Nanyang Technological University
Joker
Reference Book

Contents

1 String

1.1 KMP

```cpp
std::vector<int> kmp(std::string s) {
  int n = s.length();
  std::vector<int> pi(n);
  for (int i = 1; i < n; ++i) {
    int j = pi[i - 1];
    while (j && s[i] != s[j]) {
      j = pi[j - 1];
    }
    if (s[i] == s[j]) {
      j++;
    }
    pi[i] = j;
  }
  return pi;
}
```

1.2 Z-function

```cpp
std::vector<int> z_function(std::string s) {
  int n = s.length();
  std::vector<int> z(n);
  z[0] = n;
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = std::max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
        z[i]++;
      }
    }
    if (i + z[i] - 1 > r) {
      l = i, r = i + z[i] - 1;
    }
  }
  return z;
}
```

1.3 Aho-Corasick algorithm

```cpp
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();
```

```cpp
      for (int i = 0; i < 26; ++i) {
        if (son[now][i]) {
          fail[son[now][i]] = son[fail[now]][i];
          q.push(son[now][i]);
        } else {
          son[now][i] = son[fail[now]][i];
        }
      }
    }
  }

  void getval(std::string s) {
    int now = 0;
    for (auto c : s) {
      now = son[now][c - 'a'];
      val[now]++;
    }
  }

  void build_fail_tree() {
    for (int i = 1; i <= cnt; ++i) {
      fas[fail[i]].push_back(i);
    }
  }

  void dfs(int now = 0) {
    for (auto x : fas[now]) {
      dfs(x);
      val[now] += val[x];
    }
    if (!id[now].empty()) {
      for (auto x : id[now]) {
        ans[x] = val[now];
      }
    }
  }
};

Aho_Corasick ac;

int n;

int main() {
  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;
}
```

1.4 Manachar

```cpp
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;
}
```

1.5 SuffixArray

```
struct SuffixArray {
    static const int N = 1000005; // the length of the string

    int n, m, cnt[N], sa[N], rk[N], id[N];

    void radixSort() {
        for (int i = 0; i < m; ++i) {
            cnt[i] = 0;
        }
        for (int i = 0; i < n; ++i) {
            ++cnt[rk[i]];
        }
        for (int i = 1; i < m; ++i) {
            cnt[i] += cnt[i - 1];
        }
        for (int i = n - 1; ~i; --i) {
            sa[--cnt[rk[id[i]]]] = id[i];
        }
    }

    bool cmp(int x, int y, int l) {
        return id[x] == id[y] && id[x + l] == id[y + l];
    }

    template<typename T>
    void initSA(T first, T last) {
        n = last - first, m = 0;
        for (int i = 0; i < n; ++i) {
            rk[i] = *(first + i);
            m = std::max(m, rk[i] + 1);
            id[i] = i;
        }
        radixSort();
        for (int l = 1, p = 0; p < n && l < n; m = p, l <<= 1) {
            p = 0;
            for (int i = n - l; i < n; ++i) {
                id[p++] = i;
            }
            for (int i = 0; i < n; ++i) {
                if (sa[i] >= l && p < n) {
                    id[p++] = sa[i] - l;
                }
            }
            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
    """
```

```python
    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
    """
    for i in range(1, test_time + 1):
        x = random.randint(0, 32767) % (n - 2) + 2
        v = quickPow(x, a, n)
        if v == 1:
            continue
        j = 0
        while j < b:
            if v == n - 1:
                break
            v = v * v % n
            j = j + 1
        if j >= b:
            return False
    return True
```

2.3 Sieve of Euler

```cpp
void Euler(const int n = 100000) {
  np[1] = true;
  int cnt = 0;
  for (int i = 2; i <= n; ++i) {
    if (!np[i]) {
      prime[++cnt] = i;
    }
    for (int j = 1; j <= cnt && (LL) i * prime[j] <= n; ++j) {
      np[i * prime[j]] = true;
      if (!(i % prime[j])) {
        break;
      }
    }
  }
}
```

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:

```cpp
void pre() {
  for (int i = 1; i <= 5000000; ++i) {
    is_prime[i] = 1;
  }
  int cnt = 0;
  is_prime[1] = 0;
  phi[1] = 1;
  for (int i = 2; i <= 5000000; ++i) {
    if (is_prime[i]) {
      prime[++cnt] = i;
      phi[i] = i - 1;
    }
    for (int j = 1; j <= cnt && i * prime[j] <= 5000000; j++) {
      is_prime[i * prime[j]] = 0;
      if (i % prime[j])
        phi[i * prime[j]] = phi[i] * phi[prime[j]];
      else {
        phi[i * prime[j]] = phi[i] * prime[j];
        break;
      }
    }
  }
}
```

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:

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

When $n_1, n_2, \cdots, n_k$ are coprime.

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

```
LL CRT(int k, LL *a, LL *r) {
  LL n = 1, ans = 0;
  for (int i = 1; i <= k; ++i) {
    n = n * r[i];
  }
  for (int i = 1; i <= k; ++i) {
    LL m = n / r[i], b, y;
    exgcd(m, r[i], b, y);  // b * m mod r[i] = 1
    ans = (ans + a[i] * m * b % n) % n;
  }
  return (ans % n + n) % n;
}
```

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 $x = A\lceil\sqrt{p}\rceil - B$, $0 \le A, B \le \lceil\sqrt{p}\rceil$. 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;
  }
```

```
}

inline LL cheng(LL a, LL b, LL mod) {
  LL tmp = ((long double) a * b + .5) / mod;
  return ((a * b - tmp * mod) % mod + mod) % mod;
}

inline LL ksm(LL a, LL b, LL mod) {
  LL ans = 1;
  for (; b; b >>= 1, a = cheng(a, a, mod)) {
    if (b & 1) {
      ans = cheng(ans, a, mod);
    }
  }
  return ans;
}

inline bool witness(LL a, LL n) {
  LL u = n - 1;
  int t = 0;
  while (!(u & 1)) {
    u >>= 1;
    t++;
  }
  LL x = ksm(a, u, n);
  for (int i = 1; i <= t; ++i) {
    LL lstx = x;
    x = cheng(x, x, n);
    if (x == 1 && lstx != 1 && lstx != n - 1) {
      return false;
    }
  }
  if (x != 1) {
    return false;
  }
  return true;
}

inline bool MR(LL n) {
  if (n == 2) {
    return true;
  }
  static const int s = 5;
  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;
      }
```

```
    }
  }
  return std::max(rho(p), rho(n / p));
}

inline void solve() {
  LL n;
  read(n);
  if (MR(n)) {
    puts("Prime");
  } else {
    writeln(rho(n));
  }
}
```

3 Number-Theoretic Transform

```
#include<bits/stdc++.h>
#define ll long long
#define mod 998244353
#define maxn 400005
#define g 3
using namespace std;
inline int read(){
  int u=0,f=1;char c=getchar();
  while(c<'0'||c>'9'){if(c=='-')f=-1;c=getchar();}
  while(c>='0'&&c<='9'){u=u*10+c-'0';c=getchar();}
  return u*f;
}

int a[maxn],b[maxn];
int n,m;
inline int pw(int x,int y){
  int res=1;
  for(;y;y>>=1,x=1ll*x*x%mod)if(y&1)res=1ll*res*x%mod;
  return res;
}
int rev[maxn];
inline void ntt(int a[],int n,int  tp){
  for(int i=0;i<n;i++)if(i<rev[i])swap(a[i],a[rev[i]]);
  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

```
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.

```
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.

```
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: $L(n) = L(n-1) + L(n-2), L(0) = 2, L(1) = 1$.

```
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.

```
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
```

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) {
```

```
    if (lzy_fan[x]) {
      if (s[x][0]) {
        fan(s[x][0]);
      }
      if (s[x][1]) {
        fan(s[x][1]);
      }
      lzy_fan[x] = 0;
    }
  }

  // splay
private:
  void rotate(int x) {
    int y = fa[x], z = fa[y];
    int k = (s[y][1] == x), ss = s[x][!k];
    if (nrt(y)) {
      s[z][s[z][1] == y] = x;
    }
    fa[x] = z;
    s[x][!k] = y;
    fa[y] = x;
    s[y][k] = ss;
    if (ss) {
      fa[ss] = y;
    }
    push_up(y);
    push_up(x);
  }

  int sta[maxn];
  void splay(int x) {
    int K = x, top = 0;
    sta[++top] = K;
    while (nrt(K)) {
      sta[++top] = K = fa[K];
    }
    while (top) {
      push_down(sta[top--]);
    }
    while (nrt(x)) {
      int y = fa[x], z = fa[y];
      if (nrt(y)) {
        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;
    }
```

```
  }

  void cut(int x, int y) {
    make_root(x);
    if (find_root(y) == x && fa[y] == x && !s[y][0]) {
      fa[y] = s[x][1] = 0;
      push_up(x);
    }
  }

  void change(int x, int y) {
    splay(x);
    val[x] = y;
    push_up(x);
  }

  int ask(int x, int y) {
    split(x, y);
    return sum[y];
  }
} tr;

int main() {
  int n, m;
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= n; ++i) {
    scanf("%d", &tr.val[i]);
    tr.sum[i] = tr.val[i];
  }
  while (m--) {
    int cmd, x, y;
    scanf("%d%d%d", &cmd, &x, &y);
    switch (cmd) {
      case 0:
        printf("%d\n", tr.ask(x, y));
        break;
      case 1:
        tr.link(x, y);
        break;
      case 2:
        tr.cut(x, y);
        break;
      case 3:
        tr.change(x, y);
      }
    }
  return 0;
}
```

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 $r\,(1 \le r \le n)$，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]) {
```

```
          mx = j;
        }
      }
      if (i != mx) {
        ans *= -1;
        for (int j = i; j <= nn; ++j) {
          std::swap(mt[mx][j], mt[i][j]);
        }
      }
      if (mt[i][i] < 1e-10) {
        return 0.;
      }
      for (int j = i + 1; j <= nn; ++j) {
        double kk = mt[j][i] / mt[i][i];
        for (int k = i; k <= nn; ++k) {
          mt[j][k] -= kk * mt[i][k];
        }
      }
    }
    for (int i = 1; i <= nn; ++i) {
      ans *= mt[i][i];
    }
    return ans;
  }
} Kif;

void solve() {
  read(n), read(m);
  Kif.clear();
  for (int i = 1, u, v; i <= m; ++i) {
    read(u), read(v);
    Kif[u][u]++, Kif[v][v]++;
    Kif[u][v]--, Kif[v][u]--;
  }
  printf("%.0f\n", Kif.getans());
}
```

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 $\mathcal{O}\left(\sum\left(n+m+\min(n,a_i)^3\right)\right)$，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 $\mathcal{O}\left(\frac{m}{n} \times n^3\right) = \mathcal{O}(n^2 m)$，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) {
```

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

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

  inline int solve(int n) {
    n--;
    if (!n) {
      return 1;
    }
    int ans = 1;
    for (int i = 1; i <= n; ++i) {
      int now = 0;
      for (int j = i; j <= n; ++j) {
        if (mt[j][i]) {
          now = i;
          break;
        }
      }
      if (!now) {
        return 0;
      } else if (now != i) {
        for (int j = i; j <= n; ++j) {
          std::swap(mt[i][j], mt[now][j]);
        }
        ans *= -1;
      }
      for (int j = i + 1; j <= n; ++j) {
        while (mt[j][i]) {
          int nowk = mt[i][i] / mt[j][i];
          for (int k = i; k <= n; ++k) {
            mt[i][k] -= mt[j][k] * nowk % mod;
            if (mt[i][k] < 0) {
              mt[i][k] += mod;
            } else if (mt[i][k] >= mod) {
              mt[i][k] -= mod;
            }
            std::swap(mt[i][k], mt[j][k]);
          }
          ans *= -1;
        }
      }
    }
    for (int i = 1; i <= n; ++i) {
      (ans *= mt[i][i]) %= mod;
    }
    if (ans <= mod) {
      ans += mod;
    }
    return ans;
  }
} mat;

inline int Main() {
  read(n), read(m);
  for (int i = 1; i <= m; ++i) {
    read(e[i].u), read(e[i].v), read(e[i].d);
  }
  std::sort(e + 1, e + m + 1);
  int cnt = 0, now = 0;
  for (int i = 1; i <= m; ++i) {
    if (now < e[i].d) {
      now = e[i].d;
      cnt++;
    }
    e[i].d = cnt;
  }
  for (int i = 1; i <= n; ++i) {
    fa[i] = i;
  }
  for (int i = 1; i <= m; ++i) {
    int fax = getfa(e[i].u);
    int fay = getfa(e[i].v);
    if (fax != fay) {
      fa[fax] = fay;
      v[e[i].u].emplace_back(e[i].v, e[i].d);
      v[e[i].v].emplace_back(e[i].u, e[i].d);
    }
  }
  int ans = 1;
  for (int i = 1; i <= cnt; ++i) {
    memset(col, 0, sizeof(col));
```

```
    int cntt = 0;
    for (int j = 1; j <= n; ++j) {
      if (!col[j]) {
        dfs(j, ++cntt, i);
      }
    }
    mat.init(cntt);
    for (int j = 1; j <= m; ++j) {
      if (e[j].d == i && col[e[j].u] != col[e[j].v]) {
        mat[col[e[j].u]][col[e[j].v]]--;
        mat[col[e[j].v]][col[e[j].u]]--;
        mat[col[e[j].u]][col[e[j].u]]++;
        mat[col[e[j].v]][col[e[j].v]]++;
      }
    }
    (ans *= mat.solve(cntt)) %= mod;
  }
  writeln(ans);
  return 0;
}
```

7 Network flow

7.1 Maximum Flow Problem

```
namespace FLOW {
  const int inf = 0x3f3f3f3f;

  struct Edge {
    int to, nxt;
    int cap;
  } e[maxm << 1];

  int first[maxn];
  int first_bak[maxn];
  int cnt = -1;

  void init() {
    memset(first, 0xff, sizeof(first));
    cnt = -1;
  }

  void add_edge(int u, int v, int cap) {
    e[++cnt].nxt = first[u];
    first[u] = cnt;
    e[cnt].to = v;
    e[cnt].cap = cap;
    e[++cnt].nxt = first[v];
    first[v] = cnt;
    e[cnt].to = u;
    e[cnt].cap = 0;
  }

  int dep[maxn];

  bool bfs(int s, int t) {
    memcpy(first, first_bak, sizeof(first));
    std::queue<int> q;
    q.push(s);
    memset(dep, 0x3f, sizeof(dep));
    dep[s] = 0;
    while (!q.empty()) {
      int now = q.front();
      q.pop();
      for (int i = first[now]; ~i; i = e[i].nxt) {
        int to = e[i].to;
        if (e[i].cap && dep[to] >= inf) {
          dep[to] = dep[now] + 1;
          q.push(to);
        }
      }
    }
    return dep[t] < inf;
  }

  int dfs(int now, int t, int lim) {
    if (!lim || now == t) {
      return lim;
    }
    int flow = 0;
    for (int i = first[now]; ~i; i = e[i].nxt) {
      first[now] = i;
      if (dep[e[i].to] == dep[now] + 1) {
        int f = dfs(e[i].to, t, std::min(lim, e[i].cap));
        flow += f;
        lim -= f;
```

```
        e[i].cap -= f;
        e[i ^ 1].cap += f;
        if (!lim) {
          break;
        }
      }
    }
    return flow;
  }

  int Dinic(int s, int t) {
    memcpy(first_bak, first, sizeof(first_bak));
    int maxflow = 0;
    while (bfs(s, t)) {
      maxflow += dfs(s, t, inf);
    }
    return maxflow;
  }
}
```

7.2 Minimum-Cost Flow Problem

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
struct Edge{
    int x,y,c,nxt,cap;
    Edge(){}
    Edge(int a,int b,int _c,int d,int e){
        x=a,y=b,c=_c,cap=d,nxt=e;
    }
};
struct Network{
    static const int N=405,M=15005*2,INF=0x7FFFFFFF;
    Edge e[M];
    int n,S,T,fst[N],cur[N],cnt;
    int q[N],vis[N],head,tail;
    int MaxFlow,MinCost,dis[N];
    void clear(int _n){
        n=_n,cnt=1;
        memset(fst,0,sizeof fst);
    }
void add(int a,int b,int c,int d){
    e[++cnt]=Edge(a,b,d,c,fst[a]),fst[a]=cnt;
    e[++cnt]=Edge(b,a,-d,0,fst[b]),fst[b]=cnt;
}
void init(){
    for (int i=1;i<=n;i++)
        cur[i]=fst[i];
}
void init(int _S,int _T){
    S=_S,T=_T,MaxFlow=MinCost=0,init();
}
int SPFA(){
    for (int i=1;i<=n;i++)
        dis[i]=INF;
    memset(vis,0,sizeof vis);
    head=tail=0;
    dis[q[++tail]=T]=0;
    while (head!=tail){
        if ((++head)>=n)
            head-=n;
        int x=q[head];
        vis[x]=0;
        for (int i=fst[x];i;i=e[i].nxt){
            int y=e[i].y;
            if (e[i^1].cap&&dis[x]-e[i].c<dis[y]){
                dis[y]=dis[x]-e[i].c;
                if (!vis[y]){
                    if ((++tail)>=n)
                        tail-=n;
                    vis[q[tail]=y]=1;
                }
            }
        }
    }
    memset(vis,0,sizeof vis);
    return dis[S]<INF;
}
int dfs(int x,int Flow){
    if (x==T||!Flow)
        return Flow;
    vis[x]=1;
    int now=Flow;
    for (int &i=cur[x];i;i=e[i].nxt){
        int y=e[i].y;
```

```
                if (!vis[y]&&e[i].cap&&dis[x]-e[i].c==dis[y]){
                    int d=dfs(y,min(now,e[i].cap));
                    e[i].cap-=d,e[i^1].cap+=d;
                    if (!(now-=d))
                        break;
                }
            }
            vis[x]=0;
            return Flow-now;
        }
        void Dinic(){
            while (SPFA()){
                init();
                int now=dfs(S,INF);
                MaxFlow+=now,MinCost+=now*dis[S];
            }
        }
        void MCMF(int &_MinCost,int &_MaxFlow){
            Dinic(),_MinCost=MinCost,_MaxFlow=MaxFlow;
        }
        void Auto(int _S,int _T,int &_MinCost,int &_MaxFlow){
            init(_S,_T),MCMF(_MinCost,_MaxFlow);
        }
}g;
int read(){
    int x=0;
    char ch=getchar();
    while (!isdigit(ch))
        ch=getchar();
    while (isdigit(ch))
        x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
    return x;
}
int n,m,S,T;
int main(){
    n=read(),m=read(),S=1,T=n;
    g.clear(n);
    while (m--){
        int a=read(),b=read(),c=read(),cap=read();
        g.add(a,b,c,cap);
    }
    int MinCost,MaxFlow;
    g.Auto(S,T,MinCost,MaxFlow);
    printf("%d %d\n",MaxFlow,MinCost);
    return 0;
}
```

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

```
set tabstop=4
set nocompatible
set shiftwidth=4
set expandtab
set autoindent
set smartindent
set ruler
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>O
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
```