Leonard‘s template

基础算法

快速排序

```
void qsort(int l, int r) {
    int mid = a[l + r >> 1], i = l, j = r;
    while(i <= j) {
        while(a[i] < mid) i++;
        while(a[j] > mid) j--;
        if(i <= j) {
            swap(a[i], a[j]);
            i++;
            j--;
        }
    }
    if(l < j) qsort(l, j);
    if(i < r) qsort(i, r);
}
```

归并排序

```
void merge_sort(int l, int r) {
    int mid = l + r >> 1;
    if(l != r) merge_sort(l, mid);
    if(l != r) merge_sort(mid + 1, r);
    int k = 1, i = l, j = mid + 1;
    while(i <= mid && j <= r) {
        if(a[i] <= a[j]) tmp[k++] = a[i++];
        else tmp[k++] = a[j++];
    }
    while(i <= mid) tmp[k++] = a[i++];
    while(j <= r) tmp[k++] = a[j++];
    for(int i = l, j = 1; i <= r; i++) a[i] = tmp[j++];
}
```

逆序对的数量

```
i64 merge_sort(int l, int r) {
    i64 res = 0;
    i64 mid = l + r >> 1;
    if(l != r) res += merge_sort(l, mid);
    if(l != r) res += merge_sort(mid + 1, r);
    int k = 1, i = l, j = mid + 1;
    while(i <= mid && j <= r) {
        if(a[i] <= a[j]) tmp[k++] = a[i++];
        else {
            tmp[k++] = a[j++];
            res += mid - i + 1;
        }
    }
    while(i <= mid) tmp[k++] = a[i++];
    while(j <= r) tmp[k++] = a[j++];
    for(int i = l, j = 1; i <= r; i++) a[i] = tmp[j++];
    return res;
}
```

整数二分

```
bool check(int x)
// 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用：
int bsearch_1(int l, int r) {
    while (l < r) {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }
    return l;
}
// 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用：
int bsearch_2(int l, int r) {
    while (l < r) {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
```

```
}
```

浮点数二分

```
bool check(double x)
double bsearch(double l, double r) {
    const double eps = 1e-6;   // eps 表示精度，取决于题目对精度的要求
    while (r - l > eps) {
        double mid = (l + r) / 2;
        if (check(mid)) {
            r = mid;
        }
        else {
            l = mid;
        }
    }
    return l;
}
```

三分

三分法可以用来查找凸函数的最大（小）值。

```
lmid = left + (right - left >> 1);
rmid = lmid + (right - lmid >> 1);  // 对右侧区间取半
if (cal(lmid) > cal(rmid)) right = rmid;
else left = lmid;
```

高精度加法

```
vector<int> add(vector<int> &A, vector<int> &B) {
    if (A.size() < B.size()) return add(B, A);
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i++) {
        t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }
    if (t) C.push_back(t);
    return C;
}
int main() {
    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    for (int i = b.size() - 1; i >= 0; i--) B.push_back(b[i] - '0');
    auto C = add(A, B);
    for (int i = C.size() - 1; i >= 0; i--) cout << C[i];
    return 0;
}
```

高精度减法

```
bool cmp(vector<int> &A, vector<int> &B) {
    if (A.size() != B.size()) return A.size() > B.size();
    for (int i = A.size() - 1; i >= 0; i--)
        if (A[i] != B[i])
            return A[i] > B[i];
    return true;
}
vector<int> sub(vector<int> &A, vector<int> &B) {
    vector<int> C;
    for (int i = 0, t = 0; i < A.size(); i++) {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
int main() {
    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    for (int i = b.size() - 1; i >= 0; i--) B.push_back(b[i] - '0');
    vector<int> C;
    if (cmp(A, B)) C = sub(A, B);
    else C = sub(B, A), cout << '-';
    for (int i = C.size() - 1; i >= 0; i--) cout << C[i];
    return 0;
}
```

高精度乘法

```
vector<int> mul(vector<int> &A, int b) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size() || t; i++) {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
int main() {
    string a;
    int b;
    cin >> a >> b;
    vector<int> A;
    for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    auto C = mul(A, b);
    for (int i = C.size() - 1; i >= 0; i--) printf("%d", C[i]);
    return 0;
```

```
}
```

高精度除法

```
vector<int> div(vector<int> &A, int b, int &r) {
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; i--) {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
int main() {
    string a;
    vector<int> A;
    int B;
    cin >> a >> B;
    for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    int r;
    auto C = div(A, B, r);
    for (int i = C.size() - 1; i >= 0; i--) cout << C[i];
    cout << '\n' << r; //余数
    return 0;
}
```

一维前缀和

```
void init() {
    for (int i = 1; i <= n; i ++ ) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i ++ ) s[i] = s[i - 1] + a[i];
}
int rangeSum(int l, int r) {
    return s[r] - s[l - 1];
}
```

二维前缀和

```
void init() {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            scanf("%d", &s[i][j]);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            s[i][j] += s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1];
}
int rangeSum(int l1, int r1, int l2, int r2) {
    return s[l2][r2] - s[l1 - 1][r2] - s[l2][r1 - 1] + s[l1 - 1][r1 - 1];
}
```

一维差分

```
void insert(int l, int r, int c) {
    s[l] += c;
    s[r + 1] -= c;
}
void init() {
    for (int i = 1; i <= n; i++) s[i] += s[i - 1];
}
int rangeSum(int l, int r) {
    return s[r] - s[l - 1];
}
```

二维差分

```
void insert(int l1, int r1, int l2, int r2, int c) {
    s[l1][r1] += c;
    s[l2 + 1][r1] -= c;
    s[l1][r2 + 1] -= c;
    s[l2 + 1][r2 + 1] += c;
}
void init() {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            s[i][j] += s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1];
}
int rangeSum(int l1, int r1, int l2, int r2) {
    return s[l2][r2] - s[l1 - 1][r2] - s[l2][r1 - 1] + s[l1 - 1][r1 - 1];
}
```

搜索

全排列

```
#include <bits/stdc++.h>
using namespace std;
const int N = 10;
int n;
int path[N];
void dfs(int u, int state) {
    if (u == n) {
        for (int i = 0; i < n; i++) printf("%d ", path[i]);
        puts("");
        return;
    }
    for (int i = 0; i < n; i++)
        if (!(state >> i & 1)) {
            path[u] = i + 1;
            dfs(u + 1, state + (1 << i));
        }
}
int main() {
    scanf("%d", &n);
    dfs(0, 0);
```

```
    return 0;
}
```

n-皇后问题

```
#include <bits/stdc++.h>
using namespace std;
const int N = 20;
int n;
char g[N][N];
bool col[N], dg[N], udg[N];
void dfs(int u) {
    if (u == n) {
        for (int i = 0; i < n; i++) puts(g[i]);
        puts("");
        return;
    }
    for (int i = 0; i < n; i++)
        if (!col[i] && !dg[u + i] && !udg[n - u + i]) {
            g[u][i] = 'Q';
            col[i] = dg[u + i] = udg[n - u + i] = true;
            dfs(u + 1);
            col[i] = dg[u + i] = udg[n - u + i] = false;
            g[u][i] = '.';
        }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            g[i][j] = '.';

    dfs(0);
    return 0;
}
```

图论

邻接表

```
    int idx = 0;
    vector<int> h(n + 1, -1), e(m * 2), ne(m * 2);
    auto add = [&](int a, int b) {
        e[idx] = b, ne[idx] = h[a], h[a] = idx++;
    };
```

拓扑排序

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10, M = 1e5 + 10;
int n, m, e[M], ne[M], h[N], top[N], d[N], idx = 1, cnt;
void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
bool topsort(){
    queue <int> q;
    for(int i = 1; i <= n; i++){
        if(d[i] == 0){
            q.push(i);
            top[++cnt] = i;
        }
    }
    while(q.size()){
        int t = q.front();
        q.pop();
        for(int i = h[t]; i != -1; i = ne[i]){
            int j = e[i];
            d[j]--;
            if(d[j] == 0){
                q.push(j);
                top[++cnt] = j;
            }
        }
    }
    if(cnt == n) return 1;
    else return 0;
}
int main(){
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    for(int i = 1; i <= m; i++){
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
        d[b]++;
    }
    if(!topsort()) puts("-1");
    else{
        for(int i = 1; i <= n; i++){
            printf("%d ", top[i]);
        }
    }
    return 0;
}
```

强连通分量

```
struct SCC{
    int n;
    vector<vector<int>> g;
    vector<int> stk;
```

```
    vector<int> dfn, low, bel;
    int cur, cnt;

    SCC(int n, vector<vector<int>> &g){
        this->n = n;
        this->g = g;
        dfn.assign(n + 1, -1);
        low.resize(n + 1);
        bel.assign(n + 1, -1);
        stk.clear();
        cur = cnt = 0;
    }

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

        for (auto y : g[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = min(low[x], dfn[y]);
            }
        }

        if (dfn[x] == low[x]) {
            int y;
            cnt++;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
            } while (y != x);
        }
    }

    vector<int> work() {
        for (int i = 1; i <= n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }

    vector<vector<int>> get() {
        vector<vector<int>> v(cnt + 1);
        for(int i = 1; i <= n; i++) {
            v[bel[i]].push_back(i);
        }
        return v;
    }
};

    vector<vector<int>> g(n + 1);
    while(m--){
        int u, v;
```

```
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    SCC scc(n, g);
    auto bel = scc.work();
    auto v = scc.get();

    int cnt = scc.cnt;
    vector<> dp(cnt + 1);//int or ll
    for(int i = 1; i <= cnt; i++){
        for(auto x : v[i]){
            for(auto j : g[x]){
                if (bel[j] != i){

                }
            }
        }
    }
```

双连通分量

边双

```
void solve() {
    int n, m;
    cin >> n >> m;
    int idx = 0;
    vector<int> h(n + 1, -1), e(m * 2), ne(m * 2);
    auto add = [&](int a, int b) {
        e[idx] = b, ne[idx] = h[a], h[a] = idx++;
    };
    int timestamp = 0, top = 0, dcc_cnt = 0;
    vector<int> dfn(n + 1), low(n + 1);
    vector<int> stk(n + 1);
    vector<int> id(n + 1), is_bridge(m * 2);
    function<void(int, int)> dcc = [&](int u, int from) {
        dfn[u] = low[u] = ++timestamp;
        stk[++top] = u;
        for (int i = h[u]; ~i; i = ne[i]) {
            int j = e[i];
            if (!dfn[j]) {
                dcc(j, i);
                low[u] = min(low[u], low[j]);
                if (dfn[u] < low[j])
                    is_bridge[i] = is_bridge[i ^ 1] = true;
            }
            else if (i != (from ^ 1))
                low[u] = min(low[u], dfn[j]);
        }
        if (dfn[u] == low[u]) {
            ++dcc_cnt;
            int y;
            do {
```

```
                y = stk[top--];
                id[y] = dcc_cnt;
            } while (y != u);
        }
    };
    while (m--) {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
    dcc(1, -1);//因为边从0开始所以不能是0
    vector<vector<int>> g(dcc_cnt + 1);
    for (int i = 1; i <= n; i++) {
        for (int j = h[i]; j != -1; j = ne[j]) {
            int v = e[j];
            if (is_bridge[j]) {
                g[id[i]].push_back(id[v]);
            }
        }
    }
}
```

最短路算法

朴素dijkstra算法

时间复杂度 O($n^2 + m$), n 表示点数，m 表示边数

```
int n, m, e[N][N], d[N];
bool st[N];
void dijkstra() {
    d[1] = 0;
    for(int i = 1; i <= n; i++) {
        int t = 0;
        for(int j = 1; j <= n; j++) {
            if(!st[j] && (!t || d[j] < d[t])) t = j;
        }
        st[t] = 1;
        for(int j = 2; j <= n; j++) {
            d[j] = min(d[j], d[t] + e[t][j]);
        }
    }
}
int main() {
    cin >> n >> m;
    memset(d, 0x3f3f3f3f, sizeof d);
    memset(e, 0x3f3f3f3f, sizeof e);
    for(int i = 1; i <= n; i++) e[i][i] = 0;
    while(m--) {
        int u, v, w;
        cin >> u >> v >> w;
        e[u][v] = min(e[u][v], w);
    }
    dijkstra();
    if(d[n] == 0x3f3f3f3f) d[n] = -1;
```

```
    printf("%d", d[n]);
    return 0;
}
```

堆优化版dijkstra

时间复杂度O($mlogn$)，n 表示点数，m 表示边数。

```
auto dij = [&](auto &g, int s, int n) {
    vector<i64> d(n + 1, -1);
    vector<int> st(n + 1);
    priority_queue<array<i64, 2>, vector<array<i64, 2>>, greater<array<i64, 2>>>
pq;
    pq.push({0, s});
    while(!pq.empty()) {
        auto [dist, u] = pq.top();
        pq.pop();
        if(st[u]) continue;
        st[u] = 1;
        for(auto [v, w] : g[u]) {
            if(d[v] == -1 || d[v] > dist + w) {
                pq.push({d[v] = dist + w, v});
            }
        }
    }
    return d;
};
```

bellman-ford

有边数限制的最短路(可能存在负权回路)，时间复杂度O(nm)

```
const int N = 510, M = 10010;
int n, m, k, d[N], last[N];
struct Edge{
    int a, b, w;
}edge[M];
void bellman_ford(){
    memset(d, 0x3f, sizeof d);
    d[1] = 0;
    for(int i = 1; i <= k; i++){
        memcpy(last, d, sizeof d);
        for(int j = 1; j <= m; j++){
            auto e = edge[j];
            d[e.b] = min(d[e.b], last[e.a] + e.w);
        }
    }
}
int main(){
    scanf("%d%d%d", &n, &m, &k);
    for(int i = 1; i <= m; i++){
        scanf("%d%d%d", &edge[i].a, &edge[i].b, &edge[i].w);
    }
    bellman_ford();
    if(d[n] > 0x3f3f3f3f / 2) puts("impossible");
    else printf("%d", d[n]);
```

```
    return 0;
}
```

SPFA

```
const int N = 1e5 + 10, M = 1e5 + 10;
int n, m, e[M], ne[M], w[M], h[N], d[N], idx = 1;
bool st[N];
void add(int x, int y, int z){
    e[idx] = y, ne[idx] = h[x], w[idx] = z, h[x] = idx++;
}
int spfa(){
    memset(d, 0x3f, sizeof d);
    d[1] = 0;
    queue <int> q;
    q.push(1);
    st[1] = 1;
    while(q.size()){
        int t = q.front();
        st[t] = 0;
        q.pop();
        for(int i = h[t]; i != -1; i = ne[i]){
            int j = e[i];
            if(d[j] > d[t] + w[i]){
                d[j] = d[t] + w[i];
                if(!st[j]){
                    st[j] = 1;
                    q.push(j);
                }
            }
        }
    }
    return d[n];
}
int main(){
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    for(int i = 1; i <= m; i++){
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        add(x, y, z);
    }
    int t = spfa();
    if(t == 0x3f3f3f3f) puts("impossible");
    else printf("%d", t);
    return 0;
}
```

SPFA求负环

```
const int N = 2010, M = 1e5 + 10;
int n, m, e[M], ne[M], w[M], h[N], cnt[N], d[N], idx = 1;
bool st[N];
void add(int x, int y, int z){
    e[idx] = y, ne[idx] = h[x], w[idx] = z, h[x] = idx++;
}
```

```
bool spfa(){
    queue <int> q;
    for(int i = 1; i <= n; i++){
        q.push(i);
        st[i] = 1;
    }
    while(q.size()){
        int t = q.front();
        q.pop();
        st[t] = 0;
        for(int i = h[t]; i != -1; i = ne[i]){
            int j = e[i];
            if(d[t] + w[i] < d[j]){
                d[j] = d[t] + w[i];
                cnt[j] = cnt[t] + 1;
                if(cnt[j] == n) return 1;
                if(!st[j]){
                    q.push(j);
                    st[j] = 1;
                }
            }
        }
    }
    return 0;
}
int main(){
    memset(h, -1, sizeof h);
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= m; i++){
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        add(x, y, z);
    }
    if(spfa()) puts("Yes");
    else puts("No");
    return 0;
}
```

Floyd求最短路

时间复杂度O(n^3)

```
const int N = 210;
int n, m, k, d[N][N];
void floyd(){
    for(int k = 1; k <= n; k++){
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= n; j++){
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}
int main(){
    scanf("%d%d%d", &n, &m, &k);
    memset(d, 0x3f, sizeof d);
    for(int i = 1; i <= m; i++){
```

```
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        d[x][y] = min(d[x][y], z);
    }
    for(int i = 1; i <= n; i++) d[i][i] = 0;
    floyd();
    while(k--){
        int x, y;
        scanf("%d%d", &x, &y);
        if(d[x][y] > 0x3f3f3f3f / 2) puts("impossible");
        else printf("%d\n", d[x][y]);
    }
    return 0;
}
```

最小生成树

Kruskal

```
int n, m, p[N];
struct edge {
    int u, v, w;
}e[N * 2];
bool cmp(edge a, edge b) {
    return a.w < b.w;
}
int find(int x) {
    if(p[x] != x) p[x] = find(p[x]);
    return p[x];
}
int kruskal() {
    int cnt = 0, ans = 0;
    sort(e + 1, e + 1 + m, cmp);
    for(int i = 1; i <= m; i++) {
        int fa = find(e[i].u), fb = find(e[i].v);
        if(fa != fb) {
            p[fa] = fb;
            cnt++;
            ans += e[i].w;
        }
    }
    if(cnt == n - 1) return ans;
    else return -1;
}
int main() {
    cin >> n >> m;
    for(int i = 1; i <= n; i++) p[i] = i;
    for(int i = 1; i <= m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        e[i] = {u, v, w};
    }
    cout << kruskal();
    return 0;
}
```

Prim

```
#include <bits/stdc++.h>
using namespace std;
const int N = 510, INF = 0x3f3f3f3f;
int n, m, g[N][N], d[N];
bool st[N];
int prim(){
    memset(d, 0x3f, sizeof d);
    int res = 0;
    for(int i = 0; i < n; i++){
        int t = -1;
        for(int j = 1; j <= n; j++){
            if(!st[j] && (t == -1 || d[t] > d[j])) t = j;
        }
        if(i && d[t] == INF) return INF;
        if(i) res += d[t];
        st[t] = 1;
        for(int j = 1; j <= n; j++){
            d[j] = min(d[j], g[t][j]);
        }
    }
    return res;
}
int main(){
    memset(g, 0x3f, sizeof g);
    scanf("%d%d", &n, &m);
    for(int i = 1; i <= m; i++){
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        g[x][y] = g[y][x] = min(g[x][y], z);
    }
    int t = prim();
    if(t == INF) puts("impossible");
    else printf("%d", t);
    return 0;
}
```

dsu on tree

```
    int tim = 0;
    vector<vector<int>> g(n + 1);
    vector<int> heavy(n + 1), l(n + 1), r(n + 1), sz(n + 1, 1), id(n + 1);
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    function <void(int, int)> dfs_init = [&](int u, int fa) {
        l[u] = ++tim;
        id[tim] = u;
        for(auto v : g[u]) {
            if(v == fa)
                continue;
            dfs_init(v, u);
```

```
            sz[u] += sz[v];
            if(!heavy[u] || sz[v] > sz[heavy[u]]) heavy[u] = v;
        }
        r[u] = tim;
    };
    function <void(int, int, bool)> dfs_solve = [&](int u, int fa, bool keep) {
        for(auto v : g[u]) {
            if(v == fa || v == heavy[u])
                continue;
            dfs_solve(v, u, 0);
        }
        if(heavy[u]) dfs_solve(heavy[u], u, 1);
        auto add = [&](int x) {

        };
        auto query = [&](int x) {

        };
        auto del = [&](int x) {

        };
        for(auto v : g[u]) {
            if(v == fa || v == heavy[u])
                continue;
            for(int i = l[v]; i <= r[v]; i++)
                query(id[i]);
            for(int i = l[v]; i <= r[v]; i++)
                add(id[i]);
        }
        query(u);
        add(u);
        if(!keep) {
            for(int i = l[u]; i <= r[u]; i++)
                del(id[i]);
        }
    };
    dfs_init(1, 0);
    dfs_solve(1, 0, 0);
```

二分图

判定二分图

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10, M = 2e5 + 10;
int n, m, idx = 1, e[M], h[N], ne[M], color[N];
void add(int u, int v){
    e[idx] = v, ne[idx] = h[u], h[u] = idx++;
}
bool dfs(int u, int c){
    color[u] = c;
    for(int i = h[u]; i != -1; i = ne[i]){
        int j = e[i];
        if(!color[j]){
            if(!dfs(j, 3 - c)) return 0;
```

```
        }
        else if(color[j] == c) return 0;
    }
    return 1;
}
int main(){
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    for(int i = 1; i <= m; i++){
        int u, v;
        scanf("%d%d", &u, &v);
        add(u, v);
        add(v, u);
    }
    bool flag = 1;
    for(int i = 1; i <= n; i++){
        if(!color[i]){
            if(!dfs(i, 1)){
                flag = 0;
                break;
            }
        }
    }
    if(flag) puts("Yes");
    else puts("No");
    return 0;
}
```

增广路算法

二分图的匹配：给定一个二分图 G，在 G 的一个子图 M 中，M 的边集 $\{E\}$ 中的任意两条边都不依附于同一个顶点，则称 M 是一个匹配。

二分图的最大匹配：所有匹配中包含边数最多的一组匹配被称为二分图的最大匹配，其边数即为最大匹配数。

时间复杂度：$O(nm)$

```
int n1, n2, m;
vector<int> e[N];
int match[N];
bool st[N];
bool find(int u) {
    for(int v : e[u]) {
        if(!st[v]) {
            st[v] = true;
            if(!match[v] || find(match[v])) {
                match[v] = u;
                return true;
            }
        }
    }
    return false;
}

void solve() {
    cin >> n1 >> n2 >> m;
```

```
    for(int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        e[u].push_back(v);
    }
    int ans = 0;
    for(int i = 1; i <= n1; i++) {
        memset(st, false, sizeof st);
        if(find(i)) ans++;
    }
    cout << ans;
}
```

网络流

最大流

dinic

时间复杂度$O(n^2m)$ 二分图时间复杂度$O(\sqrt{n}m)$

可求最大独立集。**独立集**是指在图中选取一个顶点子集，使得该子集中的任意两个顶点之间都不相邻（即没有边直接连接）。二分图的最大独立集为点数-最大匹配数 例题：CF2026E

```
template<class T>
struct Flow {
    const int n;
    struct Edge {
        int to;
        T cap;
        Edge(int to, T cap) : to(to), cap(cap) {}
    };
    vector<Edge> e;
    vector<vector<int>> g;
    vector<int> cur, h;
    Flow(int n) : n(n), g(n) {}

    bool bfs(int s, int t) {
        h.assign(n, -1);
        queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
        return false;
```

```
    }

    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, 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, T c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T maxFlow(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, numeric_limits<T>::max());
        }
        return ans;
    }
};
```

费用流

spfa （不一定能用）

```
const int N = 5010, M = 100010, INF = 1e8;
int n, m, S, T;
int h[N], e[M], f[M], w[M], ne[M], idx;
int q[N], d[N], pre[N], incf[N];
bool st[N];
void add(int a, int b, int c, int d) {
    e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx++;
    e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx++;
}
bool spfa() {
    int hh = 0, tt = 1;
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    q[0] = S, d[S] = 0, incf[S] = INF;
    while (hh != tt) {
```

```
        int t = q[hh++];
        if (hh == N) hh = 0;
        st[t] = false;
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = e[i];
            if (f[i] && d[ver] > d[t] + w[i]) {
                d[ver] = d[t] + w[i];
                pre[ver] = i;
                incf[ver] = min(f[i], incf[t]);
                if (!st[ver]) {
                    q[tt++] = ver;
                    if (tt == N) tt = 0;
                    st[ver] = true;
                }
            }
        }
    }
    return incf[T] > 0;
}
void EK(int& flow, int& cost) {
    flow = cost = 0;
    while (spfa()) {
        int t = incf[T];
        flow += t, cost += t * d[T];
        for (int i = T; i != S; i = e[pre[i] ^ 1]) {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
}
int main() {
    scanf("%d%d%d%d", &n, &m, &S, &T);
    memset(h, -1, sizeof h);
    while (m--) {
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        add(a, b, c, d);
    }
    int flow, cost;
    EK(flow, cost);
    printf("%d %d\n", flow, cost);
    return 0;
}
```

skip2004的板子

$O(m^3)$

abc373G

```
namespace mcmf {
    using pr = std::pair<i64, int>;
    const int N = 10005, M = 1e6 + 10;
    struct edge {
        int to, nxt, v; i64 f;
    } e[M << 1];
    int h[N], num = 1;
    void link(int x, int y, int v, i64 f) {
```

```
        e[++num] = {y, h[x], v, f}, h[x] = num;
        e[++num] = {x, h[y], 0, -f}, h[y] = num;
    }
    i64 d[N], dis[N];
    int vis[N], fr[N];
    bool dijkstra(int s, int t) { // 正常题目不需要 dijk
        std::priority_queue<pr, std::vector<pr>, std::greater<pr>> Q;
        for(int i = s;i <= t;++i) dis[i] = d[i], d[i] = 1e18, vis[i] = fr[i] =
0; // CHECK
        for(Q.emplace(d[s] = 0, s);!Q.empty();) {
            int x = Q.top().second; Q.pop();
            if(vis[x]) continue;
            vis[x] = 1;
            for(int i = h[x];i;i = e[i].nxt) {
                const i64 v = e[i].f + dis[x] - dis[e[i].to];
                if(e[i].v && d[e[i].to] > d[x] + v) {
                    fr[e[i].to] = i;
                    Q.emplace(d[e[i].to] = d[x] + v, e[i].to);
                }
            }
        }
        for(int i = s;i <= t;++i) d[i] += dis[i]; // CHECK
        return d[t] < 1e17;
    }
    std::pair<i64, i64> EK(int s, int t) {
        i64 f = 0, c = 0;
        while(dijkstra(s, t)) { // 正常可以用 spfa
            i64 fl = 1e18;
            for(int i = fr[t]; i; i = fr[e[i ^ 1].to]) fl = std::min<i64>
(e[i].v, fl);
            for(int i = fr[t]; i; i = fr[e[i ^ 1].to]) e[i].v -= fl, e[i ^ 1].v
+= fl;
            f += fl, c += fl * d[t];
        }
        return std::make_pair(f, c);
    }
}

void solve() {

    auto [x, y] = mcmf::EK(S, T); //y是最小费用
    for(int i = 1; i <= n; i++) {
        for(int j = mcmf::h[i]; j; j = mcmf::e[j].nxt) {
            if(mcmf::e[j].to > n && mcmf::e[j].v == 0) {
                cout << mcmf::e[j].to - n << ' ';
                break;
            }
        }
    }
    //求方案
    for(int i = 1; i <= n; i++) {
        using namespace mcmf;
        for(int j = h[i]; j; j = e[j].nxt) {
            if(e[j].to > n && e[j].v == 0) {
                cout << e[j].to - n << ' ';
                break;
            }
        }
```

```
    }
    //直接using namespace mcmf 可以省很多代码 更方便但是要注意变量重名
}
```

LCA

倍增

```
struct LCA {
    int n, LOG;
    vector<vector<int>> fa;
    vector<int> d;
    LCA(int _n) : n(_n), LOG(__lg(n)), fa(n + 1, vector<int> (LOG + 1)), d(n +
1){};
    void init(int s, auto &g) {
        d[s] = 1;
        queue <int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int v : g[u]) {
                if (!d[v]) {
                    q.push(v);
                    d[v] = d[u] + 1;
                    fa[v][0] = u;
                    for (int k = 1; k <= LOG; k++) fa[v][k] = fa[fa[v][k - 1]][k
- 1];
                }
            }
        }
    };
    int anc(int x, int y) {
        if (d[x] > d[y]) swap(x, y);
        for (int k = LOG; k >= 0; k--) if (d[fa[y][k]] >= d[x]) y = fa[y][k];
        if (x == y) return x;
        for (int k = LOG; k >= 0; k--) if (fa[x][k] != fa[y][k]) x = fa[x][k], y
= fa[y][k];
        return fa[x][0];
    };
    int dist(int x, int y) {
        return d[x] + d[y] - 2 * d[anc(x, y)];
    };
};
```

树链剖分

```
    vector<vector<int>> g(n + 1);
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
```

```
    vector<int> heavy(n + 1), fa(n + 1), sz(n + 1, 1), dep(n + 1), top(n + 1);
    function <void(int, int)> dfs_init = [&](int u, int f) {
        dep[u] = dep[f] + 1;
        for(int v : g[u]) {
            if(v == f) continue;
            fa[v] = u;
            dfs_init(v, u);
            sz[u] += sz[v];
            if(!heavy[u] || sz[v] > sz[heavy[u]]) heavy[u] = v;
        }
    };
    dfs_init(1, 0);
    function <void(int, int)> dfs = [&](int u, int t) {
        top[u] = t;
        if(heavy[u]) dfs(heavy[u], t);
        for(int v : g[u]) {
            if(v == fa[u] || v == heavy[u]) continue;
            dfs(v, v);
        }
    };
    dfs(1, 1);
    auto lca = [&](int u, int v) {
        while(top[u] != top[v]) {
            if(dep[top[u]] < dep[top[v]]) v = fa[top[v]];
            else u = fa[top[u]];
        }
        return dep[u] < dep[v] ? u : v;
    };
    auto dist = [&](int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    };
```

树链剖分(jiangly)

```
struct HLD {
    int n;
    vector<int> siz, top, dep, parent, in, out, seq;
    vector<vector<int>> g;
    //vector<vector<int>> c;//存每一条链
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n + 1);
        top.resize(n + 1);
        dep.resize(n + 1);
        parent.resize(n + 1);
        in.resize(n + 1);
        out.resize(n + 1);
        seq.resize(n + 1);
        cur = 0;
        g.assign(n + 1, {});
```

```
        //c.assign(n + 1, {});
    }
    void addEdge(int u, int v) {
        g[u].push_back(v);
        g[v].push_back(u);
    }
    void work(int root = 1) {
        top[root] = root;
        dep[root] = 1;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            g[u].erase(find(g[u].begin(), g[u].end(), parent[u]));
        }
        siz[u] = 1;
        for (auto &v : g[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[g[u][0]]) {
                swap(v, g[u][0]);
            }
        }
    }
    void dfs2(int u) {
        //c[top[u]].push_back(u); 速度会慢很多慎用
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : g[u]) {
            top[v] = v == g[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }
    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }
    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }
        int d = dep[u] - k;
        while (dep[top[u]] > d) {
            u = parent[top[u]];
```

```
        }
        return seq[in[u] - dep[u] + d];
    }
    bool isAncester(int u, int v) {
        return in[u] <= in[v] && in[v] < out[u];
    }
    int rootedParent(int u, int v) {
        swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
        }
        auto it = upper_bound(g[u].begin(), g[u].end(), v, [&](int x, int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }
    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }
    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }
};
```

$O(nlogn) - O(1)$ LCA

```
vector<int> g[N];
int dep[N], dfn[N], a[N];
int anc[N][25];
int lg[N];
int chk(int x, int y) {
    return dep[x] < dep[y] ? x : y;
}
void init_lca(int n) {
    dep[1] = 1;
    int cur = 0;
    const int M = 2 * n;
    function<void(int)> dfs = [&](int u) {
        dfn[u] = ++cur;
        a[cur] = u;
        for(int v : g[u]) {
            dep[v] = dep[u] + 1;
            dfs(v);
            a[++cur] = u;
        }
    };
    dfs(1);
    lg[0] = -1;
```

```
    for(int i = 1; i <= M; i++) lg[i] = lg[i / 2] + 1, anc[i][0] = a[i];
    for(int j = 1; j <= lg[M]; j++) {
        for(int i = 1; i + (1 << j) - 1 <= M; i++) {
            if(dep[anc[i][j - 1]] < dep[anc[i + (1 << j - 1)][j - 1]]) anc[i][j]
= anc[i][j - 1];
            else anc[i][j] = anc[i + (1 << j - 1)][j - 1];
        }
    }
}
int lca(int l, int r) {
    l = dfn[l], r = dfn[r];
    if(l > r) swap(l, r);
    int k = lg[r - l + 1];
    return chk(anc[l][k], anc[r - (1 << k) + 1][k]);
}
int dis(int x, int y){
    int l = lca(x, y);
    return dep[x] + dep[y] - 2 * dep[l];
}
```

虚树

```
void build() {
    sort(a + 1, a + 1 + k, [&](int x, int y) {
        return in[x] < in[y];
    });
    stk[top = 1] = 1;
    for(int i = 1; i <= k; i++) {
        if(a[i] == 1) continue;
        int anc = lca(a[i], stk[top]);
        if(anc != stk[top]) {
            while(in[anc] < in[stk[top - 1]]) g[stk[top - 1]].push_back(stk[top-
-]);
            if(in[anc] > in[stk[top - 1]]) g[anc].push_back(stk[top]), stk[top]
= anc;
            else g[anc].push_back(stk[top--]);
        }
        stk[++top] = a[i];
    }
    for(int i = 1; i < top; i++) g[stk[i]].push_back(stk[i + 1]);
}//最后需要将虚树中的点记录下来清空边

void build() {
    sort(a + 1, a + 1 + k, [&](int x, int y) {
        return in[x] < in[y];
    });
    stk[top = 1] = 1;
    for(int i = 1; i <= k; i++) {
        if(a[i] == 1) continue;
        int anc = lca(a[i], stk[top]);
        if(anc != stk[top]) {
            while(in[anc] < in[stk[top - 1]]) g[stk[top - 1]].push_back(stk[top-
-]);
            if(in[anc] > in[stk[top - 1]]) g[anc].push_back(stk[top]), stk[top]
= anc;
```

```
            else g[anc].push_back(stk[top--]);
        }
        stk[++top] = a[i];
    }
    for(int i = 1; i < top; i++) g[stk[i]].push_back(stk[i + 1]);
}//最后需要将虚树中的点记录下来清空边
//-------//
void build_virtual_tree() {
  sort(h + 1, h + m + 1, cmp);  // 把关键点按照 dfn 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 dfn 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])
  }
}
```

三元环计数

时间复杂度 $m\sqrt{m}$

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> e(n + 1);
    vector<array<int, 2>> g(m + 1);
    vector<int> d(n + 1);
    vector<int> st(n + 1);
    for(int i = 1; i <= m; i++) {
        cin >> g[i][0] >> g[i][1];
        d[g[i][0]]++, d[g[i][1]]++;
    }
    for(int i = 1; i <= m; i++) {
        auto [x, y] = g[i];
        if(d[x] < d[y] || (d[x] == d[y] && x < y)) e[y].push_back(x);
        else e[x].push_back(y);
    }
    int res = 0;
    for(int i = 1; i <= n; i++) {
        for(int j : e[i]) st[j] = i;
        for(int j : e[i])
            for(int k : e[j])
                if(st[j] == st[k])
                    res++;
    }
    cout << res;
}
```

四元环计数

时间复杂度 $m\sqrt{m}$ 个数上界 $nm\sqrt{m}$

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> e(n + 1);
    vector<int> a(n + 1), d(n + 1), rk(n + 1), st(n + 1);
    for(int i = 1; i <= m; i++) {
        int x, y;
        cin >> x >> y;
        d[x]++, d[y]++;
        e[x].push_back(y), e[y].push_back(x);
    }
    for(int i = 1; i <= n; i++) a[i] = i;
    sort(a.begin() + 1, a.end(), [&](int x, int y) {
        return d[x] < d[y] || (d[x] == d[y] && x < y);
    });
    for(int i = 1; i <= n; i++) rk[a[i]] = i;
    long long res = 0;
    for(int i = 1; i <= n; i++) {
        for(int j : e[i])
            if(rk[i] > rk[j])
                for(int k : e[j])
                    if(rk[i] > rk[k])
                        res += st[k]++;
        for(int j : e[i])
            if(rk[i] > rk[j])
                for(int k : e[j])
                    if(rk[i] > rk[k])
                        st[k] = 0;
    }
    cout << res;
}
```

点分治

一棵树，边带权，统计树上距离不超过 k 的点对数

```
void solve() {
    int n;
    cin >> n;
    vector<vector<array<int, 2>>> g(n + 1);
    for(int i = 1; i < n; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        g[u].push_back({v, w});
        g[v].push_back({u, w});
    }
    int k;
    cin >> k;
    i64 ans = 0;
    vector<int> del(n + 1), maxs(n + 1), sz(n + 1);
    function<void(int, int)> treeDivide = [&](int u, int s) {
        int root = -1;
```

```
        function<void(int, int)> center = [&](int u, int fa) {
            sz[u] = 1, maxs[u] = 0;
            for(auto [v, w] : g[u]) {
                if(del[v] || v == fa) continue;
                center(v, u);
                sz[u] += sz[v];
                maxs[u] = max(maxs[u], sz[v]);
            }
            maxs[u] = max(maxs[u], s - sz[u]);
            if(root == -1 || maxs[u] < maxs[root]) root = u;
        };
        center(u, 0);
        vector<int> d1, d2;
        d1.push_back(0);
        auto calc = [&](vector<int> &d) {
            sort(d.begin(), d.end());
            int m = d.size(), r = m - 1;
            i64 ans = 0;
            for(int i = 0; i < m; i++) {
                while(r >= 0 && d[i] + d[r] > k) r--;
                if(i < r) ans += r - i;
            }
            return ans;
        };
        for(auto [v, w] : g[root]) {
            if(del[v]) continue;
            d2.clear();
            function<void(int, int, int)> dfs = [&](int u, int fa, int dep) {
                sz[u] = 1;
                d1.push_back(dep);
                d2.push_back(dep);
                for(auto [v, w] : g[u]) {
                    if(del[v] || v == fa) continue;
                    dfs(v, u, dep + w);
                    sz[u] += sz[v];
                }
            };
            dfs(v, root, w);
            ans -= calc(d2);
        }
        ans += calc(d1);
        del[root] = true;
        for(auto [v, w] : g[root]) {
            if(!del[v]) treeDivide(v, sz[v]);
        }
    };
    treeDivide(1, n);
    cout << ans;
}
```

动态点分治

将每一层分治重心和上一层的连边得到分治树。这就是 **点分树**。点分树是一棵重构树。

给一棵树，点有点权。q 次询问给定 x，k，求与 x 树上距离 $\leq k$ 的所有点的点权和。带修点权，强制在线。n，$q \leq 10^5$ 。

```cpp
template <typename T>
struct Fenwick {
    int n;
    vector<T> a;
    Fenwick(int _n = 0) : a(_n + 1), n(_n){}
    void add(int x, T v) {
        for (int i = x; i <= n; i += i & -i) {
            a[i] += v;
        }
    }
    T sum(int x) {
        x = min(x, n);
        auto ans = T();
        for (int i = x; i >= 1; i -= i & -i) {
            ans += a[i];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};
struct LCA {
    int n, LOG;
    vector<vector<int>> fa;
    vector<int> d;
    LCA(int _n) : n(_n), LOG(__lg(n)), fa(n + 1, vector<int> (__lg(n) + 1)), d(n
+ 1){};
    void init(int s, auto &g) {
        d[s] = 1;
        queue <int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto [v, w] : g[u]) {
                if (!d[v]) {
                    q.push(v);
                    d[v] = d[u] + 1;
                    fa[v][0] = u;
                    for (int k = 1; k <= LOG; k++) fa[v][k] = fa[fa[v][k - 1]][k
- 1];
                }
            }
        }
    };
    int anc(int x, int y) {
        if (d[x] > d[y]) swap(x, y);
        for (int k = LOG; k >= 0; k--) if (d[fa[y][k]] >= d[x]) y = fa[y][k];
        if (x == y) return x;
        for (int k = LOG; k >= 0; k--) if (fa[x][k] != fa[y][k]) x = fa[x][k], y
= fa[y][k];
        return fa[x][0];
    };
    int dist(int x, int y) {
        return d[x] + d[y] - 2 * d[anc(x, y)];
    };
};
```

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> w(n + 1);
    vector<vector<array<int, 2>>> g(n + 1);
    vector<vector<int>> e(n + 1);
    for(int i = 1; i <= n; i++) {
        cin >> w[i];
    }
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back({v, 1});
        g[v].push_back({u, 1});
    }
    LCA lca(n);
    lca.init(1, g);
    vector<Fenwick<int>> fen1(n + 1), fen2(n + 1);
    vector<int> del(n + 1), maxs(n + 1), sz(n + 1), father(n + 1);
    function<void(int, int, int, int)> treeDivide = [&](int u, int s, int rt,
int lastmax) {
        int root = -1;
        function<void(int, int)> center = [&](int u, int fa) {
            sz[u] = 1, maxs[u] = 0;
            for(auto [v, w] : g[u]) {
                if(del[v] || v == fa) continue;
                center(v, u);
                sz[u] += sz[v];
                maxs[u] = max(maxs[u], sz[v]);
            }
            maxs[u] = max(maxs[u], s - sz[u]);
            if(root == -1 || maxs[u] < maxs[root]) root = u;
        };
        center(u, 0);
        if(rt) e[rt].push_back(root), father[root] = rt;
        int mx = 0;
        for(auto [v, w] : g[root]) {
            if(del[v]) continue;
            function<void(int, int, int)> dfs = [&](int u, int fa, int dep) {
                sz[u] = 1;
                mx = max(mx, dep);
                for(auto [v, w] : g[u]) {
                    if(del[v] || v == fa) continue;
                    dfs(v, u, dep + w);
                    sz[u] += sz[v];
                }
            };
            dfs(v, root, w);
        }
        fen1[root] = Fenwick<int> (mx + 1);
        fen2[root] = Fenwick<int> (lastmax + 1);
        del[root] = true;
        for(auto [v, w] : g[root]) {
            if(!del[v]) treeDivide(v, sz[v], root, mx);
        }
    };
    treeDivide(1, n, 0, 0);
    for(int i = 1; i <= n; i++) {
```

```
        int cur = i;
        while(cur) {
            fen1[cur].add(lca.dist(i, cur) + 1, w[i]);
            if(father[cur]) fen2[cur].add(lca.dist(i, father[cur]) + 1, w[i]);
            cur = father[cur];
        }
    }
    int lastans = 0;
    while(m--) {
        int op;
        cin >> op;
        int x, k;
        cin >> x >> k;
        x ^= lastans, k ^= lastans;
        if(!op) {
            int cur = x;
            int ans = 0;
            while(cur) {
                if(lca.dist(cur, x) <= k) ans += fen1[cur].sum(k - lca.dist(cur,
x) + 1);
                if(father[cur] && lca.dist(father[cur], x) <= k) ans -=
fen2[cur].sum(k - lca.dist(father[cur], x) + 1);
                cur = father[cur];
            }
            cout << ans << '\n';
            lastans = ans;
        }
        else {
            int cur = x;
            while(cur) {
                fen1[cur].add(lca.dist(x, cur) + 1, -w[x]);
                fen1[cur].add(lca.dist(x, cur) + 1, k);
                if(father[cur]) fen2[cur].add(lca.dist(x, father[cur]) + 1, -
w[x]), fen2[cur].add(lca.dist(x, father[cur]) + 1, k);
                cur = father[cur];
            }
            w[x] = k;
        }
    }
}
```

计算几何

板子

```
const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
const int maxp = 1010;

//判断正负
inline int sgn (double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    else return 1;
}
```

```
inline double sqr (double x) {return x * x;}

//精度高时用long long!
struct Point {
    double x, y;
    Point () {}
    Point (double _x, double _y) {
        x = _x;
        y = _y;
    }
    void input () {scanf("%lf%lf", &x, &y);}
    void output () {printf("%.2lf%.2lf\n", x, y);}
    bool operator == (Point b) const {return sgn(x - b.x) == 0 && sgn(y - b.y)
== 0;}
    bool operator < (Point b) const {return sgn(x - b.x) == 0 ? sgn(y - b.y) < 0
: x < b.x;}
    Point operator + (const Point &b) const {return Point (x + b.x, y + b.y);}
    Point operator - (const Point &b) const {return Point (x - b.x, y - b.y);}
    Point operator * (const double &k) const {return Point(x * k, y * k);}
    Point operator / (const double &k) const {return Point(x / k, y / k);}

    //叉积
    double operator ^ (const Point &b) const {return x * b.y - y * b.x;}

    //点乘
    double operator * (const Point &b) const {return x * b.x + y * b.y;}

    //距离原点长度
    double len () {return hypot(x, y);}

    //距离原点长度的平方
    double len2 () {return x * x + y * y;}

    //两点距离
    double distance (Point p) {return hypot(x - p.x, y - p.y);}

    //pa和pb所成的夹角
    double rad (Point a, Point b) {
        Point p = *this;
        return fabs(atan2(fabs((a - p) ^ (b - p)), (a - p) * (b - p)));
    }

    //化为长度为r的向量
    Point trunc (double r) {
        double l = len();
        if (!sgn(l)) return *this;
        r /= l;
        return Point (x * r, y * r);
    }

    //逆时针旋转90°
    Point rotleft () {return Point(-y, x);}

    //顺时针旋转90°
    Point rotright () {return Point(y, -x);}

    //绕着p点逆时针旋转angle
    Point rotate (Point p, double angle) {
```

```
        Point v = (*this) - p;
        double c = cos(angle), s = sin(angle);
        return Point (p.x + v.x * c - v.y * s, p.y + v.x * s + v.y * c);
    }
};

//象限排序
inline int quadrant (Point a) {
    if (a.x > 0 && a.y >= 0) return 1;
    if (a.x <= 0 && a.y > 0) return 2;
    if (a.x < 0 && a.y <= 0) return 3;
    return 4;
}

//极角排序
inline bool Pcmp (Point a, Point b) {
    double aa = a ^ b;
    if (quadrant(a) == quadrant(b)) return aa > 0;
    return quadrant(a) < quadrant(b);
}

struct Line {
    Point s, e;
    Line () {}
    Line (Point _s, Point _e) {
        s = _s;
        e = _e;
    }
    bool operator == (Line v) {return (s == v.s) && (e == v.e);}

    //根据一个点和倾斜角angle确定直线， 0 <= angle < pi
    Line (Point p, double angle) {
        s = p;
        if (sgn(angle - pi / 2) == 0) e = (s + Point(0, 1));
        else e = (s + Point(1, tan(angle)));
    }
    Line (double a, double b, double c) {
        if (sgn(a) == 0) {
            s = Point(0, -c / b);
            e = Point(1, -c / b);
        } else if (sgn(b) == 0) {
            s = Point(-c / a, 0);
            e = Point(-c / a, 1);
        } else {
            s = Point(0, -c / b);
            e = Point(1, (-c - a) / b);
        }
    }
    void input () {
        s.input();
        e.input();
    }
    void adjust () {if (e < s) swap(s, e);}

    //求线段长度
    double length () {return s.distance(e);}

    //返回直线倾斜角
```

```
    double angle () {
        double k = atan2(e.y - s.y, e.x - s.x);
        if (sgn(k) < 0) k += pi;
        if (sgn(k - pi) == 0) k -= pi;
        return k;
    }

    //点和直线的关系，1在左侧，2在右侧，3在直线上
    int relation (Point p) {
        int c = sgn((p - s) ^ (e - s));
        if (c < 0) return 1;
        else if (c > 0) return 2;
        else return 3;
    }

    //点在直线上判断
    bool pointonseg (Point p) {return sgn((p - s) ^ (e - s)) == 0 && sgn((p - s)
* (p - e)) <= 0;}

    //两向量平行（直线平行或重合）
    bool parallel (Line v) {return sgn((e - s) ^ (v.e - v.s)) == 0;}

    //两线段相交判断，2规范相交，1非规范相交，0不相交
    int segcrossseg (Line v) {
        int d1 = sgn((e - s) ^ (v.s - s));
        int d2 = sgn((e - s) ^ (v.e - s));
        int d3 = sgn((v.e - v.s) ^ (s - v.s));
        int d4 = sgn((v.e - v.s) ^ (e - v.s));
        if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) return 2;
        return (d1 == 0 && sgn((v.s - s) * (v.s - e)) <= 0) ||
               (d2 == 0 && sgn((v.e - s) * (v.e - e)) <= 0) ||
               (d3 == 0 && sgn((s - v.s) * (s - v.e)) <= 0) ||
               (d4 == 0 && sgn((e - v.s) * (e - v.e)) <= 0);
    }

    //直线（*this）与线段（v）相交判断，2规范相交，1非规范相交，0不相交
    int linecrossseg (Line v) {
        int d1 = sgn((e - s) ^ (v.s - s));
        int d2 = sgn((e - s) ^ (v.e - s));
        if ((d1 ^ d2) == -2) return 2;
        return (d1 == 0 || d2 == 0);
    }

    //两直线关系，0平行，1重合，2相交
    int linecrossline (Line v) {
        if ((*this).parallel(v)) return v.relation(s) == 3;
        return 2;
    }

    //两直线交点
    Point crosspoint (Line v) {
        double a1 = (v.e - v.s) ^ (s - v.s);
        double a2 = (v.e - v.s) ^ (e - v.s);
        return Point((s.x * a2 - e.x * a1) / (a2 - a1), (s.y * a2 - e.y * a1) /
(a2 - a1));
    }

    //点到直线距离
```

```
    double dispointtoline (Point p) {return fabs((p - s) ^ (e - s) / length());}

    //点到线段距离
    double dispointtoseg (Point p) {
        if (sgn((p - s) * (e - s)) < 0 || sgn((p - e) * (s - e)) < 0)
            return min(p.distance(s), p.distance(e));
        return dispointtoline(p);
    }

    //线段到线段的距离
    double dissegtosseg (Line v) {
        return min(min(dispointtoseg(v.s), dispointtoseg(v.e)),
min(v.dispointtoseg(s), v.dispointtoseg(e)));
    }

    //返回点p在直线上的投影
    Point lineprog (Point p) {
        return s + (((e - s) * ((e - s) * (p - s))) / ((e - s).len2()));
    }

    //返回点p关于直线的对称点
    Point symmetrypoint (Point p) {
        Point q = lineprog(p);
        return Point(2 * q.x - p.x, 2 * q.y - p.y);
    }
};
struct circle {
    Point p; //圆心
    double r; //半径
    circle() {}
    circle (Point _p, double _r) {
        p = _p;
        r = _r;
    }
    circle (double x, double y, double _r) {
        p = Point(x, y);
        r = _r;
    }

    //三角形的外接圆
    //需要 Point 的 + / rotate() 以及 Line 的 crosspoint()
    //利用两条边的中垂线得到圆心
    circle (Point a, Point b, Point c) {
        Line u = Line((a + b) / 2, ((a + b) / 2) + ((b - a).rotleft()));
        Line v = Line((b + c) / 2, ((b + c) / 2) + ((c - b).rotleft()));
        p = u.crosspoint(v);
        r = p.distance(a);
    }

    //三角形的内切圆
    //参数 bool t 没有作用，只是为了和外接圆函数区别
    circle (Point a, Point b, Point c, bool t) {
        Line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x - a.x);
        u.s = a;
        u.e = u.s + Point(cos((n + m) / 2), sin((n + m) / 2));
        v.s = b;
        m = atan2 (a.y - b.y, a.x - b.x), n = atan2(c.y - b.y, c.x - b.x);
```

```
        v.e = v.s + Point(cos((n + m) / 2), sin((n + m) / 2));
        p = u.crosspoint(v);
        r = Line(a, b).dispointtoseg(p);
    }

    //输入
    void input () {
        p.input();
        scanf("%lf", &r);
    }

    //输出
    void output () {
        printf("%.2lf %.2lf %.2lf\n", p.x, p.y, r);
    }
    bool operator == (circle v) {
        return (p == v.p) && sgn(r - v.r) == 0;
    }
    bool operator < (circle v) const {
        return ((p < v.p) || ((p == v.p) && sgn(r - v.r) < 0));
    }

    //面积
    double area () {
        return pi * r * r;
    }

    //周长
    double circumference () {
        return 2 * pi * r;
    }

    //点和圆的关系
    //0 圆外
    //1 圆上
    //2 圆内
    int relation (Point b) {
        double dst = b.distance(p);
        if (sgn(dst - r) < 0) return 2;
        else if (sgn(dst - r) == 0) return 1;
        return 0;
    }

    //线段和圆的关系
    //比较的是圆心到线段的距离和半径的关系
    int relationseg (Line v) {
        double dst = v.dispointtoseg(p);
        if (sgn(dst - r) < 0) return 2;
        else if (sgn(dst - r) == 0) return 1;
        return 0;
    }

    //直线和圆的关系
    //比较的是圆心到直线的距离和半径的关系
    int relationline (Line v) {
        double dst = v.dispointtoline(p);
        if (sgn(dst - r) < 0) return 2;
        else if (sgn(dst - r) == 0) return 1;
```

```
        return 0;
    }

    //两圆的关系
    //5 相离
    //4 外切
    //3 相交
    //2 内切
    //1 内含
    //需要 Point 的 distance
    int relationcircle (circle v) {
        double d = p.distance(v.p);
        if (sgn(d - r - v.r) > 0) return 5;
        if (sgn(d - r - v.r) == 0) return 4;
        double l = fabs(r - v.r);
        if (sgn(d - r - v.r) < 0 && sgn(d - l) > 0) return 3;
        if (sgn(d - l) == 0) return 2;
        if (sgn(d - l) < 0) return 1;
    }

    //求两个圆的交点，返回 0 表示没有交点，返回 1 是一个交点，2 是两个交点
    //需要 relationcircle
    int pointcrosscircle (circle v, Point &p1, Point &p2) {
        int rel = relationcircle(v);
        if (rel == 1 || rel == 5) return 0;
        double d = p.distance(v.p);
        double l = (d * d + r * r - v.r * v.r) / (2 * d);
        double h = sqrt(r * r - l * l);
        Point tmp = p + (v.p - p).trunc(l);
        p1 = tmp + ((v.p - p).rotleft().trunc(h));
        p2 = tmp + ((v.p - p).rotright().trunc(h));
        if (rel == 2 || rel == 4) return 1;
        return 2;
    }

    //求直线和圆的交点，返回交点个数
    int pointcrossline (Line v, Point &p1, Point &p2) {
        if (!(*this).relationline(v)) return 0;
        Point a = v.lineprog(p);
        double d = v.dispointtoline(p);
        d = sqrt(r * r - d * d);
        if (sgn(d) == 0) {
            p1 = a;
            p2 = a;
            return 1;
        }
        p1 = a + (v.e - v.s).trunc(d);
        p2 = a - (v.e - v.s).trunc(d);
        return 2;
    }

    //得到过 a,b 两点，半径为 r1 的两个圆
    int getcircle (Point a, Point b, double r1, circle &c1, circle &c2) {
        circle x(a, r1), y(b, r1);
        int t = x.pointcrosscircle(y, c1.p, c2.p);
        if (!t) return 0;
        c1.r = c2.r = r1;
        return t;
```

```
    }

    //得到与直线 u 相切，过点 q，半径为 r1 的圆
    int getcircle (Line u, Point q, double r1, circle &c1, circle &c2) {
        double dis = u.dispointtoline(q);
        if (sgn(dis - r1 * 2) > 0) return 0;
        if (sgn(dis) == 0) {
            c1.p = q + ((u.e - u.s).rotleft().trunc(r1));
            c2.p = q + ((u.e - u.s).rotright().trunc(r1));
            c1.r = c2.r = r1;
            return 2;
        }
        Line u1 = Line((u.s + (u.e - u.s).rotleft().trunc(r1)), (u.e + (u.e -
u.s).rotleft().trunc(r1)));
        Line u2 = Line((u.s + (u.e - u.s).rotright().trunc(r1)), (u.e + (u.e -
u.s).rotright().trunc(r1)));
        circle cc = circle(q, r1);
        Point p1, p2;
        if (!cc.pointcrossline(u1, p1, p2)) cc.pointcrossline(u2, p1, p2);
        c1 = circle(p1, r1);
        if (p1 == p2) {
            c2 = c1;
            return 1;
        }
        c2 = circle(p2, r1);
        return 2;
    }
    //同时与直线 u,v 相切，半径为 r1 的圆
    int getcircle (Line u, Line v, double r1, circle &c1, circle &c2, circle
&c3, circle &c4) {
        if (u.parallel(v)) return 0; //两直线平行
        Line u1 = Line(u.s + (u.e - u.s).rotleft().trunc(r1), u.e + (u.e -
u.s).rotleft().trunc(r1));
        Line u2 = Line(u.s + (u.e - u.s).rotright().trunc(r1), u.e + (u.e -
u.s).rotright().trunc(r1));
        Line v1 = Line(v.s + (v.e - v.s).rotleft().trunc(r1), v.e + (v.e -
v.s).rotleft().trunc(r1));
        Line v2 = Line(v.s + (v.e - v.s).rotright().trunc(r1), v.e + (v.e -
v.s).rotright().trunc(r1));
        c1.r = c2.r = c3.r = c4.r = r1;
        c1.p = u1.crosspoint(v1);
        c2.p = u1.crosspoint(v2);
        c3.p = u2.crosspoint(v1);
        c4.p = u2.crosspoint(v2);
        return 4;
    }

    //同时与不相交圆 cx,cy 相切，半径为 r1 的圆
    int getcircle (circle cx, circle cy, double r1, circle &c1, circle &c2) {
        circle x(cx.p, r1 + cx.r), y(cy.p, r1 + cy.r);
        int t = x.pointcrosscircle(y, c1.p, c2.p);
        if (!t) return 0;
        c1.r = c2.r = r1;
        return t;
    }

    //过一点作圆的切线 (先判断点和圆的关系)
    int tangentline (Point q, Line &u, Line &v) {
```

```
        int x = relation(q);
        if (x == 2) return 0;
        if (x == 1) {
            u = Line(q, q + (q - p).rotleft());
            v = u;
            return 1;
        }
        double d = p.distance(q);
        double l = r * r / d;
        double h = sqrt(r * r - l * l);
        u = Line(q, p + ((q - p).trunc(l) + (q - p).rotleft().trunc(h)));
        v = Line(q, p + ((q - p).trunc(l) + (q - p).rotright().trunc(h)));
        return 2;
    }

    //求两圆相交的面积
    double areacircle (circle v) {
        int rel = relationcircle(v);
        if (rel >= 4) return 0.0;
        if (rel <= 2) return min(area(), v.area());
        double d = p.distance(v.p);
        double hf = (r + v.r + d) / 2.0;
        double ss = 2 * sqrt(hf * (hf - r) * (hf - v.r) * (hf - d));
        double a1 = acos((r * r + d * d - v.r * v.r) / (2.0 * r * d));
        a1 = a1 * r * r;
        double a2 = acos((v.r * v.r + d * d - r * r) / (2.0 * v.r * d));
        a2 = a2 * v.r * v.r;
        return a1 + a2 - ss;
    }

    //求圆和三角形 pab 的相交面积
    double areatriangle (Point a, Point b) {
        if (sgn((p - a) ^ (p - b)) == 0) return 0.0;
        Point q[5];
        int len = 0;
        q[len ++] = a;
        Line l(a, b);
        Point p1, p2;
        if (pointcrossline(l, q[1], q[2]) == 2) {
            if (sgn((a - q[1]) * (b - q[1])) < 0) q[len ++] = q[1];
            if (sgn((a - q[2]) * (b - q[2])) < 0) q[len ++] = q[2];
        }
        q[len ++] = b;
        if (len == 4 && sgn((q[0] - q[1]) * (q[2] - q[1])) > 0) swap(q[1],
q[2]);
        double res = 0;
        for (int i = 0; i < len - 1; i ++) {
            if (relation(q[i]) == 0 || relation(q[i + 1]) == 0) {
                double arg = p.rad(q[i], q[i + 1]);
                res += r * r * arg / 2.0;
            } else
                res += fabs((q[i] - p) ^ (q[i + 1] - p)) / 2.0;
        }
        return res;
    }
};
```

数据结构

链表

单链表

1. `H x`，表示向链表头插入一个数 x。
2. `D k`，表示删除第 k 个插入的数后面的数（当 k 为 00 时，表示删除头结点）。
3. `I k x`，表示在第 k 个插入的数后面插入一个数 x（此操作中 k 均大于 0）。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
// head 表示头结点的下标
// e[i] 表示节点i的值
// ne[i] 表示节点i的next指针是多少
// idx 存储当前已经用到了哪个点
int head, e[N], ne[N], idx;
// 初始化
void init() {
    head = -1;
    idx = 0;
}
// 将x插到头结点
void add_to_head(int x) {
    e[idx] = x, ne[idx] = head, head = idx ++ ;
}
// 将x插到下标是k的点后面
void add(int k, int x) {
    e[idx] = x, ne[idx] = ne[k], ne[k] = idx ++ ;
}
// 将下标是k的点后面的点删掉
void remove(int k) {
    ne[k] = ne[ne[k]];
}
int main() {
    int m;
    cin >> m;
    init();
    while (m--) {
        int k, x;
        char op;
        cin >> op;
        if (op == 'H') {
            cin >> x;
            add_to_head(x);
        } else if (op == 'D') {
            cin >> k;
            if (!k) head = ne[head];
            else remove(k - 1);
        } else {
            cin >> k >> x;
            add(k - 1, x);
        }
    }
```

```
    for (int i = head; i != -1; i = ne[i]) cout << e[i] << ' ';
    return 0;
}
```

双链表

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
int m;
int e[N], l[N], r[N], idx;
// 在节点a的右边插入一个数x
void insert(int a, int x) {
    e[idx] = x;
    l[idx] = a, r[idx] = r[a];
    l[r[a]] = idx, r[a] = idx ++ ;
}
// 删除节点a
void remove(int a) {
    l[r[a]] = l[a];
    r[l[a]] = r[a];
}
int main() {
    cin >> m;
    // 0是左端点，1是右端点
    r[0] = 1, l[1] = 0;
    idx = 2;
    while (m--) {
        string op;
        cin >> op;
        int k, x;
        if (op == "L") {
            cin >> x;
            insert(0, x);
        } else if (op == "R") {
            cin >> x;
            insert(l[1], x);
        } else if (op == "D") {
            cin >> k;
            remove(k + 1);
        } else if (op == "IL") {
            cin >> k >> x;
            insert(l[k + 1], x);
        } else {
            cin >> k >> x;
            insert(k + 1, x);
        }
    }
    for (int i = r[0]; i != 1; i = r[i]) cout << e[i] << ' ';
    return 0;
}
```

堆

模拟堆

维护一个集合，初始时集合为空，支持如下几种操作：

1. `I x`，插入一个数 x；
2. `PM`，输出当前集合中的最小值；
3. `DM`，删除当前集合中的最小值（数据保证此时的最小值唯一）；
4. `D k`，删除第 k 个插入的数；
5. `C k x`，修改第 k 个插入的数，将其变为 x；

现在要进行 N 次操作，对于所有第 2 个操作，输出当前集合的最小值。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
int h[N], ph[N], hp[N], cnt;
void heap_swap(int a, int b) {
    swap(ph[hp[a]], ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(h[a], h[b]);
}
void down(int u) {
    int t = u;
    if (u * 2 <= cnt && h[u * 2] < h[t]) t = u * 2;
    if (u * 2 + 1 <= cnt && h[u * 2 + 1] < h[t]) t = u * 2 + 1;
    if (u != t) {
        heap_swap(u, t);
        down(t);
    }
}
void up(int u) {
    while (u / 2 && h[u] < h[u / 2]) {
        heap_swap(u, u / 2);
        u >>= 1;
    }
}
int main() {
    int n, m = 0;
    scanf("%d", &n);
    while (n--) {
        char op[5];
        int k, x;
        scanf("%s", op);
        if (!strcmp(op, "I")) {
            scanf("%d", &x);
            cnt ++ ;
            m ++ ;
            ph[m] = cnt, hp[cnt] = m;
            h[cnt] = x;
            up(cnt);
        } else if (!strcmp(op, "PM")) printf("%d\n", h[1]);
        else if (!strcmp(op, "DM")) {
            heap_swap(1, cnt);
            cnt -- ;
            down(1);
```

```
        } else if (!strcmp(op, "D")) {
            scanf("%d", &k);
            k = ph[k];
            heap_swap(k, cnt);
            cnt -- ;
            up(k);
            down(k);
        } else {
            scanf("%d%d", &k, &x);
            k = ph[k];
            h[k] = x;
            up(k);
            down(k);
        }
    }
    return 0;
}
```

字典树

字符串字典树

```
struct trie {
    vector<vector<int>> tr;
    vector<int> val;
    int idx = 0;
    vector<int> T;
    trie(int m) : val(1), T(m, 0){
        tr.emplace_back(T);
    };//m是字符集大小
    void insert(string &s) {
        int p = 0;
        for (int i = 0; i < s.size(); i++) {
            int u = s[i] - 'a';
            if (!tr[p][u]) {
                tr[p][u] = ++idx;
                tr.emplace_back(T);
                val.emplace_back(0);
            }
            p = tr[p][u];
        }
        val[p]++;
    }
    int query(string &s) {
        int p = 0, ans = 0;
        for (int i = 0; i < s.size(); i++) {
            int u = s[i] - 'a';
            if (!tr[p][u]) return 0;
            p = tr[p][u];
        }
        return val[p];
    }
};
```

01字典树

在给定的 N 个整数 A_1，A_2 ..... A_N中选出两个进行 xor（异或）运算，得到的结果最大/最小是多少?

```
int n, a[N], tr[N * 30][2], idx;
void insert(int x) {
    int p = 0;
    for (int i = 30; i >= 0; i--) {
        int &s = tr[p][x >> i & 1];
        if (!s) {
            s = ++idx;
        }
        p = s;
    }
}
int query_mx(int x) {
    int p = 0, res = 0;
    for (int i = 30; i >= 0; i--) {
        int s = x >> i & 1;
        if (tr[p][!s]) {
            res += 1 << i;
            p = tr[p][!s];
        } else p = tr[p][s];
    }
    return res;
}
int query_mn(int x) {
    int p = 0, res = 0;
    for (int i = 30; i >= 0; i--) {
        int s = x >> i & 1;
        if (tr[p][s]) {
            p = tr[p][s];
        } else p = tr[p][!s], res += 1 << i;
    }
    return res;
}
void del(int x) {
    for(int i = 0; i <= idx; i++) tr[i][0] = tr[i][1] = 0;
}
```

并查集

```
struct DSU {
    vector<int> f, siz;
    DSU(int n) : f(n + 1), siz(n + 1, 1) { 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)]; }
};
```

可撤销并查集

```
struct DSU {
    int n;
    vector<int> fa, sz;
    stack<array<int, 2>> stk;
    void init(int n) {
        this->n = n;
        sz.assign(n + 1, 1);
        fa.resize(n + 1);
        iota(fa.begin(), fa.end(), 0);
    }
    int find(int x) {
        while(fa[x] != x) x = fa[x];
        return x;
    }
    void merge(int x, int y) {
        int fx = find(x), fy = find(y);
        if(fx != fy) {
            if(sz[fx] > sz[fy]) swap(fx, fy);
            sz[fy] += sz[fx];
            fa[fx] = fy;
            stk.push({fx, fy});
            //tag[fx] -= tag[fy];
        }
    }
    void undo(int last) {
        while((int)stk.size() > last) {
            auto [x, y] = stk.top();
            stk.pop();
            sz[y] -= sz[x];
            fa[x] = x;
            //tag[x] += tag[y];
        }
    }
};
//按秩合并并查集、可撤销并查集
DSU dsu;
```

带权并查集

```
struct DSU {
    struct T {
        int fa;
        i64 len;
    };
    vector<T> t;
    DSU(int n) : t(n + 1) {
        for (int i = 1; i <= n; i++) {
            t[i] = {i, 0};
```

```
        }
    }
    int getfa(int v) {
        if (t[v].fa == v) {
            return v;
        }
        int f = getfa(t[v].fa);
        t[v].len += t[t[v].fa].len;
        t[v].fa = f;
        return f;
    }
    bool merge(int u, int v, int w) {
        int uf = getfa(u), vf = getfa(v);
        if (uf == vf) {
            return t[u].len - t[v].len == w;
        }
        t[uf].len = w + t[v].len - t[u].len;
        t[uf].fa = vf;
        return true;
    }
};
```

可持久化并查集

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

using namespace std;
using i64 = long long;

constexpr int N = 2e5 + 10;
struct DSU {
    int node = 0, n;
    vector<int> ls, rs, f, sz, r;
    DSU(int n, int m) : n(n), ls(n * 4 + 2 * (__lg(m) + 1) * m), rs(n * 4 + 2 *
(__lg(m) + 1) * m), f(n * 4 + 2 * (__lg(m) + 1) * m), sz(n * 4 + 2 * (__lg(m) +
1) * m), r(m + 1){};
    void build(int l, int r, int &x) {
        x = ++node;
        if(l == r) {
            f[x] = l;
            sz[x] = 1;
        }
        else {
            int mid = l + r >> 1;
            build(l, mid, ls[x]);
            build(mid + 1, r, rs[x]);
        }
    }
    void modify(int pre, int &x, int l, int r, int p, int v, bool op) {
        x = ++node, ls[x] = ls[pre], rs[x] = rs[pre], f[x] = f[pre], sz[x] =
sz[pre];
        if(l == r) {
            if(op) f[x] = v;
            else sz[x] = v;
        }
```

```
        else {
            int mid = l + r >> 1;
            if(p <= mid) modify(ls[pre], ls[x], l, mid, p, v, op);
            else modify(rs[pre], rs[x], mid + 1, r, p, v, op);
        }
    }
    int query(int l, int r, int p, int x, bool op) {
        if(l == r) {
            if(op) return f[x];
            else return sz[x];
        }
        else {
            int mid = l + r >> 1;
            if(p <= mid) return query(l, mid, p, ls[x], op);
            else return query(mid + 1, r, p, rs[x], op);
        }
    }
    int find(int x, int id) {
        int fa = query(1, n, x, r[id], 1);
        if(fa == x) return x;
        else return find(fa, id);
    }
    void merge(int x, int y, int id) {
        x = find(x, id), y = find(y, id);
        if(x == y) r[id + 1] = r[id];
        else {
            int sx = query(1, n, x, r[id], 0), sy = query(1, n, y, r[id], 0);
            if(sx < sy) swap(x, y);
            modify(r[id], r[id + 1], 1, n, y, x, 1);
            modify(r[id + 1], r[id + 1], 1, n, x, sx + sy, 0);
        }
    }
    bool check(int x, int y, int id) {
        return find(x, id) == find(y, id);
    }
};

void solve() {
    int n, m;
    cin >> n >> m;
    DSU dsu(n, m);
    dsu.build(1, n, dsu.r[0]);
    int id = 0;
    while(m--) {
        ++id;
        int op;
        cin >> op;
        if(op == 1) {
            int a, b;
            cin >> a >> b;
            dsu.merge(a, b, id - 1);
        }//合并a,b所在集合
        else if(op == 2) {
            int k;
            cin >> k;
            dsu.r[id] = dsu.r[k];
        }//回到第k次操作(执行三种操作中的任意一种都记为一次操作)之后的状态
        else {
```

```
            int a, b;
            cin >> a >> b;
            dsu.r[id] = dsu.r[id - 1];
            cout << dsu.check(a, b, id) << '\n';
        }//询问a,b是否属于同一集合,如果是则输出1,否则输出0
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t;
    t = 1;

    while(t--) {
        solve();
    }

    return 0;
}
```

ST表

```
template <typename T>
struct ST_Table {
    vector<vector<T>> mx, mn, g;
    vector<int> lg;
    vector<T> a;
    int n;
    ST_Table (int n) : a(n + 1), lg(n + 1) {
        this->n = n;
        lg[0] = -1;
        for(int i = 1; i <= n; i++) lg[i] = lg[i / 2] + 1;
    }
    void buildmax() {
        mx.resize(n + 1);
        for(int i = 1; i <= n; i++) mx[i].resize(lg[n] + 1), mx[i][0] = a[i];
        for(int j = 1; j <= lg[n]; j++) {
            for(int i = 1; i + (1 << j) - 1 <= n; i++) {
                mx[i][j] = max(mx[i][j - 1], mx[i + (1 << j - 1)][j - 1]);
            }
        }
    }
    void buildmin() {
        mn.resize(n + 1);
        for(int i = 1; i <= n; i++) mn[i].resize(lg[n] + 1), mn[i][0] = a[i];
        for(int j = 1; j <= lg[n]; j++) {
            for(int i = 1; i + (1 << j) - 1 <= n; i++) {
                mn[i][j] = min(mn[i][j - 1], mn[i + (1 << j - 1)][j - 1]);
            }
        }
    }
    void buildgcd() {
        g.resize(n + 1);
```

```
        for(int i = 1; i <= n; i++) g[i].resize(lg[n] + 1), g[i][0] = a[i];
        for(int j = 1; j <= lg[n]; j++) {
            for(int i = 1; i + (1 << j) - 1 <= n; i++) {
                g[i][j] = __gcd(g[i][j - 1], g[i + (1 << j - 1)][j - 1]);
            }
        }
    }
    T querymax(int l, int r) {
        int p = lg[r - l + 1];
        return max(mx[l][p], mx[r - (1 << p) + 1][p]);
    }
    T querymin(int l, int r) {
        int p = lg[r - l + 1];
        return min(mn[l][p], mn[r - (1 << p) + 1][p]);
    }
    T querygcd(int l, int r) {
        int p = lg[r - l + 1];
        return __gcd(g[l][p], g[r - (1 << p) + 1][p]);
    }
};
```

笛卡尔树

```
struct Cartesian_tree {
    vector<int> a, l, r;
    stack<int> stk;
    int root = 0, n;
    Cartesian_tree (int n) : a(n + 1), l(n + 1), r(n + 1) {this->n = n;};
    void build() {
        for(int i = 1; i <= n; i++) {
            int lst = 0;
            while(!stk.empty() && a[stk.top()] > a[i]) {//求最大值时要改成小于号
                lst = stk.top();
                stk.pop();
            }
            if(!stk.empty()) r[stk.top()] = i;
            else root = i;
            l[i] = lst;
            stk.push(i);
        }
    }
};
```

普通莫队

```
int n, m, a[N], res[M], len, cnt[S], ans;
struct query {
    int l, r, id, b;
} q[M];
int get(int x) {
    return x / len;
}
bool cmp(query x, query y) {
    if (x.b != y.b) return x.b < y.b;
```

```
    else {
        if (x.b & 1) return x.r < y.r;
        else return x.r > y.r;
    }
}
void add(int x) {
    if (!cnt[x]) ans++;
    cnt[x]++;
}
void del(int x) {
    cnt[x]--;
    if (!cnt[x]) ans--;
}
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> a[i];
    len = max(1, (int)sqrt((double)n * n / m));
    for (int i = 1; i <= m; i++) cin >> q[i].l >> q[i].r, q[i].id = i, q[i].b =
get(q[i].l);
    sort(q + 1, q + 1 + m, cmp);
    int l = 1, r = 0;
    for (int i = 1; i <= m; i++) {
        while (r < q[i].r) add(a[++r]);
        while (l > q[i].l) add(a[--l]);
        while (l < q[i].l) del(a[l++]);
        while (r > q[i].r) del(a[r--]);
        res[q[i].id] = ans;
    }
    for (int i = 1; i <= m; i++) cout << res[i] << '\n';
    return 0;
}
```

树状数组

单点修改+区间查询

```
template <typename T>
struct Fenwick {
    int n;
    vector<T> a;
    Fenwick(int _n = 0) : n(_n), a(_n + 1){}
    void init(int n) {
        a.assign(n + 1, T());
        this->n = n;
    }
    void add(int x, T v) {
        for (int i = x; i <= n; i += i & -i) {
            a[i] += v;
        }
    }
    T sum(int x) {
        auto ans = T();
        for (int i = x; i >= 1; i -= i & -i) {
            ans += a[i];
        }
        return ans;
    }
```

```
    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
    int kth(T k) {
        int x = 0;
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i]) {
                x += i;
                k -= a[x];
            }
        }
        return x;
    }
};

struct Max {
    int v;
    Max(int x = -1E9) : v{x} {}//注意范围

    Max &operator+=(Max a) {
        v = max(v, a.v);
        return *this;
    }
};
```

区间修改+区间查询

```
template <typename T>
struct Fenwick {
    int n;
    vector<T> a, b;
    Fenwick(int _n = 0) : a(n + 1, T()), b(n + 1, T()), n(_n){}
    void rangeAdd(int l, int r, T v) {
        for (int i = l; i <= n; i += i & -i) {
            a[i] += v, b[i] += l * v;
        }
        for (int i = r + 1; i <= n; i += i & -i) {
            a[i] -= v, b[i] -= (r + 1) * v;
        }
    }
    T sum(int x) {
        auto ans = T();
        for (int i = x; i >= 1; i -= i & -i) {
            ans += (x + 1) * a[i] - b[i];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};
```

二维树状数组区间修改+区间查询

```
int tr[5][N][N];
int lowbit(int x) {
```

```
    return x & -x;
}
void add(int x, int y, int v) {
    for (int i = x; i <= n; i += lowbit(i)) {
        for (int j = y; j <= m; j += lowbit(j)) {
            tr[0][i][j] += v;
            tr[1][i][j] += v * x;
            tr[2][i][j] += v * y;
            tr[3][i][j] += v * x * y;
        }
    }
}
void range_add(int x1, int y1, int x2, int y2) {
    add(x1, y1, 1);
    add(x2 + 1, y1, -1);
    add(x1, y2 + 1, -1);
    add(x2 + 1, y2 + 1, 1);
}
int ask(int x, int y) {
    int res = 0;
    for (int i = x; i >= 1; i -= lowbit(i)) {
        for (int j = y; j >= 1; j -= lowbit(j)) {
            res += tr[0][i][j] * (x + 1) * (y + 1) - tr[1][i][j] * (y + 1) -
tr[2][i][j] * (x + 1) + tr[3][i][j];
        }
    }
    return res;
}
int range_sum(int x1, int y1, int x2, int y2) {
    return ask(x2, y2) - ask(x1 - 1, y2) - ask(x2, y1 - 1) + ask(x1 - 1, y1 -
1);
}
```

线段树

单点修改区间查询(最大值)

```
struct node {
    int l, r, v;
    #define ls u << 1
    #define rs u << 1 | 1
} tr[N << 2];
void pushup(int u) {
    tr[u].v = max(tr[ls].v, tr[rs].v);
}
void build(int u, int l, int r) {
    tr[u] = {l, r};
    if(l == r) {
        tr[u].v = a[l];
        return ;
    }
    int mid = l + r >> 1;
    build(ls, l, mid), build(rs, mid + 1, r);
    pushup(u);
}
int query(int u, int l, int r) {
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].v;
```

```
    int mid = tr[u].l + tr[u].r >> 1;
    int v = 0;
    if(l <= mid) v = max(v, query(ls, l, r));
    if(r > mid) v = max(v, query(rs, l, r));
    return v;
}
void modify(int u, int x, int v) {
    if(tr[u].l == x && tr[u].r == x) tr[u].v = v;
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if(mid >= x) modify(ls, x, v);
        else modify(rs, x, v);
        pushup(u);
    }
}
```

```
struct Info {
    int mx;
};
Info operator+(const Info &a, const Info &b) {
    return {max(a.mx, b.mx)};
}
template <class T>
struct SegmentTree {
    int n;
    vector<T> info;
    vector<T> _init;
    SegmentTree() {}
    SegmentTree(int n) {
        _init.assign(n + 1, T());
        init(_init);
    }
    SegmentTree(vector<T> &_init) {
        this->_init = _init;
        init(_init);
    }
    void init(vector<T> &_init) {
        n = (int)_init.size() - 1;
        info.assign(n * 4 + 1, T());
        build(1, 1, n);
    }
    void build(int p, int l, int r) {
        if(l == r) {
            info[p] = _init[l];
            return ;
        }
        int m = (l + r) / 2;
        build(p * 2, l, m);
        build(p * 2 + 1, m + 1, r);
        pull(p);
    };
    void pull(int p) {
        info[p] = info[p * 2] + info[p * 2 + 1];
    }
    void modify(int p, int l, int r, int x, T v) {
        if(l == r) info[p] = v;
        else {
```

```
            int m = (l + r) / 2;
            if(x <= m) modify(p * 2, l, m, x, v);
            else modify(p * 2 + 1, m + 1, r, x, v);
            pull(p);
        }
    }
    void modify(int x, T v) {
        modify(1, 1, n, x, v);
    }
    T query(int p, int l, int r, int x, int y) {
        if(l > y || r < x) return T();
        if(x <= l && y >= r) return info[p];
        int m = (l + r) / 2;
        return query(p * 2, l, m, x, y) + query(p * 2 + 1, m + 1, r, x, y);
    }
    T query(int l, int r) {
        return query(1, 1, n, l, r);
    }
};
```

ddp Info为矩阵时

```
struct Info {
    int a[2][2];
    void init() {
        for(int i = 0; i < 2; i++) {
            for(int j = 0; j < 2; j++) {
                //a[i][j] = inf - 1;
            }
        }
    }
    //array<array<int, 2>, 2> a {inf - 1, inf - 1, inf - 1, inf - 1};
};//数组最快 arrar次之 vector巨慢很容易超时
Info operator+(const Info &a, const Info &b) {
    Info c;
    c.init();
    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 2; j++) {
            for(int k = 0; k < 2; k++) {
                //c.a[i][k] = max(c.a[i][k], min(a.a[i][j], b.a[j][k]));
            }
        }
    }
    return c;
}//广义矩阵乘法
```

扫描线

```
struct node {
    int l, r;
    i64 sum, v;
    #define lc u << 1
    #define rc u << 1 | 1
}tr[N << 2];
void pushup(int u) {
    if(tr[u].sum) tr[u].v = tr[u].r - tr[u].l + 1;
```

```
    else if(tr[u].l != tr[u].r) tr[u].v = tr[lc].v + tr[rc].v;
    else tr[u].v = 0;
}
void build(int u, int l, int r) {
    if(l == r) tr[u] = {l, r, 0};
    else {
        tr[u] = {l, r};
        int mid = l + r >> 1;
        build(lc, l, mid), build(rc, mid + 1, r);
        pushup(u);
    }
}
void modify(int u, int l, int r, i64 d) {
    if(tr[u].l >= l && tr[u].r <= r) {
        tr[u].sum += d;
        pushup(u);
    }
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if(l <= mid) modify(lc, l, r, d);
        if(r > mid) modify(rc, l, r, d);
        pushup(u);
    }
}
int query(int u, int l, int r) {
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].v;
    int mid = tr[u].l + tr[u].r >> 1;
    i64 sum = 0;
    if(l <= mid) sum = sum + query(lc, l, r);
    if(r > mid) sum = sum + query(rc, l, r);
    return sum;
}
```

单点修改区间查询合并结果区间(最大连续子段和)

```
struct Node {
    int l, r;
    int sum, lmax, rmax, tmax;
    #define lc u << 1
    #define rc u << 1 | 1
}tr[N * 4];
void pushup(Node &u, Node &l, Node &r) {
    u.sum = l.sum + r.sum;
    u.lmax = max(l.lmax, l.sum + r.lmax);
    u.rmax = max(r.rmax, r.sum + l.rmax);
    u.tmax = max(max(l.tmax, r.tmax), l.rmax + r.lmax);
}
void pushup(int u) {
    pushup(tr[u], tr[lc], tr[rc]);
}
void build(int u, int l, int r) {
    if(l == r) tr[u] = {l, r, a[l], a[l], a[l], a[l]};
    else {
        tr[u].l = l, tr[u].r = r;
        int mid = (l + r) >> 1;
        build(lc, l, mid), build(rc, mid + 1, r);
        pushup(u);
```

```
    }
}
void modify(int u, int x, int v) {
    if(tr[u].l == x && tr[u].r == x) {
        tr[u] = {x, x, v, v, v, v};
    }
    else {
        int mid = (tr[u].l + tr[u].r) >> 1;
        if(x <= mid) modify(lc, x, v);
        else modify(rc, x, v);
        pushup(u);
    }
}
Node query(int u, int l, int r) {
    if(tr[u].l >= l && tr[u].r <= r) return tr[u];
    else {
        int mid = (tr[u].l + tr[u].r) >> 1;
        if(r <= mid) return query(lc, l, r);
        else if(l > mid) return query(rc, l, r);
        else {
            auto left = query(lc, l, r);
            auto right = query(rc, l, r);
            Node res;
            pushup(res, left, right);
            return res;
        }
    }
}
```

懒标记线段树(区间乘加+区间求和) 洛谷模板

```
struct Tag {
    i64 add = 0, mul = 1;
    void apply(Tag &t) {
        add = (add * t.mul + t.add) % m;
        mul = mul * t.mul % m;
    }
};
struct Info {
    i64 sum = 0, l, r;
    void apply(Tag &t) {
        sum = (sum * t.mul + t.add * (r - l + 1) % m) % m;
    }
};
Info operator+(const Info &a, const Info &b) {
    return {(a.sum + b.sum) % m, a.l, b.r};
}
template <class Info, class Tag>
struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    vector<Info> _init;
    LazySegmentTree() {}
    LazySegmentTree(int n) {
        _init.assign(n + 1, Info());
        init(_init);
```

```
    }
    LazySegmentTree(vector<Info> &_init) {
        this->_init = _init;
        init(_init);
    }
    void init(vector<Info> &_init) {
        n = (int)_init.size() - 1;
        info.assign(n * 4 + 1, Info());
        tag.assign(n * 4 + 1, Tag());
        build(1, 1, n);
    }
    void build(int p, int l, int r) {
        if(l == r) {
            info[p] = _init[l];
            return ;
        }
        int m = (l + r) / 2;
        build(p * 2, l, m);
        build(p * 2 + 1, m + 1, r);
        pull(p);
    };
    void apply(int p, Tag &t) {
        info[p].apply(t);
        tag[p].apply(t);
    }
    void push(int p) {
        apply(p * 2, tag[p]);
        apply(p * 2 + 1, tag[p]);
        tag[p] = Tag();
    }
    void pull(int p) {
        info[p] = info[p * 2] + info[p * 2 + 1];
    }
    void modify(int p, int l, int r, int x, int y, Tag &v) {
        if(l > y || r < x) return ;
        if(l >= x && r <= y) {
            apply(p, v);
        }
        else {
            int m = (l + r) / 2;
            push(p);
            modify(p * 2, l, m, x, y, v);
            modify(p * 2 + 1, m + 1, r, x, y, v);
            pull(p);
        }
    }
    void modify(int x, int y, Tag &v) {
        modify(1, 1, n, x, y, v);
    }
    Info query(int p, int l, int r, int x, int y) {
        if(l > y || r < x) return Info();
        if(x <= l && y >= r) {
            return info[p];
        }
        push(p);
        int m = (l + r) / 2;
        return query(p * 2, l, m, x, y) + query(p * 2 + 1, m + 1, r, x, y);
    }
```

```
    Info query(int l, int r) {
        return query(1, 1, n, l, r);
    }
};
```

区间加+区间查询max abc397F

```
struct Tag {
    i64 add = 0;
    void apply(Tag &t) {
        add += t.add;
    }
};
struct Info {
    i64 mx = 0, l, r;
    void apply(Tag &t) {
        mx = mx + t.add;
    }
};
Info operator+(const Info &a, const Info &b) {
    return {max(a.mx, b.mx), a.l, b.r};
}
template <class Info, class Tag>
struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    vector<Info> _init;
    LazySegmentTree() {}
    LazySegmentTree(int n) {
        _init.assign(n + 1, Info());
        init(_init);
    }
    LazySegmentTree(vector<Info> &_init) {
        this->_init = _init;
        init(_init);
    }
    void init(vector<Info> &_init) {
        n = (int)_init.size() - 1;
        info.assign(n * 4 + 1, Info());
        tag.assign(n * 4 + 1, Tag());
        build(1, 1, n);
    }
    void build(int p, int l, int r) {
        if(l == r) {
            info[p] = _init[l];
            return ;
        }
        int m = (l + r) / 2;
        build(p * 2, l, m);
        build(p * 2 + 1, m + 1, r);
        pull(p);
    };
    void apply(int p, Tag &t) {
        info[p].apply(t);
        tag[p].apply(t);
    }
```

```
    void push(int p) {
        apply(p * 2, tag[p]);
        apply(p * 2 + 1, tag[p]);
        tag[p] = Tag();
    }
    void pull(int p) {
        info[p] = info[p * 2] + info[p * 2 + 1];
    }
    void modify(int p, int l, int r, int x, int y, Tag &v) {
        if(l > y || r < x) return ;
        if(l >= x && r <= y) {
            apply(p, v);
        }
        else {
            int m = (l + r) / 2;
            push(p);
            modify(p * 2, l, m, x, y, v);
            modify(p * 2 + 1, m + 1, r, x, y, v);
            pull(p);
        }
    }
    void modify(int x, int y, Tag &v) {
        modify(1, 1, n, x, y, v);
    }
    Info query(int p, int l, int r, int x, int y) {
        if(l > y || r < x) return Info();
        if(x <= l && y >= r) {
            return info[p];
        }
        push(p);
        int m = (l + r) / 2;
        return query(p * 2, l, m, x, y) + query(p * 2 + 1, m + 1, r, x, y);
    }
    Info query(int l, int r) {
        return query(1, 1, n, l, r);
    }
};
```

懒标记线段树(jiangly)

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(vector(n_, v_));
    }
    template<class T>
```

```
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        tag.assign(4 << __lg(n), Tag());
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
```

```
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        push(p);
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
```

```
    }
};

struct Tag {
    int add = 0;
    void apply(Tag t) & {
        add += t.add;
    }
};

struct Info {
    int max = 0;
    void apply(Tag t) & {
        max += t.add;
    }
};

Info operator+(const Info &a, const Info &b) {
    return {max(a.max, b.max)};
}
```

区间修改区间查询

```
struct node {
    int l, r;
    i64 sum, add, mul;
    #define lc u << 1
    #define rc u << 1 | 1
}tr[N << 2];
void pushup(int u) {
    tr[u].sum = tr[lc].sum + tr[rc].sum;
}
void eval(node &t, i64 add, i64 mul) {
    t.sum = t.sum * mul + (t.r - t.l + 1) * add;
    t.mul = t.mul * mul;
    t.add = t.add * mul + add;
}
void pushdown(int u) {
    eval(tr[lc], tr[u].add, tr[u].mul);
    eval(tr[rc], tr[u].add, tr[u].mul);
    tr[u].add = 0, tr[u].mul = 1;
}
void build(int u, int l, int r) {
    if(l == r) tr[u] = {l, r, a[r], 0, 1};
    else {
        tr[u] = {l, r, 0, 0, 1};
        int mid = l + r >> 1;
        build(lc, l, mid), build(rc, mid + 1, r);
        pushup(u);
    }
}
void modify(int u, int l, int r, i64 add, i64 mul) {
    if(tr[u].l >= l && tr[u].r <= r) eval(tr[u], add, mul);
    else {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if(l <= mid) modify(lc, l, r, add, mul);
```

```
        if(r > mid) modify(rc, l, r, add, mul);
        pushup(u);
    }
}
int query(int u, int l, int r) {
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    i64 sum = 0;
    if(l <= mid) sum = sum + query(lc, l, r);
    if(r > mid) sum = sum + query(rc, l, r);
    return sum;
}
```

将一段区间全部改为一个值

```
struct node {
    int l, r, add, v;
#define lc u << 1
#define rc u << 1 | 1
} tr[N << 2];
void pushup(int u) {
    tr[u].v = (tr[lc].v + tr[rc].v);
}
void build(int u, int l, int r) {
    tr[u] = {l, r, -INT_MAX, 0};
    if(l == r) {
        return ;
    }
    int mid = l + r >> 1;
    build(lc, l, mid);
    build(rc, mid + 1, r);
    pushup(u);
}
void pushdown(int u) {
    if(tr[u].add == -INT_MAX) return ;
    tr[lc].v = tr[u].add * (tr[lc].r - tr[lc].l + 1), tr[rc].v = tr[u].add *
(tr[rc].r - tr[rc].l + 1);
    tr[lc].add = tr[u].add, tr[rc].add = tr[u].add;
    tr[u].add = -INT_MAX;
}
void modify(int u, int l, int r, int d) {
    if (tr[u].l >= l && tr[u].r <= r) {
        tr[u].add = d;
        tr[u].v = d * (tr[u].r - tr[u].l + 1);
    } else {
        pushdown(u);
        int mid = tr[u].l + tr[u].r >> 1;
        if (l <= mid) modify(u << 1, l, r, d);
        if (r > mid) modify(u << 1 | 1, l, r, d);
        pushup(u);
    }
}
int query(int u, int l, int r) {
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].v;
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
```

```
    int res = 0;
    if(l <= mid) res += query(lc, l, r);
    if(r > mid) res += query(rc, l, r);
    pushup(u);
    return res;
}
```

势能线段树

```
struct node {
    int l, r;
    i64 sum;
    bool op;
    #define ls u << 1
    #define rs u << 1 | 1
}tr[N << 2];
void pushup(int u) {
    tr[u].sum = tr[ls].sum + tr[rs].sum;
    tr[u].op = (tr[ls].op & tr[rs].op);
}
void build(int u, int l, int r) {
    if(l == r) tr[u] = {l, r, a[l]};
    else {
        tr[u] = {l, r, 0};
        int mid = l + r >> 1;
        build(ls, l, mid), build(rs, mid + 1, r);
        pushup(u);
    }
}
void modify(int u, int l, int r) {
    if(tr[u].l >= l && tr[u].r <= r && tr[u].l == tr[u].r && !tr[u].op) {

    }
    else if(tr[u].op) {
        return ;
    }
    else {
        int mid = tr[u].l + tr[u].r >> 1;
        if(l <= mid) modify(ls, l, r);
        if(r > mid) modify(rs, l, r);
        pushup(u);
    }
}
i64 query(int u, int l, int r) {
    if(tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
    int mid = tr[u].l + tr[u].r >> 1;
    i64 sum = 0;
    if(l <= mid) sum = sum + query(ls, l, r);
    if(r > mid) sum = sum + query(rs, l, r);
    return sum;
}
```

线段树合并

```
int cur;
int root[N], ls[N], rs[N], sum[N], mxz[N];
void pushup(int u) {
    if(sum[ls[u]] >= sum[rs[u]])
        sum[u] = sum[ls[u]], mxz[u] = mxz[ls[u]];
    else
        sum[u] = sum[rs[u]], mxz[u] = mxz[rs[u]];
}
void update(int &u, int l, int r, int z, int val) {
    if(!u) u = ++cur;
    if(l == r) {
        sum[u] += val;
        mxz[u] = z;
        return ;
    }
    int mid = l + r >> 1;
    if(z <= mid) update(ls[u], l, mid, z, val);
    else update(rs[u], mid + 1, r, z, val);
    pushup(u);
}
int merge(int u, int v, int l, int r) {
    if(!u || !v) return u + v;
    if(l == r) {
        sum[u] += sum[v];
        return u;
    }
    int mid = l + r >> 1;
    ls[u] = merge(ls[u], ls[v], l, mid);
    rs[u] = merge(rs[u], rs[v], mid + 1, r);
    pushup(u);
    return u;
}
```

线段树分治

CF1814F 需要维护每个点的tag

```
int tag[N];
vector<array<int, 2>> sgt[N << 2];
struct DSU {
    int n;
    vector<int> fa, sz;
    stack<array<int, 2>> stk;
    void init(int n) {
        this->n = n;
        sz.assign(n + 1, 1);
        fa.resize(n + 1);
        iota(fa.begin(), fa.end(), 0);
    }
    int find(int x) {
        while(fa[x] != x) x = fa[x];
        return x;
    }
    void merge(int x, int y) {
```

```
        int fx = find(x), fy = find(y);
        if(fx != fy) {
            if(sz[fx] > sz[fy]) swap(fx, fy);
            sz[fy] += sz[fx];
            fa[fx] = fy;
            stk.push({fx, fy});
            //tag[fx] -= tag[fy];
        }
    }
    void undo(int last) {
        while((int)stk.size() > last) {
            auto [x, y] = stk.top();
            stk.pop();
            sz[y] -= sz[x];
            fa[x] = x;
            //tag[x] += tag[y];
        }
    }
};
//按秩合并并查集、可撤销并查集
DSU dsu;
void insert(int u, int l, int r, int ql, int qr, int x, int y) {
    if(ql <= l && qr >= r) {
        sgt[u].push_back({x, y});
        return ;
    }
    int m = l + r >> 1;
    if(ql <= m) insert(u * 2, l, m, ql, qr, x, y);
    if(qr > m) insert(u * 2 + 1, m + 1, r, ql, qr, x, y);
}

void dfs_sgt(int u, int l, int r) {
    int last = dsu.stk.size();
    for(auto [x, y] : sgt[u]) {
        dsu.merge(x, y);
    }
    if(l == r) {
        //tag[dsu.find(1)] += 1;
        return ;
    }
    else {
        int m = l + r >> 1;
        dfs_sgt(u * 2, l, m);
        dfs_sgt(u * 2 + 1, m + 1, r);
    }
    dsu.undo(last);
}
```

可持久化线段树

```
int cur, root[N], ls[N << 5], rs[N << 5];
struct Info {
    int v;
} sgt[N << 5];
Info operator+ (const Info &a, const Info &b) {
```

```
    Info c;
    c.v = a.v + b.v;
    return c;
}
void pushup(int u) {
    sgt[u] = sgt[ls[u]] + sgt[rs[u]];
}
void build(int &u, int l, int r) {
    u = ++cur;
    if (l == r) {
        sgt[u] = {0};
        return ;
    }
    int m = l + r >> 1;
    build(ls[u], l, m);
    build(rs[u], m + 1, r);
    pushup(u);
}
void modify(int pr, int &u, int l, int r, int p) {
    u = ++cur, ls[u] = ls[pr], rs[u] = rs[pr];
    if (l == r) {
        sgt[u] = {1};
        return ;
    }
    int m = l + r >> 1;
    if (p <= m) modify(ls[pr], ls[u], l, m, p);
    else modify(rs[pr], rs[u], m + 1, r, p);
    pushup(u);
}
Info query(int u, int l, int r, int ql, int qr) {
    if(qr < l || ql > r) return Info();
    if(ql <= l && qr >= r) {
        return sgt[u];
    }
    int m = l + r >> 1;
    Info ans = Info();
    if(ql <= m) {
        ans = query(ls[u], l, m, ql, qr);
    }
    if(qr > m) {
        ans = ans + query(rs[u], m + 1, r, ql, qr);
    }
    return ans;
}
int find_first(int u, int l, int r, int ql, int qr, int k) {
    if(l == r) {
        if(k == 1) return l;
        else return -1;
    }
    int m = l + r >> 1;
    if(sgt[ls[u]].v >= k) return find_first(ls[u], l, m, ql, qr, k);
    else return find_first(rs[u], m + 1, r, ql, qr, k - sgt[ls[u]].v);
}
//动态开点线段树上二分,全局从左到右第k个,ql等于1

void solve() {
    build(root[0], 1, n); //注意范围，右边界可能不是n
    for (int i = 1; i <= n; i++) {
```

```
        modify(root[i - 1], root[i], 1, n, p);
    }
    //同样需要注意是不是1到n
}
```

指针版

主席树维护从root到各个节点的哈希值，然后在主席树上二分求出前k小

CF1957F2

```
struct Node {
    Node *l = NULL;
    Node *r = NULL;
    i64 hash = 0;
} tr[N << 5];
int tot = 0;
Node *null = &tr[0];
Node *newNode() {
    return &tr[++tot];
}
const int B = 114514;
const i64 P = i64(6e17) + 7;
i64 pw[N];
void add(Node *&t, int l, int r, int x) {
    t = new Node {*t};
    t->hash = (t->hash + pw[x]) % P;
    if (l == r) {
        return;
    }
    int m = (l + r) / 2;
    if (x <= m) {
        add(t->l, l, m, x);
    } else {
        add(t->r, m + 1, r, x);
    }
}
void find(int l, int r, Node *u1, Node *v1, Node *l1, Node *u2, Node *v2, Node
*l2) {
    if(k == 0) return ;
    i64 hash1 = (((u1->hash + v1->hash - 2 * l1->hash) % P + P) % P + (l <= al1
&& al1 <= r ? pw[al1] : 0)) % P;
    i64 hash2 = (((u2->hash + v2->hash - 2 * l2->hash) % P + P) % P + (l <= al2
&& al2 <= r ? pw[al2] : 0)) % P;
    if(hash1 == hash2) return ;
    if(l == r) {
        ans.push_back(l);
        k--;
        return ;
    }
    int m = (l + r) / 2;
    find(l, m, u1->l, v1->l, l1->l, u2->l, v2->l, l2->l);
    find(m + 1, r, u1->r, v1->r, l1->r, u2->r, v2->r, l2->r);
}
void solve() {
    null->l = null->r = null;
    vector<Node *> node(n + 1, null);
    function<void(int, int)> dfs = [&](int u, int fa) {
```

```
        add(node[u], 1, N - 1, a[u]);
        for(int v : g[u]) {
            if(v == fa) continue;
            node[v] = node[u];
            dfs(v, u);
        }
    };
    while(q--) {
        cin >> u1 >> v1 >> u2 >> v2 >> k;
        l1 = hld.lca(u1, v1), l2 = hld.lca(u2, v2);
        al1 = a[l1], al2 = a[l2];
        find(1, N - 1, node[u1], node[v1], node[l1], node[u2], node[v2],
node[l2]);
        cout << ans.size() << " ";
        for(auto x : ans) cout << x << ' ';
        cout << '\n';
        ans.clear();
    }
}
```

treap

```
const int N = 100010, INF = 1e9;
int n;
struct Node {
    int l, r;
    int key, val;
    int cnt, size;
} tr[N];
int root, idx;
void pushup(int p) {
    tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].cnt;
}
int get_node(int key) {
    tr[ ++ idx].key = key;
    tr[idx].val = rand();
    tr[idx].cnt = tr[idx].size = 1;
    return idx;
}
void zig(int &p) {  // 右旋
    int q = tr[p].l;
    tr[p].l = tr[q].r, tr[q].r = p, p = q;
    pushup(tr[p].r), pushup(p);
}
void zag(int &p) {  // 左旋
    int q = tr[p].r;
    tr[p].r = tr[q].l, tr[q].l = p, p = q;
    pushup(tr[p].l), pushup(p);
}
void build() {
    get_node(-INF), get_node(INF);
    root = 1, tr[1].r = 2;
    pushup(root);
    if (tr[1].val < tr[2].val) zag(root);
}
```

```
void insert(int &p, int key) {
    if (!p) p = get_node(key);
    else if (tr[p].key == key) tr[p].cnt++;
    else if (tr[p].key > key) {
        insert(tr[p].l, key);
        if (tr[tr[p].l].val > tr[p].val) zig(p);
    } else {
        insert(tr[p].r, key);
        if (tr[tr[p].r].val > tr[p].val) zag(p);
    }
    pushup(p);
}
void remove(int &p, int key) {
    if (!p) return;
    if (tr[p].key == key) {
        if (tr[p].cnt > 1) tr[p].cnt--;
        else if (tr[p].l || tr[p].r) {
            if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val) {
                zig(p);
                remove(tr[p].r, key);
            } else {
                zag(p);
                remove(tr[p].l, key);
            }
        } else p = 0;
    } else if (tr[p].key > key) remove(tr[p].l, key);
    else remove(tr[p].r, key);
    pushup(p);
}
int get_rank_by_key(int p, int key) {  // 通过数值找排名
    if (!p) return 0;   // 本题中不会发生此情况
    if (tr[p].key == key) return tr[tr[p].l].size + 1;
    if (tr[p].key > key) return get_rank_by_key(tr[p].l, key);
    return tr[tr[p].l].size + tr[p].cnt + get_rank_by_key(tr[p].r, key);
}
int get_key_by_rank(int p, int rank) { // 通过排名找数值
    if (!p) return INF;     // 本题中不会发生此情况
    if (tr[tr[p].l].size >= rank) return get_key_by_rank(tr[p].l, rank);
    if (tr[tr[p].l].size + tr[p].cnt >= rank) return tr[p].key;
    return get_key_by_rank(tr[p].r, rank - tr[tr[p].l].size - tr[p].cnt);
}
int get_prev(int p, int key) { // 找到严格小于key的最大数
    if (!p) return -INF;
    if (tr[p].key >= key) return get_prev(tr[p].l, key);
    return max(tr[p].key, get_prev(tr[p].r, key));
}
int get_next(int p, int key) {  // 找到严格大于key的最小数
    if (!p) return INF;
    if (tr[p].key <= key) return get_next(tr[p].r, key);
    return min(tr[p].key, get_next(tr[p].l, key));
}
int main() {
    build();
    cin >> n;
    while (n--) {
        int op, x;
        cin >> op >> x;
        if (op == 1) insert(root, x);
```

```
        else if (op == 2) remove(root, x);
        else if (op == 3) cout << get_rank_by_key(root, x) - 1 << '\n';
        else if (op == 4) cout << get_key_by_rank(root, x + 1) << '\n';
        else if (op == 5) cout << get_prev(root, x) << '\n';
        else cout << get_next(root, x) << '\n';
    }
    return 0;
}
```

CDQ分治

三位偏序，陌上花开

有 n 朵花，每朵花有三个属性：花形(s)，颜色(c)，气味(m)，用三个整数表示。

现在要对每朵花评级，一朵花的级别是它拥有的美丽能超过的花的数量。

定义一朵花 A 比另一朵花 B 要美丽,当且仅 $s_a \geq s_b, c_a \geq c_b, m_a \geq m_b$。

显然，两朵花可能有同样的属性。需要统计出评出每个等级的花的数量。

离线

```
constexpr int N = 2e5 + 10;
array<int, 5> a[N], tmp[N];
template <typename T>
struct Fenwick {
    int n;
    vector<T> a;
    Fenwick(int _n = 0) : a(_n + 1), n(_n){}
    void init(int n) {
        a.assign(n + 1, T());
        this->n = n;
    }
    void add(int x, T v) {
        for (int i = x; i <= n; i += i & -i) {
            a[i] += v;
        }
    }
    T sum(int x) {
        auto ans = T();
        for (int i = x; i >= 1; i -= i & -i) {
            ans += a[i];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};

void solve() {
    int n, k;
    cin >> n >> k;
    Fenwick<int> fen(k);
    for(int i = 1; i <= n; i++) {
        cin >> a[i][0] >> a[i][1] >> a[i][2];
        a[i][3] = 1;
```

```
    }
    sort(a + 1, a + 1 + n);
    int t = 0;
    for(int i = 1; i <= n; i++) {
        if(i == 1 || a[i][0] != a[t][0] || a[i][1] != a[t][1] || a[i][2] != a[t]
[2]) {
            a[++t] = a[i];
        }
        else {
            a[t][3]++;
        }
    }//例题题目要求去重
    function<void(int, int)> cdq = [&](int l, int r) {
        if(l == r) return ;
        int mid = l + r >> 1;
        cdq(l, mid), cdq(mid + 1, r);
        int p1 = l, p2 = mid + 1;
        int p3 = 0;
        while(p1 <= mid || p2 <= r) {
            if(p2 > r || (p1 <= mid && make_pair(a[p1][1], a[p1][2]) <=
make_pair(a[p2][1], a[p2][2]))) {
                fen.add(a[p1][2], a[p1][3]);
                tmp[++p3] = a[p1++];
            }//左半部分
            else {
                a[p2][4] += fen.sum(a[p2][2]);
                tmp[++p3] = a[p2++];
            }//右半部分
        }
        for(int i = l; i <= mid; i++) fen.add(a[i][2], -a[i][3]);
        for(int i = 1; i <= p3; i++) a[l + i - 1] = tmp[i];
    };
    cdq(1, t);
    vector<int> ans(n);
    for(int i = 1; i <= t; i++) {
        ans[a[i][3] + a[i][4] - 1] += a[i][3];
    }
    for(int i = 0; i < n; i++) cout << ans[i] << '\n';
}
```

字符串

KMP

```
vector<int> kmp(string a) {
    int n = a.size() - 1; //开头有空格
    vector<int> f(n + 1);
    for(int i = 2, j = 0; i <= n; i++) {
        while(j && a[j + 1] != a[i]) j = f[j];
        if(a[j + 1] == a[i]) f[i] = ++j;
    }
    return f;
}
```

Z函数（拓展KMP）

约定：字符串下标以 0 为起点。

对于个长度为 n 的字符串 s 。定义函数 $z[i]$ 表示 s 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。 z 被称为 s 的 **Z 函数**。特别地，$z[0]=0$。1

```
vector<int> z_function(string &s) {
    int n = (int)s.length();
    vector<int> z(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] = 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) {
            l = i, r = i + z[i] - 1;
        }
    }
    return z;
}
```

字符串哈希

```
struct Hash {
    int Base, P;//常用模数1e9+7，998244353，常用base 131，13331
    vector<int> power, sum;
    void init(int base, int p, string &s) {
        //s需要从1开始
        power.resize(s.size()), sum.resize(s.size());
        P = p, Base = base, power[0] = 1;
        for(int i = 1; i < s.size(); i++) {
            power[i] = 1ll * power[i - 1] * Base % P;
            sum[i] = (1ll * sum[i - 1] * Base + s[i]) % P;
        }
    }
    int gethash(int l, int r) {
        return (sum[r] - 1ll * sum[l - 1] * power[r - l + 1] % P + P) % P;
    }
};
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
int P1 = rng() % (int)1e9 + 1, P2 = rng() % (int)1e9 + 1;//可能会很慢
```

manacher

```
auto manacher(string &s) {
    string t = "$#";
    for(auto c : s) t += c, t += '#';
    t += '^';
    int n = t.size();
    vector<int> p(n);
```

```
    int r = 0, mid;
    for(int i = 1; i < n; i++) {
        if(i <= r) p[i] = min(p[2 * mid - i], r - i + 1);
        else p[i] = 1;
        while(t[i - p[i]] == t[i + p[i]]) p[i] += 1;
        if(i + p[i] - 1 > r) r = i + p[i] - 1, mid = i;
    }
    return p;
}
    int res = 0;
    for(int i = 1; i < p.size(); i++) {
        res = max(res, p[i] - 1);
        int l = i - p[i] + 2, r = i + p[i] - 2;//以第i个字符为中心的最长回文边界
        if(l > r) continue;
    }//最长回文串长度
```

AC自动机

```
struct AC_automation {
    vector<vector<int>> tr, g;
    vector<int> val, fail;
    int idx = 0;
    vector<int> T;
    AC_automation(int m) : val(1), fail(1), T(m, 0){
        tr.emplace_back(T);
    };
    void insert(string &s, int v) {
        int p = 0;
        for (int i = 0; i < s.size(); i++) {
            int u = s[i] - 'a';
            if (!tr[p][u]) {
                tr[p][u] = ++idx;
                tr.emplace_back(T);
                val.emplace_back(0);
                fail.emplace_back(0);
            }
            p = tr[p][u];
        }
        val[p] += v;
    }
    void build() {
        queue<int> q;
        for(int i = 0; i < 26; i++)
            if(tr[0][i])
                q.push(tr[0][i]);
        while(!q.empty()) {
            int u = q.front();
            q.pop();
            for(int i = 0; i < 26; i++) {
                int v = tr[u][i];
                if(!v)
                    tr[u][i] = tr[fail[u]][i];
                else {
                    fail[v] = tr[fail[u]][i];
                    q.push(v);
                }
            }
```

```
        }
    }
    int find(string &s) {
        int ans = 0;
        for(int i = 0, j = 0; i < s.size(); i++) {
            int u = s[i] - 'a';
            j = tr[j][u];
            int p = j;
            while(p && val[p] >= 0) {
                ans += val[p];
                val[p] = -1;//优化为trie图
                p = fail[p];
            }
        }
        return ans;
    }
    void addfail() {
        g.resize(idx + 1);
        for(int i = 1; i <= idx; i++) {
            g[fail[i]].push_back(i);
        }
    }
};
```

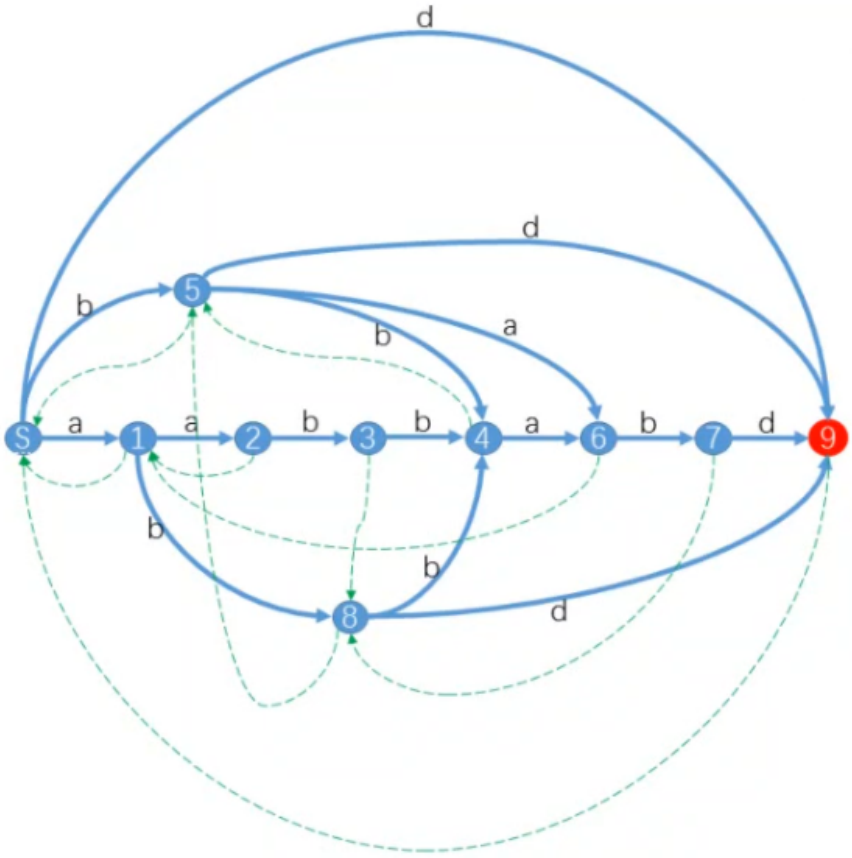
后缀数组

```
int t, n, m = 122, a[N], sa[N], c[N], x[N], y[N], rk[N], height[N];
string s;
void get_sa() {
    for(int i = 1; i <= n; i++) c[x[i] = s[i]]++;
    for(int i = 2; i <= m; i++) c[i] += c[i - 1];
    for(int i = n; i >= 1; i--) sa[c[x[i]]--] = i;
    for(int k = 1; k <= n; k <<= 1) {
        int num = 0;
        for(int i = n - k + 1; i <= n; i++) y[++num] = i;
        for(int i = 1; i <= n; i++) if(sa[i] > k) y[++num] = sa[i] - k;
        for(int i = 1; i <= m; i++) c[i] = 0;
        for(int i = 1; i <= n; i++) c[x[i]]++;
        for(int i = 2; i <= m; i++) c[i] += c[i - 1];
        for(int i = n; i >= 1; i--) sa[c[x[y[i]]]--] = y[i], y[i] = 0;
        for(int i = 1; i <= n; i++) swap(x[i], y[i]);
        x[sa[1]] = 1, num = 1;
        for(int i = 2; i <= n; i++) x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] &&
y[sa[i] + k] == y[sa[i - 1] + k]) ? num : ++num;
        if(num == n) break;
        m = num;
    }
}
void get_height() {
    for(int i = 1; i <= n; i++) rk[sa[i]] = i;
    for(int i = 1, k = 0; i <= n; i++) {
        if(rk[i] == 1) continue;
        if(k) k--;
        int j = sa[rk[i] - 1];
        while(i + k <= n && j + k <= n && s[i + k] == s[j + k]) k++;
        height[rk[i]] = k;
    }
```

```
}//排名为 i 的非空后缀与排名为 i-1 的非空后缀的最长公共前缀的长度
```

后缀自动机

比如对于字符串 $S = aabbabd$，它的后缀自动机是




```
const int N = 1e6 + 10; //endpoint的最大个数是2*n级别，N要开两倍
string s;
int tot = 1, last = 1;
int cnt[N], len[N], p[N], sum[N]; //cnt是这个子串的数量，sum是以这个子串为前缀的数量
struct Node {
    int len, fa;
    int ch[26];
} node[N];
void extend(int c) {
    int p = last, np = last = ++tot;
    cnt[tot] = 1;
    node[np].len = node[p].len + 1;
    for(; p && !node[p].ch[c]; p = node[p].fa) node[p].ch[c] = np;
    if (!p) node[np].fa = 1;
    else {
        int q = node[p].ch[c];
        if (node[q].len == node[p].len + 1) node[np].fa = q;
        else {
            int nq = ++tot;
            node[nq] = node[q], node[nq].len = node[p].len + 1;
            node[q].fa = node[np].fa = nq;
            for(; p && node[p].ch[c] == q; p = node[p].fa) node[p].ch[c] = nq;
        }
    }
}
```

```
int h[N], e[N], ne[N], idx;
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx++;
}
void build() {
    for(char c : s) extend(c - 'a');
    for(int i = 1; i <= tot; i++) len[node[i].len]++;
    for(int i = 1; i <= s.size(); i++) len[i] += len[i - 1];
    for(int i = 1; i <= tot; i++) p[len[node[i].len]--] = i;
    for(int i = tot; i >= 1; i--) cnt[node[p[i]].fa] += cnt[p[i]];
    sum[1] = cnt[1] = 0;
    for(int i = 1; i <= tot; i++)
        sum[i] = 1;
    for(int i = tot; i >= 1; i--) {
        for(int j = 0; j < 26; j++) {
            if(node[p[i]].ch[j]) {
                sum[p[i]] += sum[node[p[i]].ch[j]];
            }
        }
    }
    for(int i = 1; i <= tot; i++) {
        node[i].len = node[i].fa = cnt[i] = p[i] = len[i] = 0;
        for(int j = 0; j < 26; j++)
            node[i].ch[j] = 0;
    }//清空
}
```

回文自动机

```
string s;
int len[N], fail[N], tr[N][26], tot, cur, cnt[N];
void init() {
    for(int i = 0; i <= tot; i++)
        for(int j = 0; j < 26; j++)
            tr[i][j] = 0;
    fail[0] = 1, len[1] = -1, tot = 1, cur = 0;
}
int getfail(int x, int i) {
    while(s[i - len[x] - 1] != s[i])
        x = fail[x];
    return x;
}
void insert() {
    for(int i = 0; i < s.size(); i++) {
        int p = getfail(cur, i), u = s[i] - 'a';
        if(!tr[p][u]) {
            fail[++tot] = tr[getfail(fail[p], i)][u];
            tr[p][u] = tot;
            len[tot] = len[p] + 2;
        }
        cnt[cur = tr[p][u]]++;
    }
    for(int i = tot; i >= 2; i--)
        cnt[fail[i]] += cnt[i];
}

void solve() {
```

```
    cin >> s;
    init();
    insert();
}
```

数学知识

线性筛质数

```
int p[N], minp[N], pcnt;
void get_prime(int n) {
    for(int i = 2; i <= n; i++) {
        if(!minp[i]) {
            minp[i] = i;
            p[++pcnt] = i;
        }
        for(int j = 1; p[j] <= n / i && j <= pcnt; j++) {
            minp[i * p[j]] = p[j];
            if(i % p[j] == 0) break;
        }
    }
}
```

快速幂

```
i64 power(i64 a, i64 b) {
    a %= P;
    i64 res = 1;
    while (b) {
        if (b & 1) res = res * a % P;
        a = a * a % P;
        b >>= 1;
    }
    return res % P;
}
```

最大公约数

```
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
```

拓展欧几里得

解出来的 x 和 y 是特解，通解为 $x' = x + k \times (b/gcd(a,b)), y' = y - k \times (a/gcd(a,b))$, k是整数

```
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
    if(!b) {
        x = 1;
        y = 0;
        return a;
    }
    else {
        i64 g = exgcd(b, a % b, y, x);
        y -= a / b * x;
        return g;
    }
}
```

中国剩余定理

```
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
    if(!b) {
        x = 1;
        y = 0;
        return a;
    }
    else {
        i64 g = exgcd(b, a % b, y, x);
        y -= a / b * x;
        return g;
    }
}
int m[N], a[N];
i64 CRT(){
    int n;
    cin >> n;
    i64 mul = 1;
    for(int i = 1; i <= n; i++) {
        cin >> m[i] >> a[i];
        mul *= m[i];
    }
    i64 res = 0;
    for(int i = 1; i <= n; i++) {
        i64 M = mul / m[i];
        i64 x, y;
        exgcd(M, m[i], x, y);
        res = (res + a[i] * M * ((x % m[i] + m[i]) % m[i]));
    }
    return res % mul;
}
```

拓展中国剩余定理

```
i64 exgcd(i64 a, i64 b, i64 &x, i64 &y) {
    if(!b) {
        x = 1;
        y = 0;
        return a;
```

```
    }
    else {
        i64 g = exgcd(b, a % b, y, x);
        y -= a / b * x;
        return g;
    }
}
i64 m[N], a[N];
i64 exCRT() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> m[i] >> a[i];
    i64 res = a[1], p = m[1];
    for(int i = 2; i <= n; i++) {
        i64 x, y;
        __int128 b = (a[i] - res % m[i] + m[i]) % m[i];
        i64 g = exgcd(p, m[i], x, y);
        if(b % g) return -1;//无解
        i64 k = m[i] / g;
        x = (x % k * (b / g % k) % k + k) % k;//k很大时会爆ll，所以b需要开__int128
        res += x * p;
        p *= k;
    }
    return res;
}
```

约数个数

$N = {p_1}^{a_1} * {p_2}^{a_2} \cdots * {p_k}^{a_k}$

$ans = (a_1 + 1)(a_2 + 1) \cdots (a_k + 1)$

```
using i64 = long long;
const int P = 1e9 + 7;
int n, a[110];
unordered_map <int, int> primes;
int main(){
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        for(int j = 2; j <= a[i] / j; j++) {
            while(a[i] % j == 0) {
                a[i] /= j;
                primes[j] ++;
            }
        }
        if(a[i] != 1) primes[a[i]] ++;
    }
    i64 ans = 1;
    for(auto prime: primes) {
        ans = ans * (prime.second + 1) % P;
    }
    printf("%lld", ans);
    return 0;
}
```

约数之和

$(p_1^0+p_1^1+\ldots+p_1^{c1})*\ldots*(p_k^0+p_k^1+\ldots+p_k^{ck})$

```
using i64 = long long;
const int mod = 1e9 + 7;
int n;
unordered_map <int, int> primes;
int main(){
    scanf("%d", &n);
    while(n--) {
        int x;
        scanf("%d", &x);
        for(int i = 2; i <= x / i; i++) {
            while(x % i == 0) {
                primes[i]++;
                x /= i;
            }
        }
        if(x > 1) primes[x]++;
    }
    i64 res = 1;
    for(auto i: primes) {
        i64 t = 1;
        i64 a = i.first, b = i.second;
        while(b--) {
            t = (t * a + 1) % mod;
        }
        res = res * t % mod;
    }
    printf("%lld", res);
    return 0;
}
```

dfs求约数（利用大数约数数量大约为 $n^{1/3}$ 的结论）

```
function<void(int, i64)> dfs = [&](int u, i64 num) {
        if(u == (int)p.size()) {

            return ;
        }
        i64 res = num;
        for(int i = 0; i <= p[u].second; i++) {
            dfs(u + 1, res);
            res *= p[u].first;
        }
    };
```

欧拉函数

```
int prime[N], phi[N];
bool st[N];
void get_euler(int n) {
    phi[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!st[i]) {
            prime[++m] = i;
            phi[i] = i - 1;
        }
        for(int j = 1; prime[j] <= n / i; j++) {
            st[i * prime[j]] = 1;
            if(i % prime[j] == 0) {
                phi[i * prime[j]] = prime[j] * phi[i];
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
}
```

莫比乌斯反演

莫比乌斯反演是数论中的重要内容。对于一些函数 $f(n)$，如果很难直接求出它的值，而容易求出其倍数和或约数和 $g(n)$，那么可以通过莫比乌斯反演简化运算，求得 $f(n)$ 的值。

性质

莫比乌斯函数不仅是积性函数，还有如下性质：

$$\sum_{d|n}\mu(d)=\begin{cases}1 & n=1\\ 0 & n\neq 1\end{cases}$$

即 $\sum_{d|n}\mu(d)=\varepsilon(n)$，$\mu*1=\varepsilon$

证明

设 $n=\prod_{i=1}^{k}{p_i}^{c_i}, n'=\prod_{i=1}^{k}p_i$

那么 $\sum_{d|n}\mu(d)=\sum_{d|n'}\mu(d)=\sum_{i=0}^{k}\binom{k}{i}\cdot(-1)^i=(1+(-1))^k$

根据二项式定理，易知该式子的值在 $k=0$ 即 $n=1$ 时值为 1 否则为 0，这也同时证明了 $\sum_{d|n}\mu(d)=[n=1]=\varepsilon(n)$ 以及 $\mu*1=\varepsilon$

如果有 $f(n)=\sum_{d|n}g(d)$, 那么有 $g(n)=\sum_{d|n}\mu(d)f(\frac{n}{d})$

如果有 $f(n)=\sum_{n|d}g(d)$, 那么有 $g(n)=\sum_{n|d}\mu(\frac{d}{n})f(d)$

```
void get_mu(int n) {
    mu[1] = 1;
```

```
    for(int i = 2; i <= n; i++) {
        if(!np[i]) {
            p[++m] = i;
            mu[i] = -1;
        }
        for(int j = 1; j <= m && p[j] <= n / i; j++) {
            np[i * p[j]] = 1;
            if(i % p[j] == 0) {
                mu[i * p[j]] = 0;
                break;
            }
            mu[i * p[j]] = -mu[i];
        }
    }
    for(int i = 1; i <= n; i++) {
        sum[i] = sum[i - 1] + mu[i];
    }
}
```

逆元

线性求逆元

```
inv[1] = 1;
for (int i = 2; i <= n; ++i) {
    inv[i] = (ll)(P - P / i) * inv[P % i] % P;
}
```

费马小定理求逆元

```
i64 power(i64 a, i64 b) {
    a %= P;
    i64 res = 1;
    while (b) {
        if (b & 1) res = res * a % P;
        a = a * a % P;
        b >>= 1;
    }
    return res % P;
}
i64 inv(i64 x) {
    return power(x, P - 2);
}
```

组合数

```
constexpr int P = 1e9 + 7;
i64 power(i64 a, i64 b) {
    a %= P;
    i64 res = 1 % P;
    while (b) {
        if (b & 1) {
            res = res * a % P;
        }
        a = a * a % P;
```

```
        b >>= 1;
    }
    return res;
}
vector<i64> fact, inv_fact, _inv;
i64 c(int n, int m) {
    if(n < m || m < 0) return 0;
    return fact[n] * inv_fact[m] % P * inv_fact[n - m] % P;
}
i64 inv(i64 x) {
    return power(x, P - 2);
}
void init(int n) {
    fact.resize(n + 1);
    inv_fact.resize(n + 1);
    _inv.resize(n + 1);
    fact[0] = _inv[1] = 1;
    for (int i = 1; i <= n; i++) fact[i] = fact[i - 1] * i % P;
    inv_fact[n] = inv(fact[n]);
    for (int i = n - 1; i >= 0; i--) inv_fact[i] = inv_fact[i + 1] * (i + 1) %
P;
    for(int i = 2; i <= n; i++) _inv[i] = fact[i - 1] * inv_fact[i] % P;
}
```

矩阵快速幂

```
struct Matrix {
    vector<vector<i64>> a;
    int n, m;
    Matrix(int _n, int _m) : n(_n), m(_m), a(_n, vector<i64> (_m)){}
};
Matrix operator * (const Matrix &A, const Matrix &B) {
    Matrix C(A.n, B.m);
    for (int i = 0; i < A.n; i++) {
        for (int j = 0; j < A.m; j++) {
            for (int k = 0; k < B.m; k++) {
                C.a[i][k] = C.a[i][k] + A.a[i][j] * B.a[j][k];
            }
        }
    }
    return C;
}
Matrix operator * (const Matrix &A, const Matrix &B) {
    Matrix C(A.n, B.m);
    for (int i = 0; i < A.n; i++) {
        for (int j = 0; j < A.m; j++) {
            for (int k = 0; k < B.m; k++) {
                C.a[i][k] = max(C.a[i][k], A.a[i][j] + B.a[j][k]);
            }
        }
    }
    return C;
}//广义矩阵快速幂
    auto power = [&](Matrix a, Matrix b, i64 c) {
        c--;
        Matrix ans = a;
        while(c) {
```

```
            if(c & 1) ans = ans * b;
            b = b * b;
            c >>= 1;
        }
        return ans;
    };
```

高斯消元

```
const int N = 110;
const double eps = 1e-8;
int n;
double a[N][N];
int gauss() { // 高斯消元，答案存于a[i][n]中，0 <= i < n
    int c, r;
    for (c = 0, r = 0; c < n; c++) {
        int t = r;
        for (int i = r; i < n; i++)  if(fabs(a[i][c]) > fabs(a[t][c])) t = i; //
找绝对值最大的行
        if (fabs(a[t][c]) < eps) continue;
        for (int i = c; i <= n; i++) swap(a[t][i], a[r][i]);  // 将绝对值最大的行换
到最顶端
        for (int i = n; i >= c; i--) a[r][i] /= a[r][c];  // 将当前行的首位变成1
        for (int i = r + 1; i < n; i++)  // 用当前行将下面所有的列消成0
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j--) a[i][j] -= a[r][j] * a[i][c];
        r++;
    }
    if (r < n) {
        for (int i = r; i < n; i++)
            if (fabs(a[i][n]) > eps) return 2; // 无解
        return 1; // 有无穷多组解
    }
    for (int i = n - 1; i >= 0; i -- )
        for (int j = i + 1; j < n; j ++ )
            a[i][n] -= a[i][j] * a[j][n];
    return 0; // 有唯一解
}
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < n + 1; j ++ )
            scanf("%lf", &a[i][j]);
    int t = gauss();
    if (t == 2) puts("No solution");
    else if (t == 1) puts("Infinite group solutions");
    else {
        for (int i = 0; i < n; i ++ ) {
            if (fabs(a[i][n]) < eps) a[i][n] = 0;  // 去掉输出 -0.00 的情况
            printf("%.2lf\n", a[i][n]);
        }
    }
    return 0;
}
```

BSGS

给定正整数 a , p , b 数据保证 a 和 p 互质。

求满足 $a^x \equiv b \ (mod \ p)$ 的最小非负整数 x。

```
int bsgs(int a, int b, int p) {
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int, int> hash;
    for (int i = 0, j = b % p; i < k; i++) {
        hash[j] = i;
        j = 1i64 * j * a % p;
    }
    int ak = 1;
    for (int i = 0; i < k; i ++ ) ak = 1i64 * ak * a % p;
    for (int i = 1, j = ak; i <= k; i++) {
        if (hash.count(j)) return 1i64 * i * k - hash[j];
        j = 1i64 * j * ak % p;
    }
    return -1;
}
```

拓展BSGS

给定整数 a , p , b 。

求满足 $a^x \equiv b \ (mod \ p)$ 的最小非负整数 x。

```
const int INF = 1e8;
int exgcd(int a, int b, int& x, int& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
int bsgs(int a, int b, int p) {
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int, int> hash;
    for (int i = 0, j = b % p; i < k; i++) {
        hash[j] = i;
        j = 1i64 * j * a % p;
    }
    int ak = 1;
    for (int i = 0; i < k; i ++ ) ak = 1i64 * ak * a % p;
    for (int i = 1, j = ak; i <= k; i++) {
        if (hash.count(j)) return i * k - hash[j];
        j = 1i64 * j * ak % p;
    }
    return -INF;
}
int exbsgs(int a, int b, int p) {
    b = (b % p + p) % p;
```

```
    if (1 % p == b % p) return 0;
    int x, y;
    int d = exgcd(a, p, x, y);
    if (d > 1) {
        if (b % d) return -INF;
        exgcd(a / d, p / d, x, y);
        return exbsgs(a, 1i64 * b / d * x % (p / d), p / d) + 1;
    }
    return bsgs(a, b, p);
}
```

FFT

```
constexpr double PI = acos(-1);
struct Complex {
    double x, y;
    Complex operator+ (const Complex& t) const {
        return {x + t.x, y + t.y};
    }
    Complex operator- (const Complex& t) const {
        return {x - t.x, y - t.y};
    }
    Complex operator* (const Complex& t) const {
        return {x * t.x - y * t.y, x * t.y + y * t.x};
    }
}
auto fft = [&](auto &a, int inv, int tot, auto &rev) {
    for (int i = 0; i < tot; i++) {
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    }
    for (int mid = 1; mid < tot; mid <<= 1) {
        auto w1 = Complex({cos(PI / mid), inv * sin(PI / mid)});
        for (int i = 0; i < tot; i += mid * 2) {
            auto wk = Complex({1, 0});
            for (int j = 0; j < mid; j++, wk = wk * w1) {
                auto x = a[i + j], y = wk * a[i + j + mid];
                a[i + j] = x + y, a[i + j + mid] = x - y;
            }
        }
    }
};
void solve() {
    int n, m;
    cin >> n >> m;
    int bit = 0;
    while((1 << bit) < n + m + 1) bit++;
    int tot = 1 << bit;
    vector<Complex> a(2 * (n + m + 1) + 1), b(2 * (n + m + 1) + 1);
    for(int i = 0; i <= n; i++) cin >> a[i].x;
    for(int i = 0; i <= m; i++) cin >> b[i].x;
    for(int i = 0; i < tot; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit
- 1));
    fft(a, 1, tot, rev), fft(b, 1, tot, rev);
    for (int i = 0; i < tot; i++) a[i] = a[i] * b[i];
    fft(a, -1, tot, rev);
    for(int i = 0; i <= n + m; i++) cout << (long long)(a[i].x / tot + 0.5) << '
';//注意值域可能超出int范围
```

```
}
```

三次变两次

```
constexpr int N = 2e6 + 10; //N要大于(n+m+1)*2
constexpr double PI = acos(-1);
int n, m;
struct Complex {
    double x, y;
    Complex operator+ (const Complex& t) const {
        return {x + t.x, y + t.y};
    }
    Complex operator- (const Complex& t) const {
        return {x - t.x, y - t.y};
    }
    Complex operator* (const Complex& t) const {
        return {x * t.x - y * t.y, x * t.y + y * t.x};
    }
} a[N];
int rev[N], bit, tot;
void fft(Complex a[], int inv) {
    for(int i = 0; i < tot; i++) {
        if(i < rev[i]) {
            swap(a[i], a[rev[i]]);
        }
    }
    for(int mid = 1; mid < tot; mid <<= 1) {
        auto w1 = Complex({cos(PI / mid), inv * sin(PI / mid)});
        for(int i = 0; i < tot; i += mid * 2) {
            auto wk = Complex({1, 0});
            for(int j = 0; j < mid; j++, wk = wk * w1) {
                auto x = a[i + j], y = wk * a[i + j + mid];
                a[i + j] = x + y, a[i + j + mid] = x - y;
            }
        }
    }
    if(inv != 1) {
        for(int i = 0; i < tot; i++) {
            a[i].x /= tot, a[i].y /= tot;
        }
    }
}
void solve() {
    cin >> n >> m;
    for(int i = 0; i <= n; i++) {
        cin >> a[i].x;
    }
    for(int i = 0; i <= m; i++) {
        cin >> a[i].y;
    }
    while((1 << bit) < n + m + 1) {
        bit++;
    }
    tot = 1 << bit;
    for(int i = 0; i < tot; i++) {
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
    }
```

```
    fft(a, 1);
    for(int i = 0; i < tot; i++) {
        a[i] = a[i] * a[i];
    }
    fft(a, -1);
    for(int i = 0; i <= n + m; i++) cout << (long long)(a[i].y / 2 + 0.5) << '
';
}
```

分治FFT

```
int rev[N];
vector<Complex> cal(int l, int r) {
    if(l == r) {
        //vector<Complex> a(2);
        //a[0].x = 1 - p[l];
        //a[1].x = p[l];
        return a;
    }
    int mid = (l + r) / 2;
    auto a = cal(l, mid);
    auto b = cal(mid + 1, r);
    int bit = 0;
    int n = (int)a.size(), m = (int)b.size();
    a.resize(2 * (n + m + 1) + 1);
    b.resize(2 * (n + m + 1) + 1);
    while((1 << bit) < n + m + 1) bit++;
    int tot = 1 << bit;
    for(int i = 0; i < tot; i++) {
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
    }
    fft(a, 1, tot, rev), fft(b, 1, tot, rev);
    for (int i = 0; i < tot; i++) a[i] = a[i] * b[i];
    fft(a, -1, tot, rev);
    vector<Complex> c(n + m - 1);
    for(int i = 0; i <= n + m - 2; i++) {
        c[i].x = a[i].x / tot;
    }
    return c;
}
void solve() {
    cin >> n;
    //for(int i = 1; i <= n; i++) {
    //    cin >> p[i];
    //}
    auto ans = cal(1, n);
    double sum = 0;
    //for(int i = n / 2 + 1; i <= n; i++) {
    //    sum += ans[i].x;
    //}
    cout << fixed << setprecision(10) << sum;
}
```

NTT

```
constexpr int N = 4e6 + 10;//(n+m+1)*2
constexpr i64 P = 998244353;
constexpr int g = 3;
i64 a[N], b[N], rev[N], gn[N];
i64 power(i64 a, i64 b) {
    a %= P;
    i64 res = 1;
    while (b) {
        if (b & 1) res = (__int128)res * a % P;
        a = (__int128)a * a % P;
        b >>= 1;
    }
    return res % P;
}
void ntt(auto a[], int len, auto opt) {
    for (int i = 0; i < len; i++)
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int m = i >> 1;
        i64 gw = power(g, (opt == 1) ? ((P - 1) / (m << 1)) : (P - 1 - (P - 1) /
(m << 1)));
        gn[0] = 1;
        for (int j = 1; j < m; j++)
            gn[j] = (__int128)gn[j - 1] * gw % P;
        for (int j = 0; j < len; j += i) {
            for (int k = j; k < j + m; k++) {
                i64 x = a[k], y = (__int128)gn[k - j] * a[k + m] % P;
                a[k] = (x + y) % P;
                a[k + m] = (x - y + P) % P;
            }
        }
    }
}
void solve() {
    int n, m;
    cin >> n >> m;
    for(int i = 0; i <= n; i++) cin >> a[i];
    for(int i = 0; i <= m; i++) cin >> b[i];
    int bit = 0;
    while((1 << bit) < n + m + 1) bit++;
    int tot = 1 << bit;
    for(int i = 0; i < tot; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit
- 1));
    ntt(a, tot, 1), ntt(b, tot, 1);
    for (int i = 0; i < tot; i++)
        a[i] = (__int128)a[i] * b[i] % P;
    ntt(a, tot, -1);
    i64 inv = power(tot, P - 2);
    for (int i = 0; i <= n + m; i++) a[i] = (__int128)a[i] * inv % P;
    for (int i = 0; i <= n + m; i++) cout << a[i] << ' ';
}
```

分治NTT

```
constexpr int N = 1e6 + 10;
constexpr i64 P = 998244353;
constexpr int g = 3;
int n;
using Poly = vector<i64>;
i64 gn[N], rev[N];
i64 power(i64 a, i64 b) {
    a %= P;
    i64 res = 1;
    while (b) {
        if (b & 1) res = res * a % P;
        a = a * a % P;
        b >>= 1;
    }
    return res % P;
}
i64 inv(i64 x) {
    return power(x, P - 2);
}
int gw[2][N];
void init(int n) {
    for(int i = 1; i <= n; i *= 2) {
        gw[0][i] = power(g, (P - 1 - (P - 1) / (i << 1)));
        gw[1][i] = power(g, ((P - 1) / (i << 1)));
    }
}
void ntt(Poly &a, int len, auto opt) {
    for (int i = 0; i < len; i++)
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    for (int i = 2; i <= len; i <<= 1) {
        int m = i >> 1;
        i64 w = gw[opt == 1][m];
        //i64 gw = power(g, (opt == 1) ? ((P - 1) / (m << 1)) : (P - 1 - (P - 1)
/ (m << 1)));//预处理gw会快很多
        gn[0] = 1;
        for (int j = 1; j < m; j++)
            gn[j] = gn[j - 1] * w % P;
        for (int j = 0; j < len; j += i) {
            for (int k = j; k < j + m; k++) {
                i64 x = a[k], y = gn[k - j] * a[k + m] % P;
                a[k] = (x + y) % P;
                a[k + m] = (x - y + P) % P;
            }
        }
    }
}
Poly operator*(Poly a, Poly b) {
    int n = a.size() - 1, m = b.size() - 1;
    int bit = 0;
    while((1 << bit) < n + m + 1) bit++;
    int tot = 1 << bit;
    for(int i = 0; i < tot; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit
- 1));
    a.resize(tot);
```

```
    b.resize(tot);
    ntt(a, tot, 1), ntt(b, tot, 1);
    for (int i = 0; i < tot; i++)
        a[i] = a[i] * b[i] % P;
    ntt(a, tot, -1);
    i64 inv = power(tot, P - 2);
    for (int i = 0; i <= n + m; i++) a[i] = a[i] * inv % P;
    a.resize(n + m + 1);
    return a;

}
Poly MulAll(int l, int r, vector<Poly> &a) {
    if(l == r) return a[l];
    int m = l + r >> 1;
    auto v = MulAll(l, m, a) * MulAll(m + 1, r, a);
    return v;
}
void solve() {
    init(n); //预处理gw，n的数量级为答案的最高幂次
    vector<Poly> a(n + 1);
    for(int i = 1; i <= n; i++) {
        //int x;
        //cin >> x;
        //a[i] = {1, x};
    }
    auto v = MulAll(1, n, a);
}
```

FWT

```
    int m;
    cin >> m;
    int n = 1 << m;
    vector<i64> a(n), b(n);
    for(int i = 0; i < n; i++) {
        cin >> a[i];
    }
    for(int i = 0; i < n; i++) {
        cin >> b[i];
    }
    auto fwtor = [&](vector<i64> &a, int op) {
        for(int i = 1; i < n; i *= 2) {
            for(int p = i << 1, j = 0; j < n; j += p) {
                for(int k = 0; k < i; k++) {
                    a[i + j + k] = (a[i + j + k] + a[j + k] * op + P) % P;
                }
            }
        }
    };
    auto fwt_or = [&](vector<i64> &a, vector<i64> &b) {
        auto A = a, B = b;
        fwtor(A, 1), fwtor(B, 1);
        for(int i = 0; i < n; i++) A[i] = A[i] * B[i] % P;
        fwtor(A, -1);
        return A;
    };
    auto A = fwt_or(a, b);
```

```
    auto fwtand = [&](vector<i64> &a, int op) {
        for(int i = 1; i < n; i *= 2) {
            for(int p = i << 1, j = 0; j < n; j += p) {
                for(int k = 0; k < i; k++) {
                    a[j + k] = (a[j + k] + a[i + j + k] * op + P) % P;
                }
            }
        }
    };
    auto fwt_and = [&](vector<i64> &a, vector<i64> &b) {
        auto A = a, B = b;
        fwtand(A, 1), fwtand(B, 1);
        for(int i = 0; i < n; i++) A[i] = A[i] * B[i] % P;
        fwtand(A, -1);
        return A;
    };
    auto B = fwt_and(a, b);
    auto fwtxor = [&](vector<i64> &a, int op) {
        for(int i = 1; i < n; i *= 2) {
            for(int p = i << 1, j = 0; j < n; j += p) {
                for(int k = 0; k < i; k++) {
                    i64 x = a[j + k], y = a[i + j + k];
                    a[j + k] = (x + y) % P;
                    a[i + j + k] = (x - y + P) % P;
                    if(op == -1) {
                        a[j + k] = a[j + k] * inv2 % P;
                        a[i + j + k] = a[i + j + k] * inv2 % P;
                    }
                }
            }
        }
    };
    auto fwt_xor = [&](vector<i64> &a, vector<i64> &b) {
        auto A = a, B = b;
        fwtxor(A, 1), fwtxor(B, 1);
        for(int i = 0; i < n; i++) A[i] = A[i] * B[i] % P;
        fwtxor(A, -1);
        return A;
    };
    auto C = fwt_xor(a, b);
    for(int i = 0; i < n; i++) cout << A[i] << " ";
    cout << '\n';
    for(int i = 0; i < n; i++) cout << B[i] << " ";
    cout << '\n';
    for(int i = 0; i < n; i++) cout << C[i] << " ";
```

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;
}
vector<i64> factorize(i64 n) {
    vector<i64> p;
    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, abs(x - y), n);
                if (lam % 127 == 0) {
```

```
                    d = gcd(v, n);
                    v = 1;
                }
                if (power == lam) {
                    x = y;
                    power *= 2;
                    lam = 0;
                    d = gcd(v, n);
                    v = 1;
                }
            }
            if (d != n) {
                f(d);
                f(n / d);
                return;
            }
            ++x0;
        }
    };
    f(n);
    sort(p.begin(), p.end());
    return p;
}
```

类欧几里得算法

分数类

```
struct Frac {
    i128 x, y;
    Frac(i128 a = 0, i128 b = 1) {
        if(b < 0) a = -a, b = -b; //保证分母是正数
        x = a, y = b;
    }
};
bool operator < (const Frac &a, const Frac &b) {
    return a.x * b.y < a.y * b.x;
}
bool operator <= (const Frac &a, const Frac &b) {
    return a.x * b.y <= a.y * b.x;
}
bool operator == (const Frac &a, const Frac &b) {
    return a.x * b.y == a.y * b.x;
}
bool operator > (const Frac &a, const Frac &b) {
    return a.x * b.y > a.y * b.x;
}
bool operator >= (const Frac &a, const Frac &b) {
    return a.x * b.y >= a.y * b.x;
}
Frac operator + (const Frac &a, const Frac &b) {
    return Frac(a.x * b.y + a.y * b.x, a.y * b.y);
}
Frac operator - (const Frac &a, const Frac &b) {
    return Frac(a.x * b.y - a.y * b.x, a.y * b.y);
}
```

```
Frac operator * (const Frac &a, const Frac &b) {
    return Frac(a.x * b.x, a.y * b.y);
}
struct Line {
    Frac k, b;
} line[N];
pair<Frac, Frac> LineIntersection(Line &a, Line &b) {
    Frac frac1(b.b - a.b), frac2(a.k - b.k);
    return make_pair(frac1, frac2);
    //return make_pair(b.b - a.b, a.k - b.k);
} //求线段交点的横坐标
//用的过程中小心爆int128，在abc372G中因为求线段交点的大小关系时爆int128debug很久
```

动态规划

换根dp

有一棵 n 个节点的树，节点编号从 1 到 n。请求出每个节点到其他所有节点的距离的和。

定义两个节点的距离为它们之间的简单路径上经过了多少条边。

http://oj.daimayuan.top/course/5/problem/224

```
vector<int> g[N];
int n, sz[N];
i64 dn[N], up[N];
void dfs1(int u, int fa) {
    sz[u] = 1;
    for(int v : g[u]) {
        if(v == fa) continue;
        dfs1(v, u);
        sz[u] += sz[v];
        dn[u] += dn[v] + sz[v];
    }
}
void dfs2(int u, int fa) {
    for(int v : g[u]) {
        if(v == fa) continue;
        up[v] = up[u] + dn[u] - (dn[v] + sz[v]) + n - sz[v];
        dfs2(v, u);
    }
}

void solve() {
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }

    dfs1(1, 0);
    dfs2(1, 0);
```

```
    for(int i = 1; i <= n; i++) {
        cout << up[i] + dn[i] << '\n';
    }
    //dn是节点下面的点到该节点的距离和，up是节点上面的点到该节点的距离和
}
```

最长上升子序列

```
template <typename T>
struct Fenwick {
    int n;
    vector<T> a;
    Fenwick(int _n = 0) : a(_n + 1), n(_n){}
    void init(int n) {
        a.assign(n + 1, T());
        this->n = n;
    }
    void add(int x, T v) {
        for (int i = x; i <= n; i += i & -i) {
            a[i] = max(a[i], v);
        }
    }
    T sum(int x) {
        int ans = 0;
        for (int i = x; i >= 1; i -= i & -i) {
            ans = max(ans, a[i]);
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
    int kth(T k) {
        int x = 0;
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i]) {
                x += i;
                k -= a[x];
            }
        }
        return x;
    }
};
    vector<int> b = a;
    sort(b.begin(), b.end());
    auto find(int x) = [&](int x) {
        return lower_bound(b + 1, b + 1 + n, x) - b;
    };
    Fenwick<int> fen1(n), fen2(n);
    vector<int> l(n + 1), r(n + 1);
    for(int i = 1; i <= n; i++) {
        int pos = find(a[i]);
        int v = fen1.sum(pos - 1);
        l[i] = v + 1;
        fen1.add(pos, v + 1);
    }
    for(int i = n; i >= 1; i--) {
```

```
        int pos = n - find(a[i]) + 1;
        int v = fen2.sum(pos - 1);
        r[i] = v + 1;
        fen2.add(pos, v + 1);
    }
```

单调队列优化多重背包

时间复杂度 $O(nm)$

```
void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> dp(m + 1), ndp(m + 1);
    vector<int> q(m + 1);
    for(int i = 1; i <= n; i++) {
        int v, w, s;
        cin >> v >> w >> s;
        for(int r = 0; r < v; r++) {//r是模数
            int hh = 0, tt = -1;
            for(int j = r; j <= m; j += v) {
                while(hh <= tt && q[hh] < j - s * v) hh++;
                while(hh <= tt && dp[q[tt]] + (j - q[tt]) / v * w <= dp[j]) tt-
-;
                q[++tt] = j;
                ndp[j] = dp[q[hh]] + (j - q[hh]) / v * w;
            }
        }
        swap(dp, ndp);
    }
    cout << dp[m];
}
```

斜率优化dp

```
void solve() {
    int n;
    i64 L;
    cin >> n >> L;
    L++;
    vector<i64> dp(n + 1), s(n + 1), q(n + 1);
    for(int i = 1; i <= n; i++) {
        cin >> s[i];
        s[i] += s[i - 1] + 1;
    }
    auto sq = [&](int x) {
        return 1i64 * x * x;
    };
    auto Y = [&](int i) {
        return dp[i] + sq(s[i]);
    };
    auto X = [&](int i) {
        return s[i];
    };
```

```
    auto slope = [&](int i, int j) {
        return 1. * (Y(j) - Y(i)) / (X(j) - X(i));
    };
    int hh = 0, tt = 0;
    for(int i = 1; i <= n; i++) {
        while(hh < tt && slope(q[hh], q[hh + 1]) <= 2 * (s[i] - L)) hh++;
        int j = q[hh];
        dp[i] = dp[j] + sq(s[i] - s[j] - L);
        while(hh < tt && slope(q[tt - 1], q[tt]) >= slope(q[tt - 1], i)) tt--;
        q[++tt] = i;
    }
    cout << dp[n];
}
```

数位dp

统计0~9 在 [a,b] 中出现了多少次

```
int num[20];
i64 dp[20][20];
i64 dfs(int pos, bool limit, bool lead, int sum, int d) {
    i64 ans = 0;
    if(!pos) return sum;
    if(!limit && lead && dp[pos][sum] != -1) return dp[pos][sum];
    int up = 9;
    if(limit) up = num[pos];
    for(int j = 0; j <= up; j++) ans += dfs(pos - 1, (j == up) && limit, lead ||
j, sum + ((j || lead) && (j == d)), d);
    if(!limit && lead) dp[pos][sum] = ans;
    return ans;
}//lead=0表示存在前导0
i64 get(i64 x, int d) {
    int len = 0;
    while(x) {
        num[++len] = x % 10;
        x /= 10;
    }
    memset(dp, -1, sizeof dp);
    return dfs(len, 1, 0, 0, d);
}
```

不同子序列的个数

有一个长度为 N 的数列 A_i。

你可以进行若干次，最多 $N-1$ 次操作，选择相邻的两个数，删去他们，并在原位置放上他们两个的和。

现在你需要求出可能产生的序列个数。

做法：一种合法的序列就是由前缀和中按顺序任选其中几个确定，那本质不同的序列也就是有多少种不同的选择的方法，也就是子序列的数量。

```
const int P = 998244353;
int n;
i64 s[N];
map<i64, i64> cnt;
```

```
i64 dp[N];
void solve() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        i64 x;
        cin >> x;
        s[i] = s[i - 1] + x;
    }
    for(int i = 1; i < n; i++) {
        if(cnt.count(s[i])) {
            dp[i] = (dp[i - 1] * 2 - cnt[s[i]] + P) % P;
        }
        else {
            dp[i] = (dp[i - 1] * 2 + 1) % P;
        }
        cnt[s[i]] = dp[i - 1];
    }
    cout << (dp[n - 1] + 1) % P;//该题中 s[n] 必须要选择，遇到其他题时需注意
}
```

其他

树的拓扑序计数

有根树：$n! \times \prod_1^n \frac{1}{sz[i]}$

无根树：换根dp处理有根树的式子

切比雪夫距离转换曼哈顿距离

```
int n, x[N], y[N];
vector<array<int, 2>> v1, v2; //v1存x和y的和为偶数的点，v2存为奇数的点
i64 ans = 0;
void cal(auto a) {
    vector<ll> x, y;
    for(auto [u, v] : a) {
        x.push_back(u + v);
        y.push_back(u - v);
    }
    sort(x.begin(), x.end());
    sort(y.begin(), y.end());
    i64 s = 0;
    for(int i = 0; i < x.size(); i++) {
        ans += (x[i] * i) - s;
        s += x[i];
    }
    s = 0;
    for(int i = 0; i < y.size(); i++) {
        ans += (y[i] * i) - s;
        s += y[i];
    }
}
void solve() {
```

```
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> x[i] >> y[i];
        if((x[i] + y[i]) % 2 == 0) v1.push_back({x[i], y[i]});
        else v2.push_back({x[i], y[i]});
    }
    cal(v1), cal(v2);
    cout << ans / 2;
}
```

杂项

随机数

```
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());//使用64位时，存储要用unsigned long long
mt19937 rng(time(0));
```

unordered_map 防 TLE

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
unordered_map <int, int, custom_hash> mx;
```

初始代码模板

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

using namespace std;
using i64 = long long;

constexpr int N = 2e5 + 10;

void solve() {

}

int main() {
```

```
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t;
    t = 1;

    while (t--) {
        solve();
    }

    return 0;
}
```

int128读入输出

```
inline __int128 read(){
    __int128 x = 0, f = 1;
    char ch = getchar();
    while(ch < '0' || ch > '9'){
        if(ch == '-')
            f = -1;
        ch = getchar();
    }
    while(ch >= '0' && ch <= '9'){
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
inline void print(__int128 x){
    if(x < 0){
        putchar('-');
        x = -x;
    }
    if(x > 9)
        print(x / 10);
    putchar(x % 10 + '0');
}
```

取余模板

```
constexpr int P = 998244353;
using i64 = long long;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
```

```
T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    Z(i64 x) : x(norm(x % P)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
```

```
    friend istream &operator>>(istream &is, Z &a) {
        i64 v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend ostream &operator<<(ostream &os, const Z &a) {
        return os << a.val();
    }
};
```

lambda函数

```
实现dfs:
function<void(int, int, int)> dfs = [&](int x, int y, int z) {

};
其他:
auto fun = [&](int x, int y, int z) {

};
sort(a.begin(), a.end(), [&](int x, int y) {
   return ***;
});
```

开O2

```
#pragma GCC optimize(2)
```

windows系统下的对拍

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

using namespace std;
using i64 = long long;

constexpr int N = 2e5 + 10;

void solve() {
    for(int i = 1; i <= 100000; i++) {
        system("datamaker.exe > data.txt");
        system("A.exe < data.txt > A.txt");
        double start = clock();
        system("test.exe < data.txt > test.txt");
        double end = clock();
        if(system("fc A.txt test.txt")) {cout << "WA\n"; break;}
        else cout << "AC TIME : " << end - start << '\n';
    }
}

int main() {
    int t = 1;

    while(t--) {
```

```
        solve();
    }

    return 0;
}
```

linux系统下的对拍

需要在两个代码中读写文件，格式：

```
freopen("data.in","r",stdin);
freopen("std.out","w",stdout);
#include<bits/stdc++.h>
using namespace std;
int main() {
    for (int i = 1; i <= 100000; i++){
        printf("The result of No. %d Case is:  ", i);
        system("./datamaker");
        system("./std");
        system("./test");
        if (system("diff std.out test.out")) {
            printf("Wrong Answer\n");
            break;
        }
        else printf("Accepted\n");
    }
    return 0;
}
```

codeblocks终端修改

由于codeblocks默认的终端字体不易观看，更改步骤如下：

- 打开codeblocks
- Settings -> Environment
- 将Terminal to launch console programs栏中改为：
  gnome-terminal --geometry 80x20+100+100 --hide-menubar -t $TITLE -x

原根表

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

```
167772161           5   25  3
469762049           7   26  3
1004535809          479 21  3
2013265921          15  27  31
2281701377          17  27  3
3221225473          3   30  5
75161927681         35  31  3
77309411329         9   33  7
206158430209        3   36  22
2061584302081       15  37  7
2748779069441       5   39  3
6597069766657       3   41  5
39582418599937      9   42  5
79164837199873      9   43  5
263882790666241     15  44  7
1231453023109121    35  45  3
1337006139375617    19  46  3
3799912185593857    27  47  5
4222124650659841    15  48  19
7881299347898369    7   50  6
31525197391593473   7   52  3
180143985094819841  5   55  6
1945555039024054273 27  56  5
4179340454199820289 29  57  3
```

C++ STL简介

```
vector，变长数组，倍增的思想
    size()  返回元素个数
    empty()  返回是否为空
    clear()  清空
    front()/back()
    push_back()/pop_back()
    begin()/end()
    []
    支持比较运算，按字典序
pair<int, int>
    first，第一个元素
    second，第二个元素
    支持比较运算，以first为第一关键字，以second为第二关键字（字典序）
string，字符串
    size()/length()  返回字符串长度
    empty()
    clear()
    substr(起始下标，(子串长度))  返回子串
    c_str()  返回字符串所在字符数组的起始地址
queue，队列
    size()
    empty()
    push()  向队尾插入一个元素
    front()  返回队头元素
    back()  返回队尾元素
    pop()  弹出队头元素
priority_queue，优先队列，默认是大根堆
    size()
```

```
    empty()
    push()  插入一个元素
    top()  返回堆顶元素
    pop()  弹出堆顶元素
    定义成小根堆的方式：priority_queue<int, vector<int>, greater<int>> q;
stack, 栈
    size()
    empty()
    push()  向栈顶插入一个元素
    top()  返回栈顶元素
    pop()  弹出栈顶元素
deque, 双端队列
    size()
    empty()
    clear()
    front()/back()
    push_back()/pop_back()
    push_front()/pop_front()
    begin()/end()
    []
set, map, multiset, multimap, 基于平衡二叉树（红黑树），动态维护有序序列
    size()
    empty()
    clear()
    begin()/end()
    ++, -- 返回前驱和后继，时间复杂度 O(logn)
    set/multiset
        insert()  插入一个数
        find()  查找一个数
        count()  返回某一个数的个数
        erase()
            (1) 输入是一个数x，删除所有x   O(k + logn)
            (2) 输入一个迭代器，删除这个迭代器
        lower_bound()/upper_bound()
            lower_bound(x)  返回大于等于x的最小的数的迭代器
            upper_bound(x)  返回大于x的最小的数的迭代器
    map/multimap
        insert()  插入的数是一个pair
        erase()  输入的参数是pair或者迭代器
        find()
        []  注意multimap不支持此操作。 时间复杂度是 O(logn)
        lower_bound()/upper_bound()
unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
    和上面类似，增删改查的时间复杂度是 O(1)
    不支持 lower_bound()/upper_bound()， 迭代器的++, --
bitset, 压位
    bitset<10000> s;
    ~, &, |, ^
    >>, <<
    ==, !=
    []
    count()  返回有多少个1
    any()  判断是否至少有一个1
    none()  判断是否全为0
    set()  把所有位置成1
    set(k, v)  将第k位变成v
    reset()  把所有位变成0
    flip()  等价于~
```

`flip(k)` 把第k位取反

$n \leq$	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
$\max\{\omega(n)\}$	2	3	4	5	6	7	8	8	9
$\max\{d(n)\}$	4	12	32	64	128	240	448	768	1344
$n \leq$	10^{10}	10^{11}	10^{12}	10^{13}	10^{14}	10^{15}	10^{16}	10^{17}	10^{18}
$\max\{\omega(n)\}$	10	10	11	12	12	13	13	14	15
$\max\{d(n)\}$	2304	4032	6720	10752	17280	26880	41472	64512	103680