总

笛卡尔树,重链剖分,KMP,卢卡斯(快速求组合数，杨辉三角奇偶性),统计1~n之间所有数数位和,辛普森公式求弧长公式定积分,kruskal重构树,区间修改线段树,矩阵乘法,LCA,分块,可持久化线段树（主席树，[l, r] 区间第k小）,字符串哈希,st表,并查集（启发式合并）,数学,dp,拓欧,tarjan求强连通分量,建虚树,Manacher(回文串匹配),AC自动机,线性筛,拓扑排序,分组背包,Floyd最短路（含修改边权）

笛卡尔树

```
/**
 * 笛卡尔树
 * 将无序数组变成一个有序表
 * 构建出一棵树，
 * 树上节点按照原数组的索引(key)来看是一棵二叉搜索树
 * (二叉搜索树概念：所有节点左子树上的值均小于它自己，所有节点右孩子上的值均大于它自己)
 * 树上节点按照原数组的值(value)来看是一个(小/大)根堆
 * 建树复杂度O(n)
 */
int main() {
    // ios::sync_with_stdio(0);
    // cin.tie(0), cout.tie(0);
    int n;
    cin >> n;
    vector<int> a(n + 1, 0);
    vector<int> ls(n + 1, 0), rs(n + 1, 0);
    stack<pair<int, int>> stk;
    for (int i = 1;i <= n;i ++) cin >> a[i];
    int lst = 0;
    /**
     * 如果栈为空，则压入元素作为根
     * 压入栈的元素为栈顶元素的右孩子(要求栈顶元素的val小于压入栈元素的val)
     * 如果不满足栈顶元素val小于压入栈元素的val，则要进行弹出操作
     * 最后弹出的元素将作为压入栈元素的左孩子
     */
    for (int i = 1;i <= n;i ++) {
        lst = 0;
        if (stk.empty()) {
            stk.push({i, a[i]});
        } else {
            while (!stk.empty() && stk.top().y > a[i]) {
                lst = stk.top().x;
                stk.pop();
            }
            if (stk.empty()) {
                ls[i] = lst;
                stk.push({i, a[i]});
            } else {
                rs[stk.top().x] = i;
                ls[i] = lst;
                stk.push({i, a[i]});
            }
```

```
            }
        }
        for (int i = 1;i <= n;i ++) {
            cout << ls[i] << ' ';
        }
        cout << '\n';
        for (int i = 1;i <= n;i ++) {
            cout << rs[i] << ' ';
        }
        cout << '\n';
    }
```

重链剖分

```
// 重链剖分
struct SegTree{
    #define lc p << 1
    #define rc p << 1 | 1
    struct node{
        LL l, r, val;
        node(){l = r = val = 0;}
    };
    vector<node> f;
    vector<LL> a;
    SegTree(int x){f.resize(x << 2 | 3), a.resize(x + 1, 0);}
    void pushup(int p){
        f[p].val = max(f[lc].val, f[rc].val);
    }
    void build(int p, int l, int r){
        f[p].l = l, f[p].r = r;
        if (l == r){
            f[p].val = a[l];
            return;
        }
        int mid = l + r >> 1;
        build(lc, l, mid);
        build(rc, mid + 1, r);
        pushup(p);
    }
    void upd(int p, int id, int val){
        if (f[p].l == f[p].r){
            f[p].val = val;
            return;
        }
        int mid = f[p].l + f[p].r >> 1;
        if (id <= mid) upd(lc, id, val);
        else upd(rc, id, val);
        pushup(p);
    }
    LL qry(int p, int l, int r){
        if (l <= f[p].l && f[p].r <= r) return f[p].val;
```

```
        int mid = f[p].l + f[p].r >> 1;
        LL Max = 0;
        if (l <= mid) Max = max(Max, qry(lc, l, r));
        if (r > mid) Max = max(Max, qry(rc, l, r));
        return Max;
    }
};

int main(){
    int n, u, v;
    cin >> n;
    vector<vector<int>> g(n + 1);
    vector<int> a(n + 1, 0);
    for (int i = 1;i <= n - 1;i ++){
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for (int i = 1;i <= n;i ++) cin >> a[i];
    /*
    * dfs1
    * 1.先维护每个节点的父节点(fa)
    * 2.维护每个节点的深度(dep)
    * 3.维护每个节点的子树大小(siz)
    * 4.维护每个节点最重的孩子(son)
    */
    vector<int> fa(n + 1, 0), vis(n + 1, 0), dep(n + 1, 0), siz(n + 1, 1), son(n +
1, 0);
    auto dfs1 = [&](auto dfs1, int pu, int u) -> void {
        fa[u] = pu;
        siz[u] = 1;
        dep[u] = dep[pu] + 1;
        int Max = 0;
        for (auto v : g[u]){
            if (v == pu) continue;
            vis[v] = 1;
            dfs1(dfs1, u, v);
            siz[u] += siz[v];
            if (siz[v] > Max){
                Max = siz[v];
                son[u] = v;
            }
        }
    };
    /*
    * dfs2
    * 1.利用重孩子，求重链头节点(top)
    * 2.求dfn序(dfn)
    * 3.求dfn序对应的数值(seg)
    */
    vector<int> top(n + 1, 0), dfn(n + 1, 0), seg(n + 1, 0);
    SegTree f(n);
    f.build(1, 1, n);
    int cur = 0;
```

```
    auto dfs2 = [&](auto dfs2, int head, int u) -> void {
        top[u] = head;
        dfn[u] = ++ cur;
        seg[cur] = u;
        f.upd(1, cur, a[u]);
        if (son[u])
            dfs2(dfs2, head, son[u]);
        for (auto v : g[u]){
            if (v == fa[u] || v == son[u]) continue;
            dfs2(dfs2, v, v);
        }
    };
    dfs1(dfs1, 0, 1);
    dfs2(dfs2, 1, 1);
    /**
     * 重剖求LCA
     */
    auto lca = [&](int u, int v) -> LL {
        while (dfn[top[u]] != dfn[top[v]]){
            if (dfn[top[u]] > dfn[top[v]]){
                u = fa[top[u]];
            } else{
                v = fa[top[v]];
            }
        }
        if (dfn[u] > dfn[v]) swap(u, v);
        return u;
    };
    /**
     * 重剖求简单路径最大(点)权
     */
    auto MaxNode = [&](int u, int v) -> LL {
        LL Max = 0;
        while (dfn[top[u]] != dfn[top[v]]){
            if (dfn[top[u]] > dfn[top[v]]){
                Max = max(Max, f.qry(1, dfn[top[u]], dfn[u]));
                u = fa[top[u]];
            } else{
                Max = max(Max, f.qry(1, dfn[top[v]], dfn[v]));
                v = fa[top[v]];
            }
        }
        if (dfn[u] > dfn[v]) swap(u, v);
        Max = max(Max, f.qry(1, dfn[u], dfn[v]));
        return Max;
    };
    // int q;
    // cin >> q;
    // while (q --){
    //     int u, v;
    //     cin >> u >> v;
    //     // cout << MaxNode(u, v) << '\n';
    //     // cout << lca(u, v) << '\n';
```

```
    // }
}
```

KMP

```
vector<int> kmp(string t,string s){
    string str = t + '\0' + s;
    vector<int> pi(str.size(),0);
    for (int i=1;i<str.size();i++){
        int len = pi[i-1];
        while (len != 0 && str[i] != str[len]){
            len = pi[len - 1];
        }
        pi[i] = len + (str[i] == str[len]);
    }
    return pi;  // 最长前后缀匹配
}
```

卢卡斯(快速求组合数，杨辉三角奇偶性)

```
LL fac[N];
LL qmi(LL a,LL k,LL p){
    LL res = 1;
    while (k){
        if (k&1) (res *= a) %= p;
        a = a * a % p;
        k >>= 1;
    }
    return res;
}
LL inv(LL a,LL p){
    LL res = 1;
    LL k = p - 2;
    while (k){
        if (k & 1) (res *= a) %= p;
        (a *= a) %= p;
        k >>= 1;
    }
    return res;
}
LL C(LL m, LL n, LL p){
    if (m > n) return 0;
    return fac[n] * inv(fac[m], p) % p * inv(fac[n - m], p) % p;
}
LL lucas(LL m, LL n, LL p){
    if (m == 0) return 1;
    return lucas(m / p, n / p, p) * C(m % p, n % p, p) % p;
}
```

统计1~n之间所有数数位和

```
int main() {
    LL n;
    cin >> n;
    auto cal = [&](LL num) -> LL {
        LL base = 1, len, val = 0;
        len = to_string(num).size();
        for (int i = 1;i <= len;i ++) {
            int cur = num / base % 10;
            for (int j = 0;j <= 9;j ++) {
                if (j < cur) {
                    val += j * (num / base / 10 + 1) * base;
                } else if (j == cur) {
                    val += j * (num / base / 10) * base;
                    val += j * (num % base + 1);
                } else {
                    val += j * (num / base / 10) * base;
                }
            }
            base *= 10;
        }
        return val;
    };
    cout << cal(n) << '\n';
}
```

辛普森公式求弧长公式定积分

```
// 2025杭电暑假多校第4场, 1012
// 辛普森公式求弧长公式定积分
auto f = [&](double x) -> double {
    double y = 0;
    for (int i = m;i >= 1;i --){
        y += a[i] * cal(x, i);
    }
    return y;
};
auto df = [&](double x) -> double {
    double y = 0;
    for (int i = m;i >= 1;i --){
        y += i * a[i] * cal(x, i - 1);
    }
    return y;
};
auto fdf = [&](double x) -> double {
    double y = sqrt(1 + cal(df(x), 2));
    return y;
};
auto simpson = [&](double l, double r) -> double {
```

```
    double v = (r - l) / 6.0 * (fdf(l) + 4 * fdf((l + r) / 2) + fdf(r));
    return v;
};
auto dis = [&](auto dis, double l, double r, double v) -> double {
    double mid = (l + r) / 2;
    double L = simpson(l, mid), R = simpson(mid, r);
    if (fabs(L + R - v) > eps * 0.01){
        return dis(dis, l, mid, L) + dis(dis, mid, r, R);
    }
    return L + R + (L + R - v) / 0.01;
};
```

kruskal重构树

```
/*
* kruskal重构树
* 2025/08/09
* By Foracy
*
* 原理：图上两点间任意路径最小的最大权
*     = kruskal重构树上两点的最近公共祖先(lca)的点权
* 作用：若单次查询，可用prim求最小生成树(MST)，时间复杂度为O(n)
*     若多次查询，整体O(qn)难以通过，可构建重构树，利用lca求解，单次复杂度为O(logn)
*
* 主要流程：
* 1. 读取图的点数和边数，初始化并查集和重构树结构。
* 2. 按边权升序排序所有边，依次合并不连通的点，构建kruskal重构树。
* 3. 使用DFS预处理每个节点的深度和倍增祖先表，用于后续LCA查询。
* 4. 对每组查询，利用LCA算法求出两点在重构树上的最近公共祖先，并输出该祖先的点权（即原图两点路径的最小最大权）。
*
* 主要变量说明：
* - edge结构体：表示一条边，包含起点u、终点v和权值w。
* - f[]：并查集数组，用于维护连通性。
* - val[]：每个节点的权值，重构树中新节点权值为合并时的边权。
* - g[]：邻接表，存储重构树结构。
* - dep[]：每个节点的深度。
* - fa[][]：倍增祖先表，fa[u][i]表示u的第2^i级祖先。
*
* 关键函数说明：
* - findx：并查集查找带路径压缩。
* - merge：合并两个集合，生成新节点并更新重构树结构。
* - dfs：深度优先遍历，预处理深度和祖先表。
* - lca：倍增法求最近公共祖先。
*
* 时间复杂度：
* - 构建重构树：O(m log n)
* - 单次查询：O(log n)
*/

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

```cpp
#define all(x) begin(x), end(x)
#define siz(x) ((int) x.size())
using namespace std;
using LL = long long;

const LL inf = 1e15 + 10;

struct edge
{
    LL u, v, w;
    bool operator < (const edge&that) const {
        return w < that.w;
    }
};

int main(){
    int n, m, cur;
    cin >> n >> m;
    cur = n;
    vector<edge> p;
    vector<vector<LL>> g(n << 1 | 1);
    vector<LL> f(n << 1 | 1, 0), val(n << 1 | 1, inf);
    for (int i = 1;i <= n;i ++) f[i] = i;
    auto findx = [&](auto findx, int x) -> LL {
        if (f[x] != x){
            f[x] = findx(findx, f[x]);
        }
        return f[x];
    };
    auto merge = [&](int u, int v, int w) -> void {
        u = findx(findx, u);
        v = findx(findx, v);
        if (u != v){
            f[u] = f[v] = ++ cur;
            f[cur] = cur;
            val[cur] = w;
            g[cur].push_back(u);
            g[u].push_back(cur);
            g[cur].push_back(v);
            g[v].push_back(cur);
        }
    };
    for (int i = 1;i <= m;i ++){
        int u, v, w;
        cin >> u >> v >> w;
        p.push_back({u, v, w});
    }
    sort(all(p));
    for (int i = 0;i < m;i ++){
        if (findx(findx, p[i].u) != findx(findx, p[i].v)){
            merge(p[i].u, p[i].v, p[i].w);
        }
    }
    vector<LL> dep(n << 1 | 1, 0);
```

```
    vector<vector<LL>> fa(n << 1 | 1, vector<LL> (20, 0));
    auto dfs = [&](auto dfs, int pu, int u) -> void {
        dep[u] = dep[pu] + 1;
        fa[u][0] = pu;
        for (int i = 1;i <= 19;i ++){
            fa[u][i] = fa[fa[u][i - 1]][i - 1];
        }
        for (auto v : g[u]){
            if (v == pu) continue;
            dfs(dfs, u, v);
        }
    };
    dfs(dfs, cur, cur);
    auto lca = [&](int u, int v) -> LL {
        if (dep[u] < dep[v]) swap(u, v);
        for (int i = 19;i >= 0;i --){
            if (dep[fa[u][i]] >= dep[v]){
                u = fa[u][i];
            }
        }
        if (u == v) return u;
        for (int i = 19;i >= 0;i --){
            if (fa[u][i] != fa[v][i]){
                u = fa[u][i];
                v = fa[v][i];
            }
        }
        return fa[u][0];
    };
    int q;
    cin >> q;
    while (q --){
        int u, v;
        cin >> u >> v;
        cout << val[lca(u, v)] << '\n';
    }
}
```

区间修改线段树

```
struct SegmentTree{
    #define lc p << 1
    #define rc p << 1 | 1
    #define mid(l,r) (l + r >> 1)
    struct node{
        int l,r,val,laz;
        node(int x = 0){
            l = r = val = laz = x;
        }
        node friend operator+(node a,node b){
            node res;
```

```
            res.val = a.val + b.val;
            res.l = min(a.l, b.l);
            res.r = max(a.r, b.r);
            return res;
        }
    };
    vector<node> tr;
    vector<int> a;
    SegmentTree(int n){
        tr.resize(4 * n + 4);
        a.resize(n + 1);
    }
    void pushdown(int p){
        if (tr[p].laz){
            tr[lc].laz += tr[p].laz;
            tr[lc].val += (tr[lc].r - tr[lc].l + 1) * tr[p].laz;
            tr[rc].laz += tr[p].laz;
            tr[rc].val += (tr[rc].r - tr[rc].l + 1) * tr[p].laz;
            tr[p].laz = 0;
        }
    }
    void build(int p,int l,int r){
        tr[p].l = l, tr[p].r = r;
        if (l == r){
            tr[p].val = a[l];
            return;
        }
        build(lc, l, mid(l,r));
        build(rc, mid(l,r) + 1, r);
        tr[p] = tr[lc] + tr[rc];
    }
    void update(int p,int l,int r,int x){
        if (l <= tr[p].l && tr[p].r <= r){
            tr[p].laz += x;
            tr[p].val += x * (tr[p].r - tr[p].l + 1);
            return;
        }
        pushdown(p);
        int mid = mid(tr[p].l,tr[p].r);
        if (l <= mid) update(lc, l, r, x);
        if (r > mid) update(rc, l, r, x);
        tr[p] = tr[lc] + tr[rc];
    }
    node query(int p,int l,int r){
        if (l <= tr[p].l && tr[p].r <= r) return tr[p];
        pushdown(p);
        int mid = mid(tr[p].l, tr[p].r);
        node res;
        if (l <= mid) res = res + query(lc, l, r);
        if (r > mid) res = res + query(rc, l, r);
        return res;
    }
    #undef lc
    #undef rc
```

```
    #undef mid
};
```

矩阵乘法

```
struct Matrix{
    int R, C;
    vector<vector<LL>> mat;
    Matrix(int row = 2, int cal = 2){
        R = row;
        C = cal;
        mat.resize(row + 1, vector<LL>(cal + 1, 0));
    }
    Matrix friend operator * (Matrix A, Matrix B){
        int r = A.R;
        int c = B.C;
        int x = A.C;
        Matrix C(r, c);
        for (int i = 1;i <= r;i ++){
            for (int j = 1;j <= c;j ++){
                for (int k = 1;k <= x;k ++){
                    (C.mat[i][j] += A.mat[i][k] * B.mat[k][j] % MOD) %= MOD;
                }
            }
        }
        return C;
    }
};
```

LCA

```
vector<int> dep(n + 1, 0);
vector<vector<int>> fa(n + 1, vector<int>(20, 0));
auto dfs = [&](auto dfs, int pu, int u) -> void {
    dep[u] = dep[pu] + 1;
    fa[u][0] = pu;
    for (int i = 1;i <= 19;i ++){
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    }
    for (auto v : g[u]){
        if (v == pu) continue;
        dfs(dfs, u, v);
    }
};
auto lca = [&](int u, int v) -> int {
    if (dep[u] < dep[v]) swap(u, v);
    for (int i = 19;i >= 0;i --){
        if (dep[fa[u][i]] >= dep[v]){
            u = fa[u][i];
```

```
        }
    }
    if (u == v) return u;
    for (int i = 19;i >= 0;i --){
        if (fa[u][i] != fa[v][i]){
            u = fa[u][i];
            v = fa[v][i];
        }
    }
    return fa[u][0];
};
```

分块

- 整除分块

```
void solve()
{
    cin >> n;

    int l = 1, r;
    while(l <= n)
    {
        int r = n / (n / l);
        l = r + 1;
    }
}
```

可持久化线段树（主席树，[l, r] 区间第k小）

```
struct node
{
    int l, r;
    int cnt;
}tr[(N << 2) + N * 17];

int root[N], idx;

int find(int x)
{
    return lower_bound(vec.begin(), vec.end(), x) - vec.begin();
}

int build(int l, int r)
{
    int p = ++ idx;
    if(l == r) return p;
    int mid = l + r >> 1;
    tr[p].l = build(l, mid);
```

```
    tr[p].r = build(mid + 1, r);
    return p;
}

//动态开点
int insert(int p, int l, int r, int x)
{
    int q = ++ idx;
    tr[q] = tr[p];
    if(l == r)
    {
        tr[q].cnt ++;
        return q;
    }
    int mid = l + r >> 1;
    if(x <= mid) tr[q].l = insert(tr[p].l, l, mid, x);
    else tr[q].r = insert(tr[p].r, mid + 1, r, x);
    tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
    return q;
}

//若k比cnt小则走左子树
//若k比cnt大则走右子树，为右子树第k-cnt小
int ask(int q, int p, int l, int r, int k)
{
    if(l == r) return l;
    int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;
    int mid = l + r >> 1;
    if(k <= cnt) return ask(tr[q].l, tr[p].l, l, mid, k);
    return ask(tr[q].r, tr[p].r, mid + 1, r, k - cnt);
}

void solve()
{
    cin >> n >> m;

    for(int i = 1; i <= n; i ++ ) cin >> a[i], vec.pb(a[i]);

    sort(vec.begin(), vec.end());
    vec.erase(unique(vec.begin(), vec.end()), vec.end());
    root[0] = build(0, vec.size() - 1);

    for(int i = 1; i <= n; i ++) root[i] = insert(root[i - 1], 0, vec.size() - 1,
find(a[i]));

    while(m -- )
    {
        int l, r, k;
        cin >> l >> r >> k;
        cout << vec[ask(root[r], root[l - 1], 0, vec.size() - 1, k)] << endl;
    }
}
```

字符串哈希

```
using ULL = unsigned long long;
const int P = 131;
const int N = 1e5+10;
ULL p[N], h[N];
/*
求一个字符串的哈希值相当于求前缀和
求一个字符串的子串相当于求区间和
*/
// 预处理hash函数的前缀和
void init(){
    p[0] = 1, h[0] = 0;
    for (int i=1;i<=n;i++){
        p[i] = p[i-1] * P;
        h[i] = h[i-1] * P + s[i];
    }
}
// 计算s[l~r]的hash值
ULL get(int l,int r){
    return h[r] - h[l-1] * p[r-l+1];
}
// 判断两字串是否相同
bool substr(int l1,int r1,int l2,int r2){
    return get(l1,r1) == get(l2,r2);
}
```

st表

```
struct RMQ{
    int n;
    vector<vector<int>> st;
    RMQ(int x = 1e5){
        n = x;
        st.resize(x + 1,vector<int>(21));
    }
    void build(vector<int> arr){
        for (int i = 1;i <= n;i ++){
            st[i][0] = arr[i];
        }
        for (int i = 1;i <= 20;i ++){
            for (int j = 1;j + (1LL << i) - 1 <= n;j ++){
                st[j][i] = max(st[j][i - 1],st[j + (1LL << i - 1)][i - 1]);
            }
        }
    }
    int query(int l,int r){
        int k = log2(r - l + 1);
        return max(st[l][k], st[r - (1LL << k) + 1][k]);
```

```
    }
};
```

并查集（启发式合并）

```
struct DSU{
    vector<int> dsu;
    vector<int> siz;
    int n;
    DSU(int len = 1e5){
        n = len;
        dsu.resize(n+1,0);
        siz.resize(n+1,1);
    }
    void init(){
        for (int i = 1;i <= n;i ++) dsu[i] = i;
        for (int i = 0;i <= n;i ++) siz[i] = 1;
    }
    int findx(int x){
        if (dsu[x] != x){
            siz[x] += siz[dsu[x]];
            dsu[x] = findx(dsu[x]);
        }
        return dsu[x];
    }
    void merge(int a,int b){
        a = findx(a);
        b = findx(b);
        if (a < b) swap(a,b);
        if (a != b){
            dsu[a] = b;
            siz[b] += siz[a];
        }
    }
};
```

数学

- 欧拉降幂

```
/**
 * 求解a^k mod p
 * p 为质数，但是k非常大，数量级为10^1e5
 * 可以将k换成 k mod phi(p)
 * phi(p) 为 p 的欧拉函数值
 */
```

dp

- 数位dp

```
int cal(int num){
    vector<int> p;
    while (num){
        p.push_back(num % 10);
        num /= 10;
    }
    p.push_back(0);
    reverse(p.begin(), p.end());
    int len = p.size() - 1;
    int dp[len + 1][2][10];
    memset(dp, 0, sizeof dp);
    for (int i = 1;i <= len;i ++){
        for (int x = 1;x <= (i == 1 ? p[i] : 9);x ++){ //
            dp[i][(i == 1 && x == p[i])][x] ++;
        }
        for (int limit = 0;limit <= 1;limit ++){
            for (int x = 0;x <= (limit ? p[i] : 9);x ++){
                for (int last = 0;last <= 9;last ++){
                    dp[i][(limit && x == p[i])][x] += dp[i - 1][limit][last];
                }
            }
        }
    }
    LL sum = 0;
    for (int i = 0;i <= 9;i ++){
        sum += dp[len][0][i];
        sum += dp[len][1][i];
    }
    return sum;
}
```

拓欧

```
LL gcd(LL a, LL b){
    return (b ? gcd(b, a % b) : a);
}
LL exgcd(LL a, LL b, LL &x, LL &y){
    if (!b){
        x = 1, y = 0;
        return a;
    }
    LL d = exgcd(b, a % b, y, x); // 辗转相除，并交换系数
    /*
    裴属定理：ax + by = gcd(a, b)
    拓欧
        by + (a % b)x = gcd(b, a % b)
      = by + (a - ⌊a / b⌋ * b)x
      = by + ax - ⌊a / b⌋ * b * x
```

```
      = ax + b * (y - ⌊a / b⌋ * x)
    */
    y -= a / b * x;
    return d;
}
int main(){
    int T;
    cin >> T;
    while (T --){
        LL a, b, x, y;
        cin >> a >> b;
        exgcd(a, b, x, y);
        cout << x << ' ' << y << '\n';
    }
}
```

tarjan求强连通分量

```
int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> g(n + 1);
    for (int i = 1;i <= m;i ++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
    }
    vector<int> dfn(n + 1, 0), low(n + 1, 0), scc(n + 1, 0), vis(n + 1, 0);
    stack<int> stk;
    int cur = 0, tot = 0;
    auto tarjan = [&](auto tarjan, int u) -> void {
        dfn[u] = low[u] = ++ cur;
        vis[u] = 1;
        stk.push(u);
        for (auto v : g[u]) {
            if (!dfn[v]) {
                tarjan(tarjan, v);
                low[u] = min(low[u], low[v]);
            } else if (vis[v]) {
                low[u] = min(low[u], dfn[v]);
            }
        }
        if (low[u] == dfn[u]) {
            tot ++;
            int v;
            do {
                v = stk.top();
                stk.pop();
                scc[v] = tot;
                vis[v] = 0;
            } while (v != u);
```

```
        }
    };
    for (int i = 1;i <= n;i ++) {
        if (!dfn[i]) {
            tarjan(tarjan, i);
        }
    }
}
```

建虚树

```
int dfn[MAXN];
int h[MAXN], m, a[MAXN], len;  // 存储关键点

bool cmp(int x, int y) {
  return dfn[x] < dfn[y];  // 按照 dfs 序排序
}

void build_virtual_tree() {
  sort(h + 1, h + m + 1, cmp);  // 把关键点按照 dfs 序排序
  for (int i = 1; i < m; ++i) {
    a[++len] = h[i];
    a[++len] = lca(h[i], h[i + 1]);  // 插入 lca
  }
  a[++len] = h[m];
  sort(a + 1, a + len + 1, cmp);  // 把所有虚树上的点按照 dfs 序排序
  len = unique(a + 1, a + len + 1) - a - 1;  // 去重
  for (int i = 1, lc; i < len; ++i) {
    lc = lca(a[i], a[i + 1]);
    conn(lc, a[i + 1]);  // 连边，如有边权 就是 distance(lc,a[i+1])
  }
}
```

Manacher(回文串匹配)

```
void fxy_ac(){
    string s;
    cin >> s;
    s = ' ' + s;
    int R = 0, mid, ans = 0;
    vector<int> p(siz(s), 0);
    for (int i = 1;i < siz(s);i ++){
        if (i < R) p[i] = min(p[2 * mid - i], R - i);
        else p[i] = 1;
        while (s[i - p[i]] == s[i + p[i]]) p[i] ++;
        if (i + p[i] > R){
            R = i + p[i];
            mid = i;
        }
```

```
        ans = max(ans, p[i] * 2 - 1);
    }
    R = 0;
    for (int i = 1;i < siz(s) - 1;i ++){
        if (s[i] != s[i + 1]) continue;
        if (i < R) p[i] = min(p[2 * mid - i], R - i);
        else p[i] = 1;
        while (s[i - p[i]] == s[i + p[i] + 1]) p[i] ++;
        if (i + p[i] > R){
            R = i + p[i];
            mid = i;
        }
        ans = max(ans, p[i] * 2);
    }
    cout << ans << '\n';
}
```

AC自动机

```
constexpr int N = 2e5 + 6;
constexpr int LEN = 2e6 + 6;
constexpr int SIZE = 2e5 + 6;

int n;

namespace AC {
struct Node {
  int son[26];  // 子结点
  int ans;      // 匹配计数
  int fail;     // fail 指针
  int du;       // 入度
  int idx;

  void init() {  // 结点初始化
    memset(son, 0, sizeof(son));
    ans = fail = idx = 0;
  }
} tr[SIZE];

int tot;  // 结点总数
int ans[N], pidx;

void init() {
  tot = pidx = 0;
  tr[0].init();
}

void insert(char s[], int &idx) {
  int u = 0;
  for (int i = 1; s[i]; i++) {
    int &son = tr[u].son[s[i] - 'a'];  // 下一个子结点的引用
```

```
    if (!son) son = ++tot, tr[son].init();  // 如果没有则插入新结点，并初始化
    u = son;                                // 从下一个结点继续
  }
  // 由于有可能出现相同的模式串，需要将相同的映射到同一个编号
  if (!tr[u].idx) tr[u].idx = ++pidx;  // 第一次出现，新增编号
  idx = tr[u].idx;  // 这个模式串的编号对应这个结点的编号
}

void build() {
  queue<int> q;
  for (int i = 0; i < 26; i++)
    if (tr[0].son[i]) q.push(tr[0].son[i]);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int i = 0; i < 26; i++) {
      if (tr[u].son[i]) {                              // 存在对应子结点
        tr[tr[u].son[i]].fail = tr[tr[u].fail].son[i];  // 只用跳一次 fail 指针
        tr[tr[tr[u].fail].son[i]].du++;                 // 入度计数
        q.push(tr[u].son[i]);                           // 并加入队列
      } else
        tr[u].son[i] =
            tr[tr[u].fail]
                .son[i];  // 将不存在的字典树的状态链接到了失配指针的对应状态
    }
  }
}

void query(char t[]) {
  int u = 0;
  for (int i = 1; t[i]; i++) {
    u = tr[u].son[t[i] - 'a'];  // 转移
    tr[u].ans++;
  }
}

void topu() {
  queue<int> q;
  for (int i = 0; i <= tot; i++)
    if (tr[i].du == 0) q.push(i);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    ans[tr[u].idx] = tr[u].ans;
    int v = tr[u].fail;
    tr[v].ans += tr[u].ans;
    if (!--tr[v].du) q.push(v);
  }
}
}  // namespace AC

char s[LEN];
int idx[N];
```

```
int main() {
  AC::init();
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) {
    scanf("%s", s + 1);
    AC::insert(s, idx[i]);
    AC::ans[i] = 0;
  }
  AC::build();
  scanf("%s", s + 1);
  AC::query(s);
  AC::topu();
  for (int i = 1; i <= n; i++) {
    printf("%d\n", AC::ans[idx[i]]);
  }
  return 0;
}
```

线性基

```
using ull = unsigned long long;
ull p[64];
void insert(ull x) {
  for (int i = 63; ~i; --i) {
    if (!(x >> i))  // x 的第 i 位是 0
      continue;
    if (!p[i]) {
      p[i] = x;
      break;
    }
    x ^= p[i];
  }
}
int main() {
  int n;
  cin >> n;
  ull a;
  for (int i = 1; i <= n; ++i) {
    cin >> a;
    insert(a);
  }
  ull ans = 0;
  for (int i = 63; ~i; --i) {
    ans = max(ans, ans ^ p[i]);
  }
  cout << ans << '\n';
  return 0;
}
```

拓扑排序

```cpp
void topo() {
  vector<int> L;
  queue<int> S;
  for (int i = 1; i <= n; i++)
    if (in[i] == 0) S.push(i);
  while (!S.empty()) {
    int u = S.front();
    S.pop();
    L.push_back(u);
    for (auto v : G[u]) {
      if (--in[v] == 0) {
        S.push(v);
      }
    }
  }
  if (L.size() == n) {
    for (auto i : L) cout << i << ' ';
  }
}
```

分组背包

```cpp
for (int k = 1; k <= ts; k++)           // 循环每一组
  for (int i = m; i >= 0; i--)          // 循环背包容量
    for (int j = 1; j <= cnt[k]; j++)   // 循环该组的每一个物品
      if (i >= w[t[k][j]])              // 背包容量充足
        dp[i] = max(dp[i],dp[i - w[t[k][j]]] + c[t[k][j]]);  // 像0-1背包一样状态转
移
```

Floyd最短路（含修改边权）

```cpp
// ABC416 E
void fxy_ac(){
    int N, M;
    cin >> N >> M;
    // dp[i][j] 表示i->j的最短路径
    // dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j])
    // 先遍历k，再遍历i, j
    vector<vector<LL>> dp(N + 1, vector<LL> (N + 1, inf));
    for (int i = 0;i <= N;i ++) dp[i][i] = 0;
    for (int i = 1;i <= M;i ++){
        LL u, v, w;
        cin >> u >> v >> w;
        dp[u][v] = min(dp[u][v], w);
        dp[v][u] = min(dp[v][u], w);
    }
    LL K, T, D;
    cin >> K >> T;
```

```
    // 设0点为中转点，所有包含机场的点都会经过该中转点
    // 从机场点到中转点的边权为T，从中转点到机场点的边权为0
    for (int i = 1;i <= K;i ++){
        cin >> D;
        dp[D][0] = min(dp[D][0], T);
        dp[0][D] = 0;
    }
    for (int k = 0;k <= N;k ++){
        for (int u = 0;u <= N;u ++){
            for (int v = 0;v <= N;v ++){
                dp[u][v] = min(dp[u][v], dp[u][k] + dp[k][v]);
            }
        }
    }
    int Q;
    cin >> Q;
    while (Q --){
        int op;
        cin >> op;
        if (op == 1){
            LL u, v, w;
            cin >> u >> v >> w;
            dp[u][v] = min(dp[u][v], w);
            dp[v][u] = min(dp[v][u], w);
            for (int i = 0;i <= N;i ++){
                for (int j = 0;j <= N;j ++){
                    dp[i][j] = min(dp[i][j], dp[i][u] + dp[u][v] + dp[v][j]);
                    dp[i][j] = min(dp[i][j], dp[i][v] + dp[v][u] + dp[u][j]);
                }
            }
        } else if (op == 2){
            int d;
            cin >> d;
            dp[d][0] = min(dp[d][0], T);
            dp[0][d] = 0;
            for (int i = 0;i <= N;i ++){
                for (int j = 0;j <= N;j ++){
                    dp[i][j] = min(dp[i][j], dp[i][d] + dp[d][0] + dp[0][j]);
                    dp[i][j] = min(dp[i][j], dp[i][0] + dp[0][d] + dp[d][j]);
                }
            }
        } else if (op == 3){
            LL ans = 0;
            for (int i = 1;i <= N;i ++){
                for (int j = 1;j <= N;j ++){
                    if (dp[i][j] != inf){
                        ans += dp[i][j];
                    }
                }
            }
            cout << ans << '\n';
        }
    }
}
```

```
```

对拍

- 随机数

```
LL random(int l,int r){
    return (rand() % (r - l + 1) + l);
}
LL R1(LL mod){
    LL ans = 2147483647;
    return ans = ans * rand() % mod + 1;
}
int main(){
    struct _timeb T;
    _ftime(&T);
    srand(T.millitm);
}
// --------
    system("g++ ../std.cpp -o std.exe");
    system("g++ ../vio.cpp -o vio.exe");
    system("g++ ../dat.cpp -o dat.exe");
    for (int i = 1;i <= n;i ++){
        system("dat.exe > in.txt");
        system("vio.exe < in.txt > vio.txt");
        double begin = clock();
        printf("Running in test %d ...\n",i);
        system("std.exe < in.txt > std.txt");
        double end = clock();
        double t = end - begin;
        string info = "In Test "+to_string(i)+" Time : "+to_string(t) + " ms";
        if (system("fc std.txt vio.txt")){
            res.push_back(ColorStr(255,0,0,"Wrong Answer\n") +
ColorStr(-1,-1,-1,info));
            system("std.exe < in.txt > ../Output.txt");
            system("vio.exe < in.txt > ../Answer.txt");
            break;
        } else if (t > Timelim){
            res.push_back(ColorStr(255,0,0,"Time Limited Exceeded\n") +
ColorStr(-1,-1,-1,info));
        } else{
            res.push_back(ColorStr(0,255,0,"Accpeted\n") +
ColorStr(-1,-1,-1,info));
        }
        system("cls");
    }
```

nlogn筛

```
```

```
int Era(int n){
    int k = 0;
    for (int i=2;i*i<=n;i++){
        if (!sieve[i]){
            for (int j=i*i;j<=n;j+=i){
                sieve[j] = 1;
            }
        }
    }
    for (int i=2;i<=n;i++){
        if (!sieve[i]){
            prime[k++] = i;
        }
    }
    return k;
}
```

tricks

- 在考虑**区间和** 与 **区间长度** 相同的问题时，我们可以将式子 $$Pre_r - Pre_{l - 1} = r - (l - 1)$$ 变为 $$Pre_r - r = Pre_{l - 1} - (l - 1)$$ 对具有相同特征$Pre_i - i$进行分组并累加，即可将统计相同数量的复杂度从$O(n^2)$缩为$O(n)$// *CF1398C*

- 反 Nim 游戏
- 规定：字母 N 和 P 分别代表先手必胜与必败。

一个局面为 N 态的充要条件是有至少一条出边连接至 P 态。

一个局面为 P 态的充要条件是每一条出边都连接到 N 态。

为方便书写，用字母 T 表示 $\oplus_{i=1}^{n}a_{i}$。

- 结论：

1. 当全部 $a_{i}=1$，如果有奇数堆石子就为 P 态，有偶数堆则为 N 态。
2. 当至少一个 $a_{i}>1，T\neq 0$ 时为 N 态，否则为 P 态。

other

- 线性基

```
void ins(ll x){
    for(int i=55;i>=0;i--){
        if(!(x&(1ll<<i)))continue;
        if(d[i])x^=d[i];//eliminate the 1 on the i-th bit of x
        else{d[i]=x;break;}//successfully inserted, jump out.
    }
}
```

- 数位dp 最基本的模板，求的是区间 [0,n] 中满足条件的数的个数：

```
int dfs(int u, int high) {
    if (u == s.size()) return 1;
    if (!high && f[u] != -1) return f[u];
    int l = 0, r = high ? s[u] - '0' : 9;
    int ret = 0;
    for (int i = l; i <= r; i++) {
        ret += dfs(u + 1, high && i == r);
    }
    if (!high) f[u] = ret;
    return ret;
}
```

> 这个版本是允许数字有前导零的，可以这么做的前提是有前导零不会影响答案。
> 　　代码中的 s 是对 n 转换成字符串后的结果。
> 　　参数中的 u 是指当前在第 u 位填数字，数字是从最高位开始依次往低位填的。high 是一个 bool 变量，如果是 1 表示前面填的数字都是 n 对应位上的数字，那么第 u 位能填的数字只能是 [0,s[u]]；否则如果是 0 表示前面至少存在一个位填的数字小于 n 对应位上的数字，那么第 u 位能填的数字可以是 [0,9]。
> 　　dfs(u, high) 返回的是从第 u 位开始填，第 u 位前面填的数字是否都贴着上界，所能构造出满足条件的数的数量。边界条件是 u 等于 n 的数位长度，此时返回 1（有其他条件的话还要判断是否满足）。
> 　　l 和 r 对应第 u 位上能填的数字的范围，其中在这个模板中 l 都是 0，r 受 high 的影响。
> 　　f(u) 是为了实现记忆化搜索，其实可以把 f 扩展成 f(u,0/1)，实现时之所以不用记录 high 那维，是因为当 high=1 时必然只会搜一次（标记，这里其实我现在也不是很理解）。
> 　　最后调用的方法是 dfs(0, 1)，一开始置 high=1 是因为第 0 位能填的数字范围只能是 [0,s[0]]。

```
int dp(int n) {
    s = to_string(n);
    memset(f, -1, sizeof(f));
    return dfs(0, 1, 1);
}
```