Standard Code Library

SDU-TCS

Shandong University

October 14, 2024

Contents

一切的开始

数据结构

ST 表

```cpp
template<class T,
    class Cmp = std::less<T>>
struct RMQ {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    using u64 = unsigned long long;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    RMQ() {}
    RMQ(const std::vector<T> &v) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        pre = suf = ini = v;
        stk.resize(n);
        if (!n) {
            return;
        }
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
```

```
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};

```

线段树

```
struct SegTree {
    int l, r;
    SegTree *ls, *rs;
    int sum;
    int mx;
    int mn;
    int plus = 0;
    SegTree (const int L, const int R) : l(L), r(R) {
        plus = 0;mx = mn = 0;
        if (L == R) {
            /*Initial*/
            // sum = 1;
            ls = rs = nullptr;
        } else {
            int M = (L + R) >> 1;
            ls = new SegTree (L, M);
            rs = new SegTree (M + 1, R);
            pushup();
        }
    }
    void pushup() {
        sum = ls->sum + rs->sum;
        mn = min(ls->mn,rs->mn);
        mx = max(ls->mx,rs->mx);
    }
    void make_tag(long long w) {
        sum += (r - l + 1) * w;
        mn += w;
        mx += w;
        plus += w;
    }
    void pushdown() {
        if (plus == 0) return;
        ls->make_tag(plus);
        rs->make_tag(plus);
        plus = 0;
    }
    void upd(const int L, const int R, const int w) {
        if ((L > r) || (l > R)) return;
        if ((L <= l) && (r <= R)) {
            make_tag(w);
        } else {
            pushdown();
            ls->upd(L, R, w);
            rs->upd(L, R, w);
            pushup();
        }
    }
    void upd(const int x,const int w) {
        if((x > r) || (l > x)) return ;
        if(l == x && r == x) {
            sum = w;
        } else {
            ls->upd(x,w);
            rs->upd(x,w);
            pushup();
        }
```

```
    }
    int qry(const int L,const int R) {
        if ((L > r) || (l > R)) return 0;
        if ((L <= l) && (r <= R)) {
            return sum;
        } else {
            pushdown();
            return ls->qry(L, R) + rs->qry(L, R);
        }
    }
    bool check(int w) {
        if(mn < w - 1 || mx > w)return false;
        return true;
    }
    int findR(int L,int R,int w) {
        if ((L > r) || (l > R)) return -1;
        if ((L <= l) && (r <= R)) {
            if(check(w)) return -1;
            if(l == r) {
               return l;
            }
        }
        pushdown();
        int res = ls->findR(L,R,w);
        if(res == -1) {
            res = rs->findR(L,R,w);
        }
        return res;
    }
};
```

树上动态直径

```
#include<bits/stdc++.h>

using namespace std;

using ll = long long;
using pii = pair<int,int>;

const int maxn = 2e5 + 7;
int n,q;
ll w;
vector<pair<int,ll>> e[maxn];

ll c[maxn];
void add(int x,ll y) {
    for(;x <= n;x += (x & (-x))) c[x] += y;
}
void change(int l,int r,ll x) { // [l,r]
    add(l,x);add(r + 1,-x);
}
ll ask(int x) {
    ll ans = 0;
    for(;x > 0;x -= (x & (-x))) ans += c[x];
    return ans;
}

int dfn[maxn],mi[22][maxn],id[maxn],siz[maxn],cnt;
int get(int x, int y) {return dfn[x] < dfn[y] ? x : y;}
void dfs(int x, int fa) {
    mi[0][dfn[x] = ++cnt] = fa;
    id[cnt] = x;siz[x] = 1;
    for(auto [v,w] : e[x]) {
        if(v == fa) continue;
        dfs(v,x);
        change(dfn[v],dfn[v] + siz[v] - 1,w);
        siz[x] += siz[v];
    }
}
int lca(int u, int v) {
```

```
    if(u == v) return u;
    if((u = dfn[u]) > (v = dfn[v])) swap(u, v);
    int d = __lg(v - u++);
    return get(mi[d][u], mi[d][v - (1 << d) + 1]);
}
ll dis(pii a) {
    auto [u,v] = a;
    return ask(dfn[u]) + ask(dfn[v]) - 2 * ask(dfn[lca(u,v)]);
}

pii tr[maxn << 2];
pii merge(pii &a,pii &b) {
    pii p[6] = {a,b,{a.first,b.first},{a.first,b.second},{a.second,b.first},{a.second,b.second}};
    vector<ll> d(6);
    for(int i = 0;i < 6;i ++) d[i] = dis(p[i]);
    return p[max_element(d.begin(),d.end()) - d.begin()];
}
void build(int x,int l,int r) {
    if(l == r) {
        tr[x] = {id[l],id[l]};
        return ;
    }
    int mid = l + r >> 1;
    build(x << 1,l,mid);
    build(x << 1 | 1,mid + 1,r);
    tr[x] = merge(tr[x << 1],tr[x << 1 | 1]);
}
void update(int x,int L,int R,int l,int r) {
    if(L == l && r == R) return ;
    int mid = L + R >> 1;
    if(r <= mid) update(x << 1,L,mid,l,r);
    else if(l > mid) update(x << 1 | 1,mid + 1,R,l,r);
    else update(x << 1,L,mid,l,mid),update(x << 1 | 1,mid + 1,R,mid + 1,r);
    tr[x] = merge(tr[x << 1],tr[x << 1 | 1]);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> q >> w;
    vector<tuple<int,int,int>> edges;
    for(int i = 0;i < n - 1;i ++) {
        int x,y;ll z;
        cin >> x >> y >> z;
        e[x].push_back({y,z});
        e[y].push_back({x,z});
        edges.emplace_back(x,y,z);
    }
    dfs(1, 0);
    for(int i = 1; i <= __lg(n); i++)
      for(int j = 1; j + (1 << i) - 1 <= n; j++)
        mi[i][j] = get(mi[i - 1][j], mi[i - 1][j + (1 << i - 1)]);
    ll lastans = 0;
    build(1,1,n);
    for(int i = 0;i < q;i ++) {
        int x;ll y;
        cin >> x >> y;
        x = (x + lastans) % (n - 1);
        y = (y + lastans) % w;
        auto [son,fa,ww] = edges[x];
        if(dfn[son] < dfn[fa]) swap(son,fa);
        change(dfn[son],dfn[son] + siz[son] - 1,y - ww);
        update(1,1,n,dfn[son],dfn[son] + siz[son] - 1);
        lastans = dis(tr[1]);
        cout << lastans << "\n";
        ww = y;
        edges[x] = {son,fa,ww};
    }
}
```

扫描线

```cpp
//二维数点
struct Segment{
    int l,r,h,add;
    bool operator <(const Segment a)const{
        return h < a.h;
    }
};
struct SegTree {
    int l, r;
    SegTree *ls, *rs;
    int mn,len;
    int plus;
    SegTree (const int L, const int R) : l(L), r(R) {
        plus = 0;len = 0;
        if (L == R) {
            ls = rs = nullptr;
        } else {
            int M = (L + R) >> 1;
            ls = new SegTree (L, M);
            rs = new SegTree (M + 1, R);
            pushup();
        }
    }
    void pushup() {
        if(plus) len = r - l + 1;
        else if(l == r)len = 0;
        else len = ls->len + rs->len;
    }
    void make_tag(int w) {
        plus += w;
    }
    void pushdown() {
        if (plus == 0) return;
        ls->make_tag(plus);
        rs->make_tag(plus);
        plus = 0;
    }
    void update(const int L, const int R, const int w) {
        if ((L > r) || (l > R)) {
            return;
        }
        if ((L <= l) && (r <= R)) {
            make_tag(w);
            pushup();
            return ;
        } else {
            ls->update(L, R, w);
            rs->update(L, R, w);
            pushup();
        }
    }
};
//矩形面积并
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;
const double eps = 1e-8;
const int maxn = 2e5 + 7;
std::vector<int> x;
struct Segment{
    int l,r,h,add;
    bool operator <(const Segment a)const{
        return h < a.h;
    }
};
struct SegTree {
    int l, r;
    SegTree *ls, *rs;
```

```
    int mn,len;
    int plus;
    SegTree (const int L, const int R) : l(L), r(R) {
        plus = 0;len = 0;
        if (L == R) {
            ls = rs = nullptr;
        } else {
            int M = (L + R) >> 1;
            ls = new SegTree (L, M);
            rs = new SegTree (M + 1, R);
            pushup();
        }
    }
    void pushup() {
        if(plus) len = x[r] - x[l - 1];
        else if(l == r)len = 0;
        else len = ls->len + rs->len;
    }
    void make_tag(int w) {
        plus += w;
    }
    void pushdown() {
        if (plus == 0) return;
        ls->make_tag(plus);
        rs->make_tag(plus);
        plus = 0;
    }
    void update(const int L, const int R, const int w) {
        if ((L >= x[r]) || (x[l - 1] >= R)) {
            return;
        }
        if ((L <= x[l - 1]) && (x[r] <= R)) {
            make_tag(w);
            pushup();
            return ;
        } else {
            //pushdown();
            ls->update(L, R, w);
            rs->update(L, R, w);
            pushup();
        }
    }
};
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    vector<Segment> s;
    int n;
    cin >> n;
    for(int i = 0;i < n;i ++){
        int xa,ya,xb,yb;
        cin >> xa >> ya >> xb >> yb;
        x.push_back(xa);
        x.push_back(xb);
        s.push_back({xa,xb,ya,1});
        s.push_back({xa,xb,yb,-1});
    }
    sort(s.begin(),s.end());
    sort(x.begin(),x.end());
    x.erase(unique(x.begin(),x.end()),x.end());
    int N = x.size();
    SegTree Seg(1,N - 1);
    ll ans = 0;
    if(s.size()){
        Seg.update(s[0].l,s[0].r,s[0].add);
        for(int i = 1;i < s.size();i ++){
            ans += 1ll * Seg.len * (s[i].h - s[i - 1].h);
            Seg.update(s[i].l,s[i].r,s[i].add);
        }
    }
```

```
    cout << ans << "\n";
    return 0;
}
```

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 $V - val_A$）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。

```
#include <array>
#include <iostream>
#include <algorithm>

typedef long long int ll;

const int maxn = 500005;

ll a[maxn];

const ll inf = 0x3f3f3f3f3f3f3f3fll;

struct Node {
  Node *ls, *rs;
  int l, r, maxCnt;
  ll v, add, maxAdd, sum, maxV, maxHistory;

  Node(const int L, const int R) :
      ls(nullptr), rs(nullptr), l(L), r(R), maxCnt(0),
      v(0), add(0), maxAdd(0), sum(0), maxV(-inf), maxHistory(-inf) {}

  inline bool inRange(const int L, const int R) {
    return L <= l && r <= R;
  }
  inline bool outRange(const int L, const int R) {
    return l > R || L > r;
  }

  void addVal(const ll t, int len) {
    add += t;
    sum += len * t;
    maxV += t;
  }

  void makeAdd(const ll t, int len) {
    addVal(t, len);
    maxHistory = std::max(maxHistory, maxV);
    maxAdd = std::max(maxAdd, add);
  }
};

void pushup(Node *x, Node *y) {
  y->maxV = std::max(y->ls->maxV, y->rs->maxV);
  y->sum = y->ls->sum + y->rs->sum;
  y->maxHistory = std::max({y->maxHistory, y->ls->maxHistory, y->rs->maxHistory});
  if (x->ls->maxV != x->rs->maxV) {
    bool flag = x->ls->maxV < x->rs->maxV;
    if (flag) std::swap(x->ls, x->rs);
    x->maxV = x->ls->maxV;
    x->maxCnt = x->ls->maxCnt;
    y->maxV = std::max(y->maxV, x->rs->maxV);
    y->sum += x->rs->sum;
    x->sum = x->ls->sum;
```

```
    if (flag) std::swap(x->ls, x->rs);
  } else {
    x->maxCnt = x->ls->maxCnt + x->rs->maxCnt;
    x->sum = x->ls->sum + x->rs->sum;
    x->maxV = x->ls->maxV;
  }
  x->maxHistory = std::max({x->ls->maxHistory, x->rs->maxHistory, x->maxHistory, y->maxHistory});
}

void New(Node *&u1, Node *&u2, int L, int R) {
  u1 = new Node(L, R);
  u2 = new Node(L, R);
  if (L == R) {
    u1->v = u1->sum = u1->maxV = u1->maxHistory = a[L];
    u1->maxCnt = 1;
  } else {
    int M = (L + R) >> 1;
    New(u1->ls, u2->ls, L, M);
    New(u1->rs, u2->rs, M + 1, R);
    pushup(u1, u2);
  }
}

void pushdown(Node *x, Node *y) {
  ll val = std::max(x->ls->maxV, x->rs->maxV);
  std::array<Node*, 2> aim({y, x});
  Node *curl = aim[x->ls->maxV == val], *curr = aim[x->rs->maxV == val];
  x->ls->maxAdd = std::max(x->ls->maxAdd, x->ls->add + curl->maxAdd);
  x->ls->maxHistory = std::max(x->ls->maxHistory, x->ls->maxV + curl->maxAdd);
  x->ls->addVal(curl->add, x->ls->maxCnt);
  x->rs->maxAdd = std::max(x->rs->maxAdd, x->rs->add + curr->maxAdd);
  x->rs->maxHistory = std::max(x->rs->maxHistory, x->rs->maxV + curr->maxAdd);
  x->rs->addVal(curr->add, x->rs->maxCnt);
  y->ls->maxAdd = std::max(y->ls->maxAdd, y->ls->add + y->maxAdd);
  y->rs->maxAdd = std::max(y->rs->maxAdd, y->rs->add + y->maxAdd);
  y->ls->addVal(y->add, x->ls->r - x->ls->l + 1 - x->ls->maxCnt);
  y->rs->addVal(y->add, x->rs->r - x->rs->l + 1 - x->rs->maxCnt);
  x->add = y->add = x->maxAdd = y->maxAdd = 0;
}

void addV(Node *x, Node *y, int L, int R, ll k) {
  if (x->inRange(L, R)) {
    x->makeAdd(k, x->maxCnt);
    y->makeAdd(k, x->r - x->l + 1 - x->maxCnt);
  } else if (!x->outRange(L, R)) {
    pushdown(x, y);
    addV(x->ls, y->ls, L, R, k);
    addV(x->rs, y->rs, L, R, k);
    pushup(x, y);
  }
}

std::array<ll, 3> qry(Node *x, Node *y, const int L, const int R) {
  if (x->inRange(L, R)) return {x->sum + y->sum * ((x->r - x->l + 1) != x->maxCnt), x->maxV, x->maxHistory};
  else if (x->outRange(L, R)) return {0, -inf, -inf};
  else {
    pushdown(x, y);
    auto A = qry(x->ls, y->ls, L, R), B = qry(x->rs, y->rs, L, R);
    return {A[0] + B[0], std::max(A[1], B[1]), std::max(A[2], B[2])};
  }
}

void minV(Node *x, Node *y, const int L, const int R, int k) {
  if (x->maxV <= k) return;
  if (x->inRange(L, R) && y->maxV < k) {
    ll delta = k - x->maxV;
    x->makeAdd(delta, x->maxCnt);
  } else if (!x->outRange(L, R)) {
    pushdown(x, y);
    minV(x->ls, y->ls, L, R, k);
    minV(x->rs, y->rs, L, R, k);
```

```
    pushup(x, y);
  }
}

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  int n, m;
  std::cin >> n >> m;
  for (int i = 1; i <= n; ++i) std::cin >> a[i];
  Node *rot1, *rot2;
  New(rot1, rot2, 1, n);
  for (int op, l, r; m; --m) {
    std::cin >> op >> l >> r;
    if (op == 1) {
      std::cin >> op;
      addV(rot1, rot2, l, r, op);
    } else if (op == 2) {
      std::cin >> op;
      minV(rot1, rot2, l, r, op);
    } else {
      std::cout << qry(rot1, rot2, l, r)[op - 3] << '\n';
    }
  }
}
```

树状数组

```
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;
    Fenwick(int n) : n(n), a(n) {}
    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }
    T sum(int x) {
        T ans = 0;
        for (int i = x; i > 0; i -= i & -i) {
            ans += a[i - 1];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }
    int kth(T k) {
        int x = 0;
        // 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
        // 到最后找到的就是最大的那个 pos 并且对应的 <=x 的
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i - 1]) {
                x += i;
                k -= a[x - 1];
            }
        }
        return x;
    }//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
};
```

DSU

```
struct DSU {
    std::vector<int> f, siz;
    DSU(int n) : f(n), siz(n, 1) { std::iota(f.begin(), f.end(), 0); }
    int leader(int x) {
        while (x != f[x]) x = f[x] = f[f[x]];
        return x;
```

```
    }
    bool same(int x, int y) { return leader(x) == leader(y); }
    bool merge(int x, int y) {
        x = leader(x);
        y = leader(y);
        if (x == y) return false;
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) { return siz[leader(x)]; }
};
```

Splay

```
struct Node {
  int v, sz, sm;
  Node *ch[2], *fa;

  Node(const int V, Node *const f) : v(V), sz(1), sm(1), fa(f) {
    ch[0] = ch[1] = nullptr;
  }

  inline int GetRela(const int x) { return (v == x) ? -1 : (x > v); }

  void pushup() { sm = (ch[0] ? ch[0]->sm : 0) + (ch[1] ? ch[1]->sm : 0) + sz; }

  inline void rotate(const int x) {
    auto nrt = ch[x];
    ch[x] = nrt->ch[x ^ 1];
    nrt->ch[x ^ 1] = this;
    if (ch[x]) ch[x]->fa = this;
    nrt->fa = fa; fa = nrt;
    if (nrt->fa) nrt->fa->ch[nrt->fa->GetRela(nrt->v)] = nrt;
    pushup(); nrt->pushup();
  }

  void splay(const Node *p) {
    while (fa != p) {
      auto pa = fa->fa;
      if (pa == p) {
        fa->rotate(fa->GetRela(v));
      } else {
        int k1 = fa->GetRela(v), k2 = pa->GetRela(fa->v);
        if (k1 == k2) {
          pa->rotate(k1);
          fa->rotate(k1);
        } else {
          fa->rotate(k1);
          fa->rotate(k2);
        }
      }
    }
  }
};
```

LCT

```
struct Node {
  int v, s;
  bool tag;
  Node *ch[2], *fa;

  inline void maketag() {
    tag = !tag;
    std::swap(ch[0], ch[1]);
  }
  inline void pushup() {
    s = v;
    for (auto u : ch) if (u != nullptr) {
```

```
      s ^= u->s;
    }
  }
  inline void pushdown() {
    if (tag) {
      for (auto u : ch) if (u != nullptr) {
        u->maketag();
      }
      tag = false;
    }
  }

  inline int Getson() { return fa->ch[1] == this; }

  inline bool IsRoot() { return (fa == nullptr) || (fa->ch[Getson()] != this); }

  void rotate(const int x) {
    auto nt = ch[x];
    ch[x] = nt->ch[x ^ 1];
    nt->ch[x ^ 1] = this;
    if (ch[x]) ch[x]->fa = this;
    nt->fa = fa;
    if (!IsRoot()) { fa->ch[Getson()] = nt; }
    fa = nt;
    pushup(); nt->pushup();
  }

  void splay() {
    static Node* stk[maxn];
    int top = 0;
    stk[++top] = this;
    for (auto u = this; !u->IsRoot(); stk[++top] = u = u->fa);
    while (top) stk[top--]->pushdown();
    while (!IsRoot()) {
      if (fa->IsRoot()) {
        fa->rotate(Getson());
      } else {
        auto pa = fa->fa;
        int l1 = Getson(), l2 = fa->Getson();
        if (l1 == l2) {
          pa->rotate(l2);
          fa->rotate(l1);
        } else {
          fa->rotate(l1);
          fa->rotate(l2);
        }
      }
    }
  }
};
Node *node[maxn], Mem[maxn];

void Cut(const int x, const int y);
void Link(const int x, const int y);
void Query(const int x, const int y);
void Update(const int x, const int y);

void access(Node *u) {
  for (Node *v = nullptr; u; u = (v = u)->fa) {
    u->splay();
    u->ch[1] = v; u->pushup();
  }
}

void makeroot(Node *const u) {
  access(u);
  u->splay();
  u->maketag();
}

void Query(const int x, const int y) {
```

```
  auto u = node[x], v = node[y];
  makeroot(u);
  access(v);
  v->splay();
  qw(v->s, '\n');
}

void Link(const int x, const int y) {
  auto u = node[x], v = node[y];
  makeroot(u);
  access(v); v->splay();
  if (u->IsRoot() == false) return;
  u->fa = v;
}

void Cut(const int x, const int y) {
  auto u = node[x], v = node[y];
  makeroot(u); access(v); u->splay();
  if ((u->ch[1] != v) || (v->ch[0] != nullptr)) return;
  u->ch[1] = v->fa = nullptr;
  u->pushup();
}

// w[x] -> y
void Update(const int x, const int y) {
  auto u = node[x];
  u->splay();
  u->s ^= u->v;
  u->s ^= (u->v = a[x] = y);
}
```

珂朵莉树

```
auto getPos(int pos) {
  return --s.upper_bound({pos + 1, 0, 0});
}

void split(int pos) {
  auto it = getPos(pos);
  auto [l, r, v] = *it;
  s.erase(it);
  if (pos > l) s.insert({l, pos - 1, v});
  s.insert({pos, r, v});
}

void add(int l, int r, int v) {
  split(l); split(r + 1);
  for (auto x = getPos(l), y = getPos(r + 1); x != y; ++x) {
    x->v += v;
  }
}

void upd(int l, int r, int v) {
  split(l); split(r + 1);
  s.erase(getPos(l), getPos(r + 1));
  s.insert({l, r, v});
}
```

`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

```
constexpr long long INF = 1'000'000'000'000'000'000;
constexpr int C = 100'000;
struct Line {
    ll k,b;
    int id;
    Line(ll k,ll b,int id) : k(k), b(b), id(id) {}
```

```
};
long long f(const Line &line, int x) {
    return 1LL * line.k * x + line.b;
}
Line get(Line a, Line c, int x) {
    ll b = f(a,x), d = f(c,x);
    return b == d ? (a.id < c.id ? a : c) : b > d ? a : c;
}
struct Node {
    Node *lc, *rc;
    Line line;
    Node(const Line &line) : lc(nullptr), rc(nullptr), line(line) {}
};
void modify(Node *&p, int l, int r, Line line) {
    if (p == nullptr) {
        p = new Node(Line(0,-1e18,0));
    }
    if(l == r) {
        if(f(p -> line,l) <= f(line,l)) p -> line = line;
        return ;
    }
    int m = (l + r) / 2;
    if (f(p -> line, m) < f(line, m)) swap(p -> line, line);
    if (f(p -> line, l) < f(line, l)) modify(p -> lc, l, m, line);
    else if(f(p -> line, r) < f(line, r)) modify(p -> rc, m + 1, r, line);
}
Node *merge(Node *p, Node *q, int l, int r) {
    if (p == nullptr)
        return q;
    if (q == nullptr)
        return p;
    int m = (l + r) / 2;
    p -> lc = merge(p -> lc, q -> lc, l, m);
    p -> rc = merge(p -> rc, q -> rc, m, r);
    modify(p, l, r, q -> line);
    return p;
}
Line query(Node *p, int l, int r, int x) {
    if (p == nullptr)
        return Line(0,-1e18,0);
    if(l == r) return p -> line;
    int m = (l + r) / 2;
    if (x <= m) return get(query(p -> lc, l, m, x),p -> line,x);
    return get(query(p -> rc, m + 1, r, x),p -> line,x);
}
```

动态维护凸壳

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+m, and query maximum values at points x.
 *  Useful for dynamic programming.
 * Time: O(\log N)
 * Status: tested
 */

struct Line {
  mutable ll k, m, p;
  bool operator<(const Line &o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer: multiset<Line, less<>> {
  const ll inf = LLONG_MAX;
  ll val_offset = 0;
  void offset(ll x) {
    val_offset += x;//整体加
  }
```

```
  ll div(ll a, ll b) {
    return a / b - ((a^b) < 0 && a%b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) {
      x->p = inf;
      return 0;
    }
    if (x->k == y->k) {
      x->p = (x->m > y->m)? inf: -inf;
    } else {
      x->p = div(y->m - x->m, x->k - y->k);
    }
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m - val_offset, 0}), y = z++, x = y;//这里加减看情况
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m + val_offset;
  }
};

LineContainer* merge(LineContainer *S, LineContainer *T) {
  if (S->size() > T->size())
    swap(S, T);
  for (auto l: *S) {
    T->add(l.k, l.m + S->val_offset);
  }
  return T;
}
```

线段树合并

```
struct Info{
    int mx = 0,id = 0;
    Info() {}
    Info(int mx,int id) :mx(mx),id(id) {}
};
Info operator+(const Info a,const Info b) {
    if(a.mx > b.mx) return a;
    if(b.mx > a.mx) return b;
    if(a.id < b.id) return a;
    else return b;
}
struct Node {
    int l, r;
    Node *ls, *rs;
    Info t;
    Node(int l,int r) : l(l),r(r),ls(nullptr),rs(nullptr){}
};

void push_up(Node *&p) {
    if(p->ls == nullptr) {
        p->t = p->rs->t;return ;
    }
    if(p->rs == nullptr) {
        p->t = p->ls->t;return ;
    }
    p->t = p->ls->t + p->rs->t;
}
void upd(Node *&p,int l,int r,int x,int w) {
    if(p == nullptr) {
        p = new Node(l,r);
    }
    if(l == r) {
```

```
        p->t.mx += w;
        p->t.id = x;
        return ;
    }
    int mid = l + r >> 1;
    if(x <= mid) upd(p->ls,l,mid,x,w);
    else upd(p->rs,mid + 1,r,x,w);
    push_up(p);
}

Node* merge(Node *p,Node *q,int l,int r) {
    if(p == nullptr) return q;
    if(q == nullptr) return p;
    if(l == r) {
        p->t.mx += q->t.mx;
        return p;
    }
    int mid = l + r >> 1;
    p->ls = merge(p->ls,q->ls,l,mid);
    p->rs = merge(p->rs,q->rs,mid + 1,r);
    push_up(p);
    return p;
}
```

图论

树链剖分

```
// 重链剖分
void dfs1(int x) {
  son[x] = -1;
  siz[x] = 1;
  for (auto v:e[x])
    if (!dep[v]) {
      dep[v] = dep[x] + 1;
      fa[v] = x;
      dfs1(v);
      siz[x] += siz[v];
      if (son[x] == -1 || siz[v] > siz[son[x]]) son[x] = v;
    }
}

void dfs2(int x, int t) {
  top[x] = t;
  dfn[x] = ++ cnt;
  rnk[cnt] = x;
  if (son[x] == -1) return;
  dfs2(son[x], t);
  for (auto v:e[x])
    if (v != son[x] && v != fa[x]) dfs2(v, v);
}
int lca(int u, int v) {
  while (top[u] != top[v]) {
    if (dep[top[u]] > dep[top[v]])
      u = fa[top[u]];
    else
      v = fa[top[v]];
  }
  return dep[u] > dep[v] ? v : u;
}
```

LCA

倍增求 LCA

```
void dfs(int x){
    for(int j = 1;j <= 19;j ++){
        f[x][j] = f[f[x][j - 1]][j - 1];
    }
```

```
    for(auto v:e[x]){
        if(v == f[x][0])continue;
        f[v][0] = x;
        dep[v] = dep[x] + 1;
        dfs(v);
    }
}
int lca(int u,int v){
    if(dep[u] < dep[v])swap(u,v);
    for(int i = 0;i <= 19;i ++){
        if((dep[u] - dep[v]) & (1 << i))u = f[u][i];
    }
    if(u == v)return u;
    for(int j = 19;j >= 0; j--){
        if(f[u][j] != f[v][j]){
            u = f[u][j];
            v = f[v][j];
        }
    }
    return f[u][0];
}
int kth(int x,int k){
    for(int i = 0;i <= 19;i ++){
        if(k & (1 << i))x = f[x][i];
    }
    return x;
}
```

dfn 求 LCA

```
int get(int x, int y) {return dfn[x] < dfn[y] ? x : y;}
void dfs(int id, int f) {
  mi[0][dfn[id] = ++dn] = f;
  for(int it : e[id]) if(it != f) dfs(it, id);
}
int lca(int u, int v) {
  if(u == v) return u;
  if((u = dfn[u]) > (v = dfn[v])) swap(u, v);
  int d = __lg(v - u++);
  return get(mi[d][u], mi[d][v - (1 << d) + 1]);
}
dfs(R, 0);
for(int i = 1; i <= __lg(n); i++)
  for(int j = 1; j + (1 << i) - 1 <= n; j++)
    mi[i][j] = get(mi[i - 1][j], mi[i - 1][j + (1 << i - 1)]);
```

树哈希

```
typedef unsigned long long ull;
struct TreeHash{
    std::vector<int> hs;
    TreeHash(int n){
        hs.resize(n,0);
    }
    mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
    ull bas = rnd();
    ull H(ull x){
        return x*x*x*19890535+19260817;
    }
    ull F(ull x){
        return H(x & ((1ll << 32) - 1)) + H(x >> 32);
    }
    int flag,n;
    void dfs(int u,int fa){
        hs[u] = bas;
        for(auto v:e[u]){
            if(v == fa) continue;
            dfs(v,u);
            hs[u] += F(hs[v]);
        }
```

```
    }
};
```

虚树

```
void build_virtual_tree(vector<int> &h) {
  vector<int> a;
  sort(h.begin(), h.end(),[&](int &a,int &b){
      return dfn[a] < dfn[b];
  });  // 把关键点按照 dfn 序排序
  for (int i = 0; i < h.size(); ++i) {
    a.push_back(h[i]);
    if(i + 1 != h.size())a.push_back(lca(h[i], h[i + 1]));  // 插入 lca
  }
  sort(a.begin(), a.end(), [&](int &a,int &b){
      return dfn[a] < dfn[b];
  });  // 把所有虚树上的点按照 dfn 序排序
  a.erase(unique(a.begin(),a.end()),a.end());
  for (int i = 0; i < a.size() - 1; ++i) {
    int lc = lca(a[i], a[i + 1]);
    add(lc, a[i + 1]);  // 连边, 如有边权 就是 distance(lc,a[i+1])
  }
}
```

最小环

```
//floyd 找最小环
//dijkstra 暴力删边跑最短路-
int floyd(const int &n) {
  for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= n; ++j)
      dis[i][j] = f[i][j]; // 初始化最短路矩阵
  int ans = inf;
  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i < k; ++i)
      for (int j = 1; j < i; ++j)
        ans = std::min(ans, dis[i][j] + f[i][k] + f[k][j]); // 更新答案
    for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
        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
  }
  return ans;
}
```

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

```
bool spfa(){
    queue<int> q;
    vector<int> vis(n + 1),cnt(n + 1),dis(n + 1,1e9);
    dis[1] = 0;
    cnt[1] = 1;
    q.push(1);
    while(!q.empty()){
        int u = q.front();
        q.pop();
        vis[u] = 0;
        if(cnt[u] >= n)return 1;
        for(auto v:e[u]){
            if(dis[v] > dis[u] + len[p]){
                dis[v] = dis[u] + len[p];
                if(vis[v] == 0){
                    vis[v] = 1;
                    q.push(v);
                    cnt[v] ++;
                }
            }
        }
```

```
    }
    return 0;
}
```

最大流

```
struct Flow {
    static constexpr int INF = 1e9;
    int n;
    struct Edge {
        int to, cap;
        Edge(int to, int cap) : to(to), cap(cap) {}
    };
    vector<Edge> e;
    vector<vector<int>> g;
    vector<int> cur, h;
    Flow(int n) : n(n), g(n) {}
    void init(int n) {
        for (int i = 0; i < n; i++) g[i].clear();
        e.clear();
      }
    bool bfs(int s, int t) {
        h.assign(n, -1);
        queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            int u = que.front();
            que.pop();
            for (int i : g[u]) {
                int v = e[i].to;
                int c = e[i].cap;
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t)
                        return true;
                    que.push(v);
                }
            }
        }
        return false;
    }
    int dfs(int u, int t, int f) {
        if (u == t)
            return f;
        int r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            int j = g[u][i];
            int v = e[j].to;
            int c = e[j].cap;
            if (c > 0 && h[v] == h[u] + 1) {
                int a = dfs(v, t, std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0)
                    return f;
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, int c) {
        g[u].push_back(e.size());
        e.push_back({v, c});
        g[v].push_back(e.size());
        e.push_back({u, 0});
    }
    int maxFlow(int s, int t) {
        int ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
```

```
            ans += dfs(s, t, INF);
        }
        return ans;
    }
};
```

最小费用最大流

```
using i64 = long long;
struct MCFGraph {
    struct Edge {
        int v, c, f;
        Edge(int v, int c, int f) : v(v), c(c), f(f) {}
    };
    const int n;
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<i64> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<i64>::max());
        pre.assign(n, -1);
        priority_queue<pair<i64, int>, vector<pair<i64, int>>, greater<pair<i64, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            i64 d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] < d) continue;
            for (int i : g[u]) {
                int v = e[i].v;
                int c = e[i].c;
                int f = e[i].f;
                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
                    dis[v] = d + h[u] - h[v] + f;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<i64>::max();
    }
    MCFGraph(int n) : n(n), g(n) {}
    void addEdge(int u, int v, int c, int f) {
        if (f < 0) {
            g[u].push_back(e.size());
            e.emplace_back(v, 0, f);
            g[v].push_back(e.size());
            e.emplace_back(u, c, -f);
        } else {
            g[u].push_back(e.size());
            e.emplace_back(v, c, f);
            g[v].push_back(e.size());
            e.emplace_back(u, 0, -f);
        }
    }
    std::pair<int, i64> flow(int s, int t) {
        int flow = 0;
        i64 cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) h[i] += dis[i];
            int aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = std::min(aug, e[pre[i]].c);
            for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
                e[pre[i]].c -= aug;
                e[pre[i] ^ 1].c += aug;
            }
            flow += aug;
            cost += i64(aug) * h[t];
```

```
        }
        return std::make_pair(flow, cost);
    }
};
```

```
const int N = 5e3 + 5, M = 5e4 + 5;
struct flow {
  int cnt = 1, hd[N], nxt[M << 1], to[M << 1], limit[M << 1], cst[M << 1];
  void add(int u, int v, int w, int c) {
    nxt[++cnt] = hd[u], hd[u] = cnt, to[cnt] = v, limit[cnt] = w, cst[cnt] = c;
    nxt[++cnt] = hd[v], hd[v] = cnt, to[cnt] = u, limit[cnt] = 0, cst[cnt] = -c;
  }
  int fr[N], fl[N], in[N], dis[N];
  pair<int, int> mincost(int s, int t) {
    int flow = 0, cost = 0;
    while(1) {
      queue<int> q;
      memset(dis, 0x3f, sizeof(dis));
      memset(in, 0, sizeof(in));
      fl[s] = 1e9, dis[s] = 0, q.push(s);
      while(!q.empty()) {
        int t = q.front();
        q.pop(), in[t] = 0;
        for(int i = hd[t]; i; i = nxt[i]) {
          int it = to[i], d = dis[t] + cst[i];
          if(limit[i] && d < dis[it]) {
            fl[it] = min(limit[i], fl[t]), fr[it] = i, dis[it] = d;
            if(!in[it]) in[it] = 1, q.push(it);
          }
        }
      }
      if(dis[t] > 1e9) return make_pair(flow, cost);
      flow += fl[t], cost += dis[t] * fl[t];
      for(int u = t; u != s; u = to[fr[u] ^ 1]) limit[fr[u]] -= fl[t], limit[fr[u] ^ 1] += fl[t];
    }
  }
} g;
```

二分图最大匹配

```
auto dfs = [&](auto &&dfs, int u, int tag) -> bool {
    if (vistime[u] == tag) return false;
    vistime[u] = tag;
    for (auto v : e[u]) if (!mtch[v] || dfs(dfs, mtch[v], tag)) {
      mtch[v] = u;
      return true;
    }
    return false;
 };
```

KM(二分图最大权匹配)

```
template <typename T>
struct hungarian {  // km
  int n;
  vector<int> matchx;  // 左集合对应的匹配点
  vector<int> matchy;  // 右集合对应的匹配点
  vector<int> pre;     // 连接右集合的左点
  vector<bool> visx;   // 拜访数组 左
  vector<bool> visy;   // 拜访数组 右
  vector<T> lx;
  vector<T> ly;
  vector<vector<T> > g;
  vector<T> slack;
  T inf;
  T res;
  queue<int> q;
  int org_n;
  int org_m;

```

```
  hungarian(int _n, int _m) {
    org_n = _n;
    org_m = _m;
    n = max(_n, _m);
    inf = numeric_limits<T>::max();
    res = 0;
    g = vector<vector<T> >(n, vector<T>(n));
    matchx = vector<int>(n, -1);
    matchy = vector<int>(n, -1);
    pre = vector<int>(n);
    visx = vector<bool>(n);
    visy = vector<bool>(n);
    lx = vector<T>(n, -inf);
    ly = vector<T>(n);
    slack = vector<T>(n);
  }

  void addEdge(int u, int v, int w) {
    g[u][v] = max(w, 0);  // 负值还不如不匹配 因此设为 0 不影响
  }

  bool check(int v) {
    visy[v] = true;
    if (matchy[v] != -1) {
      q.push(matchy[v]);
      visx[matchy[v]] = true;  // in S
      return false;
    }
    // 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
    while (v != -1) {
      matchy[v] = pre[v];
      swap(v, matchx[pre[v]]);
    }
    return true;
  }

  void bfs(int i) {
    while (!q.empty()) {
      q.pop();
    }
    q.push(i);
    visx[i] = true;
    while (true) {
      while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v = 0; v < n; v++) {
          if (!visy[v]) {
            T delta = lx[u] + ly[v] - g[u][v];
            if (slack[v] >= delta) {
              pre[v] = u;
              if (delta) {
                slack[v] = delta;
              } else if (check(v)) {  // delta=0 代表有机会加入相等子图 找增广路
                                      // 找到就 return 重建交错树
                return;
              }
            }
          }
        }
      }
      // 没有增广路 修改顶标
      T a = inf;
      for (int j = 0; j < n; j++) {
        if (!visy[j]) {
          a = min(a, slack[j]);
        }
      }
      for (int j = 0; j < n; j++) {
        if (visx[j]) {  // S
          lx[j] -= a;
```

```
        }
        if (visy[j]) {  // T
          ly[j] += a;
        } else {  // T'
          slack[j] -= a;
        }
      }
      for (int j = 0; j < n; j++) {
        if (!visy[j] && slack[j] == 0 && check(j)) {
          return;
        }
      }
    }
  }

  void solve() {
    // 初始顶标
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        lx[i] = max(lx[i], g[i][j]);
      }
    }

    for (int i = 0; i < n; i++) {
      fill(slack.begin(), slack.end(), inf);
      fill(visx.begin(), visx.end(), false);
      fill(visy.begin(), visy.end(), false);
      bfs(i);
    }

    // custom
    for (int i = 0; i < n; i++) {
      if (g[i][matchx[i]] > 0) {
        res += g[i][matchx[i]];
      } else {
        matchx[i] = -1;
      }
    }
    cout << res << "\n";
    for (int i = 0; i < org_n; i++) {
      cout << matchx[i] + 1 << " ";
    }
    cout << "\n";
  }
};
```

一般图最大匹配

```
#include <bits/stdc++.h>
struct Graph {
    int n;
    std::vector<std::vector<int>> e;
    Graph(int n) : n(n), e(n + 1) {}
    void addEdge(int u, int v) {
        e[u].push_back(v);
        e[v].push_back(u);
    }
    std::vector<int> findMatching() {
        std::vector<int> match(n + 1, -1), vis(n + 1), link(n + 1), f(n + 1), dep(n + 1);
        // disjoint set union
        auto find = [&](int u) {
            while (f[u] != u)
                u = f[u] = f[f[u]];
            return u;
        };
        auto lca = [&](int u, int v) {
            u = find(u);
            v = find(v);
            while (u != v) {
                if (dep[u] < dep[v])
                    std::swap(u, v);
```

```
            u = find(link[match[u]]);
        }
        return u;
    };
    std::queue<int> q;
    auto blossom = [&](int u, int v, int p) {
        while (find(u) != p) {
            link[u] = v;
            v = match[u];
            if (vis[v] == 0) {
                vis[v] = 1;
                q.push(v);
            }
            f[u] = f[v] = p;
            u = link[v];
        }
    };
    // find an augmenting path starting from u and augment (if exist)
    auto augment = [&](int u) {
        while (!q.empty())
            q.pop();
        std::iota(f.begin(), f.end(), 0);
        // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
        std::fill(vis.begin(), vis.end(), -1);

        q.push(u);
        vis[u] = 1;
        dep[u] = 0;

        while (!q.empty()){
            int u = q.front();
            q.pop();
            for (auto v : e[u]) {
                if (vis[v] == -1) {
                    vis[v] = 0;
                    link[v] = u;
                    dep[v] = dep[u] + 1;
                    // found an augmenting path
                    if (match[v] == -1) {
                        for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]){
                            temp = match[y];
                            match[x] = y;
                            match[y] = x;
                        }
                        return;
                    }
                    vis[match[v]] = 1;
                    dep[match[v]] = dep[u] + 2;
                    q.push(match[v]);

                } else if (vis[v] == 1 && find(v) != find(u)) {
                    // found a blossom
                    int p = lca(u, v);
                    blossom(u, v, p);
                    blossom(v, u, p);
                }
            }
        }

    };
    // find a maximal matching greedily (decrease constant)
    auto greedy = [&]() {
        for (int u = 1; u <= n; ++u) {
            if (match[u] != -1)
                continue;
            for (auto v : e[u]) {
                if (match[v] == -1) {
                    match[u] = v;
                    match[v] = u;
                    break;
                }
```

```
                }
            }
        };
        greedy();
        for (int u = 1; u <= n; ++u)
            if (match[u] == -1)
                augment(u);

        return match;
    }
};
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n, m;
    std::cin >> n >> m;
    Graph g(n);
    for (int i = 0; i < m; ++i) {
        int u, v;
        std::cin >> u >> v;
        g.addEdge(u, v);
    }
    auto match = g.findMatching();
    int ans = 0;
    for (int u = 1; u <= n; ++u)
        if (match[u] != -1)
            ++ans;
    std::cout << ans / 2 << "\n";
    for (int u = 1; u <= n; ++u)
        if(match[u] != -1)std::cout << match[u] << " ";
        else std::cout << 0 << " ";
    return 0;
}
```

缩点 SCC

```
void dfs(const int u) {
  low[u] = dfn[u] = ++cnt;
  ins[stk[++top] = u] = true;
  for (auto v : e[u]) if (dfn[v] == 0) {
    dfs(v);
    low[u] = std::min(low[u], low[v]);
  } else if (ins[v]) {
    low[u] = std::min(low[u], dfn[v]);
  }
  if (low[u] == dfn[u]) {
    ++scnt; int v;
    do {
      ins[v = stk[top--]] = false;
      w[bel[v] = scnt] += a[v];
    } while (u != v);
  }
}
```

割点与桥

```
//割点
void tarjan(int u, int fa){
    dfn[u] = low[u] = ++cnt; int du = 0;
    for(for v:e[x]){
        if(v == fa) continue;
        if(!dfn[v]){ ++du;
            tarjan(v, u); low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u] && fa) vis[u] = 1;
        }
        else low[u] = min(low[u], dfn[v]);
    }
    if(!fa && du > 1) vis[u] = 1;
}
//桥
```

```cpp
void tarjan(int u, int fa) {
  f[u] = fa;
  low[u] = dfn[u] = ++cnt;
  for (auto v:e[u]) {
    if (!dfn[v]) {
      tarjan(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] > dfn[u]) {
        isbridge[v] = true;
        ++cnt_bridge;
      }
    } else if (dfn[v] < dfn[u] && v != fa) {
      low[u] = min(low[u], dfn[v]);
    }
  }
}
```

边双缩点

```cpp
void form(int x){
    std::vector<int> tmp;
    int now = 0;
    do{
        now = s[top --];
        tmp.push_back(now);
    }while(now != x);
    ans.push_back(tmp);
}
void tarjan(int x,int now){
    dfn[x] = low[x] = ++cnt;
    s[++ top] = x;
    for(auto [v,_]:e[x]){
        if(_ == now)continue;
        if(!dfn[v]){
            tarjan(v,_);
            low[x] = min(low[x],low[v]);
            if(low[v] > dfn[x]){
                form(v);
            }

        }else low[x] = min(low[x],dfn[v]);
    }
}
for(int i = 1;i <= n;i ++){
    if(dfn[i] == 0){
        tarjan(i,0);
        form(i);
    }
}
cout << ans.size() << "\n";
for(auto A:ans){
    cout << A.size() << " ";
    for(auto x:A){
        cout << x << " ";
    }cout << "\n";
}
```

圆方树

```cpp
void dfs(int u) {
    static int cnt = 0;
    dfn[u] = low[u] = ++cnt;
    for (auto [v,w]:e[u]) {
        if (v == fa[u]) continue;
        if (!dfn[v]) {
            fa[v] = u; fr[v] = w;
            dfs(v); low[u] = min(low[u], low[v]);
        }
        else low[u] = min(low[u], dfn[v]);
        if (low[v] > dfn[u]) add(u, v, w); // 圆 - 圆
```

```
    }
    for (auto [v,w]:e[u]) {
        if (u == fa[v] || dfn[v] < dfn[u]) continue;
        add(u, v, w); // 圆 - 方
    }
}
```

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

```
void dfs(int x) {
    stk.push_back(x);
    dfn[x] = low[x] = cur++;

    for (auto y : adj[x]) {
        if (dfn[y] == -1) {
            dfs(y);
            low[x] = std::min(low[x], low[y]);
            if (low[y] == dfn[x]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    edges.emplace_back(n + cnt, v);
                } while (v != y);
                edges.emplace_back(x, n + cnt);
                cnt++;
            }
        } else {
            low[x] = std::min(low[x], dfn[y]);
        }
    }
}
```

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

```
//三元环
  for (int u, v; m; --m) {
    u = A[m]; v = B[m];
    if (d[u] > d[v]) {
      std::swap(u, v);
    } else if ((d[u] == d[v]) && (u > v)) {
      std::swap(u, v);
    }
    e[u].push_back(v);
  }
  for (int u = 1; u <= n; ++u) {
    for (auto v : e[u]) vis[v] = u;
    for (auto v : e[u]) {
      for (auto w : e[v]) if (vis[w] == u) {
        ++ans;
      }
```

```
    }
  }
 // 四元环
  auto cmp = [&](int &a,int &b){
      if(d[a] != d[b])return d[a] > d[b];
      else return a < b;
  }
  for(int u = 1;u <= n;++ u) {
      for(auto v: G[u])//G 为原图
          for(auto w: e[v])
              if(cmp(u,w)) (ans += vis[w] ++)%=MOD;
      for(auto v: G[u])
          for(auto w: e[v])
              if(cmp(u,w)) vis[w] = 0;
  }
```

字符串

manacher

```
struct Manacher {
    int n, l, f[maxn * 2], Len;
    char s[maxn * 2];

    void init(char *c) {
        l = strlen(c + 1); s[0] = '~';
        for (int i = 1, j = 2; i <= l; ++i, j += 2)
            s[j] = c[i], s[j - 1] = '#';
        n = 2 * l + 1; s[n] = '#'; s[n + 1] = '\0';
    }
    void manacher() {
        int p = 0, mr = 0;
        for (int i = 1; i <= n; ++i) f[i] = 0;
        for (int i = 1; i <= n; ++i) {
            if (i < mr) f[i] = min(f[2 * p - i], mr - i);
            while (s[i + f[i]] == s[i - f[i]]) ++f[i]; --f[i];
            if (f[i] + i > mr) mr = i + f[i], p = i;
            Len = max(Len, f[i]);
        }
    }

    void solve() {
        for (int i = 1; i <= n; ++i) {
            // [1, l]
            int L = i - f[i] + 1 >> 1, R = i + f[i] - 1 >> 1;
            if (!f[i]) continue;

            // [1, 2 * l + 1]
            L = i - f[i], R = i + f[i];
        }
    }
} 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}

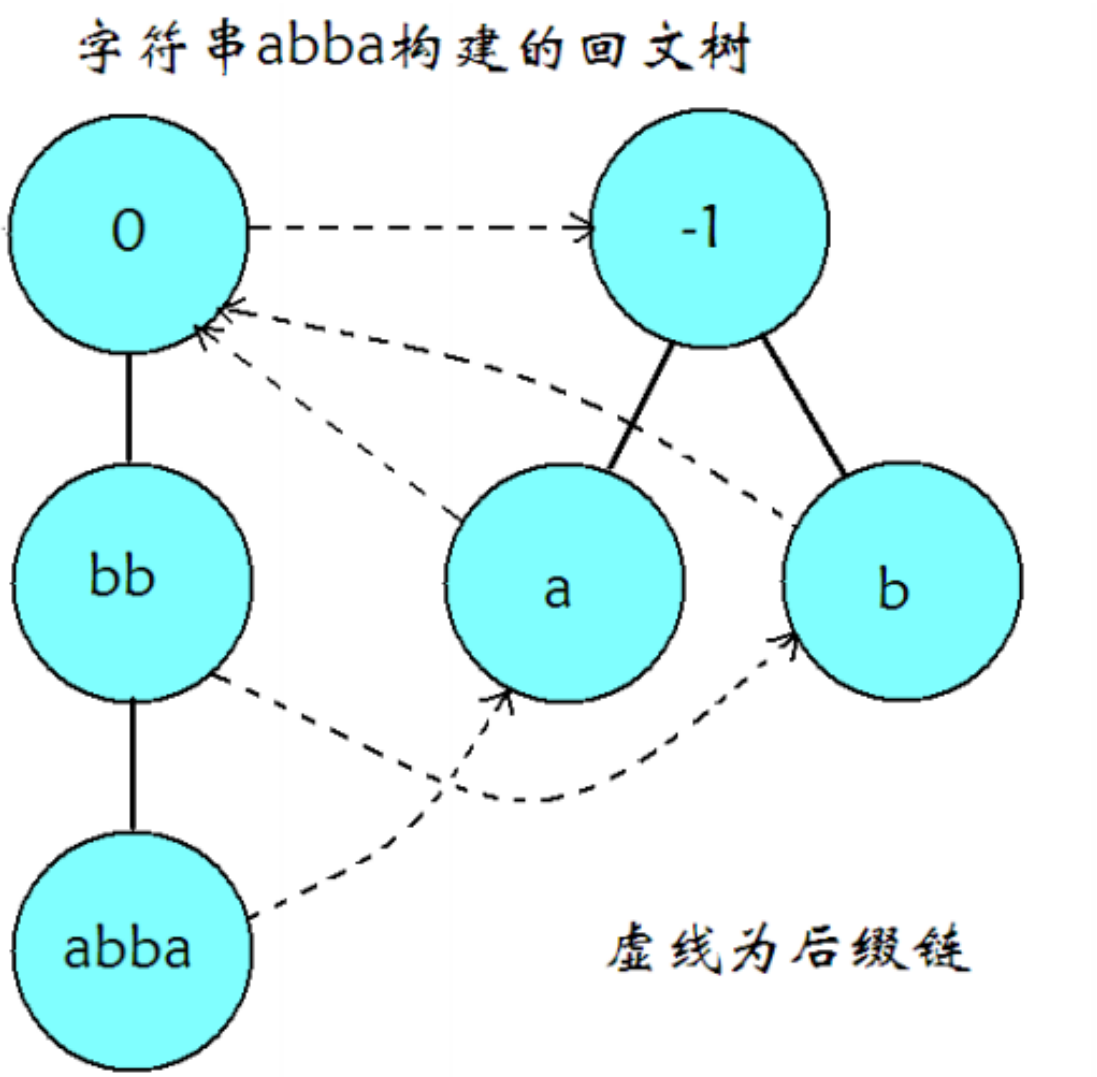
```
const int maxn = 1000005;

int sa[maxn], rnk[maxn], tax[maxn], tp[maxn], height[maxn];
void SA(string s) {
    int n = s.size();
    s = '#' + s;
    m = SIGMA_SIZE;
```

```
    vector<int> S(n + 1);
    auto RadixSort = [&]() {
        for (int i = 0; i <= m; ++i) tax[i] = 0;
        for (int i = 1; i <= n; ++i) ++tax[rnk[i]];
        for (int i = 1; i <= m; ++i) tax[i] += tax[i - 1];
        for (int i = n; i; --i) sa[tax[rnk[tp[i]]]--] = tp[i];
    };
    for (int i = 1; i <= n; ++i) {
        S[i] = s[i] - '0';
        tp[i] = i;
        rnk[i] = S[i];
    }
    RadixSort();
    for (int len = 1, p = 0; p != n; m = p, len <<= 1) {
        p = 0;
        for (int i = n - len + 1; i <= n; ++i) tp[++p] = i;
        for (int i = 1; i <= n; ++i) if (sa[i] > len) tp[++p] = sa[i] - len;
        RadixSort();
        std::swap(rnk, tp);
        p = 0;
        for (int i = 1; i <= n; ++i)
          rnk[sa[i]] = ((tp[sa[i]] == tp[sa[i-1]]) && (tp[sa[i] + len] == tp[sa[i-1] + len])) ? p : ++p;
    }
    for (int i = 1, p = 0; i <= n; ++i) {
        int pre = sa[rnk[i] - 1];
        if (p) --p;
        while (S[pre + p] == S[i + p]) ++p;
        h[0][rnk[i]] = height[rnk[i]] = p;
    }
    for (int i = 1; i <= 20; ++i) {
        memset(h[i], 0x3f, n * 4 + 4);
        for (int j = 1; j + (1 << i - 1) <= n; ++j)
            h[i][j] = min(h[i - 1][j], h[i - 1][j + (1 << i - 1)]);
    }
}
int Q(int l, int r) {
    if (l > r) swap(l, r);
    ++l;
    int k = __lg(r - l + 1);
    return min(h[k][l], h[k][r - (1 << k) + 1]);
}
int lcp(int i, int j) {
    if (i == j) return n - i + 1;
    return Q(rnk[i], rnk[j]);
}
```

PAM

转移边表示的是在原回文串的两边各加一个字符，得到长度加 2 的新回文串；`fail` 指针则指向该回文串的最长回文后缀。和其他自动机有所不同，它有两个根节点，分别代表长度为偶数的串和长度为奇数的串。它们的长度分别为 0 和 -1（注意不是 1，为了添加 2 的长度可以得到长为 1 的回文串），以下分别称为奇根和偶根。值得注意的是，偶根的 fail 指针指向的是奇根，而奇根的 `fail` 并不需在意，它的儿子中总会有长为 1 的回文串，因而不可能会失配。

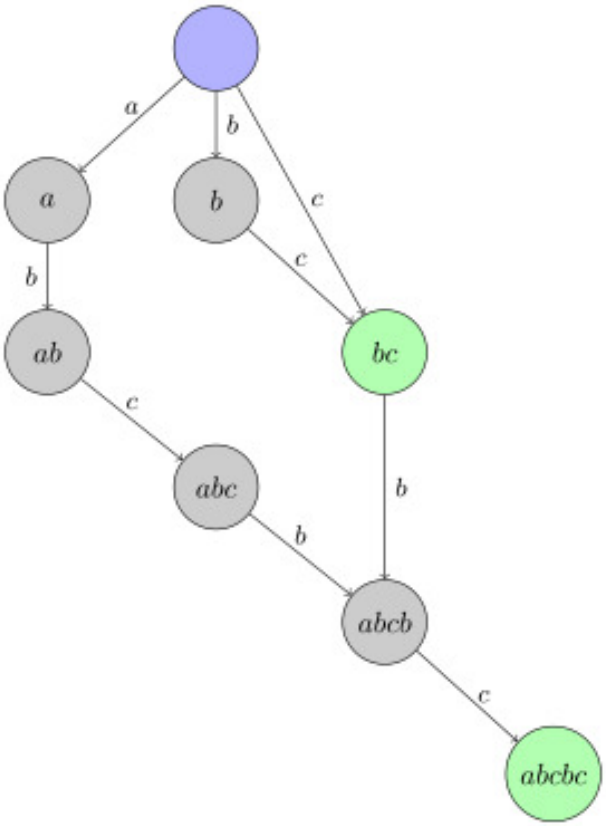


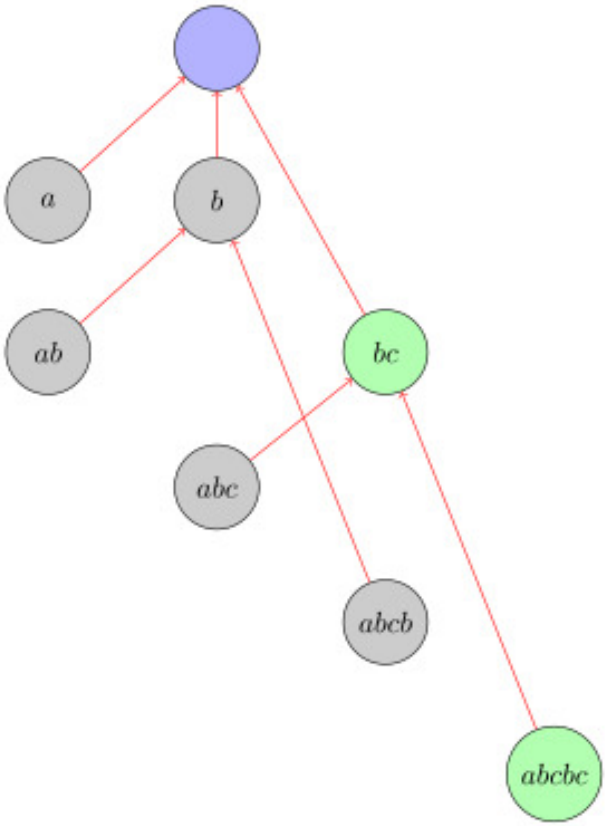

```cpp
struct PAM {
    static constexpr int ALPHABET_SIZE = 28;
    struct Node {
        int len; // 当前节点最长回文长度
        int fail;// 该回文串的最长回文后缀
        int scnt; // 当前节点表示的回文后缀的本质不同回文串个数
        int pcnt; // 当前节点回文串在字符串中出现次数，每个点代表一个不同的回文串
        std::array<int, ALPHABET_SIZE> next; // 转移边
        Node() : len{}, fail{}, scnt{}, next{}, pcnt{} {}
    };
    std::vector<Node> t;
    int last;
    std::string s;
    PAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[1].len = -1;
        last = 0;
        t[0].fail = 1;
        s = "$";
    }
    int newNode() {
        t.emplace_back();
```

```
        return t.size() - 1;
    }
    int get_fail(int x) {
        int pos = s.size() - 1;
        while(s[pos - t[x].len - 1] != s[pos]) x = t[x].fail;
        return x;
    }
    void add(char c, char offset = 'a') {
        s += c;
        int let = c - offset;
        int x = get_fail(last);
        if (!t[x].next[let]) {
            int now = newNode();
            t[now].len = t[x].len + 2;
            t[now].fail = t[get_fail(t[x].fail)].next[let];
            t[x].next[let] = now;
            t[now].scnt = t[t[now].fail].scnt + 1;
        }
        last = t[x].next[let];
        t[last].pcnt ++;
    }
};
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    string s;
    cin >> s;
    PAM pam;
    pam.init();
    for(int i = 0;i < s.size();i ++) {
        pam.add(s[i]);
        int ans = pam.t[pam.last].scnt;
        cout << ans << " ";
        if(i + 1 != s.size()) {
            s[i + 1] = (s[i + 1] - 97 + ans) % 26 + 97;
        }
    }
}
```

SAM

`fa` 为 parent 树上的父亲，`nxt` 为自动机上的指向。






```cpp
struct SAM {
    static constexpr int ALPHABET_SIZE = 26,rt = 1;
    struct Node {
        int len,fa,siz;
        std::array<int, ALPHABET_SIZE> nxt;
        Node() : len{}, fa{}, siz{}, nxt{} {}
    };
    std::vector<Node> t;
    SAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int getfa(int x){
        return t[x].fa;
    }
    int getlen(int x){
        return t[x].len;//表示该状态能够接受的最长的字符串长度。
    }
    int size(){
        return t.size();
    }
    int extend(int p, int ch) {
        int np = newNode();
        t[np].len = t[p].len + 1;t[np].siz = 1;
        while(p && !t[p].nxt[ch])t[p].nxt[ch] = np,p = t[p].fa;
        if(!p){t[np].fa = rt;return np;}
        int q = t[p].nxt[ch];
        if(t[q].len == t[p].len + 1){
            t[np].fa = q;
        }else {
```

```
            int nq = newNode();t[nq].len = t[p].len + 1;t[nq].fa = t[q].fa;
            for(int i = 0;i < 26;i ++)t[nq].nxt[i] = t[q].nxt[i];
            while(p && t[p].nxt[ch] == q)t[p].nxt[ch] = nq,p = t[p].fa;
            t[np].fa = t[q].fa = nq;
        }
        return np;
    }
    int extend_(int p, int ch) {//广义
        if(t[p].nxt[ch]){
            int q = t[p].nxt[ch];
            if(t[q].len == t[p].len + 1)return q;
            int nq = newNode();t[nq].len = t[p].len + 1;t[nq].fa = t[q].fa;
            for(int i = 0;i < 26;i ++)t[nq].nxt[i] = t[q].nxt[i];
            while(p && t[p].nxt[ch] == q)t[p].nxt[ch] = nq,p = t[p].fa;
            t[q].fa = nq;return nq;
        }
        int np = newNode();
        t[np].len = t[p].len + 1;
        while(p && !t[p].nxt[ch])t[p].nxt[ch] = np,p = t[p].fa;
        if(!p){t[np].fa = rt;return np;}
        int q = t[p].nxt[ch];
        if(t[q].len == t[p].len + 1){
            t[np].fa = q;
        }else {
            int nq = newNode();t[nq].len = t[p].len + 1;t[nq].fa = t[q].fa;
            for(int i = 0;i < 26;i ++)t[nq].nxt[i] = t[q].nxt[i];
            while(p && t[p].nxt[ch] == q)t[p].nxt[ch] = nq,p = t[p].fa;
            t[np].fa = t[q].fa = nq;
        }
        return np;
    }
    void build(vector<vector<int>> &e){
        e.resize(t.size());
        for(int i = 2;i < t.size();i ++){
            e[t[i].fa].push_back(i);
        }
    }
};
int main(){
    string s;
    cin >> s;
    int n = s.size();
    SAM sam;
    vector<int> pos(n + 1);
    pos[0] = 1;
    for(int i = 0;i < n;i ++){
        pos[i + 1] = sam.extend(pos[i],s[i] - 'a');
    }
    std::vector<std::vector<int>> e;
    sam.build(e);
    long long ans = 0;
    auto dfs = [&](auto&& self,int x)->void{
        for(auto v:e[x]){
            self(self,v);
            sam.t[x].siz += sam.t[v].siz;
        }
        if(sam.t[x].siz != 1){
            ans = max(ans,1ll * sam.t[x].siz * sam.t[x].len);
        }
    };
    dfs(dfs,1);
    cout << ans << "\n";
}
```

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

```
    int p = 1,len = 0,ans = 0;
    std::vector<int> l(m),L(m);
```

```
    for(int i = 0;i < m;i ++){
        int ch = s[i] - 'a';
        if(sam.t[p].nxt[ch]){
            p = sam.t[p].nxt[ch];len ++;
        }else {
            while(p && sam.t[p].nxt[ch] == 0){
                p = sam.t[p].fa;
            }
            if(!p)p = 1,len = 0;
            else len = sam.t[p].len + 1,p = sam.t[p].nxt[ch];
        }//其中 p 为前缀最长能匹配到的后缀所在的节点
        l[i] = len;
        L[i] = i - len + 1;
    }
```

3. 广义 SAM

```
    int main(){
        SAM sam;
        int n;
        cin >> n;
        std::vector<std::vector<int>> pos(n);
        for(int i = 0;i < n;i ++){
            string s;
            cin >> s;
            pos[i].resize(s.size() + 1);
            pos[i][0] = 1;
            for(int j = 0;j < s.size();j ++){
                pos[i][j + 1] = sam.extend_(pos[i][j],s[j] - 'a');
            }
        }
        ll ans = 0;
        for(int i = 2;i < sam.t.size();i ++){
            ans += sam.getlen(i) - sam.getlen(sam.getfa(i));
        }
        cout << ans << "\n";
        cout << sam.t.size() - 1 << "\n";
    }
```

parent 树上每个节点维护了一个区间，若 p 是 q 的父节点则有 maxp = minq - 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

AC 自动机的失配指针指向所有模式串的前缀中匹配当前状态的最长后缀。

`fail` 树上 u 和 v 的 lca 为 u 和 v 的最长公共 border。

```
const int maxn = 2e5 + 7;
#define ch s[i] - 'a'
struct AC_automaton {
    int nxt[26], cnt, fail;
} T[maxn];
int tot = 1, rt = 1, id[maxn];
void insert(string &s, int k) {
    int now = rt, l = s.size();
    for (int i = 0; i < l; ++i) {
        if (!T[now].nxt[ch]) T[now].nxt[ch] = ++tot;
        now = T[now].nxt[ch];
    } id[k] = now;
}
void init_fail() { // Trie 图
    queue<int> q;
    for (int i = 0; i < 26; ++i) {
        int &u = T[rt].nxt[i];
        if (!u) { u = rt; continue; }
        T[u].fail = rt; q.push(u);
    }
    while (!q.empty()) {
```

```
        int u = q.front(); q.pop();
        for (int i = 0; i < 26; ++i) {
            int &v = T[u].nxt[i];
            if (!v) { v = T[T[u].fail].nxt[i]; continue; }
            T[v].fail = T[T[u].fail].nxt[i]; q.push(v);
        }
    }
}
int siz[maxn];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin >> n;
    for(int i = 0;i < n;i ++) {
        string t;
        cin >> t;
        insert(t,i);
    }
    init_fail();
    string s;
    cin >> s;
    for(int u = rt,i = 0;i < s.size();i ++) {
        u = T[u].nxt[ch];
        ++ siz[u];
    }
    vector<vector<int>> e(tot + 1);
    for(int i = 2;i <= tot;i ++) e[T[i].fail].push_back(i);
    auto dfs = [&](auto self,int x) -> void  {
        for(auto v : e[x]) {
            self(self,v);
            siz[x] += siz[v];
        }
    };
    dfs(dfs,1);
    for(int i = 0;i < n;i ++)  cout << siz[id[i]] << "\n";
}
```

KMP

```
struct KMP{
    string s2;// add '#'
    std::vector<int> nxt;
    int m;
    KMP(string y) :s2(y){
        m = s2.size() - 1;
        nxt.resize(m + 1,0);
        for(int i = 2,p = 0;i <= m;i ++){
            while(p && s2[i] != s2[p + 1])p = nxt[p];
            if(s2[i] == s2[p + 1])p ++;
            nxt[i] = p;
        }
    }
    void match(string s1){
        int n = s1.size() - 1;
        for(int i = 1,p = 0;i <= n;i ++){
            while(p && s1[i] != s2[p + 1])p = nxt[p];
            if(s1[i] == s2[p + 1]){
                p ++;
                if(p == m){
                    //cout<<i - m + 1<<endl;
                    p = nxt[p];
                }
            }
        }
    }
    std::vector<int> find_border(){
        std::vector<int> v;
        for(int i = nxt[m];i;i = nxt[i])v.push_back(i);
        return v;
    }// 找该串所有的周期
```

```
    std::vector<int> calc_prefixes(){
        std::vector<int> cnt(m + 1,1);
        for(int i = m;i >= 1;i --)cnt[nxt[i]] += cnt[i];
        return cnt;
    }// 每个前缀出现次数
};
```

KMP 自动机

```
for(int i = 1, fail = 0; i <= n; i ++) {
    fail = nxt[fail][s[i]]; // 注意这一行不能和下一行互换
    nxt[i - 1][s[i]] = i;
    for(int j = 0; j < m; j ++)
        nxt[i][j] = nxt[fail][j];
}
```

Z 函数

对于一个长度为 n 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, $z[0] = 0$。

```
std::vector<int> getZ(const std::string &s) {
  int n = s.size();
  std::vector<int> Z(n);
  Z[0] = n;
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && Z[i - l] < r - i + 1) {
      Z[i] = Z[i - l];
    } else {
      Z[i] = std::max(0, r - i + 1);
      while (i + Z[i] < n && s[Z[i]] == s[i + Z[i]]) ++Z[i];
    }
    if (i + Z[i] - 1 > r) r = i + Z[l = i] - 1;
  }
  return Z;
}

std::vector<int> match(const std::string &s, const std::string &t) {
  auto Z = getZ(t);
  int n = s.size(), m = t.size();
  std::vector<int> ret(n);
  while (ret[0] < n && ret[0] < m && s[ret[0]] == t[ret[0]]) ++ret[0];
  for (int l = 0, r = ret[0] - 1, i = 1; i < n; ++i) {
    if (i <= r && Z[i - l] < r - i + 1) {
      ret[i] = Z[i - l];
    } else {
      ret[i] = std::max(0, r - i + 1);
      while (i + ret[i] < n && s[i + ret[i]] == t[ret[i]]) ++ret[i];
    }
    if (i + ret[i] - 1 > r) r = i + ret[l = i] - 1;
  }
  return ret;
}
```

LCP

```
for(int i = n;i >= 1;i --) {
    for(int j = n;j >= 1;j --) {
        if(s[i] == s[j]) {
            f[i][j] = f[i + 1][j + 1] + 1;// i-n 和 j-n 的 lcp
        }
    }
}
```

Hash

```
struct Hash {
    string s;
    using ull = unsigned long long;
    ull P1 = 998255347;
```

```
    ull P2 = 1018253347;
    ull base = 131;
    vector<ull> hs1,hs2;
    vector<ull> ps1,ps2;
    Hash(string s): s(s) {
        int n = s.size();
        hs1.resize(n);
        hs2.resize(n);
        ps1.resize(n);
        ps2.resize(n);
        ps1[0] = ps2[0] = 1;
        hs1[0] = hs2[0] = (s[0] - 'a' + 1);
        for(int i = 1;i < n;i ++) {
            hs1[i] = hs1[i - 1] * base % P1 + (s[i] - 'a' + 1);
            hs2[i] = hs2[i - 1] * base % P2 + (s[i] - 'a' + 1);
            ps1[i] = (ps1[i - 1] * base) % P1;
            ps2[i] = (ps2[i - 1] * base) % P2;
        }
    }
    pair<ull,ull> query(int l,int r) {
        ull res1 = (hs1[r] - (l == 0 ? 0 : hs1[l - 1]) * ps1[r - l + 1] % P1 + P1) % P1;
        ull res2 = (hs2[r] - (l == 0 ? 0 : hs2[l - 1]) * ps2[r - l + 1] % P2 + P2) % P2;
        return {res1,res2};
    } // [l,r]
};
```

数学

数论

扩展欧几里得（线性同余方程，斐蜀定理）

扩展欧几里得：$\gcd(a,b)=\gcd(b,a\%b)$，$ax+by=bx+(a-\lfloor\frac{a}{b}\rfloor)y$

斐蜀定理：$ax+by=c$ 若有解，则有 $(a,b)|c$

线性同余方程：$ax\equiv c\pmod b\Rightarrow ax+by=c$

```
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(b == 0){
        x = 1 ,y = 0;return a;
    }
    ll d = exgcd(b,a % b,x,y);
    ll tmp = x;
    x = y;
    y = tmp - (a / b) * y;
    return d;
}
void solve(){
    ll a,b,c;
    cin >> a >> b >> c;
    ll x0,y0;
    ll d = exgcd(a,b,x0,y0);
    if(c % d){
        cout << -1 << "\n";
        return ;
    }
    ll p = a / d,q = b / d;
    ll x = ((c / d) % q * x0 % q + q) % q;
    if(x == 0)x = q;
    ll y = (c - a * x) / b;
    if(y <= 0){
        y = ((c / d) % p * y0 % p + p) % p;
        cout << (x == 0 ? q : x) << " " << (y == 0 ? p : y) << "\n";
        return ;
    }
    ll ans_x_mn = x;
    ll ans_y_mx = y;
    y = ((c / d) % p * y0 % p + p) % p;
    if(y == 0)y = p;
```

```
    x = (c - b * y) / a;
    ll ans_x_mx = x;
    ll ans_y_mn = y;
    ll sum = min((ans_x_mx - ans_x_mn) / q,(ans_y_mx - ans_y_mn) / p);
    cout << sum + 1 << " " << ans_x_mn << " " << ans_y_mn << " " << ans_x_mx << " " << ans_y_mx << "\n";
    // 正整数解总数
}
```

费马小定理（逆元）

若 p 为素数，$\gcd(a,p)=1$，则 $a^{p-1}\equiv 1 \pmod p$

线性求逆元

```
inv[0] = inv[1] = 1;
for(int i = 2 ;i <= n ;i++) inv[i] = (p - p/i) * inv[p % i] % p;
```

CRT（中国剩余定理）

$$\begin{cases} x=b_1(\mathrm{mod} a_1) \\ x=b_2(\mathrm{mod} a_2) \\ \cdots \\ x=b_n(\mathrm{mod} a_n) \end{cases}$$

若 $ a_{1} ,a_{2},...,a_{n}$ 两两互质：

令 $M=\prod_1^n a_i, m_i'=\frac{M}{a_i}, t_i\times m_i'\equiv 1(\mathrm{mod}\ a_i)$ 则有 $x=\sum_{i=1}^n b_i\times m_i'\times t_i$ （此解为唯一解）

若 $ a_{1} ,a_{2},...,a_{n}$ 两两不互质：

合并两个方程组 $x=a_1p+b_1=a_2q+b_2$

则可将方程依次两两合并为 $x\equiv a_1p+b_1(\mathrm{mod}\mathrm{lcm}(a_1,a_2))$，其中先求解 p，再带入求 x。

```
ll r1 = B[1], m1 = A[1],r2,m2;
for(int i = 1;i < n ;i ++) {
    r2 = B[i + 1],m2 = A[i + 1];
    ll a = m1,b = m2,c = r2 - r1;
    ll d = exgcd(a,b,x,y);
    if(c % d) {
        cout<<0;return 0;
    }
    ll p = a / d,q = b / d;
    x = ((x * c / d) + q) % q;
    ll mod = lcm(m2,m1);
    ll x0 = (m1 * x + r1) % mod;
    r1 = x0 < 0 ? x0 + mod : x0;
    m1 = mod;
}
cout << r1 % m1 << "\n";
```

卢卡斯定理

$C_n^m=C_{n\bmod p}^{m\bmod p}\cdot C_{\lfloor n/p\rfloor}^{\lfloor m/p\rfloor}$，其中 p 为质数。

原根

满足同余式 $a^n\equiv 1(\mathrm{mod} m)$ 的最小正整数 n 存在，这个 n 称作 a 模 m 的阶，记作 $\delta_m(a)$，$a,a^2,\cdots,a^{\delta_m(a)}$ 模 m 两两不同余。

设 $m\in \mathbf{N}^*$，$g\in \mathbf{Z}$. 若 $(g,m)=1$，且 $\delta_m(g)=\varphi(m)$，则称 g 为模 m 的原根。

即 g 满足 $\delta_m(g)=|\mathbf{Z}_m^*|=\varphi(m)$. 当 m 是质数时，我们有 $g^i \bmod m$, $0<i<m$ 的结果互不相同。

原根判定定理：

设 $m\geq 3,(g,m)=1$，则 g 是模 m 的原根的充要条件是，对于 $\varphi(m)$ 的每个素因数 p，都有 $g^{\frac{\varphi(m)}{p}}\not\equiv 1 \pmod m$。

若一个数 m 有原根，则它原根的个数为 $\varphi(\varphi(m))$，每一个原根都形如 g^k 的形式，要求满足 $\gcd(k,\varphi(n))=1$。

原根存在定理:

一个数 m 存在原根当且仅当 $m=2,4,p^\alpha,2p^\alpha$，其中 p 为奇素数，$\alpha\in \mathbf{N}^*$。

离散对数（BSGS）

$a^x=b(\mathrm{mod} m)$

此处只解决 m 为质数，将 x 分解为 $i\times t-p$，则有 $a^{i\times t}=b\times a^p(\mathrm{mod} m)$

$t=\sqrt[2]{m}$ 时均摊复杂度最小

$ 0< p < t$ 枚举 p 计算出每一个 a^p 的值存入 hash 表

再枚举 i，算出 $a^{i\times t}$ 的值在 hash 表中查找

```
ll bsgs(ll a,ll b,ll p){
    map <ll,ll> hsh ;hsh.clear();
    ll t = sqrt(p) + 1,j;
    b %= p;
    for(int i = 0 ; i < t ;i++){
        ll tmp = b * qp(a,i,p) % p;
        hsh[tmp] = i;
    }
    a = qp(a,t,p);
    if(a == 0){
        if(b == 0)return 1;
        else return -1;
    }
    for(int i = 0 ;i <= t ;i++){
        ll tmp = qp(a,i,p);
        if(hsh.find(tmp) == hsh.end())j = -1;
        else  j = hsh[tmp];
        if(i * t - j >=0 && j >= 0)return i*t-j;
    }
    return -1;
}
```

威尔逊定理

对于素数 p 有 $(p-1)!\equiv -1\pmod p$。

数论分块

```
ll up(ll x,ll y){
    return (x + y - 1) / y;
}
ll calc_up(ll x){
    ll l = 1,r;ll ans = 1e18;
    while(l <= x){
        ll m = up(x,l);
        if(m == 1)r = x;else r = (x - 1) / (m - 1);
        l = r + 1;
    }
    return ans;
}
ll calc_down(ll x){
    ll l = 1,r;ll ans = 1;
    while(l <= x){
        r = x / (x / l);
        l = r + 1;
    }
    return ans;
}
```

积性函数

数论函数：在所有正整数上的函数被称为算术函数（数论函数）

加性函数：如果数论函数 f 对于任意两个互素的正整数 p, q 均有 $f(pq) = f(p) + f(q)$，称为加性函数

积性函数：如果数论函数 f 对于任意两个互素的正整数 p, q 均有 $f(pq) = f(p)f(q)$，称为积性函数

完全积性函数：在积性函数的基础上，p, q 对于任意正整数均成立，称为完全积性函数

若 $f(x)$ 和 $g(x)$ 均为积性函数，不难证明下列函数均为积性函数：

$$h(x) = f(x^p), h(x) = f^p(x), h(x) = \sum_{d|x} f(d), h(x) = f(x)g(x)$$

常见积性函数：

- 单位函数：$\varepsilon(n) = [n = 1]$。（完全积性）
- 恒等函数：$\mathrm{id}_k(n) = n^k$，$\mathrm{id}_1(n)$ 通常简记作 $\mathrm{id}(n)$。（完全积性）
- 常数函数：$1(n) = 1$。（完全积性）
- 除数函数：$\sigma_k(n) = \sum_{d|n} d^k$。$\sigma_0(n)$ 通常简记作 $d(n)$ 或 $\tau(n)$，$\sigma_1(n)$ 通常简记作 $\sigma(n)$。
- 欧拉函数：$\varphi(n) = \sum_{i=1}^{n} [\gcd(i, n) = 1]$
- 莫比乌斯函数：$\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 \mid n \\ (-1)^{\omega(n)} & \text{otherwise} \end{cases}$，其中 $\omega(n)$ 表示 n 的本质不同质因子个数，它是一个加性函数。

线性筛

一般情况下可通过线性筛快速筛出积性函数

```
void init(const int n){
    mu[1] = 1;phi[1] = 1;
    for(int i = 2;i <= n;i ++){
        if(!vis[i]){
            p[++ tot] = i;
            mu[i] = -1;phi[i] = i - 1;
        }
        for(int j = 1;j <= tot && i * p[j] <= n;j ++){
            vis[i * p[j]] = 1;
            if(i % p[j] == 0){
                phi[i * p[j]] = phi[i] * p[j];
                mu[i * p[j]] = 0;
                break;
            }
            mu[i * p[j]] = -mu[i];
            phi[i * p[j]] = phi[i] * phi[p[j]];
        }
    }
}
```

欧拉函数

欧拉函数（Euler’s totient function），即 $\varphi(n)$，表示的是小于等于 n 和 n 互质的数的个数。$\varphi(n) = \sum_{i=1}^{n} [\gcd(i, n) = 1]$

由唯一分解定理，设 $n = \prod_{i=1}^{s} p_i^{k_i}$，其中 p_i 是质数，有 $\varphi(n) = n \times \prod_{i=1}^{s} \frac{p_i - 1}{p_i}$。

如果 $(a, b) = 1$, $\varphi(a * b) = \varphi(a) * \varphi(b)$ 如果 a 或 b 为质数 $\varphi(a * b) = \varphi(a) * \varphi(b)$ 如果 $(a, b) \neq 1, \varphi(a * b) = \varphi(a) * b$

欧拉定理及扩展

如果 $(a, m), a^{\varphi(m)} \equiv 1 (\mathrm{mod}\ m)$ 当 $b \geq \varphi(p)$ 时 $a^b \equiv a^{b \bmod \varphi(p) + \varphi(p)} (\mathrm{mod}\ p)$ 当 $b < \varphi(p)$ 时 $a^b \equiv a^b (\mathrm{mod}\ p)$

狄利克雷卷积

对于两个数论函数 $f(x)$ 和 $g(x)$，则它们的狄利克雷卷积得到的结果 $h(x)$ 定义为：$h(x)=\sum_{d|x}f(d)g\left(\frac{x}{d}\right)=\sum_{ab=x}f(a)g(b)$ 上式可以简记为：$h=f*g$

狄利克雷卷积满足交换律，结合律，分配律

单位函数 ε 是 Dirichlet 卷积运算中的单位元，即对于任何数论函数 f，都有 $f*\varepsilon=f$

对于任何一个满足 $f(x)\neq 0$ 的数论函数，如果有另一个数论函数 $g(x)$ 满足 $f*g=\varepsilon$，则称 $g(x)$ 是 $f(x)$ 的逆元。由 **等式的性质**可知，逆元是唯一的

常见数论卷积

$\phi*1=id$

$\mu*1=\varepsilon$

$\mu*id=\phi$

两个积性函数的 Dirichlet 卷积也是积性函数

积性函数的逆元也是积性函数，且 $f(1)\neq 0$

莫比乌斯反演

莫比乌斯函数

$$\mu(n)=\begin{cases}1 & n=1\\ 0 & n\text{含有平方因子}\\ (-1)^k & k\text{为}n\text{的本质不同质因子}\end{cases}$$

$\mu*1=\varepsilon$

形式一：

$f(n)=\sum_{d|n}g(d)\Rightarrow g(n)=\sum_{d|n}\mu(d)f(\frac{n}{d})$

形式二：

$f(n)=\sum_{n|d}g(d)\Rightarrow g(n)=\sum_{n|d}\mu(\frac{d}{n})f(d)$

上两式的证明可通过等价替换和交换求和号来推导

此外我们也可以通过狄利克雷卷积来理解

$f=g*1,g=f*\mu$

实际应用中我们常使用 $\varepsilon(\gcd)=\sum_{d|\gcd}\mu(d)$，即 $\mu*1=\varepsilon$

同时也是形式 1 中 $f=\varepsilon$ 的情况

欧拉反演

$\varphi*1=id$

展开形式同莫比乌斯反演，本质上是莫比乌斯反演的进一步推导，卷积式也可用 $\mu*1=\varepsilon$ 推出

实际应用中常使用 $\gcd=\sum_{d|\gcd}\varphi(d)$

杜教筛

对于数论函数 f，要计算 $S(n)=\sum_{i=1}^{n}f(i)$

找到一个数论函数 g，有 $\sum_{i=1}^{n}(f*g)(i)=\sum_{i=1}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)$

得到 $g(1)S(n)=S(n)=\sum_{i=1}^{n}(f*g)(i)-\sum_{i=2}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)$

```cpp
const int maxn = 3e6 + 7;
int mu[maxn],p[maxn],vis[maxn],tot;
int sum_mu[maxn];
int phi[maxn];
ll sum_phi[maxn];
map <ll,ll> mp_mu,mp_phi;
void init(const int n){
    mu[1] = 1;phi[1] = 1;
    for(int i = 2;i <= n;i ++){
        if(!vis[i]){
            p[++ tot] = i;
            mu[i] = -1;phi[i] = i - 1;
        }
        for(int j = 1;j <= tot && i * p[j] <= n;j ++){
            vis[i * p[j]] = 1;
            if(i % p[j] == 0){
                phi[i * p[j]] = phi[i] * p[j];
                mu[i * p[j]] = 0;
                break;
            }
            mu[i * p[j]] = -mu[i];
            phi[i * p[j]] = phi[i] * phi[p[j]];
        }
    }
    for(int i = 1;i < n;i ++)sum_mu[i] = sum_mu[i - 1] + mu[i],sum_phi[i] = sum_phi[i - 1] + phi[i];
}
ll calc_mu(ll x){
    if(x < maxn)return sum_mu[x];
    if(mp_mu[x]) return mp_mu[x];
    ll l = 2,r;ll ans = 1;
    while(l <= x){
        r = x / (x / l);
        ans -= 1ll * (r - l + 1) * calc_mu(x / l);
        l = r + 1;
    }
    return mp_mu[x] = ans;
}
ll calc_phi(ll x){
    if(x < maxn)return sum_phi[x];
    if(mp_phi[x]) return mp_phi[x];
    ll l = 2,r;ll ans = 1ll * x * (x + 1) >> 1;
    while(l <= x){
        r = x / (x / l);
        ans -= 1ll * (r - l + 1) * calc_phi(x / l);
        l = r + 1;
    }
    return mp_phi[x] = ans;
}
```

由于符合数论反演实际意义的函数不多所以大部分数论反演题目基本上都是对上述两种反演的卷积式的应用，变形之后进行求和号交换，换元等数学手段处理等到可以快速求解的算式

Min_25

第一步

目标：求 $g(n)=\sum_{p\le n} f(p)$。

不妨设 $f(p)$ 是完全积性函数，如果不是可以尝试拆成若干项完全积性函数，分别求然后相加。

首先要线性筛求出 $\sqrt{n}$ 以内的质数。

$g(n)$ 很难直接求解，考虑用 DP 计算。设 $g(n,j)=\sum_{i=1}^{n} f(i)[i\text{是质数或其最小质因子}>p_j]$，其中 p_j 表示第 j 个质数，那么我们要的就是 $g(n,k),k$为最小的满足$p_k\ge\sqrt{n}$。考虑从 $j-1$ 变到 j，那么最小质因子为 p_j 的合数会被筛掉，那么它们的贡献要减去。则有转移

$$g(n,j)=g(n,j-1)-f(p_j)\left(g\left(\left\lfloor\frac{n}{p_j}\right\rfloor,j-1\right)-g(p_{j-1},j-1)\right)$$

系数 $f(p_j)$ 表示由于 $f(p)$ 是完全积性函数，所以可以把它从后面提出来。$g\left(\left\lfloor\frac{n}{p_j}\right\rfloor, j-1\right)$ 表示考虑所有 p_j 的倍数，它们除以 p_j 之后，最小质因子 $> p_{j-1}$ 的合数**以及所有质数**的贡献，应当减去。但是，这些**质数**中可能有 $\le p_{j-1}$ 的，它们在之前就被筛掉过了，所以要加回来，也就是 $g(p_{j-1}, j-1)$。

由于有公式 $\left\lfloor\frac{\lfloor\frac{a}{b}\rfloor}{c}\right\rfloor = \left\lfloor\frac{a}{bc}\right\rfloor$，因此容易发现上述式子只会用到形如 $\left\lfloor\frac{n}{x}\right\rfloor, x \le n$ 的点处的 DP 值，即第一项的状态数是 $O(\sqrt{n})$（实际实现的时候注意状态数是 $2\sqrt{n}$）。我们预处理出这 $O(\sqrt{n})$ 个数，把他们离散化，顺带求出 $g(x, 0)$，然后 DP 即可。

第二步

目标：求 $S(n) = \sum_{i \le n} f(i)$。与第一步类似，设 $S(n, j) = \sum_{i=1}^{n} f(i)[i\text{的最小质因子} > p_j]$。但此处 f 不需要再拆分成单项式，直接是原函数即可（因为不需要依赖于**完全积性**，只需要**积性**即可）（但要能快速计算 $f(p^k)$ 的值）。

考虑把贡献拆成质数的和合数的，合数枚举最小质因子以及次数，于是有转移：

$$S(n, j) = g(n) - g(p_j) + \sum_{j<k, p_k \le \sqrt{n}, 1 \le e, p_k^e \le n} f(p_k^e)\left(S\left(\left\lfloor\frac{n}{p_k^e}\right\rfloor, k\right) + [e \ne 1]\right)$$

最后一项 $[e \ne 1]$ 的意思是，对于 $e = 1$ 的情况，S 没有计算 1 贡献，刚好，因为此时 $p_k \times 1$ 是质数，其贡献在之前计算过；对于 $e > 1$ 的情况，$p_k^e \times 1$ 是合数，贡献算漏了，要补上。直接暴力递归计算（并且不需要记忆化）。

```cpp
#include<bits/stdc++.h>

using namespace std;
using ll = long long;
const int mo = 1e9 + 7;
std::vector<int> minp, primes;
void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();
    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }
        for (auto p : primes) {
            if (i * p > n) break;
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
int qp(int a,int b) {
    int ans = 1,base = a;
    while(b != 0) {
        if(b & 1) ans = 1ll * ans * base % mo;
        base = 1ll * base * base % mo;
        b >>= 1;
    }
    return ans;
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    ll n;
    cin >> n;
    ll s = sqrt(n);
    sieve(s);
    vector<ll> v;
    for(ll l = 1,r;l <= n;l = r + 1) {
        v.push_back(n / l);
        r = n / (n / l);
    }
    int inv2 = qp(2,mo - 2);
    int inv6 = qp(6,mo - 2);
    auto calc1 = [&](ll x) ->ll {
        x %= mo;
```

```
        return (x * (x + 1) % mo) * inv2 % mo;
    };
    auto calc2 = [&](ll x) ->ll {
        x %= mo;
        return (x * (x + 1) % mo * (2 * x + 1) % mo) * inv6 % mo;
    };
    vector<ll> f1(v.size() + 1),f2(v.size() + 1);
    auto getid = [&](ll x) -> ll {
        if(x <= s) return v.size() - x;
        else return n / x - 1;
    };
    for(int i = 0;i < v.size();i ++) {
        ll x = v[i];
        x %= mo;
        f1[i] = (calc1(x) + mo - 1) % mo;// \sum_2^n i^k = \sum_1^n i^k - 1
        f2[i] = (calc2(x) + mo - 1) % mo;
    }
    ll ps1 = 0,ps2 = 0;
    for(auto p : primes) {
        for(int i = 0;i < v.size();i ++) {
            if(1ll * p * p > v[i]) break;
            f1[i] -= 1ll * p * (f1[getid(v[i] / p)] - ps1 + mo) % mo;
            f1[i] += mo;f1[i] %= mo;
            f2[i] -= 1ll * p * p % mo * (f2[getid(v[i] / p)] - ps2 + mo) % mo;
            f2[i] += mo;f2[i] %= mo;
        }
        ps1 += p;ps1 %= mo;
        ps2 += 1ll * p * p % mo;ps2 %= mo;
    }
    auto F = [](ll x) -> ll { // targeted function
        x %= mo;
        return (x * x % mo - x + mo) % mo;
    };
    auto S = [&](auto self,ll x,int y) -> ll {
        int p = y == 0 ? 0 : primes[y - 1];
        if(p >= x) return 0ll;
        ll res = (f2[getid(x)] - f2[getid(p)] - (f1[getid(x)] - f1[getid(p)]) + mo + mo) % mo;
        for(int i = y;i < primes.size() && primes[i] <= x / primes[i];i ++) {
            ll w = primes[i];
            for(int j = 1;w <= x;j ++,w = w * primes[i]) {
                res = (res + F(w) * (self(self,x / w,i + 1) % mo + (j != 1)) % mo) % mo;
            }
        }
        return res;
    };
    cout << (S(S,n,0) + 1) % mo << "\n";
}
```

素数测试与因式分解（Miller-Rabin & Pollard-Rho）

```
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
```

```
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
std::vector<i64> factorize(i64 n) {
    std::vector<i64> p;
    std::function<void(i64)> f = [&](i64 n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
                p.push_back(n);
            return;
        }
        if (isprime(n)) {
            p.push_back(n);
            return;
        }
        auto g = [&](i64 x) {
            return (mul(x, x, n) + 1) % n;
        };
        i64 x0 = 2;
        while (true) {
            i64 x = x0;
            i64 y = x0;
            i64 d = 1;
            i64 power = 1, lam = 0;
            i64 v = 1;
            while (d == 1) {
                y = g(y);
                ++lam;
                v = mul(v, std::abs(x - y), n);
                if (lam % 127 == 0) {
                    d = std::gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
                    lam = 0;
                    d = std::gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                f(d);
                f(n / d);
                return;
            }
            ++x0;
        }
    };
    f(n);
    std::sort(p.begin(), p.end());
    return p;
}
```

公式

一些数论公式

- 当 $x \geq \phi(p)$ 时有 $a^x \equiv a^{x \bmod \phi(p)+\phi(p)} \pmod p$
- $\mu^2(n) = \sum_{d^2|n} \mu(d)$
- $\sum_{d|n} \varphi(d) = n$
- $\sum_{d|n} 2^{\omega(d)} = \sigma_0(n^2)$，其中 ω 是不同素因子个数
- $\sum_{d|n} \mu^2(d) = 2^{\omega(d)}$

一些数论函数求和的例子

- $\sum_{i=1}^{n} i[gcd(i,n)=1] = \frac{n\varphi(n)+[n=1]}{2}$
- $\sum_{i=1}^{n}\sum_{j=1}^{m}[gcd(i,j)=x] = \sum_d \mu(d)\lfloor\frac{n}{dx}\rfloor\lfloor\frac{m}{dx}\rfloor$
- $\sum_{i=1}^{n}\sum_{j=1}^{m} gcd(i,j) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{d|gcd(i,j)}\varphi(d) = \sum_d \varphi(d)\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor$
- $S(n) = \sum_{i=1}^{n}\mu(i) = 1 - \sum_{i=1}^{n}\sum_{d|i,d<i}\mu(d) \overset{t=\frac{i}{d}}{=} 1 - \sum_{t=2}^{n} S(\lfloor\frac{n}{t}\rfloor)$
  - 利用 $[n=1] = \sum_{d|n}\mu(d)$
- $S(n) = \sum_{i=1}^{n}\varphi(i) = \sum_{i=1}^{n} i - \sum_{i=1}^{n}\sum_{d|i,d<i}\varphi(i) \overset{t=\frac{i}{d}}{=} \frac{i(i+1)}{2} - \sum_{t=2}^{n} S(\frac{n}{t})$
  - 利用 $n = \sum_{d|n}\varphi(d)$
- $\sum_{i=1}^{n}\mu^2(i) = \sum_{i=1}^{n}\sum_{d^2|n}\mu(d) = \sum_{d=1}^{\lfloor\sqrt{n}\rfloor}\mu(d)\lfloor\frac{n}{d^2}\rfloor$
- $\sum_{i=1}^{n}\sum_{j=1}^{n} gcd^2(i,j) = \sum_d d^2\sum_t \mu(t)\lfloor\frac{n}{dt}\rfloor^2$
  $\overset{x=dt}{=} \sum_x \lfloor\frac{n}{x}\rfloor^2 \sum_{d|x} d^2\mu(\frac{x}{d})$
- $\sum_{i=1}^{n}\varphi(i) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}[i\perp j] - 1 = \frac{1}{2}\sum_{i=1}^{n}\mu(i)\cdot\lfloor\frac{n}{i}\rfloor^2 - 1$

斐波那契数列性质

- $F_{a+b} = F_{a-1}\cdot F_b + F_a \cdot F_{b+1}$
- $F_1 + F_3 + \cdots + F_{2n-1} = F_{2n}, F_2 + F_4 + \cdots + F_{2n} = F_{2n+1} - 1$
- $\sum_{i=1}^{n} F_i = F_{n+2} - 1$
- $\sum_{i=1}^{n} F_i^2 = F_n \cdot F_{n+1}$
- $F_n^2 = (-1)^{n-1} + F_{n-1}\cdot F_{n+1}$
- $gcd(F_a, F_b) = F_{gcd(a,b)}$
- 模 n 周期（皮萨诺周期）
  - $\pi(p^k) = p^{k-1}\pi(p)$
  - $\pi(nm) = lcm(\pi(n),\pi(m)), \forall n \perp m$
  - $\pi(2) = 3, \pi(5) = 20$
  - $\forall p \equiv \pm 1 \pmod{10}, \pi(p) | p-1$
  - $\forall p \equiv \pm 2 \pmod 5, \pi(p) | 2p+2$

组合数学

组合化简技巧

$$\binom{n}{m} = \binom{n}{n-m} \tag{1}$$

相当于将选出的集合对全集取补集，故数值不变。（对称性）

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1} \tag{2}$$

由定义导出的递推式。

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1} \tag{3}$$

组合数的递推式（杨辉三角的公式表达）。我们可以利用这个式子，在 $O(n^2)$ 的复杂度下推导组合数。

$$\binom{n}{0}+\binom{n}{1}+\cdots+\binom{n}{n}=\sum_{i=0}^{n}\binom{n}{i}=2^n \tag{4}$$

这是二项式定理的特殊情况。取 $a=b=1$ 就得到上式。

$$\sum_{i=0}^{n}(-1)^i\binom{n}{i}=[n=0] \tag{5}$$

二项式定理的另一种特殊情况，可取 $a=1,b=-1$。式子的特殊情况是取 $n=0$ 时答案为 1。

$$\sum_{i=0}^{m}\binom{n}{i}\binom{m}{m-i}=\binom{m+n}{m}\quad(n\geq m) \tag{6}$$

拆组合数的式子，在处理某些数据结构题时会用到。

$$\sum_{i=0}^{n}\binom{n}{i}^2=\binom{2n}{n} \tag{7}$$

这是 (6) 的特殊情况，取 $n=m$ 即可。

$$\sum_{i=0}^{n}i\binom{n}{i}=n2^{n-1} \tag{8}$$

带权和的一个式子，通过对 (3) 对应的多项式函数求导可以得证。

$$\sum_{i=0}^{n}i^2\binom{n}{i}=n(n+1)2^{n-2} \tag{9}$$

与上式类似，可以通过对多项式函数求导证明。

$$\sum_{l=0}^{n}\binom{l}{k}=\binom{n+1}{k+1} \tag{10}$$

通过组合分析一一考虑 $S=\{a_1,a_2,\cdots,a_{n+1}\}$ 的 $k+1$ 子集数可以得证，在恒等式证明中比较常用。

$$\binom{n}{r}\binom{r}{k}=\binom{n}{k}\binom{n-k}{r-k} \tag{11}$$

通过定义可以证明。

$$\sum_{i=0}^{n}\binom{n-i}{i}=F_{n+1} \tag{12}$$

其中 F 是斐波那契数列。

鸽巢原理

把 $n+1$ 个物品放进 n 个盒子，至少有一个盒子包含两个或更多的物品

加强形式：

令 $q_1, q_2 \dots q_n$ 为正整数，如果将 $q_1+q_2+\cdots+q_n-n+1$ 个物品放入 n 个盒子，那么，或者第 1 个盒子至少含有 q_1 个物品，或者第 2 个盒子至少含有 q_2 个物品，$\cdots$，或者第 n 个盒子至少含有 q_n 个物品。

容斥原理

$\bigcup_{i=1}^{n} S_i = \sum_{m=1}^{n}(-1)^{m-1}\sum_{a_i<a_{i+1}}\bigcap_{i=1}^{m} S_{a_i}$

二项式定理

$(1+x)^n = \sum_{r=0}^{n} C_n^r x^r$

多重集的排列数 | 多重组合数

请大家一定要区分 **多重组合数**与 **多重集的组合数**！两者是完全不同的概念！

多重集是指包含重复元素的广义集合。设 $S=\{n_1\cdot a_1, n_2\cdot a_2, \cdots, n_k\cdot a_k\}$ 表示由 n_1 个 a_1，n_2 个 a_2，...，n_k 个 a_k 组成的多重集，S 的全排列个数为

$$\frac{n!}{\prod_{i=1}^{k} n_i!} = \frac{n!}{n_1!n_2!\cdots n_k!}$$

相当于把相同元素的排列数除掉了。具体地，你可以认为你有 k 种不一样的球，每种球的个数分别是 $n_1, n_2, \cdots, n_k$，且 $n = n_1+n_2+\ldots+n_k$。这 n 个球的全排列数就是 **多重集的排列数**。多重集的排列数常被称作 **多重组合数**。我们可以用多重组合数的符号表示上式：

$$\binom{n}{n_1, n_2, \cdots, n_k} = \frac{n!}{\prod_{i=1}^{k} n_i!}$$

可以看出，$\binom{n}{m}$ 等价于 $\binom{n}{m, n-m}$，只不过后者较为繁琐，因而不采用。

多重集的组合数 1

设 $S=\{n_1\cdot a_1, n_2\cdot a_2, \cdots, n_k\cdot a_k\}$ 表示由 n_1 个 a_1，n_2 个 a_2，...，n_k 个 a_k 组成的多重集。那么对于整数 $r(r<n_i, \forall i\in[1,k])$，从 S 中选择 r 个元素组成一个多重集的方案数就是 **多重集的组合数**。这个问题等价于 $x_1+x_2+\cdots+x_k=r$ 的非负整数解的数目，可以用插板法解决，答案为

$$\binom{r+k-1}{k-1}$$

多重集的组合数 2

考虑这个问题：设 $S=\{n_1\cdot a_1, n_2\cdot a_2, \cdots, n_k\cdot a_k,\}$ 表示由 n_1 个 a_1，n_2 个 a_2，...，n_k 个 a_k 组成的多重集。那么对于正整数 r，从 S 中选择 r 个元素组成一个多重集的方案数。

这样就限制了每种元素的取的个数。同样的，我们可以把这个问题转化为带限制的线性方程求解：

$$\forall i\in[1,k],\ x_i\le n_i,\ \sum_{i=1}^{k} x_i = r$$

于是很自然地想到了容斥原理。容斥的模型如下：

1. 全集：$\sum_{i=1}^{k} x_i = r$ 的非负整数解。
2. 属性：$x_i \le n_i$。

于是设满足属性 i 的集合是 S_i，$\overline{S_i}$ 表示不满足属性 i 的集合，即满足 $x_i \ge n_i + 1$ 的集合（转化为上面插板法的问题三）。那么答案即为

$$\left|\bigcap_{i=1}^{k} S_i\right| = |U| - \left|\bigcup_{i=1}^{k} \overline{S_i}\right|$$

根据容斥原理，有：

$$\begin{aligned}
\left|\bigcup_{i=1}^{k} \overline{S_i}\right| &= \sum_i |\overline{S_i}| - \sum_{i,j} |\overline{S_i} \cap \overline{S_j}| + \sum_{i,j,k} |\overline{S_i} \cap \overline{S_j} \cap \overline{S_k}| - \cdots && (1)\\
&\quad + (-1)^{k-1} \left|\bigcap_{i=1}^{k} \overline{S_i}\right| && (2)\\
&= \sum_i \binom{k+r-n_i-2}{k-1} - \sum_{i,j} \binom{k+r-n_i-n_j-3}{k-1} + \sum_{i,j,k} \binom{k+r-n_i-n_j-n_k-4}{k-1} - \cdots && (3)\\
&\quad + (-1)^{k-1} \binom{k+r-\sum_{i=1}^{k} n_i - k - 1}{k-1} && (4)
\end{aligned}$$

拿全集 $|U| = \binom{k+r-1}{k-1}$ 减去上式，得到多重集的组合数

$$Ans = \sum_{p=0}^{k} (-1)^p \sum_{A} \binom{k+r-1-\sum_A n_{A_i} - p}{k-1}$$

其中 A 是充当枚举子集的作用，满足 $|A| = p,\ A_i < A_{i+1}$。

圆排列

n 个人全部来围成一圈，所有的排列数记为 Q_n^n。考虑其中已经排好的一圈，从不同位置断开，又变成不同的队列。所以有

$$\mathrm{Q}_n^n \times n = \mathrm{A}_n^n \Longrightarrow \mathrm{Q}_n = \frac{\mathrm{A}_n^n}{n} = (n-1)!$$

由此可知部分圆排列的公式：

$$\mathrm{Q}_n^r = \frac{\mathrm{A}_n^r}{r} = \frac{n!}{r \times (n-r)!}$$

错排

$D_n = (n-1)(D_{n-1} + D_{n-2})$

$D_n = nD_{n-1} + (-1)^n$

Catalan 数

$C_{2n}^n - C_{2n}^{n-1} = \frac{C_{2n}^n}{n+1}$

Stirling 数

第一类斯特林数（感觉 useless）

把 n 个不同元素分配到 k 个圆排列里，圆不能为空

$s_{n,k} = s_{n-1,k-1} + (n-1) \times s_{n-1,k}$

第二类斯特林数

将 n 个物体划分成 k 个非空的没有区别的集合的方法数，等同于把 n 个不同的小球放入 m 个相同的盒子中（且盒子不能为空）的方案数。递推公式为

$s_{i,j} = s_{i-1,j} \times j + s_{i-1,j-1}$ （$s_{i,j}$ 表示前 i 个小球放到前 j 个盒子里的方案数）

我们可以这样理解：对于一个 $s_{i,j}$，你接下来要再放一个小球，你可以放到前 j 个盒子里，方案数为 $j \times s_{i-1,j-1}$，也可以放到下一个盒子里，方案数为 $s_{i,j+1}$。

此外这个部分还有一种做法。考虑容斥：枚举多少个盒子空了，然后剩下的部分就是第三种情况了。然后就可以得到下面这个式子：

$S_{n,m} = \frac{1}{m!} \sum_{i=0}^{m} (-1)^i \binom{m}{i} (m-i)^n$

高维前缀和

对于所有的 $i, 0 \le i \le 2^n - 1$，求解 $\sum_{j \subset i} a_j$

去除原有二维前缀和的容斥求法，对每一维采用类似一维前缀和的方式依次累加进行计算，每次提高一个维度不断累加。实际上就等价于 f_i 加上 $f_{i \oplus (2^j)}$，类似 or 运算

```
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(i >> j & 1) f[i] += f[i ^ (1 << j)];
```

对超集求和，也为高维后缀和，类似 and 运算

```
for(int j = 0; j < n; j++)
    for(int i = 0; i < 1 << n; i++)
        if(!(i >> j & 1)) f[i] += f[i ^ (1 << j)];
```

二项式反演

三种形式

$$f(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} g(k) \iff g(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} f(k)$$

至多和恰好的转换

$$f(n) = \sum_{k=0}^{n} \binom{n}{k} g(k) \iff g(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} f(k)$$

至少和恰好的转换

$$f(n) = \sum_{k=n}^{\infty} \binom{k}{n} g(k) \iff g(n) = \sum_{k=n}^{\infty} (-1)^{k-n} \binom{k}{n} f(k)$$

斯特林反演

$$f(n) = \sum_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} g(k) \iff g(n) = \sum_{k=0}^{n} (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix} f(k)$$

$$f(n) = \sum_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} g(k) \iff g(n) = \sum_{k=0}^{n} (-1)^{n-k} \begin{bmatrix} n \\ k \end{bmatrix} f(k)$$

子集反演

$$f(S)=\sum_{T\subseteq S}g(T)\Longrightarrow g(S)=\sum_{T\subseteq S}(-1)^{|S|-|T|}f(T)$$

最值反演（min-max 容斥）

$$\max S=\sum_{T\subseteq S}(-1)^{|T|-1}\min T$$

$$\min S=\sum_{T\subseteq S}(-1)^{|T|-1}\max T$$

拓展

$$k^{th}\max S=\sum_{T\subseteq S}(-1)^{|T|-k}\binom{|T|-1}{k-1}\min T$$

反之同理

此外上述两种 min-max 容斥在期望上仍然成立

线性代数

高斯消元

```cpp
typedef long long ll;
const int mo = 1e9 + 7;
using i64 = long long;

template<typename T>
struct Gauss{
    int n,m;// n 行 m 列
    std::vector<std::vector<T>> a;
    Gauss(int n,int m,const vector<vector<T>> &a):n(n),m(m),a(a){}
    vector<double> gauss(){
        std::vector<double> sol;
        int r = 0;
        for(int i = 0;i < m;i ++){
            if(n < m && i >= n) break;
            int mx = i;//最大数的行号
            for(int j = i + 1;j < n;j ++)
                if(fabs(a[mx][i]) < fabs(a[j][i])) mx = j;
            if(fabs(a[mx][i]) < eps) continue;
            if(mx != i) swap(a[mx],a[i]);
            double t = a[i][i];
            for(int j = i;j < m + 1;j ++){
                a[i][j] /= t;
            }
            for(int j = i + 1;j < n;j ++){
                t = a[j][i];
                for(int k = i;k < m + 1;k ++){
                    a[j][k] -= a[i][k] * t;
                }
            }
            r ++;
        }
        for(int i = m;i < n;i ++){
            if(fabs(a[i][m]) > eps)return sol;// 无解
        }
        if(r < m) return sol;// 无穷解
        sol.resize(m);
        sol[m - 1] = a[m - 1][m];
        for(int i = n - 2;i >= 0;i --){
            sol[i] = a[i][m];
            for(int j = i + 1;j < m;j ++){
                sol[i] -= a[i][j] * sol[j];
```

```
            }
        }
        return sol;
    }
    int gauss_det(){
        assert(n == m);
        int det = 1;
        for(int i = 0;i < n;i ++){
            for(int j = i + 1;j < n;j ++){
                while(a[j][i] != 0){
                    int k = a[i][i] / a[j][i];
                    for(int h = i;h < n;h ++){
                        a[i][h] -= a[j][h] * k;
                        std::swap(a[i][h],a[j][h]);
                    }
                    det = -det;
                }
            }
        }
        for(int i = 0;i < n;i ++) det *= a[i][i];
        return det;
    }

    T gauss_det_mod(int st = 0){
        T det = T(1);
        for(int i = st;i < n;i ++){
            det *= a[i][i];
            //cerr << i << " " << a[i][i].a << " " << a[i][i].b << "\n";
            T tmp = T(1) / a[i][i];
            for(int j = i;j < n;j ++) a[i][j] *= tmp;
            for(int j = i + 1;j < n;j ++){
                tmp = a[j][i];
                for(int k = i;k < n;k ++) a[j][k] = a[j][k] - tmp * a[i][k];
            }
        }
        return det;
    }
};
```

线性基

```
struct LinearBasis {
    static const int n = 60;
    bool O; ll a[n + 1]; int cnt;

    void clear() { O = 0; cnt = 0; for (int i = 0; i <= n; ++i) a[i] = 0; }
    void insert(ll v) {
        for (int i = n; ~i; --i) {
            if (!(v >> i & 1)) continue;
            if (!a[i]) { a[i] = v; break; }
            v ^= a[i];
        } O |= !v;
    }
    void work() {
        for (int i = 0; i <= n; cnt += a[i++] > 0)
            for (int j = 0; j < i; ++j)
                if (a[i] >> j & 1) a[i] ^= a[j];
    }
    ll get_max(ll v = 0) {
        for (int i = n; ~i; --i) v = max(v, v ^ a[i]);
        return v;
    }
    ll get_min() {
        if (O) return 0;
        for (int i = 0; i <= n; ++i)
            if (a[i]) return a[i];
    }
    ll get_min(ll v) {
        for (int i = n; ~i; --i) v = min(v, v ^ a[i]);
        return v;
    }
```

```
    ll get_kth(ll k) {
        ll ans = 0; k -= 0;
        for (int i = 0; i <= n; ++i) {
            if (!a[i]) continue;
            if (k & 1) ans ^= a[i];
            k >>= 1;
        }
        if (k > 0) return -1;
        else return ans;
    }
    bool check(ll v) {
        for (int i = n; ~i; --i) {
            if (!(v >> i & 1)) continue;
            if (!a[i]) return 0;
            v ^= a[i];
        } return 1;
    }
};
```

Prüfer 序列

Prüfer 是这样建立的：每次选择一个编号最小的叶结点并删掉它，然后在序列中记录下它连接到的那个结点。重复 $n-2$ 次后就只剩下两个结点。

- **重要性质：prufer 序列与无根树一一对应。**
- **度数为 d_i 的节点会在 prufer 序列中出现 d_i-1 次。**
- **一个 n 个节点的完全图的生成树个数为 n^{n-2}。**
- **对于给定度数为 $d_{1\dots n}$ 的一棵无根树共有 $\frac{(n-2)!}{\prod(d_i-1)!}$ 种情况。**
- **n 个点 m 条边的带标号无向图有 k 个连通块。我们希望添加 $k-1$ 条边使得整个图连通。方案数为 $n^{k-2}\prod_{i=1}^{k} s_i$。**

LGV 引理

LGV 引理仅适用于 **有向无环图**。

定义

$\omega(P)$ 表示 P 这条路径上所有边的边权之积。（路径计数时，可以将边权都设为 1）（事实上，边权可以为生成函数）

$e(u,v)$ 表示 u 到 v 的 **每一条**路径 P 的 $\omega(P)$ 之和，即 $e(u,v)=\sum\limits_{P:u\to v}\omega(P)$。

起点集合 A，是有向无环图点集的一个子集，大小为 n。

终点集合 B，也是有向无环图点集的一个子集，大小也为 n。

一组 $A\to B$ 的不相交路径 S：S_i 是一条从 A_i 到 $B_{\sigma(S)_i}$ 的路径（$\sigma(S)$ 是一个排列），对于任何 $i\neq j$，S_i 和 S_j 没有公共顶点。

$N(\sigma)$ 表示排列 σ 的逆序对个数。

引理

$$M=\begin{bmatrix} e(A_1,B_1) & e(A_1,B_2) & \cdots & e(A_1,B_n)\\ e(A_2,B_1) & e(A_2,B_2) & \cdots & e(A_2,B_n)\\ \vdots & \vdots & \ddots & \vdots\\ e(A_n,B_1) & e(A_n,B_2) & \cdots & e(A_n,B_n)\end{bmatrix}$$

$$\det(M)=\sum_{S:A\to B}(-1)^{N(\sigma(S))}\prod_{i=1}^{n}\omega(S_i)$$

其中 $\sum_{S:A\to B}$ 表示满足上文要求的 $A\to B$ 的每一组不相交路径 S。

矩阵树定理

定理 1（矩阵树定理，无向图行列式形式）对于任意的 i，都有

$$t(G)=\det L(G)\binom{1,2,\cdots,i-1,i+1,\cdots,n}{1,2,\cdots,i-1,i+1,\cdots,n}$$

其中记号 $L(G)\binom{1,2,\cdots,i-1,i+1,\cdots,n}{1,2,\cdots,i-1,i+1,\cdots,n}$ 表示矩阵 $L(G)$ 的第 $1,\cdots,i-1,i+1,\cdots,n$ 行与第 $1,\cdots,i-1,i+1,\cdots,n$ 列构成的子矩阵。也就是说，无向图的 Laplace 矩阵具有这样的性质，它的所有 $n-1$ 阶主子式都相等。

定理 2（矩阵树定理，无向图特征值形式）设 $\lambda_1,\lambda_2,\cdots,\lambda_{n-1}$ 为 $L(G)$ 的 $n-1$ 个非零特征值，那么有

$t(G)=\frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}$

定理 3（矩阵树定理，有向图根向形式）对于任意的 k，都有

$$t^{root}(G,k)=\det L^{out}(G)\binom{1,2,\cdots,k-1,k+1,\cdots,n}{1,2,\cdots,k-1,k+1,\cdots,n}$$

因此如果要统计一张图所有的根向树形图，只要枚举所有的根 k 并对 $t^{root}(G,k)$ 求和即可。

定理 4（矩阵树定理，有向图叶向形式）对于任意的 k，都有

$$t^{leaf}(G,k)=\det L^{in}(G)\binom{1,2,\cdots,k-1,k+1,\cdots,n}{1,2,\cdots,k-1,k+1,\cdots,n}$$

因此如果要统计一张图所有的叶向树形图，只要枚举所有的根 k 并对 $t^{leaf}(G,k)$ 求和即可。

BEST 定理

定理 5 （BEST 定理）设 G 是有向欧拉图，那么 G 的不同欧拉回路总数 $ec(G)$ 是

$$ec(G)=t^{root}(G,k)\prod_{v\in V}(\deg(v)-1)!$$

注意，对欧拉图 G 的任意两个节点 k,k'，都有 $t^{root}(G,k)=t^{root}(G,k')$，且欧拉图 G 的所有节点的入度和出度相等。

博弈

Nim 游戏：每轮从若干堆石子中的一堆取走若干颗。先手必胜条件为石子数量异或和非零。

阶梯 Nim 游戏：可以选择阶梯上某一堆中的若干颗向下推动一级，直到全部推下去。先手必胜条件是奇数阶梯的异或和非零（对于偶数阶梯的操作可以模仿）。

Anti-SG：无法操作者胜。先手必胜的条件是：

- SG 不为 0 且某个单一游戏的 SG 大于 1 。
- SG 为 0 且没有单一游戏的 SG 大于 1。

Every-SG：对所有单一游戏都要操作。先手必胜的条件是单一游戏中的最大 step 为奇数。

- 对于终止状态 step 为 0
- 对于 SG 为 0 的状态，step 是最大后继 step +1
- 对于 SG 非 0 的状态，step 是最小后继 step +1

树上删边：叶子 SG 为 0，非叶子结点为所有子结点的 SG 值加 1 后的异或和。