ICPC Templates

Beyond List @ SDNU

June 16, 2024

Contents

1 Math

1.1 欧拉筛

```cpp
struct Sieve {
   std::vector<int> P, v;

   Sieve(int n)
      : v(n) {
      for (int i = 2; i < n; i++) {
         if (v[i] == 0) {
            P.push_back(i);
            v[i] = i;
         }
         for (int j = 0; j < P.size() and i * P[j] < n; j++) {
            v[i * P[j]] = P[j];
            if (P[j] == v[i]) break;
         }
      }
   }

   // 求所有约数
   auto getDiv(int x) const {
      std::vector<int> _div(1, 1);
      while (x > 1) {
         int D = v[x];
         int l = 0, r = _div.size();
         while (x % D == 0) {
            for (int k = l; k < r; k++) _div.push_back(_div[k] * D);
            x /= D, l = r, r = _div.size();
         }
      }
      return _div;
   }
};
```

1.2 组合数

```cpp
template<class T, T P> class Comb
{
   static constexpr int multip(const int& a, const int& b) { return 1ll * a
       * b % P; }
   static constexpr i64 multip(const i64& a, const i64& b) {
      i64 res = a * b - i64(1.L * a * b / P) * P;
      res %= P;
      res += (res < 0 ? P : 0);
      return res;
   }

   int n;
   std::vector<T> _jc, _ijc, _inv;

```

```
14 public:
15    constexpr Comb()
16       : n{0}
17       , _jc{1}
18       , _ijc{1}
19       , _inv{0} {}
20    Comb(int n)
21       : Comb() {
22       init(n);
23    }
24
25    static constexpr T powp(T a, i64 mi) {
26       T ans = 1;
27       for (; mi; mi >>= 1, a = multip(a, a))
28          if (mi & 1) ans = multip(ans, a);
29       return ans;
30    }
31
32    void init(int m) {
33       m = std::min(m, P - 1);
34       if (m <= n) return;
35
36       _jc.resize(m + 1);
37       _ijc.resize(m + 1);
38       _inv.resize(m + 1);
39
40       for (int i = n + 1; i <= m; i++) {
41          _jc[i] = multip(i, _jc[i - 1]);
42       }
43       _ijc.back() = powp(_jc.back(), P - 2);
44       for (int i = m; i > n; i--) {
45          _ijc[i - 1] = multip(i, _ijc[i]);
46          _inv[i] = multip(_ijc[i], _jc[i - 1]);
47       }
48
49       n = m;
50    }
51
52    T jc(int x) {
53       if (x > n) init(x << 1);
54       return _jc[x];
55    }
56    T ijc(int x) {
57       if (x > n) init(x << 1);
58       return _ijc[x];
59    }
60    T inv(int x) {
61       if (x > n) init(x << 1);
62       return _inv[x];
63    }
64
65    T A(int a, int b) {
66       if (a < b or b < 0) return 0;
```

```
67      return multip(jc(a), ijc(a - b));
68   }
69   T C(int a, int b) {
70      if (a < b or b < 0) return 0;
71      return multip(A(a, b), ijc(b));
72   }
73 };
74 constexpr int P = 998244353;
75 Comb<int, P> comb;
76
77 // 取模加法
78 int add(int a, int b) {
79    a += b;
80
81    if (a >= P) {
82       a -= P;
83    }
84
85    if (a < 0) {
86       a += P;
87    }
88
89    return a;
90 }
```

1.3 拓展欧几里得

对于方程 $ax + by + c$, 调用 '$exgcd$', 求出 x_0 和 y_0, 使得 $ax_0 + by_0 = gcd(a, c)$。

在 $gcd(a, b)|c$ 的情况下，方程有通解:

$x = x_0 * \frac{c}{gcd(a,b)} + k * \frac{b}{gcd(a,b)}$ 和 $y = y_0 * \frac{c}{gcd(a,b)} - k * \frac{a}{gcd(a,b)}$

```
1  template <class T>
2  struct ExGcd {
3     T operator()(const T &a, const T &b, T &x, T &y) {
4        if (b == 0)
5           return (x = 1, y = 0, a);
6        T g = (*this)(b, a % b, y, x);
7        y -= a / b * x;
8        return g;
9     }
10 };
11 ExGcd<int> exgcd;
```

1.4 中国剩余定理

```
1  template<class T, class G> struct ExCrt : public ExGcd<T> {
2     std::vector<std::pair<T, T>> q;
3     void insert(T a, T mod) { q.push_back({a, mod}); }
4
5     // 方程组 x ≡ a (模 mod) 返回最小正解
6     // 无解返回 -1
```

```
   T get() {
      T res = 0, M = 1;
      for (auto [a, mod] : q) {
         T r = (a - res) % mod;
         r += (r < 0 ? mod : 0);

         T x, y;
         T g = (*this)(M, mod, x, y);
         if (r % g) {
            q.clear();
            return -1;
         }

         x = (G(x) * r / g % (mod / g));
         x += (x < 0 ? mod / g : 0);

         T Last = M;
         M = M / g * mod;
         res = (G(x) * Last % M + res) % M;
      }
      q.clear();
      return res;
   }
};

ExCrt<i64, __int128> crt;
```

1.5 RandomTheory

1.5.1 RandomNumber

```
template<class T> struct Rand {
   std::mt19937 myrand;
   Rand(const i64 seed = time(0))
      : myrand(seed) {}
   T operator()(T l, T r) { return std::uniform_int_distribution<T>(l, r)(
       myrand); }
};
Rand<int> rd;

// std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().
    count());
```

1.5.2 MillerRabin

```
/*
维基百科 :
n < 4e9, Prime = [2, 7, 61]
n < 3e14, Prime = [2, 3, 5, 7, 11, 13, 17]
n < 3e18, Prime = [2, 3, 5, 7, 11, 13, 17, 19, 23]
n < 3e23, Prime = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
*/
```

```cpp
template<class T> struct MillerRabin {
   const std::vector<int> Prime;
   MillerRabin()
      : Prime({2, 3, 5, 7, 11, 13, 17, 19, 23}) {}

   static constexpr int mulp(const int& a, const int& b, const int& P) { return 1ll * a * b % P; }
   static constexpr i64 mulp(const i64& a, const i64& b, const i64& P) {
      i64 res = a * b - i64(1.L * a * b / P) * P;
      res %= P;
      res += (res < 0 ? P : 0);
      return res;
   }

   static constexpr T powp(T a, T mi, const T& mod) {
      T ans = 1;
      for (; mi; mi >>= 1) {
         if (mi & 1) ans = mulp(ans, a, mod);
         a = mulp(a, a, mod);
      }
      return ans;
   }

   constexpr bool operator()(const T& v) { // 判断v是不是质数
      if (v < 2 or v != 2 and v % 2 == 0) return false;
      T s = v - 1;
      while (!(s & 1)) s >>= 1;
      for (int x : Prime) {
         if (v == x) return true;
         T t = s, m = powp(x, s, v);
         while (t != v - 1 and m != 1 and m != v - 1) m = mulp(m, m, v), t <<= 1;
         if (m != v - 1 and !(t & 1)) return false;
      }
      return true;
   }
};
MillerRabin<i64> isp;
```

1.5.3 PollardRho

如果 n 是质数 (MillerRabin 判断), 返回 n, 否则返回 n 的随机一个 $[2, n-1]$ 的因子。

复杂度略微高于 $O(n^{\frac{1}{4}}logn)$

```cpp
/*
维基百科 :
n < 4e9, Prime = [2, 7, 61]
n < 3e14, Prime = [2, 3, 5, 7, 11, 13, 17]
n < 3e18, Prime = [2, 3, 5, 7, 11, 13, 17, 19, 23]
n < 3e23, Prime = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
*/
template<class T> struct MillerRabin {
```

```
    const std::vector<int> Prime;
    MillerRabin()
        : Prime({2, 3, 5, 7, 11, 13, 17, 19, 23}) {}

    static constexpr int mulp(const int& a, const int& b, const int& P) {
        return 1ll * a * b % P; }
    static constexpr i64 mulp(const i64& a, const i64& b, const i64& P) {
        i64 res = a * b - i64(1.L * a * b / P) * P;
        res %= P;
        res += (res < 0 ? P : 0);
        return res;
    }

    static constexpr T powp(T a, T mi, const T& mod) {
        T ans = 1;
        for (; mi; mi >>= 1) {
            if (mi & 1) ans = mulp(ans, a, mod);
            a = mulp(a, a, mod);
        }
        return ans;
    }

    constexpr bool operator()(const T& v) { // 判断v是不是质数
        if (v < 2 or v != 2 and v % 2 == 0) return false;
        T s = v - 1;
        while (!(s & 1)) s >>= 1;
        for (int x : Prime) {
            if (v == x) return true;
            T t = s, m = powp(x, s, v);
            while (t != v - 1 and m != 1 and m != v - 1) m = mulp(m, m, v), t
                <<= 1;
            if (m != v - 1 and !(t & 1)) return false;
        }
        return true;
    }
};
MillerRabin<i64> isp;
```

2 String

2.1 最小表示法

```
std::vector<int> minimalString(std::vector<int>& a) {
    int n = a.size();
    int i = 0, j = 1, k = 0;
    while (k < n and i < n and j < n) {
        if (a[(i + k) % n] == a[(j + k) % n])
            k++;
        else {
            (a[(i + k) % n] > a[(j + k) % n] ? i : j) += k + 1;
            i += (i == j);
            k = 0;
```

```
    }
  }
  k = std::min(i, j);
  std::vector<int> ans(n);
  for (int i = 0; i < n; i++) ans[i] = a[(i + k) % n];
  return ans;
}
// 直接返回字典序最小循环同构串
// 4321的循环同构串有 3214 2134 1432，最小为1432
```

2.2 字符串哈希

```
// -std=c++20
template<int D, std::array<int, D> B, std::array<int, D> P> struct
    StringHash {
  std::vector<std::array<int, D>> h;

  template<class T>
  StringHash(const T& s)
    : h(s.size() + 1) {
    for (int i = 0; i < s.size(); i++) {
      for (int k = 0; k < D; k++) {
        h[i + 1][k] = (1ll * h[i][k] * B[k] + s[i] + 1) % P[k];
      }
    }
  }

  // [l, r)
  std::array<int, D> get(int l, int r) {
    static std::vector<std::array<int, D>> spow(1);
    if (r - l < 0) throw -1;

    if (spow.size() < r - l + 1) {
      if (spow[0][0] == 0) {
        spow[0].fill(1);
      }
      int n = spow.size();
      spow.resize(r - l + 1);
      for (int i = n; i < spow.size(); i++) {
        for (int k = 0; k < D; k++) {
          spow[i][k] = 1ll * spow[i - 1][k] * B[k] % P[k];
        }
      }
    }

    std::array<int, D> res = {};
    for (int k = 0; k < D; k++) {
      res[k] = h[r][k] - 1ll * h[l][k] * spow[r - l][k] % P[k];
      res[k] += (res[k] < 0 ? P[k] : 0);
    }
    return res;
  }
```

```
};
using Hash = StringHash<2, {133, 331}, {int(1e9 + 21), int(1e9 + 33)}>;
```

2.3 KMP

```
auto kmp(const std::string& s) {
   int n = s.size();
   std::vector<int> link(n);
   for (int i = 1, j = 0; i < n; i++) {
      while (j and s[i] ≠ s[j]) j = link[j - 1];
      j += (s[i] == s[j]);
      link[i] = j;
   }
   return link;
}

void find(const std::string& s, const std::string& p, const std::vector<int
    >& link) {
   for (int i = 0, j = 0; i < (int)s.size(); ++i) {
      while (j && s[i] ≠ p[j]) j = link[j - 1];
      j += (s[i] == p[j]);
      if (j == (int)p.size()) {
         std::cout << i - j + 2 << '\n';
         j = link[j - 1];
      }
   }
}
```

2.4 字典树

```
struct Trie {
   int ch[N][63], cnt[N], idx = 0;
   map<char, int> mp;
   void init() {
      LL id = 0;
      for (char c = 'a'; c <= 'z'; c++) mp[c] = ++id;
      for (char c = 'A'; c <= 'Z'; c++) mp[c] = ++id;
      for (char c = '0'; c <= '9'; c++) mp[c] = ++id;
   }
   void insert(string s) {
      int u = 0;
      for (int i = 0; i < s.size(); i++) {
         int v = mp[s[i]];
         if (!ch[u][v]) ch[u][v] = ++idx;
         u = ch[u][v];
         cnt[u]++;
      }
   }
   LL query(string s) {
      int u = 0;
      for (int i = 0; i < s.size(); i++) {
```

```
22          int v = mp[s[i]];
23          if (!ch[u][v]) return 0;
24          u = ch[u][v];
25       }
26       return cnt[u];
27    }
28    void Clear() {
29       for (int i = 0; i <= idx; i++) {
30          cnt[i] = 0;
31          for (int j = 0; j <= 62; j++) {
32             ch[i][j] = 0;
33          }
34       }
35       idx = 0;
36    }
37 } trie;
```

2.5 01 字典树

```
1  struct Trie {
2     int n, idx;
3     std::vector<std::vector<int>> ch;
4     Trie(int n) {
5        this->n = n;
6        idx = 0;
7        ch.resize(30 * (n + 1), std::vector<int>(2));
8     }
9     void insert(int x) {
10       int u = 0;
11       for (int i = 30; ~i; i--) {
12          int& v = ch[u][x >> i & 1];
13          if (!v) v = ++idx;
14          u = v;
15       }
16    }
17    int query(int x) {
18       int u = 0, res = 0;
19       for (int i = 30; ~i; i--) {
20          int v = x >> i & 1;
21          if (ch[u][!v]) {
22             res += (1 << i);
23             u = ch[u][!v];
24          } else {
25             u = ch[u][v];
26          }
27       }
28       return res;
29    }
30 };
```

2.6 AC 自动机

```cpp
// Trie+Kmp, 多模式串匹配
struct ACAutomaton {
   static constexpr int N = 1e6 + 10;
   int ch[N][26], fail[N], cntNodes;
   int cnt[N];
   ACAutomaton() { cntNodes = 1; }
   void insert(string s) {
      int u = 1;
      for (auto c : s) {
         int& v = ch[u][c - 'a'];
         if (!v) v = ++cntNodes;
         u = v;
      }
      cnt[u]++;
   }
   void build() {
      fill(ch[0], ch[0] + 26, 1);
      queue<int> q;
      q.push(1);
      while (!q.empty()) {
         int u = q.front();
         q.pop();
         for (int i = 0; i < 26; i++) {
            int& v = ch[u][i];
            if (!v)
               v = ch[fail[u]][i];
            else {
               fail[v] = ch[fail[u]][i];
               q.push(v);
            }
         }
      }
   }
   LL query(string t) {
      LL ans = 0;
      int u = 1;
      for (auto c : t) {
         u = ch[u][c - 'a'];
         for (int v = u; v && ~cnt[v]; v = fail[v]) {
            ans += cnt[v];
            cnt[v] = -1;
         }
      }
      return ans;
   }
};
```

2.7 AC 自动机 2

```cpp
template<int Z, char Base> struct AcAutomaton {
   std::vector<std::array<int, Z>> son;
```

```
3     std::vector<std::vector<int>> ID;
4     std::vector<int> link;
5     int SIZE = 0, tot = 0;
6
7     AcAutomaton(const std::vector<std::string>& s) {
8        for (auto t : s) SIZE += t.size();
9        son.resize(SIZE + 1);
10       ID.resize(SIZE + 1);
11       link.resize(SIZE + 1);
12
13       for (int i = 0; i < s.size(); i++) insert(i, s[i]);
14       build();
15    }
16
17    void insert(int id, const std::string& s) {
18       int p = 0;
19       for (char c : s) {
20          c -= Base;
21          if (!son[p][c]) son[p][c] = ++tot;
22          p = son[p][c];
23       }
24       ID[p].push_back(id);
25    }
26
27    void build() {
28       std::queue<int> q;
29       for (int& y : son[0])
30          if (y) {
31             q.push(y);
32          }
33       while (!q.empty()) {
34          int x = q.front();
35          q.pop();
36
37          for (int c = 0; int& y : son[x]) {
38             if (y) {
39                link[y] = son[link[x]][c];
40                q.push(y);
41             } else
42                y = son[link[x]][c];
43             c++;
44          }
45       }
46    }
47 };
```

2.8 Z-Function

Z_i 是 S 与 $S[i \dots n-1]$ 的最长公共前缀

```
1  auto zFunction(const std::string& s) {
2     int n = s.size();
```

```
    std::vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i < r) z[i] = std::min(z[i - l], r - i);
        while (i + z[i] < n and s[i + z[i]] == s[z[i]]) z[i]++;
        if (i + z[i] > r) l = i, r = i + z[i];
    }
    return z;
    // S : aabcaabcaaaab
    // Z : 0100610022310
}
```

2.9 马拉车

```
struct Manacher {
    const int n;
    std::vector<int> r, f;

    Manacher(const std::string& t)
        : n(t.size())
        , r(2 * t.size() + 3)
        , f(2 * t.size() + 3) {
        std::string s = "[-";
        for (int i = 0; i < n; i++) {
            s += t[i];
            s += '-';
        }
        s.push_back(']');

        int mid = 1, far = 1;
        for (int i = 1; i < s.size(); i++) {
            r[i] = std::min(r[2 * mid - i], far - i);
            while (s[i + r[i]] == s[i - r[i]]) r[i] += 1;
            if (far < i + r[i]) mid = i, far = i + r[i];
            f[i + r[i] - 1] = std::max(f[i + r[i] - 1], r[i]);
        }
        for (int i = f.size() - 2; i; i--) f[i] = std::max(f[i], f[i + 1] -
            1);
    }

    // 返回以i为中心的最长回文字符串长度，center为以+0.5为中心
    // aaa (0, 1) = 2
    int getPalinLenFromCenter(int i, int center) const {
        assert(!center and 0 <= i and i < n or center and 0 <= i and i < n -
            1);

        return r[2 * (i + 1) + center] - 1;
    }

    // 返回以i结尾的最长回文字符串长度
    int getPalinLenFromTail(int i) const {
        assert(0 <= i and i < n);
        return f[2 * (i + 1)];
```

```
    }
};
```

2.10 后缀数组

字符串本质不同的子串数量为 $\frac{n(n-1)}{2} - \sum h[i]$

两个子串的 LCP 为 $\min_{l_1 \le k \le l_2} h[k]$

```
struct SuffixArray {
    std::vector<int> sa, rk, h;
    // sa: S的所有后缀按字典序排序
    // h: LCP(sa[i], sa[i - 1]), 即sa[i]和sa[i - 1]的最长公共前缀

    template<class T>
    SuffixArray(const T& s)
        : n(s.size())
        , sa(s.size())
        , rk(s.size())
        , id(s.size())
        , tmp(s.size()) {

        std::iota(begin(id), end(id), 0);
        for (int i = 0; i < n; i++) rk[i] = s[i];

        countSort();

        for (int w = 1;; w <<= 1) {
            std::iota(begin(id), begin(id) + w + 1, n - w);
            for (int i = 0, p = w; i < n; i++)
                if (sa[i] >= w) id[p++] = sa[i] - w;

            countSort();
            oldrk = rk;

            rk[sa[0]] = 0;
            for (int i = 1, p = 0; i < n; i++) rk[sa[i]] = equal(sa[i], sa[i -
                1], w) ? p : ++p;

            if (rk[sa.back()] + 1 == n) break;
        }

        calcHeight(s);
    }

private:
    const int n;
    std::vector<int> oldrk, id, tmp, cnt;

    template<class T> inline void calcHeight(const T& s) {
        h.assign(n, 0);
        for (int i = 0, k = 0; i < n; i++) {
            if (rk[i] == 0) continue;
```

```
44          k -= bool(k);
45          while (s[i + k] == s[sa[rk[i] - 1] + k]) k += 1;
46          h[rk[i]] = k;
47       }
48    }
49
50    // 计数排序
51    inline void countSort() {
52       int m = *std::max_element(begin(rk), end(rk));
53       cnt.assign(m + 1, 0);
54       for (int i = 0; i < n; i++) cnt[tmp[i] = rk[id[i]]] += 1;
55       for (int i = 1; i < cnt.size(); i++) cnt[i] += cnt[i - 1];
56       for (int i = n - 1; i >= 0; i--) sa[--cnt[tmp[i]]] = id[i];
57    }
58
59    inline bool equal(int x, int y, int w) {
60       int rkx = (x + w < n ? oldrk[x + w] : -1);
61       int rky = (y + w < n ? oldrk[y + w] : -1);
62       return oldrk[x] == oldrk[y] and rkx == rky;
63    }
64 };
```

3 DataStruct

3.1 RMQ

```
1  template<class T, class Cmp = std::less<T>> struct RMQ {
2     const Cmp cmp = Cmp();
3     std::vector<std::vector<T>> ST;
4
5     RMQ(const std::vector<T>& a) {
6        int n = a.size(), logn = std::__lg(n);
7        ST.assign(n, std::vector<T>(logn + 1));
8        for (int i = 0; i < n; i++) ST[i][0] = a[i];
9        for (int j = 0; j < logn; j++) {
10          for (int i = 0; i + (1 << (j + 1)) - 1 < n; i++) {
11             ST[i][j + 1] = std::min(ST[i][j], ST[i + (1 << j)][j], cmp);
12          }
13       }
14    }
15
16    // [l, r)
17    T operator()(int l, int r) {
18       int log = std::__lg(r - l);
19       return std::min(ST[l][log], ST[r - (1 << log)][log], cmp);
20    }
21 };
```

3.2 Heap

```cpp
template<class T, class Cmp = std::less<T>> struct Heap {
   std::priority_queue<T, std::vector<T>, Cmp> qPush, qErase; // Heap=qPush-qErase

   void push(T x) { qPush.push(x); }

   void erase(T x) { qErase.push(x); }

   T top() {
      while (!qErase.empty() && qPush.top() == qErase.top()) qPush.pop(), qErase.pop();
      return qPush.top();
   }

   void pop() {
      while (!qErase.empty() && qPush.top() == qErase.top()) qPush.pop(), qErase.pop();
      qPush.pop();
   }

   int size() { return qPush.size() - qErase.size(); }
};
```

3.3 并查集

```cpp
struct DSU {
   std::vector<int> f;
   std::vector<int> size;

   DSU(int n)
      : f(n)
      , size(n) {
      std::iota(f.begin(), f.end(), 0);
      std::fill(size.begin(), size.end(), 1);
   }

   int find(int x) {
      while (x != f[x]) x = f[x] = f[f[x]];
      return x;
   }

   void Union(int x, int y) {
      x = find(x), y = find(y);
      if (x == y) return;

      if (size[x] < size[y]) std::swap(x, y);

      size[x] += size[y];
      f[y] = x;
   }
};
```

3.4 带权并查集

```cpp
template<class T> struct DSU {
   std::vector<int> f;
   std::vector<int> size;
   std::vector<T> w;

   DSU(int n)
      : f(n)
      , size(n)
      , w(n) {
      std::iota(f.begin(), f.end(), 0);
      std::fill(size.begin(), size.end(), 1);
   }

   int find(int x) {
      if (f[x] == x) return x;
      int pr = f[x], anc = find(pr);

      w[x] = w[x] + w[pr];

      return f[x] = anc;
   }

   void Union(int x, int y, const T& z) {
      T road = w[x] + z, lastWy = w[y];
      x = find(x), y = find(y);
      if (x == y) return;

      w[y] = road - lastWy;

      size[x] += size[y];
      f[y] = x;
   }
};

struct Info {
   int val;

   Info(int x = 0)
      : val(x) {}

   bool operator==(const Info& a) const { return val == a.val; }

   Info operator+(const Info& a) const {}

   Info operator-(const Info& a) const {}
};
```

3.5 pbdsTree

```cpp
#include <bits/extc++.h>
```

```
namespace pb = __gnu_pbds;

template<class T, class Cmp = std::less<T>>
using RedBlackTree =
   pb::tree<T, pb::null_type, Cmp, pb::rb_tree_tag, pb::
       tree_order_statistics_node_update>;

/**
 * order_of_key(x) -> 查询有多少元素比x小
 * find_by_order(x) -> 查询有x个元素比它小的元素的迭代器
 */
```

3.6 SegmentTree

3.6.1 SegTree

```
template<class T, class Merge = std::plus<T>> struct SegT {
   const Merge merge;
   const int n;
   std::vector<T> t;

   SegT(int n)
      : n(n)
      , t(4 << std::__lg(n))
      , merge(Merge()) {}

   SegT(const std::vector<T>& a)
      : SegT(a.size()) {
      std::function<void(int, int, int)> build = [&](int i, int l, int r) {
         if (r - l == 1) {
            t[i] = a[l];
            return;
         }
         int mid = l + r >> 1;
         build(i << 1, l, mid);
         build(i << 1 | 1, mid, r);
         up(i);
      };
      build(1, 0, n);
   }

   void up(int i) { t[i] = merge(t[i << 1], t[i << 1 | 1]); }

   // 默认单点赋值
   void modify(int x, const T& v) { modify(1, 0, n, x, v); }

   void modify(int i, int l, int r, int x, const T& v) {
      if (r - l == 1) {
         t[i] = v;
         return;
      }
      int mid = l + r >> 1;
```

```
37      if (x < mid)
38         modify(i << 1, l, mid, x, v);
39      else
40         modify(i << 1 | 1, mid, r, x, v);
41      up(i);
42   }
43
44   // [l, r)
45   T rangeQuery(int l, int r) { return rangeQuery(1, 0, n, l, r); }
46
47   T rangeQuery(int i, int l, int r, int tl, int tr) {
48      if (tl <= l and r <= tr) {
49         return t[i];
50      }
51      int mid = l + r >> 1;
52      return merge((tl < mid ? rangeQuery(i << 1, l, mid, tl, tr) : T()),
53                (mid < tr ? rangeQuery(i << 1 | 1, mid, r, tl, tr) : T()));
54   }
55 };
```

3.6.2 LazySegTree

```
1  template<class T, class Tag> struct LazySegT {
2     int n;
3     std::vector<T> info;
4     std::vector<Tag> tag;
5
6     LazySegT(int n, T v = T())
7        : LazySegT(std::vector(n, v)) {}
8
9     template<class G>
10    LazySegT(const std::vector<G>& a)
11       : n(a.size()) {
12       info.assign(4 << std::__lg(n), T());
13       tag.assign(4 << std::__lg(n), Tag());
14       std::function<void(int, int, int)> build = [&](int i, int l, int r) {
15          if (r - l == 1) {
16             info[i] = a[l];
17             return;
18          }
19          int mid = l + r >> 1;
20          build(i << 1, l, mid);
21          build(i << 1 | 1, mid, r);
22          up(i);
23       };
24       build(1, 0, n);
25    }
26
27    void up(int i) { info[i] = info[i << 1] + info[i << 1 | 1]; }
28    void apply(int i, const Tag& v) {
29       info[i].apply(v);
30       tag[i].apply(v);
```

```
31    }
32    void down(int i) {
33       apply(i << 1, tag[i]);
34       apply(i << 1 | 1, tag[i]);
35       tag[i] = Tag();
36    }
37
38    // 单点修改
39    void modify(int i, const T& v) { modify(1, 0, n, i, v); }
40    void modify(int i, int l, int r, int x, const T& v) {
41       if (r - l == 1) {
42          info[i] = v;
43          return;
44       }
45       int mid = l + r >> 1;
46       down(i);
47       if (x < mid) {
48          modify(i << 1, l, mid, x, v);
49       } else {
50          modify(i << 1 | 1, mid, r, x, v);
51       }
52       up(i);
53    }
54
55    // 区间查询 [l, r)
56    T rangeQuery(int l, int r) { return rangeQuery(1, 0, n, l, r); }
57    T rangeQuery(int i, int l, int r, int tl, int tr) {
58
59       if (tl <= l and r <= tr) return info[i];
60
61       down(i);
62       int mid = l + r >> 1;
63
64       return (tl < mid ? rangeQuery(i << 1, l, mid, tl, tr) : T()) +
65            (mid < tr ? rangeQuery(i << 1 | 1, mid, r, tl, tr) : T());
66    }
67
68    // 区间修改 [l, r)
69    void rangeModify(int l, int r, const Tag& v) { return rangeModify(1, 0,
          n, l, r, v); }
70    void rangeModify(int i, int l, int r, int tl, int tr, const Tag& v) {
71
72       if (tl <= l and r <= tr) {
73          apply(i, v);
74          return;
75       }
76       down(i);
77       int mid = l + r >> 1;
78
79       if (tl < mid) rangeModify(i << 1, l, mid, tl, tr, v);
80       if (mid < tr) rangeModify(i << 1 | 1, mid, r, tl, tr, v);
81       up(i);
82    }
```

```
83
84      // 区间左边第一个满足条件的下标
85      template<class F> int findFirst(int l, int r, F pred) { return findFirst
            (1, 0, n, l, r, pred); }
86      template<class F> int findFirst(int i, int l, int r, int tl, int tr, F
            pred) {
87          if (l >= tr || r <= tl || !pred(info[i])) {
88              return -1;
89          }
90          if (r - l == 1) {
91              return l;
92          }
93          int mid = l + r >> 1;
94          down(i);
95          int res = findFirst(i << 1, l, mid, tl, tr, pred);
96          if (res == -1) {
97              res = findFirst(i << 1 | 1, mid, r, tl, tr, pred);
98          }
99          return res;
100     }
101
102     // 区间右边第一个满足条件的下标
103     template<class F> int findLast(int l, int r, F pred) { return findLast
            (1, 0, n, l, r, pred); }
104     template<class F> int findLast(int i, int l, int r, int tl, int tr, F
            pred) {
105         if (l >= tr || r <= tl || !pred(info[i])) {
106             return -1;
107         }
108         if (r - l == 1) {
109             return l;
110         }
111         int mid = l + r >> 1;
112         down(i);
113         int res = findLast(i << 1 | 1, mid, r, tl, tr, pred);
114         if (res == -1) {
115             res = findLast(i << 1, l, mid, tl, tr, pred);
116         }
117         return res;
118     }
119 };
120
121 struct Tag {
122     int add;
123
124     Tag(const int& add = 0)
125         : add(add) {}
126
127     void apply(const Tag& tag) {
128         if (tag.add) {}
129     }
130 };
131
```

```
132 struct Node {
133    i64 val;
134    int len;
135
136    Node(const i64& val = 0, const int& len = 1)
137       : val(val)
138       , len(len) {}
139
140    void apply(const Tag& tag) {
141       if (tag.add) {
142          val += 1ll * tag.add * len;
143       }
144    }
145
146    Node operator+(const Node& a) { return Node(val + a.val, len + a.len); }
147 };
```

3.6.3 主席树

```
1  template<class T> class PresidentTree
2  {
3
4     using NodeIndex = int;
5
6     struct Node {
7        int val;
8        NodeIndex l, r;
9
10       Node(int val = 0)
11          : val{val}
12          , l{0}
13          , r{0} {}
14    };
15
16    std::vector<Node> t; // memory pool
17
18    const T Start, Last;
19    std::vector<NodeIndex> root;
20
21    constexpr NodeIndex newNode(int val = 0) {
22       t.emplace_back(val);
23       return (int)t.size() - 1;
24    }
25
26    constexpr void up(NodeIndex i) { t[i].val = t[t[i].l].val + t[t[i].r].
          val; }
27
28    constexpr void modify(NodeIndex& p, T l, T r, T x) {
29       if (p == 0) {
30          p = newNode();
31       }
32       if (r - l == 1) {
```

```
33          t[p].val++;
34          return;
35       }
36
37       T mid = (0LL + l + r) / 2;
38
39       if (x < mid)
40          modify(t[p].l, l, mid, x);
41       else
42          modify(t[p].r, mid, r, x);
43       up(p);
44    }
45    constexpr NodeIndex merge(NodeIndex x, NodeIndex y, T l, T r) {
46       if (!x or !y) return (x ? x : y);
47
48       // 每次把 x 修改
49       if (r - l == 1) {
50          t[x].val += t[y].val;
51          return x;
52       }
53
54       T mid = (0LL + l + r) / 2;
55       t[x].l = merge(t[x].l, t[y].l, l, mid);
56       t[x].r = merge(t[x].r, t[y].r, mid, r);
57       return up(x), x;
58    }
59
60    constexpr int getRange(NodeIndex x, NodeIndex y, T l, T r, T tl, T tr) {
61       if (tl <= l and r <= tr) {
62          return t[y].val - t[x].val;
63       }
64       T mid = (0LL + l + r) / 2;
65       return (tl < mid ? getRange(t[x].l, t[y].l, l, mid, tl, tr) : 0) +
66            (mid < tr ? getRange(t[x].r, t[y].r, mid, r, tl, tr) : 0);
67    }
68
69    constexpr T getKth(NodeIndex x, NodeIndex y, T l, T r, int k) {
70       if (r - l == 1) return l;
71       T mid = (0LL + l + r) / 2;
72       int L = t[t[y].l].val - t[t[x].l].val;
73
74       return (L >= k ? getKth(t[x].l, t[y].l, l, mid, k) : getKth(t[x].r, t
             [y].r, mid, r, k - L));
75    }
76
77 public:
78    constexpr PresidentTree(const std::vector<T>& a, T min, T max)
79       : root(a.size() + 1)
80       , Start(min)
81       , Last(max + 1)
82       , t(1) {
83
84       t.reserve(a.size() * std::__lg(a.size() * 2));
```

```
85
86       root[0] = newNode();
87       for (int i = 1; i <= a.size(); i++) {
88          if (t.capacity() <= t.size() + 64) {
89             t.reserve(std::max(2 * t.capacity(), t.capacity() + 64));
90          }
91          modify(root[i], Start, Last, a[i - 1]);
92          root[i] = merge(root[i], root[i - 1], Start, Last);
93       }
94    }
95    // [l, r), [tl, tr)
96    constexpr int getRange(int l, int r, T tl, T tr) {
97       return getRange(root[l], root[r], Start, Last, tl, tr);
98    }
99    // [l, r)
100   constexpr T getKth(int l, int r, int k) { return getKth(root[l], root[r
          ], Start, Last, k); }
101 };
```

3.7 BIT

3.7.1 BIT

```
1  template<class T, class Cmp = std::greater<T>> struct Max {
2     const Cmp cmp = Cmp();
3     constexpr T operator()(const T& a, const T& b) const { return std::min(a
          , b, cmp); }
4  };
5
6  template<class T, class Merge = std::plus<T>> struct BIT {
7     const Merge merge;
8     std::vector<T> t;
9
10    BIT(int n)
11       : t(n + 1)
12       , merge(Merge()) {}
13
14    // O(n) build BIT
15    BIT(const std::vector<T>& a)
16       : BIT(a.size()) {
17       int n = a.size();
18       for (int i = 1; i <= n; i++) {
19          t[i] = merge(t[i], a[i - 1]);
20          int j = i + (i & -i);
21          if (j <= n) t[j] = merge(t[j], t[i]);
22       }
23    }
24
25    void modify(int i, const T& x) {
26       for (i += 1; i < t.size(); i += i & -i) t[i] = merge(t[i], x);
27    }
28
```

```
29     T posQuery(int i) {
30         T res = T();
31         for (i += 1; i; i -= i & -i) res = merge(res, t[i]);
32         return res;
33     }
34 
35     // [l, r)
36     T rangeQuery(int l, int r) { return posQuery(r - 1) - posQuery(l - 1); }
37 };
```

3.7.2 RangeBIT

```
1  template<class T> struct RangeBIT {
2      BIT<T, std::plus<T>> d, s;
3  
4      RangeBIT(int n)
5          : d(n)
6          , s(n) {}
7  
8      // O(n) build RangeBIT
9      RangeBIT(std::vector<T> a)
10         : d(diff(a))
11         , s(multIndex(diff(a))) {}
12 
13     static std::vector<T> diff(std::vector<T> a) {
14         std::adjacent_difference(begin(a), end(a), begin(a));
15         return a;
16     }
17 
18     static std::vector<T> multIndex(std::vector<T> a) {
19         for (int i = 0; i < a.size(); i++) {
20             a[i] *= i;
21         }
22         return a;
23     }
24 
25     // [l, r)
26     void rangeModify(int l, int r, const T& x) {
27         d.modify(l, x), d.modify(r, -x);
28         s.modify(l, l * x), s.modify(r, -r * x);
29     }
30 
31     // [l, r)
32     T rangeQuery(int l, int r) {
33         T res1 = r * d.posQuery(r - 1) - s.posQuery(r - 1);
34         T res2 = l * d.posQuery(l - 1) - s.posQuery(l - 1);
35         return res1 - res2;
36     }
37 };
```

3.7.3 MatBIT

```cpp
template<class T, class Merge = std::plus<T>> struct MatBIT {
   const Merge merge;
   std::vector<BIT<T, Merge>> t;

   MatBIT(int n, int m)
      : t(n + 1, BIT<T>(m))
      , merge(Merge()) {}

   void modify(int x, int y, const T& v) {
      for (int i = x + 1; i < t.size(); i += i & -i) {
         t[i].modify(y, v);
      }
   }

   T posQuery(int x, int y) {
      T res = T();
      for (int i = x + 1; i; i -= i & -i) {
         res = merge(res, t[i].posQuery(y));
      }
      return res;
   }

   // [u, d), [l, r)
   T rangeQuery(int u, int l, int d, int r) {
      u -= 1, l -= 1, d -= 1, r -= 1;
      T res1 = posQuery(d, r) + posQuery(u, l);
      T res2 = posQuery(d, l) + posQuery(u, r);
      return res1 - res2;
   }
};
```

3.7.4 RangeMatBIT

```cpp
template<class T> struct RangeMatBIT {
   MatBIT<T> p, px, py, pxy;

   RangeMatBIT(int n, int m)
      : p(n, m)
      , px(n, m)
      , py(n, m)
      , pxy(n, m) {}

   // [u, d), [l, r)
   void rangeModify(int u, int l, int d, int r, const T& v) {
      modify(u, l, v);
      modify(d, r, v);
      modify(u, r, -v);
      modify(d, l, -v);
   }

   // [u, d), [l, r)
   T rangeQuery(int u, int l, int d, int r) {
```

```
20       u -= 1, l -= 1, d -= 1, r -= 1;
21       return query(u, l) + query(d, r) - query(d, l) - query(u, r);
22    }
23
24 private:
25    void modify(int x, int y, const T& v) {
26       p.modify(x, y, v);
27       px.modify(x, y, v * x);
28       py.modify(x, y, v * y);
29       pxy.modify(x, y, v * x * y);
30    }
31
32    T query(int x, int y) {
33       T res = T();
34       res += p.posQuery(x, y) * (x + 1) * (y + 1);
35       res -= px.posQuery(x, y) * (y + 1);
36       res -= py.posQuery(x, y) * (x + 1);
37       res += pxy.posQuery(x, y);
38       return res;
39    }
40 };
```

3.8 Block

```
1  // O(sqrt(n)) 区间加, O(1) 单点查
2  template<class T, class Merge = std::plus<T>> struct Block {
3     const int n, B;
4     const Merge merge;
5     std::vector<T> a, b;
6
7     Block(int n, const T& v = T())
8        : Block(std::vector<T>(n, v)) {}
9
10    Block(const std::vector<T>& _init)
11       : n(_init.size())
12       , B(sqrt(2 * _init.size()))
13       , a(_init)
14       , merge(Merge()) {
15       b.assign(n / B + 1, T());
16    }
17
18    // [l, r)
19    void add(int l, int r, const T& v) {
20       for (; l / B == (l - 1) / B and l < r; l++) {
21          a[l] = merge(a[l], v);
22       }
23       for (; r / B == (r - 1) / B and l < r; r--) {
24          a[r - 1] = merge(a[r - 1], v);
25       }
26       for (int i = l / B; i < r / B; i++) {
27          b[i] = merge(b[i], v);
28       }
```

```
29     }
30
31     T get(int x) { return merge(a[x], b[x / B]); }
32 };
```

4 Graph

4.1 树剖

```
1  template<class T> class TreePre
2  {
3     static constexpr int endPoint(int x) { return x; }
4     template<class G> static constexpr int endPoint(const std::pair<int, G>&
          pr) {
5        return pr.first;
6     }
7
8     void dfs1(int x, int f) {
9        fa[x] = f;
10
11       for (auto&& p : e[x]) {
12          int&& y = endPoint(p);
13          if (y != f) {
14             dep[y] = dep[x] + 1;
15             dfs1(y, x);
16
17             size[x] += size[y];
18             if (big[x] == -1 or size[y] > size[big[x]]) big[x] = y;
19          }
20       }
21    }
22    void dfs2(int x, int top) {
23       dfn[x] = cur++;
24       idfn[dfn[x]] = x;
25       tp[x] = top;
26       if (big[x] != -1) dfs2(big[x], top);
27
28       for (auto&& p : e[x]) {
29          int&& y = endPoint(p);
30          if (y != big[x] and y != fa[x]) dfs2(y, y);
31       }
32    }
33    const std::vector<std::vector<T>>& e;
34
35    const int n;
36
37 public:
38    std::vector<int> size, big, dep, tp, fa, dfn, idfn;
39    // dfn begin from 0
40    int cur = 0;
41
42    TreePre(const std::vector<std::vector<T>>& g, int root)
```

```
43      : e(g)
44      , n{g.size()}
45      , big(n, -1)
46      , size(n, 1)
47      , tp(n)
48      , dep(n)
49      , fa(n)
50      , dfn(n)
51      , idfn(n) {
52      // dep begin from 0
53      // dep[root] = 0;
54      dfs1(root, -1);
55      dfs2(root, root);
56   }
57
58   int getLca(int x, int y) {
59      while (tp[x] ≠ tp[y]) (dep[tp[x]] > dep[tp[y]] ? x = fa[tp[x]] : y =
            fa[tp[y]]);
60      return (dep[x] < dep[y] ? x : y);
61   }
62
63   int dist(int x, int y) {
64      int lca = getLca(x, y);
65      return dep[x] + dep[y] - 2 * dep[lca];
66   }
67
68   // x→y路径剖分的dfn号区间[l, r], l > r 说明这是上升段
69   auto getRoad(int x, int y) {
70      int lca = getLca(x, y);
71      std::vector<std::pair<int, int>> vec1, vec2;
72      while (tp[x] ≠ tp[lca]) {
73         vec1.push_back({dfn[x], dfn[tp[x]]});
74         x = fa[tp[x]];
75      }
76
77      if (x ≠ lca) {
78         vec1.push_back({dfn[x], dfn[lca] + 1});
79      }
80
81      vec1.push_back({dfn[lca], dfn[lca]});
82
83      while (tp[y] ≠ tp[lca]) {
84         vec2.push_back({dfn[tp[y]], dfn[y]});
85         y = fa[tp[y]];
86      }
87
88      if (y ≠ lca) {
89         vec2.push_back({dfn[lca] + 1, dfn[y]});
90         y = fa[tp[y]];
91      }
92
93      vec1.insert(end(vec1), rbegin(vec2), rend(vec2));
94      return vec1;
```

```
95     }
96
97     int kthAncester(int x, int k) {
98        if (dep[x] < k) {
99           return -1;
100       }
101
102       int d = dep[x] - k;
103
104       while (dep[tp[x]] > d) {
105          x = fa[tp[x]];
106       }
107
108       return idfn[dfn[x] - dep[x] + d];
109    }
110
111    // x is y's ancester
112    bool isAncester(int x, int y) { return dfn[x] <= dfn[y] and dfn[y] < dfn
           [x] + size[x]; }
113 };
```

4.2 树的直径

```
1  template<class T> class TreeDiameter
2  {
3     static constexpr std::pair<int, int> edge(int x) { return {x, 1}; }
4     template<class G> static constexpr int edge(const std::pair<int, G>& pr)
           { return pr; }
5
6     const std::vector<T>& e;
7     std::vector<i64> dis;
8
9     void dfs(int x, int fa) {
10       for (auto p : e[x]) {
11          auto [y, w] = edge(p);
12          if (y ≠ fa) {
13             dis[y] = dis[x] + w;
14             dfs(y, x);
15          }
16       }
17    }
18
19 public:
20    int v1 = 0, v2 = 0;
21    i64 diameter = 0;
22
23    TreeDiameter(const std::vector<T>& e)
24       : e(e)
25       , dis(e.size()) {
26       dfs(0, -1);
27       v1 = std::max_element(begin(dis), end(dis)) - begin(dis);
28       dis[v1] = 0;
```

```
29      dfs(v1, -1);
30      v2 = std::max_element(begin(dis), end(dis)) - begin(dis);
31      diameter = dis[v2];
32   }
33 };
```

4.3 树哈希

```
1  using u64 = unsigned long long;
2
3  template<class T> struct Rand {
4     std::mt19937 myrand;
5     Rand(const i64 seed = time(0))
6        : myrand(seed) {}
7     T operator()(T l, T r) { return std::uniform_int_distribution<T>(l, r)(
          myrand); }
8  };
9  Rand<u64> rd;
10
11 u64 f(u64 x) {
12    const static u64 r1 = rd(1 << 20, 1 << 24);
13    const static u64 r2 = rd(1 << 25, 1 << 30);
14    const static u64 mask = (1ll << 31) - 1;
15
16    auto h = [&](u64 y) { return (u64)y * y * y * r1 + r2; };
17    return h(x & mask) + h(x >> 31);
18 }
```

4.4 最短路

```
1  template<class T, class G> class Dijkstra
2  {
3     const std::vector<std::vector<std::pair<int, T>>>& e;
4     std::vector<std::vector<G>> dis;
5
6     auto get(int s) {
7        std::vector<G> dis(e.size(), std::numeric_limits<G>::max() / 2);
8
9        using pii = std::pair<G, int>;
10       std::priority_queue<pii, std::vector<pii>, std::greater<>> q;
11
12       dis[s] = G();
13       q.push({dis[s], s});
14
15       while (!q.empty()) {
16          auto [D, x] = q.top();
17          q.pop();
18
19          if (D > dis[x]) continue;
20
21          for (auto&& [y, w] : e[x]) {
```

```
            if (dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                q.push({dis[y], y});
            }
        }
    }
    return dis;
}

public:
Dijkstra(const std::vector<std::vector<std::pair<int, T>>>& g)
    : e(g)
    , dis(g.size()) {}

G operator()(int x, int y) {
    if (dis[x].empty()) dis[x] = get(x);
    return dis[x][y];
}
};
```

4.5 二分图染色

```
struct BiGraphColor {
    std::vector<int> col;
    bool isBiGraph;

    BiGraphColor(const std::vector<std::vector<int>>& e)
        : col(e.size(), -1)
        , isBiGraph(true) {

        int n = e.size();
        std::function<void(int)> dfs = [&](int x) {
            if (!isBiGraph) return;
            for (int y : e[x]) {
                if (col[y] == -1) {
                    col[y] = col[x] ^ 1;
                    dfs(y);
                } else if (col[y] == col[x]) {
                    isBiGraph = false;
                    return;
                }
            }
        };

        for (int i = 0; i < n; i++) {
            if (col[i] == -1) {
                col[i] = 0;
                dfs(i);
            }
            if (!isBiGraph) return;
        }
    }
```

```
};
```

4.6 拓扑排序

```
class TopSort
{
   static constexpr int endPoint(int x) { return x; }
   template<class G> static constexpr int endPoint(const std::pair<int, G>&
       pr) {
      return pr.first;
   }

public:
   template<class T> std::vector<int> operator()(const std::vector<T>& e)
       const {
      int n = e.size();
      std::vector<int> ind(n);
      for (int x = 0; x < n; x++) {
         for (auto p : e[x]) {
            ind[endPoint(p)] += 1;
         }
      }

      std::vector<int> q;
      for (int x = 0; x < n; x++) {
         if (ind[x] == 0) {
            q.push_back(x);
         }
      }

      std::vector<int> res;
      while (!q.empty()) {
         int x = q.back();
         res.push_back(x);
         q.pop_back();

         for (auto p : e[x]) {
            int y = endPoint(p);
            ind[y] -= 1;
            if (ind[y] == 0) {
               q.push_back(y);
            }
         }
      }

      return res;
   }
};
const TopSort topSort;
```

4.7 连通性

4.7.1 强连通分量

```cpp
class SCC
{
   const std::vector<std::vector<int>>& e;
   std::vector<int> q; // stack
   int r = 0, cur = 0;

   void dfs(int x) {
      dfn[x] = low[x] = cur++;
      q[++r] = x;

      for (int y : e[x]) {
         if (dfn[y] == -1) {
            dfs(y);
            low[x] = std::min(low[x], low[y]);
         } else if (bel[y] == -1) {
            low[x] = std::min(low[x], dfn[y]);
         }
      }

      if (dfn[x] == low[x]) {
         int y;
         do {
            y = q[r--];
            bel[y] = cntBlock;
         } while (y != x);
         cntBlock += 1;
      }
   }

public:
   // original graph
   std::vector<int> dfn, low, bel;

   // shrinking graph
   std::vector<std::vector<int>> g;
   int cntBlock = 0;

   SCC(const std::vector<std::vector<int>>& e)
      : e(e)
      , dfn(e.size(), -1)
      , low(e.size())
      , bel(e.size(), -1) {
      int n = e.size();
      q.assign(n + 1, 0);

      for (int i = 0; i < n; i++) {
         if (dfn[i] == -1) {
            dfs(i);
         }
```

```
50        }
51
52        g.resize(cntBlock);
53        for (int x = 0; x < n; x++) {
54           for (int y : e[x]) {
55              if (bel[x] == bel[y]) continue;
56              g[bel[x]].push_back(bel[y]);
57           }
58        }
59
60        // for (int x = 0; x < cntBlock; x++) {
61        // std::sort(begin(g[x]), end(g[x]));
62        // g[x].erase(std::unique(begin(g[x]), end(g[x])), end(g[x]));
63        // }
64     }
65  };
```

4.7.2 割点

```
1   class VertexBC
2   {
3      const std::vector<std::vector<int>>& e;
4      int cur = 0;
5
6      void dfs(int x, int root) {
7         dfn[x] = low[x] = cur++;
8
9         int sonNum = 0;
10        for (int y : e[x]) {
11           if (dfn[y] == -1) {
12              sonNum += 1;
13              dfs(y, root);
14              low[x] = std::min(low[x], low[y]);
15
16              if (low[y] >= dfn[x] and x != root) {
17                 cutDeg[x] += 1;
18              }
19           } else {
20              low[x] = std::min(low[x], dfn[y]);
21           }
22        }
23
24        if (x == root) {
25           cutDeg[x] = sonNum - 1;
26        }
27     }
28
29  public:
30     // original graph
31     std::vector<int> dfn, low, cutDeg;
32     int componentNum = 0;
33
```

```
34     VertexBC(const std::vector<std::vector<int>>& e)
35        : e(e)
36        , dfn(e.size(), -1)
37        , low(e.size())
38        , cutDeg(e.size()) {
39        int n = e.size();
40
41        for (int i = 0; i < n; i++) {
42           if (dfn[i] == -1) {
43              componentNum += 1;
44              dfs(i, i);
45           }
46        }
47     }
48 };
```

4.7.3 割边

```
1  class EdgeBC
2  {
3     const std::vector<std::vector<int>>& e;
4     std::vector<int> q; // stack
5     int r = 0, cur = 0;
6
7     void dfs(int x, int fa) {
8        dfn[x] = low[x] = cur++;
9        q[++r] = x;
10
11       for (int y : e[x]) {
12          if (y == fa) {
13             fa = ~fa;
14             continue;
15          }
16          if (dfn[y] == -1) {
17             dfs(y, x);
18             low[x] = std::min(low[x], low[y]);
19          } else {
20             low[x] = std::min(low[x], dfn[y]);
21          }
22       }
23
24       if (dfn[x] == low[x]) {
25          int y;
26          do {
27             y = q[r--];
28             bel[y] = cntBlock;
29          } while (y != x);
30          cntBlock += 1;
31       }
32    }
33
34 public:
```

```
35    // original graph
36    std::vector<int> dfn, low, bel, cutDeg;
37
38    // shrinking graph
39    std::vector<std::vector<int>> g;
40    int cntBlock = 0, componentNum = 0;
41
42    EdgeBC(const std::vector<std::vector<int>>& e)
43       : e(e)
44       , dfn(e.size(), -1)
45       , low(e.size())
46       , bel(e.size(), -1)
47       , cutDeg(e.size()) {
48       int n = e.size();
49       q.assign(n + 1, 0);
50
51       for (int i = 0; i < n; i++) {
52          if (dfn[i] == -1) {
53             componentNum += 1;
54             dfs(i, -1);
55          }
56       }
57
58       g.resize(cntBlock);
59       for (int x = 0; x < n; x++) {
60          for (int y : e[x]) {
61             if (bel[x] == bel[y]) continue;
62             g[bel[x]].push_back(bel[y]);
63          }
64       }
65    }
66 };
```

4.8 网络流

4.8.1 最大流

```
1  template<class T> struct Flow {
2     const int n;
3
4     std::vector<std::pair<int, T>> e;
5     std::vector<std::vector<int>> g;
6     std::vector<int> cur, dep;
7
8     Flow(int n)
9        : n(n)
10       , g(n) {}
11
12    bool bfs(int s, int t) {
13       dep.assign(n, -1);
14       std::queue<int> q;
15       dep[s] = 0;
```

```
16
17       q.push(s);
18       while (!q.empty()) {
19          const int u = q.front();
20          q.pop();
21
22          for (int i : g[u]) {
23             auto [v, c] = e[i];
24
25             if (c > 0 and dep[v] == -1) {
26                dep[v] = dep[u] + 1;
27                if (v == t) return true;
28                q.push(v);
29             }
30          }
31       }
32
33       return false;
34    }
35
36    T dfs(int u, int t, T f) {
37       if (u == t) {
38          return f;
39       }
40       T res = f;
41       for (int& i = cur[u]; i < g[u].size(); i++) {
42          const int j = g[u][i];
43          auto [v, c] = e[j];
44
45          if (c > 0 and dep[v] == dep[u] + 1) {
46             T out = dfs(v, t, std::min(res, c));
47             e[j].second -= out;
48             e[j ^ 1].second += out;
49
50             res -= out;
51             if (res == 0) {
52                return f;
53             }
54          }
55       }
56       return f - res;
57    }
58
59    void add(int u, int v, T c) {
60       g[u].push_back(e.size());
61       e.emplace_back(v, c);
62       g[v].push_back(e.size());
63       e.emplace_back(u, 0);
64    }
65
66    T work(int s, int t) {
67       T ans = 0;
68       while (bfs(s, t)) {
```

```
69          cur.assign(n, 0);
70          ans += dfs(s, t, std::numeric_limits<T>::max());
71       }
72       return ans;
73    }
74 };
```

4.8.2 最小费用流

```
1  template<class T = i64> struct MCFGraph {
2     struct Edge {
3        int y, c, f;
4     };
5     const int n;
6     std::vector<Edge> e;
7     std::vector<std::vector<int>> g;
8     std::vector<T> h, dis;
9     std::vector<int> pre;
10
11    bool dijkstra(int s, int t) {
12       dis.assign(n, std::numeric_limits<T>::max());
13       pre.assign(n, -1);
14       using pii = std::pair<T, int>;
15       std::priority_queue<pii, std::vector<pii>, std::greater<>> q;
16       dis[s] = 0;
17       q.emplace(0, s);
18
19       while (!q.empty()) {
20          auto [D, x] = q.top();
21          q.pop();
22
23          if (dis[x] < D) continue;
24          for (int i : g[x]) {
25             const auto& [y, c, f] = e[i];
26             if (c > 0 and dis[y] > D + h[x] - h[y] + f) {
27                dis[y] = D + h[x] - h[y] + f;
28                pre[y] = i;
29                q.emplace(dis[y], y);
30             }
31          }
32       }
33       return dis[t] != std::numeric_limits<T>::max();
34    }
35    MCFGraph(int n)
36       : n(n)
37       , g(n) {}
38    void add(int x, int y, int c, int f) {
39       if (f < 0) { // ** 删除 <=> 最大流
40          g[x].push_back(e.size());
41          e.emplace_back(y, 0, f);
42          g[y].push_back(e.size());
43          e.emplace_back(x, c, -f);
```

```
44        } else // **
45           g[x].push_back(e.size()), e.emplace_back(y, c, f), g[y].push_back(
                 e.size()),
46              e.emplace_back(x, 0, -f);
47     }
48     std::pair<int, T> work(int s, int t) {
49        int flow = 0;
50        T cost = 0;
51        h.assign(n, 0);
52        while (dijkstra(s, t)) {
53           for (int i = 0; i < n; ++i) h[i] += dis[i];
54           int aug = std::numeric_limits<int>::max();
55           for (int i = t; i != s; i = e[pre[i] ^ 1].y) aug = std::min(aug, e
                 [pre[i]].c);
56           for (int i = t; i != s; i = e[pre[i] ^ 1].y) {
57              e[pre[i]].c -= aug;
58              e[pre[i] ^ 1].c += aug;
59           }
60           flow += aug;
61           cost += T(aug) * h[t];
62        }
63        return std::pair(flow, cost);
64     }
65  };
```

5 Optimize

5.1 快读快写

```
1  char buf[1 << 22], *p1 = buf, *p2 = buf;
2  // p1 means start-pointer, p2 means end-pointer
3
4  char gc() {
5     if (p1 == p2) p2 = (p1 = buf) + fread(buf, 1, 1 << 21, stdin);
6     return *(p1++);
7  }
8
9  template<typename T> T read() {
10    T sum = 0, fl = 1; // 将 sum,fl 和 ch 以输入的类型定义
11    int ch = gc();
12    for (; !isdigit(ch); ch = gc())
13       if (ch == '-') fl = -1;
14    for (; isdigit(ch); ch = gc()) sum = sum * 10 + ch - '0';
15    return sum * fl;
16 }
17
18 template<typename T> void write(T x) {
19    if (x < 0) x = -x, putchar('-'); // 负数输出
20    static T sta[35];
21    T top = 0;
22    do {
23       sta[top++] = x % 10, x /= 10;
```

```
24      } while (x);
25      while (top) putchar(sta[--top] + '0');
26  }
```

5.2 手写哈希

```
1   struct myhash {
2      static uint64_t hash(uint64_t x) {
3         x += 0x9e3779b97f4a7c15;
4         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
5         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6         return x ^ (x >> 31);
7      }
8      size_t operator()(uint64_t x) const {
9         static const uint64_t SEED = chrono::steady_clock::now().
              time_since_epoch().count();
10        return hash(x + SEED);
11     }
12     size_t operator()(pair<uint64_t, uint64_t> x) const {
13        static const uint64_t SEED = chrono::steady_clock::now().
              time_since_epoch().count();
14        return hash(x.first + SEED) ^ (hash(x.second + SEED) >> 1);
15     }
16  };
```