

Standard Code Library

Your TeamName

Your School

May 12, 2024

Contents

一切的开始	3
数据结构	3
ST 表	3
线段树	3
树状数组	4
DSU	4
Splay	4
LCT	5
扫描线	7
Seg beats	9
珂朵莉树	11
李超树	11
动态维护凸壳	12
图论	13
树链剖分	13
LCA	14
倍增求 LCA	14
dfn 求 LCA	14
树哈希	15
虚树	15
Dijkstra	15
最小环	16
差分约束	16
最大流	16
最小费用最大流	17
二分图最大匹配	18
KM(二分图最大权匹配)	19
一般图最大匹配	20
缩点 SCC	22
割点与桥	23
边双缩点	23
圆方树	24
广义圆方树	24
2-SAT	25
环计数	25
字符串	25
manacher	25
SA	26
PAM	27
SAM	27
ACAM	29
KMP	30
Z 函数	30
LCP	31
Hash	31
计算几何	31
二维计算几何	31
动态凸包	39
最小圆覆盖	40
杂项	41
大质数和原根	41

约瑟夫问题	42
辛普森积分	42
unordered_map	43
位运算	43
int128 输出	44
随机生成质数	44
bitset	44
string	46
pb_ds	47
__gnu_pbds :: tree	47
__gnu_pbds :: priority_queue	49
hash 表	50
rope	50
对拍	51
火车头	51
Sublime	51
卡常	52
注意事项	52
策略	52

一切的开始

数据结构

ST 表

```
1 struct ST{
2     int n;
3     std::vector<array<int,21>> st;
4     ST(int n):n(n),st(n + 1) {}
5     void init(vector<int>& a){
6         for(int i = 1;i <= n;i ++){st[i][0] = a[i - 1];
7             for(int j = 1;j <= 18;j ++){
8                 for(int i = 1;i + (1 << j) <= n + 1;i ++){
9                     st[i][j] = max(st[i][j - 1],st[i + (1 << (j - 1))][j - 1]);
10                }
11            }
12        }
13        int rmq(int l,int r){
14            int j = log(r - l + 1)/log(2);
15            return max(st[l][j],st[r - (1 << j) + 1][j]);
16        }
17    };
```

线段树

```
1 struct SegTree {
2     int l, r;
3     SegTree *ls, *rs;
4     ll sum;
5     ll plus;
6     SegTree (const int L, const int R) : l(L), r(R) {
7         plus = 0;
8         if (L == R) {
9             /*Initial*/
10            ls = rs = nullptr;
11        } else {
12            int M = (L + R) >> 1;
13            ls = new SegTree (L, M);
14            rs = new SegTree (M + 1, R);
15            pushup();
16        }
17    }
18    void pushup() {
19        sum = ls->sum + rs->sum;
20        // std::cerr << "AAA" << l << ' ' << r << ' ' << sum;
21    }
22    void make_tag(long long w) {
23        sum += (r - l + 1) * w;
24        plus += w;
25    }
26    void pushdown() {
27        if (plus == 0) return;
28        ls->make_tag(plus);
29        rs->make_tag(plus);
30        plus = 0;
31    }
32    void upd(const int L, const int R, const int w) {
33        if ((L > r) || (l > R)) return;
34        if ((L <= l) && (r <= R)) {
35            make_tag(w);
36        } else {
37            pushdown();
38            ls->upd(L, R, w);
39            rs->upd(L, R, w);
40            pushup();
41        }
42    }
43 };
```

树状数组

```
1  template <typename T>
2  struct Fenwick {
3      int n;
4      std::vector<T> a;
5      Fenwick(int n) : n(n), a(n) {}
6      void add(int x, T v) {
7          for (int i = x + 1; i <= n; i += i & -i) {
8              a[i - 1] += v;
9          }
10     }
11     T sum(int x) {
12         T ans = 0;
13         for (int i = x; i > 0; i -= i & -i) {
14             ans += a[i - 1];
15         }
16         return ans;
17     }
18     T rangeSum(int l, int r) {
19         return sum(r) - sum(l);
20     }
21     int kth(T k) {
22         int x = 0;
23         // 先从高位开始取, 如果当前这一位可以取, 那么就考虑下一位是取 1 还是 0
24         // 到最后找到的就是最大的那个 pos 并且对应的 <=x 的
25         for (int i = 1 << std::lg(n); i; i /= 2) {
26             if (x + i <= n && k >= a[x + i - 1]) {
27                 x += i;
28                 k -= a[x - 1];
29             }
30         }
31         return x;
32     } // 树状数组上倍增本质上是通过倍增来快速找出对应的区间
33 };
```

DSU

```
1  struct DSU {
2      std::vector<int> f, siz;
3      DSU(int n) : f(n), siz(n, 1) { std::iota(f.begin(), f.end(), 0); }
4      int leader(int x) {
5          while (x != f[x]) x = f[x] = f[f[x]];
6          return x;
7      }
8      bool same(int x, int y) { return leader(x) == leader(y); }
9      bool merge(int x, int y) {
10         x = leader(x);
11         y = leader(y);
12         if (x == y) return false;
13         siz[x] += siz[y];
14         f[y] = x;
15         return true;
16     }
17     int size(int x) { return siz[leader(x)]; }
18 };
```

Splay

```
1  struct Node {
2      int v, sz, sm;
3      Node *ch[2], *fa;
4
5      Node(const int V, Node *const f) : v(V), sz(1), sm(1), fa(f) {
6          ch[0] = ch[1] = nullptr;
7      }
8
9      inline int GetRela(const int x) { return (v == x) ? -1 : (x > v); }
10
11     void pushup() { sm = (ch[0] ? ch[0]->sm : 0) + (ch[1] ? ch[1]->sm : 0) + sz; }
```

```

12
13 inline void rotate(const int x) {
14     auto nrt = ch[x];
15     ch[x] = nrt->ch[x ^ 1];
16     nrt->ch[x ^ 1] = this;
17     if (ch[x]) ch[x]->fa = this;
18     nrt->fa = fa; fa = nrt;
19     if (nrt->fa) nrt->fa->ch[nrt->fa->GetRela(nrt->v)] = nrt;
20     pushup(); nrt->pushup();
21 }
22
23 void splay(const Node *p) {
24     while (fa != p) {
25         auto pa = fa->fa;
26         if (pa == p) {
27             fa->rotate(fa->GetRela(v));
28         } else {
29             int k1 = fa->GetRela(v), k2 = pa->GetRela(fa->v);
30             if (k1 == k2) {
31                 pa->rotate(k1);
32                 fa->rotate(k1);
33             } else {
34                 fa->rotate(k1);
35                 fa->rotate(k2);
36             }
37         }
38     }
39 }
40 };

```

LCT

```

1 struct Node {
2     int v, s;
3     bool tag;
4     Node *ch[2], *fa;
5
6     inline void maketag() {
7         tag = !tag;
8         std::swap(ch[0], ch[1]);
9     }
10    inline void pushup() {
11        s = v;
12        for (auto u : ch) if (u != nullptr) {
13            s ^= u->s;
14        }
15    }
16    inline void pushdown() {
17        if (tag) {
18            for (auto u : ch) if (u != nullptr) {
19                u->maketag();
20            }
21            tag = false;
22        }
23    }
24
25    inline int Getson() { return fa->ch[1] == this; }
26
27    inline bool IsRoot() { return (fa == nullptr) || (fa->ch[Getson()]) != this; }
28
29    void rotate(const int x) {
30        auto nt = ch[x];
31        ch[x] = nt->ch[x ^ 1];
32        nt->ch[x ^ 1] = this;
33        if (ch[x]) ch[x]->fa = this;
34        nt->fa = fa;
35        if (!IsRoot()) { fa->ch[Getson()] = nt; }
36        fa = nt;
37        pushup(); nt->pushup();
38    }
39

```

```

40 void splay() {
41     static Node* stk[maxn];
42     int top = 0;
43     stk[++top] = this;
44     for (auto u = this; !u->IsRoot(); stk[++top] = u = u->fa);
45     while (top) stk[top--]>pushdown();
46     while (!IsRoot()) {
47         if (fa->IsRoot()) {
48             fa->rotate(Getson());
49         } else {
50             auto pa = fa->fa;
51             int l1 = Getson(), l2 = fa->Getson();
52             if (l1 == l2) {
53                 pa->rotate(l2);
54                 fa->rotate(l1);
55             } else {
56                 fa->rotate(l1);
57                 fa->rotate(l2);
58             }
59         }
60     }
61 }
62 };
63 Node *node[maxn], Mem[maxn];
64
65 void Cut(const int x, const int y);
66 void Link(const int x, const int y);
67 void Query(const int x, const int y);
68 void Update(const int x, const int y);
69
70 void access(Node *u) {
71     for (Node *v = nullptr; u; u = (v = u)->fa) {
72         u->splay();
73         u->ch[1] = v; u->pushup();
74     }
75 }
76
77 void makeroot(Node *const u) {
78     access(u);
79     u->splay();
80     u->maketag();
81 }
82
83 void Query(const int x, const int y) {
84     auto u = node[x], v = node[y];
85     makeroot(u);
86     access(v);
87     v->splay();
88     qw(v->s, '\n');
89 }
90
91 void Link(const int x, const int y) {
92     auto u = node[x], v = node[y];
93     makeroot(u);
94     access(v); v->splay();
95     if (u->IsRoot() == false) return;
96     u->fa = v;
97 }
98
99 void Cut(const int x, const int y) {
100     auto u = node[x], v = node[y];
101     makeroot(u); access(v); u->splay();
102     if ((u->ch[1] != v) || (v->ch[0] != nullptr)) return;
103     u->ch[1] = v->fa = nullptr;
104     u->pushup();
105 }
106
107 // w[x] -> y
108 void Update(const int x, const int y) {
109     auto u = node[x];
110     u->splay();

```

```

111     u->s ^ = u->v;
112     u->s ^ = (u->v = a[x] = y);
113 }

```

扫描线

```

1 //二维数点
2 struct Segment{
3     int l,r,h,add;
4     bool operator <(const Segment a)const{
5         return h < a.h;
6     }
7 };
8 struct SegTree {
9     int l, r;
10    SegTree *ls, *rs;
11    int mn,len;
12    int plus;
13    SegTree (const int L, const int R) : l(L), r(R) {
14        plus = 0;len = 0;
15        if (L == R) {
16            ls = rs = nullptr;
17        } else {
18            int M = (L + R) >> 1;
19            ls = new SegTree (L, M);
20            rs = new SegTree (M + 1, R);
21            pushup();
22        }
23    }
24    void pushup() {
25        if(plus) len = r - l + 1;
26        else if(l == r)len = 0;
27        else len = ls->len + rs->len;
28    }
29    void make_tag(int w) {
30        plus += w;
31    }
32    void pushdown() {
33        if (plus == 0) return;
34        ls->make_tag(plus);
35        rs->make_tag(plus);
36        plus = 0;
37    }
38    void update(const int L, const int R, const int w) {
39        if ((L > r) || (l > R)) {
40            return;
41        }
42        if ((L <= l) && (r <= R)) {
43            make_tag(w);
44            pushup();
45            return ;
46        } else {
47            ls->update(L, R, w);
48            rs->update(L, R, w);
49            pushup();
50        }
51    }
52 };
53 //矩形面积并
54 #include<bits/stdc++.h>
55
56 using namespace std;
57 typedef long long ll;
58 const double eps = 1e-8;
59 const int maxn = 2e5 + 7;
60 std::vector<int> x;
61 struct Segment{
62     int l,r,h,add;
63     bool operator <(const Segment a)const{
64         return h < a.h;
65     }

```



```

66 };
67 struct SegTree {
68     int l, r;
69     SegTree *ls, *rs;
70     int mn, len;
71     int plus;
72     SegTree (const int L, const int R) : l(L), r(R) {
73         plus = 0; len = 0;
74         if (L == R) {
75             ls = rs = nullptr;
76         } else {
77             int M = (L + R) >> 1;
78             ls = new SegTree (L, M);
79             rs = new SegTree (M + 1, R);
80             pushup();
81         }
82     }
83     void pushup() {
84         if(plus) len = x[r] - x[l - 1];
85         else if(l == r) len = 0;
86         else len = ls->len + rs->len;
87     }
88     void make_tag(int w) {
89         plus += w;
90     }
91     void pushdown() {
92         if (plus == 0) return;
93         ls->make_tag(plus);
94         rs->make_tag(plus);
95         plus = 0;
96     }
97     void update(const int L, const int R, const int w) {
98         if ((L >= x[r]) || (x[l - 1] >= R)) {
99             return;
100         }
101         if ((L <= x[l - 1]) && (x[r] <= R)) {
102             make_tag(w);
103             pushup();
104             return ;
105         } else {
106             //pushdown();
107             ls->update(L, R, w);
108             rs->update(L, R, w);
109             pushup();
110         }
111     }
112 };
113 int main(){
114     ios::sync_with_stdio(false);
115     cin.tie(0);
116
117     vector<Segment> s;
118     int n;
119     cin >> n;
120     for(int i = 0; i < n; i++){
121         int xa, ya, xb, yb;
122         cin >> xa >> ya >> xb >> yb;
123         x.push_back(xa);
124         x.push_back(xb);
125         s.push_back({xa, xb, ya, 1});
126         s.push_back({xa, xb, yb, -1});
127     }
128     sort(s.begin(), s.end());
129     sort(x.begin(), x.end());
130     x.erase(unique(x.begin(), x.end()), x.end());
131     int N = x.size();
132     SegTree Seg(1, N - 1);
133     ll ans = 0;
134     if(s.size()){
135         Seg.update(s[0].l, s[0].r, s[0].add);
136         for(int i = 1; i < s.size(); i++){

```

```

137         ans += 1ll * Seg.len * (s[i].h - s[i - 1].h);
138         Seg.update(s[i].l, s[i].r, s[i].add);
139     }
140 }
141 cout << ans << "\n";
142 return 0;
143 }

```

Seg beats

本质上是维护了两棵线段树，A 树维护区间内最大值产生的贡献，B 树维护剩下树的贡献。注意 A 树某节点的孩子不一定全部能贡献到该节点，因为孩子的最大值不一定是父亲的最大值。所以要注意下传标记时，A 树的孩子下传的可能是 B 的标记。

beats 的部分是，每次让序列里每个数对另一个数 V 取 \min ，则直接暴力递归到 inRange 且 B 的最大值小于 V 的那些节点上，转化成对 A 那个节点的区间加法（加上 $V - \text{val}_A$ ）即可。这么做的均摊复杂度是 $O(\log n)$ 。

做区间历史最大值的方法是，维护两个标记 x, y ， x 是真正的加标记， y 是 x 在上次下传结束并清零后的历史最大值。下传时注意先下传 y 再下传 x 。实现历史最值是平凡的，不需要 beats。beats 解决的仅是取 \min 的操作。

下面五个操作分别是：区间加，区间对 k 取 \min ，区间求和，区间最大值，区间历史最大值。

```

1  #include <array>
2  #include <iostream>
3  #include <algorithm>
4
5  typedef long long int ll;
6
7  const int maxn = 500005;
8
9  ll a[maxn];
10
11 const ll inf = 0x3f3f3f3f3f3f3f3fll;
12
13 struct Node {
14     Node *ls, *rs;
15     int l, r, maxCnt;
16     ll v, add, maxAdd, sum, maxV, maxHistory;
17
18     Node(const int L, const int R) :
19         ls(nullptr), rs(nullptr), l(L), r(R), maxCnt(0),
20         v(0), add(0), maxAdd(0), sum(0), maxV(-inf), maxHistory(-inf) {}
21
22     inline bool inRange(const int L, const int R) {
23         return L <= l && r <= R;
24     }
25     inline bool outRange(const int L, const int R) {
26         return l > R || L > r;
27     }
28
29     void addVal(const ll t, int len) {
30         add += t;
31         sum += len * t;
32         maxV += t;
33     }
34
35     void makeAdd(const ll t, int len) {
36         addVal(t, len);
37         maxHistory = std::max(maxHistory, maxV);
38         maxAdd = std::max(maxAdd, add);
39     }
40 };
41
42 void pushup(Node *x, Node *y) {
43     y->maxV = std::max(y->ls->maxV, y->rs->maxV);
44     y->sum = y->ls->sum + y->rs->sum;
45     y->maxHistory = std::max({y->maxHistory, y->ls->maxHistory, y->rs->maxHistory});
46     if (x->ls->maxV != x->rs->maxV) {
47         bool flag = x->ls->maxV < x->rs->maxV;
48         if (flag) std::swap(x->ls, x->rs);
49         x->maxV = x->ls->maxV;

```

```

50     x->maxCnt = x->ls->maxCnt;
51     y->maxV = std::max(y->maxV, x->rs->maxV);
52     y->sum += x->rs->sum;
53     x->sum = x->ls->sum;
54     if (flag) std::swap(x->ls, x->rs);
55 } else {
56     x->maxCnt = x->ls->maxCnt + x->rs->maxCnt;
57     x->sum = x->ls->sum + x->rs->sum;
58     x->maxV = x->ls->maxV;
59 }
60 x->maxHistory = std::max({x->ls->maxHistory, x->rs->maxHistory, x->maxHistory, y->maxHistory});
61 }
62
63 void New(Node *&u1, Node *&u2, int L, int R) {
64     u1 = new Node(L, R);
65     u2 = new Node(L, R);
66     if (L == R) {
67         u1->v = u1->sum = u1->maxV = u1->maxHistory = a[L];
68         u1->maxCnt = 1;
69     } else {
70         int M = (L + R) >> 1;
71         New(u1->ls, u2->ls, L, M);
72         New(u1->rs, u2->rs, M + 1, R);
73         pushup(u1, u2);
74     }
75 }
76
77 void pushdown(Node *x, Node *y) {
78     ll val = std::max(x->ls->maxV, x->rs->maxV);
79     std::array<Node*, 2> aim({y, x});
80     Node *curl = aim[x->ls->maxV == val], *curr = aim[x->rs->maxV == val];
81     x->ls->maxAdd = std::max(x->ls->maxAdd, x->ls->add + curl->maxAdd);
82     x->ls->maxHistory = std::max(x->ls->maxHistory, x->ls->maxV + curl->maxAdd);
83     x->ls->addVal(curl->add, x->ls->maxCnt);
84     x->rs->maxAdd = std::max(x->rs->maxAdd, x->rs->add + curr->maxAdd);
85     x->rs->maxHistory = std::max(x->rs->maxHistory, x->rs->maxV + curr->maxAdd);
86     x->rs->addVal(curr->add, x->rs->maxCnt);
87     y->ls->maxAdd = std::max(y->ls->maxAdd, y->ls->add + y->maxAdd);
88     y->rs->maxAdd = std::max(y->rs->maxAdd, y->rs->add + y->maxAdd);
89     y->ls->addVal(y->add, x->ls->r - x->ls->l + 1 - x->ls->maxCnt);
90     y->rs->addVal(y->add, x->rs->r - x->rs->l + 1 - x->rs->maxCnt);
91     x->add = y->add = x->maxAdd = y->maxAdd = 0;
92 }
93
94 void addV(Node *x, Node *y, int L, int R, ll k) {
95     if (x->inRange(L, R)) {
96         x->makeAdd(k, x->maxCnt);
97         y->makeAdd(k, x->r - x->l + 1 - x->maxCnt);
98     } else if (!x->outRange(L, R)) {
99         pushdown(x, y);
100         addV(x->ls, y->ls, L, R, k);
101         addV(x->rs, y->rs, L, R, k);
102         pushup(x, y);
103     }
104 }
105
106 std::array<ll, 3> qry(Node *x, Node *y, const int L, const int R) {
107     if (x->inRange(L, R)) return {x->sum + y->sum * ((x->r - x->l + 1) != x->maxCnt), x->maxV, x->maxHistory};
108     else if (x->outRange(L, R)) return {0, -inf, -inf};
109     else {
110         pushdown(x, y);
111         auto A = qry(x->ls, y->ls, L, R), B = qry(x->rs, y->rs, L, R);
112         return {A[0] + B[0], std::max(A[1], B[1]), std::max(A[2], B[2])};
113     }
114 }
115
116 void minV(Node *x, Node *y, const int L, const int R, int k) {
117     if (x->maxV <= k) return;
118     if (x->inRange(L, R) && y->maxV < k) {
119         ll delta = k - x->maxV;
120         x->makeAdd(delta, x->maxCnt);

```

```

121     } else if (!x->outRange(L, R)) {
122         pushdown(x, y);
123         minV(x->ls, y->ls, L, R, k);
124         minV(x->rs, y->rs, L, R, k);
125         pushup(x, y);
126     }
127 }
128
129 int main() {
130     std::ios::sync_with_stdio(false);
131     std::cin.tie(nullptr);
132     int n, m;
133     std::cin >> n >> m;
134     for (int i = 1; i <= n; ++i) std::cin >> a[i];
135     Node *rot1, *rot2;
136     New(rot1, rot2, 1, n);
137     for (int op, l, r; m; --m) {
138         std::cin >> op >> l >> r;
139         if (op == 1) {
140             std::cin >> op;
141             addV(rot1, rot2, l, r, op);
142         } else if (op == 2) {
143             std::cin >> op;
144             minV(rot1, rot2, l, r, op);
145         } else {
146             std::cout << qry(rot1, rot2, l, r)[op - 3] << '\n';
147         }
148     }
149 }

```

珂朵莉树

```

1  auto getPos(int pos) {
2      return --s.upper_bound({pos + 1, 0, 0});
3  }
4
5  void split(int pos) {
6      auto it = getPos(pos);
7      auto [l, r, v] = *it;
8      s.erase(it);
9      if (pos > l) s.insert({l, pos - 1, v});
10     s.insert({pos, r, v});
11 }
12
13 void add(int l, int r, int v) {
14     split(l); split(r + 1);
15     for (auto x = getPos(l), y = getPos(r + 1); x != y; ++x) {
16         x->v += v;
17     }
18 }
19
20 void upd(int l, int r, int v) {
21     split(l); split(r + 1);
22     s.erase(getPos(l), getPos(r + 1));
23     s.insert({l, r, v});
24 }

```

getPos(pos): 找到 pos 所在的迭代器 split(pos): 把 pos 所在的迭代器区间 [l, r] 分成 [l, pos - 1] 和 [pos, r] 两个

李超树

插入线段 $kx + b$ 求某点最值

```

1  constexpr long long INF = 1'000'000'000'000'000'000;
2  constexpr int C = 100'000;
3  struct Line {
4      int k;
5      long long b;
6      Line(int k, long long b) : k(k), b(b) {}
7  };

```

```

8  long long f(const Line &line, int x) {
9      return 1LL * line.k * x + line.b;
10 }
11 struct Node {
12     Node *lc, *rc;
13     Line line;
14     Node(const Line &line) : lc(nullptr), rc(nullptr), line(line) {}
15 };
16 void modify(Node *&p, int l, int r, Line line) {
17     if (p == nullptr) {
18         p = new Node(line);
19         return;
20     }
21     int m = (l + r) / 2;
22     bool le = f(p -> line, l) < f(line, l);
23     bool mi = f(p -> line, m) < f(line, m);
24     if (!mi)
25         std::swap(p -> line, line);
26     if (r - l == 1)
27         return;
28     if (le != mi) {
29         modify(p -> lc, l, m, line);
30     } else {
31         modify(p -> rc, m, r, line);
32     }
33 }
34 Node *merge(Node *p, Node *q, int l, int r) {
35     if (p == nullptr)
36         return q;
37     if (q == nullptr)
38         return p;
39     int m = (l + r) / 2;
40     p -> lc = merge(p -> lc, q -> lc, l, m);
41     p -> rc = merge(p -> rc, q -> rc, m, r);
42     modify(p, l, r, q -> line);
43     return p;
44 }
45 long long query(Node *p, int l, int r, int x) {
46     if (p == nullptr)
47         return INF;
48     long long ans = f(p -> line, x);
49     if (r - l == 1)
50         return ans;
51     int m = (l + r) / 2;
52     if (x < m) {
53         return std::min(ans, query(p -> lc, l, m, x));
54     } else {
55         return std::min(ans, query(p -> rc, m, r, x));
56     }
57 }

```

动态维护凸壳

```

1  /**
2   * Author: Simon Lindholm
3   * Date: 2017-04-20
4   * License: CC0
5   * Source: own work
6   * Description: Container where you can add lines of the form  $kx+m$ , and query maximum values at points  $x$ .
7   * Useful for dynamic programming.
8   * Time:  $O(\log N)$ 
9   * Status: tested
10 */
11
12 struct Line {
13     mutable ll k, m, p;
14     bool operator<(const Line &o) const { return k < o.k; }
15     bool operator<(ll x) const { return p < x; }
16 };
17
18 struct LineContainer: multiset<Line, less<>> {

```

```

19  const ll inf = LLONG_MAX;
20  ll val_offset = 0;
21  void offset(ll x) {
22      val_offset += x; //整体加
23  }
24  ll div(ll a, ll b) {
25      return a / b - ((a^b) < 0 && a%b);
26  }
27  bool isect(iterator x, iterator y) {
28      if (y == end()) {
29          x->p = inf;
30          return 0;
31      }
32      if (x->k == y->k) {
33          x->p = (x->m > y->m)? inf: -inf;
34      } else {
35          x->p = div(y->m - x->m, x->k - y->k);
36      }
37      return x->p >= y->p;
38  }
39  void add(ll k, ll m) {
40      auto z = insert({k, m - val_offset, 0}), y = z++, x = y; //这里加减看情况
41      while (isect(y, z)) z = erase(z);
42      if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
43      while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
44  }
45  ll query(ll x) {
46      assert(!empty());
47      auto l = *lower_bound(x);
48      return l.k * x + l.m + val_offset;
49  }
50 };
51
52 LineContainer* merge(LineContainer *S, LineContainer *T) {
53     if (S->size() > T->size())
54         swap(S, T);
55     for (auto l: *S) {
56         T->add(l.k, l.m + S->val_offset);
57     }
58     return T;
59 }

```

TODO

线段树合并和分裂

图论

树链剖分

```

1  // 重链剖分
2  void dfs1(int x) {
3      son[x] = -1;
4      siz[x] = 1;
5      for (auto v:e[x])
6          if (!dep[v]) {
7              dep[v] = dep[x] + 1;
8              fa[v] = x;
9              dfs1(v);
10             siz[x] += siz[v];
11             if (son[x] == -1 || siz[v] > siz[son[x]]) son[x] = v;
12         }
13  }
14
15  void dfs2(int x, int t) {
16      top[x] = t;
17      dfn[x] = ++ cnt;
18      rnk[cnt] = x;
19      if (son[x] == -1) return;
20      dfs2(son[x], t);

```

```

21     for (auto v:e[x])
22         if (v != son[x] && v != fa[x]) dfs2(v, v);
23     }
24     int lca(int u, int v) {
25         while (top[u] != top[v]) {
26             if (dep[top[u]] > dep[top[v]])
27                 u = fa[top[u]];
28             else
29                 v = fa[top[v]];
30         }
31         return dep[u] > dep[v] ? v : u;
32     }

```

LCA

倍增求 LCA

```

1  void dfs(int x){
2      for(int j = 1;j <= 19;j++){
3          f[x][j] = f[f[x][j-1]][j-1];
4      }
5      for(auto v:e[x]){
6          if(v == f[x][0])continue;
7          f[v][0] = x;
8          dep[v] = dep[x] + 1;
9          dfs(v);
10     }
11 }
12 int lca(int u,int v){
13     if(dep[u] < dep[v])swap(u,v);
14     for(int i = 0;i <= 19;i++){
15         if((dep[u] - dep[v]) & (1 << i))u = f[u][i];
16     }
17     if(u == v)return u;
18     for(int j = 19;j >= 0; j--){
19         if(f[u][j] != f[v][j]){
20             u = f[u][j];
21             v = f[v][j];
22         }
23     }
24     return f[u][0];
25 }
26 int kth(int x,int k){
27     for(int i = 0;i <= 19;i++){
28         if(k & (1 << i))x = f[x][i];
29     }
30     return x;
31 }

```

dfn 求 LCA

```

1  int get(int x, int y) {return dfn[x] < dfn[y] ? x : y;}
2  void dfs(int id, int f) {
3      mi[0][dfn[id] = ++dn] = f;
4      for(int it : e[id]) if(it != f) dfs(it, id);
5  }
6  int lca(int u, int v) {
7      if(u == v) return u;
8      if((u = dfn[u]) > (v = dfn[v])) swap(u, v);
9      int d = __lg(v - u++);
10     return get(mi[d][u], mi[d][v - (1 << d) + 1]);
11 }
12 dfs(R, 0);
13 for(int i = 1; i <= __lg(n); i++)
14     for(int j = 1; j + (1 << i) - 1 <= n; j++)
15         mi[i][j] = get(mi[i-1][j], mi[i-1][j + (1 << i - 1)]);

```

树哈希

```
1 typedef unsigned long long ull;
2 struct TreeHash{
3     std::vector<int> hs;
4     TreeHash(int n){
5         hs.resize(n,0);
6     }
7     mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
8     ull bas = rnd();
9     ull H(ull x){
10         return x*x*x*19890535+19260817;
11     }
12     ull F(ull x){
13         return H(x & ((1ll << 32) - 1)) + H(x >> 32);
14     }
15     int flag,n;
16     void dfs(int u,int fa){
17         hs[u] = bas;
18         for(auto v:e[u]){
19             if(v == fa) continue;
20             dfs(v,u);
21             hs[u] += F(hs[v]);
22         }
23     }
24 };
```

虚树

```
1 void build_virtual_tree(vector<int> &h) {
2     vector<int> a;
3     sort(h.begin(), h.end(), [&](int &a,int &b){
4         return dfn[a] < dfn[b];
5     }); // 把关键点按照 dfn 序排序
6     for (int i = 0; i < h.size(); ++i) {
7         a.push_back(h[i]);
8         if(i + 1 != h.size())a.push_back(lca(h[i], h[i + 1])); // 插入 lca
9     }
10    sort(a.begin(), a.end(), [&](int &a,int &b){
11        return dfn[a] < dfn[b];
12    }); // 把所有虚树上的点按照 dfn 序排序
13    a.erase(unique(a.begin(),a.end()),a.end());
14    for (int i = 0; i < a.size() - 1; ++i) {
15        int lc = lca(a[i], a[i + 1]);
16        add(lc, a[i + 1]); // 连边, 如有边权 就是 distance(lc,a[i+1])
17    }
18 }
```

Dijkstra

```
1 void dijkstra(int s) {
2     memset(dis, 0x3f, sizeof(dis));
3     dis[s] = 0;
4     priority_queue<pair<int,int>> q;
5     q.push(make_pair(0, s));
6     while(!q.empty()) {
7         auto x = q.top().second;
8         q.pop();
9         if(vis[x]) continue;
10        vis[x] = 1;
11        for(auto [v,w] : e[x]) {
12            if(dis[v] > dis[x] + w) {
13                dis[v] = dis[x] + w;
14                q.push({-dis[v],v});
15            }
16        }
17    }
18 }
```


最小环

```
1 //floyd 找最小环
2 //dijkstra 暴力删边跑最短路-
3 int floyd(const int &n) {
4     for (int i = 1; i <= n; ++i)
5         for (int j = 1; j <= n; ++j)
6             dis[i][j] = f[i][j]; // 初始化最短路矩阵
7     int ans = inf;
8     for (int k = 1; k <= n; ++k) {
9         for (int i = 1; i < k; ++i)
10            for (int j = 1; j < i; ++j)
11                ans = std::min(ans, dis[i][j] + f[i][k] + f[k][j]); // 更新答案
12        for (int i = 1; i <= n; ++i)
13            for (int j = 1; j <= n; ++j)
14                dis[i][j] = std::min(dis[i][j], dis[i][k] + dis[k][j]); // 正常的 floyd 更新最短路矩阵
15    }
16    return ans;
17 }
```

差分约束

$$x_i + C \geq x_j$$

最短路-> 最大解

最长路-> 最小解

判负环或正环即可

```
1 bool spfa(){
2     queue<int> q;
3     vector<int> vis(n + 1), cnt(n + 1), dis(n + 1, 1e9);
4     dis[1] = 0;
5     cnt[1] = 1;
6     q.push(1);
7     while(!q.empty()){
8         int u = q.front();
9         q.pop();
10        vis[u] = 0;
11        if(cnt[u] >= n) return 1;
12        for(auto v:e[u]){
13            if(dis[v] > dis[u] + len[p]){
14                dis[v] = dis[u] + len[p];
15                if(vis[v] == 0){
16                    vis[v] = 1;
17                    q.push(v);
18                    cnt[v] ++;
19                }
20            }
21        }
22    }
23    return 0;
24 }
```

最大流

```
1 struct Flow {
2     static constexpr int INF = 1e9;
3     int n;
4     struct Edge {
5         int to, cap;
6         Edge(int to, int cap) : to(to), cap(cap) {}
7     };
8     vector<Edge> e;
9     vector<vector<int>> g;
10    vector<int> cur, h;
11    Flow(int n) : n(n), g(n) {}
12    void init(int n) {
13        for (int i = 0; i < n; i++) g[i].clear();
14        e.clear();
15    }
```

```

15     }
16     bool bfs(int s, int t) {
17         h.assign(n, -1);
18         queue<int> que;
19         h[s] = 0;
20         que.push(s);
21         while (!que.empty()) {
22             int u = que.front();
23             que.pop();
24             for (int i : g[u]) {
25                 int v = e[i].to;
26                 int c = e[i].cap;
27                 if (c > 0 && h[v] == -1) {
28                     h[v] = h[u] + 1;
29                     if (v == t)
30                         return true;
31                     que.push(v);
32                 }
33             }
34         }
35         return false;
36     }
37     int dfs(int u, int t, int f) {
38         if (u == t)
39             return f;
40         int r = f;
41         for (int &i = cur[u]; i < int(g[u].size()); ++i) {
42             int j = g[u][i];
43             int v = e[j].to;
44             int c = e[j].cap;
45             if (c > 0 && h[v] == h[u] + 1) {
46                 int a = dfs(v, t, std::min(r, c));
47                 e[j].cap -= a;
48                 e[j ^ 1].cap += a;
49                 r -= a;
50                 if (r == 0)
51                     return f;
52             }
53         }
54         return f - r;
55     }
56     void addEdge(int u, int v, int c) {
57         g[u].push_back(e.size());
58         e.push_back({v, c});
59         g[v].push_back(e.size());
60         e.push_back({u, 0});
61     }
62     int maxFlow(int s, int t) {
63         int ans = 0;
64         while (bfs(s, t)) {
65             cur.assign(n, 0);
66             ans += dfs(s, t, INF);
67         }
68         return ans;
69     }
70 };

```

最小费用最大流

```

1  using i64 = long long;
2
3  struct MCFGraph {
4      struct Edge {
5          int v, c, f;
6          Edge(int v, int c, int f) : v(v), c(c), f(f) {}
7      };
8      const int n;
9      std::vector<Edge> e;
10     std::vector<std::vector<int>> g;
11     std::vector<i64> h, dis;
12     std::vector<int> pre;

```

```

13 bool dijkstra(int s, int t) {
14     dis.assign(n, std::numeric_limits<i64>::max());
15     pre.assign(n, -1);
16     priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<pair<i64, int>>> que;
17     dis[s] = 0;
18     que.emplace(0, s);
19     while (!que.empty()) {
20         i64 d = que.top().first;
21         int u = que.top().second;
22         que.pop();
23         if (dis[u] < d) continue;
24         for (int i : g[u]) {
25             int v = e[i].v;
26             int c = e[i].c;
27             int f = e[i].f;
28             if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
29                 dis[v] = d + h[u] - h[v] + f;
30                 pre[v] = i;
31                 que.emplace(dis[v], v);
32             }
33         }
34     }
35     return dis[t] != std::numeric_limits<i64>::max();
36 }
37 MCFGGraph(int n) : n(n), g(n) {}
38 void addEdge(int u, int v, int c, int f) {
39     if (f < 0) {
40         g[u].push_back(e.size());
41         e.emplace_back(v, 0, f);
42         g[v].push_back(e.size());
43         e.emplace_back(u, c, -f);
44     } else {
45         g[u].push_back(e.size());
46         e.emplace_back(v, c, f);
47         g[v].push_back(e.size());
48         e.emplace_back(u, 0, -f);
49     }
50 }
51 std::pair<int, i64> flow(int s, int t) {
52     int flow = 0;
53     i64 cost = 0;
54     h.assign(n, 0);
55     while (dijkstra(s, t)) {
56         for (int i = 0; i < n; ++i) h[i] += dis[i];
57         int aug = std::numeric_limits<int>::max();
58         for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug, e[pre[i]].c);
59         for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
60             e[pre[i]].c -= aug;
61             e[pre[i] ^ 1].c += aug;
62         }
63         flow += aug;
64         cost += i64(aug) * h[t];
65     }
66     return std::make_pair(flow, cost);
67 }
68 };

```

二分图最大匹配

```

1 auto dfs = [&](auto &&dfs, int u, int tag) -> bool {
2     if (vistime[u] == tag) return false;
3     vistime[u] = tag;
4     for (auto v : e[u]) if (!mtch[v] || dfs(dfs, mtch[v], tag)) {
5         mtch[v] = u;
6         return true;
7     }
8     return false;
9 };

```

KM(二分图最大权匹配)

```
1  template <typename T>
2  struct hungarian { // km
3      int n;
4      vector<int> matchx; // 左集合对应的匹配点
5      vector<int> matchy; // 右集合对应的匹配点
6      vector<int> pre; // 连接右集合的左点
7      vector<bool> visx; // 拜访数组 左
8      vector<bool> visy; // 拜访数组 右
9      vector<T> lx;
10     vector<T> ly;
11     vector<vector<T> > g;
12     vector<T> slack;
13     T inf;
14     T res;
15     queue<int> q;
16     int org_n;
17     int org_m;
18
19     hungarian(int _n, int _m) {
20         org_n = _n;
21         org_m = _m;
22         n = max(_n, _m);
23         inf = numeric_limits<T>::max();
24         res = 0;
25         g = vector<vector<T> >(n, vector<T>(n));
26         matchx = vector<int>(n, -1);
27         matchy = vector<int>(n, -1);
28         pre = vector<int>(n);
29         visx = vector<bool>(n);
30         visy = vector<bool>(n);
31         lx = vector<T>(n, -inf);
32         ly = vector<T>(n);
33         slack = vector<T>(n);
34     }
35
36     void addEdge(int u, int v, int w) {
37         g[u][v] = max(w, 0); // 负值还不如不匹配 因此设为 0 不影响
38     }
39
40     bool check(int v) {
41         visy[v] = true;
42         if (matchy[v] != -1) {
43             q.push(matchy[v]);
44             visx[matchy[v]] = true; // in S
45             return false;
46         }
47         // 找到新的未匹配点 更新匹配点 pre 数组记录着"非匹配边"上与之相连的点
48         while (v != -1) {
49             matchy[v] = pre[v];
50             swap(v, matchx[pre[v]]);
51         }
52         return true;
53     }
54
55     void bfs(int i) {
56         while (!q.empty()) {
57             q.pop();
58         }
59         q.push(i);
60         visx[i] = true;
61         while (true) {
62             while (!q.empty()) {
63                 int u = q.front();
64                 q.pop();
65                 for (int v = 0; v < n; v++) {
66                     if (!visy[v]) {
67                         T delta = lx[u] + ly[v] - g[u][v];
68                         if (slack[v] >= delta) {
69                             pre[v] = u;
```

```

70         if (delta) {
71             slack[v] = delta;
72         } else if (check(v)) { // delta=0 代表有机会加入相等子图 找增广路
73             // 找到就 return 重建交错树
74             return;
75         }
76     }
77 }
78 }
79 }
80 // 没有增广路 修改顶标
81 T a = inf;
82 for (int j = 0; j < n; j++) {
83     if (!visy[j]) {
84         a = min(a, slack[j]);
85     }
86 }
87 for (int j = 0; j < n; j++) {
88     if (visx[j]) { // S
89         lx[j] -= a;
90     }
91     if (visy[j]) { // T
92         ly[j] += a;
93     } else { // T'
94         slack[j] -= a;
95     }
96 }
97 for (int j = 0; j < n; j++) {
98     if (!visy[j] && slack[j] == 0 && check(j)) {
99         return;
100     }
101 }
102 }
103 }
104
105 void solve() {
106     // 初始顶标
107     for (int i = 0; i < n; i++) {
108         for (int j = 0; j < n; j++) {
109             lx[i] = max(lx[i], g[i][j]);
110         }
111     }
112
113     for (int i = 0; i < n; i++) {
114         fill(slack.begin(), slack.end(), inf);
115         fill(visx.begin(), visx.end(), false);
116         fill(visy.begin(), visy.end(), false);
117         bfs(i);
118     }
119
120     // custom
121     for (int i = 0; i < n; i++) {
122         if (g[i][matchx[i]] > 0) {
123             res += g[i][matchx[i]];
124         } else {
125             matchx[i] = -1;
126         }
127     }
128     cout << res << "\n";
129     for (int i = 0; i < org_n; i++) {
130         cout << matchx[i] + 1 << " ";
131     }
132     cout << "\n";
133 }
134 };

```

一般图最大匹配

```

1 #include <bits/stdc++.h>
2 struct Graph {
3     int n;

```

```

4     std::vector<std::vector<int>> e;
5     Graph(int n) : n(n), e(n + 1) {}
6     void addEdge(int u, int v) {
7         e[u].push_back(v);
8         e[v].push_back(u);
9     }
10    std::vector<int> findMatching() {
11        std::vector<int> match(n + 1, -1), vis(n + 1), link(n + 1), f(n + 1), dep(n + 1);
12
13        // disjoint set union
14        auto find = [&](int u) {
15            while (f[u] != u)
16                u = f[u] = f[f[u]];
17            return u;
18        };
19
20        auto lca = [&](int u, int v) {
21            u = find(u);
22            v = find(v);
23            while (u != v) {
24                if (dep[u] < dep[v])
25                    std::swap(u, v);
26                u = find(link[match[u]]);
27            }
28            return u;
29        };
30
31        std::queue<int> q;
32        auto blossom = [&](int u, int v, int p) {
33            while (find(u) != p) {
34                link[u] = v;
35                v = match[u];
36                if (vis[v] == 0) {
37                    vis[v] = 1;
38                    q.push(v);
39                }
40                f[u] = f[v] = p;
41                u = link[v];
42            }
43        };
44
45        // find an augmenting path starting from u and augment (if exist)
46        auto augment = [&](int u) {
47
48            while (!q.empty())
49                q.pop();
50
51            std::iota(f.begin(), f.end(), 0);
52
53            // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
54            std::fill(vis.begin(), vis.end(), -1);
55
56            q.push(u);
57            vis[u] = 1;
58            dep[u] = 0;
59
60            while (!q.empty()) {
61                int u = q.front();
62                q.pop();
63                for (auto v : e[u]) {
64                    if (vis[v] == -1) {
65
66                        vis[v] = 0;
67                        link[v] = u;
68                        dep[v] = dep[u] + 1;
69                        // found an augmenting path
70                        if (match[v] == -1) {
71                            for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
72                                temp = match[y];
73                                match[x] = y;
74                                match[y] = x;

```

```

75         }
76         return;
77     }
78     vis[match[v]] = 1;
79     dep[match[v]] = dep[u] + 2;
80     q.push(match[v]);
81
82     } else if (vis[v] == 1 && find(v) != find(u)) {
83         // found a blossom
84         int p = lca(u, v);
85         blossom(u, v, p);
86         blossom(v, u, p);
87     }
88 }
89 }
90
91 };
92
93 // find a maximal matching greedily (decrease constant)
94 auto greedy = [&]() {
95
96     for (int u = 1; u <= n; ++u) {
97         if (match[u] != -1)
98             continue;
99         for (auto v : e[u]) {
100             if (match[v] == -1) {
101                 match[u] = v;
102                 match[v] = u;
103                 break;
104             }
105         }
106     }
107 };
108
109 greedy();
110
111 for (int u = 1; u <= n; ++u)
112     if (match[u] == -1)
113         augment(u);
114
115 return match;
116 }
117 };
118 int main() {
119     std::ios::sync_with_stdio(false);
120     std::cin.tie(nullptr);
121     int n, m;
122     std::cin >> n >> m;
123     Graph g(n);
124     for (int i = 0; i < m; ++i) {
125         int u, v;
126         std::cin >> u >> v;
127         g.addEdge(u, v);
128     }
129     auto match = g.findMatching();
130     int ans = 0;
131     for (int u = 1; u <= n; ++u)
132         if (match[u] != -1)
133             ++ans;
134     std::cout << ans / 2 << "\n";
135     for (int u = 1; u <= n; ++u)
136         if (match[u] != -1) std::cout << match[u] << " ";
137     else std::cout << 0 << " ";
138     return 0;
139 }

```

缩点 SCC

```

1 void dfs(const int u) {
2     low[u] = dfn[u] = ++cnt;
3     ins[stk[++top] = u] = true;

```

```

4     for (auto v : e[u]) if (dfn[v] == 0) {
5         dfs(v);
6         low[u] = std::min(low[u], low[v]);
7     } else if (ins[v]) {
8         low[u] = std::min(low[u], dfn[v]);
9     }
10    if (low[u] == dfn[u]) {
11        ++scnt; int v;
12        do {
13            ins[v = stk[top--]] = false;
14            w[bel[v] = scnt] += a[v];
15        } while (u != v);
16    }
17 }

```

割点与桥

```

1 //割点
2 void tarjan(int u, int fa){
3     dfn[u] = low[u] = ++cnt; int du = 0;
4     for (for v:e[x]){
5         if(v == fa) continue;
6         if(!dfn[v]){ ++du;
7             tarjan(v, u); low[u] = min(low[u], low[v]);
8             if(low[v] >= dfn[u] && fa) vis[u] = 1;
9         }
10        else low[u] = min(low[u], dfn[v]);
11    }
12    if(!fa && du > 1) vis[u] = 1;
13 }
14 //桥
15 void tarjan(int u, int fa) {
16     f[u] = fa;
17     low[u] = dfn[u] = ++cnt;
18     for (auto v:e[u]) {
19         if (!dfn[v]) {
20             tarjan(v, u);
21             low[u] = min(low[u], low[v]);
22             if (low[v] > dfn[u]) {
23                 isbridge[v] = true;
24                 ++cnt_bridge;
25             }
26         } else if (dfn[v] < dfn[u] && v != fa) {
27             low[u] = min(low[u], dfn[v]);
28         }
29     }
30 }

```

边双缩点

```

1 void form(int x){
2     std::vector<int> tmp;
3     int now = 0;
4     do{
5         now = s[top --];
6         tmp.push_back(now);
7     }while(now != x);
8     ans.push_back(tmp);
9 }
10 void tarjan(int x,int now){
11     dfn[x] = low[x] = ++cnt;
12     s[++ top] = x;
13     for(auto [v,_]:e[x]){
14         if(_ == now)continue;
15         if(!dfn[v]){
16             tarjan(v,_);
17             low[x] = min(low[x],low[v]);
18             if(low[v] > dfn[x]){
19                 form(v);
20             }

```



```

21         }else low[x] = min(low[x],dfn[v]);
22     }
23 }
24 }
25 for(int i = 1;i <= n;i ++){
26     if(dfn[i] == 0){
27         tarjan(i,0);
28         form(i);
29     }
30 }
31 cout << ans.size() << "\n";
32 for(auto A:ans){
33     cout << A.size() << " ";
34     for(auto x:A){
35         cout << x << " ";
36     }cout << "\n";
37 }

```

圆方树

```

1 void dfs(int u) {
2     static int cnt = 0;
3     dfn[u] = low[u] = ++cnt;
4     for (auto [v,w]:e[u]) {
5         if (v == fa[u]) continue;
6         if (!dfn[v]) {
7             fa[v] = u; fr[v] = w;
8             dfs(v); low[u] = min(low[u], low[v]);
9         }
10        else low[u] = min(low[u], dfn[v]);
11        if (low[v] > dfn[u]) add(u, v, w); // 圆 - 圆
12    }
13    for (auto [v,w]:e[u]) {
14        if (u == fa[v] || dfn[v] < dfn[u]) continue;
15        add(u, v, w); // 圆 - 方
16    }
17 }

```

广义圆方树

跟普通圆方树没有太大的区别，大概就是对于每个点双新建一个方点，然后将点双中的所有点向方点连边

需要注意的是我的写法中，两个点一条边也视为一个点双

性质

1. 树上的每一条边都连接了一个圆点和一个方点
2. 每个点双有唯一的方点
3. 一条从圆点到圆点的树上简单路径代表原图的中的一堆路径，其中圆点是必须经过的，而方点 (指的是与方点相连的点双) 是可以随便走的，也可以理解成原图中两点简单路径的并

```

1 void dfs(int x) {
2     stk.push_back(x);
3     dfn[x] = low[x] = cur++;
4
5     for (auto y : adj[x]) {
6         if (dfn[y] == -1) {
7             dfs(y);
8             low[x] = std::min(low[x], low[y]);
9             if (low[y] == dfn[x]) {
10                 int v;
11                 do {
12                     v = stk.back();
13                     stk.pop_back();
14                     edges.emplace_back(n + cnt, v);
15                 } while (v != y);
16                 edges.emplace_back(x, n + cnt);
17                 cnt++;
18             }
19         } else {

```

```

20         low[x] = std::min(low[x], dfn[y]);
21     }
22 }
23 }

```

2-SAT

输出方案时可以通过变量在图中的拓扑序确定该变量的取值。如果变量 x 的拓扑序在 $\neg x$ 之后，那么取 x 值为真。应用到 Tarjan 算法的缩点，即 x 所在 SCC 编号在 $\neg x$ 之前时，取 x 为真。因为 Tarjan 算法求强连通分量时使用了栈，所以 Tarjan 求得的 SCC 编号相当于反拓扑序。

环计数

```

1 //三元环
2 for (int u, v; m; --m) {
3     u = A[m]; v = B[m];
4     if (d[u] > d[v]) {
5         std::swap(u, v);
6     } else if ((d[u] == d[v]) && (u > v)) {
7         std::swap(u, v);
8     }
9     e[u].push_back(v);
10 }
11 for (int u = 1; u <= n; ++u) {
12     for (auto v : e[u]) vis[v] = u;
13     for (auto v : e[u]) {
14         for (auto w : e[v]) if (vis[w] == u) {
15             ++ans;
16         }
17     }
18 }
19 // 四元环
20 auto cmp = [&](int &a, int &b){
21     if(d[a] != d[b])return d[a] > d[b];
22     else return a < b;
23 }
24 for(int u = 1; u <= n; ++u) {
25     for(auto v: G[u])//G 为原图
26         for(auto w: e[v])
27             if(cmp(u,w)) (ans += vis[w]++)%MOD;
28     for(auto v: G[u])
29         for(auto w: e[v])
30             if(cmp(u,w)) vis[w] = 0;
31 }

```

字符串

manacher

```

1 struct Manacher {
2     int n, l, f[maxn * 2], Len;
3     char s[maxn * 2];
4
5     void init(char *c) {
6         l = strlen(c + 1); s[0] = '~';
7         for (int i = 1, j = 2; i <= l; ++i, j += 2)
8             s[j] = c[i], s[j - 1] = '#';
9         n = 2 * l + 1; s[n] = '#'; s[n + 1] = '\0';
10    }
11    void manacher() {
12        int p = 0, mr = 0;
13        for (int i = 1; i <= n; ++i) f[i] = 0;
14        for (int i = 1; i <= n; ++i) {
15            if (i < mr) f[i] = min(f[2 * p - i], mr - i);
16            while (s[i + f[i]] == s[i - f[i]]) ++f[i]; --f[i];
17            if (f[i] + i > mr) mr = i + f[i], p = i;
18            Len = max(Len, f[i]);
19        }
20    }
21 }

```

```

20     }
21
22     void solve() {
23         for (int i = 1; i <= n; ++i) {
24             // [1, l]
25             int L = i - f[i] + 1 >> 1, R = i + f[i] - 1 >> 1;
26             if (!f[i]) continue;
27
28             // [1, 2 * l + 1]
29             L = i - f[i], R = i + f[i];
30         }
31     }
32 } M;

```

SA

sa_i 表示排名为 i 的后缀。

rnk_i 表示 $[i, n]$ 这个后缀的排名（在 SA 里的下标）。

$height_i$ 是 sa_i 和 sa_{i-1} 的 LCP 长度。换句话说，向求排名为 i 的后缀和排名为 $i-1$ 的后缀的 LCP 直接就是 $height_i$ ；求 $[i, n]$ 这个后缀和它在 sa 里前一个串的 LCP 就是 $height_{rnk_i}$

```

1  const int maxn = 1000005;
2
3  int sa[maxn], rnk[maxn], tax[maxn], tp[maxn], height[maxn];
4  void SA(string s) {
5      int n = s.size();
6      s = '#' + s;
7      m = SIGMA_SIZE;
8      vector<int> S(n + 1);
9      auto RadixSort = [&]() {
10         for (int i = 0; i <= m; ++i) tax[i] = 0;
11         for (int i = 1; i <= n; ++i) ++tax[rnk[i]];
12         for (int i = 1; i <= m; ++i) tax[i] += tax[i - 1];
13         for (int i = n; i; --i) sa[tax[rnk[tp[i]]]--] = tp[i];
14     };
15     for (int i = 1; i <= n; ++i) {
16         S[i] = s[i] - '0';
17         tp[i] = i;
18         rnk[i] = S[i];
19     }
20     RadixSort();
21     for (int len = 1, p = 0; p != n; m = p, len <= 1) {
22         p = 0;
23         for (int i = n - len + 1; i <= n; ++i) tp[++p] = i;
24         for (int i = 1; i <= n; ++i) if (sa[i] > len) tp[++p] = sa[i] - len;
25         RadixSort();
26         std::swap(rnk, tp);
27         p = 0;
28         for (int i = 1; i <= n; ++i)
29             rnk[sa[i]] = ((tp[sa[i]] == tp[sa[i-1]]) && (tp[sa[i] + len] == tp[sa[i-1] + len])) ? p : ++p;
30     }
31     for (int i = 1, p = 0; i <= n; ++i) {
32         int pre = sa[rnk[i] - 1];
33         if (p) --p;
34         while (S[pre + p] == S[i + p]) ++p;
35         h[0][rnk[i]] = height[rnk[i]] = p;
36     }
37     for (int i = 1; i <= 20; ++i) {
38         memset(h[i], 0x3f, n * 4 + 4);
39         for (int j = 1; j + (1 << i - 1) <= n; ++j)
40             h[i][j] = min(h[i-1][j], h[i-1][j + (1 << i - 1)]);
41     }
42 }
43 int Q(int l, int r) {
44     if (l > r) swap(l, r);
45     ++l;
46     int k = __lg(r - l + 1);
47     return min(h[k][l], h[k][r - (1 << k) + 1]);

```

```

48 }
49 int lcp(int i, int j) {
50     if (i == j) return n - i + 1;
51     return Q(rnk[i], rnk[j]);
52 }

```

PAM

```

1 struct PAM {
2     static constexpr int ALPHABET_SIZE = 28;
3     struct Node {
4         int len; // 当前节点最长回文长度
5         int fail; // 回文树边
6         int scnt; // 当前节点表示的回文后缀的本质不同回文串个数
7         int pcnt; // 当前节点回文串在字符串中出现次数, 每个点代表一个不同的回文串
8         std::array<int, ALPHABET_SIZE> next;
9         Node() : len{}, fail{}, scnt{}, next{}, pcnt{} {}
10    };
11    std::vector<Node> t;
12    int last;
13    std::string s;
14    PAM() {
15        init();
16    }
17    void init() {
18        t.assign(2, Node());
19        t[1].len = -1;
20        last = 0;
21        t[0].fail = 1;
22        s = "$";
23    }
24    int newNode() {
25        t.emplace_back();
26        return t.size() - 1;
27    }
28    int get_fail(int x) {
29        int pos = s.size() - 1;
30        while(s[pos - t[x].len - 1] != s[pos]) x = t[x].fail;
31        return x;
32    }
33    void add(char c, char offset = 'a') {
34        s += c;
35        int let = c - offset;
36        int x = get_fail(last);
37        if (!t[x].next[let]) {
38            int now = newNode();
39            t[now].len = t[x].len + 2;
40            t[now].fail = t[get_fail(t[x].fail)].next[let];
41            t[x].next[let] = now;
42            t[now].scnt = t[t[now].fail].scnt + 1;
43        }
44        last = t[x].next[let];
45        t[last].pcnt++;
46    }
47 };

```

SAM

```

1 struct SAM {
2     static constexpr int ALPHABET_SIZE = 26, rt = 1;
3     struct Node {
4         int len, fa, siz;
5         std::array<int, ALPHABET_SIZE> nxt;
6         Node() : len{}, fa{}, siz{}, nxt{} {}
7     };
8     std::vector<Node> t;
9     SAM() {
10        init();
11    }
12    void init() {

```

```

13     t.assign(2, Node());
14 }
15 int newNode() {
16     t.emplace_back();
17     return t.size() - 1;
18 }
19 int getfa(int x){
20     return t[x].fa;
21 }
22 int getlen(int x){
23     return t[x].len; //表示该状态能够接受的最长的字符串长度。
24 }
25 int size(){
26     return t.size();
27 }
28 int extend(int p, int ch) {
29     int np = newNode();
30     t[np].len = t[p].len + 1; t[np].siz = 1;
31     while(p && !t[p].nxt[ch]) t[p].nxt[ch] = np, p = t[p].fa;
32     if(!p) { t[np].fa = rt; return np; }
33     int q = t[p].nxt[ch];
34     if(t[q].len == t[p].len + 1) {
35         t[np].fa = q;
36     } else {
37         int nq = newNode(); t[nq].len = t[p].len + 1; t[nq].fa = t[q].fa;
38         for(int i = 0; i < 26; i++) t[nq].nxt[i] = t[q].nxt[i];
39         while(p && t[p].nxt[ch] == q) t[p].nxt[ch] = nq, p = t[p].fa;
40         t[np].fa = t[q].fa = nq;
41     }
42     return np;
43 }
44 int extend_(int p, int ch) { // 叉
45     if(t[p].nxt[ch]) {
46         int q = t[p].nxt[ch];
47         if(t[q].len == t[p].len + 1) return q;
48         int nq = newNode(); t[nq].len = t[p].len + 1; t[nq].fa = t[q].fa;
49         for(int i = 0; i < 26; i++) t[nq].nxt[i] = t[q].nxt[i];
50         while(p && t[p].nxt[ch] == q) t[p].nxt[ch] = nq, p = t[p].fa;
51         t[q].fa = nq; return nq;
52     }
53     int np = newNode();
54     t[np].len = t[p].len + 1;
55     while(p && !t[p].nxt[ch]) t[p].nxt[ch] = np, p = t[p].fa;
56     if(!p) { t[np].fa = rt; return np; }
57     int q = t[p].nxt[ch];
58     if(t[q].len == t[p].len + 1) {
59         t[np].fa = q;
60     } else {
61         int nq = newNode(); t[nq].len = t[p].len + 1; t[nq].fa = t[q].fa;
62         for(int i = 0; i < 26; i++) t[nq].nxt[i] = t[q].nxt[i];
63         while(p && t[p].nxt[ch] == q) t[p].nxt[ch] = nq, p = t[p].fa;
64         t[np].fa = t[q].fa = nq;
65     }
66     return np;
67 }
68 void build(vector<vector<int>> &e){
69     e.resize(t.size());
70     for(int i = 2; i < t.size(); i++){
71         e[t[i].fa].push_back(i);
72     }
73 }
74 };

```

1. 本质不同的子串个数

这个显然就是所有状态所对应的 endpos 集合的大小的和也等价于每个节点的 len 减去 parent 树上的父亲的 len

2. 求两个串的最长公共子串

```

1     int p = 1, len = 0, ans = 0;
2     std::vector<int> l(m), L(m);
3     for(int i = 0; i < m; i++){

```

```

4         int ch = s[i] - 'a';
5         if(sam.t[p].nxt[ch]){
6             p = sam.t[p].nxt[ch]; len ++;
7         }else {
8             while(p && sam.t[p].nxt[ch] == 0){
9                 p = sam.t[p].fa;
10            }
11            if(!p)p = 1, len = 0;
12            else len = sam.t[p].len + 1, p = sam.t[p].nxt[ch];
13        } //其中 p 为前缀最长能匹配到的后缀所在的节点
14        l[i] = len;
15        L[i] = i - len + 1;
16    }

```

parent 树上每个节点维护了一个区间，若 p 是 q 的父节点则有 $\max p = \min q - 1$

每个节点的 endpos 集合为该节点 parent 树上的子树 siz 大小

反串的 SAM 的 parent 树是原串的后缀树

ACAM

```

1  #define ch s[i] - 'a'
2  struct AC_automaton {
3      int nxt[26], Nxt[26], cnt, fail;
4  } T[maxn]; int top = 1, rt = 1, id[maxn];
5  void insert(char *s, int k) {
6      int now = rt, l = strlen(s);
7      for (int i = 0; i < l; ++i) {
8          if (!T[now].nxt[ch]) T[now].nxt[ch] = ++top;
9          now = T[now].nxt[ch];
10     } id[k] = now;
11 }
12
13 void init_fail() { // Trie 图
14     queue<int> Q;
15     for (int i = 0; i < 26; ++i) {
16         int &u = T[rt].nxt[i];
17         if (!u) { u = rt; continue; }
18         T[u].fail = rt; Q.push(u);
19     }
20     while (!Q.empty()) {
21         int u = Q.front(); Q.pop();
22         for (int i = 0; i < 26; ++i) {
23             int &v = T[u].nxt[i];
24             if (!v) { v = T[T[u].fail].nxt[i]; continue; }
25             T[v].fail = T[T[u].fail].nxt[i]; Q.push(v);
26         }
27     }
28 }
29
30 void init_fail() {
31     queue<int> Q;
32     for (int i = 0; i < 26; ++i) {
33         int u = T[rt].nxt[i]; if (!u) { T[rt].Nxt[i] = rt; continue; }
34         T[rt].Nxt[i] = u; T[u].fail = rt; Q.push(u);
35     }
36     while (!Q.empty()) {
37         int u = Q.front(); Q.pop();
38         for (int i = 0; i < 26; ++i) {
39             int v = T[u].nxt[i];
40             if (!v) { T[u].Nxt[i] = T[T[u].fail].Nxt[i]; continue; }
41             T[u].Nxt[i] = v; T[v].fail = T[T[u].fail].Nxt[i]; Q.push(v);
42         }
43     }
44 }

```

KMP

```
1 struct KMP{
2     string s2;// add '#'
3     std::vector<int> nxt;
4     int m;
5     KMP(string y) :s2(y){
6         m = s2.size() - 1;
7         nxt.resize(m + 1,0);
8         for(int i = 2,p = 0;i <= m;i++){
9             while(p && s2[i] != s2[p + 1])p = nxt[p];
10            if(s2[i] == s2[p + 1])p++;
11            nxt[i] = p;
12        }
13    }
14    void match(string s1){
15        int n = s1.size() - 1;
16        for(int i = 1,p = 0;i <= n;i++){
17            while(p && s1[i] != s2[p + 1])p = nxt[p];
18            if(s1[i] == s2[p + 1]){
19                p++;
20                if(p == m){
21                    //cout<<i - m + 1<<endl;
22                    p = nxt[p];
23                }
24            }
25        }
26    }
27    std::vector<int> find_border(){
28        std::vector<int> v;
29        for(int i = nxt[m];i;i = nxt[i])v.push_back(i);
30        return v;
31    }// 找该串所有的周期
32    std::vector<int> calc_prefixes(){
33        std::vector<int> cnt(m + 1,1);
34        for(int i = m;i >= 1;i--)cnt[nxt[i]] += cnt[i];
35        return cnt;
36    }// 每个前缀出现次数
37};
```

Z 函数

对于一个长度为 nn 的字符串 s , 定义函数 $z[i]$ 表示和 $s[i, n - 1]$ (即以 $s[i]$ 开头的后缀) 的最长公共前缀 (LCP) 的长度, 特别地, $z[0] = 0$ 。

```
1 std::vector<int> getZ(const std::string &s) {
2     int n = s.size();
3     std::vector<int> Z(n);
4     Z[0] = n;
5     for (int i = 1, l = 0, r = 0; i < n; ++i) {
6         if (i <= r && Z[i - l] < r - i + 1) {
7             Z[i] = Z[i - l];
8         } else {
9             Z[i] = std::max(0, r - i + 1);
10            while (i + Z[i] < n && s[Z[i]] == s[i + Z[i]]) ++Z[i];
11        }
12        if (i + Z[i] - 1 > r) r = i + Z[l = i] - 1;
13    }
14    return Z;
15 }
16
17 std::vector<int> match(const std::string &s, const std::string &t) {
18     auto Z = getZ(t);
19     int n = s.size(), m = t.size();
20     std::vector<int> ret(n);
21     while (ret[0] < n && ret[0] < m && s[ret[0]] == t[ret[0]]) ++ret[0];
22     for (int l = 0, r = ret[0] - 1, i = 1; i < n; ++i) {
23         if (i <= r && Z[i - l] < r - i + 1) {
24             ret[i] = Z[i - l];
25         } else {
26             ret[i] = std::max(0, r - i + 1);
```

```

27     while (i + ret[i] < n && s[i + ret[i]] == t[ret[i]]) ++ret[i];
28 }
29 if (i + ret[i] - 1 > r) r = i + ret[l = i] - 1;
30 }
31 return ret;
32 }

```

LCP

```

1 for(int i = n; i >= 1; i --) {
2     for(int j = n; j >= 1; j --) {
3         if(s[i] == s[j]) {
4             f[i][j] = f[i + 1][j + 1] + 1; // i-n 和 j-n 的 lcp
5         }
6     }
7 }

```

Hash

```

1 struct Hash {
2     string s;
3     using ull = unsigned long long;
4     ull P1 = 998255347;
5     ull P2 = 1018253347;
6     ull base = 131;
7     vector<ull> hs1, hs2;
8     vector<ull> ps1, ps2;
9     Hash(string s): s(s) {
10         int n = s.size();
11         hs1.resize(n);
12         hs2.resize(n);
13         ps1.resize(n);
14         ps2.resize(n);
15         ps1[0] = ps2[0] = 1;
16         hs1[0] = hs2[0] = (s[0] - 'a');
17         for(int i = 1; i < n; i++) {
18             hs1[i] = hs1[i - 1] * base % P1 + (s[i] - 'a');
19             hs2[i] = hs2[i - 1] * base % P2 + (s[i] - 'a');
20             ps1[i] = (ps1[i - 1] * base) % P1;
21             ps2[i] = (ps2[i - 1] * base) % P2;
22         }
23     }
24     pair<ull, ull> query(int l, int r) {
25         ull res1 = (hs1[r] - (l == 0 ? 0 : hs1[l - 1]) * ps1[r - l + 1] % P1 + P1) % P1;
26         ull res2 = (hs2[r] - (l == 0 ? 0 : hs2[l - 1]) * ps2[r - l + 1] % P2 + P2) % P2;
27         return {res1, res2};
28     } // [l, r]
29 };

```

计算几何

tips:

直线上两点整点坐标范围在 $[-10^6, 10^6]$, 直线交点范围在 $[-10^{18}, 10^{18}]$

Pick 定理: 给定顶点均为整点的简单多边形, 其面积 A 和内部格点数目 i 、边上格点数目 b 的关系为 $A = i + \frac{b}{2} - 1$

曼哈顿转切比雪夫: (x, y) 变为 $(\frac{x+y}{2}, \frac{x-y}{2})$

二维计算几何

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 constexpr double eps = 1e-7;
5 constexpr double PI = acos(-1);
6 constexpr double inf = 1e9;
7 struct Point { double x, y; }; // 点

```



```

8 using Vec = Point; // 向量
9 struct Line { Point P; Vec v; }; // 直线 (点向式), 射线时为 A->B
10 struct Seg { Point A, B; }; // 线段 (存两个端点)
11 struct Circle { Point O; double r; }; // 圆 (存圆心和半径)
12 using Points = std::vector<Point>;
13 using ConvexHull = std::vector<Point>;
14 const Point O = {0, 0}; // 原点
15 const Line Ox = {O, {1, 0}}, Oy = {O, {0, 1}}; // 坐标轴
16
17 bool eq(double a, double b) { return abs(a - b) < eps; } // ==
18 bool gt(double a, double b) { return a - b > eps; } // >
19 bool lt(double a, double b) { return a - b < -eps; } // <
20 bool ge(double a, double b) { return a - b > -eps; } // >=
21 bool le(double a, double b) { return a - b < eps; } // <=
22 Vec operator + (const Vec &a, const Vec &b) { return (Vec){a.x + b.x, a.y + b.y}; }
23 Vec operator - (const Vec &a, const Vec &b) { return (Vec){a.x - b.x, a.y - b.y}; }
24 Vec operator * (const Vec &a, const double &b) { return (Vec){b * a.x, b * a.y}; }
25 Vec operator * (const double &a, const Vec &b) { return (Vec){a * b.x, a * b.y}; }
26 Vec operator / (const Vec &a, const double &b) { return (Vec){a.x / b, a.y / b}; }
27 double operator * (const Point &a, const Point &b) { return a.x * b.x + a.y * b.y; } // dot // 点乘
28 double operator ^ (const Point &a, const Point &b) { return a.x * b.y - a.y * b.x; } // cross // 叉乘
29 bool operator < (const Point& a, const Point& b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
30 double len(const Vec &a) { return sqrt(a * a); }
31
32 ll cross(Point a, Point b) { return (ll)a.x * (ll)b.y - (ll)a.y * (ll)b.x; }
33 ll dot(Point a, Point b) { return (ll)a.x * (ll)b.x + (ll)a.y * (ll)b.y; }
34
35 double angle(const Vec &a, const Vec &b) { return acos(a * b / len(a) / len(b)); }
36
37 double Polar_angle(Vec &v) { return atan2(v.y, v.x); }
38
39 int sgn(double x) {
40     if(fabs(x) < eps)
41         return 0;
42     if(x < 0)
43         return -1;
44     return 1;
45 }
46
47 Vec r90a(Vec v) { return {-v.y, v.x}; } // 逆时针旋转 90 度的向量
48 Vec r90c(Vec v) { return {v.y, -v.x}; } // 顺时针旋转 90 度的向量
49
50 // 两向量的夹角余弦
51 // DEPENDS len, V*V
52 double cos_t(Vec u, Vec v) { return u * v / len(u) / len(v); }
53
54 // 归一化向量 (与原向量方向相同的单位向量)
55 // DEPENDS len
56 Vec norm(Vec v) { return {v.x / len(v), v.y / len(v)}; }
57
58 // 与原向量平行且横坐标大于等于 0 的单位向量
59 // DEPENDS d*V, len
60 Vec pnorm(Vec v) { return (v.x < 0 ? -1 : 1) / len(v) * v; }
61
62 // 线段的方向向量
63 // DEPENDS V-V
64 // NOTE 直线的方向向量直接访问属性 v
65 Vec dvec(Seg l) { return l.B - l.A; }
66 //-----//
67 Line line(Point A, Point B) { return {A, B - A}; }
68
69 // 斜截式直线
70 Line line(double k, double b) { return {{0, b}, {1, k}}; }
71
72 // 点斜式直线
73 Line line(Point P, double k) { return {P, {1, k}}; }
74
75 // 线段所在直线
76 // DEPENDS V-V
77 Line line(Seg l) { return {l.A, l.B - l.A}; }
78

```

```

79 // 给定直线的横坐标求纵坐标
80 // NOTE 请确保直线不与 y 轴平行
81 double at_x(Line l, double x) { return l.P.y + (x - l.P.x) * l.v.y / l.v.x; }
82
83 // 给定直线的纵坐标求横坐标
84 // NOTE 请确保直线不与 x 轴平行
85 double at_y(Line l, double y) { return l.P.x - (y - l.P.y) * l.v.x / l.v.y; }
86
87 // 点到直线的垂足
88 // DEPENDS V-V, V*V, d*V
89 Point pedal(Point P, Line l) { return l.P - (l.P - P) * l.v / (l.v * l.v) * l.v; }
90
91 // 过某点作直线的垂线
92 // DEPENDS r90c
93 Line perp(Line l, Point P) { return {P, r90c(l.v)}; }
94
95 // 角平分线
96 // DEPENDS V+V, len, norm
97 Line bisec(Point P, Vec u, Vec v) { return {P, norm(u) + norm(v)}; }
98
99 //seg-----//
100
101 // 线段的方向向量
102 // DEPENDS V-V
103 // NOTE 直线的方向向量直接访问属性 v
104 //Vec dvec(Seg l) { return l.B - l.A; }
105
106 // 线段中点
107 Point midp(Seg l) { return {(l.A.x + l.B.x) / 2, (l.A.y + l.B.y) / 2}; }
108
109 // 线段中垂线
110 // DEPENDS r90c, V-V, midp
111 Line perp(Seg l) { return {midp(l), r90c(l.B - l.A)}; }
112 //-----//
113 // 向量是否互相垂直
114 // DEPENDS eq, V*V
115 bool verti(Vec u, Vec v) { return eq(u * v, 0); }
116
117 // 向量是否互相平行
118 // DEPENDS eq, V^V
119 bool paral(Vec u, Vec v) { return eq(u ^ v, 0); }
120
121 // 向量是否与 x 轴平行
122 // DEPENDS eq
123 bool paral_x(Vec v) { return eq(v.y, 0); }
124
125 // 向量是否与 y 轴平行
126 // DEPENDS eq
127 bool paral_y(Vec v) { return eq(v.x, 0); }
128
129 // 点是否在直线上
130 // DEPENDS eq
131 bool on(Point P, Line l) { return eq((P.x - l.P.x) * l.v.y, (P.y - l.P.y) * l.v.x); }
132
133
134 // 点是否在射线上
135 // DEPENDS eq
136 bool on_ray(Point P, Line l) { return on(P,l) && ((P - l.P) * l.v) >= 0; }
137
138 // 点是否在线段上
139 // DEPENDS eq, len, V-V
140 bool on(Point P, Seg l) { return eq(len(P - l.A) + len(P - l.B), len(l.A - l.B)); }
141
142 // 两个点是否重合
143 // DEPENDS eq
144 bool operator==(Point A, Point B) { return eq(A.x, B.x) && eq(A.y, B.y); }
145
146 // 两条直线是否重合
147 // DEPENDS eq, on(L)
148 bool operator==(Line a, Line b) { return on(a.P, b) && on(a.P + a.v, b); }
149

```

```

150 // 两条线段是否重合
151 // DEPENDS eq, P==P
152 bool operator==(Seg a, Seg b) { return (a.A == b.A && a.B == b.B) || (a.A == b.B && a.B == b.A); }
153
154 // 以横坐标为第一关键词、纵坐标为第二关键词比较两个点
155 // DEPENDS eq, lt
156 bool operator<(Point A, Point B) { return lt(A.x, B.x) || (eq(A.x, B.x) && lt(A.y, B.y)); }
157
158 // 直线与圆是否相切
159 // DEPENDS eq, V^V, len
160 bool tangency(Line l, Circle C) { return eq(abs((C.O ^ l.v) - (l.P ^ l.v)), C.r * len(l.v)); }
161
162 // 圆与圆是否相切
163 // DEPENDS eq, V-V, len
164 bool tangency(Circle C1, Circle C2) { return eq(len(C1.O - C2.O), C1.r + C2.r); }
165 //-----//
166 // 两点间的距离
167 // DEPENDS len, V-V
168 double dis(Point A, Point B) { return len(A - B); }
169
170 // 点到直线的距离
171 // DEPENDS V^V, len
172 double dis(Point P, Line l) { return abs((P ^ l.v) - (l.P ^ l.v)) / len(l.v); }
173
174 // 点到线段的距离
175 double dis(Point P, Seg l) {
176     if(((P - l.A) * (l.B - l.A)) < 0 || ((P - l.B) * (l.A - l.B)) < 0){
177         return min(dis(P, l.A), dis(P, l.B));
178     }else {
179         Line ll = line(l);
180         return dis(P, l);
181     }
182 }
183 // 平行直线间的距离
184 // DEPENDS d*V, V^V, len, pnorm
185 // NOTE 请确保两直线是平行的
186 double dis(Line a, Line b) { return abs((a.P ^ pnorm(a.v)) - (b.P ^ pnorm(b.v))); }
187
188 // 平移
189 // DEPENDS V+V
190 Line operator+(Line l, Vec v) { return {l.P + v, l.v}; }
191 Seg operator+(Seg l, Vec v) { return {l.A + v, l.B + v}; }
192
193 // 旋转 逆时针
194 // DEPENDS V+V, V-V
195 Point rotate(Point P, double rad) { return {cos(rad) * P.x - sin(rad) * P.y, sin(rad) * P.x + cos(rad) * P.y}; }
196 Point rotate(Point P, double rad, Point C) { return C + rotate(P - C, rad); } // DEPENDS ^1
197 Line rotate(Line l, double rad, Point C = 0) { return {rotate(l.P, rad, C), rotate(l.v, rad)}; } // DEPENDS ^1, ^2
198 Seg rotate(Seg l, double rad, Point C = 0) { return {rotate(l.A, rad, C), rotate(l.B, rad, C)}; } // DEPENDS ^1, ^2
199
200 // 直线与直线交点
201 // DEPENDS eq, d*V, V*V, V+V, V^V
202 Points inter(Line a, Line b){
203     double c = a.v ^ b.v;
204     if (eq(c, 0)) {return {}};
205     Vec v = 1 / c * Vec{a.P ^ (a.P + a.v), b.P ^ (b.P + b.v)};
206     return {{v * Vec{-b.v.x, a.v.x}, v * Vec{-b.v.y, a.v.y}}};
207 }
208
209 // 线段与线段是否相交
210 bool cross_seg(Seg A, Seg B){
211     Point a = A.A, b = A.B, c = B.A, d = B.B;
212     double c1 = (b - a) ^ (c - a), c2 = (b - a) ^ (d - a);
213     double d1 = (d - c) ^ (a - c), d2 = (d - c) ^ (b - c);
214     return sgn(c1) * sgn(c2) < 0 && sgn(d1) * sgn(d2) < 0;
215 }
216
217 // 直线与线段相交 => 直线与直线相交 + 点是否在线段上
218 // bool cross_line_seg(Line A, Seg B){

```

```

221 // Line BB = {B.A,B.B};
222 // Points tmp = inter(A,BB);
223 // if(tmp.size() == 0)return false;
224 // return on(tmp[0],B);
225 // }
226 bool cross_line_seg(Line A,Seg B){
227     if(fabs(A.v ^ (B.A - B.B)) < eps)return false;// 平行
228     Vec v1 = B.A - A.P,v2 = B.B - A.P;
229     if((v2 ^ v1) < 0){
230         swap(v1,v2);
231     }else if(fabs(v2 ^ v1) < eps){
232         if((v1 * v2) <= 0)return true;
233         else return false;
234     }// 保证 v2 在 v1 下面
235     int d1 = sgn(A.v ^ v1);
236     int d2 = sgn(A.v ^ v2);
237     if(d1 * d2 <= 0)return true;
238     return false;
239 }
240
241 // 射线与射线交
242 bool cross_ray_ray(Line A,Line B){
243     Points tmp = inter(A,B);
244     if(tmp.size() == 0)return false;//注意重合
245     int d1 = sgn((tmp[0] - A.P) * A.v);
246     int d2 = sgn((tmp[0] - B.P) * B.v);
247     return d1 >= 0 && d2 >= 0;
248 }
249
250 // 射线与线段交
251 // bool cross_ray_seg(Line A,Seg B){
252 //     Line BB = {B.A,B.B};
253 //     Points tmp = inter(A,BB);
254 //     if(tmp.size() == 0)return false;//注意重合
255 //     int d = sgn((tmp[0] - A.P) * A.v);
256 //     return on(tmp[0],B) && d >= 0;
257 // }
258 bool cross_ray_seg(Line A,Seg B){
259     if(fabs(A.v ^ (B.A - B.B)) < eps)return false;// 平行
260     Vec v1 = B.A - A.P,v2 = B.B - A.P;
261     if((v2 ^ v1) < 0){
262         swap(v1,v2);
263     }else if(fabs(v2 ^ v1) < eps){
264         if((v1 * v2) <= 0)return true;
265         else return false;
266     }// 保证 v2 在 v1 下面
267     int d1 = sgn(A.v ^ v1);
268     int d2 = sgn(A.v ^ v2);
269     if(d1 >= 0 && d2 <= 0)return true;
270     return false;
271 }
272
273 // 射线与直线交
274 bool cross_ray_line(Line A,Line B){ // A 为射线
275     Points tmp = inter(A,B);
276     if(tmp.size() == 0)return false;
277     int d = sgn((tmp[0] - A.P) * A.v);
278     return d >= 0;
279 }
280
281 // 直线与圆交点
282 // DEPENDS eq, gt, V+V, V-V, V*V, d*V, len, pedal
283 std::vector<Point> inter(Line l, Circle C){
284     Point P = pedal(C.O, l);
285     double h = len(P - C.O);
286     if (gt(h, C.r)) return {};
287     if (eq(h, C.r)) return {P};
288     double d = sqrt(C.r * C.r - h * h);
289     Vec vec = d / len(l.v) * l.v;
290     return {P + vec, P - vec};
291 }

```

```

292 // 圆与圆的交点
293 // DEPENDS eq, gt, V+V, V-V, d*V, len, r90c
294 std::vector<Point> inter(Circle C1, Circle C2){
295     Vec v1 = C2.O - C1.O, v2 = r90c(v1);
296     double d = len(v1);
297     if (gt(d, C1.r + C2.r) || gt(abs(C1.r - C2.r), d)) return {};
298     if (eq(d, C1.r + C2.r) || eq(d, abs(C1.r - C2.r))) return {C1.O + C1.r / d * v1};
299     double a = ((C1.r * C1.r - C2.r * C2.r) / d + d) / 2;
300     double h = sqrt(C1.r * C1.r - a * a);
301     Vec av = a / len(v1) * v1, hv = h / len(v2) * v2;
302     return {C1.O + av + hv, C1.O + av - hv};
303 }
304
305 // 三角形的重心
306 Point barycenter(Point A, Point B, Point C){
307     return {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};
308 }
309
310 // 三角形的外心
311 // DEPENDS r90c, V*V, d*V, V-V, V+V
312 // NOTE 给定圆上三点求圆, 要先判断是否三点共线
313 Point circumcenter(Point A, Point B, Point C){
314     double a = A * A, b = B * B, c = C * C;
315     double d = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
316     return 1 / d * r90c(a * (B - C) + b * (C - A) + c * (A - B));
317 }
318
319 // 三角形的内心
320 // DEPENDS len, d*V, V-V, V+V
321 Point incenter(Point A, Point B, Point C){
322     double a = len(B - C), b = len(A - C), c = len(A - B);
323     double d = a + b + c;
324     return 1 / d * (a * A + b * B + c * C);
325 }
326
327 // 三角形的垂心
328 // DEPENDS V*V, d*V, V-V, V^V, r90c
329 Point orthocenter(Point A, Point B, Point C){
330     double n = B * (A - C), m = A * (B - C);
331     double d = (B - C) ^ (A - C);
332     return 1 / d * r90c(n * (C - B) - m * (C - A));
333 }
334
335 // Graham 扫描法
336 // DEPENDS eq, lt, cross, V-V, P<P
337
338 double theta(Point p) { return p == 0 ? -1 / 0. : atan2(p.y, p.x); } // 求极角
339 void psort(Points &ps, Point c = 0) { // 极角排序
340     sort(ps.begin(), ps.end(), [&](auto a, auto b) {
341         return lt(theta(a - c), theta(b - c));
342     });
343 }
344
345 //极角排序
346 int qua(const Point &P){
347     if(P.x == 0 && P.y == 0) return 0;
348     if(P.x >= 0 && P.y >= 0) return 1;
349     if(P.x < 0 && P.y >= 0) return 2;
350     if(P.x < 0 && P.y < 0) return 3;
351     if(P.x >= 0 && P.y < 0) return 4;
352     exit(-1);
353 }
354
355 void psort(Points &ps, Point c = 0) { // 极角排序
356     stable_sort(ps.begin(), ps.end(), [&](auto p1, auto p2) {
357         return qua(p1 - c) < qua(p2 - c) || qua(p1 - c) == qua(p2 - c) && gt((Point)(p1 - c) ^ (Point)(p2 - c), 0);
358     });
359 }

```

```

363 }
364
365 // 检查向量夹角 acb 小于 180
366 bool check1(Point a, Point b, Point c){//
367     ll d = (a - c) ^ (b - c);
368     if(d > 0) return true;
369     if(d < 0) return false;
370     return (a - c) * (b - c) > 0;
371 }
372
373
374
375
376 bool check(Point p, Point q, Point r){ // 检查三个点组成的两个向量的旋转方向是否为逆时针
377     return lt(0, (q - p) ^ (r - q));
378 }
379 ConvexHull Andrew(Points &ps){
380     if(ps.size() == 1){
381         return ps;
382     }
383     sort(ps.begin(), ps.end());
384     std::vector<int> I{0}, used(ps.size());
385     for (int i = 1; i < ps.size(); i++){
386         //std::cout << ps[i].x << " " << ps[i].y << "\n";
387         while (I.size() > 1 && !check(ps[I[I.size() - 2]], ps[I.back()], ps[i]))
388             used[I.back()] = 0, I.pop_back();
389         used[i] = 1, I.push_back(i);
390     }
391     int tmp = I.size();
392     for (int i = ps.size() - 2; i >= 0; i--){
393         if (used[i])
394             continue;
395         while (I.size() > tmp && !check(ps[I[I.size() - 2]], ps[I.back()], ps[i]))
396             used[I.back()] = 0, I.pop_back();
397         used[i] = 1, I.push_back(i);
398     }
399     Points H;
400     for (int i = 0; i < I.size() - 1; i++)
401         H.push_back(ps[I[i]]);
402     return H;
403 }//逆时针
404 ConvexHull chull(Points &ps){
405     psort(ps, *min_element(ps.begin(), ps.end())); // 以最左下角的点为极角排序
406     Points H{ps[0]};
407     for (int i = 1; i < ps.size(); i++){
408         while (H.size() > 1 && !check(H[H.size() - 2], H.back(), ps[i]))
409             H.pop_back();
410         H.push_back(ps[i]);
411     }
412     return H;
413 }
414 ConvexHull operator+(const ConvexHull &A, const ConvexHull B){
415     int n = A.size();
416     int m = B.size();
417     std::vector<Point> v1(n), v2(m);
418     for(int i = 0; i < n; i++){
419         v1[i] = A[(i + 1) % n] - A[i];
420     }
421     for(int i = 0; i < m; i++){
422         v2[i] = B[(i + 1) % m] - B[i];
423     }
424     ConvexHull C;
425     C.push_back(A[0] + B[0]);
426     int p1 = 0, p2 = 0;
427     while(p1 < n && p2 < m){
428         C.push_back(C.back() + ((v1[p1] ^ v2[p2]) >= 0 ? v1[p1++] : v2[p2++]));
429     }// 对上凸壳做闵可夫斯基和时将 >= 改为 <= 并且合并凸包时不需要排序
430     while(p1 < n) C.push_back(C.back() + v1[p1++]);
431     while(p2 < m) C.push_back(C.back() + v2[p2++]);
432     C = chull(C);
433     return C;

```

```

434 }
435 void test(Points a, Point b){
436     int n = a.size();
437     int r = 0;
438     for(int l = 0; l < n; l++){
439         auto nxt = [&](int x){
440             return (x + 1) % n;
441         };
442         while(nxt(r) != l && check(a[l], a[nxt(r)], b)){
443             // b 为轴点
444             r = nxt(r);
445
446             if(l == r) break;
447         }
448     } //极角排序 转半平面
449 }
450
451 // 半平面交
452 int sgn(Point a) {
453     return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
454 }
455 bool pointOnLineLeft(Point p, Line l) {
456     return (l.v ^ (p - l.P)) > eps;
457 }
458 Point lineIntersection(Line l1, Line l2) {
459     return l1.P + l1.v * (cross(l2.v, l1.P - l2.P) / cross(l2.v, 0 - l1.v));
460 }
461 std::vector<Point> hp(std::vector<Line> lines) {
462     std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
463         auto d1 = l1.v;
464         auto d2 = l2.v;
465
466         if (sgn(d1) != sgn(d2)) {
467             return sgn(d1) == 1;
468         }
469
470         return cross(d1, d2) > 0;
471     });
472     std::deque<Line> ls;
473     std::deque<Point> ps;
474     for (auto l : lines) {
475         if (ls.empty()) {
476             ls.push_back(l);
477             continue;
478         }
479         while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
480             ps.pop_back(); ls.pop_back();
481         }
482
483         while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
484             ps.pop_front(); ls.pop_front();
485         }
486         if (fabs(cross(l.v, ls.back().v)) < eps) {
487
488             if ((l.v * ls.back().v) > eps) {
489                 //continue;
490                 if (!pointOnLineLeft(ls.back().P, l)) {
491                     assert(ls.size() == 1);
492                     ls[0] = l;
493                 }
494                 continue;
495             }
496             return {};
497         }
498         auto now = inter(ls.back(), l);
499         ps.push_back(now[0]);
500         // ps.push_back(lineIntersection(ls.back(), l));
501         ls.push_back(l);
502     }
503
504     while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {

```

```

505     ps.pop_back();ls.pop_back();
506 }
507 if (ls.size() <= 2) {
508     return {};
509 }
510 auto now = inter(ls[0],ls.back());
511 ps.push_back(now[0]);
512 // ps.push_back(lineIntersection(ls[0], ls.back()));
513 return std::vector(ps.begin(), ps.end());
514 }
515
516 // int sta[N], top; // 将凸包上的节点编号存在栈里, 第一个和最后一个节点编号相同
517 // bool is[N];
518
519 // ll pf(ll x) { return x * x; }
520
521 // ll dis(int p, int q) { return pf(a[p].x - a[q].x) + pf(a[p].y - a[q].y); }
522
523 // ll sqr(int p, int q, int y) { return abs((a[q] - a[p]) * (a[y] - a[q])); }
524
525 // ll mx;
526
527 // void get_longest() { // 求凸包直径
528 //     int j = 3;
529 //     if (top < 4) {
530 //         mx = dis(sta[1], sta[2]);
531 //         return;
532 //     }
533 //     for (int i = 1; i <= top; ++i) {
534 //         while (sqr(sta[i], sta[i + 1], sta[j]) <=
535 //             sqr(sta[i], sta[i + 1], sta[j % top + 1]))
536 //             j = j % top + 1;
537 //         mx = max(mx, max(dis(sta[i + 1], sta[j]), dis(sta[i], sta[j]))));
538 //     }
539 // }
540
541

```

动态凸包

```

1  struct Item {
2      P p;
3      mutable P vec;
4      int q = 0;
5  };
6
7  bool operator<(const Item &a, const Item &b) {
8      if (!b.q) {
9          return a.p.x < b.p.x;
10     }
11     return dot(a.vec, b.p) > 0;
12 }
13
14 struct Hull {
15     std::set<Item> s;
16     i128 dx = 0;
17     i128 dy = 0;
18 };
19
20 void print(const Hull &h) {
21     for (auto it : h.s) {
22         std::cerr << "(" << i64(it.p.x + h.dx) << ", " << i64(it.p.y + h.dy) << ") ";
23     }
24     std::cerr << "\n";
25 }
26
27 constexpr i64 inf = 2E18;
28
29 void insert(Hull &h, P p) {
30     p.x -= h.dx;
31     p.y -= h.dy;

```



```

32     h.s.insert({p});
33     auto it = h.s.lower_bound({p});
34     if (it != h.s.end() && it->p.x == p.x) {
35         if (it->p.y > p.y) {
36             return;
37         }
38         it = h.s.erase(it);
39     }
40     if (it != h.s.begin() && it != h.s.end()
41         && cross(p - std::prev(it)->p, it->p - p) >= 0) {
42         return;
43     }
44     it = h.s.insert({p}).first;
45     auto r = std::next(it);
46     if (r != h.s.end()) {
47         while (cross(r->p - p, r->vec) >= 0) {
48             r = h.s.erase(r);
49         }
50         it->vec = r->p - p;
51     } else {
52         it->vec = P(0, -inf);
53     }
54
55     if (it != h.s.begin()) {
56         auto l = std::prev(it);
57         while (l != h.s.begin()) {
58             auto a = std::prev(l);
59             if (cross(a->vec, p - l->p) < 0) {
60                 break;
61             }
62             h.s.erase(l);
63             l = a;
64         }
65         l->vec = p - l->p;
66     }
67 }
68
69 i64 query(const Hull &h, i64 x) {
70     if (h.s.empty()) {
71         return 0LL;
72     }
73     auto it = h.s.lower_bound({P(x, 1), P{}, 1});
74     assert(it != h.s.end());
75     auto p = it->p;
76     p.x += h.dx;
77     p.y += h.dy;
78     return p.x * x + p.y;
79 }

```

最小圆覆盖

```

1  int n;
2  double r;
3
4  struct point {
5      double x, y;
6  } p[100005], o;
7
8  double sqr(double x) { return x * x; }
9
10 double dis(point a, point b) { return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)); }
11
12 bool cmp(double a, double b) { return fabs(a - b) < 1e-8; }
13
14 point geto(point a, point b, point c) {
15     double a1, a2, b1, b2, c1, c2;
16     point ans;
17     a1 = 2 * (b.x - a.x), b1 = 2 * (b.y - a.y),
18     c1 = sqr(b.x) - sqr(a.x) + sqr(b.y) - sqr(a.y);
19     a2 = 2 * (c.x - a.x), b2 = 2 * (c.y - a.y),
20     c2 = sqr(c.x) - sqr(a.x) + sqr(c.y) - sqr(a.y);

```

```

21     if (cmp(a1, 0)) {
22         ans.y = c1 / b1;
23         ans.x = (c2 - ans.y * b2) / a2;
24     } else if (cmp(b1, 0)) {
25         ans.x = c1 / a1;
26         ans.y = (c2 - ans.x * a2) / b2;
27     } else {
28         ans.x = (c2 * b1 - c1 * b2) / (a2 * b1 - a1 * b2);
29         ans.y = (c2 * a1 - c1 * a2) / (b2 * a1 - b1 * a2);
30     }
31     return ans;
32 }
33
34 int main() {
35     scanf("%d", &n);
36     for (int i = 1; i <= n; i++) scanf("%lf%lf", &p[i].x, &p[i].y);
37     for (int i = 1; i <= n; i++) swap(p[rand() % n + 1], p[rand() % n + 1]);
38     o = p[1];
39     for (int i = 1; i <= n; i++) {
40         if (dis(o, p[i]) < r || cmp(dis(o, p[i]), r)) continue;
41         o.x = (p[i].x + p[1].x) / 2;
42         o.y = (p[i].y + p[1].y) / 2;
43         r = dis(p[i], p[1]) / 2;
44         for (int j = 2; j < i; j++) {
45             if (dis(o, p[j]) < r || cmp(dis(o, p[j]), r)) continue;
46             o.x = (p[i].x + p[j].x) / 2;
47             o.y = (p[i].y + p[j].y) / 2;
48             r = dis(p[i], p[j]) / 2;
49             for (int k = 1; k < j; k++) {
50                 if (dis(o, p[k]) < r || cmp(dis(o, p[k]), r)) continue;
51                 o = geto(p[i], p[j], p[k]);
52                 r = dis(o, p[i]);
53             }
54         }
55     }
56     printf("%.10lf\n%.10lf %.10lf", r, o.x, o.y);
57     return 0;
58 }

```

杂项

大质数和原根

$$p = r \times 2^k + 1$$

prime	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
1004535809	479	21	3

2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

约瑟夫问题

```

1 //约瑟夫问题
2 int josephus(int n, int k) {
3     int res = 0;
4     for (int i = 1; i <= n; ++i) res = (res + k) % i;
5     return res;
6 }
7 int josephus(int n, int k) {
8     if (n == 1) return 0;
9     if (k == 1) return n - 1;
10    if (k > n) return (josephus(n - 1, k) + k) % n; // 线性算法
11    int res = josephus(n - n / k, k);
12    res -= n % k;
13    if (res < 0)
14        res += n; // mod n
15    else
16        res += res / (k - 1); // 还原位置
17    return res;
18 }

```

辛普森积分

```

1 const int N = 1000 * 1000;
2
3 double simpson_integration(double a, double b) {
4     double h = (b - a) / N;
5     double s = f(a) + f(b);
6     for (int i = 1; i <= N - 1; ++i) {
7         double x = a + h * i;
8         s += f(x) * ((i & 1) ? 4 : 2);
9     }
10    s *= h / 3;
11    return s;
12 }
13
14
15 //自适应
16 double simpson(double l, double r) {
17     double mid = (l + r) / 2;
18     return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6; // 辛普森公式
19 }
20

```

```

21 double asr(double l, double r, double eps, double ans, int step) {
22     double mid = (l + r) / 2;
23     double fl = simpson(l, mid), fr = simpson(mid, r);
24     if (abs(fl + fr - ans) <= 15 * eps && step < 0)
25         return fl + fr + (fl + fr - ans) / 15; // 足够相似的话就直接返回
26     return asr(l, mid, eps / 2, fl, step - 1) +
27         asr(mid, r, eps / 2, fr, step - 1); // 否则分割成两段递归求解
28 }
29
30 double calc(double l, double r, double eps) {
31     return asr(l, r, eps, simpson(l, r), 12);
32 }

```

unordered_map

```

1 struct custom_hash {
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
10    size_t operator()(uint64_t x) const {
11        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
12        return splitmix64(x + FIXED_RANDOM);
13    }
14 };
15 // pair
16 // 分别计算出内置类型的 Hash Value 然后对它们进行 Combine 得到一个哈希值
17 // 一般直接采用移位加异或 (XOR) 得到哈希值
18 struct HashFunc
19 {
20     template<typename T, typename U>
21     size_t operator()(const std::pair<T, U>& p) const {
22         return std::hash<T>()(p.first) ^ std::hash<U>()(p.second);
23     }
24 };
25
26 // 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
27 struct EqualKey {
28     template<typename T, typename U>
29     bool operator()(const std::pair<T, U>& p1, const std::pair<T, U>& p2) const {
30         return p1.first == p2.first && p1.second == p2.second;
31     }
32 };

```

位运算

1. `int __builtin_ffs(int x)`: 返回 x 的二进制末尾最后一个 1 的位置, 位置的编号从 1 开始 (最低位编号为 1)。当 x 为 0 时返回 0。
2. `int __builtin_clz(unsigned int x)`: 返回 x 的二进制的前导 0 的个数。当 x 为 0 时, 结果未定义。
3. `int __builtin_ctz(unsigned int x)`: 返回 x 的二进制末尾连续 0 的个数。当 x 为 0 时, 结果未定义。
4. `int __builtin_clrsb(int x)`: 当 x 的符号位为 0 时返回 0 的二进制的前导 0 的个数减一, 否则返回 x 的二进制的前导 1 的个数减一。
5. `int __builtin_popcount(unsigned int x)`: 返回 x 的二进制中 1 的个数。
6. `int __builtin_parity(unsigned int x)`: 判断 x 的二进制中的个数的奇偶性。

模 2 的次幂

```
1 int modPowerOfTwo(int x, int mod) { return x & (mod - 1); }
```

2 的次幂判断

```
1 bool isPowerOfTwo(int n) { return n > 0 && (n & (n - 1)) == 0; }
```

子集枚举

```
1 for (int i = 0; i < (1 << n); i++)
2     for (int s = i; s; s = (s - 1) & i)
```

int128 输出

```
1 using i128 = __int128;
2
3 std::ostream &operator<<(std::ostream &os, i128 n) {
4     std::string s;
5     while (n) {
6         s += '0' + n % 10;
7         n /= 10;
8     }
9     std::reverse(s.begin(), s.end());
10    return os << s;
11 }
```

随机生成质数

```
1 bool isprime(int n) {
2     if (n <= 1) {
3         return false;
4     }
5     for (int i = 2; i * i <= n; i++) {
6         if (n % i == 0) {
7             return false;
8         }
9     }
10    return true;
11 }
12
13 int findPrime(int n) {
14     while (!isprime(n)) {
15         n++;
16     }
17     return n;
18 }
19 std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());
20 const int P = findPrime(rng() % 9000000000 + 1000000000);
```

bitset

构造函数

- `bitset()`: 每一位都是 `false`。
- `bitset(unsigned long val)`: 设为 `val` 的二进制形式。
- `bitset(const string& str)`: 设为 01 串 `str`。

运算符

- `operator []`: 访问其特定的一位。
- `operator ==/!=`: 比较两个 `bitset` 内容是否完全一样。
- `operator &/&=||| =/^/^/=/~`: 进行按位与/或/异或/取反操作。**bitset 只能与 bitset 进行位运算**，若要和整型进行位运算，要先将整型转换为 `bitset`。
- `operator <>/<<=/>>=`: 进行二进制左移/右移。
- `operator <>`: 流运算符，这意味着你可以通过 `cin/cout` 进行输入输出。

成员函数

- `count()`: 返回 `true` 的数量。
- `size()`: 返回 `bitset` 的大小。
- `test(pos)`: 它和 `vector` 中的 `at()` 的作用是一样的，和 `[]` 运算符的区别就是越界检查。
- `any()`: 若存在某一位是 `true` 则返回 `true`，否则返回 `false`。

- none(): 若所有位都是 false 则返回 true, 否则返回 false。
- all(): C++11, 若所有位都是 true 则返回 true, 否则返回 false。
- 1. set(): 将整个 bitset 设置成 true。
2. set(pos, val = true): 将某一位设置成 true/false。
- 1. reset(): 将整个 bitset 设置成 false。
2. reset(pos): 将某一位设置成 false。相当于 set(pos, false)。
- 1. flip(): 翻转每一位。0 ↔ 1, 相当于异或一个全是 1 的 bitset)
2. flip(pos): 翻转某一位。
- to_string(): 返回转换成的字符串表达。
- to_ulong(): 返回转换成的 unsigned long 表达 (long 在 NT 及 32 位 POSIX 系统下与 int 一样, 在 64 位 POSIX 下与 long long 一样)。
- to_ullong(): C++11, 返回转换成的 unsigned long long 表达。

一些文档中没有的成员函数:

- _Find_first(): 返回 bitset 第一个 true 的下标, 若没有 true 则返回 bitset 的大小。
- _Find_next(pos): 返回 pos 后面(下标严格大于 pos 的位置)第一个 true 的下标, 若 pos 后面没有 true 则返回 bitset 的大小。

手写 bitset

```

1  #include<vector>
2  using ull = unsigned long long;
3  struct Bit {
4      ull mi[65];
5      // ull bit[15626];
6      std::vector<ull> bit; int len;
7      Bit() {
8          len = 10;
9          bit.resize(len);
10         for(int i = 0; i <= 63; i++) mi[i] = (1ull << i);
11     }
12     Bit(int len) : len(len){
13         bit.resize(len);
14         for(int i = 0; i <= 63; i++) mi[i] = (1ull << i);
15     }
16     void reset() {bit.assign(len,0);}
17     void set1(int x) { bit[x>>6] |= mi[x&63];}
18     void set0(int x) { bit[x>>6] &= ~mi[x&63];}
19     void flip(int x) { bit[x>>6] ^= mi[x&63];}
20     bool operator [](int x) {
21         return (bit[x>>6] >> (x&63)) & 1;
22     }
23     int count() {
24         int s = 0;
25         for(int i = 0; i < len; i++) s += __builtin_popcountll(bit[i]);
26         return s;
27     }
28     Bit operator ~ (void) const {
29         Bit res;
30         for (int i = 0; i < len; i++) res.bit[i] = ~bit[i];
31         return res;
32     }
33
34     Bit operator & (const Bit &b) const {
35         Bit res;
36         for (int i = 0; i < len; i++) res.bit[i] = bit[i] & b.bit[i];
37         return res;
38     }
39
40     Bit operator | (const Bit &b) const {
41         Bit res;
42         for (int i = 0; i < len; i++) res.bit[i] = bit[i] | b.bit[i];
43         return res;

```

```

44     }
45
46     Bit operator ^ (const Bit &b) const {
47         Bit res;
48         for (int i = 0; i < len; i++) res.bit[i] = bit[i] ^ b.bit[i];
49         return res;
50     }
51
52     void operator &= (const Bit &b) {
53         for (int i = 0; i < len; i++) bit[i] &= b.bit[i];
54     }
55
56     void operator |= (const Bit &b) {
57         for (int i = 0; i < len; i++) bit[i] |= b.bit[i];
58     }
59
60     void operator ^= (const Bit &b) {
61         for (int i = 0; i < len; i++) bit[i] ^= b.bit[i];
62     }
63
64     Bit operator << (const int t) const {
65         Bit res; int high = t >> 6, low = t & 63;
66         ull last = 0;
67         for (int i = 0; i + high < len; i++) {
68             res.bit[i + high] = (last | (bit[i] << low));
69             if (low) last = (bit[i] >> (64 - low));
70         }
71         return res;
72     }
73
74     Bit operator >> (const int t) const {
75         Bit res; int high = t >> 6, low = t & 63;
76         ull last = 0;
77         for (int i = len - 1; i >= high; i--) {
78             res.bit[i - high] = last | (bit[i] >> low);
79             if (low) last = bit[i] << (64 - low);
80         }
81         return res;
82     }
83
84     void operator <<= (const int t) {
85         int high = t >> 6, low = t & 63;
86         for (int i = len - high - 1; ~i; i--) {
87             bit[i + high] = (bit[i] << low);
88             if (low && i) bit[i + high] |= bit[i - 1] >> (64 - low);
89         }
90         for (int i = 0; i < high; i++) bit[i] = 0;
91     }
92 };

```

string

转 char 数组

string 有两个成员函数能够将自己转换为 char 指针——data()/c_str() (它们几乎是一样的, 但最好使用 c_str(), 因为 c_str() 保证末尾有空字符, 而 data() 则不保证)

寻找某字符 (串) 第一次出现的位置

find(str, pos) 函数可以用来查找字符串中一个字符/字符串在 pos (含) 之后第一次出现的位置 (若不传参给 pos 则默认为 0)。如果没有出现, 则返回 string::npos (被定义为 -1, 但类型仍为 size_t/unsigned long)。

截取子串

substr(pos, len) 函数的参数返回从 pos 位置开始截取最多 len 个字符组成的字符串 (如果从 pos 开始的后缀长度不足 len 则截取这个后缀)。

插入/删除字符 (串)

`insert(index, count, ch)` 和 `insert(index, str)` 是比较常见的插入函数。它们分别表示在 `index` 处连续插入 `count` 次字符串 `ch` 和插入字符串 `str`。

`erase(index, count)` 函数将字符串 `index` 位置开始（含）的 `count` 个字符删除（若不传参给 `count` 则表示删去 `count` 位置及以后的所有字符）。

替换字符（串）

`replace(pos, count, str)` 和 `replace(first, last, str)` 是比较常见的替换函数。它们分别表示将从 `pos` 位置开始 `count` 个字符的子串替换为 `str` 以及将以 `first` 开始（含）、`last` 结束（不含）的子串替换为 `str`，其中 `first` 和 `last` 均为迭代器。

STL

- `sort`: 排序。 `sort(v.begin(), v.end(), cmp)` 或 `sort(a + begin, a + end, cmp)`，其中 `end` 是排序的数组最后一个元素的后一位，`cmp` 为自定义的比较函数。
- `stable_sort`: 稳定排序，用法同 `sort()`。
- `nth_element`: 按指定范围进行分类，即找出序列中第 n 大的元素，使其左边均为小于它的数，右边均为大于它的数。
`nth_element(v.begin(), v.begin() + mid, v.end(), cmp)` 或 `nth_element(a + begin, a + begin + mid, a + end, cmp)`。
- `binary_search`: 二分查找。 `binary_search(v.begin(), v.end(), value)`，其中 `value` 为需要查找的值。
- `merge`: 将两个（已排序的）序列有序合并到第三个序列的插入迭代器上。 `merge(v1.begin(), v1.end(), v2.begin(), v2.end(), back_inserter(v3))`。
- `inplace_merge`: 将两个（已按小于运算符排序的）: `[first, middle)`, `[middle, last)` 范围原地合并为一个有序序列。 `inplace_merge(v.begin(), v.begin() + middle, v.end())`。
- `lower_bound`: 在一个有序序列中进行二分查找，返回指向第一个大于等于 x 的元素的位置的迭代器。如果不存在这样的元素，则返回尾迭代器。 `lower_bound(v.begin(), v.end(), x)`。
- `upper_bound`: 在一个有序序列中进行二分查找，返回指向第一个大于 x 的元素的位置的迭代器。如果不存在这样的元素，则返回尾迭代器。 `upper_bound(v.begin(), v.end(), x)`。
- `next_permutation`: 将当前排列更改为全排列中的下一个排列。如果当前排列已经是全排列中的最后一个排列（元素完全从大到小排列），函数返回 `false` 并将排列更改为全排列中的第一个排列（元素完全从小到大排列）；否则，函数返回 `true`。
`next_permutation(v.begin(), v.end())` 或 `next_permutation(v + begin, v + end)`。
- `partial_sum`: 求前缀和。设源容器为 x ，目标容器为 y ，则令 $y[i] = x[0] + x[1] + \dots + x[i]$ 。 `partial_sum(src.begin(), src.end(), back_inserter(dst))`。

pb_ds

`__gnu_pbds::tree`

```
1 #include <ext/pb_ds/assoc_container.hpp> // 因为 tree 定义在这里 所以需要包含这个头文件
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 __gnu_pbds::tree<Key, Mapped, Cmp_Fn = std::less<Key>, Tag = rb_tree_tag,
5     Node_Update = null_tree_node_update,
6     Allocator = std::allocator<char> >
```

模板形参

- `Key`: 储存的元素类型，如果想要存储多个相同的 `Key` 元素，则需要使用类似于 `std::pair` 和 `struct` 的方法，并配合使用 `lower_bound` 和 `upper_bound` 成员函数进行查找
- `Mapped`: 映射规则（Mapped-Policy）类型，如果要指示关联容器是集合，类似于存储元素在 `std::set` 中，此处填入 `null_type`，低版本 `g++` 此处为 `null_mapped_type`；如果要指示关联容器是带值的集合，类似于存储元素在 `std::map` 中，此处填入类似于 `std::map<Key, Value>` 的 `Value` 类型
- `Cmp_Fn`: 关键字比较函子，例如 `std::less<Key>`
- `Tag`: 选择使用何种底层数据结构类型，默认是 `rb_tree_tag`。 `__gnu_pbds` 提供不同的三种平衡树，分别是：
 - `rb_tree_tag`: 红黑树，一般使用这个，后两者的性能一般不如红黑树
 - `splay_tree_tag`: splay 树

- `ov_tree_tag`: 有序向量树，只是一个由 `vector` 实现的有序结构，类似于排序的 `vector` 来实现平衡树，性能取决于数据想不想卡你
- `Node_Update`: 用于更新节点的策略，默认使用 `null_node_update`，若要使用 `order_of_key` 和 `find_by_order` 方法，需要使用 `tree_order_statistics_node_update`
- `Allocator`: 空间分配器类型

构造方式

```

1 __gnu_pbds::tree<std::pair<int, int>, __gnu_pbds::null_type,
2     std::less<std::pair<int, int> >, __gnu_pbds::rb_tree_tag,
3     __gnu_pbds::tree_order_statistics_node_update>
4     trr;

```

成员函数

- `insert(x)`: 向树中插入一个元素 `x`，返回 `std::pair<point_iterator, bool>`。
- `erase(x)`: 从树中删除一个元素/迭代器 `x`，返回一个 `bool` 表明是否删除成功。
- `order_of_key(x)`: 返回 `x` 以 `Cmp_Fn` 比较的排名。
- `find_by_order(x)`: 返回 `Cmp_Fn` 比较的排名所对应元素的迭代器。
- `lower_bound(x)`: 以 `Cmp_Fn` 比较做 `lower_bound`，返回迭代器。
- `upper_bound(x)`: 以 `Cmp_Fn` 比较做 `upper_bound`，返回迭代器。
- `join(x)`: 将 `x` 树并入当前树，前提是两棵树的类型一样，`x` 树被删除。
- `split(x,b)`: 以 `Cmp_Fn` 比较，小于等于 `x` 的属于当前树，其余的属于 `b` 树。
- `empty()`: 返回是否为空。
- `size()`: 返回大小。

示例

```

1 // Common Header Simple over C++11
2 #include <bits/stdc++.h>
3 using namespace std;
4 typedef long long ll;
5 typedef unsigned long long ull;
6 typedef long double ld;
7 typedef pair<int, int> pii;
8 #define pb push_back
9 #define mp make_pair
10 #include <ext/pb_ds/assoc_container.hpp>
11 #include <ext/pb_ds/tree_policy.hpp>
12 __gnu_pbds::tree<pair<int, int>, __gnu_pbds::null_type, less<pair<int, int> >,
13     __gnu_pbds::rb_tree_tag,
14     __gnu_pbds::tree_order_statistics_node_update>
15     trr;
16
17 int main() {
18     int cnt = 0;
19     trr.insert(mp(1, cnt++));
20     trr.insert(mp(5, cnt++));
21     trr.insert(mp(4, cnt++));
22     trr.insert(mp(3, cnt++));
23     trr.insert(mp(2, cnt++));
24     // 树上元素 {{1,0},{2,4},{3,3},{4,2},{5,1}}
25     auto it = trr.lower_bound(mp(2, 0));
26     trr.erase(it);
27     // 树上元素 {{1,0},{3,3},{4,2},{5,1}}
28     auto it2 = trr.find_by_order(1);
29     cout << (*it2).first << endl;
30     // 输出排名 0 1 2 3 中的排名 1 的元素的 first:1
31     int pos = trr.order_of_key(*it2);
32     cout << pos << endl;
33     // 输出排名
34     decltype(trr) newtr;
35     trr.split(*it2, newtr);
36     for (auto i = newtr.begin(); i != newtr.end(); ++i) {
37         cout << (*i).first << ' ';
38     }

```

```

39     cout << endl;
40     // {4,2},{5,1} 被放入新树
41     trr.join(newtr);
42     for (auto i = trr.begin(); i != trr.end(); ++i) {
43         cout << (*i).first << ' ';
44     }
45     cout << endl;
46     cout << newtr.size() << endl;
47     // 将 newtr 树并入 trr 树, newtr 树被删除。
48     return 0;
49 }

```

__gnu_pbds::priority_queue

```

1  #include <ext/pb_ds/priority_queue.hpp>
2  using namespace __gnu_pbds;
3  __gnu_pbds::priority_queue<T, Compare, Tag, Allocator>

```

模板形参

- T: 储存的元素类型
- Compare: 提供严格的弱序比较类型
- Tag: 是 __gnu_pbds 提供的不同的五种堆, Tag 参数默认是 pairing_heap_tag 五种分别是:
 - pairing_heap_tag: 配对堆官方文档认为在非原生元素 (如自定义结构体/std::string/pair) 中, 配对堆表现最好
 - binary_heap_tag: 二叉堆官方文档认为在原生元素中二叉堆表现最好, 不过我测试的表现并没有那么好
 - binomial_heap_tag: 二项堆二项堆在合并操作的表现要优于二叉堆, 但是其取堆顶元素操作的复杂度比二叉堆高
 - rc_binomial_heap_tag: 冗余计数二项堆
 - thin_heap_tag: 除了合并的复杂度都和 Fibonacci 堆一样的一个 tag
- Allocator: 空间配置器, 由于 OI 中很少出现, 故这里不做讲解

构造方式

要注明命名空间因为和 std 的类名称重复。

```

__gnu_pbds::priority_queue<int> __gnu_pbds::priority_queue<int, greater<int> >
__gnu_pbds::priority_queue<int, greater<int>, pairing_heap_tag>
__gnu_pbds::priority_queue<int>::point_iterator id; // 点类型迭代器
// 在 modify 和 push 的时候都会返回一个 point_iterator, 下文会详细的讲使用方法
id = q.push(1);

```

成员函数

- push(): 向堆中压入一个元素, 返回该元素位置的迭代器。
- pop(): 将堆顶元素弹出。
- top(): 返回堆顶元素。
- size() 返回元素个数。
- empty() 返回是否非空。
- modify(point_iterator, const key): 把迭代器位置的 key 修改为传入的 key, 并对底层储存结构进行排序。
- erase(point_iterator): 把迭代器位置的键值从堆中擦除。
- join(__gnu_pbds::priority_queue &other): 把 other 合并到 *this 并把 other 清空。

使用的 tag 决定了每个操作的时间复杂度: pairing_heap_tag push: $O(1)$ pop: 最坏 $\Theta(n)$ 均摊 $\Theta(\log(n))$ modify: 最坏 $\Theta(n)$ 均摊 $\Theta(\log(n))$ erase: 最坏 $\Theta(n)$ 均摊 $\Theta(\log(n))$ join: $O(1)$

示例

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <ext/pb_ds/priority_queue.hpp>
4  #include <iostream>
5  using namespace __gnu_pbds;
6  // 由于面向 OIer, 本文以常用堆: pairing_heap_tag 作为范例

```

```

7 // 为了更好的阅读体验，定义宏如下：
8 #define pair_heap __gnu_pbds::priority_queue<int>
9 pair_heap q1; // 大根堆，配对堆
10 pair_heap q2;
11 pair_heap::point_iterator id; // 一个迭代器
12
13 int main() {
14     id = q1.push(1);
15     // 堆中元素 : [1];
16     for (int i = 2; i <= 5; i++) q1.push(i);
17     // 堆中元素 : [1, 2, 3, 4, 5];
18     std::cout << q1.top() << std::endl;
19     // 输出结果 : 5;
20     q1.pop();
21     // 堆中元素 : [1, 2, 3, 4];
22     id = q1.push(10);
23     // 堆中元素 : [1, 2, 3, 4, 10];
24     q1.modify(id, 1);
25     // 堆中元素 : [1, 1, 2, 3, 4];
26     std::cout << q1.top() << std::endl;
27     // 输出结果 : 4;
28     q1.pop();
29     // 堆中元素 : [1, 1, 2, 3];
30     id = q1.push(7);
31     // 堆中元素 : [1, 1, 2, 3, 7];
32     q1.erase(id);
33     // 堆中元素 : [1, 1, 2, 3];
34     q2.push(1), q2.push(3), q2.push(5);
35     // q1 中元素 : [1, 1, 2, 3], q2 中元素 : [1, 3, 5];
36     q2.join(q1);
37     // q1 中无元素, q2 中元素 : [1, 1, 1, 2, 3, 3, 5];
38 }

```

hash 表

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
4 struct chash {
5     int operator()(int x) const { return x ^ RANDOM; }
6 };
7 typedef gp_hash_table<int, int, chash> hash_t;

```

rope

```

1 #include <ext/rope>
2 using namespace __gnu_cxx;

```

1) 运算符: rope 支持 operator +=, -=, +, -, <, ==

2) 输入输出: 可以用 << 运算符由输入输出流读入或输出。

3) 长度/大小: 调用 length(), size() 都可以

4) 插入/添加等:

push_back(x): 在末尾添加 x

insert(pos, x): 在 pos 插入 x, 自然支持整个 char 数组的一次插入

erase(pos, x): 从 pos 开始删除 x 个

copy(pos, len, x): 从 pos 开始到 pos+len 为止用 x 代替

replace(pos, x): 从 pos 开始换成 x

substr(pos, x): 提取 pos 开始 x 个

at(x)/[x]: 访问第 x 个元素

对拍

```
1  #!/bin/bash
2  while true; do
3      ./data > data.in
4      ./std <data.in >std.out
5      ./Todobc <data.in >Todobc.out
6      if diff std.out Todobc.out; then
7          printf "AC\n"
8      else
9          printf "WA\n"
10         exit 0
11     fi
12 done
```

火车头

```
1  #pragma GCC optimize(3)
2  #pragma GCC target("avx")
3  #pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
4  #pragma GCC optimize("unroll-loops")
5  #pragma GCC optimize("Ofast")
6  #pragma GCC optimize("inline")
7  #pragma GCC optimize("-fgcse")
8  #pragma GCC optimize("-fgcse-lm")
9  #pragma GCC optimize("-fipa-sra")
10 #pragma GCC optimize("-ftree-pre")
11 #pragma GCC optimize("-ftree-vrp")
12 #pragma GCC optimize("-fpeephole2")
13 #pragma GCC optimize("-ffast-math")
14 #pragma GCC optimize("-fsched-spec")
15 #pragma GCC optimize("-falign-jumps")
16 #pragma GCC optimize("-falign-loops")
17 #pragma GCC optimize("-falign-labels")
18 #pragma GCC optimize("-fdevirtualize")
19 #pragma GCC optimize("-fcaller-saves")
20 #pragma GCC optimize("-fcrossjumping")
21 #pragma GCC optimize("-fthread-jumps")
22 #pragma GCC optimize("-funroll-loops")
23 #pragma GCC optimize("-fwhole-program")
24 #pragma GCC optimize("-freorder-blocks")
25 #pragma GCC optimize("-fschedule-insns")
26 #pragma GCC optimize("inline-functions")
27 #pragma GCC optimize("-ftree-tail-merge")
28 #pragma GCC optimize("-fschedule-insns2")
29 #pragma GCC optimize("-fstrict-aliasing")
30 #pragma GCC optimize("-fstrict-overflow")
31 #pragma GCC optimize("-falign-functions")
32 #pragma GCC optimize("-fcse-skip-blocks")
33 #pragma GCC optimize("-fcse-follow-jumps")
34 #pragma GCC optimize("-fsched-interblock")
35 #pragma GCC optimize("-fpartial-inlining")
36 #pragma GCC optimize("no-stack-protector")
37 #pragma GCC optimize("-freorder-functions")
38 #pragma GCC optimize("-findirect-inlining")
39 #pragma GCC optimize("-fhoist-adjacent-loads")
40 #pragma GCC optimize("-frerun-cse-after-loop")
41 #pragma GCC optimize("inline-small-functions")
42 #pragma GCC optimize("-finline-small-functions")
43 #pragma GCC optimize("-ftree-switch-conversion")
44 #pragma GCC optimize("-foptimize-sibling-calls")
45 #pragma GCC optimize("-fexpensive-optimizations")
46 #pragma GCC optimize("-funsafe-loop-optimizations")
47 #pragma GCC optimize("inline-functions-called-once")
48 #pragma GCC optimize("-fdelete-null-pointer-checks")
49 #pragma GCC optimize(2)
```

Sublime

```
{
```

```

"encoding": "utf-8",
"working_dir": "${file_path}",
"shell_cmd": "g++ -Wall -std=c++2a \"${file}\" -o \"${file_path}/${file_base_name}\",
"file_regex": "^(..[^:]*):([0-9]+):?([0-9]+)??:? (.*)$",
"selector": "source.c++",

"variants":
[
    {
        "name": "Run",
        "shell_cmd": "g++ -Wall -std=c++2a \"${file}\" -o \"${file_base_name}\" && start cmd /c \"\"${file_
    }
]
}

```

卡常

vector 的调用空间和运算的效率并不低，主要是多次的加入元素过程效率很低。

减少申请空间的次数，例如不要循环内开 vector

```

1 // vector f(n,vector<int>(m,0));
2 f.assign(n, std::vector(m, 0));

```

数组访问尽量连续

被卡常时，不要爆交，先多想想剪枝

注意事项

- 相信所有题都是可做的。
- 认真读题，模拟完样例再写程序。
- 热身赛，测试机器速度，重点测 $O(n \log n)$, $O(n \log^2 n)$, $O(n^3)$, $O(n^2 \log n)$ 。
- 感觉不可做的，有较高多项式复杂度暴力的题，思考：分治、贪心、dp、线段树。
- 感觉不可做的，只有指数级复杂度暴力的最优化题，思考：贪心、dp、流和割、暴搜加优化。
- 感觉不可做的，只有指数级暴力的数数题，思考：dp、行列式、暴搜加优化、拉格朗日插值、容斥、造自动机。
- 构造、交互题，考虑：增量法、分治、暴搜策略。
- dp 优化：凸优化（wqs，闵可夫斯基和，李超树）、斜率优化，决策单调性、交换状态和值域、减少状态（包含常数上的）。
- 感觉不可做的题，考虑各个元素/集合之间有什么关系。
- 对于复杂度比较顶的做法，一定要充分沟通后再上机
- `int(v.size())` 切记不能 `ull` 减 `int`
- `__builtin_popcount` 和 `__builtin_popcountll`
- `sqrt` 和 `sqrtl`，`sqrtl` 返回 `long double`
- 几何题注意是不是可能返回 `nan`
- 不能 `x * 1ll` 而是 `1ll * x`
- 比较长的题，写一部分测一部分不要最后一块测
- 任何 n 较大的，可以快速算单项的东西考虑分段打表。

策略

签到题不会做，先确认题面，题面无误看看是不是想难了或者暴力很有道理。

沟通题意前切记确认题面，对着题面和队友讲题意

咕宝主要问题在于不能陷入长时间无效思考状态，所以咕宝优先读题。

我和胜利爷有时候会在一些思路堵死，要及时跳出来误区。

榜上有简单题做不出来的时候，切记转换一下思路或立即拉另一个人过来重新想（先不要交流思路）

对于签到，和咕宝沟通下题意直接让他写。

对于铜银题，板子比较多的我来写，比较有细节的胜利爷来写，剩下的咕宝来写

对于金牌题，最后大伙一块 all in，交给咕宝来写