

Standard Code Library

Kyooma

HZCU

October 3, 2023

Contents

一切的开始	3
快读	3
代码模板	3
数据结构	3
st 表	3
树状数组	4
主席树	4
树链剖分	5
平衡树 Treap	7
数学	10
组合数预处理	10
Exgcd	11
Lucas 定理	11
欧拉筛	11
线性基	12
欧拉降幂	12
矩阵快速幂	13
中国剩余定理	13
整除分块	14
拉格朗日插值	14
FFT	15
NTT	16
高斯消元	17
图论	18
LCA	18
倍增求法	18
欧拉序求法	18
Tarjan	19
求割点割边点双	19
求有向图强连通分量 (scc)	20
判断仙人掌图	22
树哈希	23
二分图	24
最大匹配 (匈牙利)	24
最大权匹配	25
欧拉回路	26
最大流	26
最小费用最大流 (费用流)	27
SPFA(时间复杂度 $n \times e \times f$)	27
dij (边权为正, 时间复杂度 $e \times \log(n) \times f$)	28
竞赛图	34
计算几何	35
二维几何	35
Andrew	39
CHT	40
字符串	41
KMP	41
序列自动机	41
字典树	42
字符串双哈希	42

杂项	43
莫队	43
unordered_map	43
DSU	44
Floyd 判圈	44
三分搜索	45
随机器	45

一切的开始

快读

```
1  #define gc() (is==it?it=(is=in)+fread(in,1,Q,stdin),(is==it?EOF:*is++):*is++)
2  const int Q=(1<<24)+1;
3  char in[Q],*is=in,*it=in,c;
4  void read(long long &n){
5      for(n=0;(c=gc())<'0' || c>'9');
6      for(;c<='9'&& c>='0';c=gc())n=n*10+c-48;
7  }
```

代码模板

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  #define dbg(x...) \
5      do { \
6          cout << #x << " -> "; \
7          err(x); \
8      } while (0)
9
10 void err() {
11     cout << endl;
12 }
13
14 template<class T, class... Ts>
15 void err(T arg, Ts &... args) {
16     cout << arg << ' ';
17     err(args...);
18 }
19
20 typedef long long ll;
21 typedef pair<int, int> pii;
22 const int N = 1e5 + 10, Log = 20, inf = 0x3f3f3f3f;
23
24 void solve() {
25
26 }
27
28 int main() {
29     int T = 1;
30     ios::sync_with_stdio(false);
31     cin >> T;
32     while (T--) solve();
33     return 0;
34 }
```

数据结构

st 表

$st[i][j]$ 表示区间 $[i, i + 2^j - 1]$ 的 gcd

```
1  int st[N][Log + 5], logx[N];
2
3  void init(int n) {
4      logx[0] = -1;
5      for (int i = 1; i <= n; i++) logx[i] = logx[i >> 1] + 1;
6      for (int i = 1; i <= n; i++) st[i][0] = i;
7      for (int j = 1; (1 << j) <= n; j++) {
8          for (int i = 1; i + (1 << j) - 1 <= n; i++) {
9              st[i][j] = __gcd(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
10         }
11     }
12 }
13
14 int query(int l, int r){
```

```

15     int k = logx[r - l + 1];
16     return __gcd(st[l][k], st[r - (1 << k) + 1][k]);
17 }

```

树状数组

```

1  template <typename T>
2  struct Fenwick {
3      const int n;
4      vector<T> a;
5      Fenwick(int n) : n(n), a(n + 1) {}
6      void add(int x, T v) {
7          while(x <= n){
8              a[x] += v;
9              x += x & -x;
10         }
11     }
12     T sum(int x) {
13         T ans = 0;
14         for (int i = x; i; i -= i & -i) {
15             ans += a[i];
16         }
17         return ans;
18     }
19     T rangeSum(int l, int r) {
20         return sum(r) - sum(l - 1);
21     }
22 };

```

主席树

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  typedef long long ll;
5  typedef pair<int, int> pii;
6  const int N = 200010;
7  int root[N], tot = 0, num[N], len, a[N];
8  struct Info {
9      int sum, l, r;
10 } info[N << 5];
11
12 int getid(int x) {
13     return lower_bound(num + 1, num + len + 1, x) - num;
14 }
15
16 void build(int &x, int l, int r) { //创建空树
17     x = ++tot;
18     info[x].sum = 0;
19     if (l == r) return;
20     int mid = (l + r) / 2;
21     build(info[x].l, l, mid);
22     build(info[x].r, mid + 1, r);
23 }
24
25 void update(int pre, int &now, int l, int r, int q) { //更新
26     now = ++tot;
27     info[now] = info[pre];
28     info[now].sum++;
29     if (l == r) return;
30     int mid = (l + r) / 2;
31     if (mid >= q) update(info[pre].l, info[now].l, l, mid, q);
32     else update(info[pre].r, info[now].r, mid + 1, r, q);
33 }
34
35 int query(int pre, int now, int l, int r, int k) { //求第 k 小
36     if (l == r) return l;
37     int delta = info[info[now].l].sum - info[info[pre].l].sum;
38     int mid = (l + r) / 2;
39     if (delta >= k) return query(info[pre].l, info[now].l, l, mid, k);

```

```

40     else return query(info[pre].r, info[now].r, mid + 1, r, k - delta);
41 }
42
43 int query_sum(int pre, int now, int l, int r, int k) { // 求小于等于 k 的个数
44     if (l == r) return info[now].sum - info[pre].sum;
45     int mid = (l + r) >> 1;
46     if (k <= mid) return query_sum(info[pre].l, info[now].l, l, mid, k);
47     else return (info[info[now].l].sum - info[info[pre].l].sum) + query_sum(info[pre].r, info[now].r, mid + 1, r, k);
48 }
49 /*
50  先进行离散化
51  sort(num + 1, num + 1 + n);
52  len = unique(num + 1, num + 1 + n) - num - 1;
53  建空树
54  build(root[0], 1, len);
55  更新
56  update(root[i - 1], root[i], 1, len, getid(a[i]));
57  查询 [l, r]
58  query_sum(root[l - 1], root[r], 1, len, k)
59  query(root[l - 1], root[r], 1, len, k)
60  */

```

树链剖分

```

1  vector<int> e[N];
2
3  int n;
4  int sz[N], f[N], son[N];
5  int top[N], dfn[N], rk[N], tot, ru[N];
6  int a[N];
7
8  void dfs(int u, int fa){
9      sz[u] = 1;
10     f[u] = fa;
11     son[u] = -1;
12     for(int i : e[u]){
13         if(i == fa) continue;
14         dfs(i, u);
15         sz[u] += sz[i];
16         if(son[u] == -1 || sz[i] > sz[son[u]]) son[u] = i;
17     }
18 }
19
20 void dfs1(int u, int t){
21     top[u] = t;
22     dfn[u] = ++tot;
23     rk[tot] = u;
24     if(son[u] == -1){
25         ru[u] = dfn[u];
26         return;
27     }
28     dfs1(son[u], t);
29     for(int i : e[u]){
30         if(i == f[u] || i == son[u]) continue;
31         dfs1(i, i);
32     }
33     ru[u] = tot;
34 }
35
36 template<typename T>
37 struct SegmentTree{
38     T sum[N << 2], lz[N << 2];
39     void apply(int k, int l, int r, T x){
40         sum[k] += (r - l + 1) * x;
41         lz[k] += x;
42     }
43     void pd(int k, int l, int r){ // push down
44         int mid = (l + r) >> 1;
45         apply(k << 1, l, mid, lz[k]);
46         apply(k << 1 | 1, mid + 1, r, lz[k]);
47         lz[k] = 0;

```

```

48     }
49     void pu(int k){// push up
50         sum[k] = sum[k << 1] + sum[k << 1 | 1];
51     }
52     void build(int k, int l, int r){
53         if(l == r){
54             sum[k] = a[rk[l]];
55             lz[k] = 0;
56             return;
57         }
58         int mid = (l + r) >> 1;
59         build(k << 1, l, mid);
60         build(k << 1 | 1, mid + 1, r);
61         pu(k);
62     }
63     void mdf(int k, int l, int r, int ql, int qr, T x){// modify [ql, qr] add x
64         if(l > qr || r < ql) return;
65         if(l >= ql && r <= qr){
66             sum[k] += (r - l + 1) * x;
67             lz[k] += x;
68             return;
69         }
70         pd(k, l, r);
71         int mid = (l + r) >> 1;
72         mdf(k << 1, l, mid, ql, qr, x);
73         mdf(k << 1 | 1, mid + 1, r, ql, qr, x);
74         pu(k);
75     }
76     T query(int k, int l, int r, int ql, int qr){
77         if(l > qr || r < ql) return 0;
78         if(l >= ql && r <= qr){
79             return sum[k];
80         }
81         pd(k, l, r);
82         int mid = (l + r) >> 1;
83         return query(k << 1, l, mid, ql, qr) + query(k << 1 | 1, mid + 1, r, ql, qr);
84     }
85 };
86
87 SegmentTree<ll> seg;
88
89 int qrysum(int u, int v){
90     int fu = top[u], fv = top[v], ret = 0;
91     while(fu != fv){
92         if(dfn[fu] > dfn[fv]){
93             ret += seg.query(1, 1, n, dfn[fu], dfn[u]);
94             u = f[fu];
95         }else{
96             ret += seg.query(1, 1, n, dfn[fv], dfn[v]);
97             v = f[fv];
98         }
99         fu = top[u];
100        fv = top[v];
101    }
102    if(dfn[u] > dfn[v]) swap(u, v);
103    ret += seg.query(1, 1, n, dfn[u], dfn[v]);
104    return ret;
105 }
106
107 void solve() {
108     int m, rt;
109     cin >> n;
110     for(int i = 1; i <= n; i++) cin >> a[i];
111     for(int i = 0, u, v; i < n - 1; i++){
112         cin >> u >> v;
113         e[u].push_back(v);
114         e[v].push_back(u);
115     }
116     dfs(1, 0);
117     dfs1(1, 1);
118     seg.build(1, 1, n);

```

119 }

平衡树 Treap

普通平衡树

```
1 mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());
2
3 int rng(int l, int r) {
4     uniform_int_distribution<int> uni(l, r);
5     return uni(mt);
6 }
7
8 struct Node{
9     Node *lt, *rt; // 左右子结点
10    int val, prio; // 值, 优先级
11    int cnt, sz; // 重复次数, 子树大小
12
13    Node(int _val) : val(_val), cnt(1), sz(1) {
14        lt = rt = nullptr;
15        prio = rng(1, 1e9);
16    }
17
18    void upd(){
19        sz = cnt;
20        if(lt != nullptr) sz += lt->sz;
21        if(rt != nullptr) sz += rt->sz;
22    }
23 };
24
25 struct Treap{
26     int siz(Node *p){
27         if(p == nullptr) return 0;
28         return p->sz;
29     }
30
31     Node *root;
32
33     pair<Node *, Node *> split(Node *cur, int key) { // 根据 val 分裂成 小于等于 key 和 大于 key 的两个 treap
34         if (cur == nullptr) return {nullptr, nullptr};
35         if (cur->val <= key) { // 当前属于第一个 treap
36             auto temp = split(cur->rt, key);
37             cur->rt = temp.first;
38             cur->upd();
39             return {cur, temp.second};
40         } else { // 当前属于第二个 treap
41             auto temp = split(cur->lt, key);
42             cur->lt = temp.second;
43             cur->upd();
44             return {temp.first, cur};
45         }
46     }
47
48     tuple<Node *, Node *, Node *> split_by_rk(Node *cur, int rk) { // 根据 rk 分裂成 小于 rk 和 等于 rk 和 大于 rk 的三个
49     ↪ treap, 其中第二个只有一个结点
50         if (cur == nullptr) return {nullptr, nullptr, nullptr};
51         int ls_siz = siz(cur->lt); // 左子树大小
52         if (rk <= ls_siz) { // 当前属于第三个 treap
53             Node *l, *mid, *r;
54             tie(l, mid, r) = split_by_rk(cur->lt, rk);
55             cur->lt = r;
56             cur->upd();
57             return {l, mid, cur};
58         } else if (rk <= ls_siz + cur->cnt) { // 当前属于第二个 treap
59             Node *lt = cur->lt;
60             Node *rt = cur->rt;
61             cur->lt = cur->rt = nullptr;
62             return {lt, cur, rt};
63         } else { // 当前属于第一个 treap
64             Node *l, *mid, *r;
65             tie(l, mid, r) = split_by_rk(cur->rt, rk - ls_siz - cur->cnt);
```



```

65         cur->rt = l;
66         cur->upd();
67         return {cur, mid, r};
68     }
69 }
70
71 Node *merge(Node *u, Node *v) { // 按照 prio 小根堆合并
72     if (u == nullptr && v == nullptr) return nullptr;
73     if (u != nullptr && v == nullptr) return u;
74     if (v != nullptr && u == nullptr) return v;
75     if (u->prio < v->prio) {
76         u->rt = merge(u->rt, v);
77         u->upd();
78         return u;
79     } else {
80         v->lt = merge(u, v->lt);
81         v->upd();
82         return v;
83     }
84 }
85
86 void insert(int val) { // 插入
87     auto temp = split(root, val);
88     auto l_tr = split(temp.first, val - 1);
89     Node *new_node;
90     if (l_tr.second == nullptr) {
91         new_node = new Node(val);
92     } else {
93         l_tr.second->cnt++;
94         l_tr.second->upd();
95     }
96     Node *l_tr_combined = merge(l_tr.first, l_tr.second == nullptr ? new_node : l_tr.second);
97     root = merge(l_tr_combined, temp.second);
98 }
99
100 void del(int val) { // 删除
101     auto temp = split(root, val);
102     auto l_tr = split(temp.first, val - 1);
103     if (l_tr.second == nullptr) {
104         root = merge(l_tr.first, temp.second);
105         return;
106     }
107     if (l_tr.second->cnt > 1) {
108         l_tr.second->cnt--;
109         l_tr.second->upd();
110         l_tr.first = merge(l_tr.first, l_tr.second);
111     } else {
112         if (temp.first == l_tr.second) {
113             temp.first = nullptr;
114         }
115         delete l_tr.second;
116         l_tr.second = nullptr;
117     }
118     root = merge(l_tr.first, temp.second);
119 }
120
121 int qrank_by_val(Node *cur, int val) { // 查询 val 的 rk
122     auto temp = split(cur, val - 1);
123     int ret = siz(temp.first) + 1;
124     root = merge(temp.first, temp.second);
125     return ret;
126 }
127
128 int qval_by_rank(Node *cur, int rk) { // 查询 rk 的 val 第 rk 大的值
129     Node *l, *mid, *r;
130     tie(l, mid, r) = split_by_rk(cur, rk);
131     int ret = (mid == nullptr ? -114514 : mid->val);
132     root = merge(merge(l, mid), r);
133     return ret;
134 }
135

```

```

136     int qprev(int val) { // 查询第一个比 val 小的值
137         auto temp = split(root, val - 1);
138         int ret = qval_by_rank(temp.first, temp.first->sz);
139         root = merge(temp.first, temp.second);
140         return ret;
141     }
142
143     int qnex(int val) { // 查询第一个比 val 大的值
144         auto temp = split(root, val);
145         int ret = qval_by_rank(temp.second, 1);
146         root = merge(temp.first, temp.second);
147         return ret;
148     }
149 };

```

区间翻转

```

1  mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());
2
3  int rng(int l, int r) {
4      uniform_int_distribution<int> uni(l, r);
5      return uni(mt);
6  }
7
8  struct Node{
9      Node *lt, *rt; // 左右子结点
10     int val, prio; // 值, 优先级
11     int cnt, sz; // 重复次数, 子树大小
12     bool rev; // 是否翻转
13
14     Node(int _val) : val(_val), cnt(1), sz(1) {
15         lt = rt = nullptr;
16         rev = false;
17         prio = rng(1, 1e9);
18     }
19
20     void pu(){
21         sz = cnt;
22         if(lt != nullptr) sz += lt->sz;
23         if(rt != nullptr) sz += rt->sz;
24     }
25
26     void pd(){
27         if(rev){
28             swap(lt, rt);
29             if(lt != nullptr) lt->rev ^= 1;
30             if(rt != nullptr) rt->rev ^= 1;
31             rev = false;
32         }
33     }
34 };
35
36 struct Treap{
37     Node* root;
38     int siz(Node *p){
39         if(p == nullptr) return 0;
40         return p->sz;
41     }
42
43     pair<Node *, Node *> split(Node *cur, int sz){
44         if(cur == nullptr) return {nullptr, nullptr};
45         cur->pd();
46         int lc = siz(cur->lt);
47         if(sz <= lc){
48             auto temp = split(cur->lt, sz);
49             cur->lt = temp.second;
50             cur->pu();
51             return {temp.first, cur};
52         }else{
53             auto temp = split(cur->rt, sz - lc - cur->cnt);
54             cur->rt = temp.first;

```

```

55         cur->pu();
56         return {cur, temp.second};
57     }
58 }
59
60 Node* merge(Node* u, Node* v) { // u 小 v 大
61     if (u == nullptr && v == nullptr) return nullptr;
62     if (u != nullptr && v == nullptr) return u;
63     if (u == nullptr && v != nullptr) return v;
64     u->pd(), v->pd();
65     if (u->prio < v->prio) { // u 为根
66         u->rt = merge(u->rt, v);
67         u->pu();
68         return u;
69     } else {
70         v->lt = merge(u, v->lt);
71         v->pu();
72         return v;
73     }
74 }
75
76 void insert(int val){
77     root = merge(root, new Node(val));
78 }
79
80 void seg_rev(int l, int r) {
81     auto res = split(root, l - 1); // [1, l - 1] [l, n]
82     auto ans = split(res.second, r - l + 1); // [l, r] [r + 1, n]
83     ans.first->rev = true;
84     root = merge(res.first, merge(ans.first, ans.second));
85 }
86
87 void print(Node* cur) {
88     if (cur == nullptr) return;
89     cur->pd();
90     print(cur->lt);
91     printf("%d ", cur->val);
92     print(cur->rt);
93 }
94 };

```

数学

组合数预处理

```

1  ll f[N], inv[N];
2
3  ll qpow(ll a, ll b) {
4      ll res = 1;
5      while (b) {
6          if (b & 1) res = res * a % mod;
7          a = a * a % mod;
8          b /= 2;
9      }
10     return res;
11 }
12
13 ll C(ll n, ll m) {
14     return f[n] * inv[m] % mod * inv[n - m] % mod;
15 }
16
17 void init(int M) {
18     f[0] = 1;
19     for (int i = 1; i <= M; i++) f[i] = f[i - 1] * i % mod;
20     inv[M] = qpow(f[M], mod - 2);
21     for (int i = M - 1; i >= 0; i--) inv[i] = inv[i + 1] * (i + 1) % mod;
22 }

```

Exgcd

求解 $xa + yb = c$

有解需满足 $\gcd(a, b) | c$

设解出的一组特解为 x_0, y_0 则通解为 $x = x_0 + tb, y = y_0 - ta$

```
1 ll exgcd(ll a, ll b, ll &x, ll &y) {
2     if (!b) {
3         x = 1;
4         y = 0;
5         return a;
6     } else {
7         ll g = exgcd(b, a % b, x, y);
8         ll t = x;
9         x = y;
10        y = t - a / b * y;
11        return g;
12    }
13 }
14
15 ll upper(ll m, ll n) { //向上取整
16     if (m <= 0) return m / n;
17     return (m - 1) / n + 1;
18 }
19
20 ll lower(ll m, ll n) { //向下取整
21     if (m >= 0) return m / n;
22     return (m + 1) / n - 1;
23 }
```

Lucas 定理

适用于模数为小质数

$$C_n^m \bmod p = C_{n \bmod p}^{m \bmod p} \times C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} \bmod p$$

```
1 ll C(ll n, ll r, ll p) {
2     if (r > n || r < 0) return 0;
3     return f[n] * inv[r] % p * inv[n - r] % p;
4 }
5
6 ll Lucas(ll n, ll m, ll p) {
7     if (m == 0) return 1;
8     return (C(n % p, m % p, p) * Lucas(n / p, m / p, p)) % p;
9 }
```

欧拉筛

```
1 const int N = 1e4 + 10, M = 10000;
2 vector<int> p;
3 int vis[N];
4
5 void init() {
6     for (int i = 2; i <= M; i++) {
7         if (!vis[i]) {
8             p.push_back(i);
9         }
10        for (int j = 0; j < p.size() && p[j] * i <= M; j++) {
11            vis[p[j] * i] = 1;
12            if (i % p[j] == 0) {
13                break;
14            }
15        }
16    }
17 }
```

求欧拉函数: $\phi(n) = n \prod (1 - \frac{1}{p_i})$

```

1  const int N = 1e4 + 10, M = 10000;
2  vector<int> p;
3  int phi[N], vis[N];
4
5  void rua() { //欧拉筛 以及 求欧拉函数
6      for (int i = 2; i <= M; i++) {
7          if (!vis[i]) {
8              p.push_back(i);
9              phi[i] = i - 1;
10         }
11         for (int j = 0; j < p.size() && p[j] * i <= M; j++) {
12             vis[p[j] * i] = 1;
13             if (i % p[j] == 0) {
14                 phi[i * p[j]] = phi[i] * p[j];
15                 break;
16             } else {
17                 phi[i * p[j]] = phi[i] * phi[p[j]];
18             }
19         }
20     }
21 }

```

线性基

线性基是一个数的集合，并且每个序列都拥有至少一个线性基，取线性基中若干个数异或起来可以得到原序列中的任何一个数。原序列里面的任意一个数都可以由线性基里面的一些数异或得到线性基里面的任意一些数异或起来都不能得到 0 线性基里面的数的个数唯一，并且在保持性质一的前提下，数的个数是最少的

```

1  ll d[Log + 5];
2
3  void add(ll x) { // 线性基插入
4      for (int i = Log; i >= 0; i--) {
5          if ((x >> i) & 1) {
6              if (d[i]) x ^= d[i];
7              else {
8                  d[i] = x; // 插入成功
9                  break;
10             }
11         }
12     }
13 }

```

欧拉降幂

$$a^b \pmod m \equiv a^{b \bmod \phi(m) + \phi(m)} \pmod m [b \geq \phi(m)]$$

以下代码以计算 $a_l^{a_{l+2}^{a_{l+1}^{a_r}}}$ 为例

```

1  unordered_map<ll, ll> mp;
2  ll a[N];
3  ll MOD(ll x, ll mod) { return x < mod ? x : x % mod + mod; }
4  ll qpow(ll a, ll b, ll mod) {
5      ll res = 1;
6      while (b) {
7          if (b & 1) res = MOD(res * a, mod);
8          b /= 2;
9          a = MOD(a * a, mod);
10     }
11     return res;
12 }
13 ll phi(ll x) {
14     if (mp[x]) return mp[x];
15     ll res = x;
16     for (ll i = 2; i * i <= x; i++) {
17         if (x % i == 0) {
18             res -= res / i;
19             while (x % i == 0) x /= i;
20         }
21     }

```

```

22     if (x > 1) {
23         res -= res / x;
24     }
25     return mp[x] = res;
26 }
27 ll solve(int l, int r, ll p) {
28     if (p == 1) return MOD(a[l], p);
29     if (l == r) return MOD(a[l], p);
30
31     return qpow(a[l], solve(l + 1, r, phi(p)), p);
32 }

```

矩阵快速幂

```

1  const int MOD = 1e9 + 7;
2
3  struct mat {
4      int n;
5      vector<vector<int>> a;
6
7      mat(int n): n(n), a(n, vector<int>(n)){}
8
9      mat operator*(const mat& b) const {
10         mat res(n);
11         for (int i = 0; i < n; i++) {
12             for (int j = 0; j < n; j++) {
13                 for (int k = 0; k < n; k++) {
14                     (res.a[i][j] += 1ll * a[i][k] * b.a[k][j] % MOD) %= MOD;
15                 }
16             }
17         }
18         return res;
19     }
20
21     void print(){
22         for(int i = 0; i < n; i++){
23             for(int j = 0; j < n; j++){
24                 cout << a[i][j] << ' ';
25             }
26             cout << '\n';
27         }
28         cout << '\n';
29     }
30 };
31
32 mat qpow(mat a, ll b) {
33     mat res(a.n);
34     for (int i = 0; i < a.n; i++) {
35         res.a[i][i] = 1;
36     }
37     while (b) {
38         if (b & 1) res = res * a;
39         a = a * a, b >>= 1;
40     }
41     return res;
42 }

```

中国剩余定理

$$x = num_i(mod\ r_i)$$

```

1  ll CRT(int n) { //适用于 ri 两两互质
2      ll N = 1, res = 0;
3      for (int i = 1; i <= n; i++) N *= r[i];
4      for (int i = 1; i <= n; i++) {
5          ll m = N / r[i], x, y;
6          exgcd(m, r[i], x, y);
7          res = (res + num[i] * m % N * x % N) % N;
8      }

```

```

9     return (res + N) % N;
10 }

```

通解解法

$$x = a_1 (\text{mod } m_1)$$

$$x = a_2 (\text{mod } m_2)$$

$$x = k_1 \times m_1 + a_1 = k_2 \times m_2 + a_2$$

$$k_1 \times m_1 - k_2 \times m_2 = a_2 - a_1$$

运用 `exgcd` 可求得一组解 (k_1, k_2) 可将上述两方程化为

$$x = k_1 \times m_1 + a_1 (\text{mod } \text{lcm}(m_1, m_2))$$

若有多个方程依次两两合并即可

整除分块

$$\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$$

```

1  ans = 0;
2  for(int l = 1, r; l <= n; l = r + 1) {
3      r = n / (n / l);
4      ans += n / l * (r - l + 1);
5  }

```

拉格朗日插值

设要求的 n 次多项式为 $f(k)$, 已知 $f(x_i)$ ($1 \leq i \leq n+1$)

$$f(k) = \sum_{i=1}^{n+1} f(x_i) \prod_{j \neq i} \frac{k - x_j}{x_i - x_j}$$

设要求的 n 次多项式为 $f(k)$, 已知 $f(i)$ ($1 \leq i \leq n+1$)

$$f(k) = \sum_{i=1}^{n+1} f(i) \times \frac{\prod_{j=1}^{n+1} (x-j)}{(x-i) \times (-1)^{n+1-i} \times (i-1)! \times (n+1-i)!}$$

以下代码求 $\sum_{i=1}^n i^k$

```

1  ll f[N], inv[N];
2
3  ll qpow(ll a, ll b) {
4      ll res = 1;
5      while (b) {
6          if (b & 1) res = res * a % mod;
7          a = a * a % mod;
8          b /= 2;
9      }
10     return res;
11 }
12
13 ll C(ll n, ll m) {
14     return f[n] * inv[m] % mod * inv[n - m] % mod;
15 }
16
17 void init(int M) {
18     f[0] = 1;
19     for (int i = 1; i <= M; i++) f[i] = f[i - 1] * i % mod;
20     inv[M] = qpow(f[M], mod - 2);
21     for (int i = M - 1; i >= 0; i--) inv[i] = inv[i + 1] * (i + 1) % mod;
22 }
23
24 void solve() { // 对 k+1 次多项式插值, 且横坐标连续
25     int n, k;
26     cin >> n >> k;

```

```

27     vector<ll> y(k + 3);
28     for(int i = 1; i <= k + 2; i++){ // 前 k+2 项
29         y[i] = (y[i - 1] + qpow(i, k)) % mod;
30     }
31     if(n <= k + 2){
32         cout << y[n] << '\n';
33         return;
34     }
35     init(2e6);
36     vector<ll> p(k + 3);
37     ll sum = 1;
38     for(int i = 1; i <= k + 2; i++){
39         p[i] = qpow(n - i, mod - 2);
40         sum = sum * (n - i) % mod;
41     }
42     ll ans = 0;
43     for(int i = 1; i <= k + 2; i++){
44         ll tmp = y[i] * sum % mod * p[i] % mod * inv[i - 1] % mod * inv[k + 2 - i] % mod;
45         if((k + 2 - i) & 1) ans -= tmp;
46         else ans += tmp;
47         ans %= mod;
48         if(ans < 0) ans += mod;
49     }
50     cout << ans;
51 }

```

FFT

```

1  typedef vector<int> vi;
2  typedef long long ll;
3  typedef pair<int, int> pii;
4
5  typedef complex<double> C;
6  typedef vector<double> vd;
7
8  void fft(vector<C> &a) {
9      int n = (int) a.size(), L = 31 - __builtin_clz(n);
10     static vector<complex<long double>> R(2, 1);
11     static vector<C> rt(2, 1);
12     for (static int k = 2; k < n; k *= 2) {
13         R.resize(n);
14         rt.resize(n);
15         auto x = polar(1.0L, acos(-1.0L) / k);
16         for (int i = k; i < 2 * k; i++) {
17             rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
18         }
19     }
20     vi rev(n);
21     for (int i = 0; i < n; i++) {
22         rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
23     }
24     for (int i = 0; i < n; i++) {
25         if (i < rev[i]) swap(a[i], a[rev[i]]);
26     }
27     for (int k = 1; k < n; k *= 2) {
28         for (int i = 0; i < n; i += 2 * k) {
29             for (int j = 0; j < k; j++) {
30                 C z = rt[j + k] * a[i + j + k];
31                 a[i + j + k] = a[i + j] - z;
32                 a[i + j] += z;
33             }
34         }
35     }
36 }
37
38 vd conv(const vd &a, const vd &b) {
39     if (a.empty() || b.empty()) return {};
40     vd res((int) a.size() + (int) b.size() - 1);
41     int L = 32 - __builtin_clz((int) res.size()), n = 1 << L;
42     vector<C> in(n), out(n);
43     copy(a.begin(), a.end(), begin(in));

```



```

44     for (int i = 0; i < (int) b.size(); i++) {
45         in[i].imag(b[i]);
46     }
47     fft(in);
48     for (C &x: in) x *= x;
49     for (int i = 0; i < n; i++) {
50         out[i] = in[-i & (n - 1)] - conj(in[i]);
51     }
52     fft(out);
53     for (int i = 0; i < (int) res.size(); i++) {
54         res[i] = imag(out[i]) / (4 * n);
55     }
56     return res;
57 }
58
59 using vll = vector<ll>;
60
61 vll gao(const vi &a, const vi &b) { //a 和 b 的卷积
62     vd aa((int) a.size()), bb((int) b.size());
63     for (int i = 0; i < (int) a.size(); i++) aa[i] = a[i];
64     for (int j = 0; j < (int) b.size(); j++) bb[j] = b[j];
65
66     vd cc = conv(aa, bb);
67     vll c((int) cc.size());
68     for (int i = 0; i < (int) c.size(); i++) c[i] = round(cc[i]);
69     return c;
70 }

```

NTT

```

1  typedef long long ll;
2  typedef __int128 i128;
3  const int N = 1e6 + 5;
4  const ll mod = 4179340454199820289, G = 3, Gi = 1393113484733273430;
5
6  ll qpow(ll a, ll b, ll p) {
7      ll res = 1;
8      while (b) {
9          if (b & 1) res = (i128) res * a % p;
10         b >>= 1;
11         a = (i128) a * a % p;
12     }
13     return res;
14 }
15
16 ll f[N], g[N];
17 int bit, tot, rev[N];
18
19 void NTT(ll a[], int type) {
20     for (int i = 0; i < tot; i++)
21         if (i > rev[i])
22             swap(a[i], a[rev[i]]);
23     for (int mid = 1; mid < tot; mid <= 1) {
24         ll w1 = qpow(type == 1 ? G : Gi, (mod - 1) / (mid * 2), mod);
25         for (int i = 0; i < tot; i += mid * 2) {
26             ll wk = 1;
27             for (int j = 0; j < mid; j++, wk = (i128) wk * w1 % mod) {
28                 ll x = a[i + j], y = (i128) wk * a[i + j + mid] % mod;
29                 a[i + j] = (x + y) % mod, a[i + j + mid] = (x - y + mod) % mod;
30             }
31         }
32     }
33     if (type == -1) {
34         ll inv = qpow(tot, mod - 2, mod);
35         for (int i = 0; i < tot; i++)
36             a[i] = (i128) a[i] * inv % mod;
37     }
38 }
39
40 void gao(){
41     int n = 0, m = 0; // f, g 长度

```

```

42     for(int i = 0; i < n; i++) f[i] = 0;
43     for (int i = 0; i < m; i++) g[i] = 0;
44     while ((1 << bit) <= n + m) bit++;
45     tot = 1 << bit;
46     for (int i = 0; i < tot; i++)
47         rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
48     for (int i = n; i < tot; i++) f[i] = 0;
49     for (int i = m; i < tot; i++) g[i] = 0;
50     NTT(f, 1), NTT(g, 1);
51     for (int i = 0; i < tot; i++) f[i] = (i128) f[i] * g[i] % mod;
52     NTT(f, -1);
53 }

```

高斯消元

```

1  const double eps = 1e-8;
2
3  int sgn(double x) {
4      if (fabs(x) < eps) return 0;
5      if (x < 0) return -1;
6      return 1;
7  }
8
9  double a[4][4], x[4], b[4][4], y[4];
10 int equ, var;
11
12 int Gauss() {
13     int i, j, k, col, max_r;
14     for (k = 0, col = 0; k < equ && col < var; ++k, ++col) {
15         max_r = k;
16         for (i = k + 1; i < equ; ++i) if (fabs(a[i][col]) > fabs(a[max_r][col])) max_r = i;
17         if (fabs(a[max_r][col]) < eps) return 0;
18         if (k != max_r) {
19             for (j = col; j < var; ++j) swap(a[k][j], a[max_r][j]);
20             swap(x[k], x[max_r]);
21         }
22         x[k] /= a[k][col];
23         for (j = col + 1; j < var; ++j) a[k][j] /= a[k][col];
24         a[k][col] = 1;
25         for (i = 0; i < equ; ++i) {
26             if (i != k) {
27                 x[i] -= x[k] * a[i][col];
28                 for (j = col + 1; j < var; ++j) a[i][j] -= a[k][j] * a[i][col];
29                 a[i][col] = 0;
30             }
31         }
32     }
33     return 1;
34 }
35
36 int Gauss(int n, int m){ // equ var
37     equ = n;
38     var = m;
39     for(int i = 0; i < n; i++){
40         for(int j = 0; j < m; j++){
41             a[i][j] = b[i][j];
42         }
43         x[i] = y[i];
44     }
45     if(!Gauss()) return 0;
46     for(int i = 0; i < n; i++){
47         double res = 0;
48         for(int j = 0; j < m; j++){
49             res += x[j] * b[i][j];
50         }
51         if(sgn(res - y[i])) return 0;
52     }
53     return 1;
54 }

```

图论

LCA

倍增求法

```
1  const int N = 100010, Log = 20;
2  int anc[N][Log + 5], depth[N];
3  vector<int> e[N];
4
5  void dfs(int u, int fa) {
6      anc[u][0] = fa;
7      depth[u] = depth[fa] + 1;
8      for (int i : e[u]) {
9          if(i == fa) continue;
10         dfs(i, u);
11     }
12 }
13
14 void init(int root, int n) { //初始化
15     depth[0] = 0;
16     dfs(root, 0);
17     for (int j = 1; j <= Log; j++) {
18         for (int i = 1; i <= n; i++) {
19             anc[i][j] = anc[anc[i][j - 1]][j - 1];
20         }
21     }
22 }
23
24 int rush(int u, int h) {
25     for (int i = 0; i <= Log; i++) {
26         if (h >> i & 1) u = anc[u][i];
27     }
28     return u;
29 }
30
31 int qry(int x, int y) { // 求 x 和 y 的 lca
32     if (depth[x] < depth[y]) swap(x, y);
33     x = rush(x, depth[x] - depth[y]);
34     if (x == y) return x;
35     for (int i = Log; i >= 0; i--) {
36         if (anc[x][i] != anc[y][i]) {
37             x = anc[x][i];
38             y = anc[y][i];
39         }
40     }
41     return anc[x][0];
42 }
```

欧拉序求法

```
1  const int N = 100010, Log = 20;
2  int logx[N], st[N][Log]; //logx[i] 即 log(i) 向下取整 st[i][j] 表示 i 为起点长度为 2^j 区间最值
3  int first[N], id[N], tot, depth[N]; //id 为访问时间戳
4  vector<int> e[N];
5
6  void dfs(int u, int fa, int d) {
7      id[++tot] = u; //存储欧拉序所对应的树的节点编号
8      depth[tot] = d; //存储每个 dfs 遍历序列号的深度
9      first[u] = tot; //表示树的第 x 节点在序列第一次出现的时间戳 y
10     for (int v : e[u]) {
11         if(v == fa) continue;
12         dfs(v, u, d + 1);
13         id[++tot] = u;
14         depth[tot] = d;
15     }
16 }
17
18 int Min(int x, int y) {
19     return depth[x] > depth[y] ? y : x;
20 }
```

```

21
22 void init(int root, int n) {
23     dfs(root, 0, 0);
24     n = n * 2 - 1; // 欧拉序长度
25     logx[0] = -1;
26     for (int i = 1; i <= n; i++) logx[i] = logx[i >> 1] + 1;
27     for (int i = 1; i <= n; i++) st[i][0] = i;
28     for (int j = 1; (1 << j) <= n; j++) {
29         for (int i = 1; i + (1 << j) - 1 <= n; i++) {
30             st[i][j] = Min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
31         }
32     }
33 }
34
35 int qry(int u, int v) { //求 u 和 v 节点的 lca
36     int l = first[u], r = first[v];
37     if (l > r) swap(l, r);
38     int k = logx[r - l + 1];
39     return id[Min(st[l][k], st[r - (1 << k) + 1][k])];
40 }

```

Tarjan

求割点割边点双

```

1 // 无向图
2 const int N = 1e3 + 10, M = 1e6 + 10;
3
4 struct Edge{
5     int v, id;
6 };
7
8 vector<Edge> e[N];
9 vector<int> bcc[N]; //点双
10 bool cut[N], cut_edge[M]; // 割点 割边
11 int low[N], dfn[N], tot, bcc_cnt, sta[N], top;
12
13 void tarjan(int u, int fa) {
14     low[u] = dfn[u] = ++tot;
15     sta[++top] = u;
16     int child = 0, x;
17     for (Edge i : e[u]) {
18         int v = i.v, id = i.id;
19         if (!dfn[v]) {
20             child++;
21             tarjan(v, u);
22             low[u] = min(low[v], low[u]);
23             if ((!fa && child > 1) || (fa && low[v] >= dfn[u])) { //割点
24                 cut[u] = true;
25             }
26             if (low[v] > dfn[u]) { //割边
27                 cut_edge[id] = true;
28             }
29             if (low[v] >= dfn[u]) { //点双
30                 bcc_cnt++;
31                 do{
32                     x = sta[top--];
33                     bcc[bcc_cnt].push_back(x);
34                 }while(x != v);
35                 bcc[bcc_cnt].push_back(u);
36             }
37         } else if (v != fa) {
38             low[u] = min(low[u], dfn[v]);
39         }
40     }
41 }
42
43 void solve() {
44     int n, m;
45     cin >> n >> m;
46     for(int i = 0, u, v; i < m; i++){

```

```

47         cin >> u >> v;
48         e[u].push_back({v, i});
49         e[v].push_back({u, i});
50     }
51     for(int i = 1; i <= n; i++){
52         if(!dfn[i]){
53             top = 0;
54             tarjan(i, 0);
55         }
56     }
57 }

```

求有向图强连通分量 (scc)

2-sat 问题:

- $a \vee b : \neg a \longrightarrow b, \neg b \longrightarrow a$
- $\neg(a \wedge b) : a \longrightarrow \neg b, b \longrightarrow \neg a$

```

1 //有 n 对点, 每对点只能选一个, m 对关系, 每对关系给出 u, v 两点, 表示 u 和 v 不能同时选
2 //输出方案或不成立 (NIE)
3 //编号为 2i-1 和 2i 的代表属于第 i 对点
4 #include <bits/stdc++.h>
5
6 using namespace std;
7
8 const int N = 1e5 + 10, M = 1e6 + 10;
9
10 vector<int> e[N];
11 int low[N], dfn[N], tot, sta[N], top;
12 int scc_cnt, scc[N], in[N];
13
14 void tarjan(int u) {
15     low[u] = dfn[u] = ++tot;
16     sta[++top] = u;
17     int x;
18     in[u] = 1;
19     for (int v : e[u]) {
20         if (!dfn[v]) {
21             tarjan(v);
22             low[u] = min(low[v], low[u]);
23         } else if (in[v]) {
24             low[u] = min(low[u], dfn[v]);
25         }
26     }
27     if (dfn[u] == low[u]) { // scc 强连通分量
28         scc_cnt++;
29         do {
30             x = sta[top--];
31             in[x] = 0;
32             scc[x] = scc_cnt; // 染色
33         } while (x != u);
34     }
35 }
36
37 int re(int x){
38     return ((x & 1) ? (x + 1) : (x - 1));
39 }
40
41 void solve() {
42     int n, m;
43     cin >> n >> m;
44     for(int i = 0, u, v; i < m; i++){
45         cin >> u >> v;
46         e[u].push_back(re(v));
47         e[v].push_back(re(u));
48     }
49     for(int i = 1; i <= n * 2; i++){
50         if(!dfn[i]){
51             top = 0;
52             tarjan(i);

```

```

53     }
54 }
55 for(int i = 1; i <= n * 2; i += 2){
56     if(scc[i] == scc[i + 1]){
57         cout << "NIE\n";
58         return;
59     }
60 }
61 for(int i = 1; i <= n * 2; i += 2){
62     int f1 = scc[i], f2 = scc[i + 1];
63     if(f1 < f2){
64         cout << i << '\n';
65     }else{
66         cout << i + 1 << '\n';
67     }
68 }
69 }
70
71 int main() {
72     int T = 1;
73     ios::sync_with_stdio(false);
74     //cin >> T;
75     while (T--) solve();
76     return 0;
77 }

```

1 /*
2 落谷 P6378
3 n 个点 m 条边的无向图被分成 k 个部分 (点集)。每个部分包含一些点。
4 请选择一些关键点, 使得每个部分恰有一个关键点, 且每条边至少有一个端点是关键点。
5
6 做法:
7 每条边就是 $(u \text{ or } v)$ 的关系。
8 对于每个点集来说使用前缀和优化可以变为线性复杂度。
9 设当前点为 u , 前一个点为 v , $preu$ 和 $prev$ 为对应点前缀和点, re 表示不选。
10 $u \rightarrow preu, re(preu) \rightarrow re(u)$
11 $prev \rightarrow preu, re(preu) \rightarrow re(prev)$
12 $prev \rightarrow re(u), u \rightarrow re(prev)$
13 */
14 #include <bits/stdc++.h>
15
16 using namespace std;
17 const int N = 4e6 + 10, Log = 20, inf = 0x3f3f3f3f;
18
19 int n, m, k;
20
21 vector<int> e[N];
22 int low[N], dfn[N], tot, sta[N], top;
23 int scc_cnt, scc[N], in[N];
24
25 void tarjan(int u) {
26 low[u] = dfn[u] = ++tot;
27 sta[++top] = u;
28 int x;
29 in[u] = 1;
30 for (int v : e[u]) {
31 if (!dfn[v]) {
32 tarjan(v);
33 low[u] = min(low[v], low[u]);
34 } else if (in[v]) {
35 low[u] = min(low[u], dfn[v]);
36 }
37 }
38 if (dfn[u] == low[u]) { // scc 强连通分量
39 scc_cnt++;
40 do {
41 x = sta[top--];
42 in[x] = 0;
43 scc[x] = scc_cnt; // 染色
44 } while (x != u);
45 }
46 }

```

47
48 int re(int x){
49     return x > n ? x - n : x + n;
50 }
51
52 void solve() {
53     scanf("%d%d%d", &n, &m, &k);
54     n *= 2;
55     for(int i = 0, u, v; i < m; i++){
56         scanf("%d%d", &u, &v);
57         e[re(u)].push_back(v);
58         e[re(v)].push_back(u);
59     }
60     for(int i = 0, cnt; i < k; i++){
61         scanf("%d", &cnt);
62         int pre = -1;
63         for(int j = 0, u; j < cnt; j++){
64             scanf("%d", &u);
65             int prefix = u + n / 2;
66             e[u].push_back(prefix);
67             e[re(prefix)].push_back(re(u));
68             if(pre != -1){
69                 e[pre].push_back(prefix);
70                 e[re(prefix)].push_back(re(pre));
71                 e[pre].push_back(re(u));
72                 e[u].push_back(re(pre));
73             }
74             pre = prefix;
75         }
76     }
77     for(int i = 1; i <= n * 2; i++){
78         if(!dfn[i]) tarjan(i);
79     }
80     for(int i = 1; i <= n; i++){
81         if(scc[i] == scc[i + n]){
82             puts("NIE");
83             return;
84         }
85     }
86     puts("TAK");
87 }
88
89 int main() {
90     int T = 1;
91     // ios::sync_with_stdio(false);
92     // cin >> T;
93     while (T--) solve();
94     return 0;
95 }

```

判断仙人掌图

```

1 // 无向图判断仙人掌
2 vector<int> e[N];
3 int dfn[N], dep[N], fa[N], low[N], tot;
4 int du[N], is_cactus = true;
5
6 void DP(int rt, int v){
7     int num = dep[v] - dep[rt] + 1; // 环大小
8     for(int i = v; i != rt; i = fa[i]){
9         du[i]++;
10        if(du[i] > 1){
11            is_cactus = false;
12            return;
13        }
14    }
15 }
16
17 void tarjan(int x){
18     dfn[x] = low[x] = ++tot;
19     cc.push_back(x);

```

```

20     for(int v : e[x]){
21         if(v == fa[x]) continue;
22         if(!dfn[v]){
23             fa[v] = x;
24             dep[v] = dep[x] + 1;
25             tarjan(v);
26             low[x] = min(low[x], low[v]);
27         }else{
28             low[x] = min(low[x], dfn[v]);
29         }
30     }
31     if(!is_cactus) return;
32     for(int v : e[x]){
33         if(fa[v] != x && dfn[x] < dfn[v]) DP(x, v);
34     }
35 }

```

树哈希

p_i 表示第 i 个质数。

- $f_u = 1 + \sum_{v \in son_u} f_v \times p_{sz_v}$
- $f_u = \prod_{v \in son_u} f_v + p_{sz_u}$

对于无根树，找重心 (对于一棵树 n 个节点的无根树，找到一个点，将无根树变为以该点为根的有根树时，最大子树的结点数最小，这个点就是重心)，一颗树的重心最多只有两个，分别比较即可。

以下为另一种树哈希。

```

1 //copy from https://uoj.ac/submission/579874
2 //树哈希
3 #include <cstdio>
4 #include <cctype>
5 #include <chrono>
6 #include <random>
7 #include <algorithm>
8
9 typedef unsigned uint;
10 typedef long long unsigned uint64;
11
12 uint64 xorshift(uint64 x) {
13     x ^= x << 13;
14     x ^= x >> 7;
15     x ^= x << 17;
16     return x;
17 }
18
19 const int Max_N = 1000005;
20
21 std::mt19937_64 engine((std::random_device())() ^ std::chrono::steady_clock::now().time_since_epoch().count() ^
22     ↪ size_t(new char));
23 uint64 S = std::uniform_int_distribution<uint64>(1, -1)(engine);
24
25 int N;
26 int first[Max_N];
27 int Next[Max_N * 2], To[Max_N * 2];
28 int ecnt;
29
30 int hfirst[1 << 20];
31 int hNext[Max_N];
32 uint64 hTo[Max_N];
33 int hcnt;
34
35 void insert(uint64 x) {
36     int b = x & ((1 << 20) - 1);
37     for (int e = hfirst[b]; e; e = hNext[e])
38         if (x == hTo[e])
39             return;
40     ++hcnt, hNext[hcnt] = hfirst[b], hTo[hcnt] = x, hfirst[b] = hcnt;

```



```

40 }
41
42 uint64 dfs(const int v, const int p) {
43     uint64 h = S;
44     for (int e = first[v]; e; e = Next[e])
45         if (To[e] != p)
46             h += xorshift(dfs(To[e], v));
47     insert(h);
48     return h;
49 }
50
51 int main(int argc, char **argv) {
52     cin >> N;
53     ecnt = 0;
54     for (int i = 1; i != N; ++i) {
55         int v, w;
56         cin >> v >> w;
57         ++ecnt, Next[ecnt] = first[v], To[ecnt] = w, first[v] = ecnt;
58         ++ecnt, Next[ecnt] = first[w], To[ecnt] = v, first[w] = ecnt;
59     }
60
61     dfs(1, 0);
62
63     printf("%d\n", hcnt);
64
65     return 0;
66 }

```

二分图

最大匹配 (匈牙利)

- k-正则图：各顶点的度均为 k 的无向简单图
- 最大匹配数：最大匹配的匹配边的数目
- 最大独立集数：选取最多的点集，使点集中任意两点均不相连
- 最小点覆盖数：选取最少的点集，使任意一条边都至少有一个端点在点集中
- 最大匹配数 = 最小点覆盖数
- 最大独立集数 = 顶点数 - 最大匹配数

```

1  int n, m;
2  int mp[N][N], link[N]; // 存图 link i 右部图 i 点在左部图的连接点
3  bool vis[N]; // 是否在交替路中
4
5  bool dfs(int u) {
6      for (int v = 1; v <= m; v++) {
7          if (vis[v] || !mp[u][v]) continue;
8          vis[v] = true;
9          if (link[v] == -1 || dfs(link[v])) {
10             link[v] = u;
11             return true;
12         }
13     }
14     return false;
15 }
16
17 int hungarian() {
18     int ans = 0;
19     for (int i = 1; i <= m; i++) link[i] = -1;
20     for (int i = 1; i <= n; i++) {
21         for (int j = 1; j <= m; j++) vis[j] = false;
22         if (dfs(i)) ans++;
23     }
24     return ans;
25 }
26
27 void solve() {
28     int e;
29     cin >> n >> m >> e;
30     for (int i = 0, u, v; i < e; i++) {

```

```

31         cin >> u >> v;
32         mp[u][v] = true;
33     }
34     cout << hungarian();
35 }

```

也可建立一个源点和汇点, 将源点连向所有左部点, 左部点连向右部点, 右部点连向汇点, 且所有流量为 1, 然后跑最大流即为最大匹配

最大权匹配

KM (时间复杂度 n^3)

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  typedef long long ll;
6  typedef pair<int, int> pii;
7  //Data
8  const int N = 500 + 10;
9  const ll inf = 1e11;
10 int nx, ny;
11
12 //KM
13 ll c[N], e[N][N], kb[N], ka[N];
14 int mb[N], p[N], vb[N];
15
16 void Bfs(int u) {
17     int a, v, vl = 0;
18     ll d;
19     for (int i = 1; i <= nx; i++) p[i] = 0, c[i] = inf;
20     mb[v] = u;
21     do {
22         a = mb[v], d = inf, vb[v] = 1;
23         for (int b = 1; b <= nx; b++)
24             if (!vb[b]) {
25                 if (c[b] > ka[a] + kb[b] - e[a][b])
26                     c[b] = ka[a] + kb[b] - e[a][b], p[b] = v;
27                 if (c[b] < d) d = c[b], vl = b;
28             }
29         for (int b = 0; b <= nx; b++)
30             if (vb[b]) ka[mb[b]] -= d, kb[b] += d;
31             else c[b] -= d;
32         v = vl;
33     } while (mb[v]);
34     while (v) mb[v] = mb[p[v]], v = p[v];
35 }
36
37 ll KM() {
38     for (int i = 1; i <= nx; i++) mb[i] = 0, ka[i] = kb[i] = 0;
39     for (int a = 1; a <= nx; a++) {
40         for (int b = 1; b <= nx; b++) vb[b] = 0;
41         Bfs(a);
42     }
43     ll res = 0;
44     for (int b = 1; b <= nx; b++) res += e[mb[b]][b];
45     return res;
46 }
47
48 void solve() {
49     int n, m;
50     scanf("%d%d", &n, &m);
51     nx = n, ny = n;
52     for (int a = 1; a <= nx; a++)
53         for (int b = 1; b <= nx; b++) e[a][b] = -inf;
54     for (int i = 1, u, v, w; i <= m; i++) {
55         scanf("%d%d%d", &u, &v, &w);
56         e[u][v] = max(e[u][v], w * 1ll);
57     }
58     printf("%lld\n", KM());
59     for (int u = 1; u <= ny; u++) printf("%d ", mb[u]);

```

```

60     puts("");
61 }
62
63 int main() {
64     solve();
65     return 0;
66 }

```

费用流 (时间复杂度 $n \times e \times f$ 或 $e \times \log(n) \times f$)

欧拉回路

```

1 // 若有奇数度数的点 可先建若干条虚边使其度数变为偶数
2 const int N = 5e5 + 10;
3
4 struct Edge{
5     int to, next;
6     int index; // 边在图中编号
7     int dir; // 方向
8     bool flag;
9 }edge[N];
10 int head[N], tot;
11
12 void init(){
13     memset(head, -1, sizeof(head));
14     tot = 0;
15 }
16
17 void add(int u, int v, int index){
18     edge[tot] = {v, head[u], index, 0, false};
19     head[u] = tot++;
20     edge[tot] = {u, head[v], index, 1, false};
21     head[v] = tot++;
22 }
23
24 int du[N]; // 点的度
25 vector<int> ans;
26
27 void dfs(int u){
28     for(int i = head[u]; i != -1; i = edge[i].next){
29         if(!edge[i].flag){
30             edge[i].flag = true;
31             edge[i ^ 1].flag = true;
32             dfs(edge[i].to);
33             ans.push_back(i);
34         }
35     }
36 }

```

最大流

```

1 template<int V>
2 struct MF {
3     using U = int; //流量类型
4     const U INF = 0x3f3f3f3f;
5     struct Edge {
6         int v, inv; //有向边指向 v 点, 反向边在 v 邻接表中的位置 inv
7         U w; //流量
8     };
9     int s, t, n; //源点、汇点、总数
10    vector<Edge> e[V];
11    int cur[V], qe[V];
12    U dis[V];
13    bool vis[V];
14
15    void add(int u, int v, U w, U rw = 0) {
16        e[u].push_back({v, (int) e[v].size(), w});
17        e[v].push_back({u, (int) e[u].size() - 1, rw});
18    }

```

```

19
20 bool bfs() {
21     for (int i = 1; i <= n; i++) vis[i] = false;
22     int lt = 0, rt = 0;
23     qe[rt++] = s;
24     vis[s] = true;
25     dis[s] = 0;
26     while (lt < rt) {
27         int u = qe[lt++];
28         for(auto [v, inv, w] : e[u]){
29             if(!vis[v] && w){
30                 vis[v] = true;
31                 dis[v] = dis[u] + 1;
32                 qe[rt++] = v;
33             }
34         }
35     }
36     return vis[t];
37 }
38
39 U dfs(int x, U flow) {
40     if (x == t || !flow) return flow;
41     U delta = 0, f;
42     for (int i = cur[x]; i < e[x].size(); i++) {
43         auto [v, inv, w] = e[x][i];
44         cur[x] = i;
45         if (dis[v] == dis[x] + 1 && (f = dfs(v, min(flow, w))) > 0) {
46             e[x][i].w -= f;
47             e[v][inv].w += f;
48             flow -= f;
49             delta += f;
50             if (flow == 0) break;
51         }
52     }
53     return delta;
54 }
55
56 U MaxFlow() {
57     U ans = 0;
58     while (bfs()) {
59         for (int i = 1; i <= n; i++) cur[i] = 0;
60         ans += dfs(s, inf);
61     }
62     return ans;
63 }
64
65 void init(){
66     for(int i = 1; i <= n; i++){
67         vector<Edge>().swap(e[i]);
68     }
69 }
70 };

```

最小费用最大流 (费用流)

SPFA(时间复杂度 $n \times e \times f$)

```

1  const int inf = 0x3f3f3f3f, N = 100010;
2  struct edge {
3      int to, next;
4      ll w, fee; //w 为流量 fee 为费用
5  } e[N];
6  int head[N], idx;
7  int pre[N], id[N]; //pre 前一个节点 id 当前节点的边的 idx
8  int s, t, n;
9  ll dist[N], flow[N]; //dist 费用 (距离) flow 流量
10 bool vis[N];
11
12 void init() {
13     idx = 0;
14     for (int i = 1; i <= n; i++) head[i] = -1;

```

```

15 }
16
17 void add(int a, int b, ll c, ll fee) {
18     e[idx].to = b;
19     e[idx].w = c;
20     e[idx].next = head[a];
21     e[idx].fee = fee;
22     head[a] = idx++;
23 }
24
25 bool spfa() {
26     for (int i = 1; i <= n; i++) {
27         vis[i] = false;
28         dist[i] = inf;
29         flow[i] = inf;
30     }
31     queue<int> q;
32     q.push(s);
33     vis[s] = true;
34     pre[t] = -1;
35     dist[s] = 0;
36     while (!q.empty()) {
37         int x = q.front();
38         q.pop();
39         vis[x] = false;
40         for (int i = head[x]; i != -1; i = e[i].next) {
41             int to = e[i].to;
42             ll w = e[i].w, fee = e[i].fee;
43             if (w && dist[to] > dist[x] + fee) {
44                 dist[to] = dist[x] + fee;
45                 flow[to] = min(flow[x], w);
46                 pre[to] = x;
47                 id[to] = i;
48                 if (!vis[to]) {
49                     q.push(to);
50                     vis[to] = true;
51                 }
52             }
53         }
54     }
55     return dist[t] != inf;
56 }
57
58 void MinFee() {
59     ll minfee = 0, maxflow = 0;
60     while (spfa()) {
61         int now = t;
62         maxflow += flow[t];
63         minfee += flow[t] * dist[t];
64         while (now != s) {
65             e[id[now]].w -= flow[t];
66             e[id[now] ^ 1].w += flow[t];
67             now = pre[now];
68         }
69     }
70     printf("%lld %lld\n", maxflow, minfee);
71 }

```

dij (边权为正, 时间复杂度 $e \times \log(n) \times f$)

```

1 //适用于正边权
2 template<int V>
3 struct MCMF {
4     using U = int; //流量类型
5     using T = int; //费用类型
6     using P = pair<T, int>;
7     const U INF = 0x3f3f3f3f;
8     const T FINF = 0x3f3f3f3f;
9     struct Edge {
10         int v, inv; //有向边指向 v 点, 反向边在 v 邻接表中的位置 inv
11         U w; //流量

```

```

12     T fee; //费用
13 };
14 int s, t, n; //源点、汇点、总数
15 vector<Edge> e[V];
16 int pre[V], id[V]; //边: pre[v] -> v, id[v] 表示边在 pre[v] 中的下标
17 U flow[V];
18 T dis[V], h[V];
19 bool vis[V];
20
21 void add(int u, int v, U w, T fee) {
22     e[u].push_back({v, (int) e[v].size(), w, fee});
23     e[v].push_back({u, (int) e[u].size() - 1, 0, -fee});
24 }
25
26 bool dij() {
27     for (int i = 1; i <= n; i++) {
28         vis[i] = false;
29         dis[i] = FINF;
30         flow[i] = INF;
31     }
32     dis[s] = 0;
33     priority_queue<P, vector<P>, greater<>> q;
34     q.push({dis[s], s});
35     while (!q.empty()) {
36         auto [d, u] = q.top();
37         q.pop();
38         if (vis[u]) continue;
39         vis[u] = true;
40         for (int i = 0; i < e[u].size(); i++) {
41             auto [v, inv, w, fee] = e[u][i];
42             if (w && dis[v] > dis[u] + h[u] - h[v] + fee) {
43                 dis[v] = dis[u] + h[u] - h[v] + fee;
44                 flow[v] = min(flow[u], w);
45                 pre[v] = u;
46                 id[v] = i;
47                 q.push({dis[v], v});
48             }
49         }
50     }
51     return dis[t] < FINF;
52 }
53
54 void init(){
55     for(int i = 1; i <= n; i++){
56         h[i] = FINF;
57         vis[i] = false;
58     }
59     queue<int> q;
60     h[s] = 0, vis[s] = true;
61     q.push(s);
62     while(!q.empty()){
63         int u = q.front();
64         q.pop();
65         vis[u] = false;
66         for(auto [v, inv, w, fee] : e[u]){
67             if(w && h[v] > h[u] + fee){
68                 h[v] = h[u] + fee;
69                 if(!vis[v]){
70                     vis[v] = true;
71                     q.push(v);
72                 }
73             }
74         }
75     }
76 }
77
78 pair<U, T> MinCostMaxFlow() {
79     T minfee = 0;
80     U maxflow = 0;
81     init();
82     while (dij()) {

```

```

83         for (int i = 1; i <= n; i++) {
84             h[i] += dis[i];
85         }
86         int v = t;
87         maxflow += flow[t];
88         minfee += flow[t] * h[t];
89         while (v != s) {
90             int u = pre[v];
91             //u -> v
92             e[u][id[v]].w -= flow[t];
93             int inv = e[u][id[v]].inv;
94             e[v][inv].w += flow[t];
95             v = u;
96         }
97     }
98     return {maxflow, minfee};
99 }
100 };

1 //from reborn
2 //适用于正边权
3 #pragma GCC optimize(2)
4 #pragma GCC optimize(3,"Ofast","inline")
5 #include <bits/stdc++.h>
6 using namespace std;
7 const int N = 30005;
8 typedef long long ll;
9 ll tot = 2;
10 #define int long long
11
12 const int INF = 0x3f3f3f3f3f3f3f3f;
13
14 #define V vector
15 #define pb push_back
16 #define eb emplace_back
17 #define sz(x) (int)size(x)
18
19 // Min cost max flow {{{
20 template<typename flow_t = int, typename cost_t = int>
21 struct MinCostFlow {
22     struct Edge {
23         cost_t c;
24         flow_t f; // DO NOT USE THIS DIRECTLY. SEE getFlow(Edge const& e)
25         int to, rev;
26         Edge(int _to, cost_t _c, flow_t _f, int _rev) : c(_c), f(_f), to(_to), rev(_rev) {}
27     };
28
29     int N, S, T;
30     vector<vector<Edge>> > G;
31     MinCostFlow(int _N, int _S, int _T) : N(_N), S(_S), T(_T), G(_N), eps(0) {}
32
33     void AddEdge(int a, int b, flow_t cap, cost_t cost) {
34         assert(cap >= 0);
35         assert(a >= 0 && a < N && b >= 0 && b < N);
36         if (a == b) { assert(cost >= 0); return; }
37         cost *= N;
38         eps = max(eps, abs(cost));
39         G[a].emplace_back(b, cost, cap, G[b].size());
40         G[b].emplace_back(a, -cost, 0, G[a].size() - 1);
41     }
42
43     flow_t getFlow(Edge const &e) {
44         return G[e.to][e.rev].f;
45     }
46
47     pair<flow_t, cost_t> minCostMaxFlow() {
48         cost_t retCost = 0;
49         for (int i = 0; i < N; ++i) {
50             for (Edge &e : G[i]) {
51                 retCost += e.c*(e.f);
52             }
53         }

```

```

54 //find max-flow
55 flow_t retFlow = max_flow();
56 h.assign(N, 0); ex.assign(N, 0);
57 isq.assign(N, 0); cur.assign(N, 0);
58 queue<int> q;
59 for (; eps; eps >>= scale) {
60     //refine
61     fill(cur.begin(), cur.end(), 0);
62     for (int i = 0; i < N; ++i) {
63         for (auto &e : G[i]) {
64             if (h[i] + e.c - h[e.to] < 0 && e.f) push(e, e.f);
65         }
66     }
67     for (int i = 0; i < N; ++i) {
68         if (ex[i] > 0){
69             q.push(i);
70             isq[i] = 1;
71         }
72     }
73     // make flow feasible
74     while (!q.empty()) {
75         int u = q.front(); q.pop();
76         isq[u]=0;
77         while (ex[u] > 0) {
78             if (cur[u] == G[u].size()) {
79                 relabel(u);
80             }
81             for (unsigned int &i=cur[u], max_i = G[u].size(); i < max_i; ++i) {
82                 Edge &e = G[u][i];
83                 if (h[u] + e.c - h[e.to] < 0) {
84                     push(e, ex[u]);
85                     if (ex[e.to] > 0 && isq[e.to] == 0) {
86                         q.push(e.to);
87                         isq[e.to] = 1;
88                     }
89                     if (ex[u] == 0) break;
90                 }
91             }
92         }
93     }
94     if (eps > 1 && eps>>scale == 0) {
95         eps = 1<<scale;
96     }
97 }
98 for (int i = 0; i < N; ++i) {
99     for (Edge &e : G[i]) {
100         retCost -= e.c*(e.f);
101     }
102 }
103 return make_pair(retFlow, retCost / 2 / N);
104 }
105
106 private:
107 static constexpr cost_t INFCOST = numeric_limits<cost_t>::max()/2;
108 static constexpr int scale = 2;
109
110 cost_t eps;
111 vector<unsigned int> isq, cur;
112 vector<flow_t> ex;
113 vector<cost_t> h;
114 vector<vector<int>> > hs;
115 vector<int> co;
116
117 void add_flow(Edge& e, flow_t f) {
118     Edge &back = G[e.to][e.rev];
119     if (!ex[e.to] && f) {
120         hs[h[e.to]].push_back(e.to);
121     }
122     e.f -= f; ex[e.to] += f;
123     back.f += f; ex[back.to] -= f;
124 }

```



```

125
126 void push(Edge &e, flow_t amt) {
127     if (e.f < amt) amt = e.f;
128     e.f -= amt; ex[e.to] += amt;
129     G[e.to][e.rev].f += amt; ex[G[e.to][e.rev].to] -= amt;
130 }
131
132 void relabel(int vertex){
133     cost_t newHeight = -INFCOST;
134     for (unsigned int i = 0; i < G[vertex].size(); ++i){
135         Edge const&e = G[vertex][i];
136         if(e.f && newHeight < h[e.to] - e.c){
137             newHeight = h[e.to] - e.c;
138             cur[vertex] = i;
139         }
140     }
141     h[vertex] = newHeight - eps;
142 }
143
144 flow_t max_flow() {
145     ex.assign(N, 0);
146     h.assign(N, 0); hs.resize(2*N);
147     co.assign(2*N, 0); cur.assign(N, 0);
148     h[S] = N;
149     ex[T] = 1;
150     co[0] = N-1;
151     for (auto &e : G[S]) {
152         add_flow(e, e.f);
153     }
154     if (hs[0].size()) {
155         for (int hi = 0; hi >= 0; ) {
156             int u = hs[hi].back();
157             hs[hi].pop_back();
158             while (ex[u] > 0) { // discharge u
159                 if (cur[u] == G[u].size()) {
160                     h[u] = 1e9;
161                     for (unsigned int i = 0; i < G[u].size(); ++i) {
162                         auto &e = G[u][i];
163                         if (e.f && h[u] > h[e.to]+1) {
164                             h[u] = h[e.to]+1, cur[u] = i;
165                         }
166                     }
167                     if (++co[h[u]], !--co[hi] && hi < N) {
168                         for (int i = 0; i < N; ++i) {
169                             if (hi < h[i] && h[i] < N) {
170                                 --co[h[i]];
171                                 h[i] = N + 1;
172                             }
173                         }
174                     }
175                     hi = h[u];
176                 } else if (G[u][cur[u]].f && h[u] == h[G[u][cur[u]].to]+1) {
177                     add_flow(G[u][cur[u]], min(ex[u], G[u][cur[u]].f));
178                 } else {
179                     ++cur[u];
180                 }
181             }
182             while (hi >= 0 && hs[hi].empty()) {
183                 --hi;
184             }
185         }
186     }
187     return -ex[S];
188 }
189 };
190
191 //from bbg
192 //可以跑负权
193 #include <bits/stdc++.h>
194
195 using ll = long long;
196 using ull = unsigned long long;

```

```

7
8 struct MCMF {
9     struct Edge {
10         int nxt, to;
11         ll cap, cost;
12     };
13     std::vector<Edge> edges;
14     std::vector<int> head, fa, fe;
15     std::vector<ll> dual, mark, cyc;
16     ll ti, sum;
17
18     MCMF(int n) : head(n, 0), fa(n), fe(n), dual(n), mark(n), cyc(n + 1), ti(0) {
19         edges.push_back({0, 0, 0, 0});
20         edges.push_back({0, 0, 0, 0});
21     }
22
23     int addEdge(int u, int v, ll cap, ll cost) {
24         sum += std::abs(cost);
25         assert(edges.size() % 2 == 0);
26         int e = edges.size();
27         edges.push_back({head[u], v, cap, cost});
28         head[u] = e;
29         edges.push_back({head[v], u, 0, -cost});
30         head[v] = e + 1;
31         return e;
32     }
33
34     void initTree(int x) {
35         mark[x] = 1;
36         for (int i = head[x]; i; i = edges[i].nxt) {
37             int v = edges[i].to;
38             if (!mark[v] and edges[i].cap) {
39                 fa[v] = x, fe[v] = i;
40                 initTree(v);
41             }
42         }
43     }
44
45     int phi(int x) {
46         if (mark[x] == ti)
47             return dual[x];
48         return mark[x] = ti, dual[x] = phi(fa[x]) - edges[fe[x]].cost;
49     }
50
51     void pushFlow(int e, ll &cost) {
52         int pen = edges[e ^ 1].to, lca = edges[e].to;
53         ti++;
54         while (pen)
55             mark[pen] = ti, pen = fa[pen];
56         while (mark[lca] != ti)
57             mark[lca] = ti, lca = fa[lca];
58
59         int e2 = 0;
60         ll f = edges[e].cap;
61         int path = 2, clen = 0;
62         for (int i = edges[e ^ 1].to; i != lca; i = fa[i]) {
63             cyc[++clen] = fe[i];
64             if (edges[fe[i]].cap < f)
65                 f = edges[fe[e2 = i] ^ (path = 0)].cap;
66         }
67         for (int i = edges[e].to; i != lca; i = fa[i]) {
68             cyc[++clen] = fe[i] ^ 1;
69             if (edges[fe[i] ^ 1].cap <= f)
70                 f = edges[fe[e2 = i] ^ (path = 1)].cap;
71         }
72         cyc[++clen] = e;
73
74         for (int i = 1; i <= clen; ++i) {
75             edges[cyc[i]].cap -= f, edges[cyc[i] ^ 1].cap += f;
76             cost += edges[cyc[i]].cost * f;
77         }

```

```

78         if (path == 2)
79             return;
80
81         int laste = e ^ path, last = edges[laste].to, cur = edges[laste ^ 1].to;
82         while (last != e2) {
83             mark[cur]--;
84             laste ^= 1;
85             std::swap(laste, fe[cur]);
86             std::swap(last, fa[cur]);
87             std::swap(last, cur);
88         }
89     }
90
91     std::pair<ll, ll> compute(int s, int t) {
92         ll tot = sum;
93         int ed = addEdge(t, s, 1e18, -tot);
94         ll cost = 0;
95         initTree(0);
96         mark[0] = ti = 2;
97         fa[0] = cost = 0;
98         int ncnt = edges.size() - 1;
99         for (int i = 2, pre = ncnt; i != pre; i = i == ncnt ? 2 : i + 1) {
100             if (edges[i].cap and
101                 edges[i].cost < phi(edges[i ^ 1].to) - phi(edges[i].to))
102                 pushFlow(pre = i, cost);
103         }
104         ll flow = edges[ed ^ 1].cap;
105         cost += tot * flow;
106         return {cost, flow};
107     }
108 };
109
110 void run(int tCase) {
111     int n, m, s, t;
112     std::cin >> n >> m >> s >> t;
113     s--, t--;
114     MCMF mcmf(n);
115     for (int i = 0; i < m; ++i) {
116         int u, v, cap, cost;
117         std::cin >> u >> v >> cap >> cost;
118         u--, v--;
119         mcmf.addEdge(u, v, cap, cost);
120     }
121     auto [cost, flow] = mcmf.compute(s, t);
122     std::cout << flow << ' ' << cost << '\n';
123 }
124
125 int main() {
126     std::ios_base::sync_with_stdio(false);
127     std::cin.tie(nullptr);
128     int T = 1;
129     // std::cin >> T;
130     for (int t = 1; t <= T; ++t) {
131         run(t);
132     }
133     return 0;
134 }

```

竞赛图

一张完全有向图。其中一条由 u 指向 v 的边表示 u 能打败 v 。

经典结论：

1. 缩点之后是一条链，拓扑序唯一。
2. 拓扑序在前的 SCC 的任意节点的入度严格小于拓扑序在后的 SCC 的节点。
3. 若按照入度从小到大排序后，前 i 个点的入度和等于 $\frac{i \times (i-1)}{2}$ 时，说明出现了一个新的 SCC。

```

1 //cf1498E
2 #include <bits/stdc++.h>

```

```

3
4 using namespace std;
5 typedef pair<int, int> pii;
6 const int N = 1e5 + 10, Log = 20, inf = 0x3f3f3f3f;
7
8 void solve() {
9     int n;
10    cin >> n;
11    string s(n, '1');
12    vector<pii> v;
13    for(int i = 1, x; i <= n; i++){
14        cin >> x;
15        v.push_back({x, i});
16    }
17    sort(v.begin(), v.end());
18    int x = 0, y = 0, mx = -1;
19    for(int i = 0, sum = 0, l = 0; i < n; i++){
20        sum += v[i].first;
21        if(sum == (i + 1) * i / 2){
22            if(i > l && v[i].first - v[l].first > mx){
23                mx = v[i].first - v[l].first;
24                x = v[l].second;
25                y = v[i].second;
26            }
27            l = i + 1;
28        }
29    }
30    cout << "! " << x << " " << y << endl;
31 }
32
33 int main() {
34     int T = 1;
35     ios::sync_with_stdio(false);
36     // cin >> T;
37     while (T--) solve();
38     return 0;
39 }

```

计算几何

二维几何

```

1 namespace Geometry {
2 using T = ll;
3 constexpr T eps = 0;
4
5 bool eq(const T &x, const T &y) { return abs(x - y) <= eps; }
6 inline constexpr int type(T x, T y) {
7     if(x == 0 and y == 0) return 0;
8     if(y < 0 or (y == 0 and x > 0)) return -1;
9     return 1;
10 }
11 struct Point {
12     T x, y;
13     constexpr Point(T _x = 0, T _y = 0) : x(_x), y(_y) {}
14     constexpr Point operator+() const noexcept { return *this; }
15     constexpr Point operator-() const noexcept { return Point(-x, -y); }
16     constexpr Point operator+(const Point &p) const { return Point(x + p.x, y + p.y); }
17     constexpr Point operator-(const Point &p) const { return Point(x - p.x, y - p.y); }
18     constexpr Point &operator+=(const Point &p) { return x += p.x, y += p.y, *this; }
19     constexpr Point &operator-=(const Point &p) { return x -= p.x, y -= p.y, *this; }
20     constexpr T operator*(const Point &p) const { return x * p.x + y * p.y; }
21     constexpr Point &operator*=(const T &k) { return x *= k, y *= k, *this; }
22     constexpr Point operator*(const T &k) { return Point(x * k, y * k); }
23     constexpr bool operator==(const Point &r) const noexcept { return r.x == x and r.y == y; }
24     constexpr T cross(const Point &r) const { return x * r.y - y * r.x; }
25
26     constexpr bool operator<(const Point &r) const { return pair(x, y) < pair(r.x, r.y); }
27
28     // 1 : left, 0 : same, -1 : right

```

```

29     constexpr int toleft(const Point &r) const {
30         auto t = cross(r);
31         return t > eps ? 1 : t < -eps ? -1 : 0;
32     }
33
34     constexpr bool arg_cmp(const Point &r) const {
35         int L = type(x, y), R = type(r.x, r.y);
36         if(L != R) return L < R;
37
38         T X = x * r.y, Y = r.x * y;
39         if(X != Y) return X > Y;
40         return x < r.x;
41     }
42 };
43 bool arg_cmp(const Point &l, const Point &r) { return l.arg_cmp(r); }
44 ostream &operator<<(ostream &os, const Point &p) { return os << p.x << " " << p.y; }
45 istream &operator>>(istream &is, Point &p) {
46     is >> p.x >> p.y;
47     return is;
48 }
49
50 struct Line {
51     Point a, b;
52     Line() = default;
53     Line(Point a, Point b) : a(a), b(b) {}
54     // ax + by = c
55     Line(T A, T B, T C) {
56         if(A == 0) {
57             a = Point(0, C / B), b = Point(1, C / B);
58         } else if(B == 0) {
59             a = Point(C / A, 0), b = Point(C / A, 1);
60         } else {
61             a = Point(0, C / B), b = Point(C / A, 0);
62         }
63     }
64     // 1 : left, 0 : same, -1 : right
65     constexpr int toleft(const Point &r) const {
66         auto t = (b - a).cross(r - a);
67         return t > eps ? 1 : t < -eps ? -1 : 0;
68     }
69
70     friend std::ostream &operator<<(std::ostream &os, Line &ls) {
71         return os << "{"
72             << "(" << ls.a.x << ", " << ls.a.y << "), (" << ls.b.x << ", " << ls.b.y << ")}";
73     }
74 };
75 istream &operator>>(istream &is, Line &p) { return is >> p.a >> p.b; }
76
77 struct Segment : Line {
78     Segment() = default;
79     Segment(Point a, Point b) : Line(a, b) {}
80 };
81
82 ostream &operator<<(ostream &os, Segment &p) { return os << p.a << " to " << p.b; }
83 istream &operator>>(istream &is, Segment &p) {
84     is >> p.a >> p.b;
85     return is;
86 }
87
88 struct Circle {
89     Point p;
90     T r;
91     Circle() = default;
92     Circle(Point p, T r) : p(p), r(r) {}
93 };
94
95 using pt = Point;
96 using Points = vector<pt>;
97 using Polygon = Points;
98 T cross(const pt &x, const pt &y) { return x.x * y.y - x.y * y.x; }
99 T dot(const pt &x, const pt &y) { return x.x * y.x + x.y * y.y; }

```

```

100
101 T abs2(const pt &x) { return dot(x, x); }
102 // http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_1_C
103 // 点の回転方向
104 int ccw(const Point &a, Point b, Point c) {
105     b = b - a, c = c - a;
106     if(cross(b, c) > 0) return +1; // "COUNTER_CLOCKWISE"
107     if(cross(b, c) < 0) return -1; // "CLOCKWISE"
108     if(dot(b, c) < 0) return +2; // "ONLINE_BACK"
109     if(abs2(b) < abs2(c)) return -2; // "ONLINE_FRONT"
110     return 0; // "ON_SEGMENT"
111 }
112
113 // http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_A
114 // 平行判定
115 bool parallel(const Line &a, const Line &b) { return (cross(a.b - a.a, b.b - b.a) == 0); }
116
117 // http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_A
118 // 垂直判定
119 bool orthogonal(const Line &a, const Line &b) { return (dot(a.a - a.b, b.a - b.b) == 0); }
120
121 bool intersect(const Line &l, const Point &p) { return abs(ccw(l.a, l.b, p)) != 1; }
122
123 bool intersect(const Line &l, const Line &m) { return !parallel(l, m); }
124
125 bool intersect(const Segment &s, const Point &p) { return ccw(s.a, s.b, p) == 0; }
126
127 bool intersect(const Line &l, const Segment &s) { return cross(l.b - l.a, s.a - l.a) * cross(l.b - l.a, s.b - l.a) <=
    ↪ 0; }
128
129 // http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_B
130 bool intersect(const Segment &s, const Segment &t) { return ccw(s.a, s.b, t.a) * ccw(s.a, s.b, t.b) <= 0 && ccw(t.a,
    ↪ t.b, s.a) * ccw(t.a, t.b, s.b) <= 0; }
131
132 bool intersect(const Polygon &ps, const Polygon &q) {
133     int pl = si(ps), ql = si(q), i = 0, j = 0;
134     while((i < pl || j < ql) && (i < 2 * pl) && (j < 2 * ql)) {
135         auto ps0 = ps[(i + pl - 1) % pl], ps1 = ps[i % pl];
136         auto qs0 = q[(j + ql - 1) % ql], qs1 = q[j % ql];
137         if(intersect(Segment(ps0, ps1), Segment(qs0, qs1))) return true;
138         Point a = ps1 - ps0;
139         Point b = qs1 - qs0;
140         T v = cross(a, b);
141         T va = cross(qs1 - qs0, ps1 - qs0);
142         T vb = cross(ps1 - ps0, qs1 - ps0);
143
144         if(!v && va < 0 && vb < 0) return false;
145         if(!v && !va && !vb) {
146             i += 1;
147         } else if(v > 0) {
148             if(vb > 0)
149                 i += 1;
150             else
151                 j += 1;
152         } else {
153             if(va > 0)
154                 j += 1;
155             else
156                 i += 1;
157         }
158     }
159     return false;
160 }
161
162 T norm(const Point &p) { return p.x * p.x + p.y * p.y; }
163 Point projection(const Segment &l, const Point &p) {
164     T t = dot(p - l.a, l.a - l.b) / norm(l.a - l.b);
165     return l.a + (l.a - l.b) * t;
166 }
167
168 Point crosspoint(const Line &l, const Line &m) {

```

```

169     T A = cross(l.b - l.a, m.b - m.a);
170     T B = cross(l.b - l.a, l.b - m.a);
171     if(A == 0 and B == 0) return m.a;
172     return m.a + (m.b - m.a) * (B / A);
173 }
174
175 // http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_2_C
176 Point crosspoint(const Segment &l, const Segment &m) { return crosspoint(Line(l), Line(m)); }
177
178 // http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=CGL_3_B
179 // 凸性判定
180 bool is_convex(const Points &p) {
181     int n = (int)p.size();
182     for(int i = 0; i < n; i++) {
183         if(ccw(p[(i + n - 1) % n], p[i], p[(i + 1) % n]) == -1) return false;
184     }
185     return true;
186 }
187
188 Points convex_hull(Points p) {
189     int n = p.size(), k = 0;
190     if(n <= 2) return p;
191     sort(begin(p), end(p), [](pt x, pt y) { return (x.x != y.x ? x.x < y.x : x.y < y.y); });
192     Points ch(2 * n);
193     for(int i = 0; i < n; ch[k++] = p[i++]) {
194         while(k >= 2 && cross(ch[k - 1] - ch[k - 2], p[i] - ch[k - 1]) <= 0) --k;
195     }
196     for(int i = n - 2, t = k + 1; i >= 0; ch[k++] = p[i--]) {
197         while(k >= t && cross(ch[k - 1] - ch[k - 2], p[i] - ch[k - 1]) <= 0) --k;
198     }
199     ch.resize(k - 1);
200     return ch;
201 }
202
203 // 面積の 2 倍
204 T area2(const Points &p) {
205     T res = 0;
206     rep(i, si(p)) { res += cross(p[i], p[(i == si(p) - 1 ? 0 : i + 1)]); }
207     return res;
208 }
209
210 enum { _OUT, _ON, _IN };
211
212 int contains(const Polygon &Q, const Point &p) {
213     bool in = false;
214     for(int i = 0; i < Q.size(); i++) {
215         Point a = Q[i] - p, b = Q[(i + 1) % Q.size()] - p;
216         if(a.y > b.y) swap(a, b);
217         if(a.y <= 0 && 0 < b.y && cross(a, b) < 0) in = !in;
218         if(cross(a, b) == 0 && dot(a, b) <= 0) return _ON;
219     }
220     return in ? _IN : _OUT;
221 }
222
223 Polygon Minkowski_sum(const Polygon &P, const Polygon &Q) {
224     vector<Segment> e1(P.size()), e2(Q.size()), ed(P.size() + Q.size());
225     const auto cmp = [](const Segment &u, const Segment &v) { return (u.b - u.a).arg_cmp(v.b - v.a); };
226     rep(i, P.size()) e1[i] = {P[i], P[(i + 1) % P.size()]};
227     rep(i, Q.size()) e2[i] = {Q[i], Q[(i + 1) % Q.size()]};
228     rotate(begin(e1), min_element(all(e1), cmp), end(e1));
229     rotate(begin(e2), min_element(all(e2), cmp), end(e2));
230     merge(all(e1), all(e2), begin(ed), cmp);
231     const auto check = [](const Points &res, const Point &u) {
232         const auto back1 = res.back(), back2 = *prev(end(res), 2);
233         return eq(cross(back1 - back2, u - back2), eps) and dot(back1 - back2, u - back1) >= -eps;
234     };
235     auto u = e1[0].a + e2[0].a;
236     Points res{u};
237     res.reserve(P.size() + Q.size());
238     for(const auto &v : ed) {
239         u = u + v.b - v.a;

```

```

240     while(si(res) >= 2 and check(res, u)) res.pop_back();
241     res.eb(u);
242 }
243 if(res.size() and check(res, res[0])) res.pop_back();
244 return res;
245 }
246
247 // -1 : on, 0 : out, 1 : in
248 // O(log(n))
249 int is_in(const Polygon &p, const Point &a) {
250     if(p.size() == 1) return a == p[0] ? -1 : 0;
251     if(p.size() == 2) return intersect(Segment(p[0], p[1]), a);
252     if(a == p[0]) return -1;
253     if((p[1] - p[0]).toleft(a - p[0]) == -1 || (p.back() - p[0]).toleft(a - p[0]) == 1) return 0;
254     const auto cmp = [&](const Point &u, const Point &v) { return (u - p[0]).toleft(v - p[0]) == 1; };
255     const size_t i = lower_bound(p.begin() + 1, p.end(), a, cmp) - p.begin();
256     if(i == 1) return intersect(Segment(p[0], p[i]), a) ? -1 : 0;
257     if(i == p.size() - 1 && intersect(Segment(p[0], p[i]), a)) return -1;
258     if(intersect(Segment(p[i - 1], p[i]), a)) return -1;
259     return (p[i] - p[i - 1]).toleft(a - p[i - 1]) > 0;
260 }
261
262 Points halplane_intersection(vector<Line> L, const T inf = 1e9) {
263     Point box[4] = {Point(inf, inf), Point(-inf, inf), Point(-inf, -inf), Point(inf, -inf)};
264     rep(i, 4) { L.emplace_back(box[i], box[(i + 1) % 4]); }
265     sort(all(L), [](const Line &l, const Line &r) { return (l.b - l.a).arg_cmp(r.b - r.a); });
266     deque<Line> dq;
267     int len = 0;
268     auto check = [](const Line &a, const Line &b, const Line &c) { return a.toleft(crosspoint(b, c)) == -1; };
269     rep(i, L.size()) {
270         while(dq.size() > 1 and check(L[i], *(end(dq) - 2), *(end(dq) - 1))) dq.pop_back();
271         while(dq.size() > 1 and check(L[i], dq[0], dq[1])) dq.pop_front();
272         // dump(L[i], si(dq));
273
274         if(dq.size() and eq(cross(L[i].b - L[i].a, dq.back().b - dq.back().a), 0)) {
275             if(dot(L[i].b - L[i].a, dq.back().b - dq.back().a) < eps) return {};
276             if(L[i].toleft(dq.back().a) == -1)
277                 dq.pop_back();
278             else
279                 continue;
280         }
281         dq.emplace_back(L[i]);
282     }
283
284     while(dq.size() > 2 and check(dq[0], *(end(dq) - 2), *(end(dq) - 1))) dq.pop_back();
285     while(dq.size() > 2 and check(dq.back(), dq[0], dq[1])) dq.pop_front();
286     if(si(dq) < 3) return {};
287     Polygon ret(dq.size());
288     rep(i, dq.size()) ret[i] = crosspoint(dq[i], dq[(i + 1) % dq.size()]);
289     return ret;
290 }
291 } // namespace Geometry
292
293 using namespace Geometry;

```

Andrew

```

1  const double eps = 1e-9, pi = acos(-1.0);
2  const int N = 1e5 + 10;
3
4  int n, cnt, m;
5
6  int sgn(double x) {
7      if(fabs(x) < eps) return 0;
8      if(x > 0) return 1;
9      return -1;
10 }
11
12 struct point {
13     double x, y;
14     point(double a = 0.0, double b = 0.0) : x(a), y(b) {}

```



```

15     bool operator < (point t) {
16         if(sgn(x - t.x) == 0) return y < t.y;
17         return x < t.x;
18     }
19     point operator - (point p){
20         return {x - p.x, y - p.y};
21     }
22     double operator ^ (point p){
23         return x * p.y - y * p.x;
24     }
25 }p[N], ans[N];
26
27 double dis(point a, point b) {
28     a = a - b;
29     return sqrt(a.x * a.x + a.y * a.y);
30 }
31
32 void Andrew() {
33     sort(p, p + n);
34     int p1 = 0, p2;
35     for(int i = 0; i < n; i++) {
36         while(p1 > 1 && sgn((ans[p1] - ans[p1 - 1]) ^ (p[i] - ans[p1 - 1])) <= 0) p1--;
37         ans[++p1] = p[i];
38     }
39     p2 = p1;
40     for(int i = n - 2; i >= 0; i--) {
41         while(p2 > p1 && sgn((ans[p2] - ans[p2 - 1]) ^ (p[i] - ans[p2 - 1])) <= 0) p2--;
42         ans[++p2] = p[i];
43     }
44     double target = 0.0;
45     for(int i = 1; i < p2; i++){
46         target += dis(ans[i], ans[i + 1]);
47     }
48     printf("%.2f\n", target);
49 }
50 /*
51 usage:
52 scanf("%d", &n);
53 for(int i = 0; i < n; i++)
54     scanf("%lf%lf", &p[i].x, &p[i].y);
55 Andrew();
56 */

```

CHT

```

1 // 维护上凸壳
2 struct Line {
3     ll k, b;
4     double intersect(Line l) {
5         //交点 x 坐标
6         double db = l.b - b;
7         double dk = k - l.k;
8         return db / dk;
9     }
10
11     ll calc (int x) {
12         return k * x + b;
13     }
14 };
15
16 struct CHT {
17     vector<double> x; // 相邻线交点
18     vector<Line> line; // 线
19
20     void init(Line l) {
21         x.push_back(-inf);
22         line.push_back(l);
23     }
24
25     void addLine(Line l) {
26         while (line.size() >= 2 && l.intersect(line[line.size() - 2]) <= x.back()) {

```

```

27         x.pop_back();
28         line.pop_back();
29     }
30     x.push_back(l.intersect(line.back()));
31     line.push_back(l);
32 }
33
34 ll query(int qx) {
35     int id = upper_bound(x.begin(), x.end(), qx) - x.begin() - 1; // 计算点属于的线 id
36     return line[id].calc(qx);
37 }
38 };

```

字符串

KMP

```

1  int nxt[N];
2  string a, b;
3  //a 为模式串 b 为匹配串
4
5  int kmp(int n, int m){
6      int res = 0;
7      nxt[0] = -1;
8      for(int j = -1, i = 0; i < n;){
9          if(j == -1 || a[j] == a[i]){
10             i++;j++;
11             nxt[i] = j;
12          }else{
13             j = nxt[j];
14          }
15      }
16      //i 模式串 j 匹配串
17      for(int i = 0, j = 0; j < m; ){
18          if(i == -1 || a[i] == b[j]){
19             i++;j++;
20          }else i = nxt[i];
21          if(i == n){
22             res += 1;
23             // position:: j - n + 1
24             i = nxt[i];
25          }
26      }
27      return res;
28  }

```

序列自动机

```

1  构建:
2  for(int i = n; i >= 1; i--){
3      for(int j = 0; j < 26; j++) ne[i - 1][j] = ne[i][j];
4      ne[i - 1][s[i - 1] - 'a'] = i;
5  }
6
7  求三 (或多个) 个串的公共子序列个数:
8  int dfs(int p1, int p2, int p3){
9      if(f[p1][p2][p3]) return f[p1][p2][p3];
10     for(int i = 0; i < 26; i++){
11         if(ne[0][p1][i] && ne[1][p2][i] && ne[2][p3][i]){
12             f[p1][p2][p3] = (f[p1][p2][p3] + dfs(ne[0][p1][i], ne[1][p2][i], ne[2][p3][i])) % mod;
13         }
14     }
15     f[p1][p2][p3] = (f[p1][p2][p3] + 1) % mod;
16     return f[p1][p2][p3];
17 }

```

字典树

```
1 //对数排序 查找排序后第 k 个数 每个数 <= 1e9
2 const int N = 5000010; // 总长度
3 int trie[N][10], tot, sum[N][10], ssum = 0;
4 int color[N];
5
6 void insert(int x){
7     int r = 1e9, p = 0;
8     ssum++;
9     for(int i = 0; i < 10; i++, r /= 10){
10         int c = x / r;
11         c %= 10;
12         sum[p][c]++;
13         if(!trie[p][c]) trie[p][c] = ++tot;
14         p = trie[p][c];
15     }
16     color[p]++;
17 }
18
19 int find(int k){
20     int res = 0, p = 0;
21     while(k > 0){
22         for(int i = 0; i < 10; i++){
23             if(sum[p][i] < k) k -= sum[p][i];
24             else{
25                 res = res * 10 + i;
26                 p = trie[p][i];
27                 k -= color[p];
28                 break;
29             }
30         }
31     }
32     return res;
33 }
```

字符串双哈希

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 typedef long long ll;
5 typedef pair<int, int> pii;
6 typedef pair<ll, ll> pll;
7
8 const int N = 1e5 + 10;
9 const pii mod = {1e9 + 7, 1e9 + 9};
10 const pii base = {131, 251};
11 pll pw[N];
12
13 pll operator*(const pll &p1, const pll &p2) {
14     return {p1.first * p2.first % mod.first, p1.second * p2.second % mod.second};
15 }
16
17 pll operator+(const pll &p1, const pll &p2) {
18     return {(p1.first + p2.first) % mod.first, (p1.second + p2.second) % mod.second};
19 }
20
21 pll operator-(const pll &p1, const pll &p2) {
22     return {(p1.first - p2.first + mod.first) % mod.first, (p1.second - p2.second + mod.second) % mod.second};
23 }
24
25 struct Hash {
26     vector<pll> f;
27     int n{};
28
29     void init(ll ss[], int _n) {
30         n = _n;
31         f.resize(n + 1, {0, 0});
32         for (int i = 1; i <= n; i++) {
33             ll ch = ss[i];
```

```

34         f[i] = f[i - 1] * base + pll{ch, ch};
35     }
36 }
37
38 pll ask(int l, int r) {[l, r]
39     return f[r] - f[l - 1] * pw[r - l + 1];
40 }
41 };
42 //记得初始化 pw
43 //pw[0] = {1, 1};
44 //for (int i = 1; i <= n; i++) pw[i] = pw[i - 1] * base;

```

杂项

莫队

时间复杂度 $O(\frac{n^2}{S} + mS)$, n 为长度, m 个询问, 块长为 S (一般取 \sqrt{n} 或 $\frac{n}{\sqrt{m}}$)

```

1  int unit;
2  int a[N];
3
4  struct node {
5      int l, r, id;
6
7      bool operator < (const node &k) const {
8          if (l / unit != k.l / unit) return l / unit < k.l / unit;
9          return r < k.r;
10     }
11 } q[N];
12 void add(int i) {
13
14 }
15
16 void sub(int i) {
17
18 }
19 void solve(){
20     unit = (int)sqrt(m); // m 个区间
21     sort(q + 1, q + 1 + m);
22     int L = 1, R = 0;
23     for (int i = 1; i <= m; i++) {
24         while (R < q[i].r) {
25             R++;
26             add(R);
27         }
28         while (R > q[i].r) {
29             sub(R);
30             R--;
31         }
32         while (L > q[i].l) {
33             L--;
34             add(L);
35         }
36         while (L < q[i].l) {
37             sub(L);
38             L++;
39         }
40     }
41 }

```

unordered_map

```

1  struct HashFunc{
2      static uint64_t splitmix64(uint64_t x) {
3          // http://xorshift.di.unimi.it/splitmix64.c
4          x += 0x9e3779b97f4a7c15;
5          x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
6          x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
7          return x ^ (x >> 31);
8      }
9  };

```

```

8     }
9     template<typename T, typename U>
10    size_t operator()(const std::pair<T, U>& p) const {
11        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
12        return splitmix64(p.first + FIXED_RANDOM) ^ splitmix64(p.second + FIXED_RANDOM);
13    }
14 };
15
16 // 键值比较, 哈希碰撞的比较定义, 需要直到两个自定义对象是否相等
17 struct EqualKey {
18     template<typename T, typename U>
19     bool operator()(const std::pair<T, U>& p1, const std::pair<T, U>& p2) const {
20         return p1.first == p2.first && p1.second == p2.second;
21     }
22 };
23 unordered_map<pii, int, HashFunc, EqualKey> mp;

```

DSU

```

1 struct DSU{
2     int f[N];
3     void init(int n){
4         for(int i = 0; i <= n; i++) f[i] = -1;
5     }
6     int find(int x){
7         return f[x] < 0 ? x : f[x] = find(f[x]);
8     }
9     void merge(int x, int y){
10        int fx = find(x), fy = find(y);
11        if(fx == fy) return;
12        if(f[fx] > f[fy]) swap(fx, fy);
13        f[fx] += f[fy];
14        f[fy] = fx;
15    }
16 }dsu;

```

Floyd 判圈

```

1 //适用于每个点出度唯一的图找环
2 const ll mod = 1099511627776;
3
4 ll calc(ll x){
5     return (x + (x >> 20) + 12345) % mod;
6 }
7
8 void Floyd_Cycle_Detection_Algorithm(){
9     ll p1 = 1611516670, p2 = 1611516670; // 起始点
10    do{
11        p1 = calc(p1); // 移动一次
12        p2 = calc(calc(p2)); // 移动两次
13    }while(p1 != p2);
14    // 存在环
15    ll len = 0; // 环长
16    do{
17        p2 = calc(p2);
18        len++;
19    }while(p1 != p2);
20    p1 = 1611516670; // 寻找环起点
21    ll c1 = 0; // 起点到环起点的距离
22    while(p1 != p2){
23        p1 = calc(p1);
24        p2 = calc(p2);
25        c1++;
26    }
27    cout << p1 << ' ' << len << ' ' << c1 << '\n';
28 }

```

三分搜索

```
1 auto ternary_search = [&](ld l, ld r) {
2     int it = 300;           //set the error limit here
3     while (it--) {
4         ld m1 = l + (r - l) / 3;
5         ld m2 = r - (r - l) / 3;
6         ld f1 = f(m1);      //evaluates the function at m1
7         ld f2 = f(m2);      //evaluates the function at m2
8         if (f1 < f2)
9             l = m1;
10        else
11            r = m2;
12    }
13    return l;                //return the maximum of f(x) in [l, r]
14 };

1 auto ternary_search = [&](int l, int r) {
2     int it = 300;           //set the error limit here
3     while (it--) {
4         int m1 = (l + r) >> 1;
5         int m2 = m1 + 1;
6         ld f1 = f(m1);      //evaluates the function at m1
7         ld f2 = f(m2);      //evaluates the function at m2
8         if (f1 < f2)
9             l = m1;
10        else
11            r = m2;
12    }
13    return l;                //return the maximum of f(x) in [l, r]
14 };
```

随机器

```
1 // random shuffle
2 random_device rd;
3 mt19937 rng(rd());
4 shuffle(a + 1, a + 1 + n, rng);

1 // 区间随机
2 mt19937 mt(chrono::steady_clock::now().time_since_epoch().count());
3
4 int rng(int l, int r) {
5     uniform_int_distribution<int> uni(l, r);
6     return uni(mt);
7 }
```