ICPC Templates

我们需要更深入浅出一些

Mine_King, Tx_Lcy, 369pai

洛谷科技

October 20, 2024

Contents

Chapter 1

字符串

1.1 最小表示法

```
// == Main ==
int i = 0, j = 1, k = 0;
while (k < n && i < n && j < n)
    if (a[(i + k) % n] == a[(j + k) % n]) k++;
    else {
        if (a[(i + k) % n] > a[(j + k) % n]) i = i + k + 1;
        else j = j + k + 1;
        if (i == j) i++;
        k = 0;
    }
ans = min(i, j);
```

1.2 Border 理论

1.2.1 关键结论

定理 1：对于一个字符串 s，若用 t 表示其最长的 Border，则有 $\mathcal{B}(s) = \mathcal{B}(t) \cup \{t\}$。

定理 2：一个字符串的 Border 与 Period 一一对应。具体地，$\mathrm{pre}(s, i) \in \mathcal{B}(s) \iff |s| - i \in \mathcal{P}(s)$。

弱周期引理：

$$\forall p, q \in \mathcal{P}(s), p + q \le |s| \implies \gcd(p, q) \in \mathcal{P}(s)$$

定理 3：若字符串 t 是字符串 s 的前缀，且 $a \in \mathcal{P}(s), b \in \mathcal{P}(t), b \mid a, |t| \geq a$，且 b 是 t 的整周期，则有 $b \in \mathcal{P}(s)$。

周期引理：

$$\forall p, q \in \mathcal{P}(s), p + q - \gcd(p, q) \leq |s| \implies \gcd(p, q) \in \mathcal{P}(s)$$

定理 4：对于文本串 s 和模式串 t，若 $|t| \geq \frac{|s|}{2}$，且 t 在 s 中至少成功匹配了 3 次，则每次匹配的位置形成一个等差数列，且公差为 t 的最小周期。

定理 5：一个字符串 s 的所有长度不小于 $\frac{|s|}{2}$ 的 Border 的长度构成一个等差数列。

定理 6：一个字符串的所有 Border 的长度排序后可以划分成 $\lceil \log_2 |s| \rceil$ 个连续段，使得每段都是一个等差数列。

定理 7：回文串的回文前/后缀即为该串的 Border。

定理 8：若回文串 s 有周期 p，则可以把 $\operatorname{pre}(s, p)$ 划分成长度为 $|s| \bmod p$ 的前缀和长度为 $p - |s| \bmod p$ 的后缀，使得它们都是回文串。

定理 9：若 t 是回文串 s 的最长 Border 且 $|t| \geq \frac{|s|}{2}$，则 t 在 s 中只能匹配 2 次。

定理 10：对于任意一个字符串以及 $u, v \in \operatorname{Ssuf}(s), |u| < |v|$，一定有 u 是 v 的 Border。

定理 11：对于任意一个字符串 s 以及 $u, v \in \operatorname{Ssuf}(s), |u| < |v|$，一定有 $2|u| \leq |v|$。

定理 12：$|\operatorname{Ssuf}(s)| \leq \log_2 |s|$。

1.2.2　KMP

```
// == Main ==
// n is |s|, m is |t|.
for (int i = 1, j = 0; i <= n; i++) {
    while (j && t[j + 1] != s[i]) j = pi[j];
    if (t[j + 1] == s[i])
        if (++j == m) {
```

```
7          // ...
8          j = nxt[j];
9      }
10 }
```

1.3   Z 函数

```
1 // == Main ==
2 // n is |s|.
3 for (int i = 2, j = 0; i <= n; i++) {
4     if (i < j + z[j]) z[i] = min(z[i - j + 1], j + z[j] - i);
5     while (i + z[i] <= n && s[i + z[i]] == s[1 + z[i]]) z[i]++;
6     if (i + z[i] > j + z[j]) j = i;
7 }
```

1.4   Manacher

```
1 // == Main ==
2 // t is the original string, n is |t|.
3 string s = "^#";
4 for (char i : t) s.push_back(i), s.push_back('#');
5 s.push_back ('@');
6 for (int i = 1, j = 0; i <= 2 * n + 1; i++) {
7     if (i <= j + p[j]) p[i] = min (p[2 * j - i], j + p[j] - i);
8     while (s[i - p[i] - 1] == s[i + p[i] + 1]) p[i]++;
9     if (i + p[i] > j + p[j]) j = i;
10 }
```

1.5   AC 自动机

```
1 // == Preparations ==
2 #include <queue>
3 // == Main ==
4 struct ACAM {
5     int tot, fail[200005], delta[200005][26];
6
7     void insert(string s, int id) {
8         int now = 0;
9         for (char c : s) {
10             int v = c - 'a';
11             if (!delta[now][v]) delta[now][v] = ++tot;
```

```
12              now = delta[now][v];
13          }
14          return;
15      }
16      void build() {
17          queue<int> q;
18          for (int c = 0; c < 26; c++)
19              if (delta[0][c]) q.push(delta[0][c]);
20          while (!q.empty()) {
21              int now = q.front();
22              q.pop();
23              for (int c = 0; c < 26; c++)
24                  if (delta[now][c]) fail[delta[now][c]] = delta[fail[now]][c],
                  ↪  q.push(delta[now][c]);
25                  else delta[now][c] = delta[fail[now]][c];
26          }
27          return;
28      }
29  } ac;
```

1.6   回文自动机 PAM

```
1  // == Main ==
2  struct PAM {
3      int tot, delta[500005][26], len[500005], fail[500005], ans[500005];
4      string s;
5      int lst;
6
7      PAM() {tot = 1; len[0] = 0; len[1] = -1; fail[0] = fail[1] = 1;}
8      int getfail(int now, int i) {
9          while (s[i - len[now] - 1] != s[i]) now = fail[now];
10          return now;
11      }
12      void insert(int i) {
13          int now = getfail(lst, i);
14          if (!delta[now][s[i] - 'a']) {
15              len[++tot] = len[now] + 2;
16              fail[tot] = delta[getfail(fail[now], i)][s[i] - 'a'];
17              delta[now][s[i] - 'a'] = tot;
18              ans[tot] = ans[fail[tot]] + 1;
19          }
20          lst = delta[now][s[i] - 'a'];
21          return;
22      }
23  } p;
```

1.7 后缀自动机

1.7.1 普通 SAM

```cpp
// == Main ==
struct SAM {
    int tot, lst;
    int len[2000005], siz[2000005], link[2000005];
    int delta[2000005][26];

    SAM() {link[0] = -1;}
    void insert(char ch) {
        int c = ch - 'a', now = ++tot;
        len[now] = len[lst] + 1;
        siz[now] = 1;
        for (int p = lst; p != -1; p = link[p])
            if (!delta[p][c]) delta[p][c] = tot;
            else if (len[delta[p][c]] == len[p] + 1) {link[now] = delta[p][c];
            ↪  break;}
            else {
                int q = delta[p][c], v = ++tot;
                len[v] = len[p] + 1;
                memcpy(delta[v], delta[q], sizeof(delta[v]));
                link[v] = link[q], link[q] = v, link[now] = v;
                for (int i = p; delta[i][c] == q; i = link[i]) delta[i][c] = v;
                break;
            }
        lst = now;
        return ;
    }
} sam;
```

1.7.2 广义 SAM

注意自动机空间要开 Trie 的两倍。

```cpp
// == Main ==
struct GSAM {
    int tot;
    int delta[2000005][26], link[2000005], len[2000005];
    struct Trie {
        int tot, trie[1000005][26], st[1000005];

        void insert(string s) {
            int now = 0;
            for (char c : s) {
```

```
11                    int id = c - 'a';
12                    if (!trie[now][id]) trie[now][id] = ++tot;
13                    now = trie[now][id];
14                }
15                return;
16            }
17        } tr;
18
19        GSAM() {link[0] = -1;}
20        int insert(int c, int lst) {
21            int now = ++tot;
22            len[now] = len[lst] + 1;
23            for (int p = lst; p != -1; p = link[p])
24                if (!delta[p][c]) delta[p][c] = now;
25                else if (len[delta[p][c]] == len[p] + 1) {link[now] = delta[p][c];
      ↪   break;}
26                else {
27                    int q = delta[p][c], v = ++tot;
28                    len[v] = len[p] + 1;
29                    memcpy(delta[v], delta[q], sizeof(delta[v]));
30                    link[v] = link[q], link[q] = v, link[now] = v;
31                    for (int i = p; i != -1 && delta[i][c] == q; i = link[i]) delta[i][c]
      ↪   = v;
32                    break;
33                }
34            return now;
35        }
36        void build() {
37            queue<int> q;
38            tr.st[0] = 0;
39            q.push(0);
40            while (!q.empty()) {
41                int now = q.front();
42                q.pop();
43                for (int i = 0; i < 26; i++)
44                    if (tr.trie[now][i])
45                        tr.st[tr.trie[now][i]] = insert(i, tr.st[now]),
      ↪   q.push(tr.trie[now][i]);
46            }
47            return;
48        }
49    } gsam;
```

1.8 后缀排序

```
1  // == Preparations ==
2  int sa[2000005], rk[2000005], b[1000005], cp[2000005];
3  // == Main ==
4  for (int i = 1; i <= n; i++) b[rk[i] = s[i]]++;
5  for (int i = 1; i < 128; i++) b[i] += b[i - 1];
6  for (int i = n; i >= 1; i--) sa[b[rk[i]]--] = i;
7  memcpy(cp, rk, sizeof(cp));
8  for (int i = 1, j = 0; i <= n; i++)
9      if (cp[sa[i]] == cp[sa[i - 1]]) rk[sa[i]] = j;
10     else rk[sa[i]] = ++j;
11 for (int w = 1; w < n; w <<= 1) {
12     memcpy(cp, sa, sizeof(cp));
13     memset(b, 0, sizeof(b));
14     for (int i = 1; i <= n; i++) b[rk[cp[i] + w]]++;
15     for (int i = 1; i <= n; i++) b[i] += b[i - 1];
16     for (int i = n; i >= 1; i--) sa[b[rk[cp[i] + w]]--] = cp[i];
17     memcpy(cp, sa, sizeof(cp));
18     memset(b, 0, sizeof(b));
19     for (int i = 1; i <= n; i++) b[rk[cp[i]]]++;
20     for (int i = 1; i <= n; i++) b[i] += b[i - 1];
21     for (int i = n; i >= 1; i--) sa[b[rk[cp[i]]]--] = cp[i];
22     memcpy(cp, rk, sizeof(cp));
23     for (int i = 1, j = 0; i <= n; i++)
24         if (cp[sa[i]] == cp[sa[i - 1]] && cp[sa[i] + w] == cp[sa[i - 1] + w])
           ↪  rk[sa[i]] = j;
25         else rk[sa[i]] = ++j;
26 }
```

Chapter 2

数论

2.1 Miller Rabin 和 Pollard Rho

2.1.1 Miller Rabin

```
1  // == Preparations ==
2  const int prime[] = {2, 3, 5, 7, 9, 11, 13, 17, 19, 23, 29, 31, 37};
3
4  long long power(long long a, long long b, long long mod) {
5      long long ans = 1;
6      while (b) {
7          if (b & 1) ans = (__int128)ans * a % mod;
8          a = (__int128)a * a % mod;
9          b >>= 1;
10     }
11     return ans % mod;
12 }
13 // == Main ==
14 inline int Miller_Rabin(long long n) {
15     if (n == 1) return 0;
16     if (n == 2) return 1;
17     if (n % 2 == 0) return 0;
18     long long u = n - 1, t = 0;
19     while (u % 2 == 0) u /= 2, t++;
20     for (int i = 0; i < 12; i++) {
21         if (prime[i] % n == 0) continue;
22         long long x = power(prime[i] % n, u, n);
23         if (x == 1) continue;
24         int flag = 0;
25         for (int j = 1; j <= t; j++) {
26             if (x == n - 1) {flag = 1; break;}
27             x = (__int128)x * x % n;
28         }
29         if (!flag) return 0;
```

```
30      }
31      return 1;
32 }
```

2.1.2 Pollard Rho

```
1  // == Preparations ==
2  #include <chrono>
3  #include <random>
4
5  mt19937_64 gen(chrono::system_clock::now().time_since_epoch().count());
6  /*
7  Miller Rabin
8  */
9  // == Main ==
10 long long Pollard_Rho(long long n) {
11     long long s = 0, t = 0, c = gen() % (n - 1) + 1;
12     for (int goal = 1; ; goal <<= 1, s = t) {
13         long long val = 1;
14         for (int step = 1; step <= goal; step++) {
15             t = ((__int128)t * t + c) % n;
16             val = (__int128)val * abs(t - s) % n;
17             if (!val) return n;
18             if (step % 127 == 0) {
19                 long long d = __gcd(val, n);
20                 if (d > 1) return d;
21             }
22         }
23         long long d = __gcd(val, n);
24         if (d > 1) return d;
25     }
26 }
27 void factor(long long n) {
28     if (n < ans) return ;
29     if (n == 1 || Miller_Rabin(n)) {
30         // n = 1 或 n 是一个质因子。
31         // ...
32     }
33     long long p;
34     do p = Pollard_Rho(n);
35     while (p == n);
36     while (n % p == 0) n /= p;
37     factor(n), factor(p);
38     return ;
39 }
```

2.2 常用数论算法

2.2.1 exgcd

求出来的解满足 $|x| \leq b, |y| \leq a$。

```cpp
// == Main ==
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {x = 1; y = 0; return a;}
    int m = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return m;
}
```

2.2.2 CRT

没用。

```cpp
// == Preparations ==
#include <vector>
// == Main ==
int CRT(vector<pair<int, int>> &a) {
    int M = 1;
    for (auto i : a) M *= i.second;
    int res = 0;
    for (auto i : a) {
        int t = inv(M / i.second, i.second);
        res = (res + (long long)i.first * (M / i.second) % M * t % M) % M;
    }
    return res;
}
```

2.2.3 exCRT

需保证 lcm 在 long long 范围内。

```cpp
// == Preparations ==
long long exgcd(long long a, long long b, long long &x, long long &y);
// == Main ==
long long exCRT(vector<pair<long long, long long>> vec) {
    long long ans = vec[0].first, mod = vec[0].second;
    for (int i = 1; i < (int)vec.size(); i++) {
        long long a = mod, b = vec[i].second, c = vec[i].first - ans % b;
        long long x, y;
```

```
 9        long long g = exgcd(a, b, x, y);
10        if (c % g != 0) return -1;
11        b /= g;
12        x = (__int128)x * (c / g) % b;
13        ans += x * mod;
14        mod *= b;
15        ans = (ans % mod + mod) % mod;
16    }
17    return ans;
18 }
```

2.2.4 exLucas

```
 1 // == Preparations ==
 2 int power(int a, long long b, int mod);
 3 int exgcd(int a, int b, int &x, int &y);
 4 int CRT(vector<pair<int, int>> &a);
 5 // == Main ==
 6 int inv(int n, int p) {
 7    int x, y;
 8    exgcd(n, p, x, y);
 9    return (x % p + p) % p;
10 }
11 int fac(long long n, int p, int pk) {
12    if (n == 0) return 1;
13    int res = 1;
14    for (int i = 1; i < pk; i++)
15        if (i % p != 0) res = (long long)res * i % pk;
16    res = power(res, n / pk, pk);
17    for (int i = 1; i <= n % pk; i++)
18        if (i % p != 0) res = (long long)res * i % pk;
19    return (long long)res * fac(n / p, p, pk) % pk;
20 }
21 int C(long long n, long long m, int p, int pk) {
22    long long x = n, y = m, z = n - m;
23    int res = (long long)fac(x, p, pk) * inv(fac(y, p, pk), pk) % pk * inv(fac(z, p,
      ↪  pk), pk) % pk;
24    long long e = 0;
25    while (x) e += x / p, x /= p;
26    while (y) e -= y / p, y /= p;
27    while (z) e -= z / p, z /= p;
28    return (long long)res * power(p, e, pk) % pk;
29 }
30 int exLucas(long long n, long long m, int p) {
31    vector<pair<int, int>> a;
32    for (int i = 2; i * i <= p; i++)
33        if (p % i == 0) {
```

```
34          int pk = 1;
35          while (p % i == 0) pk *= i, p /= i;
36          a.emplace_back(C(n, m, i, pk), pk);
37      }
38    if (p != 1) a.emplace_back(C(n, m, p, p), p);
39    return CRT(a);
40 }
```

2.3 万能欧几里得算法

> 问题描述：
> 给出一个幺半群 $(S, \cdot)$ 和元素 $u, r \in S$，以及一条直线 $y = \frac{ax+b}{c}$。
> 画出平面中所有坐标为正整数的横线和竖线，维护一个 f，初值为单位元 e。
> 从原点出发，先向 y 轴正方向走直到到达直线与 y 的交点，然后沿直线走一直走到与 $x = n$ 的交点为止。
> 每当经过一条横线时，执行 $f \leftarrow fu$，经过一条竖线时执行 $f \leftarrow fr$。特别地，在 y 轴上行走时不考虑竖线，同时经过横线和竖线时先执行前者。
> 求最终的 f。记为 $\text{euclid}(a, b, c, n, u, r)$。
> 其中 $a, b \geq 0, n, c > 0$。

做法：

$$\text{euclid}(a, b, c, n, u, r) = \begin{cases} r^n & m = 0 \\ u^{\lfloor \frac{b}{c} \rfloor} \cdot \text{euclid}(a \bmod c, b \bmod c, c, n, u, u^{\lfloor \frac{a}{c} \rfloor} r) & a \geq c \vee b \geq c \\ r^{\lfloor \frac{c-b-1}{a} \rfloor} u \cdot \text{euclid}(c, (c-b-1) \bmod a, a, m-1, r, u) \cdot r^{n - \lfloor \frac{cm-b-1}{a} \rfloor} & \text{otherwise} \end{cases}$$

设一次乘法的复杂度为 $O(T)$，则复杂度为 $O(T \log(a+c) \log(a+n+c))$。

```
1  // == Preparations ==
2  struct Node {
3      // ...
4
5      Node operator*(const Node &x) const {
6          // ...
7      }
8  };
9  // == Main ==
10 Node power(Node a, long long b) {
11     Node ans = Node(/* 幺元 */);
12     while (b) {
13         if (b & 1) ans = ans * a;
14         a = a * a;
```

```
15          b >>= 1;
16      }
17      return ans;
18  }
19  Node Euclid(int a, int b, int c, long long n, Node r, Node u) {
20      long long m = (a * n + b) / c;
21      if (!m) return power(r, n);
22      if (a >= c || b >= c)
23          return power(u, b / c) * Euclid(a % c, b % c, c, n, power(u, a / c) * r, u);
24      return power(r, (c - b - 1) / a) * u *
25          Euclid(c, (c - b - 1) % a, a, m - 1, u, r) * power(r, n - (c * m - b - 1) /
            ↪    a);
26  }
```

Chapter 3

数据结构

3.1　K-D Tree

```cpp
// == Main ==
template<const int Dim = 2>
struct KDTree {
    using point = array<int, Dim>;
    struct node {
        point p, l, r;
        int val, siz, sum;
        node *ls, *rs;

        node() = default;
        node(point _p, int _val = 0):
            p(_p), l(_p), r(_p), val(_val), siz(1), sum(_val), ls(nullptr),
            ↪  rs(nullptr) {}
        void pushup() {
            l = r = p, siz = 1, sum = val;
            for (int i = 0; i < Dim; i++) {
                if (ls) l[i] = min(l[i], ls->l[i]), r[i] = max(r[i], ls->r[i]);
                if (rs) l[i] = min(l[i], rs->l[i]), r[i] = max(r[i], rs->r[i]);
            }
            if (ls) siz += ls->siz, sum += ls->sum;
            if (rs) siz += rs->siz, sum += rs->sum;
            return ;
        }
    };
    vector<node *> root;
    using itor = typename vector<node>::iterator;

    node *build(itor l, itor r, int dim = 0) {
        if (l == r) return nullptr;
        int mid = (r - l) / 2;
        nth_element(l, l + mid, r, [&dim](const node &x, const node &y) {return
            ↪  x.p[dim] < y.p[dim];});
```

```
31          node *now = new node(*(l + mid));
32          now->ls = build(l, l + mid, (dim + 1) % Dim);
33          now->rs = build(l + mid + 1, r, (dim + 1) % Dim);
34          now->pushup();
35          return now;
36      }
37      void getnode(node *now, vector<node> &vec) {
38          if (!now) return ;
39          vec.push_back(*now);
40          getnode(now->ls, vec), getnode(now->rs, vec);
41          delete now;
42          return ;
43      }
44      void insert(point p, int val) {
45          vector<node> tmp({node(p, val)});
46          while (!root.empty() && root.back()->siz == (int)tmp.size())
47              getnode(root.back(), tmp), root.pop_back();
48          sort(tmp.begin(), tmp.end(), [](const node &x, const node &y) {return x.p <
    ↪     y.p;});
49          vector<node> vec;
50          for (node i : tmp)
51              if (!vec.empty() && vec.back().p == i.p) vec.back().val += i.val;
52              else vec.push_back(i);
53          root.push_back(build(vec.begin(), vec.end()));
54          return ;
55      }
56      int query(point ll, point rr, node *now) {
57          if (!now) return 0;
58          int flag = 1;
59          for (int i = 0; i < Dim; i++)
60              if (now->r[i] < ll[i] || now->l[i] > rr[i]) return 0;
61              else flag &= ll[i] <= now->l[i] && now->r[i] <= rr[i];
62          if (flag) return now->sum;
63          flag = 1;
64          for (int i = 0; i < Dim; i++) flag &= ll[i] <= now->p[i] && now->p[i] <=
    ↪     rr[i];
65          return flag * now->val + query(ll, rr, now->ls) + query(ll, rr, now->rs);
66      }
67      int query(point ll, point rr) {
68          int ans = 0;
69          for (node *rt : root) ans += query(ll, rr, rt);
70          return ans;
71      }
72      ~KDTree() {
73          vector<node> tmp;
74          for (node *rt : root) getnode(rt, tmp);
75      }
76 };
```

3.2 Link Cut Tree

代码维护的是点权异或和。

```cpp
// == Main ==
struct LinkCutTree {
    int fa[100005], son[100005][2], siz[100005], swp[100005];
    int val[100005], Xor[100005];

    void pushup(int now) {
        siz[now] = siz[son[now][0]] + siz[son[now][1]] + 1;
        Xor[now] = Xor[son[now][0]] ^ val[now] ^ Xor[son[now][1]]; // 此处更新信息。
        return;
    }
    void pushdown(int now) {
        if(!swp[now]) return ;
        swap(son[now][0], son[now][1]);
        swp[son[now][0]] ^= 1, swp[son[now][1]] ^= 1;
        swp[now] = 0;
        // 此处将信息 pushdown
        return;
    }
    int isRoot(int now) {return now != son[fa[now]][0] && now != son[fa[now]][1];}
    int get(int now) {return now == son[fa[now]][1];}
    void rotate(int x) {
        int y = fa[x], z = fa[fa[x]], chk = get(x);
        if (!isRoot(y)) son[z][get(y)] = x;
        son[y][chk] = son[x][chk ^ 1], fa[son[x][chk ^ 1]] = y;
        son[x][chk ^ 1] = y, fa[y] = x;
        fa[x] = z;
        pushup(y), pushup(x);
        return;
    }
    void splay(int now) {
        vector<int> stk;
        stk.push_back(now);
        for (int i = now; !isRoot(i); i = fa[i]) stk.push_back(fa[i]);
        while (!stk.empty()) pushdown(stk.back()), stk.pop_back();
        for (int f; f = fa[now], !isRoot(now); rotate(now))
            if (!isRoot(f)) rotate(get(f) == get(now) ? f : now);
        return;
    }
    void access(int now) { // 打通到根的链
        for (int lst = 0; now; lst = now, now = fa[now]) splay(now), son[now][1] =
        ↪  lst, pushup(now);
        return;
    }
```

```
43    void makeRoot(int now) {access(now); splay(now); swp[now] ^= 1; return;} // 设置
   ↪    根
44    void link(int u, int v) {makeRoot(u); fa[u] = v; return;} // 连接
45    void cut(int u, int v) {makeRoot(u); access(v); splay(v); son[v][0] = fa[u] = 0;
   ↪    return;} // 切割
46    int find(int now) { // 找根
47        access(now), splay(now);
48        pushdown(now);
49        while (son[now][0]) now = son[now][0], pushdown(now);
50        splay(now);
51        return now;
52    }
53    void split(int u, int v) {makeRoot(u); access(v); splay(u); return;} // 剖出 u ~
   ↪    v 的链
54    void update(int u, int _val) {split(u, u); val[u] = Xor[u] = _val; return ;} //
   ↪    修改操作，split 后做就行了，此处为单点修改。
55    int query(int u, int v) {split(u, v); return Xor[u];} // 查询操作，split 后做就行
   ↪    了。
56    int isConnected(int u, int v) {return find(u) == find(v);} // 查询两个点是否连通。
57 };
```

Chapter 4

图论

4.1   Tarjan

4.1.1   强连通分量

```cpp
// == Main ==
void Tarjan(int now) {
    dfn[now] = low[now] = ++Index;
    s.push(now);
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (!dfn[g.to[i]]) {
            Tarjan(g.to[i]);
            low[now] = min(low[now], low[g.to[i]]);
        } else if (!scc[g.to[i]]) low[now] = min(low[now], dfn[g.to[i]]);
    if (low[now] == dfn[now]) {
        scc_cnt++;
        for (int x = 0; x != now; s.pop()) {
            x = s.top();
            scc[x] = scc_cnt;
        }
    }
    return ;
}
```

4.1.2   割边与边双

割边：

```cpp
// == Main ==
void Tarjan(int now, int fa) {
    dfn[now] = low[now] = ++Index;
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (!dfn[g.to[i]]) {
```

```
6            Tarjan(g.to[i], now);
7            low[now] = min(low[now], low[g.to[i]]);
8            if (low[g.to[i]] > dfn[now])
9                printf("A Bridge of the Input Garph is (%d, %d)\n", now, g.to[i]);
10        } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
11    return ;
12 }
```

边双：

```
1  // == Main ==
2  void Tarjan(int now, int fa) {
3      dfn[now] = low[now] = ++Index;
4      s.push(now);
5      for (int i = g.hd[now]; i; i = g.nxt[i])
6          if (!dfn[g.to[i]]) {
7              Tarjan(g.to[i], now);
8              low[now] = min(low[now], low[g.to[i]]);
9          } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
10     if (low[now] == dfn[now]) {
11         bcc_cnt++;
12         for (int x = 0; x != now; s.pop()) {
13             x = s.top();
14             bcc[x] = bcc_cnt;
15         }
16     }
17     return ;
18 }
```

4.1.3 割点与点双

割点：

```
1  // == Main ==
2  void Tarjan(int now, int root) {
3      dfn[now] = low[now] = ++Index;
4      int sons=0, flag=0;
5      for (int i=g.hd[now]; i; i = g.nxt[i], sons++)
6          if (!dfn[g.to[i]]) {
7              Tarjan(g.to[i], now);
8              low[now] = min(low[now], low[g.to[i]]);
9              if (now!=root && low[g.to[i]] == dfn[now] && !flag)
10                 printf("A Cut Vertex of the Input Graph is %d.", now), flag=1;
11         } else low[now] = min(low[now], dfn[g.to[i]]);
12     if (now == root && sons >= 2)
13         printf("A Cut Vertex of the Input Graph is %d.", now);
```

```
14      return ;
15  }
```

点双：

```
1   // == Main ==
2   void Tarjan(int now) {
3       dfn[now] = low[now] = ++Index;
4       s.push(now);
5       for (int i = g.hd[now]; i; i = g.nxt[i], sons++)
6           if (!dfn[g.to[i]]) {
7               Tarjan(g.to[i]);
8               low[now] = min(low[now], low[g.to[i]]);
9               if (low[g.to[i]] == dfn[now]) {
10                  printf("BCC #%d:\n", ++bcc_cnt);
11                  for (int x = 0; x != g.to[i]; s.pop())
12                      printf("%d ", x = s.top());
13                  printf("%d\n", now);
14              }
15          } else low[now] = min(low[now], dfn[g.to[i]]);
16      return ;
17  }
```

4.2 欧拉路径

```
1   // == Preparations ==
2   #include <vector>
3
4   vector<int> g[100005], ans;
5   // == Main ==
6   void dfs(int now) {
7       while (!g[now].empty()) {
8           int to = g[now].back();
9           g[now].pop_back();
10          dfs(to);
11      }
12      ans.push_back(now);
13      return ;
14  }
```

4.3 二分图匹配

4.3.1 最大匹配

```
1  // == Preparations ==
2  int chos[100005], vis[100005];
3  struct graph {/* ... */} g;
4  // == Main ==
5  int dfs(int now) {
6      for (int i = g.hd[now]; i; i = g.nxt[i]) {
7          if (vis[g.to[i]]) continue;
8          vis[g.to[i]] = true;
9          if (!chos[g.to[i]] || dfs(chos[g.to[i]])) {
10             chos[g.to[i]] = now;
11             return 1;
12         }
13     }
14     return 0;
15 }
16
17 for (int i = 1; i <= n; i++) {
18     memset(vis, 0, sizeof(vis));
19     ans += dfs(i);
20 }
```

4.3.2 最大权匹配

```
1  // == Preparations ==
2  int vis[1005], mat[1005], pre[1005];
3  long long g[505][1005];
4  long long w[1005], slack[1005];
5  // edge: g[u][n + v] = w;
6  // == Main ==
7  for (int i = 1; i <= n; i++) {
8      w[i] = ~0x3f3f3f3f3f3f3f3f;
9      for (int j = n + 1; j <= n + n; j++) w[i] = max(w[i], (long long)g[i][j]);
10 }
11 for (int i = 1; i <= n; i++) {
12     memset(vis, 0, sizeof(vis));
13     memset(slack, 0x3f, sizeof(slack));
14     memset(pre, 0, sizeof(pre));
15     int now = i, ri = 0;
16     while (1) {
17         int id = 0;
18         long long delta = 0x3f3f3f3f3f3f3f3f;
19         for (int j = n + 1; j <= n + n; j++)
```

```
20              if (!vis[j]) {
21                  long long val = w[now] + w[j] - g[now][j];
22                  if (val < slack[j]) slack[j] = val, pre[j] = ri;
23                  if (slack[j] < delta) delta = slack[j], id = j;
24              }
25          w[i] -= delta;
26          for (int j = n + 1; j <= n + n; j++)
27              if (vis[j]) w[j] += delta, w[mat[j]] -= delta;
28              else slack[j] -= delta;
29          vis[ri = id] = 1;
30          if (mat[ri]) now = mat[ri];
31          else break;
32      }
33      while (ri) {
34          mat[ri] = mat[pre[ri]];
35          if (!pre[ri]) {mat[ri] = i; break;}
36          ri = pre[ri];
37      }
38 }
39 long long ans = 0;
40 for (int i = 1; i <= n + n; i++) ans += w[i];
41 printf("%lld\n", ans);
42 for (int i = n + 1; i <= n + n; i++) printf("%d ", mat[i]);
43 puts("");
```

4.4 网络流

4.4.1 最大流

```
1 // == Preparations ==
2 #include <queue>
3 // == Main ==
4 struct Dinic {
5     int s, t;
6     struct graph {
7         int tot, hd[205];
8         int nxt[10005], to[10005], dt[10005];
9         graph() {tot = 1;}
10         void add(int u, int v, int w) {
11             nxt[++tot] = hd[u];
12             hd[u] = tot;
13             to[tot] = v;
14             dt[tot] = w;
15             return ;
16         }
17     } g;
```

22

```
18        int cur[205], dis[205];
19
20        void add_edge(int u, int v, int f) {g.add(u, v, f), g.add(v, u, 0); return;}
21        int bfs() {
22            memset(dis, 0, sizeof(dis));
23            queue<int>q;
24            q.push(s);
25            dis[s] = 1;
26            while (!q.empty()) {
27                int now = q.front();
28                q.pop();
29                cur[now] = g.hd[now];
30                for (int i = g.hd[now]; i; i = g.nxt[i])
31                    if (g.dt[i] && !dis[g.to[i]]) dis[g.to[i]] = dis[now] + 1,
                    ↪   q.push(g.to[i]);
32            }
33            return dis[t];
34        }
35        long long dinic(int now, long long flow) {
36            if (now == t) return flow;
37            long long used = 0;
38            for (int i = cur[now]; i && used < flow; i = g.nxt[i])
39                if (g.dt[i] && dis[g.to[i]] == dis[now] + 1) {
40                    long long k = dinic(g.to[i], min(flow - used, (long long)g.dt[i]));
41                    g.dt[i] -= k, g.dt[i ^ 1] += k;
42                    used += k;
43                    cur[now] = i;
44                }
45            if (used == 0) dis[now] = 0;
46            return used;
47        }
48        long long solve() {
49            long long ans = 0;
50            while (bfs()) ans += dinic(s, 0x3f3f3f3f3f3f3f3f);
51            return ans;
52        }
53  } F;
```

4.4.2 费用流

原始对偶：

```
1  // == Preparations ==
2  #include <queue>
3  // == Main ==
4  struct PrimalDual {
5      int n, s, t;
```

```
 6    struct graph {
 7        int tot, hd[805];
 8        int nxt[30005], to[30005], flw[30005], cst[30005];
 9
10        graph() {tot = 1;}
11        void add(int u, int v, int f, int c) {
12            nxt[++tot] = hd[u];
13            hd[u] = tot;
14            to[tot] = v;
15            flw[tot] = f;
16            cst[tot] = c;
17            return ;
18        }
19    } g;
20    int h[805], dis[805], f[805], pre[805];
21    struct node {
22        int id, val;
23
24        node() = default;
25        node(int _id, int _val): id(_id), val(_val) {}
26        bool operator<(const node &x) const {return val > x.val;}
27    };
28
29    void add_edge(int u, int v, int f, int c) {g.add(u, v, f, c), g.add(v, u, 0, -c);
   ↪    return;}
30    void spfa() {
31        queue<int> q;
32        memset(h, 0x3f, sizeof(h));
33        h[s] = 0;
34        q.push(s);
35        while (!q.empty()) {
36            int now = q.front();
37            q.pop();
38            f[now] = 0;
39            for (int i = g.hd[now]; i; i = g.nxt[i])
40                if (g.flw[i] && h[g.to[i]] > h[now] + g.cst[i]) {
41                    h[g.to[i]] = h[now] + g.cst[i];
42                    if (!f[g.to[i]]) q.push(g.to[i]), f[g.to[i]] = 1;
43                }
44        }
45        return ;
46    }
47    int dijkstra() {
48        priority_queue<node> q;
49        memset(dis, 0x3f, sizeof(dis));
50        memset(pre, 0, sizeof(pre));
51        q.emplace(s, dis[s] = 0);
52        while (!q.empty()) {
53            int now = q.top().id, tmp = q.top().val;
```

```
54              q.pop();
55              if (dis[now] != tmp) continue;
56              for (int i = g.hd[now]; i; i = g.nxt[i])
57                  if (g.flw[i] && dis[g.to[i]] > dis[now] + g.cst[i] + h[now] -
                     ↪  h[g.to[i]]) {
58                      q.emplace(g.to[i], dis[g.to[i]] = dis[now] + g.cst[i] + h[now] -
                         ↪  h[g.to[i]]);
59                      pre[g.to[i]] = i ^ 1;
60                  }
61          }
62          return pre[t];
63      }
64      pair<int, int> solve() {
65          int flow = 0, cost = 0;
66          spfa();
67          while (dijkstra()) {
68              for (int i = 1; i <= n; i++)
69                  if (dis[i] < 0x3f3f3f3f) h[i] += dis[i];
70              int mnflow = 0x3f3f3f3f;
71              for (int i = t; i != s; i = g.to[pre[i]]) mnflow = min(mnflow,
                 ↪  g.flw[pre[i] ^ 1]);
72              for (int i = t; i != s; i = g.to[pre[i]]) g.flw[pre[i] ^ 1] -= mnflow,
                 ↪  g.flw[pre[i]] += mnflow;
73              flow += mnflow;
74              cost += mnflow * h[t];
75          }
76          return {flow, cost};
77      }
78 } F;
```

4.4.3 上下界

> $f(u, v)$ 表示边 (u, v) 的流量，$f(u)$ 表示 u 的出流减入流，$c(u, v)$ 表示边 (u, v) 的容量。
> 对于每条边给定一个流量下界 $b(u, v)$，需要额外满足 $\forall (u, v), b(u, v) \le f(u, v) \le c(u, v)$。

无源汇上下界可行流

没有源点和汇点，对于所有点满足 $f(u) = 0$，求一个可行的流。

先强制每条边流到流量下界，建立虚拟源汇点 s, t，对于每个点 u 考虑此时的净流量：

- $f(u) = 0$：满足条件，不用管。

- $f(u) > 0$：出流大于入流，从 u 向 t 连容量为 $f(u)$ 的边。

- $f(u) < 0$：入流大于出流，从 s 向 u 连容量为 $-f(u)$ 的边。

将原图中每条边的容量设为 $c(u,v) - b(u,v)$，则从 s 到 t 的流相当于增加调整流量的过程。

若 s 的出边流满（等同于 t 的入边流满），则找到了一条可行流。

有源汇上下界可行流

连一条 t 到 s 容量正无穷下界为 0 的边，然后跑无源汇上下界可行流即可，流量为新增边的流量。

有源汇上下界最大流

求出可行流后删掉 t 到 s 的边，在残量网络上跑 s 到 t 的最大流，该最大流加上原本的可行流即为答案。

有源汇上下界最小流

同理，改成求 t 到 s 的最大流，原可行流减去该最大流即为答案。

有源汇上下界最小费用流

做法是一样的，所有新增边费用为 0。

需要注意求最小流时需要改成费用最大。

4.4.4 有负圈的最小费用最大流

先钦定所有负圈边流满，即上下界均为流量。然后对于负边建反向、容量相同、费用为相反数的边用于退流原边。

这样就转化成了有源汇上下界最小费用最大流。

4.5 k 短路

复杂度为 $O((n+m)\log n + k\log k)$。

```cpp
// == Preparations ==
int ontree[200005];
struct graph {
    int tot, hd[5005];
    int nxt[200005], to[200005];
    long long dt[200005];

    void add(int u, int v, long long w) {
        nxt[++tot] = hd[u];
        hd[u] = tot;
        to[tot] = v;
        dt[tot] = w;
        return ;
    }
```

```cpp
15  } g;
16  long long dis[5005];
17  struct node {
18      int id;
19      long long val;
20
21      node() = default;
22      node(int _id, long long _val): id(_id), val(_val) {}
23      bool operator<(const node &x) const {return val > x.val;}
24  };
25  priority_queue<node> q;
26  int vis[5005];
27
28  // 以下左偏树
29  struct HeapNode {
30      long long val;
31      int to, dist;
32      HeapNode *ls, *rs;
33
34      HeapNode() = default;
35      HeapNode(long long _val, int _to): val(_val), to(_to), dist(1), ls(nullptr),
        ↪   rs(nullptr) {}
36  };
37  struct Heap {
38      HeapNode *root[5005];
39
40      HeapNode *merge(HeapNode *u, HeapNode *v) {
41          if (!u) return v;
42          if (!v) return u;
43          if (u->val > v->val) swap(u, v);
44          HeapNode *p = new HeapNode(*u);
45          p->rs = merge(u->rs, v);
46          if (!p->ls || p->ls->dist < p->rs->dist) swap(p->ls, p->rs);
47          if (p->rs) p->dist = p->rs->dist + 1;
48          else p->dist = 1;
49          return p;
50      }
51  } h;
52
53  struct Node {
54      HeapNode *id;
55      long long val;
56
57      Node() = default;
58      Node(HeapNode *_id, long long _val): id(_id), val(_val) {}
59      bool operator<(const Node &x) const {return val > x.val;}
60  };
61  priority_queue<Node> Q;
62  // == Main ==
```

```
63  void dfs(int now) {
64      vis[now] = 1;
65      for (int i = g.hd[now]; i; i = g.nxt[i])
66          if (!vis[g.to[i]] && dis[g.to[i]] == dis[now] + g.dt[i]) ontree[i] = 1,
        ↪   dfs(g.to[i]);
67      return ;
68  }
69  void dfs2(int now) {
70      for (int i = g.hd[now]; i; i = g.nxt[i])
71          if (ontree[i]) h.root[g.to[i]] = h.merge(h.root[g.to[i]], h.root[now]),
        ↪   dfs2(g.to[i]);
72      return;
73  }
74
75  memset(dis, 0x3f, sizeof(dis));
76  q.emplace(n, dis[n] = 0);
77  while (!q.empty()) {
78      int now = q.top().id;
79      long long tmp = q.top().val;
80      q.pop();
81      if (tmp != dis[now]) continue;
82      for (int i = g.hd[now]; i; i = g.nxt[i])
83          if (dis[g.to[i]] > dis[now] + g.dt[i]) q.emplace(g.to[i], dis[g.to[i]] =
        ↪   dis[now] + g.dt[i]);
84  }
85  dfs(n);
86  for (int i = 1; i <= n; i++)
87      for (int j = g.hd[i]; j; j = g.nxt[j])
88          if (!ontree[j] && g.to[j] != n)
89              h.root[g.to[j]] = h.merge(h.root[g.to[j]], new HeapNode(dis[i] + g.dt[j]
              ↪   - dis[g.to[j]], i));
90  dfs2(n);
91  if (h.root[1]) Q.emplace(h.root[1], dis[1] + h.root[1]->val);
92  while (!Q.empty()) { // 每次取出来一条路径
93      HeapNode *now = Q.top().id;
94      long long d = Q.top().val;
95      Q.pop();
96      if (now->ls) Q.emplace(now->ls, d - now->val + now->ls->val);
97      if (now->rs) Q.emplace(now->rs, d - now->val + now->rs->val);
98      HeapNode *tmp = h.root[now->to];
99      if (tmp) Q.emplace(tmp, d + tmp->val);
100 }
```

4.6 全局最小割

时间复杂度为 $O(|V|^3)$。

```
1  // == Preparations ==
2  int g[605][605], vis1[605], vis2[605];
3  long long w[605];
4  // == Main ==
5  long long Stoer_Wagner() {
6      long long ans = 0x3f3f3f3f3f3f3f3f;
7      for (int i = 1; i < n; i++) {
8          int s = 0, t = 0;
9          memset(vis2, 0, sizeof(vis2));
10         memset(w, 0, sizeof(w));
11         for (int j = 1; j <= n - i + 1; j++) {
12             int now = 0;
13             for (int k = 1; k <= n; k++)
14                 if (!vis1[k] && !vis2[k] && w[k] >= w[now]) now = k;
15             s = t, t = now;
16             vis2[now] = 1;
17             for (int k = 1; k <= n; k++) w[k] += g[k][now];
18         }
19         ans = min(ans, w[t]);
20         vis1[t] = 1;
21         for (int j = 1; j <= n; j++)
22             if (j != s) g[s][j] += g[t][j], g[j][s] += g[j][t];
23     }
24     return ans;
25 }
```

4.7   支配树

$idom_u$ 为 u 在支配树上的父亲。

最后 id 形成 dfs 序。

```
1  // == Preparations ==
2  #include <vector>
3
4  struct graph {
5      int tot, hd[200005];
6      int nxt[300005], to[300005];
7
8      void add(int u, int v) {
9          nxt[++tot] = hd[u];
10         hd[u] = tot;
11         to[tot] = v;
12         return ;
13     }
14 } g, fg;
```

```
15  int timer, fa[200005], dfn[200005], id[200005];
16  int sdom[200005], idom[200005];
17  struct dsu {
18      int fa[200005], mn[200005];
19
20      dsu() {for (int i = 1; i < 200005; i++) fa[i] = mn[i] = i;}
21      int find(int x) {
22          if (x == fa[x]) return x;
23          int tmp = find(fa[x]);
24          if (dfn[sdom[mn[fa[x]]]] < dfn[sdom[mn[x]]]) mn[x] = mn[fa[x]];
25          return fa[x] = tmp;
26      }
27  } d;
28  vector<int> vec[200005];
29  int siz[200005];
30  // == Main ==
31  void dfs(int now) {
32      id[dfn[now] = ++timer] = now;
33      for (int i = g.hd[now]; i; i = g.nxt[i])
34          if (!dfn[g.to[i]]) fa[g.to[i]] = now, dfs(g.to[i]);
35      return ;
36  }
37  void solve() {
38      dfs(1);
39      for (int i = 1; i <= n; i++) sdom[i] = i;
40      for (int i = timer; i >= 1; i--) {
41          int u = id[i];
42          for (int v : vec[u]) {
43              d.find(v);
44              if (sdom[d.mn[v]] == u) idom[v] = u;
45              else idom[v] = d.mn[v];
46          }
47          if (i == 1) continue;
48          for (int j = fg.hd[u]; j; j = fg.nxt[j]) {
49              if (!dfn[fg.to[j]]) continue;
50              if (dfn[fg.to[j]] < dfn[sdom[u]]) sdom[u] = fg.to[j];
51              else if (dfn[fg.to[j]] > dfn[u]) {
52                  d.find(fg.to[j]);
53                  if (dfn[sdom[d.mn[fg.to[j]]]] < dfn[sdom[u]]) sdom[u] =
                      ↪   sdom[d.mn[fg.to[j]]];
54              }
55          }
56          vec[sdom[u]].push_back(u);
57          d.fa[u] = fa[u];
58      }
59      for (int i = 2; i <= timer; i++)
60          if (idom[id[i]] != sdom[id[i]]) idom[id[i]] = idom[idom[id[i]]];
61      return ;
```

```
62 }
```

4.8 弦图

4.8.1 MCS 最大势算法。

```cpp
1  // == Preparations ==
2  #include <vector>
3
4  int pos[/* ... */], p[/* ... */];
5  vector<int> vec[/* ... */];
6  // == Main ==
7  for (int i = 1; i <= n; i++) pos[i] = vec[0].size(), vec[0].push_back(i);
8  for (int i = 1, j = 0; i <= n; i++, j++) {
9      while (vec[j].empty()) j--;
10     int u = p[i] = vec[j].back();
11     vec[j].pop_back();
12     pos[u] = -1;
13     for (int k = g.hd[u]; k; k = g.nxt[k])
14         if (pos[g.to[k]] != -1) {
15             int v = g.to[k];
16             pos[vec[l[v]].back()] = pos[v];
17             swap(vec[l[v]][pos[v]], vec[l[v]].back());
18             vec[l[v]].pop_back();
19             pos[v] = vec[++l[v]].size();
20             vec[l[v]].push_back(v);
21         }
22 }
23 reverse(p + 1, p + n + 1);
```

4.8.2 弦图判定

跑 MCS，然后判断是否为完美消除序列。

具体地，对于每个 p_i，找到与之相连且在它之后出现的点，按出现顺序记为 $c_1, c_2, \ldots, c_k$，我们只需要判断 c_1 与 c_j 之间是否有边即可。因为这个团中其他边会在 $p_{c_2}, p_{c_3}, \ldots, p_{c_k}$ 中被判断。

4.8.3 求弦图的团数与色数

求团数：

设 $N(x)$ 为完美消除序列中在 x 之后且与 x 相连的点的集合，则弦图的最大团一定可以被表示为 $\{x\} + N(x)$，则 $|\{x\} + N(x)|$ 的最大值就是弦图的团数。

求色数：

考虑按完美消除序列从后往前考虑，贪心染 mex，这样需要的颜色数量等于团数。由于团数小于等于

31

色数，这样取到等号，一定最小。

4.8.4 求弦图的最大独立集和最小团覆盖

最大独立集：

按完美消除序列从前往后贪心。正确性证明：每次考虑最靠前的极大团，选最前面的点不劣于选其他点，且优于不选点。

最小团覆盖：

取最大独立集中的每个点 x 对应的团 $\{x\} + N(x)$，这样需要的团的数量等于最大独立集的大小。由于最大独立集小于等于最小团覆盖，这样取到等号，一定最小。

4.8.5 tricks

区间图是弦图，完美消除序列为按区间右端点从小到大排序。

树上距离不超过 k 的点连边是弦图，完美消除序列为 bfs 序的逆序。

Chapter 5

多项式

5.1 牛顿迭代

用于解决下列问题：

> 已知函数 G 且 $G(F(x)) = 0$，求多项式 F（$\mod x^n$）。

结论：

$$F(x) = F_*(x) - \frac{G(F_*(x))}{G'(F_*(x))} \pmod{x^n}$$

其中 $F_*(x)$ 为做到 $x^{n/2}$ 时的答案。

5.2 FFT

```cpp
// == Preparations ==
struct complex {
    double a, b;

    complex() = default;
    complex(double _a, double _b): a(_a), b(_b) {}
    complex operator+(const complex &x) const {return complex(a + x.a, b + x.b);}
    complex operator-(const complex &x) const {return complex(a - x.a, b - x.b);}
    complex operator*(const complex &x) const {return complex(a * x.a - b * x.b, a *
    ↪  x.b + b * x.a);}
    complex operator/(const complex &x) const {
        double t = b * b + x.b * x.b;
        return complex((a * x.a + b * x.b) / t, (b * x.a - a * x.b) / t);
    }
    complex &operator+=(const complex &x) {return *this = *this + x;}
    complex &operator-=(const complex &x) {return *this = *this - x;}
    complex &operator*=(const complex &x) {return *this = *this * x;}
```

```
17        complex &operator/=(const complex &x) {return *this = *this / x;}
18  };
19  // == Main ==
20  void FFT(vector<complex> &f, int flag) const {
21      int n = f.size();
22      vector<int> swp(n);
23      for (int i = 0; i < n; i++) {
24          swp[i] = swp[i >> 1] >> 1 | ((i & 1) * (n >> 1));
25          if (i < swp[i]) std::swap(f[i], f[swp[i]]);
26      }
27      for (int mid = 1; mid < n; mid <<= 1) {
28          complex w1(cos(pi / mid), flag * sin(pi / mid));
29          for (int i = 0; i < n; i += mid << 1) {
30              complex w(1, 0);
31              for (int j = 0; j < mid; j++, w *= w1) {
32                  complex x = f[i + j], y = w * f[i + mid + j];
33                  f[i + j] = x + y, f[i + mid + j] = x - y;
34              }
35          }
36      }
37      return;
38  }
```

5.3   常用 NTT 模数及其原根

模数	原根	分解
167772161	3	$5 \times 2^{25} + 1$
469762049	3	$7 \times 2^{26} + 1$
998244353	3	$119 \times 2^{23} + 1$
1004535809	3	$479 \times 2^{21} + 1$
2013265921	31	$15 \times 2^{27} + 1$
2281701377	3	$17 \times 2^{27} + 1$

5.4   多项式模板

```
1  // == Preparations ==
2  #include <vector>
3  // == Main ==
4  namespace Poly {
5      const int mod = 998244353, G = 3, invG = 332748118;
6
7      inline int power(int a, int b) {
8          int ans = 1;
9          while (b) {
```

```
10          if (b & 1) ans = (long long)ans * a % mod;
11          a = (long long)a * a % mod;
12          b >>= 1;
13      }
14      return ans % mod;
15  }
16
17  struct poly: vector<int> {
18      poly(initializer_list<int> &&arg): vector<int>(arg) {}
19      template<typename... argT>
20      poly(argT &&...args): vector<int>(forward<argT>(args)...) {}
21
22      poly operator+(const poly &b) const {
23          const poly &a = *this;
24          poly ans(max(a.size(), b.size()));
25          for (int i = 0; i < (int)ans.size(); i++)
26              ans[i] = ((i < (int)a.size() ? a[i] : 0) + (i < (int)b.size() ? b[i]
                  ↪   : 0)) % mod;
27          return ans;
28      }
29      poly operator+=(const poly &b) {return *this = *this + b;}
30      poly operator-(const poly &b) const {
31          const poly &a = *this;
32          poly ans(max(a.size(), b.size()));
33          for (int i = 0; i < (int)ans.size(); i++)
34              ans[i] = ((i < (int)a.size() ? a[i] : 0) - (i < (int)b.size() ? b[i]
                  ↪   : 0) + mod) % mod;
35          return ans;
36      }
37      poly operator-=(const poly &b) {return *this = *this - b;}
38      void NTT(poly &g, int flag) const {
39          int n = g.size();
40          vector<unsigned long long> f(g.begin(), g.end());
41          vector<int> swp(n);
42          for (int i = 0; i < n; i++) {
43              swp[i] = swp[i >> 1] >> 1 | ((i & 1) * (n >> 1));
44              if (i < swp[i]) std::swap(f[i], f[swp[i]]);
45          }
46          for (int mid = 1; mid < n; mid <<= 1) {
47              int w1 = power(flag ? G : invG, (mod - 1) / mid / 2);
48              vector<int> w(mid);
49              w[0] = 1;
50              for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 % mod;
51              for (int i = 0; i < n; i += mid << 1)
52                  for (int j = 0; j < mid; j++) {
53                      int t = (long long)w[j] * f[i + mid + j] % mod;
54                      f[i + mid + j] = f[i + j] - t + mod;
55                      f[i + j] += t;
56                  }
```

```
57              if (mid == 1 << 10)
58                  for (int i = 0; i < n; i++) f[i] %= mod;
59          }
60          int inv = flag ? 1 : power(n, mod - 2);
61          for (int i = 0; i < n; i++) g[i] = f[i] % mod * inv % mod;
62          return;
63      }
64
65      // 下面是基于转置原理的 NTT，相对朴素版本效率更高。
66      void NTT(poly &g, int flag) const {
67          int n = g.size();
68          vector<int> f(g.begin(), g.end());
69          if (flag) {
70              for (int mid = n >> 1; mid >= 1; mid >>= 1) {
71                  int w1 = power(G, (mod - 1) / mid / 2);
72                  vector<int> w(mid);
73                  w[0] = 1;
74                  for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 %
                      ↪ mod;
75                  for (int i = 0; i < n; i += mid << 1)
76                      for (int j = 0; j < mid; j++) {
77                          int t = (long long)(f[i + j] - f[i + mid + j] + mod) *
                              ↪ w[j] % mod;
78                          f[i + j] = f[i + j] + f[i + mid + j] >= mod ?
79                              f[i + j] + f[i + mid + j] - mod : f[i + j] + f[i +
                                  ↪ mid + j];
80                          f[i + mid + j] = t;
81                      }
82              }
83              for (int i = 0; i < n; i++) g[i] = f[i];
84          } else {
85              for (int mid = 1; mid < n; mid <<= 1) {
86                  int w1 = power(invG, (mod - 1) / mid / 2);
87                  vector<int> w(mid);
88                  w[0] = 1;
89                  for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 %
                      ↪ mod;
90                  for (int i = 0; i < n; i += mid << 1)
91                      for (int j = 0; j < mid; j++) {
92                          int t = (long long)w[j] * f[i + mid + j] % mod;
93                          f[i + mid + j] = f[i + j] - t < 0 ? f[i + j] - t + mod :
                              ↪ f[i + j] - t;
94                          f[i + j] = f[i + j] + t >= mod ? f[i + j] + t - mod : f[i
                              ↪ + j] + t;
95                      }
96              }
97              int inv = power(n, mod - 2);
98              for (int i = 0; i < n; i++) g[i] = (long long)f[i] * inv % mod;
99          }
```

```
100          return;
101       }
102
103       poly operator*(poly b) const {
104          poly a(*this);
105          int n = 1, len = (int)(a.size() + b.size()) - 1;
106          while (n < len) n <<= 1;
107          a.resize(n), b.resize(n);
108          NTT(a, 1), NTT(b, 1);
109          poly c(n);
110          for (int i = 0; i < n; i++) c[i] = (long long)a[i] * b[i] % mod;
111          NTT(c, 0);
112          c.resize(len);
113          return c;
114       }
115       poly operator*=(const poly &b) {return *this = *this * b;}
116       poly inv() const {
117          poly f = *this, g;
118          g.push_back(power(f[0], mod - 2));
119          int n = 1;
120          while (n < (int)f.size()) n <<= 1;
121          f.resize(n << 1);
122          for (int len = 2; len <= n; len <<= 1) {
123             poly tmp(len), ff(len << 1);
124             for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
125             for (int i = 0; i < len; i++) ff[i] = f[i];
126             g.resize(len << 1);
127             NTT(g, 1), NTT(ff, 1);
128             for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] %
                   ↪  mod * ff[i] % mod;
129             NTT(g, 0);
130             g.resize(len);
131             for (int i = 0; i < len; i++) g[i] = (tmp[i] - g[i] + mod) % mod;
132          }
133          g.resize(size());
134          return g;
135       }
136       poly sqrt() const { // need F[0] = 1.
137          poly f = *this, g;
138          g.push_back(1);
139          int n = 1;
140          while (n < (int)f.size()) n <<= 1;
141          f.resize(n << 1);
142          for (int len = 2; len <= n; len <<= 1) {
143             poly tmp(len), ff(len << 1);
144             for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
145             for (int i = 0; i < len; i++) ff[i] = f[i];
146             g.resize(len << 1);
147             NTT(g, 1);
```

```
148        for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] %
       ↪   mod;
149        NTT(g, 0);
150        g += ff;
151        g *= tmp.inv();
152        g.resize(len);
153        }
154        g.resize(size());
155        return g;
156    }
157    poly derivative() const {
158        poly f(*this);
159        for (int i = 1; i < (int)f.size(); i++) f[i - 1] = (long long)f[i] * i %
       ↪   mod;
160        f.pop_back();
161        return f;
162    }
163    poly integral() const {
164        poly f(*this);
165        f.push_back(0);
166        for (int i = f.size() - 1; i >= 1; i--) f[i] = (long long)f[i - 1] *
       ↪   power(i, mod - 2) % mod;
167        f[0] = 0;
168        return f;
169    }
170    poly ln() const {
171        poly f((derivative() * inv()).integral());
172        f.resize(size());
173        return f;
174    }
175    poly exp() const { // 需要满足 F[0] = 0
176        poly f(*this), g;
177        g.push_back(1);
178        int n = 1;
179        while (n < (int)size()) n <<= 1;
180        f.resize(n);
181        for (int len = 2; len <= n; len <<= 1) {
182            poly tmp(g);
183            g.resize(len);
184            g = g.ln();
185            for (int i = 0; i < len; i++) g[i] = (f[i] - g[i] + mod) % mod;
186            g[0] = (g[0] + 1) % mod;
187            g *= tmp;
188            g.resize(len);
189        }
190        g.resize(size());
191        return g;
192    }
193  };
```

```
194
195  inline poly power(poly f, int b) { // 需要满足 F[0] = 1
196      f = f.ln();
197      for (int i = 0; i < (int)f.size(); i++) f[i] = (long long)f[i] * b % mod;
198      f = f.exp();
199      return f;
200  }
201  // 不要求 F[0] = 1 的多项式快速幂，但是我忘记怎么用了，记得去回顾一下！
202  poly power(poly f, int b1, int b2 = -1) {
203      if (b2 == -1) b2 = b1;
204      int n = f.size(), p = 0;
205      reverse(f.begin(), f.end());
206      while (!f.empty() && !f.back()) f.pop_back(), p++;
207      if (f.empty() || (long long)p * b1 >= n) return poly(n);
208      int v = f.back();
209      int inv = power(v, mod - 2);
210      for (int &i : f) i = (long long)i * inv % mod;
211      reverse(f.begin(), f.end());
212      f = f.ln();
213      for (int &i : f) i = (long long)i * b1 % mod;
214      f = f.exp();
215      reverse(f.begin(), f.end());
216      for (int i = 1; i <= p * b1; i++) f.push_back(0);
217      reverse(f.begin(), f.end());
218      f.resize(n);
219      v = power(v, b2);
220      for (int &i : f) i = (long long)i * v % mod;
221      return f;
222  }
223 }
```

Chapter 6

杂项

6.1　取模类

```cpp
// == Main ==
struct mint {
    static const int mod = 998244353;
    int v;

    mint() = default;
    mint(int _v): v((_v % mod + mod) % mod) {}
    explicit operator int() const {return v;}
    mint operator+(const mint &x) const {return v + x.v - (v + x.v < mod ? 0 : mod);}
    mint &operator+=(const mint &x) {return *this = *this + x;}
    mint operator-(const mint &x) const {return v - x.v + (v - x.v >= 0 ? 0 : mod);}
    mint &operator-=(const mint &x) {return *this = *this - x;}
    mint operator*(const mint &x) const {return (long long)v * x.v % mod;}
    mint &operator*=(const mint &x) {return *this = *this * x;}
    mint inv() const {
        mint a(*this), ans(1);
        int b(mod - 2);
        while (b) {
            if (b & 1) ans *= a;
            a *= a;
            b >>= 1;
        }
        return ans;
    }
    mint operator/(const mint &x) const {return *this * x.inv();}
    mint &operator/=(const mint &x) {return *this = *this / x;}
    mint operator-() {return mint(-v);}
};
```

6.1.1 Barrett 约减

当模数不固定时可以加速。

用法：在构造函数中传模数，使用方法为 F.reduce(x)，其中 x 是需要取模的数。

```cpp
// == Main ==
struct Barrett {
    unsigned long long b, m;
    Barrett(unsigned long long b = 2): b(b), m((__uint128_t(1) << 64) / b) {}
    unsigned long long reduce(long long x) {
        unsigned long long r = (__uint128_t(x + b) * m) >> 64;
        unsigned long long q = (x + b) - b * r;
        return q >= b ? q - b : q;
    }
} F;
```

6.2 对拍脚本

```bash
#!/usr/bin/bash

declare -i num=0

while [ true ]; do
    ./mkdata > in.txt
    time ./mine < in.txt > out.txt
    ./correct < in.txt > ans.txt
    diff out.txt ans.txt
    if [ $? -ne 0 ]; then
        echo "WA"
        break
    fi
    num=num+1
    echo "Passed $num tests."
done
```

6.3 VS Code 配置

6.3.1 User Tasks

```json
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
```

```
 4        "version": "2.0.0",
 5        "tasks": [
 6            {
 7                "type": "shell",
 8                "label": "My C++ Runner",
 9                "detail": "Build and Run Current C++ Program",
10                "command": [ // 三个编译方式保留一个即可。
11                    "clear",
12                    "&&",
13                    "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension}
                    ↪  -std=c++14 -Wall -Wextra && echo '== Normal =='",
14                    "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension}
                    ↪  -std=c++14 -Wall -Wextra -O2 && echo '== O2 =='",
15                    "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension}
                    ↪  -std=c++14 -Wall -Wextra -fsanitize=undefined,address && echo
                    ↪  '== UB Check =='",
16                    "&&",
17                    "gnome-terminal -- bash -c \"ulimit -s 524288; time
                    ↪  ${fileDirname}/${fileBasenameNoExtension}; read -p 'Press ENTER
                    ↪  to continue...'; exit\""
18                ],
19                "problemMatcher": [ // 非必要
20                    "$gcc"
21                ],
22                "group": { // 非必要
23                    "kind": "build",
24                    "isDefault": true
25                },
26                "presentation": { // 非必要
27                    "showReuseMessage": false
28                }
29            }
30        ]
31    }
```

6.3.2 设置

- 字体大小：16。（"editor.fontSize": 16）

- 添加多个光标的方式：ctrl。（"editor.multiCursorModifier": "ctrlCmd"）

- 不适用空格代替 Tab。（"editor.insertSpaces": false）

- 不允许 Enter 进行代码补全。（"editor.acceptSuggestionOnEnter": "off"）

- 标尺：110。（"editor.rulers": [110]）

- 平滑。（"editor.cursorSmoothCaretAnimation": "on"）

42

- 标题栏外观。（"window.titleBarStyle": "custom"）

totally：

```
{
    "editor.fontSize": 16,
    "editor.multiCursorModifier": "ctrlCmd",
    "editor.insertSpaces": false,
    "editor.acceptSuggestionOnEnter": "off",
    "editor.rulers": [110],
    "editor.cursorSmoothCaretAnimation": "on",
    "window.commandCenter": false
}
```

6.3.3 快捷键

- 切换块注释：Ctrl+Shift+A -> Ctrl+Shift+/

- 运行任务：Ctrl+Shift+B -> F11

- 向上移动行：Alt+up -> Ctrl+Shift+up

- 向下移动行：Alt+down -> Ctrl+Shift+down