

Standard Code Library

SDU-TCS

Shandong University

October 14, 2024

Contents

一切的开始	2
数据结构	2
ST 表	2
线段树	3
树上动态直径	4
扫描线	6
Seg beats	8
树状数组	10
DSU	10
Splay	11
LCT	11
珂朵莉树	13
李超树	13
动态维护凸壳	14
线段树合并	15
图论	16
树链剖分	16
LCA	16
倍增求 LCA	16
dfn 求 LCA	17
树哈希	17
虚树	18
最小环	18
差分约束	18
最大流	19
最小费用最大流	20
二分图最大匹配	21
KM(二分图最大权匹配)	21
一般图最大匹配	23
缩点 SCC	25
割点与桥	25
边双缩点	26
圆方树	26
广义圆方树	27
2-SAT	27
环计数	27

一切的开始

数据结构

ST 表

```
1  template<class T,  
2      class Cmp = std::less<T>>  
3  struct RMQ {  
4      const Cmp cmp = Cmp();  
5      static constexpr unsigned B = 64;  
6      using u64 = unsigned long long;  
7      int n;  
8      std::vector<std::vector<T>> a;  
9      std::vector<T> pre, suf, ini;  
10     std::vector<u64> stk;  
11     RMQ() {}  
12     RMQ(const std::vector<T> &v) {  
13         init(v);  
14     }  
15     void init(const std::vector<T> &v) {  
16         n = v.size();  
17         pre = suf = ini = v;  
18         stk.resize(n);  
19         if (!n) {  
20             return;  
21         }  
22         const int M = (n - 1) / B + 1;  
23         const int lg = std::__lg(M);  
24         a.assign(lg + 1, std::vector<T>(M));  
25         for (int i = 0; i < M; i++) {  
26             a[0][i] = v[i * B];  
27             for (int j = 1; j < B && i * B + j < n; j++) {  
28                 a[0][i] = std::min(a[0][i], v[i * B + j], cmp);  
29             }  
30         }  
31         for (int i = 1; i < n; i++) {  
32             if (i % B) {  
33                 pre[i] = std::min(pre[i], pre[i - 1], cmp);  
34             }  
35         }  
36         for (int i = n - 2; i >= 0; i--) {  
37             if (i % B != B - 1) {  
38                 suf[i] = std::min(suf[i], suf[i + 1], cmp);  
39             }  
40         }  
41         for (int j = 0; j < lg; j++) {  
42             for (int i = 0; i + (2 << j) <= M; i++) {  
43                 a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);  
44             }  
45         }  
46         for (int i = 0; i < M; i++) {  
47             const int l = i * B;  
48             const int r = std::min(1U * n, l + B);  
49             u64 s = 0;  
50             for (int j = l; j < r; j++) {  
51                 while (s && cmp(v[j], v[std::__lg(s) + l])) {  
52                     s ^= 1ULL << std::__lg(s);  
53                 }  
54                 s |= 1ULL << (j - l);  
55                 stk[j] = s;  
56             }  
57         }  
58     }  
59     T operator()(int l, int r) {  
60         if (l / B != (r - 1) / B) {  
61             T ans = std::min(suf[l], pre[r - 1], cmp);  
62             l = l / B + 1;  
63             r = r / B;  
64             if (l < r) {
```

```

65         int k = std::__lg(r - l);
66         ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
67     }
68     return ans;
69 } else {
70     int x = B * (l / B);
71     return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + 1];
72 }
73 }
74 };
75

```

线段树

```

1  struct SegTree {
2      int l, r;
3      SegTree *ls, *rs;
4      int sum;
5      int mx;
6      int mn;
7      int plus = 0;
8      SegTree(const int L, const int R) : l(L), r(R) {
9          plus = 0; mx = mn = 0;
10         if (L == R) {
11             /*Initial*/
12             // sum = 1;
13             ls = rs = nullptr;
14         } else {
15             int M = (L + R) >> 1;
16             ls = new SegTree(L, M);
17             rs = new SegTree(M + 1, R);
18             pushup();
19         }
20     }
21     void pushup() {
22         sum = ls->sum + rs->sum;
23         mn = min(ls->mn, rs->mn);
24         mx = max(ls->mx, rs->mx);
25     }
26     void make_tag(long long w) {
27         sum += (r - l + 1) * w;
28         mn += w;
29         mx += 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 upd(const int L, const int R, const int w) {
39         if ((L > r) || (l > R)) return;
40         if ((L <= l) && (r <= R)) {
41             make_tag(w);
42         } else {
43             pushdown();
44             ls->upd(L, R, w);
45             rs->upd(L, R, w);
46             pushup();
47         }
48     }
49     void upd(const int x, const int w) {
50         if ((x > r) || (l > x)) return;
51         if (l == x && r == x) {
52             sum = w;
53         } else {
54             ls->upd(x, w);
55             rs->upd(x, w);
56             pushup();
57         }
58     }
59 }

```

```

58     }
59     int qry(const int L,const int R) {
60         if ((L > r) || (l > R)) return 0;
61         if ((L <= l) && (r <= R)) {
62             return sum;
63         } else {
64             pushdown();
65             return ls->qry(L, R) + rs->qry(L, R);
66         }
67     }
68     bool check(int w) {
69         if(mn < w - 1 || mx > w)return false;
70         return true;
71     }
72     int findR(int L,int R,int w) {
73         if ((L > r) || (l > R)) return -1;
74         if ((L <= l) && (r <= R)) {
75             if(check(w)) return -1;
76             if(l == r) {
77                 return l;
78             }
79         }
80         pushdown();
81         int res = ls->findR(L,R,w);
82         if(res == -1) {
83             res = rs->findR(L,R,w);
84         }
85         return res;
86     }
87 };

```

树上动态直径

```

1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  using ll = long long;
6  using pii = pair<int,int>;
7
8  const int maxn = 2e5 + 7;
9  int n,q;
10 ll w;
11 vector<pair<int,ll>> e[maxn];
12
13 ll c[maxn];
14 void add(int x,ll y) {
15     for(;x <= n;x += (x & (-x))) c[x] += y;
16 }
17 void change(int l,int r,ll x) { // [l,r]
18     add(l,x);add(r + 1,-x);
19 }
20 ll ask(int x) {
21     ll ans = 0;
22     for(;x > 0;x -= (x & (-x))) ans += c[x];
23     return ans;
24 }
25
26 int dfn[maxn],mi[22][maxn],id[maxn],siz[maxn],cnt;
27 int get(int x, int y) {return dfn[x] < dfn[y] ? x : y;}
28 void dfs(int x, int fa) {
29     mi[0][dfn[x]] = ++cnt = fa;
30     id[cnt] = x;siz[x] = 1;
31     for(auto [v,w] : e[x]) {
32         if(v == fa) continue;
33         dfs(v,x);
34         change(dfn[v],dfn[v] + siz[v] - 1,w);
35         siz[x] += siz[v];
36     }
37 }
38 int lca(int u, int v) {

```

```

39     if(u == v) return u;
40     if((u = dfn[u]) > (v = dfn[v])) swap(u, v);
41     int d = __lg(v - u++);
42     return get(mi[d][u], mi[d][v - (1 << d) + 1]);
43 }
44 ll dis(pii a) {
45     auto [u,v] = a;
46     return ask(dfn[u]) + ask(dfn[v]) - 2 * ask(dfn[lca(u,v)]);
47 }
48
49 pii tr[maxn << 2];
50 pii merge(pii &a,pii &b) {
51     pii p[6] = {a,b,{a.first,b.first},{a.first,b.second},{a.second,b.first},{a.second,b.second}};
52     vector<ll> d(6);
53     for(int i = 0;i < 6;i++) d[i] = dis(p[i]);
54     return p[max_element(d.begin(),d.end()) - d.begin()];
55 }
56 void build(int x,int l,int r) {
57     if(l == r) {
58         tr[x] = {id[l],id[l]};
59         return ;
60     }
61     int mid = l + r >> 1;
62     build(x << 1,l,mid);
63     build(x << 1 | 1,mid + 1,r);
64     tr[x] = merge(tr[x << 1],tr[x << 1 | 1]);
65 }
66 void update(int x,int L,int R,int l,int r) {
67     if(L == l && r == R) return ;
68     int mid = L + R >> 1;
69     if(r <= mid) update(x << 1,L,mid,l,r);
70     else if(l > mid) update(x << 1 | 1,mid + 1,R,l,r);
71     else update(x << 1,L,mid,l,mid),update(x << 1 | 1,mid + 1,R,mid + 1,r);
72     tr[x] = merge(tr[x << 1],tr[x << 1 | 1]);
73 }
74
75 int main() {
76     ios::sync_with_stdio(false);
77     cin.tie(0);
78     cin >> n >> q >> w;
79     vector<tuple<int,int,int>> edges;
80     for(int i = 0;i < n - 1;i++) {
81         int x,y;ll z;
82         cin >> x >> y >> z;
83         e[x].push_back({y,z});
84         e[y].push_back({x,z});
85         edges.emplace_back(x,y,z);
86     }
87     dfs(1, 0);
88     for(int i = 1; i <= __lg(n); i++)
89         for(int j = 1; j + (1 << i) - 1 <= n; j++)
90             mi[i][j] = get(mi[i - 1][j], mi[i - 1][j + (1 << i - 1)]);
91     ll lastans = 0;
92     build(1,1,n);
93     for(int i = 0;i < q;i++) {
94         int x;ll y;
95         cin >> x >> y;
96         x = (x + lastans) % (n - 1);
97         y = (y + lastans) % w;
98         auto [son,fa,ww] = edges[x];
99         if(dfn[son] < dfn[fa]) swap(son,fa);
100         change(dfn[son],dfn[son] + siz[son] - 1,y - ww);
101         update(1,1,n,dfn[son],dfn[son] + siz[son] - 1);
102         lastans = dis(tr[1]);
103         cout << lastans << "\n";
104         ww = y;
105         edges[x] = {son,fa,ww};
106     }
107 }

```

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 }

```

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 makeroor(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 }

```

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      ll k, b;
5      int id;
6      Line(ll k, ll b, int id) : k(k), b(b), id(id) {}

```

```

7   };
8   long long f(const Line &line, int x) {
9       return 1LL * line.k * x + line.b;
10  }
11  Line get(Line a, Line c, int x) {
12      ll b = f(a,x), d = f(c,x);
13      return b == d ? (a.id < c.id ? a : c) : b > d ? a : c;
14  }
15  struct Node {
16      Node *lc, *rc;
17      Line line;
18      Node(const Line &line) : lc(nullptr), rc(nullptr), line(line) {}
19  };
20  void modify(Node *&p, int l, int r, Line line) {
21      if (p == nullptr) {
22          p = new Node(Line(0,-1e18,0));
23      }
24      if(l == r) {
25          if(f(p -> line,l) <= f(line,l)) p -> line = line;
26          return ;
27      }
28      int m = (l + r) / 2;
29      if (f(p -> line, m) < f(line, m)) swap(p -> line, line);
30      if (f(p -> line, l) < f(line, l)) modify(p -> lc, l, m, line);
31      else if(f(p -> line, r) < f(line, r)) modify(p -> rc, m + 1, r, line);
32  }
33  Node *merge(Node *p, Node *q, int l, int r) {
34      if (p == nullptr)
35          return q;
36      if (q == nullptr)
37          return p;
38      int m = (l + r) / 2;
39      p -> lc = merge(p -> lc, q -> lc, l, m);
40      p -> rc = merge(p -> rc, q -> rc, m, r);
41      modify(p, l, r, q -> line);
42      return p;
43  }
44  Line query(Node *p, int l, int r, int x) {
45      if (p == nullptr)
46          return Line(0,-1e18,0);
47      if(l == r) return p -> line;
48      int m = (l + r) / 2;
49      if (x <= m) return get(query(p -> lc, l, m, x),p -> line,x);
50      return get(query(p -> rc, m + 1, r, x),p -> line,x);
51  }

```

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 }

```

线段树合并

```

1     struct Info{
2         int mx = 0, id = 0;
3         Info() {}
4         Info(int mx, int id) : mx(mx), id(id) {}
5     };
6     Info operator+(const Info a, const Info b) {
7         if(a.mx > b.mx) return a;
8         if(b.mx > a.mx) return b;
9         if(a.id < b.id) return a;
10        else return b;
11    }
12    struct Node {
13        int l, r;
14        Node *ls, *rs;
15        Info t;
16        Node(int l, int r) : l(l), r(r), ls(nullptr), rs(nullptr) {}
17    };
18
19    void push_up(Node *&p) {
20        if(p->ls == nullptr) {
21            p->t = p->rs->t; return ;
22        }
23        if(p->rs == nullptr) {
24            p->t = p->ls->t; return ;
25        }
26        p->t = p->ls->t + p->rs->t;
27    }
28    void upd(Node *&p, int l, int r, int x, int w) {
29        if(p == nullptr) {
30            p = new Node(l, r);
31        }
32        if(l == r) {

```



```

33     p->t.mx += w;
34     p->t.id = x;
35     return ;
36 }
37 int mid = l + r >> 1;
38 if(x <= mid) upd(p->ls,l,mid,x,w);
39 else upd(p->rs,mid + 1,r,x,w);
40 push_up(p);
41 }
42
43 Node* merge(Node *p,Node *q,int l,int r) {
44     if(p == nullptr) return q;
45     if(q == nullptr) return p;
46     if(l == r) {
47         p->t.mx += q->t.mx;
48         return p;
49     }
50     int mid = l + r >> 1;
51     p->ls = merge(p->ls,q->ls,l,mid);
52     p->rs = merge(p->rs,q->rs,mid + 1,r);
53     push_up(p);
54     return p;
55 }

```

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 }

```

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 }

```

```

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     }
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  struct MCFGraph {
3      struct Edge {
4          int v, c, f;
5          Edge(int v, int c, int f) : v(v), c(c), f(f) {}
6      };
7      const int n;
8      std::vector<Edge> e;
9      std::vector<std::vector<int>> g;
10     std::vector<i64> h, dis;
11     std::vector<int> pre;
12     bool dijkstra(int s, int t) {
13         dis.assign(n, std::numeric_limits<i64>::max());
14         pre.assign(n, -1);
15         priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<pair<i64, int>>> que;
16         dis[s] = 0;
17         que.emplace(0, s);
18         while (!que.empty()) {
19             i64 d = que.top().first;
20             int u = que.top().second;
21             que.pop();
22             if (dis[u] < d) continue;
23             for (int i : g[u]) {
24                 int v = e[i].v;
25                 int c = e[i].c;
26                 int f = e[i].f;
27                 if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
28                     dis[v] = d + h[u] - h[v] + f;
29                     pre[v] = i;
30                     que.emplace(dis[v], v);
31                 }
32             }
33         }
34         return dis[t] != std::numeric_limits<i64>::max();
35     }
36     MCFGraph(int n) : n(n), g(n) {}
37     void addEdge(int u, int v, int c, int f) {
38         if (f < 0) {
39             g[u].push_back(e.size());
40             e.emplace_back(v, 0, f);
41             g[v].push_back(e.size());
42             e.emplace_back(u, c, -f);
43         } else {
44             g[u].push_back(e.size());
45             e.emplace_back(v, c, f);
46             g[v].push_back(e.size());
47             e.emplace_back(u, 0, -f);
48         }
49     }
50     std::pair<int, i64> flow(int s, int t) {
51         int flow = 0;
52         i64 cost = 0;
53         h.assign(n, 0);
54         while (dijkstra(s, t)) {
55             for (int i = 0; i < n; ++i) h[i] += dis[i];
56             int aug = std::numeric_limits<int>::max();
57             for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug, e[pre[i]].c);
58             for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
59                 e[pre[i]].c -= aug;
60                 e[pre[i] ^ 1].c += aug;
61             }
62             flow += aug;
63             cost += i64(aug) * h[t];

```

```

64     }
65     return std::make_pair(flow, cost);
66 }
67 };

1  const int N = 5e3 + 5, M = 5e4 + 5;
2  struct flow {
3      int cnt = 1, hd[N], nxt[M << 1], to[M << 1], limit[M << 1], cst[M << 1];
4      void add(int u, int v, int w, int c) {
5          nxt[++cnt] = hd[u], hd[u] = cnt, to[cnt] = v, limit[cnt] = w, cst[cnt] = c;
6          nxt[++cnt] = hd[v], hd[v] = cnt, to[cnt] = u, limit[cnt] = 0, cst[cnt] = -c;
7      }
8      int fr[N], fl[N], in[N], dis[N];
9      pair<int, int> mincost(int s, int t) {
10         int flow = 0, cost = 0;
11         while(1) {
12             queue<int> q;
13             memset(dis, 0x3f, sizeof(dis));
14             memset(in, 0, sizeof(in));
15             fl[s] = 1e9, dis[s] = 0, q.push(s);
16             while(!q.empty()) {
17                 int t = q.front();
18                 q.pop(), in[t] = 0;
19                 for(int i = hd[t]; i; i = nxt[i]) {
20                     int it = to[i], d = dis[t] + cst[i];
21                     if(limit[i] && d < dis[it]) {
22                         fl[it] = min(limit[i], fl[t]), fr[it] = i, dis[it] = d;
23                         if(!in[it]) in[it] = 1, q.push(it);
24                     }
25                 }
26             }
27             if(dis[t] > 1e9) return make_pair(flow, cost);
28             flow += fl[t], cost += dis[t] * fl[t];
29             for(int u = t; u != s; u = to[fr[u] ^ 1]) limit[fr[u]] -= fl[t], limit[fr[u] ^ 1] += fl[t];
30         }
31     }
32 } g;

```

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         // disjoint set union
13         auto find = [&](int u) {
14             while (f[u] != u)
15                 u = f[u] = f[f[u]];
16             return u;
17         };
18         auto lca = [&](int u, int v) {
19             u = find(u);
20             v = find(v);
21             while (u != v) {
22                 if (dep[u] < dep[v])
23                     std::swap(u, v);

```



```

24         u = find(link[match[u]]);
25     }
26     return u;
27 };
28 std::queue<int> q;
29 auto blossom = [&](int u, int v, int p) {
30     while (find(u) != p) {
31         link[u] = v;
32         v = match[u];
33         if (vis[v] == 0) {
34             vis[v] = 1;
35             q.push(v);
36         }
37         f[u] = f[v] = p;
38         u = link[v];
39     }
40 };
41 // find an augmenting path starting from u and augment (if exist)
42 auto augment = [&](int u) {
43     while (!q.empty())
44         q.pop();
45     std::iota(f.begin(), f.end(), 0);
46     // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
47     std::fill(vis.begin(), vis.end(), -1);
48
49     q.push(u);
50     vis[u] = 1;
51     dep[u] = 0;
52
53     while (!q.empty()){
54         int u = q.front();
55         q.pop();
56         for (auto v : e[u]) {
57             if (vis[v] == -1) {
58                 vis[v] = 0;
59                 link[v] = u;
60                 dep[v] = dep[u] + 1;
61                 // found an augmenting path
62                 if (match[v] == -1) {
63                     for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]){
64                         temp = match[y];
65                         match[x] = y;
66                         match[y] = x;
67                     }
68                     return;
69                 }
70                 vis[match[v]] = 1;
71                 dep[match[v]] = dep[u] + 2;
72                 q.push(match[v]);
73             } else if (vis[v] == 1 && find(v) != find(u)) {
74                 // found a blossom
75                 int p = lca(u, v);
76                 blossom(u, v, p);
77                 blossom(v, u, p);
78             }
79         }
80     }
81 }
82
83 };
84 // find a maximal matching greedily (decrease constant)
85 auto greedy = [&]() {
86     for (int u = 1; u <= n; ++u) {
87         if (match[u] != -1)
88             continue;
89         for (auto v : e[u]) {
90             if (match[v] == -1) {
91                 match[u] = v;
92                 match[v] = u;
93                 break;
94             }
95         }
96     }
97 }

```

```

95         }
96     }
97 };
98 greedy();
99 for (int u = 1; u <= n; ++u)
100     if (match[u] == -1)
101         augment(u);
102
103     return match;
104 }
105 };
106 int main() {
107     std::ios::sync_with_stdio(false);
108     std::cin.tie(nullptr);
109     int n, m;
110     std::cin >> n >> m;
111     Graph g(n);
112     for (int i = 0; i < m; ++i) {
113         int u, v;
114         std::cin >> u >> v;
115         g.addEdge(u, v);
116     }
117     auto match = g.findMatching();
118     int ans = 0;
119     for (int u = 1; u <= n; ++u)
120         if (match[u] != -1)
121             ++ans;
122     std::cout << ans / 2 << "\n";
123     for (int u = 1; u <= n; ++u)
124         if (match[u] != -1) std::cout << match[u] << " ";
125         else std::cout << 0 << " ";
126     return 0;
127 }

```

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 }

```

```

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 }

```