

Standard Code Library

Your TeamName

Your School

May 12, 2024

Contents

一切的开始	2
数据结构	2
ST 表	2
线段树	2
树状数组	3
DSU	3
Splay	3
LCT	4
扫描线	6
Seg beats	8
珂朵莉树	10
李超树	10
动态维护凸壳	11
图论	12
树链剖分	12
LCA	13
倍增求 LCA	13
dfn 求 LCA	13
树哈希	14
虚树	14
Dijkstra	14
最小环	15
差分约束	15
最大流	15
最小费用最大流	16
二分图最大匹配	17
KM(二分图最大权匹配)	18
一般图最大匹配	19
缩点 SCC	21
割点与桥	22
边双缩点	22
圆方树	23
广义圆方树	23
2-SAT	24
环计数	24
字符串	24
manacher	24
SA	25
PAM	26
SAM	26
ACAM	28
KMP	29
Z 函数	29
LCP	30
Hash	30

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 ++){
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 MCFGraph(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[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 };

```