                     mx = j;
```

```

28     }
29 }
30 if (i != mx) {
31     ans *= -1;
32     for (int j = i; j <= nn; ++j) {
33         std::swap(mt[mx][j], mt[i][j]);
34     }
35 }
36 if (mt[i][i] < 1e-10) {
37     return 0.;
38 }
39 for (int j = i + 1; j <= nn; ++j) {
40     double kk = mt[j][i] / mt[i][i];
41     for (int k = i; k <= nn; ++k) {
42         mt[j][k] -= kk * mt[i][k];
43     }
44 }
45 }
46 for (int i = 1; i <= nn; ++i) {
47     ans *= mt[i][i];
48 }
49 return ans;
50 }
51 } Kif;
52
53 void solve() {
54     read(n), read(m);
55     Kif.clear();
56     for (int i = 1, u, v; i <= m; ++i) {
57         read(u), read(v);
58         Kif[u][u]++, Kif[v][v]++;
59         Kif[u][v]--, Kif[v][u]--;
60     }
61     printf("%.0f\n", Kif.getans());
62 }

```

5.1.2 最小生成树计数

然后是最小生成树计数。这个大概就是发现每个最小生成树每种边权的边数应该是一样的，且将这些边去掉后所得的连通块相同。

于是我们考虑建出一棵最小生成树，枚举边权然后把原来最小生成树上该边权的边删掉，然后跑矩阵树。

复杂度？假设离散之后边权 i 共有 a_i 条边，那么显然 $\sum a_i = m$ 。如果图没有重边，则 Kruskal 复杂度 $\mathcal{O}(m \log m)$ ，矩阵树复杂度为 $\mathcal{O}(\sum (n + m + \min(n, a_i)^3))$ ，由于没有重边，前面的 $n + m$ 那一项卡满不过 $\mathcal{O}(m \times (n + m)) = \mathcal{O}(m^2) = \mathcal{O}(n^2 m)$ ，而后面那一项当每个 a_i 取

到 n 时最大, 即 $\mathcal{O}\left(\frac{m}{n} \times n^3\right) = \mathcal{O}(n^2m)$, 所以总复杂度 $\mathcal{O}(n^2m)$ 。

```

1  const int maxn = 105;
2  const int maxm = 1005;
3  const int mod = 31011;
4
5  int n, m;
6
7  struct Edge {
8      int u, v, d;
9
10     friend bool operator < (const Edge& a, const Edge& b) {
11         return a.d < b.d;
12     }
13 } e[maxm];
14
15 std::vector<std::pair<int, int>> v[maxn];
16
17 int col[maxn];
18
19 int fa[maxn];
20
21 inline int getfa(int x) {
22     return fa[x] == x ? x : fa[x] = getfa(fa[x]);
23 }
24
25 inline void dfs(int now, int ccol, int bx) {
26     col[now] = ccol;
27     for (auto to : v[now]) {
28         if (!col[to.first] && to.second != bx) {
29             dfs(to.first, ccol, bx);
30         }
31     }
32 }
33
34 struct Matrix {
35     int mt[maxn][maxn];
36
37     inline void init(int n) {
38         for (int i = 1; i <= n; ++i) {
39             for (int j = 1; j <= n; ++j) {
40                 mt[i][j] = 0;
41             }
42         }
43     }
44
45     inline int* operator [] (int x) {
46         return mt[x];

```

```
47 }
48
49 inline int solve(int n) {
50     n--;
51     if (!n) {
52         return 1;
53     }
54     int ans = 1;
55     for (int i = 1; i <= n; ++i) {
56         int now = 0;
57         for (int j = i; j <= n; ++j) {
58             if (mt[j][i]) {
59                 now = i;
60                 break;
61             }
62         }
63         if (!now) {
64             return 0;
65         } else if (now != i) {
66             for (int j = i; j <= n; ++j) {
67                 std::swap(mt[i][j], mt[now][j]);
68             }
69             ans *= -1;
70         }
71         for (int j = i + 1; j <= n; ++j) {
72             while (mt[j][i]) {
73                 int nowk = mt[i][i] / mt[j][i];
74                 for (int k = i; k <= n; ++k) {
75                     mt[i][k] -= mt[j][k] * nowk % mod;
76                     if (mt[i][k] < 0) {
77                         mt[i][k] += mod;
78                     } else if (mt[i][k] >= mod) {
79                         mt[i][k] -= mod;
80                     }
81                     std::swap(mt[i][k], mt[j][k]);
82                 }
83                 ans *= -1;
84             }
85         }
86     }
87     for (int i = 1; i <= n; ++i) {
88         (ans *= mt[i][i]) %= mod;
89     }
90     if (ans <= mod) {
91         ans += mod;
92     }
93     return ans;
```

```

94     }
95 } mat;
96
97 inline int Main() {
98     read(n), read(m);
99     for (int i = 1; i <= m; ++i) {
100         read(e[i].u), read(e[i].v), read(e[i].d);
101     }
102     std::sort(e + 1, e + m + 1);
103     int cnt = 0, now = 0;
104     for (int i = 1; i <= m; ++i) {
105         if (now < e[i].d) {
106             now = e[i].d;
107             cnt++;
108         }
109         e[i].d = cnt;
110     }
111     for (int i = 1; i <= n; ++i) {
112         fa[i] = i;
113     }
114     for (int i = 1; i <= m; ++i) {
115         int fax = getfa(e[i].u);
116         int fay = getfa(e[i].v);
117         if (fax != fay) {
118             fa[fax] = fay;
119             v[e[i].u].emplace_back(e[i].v, e[i].d);
120             v[e[i].v].emplace_back(e[i].u, e[i].d);
121         }
122     }
123     int ans = 1;
124     for (int i = 1; i <= cnt; ++i) {
125         memset(col, 0, sizeof(col));
126         int cntt = 0;
127         for (int j = 1; j <= n; ++j) {
128             if (!col[j]) {
129                 dfs(j, ++cntt, i);
130             }
131         }
132         mat.init(cntt);
133         for (int j = 1; j <= m; ++j) {
134             if (e[j].d == i && col[e[j].u] != col[e[j].v]) {
135                 mat[col[e[j].u]][col[e[j].v]]--;
136                 mat[col[e[j].v]][col[e[j].u]]--;
137                 mat[col[e[j].u]][col[e[j].u]]++;
138                 mat[col[e[j].v]][col[e[j].v]]++;
139             }
140         }

```

```

141     (ans *= mat.solve(cntt)) %= mod;
142 }
143 writeln(ans);
144 return 0;
145 }

```

5.2 网络流

5.2.1 最大流

```

1  #include <cstdio>
2  #include <cstring>
3  #include <queue>
4
5  using namespace std;
6
7  #define fill(_a) memset(_a, 0x3f, sizeof(_a))
8  #define clr(_a) memset(_a, 0, sizeof(_a))
9  #define copy(_a, _b) memcpy(_a, _b, sizeof(_b))
10
11 const int inf = 0x3f3f3f3f;
12 const int maxn = 10005;
13 const int maxm = 100005;
14
15 int n, m, s, t;
16
17 struct EDGE
18 {
19     int to, nxt;
20     int dist;
21 } e[maxm<<1];
22
23 int first[maxn];
24 int tmp[maxn];
25 int _cnt = -1;
26
27 inline void init()
28 {
29     _cnt = -1;
30     memset(first, 0xff, sizeof(first));
31 }
32
33 inline void add_edge(int f, int t, int dist)
34 {
35     e[++_cnt].to = t;
36     e[_cnt].nxt = first[f];

```

```
37     first[f] = _cnt;
38     e[_cnt].dist = dist;
39     e[++_cnt].to = f;
40     e[_cnt].nxt = first[t];
41     first[t] = _cnt;
42     e[_cnt].dist = 0;
43 }
44
45 int deep[maxn];
46
47 inline bool bfs(int s, int t)
48 {
49     copy(first, tmp);
50     int now = s;
51     queue<int> q;
52     q.push(now);
53     fill(deep);
54     deep[s] = 0;
55     while(!q.empty())
56     {
57         now = q.front();
58         q.pop();
59         for(int i = first[now]; ~i; i = e[i].nxt)
60         {
61             if(e[i].dist && deep[e[i].to] >= inf)
62             {
63                 deep[e[i].to] = deep[now] + 1;
64                 q.push(e[i].to);
65             }
66         }
67     }
68     return deep[t] < inf;
69 }
70
71 inline int dfs(int now, int t, int limit)
72 {
73     if(!limit || now == t)
74         return limit;
75     int flow = 0, f;
76     for(int i = first[now]; ~i; i = e[i].nxt)
77     {
78         first[now] = i;
79         if(deep[e[i].to] == deep[now]+1 && (f = dfs(e[i].to, t, min(limit, e[i].dist)
80             )))
81         {
82             flow += f;
83             limit -= f;
```

```

83     e[i].dist -= f;
84     e[i^1].dist += f;
85     if(!limit)
86         break;
87     }
88 }
89 return flow;
90 }
91
92 inline int Dinic(int s, int t)
93 {
94     int maxflow = 0;
95     while(bfs(s, t))
96         maxflow += dfs(s, t, inf);
97     return maxflow;
98 }
99
100 int main()
101 {
102     scanf("%d%d%d", &n, &m, &s, &t);
103     init();
104     for(int i = 1, _f, _t, d; i <= m; ++i)
105     {
106         scanf("%d%d", &_f, &_t, &d);
107         add_edge(_f, _t, d);
108     }
109     copy(tmp, first);
110     printf("%d", Dinic(s, t));
111     return 0;
112 }

```

```

1  #include <cstdio>
2  #include <cstring>
3  #include <queue>
4
5  using namespace std;
6
7  #define fill(_a) memset(_a, 0x3f, sizeof(_a))
8  #define clr(_a) memset(_a, 0, sizeof(_a))
9  #define copy(_a, _b) memcpy(_a, _b, sizeof(_b))
10
11 const int inf = 0x3f3f3f3f;
12 const int maxn = 10005;
13 const int maxm = 100005;
14
15 int n, m, s, t;
16

```

```
17 struct EDGE
18 {
19     int to, nxt;
20     int dist;
21 } e[maxm<<1];
22
23 int first[maxn];
24 int tmp[maxn];
25 int _cnt = -1;
26
27 inline void init()
28 {
29     _cnt = -1;
30     memset(first, 0xff, sizeof(first));
31 }
32
33 inline void add_edge(int f, int t, int dist)
34 {
35     e[++_cnt].to = t;
36     e[_cnt].nxt = first[f];
37     first[f] = _cnt;
38     e[_cnt].dist = dist;
39     e[++_cnt].to = f;
40     e[_cnt].nxt = first[t];
41     first[t] = _cnt;
42     e[_cnt].dist = 0;
43 }
44
45 int deep[maxn];
46
47 inline bool bfs(int s, int t)
48 {
49     copy(first, tmp);
50     int now = s;
51     queue<int> q;
52     q.push(now);
53     fill(deep);
54     deep[s] = 0;
55     while(!q.empty())
56     {
57         now = q.front();
58         q.pop();
59         for(int i = first[now]; ~i; i = e[i].nxt)
60         {
61             if(e[i].dist && deep[e[i].to] >= inf)
62             {
63                 deep[e[i].to] = deep[now] + 1;
```

```

64         q.push(e[i].to);
65     }
66 }
67 }
68 return deep[t] < inf;
69 }
70
71 inline int dfs(int now, int t, int limit)
72 {
73     if(!limit || now == t)
74         return limit;
75     int flow = 0, f;
76     for(int i = first[now]; ~i; i = e[i].nxt)
77     {
78         first[now] = i;
79         if(deep[e[i].to] == deep[now]+1 && (f = dfs(e[i].to, t, min(limit, e[i].dist)
80             )))
81         {
82             flow += f;
83             limit -= f;
84             e[i].dist -= f;
85             e[i^1].dist += f;
86             if(!limit)
87                 break;
88         }
89     }
90     return flow;
91 }
92
93 inline int Dinic(int s, int t)
94 {
95     int maxflow = 0;
96     while(bfs(s, t))
97         maxflow += dfs(s, t, inf);
98     return maxflow;
99 }
100
101 int main()
102 {
103     scanf("%d%d%d", &n, &m, &s, &t);
104     init();
105     for(int i = 1, _f, _t, d; i <= m; ++i)
106     {
107         scanf("%d%d", &_f, &_t, &d);
108         add_edge(_f, _t, d);
109     }
110     copy(tmp, first);

```



```
110 | printf("%d", Dinic(s, t));  
111 | return 0;  
112 | }
```


第六章 其他

6.1 读入输出优化

```
1 inline char gc() {
2     static const int L = 23333;
3     static char sxd[L], *sss = sxd, *ttt = sxd;
4     if (sss == ttt) {
5         ttt = (sss = sxd) + fread(sxd, 1, L, stdin);
6         if (sss == ttt) {
7             return EOF;
8         }
9     }
10    return *sss++;
11 }
12
13 #ifndef Debug
14 #define dd c = getchar()
15 #else
16 #define dd c = gc()
17 #endif
18 template <class T>
19 inline bool read(T& x) {
20     x = 0;
21     char dd;
22     bool flg = false;
23     for (; !isdigit(c); dd) {
24         if (c == '-') {
25             flg = true;
26         } else if (c == EOF) {
27             return false;
28         }
29     }
30     for (; isdigit(c); dd) {
31         x = (x * 10) + (c ^ 48);
32     }
33     if (flg) {
34         x = -x;
35     }
```

```
36     return true;
37 }
38 #undef dd
39
40 template <class T>
41 inline void write(T x) {
42     if (x < 0) {
43         x = -x;
44         putchar('-');
45     }
46     if (x > 9) {
47         write(x / 10);
48         x %= 10;
49     }
50     putchar(x | 48);
51 }
52
53 template <class T>
54 inline void writeln(T x) {
55     write(x);
56     puts("");
57 }
58
59 template <class T>
60 inline void writesp(T x) {
61     write(x);
62     putchar(' ');
63 }
```