

Nanyang Technological University
Joker
Reference Book



Contents

1 String	3	4.2 Catalan Numbers	9
1.1 KMP	3	4.3 Bell or Exponential Numbers	10
1.2 Z-function	3	4.4 Bell or Exponential Numbers	10
1.3 Aho-Corasick algorithm	3	4.5 Lucas numbers	10
1.4 Manachar	4	4.6 Derangement	10
1.5 SuffixArray	5	4.7 Prufer	10
2 Number Theory	5	5 Data Structures	10
2.1 Extended Euclidean Algorithm	5	5.1 Link-Cut Tree	10
2.2 Miller-Rabin primality test	5	6 Graph Theory	12
2.3 Sieve of Euler	6	6.1 矩阵树	12
2.4 Euler's Totient Function	6	6.2 最小生成树计数	13
2.5 Euler's theorem	6	7 Network flow	15
2.6 Lucas	7	7.1 Maximum Flow Problem	15
2.7 Chinese Remainder Theorem	7	7.2 Minimum-Cost Flow Problem	16
2.8 Wilson's theorem	7	7.3 无源汇上下界可行流	17
2.9 Baby-Step Giant-Step	7	7.4 有源汇上下界可行流	17
2.10 Pollard-Rho	7	7.5 有源汇上下界最大流	17
3 Number-Theoretic Transform	9	7.6 有源汇上下界最小流	18
4 OEIS	9	8 Others	18
4.1 Fibonacci Numbers	9	8.1 vim	18

1 String

1.1 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 }

```

1.2 Z-function

```

16 std::vector<int> z_function(std::string s) {
17     int n = s.length();
18     std::vector<int> z(n);
19     z[0] = n;
20     for (int i = 1, l = 0, r = 0; i < n; ++i) {
21         if (i <= r && z[i - l] < r - i + 1) {
22             z[i] = z[i - l];
23         } else {
24             z[i] = std::max(0, r - i + 1);
25             while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
26                 z[i]++;
27             }
28         }
29         if (i + z[i] - 1 > r) {
30             l = i, r = i + z[i] - 1;
31         }
32     }
33     return z;
34 }

```

1.3 Aho-Corasick algorithm

```

const int maxn = 200005;

int ans[maxn];

struct Aho_Corasick {
    std::vector<int> id[maxn];
    int son[maxn][26];
    int fail[maxn];
    int val[maxn];
    int cnt;

    Aho_Corasick() {
        cnt = 0;
        memset(son, 0, sizeof(son));
        memset(fail, 0, sizeof(fail));
        memset(val, 0, sizeof(val));
    }

    void insert(std::string s, int _id) {
        int now = 0;
        for (auto c : s) {
            const int x = c - 'a';
            if (!son[now][x]) {
                son[now][x] = ++cnt;
            }
            now = son[now][x];
        }
        id[now].push_back(_id);
    }

    std::vector<int> fas[maxn];

    void build() {
        std::queue<int> q;
        for (int i = 0; i < 26; ++i) {
            if (son[0][i]) {
                q.push(son[0][i]);
            }
        }
        while (!q.empty()) {
            int now = q.front();
            q.pop();

```

35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76

```

77     for (int i = 0; i < 26; ++i) {
78         if (son[now][i]) {
79             fail[son[now][i]] = son[fail[now]][i];
80             q.push(son[now][i]);
81         } else {
82             son[now][i] = son[fail[now]][i];
83         }
84     }
85 }
86 }
87
88 void getval(std::string s) {
89     int now = 0;
90     for (auto c : s) {
91         now = son[now][c - 'a'];
92         val[now]++;
93     }
94 }
95
96 void build_fail_tree() {
97     for (int i = 1; i <= cnt; ++i) {
98         fas[fail[i]].push_back(i);
99     }
100 }
101
102 void dfs(int now = 0) {
103     for (auto x : fas[now]) {
104         dfs(x);
105         val[now] += val[x];
106     }
107     if (!id[now].empty()) {
108         for (auto x : id[now]) {
109             ans[x] = val[now];
110         }
111     }
112 }
113 };
114
115 Aho_Corasick ac;
116
117 int n;
118
119 int main() {
120     std::cin >> n;

```

```

    for (int i = 1; i <= n; ++i) {
        std::string s;
        std::cin >> s;
        ac.insert(s, i);
    }
    ac.build();
    std::string s;
    std::cin >> s;
    ac.getval(s);
    ac.build_fail_tree();
    ac.dfs();
    for (int i = 1; i <= n; ++i) {
        std::cout << ans[i] << std::endl;
    }
    return 0;
}

```

```

121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136

```

1.4 Manacher

```

int manacher() {
    int i, p, ans = 0;
    r[1] = 0, p = 1;
    for (i = 2; i <= n; ++i) {
        if (i <= p + r[p]) {
            r[i] = min(r[2 * p - i], p + r[p] - i);
        } else {
            r[i] = 1;
        }
        while (st[i - r[i]] == st[i + r[i]]) {
            ++r[i];
        }
        --r[i];
        if (i + r[i] > p + r[p]) {
            p = i;
        }
        ans = max(ans, r[i]);
    }
    return ans;
}

```

```

137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156

```

1.5 SuffixArray

```

157 struct SuffixArray {
158     static const int N = 1000005; // the length of the string
159
160     int n, m, cnt[N], sa[N], rk[N], id[N];
161
162     void radixSort() {
163         for (int i = 0; i < m; ++i) {
164             cnt[i] = 0;
165         }
166         for (int i = 0; i < n; ++i) {
167             ++cnt[rk[i]];
168         }
169         for (int i = 1; i < m; ++i) {
170             cnt[i] += cnt[i - 1];
171         }
172         for (int i = n - 1; ~i; --i) {
173             sa[—cnt[rk[id[i]]]] = id[i];
174         }
175     }
176
177     bool cmp(int x, int y, int l) {
178         return id[x] == id[y] && id[x + l] == id[y + l];
179     }
180
181     template<typename T>
182     void initSA(T first, T last) {
183         n = last - first, m = 0;
184         for (int i = 0; i < n; ++i) {
185             rk[i] = *(first + i);
186             m = std::max(m, rk[i] + 1);
187             id[i] = i;
188         }
189         radixSort();
190         for (int l = 1, p = 0; p < n && l < n; m = p, l <= 1) {
191             p = 0;
192             for (int i = n - l; i < n; ++i) {
193                 id[p++] = i;
194             }
195             for (int i = 0; i < n; ++i) {
196                 if (sa[i] >= l && p < n) {
197                     id[p++] = sa[i] - l;
198                 }

```

```

        }
        radixSort();
        for (int i = 0; i < n; ++i) id[i] = rk[i];
        p = 1, rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i) {
            if (!cmp(sa[i - 1], sa[i], l) && p < n) ++p;
            rk[sa[i]] = p - 1;
        }
    }
} SA;

int main() {
    n = readStr(s);
    SA.initSA(s, s + n);
    for (int i = 0; i < n; ++i) {
        print(SA.sa[i] + 1, ' ');
    }
    putchar('\n');
}

```

2 Number Theory

2.1 Extended Euclidean Algorithm

```

def Exgcd(a, b):
    if b == 0:
        return a, 1, 0
    d, x, y = Exgcd(b, a % b)
    return d, y, x - (a // b) * y

```

2.2 Miller-Rabin primality test

```

def millerRabin(n):
    if n < 3 or n % 2 == 0:
        return n == 2
    a, b = n - 1, 0
    while a % 2 == 0:
        a = a // 2
        b = b + 1
    ""

```

```

232  test_time is the number of tests, it is recommended to set it to an integer
      not less than 8 to ensure the correct rate, but it should not be too
      large, otherwise it will affect the efficiency
233  """
234  for i in range(1, test_time + 1):
235      x = random.randint(0, 32767) % (n - 2) + 2
236      v = quickPow(x, a, n)
237      if v == 1:
238          continue
239      j = 0
240      while j < b:
241          if v == n - 1:
242              break
243          v = v * v % n
244          j = j + 1
245      if j >= b:
246          return False
247  return True

```

2.3 Sieve of Euler

```

248 void Euler(const int n = 100000) {
249     np[1] = true;
250     int cnt = 0;
251     for (int i = 2; i <= n; ++i) {
252         if (!np[i]) {
253             prime[++cnt] = i;
254         }
255         for (int j = 1; j <= cnt && (LL) i * prime[j] <= n; ++j) {
256             np[i * prime[j]] = true;
257             if (!(i % prime[j])) {
258                 break;
259             }
260         }
261     }
262 }

```

2.4 Euler's Totient Function

In number theory, Euler's totient function counts the positive integers up to a given integer n that are relatively prime to n .

$$\varphi(n) = \sum_{i=1}^n [\gcd(i, n) = 1] = n \times \prod \left(1 - \frac{1}{p_i}\right)$$

Get φ use sieve of Euler:

```

263 void pre() {
264     for (int i = 1; i <= 5000000; ++i) {
265         is_prime[i] = 1;
266     }
267     int cnt = 0;
268     is_prime[1] = 0;
269     phi[1] = 1;
270     for (int i = 2; i <= 5000000; ++i) {
271         if (is_prime[i]) {
272             prime[++cnt] = i;
273             phi[i] = i - 1;
274         }
275         for (int j = 1; j <= cnt && i * prime[j] <= 5000000; j++) {
276             is_prime[i * prime[j]] = 0;
277             if (i % prime[j])
278                 phi[i * prime[j]] = phi[i] * phi[prime[j]];
279             else {
280                 phi[i * prime[j]] = phi[i] * prime[j];
281                 break;
282             }
283         }
284     }
285 }

```

2.5 Euler's theorem

$$a^{p-1} \equiv 1 \pmod{p}$$

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)}, & \gcd(a, m) = 1 \\ a^b, & \gcd(a, m) \neq 1, b < \varphi(m) \\ a^{(b \bmod \varphi(m)) + \varphi(m)}, & \gcd(a, m) \neq 1, b \geq \varphi(m). \end{cases} \pmod{m}$$

2.6 Lucas

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

2.7 Chinese Remainder Theorem

Solve:

$$x \equiv \begin{cases} a_1 & (\bmod n_1) \\ a_2 & (\bmod n_2) \\ \vdots & \\ a_k & (\bmod n_k) \end{cases}$$

When n_1, n_2, \dots, n_k are coprime.

$$\begin{cases} n &= \prod_{i=1}^k n_i \\ m_i &= \frac{n}{n_i} \\ M_i &\equiv m_i^{-1} \pmod{n_i} \\ x &\equiv \sum_{i=1}^k a_i m_i M_i \pmod{n} \end{cases}$$

```

286 LL CRT(int k, LL *a, LL *r) {
287     LL n = 1, ans = 0;
288     for (int i = 1; i <= k; ++i) {
289         n = n * r[i];
290     }
291     for (int i = 1; i <= k; ++i) {
292         LL m = n / r[i], b, y;
293         exgcd(m, r[i], b, y); // b * m mod r[i] = 1
294         ans = (ans + a[i] * m * b % n) % n;
295     }
296     return (ans % n + n) % n;
297 }
```

2.8 Wilson's theorem

$$(p-1)! \equiv -1 \pmod{p}$$

2.9 Baby-Step Giant-Step

Get all $x \in [0, p)$ for:

$$a^x \equiv b \pmod{p},$$

where a and p are coprime.

Let $x = A \lceil \sqrt{p} \rceil - B$, $0 \leq A, B \leq \lceil \sqrt{p} \rceil$. We have $a^{A \lceil \sqrt{p} \rceil - B} \equiv b \pmod{p}$. So $a^{A \lceil \sqrt{p} \rceil} \equiv ba^B \pmod{p}$.

Enumerate all A and put them into hash map. Then enumerate B to get the answer.

2.10 Pollard-Rho

```

typedef unsigned long long ULL;
typedef long long LL;
```

```
std::set<int> ans;
```

```

inline ULL rnd() {
    static ULL seed = 2333;
    seed ^= seed << 40;
    seed ^= seed >> 23;
    seed ^= seed << 7;
    return seed;
}
```

```

template <typename T>
inline T gcd(T a, T b) {
    while (b) {
        T t = a % b;
        a = b;
        b = t;
    }
    return a < 0 ? -a : a;
}
```

```

template <typename T>
inline void add(T& x, T y, T mod) {
    x += y;
    if (x >= mod) {
        x -= mod;
    } else if (x < 0) {
        x += mod;
    }
}
```

298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328

```

329 }
330
331 inline LL cheng(LL a, LL b, LL mod) {
332     LL tmp = ((long double) a * b + .5) / mod;
333     return ((a * b - tmp * mod) % mod + mod) % mod;
334 }
335
336 inline LL ksm(LL a, LL b, LL mod) {
337     LL ans = 1;
338     for (; b >>= 1, a = cheng(a, a, mod)) {
339         if (b & 1) {
340             ans = cheng(ans, a, mod);
341         }
342     }
343     return ans;
344 }
345
346 inline bool witness(LL a, LL n) {
347     LL u = n - 1;
348     int t = 0;
349     while (!(u & 1)) {
350         u >>= 1;
351         t++;
352     }
353     LL x = ksm(a, u, n);
354     for (int i = 1; i <= t; ++i) {
355         LL lstx = x;
356         x = cheng(x, x, n);
357         if (x == 1 && lstx != 1 && lstx != n - 1) {
358             return false;
359         }
360     }
361     if (x != 1) {
362         return false;
363     }
364     return true;
365 }
366
367 inline bool MR(LL n) {
368     if (n == 2) {
369         return true;
370     }
371     static const int s = 5;
372     for (int i = 1; i <= s; ++i) {

```

```

        if (!witness(rnd() % (n - 1) + 1, n)) {
            return false;
        }
    }
    return true;
}

inline LL rho(LL n) {
    if (MR(n)) {
        return n;
    }
    LL x = rnd() % n;
    LL y = x;
    LL p = (n & 1) ? 1 : 2;
    while (p == 1) {
        LL cc = rnd() % n;
        while (true) {
            int bitt = 127;
            LL xx = 1;
            while (bitt) {
                x = cheng(x, x, n);
                add(x, cc, n);
                y = cheng(y, y, n);
                add(y, cc, n);
                y = cheng(y, y, n);
                add(y, cc, n);
                if (x == y) {
                    break;
                }
                LL tx = (__int128) xx * (y - x) % n;
                if (tx) {
                    xx = tx;
                } else {
                    break;
                }
            }
        }
        LL d = gcd((LL) xx, n);
        if (d != 1 && d != n) {
            p = d;
            break;
        }
    }
    if (x == y) {
        break;
    }
}

```



```

417     }
418 }
419 return std::max(rho(p), rho(n / p));
420 }
421
422 inline void solve() {
423     LL n;
424     read(n);
425     if (MR(n)) {
426         puts("Prime");
427     } else {
428         writeln(rho(n));
429     }
430 }

```

3 Number-Theoretic Transform

```

431 #include<bits/stdc++.h>
432 #define ll long long
433 #define mod 998244353
434 #define maxn 400005
435 #define g 3
436 using namespace std;
437 inline int read(){
438     int u=0,f=1;char c=getchar();
439     while(c<'0' || c>'9'){if(c=='-')f=-1;c=getchar();}
440     while(c>='0'&&c<='9'){u=u*10+c-'0';c=getchar();}
441     return u*f;
442 }
443
444 int a[maxn],b[maxn];
445 int n,m;
446 inline int pw(int x,int y){
447     int res=1;
448     for(;y>=1,x=1ll*x*x%mod;if(y&1)res=1ll*res*x%mod;
449     return res;
450 }
451 int rev[maxn];
452 inline void ntt(int a[],int n,int tp){
453     for(int i=0;i<n;i++)if(i<rev[i])swap(a[i],a[rev[i]]);
454     for(int k=2;k<=n;k<=1){

```

```

int wn=pw(g,(mod-1)/k);
if(tp==1)wn=pw(wn,mod-2);
for(int i=0;i<n;i+=k){
    int w=1;
    for(int j=0;j<(k>>1);j++,w=1ll*w*wn%mod){
        int x=a[i+j],y=1ll*w*a[i+j+(k>>1)]%mod;
        a[i+j]=(x+y)%mod;
        a[i+j+(k>>1)]=(x-y+mod)%mod;
    }
}
}
if(tp==1){
    int inv=pw(n,mod-2);
    for(int i=0;i<n;i++)a[i]=1ll*a[i]*inv%mod;
}
}
int main(){
    n=read();m=read();
    for(int i=0;i<n;i++)a[i]=read();
    for(int i=0;i<m;i++)b[i]=read();
    int l=1,cnt=0;
    while(l<n+m)l<=1,cnt++;
    for(int i=1;i<l;i++)rev[i]=(rev[i>>1]>>1|((i&1)<<(cnt-1)));
    ntt(a,l,1);ntt(b,l,1);
    for(int i=0;i<l;i++)a[i]=1ll*a[i]*b[i]%mod;
    ntt(a,l,-1);
    for(int i=0;i<n+m-1;i++)cout<<a[i]<<" ";
    return 0;
}

```

4 OEIS

4.1 Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418

$$f_n = f_{n-1} + f_{n-2}$$

4.2 Catalan Numbers

485 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440

$$C_n = \sum_{i=1}^n H_{i-1} H_{n-i} = \frac{\binom{2n}{n}}{n+1} = \binom{2n}{n} - \binom{2n}{n-1}$$

4.3 Bell or Exponential Numbers

Number of ways to partition a set of n labeled elements.

486 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

4.4 Bell or Exponential Numbers

Number of ways to partition a set of n labeled elements.

487 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

4.5 Lucas numbers

Lucas numbers beginning at 2: $L(n) = L(n-1) + L(n-2)$, $L(0) = 2$, $L(1) = 1$.

488 2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349, 15127, 24476, 39603, 64079, 103682, 167761, 271443, 439204

4.6 Derangement

Subfactorial or rencontres numbers, or derangements: number of permutations of n elements with no fixed points.

489 1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961

$$D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$$

4.7 Prufer

Number of labeled rooted trees with n nodes: n^{n-1} .

1, 2, 9, 64, 625, 7776, 117649, 2097152, 43046721

490

5 Data Structures

5.1 Link-Cut Tree

```
#include <cstdio>
#include <iostream>
#include <algorithm>

using namespace std;

const int maxn = 300005;

class LCT {
    // node

public:
    int sum[maxn], val[maxn];
    int s[maxn][2], fa[maxn];

private:
    bool lzy_fan[maxn];

    void push_up(int x) {
        sum[x] = val[x] ^ sum[s[x][0]] ^ sum[s[x][1]];
    }

    bool nrt(int x) {
        return s[fa[x]][0] == x || s[fa[x]][1] == x;
    }

    void fan(int x) {
        swap(s[x][0], s[x][1]);
        lzy_fan[x] ^= 1;
    }

    void push_down(int x) {
```

491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522

```

523     if (lzy_fan[x]) {
524         if (s[x][0]) {
525             fan(s[x][0]);
526         }
527         if (s[x][1]) {
528             fan(s[x][1]);
529         }
530         lzy_fan[x] = 0;
531     }
532 }
533
534 // splay
535 private:
536 void rotate(int x) {
537     int y = fa[x], z = fa[y];
538     int k = (s[y][1] == x), ss = s[x][!k];
539     if (nrt(y)) {
540         s[z][s[z][1] == y] = x;
541     }
542     fa[x] = z;
543     s[x][!k] = y;
544     fa[y] = x;
545     s[y][k] = ss;
546     if (ss) {
547         fa[ss] = y;
548     }
549     push_up(y);
550     push_up(x);
551 }
552
553 int sta[maxn];
554 void splay(int x) {
555     int K = x, top = 0;
556     sta[++top] = K;
557     while (nrt(K)) {
558         sta[++top] = K = fa[K];
559     }
560     while (top) {
561         push_down(sta[top--]);
562     }
563     while (nrt(x)) {
564         int y = fa[x], z = fa[y];
565         if (nrt(y)) {
566             rotate((s[y][0] == x) ^ (s[z][0] == y)) ? x : y);

```

```

    }
    rotate(x);
}

// LCT
private:
void access(int x) {
    for (int y = 0; x; x = fa[y = x]) {
        splay(x);
        s[x][1] = y;
        push_up(x);
    }
}

void make_root(int x) {
    access(x);
    splay(x);
    fan(x);
}

int find_root(int x) {
    access(x);
    splay(x);
    while (s[x][0]) {
        push_down(x);
        x = s[x][0];
    }
    splay(x);
    return x;
}

void split(int x, int y) {
    make_root(x);
    access(y);
    splay(y);
}

public:
void link(int x, int y) {
    make_root(x);
    if (find_root(y) != x) {
        fa[x] = y;
    }
}

```

```

611 }
612
613 void cut(int x, int y) {
614     make_root(x);
615     if (find_root(y) == x && fa[y] == x && !s[y][0]) {
616         fa[y] = s[x][1] = 0;
617         push_up(x);
618     }
619 }
620
621 void change(int x, int y) {
622     splay(x);
623     val[x] = y;
624     push_up(x);
625 }
626
627 int ask(int x, int y) {
628     split(x, y);
629     return sum[y];
630 }
631 } tr;
632
633 int main() {
634     int n, m;
635     scanf("%d%d", &n, &m);
636     for (int i = 1; i <= n; ++i) {
637         scanf("%d", &tr.val[i]);
638         tr.sum[i] = tr.val[i];
639     }
640     while (m--) {
641         int cmd, x, y;
642         scanf("%d%d%d", &cmd, &x, &y);
643         switch (cmd) {
644             case 0:
645                 printf("%d\n", tr.ask(x, y));
646                 break;
647             case 1:
648                 tr.link(x, y);
649                 break;
650             case 2:
651                 tr.cut(x, y);
652                 break;
653             case 3:
654                 tr.change(x, y);

```

```

    }
    }
    return 0;
}

```

```

655
656
657
658

```

6 Graph Theory

6.1 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 。

```

const int maxn = 13;

int n, m;

struct Matrix {
    double mt[maxn][maxn];

    inline double* operator [] (int x) {
        return mt[x];
    }

    inline void clear() {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                mt[i][j] = 0;
            }
        }
    }

    inline double getans() {
        int nn = n - 1;
        double ans = 1.;
        for (int i = 1; i <= nn; ++i) {
            int mx = i;
            for (int j = i + 1; j <= nn; ++j) {
                if (mt[mx][i] < mt[j][i]) {

```

```

659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684

```

```

685     mx = j;
686 }
687 }
688 if (i != mx) {
689     ans *= -1;
690     for (int j = i; j <= nn; ++j) {
691         std::swap(mt[mx][j], mt[i][j]);
692     }
693 }
694 if (mt[i][i] < 1e-10) {
695     return 0.;
696 }
697 for (int j = i + 1; j <= nn; ++j) {
698     double kk = mt[j][i] / mt[i][i];
699     for (int k = i; k <= nn; ++k) {
700         mt[j][k] -= kk * mt[i][k];
701     }
702 }
703 }
704 for (int i = 1; i <= nn; ++i) {
705     ans *= mt[i][i];
706 }
707 return ans;
708 }
709 } Kif;
710
711 void solve() {
712     read(n), read(m);
713     Kif.clear();
714     for (int i = 1, u, v; i <= m; ++i) {
715         read(u), read(v);
716         Kif[u][u]++, Kif[v][v]++;
717         Kif[u][v]--, Kif[v][u]--;
718     }
719     printf("%.0f\n", Kif.getans());
720 }

```

6.2 最小生成树计数

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 Kruscal 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 $\mathcal{O}(\frac{m}{n} \times n^3) = \mathcal{O}(n^2 m)$ ，所以总复杂度 $\mathcal{O}(n^2 m)$ 。

```

const int maxn = 105;
const int maxm = 1005;
const int mod = 31011;

int n, m;

struct Edge {
    int u, v, d;

    friend bool operator < (const Edge& a, const Edge& b) {
        return a.d < b.d;
    }
} e[maxm];

std::vector<std::pair<int, int>> v[maxn];

int col[maxn];

int fa[maxn];

inline int getfa(int x) {
    return fa[x] == x ? x : fa[x] = getfa(fa[x]);
}

inline void dfs(int now, int ccol, int bx) {
    col[now] = ccol;
    for (auto to : v[now]) {
        if (!col[to.first] && to.second != bx) {
            dfs(to.first, ccol, bx);
        }
    }
}

struct Matrix {
    int mt[maxn][maxn];

    inline void init(int n) {

```

721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757

```

758     for (int i = 1; i <= n; ++i) {
759         for (int j = 1; j <= n; ++j) {
760             mt[i][j] = 0;
761         }
762     }
763 }
764
765 inline int* operator [] (int x) {
766     return mt[x];
767 }
768
769 inline int solve(int n) {
770     n--;
771     if (!n) {
772         return 1;
773     }
774     int ans = 1;
775     for (int i = 1; i <= n; ++i) {
776         int now = 0;
777         for (int j = i; j <= n; ++j) {
778             if (mt[j][i]) {
779                 now = i;
780                 break;
781             }
782         }
783         if (!now) {
784             return 0;
785         } else if (now != i) {
786             for (int j = i; j <= n; ++j) {
787                 std::swap(mt[i][j], mt[now][j]);
788             }
789             ans *= -1;
790         }
791         for (int j = i + 1; j <= n; ++j) {
792             while (mt[j][i]) {
793                 int nowk = mt[i][i] / mt[j][i];
794                 for (int k = i; k <= n; ++k) {
795                     mt[i][k] -= mt[j][k] * nowk % mod;
796                     if (mt[i][k] < 0) {
797                         mt[i][k] += mod;
798                     } else if (mt[i][k] >= mod) {
799                         mt[i][k] -= mod;
800                     }
801                     std::swap(mt[i][k], mt[j][k]);

```

```

802     }
803     ans *= -1;
804 }
805 }
806 }
807 for (int i = 1; i <= n; ++i) {
808     (ans *= mt[i][i]) %= mod;
809 }
810 if (ans <= mod) {
811     ans += mod;
812 }
813 return ans;
814 }
815 } mat;
816
817 inline int Main() {
818     read(n), read(m);
819     for (int i = 1; i <= m; ++i) {
820         read(e[i].u), read(e[i].v), read(e[i].d);
821     }
822     std::sort(e + 1, e + m + 1);
823     int cnt = 0, now = 0;
824     for (int i = 1; i <= m; ++i) {
825         if (now < e[i].d) {
826             now = e[i].d;
827             cnt++;
828         }
829         e[i].d = cnt;
830     }
831     for (int i = 1; i <= n; ++i) {
832         fa[i] = i;
833     }
834     for (int i = 1; i <= m; ++i) {
835         int fax = getfa(e[i].u);
836         int fay = getfa(e[i].v);
837         if (fax != fay) {
838             fa[fax] = fay;
839             v[e[i].u].emplace_back(e[i].v, e[i].d);
840             v[e[i].v].emplace_back(e[i].u, e[i].d);
841         }
842     }
843     int ans = 1;
844     for (int i = 1; i <= cnt; ++i) {
845         memset(col, 0, sizeof(col));

```

```

846     int cntt = 0;
847     for (int j = 1; j <= n; ++j) {
848         if (!col[j]) {
849             dfs(j, ++cntt, i);
850         }
851     }
852     mat.init(cntt);
853     for (int j = 1; j <= m; ++j) {
854         if (e[j].d == i && col[e[j].u] != col[e[j].v]) {
855             mat[col[e[j].u]][col[e[j].v]]--;
856             mat[col[e[j].v]][col[e[j].u]]--;
857             mat[col[e[j].u]][col[e[j].u]]++;
858             mat[col[e[j].v]][col[e[j].v]]++;
859         }
860     }
861     (ans *= mat.solve(cntt)) %= mod;
862 }
863 writeln(ans);
864 return 0;
865 }

```

7 Network flow

7.1 Maximum Flow Problem

```

866 namespace FLOW {
867     const int inf = 0x3f3f3f3f;
868
869     struct Edge {
870         int to, nxt;
871         int cap;
872     } e[maxm << 1];
873
874     int first[maxn];
875     int first_bak[maxn];
876     int cnt = -1;
877
878     void init() {
879         memset(first, 0xff, sizeof(first));
880         cnt = -1;
881     }
882

```

```

883 void add_edge(int u, int v, int cap) {
884     e[++cnt].nxt = first[u];
885     first[u] = cnt;
886     e[cnt].to = v;
887     e[cnt].cap = cap;
888     e[++cnt].nxt = first[v];
889     first[v] = cnt;
890     e[cnt].to = u;
891     e[cnt].cap = 0;
892 }
893
894 int dep[maxn];
895
896 bool bfs(int s, int t) {
897     memcpy(first, first_bak, sizeof(first));
898     std::queue<int> q;
899     q.push(s);
900     memset(dep, 0x3f, sizeof(dep));
901     dep[s] = 0;
902     while (!q.empty()) {
903         int now = q.front();
904         q.pop();
905         for (int i = first[now]; ~i; i = e[i].nxt) {
906             int to = e[i].to;
907             if (e[i].cap && dep[to] >= inf) {
908                 dep[to] = dep[now] + 1;
909                 q.push(to);
910             }
911         }
912     }
913     return dep[t] < inf;
914 }
915
916 int dfs(int now, int t, int lim) {
917     if (!lim || now == t) {
918         return lim;
919     }
920     int flow = 0;
921     for (int i = first[now]; ~i; i = e[i].nxt) {
922         first[now] = i;
923         if (dep[e[i].to] == dep[now] + 1) {
924             int f = dfs(e[i].to, t, std::min(lim, e[i].cap));
925             flow += f;
926             lim -= f;

```

```

927     e[i].cap -= f;
928     e[i ^ 1].cap += f;
929     if (!lim) {
930         break;
931     }
932 }
933 }
934 return flow;
935 }
936
937 int Dinic(int s, int t) {
938     memcpy(first_bak, first, sizeof(first_bak));
939     int maxflow = 0;
940     while (bfs(s, t)) {
941         maxflow += dfs(s, t, inf);
942     }
943     return maxflow;
944 }
945 }

```

7.2 Minimum-Cost Flow Problem

```

946 #include <bits/stdc++.h>
947 using namespace std;
948 typedef long long LL;
949 struct Edge{
950     int x,y,c,nxt,cap;
951     Edge(){}
952     Edge(int a,int b,int _c,int d,int e){
953         x=a,y=b,c=_c,cap=d,nxt=e;
954     }
955 };
956 struct Network{
957     static const int N=405,M=15005*2,INF=0x7FFFFFFF;
958     Edge e[M];
959     int n,S,T,fst[N],cur[N],cnt;
960     int q[N],vis[N],head,tail;
961     int MaxFlow,MinCost,dis[N];
962     void clear(int _n){
963         n=_n,cnt=1;
964         memset(fst,0,sizeof fst);
965     }

```

```

966 void add(int a,int b,int c,int d){
967     e[++cnt]=Edge(a,b,d,c,fst[a]),fst[a]=cnt;
968     e[++cnt]=Edge(b,a,-d,0,fst[b]),fst[b]=cnt;
969 }
970 void init(){
971     for (int i=1;i<=n;i++)
972         cur[i]=fst[i];
973 }
974 void init(int _S,int _T){
975     S=_S,T=_T,MaxFlow=MinCost=0,init();
976 }
977 int SPFA(){
978     for (int i=1;i<=n;i++)
979         dis[i]=INF;
980     memset(vis,0,sizeof vis);
981     head=tail=0;
982     dis[q[++tail]=T]=0;
983     while (head!=tail){
984         if ((++head)>=n)
985             head=-n;
986         int x=q[head];
987         vis[x]=0;
988         for (int i=fst[x];i;i=e[i].nxt){
989             int y=e[i].y;
990             if (e[i^1].cap&&dis[x]-e[i].c<dis[y]){
991                 dis[y]=dis[x]-e[i].c;
992                 if (!vis[y]){
993                     if ((++tail)>=n)
994                         tail=-n;
995                     vis[q[tail]=y]=1;
996                 }
997             }
998         }
999     }
1000     memset(vis,0,sizeof vis);
1001     return dis[S]<INF;
1002 }
1003 int dfs(int x,int Flow){
1004     if (x==T||!Flow)
1005         return Flow;
1006     vis[x]=1;
1007     int now=Flow;
1008     for (int &i=cur[x];i;i=e[i].nxt){
1009         int y=e[i].y;

```



```

1010         if (!vis[y]&&e[i].cap&&dis[x]-e[i].c==dis[y]){
1011             int d=dfs(y,min(now,e[i].cap));
1012             e[i].cap-=d,e[i^1].cap+=d;
1013             if (!(now-=d))
1014                 break;
1015         }
1016     }
1017     vis[x]=0;
1018     return Flow-now;
1019 }
1020 void Dinic(){
1021     while (SPFA()){
1022         init();
1023         int now=dfs(S,INF);
1024         MaxFlow+=now,MinCost+=now*dis[S];
1025     }
1026 }
1027 void MCMF(int &_MinCost,int &_MaxFlow){
1028     Dinic(),_MinCost=MinCost,_MaxFlow=MaxFlow;
1029 }
1030 void Auto(int _S,int _T,int &_MinCost,int &_MaxFlow){
1031     init(_S,_T),MCMF(_MinCost,_MaxFlow);
1032 }
1033 }g;
1034 int read(){
1035     int x=0;
1036     char ch=getchar();
1037     while (!isdigit(ch))
1038         ch=getchar();
1039     while (isdigit(ch))
1040         x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
1041     return x;
1042 }
1043 int n,m,S,T;
1044 int main(){
1045     n=read(),m=read(),S=1,T=n;
1046     g.clear(n);
1047     while (m--){
1048         int a=read(),b=read(),c=read(),cap=read();
1049         g.add(a,b,c,cap);
1050     }
1051     int MinCost,MaxFlow;
1052     g.Auto(S,T,MinCost,MaxFlow);
1053     printf("%d_%d\n",MaxFlow,MinCost);

```

```

return 0;
}

```

1054
1055

7.3 无源汇上下界可行流

给定无源汇流量网络 G 。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时每一个点流量平衡。

不妨假设每条边已经流了 $b(u,v)$ 的流量，设其为初始流。同时我们在新图中加入 u 连向 v 的流量为 $c(u,v) - b(u,v)$ 的边。考虑在新图上进行调整。

由于最大流需要满足初始流量平衡条件（最大流可以看成是下界为 0 的上下界最大流），但是构造出来的初始流很有可能不满足初始流量平衡。假设一个点初始流入流量减初始流出流量为 M 。

若 $M = 0$ ，此时流量平衡，不需要附加边。

若 $M > 0$ ，此时入流量过大，需要新建附加源点 S' ， S' 向其连流量为 M 的附加边。

若 $M < 0$ ，此时出流量过大，需要新建附加汇点 T' ，其向 T' 连流量为 $-M$ 的附加边。

如果附加边满流，说明这一个点的流量平衡条件可以满足，否则这个点的流量平衡条件不满足。（因为原图加上附加流之后才会满足原图中的流量平衡。）

在建图完毕之后跑 S' 到 T' 的最大流，若 S' 连出去的边全部满流，则存在可行流，否则不存在。

7.4 有源汇上下界可行流

给定有源汇流量网络 G 。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。

假设源点为 S ，汇点为 T 。

则我们可以加入一条 T 到 S 的上界为 ∞ ，下界为 0 的边转化为无源汇上下界可行流问题。

若有解，则 S 到 T 的可行流流量等于 T 到 S 的附加边的流量。

7.5 有源汇上下界最大流

给定有源汇流量网络 G 。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。如果存在，询问满足标定的最大流量。

我们找到网络上的任意一个可行流。如果找不到解就可以直接结束。

否则我们考虑删去所有附加边之后的残量网络并且在网络上进行调整。

我们在残量网络上再跑一次 S 到 T 的最大流，将可行流流量和最大流流量相加即为答案。

S 到 T 的最大流千万不可以在直接在跑完有源汇上下界可行的残量网络上跑。

7.6 有源汇上下界最小流

给定有源汇流量网络 G 。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。如果存在，询问满足标定的最小流量。

类似的，我们考虑将残量网络中不需要的流退掉。

我们找到网络上的任意一个可行流。如果找不到解就可以直接结束。

否则我们考虑删去所有附加边之后的残量网络。

我们在残量网络上再跑一次 T 到 S 的最大流，将可行流流量减去最大流流量即为答案。

8 Others

8.1 vim

```
1056 set tabstop=4
1057 set nocompatible
1058 set shiftwidth=4
1059 set expandtab
1060 set autoindent
1061 set smartindent
1062 set ruler
1063 set showcmd
```

```
set incsearch
set shellslash
set number
set relativenumber
set cino+=L0
set splitright
filetype indent on
filetype off

colorscheme evening

imap jk      <Esc>

inoremap {<CR>  {<CR>}<Esc>0
inoremap {      {}<left>
inoremap {}      {}
inoremap (      ()<left>
inoremap ()      ()

setlocal makeprg=g++\ -O2\ -Wall\ --std=c++17\ -Wno-unused-result\ %:r.cpp\ -o\
%:r
nmap <F2> <cmd>vs %:r.in<CR>
nmap <F3> <cmd>!%:r < %:r.in <CR>
nmap <F4> <cmd>w<CR><cmd>make<CR>
nmap <F5> <cmd>w<CR><cmd>make<CR><cmd>!%:r < %:r.in<CR>
syntax on
```