ACM-Template

This is tllwtg's ACM Templates.

Base-Algorithm	DP	DS	Game	Geometry	Graph	Math	Others	Snippets	String

Full List

- Aho-Corasick_Automaton, Anti-SG, Augmenting_Path_Algorithm
- Bellman_Ford, Biconnected_Component, Big_Integer, Binary_Indexed_Tree, Binary_Search
- Calculation_2D, Calculation_3D, Cantor_Expansion, Chinese_Remainder_Theorem, Classical_Game, Combination
- Data, Difference, Dijkstra, Dominator_Tree, DP_Digit, DP_Knapsack, DP_Range, DP_Tree, DSU
- Euler's_Totient_Function, Every-SG, Exgcd
- Fast_IO, Floyd, Functions
- Gauss_Jordan_Elimination, GCD, Gospers_Hack, Graph
- Hash, Heap, Heavy_Path_Decomposition, Heuristic_Merge, Hungarian_Algorithm
- Johnson
- k-D_Tree, KMP, Kosaraju, Kruskal
- LCA, LCM, Lucas
- Matching, MaxFlow, Mo's_Algorithm, ModInt, Modular_Multiplicative_Inverse, Modulo_For_Rational_Numbers, Monotone_Queue, Monotone_Stack, Multi-SG
- n_D_Prefix_Sum, Numbers
- Persistent_DS, Pigeonhole_Principle, Pollard_Rho, Poly, Prim, Prime
- Quick_Pow
- Rotating_Calipers
- Search, Segment_Tree, Segment_Tree_Beats, Segment_Tree_EX, SG, Snippets_C, Snippets_Cpp, Snippets_Cpp_win, Snippets_others, Sort, Sparse_Table, Splay, Sweep_Line
- Tarjan, Ternary_Search, Topology_Sort, Tricks, Trie

LICENSE

> some of the code originates from the internet.

Table of Content

1. Base-Algorithm

Binary_Search

1. `[left, right)` version

```
int binary_search1(int l, int r, int key)
{
    ++r;
    int res = -1, mid;
    while (l < r)
    {
        mid = l + ((r - l) >> 1);
```

```
        if (arr[mid] < key)
            l = mid + 1;
        else if (arr[mid] > key)
            r = mid;
        else
        {
            res = mid;
            break; // r = mid or l = mid + 1
        }
    }
    return res;
}

int binary_ans1(int l, int r)
{
    ++r;
    int res = -1, mid;
    while (l < r)
    {
        mid = l + ((r - l) >> 1);
        if (check(arr[mid]))
        {
            res = mid;
            l = mid + 1;
        }
        else
            r = mid;
    }
    return res;
}
```

2. `[left, right]` version

```
int binary_search2(int l, int r, int key)
{
    int res = -1, mid;
    while (l <= r)
    {
        mid = l + ((r - l) >> 1);
        if (arr[mid] < key)
            l = mid + 1;
        else if (arr[mid] > key)
            r = mid - 1;
        else
        {
            res = mid;
            break; // r = mid - 1 or l = mid + 1
        }
    }
    return res;
}

int binary_ans2(int l, int r)
{
    int res = -1, mid;
```

```
    while (l <= r)
    {
        mid = l + ((r - l) >> 1);
        if (check(arr[mid]))
        {
            res = mid;
            l = mid + 1;
        }
        else
            r = mid - 1;
    }
    return res;
}
```

Difference

1. 1d

```
int diff_1[MAXN];

void insert_1(int l, int r, int val)
{
    diff_1[l] += val;
    diff_1[r + 1] -= val;
}

void build_1(int n)
{
    int cur = 0;
    for (int i = 1; i <= n; ++i)
    {
        cur += diff_1[i];
        arr[i] = cur;
    }
}
```

2. 2d

```
// 2d -> 1d

int diff_2[MAXN][MAXN];

void insert_2_v1(int x1, int y1, int x2, int y2, int val)
{
    for (int i = x1; i <= x2; ++i)
    {
        diff_2[i][y1] += val;
        diff_2[i][y2 + 1] -= val;
    }
}

void build_2_v1(int n, int m)
{
    for (int i = 1; i <= n; ++i)
```

```
    {
        int cur = 0;
        for (int j = 1; j <= m; ++j)
        {
            cur += diff_2[i][j];
            mat[i][j] = cur;
        }
    }
}

// 2d

void insert_2_v2(int x1, int y1, int x2, int y2, int val)
{
    diff_2[x1][y1] += val;
    diff_2[x2 + 1][y2 + 1] += val;
    diff_2[x1][y2 + 1] -= val;
    diff_2[x2 + 1][y1] -= val;
}

void build_2_v2(int n, int m)
{
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            mat[i][j] = diff_2[i][j] + mat[i - 1][j] + mat[i][j - 1] - mat[i -
1][j - 1];
}
```

3. 对角线差分

`rl[x + y], lr[x - y + n]`

4. 差分用于区间去重

```
// 对不符合的区间加上 1，则符合的区间累加后一定为 0，由此可以进行计数
int lr[MAXN][MAXN];
for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= n; ++j)
    {
        lr[i][j] += lr[i][j - 1];
        if (!lr[i][j] && i <= j)
            ++ans;
    }
```

Quick_Pow

```
ll qpow(ll a, ll t, ll mod)
{
    a %= mod;
    ll res = 1;
    while (t)
    {
        if (t & 1)
            res = res * a % mod;
        a = a * a % mod;
```

```
        t >>= 1;
    }
    return res;
}

// if mod is a prime number
// t %= (mod - 1);
// a^(t^tt) == a^(t^tt % (mod - 1))

// 矩阵快速幂
namespace MatPow
{
    const ll N = 105, MOD = 1e9 + 7;
    int n = 10;

    struct Mat
    {
        ll v[N][N];
    };

    Mat operator *(const Mat &x, const Mat &y)
    {
        Mat ans;
        ll tmp = 0;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
            {
                tmp = 0;
                for (int k = 0; k < n; ++k)
                    tmp += x.v[i][k] * y.v[k][j] % MOD;
                ans.v[i][j] = tmp % MOD;
            }
        return ans;
    }

    Mat qpow(Mat a, ll k)
    {
        Mat res;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                res.v[i][j] = 0;
        for (int i = 0; i < n; ++i)
            res.v[i][i] = 1;

        while(k)
        {
            if (k & 1)
                res = res * a;
            a = a * a;
            k >>= 1;
        }
        return res;
    }
}
```

Search

1. Meet in the middle(BFS version)

```
// (a). one queue and 2 different marks
//
// (b). two queues and two maps(marks)
//      each time search the smaller size part for 1 level
//      (more optimized)
```

2. IDS(iterative deepening depth-first search)

```
// for (int limit = 1; ; ++limit)
//      search
//      if found:
//          break;
```

3. A* search algorithm

```
//      each time search the smallest f(x) = g(x) + h(x)
//      with open set and closed set (?)
```

Sort

1. Quicksort

> Quicksort can find the k-th element

```
int arr[MAXN];

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

1. Merge sort

> Merge sort can count Inversions

```cpp
int arr[MAXN], brr[MAXN];

void msort(int b, int e)
{
    if (b == e)
        return;
    int mid = (b + e) / 2, i = b, j = mid + 1, k = b;
    msort(b, mid), msort(mid + 1, e);
    while (i <= mid && j <= e)
    {
        if (arr[i] <= arr[j])
            brr[k++] = arr[i++];
        else
            brr[k++] = arr[j++];
    }
    while (i <= mid)
        brr[k++] = arr[i++];
    while (j <= e)
        brr[k++] = arr[j++];
    for (int l = b; l <= e; ++l)
        arr[l] = brr[l];
}
```

Ternary_Search

```cpp
// Min Version

double f(double x) {}
int run(int x) {}

double ternary_search()
{
    const double eps = 1e-6;
    double l, r, mid, lmid, rmid;
    while (r - l > eps)
    {
        mid = (lmid + rmid) / 2;
        lmid = mid - eps;
        rmid = mid + eps;
        if (f(lmid) < f(rmid))
            r = mid; // inaccuracy but ok
        else
            l = mid;
    }
    return l; // ?
}

int ternary_ans()
{
    int l = 0, r = 1e8, mid;
    int ans = 1e9;
    while (l + 1 < r)
    {
        mid = (l + r) / 2;
```

```
        int a1 = run(mid - 1), a2 = run(mid + 1);
        if (a1 > a2)
            ans = min(ans, a1), l = mid;
        else
            ans = min(ans, a2), r = mid;
    }
    ans = min(ans, run(l));
    ans = min(ans, run(r));
    return ans;
}
```

n_D_Prefix_Sum

1. with Principle of inclusion-exclusion

```
// s[i]=a[i]+s[i-1]
// s[i][j]=a[i][j]+s[i-1][j]+s[i][j-1]-s[i-1][j-1]
// s[i][j][k]=a[i][j][k]+s[i-1][j][k]+s[i][j-1][k]+s[i][j][k-1]-s[i-1][j-1][k]
-s[i-1][j][k-1]-s[i][j-1][j-1]+s[i-1][j-1][k-1]
// ......
```

2. DP

```
void prefix_2D()
{
    int a[MAXN][MAXN], n, m;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            a[i][j] = a[i][j] + a[i][j - 1];
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            a[i][j] = a[i][j] + a[i - 1][j];
}

void prefix_3D()
{
    int a[MAXN][MAXN][MAXN], n, m, p;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            for (int k = 1; k <= p; ++k)
                a[i][j][k] += a[i - 1][j][k];
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            for (int k = 1; k <= p; ++k)
                a[i][j][k] += a[i][j - 1][k];
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            for (int k = 1; k <= p; ++k)
                a[i][j][k] += a[i][j][k - 1];
}
```

2. DP

DP_Digit

> 记忆化搜索或循环迭代递推或（人类智慧）
>
> 求 [l, r] 答案时，可考虑 [0, r] - [0, l - 1]
>
> 数字间无关系时，可考虑每个数字分别 dp 一次
>
> 考虑贡献
>
> 可拓展到其他进制

1. 给定正整数 n，返回在 [1, n] 范围内无重复数字的正整数的个数。

```
ll digitDP(ll n)
{
    string s = to_string(n);
    int len = s.length();
    vector<vector<ll>> dp(len, vector<ll>(1 << 10, -1));
    // is_limit 是否受 n 限制，is_num 前面是否已经填了数字
    function<ll(int, int, bool, bool)> f = [&](int cur, int mask, bool is_limit,
bool is_num) -> ll
    {
        if (cur == len)
            return is_num; // is_num 为 true 表示得到了一个合法数字
        if (!is_limit && is_num && dp[cur][mask] != -1)
            return dp[cur][mask];
        ll res = 0;
        if (!is_num) // 可以跳过当前数位
            res += f(cur + 1, mask, false, false);
        int up = is_limit ? s[cur] - '0' : 9;  // 如果前面填的数字都和 n 的一样，那么
这一位至多填数字 s[cur]
        for (int d = 1 - is_num; d <= up; ++d) // 枚举要填入的数字 d
            if ((mask >> d & 1) == 0)          // d 不在 mask 中
                res += f(cur + 1, mask | (1 << d), is_limit && d == up, true);
        if (!is_limit && is_num)
            dp[cur][mask] = res;
        return res;
    };
    return f(0, 0, true, false);
}
```

2. 求 [a,b] 范围的所有整数中，每个数码(digit)各出现了多少次

```
namespace count
{
    vector<ll> digitDP(ll n)
    {
        string s = to_string(n);
        int len = s.length();
        vector<vector<ll>> dp(len, vector<ll>(14, -1));
        vector<ll> ans(10);
        function<ll(int, bool, bool, int, int)> f = [&](int cur, bool is_limit,
bool is_num, int d, int sumi) -> ll
        {
            if (cur == len)
```

```
                return sumi;
            if (!is_limit && is_num && dp[cur][sumi] != -1)
                return dp[cur][sumi];
            ll res = 0;
            if (!is_num)
                res += f(cur + 1, false, false, d, sumi);
            int up = is_limit ? s[cur] - '0' : 9;
            for (int dd = 1 - is_num; dd <= up; ++dd)
                res += f(cur + 1, is_limit && dd == up, true, d, sumi + (dd ==
d));
            if (!is_limit && is_num)
                dp[cur][sumi] = res;
            return res;
        };
        for (int i = 0; i < 10; ++i)
        {
            dp.assign(len, vector<ll>(14, -1));
            ans[i] = f(0, true, false, i, 0);
        }
        return ans;
    }

    int main()
    {
        // clock_t st = clock(), ed;
        ios::sync_with_stdio(0);
        cin.tie(0);
        // cout << setprecision(15) << fixed;
        ll a, b;
        cin >> a >> b;
        vector<ll> aa(digitDP(a - 1));
        vector<ll> bb(digitDP(b));
        for (int i = 0; i < 10; ++i)
            cout << bb[i] - aa[i] << " ";
        cout << endl;
        // ed = clock();
        // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
        // cout << "Total time: " << endtime << endl;
        return 0;
    }
}
```

DP_Knapsack

```
// 1. 01

void dp1(int n)
{
    for (int i = 1; i <= n; ++i)
        for (int j = W; j >= w[i]; --j)
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
}

// 2. inf
```

```cpp
void dp2(int n)
{
    for (int i = 1; i <= n; ++i)
        for (int j = w[i]; j <= W; ++j)
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
}

// 3. n kinds but k things

void dp3(int m)
{
    // preprocess
    int index = 0;
    for (int i = 1; i <= m; i++)
    {
        int c = 1, p, h, k;
        cin >> p >> h >> k;
        while (k > c)
        {
            k -= c;
            w[++index] = c * p;
            v[index] = c * h;
            c *= 2;
        }
        w[++index] = p * k;
        v[index] = h * k;
    }
    for (int i = 1; i <= index; ++i)
        for (int j = W; j >= w[i]; --j)
            dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
}

// 4. combined
// if ... else if ... else ...

// 5. 2 dimensions

int dpdp[MAXN][MAXN], c[MAXN], C;

void dp5(int n)
{
    for (int i = 1; i <= n; ++i)
        for (int j = W; j >= w[i]; --j)
            for (int k = C; k >= c[i]; --k)
                dpdp[j][k] = max(dpdp[j][k], dpdp[j - w[i]][k - c[i]] + w[i]);
}

// 6. groups

int t[MAXN][MAXN], cnt[MAXN], tsz;

void dp6(int n)
{
    for (int k = 1; k <= tsz; ++k)
        for (int j = W; j >= 0; --j)
```

```
            for (int i = 1; i <= cnt[k]; ++i)
                if (j >= w[t[k][j]])
                    dp[j] = max(dp[j], dp[j - w[t[k][i]]] + v[t[k][i]]);
}

// record for 01
// int g[MAXN][MAXN] 0 for fi, 1 for se
// int w = W;
// loop from last to first
// if choose, w -= w[i]; and record  that

// 可行性 01 背包问题可以用 bitset 优化
// bitset<MAXN> dp;
// dp[0] = 1;
// dp |= dp << v[i];
// if (dp.test[v[i]]) ...
// example: v[1] = 2, v[2] = 3
// 00000001
// 00000101
// 00101101
```

DP_Range

```
// luogu P1880 (max)

int dp(int n)
{
    vector<int> arr(2 * n + 1), sumi(2 * n + 1, 0);
    vector<vector<int>> dp(2 * n + 1, vector<int>(2 * n + 1, 0));
    for (int i = 1; i <= n; ++i)
        cin >> arr[i], arr[i + n] = arr[i];
    for (int i = 1; i <= 2 * n; ++i)
        sumi[i] = sumi[i - 1] + arr[i];
    for (int len = 1; len <= n; ++len)
        for (int i = 1, j = len + 1; j < 2 * n; ++i, ++j)
            for (int k = i; k < j; ++k)
                dp[i][j] = max(dp[i][j], dp[i][k] + dp[k + 1][j] + sumi[j] -
sumi[i - 1]);
    int ans = 0;
    for (int i = 1, j = n; j < 2 * n; ++i, ++j)
        ans = max(ans, dp[i][j]);
    return ans;
}
```

DP_Tree

1. luogu P1352

```
int dp[10005][2], isnhead[10005], vis[10005];

vector<vector<int>> v(10005);

void dfs(int x)
{
```

```
    vis[x] = 1;
    for (auto &vv : v[x])
    {
        if (vis[vv])
            continue;
        dfs(vv);
        dp[x][1] += dp[vv][0];
        dp[x][0] += max(dp[vv][0], dp[vv][1]);
    }
}

void dp_tree()
{
    int n, x, y;
    cin >> n;
    for (int i = 1; i <= n; ++i)
        cin >> dp[i][1];
    for (int i = 1; i <= n - 1; ++i)
    {
        cin >> y >> x;
        v[x].pb(y);
        isnhead[y] = 1;
    }
    for (int i = 1; i <= n; ++i)
    {
        if (!isnhead[i])
        {
            dfs(i);
            cout << max(dp[i][0], dp[i][1]) << endl;
            break;
        }
    }
}
```

2. P3478 preprocess tree dp

```
vector<vector<ll>> v2(1e6 + 5);

vector<ll> dp2(1e6 + 5, 0), sz2(1e6 + 5, 0), dep2(1e6 + 5, 0);
ll n, x, y;
void dfs1(int u, int fa)
{
    dep2[u] = dep2[fa] + 1;
    sz2[u] = 1;
    for (auto &x : v[u])
    {
        if (x == fa)
            continue;
        dfs1(x, u);
        sz2[u] += sz2[x];
    }
}

void dfs2(int u, int fa)
{
```

```
    for (auto &x : v[u])
    {
        if (x == fa)
            continue;
        dp2[x] = dp2[u] + n - 2 * sz2[x];
        dfs2(x, u);
    }
}

void dp_tree2()
{
    cin >> n;
    for (int i = 1; i <= n - 1; ++i)
    {
        cin >> x >> y;
        v[x].pb(y);
        v[y].pb(x);
    }
    dfs1(1, 1);
    ll ans = -1e9, id = -1;
    for (int i = 1; i <= n; ++i)
        dp2[1] += dep2[i];
    dfs2(1, 1);

    for (int i = 1; i <= n; ++i)
        if (dp2[i] > ans)
        {
            ans = dp2[i];
            id = i;
        }
    cout << id << endl;
}
```

3. CF 1822F(max distance)

```
vector<vector<int>> e(2e5 + 5);
vector<ll> dep(2e5 + 5, 0), down(2e5 + 5), up(2e5 + 5);

ll n;

void predfs(int u, int fa)
{
    dep[u] = dep[fa] + 1;
    down[u] = up[u] = 0;
    for (auto &x : e[u])
    {
        if (x == fa)
            continue;
        predfs(x, u);
        down[u] = max(down[u], down[x] + 1);
    }
}

void dfs(int u, int fa)
{
```

```
    ll max1 = -n - 1, max2 = -n - 1;
    for (auto &x : e[u])
    {
        if (x == fa)
            continue;
        if (down[x] > max1)
        {
            max2 = max1;
            max1 = down[x];
        }
        else
        {
            max2 = max(max2, down[x]);
        }
    }
    for (auto &x : e[u])
    {
        if (x == fa)
            continue;
        up[x] = max(up[u] + 1, (down[x] == max1 ? (max2 + 2) : (max1 + 2)));
        dfs(x, u);
    }
}

int main()
{
    // clock_t st = clock(), ed;
    ios::sync_with_stdio(0);
    cin.tie(0);
    // cout << setprecision(15) << fixed;
    ll T, n, k, c, u, v;
    dep[0] = -1;
    cin >> T;
    while (T--)
    {
        cin >> n >> k >> c;
        for (int i = 1; i < n; ++i)
        {
            cin >> u >> v;
            e[u].pb(v);
            e[v].pb(u);
        }
        predfs(1, 0);
        ll ans = -1;
        dfs(1, 0);
        for (int i = 1; i <= n; ++i)
            ans = max(ans, k * max(down[i], up[i]) - c * dep[i]);
        cout << ans << endl;
        for (int i = 1; i <= n; i++)
            e[i].clear();
    }
    // ed = clock();
    // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
    // cout << "Total time: " << endtime << endl;
    return 0;
```

```
}
```

4. 直径

```
int ans = 0;
vector<vector<int>> to(n + 1);
vector<int> d1(n + 1), d2(n + 1);

function<void(int u, int fa)> dfs_1 = [&](int u, int fa)
{
    d1[u] = d2[u] = 0;
    for (auto v: to[u])
    {
        if (v == fa)
            continue;
        int t = d1[v] + 1;
        if (t > d1[u])
            d2[u] = d1[u], d1[u] = t;
        else if (t > d2[u])
            d2[u] = t;
    }
    ans = max(ans, d1[u] + d2[u]);
};
```

3. DS

Binary_Indexed_Tree

> prefix sum version
>
> also able to update range with difference

```
class biTree
{
private:
    int MAXN;
    vector<ll> tree;
    ll lowbit(ll x)
    {
        return x & (-x);
    }

public:
    biTree(int _MAXN = 100005) : MAXN(_MAXN)
    {
        tree.resize(MAXN);
    }
    void update(ll index, ll x)
    {
        for (ll pos = index; pos < MAXN; pos += lowbit(pos))
            tree[pos] += x;
    }
```

```
    ll query(ll n)
    {
        ll sum = 0;
        for (ll i = n; i; i -= lowbit(i))
            sum += tree[i];
        return sum;
    }
    // (a, b]
    ll query(ll a, ll b)
    {
        return query(b) - query(a);
    }
};
```

DSU

1. 普通并查集

```
class DSU
{
private:
    int n;
    vector<int> fa, sz;

public:
    DSU(int _n) : n(_n)
    {
        fa.resize(n + 1);
        sz.resize(n + 1);
        for (int i = 1; i <= n; ++i)
        {
            fa[i] = i;
            sz[i] = 1;
        }
    }
    int find(int x)
    {
        if (fa[x] == x)
            return x;
        else
        {
            fa[x] = find(fa[x]);
            return fa[x];
        }
    }
    void merge(int x, int y)
    {
        int xfa = find(x), yfa = find(y);
        if (xfa == yfa)
            return;
        if (sz[xfa] <= sz[yfa])
        {
            sz[yfa] += sz[xfa];
            fa[xfa] = yfa;
        }
```

```
        else
        {
            sz[xfa] += sz[yfa];
            fa[yfa] = xfa;
        }
    }
    int get_size(int x)
    {
        return sz[find(x)];
    }
};
```

2. 带权并查集（以点到根的距离为例）

```
// 1. 路径压缩
int FindSet(int x) {
    if (x == parent[x])
        return x;
    else {
        int t = parent[x]; //记录原父节点编号
        parent[x] = FindSet(parent[x]); //父节点变为根节点，此时value[x]=父节点到根节点的权值
        value[x] += value[t]; //当前节点的权值加上原本父节点的权值
         return parent[x]
    }
}

// 2. 一种合并方式（依题目变化）
//
// 已知 x，y 根节点分别为 xRoot，yRoot，如果有了 x、y 之间的关系，
// 得求出 xRoot 与 yRoot 这条边的权值是多少，很显然 x 到 yRoot 两条路径的权值之和应该相同
//  int xRoot = FindSet(x);
//  int yRoot = FindSet(y);
//  if (xRoot != yRoot)
//  {
//      parent[xRoot] = yRoot;
//      value[xRoot] = -value[x] + value[y] + s;
//  }
```

3. 种类并查集（以食物链为例）

> 我们可以用一个三倍大小的并查集进行维护，用 i+n 表示 i 的捕食对象，而 i+2n 表示 i 的天敌

```
int main()
{
    int n = read(), m = read(), ans = 0;
    init(n * 3); //i吃i+n，被i+2n吃
    for (int i = 0; i < m; ++i)
    {
        int opr, x, y;
        scanf("%d%d%d", &opr, &x, &y);
        if (x > n || y > n) //特判x或y不在食物链中的情况
            ans++;
        else if (opr == 1)
        {
```

```
            if (query(x, y + n) || query(x, y + 2 * n)) //如果已知x吃y，或者x被y吃，说明这是假话
                ans++;
            else
            {
                merge(x, y);                  //这是真话，则x和y是一族
                merge(x + n, y + n);          //x的猎物和y的猎物是一族
                merge(x + 2 * n, y + 2 * n); //x的天敌和y的天敌是一族
            }
        }
        else if (opr == 2)
        {
            if (query(x, y) || query(x, y + 2 * n)) //如果已知x与y是一族，或者x被y吃，说明这是假话
                ans++;
            else
            {
                merge(x, y + n);          //这是真话，则x吃y
                merge(x + n, y + 2 * n); //x的猎物吃y的猎物
                merge(x + 2 * n, y);      //x的天敌吃y的天敌，或者说y吃x的天敌
            }
        }
    }
    printf("%d\n", ans);
    return 0;
}
```

Dominator_Tree

1. version 1

> index start from 0
>
> must be connected gragh

```
class DomTree
{
public:
    vector<vector<int>> dom;
    vector<int> idom, sz;
    void Build(const vector<vector<int>> &G, int entry)
    {
        int n = G.size(), cnt = 0;
        vector<vector<int>> RG(n), S(n);
        for (int u = 0; u < n; ++u)
        {
            for (auto v : G[u])
                RG[v].pb(u);
        }
        idom.assign(n, 0), dom.assign(n, {}), sz.assign(n, 0);
        vector<int> dfn(n, -1), rev(n), par(n);
        function<void(int)> dfs = [&](int x)
        {
            rev[dfn[x] = cnt++] = x;
            for (int y : G[x])
```

```
        {
            if (~dfn[y])
                continue;
            par[y] = x, dfs(y);
        }
    };
    dfs(entry);
    vector<int> semi(n);
    iota(semi.begin(), semi.end(), 0);
    vector<int> mn(semi), f(semi);
    function<int(int)> merge = [&](int x)
    {
        if (x == f[x])
            return x;
        int t = f[x];
        f[x] = merge(f[x]);
        if (dfn[semi[mn[t]]] < dfn[semi[mn[x]]])
            mn[x] = mn[t];
        return f[x];
    };
    for (int i = n - 1; i; --i)
    {
        int tmp = n, u = rev[i];
        for (int j : RG[rev[i]])
        {
            if (dfn[j] < i)
                tmp = min(tmp, dfn[j]);
            else
            {
                merge(j);
                tmp = min(tmp, dfn[semi[mn[j]]]);
            }
        }
        semi[u] = rev[tmp];
        f[u] = par[u];
        S[semi[u]].pb(u);
        for (int j : S[rev[i - 1]])
        {
            merge(j);
            if (semi[mn[j]] == semi[j])
                idom[j] = semi[j];
            else
                idom[j] = mn[j];
        }
    }
    for (auto u : rev)
    {
        if (idom[u] != semi[u])
            idom[u] = idom[idom[u]];
    }
    for (int i = 0; i < n; ++i)
    {
        if (i == entry)
            continue;
        dom[idom[i]].pb(i);
```

```
        }
        function<void(int, int)> getsize = [&](int u, int fa)
        {
            sz[u] = 1;
            for (int v : dom[u])
            {
                if (v == fa)
                    continue;
                getsize(v, u);
                sz[u] += sz[v];
            }
        };
        getsize(entry, 0);
    }
};
```

2. version 2

> only for DAG
>
> index start from 1
>
> work for forest

```
namespace DomTree_dag
{
    const int MAX = 65536;
    int n, tot;
    int d[MAX], w[MAX], sz[MAX], p[MAX], f[MAX][17];
    vector<int> e[MAX], g[MAX], h[MAX];
    stack<int> s;
    void topo()
    {
        s.push(0);
        for (int i = 1; i <= n; ++i)
        {
            if (!w[i])
            {
                e[0].pb(i);
                g[i].pb(0);
                ++w[i];
            }
        }
        while (!s.empty())
        {
            int x = s.top();
            s.pop();
            p[++tot] = x;
            for (int i : e[x])
            {
                --w[i];
                if (!w[i])
                {
                    s.push(i);
                }
            }
```

```
        }
    }
    int lca(int u, int v)
    {
        if (d[u] < d[v])
        {
            swap(u, v);
        }
        for (int i = 15; i >= 0; --i)
        {
            if (d[f[u][i]] >= d[v])
            {
                u = f[u][i];
            }
        }
        if (u == v)
        {
            return u;
        }
        for (int i = 15; i >= 0; --i)
        {
            if (f[u][i] != f[v][i])
            {
                u = f[u][i];
                v = f[v][i];
            }
        }
        return f[u][0];
    }
    void dfs(int x)
    {
        sz[x] = 1;
        for (int i : h[x])
        {
            dfs(i);
            sz[x] += sz[i];
        }
    }
    void build()
    {
        for (int i = 2; i <= n + 1; ++i)
        {
            int x = p[i], y = g[x][0];
            for (int j = 1, q = g[x].size(); j < q; ++j)
            {
                y = lca(y, g[x][j]);
            }
            h[y].pb(x);
            d[x] = d[y] + 1;
            f[x][0] = y;
            for (int i = 1; i <= 15; ++i)
            {
                f[x][i] = f[f[x][i - 1]][i - 1];
            }
        }
```

```
    }
    // example

    int main()
    {
        // clock_t st = clock(), ed;
        ios::sync_with_stdio(0);
        cin.tie(0);
        // cout << setprecision(15) << fixed;
        int n, x;
        cin >> n;
        DomTree_dag::n = n;
        for (int i = 1; i <= n; ++i)
        {
            while (true)
            {
                cin >> x;
                if (!x)
                {
                    break;
                }
                DomTree_dag::e[x].pb(i);
                DomTree_dag::g[i].pb(x);
                ++DomTree_dag::w[i];
            }
        }
        DomTree_dag::topo();
        DomTree_dag::build();
        DomTree_dag::dfs(0);
        for (int i = 1; i <= n; ++i)
            cout << DomTree_dag::sz[i] - 1 << endl;
        // ed = clock();
        // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
        // cout << "Total time: " << endtime << endl;
        return 0;
    }
}
```

Heap

1. 普通堆

```
class heap
{
private:
    vector<int> h;
    void up(int pos)
    {
        while (pos > 1)
        {
            if (h[pos >> 1] < h[pos])
            {
                swap(h[pos >> 1], h[pos]);
                pos /= 2;
            }
```

```
            else
                break;
        }
    }
    void down(int pos)
    {
        while (true)
        {
            int tpos = pos;
            int ls = pos * 2, rs = pos * 2 + 1;
            if (ls <= sz && h[ls] > h[tpos])
                tpos = ls;
            if (rs <= sz && h[rs] > h[tpos])
                tpos = rs;
            if (tpos == pos)
                break;
            swap(h[pos], h[tpos]);
            pos = tpos;
        }
    }

public:
    int sz;
    heap(int MAXN = 1000005) : sz(0)
    {
        h.resize(MAXN);
    }
    heap(const vector<int> &arr, int MAXN = 1000005)
    {
        h.resize(MAXN);
        sz = arr.size();
        for (int i = 0; i < sz; ++i)
            h[i + 1] = arr[i];
        for (int i = sz; i >= 1; --i)
            down(i);
    }
    void push(int x)
    {
        h[++sz] = x;
        up(sz);
    }
    void pop()
    {
        if (sz == 0)
            return;
        h[1] = h[sz--];
        down(1);
    }
    int top()
    {
        if (sz == 0)
            return 0;
        return h[1];
    }
};
```

2. 对顶堆（kth element）

```
class kth_heap
{
private:
    priority_queue<int, vector<int>, greater<int>> q1;
    priority_queue<int, vector<int>, less<int>> q2;

public:
    int sz, k;
    kth_heap(int x) : sz(0), k(x) {}
    bool isok()
    {
        return sz >= k;
    }
    void push(int x)
    {
        if (sz < k)
        {
            q1.push(x);
            ++sz;
            return;
        }
        ++sz;
        q2.push(x);
        while (q1.top() < q2.top())
        {
            int temp = q1.top();
            q1.pop();
            q2.push(temp);
            q1.push(q2.top());
            q2.pop();
        }
    }
    void pop()
    {
        if (sz == 0)
            return;
        --sz;
        q1.pop();
        if (q2.empty())
            return;
        q1.push(q2.top());
        q2.pop();
    }
    int top()
    {
        return q1.top();
    }
};
```

Monotone_Queue

```
int arr[MAXN], k;
deque<int> q;

void func()
{
    for (int i = 1; i <= MAXN; ++i)
    {
        if (!q.empty() && i - q.front() >= k)
            q.pop_front();
        while (!q.empty() && arr[q.back()] < arr[i])
            q.pop_back();
        q.push_back(i);
        if (i >= k)
            cout << q.front() << endl;
    }
}
```

Monotone_Stack

```
int arr[MAXN], ans[MAXN];
stack<int> st;

void func()
{
    for (int i = 1; i <= MAXN; ++i)
    {
        while (!st.empty() && arr[i] > arr[st.top()])
        {
            ans[st.top()] = i;
            st.pop();
        }
        st.push(i);
    }
}
```

Persistent_DS

1. persistent array

> partially persistent

```
namespace pa
{
    vector<pii> arr[100005];
    int get_item(int index, int time)
    {
        auto ub = upper_bound(all(arr[index]), make_pair(time, INT_MAX));
        return prev(ub)->se;
    }
    void update_item(int index, int value, int time)
    {
```

```
        assert(arr[index].back().fi < time);
        arr[index].pb({time, value});
    }
    void init_arr(int n, const vector<int> &init)
    {
        for (int i = 0; i < n; ++i)
            arr[i].pb({0, init[i]});
    }
}
```

2. persistent segment tree

```
namespace PST
{
#define ls(x) tree[x].ls
#define rs(x) tree[x].rs
#define val(x) tree[x].val
#define mark(x) tree[x].mark
    const int MAXV = 20000000, MAXN = 1000005;
    struct node
    {
        int val, ls, rs;
    } tree[MAXV];
    int A[MAXN], roots[MAXN], n, m, cnt = 1; // roots记录每个历史版本的根节点
    void build(int l = 1, int r = n, int p = 1)
    {
        if (l == r)
            val(p) = A[l];
        else
        {
            ls(p) = ++cnt, rs(p) = ++cnt;
            int mid = (l + r) / 2;
            build(l, mid, ls(p));
            build(mid + 1, r, rs(p));
            val(p) = val(ls(p)) + val(rs(p));
        }
    }
    void update(int x, int d, int p, int q, int cl = 1, int cr = n) // 单点修改
    {
        if (cl == cr)
            val(q) = d;
        else
        {
            ls(q) = ls(p), rs(q) = rs(p);
            int mid = (cl + cr) / 2;
            if (x <= mid)
                ls(q) = ++cnt, update(x, d, ls(p), ls(q), cl, mid);
            else
                rs(q) = ++cnt, update(x, d, rs(p), rs(q), mid + 1, cr);
            val(q) = val(ls(q)) + val(rs(q));
        }
    }
    int query(int l, int r, int p, int cl = 1, int cr = n) // 区间查询
    {
        if (cl > r || cr < l)
```

```
            return 0;
        else if (cl >= l && cr <= r)
            return val(p);
        else
        {
            int mid = (cl + cr) / 2;
            return query(l, r, ls(p), cl, mid) + query(l, r, rs(p), mid + 1,
cr);
        }
    }

    // example
    void solve()
    {
        cin >> n >> m;
        for (int i = 1; i <= n; ++i)
            cin >> A[i];
        build();
        roots[0] = 1;
        for (int t = 1; t <= m; ++t)
        {
            int v, o;
            cin >> v >> o;
            if (o == 1)
            {
                int x, d;
                cin >> x >> d;
                roots[t] = ++cnt; // 新建节点
                update(x, d, roots[v], roots[t]);
            }
            else
            {
                int x;
                cin >> x;
                roots[t] = roots[v]; // 复用v号版本
                cout << query(x, x, roots[v]) << endl;
            }
        }
    }
}
```

3. president tree

```
namespace PT
{
#define ls(x) (tr[x].l)
#define rs(x) (tr[x].r)
#define sum(x) tr[x].sum
    const int N = 2e5 + 5;
    const int M = 1e6 + 5;
    struct node
    {
        int sum = 0;
        int l = 0, r = 0;
    } tr[M * 30];
```

```
int tot = 1;
int root[N], a[N], n, m;
void pushup(int x)
{
    sum(x) = sum(ls(x)) + sum(rs(x));
}
void upd(int last, int now, int pos, int k, int l, int r)
{
    // 过去的节点 现在的节点 修改的位置，k ，当前节点表示的区间[l,r]
    if (l == r)
    {
        sum(now) = sum(last) + k;
    }
    else
    {
        ls(now) = ls(last), rs(now) = rs(last);
        int mid = (l + r - 1) / 2;
        if (pos <= mid)
            ls(now) = ++tot, upd(ls(last), ls(now), pos, k, l, mid);
        else
            rs(now) = ++tot, upd(rs(last), rs(now), pos, k, mid + 1, r);
        pushup(now);
    }
}
const int up = 1e9 + 5;
const int down = -(1e9 + 5);
void upd(int last, int now, int pos, int k)
{
    upd(last, now, pos, k, down, up);
}
int kth(int last, int now, int k, int l, int r)
{
    if (l == r)
        return l;
    int mid = (l + r - 1) / 2;
    int val = sum(ls(now)) - sum(ls(last));
    if (val >= k)
        return kth(ls(last), ls(now), k, l, mid);
    else
        return kth(rs(last), rs(now), k - val, mid + 1, r);
}
int kth(int last, int now, int k)
{
    return kth(last, now, k, down, up);
}

// example
void solve()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= n; i++)
    {
        root[i] = ++tot;
```

```
            upd(root[i - 1], root[i], a[i], 1);
        }
        while (m--)
        {
            int L, R, k;
            cin >> L >> R >> k;
            cout << kth(root[L - 1], root[R], k) << endl;
        }
    }
}
```

Segment_Tree

> if data is complex, use node struct to manage them

1. add and sum version

```
template <typename T>
class SegmentTree
{
private:
    int n;
    vector<T> tree, mark;
    void push_down(int p, int len)
    {
        mark[p * 2] += mark[p];
        mark[p * 2 + 1] += mark[p];
        tree[p * 2] += mark[p] * (len - len / 2);
        tree[p * 2 + 1] += mark[p] * (len / 2);
        mark[p] = 0;
    }

public:
    vector<T> arr;
    SegmentTree(int _n) : n(_n)
    {
        tree.resize(4 * n + 10, 0);
        mark.resize(4 * n + 10, 0);
        arr.resize(n + 10);
    }
    void build(int l, int r, int p = 1)
    {
        if (l == r)
            tree[p] = arr[l];
        else
        {
            int middle = (l + r) / 2;
            build(l, middle, 2 * p);
            build(middle + 1, r, 2 * p + 1);
            tree[p] = tree[2 * p] + tree[2 * p + 1];
        }
    }
    void update(int l, int r, int cl, int cr, T d, int p = 1)
    {
        if (cr < l || cl > r)
```

```
            return;
        else if (l <= cl && cr <= r)
        {
            tree[p] += d * (cr - cl + 1);
            if (cr > cl)
                mark[p] += d;
        }
        else
        {
            int middle = (cl + cr) / 2;
            push_down(p, cr - cl + 1);
            update(l, r, cl, middle, d, p * 2);
            update(l, r, middle + 1, cr, d, p * 2 + 1);
            tree[p] = tree[p * 2] + tree[p * 2 + 1];
        }
    }
    T query(int l, int r, int cl, int cr, int p = 1)
    {
        if (cl > r || cr < l)
            return 0;
        else if (cl >= l && cr <= r)
            return tree[p];
        else
        {
            int mid = (cl + cr) / 2;
            push_down(p, cr - cl + 1);
            return query(l, r, cl, mid, p * 2) + query(l, r, mid + 1, cr, p * 2
+ 1);
        }
    }
};
```

2. change and sum version

```
template <typename T>
class SegTreeLazyRangeSet
{
private:
    vector<T> tree, lazy;
    vector<T> *arr;
    int n, root, n4, end;
    void maintain(int cl, int cr, int p)
    {
        int cm = cl + (cr - cl) / 2;
        if (cl != cr && lazy[p])
        {
            lazy[p * 2] = lazy[p];
            lazy[p * 2 + 1] = lazy[p];
            tree[p * 2] = lazy[p] * (cm - cl + 1);
            tree[p * 2 + 1] = lazy[p] * (cr - cm);
            lazy[p] = 0;
        }
    }
    T range_sum(int l, int r, int cl, int cr, int p)
    {
```

```
        if (l <= cl && cr <= r)
            return tree[p];
        int m = cl + (cr - cl) / 2;
        T sum = 0;
        maintain(cl, cr, p);
        if (l <= m)
            sum += range_sum(l, r, cl, m, p * 2);
        if (r > m)
            sum += range_sum(l, r, m + 1, cr, p * 2 + 1);
        return sum;
    }
    void range_set(int l, int r, T val, int cl, int cr, int p)
    {
        if (l <= cl && cr <= r)
        {
            lazy[p] = val;
            tree[p] = (cr - cl + 1) * val;
            return;
        }
        int m = cl + (cr - cl) / 2;
        maintain(cl, cr, p);
        if (l <= m)
            range_set(l, r, val, cl, m, p * 2);
        if (r > m)
            range_set(l, r, val, m + 1, cr, p * 2 + 1);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
    }
    void build(int s, int t, int p)
    {
        if (s == t)
        {
            tree[p] = (*arr)[s];
            return;
        }
        int m = s + (t - s) / 2;
        build(s, m, p * 2);
        build(m + 1, t, p * 2 + 1);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
    }

public:
    explicit SegTreeLazyRangeSet<T>(vector<T> v)
    {
        n = v.size();
        n4 = n * 4;
        tree = vector<T>(n4, 0);
        lazy = vector<T>(n4, 0);
        arr = &v;
        end = n - 1;
        root = 1;
        build(0, end, 1);
        arr = nullptr;
    }
    void show(int p, int depth = 0)
    {
```

```
        if (p > n4 || tree[p] == 0)
            return;
        show(p * 2, depth + 1);
        for (int i = 0; i < depth; ++i)
            putchar('\t');
        printf("%d:%d\n", tree[p], lazy[p]);
        show(p * 2 + 1, depth + 1);
    }
    T range_sum(int l, int r) { return range_sum(l, r, 0, end, root); }
    void range_set(int l, int r, int val) { range_set(l, r, val, 0, end, root);
}
};
```

3. add, mul and sum version

```
template <typename T>
class SegmentTree_pro
{
private:
    int n, mod;
    vector<T> tree, mark_p, mark_m;
    void push_down(int p, int len)
    {
        mark_m[p * 2] = mark_m[p * 2] * mark_m[p] % mod;
        mark_m[p * 2 + 1] = mark_m[p * 2 + 1] * mark_m[p] % mod;
        mark_p[p * 2] = (mark_p[p * 2] * mark_m[p] % mod + mark_p[p]) % mod;
        mark_p[p * 2 + 1] = (mark_p[p * 2 + 1] * mark_m[p] % mod + mark_p[p]) %
mod;
        tree[p * 2] = (tree[p * 2] * mark_m[p] % mod + mark_p[p] * (len - len /
2) % mod) % mod;
        tree[p * 2 + 1] = (tree[p * 2 + 1] * mark_m[p] % mod + mark_p[p] * (len
/ 2) % mod) % mod;
        mark_p[p] = 0;
        mark_m[p] = 1;
    }

public:
    vector<T> arr;
    SegmentTree_pro(int _n, int _mod) : n(_n), mod(_mod)
    {
        tree.resize(4 * n + 10, 0);
        mark_p.resize(4 * n + 10, 0);
        mark_m.resize(4 * n + 10, 1);
        arr.resize(n + 10);
    }
    void build(int l, int r, int p = 1)
    {
        if (l == r)
            tree[p] = arr[l] % mod;
        else
        {
            int middle = (l + r) / 2;
            build(l, middle, 2 * p);
            build(middle + 1, r, 2 * p + 1);
            tree[p] = (tree[2 * p] + tree[2 * p + 1]) % mod;
```

```
        }
    }
    void update_add(int l, int r, int cl, int cr, T d, int p = 1)
    {
        if (cr < l || cl > r)
            return;
        else if (l <= cl && cr <= r)
        {
            tree[p] = (tree[p] + d * (cr - cl + 1)) % mod;
            if (cr > cl)
                mark_p[p] = (mark_p[p] + d) % mod;
        }
        else
        {
            int middle = (cl + cr) / 2;
            push_down(p, cr - cl + 1);
            update_add(l, r, cl, middle, d, p * 2);
            update_add(l, r, middle + 1, cr, d, p * 2 + 1);
            tree[p] = (tree[p * 2] + tree[p * 2 + 1]) % mod;
        }
    }
    void update_mul(int l, int r, int cl, int cr, T d, int p = 1)
    {
        if (cr < l || cl > r)
            return;
        else if (l <= cl && cr <= r)
        {
            tree[p] = tree[p] * d % mod;
            if (cr > cl)
            {
                mark_p[p] = mark_p[p] * d % mod;
                mark_m[p] = mark_m[p] * d % mod;
            }
        }
        else
        {
            T middle = (cl + cr) / 2;
            push_down(p, cr - cl + 1);
            update_mul(l, r, cl, middle, d, p * 2);
            update_mul(l, r, middle + 1, cr, d, p * 2 + 1);
            tree[p] = (tree[p * 2] + tree[p * 2 + 1]) % mod;
        }
    }
    T query(int l, int r, int cl, int cr, int p = 1)
    {
        if (cl > r || cr < l)
            return 0;
        else if (cl >= l && cr <= r)
            return tree[p];
        else
        {
            int mid = (cl + cr) / 2;
            push_down(p, cr - cl + 1);
            return (query(l, r, cl, mid, p * 2) + query(l, r, mid + 1, cr, p * 2
+ 1)) % mod;
```

```
        }
    }
};
```

4. add and max version (no sum)

```
template <typename T>
class SegmentTree_max
{
private:
    int n;
    vector<T> tree, mark;
    void push_down(int p)
    {
        mark[p * 2] += mark[p];
        mark[p * 2 + 1] += mark[p];
        tree[p * 2] += mark[p];
        tree[p * 2 + 1] += mark[p];
        mark[p] = 0;
    }

public:
    vector<T> arr;
    SegmentTree_max(int _n) : n(_n)
    {
        tree.resize(4 * n + 10, 0);
        mark.resize(4 * n + 10, 0);
        arr.resize(n + 10);
    }
    void build(int l, int r, int p = 1)
    {
        if (l == r)
            tree[p] = arr[l];
        else
        {
            int middle = (l + r) / 2;
            build(l, middle, 2 * p);
            build(middle + 1, r, 2 * p + 1);
            tree[p] = max(tree[2 * p], tree[2 * p + 1]);
        }
    }
    void update(int l, int r, int cl, int cr, T d, int p = 1)
    {
        if (cr < l || cl > r)
            return;
        else if (l <= cl && cr <= r)
        {
            tree[p] += d;
            if (cr > cl)
                mark[p] += d;
        }
        else
        {
            int middle = (cl + cr) / 2;
            push_down(p);
```

```
            update(l, r, cl, middle, d, p * 2);
            update(l, r, middle + 1, cr, d, p * 2 + 1);
            tree[p] = max(tree[p * 2], tree[p * 2 + 1]);
        }
    }
    T query(int l, int r, int cl, int cr, int p = 1)
    {
        if (cl > r || cr < l)
            return 0;
        else if (cl >= l && cr <= r)
            return tree[p];
        else
        {
            int mid = (cl + cr) / 2;
            push_down(p);
            return max(query(l, r, cl, mid, p * 2), query(l, r, mid + 1, cr, p *
2 + 1));
        }
    }
};
```

5. 区间全相等、区间单增、单减

```
template <typename T>
class SegmentTree1
{
public:
    int n;
    // vector<T> tree, mark;
    struct node
    {
        T val, mark;
        bool iseq;
    };

    vector<node> tree;

    struct qres1
    {
        T val;
        bool ok, iseq;
    };

    void push_down(int p)
    {
        tree[2 * p].val += tree[p].mark;
        tree[2 * p + 1].val += tree[p].mark;
        tree[2 * p].mark += tree[p].mark;
        tree[2 * p + 1].mark += tree[p].mark;
        tree[p].mark = 0;
    }

public:
    vector<T> arr;
    SegmentTree1(int _n) : n(_n)
```

```
    {
        tree.resize(4 * n + 10);
        arr.resize(n + 10);
    }
    void build(int l, int r, int p = 1)
    {
        if (l == r)
        {
            tree[p].val = arr[l];
            tree[p].mark = 0;
            tree[p].iseq = true;
        }
        else
        {
            int middle = (l + r) / 2;
            build(l, middle, 2 * p);
            build(middle + 1, r, 2 * p + 1);
            tree[p].val = tree[2 * p].val;
            tree[p].mark = 0;
            tree[p].iseq = tree[2 * p].val == tree[2 * p + 1].val && tree[2 *
p].iseq && tree[2 * p + 1].iseq;
        }
    }
    void update(int l, int r, int cl, int cr, T d, int p = 1)
    {
        if (cr < l || cl > r)
            return;
        else if (l <= cl && cr <= r)
        {
            tree[p].val += d;
            if (cr > cl)
                tree[p].mark += d;
        }
        else
        {
            int middle = (cl + cr) / 2;
            push_down(p);
            update(l, r, cl, middle, d, p * 2);
            update(l, r, middle + 1, cr, d, p * 2 + 1);
            tree[p].val = tree[2 * p].val;
            tree[p].iseq = tree[2 * p].val == tree[2 * p + 1].val && tree[2 *
p].iseq && tree[2 * p + 1].iseq;
        }
    }
    qres1 query(int l, int r, int cl, int cr, int p = 1)
    {
        if (cl > r || cr < l)
            return (qres1){0, false, true};
        else if (cl >= l && cr <= r)
        {
            return (qres1){tree[p].val, true, tree[p].iseq};
        }
        else
        {
            int mid = (cl + cr) / 2;
```

```
            push_down(p);
            // return query(l, r, cl, mid, p * 2) && query(l, r, mid + 1, cr, p
* 2 + 1) && tree[2 * p].val == tree[2 * p + 1].val;
            qres1 res1 = query(l, r, cl, mid, p * 2);
            qres1 res2 = query(l, r, mid + 1, cr, p * 2 + 1);

            if (res1.ok && res2.ok)
            {
                return (qres1){res1.val, true, res1.iseq && res2.iseq &&
res1.val == res2.val};
            }
            else if (res1.ok && !res2.ok)
            {
                return (qres1){res1.val, true, res1.iseq};
            }
            else if (!res1.ok && res2.ok)
            {
                return (qres1){res2.val, true, res2.iseq};
            }
            else
            {
                return (qres1){0, false, true};
            }
        }
    }
};

template <typename T>
class SegmentTree2
{
private:
    int n;
    // vector<T> tree, mark;
    struct node
    {
        T lval, rval, mark;
        bool isgt;
    };
    vector<node> tree;

    struct qres2
    {
        T lval, rval;
        bool ok, isgt;
    };

    void push_down(int p)
    {
        tree[2 * p].lval += tree[p].mark;
        tree[2 * p + 1].lval += tree[p].mark;
        tree[2 * p].rval += tree[p].mark;
        tree[2 * p + 1].rval += tree[p].mark;
        tree[2 * p].mark += tree[p].mark;
        tree[2 * p + 1].mark += tree[p].mark;
        tree[p].mark = 0;
```

```
    }

public:
    vector<T> arr;
    SegmentTree2(int _n) : n(_n)
    {
        tree.resize(4 * n + 10);
        arr.resize(n + 10);
    }
    void build(int l, int r, int p = 1)
    {
        if (l == r)
        {
            tree[p].lval = arr[l];
            tree[p].rval = arr[l];
            tree[p].mark = 0;
            tree[p].isgt = true;
        }
        else
        {
            int middle = (l + r) / 2;
            build(l, middle, 2 * p);
            build(middle + 1, r, 2 * p + 1);
            tree[p].lval = tree[2 * p].lval;
            tree[p].rval = tree[2 * p + 1].rval;
            tree[p].mark = 0;
            tree[p].isgt = tree[2 * p].rval < tree[2 * p + 1].lval && tree[2 *
p].isgt && tree[2 * p + 1].isgt;
        }
    }
    void update(int l, int r, int cl, int cr, T d, int p = 1)
    {
        if (cr < l || cl > r)
            return;
        else if (l <= cl && cr <= r)
        {
            tree[p].lval += d;
            tree[p].rval += d;
            if (cr > cl)
                tree[p].mark += d;
        }
        else
        {
            int middle = (cl + cr) / 2;
            push_down(p);
            update(l, r, cl, middle, d, p * 2);
            update(l, r, middle + 1, cr, d, p * 2 + 1);
            tree[p].lval = tree[2 * p].lval;
            tree[p].rval = tree[2 * p + 1].rval;
            tree[p].isgt = tree[2 * p].rval < tree[2 * p + 1].lval && tree[2 *
p].isgt && tree[2 * p + 1].isgt;
        }
    }
    qres2 query(int l, int r, int cl, int cr, int p = 1)
    {
```

```
        if (cl > r || cr < l)
            return (qres2){0, 0, false, true};
        else if (cl >= l && cr <= r)
            return (qres2){tree[p].lval, tree[p].rval, true, tree[p].isgt};
        else
        {
            int mid = (cl + cr) / 2;
            push_down(p);
            // return query(l, r, cl, mid, p * 2) && query(l, r, mid + 1, cr, p
* 2 + 1) && tree[2 * p].rval < tree[2 * p + 1].lval;
//            return query(l, r, cl, mid, p * 2) && query(l, r, mid + 1, cr, p *
2 + 1);
            qres2 res1 = query(l, r, cl, mid, p * 2);
            qres2 res2 = query(l, r, mid + 1, cr, p * 2 + 1);

            if (res1.ok && res2.ok)
            {
                return (qres2){res1.lval, res2.rval, true, res1.isgt &&
res2.isgt && res1.rval < res2.lval};
            }
            else if (res1.ok && !res2.ok)
            {
                return (qres2){res1.lval, res2.rval, true, res1.isgt};
            }
            else if (!res1.ok && res2.ok)
            {
                return (qres2){res2.lval, res2.rval, true, res2.isgt};
            }
            else
            {
                return (qres2){0, 0, false, true};
            }
        }
    }
};

template <typename T>
class SegmentTree3
{
private:
    int n;
    // vector<T> tree, mark;
    struct node
    {
        T lval, rval, mark;
        bool islt;
    };
    vector<node> tree;
    struct qres3
    {
        T lval, rval;
        bool ok, islt;
    };

    void push_down(int p)
```

```
    {
        tree[2 * p].lval += tree[p].mark;
        tree[2 * p + 1].lval += tree[p].mark;
        tree[2 * p].rval += tree[p].mark;
        tree[2 * p + 1].rval += tree[p].mark;
        tree[2 * p].mark += tree[p].mark;
        tree[2 * p + 1].mark += tree[p].mark;
        tree[p].mark = 0;
    }

public:
    vector<T> arr;
    SegmentTree3(int _n) : n(_n)
    {
        tree.resize(4 * n + 10);
        arr.resize(n + 10);
    }
    void build(int l, int r, int p = 1)
    {
        if (l == r)
        {
            tree[p].lval = arr[l];
            tree[p].rval = arr[l];
            tree[p].mark = 0;
            tree[p].islt = true;
        }
        else
        {
            int middle = (l + r) / 2;
            build(l, middle, 2 * p);
            build(middle + 1, r, 2 * p + 1);
            tree[p].lval = tree[2 * p].lval;
            tree[p].rval = tree[2 * p + 1].rval;
            tree[p].mark = 0;
            tree[p].islt = tree[2 * p].rval > tree[2 * p + 1].lval && tree[2 *
p].islt && tree[2 * p + 1].islt;
        }
    }
    void update(int l, int r, int cl, int cr, T d, int p = 1)
    {
        if (cr < l || cl > r)
            return;
        else if (l <= cl && cr <= r)
        {
            tree[p].lval += d;
            tree[p].rval += d;
            if (cr > cl)
                tree[p].mark += d;
        }
        else
        {
            int middle = (cl + cr) / 2;
            push_down(p);
            update(l, r, cl, middle, d, p * 2);
            update(l, r, middle + 1, cr, d, p * 2 + 1);
```

```
            tree[p].lval = tree[2 * p].lval;
            tree[p].rval = tree[2 * p + 1].rval;
            tree[p].islt = tree[2 * p].rval > tree[2 * p + 1].lval && tree[2 *
p].islt && tree[2 * p + 1].islt;
        }
    }
    qres3 query(int l, int r, int cl, int cr, int p = 1)
    {
        if (cl > r || cr < l)
            return (qres3){0, 0, false, true};
        else if (cl >= l && cr <= r)
            return (qres3){tree[p].lval, tree[p].rval, true, tree[p].islt};
        else
        {
            int mid = (cl + cr) / 2;
            push_down(p);
            // return query(l, r, cl, mid, p * 2) && query(l, r, mid + 1, cr, p
* 2 + 1) && tree[2 * p].rval > tree[2 * p + 1].lval;
//            return query(l, r, cl, mid, p * 2) && query(l, r, mid + 1, cr, p *
2 + 1);
            qres3 res1 = query(l, r, cl, mid, p * 2);
            qres3 res2 = query(l, r, mid + 1, cr, p * 2 + 1);

            if (res1.ok && res2.ok)
            {
                return (qres3){res1.lval, res2.rval, true, res1.islt &&
res2.islt && res1.rval > res2.lval};
            }
            else if (res1.ok && !res2.ok)
            {
                return (qres3){res1.lval, res2.rval, true, res1.islt};
            }
            else if (!res1.ok && res2.ok)
            {
                return (qres3){res2.lval, res2.rval, true, res2.islt};
            }
            else
            {
                return (qres3){0, 0, false, true};
            }
        }
    }
};
```

6. 上面三棵树维护区间信息的另一种思路：维护差分数组的正负个数，区间最值（都为0则相等）

Segment_Tree_Beats

> 可以以O((n + q)logn)的复杂度处理区间cmax / cmin, 区间求和的问题
>
> 如果在以上操作加上区间加, 复杂度上界为O((n + q)log^2n), 实际运用可能为O((n + q)logn)

```
#define maxn 100005

int N;
```

```
// 初始数组
int a[maxn];

class sgt
{
    // maxi: 子区间最大值, maxi2: 子区间严格次大值
    // maxi_cnt: ma的个数, tag: 是否需要push_down
    // sumi: 子区间求和
    struct tree
    {
        int maxi, maxi2;
        int maxi_cnt, tag;
        ll sumi;
    };
    tree t[maxn << 3];

public:
    void build(int p = 1, int cl = 1, int cr = N)
    {
        t[p].tag = 0;
        if (cl == cr)
        {
            t[p].sumi = t[p].maxi = a[cl];
            t[p].maxi_cnt = 1;
            return;
        }
        int lc = p << 1, rc = lc + 1, mid = (cl + cr) >> 1;
        build(lc, cl, mid);
        build(rc, mid + 1, cr);
        merge(p);
    }
    // 合并左右子区间信息(最大值、次大值等)
    void merge(int p)
    {
        int lc = p << 1, rc = lc + 1;
        t[p].sumi = t[lc].sumi + t[rc].sumi;
        if (t[lc].maxi > t[rc].maxi)
        {
            t[p].maxi = t[lc].maxi;
            t[p].maxi_cnt = t[lc].maxi_cnt;
            t[p].maxi2 = max(t[lc].maxi2, t[rc].maxi);
        }
        else if (t[rc].maxi > t[lc].maxi)
        {
            t[p].maxi = t[rc].maxi;
            t[p].maxi_cnt = t[rc].maxi_cnt;
            t[p].maxi2 = max(t[rc].maxi2, t[lc].maxi);
        }
        else
        {
            t[p].maxi = t[lc].maxi;
            t[p].maxi_cnt = t[lc].maxi_cnt + t[rc].maxi_cnt;
            t[p].maxi2 = max(t[lc].maxi2, t[rc].maxi2);
        }
    }
```

```
// 更新左右子节点区间
void push_down(int p)
{
    if (!t[p].tag)
        return;
    t[p].tag = 0;
    int lc = p << 1, rc = lc + 1;
    if (t[lc].maxi > t[p].maxi)
    {
        t[lc].sumi -= 1ll * t[lc].maxi_cnt * (t[lc].maxi - t[p].maxi);
        t[lc].maxi = t[p].maxi;
        t[lc].tag = 1;
    }
    if (t[rc].maxi > t[p].maxi)
    {
        t[rc].sumi -= 1ll * t[rc].maxi_cnt * (t[rc].maxi - t[p].maxi);
        t[rc].maxi = t[p].maxi;
        t[rc].tag = 1;
    }
}
// for(a[l....r], cmin(a[i], x))
// 如果最大值小于x, 直接返回
// 如果x大于次大值，小于最大值, 更新sum
// 如果x小于等于次大值, 递归更新
void update_cmin(int l, int r, int x, int p = 1, int cl = 1, int cr = N)
{
    if (cl > r || cr < l)
        return;
    if (cl != cr)
        push_down(p);
    if (cl >= l && cr <= r)
    {
        if (t[p].maxi <= x)
            return;
        if (t[p].maxi2 < x)
        {
            t[p].sumi -= 1ll * t[p].maxi_cnt * (t[p].maxi - x);
            t[p].maxi = x;
            t[p].tag = 1;
            return;
        }
    }
    int lc = p << 1, rc = lc + 1, mid = (cl + cr) / 2;
    update_cmin(l, r, x, lc, cl, mid);
    update_cmin(l, r, x, rc, mid + 1, cr);
    merge(p);
}
ll query_sum(int l, int r, int p = 1, int cl = 1, int cr = N)
{
    if (cl > r || cr < l)
        return 0;
    push_down(p);
    if (cl >= l && cr <= r)
        return t[p].sumi;
    int lc = p << 1, rc = lc + 1, mid = (cl + cr) / 2;
```

```
        return query_sum(l, r, lc, cl, mid) + query_sum(l, r, rc, mid + 1, cr);
    }
    ll query_max(int l, int r, int p = 1, int cl = 1, int cr = N)
    {
        if (cl > r || cr < l)
            return 0;
        push_down(p);
        if (cl >= l && cr <= r)
            return t[p].maxi;
        int lc = p << 1, rc = lc + 1, mid = (cl + cr) / 2;
        return max(query_max(l, r, lc, cl, mid), query_max(l, r, rc, mid + 1,
cr));
    }
    void clear()
    {
        memset(t + 1, 0, sizeof(tree) * (N << 2));
    }
};
```

Segment_Tree_EX

1. dynamic node (and cnt) SegmentTree

```
namespace DST
{
#define ls(x) tree[x].ls
#define rs(x) tree[x].rs
#define val(x) tree[x].val
#define mark(x) tree[x].mark
    const int MAXV = 8e6;
    int L = -1e5, R = 1e5, cnt = 1;
    struct node
    {
        ll val, mark;
        int ls, rs;
    } tree[MAXV];
    void upd(int &p, int x, int len)
    {
        if (!p)
            p = ++cnt;
        val(p) += x * len;
        mark(p) += x;
    }
    void push_down(int p, int cl, int cr)
    {
        if (cl >= cr)
            return;
        int mid = (cl + cr - 1) / 2;
        upd(ls(p), mark(p), mid - cl + 1);
        upd(rs(p), mark(p), cr - mid);
        mark(p) = 0;
    }
    ll query(int l, int r, int p = 1, int cl = L, int cr = R)
    {
        if (cl >= l && cr <= r)
```

```
            return val(p);
        push_down(p, cl, cr);
        ll mid = (cl + cr - 1) / 2, ans = 0;
        if (mid >= l)
            ans += query(l, r, ls(p), cl, mid);
        if (mid < r)
            ans += query(l, r, rs(p), mid + 1, cr);
        return ans;
    }
    void update(int l, int r, int d, int p = 1, int cl = L, int cr = R)
    {
        if (cl >= l && cr <= r)
            return (void)(val(p) += d * (cr - cl + 1), mark(p) += d);
        push_down(p, cl, cr);
        int mid = (cl + cr - 1) / 2;
        if (mid >= l)
            update(l, r, d, ls(p), cl, mid);
        if (mid < r)
            update(l, r, d, rs(p), mid + 1, cr);
        val(p) = val(ls(p)) + val(rs(p));
    }
    // cnt part
    void insert(int v)
    {
        update(v, v, 1);
    }
    void remove(int v)
    {
        update(v, v, -1);
    }
    int countl(int v)
    {
        return query(L, v - 1);
    }
    int countg(int v)
    {
        return query(v + 1, R);
    }
    int rank(int v)
    {
        return countl(v) + 1;
    }
    int kth(int k, int p = 1, int cl = L, int cr = R)
    {
        if (cl == cr)
            return cl;
        int mid = (cl + cr - 1) / 2;
        if (val(ls(p)) >= k)
            return kth(k, ls(p), cl, mid);
        else
            return kth(k - val(ls(p)), rs(p), mid + 1, cr);
    }
    int pre(int v)
    {
        int r = countl(v);
```

```
        return kth(r);
    }
    int suc(int v)
    {
        int r = val(1) - countg(v) + 1;
        return kth(r);
    }
}
```

2. merge segment tree

```
// 1. new root
int merge(int a, int b, int l = 1, int r = nn)
{
    if (!a || !b)
        return a + b; // 如果有一个为空，就返回不为空的；如果都为空就返回空
    int c = ++cnt;
    if (l == r)
        return T[c].v = T[a].v + T[b].v, c;
    int mid = l + r >> 1;
    T[c].ls = merge(T[a].ls, T[b].ls, l, mid);
    T[c].rs = merge(T[a].rs, T[b].rs, mid + 1, r);
    pushup(c);
    return c;
}

// 2. use a's root
more efficient

int merge(int a, int b, int l = 1, int r = nn)
{
    if (!a || !b)
        return a + b;
    if (l == r)
        return T[a].v += T[b].v, a;
    int mid = l + r >> 1;
    T[a].ls = merge(T[a].ls, T[b].ls, l, mid);
    T[a].rs = merge(T[a].rs, T[b].rs, mid + 1, r);
    pushup(a);
    return a;
}
```### Sparse_Table

```cpp
class ST
{
private:
    int n, MAXN;
    vector<vector<int>> stable;

public:
    vector<int> arr;
    ST(int _n) : n(_n), MAXN(_n + 5)
    {
        arr.resize(MAXN);
```

```
        stable = vector<vector<int>>(__lg(MAXN) + 1, vector<int>(MAXN));
    }
    void init()
    {
        for (int i = 1; i <= n; ++i)
            stable[0][i] = arr[i];
        for (int i = 1; i <= __lg(MAXN); ++i)
            for (int j = 1; j + (1 << i) - 1 <= n; ++j)
                stable[i][j] = max(stable[i - 1][j], stable[i - 1][j + (1 << (i
- 1))]);
    }
    int query(int l, int r)
    {
        int s = __lg(r - l + 1);
        return max(stable[s][l], stable[s][r - (1 << s) + 1]);
    }
};
```

Splay

```
template <typename T>
class Balanced_Binary_Tree
{
private:
    // N + M
    static const int N = 200005;
    struct Point
    {
        T value;
        int count;
        int size;
        int son[2], fa;
        Point() { size = count = 0, fa = son[1] = son[0] = 0; }
        Point(T value)
        {
            this->value = value,
            size = count = 1, fa = son[1] = son[0] = 0;
        }
    } tree[N];
    int length, size, root;
    bool get_id(int x)
    {
        return tree[tree[x].fa].son[1] == x;
    }
    void upto(int x)
    {
        tree[x].size =
            tree[tree[x].son[0]].size + tree[tree[x].son[1]].size +
tree[x].count;
    }
    void connect(int x, int y, bool id)
    {
        tree[x].fa = y, tree[y].son[id] = x;
    }
    void rotate(int x)
```

```
    {
        int m(tree[x].fa), g(tree[m].fa);
        bool id(get_id(x)), mid(get_id(m));
        connect(tree[x].son[!id], m, id), upto(m),
            connect(m, x, !id), upto(x), connect(x, g, mid);
    }
    void splay(int x, int root)
    {
        root = tree[root].fa;
        while (tree[x].fa not_eq root)
            if (tree[tree[x].fa].fa == root)
                rotate(x);
            else if (get_id(x) == get_id(tree[x].fa))
                rotate(tree[x].fa), rotate(x);
            else
                rotate(x), rotate(x);
    }
    void Splay(int x)
    {
        splay(x, root), root = x;
    }

public:
    Balanced_Binary_Tree() { size = length = 0; }
    void insert(T x)
    {
        if (not size++)
        {
            tree[root = ++length] = Point(x), connect(root, 0, 0);
            return;
        }
        for (int i(root); ++tree[i].size, true;)
        {
            if (tree[i].value == x)
            {
                ++tree[i].count, Splay(i);
                return;
            }
            else
            {
                bool id(tree[i].value < x);
                if (not tree[i].son[id])
                {
                    tree[++length] = Point(x), connect(length, i, id),
                    Splay(length);
                    return;
                }
                else
                    i = tree[i].son[id];
            }
        }
    }
    void erase(T x)
    {
        --size;
```

```
        for (
            int i(root);
            tree[i].size--;
            i = tree[i].son[x > tree[i].value])
            if (tree[i].value == x)
            {
                if (--tree[i].count)
                {
                    Splay(i);
                    return;
                }
                Splay(i);
                if (not tree[i].son[0])
                {
                    connect(root = tree[i].son[1], 0, 0);
                    return;
                }
                if (not tree[i].son[1])
                {
                    connect(root = tree[i].son[0], 0, 0);
                    return;
                }
                int j(tree[i].son[0]);
                while (tree[j].son[1])
                    j = tree[j].son[1];
                splay(j, tree[i].son[0]);
                connect(tree[i].son[1], tree[i].son[0], 1),
                    upto(tree[i].son[0]),
                    connect(root = tree[i].son[0], 0, 0);
                return;
            }
    }
    int rank(T x)
    {
        int r(1);
        for (int i(root); i;)
            if (tree[i].value == x)
            {
                Splay(i);
                return tree[tree[i].son[0]].size + 1;
            }
            else if (tree[i].value < x)
                r += tree[tree[i].son[0]].size + tree[i].count,
                    i = tree[i].son[1];
            else
                i = tree[i].son[0];
        return r;
    }
    T kth(int x)
    {
        int r(1);
        for (int i(root); true;)
        {
            if (
```

```
                r + tree[tree[i].son[0]].size <= x and x < r +
tree[tree[i].son[0]].size + tree[i].count)
            {
                Splay(i);
                return tree[i].value;
            }
            else if (x < r + tree[tree[i].son[0]].size)
                i = tree[i].son[0];
            else
                r += tree[tree[i].son[0]].size + tree[i].count,
                    i = tree[i].son[1];
        }
    }
    T lower(T x)
    {
        T r;
        int li;
        for (int i(root); i;)
            if (tree[i].value >= x)
                li = i, i = tree[i].son[0];
            else
                r = tree[li = i].value, i = tree[i].son[1];
        Splay(li);
        return r;
    }
    T upper(T x)
    {
        T r;
        int li;
        for (int i(root); i;)
            if (tree[i].value <= x)
                li = i, i = tree[i].son[1];
            else
                r = tree[li = i].value, i = tree[i].son[0];
        Splay(li);
        return r;
    }
};
```

k-D_Tree

> 查询某个点集离某个点距离前m小的点

```
namespace KDtree
{
    const int maxn = 2e5 + 100;
    const int K = 3;
    int idx; // 存储超平面的维度
    struct Node
    {
        ll a[K]; // 分维度位置
        int id;  // 输入数据的标识
        int way; // 超平面的维度号
        bool operator<(const Node &oth) const
        { // 根据超平面维度决定比较函数
```

```
        return a[idx] < oth.a[idx];
    }
} p[maxn];          // 存初始数据
Node pt[4 * maxn];  // 存KDTree数据
int flag[4 * maxn]; // 存节点是否存在数据
priority_queue<pair<ll, Node>> pq;
ll dis_sqr(ll a, ll b)
{ // 距离平方
    return (a - b) * (a - b);
}
double var[K];
void build(int l, int r, int rt, int dept)
{ // 注意l要从0开始
    if (l > r)
        return;
    flag[rt] = 1; // 当前节点存在
    int lc = rt << 1;
    int rc = rt << 1 | 1;
    flag[lc] = flag[rc] = -1; // 初始化子节点不存在
    /*                                                  //根据方差选择超平面
    int maxx = 0;
    for(int i = 0; i < K; i++){
        double avg = var[i] = 0.0;
        for(int j = l; j <= r; j++){
            avg += p[j].a[i];
        }
        avg /= (r - l + 1);
        for(int j = l; j <= r; j++){
            var[i] += (avg - p[j].a[i]) * (avg - p[j].a[i]);
        }
        var[i] /= (r - l + 1);
        if(var[i] > var[maxx]) maxx = i;
    }
    idx = maxx;
    */
    idx = dept % K; // 随机选择超平面
    int mid = (l + r) >> 1;
    nth_element(p + l, p + mid, p + r + 1); // 分为左右子树
    pt[rt] = p[mid];
    pt[rt].way = idx; // 若采用方差选择超平面，则记录使用的哪一维
    build(l, mid - 1, lc, dept + 1);
    build(mid + 1, r, rc, dept + 1);
}
void query(Node &nd, int m, int rt, int dept)
{ // 查询离nd点前m小的点
    if (flag[rt] == -1)
        return;                     // 节点不存在
    pair<ll, Node> cur(0, pt[rt]); // 计算当前节点的距离
    for (int i = 0; i < K; i++)
    {
        cur.first += dis_sqr(pt[rt].a[i], nd.a[i]);
    }
    int dim = dept % K; // 随机选择超平面
    // int dim = pt[rt].way;                            //方差选择超平面
    bool fg = 0; // 判断右儿子是否存在
```

```
        int lc = rt << 1;
        int rc = rt << 1 | 1;
        if (nd.a[dim] >= pt[rt].a[dim])
            swap(lc, rc); // nd在当前分维上的大小大于等于rt在当前分维上的大小
        if (~flag[lc])
            query(nd, m, lc, dept + 1); // lc != -1
        if (pq.size() < m)
        {
            pq.push(cur);
            fg = 1;
        } // 队列未满
        else
        {
            if (cur.first < pq.top().first)
            { // 当前节点更优
                pq.pop();
                pq.push(cur);
            }
            if (dis_sqr(nd.a[dim], pt[rt].a[dim]) < pq.top().first)
            { // 右儿子内可能存在更优点
                fg = 1;
            }
        }
        if (~flag[rc] && fg)
        { // 访问右儿子
            query(nd, m, rc, dept + 1);
        }
    }
    // example
    void solve()
    {
        int n, m;
        cin >> n >> m;
        for (int i = 0; i < n; i++)
        { // 输入原有的n个点
            for (int j = 0; j < 3; j++)
            {
                cin >> p[i].a[j];
            }
            p[i].id = i;
        }
        build(0, n - 1, 1, 0); // 建树
        while (m--)
        { // 对于新的m个点查询
            while (!pq.empty())
                pq.pop();
            Node nd;
            cin >> nd.a[0] >> nd.a[1] >> nd.a[2];
            query(nd, 1, 1, 0);
            nd = pq.top().second;
            cout << nd.a[0] << ' ' << nd.a[1] << ' ' << nd.a[2] << '\n';
        }
    }
}
```

4. Game

Anti-SG

1. Anti-SG 游戏

> 决策集合为空的游戏者赢，其余规则与 SG 游戏相同。

2. SJ 定理

> 对于 Anti-SG 游戏，如果我们规定当局面中所有单一游戏的 SG 值为 0 时，游戏结束，则先手必胜当且仅当：
>
> 游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数值大于 1。
>
> 游戏的 SG 函数为 0 且没有某个单一游戏的 SG 函数大于 1。

3. anti-nim

> 有 n 堆石子，两个人可以从任意一堆石子中拿任意多个石子(不能不拿)，拿走最后一个石子的人失败。
>
> 对于这个游戏，先手必胜有两种情况：
>
> 当每堆石子都只有一个，且游戏的 SG 值为 0
>
> 至少一堆石子多于一个，且游戏的 SG 值不为 0

```
bool anti_nim()
{
    int n, cnt = 0, sg = 0, x;
    cin >> n;
    while (n--)
    {
        cin >> x;
        if (x > 1)
            ++cnt;
        sg ^= x;
    }
    if ((!cnt && sg == 0) || (cnt && sg != 0))
        return true;
    else
        return false;
}
```

Classical_Game

1. nim 游戏

> SG(x) = x
>
> a1^a2^...^an != 0 即必胜

```cpp
bool nim(int n, vector<int> &arr)
{
    int ans = 0;
    for (int i = 1; i <= n; ++i)
        ans ^= arr[i];
    return ans;
}
```

2. 威佐夫博弈

> 有两堆各若干个物品，两个人轮流从任一堆取至少一个或同时从两堆中取同样多的物品
>
> 每次至少取一个，至多不限，最后取完者胜利。
>
> 若两堆物品的初始值为 (x,y)，且 x < y，则令 z = y-x；
>
> 记 w = (int)((sqrt(5)+1)/2*z)
>
> 若 w == x，则先手必败，否则先手必胜。

```cpp
bool Wythoff(int x, int y)
{
    if (x > y)
        swap(x, y);
    int z = y - x;
    if ((int)((sqrt(5) + 1) / 2 * z) == x)
        return false; // 先手必败
    else
        return true; // 先手必胜
}
```

3. 巴什博弈

> 一堆 n 个物品，两个人轮流从中取出 1~m 个，最后取完者胜利。
>
> 同余定理：n = k*(m+1)+r，先者拿走 r 个，那么后者无论拿走几个，先者只要的数目使和为 m+1，那么先手必赢。
>
> 反之若 n = k*(m+1)，那么先手无论怎样都会输。

```cpp
bool Bash(int n, int m)
{
    if (n % (m + 1))
        return true; // 先手获胜
    else
        return false; // 后手获胜
}
```

4. 斐波那契博弈

> 有 n 个物品，两人轮流取物品，先手最少取一个，至多无上限，但不能把物品取完，
>
> 之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件，取走最后一件物品的人获胜。
>
> 先手胜当且仅当 n 不是斐波那契数。

```
bool fib(int n)
{
    int a = 1, b = 1, c = 1;
    while (c <= n)
    {
        if (c == n)
            return false; // 后手获胜
        c = a + b;
        a = b;
        b = c;
    }
    return true; // 先手获胜
}
```

5. 树的删边游戏

> 给出一个有 N 个点的树，有一个点作为树的根节点。
>
> 游戏者轮流从树中删去边，删去一条边后，不与根节点相连的部分将被移走。最后删去者胜利。
>
> 叶子节点的 SG 值为 0；中间节点的 SG 值为它的所有子节点的 SG 值加 1 后的异或和。

6. 无向图的删边游戏

> 给出一个有 N 个点的树，有一个点作为树的根节点。
>
> 游戏者轮流从树中删去边，删去一条边后，不与根节点相连的部分将被移走。谁无法移动谁输。
>
> 由 Fusion Principle，
>
> 我们可以对无向图做如下改动：将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；
>
> 所有连到原先环上的边全部改为与新点相连。这样的改动不会影响图的 SG 值。
>
> 这样的话，我们可以将任意一个无向图改成树结构，“无向图的删边游戏”就变成了“树的删边游戏”。

Every-SG

> 对于还没有结束的单一游戏，游戏者必须对该游戏进行一步决策。
>
> 其他规则与普通 SG 游戏相同。
>
> (由很多单一游戏组成，每次每个玩家需要在每一个能够操作的单一游戏上进行操作，最后结束的那个单一游戏的胜负决定整个游戏的胜负。)
>
> 那么对于每个玩家来说，都希望自己必胜的游戏玩得尽量长，自己必败的游戏玩的尽量短。
>
> 对于 SG 值为 0 的点，我们需要知道最少需要多少步才能走到结束，对于 SG 值不为 0 的点，我们需要知道最多需要多少步结束。

```
            0                (v 为终止状态)
step(v)=    max{step(u)}+1   (SG(v)>0 且 u 为 v 的后继状态 且 SG(u)=0)
            min{step(u)}+1   (SG(v)=0 且 u 为 v 的后继状态)
```

> 先手必胜当且仅当单一游戏中最大的 step 为奇数。

Multi-SG

1. Multi-SG 游戏

> Multi-SG 游戏规定，在符合拓扑原则的前提下，一个单一游戏的后继可以为多个单一游戏
>
> Multi-SG 其他规则与 SG 游戏相同。
>
> 对于一个状态来说，不同的划分方法会产生多个不同的后继，而在一个后继中可能含有多个独立的游戏
>
> 一个后继状态的 SG 值即为后继状态中独立游戏的异或和
>
> 该状态的 SG 值即为后继状态的SG值中未出现过的最小值

2. multi-nim

> 有 n 堆石子，两个人可以从任意一堆石子中拿任意多个石子(不能不拿)，
>
> 或把一堆数量不少于 2 石子分为两堆不为空的石子，最后取完者胜利。
>
> 操作一与普通的 nim 游戏等价，
>
> 操作二实际上是将一个游戏分解为两个游戏，根据 SG 定理，我们可以通过异或运算把两个游戏连接到一起，作为一个后继状态。
>
> SG(3)的后继状态有{(0), (1), (2), (1, 2)}他们的 SG 值分别为{0, 1, 2, 3}，因此 SG(3) = mex{0, 1, 2, 3} = 4。

```
        = x-1   (x%4 == 0)
SG(x)   = x     (x%4 == 1 or 2)
        = x+1   (x%4 == 3)
```

> 最后规律如上式。

```
bool multi_nim()
{
    int n, sg = 0, x;
    cin >> n;
    while (n--)
    {
        cin >> x;
        if (x % 4 == 0)
            sg ^= x - 1;
        else if (x % 4 == 3)
            sg ^= x + 1;
        else
            sg ^= x;
    }
    return sg;
}
```

SG

1. 公平组合游戏

> 1. 游戏有两个人参与，二者轮流做出决策，双方均知道游戏的完整信息；
> 2. 任意一个游戏者在某一确定状态可以作出的决策集合只与当前的状态有关，而与游戏者无关；
> 3. 游戏中的同一个状态不可能多次抵达，游戏以玩家无法行动为结束，且游戏一定会在有限步后以非平局结束。
>
> P-position(Previous-position)：必败态(简记为 P) 。
>
> N-position(Next-position)：必胜态(简记为 N)。

2. 博弈图

> 1. 没有后继状态的状态是必败状态。
> 2. 一个状态是必胜状态当且仅当存在至少一个必败状态为它的后继状态。
> 3. 一个状态是必败状态当且仅当它的所有后继状态均为必胜状态。

3. Sprague-Grundy 定理

> SG(x) = mex({SG(y)|x->y})
>
> SG(x) = 0 时，称 x 为必败态，反之为必胜态
>
> SG(x+y) = SG(x)^SG(y)
>
> 对于由 n 个有向图游戏组成的组合游戏，设它们的起点分别为 s1, s2,..., sn，
>
> 则当且仅当 SG(s1)^SG(s2)^...^SG(sn) != 0 时，这个游戏是先手必胜的。

4. mex 函数

```
int mex(auto v)
{
    unordered_set<int> S;
    for (auto e : v)
        S.insert(e);
    for (int i = 0;; ++i)
        if (S.find(i) == S.end())
            return i;
}
```

5. multi-nim 打表

```
void SG()
{
    vector<int> sg(510, 0), vis;
    sg[0] = 0, sg[1] = 1;
    for (int i = 2; i <= 505; ++i)
    {
        vis.assign(510, 0);
        for (int j = 1; j <= i; ++j)
            vis[sg[i - j]] = true;
        for (int j = 1; j < i; ++j)
            vis[sg[j] ^ sg[i - j]] = true;
        int j = 0;
```

```
        while (vis[j])
            sg[i] = ++j;
    }
}
```

5. Geometry

Calculation_2D

```
// kuangbin

namespace D2
{
    const double eps = 1e-8;
    const double inf = 1e20;
    const double pi = acos(-1.0);
    const int maxp = 1010;
    // Compares a double to zero
    int sgn(double x)
    {
        if (fabs(x) < eps)
            return 0;
        if (x < 0)
            return -1;
        else
            return 1;
    }
    // square of a double
    inline double sqr(double x) { return x * x; }
    /*
     * Point
     * Point()                - Empty constructor
     * Point(double _x,double _y)  - constructor
     * input()              - double input
     * output()             - %.2f output
     * operator ==          - compares x and y
     * operator <           - compares first by x, then by y
     * operator -           - return new Point after subtracting curresponging x
and y
     * operator ^           - cross product of 2d points
     * operator *           - dot product
     * len()                - gives length from origin
     * len2()               - gives square of length from origin
     * distance(Point p)    - gives distance from p
     * operator + Point b   - returns new Point after adding curresponging x and
y
     * operator * double k - returns new Point after multiplieing x and y by k
     * operator / double k - returns new Point after divideing x and y by k
     * rad(Point a,Point b)- returns the angle of Point a and Point b from this
Point
     * trunc(double r)      - return Point that if truncated the distance from
center to r
```

```
     * rotleft()           - returns 90 degree ccw rotated point
     * rotright()          - returns 90 degree cw rotated point
     * rotate(Point p,double angle) - returns Point after rotateing the Point
centering at p by angle radian ccw
     */
    struct Point
    {
        double x, y;
        Point() {}
        Point(double _x, double _y)
        {
            x = _x;
            y = _y;
        }
        void input()
        {
            cin >> x >> y;
        }
        void output()
        {
            cout << x << " " << y << endl;
        }
        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);
        }
        // 叉积
        double operator^(const Point &b) const
        {
            return x * b.y - y * b.x;
        }
        // 点积
        double operator*(const Point &b) const
        {
            return x * b.x + y * b.y;
        }
        // 返回长度
        double len()
        {
            return hypot(x, y); // 库函数
        }
        // 返回长度的平方
        double len2()
        {
            return x * x + y * y;
        }
        // 返回两点的距离
```

```
    double distance(Point p)
    {
        return hypot(x - p.x, y - p.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);
    }
    // 计算pa  和  pb 的夹角
    // 就是求这个点看a,b 所成的夹角
    // 测试 LightOJ1203
    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度
    Point rotleft()
    {
        return Point(-y, x);
    }
    // 顺时针旋转90度
    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);
    }
};
/*
 * Stores two points
 * Line()                          - Empty constructor
 * Line(Point _s,Point _e)         - Line through _s and _e
```

```
 * operator ==                     - checks if two points are same
 * Line(Point p,double angle)      - one end p , another end at angle degree
 * Line(double a,double b,double c) - Line of equation ax + by + c = 0
 * input()                         - inputs s and e
 * adjust()                        - orders in such a way that s < e
 * length()                        - distance of se
 * angle()                         - return 0 <= angle < pi
 * relation(Point p)               - 3 if point is on line
 *                                   1 if point on the left of line
 *                                   2 if point on the right of line
 * pointonseg(double p)            - return true if point on segment
 * parallel(Line v)                - return true if they are parallel
 * segcrossseg(Line v)             - returns 0 if does not intersect
 *                                   returns 1 if non-standard intersection
 *                                   returns 2 if intersects
 * linecrossseg(Line v)            - line and seg
 * linecrossline(Line v)           - 0 if parallel
 *                                   1 if coincides
 *                                   2 if intersects
 * crosspoint(Line v)              - returns intersection point
 * dispointtoline(Point p)         - distance from point p to the line
 * dispointtoseg(Point p)          - distance from p to the segment
 * dissegtoseg(Line v)             - distance of two segment
 * lineprog(Point p)               - returns projected point p on se line
 * symmetrypoint(Point p)          - returns reflection point of p over se
 *
 */
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)));
        }
    }
    // ax+by+c=0
    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 0<=angle<pi
        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;
```

```
}
// 点在线段上的判断
bool pointonseg(Point p)
{
    return sgn((p - s) ^ (e - s)) == 0 && sgn((p - s) * (p - e)) <= 0;
}
// 两向量平行(对应直线平行或重合)
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);
}
// 直线和线段相交判断
//-*this line   -v seg
// 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;
}
// 求两直线的交点
// 要保证两直线不平行或重合
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));
        }
        // 点到直线的距离
        double dispointtoline(Point p)
        {
            return fabs((p - s) ^ (e - s)) / length();
        }
        // 点到线段的距离
        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);
        }
        // 返回线段到线段的距离
        // 前提是两线段不相交，相交距离就是0了
        double dissegtoseg(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);
        }
    };
    // 圆
    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
```

```
        // 测试：UVA12304
        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，只是为了和上面外接圆函数区别
        // 测试：UVA12304
        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();
            cin >> r;
        }
        // 输出
        void output()
        {
            cout << p.x << " " << p.y << " " << r << endl;
        }
        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
        // 测试：UVA12304
        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
// 测试：UVA12304
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 gercircle(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 = r;
    return t;
}
// 得到与直线u相切，过点q,半径为r1的圆
// 测试：UVA12304
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
        // 测试：UVA12304
        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
        // 测试：UVA12304
        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)
// 测试：UVA12304
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
// 测试：POJ3675 HDU3982 HDU2892
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;
        }
    };

    /*
     * n,p  Line l for each side
     * input(int _n)                        - inputs _n size polygon
     * add(Point q)                         - adds a point at end of the list
     * getline()                            - populates line array
     * cmp                                  - comparision in convex_hull order
     * norm()                               - sorting in convex_hull order
     * getconvex(polygon &convex)           - returns convex hull in convex
     * Graham(polygon &convex)              - returns convex hull in convex
     * isconvex()                           - checks if convex
     * relationpoint(Point q)               - returns 3 if q is a vertex
     *                                                2 if on a side
     *                                                1 if inside
     *                                                0 if outside
     * convexcut(Line u,polygon &po)        - left side of u in po
     * gercircumference()                   - returns side length
     * getarea()                            - returns area
     * getdir()                             - returns 0 for cw, 1 for ccw
     * getbarycentre()                      - returns barycenter
     *
     */
    struct polygon
    {
        int n;
        Point p[maxp];
        Line l[maxp];
        void input(int _n)
        {
            n = _n;
            for (int i = 0; i < n; i++)
                p[i].input();
        }
        void add(Point q)
```

```
{
    p[n++] = q;
}
void getline()
{
    for (int i = 0; i < n; i++)
    {
        l[i] = Line(p[i], p[(i + 1) % n]);
    }
}
struct cmp
{
    Point p;
    cmp(const Point &p0) { p = p0; }
    bool operator()(const Point &aa, const Point &bb)
    {
        Point a = aa, b = bb;
        int d = sgn((a - p) ^ (b - p));
        if (d == 0)
        {
            return sgn(a.distance(p) - b.distance(p)) < 0;
        }
        return d > 0;
    }
};
// 进行极角排序
// 首先需要找到最左下角的点
// 需要重载号好Point的 < 操作符(min函数要用)
void norm()
{
    Point mi = p[0];
    for (int i = 1; i < n; i++)
        mi = min(mi, p[i]);
    sort(p, p + n, cmp(mi));
}
// 得到凸包
// 得到的凸包里面的点编号是0$\sim$n-1的
// 两种凸包的方法
// 注意如果有影响，要特判下所有点共点，或者共线的特殊情况
// 测试 LightOJ1203  LightOJ1239
void getconvex(polygon &convex)
{
    sort(p, p + n);
    convex.n = n;
    for (int i = 0; i < min(n, 2); i++)
    {
        convex.p[i] = p[i];
    }
    if (convex.n == 2 && (convex.p[0] == convex.p[1]))
        convex.n--; // 特判
    if (n <= 2)
        return;
    int &top = convex.n;
    top = 1;
    for (int i = 2; i < n; i++)
```

```
            {
                while (top && sgn((convex.p[top] - p[i]) ^ (convex.p[top - 1] -
p[i])) <= 0)
                    top--;
                convex.p[++top] = p[i];
            }
            int temp = top;
            convex.p[++top] = p[n - 2];
            for (int i = n - 3; i >= 0; i--)
            {
                while (top != temp && sgn((convex.p[top] - p[i]) ^ (convex.p[top
- 1] - p[i])) <= 0)
                    top--;
                convex.p[++top] = p[i];
            }
            if (convex.n == 2 && (convex.p[0] == convex.p[1]))
                convex.n--; // 特判
            convex.norm();  // 原来得到的是顺时针的点，排序后逆时针
        }
        // 得到凸包的另外一种方法
        // 测试 LightOJ1203  LightOJ1239
        void Graham(polygon &convex)
        {
            norm();
            int &top = convex.n;
            top = 0;
            if (n == 1)
            {
                top = 1;
                convex.p[0] = p[0];
                return;
            }
            if (n == 2)
            {
                top = 2;
                convex.p[0] = p[0];
                convex.p[1] = p[1];
                if (convex.p[0] == convex.p[1])
                    top--;
                return;
            }
            convex.p[0] = p[0];
            convex.p[1] = p[1];
            top = 2;
            for (int i = 2; i < n; i++)
            {
                while (top > 1 && sgn((convex.p[top - 1] - convex.p[top - 2]) ^
(p[i] - convex.p[top - 2])) <= 0)
                    top--;
                convex.p[top++] = p[i];
            }
            if (convex.n == 2 && (convex.p[0] == convex.p[1]))
                convex.n--; // 特判
        }
        // 判断是不是凸的
```

```
bool isconvex()
{
    bool s[3];
    memset(s, false, sizeof(s));
    for (int i = 0; i < n; i++)
    {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        s[sgn((p[j] - p[i]) ^ (p[k] - p[i])) + 1] = true;
        if (s[0] && s[2])
            return false;
    }
    return true;
}
// 判断点和任意多边形的关系
//  3 点上
//  2 边上
//  1 内部
//  0 外部
int relationpoint(Point q)
{
    for (int i = 0; i < n; i++)
    {
        if (p[i] == q)
            return 3;
    }
    getline();
    for (int i = 0; i < n; i++)
    {
        if (l[i].pointonseg(q))
            return 2;
    }
    int cnt = 0;
    for (int i = 0; i < n; i++)
    {
        int j = (i + 1) % n;
        int k = sgn((q - p[j]) ^ (p[i] - p[j]));
        int u = sgn(p[i].y - q.y);
        int v = sgn(p[j].y - q.y);
        if (k > 0 && u < 0 && v >= 0)
            cnt++;
        if (k < 0 && v < 0 && u >= 0)
            cnt--;
    }
    return cnt != 0;
}
// 直线u切割凸多边形左侧
// 注意直线方向
// 测试：HDU3982
void convexcut(Line u, polygon &po)
{
    int &top = po.n; // 注意引用
    top = 0;
    for (int i = 0; i < n; i++)
    {
```

```
            int d1 = sgn((u.e - u.s) ^ (p[i] - u.s));
            int d2 = sgn((u.e - u.s) ^ (p[(i + 1) % n] - u.s));
            if (d1 >= 0)
                po.p[top++] = p[i];
            if (d1 * d2 < 0)
                po.p[top++] = u.crosspoint(Line(p[i], p[(i + 1) % n]));
        }
    }
    // 得到周长
    // 测试 LightOJ1239
    double getcircumference()
    {
        double sum = 0;
        for (int i = 0; i < n; i++)
        {
            sum += p[i].distance(p[(i + 1) % n]);
        }
        return sum;
    }
    // 得到面积
    double getarea()
    {
        double sum = 0;
        for (int i = 0; i < n; i++)
        {
            sum += (p[i] ^ p[(i + 1) % n]);
        }
        return fabs(sum) / 2;
    }
    // 得到方向
    //  1 表示逆时针，0表示顺时针
    bool getdir()
    {
        double sum = 0;
        for (int i = 0; i < n; i++)
            sum += (p[i] ^ p[(i + 1) % n]);
        if (sgn(sum) > 0)
            return 1;
        return 0;
    }
    // 得到重心
    Point getbarycentre()
    {
        Point ret(0, 0);
        double area = 0;
        for (int i = 1; i < n - 1; i++)
        {
            double tmp = (p[i] - p[0]) ^ (p[i + 1] - p[0]);
            if (sgn(tmp) == 0)
                continue;
            area += tmp;
            ret.x += (p[0].x + p[i].x + p[i + 1].x) / 3 * tmp;
            ret.y += (p[0].y + p[i].y + p[i + 1].y) / 3 * tmp;
        }
        if (sgn(area))
```

```
                ret = ret / area;
            return ret;
        }
        // 多边形和圆交的面积
        // 测试：POJ3675 HDU3982 HDU2892
        double areacircle(circle c)
        {
            double ans = 0;
            for (int i = 0; i < n; i++)
            {
                int j = (i + 1) % n;
                if (sgn((p[j] - c.p) ^ (p[i] - c.p)) >= 0)
                    ans += c.areatriangle(p[i], p[j]);
                else
                    ans -= c.areatriangle(p[i], p[j]);
            }
            return fabs(ans);
        }
        // 多边形和圆关系
        //  2 圆完全在多边形内
        //  1 圆在多边形里面，碰到了多边形边界
        //  0 其它
        int relationcircle(circle c)
        {
            getline();
            int x = 2;
            if (relationpoint(c.p) != 1)
                return 0; // 圆心不在内部
            for (int i = 0; i < n; i++)
            {
                if (c.relationseg(l[i]) == 2)
                    return 0;
                if (c.relationseg(l[i]) == 1)
                    x = 1;
            }
            return x;
        }
    };
    // AB X AC
    double cross(Point A, Point B, Point C)
    {
        return (B - A) ^ (C - A);
    }
    // AB*AC
    double dot(Point A, Point B, Point C)
    {
        return (B - A) * (C - A);
    }
    // 最小矩形面积覆盖
    //  A 必须是凸包(而且是逆时针顺序)
    //  测试 UVA 10173
    double minRectangleCover(polygon A)
    {
        // 要特判A.n < 3的情况
        if (A.n < 3)
```

```
            return 0.0;
        A.p[A.n] = A.p[0];
        double ans = -1;
        int r = 1, p = 1, q;
        for (int i = 0; i < A.n; i++)
        {
            // 卡出离边A.p[i] - A.p[i+1]最远的点
            while (sgn(cross(A.p[i], A.p[i + 1], A.p[r + 1]) - cross(A.p[i],
A.p[i + 1], A.p[r])) >= 0)
                r = (r + 1) % A.n;
            // 卡出A.p[i] - A.p[i+1]方向上正向n最远的点
            while (sgn(dot(A.p[i], A.p[i + 1], A.p[p + 1]) - dot(A.p[i], A.p[i +
1], A.p[p])) >= 0)
                p = (p + 1) % A.n;
            if (i == 0)
                q = p;
            // 卡出A.p[i] - A.p[i+1]方向上负向最远的点
            while (sgn(dot(A.p[i], A.p[i + 1], A.p[q + 1]) - dot(A.p[i], A.p[i +
1], A.p[q])) <= 0)
                q = (q + 1) % A.n;
            double d = (A.p[i] - A.p[i + 1]).len2();
            double tmp = cross(A.p[i], A.p[i + 1], A.p[r]) *
                         (dot(A.p[i], A.p[i + 1], A.p[p]) - dot(A.p[i], A.p[i +
1], A.p[q])) / d;
            if (ans < 0 || ans > tmp)
                ans = tmp;
        }
        return ans;
    }

    // 直线切凸多边形
    // 多边形是逆时针的，在q1q2的左侧
    // 测试:HDU3982
    vector<Point> convexCut(const vector<Point> &ps, Point q1, Point q2)
    {
        vector<Point> qs;
        int n = ps.size();
        for (int i = 0; i < n; i++)
        {
            Point p1 = ps[i], p2 = ps[(i + 1) % n];
            int d1 = sgn((q2 - q1) ^ (p1 - q1)), d2 = sgn((q2 - q1) ^ (p2 -
q1));
            if (d1 >= 0)
                qs.push_back(p1);
            if (d1 * d2 < 0)
                qs.push_back(Line(p1, p2).crosspoint(Line(q1, q2)));
        }
        return qs;
    }
    // 半平面交
    // 测试 POJ3335 POJ1474 POJ1279
    //*****************************
    struct halfplane : public Line
    {
        double angle;
```

```
        halfplane() {}
        // 表示向量s->e逆时针(左侧)的半平面
        halfplane(Point _s, Point _e)
        {
            s = _s;
            e = _e;
        }
        halfplane(Line v)
        {
            s = v.s;
            e = v.e;
        }
        void calcangle()
        {
            angle = atan2(e.y - s.y, e.x - s.x);
        }
        bool operator<(const halfplane &b) const
        {
            return angle < b.angle;
        }
    };
    struct halfplanes
    {
        int n;
        halfplane hp[maxp];
        Point p[maxp];
        int que[maxp];
        int st, ed;
        void push(halfplane tmp)
        {
            hp[n++] = tmp;
        }
        // 去重
        void unique()
        {
            int m = 1;
            for (int i = 1; i < n; i++)
            {
                if (sgn(hp[i].angle - hp[i - 1].angle) != 0)
                    hp[m++] = hp[i];
                else if (sgn((hp[m - 1].e - hp[m - 1].s) ^ (hp[i].s - hp[m -
1].s)) > 0)
                    hp[m - 1] = hp[i];
            }
            n = m;
        }
        bool halfplaneinsert()
        {
            for (int i = 0; i < n; i++)
                hp[i].calcangle();
            sort(hp, hp + n);
            unique();
            que[st = 0] = 0;
            que[ed = 1] = 1;
            p[1] = hp[0].crosspoint(hp[1]);
```

```
            for (int i = 2; i < n; i++)
            {
                while (st < ed && sgn((hp[i].e - hp[i].s) ^ (p[ed] - hp[i].s)) <
0)
                    ed--;
                while (st < ed && sgn((hp[i].e - hp[i].s) ^ (p[st + 1] -
hp[i].s)) < 0)
                    st++;
                que[++ed] = i;
                if (hp[i].parallel(hp[que[ed - 1]]))
                    return false;
                p[ed] = hp[i].crosspoint(hp[que[ed - 1]]);
            }
            while (st < ed && sgn((hp[que[st]].e - hp[que[st]].s) ^ (p[ed] -
hp[que[st]].s)) < 0)
                ed--;
            while (st < ed && sgn((hp[que[ed]].e - hp[que[ed]].s) ^ (p[st + 1] -
hp[que[ed]].s)) < 0)
                st++;
            if (st + 1 >= ed)
                return false;
            return true;
        }
        // 得到最后半平面交得到的凸多边形
        // 需要先调用halfplaneinsert() 且返回true
        void getconvex(polygon &con)
        {
            p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
            con.n = ed - st + 1;
            for (int j = st, i = 0; j <= ed; i++, j++)
                con.p[i] = p[j];
        }
    };
    //*****************************

    const int maxn = 1010;
    struct circles
    {
        circle c[maxn];
        double ans[maxn]; // ans[i]表示被覆盖了i次的面积
        double pre[maxn];
        int n;
        circles() {}
        void add(circle cc)
        {
            c[n++] = cc;
        }
        // x包含在y中
        bool inner(circle x, circle y)
        {
            if (x.relationcircle(y) != 1)
                return 0;
            return sgn(x.r - y.r) <= 0 ? 1 : 0;
        }
        // 圆的面积并去掉内含的圆
```

```
void init_or()
{
    bool mark[maxn] = {0};
    int i, j, k = 0;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            if (i != j && !mark[j])
            {
                if ((c[i] == c[j]) || inner(c[i], c[j]))
                    break;
            }
        if (j < n)
            mark[i] = 1;
    }
    for (i = 0; i < n; i++)
        if (!mark[i])
            c[k++] = c[i];
    n = k;
}
// 圆的面积交去掉内含的圆
void init_add()
{
    int i, j, k = 0;
    bool mark[maxn] = {0};
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            if (i != j && !mark[j])
            {
                if ((c[i] == c[j]) || inner(c[j], c[i]))
                    break;
            }
        if (j < n)
            mark[i] = 1;
    }
    for (i = 0; i < n; i++)
        if (!mark[i])
            c[k++] = c[i];
    n = k;
}
// 半径为r的圆，弧度为th对应的弓形的面积
double areaarc(double th, double r)
{
    return 0.5 * r * r * (th - sin(th));
}
// 测试SPOJVCIRCLES SPOJCIRUT
// SPOJVCIRCLES求n个圆并的面积，需要加上init\_or()去掉重复圆（否则WA）
// SPOJCIRUT 是求被覆盖k次的面积，不能加init\_or()
// 对于求覆盖多少次面积的问题，不能解决相同圆，而且不能init\_or()
// 求多圆面积并，需要init\_or,其中一个目的就是去掉相同圆
void getarea()
{
    memset(ans, 0, sizeof(ans));
    vector<pair<double, int>> v;
```

```
            for (int i = 0; i < n; i++)
            {
                v.clear();
                v.push_back(make_pair(-pi, 1));
                v.push_back(make_pair(pi, -1));
                for (int j = 0; j < n; j++)
                    if (i != j)
                    {
                        Point q = (c[j].p - c[i].p);
                        double ab = q.len(), ac = c[i].r, bc = c[j].r;
                        if (sgn(ab + ac - bc) <= 0)
                        {
                            v.push_back(make_pair(-pi, 1));
                            v.push_back(make_pair(pi, -1));
                            continue;
                        }
                        if (sgn(ab + bc - ac) <= 0)
                            continue;
                        if (sgn(ab - ac - bc) > 0)
                            continue;
                        double th = atan2(q.y, q.x), fai = acos((ac * ac + ab *
ab - bc * bc) / (2.0 * ac * ab));
                        double a0 = th - fai;
                        if (sgn(a0 + pi) < 0)
                            a0 += 2 * pi;
                        double a1 = th + fai;
                        if (sgn(a1 - pi) > 0)
                            a1 -= 2 * pi;
                        if (sgn(a0 - a1) > 0)
                        {
                            v.push_back(make_pair(a0, 1));
                            v.push_back(make_pair(pi, -1));
                            v.push_back(make_pair(-pi, 1));
                            v.push_back(make_pair(a1, -1));
                        }
                        else
                        {
                            v.push_back(make_pair(a0, 1));
                            v.push_back(make_pair(a1, -1));
                        }
                    }
                sort(v.begin(), v.end());
                int cur = 0;
                for (int j = 0; j < v.size(); j++)
                {
                    if (cur && sgn(v[j].first - pre[cur]))
                    {
                        ans[cur] += areaarc(v[j].first - pre[cur], c[i].r);
                        ans[cur] += 0.5 * (Point(c[i].p.x + c[i].r *
cos(pre[cur]), c[i].p.y + c[i].r * sin(pre[cur])) ^ Point(c[i].p.x + c[i].r *
cos(v[j].first), c[i].p.y + c[i].r * sin(v[j].first)));
                    }
                    cur += v[j].second;
                    pre[cur] = v[j].first;
                }
```

```
            }
            for (int i = 1; i < n; i++)
                ans[i] -= ans[i + 1];
        }
    };
}

// 平面最近点对
namespace closestPoints
{
    const int MAXN = 100010;
    const double eps = 1e-8;
    const double INF = 1e20;
    struct Point
    {
        double x, y;
        void input()
        {
            cin >> x >> y;
        }
    };
    double dist(Point a, Point b)
    {
        return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
    }
    Point p[MAXN];
    Point tmpt[MAXN];
    bool cmpx(Point a, Point b)
    {
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
    bool cmpy(Point a, Point b)
    {
        return a.y < b.y || (a.y == b.y && a.x < b.x);
    }
    double Closest_Pair(int left, int right)
    {
        double d = INF;
        if (left == right)
            return d;
        if (left + 1 == right)
            return dist(p[left], p[right]);
        int mid = (left + right) / 2;
        double d1 = Closest_Pair(left, mid);
        double d2 = Closest_Pair(mid + 1, right);
        d = min(d1, d2);
        int cnt = 0;
        for (int i = left; i <= right; i++)
        {
            if (fabs(p[mid].x - p[i].x) <= d)
                tmpt[cnt++] = p[i];
        }
        sort(tmpt, tmpt + cnt, cmpy);
        for (int i = 0; i < cnt; i++)
        {
```

```
            for (int j = i + 1; j < cnt && tmpt[j].y - tmpt[i].y < d; j++)
                d = min(d, dist(tmpt[i], tmpt[j]));
        }
        return d;
    }
    int main()
    {
        int n;
        while (cin >> n)
        {
            for (int i = 0; i < n; i++)
                p[i].input();
            sort(p, p + n, cmpx);
            cout << Closest_Pair(0, n - 1) << endl;
        }
        return 0;
    }
}
```

Calculation_3D

```
// kuangbin

namespace D3
{

    const double eps = 1e-8;
    int sgn(double x)
    {
        if (fabs(x) < eps)
            return 0;
        if (x < 0)
            return -1;
        else
            return 1;
    }
    struct Point3
    {
        double x, y, z;
        Point3(double _x = 0, double _y = 0, double _z = 0)
        {
            x = _x;
            y = _y;
            z = _z;
        }
        void input()
        {
            cin >> x >> y >> z;
        }
        void output()
        {
            cout << x << " " << y << " " << z << endl;
        }
        bool operator==(const Point3 &b) const
        {
```

```
            return sgn(x - b.x) == 0 && sgn(y - b.y) == 0 && sgn(z - b.z) == 0;
        }
        bool operator<(const Point3 &b) const
        {
            return sgn(x - b.x) == 0 ? (sgn(y - b.y) == 0 ? sgn(z - b.z) < 0 : y
< b.y) : x < b.x;
        }
        double len()
        {
            return sqrt(x * x + y * y + z * z);
        }
        double len2()
        {
            return x * x + y * y + z * z;
        }
        double distance(const Point3 &b) const
        {
            return sqrt((x - b.x) * (x - b.x) + (y - b.y) * (y - b.y) + (z -
b.z) * (z - b.z));
        }
        Point3 operator-(const Point3 &b) const
        {
            return Point3(x - b.x, y - b.y, z - b.z);
        }
        Point3 operator+(const Point3 &b) const
        {
            return Point3(x + b.x, y + b.y, z + b.z);
        }
        Point3 operator*(const double &k) const
        {
            return Point3(x * k, y * k, z * k);
        }
        Point3 operator/(const double &k) const
        {
            return Point3(x / k, y / k, z / k);
        }
        // 点乘
        double operator*(const Point3 &b) const
        {
            return x * b.x + y * b.y + z * b.z;
        }
        // 叉乘
        Point3 operator^(const Point3 &b) const
        {
            return Point3(y * b.z - z * b.y, z * b.x - x * b.z, x * b.y - y *
b.x);
        }
        double rad(Point3 a, Point3 b)
        {
            Point3 p = (*this);
            return acos(((a - p) * (b - p)) / (a.distance(p) * b.distance(p)));
        }
        // 变换长度
        Point3 trunc(double r)
        {
```

```
        double l = len();
        if (!sgn(l))
            return *this;
        r /= l;
        return Point3(x * r, y * r, z * r);
    }
};
struct Line3
{
    Point3 s, e;
    Line3() {}
    Line3(Point3 _s, Point3 _e)
    {
        s = _s;
        e = _e;
    }
    bool operator==(const Line3 v)
    {
        return (s == v.s) && (e == v.e);
    }
    void input()
    {
        s.input();
        e.input();
    }
    double length()
    {
        return s.distance(e);
    }
    // 点到直线距离
    double dispointtoline(Point3 p)
    {
        return ((e - s) ^ (p - s)).len() / s.distance(e);
    }
    // 点到线段距离
    double dispointtoseg(Point3 p)
    {
        if (sgn((p - s) * (e - s)) < 0 || sgn((p - e) * (s - e)) < 0)
            return min(p.distance(s), e.distance(p));
        return dispointtoline(p);
    }
    // 返回点p在直线上的投影
    Point3 lineprog(Point3 p)
    {
        return s + (((e - s) * ((e - s) * (p - s))) / ((e - s).len2()));
    }
    // p绕此向量逆时针arg角度
    Point3 rotate(Point3 p, double ang)
    {
        if (sgn(((s - p) ^ (e - p)).len()) == 0)
            return p;
        Point3 f1 = (e - s) ^ (p - s);
        Point3 f2 = (e - s) ^ (f1);
        double len = ((s - p) ^ (e - p)).len() / s.distance(e);
        f1 = f1.trunc(len);
```

```
            f2 = f2.trunc(len);
            Point3 h = p + f2;
            Point3 pp = h + f1;
            return h + ((p - h) * cos(ang)) + ((pp - h) * sin(ang));
        }
        // 点在直线上
        bool pointonseg(Point3 p)
        {
            return sgn(((s - p) ^ (e - p)).len()) == 0 && sgn((s - p) * (e - p))
== 0;
        }
    };
    struct Plane
    {
        Point3 a, b, c, o; // 平面上的三个点，以及法向量
        Plane() {}
        Plane(Point3 _a, Point3 _b, Point3 _c)
        {
            a = _a;
            b = _b;
            c = _c;
            o = pvec();
        }
        Point3 pvec()
        {
            return (b - a) ^ (c - a);
        }
        // ax+by+cz+d = 0
        Plane(double _a, double _b, double _c, double _d)
        {
            o = Point3(_a, _b, _c);
            if (sgn(_a) != 0)
                a = Point3((-_d - _c - _b) / _a, 1, 1);
            else if (sgn(_b) != 0)
                a = Point3(1, (-_d - _c - _a) / _b, 1);
            else if (sgn(_c) != 0)
                a = Point3(1, 1, (-_d - _a - _b) / _c);
        }
        // 点在平面上的判断
        bool pointonplane(Point3 p)
        {
            return sgn((p - a) * o) == 0;
        }
        // 两平面夹角
        double angleplane(Plane f)
        {
            return acos(o * f.o) / (o.len() * f.o.len());
        }
        // 平面和直线的交点，返回值是交点个数
        int crossline(Line3 u, Point3 &p)
        {
            double x = o * (u.e - a);
            double y = o * (u.s - a);
            double d = x - y;
            if (sgn(d) == 0)
```

```
                return 0;
            p = ((u.s * x) - (u.e * y)) / d;
            return 1;
        }
        // 点到平面最近点(也就是投影)
        Point3 pointtoplane(Point3 p)
        {
            Line3 u = Line3(p, p + o);
            crossline(u, p);
            return p;
        }
        // 平面和平面的交线
        int crossplane(Plane f, Line3 &u)
        {
            Point3 oo = o ^ f.o;
            Point3 v = o ^ oo;
            double d = fabs(f.o * v);
            if (sgn(d) == 0)
                return 0;
            Point3 q = a + (v * (f.o * (f.a - a)) / d);
            u = Line3(q, q + oo);
            return 1;
        }
    };
}

// 三维凸包
namespace convexHull
{
    const double eps = 1e-8;
    const int MAXN = 550;
    int sgn(double x)
    {
        if (fabs(x) < eps)
            return 0;
        if (x < 0)
            return -1;
        else
            return 1;
    }
    struct Point3
    {
        double x, y, z;
        Point3(double _x = 0, double _y = 0, double _z = 0)
        {
            x = _x;
            y = _y;
            z = _z;
        }
        void input()
        {
            cin >> x >> y >> z;
        }
        bool operator==(const Point3 &b) const
        {
```

```
            return sgn(x - b.x) == 0 && sgn(y - b.y) == 0 && sgn(z - b.z) == 0;
        }
        double len()
        {
            return sqrt(x * x + y * y + z * z);
        }
        double len2()
        {
            return x * x + y * y + z * z;
        }
        double distance(const Point3 &b) const
        {
            return sqrt((x - b.x) * (x - b.x) + (y - b.y) * (y - b.y) + (z -
b.z) * (z - b.z));
        }
        Point3 operator-(const Point3 &b) const
        {
            return Point3(x - b.x, y - b.y, z - b.z);
        }
        Point3 operator+(const Point3 &b) const
        {
            return Point3(x + b.x, y + b.y, z + b.z);
        }
        Point3 operator*(const double &k) const
        {
            return Point3(x * k, y * k, z * k);
        }
        Point3 operator/(const double &k) const
        {
            return Point3(x / k, y / k, z / k);
        }
        // 点乘
        double operator*(const Point3 &b) const
        {
            return x * b.x + y * b.y + z * b.z;
        }
        // 叉乘
        Point3 operator^(const Point3 &b) const
        {
            return Point3(y * b.z - z * b.y, z * b.x - x * b.z, x * b.y - y *
b.x);
        }
    };
    struct CH3D
    {
        struct face
        {
            // 表示凸包一个面上的三个点的编号
            int a, b, c;
            // 表示该面是否属于最终的凸包上的面
            bool ok;
        };
        // 初始顶点数
        int n;
        Point3 P[MAXN];
```

```
// 凸包表面的三角形数
int num;
// 凸包表面的三角形
face F[8 * MAXN];
int g[MAXN][MAXN];
// 叉乘
Point3 cross(const Point3 &a, const Point3 &b, const Point3 &c)
{
    return (b - a) ^ (c - a);
}
// 三角形面积*2
double area(Point3 a, Point3 b, Point3 c)
{
    return ((b - a) ^ (c - a)).len();
}
// 四面体有向面积*6
double volume(Point3 a, Point3 b, Point3 c, Point3 d)
{
    return ((b - a) ^ (c - a)) * (d - a);
}
// 正：点在面同向
double dblcmp(Point3 &p, face &f)
{
    Point3 p1 = P[f.b] - P[f.a];
    Point3 p2 = P[f.c] - P[f.a];
    Point3 p3 = p - P[f.a];
    return (p1 ^ p2) * p3;
}
void deal(int p, int a, int b)
{
    int f = g[a][b];
    face add;
    if (F[f].ok)
    {
        if (dblcmp(P[p], F[f]) > eps)
            dfs(p, f);
        else
        {
            add.a = b;
            add.b = a;
            add.c = p;
            add.ok = true;
            g[p][b] = g[a][p] = g[b][a] = num;
            F[num++] = add;
        }
    }
}
// 递归搜索所有应该从凸包内删除的面
void dfs(int p, int now)
{
    F[now].ok = false;
    deal(p, F[now].b, F[now].a);
    deal(p, F[now].c, F[now].b);
    deal(p, F[now].a, F[now].c);
}
```

```
bool same(int s, int t)
{
    Point3 &a = P[F[s].a];
    Point3 &b = P[F[s].b];
    Point3 &c = P[F[s].c];
    return fabs(volume(a, b, c, P[F[t].a])) < eps &&
           fabs(volume(a, b, c, P[F[t].b])) < eps &&
           fabs(volume(a, b, c, P[F[t].c])) < eps;
}
// 构建三维凸包
void create()
{
    num = 0;
    face add;

    //*********************************
    // 此段是为了保证前四个点不共面
    bool flag = true;
    for (int i = 1; i < n; i++)
    {
        if (!(P[0] == P[i]))
        {
            swap(P[1], P[i]);
            flag = false;
            break;
        }
    }
    if (flag)
        return;
    flag = true;
    for (int i = 2; i < n; i++)
    {
        if (((P[1] - P[0]) ^ (P[i] - P[0])).len() > eps)
        {
            swap(P[2], P[i]);
            flag = false;
            break;
        }
    }
    if (flag)
        return;
    flag = true;
    for (int i = 3; i < n; i++)
    {
        if (fabs(((P[1] - P[0]) ^ (P[2] - P[0])) * (P[i] - P[0])) > eps)
        {
            swap(P[3], P[i]);
            flag = false;
            break;
        }
    }
    if (flag)
        return;
    //*********************************
```

```
        for (int i = 0; i < 4; i++)
        {
            add.a = (i + 1) % 4;
            add.b = (i + 2) % 4;
            add.c = (i + 3) % 4;
            add.ok = true;
            if (dblcmp(P[i], add) > 0)
                swap(add.b, add.c);
            g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
            F[num++] = add;
        }
        for (int i = 4; i < n; i++)
            for (int j = 0; j < num; j++)
                if (F[j].ok && dblcmp(P[i], F[j]) > eps)
                {
                    dfs(i, j);
                    break;
                }
        int tmp = num;
        num = 0;
        for (int i = 0; i < tmp; i++)
            if (F[i].ok)
                F[num++] = F[i];
    }
    // 表面积
    // 测试：HDU3528
    double area()
    {
        double res = 0;
        if (n == 3)
        {
            Point3 p = cross(P[0], P[1], P[2]);
            return p.len() / 2;
        }
        for (int i = 0; i < num; i++)
            res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
        return res / 2.0;
    }
    double volume()
    {
        double res = 0;
        Point3 tmp = Point3(0, 0, 0);
        for (int i = 0; i < num; i++)
            res += volume(tmp, P[F[i].a], P[F[i].b], P[F[i].c]);
        return fabs(res / 6);
    }
    // 表面三角形个数
    int triangle()
    {
        return num;
    }
    // 表面多边形个数
    // 测试：HDU3662
    int polygon()
    {
```

```
            int res = 0;
            for (int i = 0; i < num; i++)
            {
                bool flag = true;
                for (int j = 0; j < i; j++)
                    if (same(i, j))
                    {
                        flag = 0;
                        break;
                    }
                res += flag;
            }
            return res;
        }
        // 重心
        // 测试：HDU4273
        Point3 barycenter()
        {
            Point3 ans = Point3(0, 0, 0);
            Point3 o = Point3(0, 0, 0);
            double all = 0;
            for (int i = 0; i < num; i++)
            {
                double vol = volume(o, P[F[i].a], P[F[i].b], P[F[i].c]);
                ans = ans + (((o + P[F[i].a] + P[F[i].b] + P[F[i].c]) / 4.0) *
vol);
                all += vol;
            }
            ans = ans / all;
            return ans;
        }
        // 点到面的距离
        // 测试：HDU4273
        double ptoface(Point3 p, int i)
        {
            double tmp1 = fabs(volume(P[F[i].a], P[F[i].b], P[F[i].c], p));
            double tmp2 = ((P[F[i].b] - P[F[i].a]) ^ (P[F[i].c] -
P[F[i].a])).len();
            return tmp1 / tmp2;
        }
    };
    CH3D hull;
    int main()
    {
        while (cin >> hull.n)
        {
            for (int i = 0; i < hull.n; i++)
                hull.P[i].input();
            hull.create();
            Point3 p = hull.barycenter();
            double ans = 1e20;
            for (int i = 0; i < hull.num; i++)
                ans = min(ans, hull.ptoface(p, i));
            cout << ans << endl;
        }
```

```
        return 0;
    }
}
```

Rotating_Calipers

> 不要求顺序

```
namespace RC
{
    const int maxn = 1e6 + 5;
    const long double eps=1e-11;
    int n;

    int dcmp(long double x){
        return (fabs(x)<=eps)?0:(x<0?-1:1);
    }
    struct Point{
        long double x,y;
        Point(long double X=0,long double Y=0){x=X,y=Y;}
    };
    struct Vector{
        long double x,y;
        Vector(long double X=0,long double Y=0){x=X,y=Y;}
    };
    inline Vector operator-(Point x,Point y){// 点-点=向量
        return Vector(x.x-y.x,x.y-y.y);
    }
    inline long double cross(Vector x,Vector y){ // 向量叉积
        return x.x*y.y-x.y*y.x;
    }
    inline long double operator*(Vector x,Vector y){ // 向量叉积
        return cross(x,y);
    }
    inline long double len(Vector x){ // 向量模长
        return sqrt(x.x*x.x+x.y*x.y);
    }

    int stk[maxn];
    bool used[maxn];
    Point poly[1000005];

    vector<Point> ConvexHull(Point* poly, int n){ // Andrew算法求凸包
        int top=0;
        sort(poly+1,poly+n+1,[&](Point x,Point y){
            return (x.x==y.x)?(x.y<y.y):(x.x<y.x);
        });
        stk[++top]=1;
        for(int i=2;i<=n;i++){
            while(top>1&&dcmp((poly[stk[top]]-poly[stk[top-1]])*(poly[i]-
poly[stk[top]]))<=0){
                used[stk[top--]]=0;
            }
            used[i]=1;
            stk[++top]=i;
```

```
        }
        int tmp=top;
        for(int i=n-1;i;i--){
            if(used[i]) continue;
            while(top>tmp&&dcmp((poly[stk[top]]-poly[stk[top-1]])*(poly[i]-
poly[stk[top]]))<=0){
                used[stk[top--]]=0;
            }
            used[i]=1;
            stk[++top]=i;
        }
        vector<Point> a;
        for(int i=1;i<=top;i++){
            a.push_back(poly[stk[i]]);
        }
        return a;
    }

    struct Line{
        Point x;Vector y;
        Line(Point X,Vector Y){x=X,y=Y;}
        Line(Point X,Point Y){x=X,y=Y-X;}
    };

    inline long double DistanceToLine(Point P,Line x){// 点到直线的距离
        Vector v1=x.y, v2=P-x.x;
        return fabs(cross(v1,v2))/len(v1);
    }

    long double RoatingCalipers(vector<Point> poly){// 旋转卡壳
        if(poly.size()==3) return len(poly[1]-poly[0]);
        int cur=0;
        long double ans=0;
        for(int i=0;i<poly.size()-1;i++){
            Line line(poly[i],poly[i+1]);
            while(DistanceToLine(poly[cur], line) <=
DistanceToLine(poly[(cur+1)%poly.size()], line)){
                cur=(cur+1)%poly.size();
            }
            ans=max(ans, max(len(poly[i]-poly[cur]), len(poly[i+1]-poly[cur])));
        }
        return ans;
    }

    long double solve()
    {
        cin >> n;
        for(int i = 1; i <= n; ++i)
            cin >> poly[i].x >> poly[i].y;
        long double v = RoatingCalipers(ConvexHull(poly, n));
        return v;
    }
}
```

Sweep_Line

```
#define maxn 300

int lazy[maxn << 3];  // 标记了这条线段出现的次数
double s[maxn << 3];

struct node1 {
    double l, r;
    double sum;
} cl[maxn << 3];  // 线段树

struct node2 {
    double x, y1, y2;
    int flag;
} p[maxn << 3];  // 坐标

// 定义sort比较
bool cmp(node2 a, node2 b) { return a.x < b.x; }

// 上传
void pushup(int rt) {
    if (lazy[rt] > 0)
        cl[rt].sum = cl[rt].r - cl[rt].l;
    else
        cl[rt].sum = cl[rt * 2].sum + cl[rt * 2 + 1].sum;
}

// 建树
void build(int rt, int l, int r) {
    if (r - l > 1) {
        cl[rt].l = s[l];
        cl[rt].r = s[r];
        build(rt * 2, l, (l + r) / 2);
        build(rt * 2 + 1, (l + r) / 2, r);
        pushup(rt);
    } else {
        cl[rt].l = s[l];
        cl[rt].r = s[r];
        cl[rt].sum = 0;
    }
    return;
}

// 更新
void update(int rt, double y1, double y2, int flag) {
    if (cl[rt].l == y1 && cl[rt].r == y2) {
        lazy[rt] += flag;
        pushup(rt);
        return;
    } else {
        if (cl[rt * 2].r > y1) update(rt * 2, y1, min(cl[rt * 2].r, y2), flag);
        if (cl[rt * 2 + 1].l < y2)
        update(rt * 2 + 1, max(cl[rt * 2 + 1].l, y1), y2, flag);
        pushup(rt);
```

```
    }
}

int main() {
    int temp = 1, n;
    double x1, y1, x2, y2, ans;
    while (scanf("%d", &n) && n) {
        ans = 0;
        for (int i = 0; i < n; i++) {
            scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2);
            p[i].x = x1;
            p[i].y1 = y1;
            p[i].y2 = y2;
            p[i].flag = 1;
            p[i + n].x = x2;
            p[i + n].y1 = y1;
            p[i + n].y2 = y2;
            p[i + n].flag = -1;
            s[i + 1] = y1;
            s[i + n + 1] = y2;
        }
        sort(s + 1, s + (2 * n + 1));  // 离散化
        sort(p, p + 2 * n, cmp);  // 把矩形的边的横坐标从小到大排序
        build(1, 1, 2 * n);       // 建树
        memset(lazy, 0, sizeof(lazy));
        update(1, p[0].y1, p[0].y2, p[0].flag);
        for (int i = 1; i < 2 * n; i++) {
            ans += (p[i].x - p[i - 1].x) * cl[1].sum;
            update(1, p[i].y1, p[i].y2, p[i].flag);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n", temp++, ans);
    }
    return 0;
}
```

6.Graph

Augmenting_Path_Algorithm

> index starts from 1
>
> only save oriented graph
>
> match: n2 part to n1 part

```
void solve()
{
    const int maxn = 505;
    int n1, n2, m, u, v, ans = 0;
    cin >> n1 >> n2 >> m;
    vector<vector<int>> to(maxn);
    vector<int> match(maxn, 0), st(maxn, 0);
    for (int i = 1; i <= m; ++i)
```

```
    {
        cin >> u >> v;
        to[u].pb(v);
    }
    function<bool(int x)> find = [&](int x)->bool
    {
        for (auto v: to[x])
        {
            if (st[v])
                continue;
            st[v] = true;
            if (match[v] == 0 || find(match[v]))
            {
                match[v] = x;
                return true;
            }
        }
        return false;
    };
    for (int i = 1; i <= n1; ++i)
    {
        fill(all(st), 0);
        if (find(i))
            ++ans;
    }
    cout << ans << endl;
}
```

Bellman_Ford

```
#define MAXN 10005

struct edge
{
    int v, w;
};

vector<vector<edge>> e(MAXN);
int dis[MAXN];
const int inf = 0x3f3f3f3f;

bool bellmanford(int n, int s)
{
    memset(dis, 0x3f, sizeof(dis));
    dis[s] = 0;
    bool flag;
    for (int i = 1; i <= n; ++i)
    {
        flag = false;
        for (int u = 1; u <= n; ++u)
        {
            if (dis[u] == inf)
                continue;
            for (auto &x : e[u])
            {
```

```
                int v = x.v, w = x.w;
                if (dis[v] > dis[u] + w)
                {
                    dis[v] = dis[u] + w;
                    flag = true;
                }
            }
        }
        if (!flag)
            break;
    }
    return flag;
}

//

int vis[MAXN], cnt[MAXN];
queue<int> q;

bool spfa(int n, int s)
{
    memset(dis, 0x3f, sizeof(dis));
    dis[s] = 0, vis[s] = 1;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (auto &x : e[u])
        {
            int v = x.v, w = x.w;
            if (dis[v] > dis[u] + w)
            {
                dis[v] = dis[u] + w;
                cnt[v] = cnt[u] + 1;
                if (cnt[v] >= n)
                    return false;
                if (!vis[v])
                    q.push(v), vis[v] = 1;
            }
        }
    }
    return true;
}
```

Biconnected_Component

1. Edge-bcc

```
namespace Ebcc
{
    const int N = 5e5 + 5, M = 2e6 + 5;
    int n, m;
```

```
    struct edge {
        int to, nt;
    } e[M << 2];

    int hd[N << 1], tot = 1;

    void add(int u, int v) { e[++tot] = (edge){v, hd[u]}, hd[u] = tot; }
    void uadd(int u, int v) { add(u, v), add(v, u); }

    int bcc_cnt, sum;
    int dfn[N], low[N];
    bool vis[N];
    vector<vector<int>> ans;
    stack<int> st;

    void tarjan(int u, int in) {
        low[u] = dfn[u] = ++bcc_cnt;
        st.push(u), vis[u] = 1;
        for (int i = hd[u]; i; i = e[i].nt) {
            int v = e[i].to;
            if (i == (in ^ 1)) continue;
            if (!dfn[v])
                tarjan(v, i), low[u] = min(low[u], low[v]);
            else if (vis[v])
                low[u] = min(low[u], dfn[v]);
        }
        if (dfn[u] == low[u]) {
            vector<int> t;
            t.push_back(u);
            while (st.top() != u)
                t.push_back(st.top()), vis[st.top()] = 0, st.pop();
            st.pop(), ans.push_back(t);
        }
    }

    int main() {
        scanf("%d%d", &n, &m);
        int u, v;
        for (int i = 1; i <= m; i++) {
            scanf("%d%d", &u, &v);
            if (u != v) uadd(u, v);
        }
        for (int i = 1; i <= n; i++)
            if (!dfn[i]) tarjan(i, 0);
        printf("%llu\n", ans.size());
        for (int i = 0; i < ans.size(); i++) {
            printf("%llu ", ans[i].size());
            for (int j = 0; j < ans[i].size(); j++) printf("%d ", ans[i][j]);
            printf("\n");
        }
        return 0;
    }
}
```

2. Vetex-bcc

```cpp
namespace Vbcc
{
    const int N = 5e5 + 5, M = 2e6 + 5;
    int n, m;

    struct edge {
        int to, nt;
    } e[M << 1];

    int hd[N], tot = 1;

    void add(int u, int v) { e[++tot] = (edge){v, hd[u]}, hd[u] = tot; }
    void uadd(int u, int v) { add(u, v), add(v, u); }

    int ans;
    int dfn[N], low[N], bcc_cnt;
    int sta[N], top, cnt;
    bool cut[N];
    vector<int> dcc[N];
    int root;

    void tarjan(int u) {
        dfn[u] = low[u] = ++bcc_cnt, sta[++top] = u;
        if (u == root && hd[u] == 0) {
            dcc[++cnt].push_back(u);
            return;
        }
        int f = 0;
        for (int i = hd[u]; i; i = e[i].nt) {
            int v = e[i].to;
            if (!dfn[v]) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    if (++f > 1 || u != root) cut[u] = true;
                    cnt++;
                    do
                        dcc[cnt].push_back(sta[top--]);
                    while (sta[top + 1] != v);
                    dcc[cnt].push_back(u);
                }
            } else
                low[u] = min(low[u], dfn[v]);
        }
    }

    int main() {
        scanf("%d%d", &n, &m);
        int u, v;
        for (int i = 1; i <= m; i++) {
            scanf("%d%d", &u, &v);
            if (u != v) uadd(u, v);
        }
        for (int i = 1; i <= n; i++)
            if (!dfn[i]) root = i, tarjan(i);
```

```
        printf("%d\n", cnt);
        for (int i = 1; i <= cnt; i++) {
            printf("%llu ", dcc[i].size());
            for (int j = 0; j < dcc[i].size(); j++) printf("%d ", dcc[i][j]);
            printf("\n");
        }
        return 0;
    }
}
```

Dijkstra

1. 堆优化

```
class Dij
{
public:
    Dij(int _n): n(_n)
    {
        e.resize(n + 5);
        dis.resize(n + 5, 0x3f3f3f3f);
        vis.resize(n + 5, 0);
    }

    struct edge
    {
        int v, w;
    };
    struct node
    {
        int dis, u;
        bool operator>(const node &b) const { return dis > b.dis; }
    };
    vector<vector<edge>> e;
    vector<int> dis, vis;
    priority_queue<node, vector<node>, greater<node>> q;

    void dij(int s)
    {
        dis[s] = 0;
        q.push({0, s});
        while (!q.empty())
        {
            int u = q.top().u;
            q.pop();
            if (vis[u])
                continue;
            vis[u] = 1;
            for (auto x : e[u])
            {
                int v = x.v, w = x.w;
                if (dis[v] > dis[u] + w)
                {
                    dis[v] = dis[u] + w;
                    q.push({dis[v], v});
```

```
                }
            }
        }
    }

private:
    int n;
};
```

2. 直接遍历（m >> n）

```
class Dij_2
{
public:
    Dij_2(int _n): n(_n)
    {
        e.resize(n + 5);
        dis.resize(n + 5, 0x3f3f3f3f);
        vis.resize(n + 5, 0);
    }

    struct edge
    {
        int v, w;
    };
    struct node
    {
        int dis, u;
        bool operator>(const node &b) const { return dis > b.dis; }
    };
    vector<vector<edge>> e;
    vector<int> dis, vis;

    void dij2(int s)
    {
        dis.resize(n + 5, 0x3f3f3f3f);
        dis[s] = 0;
        for (int i = 1; i <= n; i++)
        {
            int u = 0, mind = 0x3f3f3f3f;
            for (int j = 1; j <= n; j++)
                if (!vis[j] && dis[j] < mind)
                    u = j, mind = dis[j];
            vis[u] = 1;
            for (auto x : e[u])
            {
                int v = x.v, w = x.w;
                if (dis[v] > dis[u] + w)
                    dis[v] = dis[u] + w;
            }
        }
    }

private:
    int n;
```

```
};
```

Floyd

```
#define MAXN 10005

struct edge
{
    int v, w;
};

vector<vector<edge>> e(MAXN);
int dp[MAXN][MAXN];

void Floyd(int n)
{
    memset(dp, 0x3f, sizeof(dp));
    for (int i = 1; i <= n; ++i)
        dp[i][i] = 0;
    for (int i = 1; i <= n; ++i)
        for (auto &x : e[i])
            dp[i][x.v] = x.w;
    for (int k = 1; k <= n; ++k)
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
                dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
}
```

Graph

1. 邻接表

```
struct edge
{
    int v, w;
};

vector<vector<edge>> e1(MAXN);
```

2. 邻接矩阵

```
int e2[MAXN][MAXN];
```

3. 链式前向星

```
int n, m, cnt = 0;

struct Edge
{
    int to;
    int w;
    int nxt;
};
```

```
vector<Edge> e(m * 2 + 10);
vector<int> head(n + 1, 0);

function<void(int u, int v, int w)> add = [&](int u, int v, int w)
{
    ++cnt;
    e[cnt].to = v;
    e[cnt].w = w;
    e[cnt].nxt = head[u];
    head[u] = cnt;
};

bool find_edge(int u, int v)
{
    for (int eg = head[u]; eg; eg = e[eg].nxt)
        if (e[eg].to == v)
            return true;
    return false;
}
```

Heavy_Path_Decomposition

1. 合并

```
namespace hsonMerge
{
    vector<vector<int>> to;
    vector<int> sz, hson, ans;
    multiset<int> book;
    ll sumi;

    void init() {}

    void dfs0(int u, int fa = -1)
    {
        int mx = 0, size = 1;
        for (auto v : to[u])
        {
            if (v != fa)
            {
                dfs0(v, u);
                size += sz[v];
                if (sz[v] > mx)
                    hson[u] = v, mx = sz[v];
            }
        }
        sz[u] = size;
    }

    void add(int u) {}
    void del(int u) {}

    void addsubtree(int u, int fa = -1)
    {
```

```
        add(u);
        for (auto v : to[u])
        {
            if (v != fa)
                addsubtree(v, u);
        }
    }
    void delsubtree(int u, int fa = -1)
    {
        del(u);
        for (auto v : to[u])
        {
            if (v != fa)
                delsubtree(v, u);
        }
    }

    void dfs(int u, int fa = -1, bool keep = -1)
    {
        // 先遍历轻儿子
        for (auto v : to[u])
        {
            if (v != fa && v != hson[u])
                dfs(v, u, 0);
        }
        // 最后遍历重儿子
        if (hson[u])
            dfs(hson[u], u, 1);
        add(u);
        for (auto v : to[u])
        {
            if (v != fa && v != hson[u])
                addsubtree(v, u);
        } // 重儿子的信息没有删除，因此不必再添加
        ans[u] = sumi;
        if (!keep)
        {
            init();
            sumi = 0;
        }
    }
}
```

2. 重链剖分+线段树维护

```
namespace PD
{
    template <typename T>
    class SegmentTree
    {
    private:
        int n;
        vector<T> tree, mark;
        void push_down(int p, int len)
        {
```

```
            mark[p * 2] += mark[p];
            mark[p * 2 + 1] += mark[p];
            tree[p * 2] += mark[p] * (len - len / 2);
            tree[p * 2 + 1] += mark[p] * (len / 2);
            mark[p] = 0;
        }

    public:
        vector<T> arr;
        SegmentTree(int _n) : n(_n)
        {
            tree.resize(4 * n + 10, 0);
            mark.resize(4 * n + 10, 0);
            arr.resize(n + 10);
        }
        void build(int l, int r, int p = 1)
        {
            if (l == r)
                tree[p] = arr[l];
            else
            {
                int middle = (l + r) / 2;
                build(l, middle, 2 * p);
                build(middle + 1, r, 2 * p + 1);
                tree[p] = tree[2 * p] + tree[2 * p + 1];
            }
        }
        void update(int l, int r, int cl, int cr, T d, int p = 1)
        {
            if (cr < l || cl > r)
                return;
            else if (l <= cl && cr <= r)
            {
                tree[p] += d * (cr - cl + 1);
                if (cr > cl)
                    mark[p] += d;
            }
            else
            {
                int middle = (cl + cr) / 2;
                push_down(p, cr - cl + 1);
                update(l, r, cl, middle, d, p * 2);
                update(l, r, middle + 1, cr, d, p * 2 + 1);
                tree[p] = tree[p * 2] + tree[p * 2 + 1];
            }
        }
        T query(int l, int r, int cl, int cr, int p = 1)
        {
            if (cl > r || cr < l)
                return 0;
            else if (cl >= l && cr <= r)
                return tree[p];
            else
            {
                int mid = (cl + cr) / 2;
```

```
                push_down(p, cr - cl + 1);
                return query(l, r, cl, mid, p * 2) + query(l, r, mid + 1, cr, p
* 2 + 1);
            }
        }
    };
    const int MAXN = 100005;
    int n;
    int tot = 0;
    vector<vector<int>> e(MAXN);
    vector<ll> val(MAXN);
    vector<int> fa(MAXN), hson(MAXN), sz(MAXN), dep(MAXN);
    vector<int> dfn(MAXN), rank(MAXN), rdfn(MAXN), top(MAXN);
    SegmentTree<ll> tree(MAXN);
    void dfs1(int cur, int last, int d = 1)
    {
        int maxi = 0;
        sz[cur] = 1;
        dep[cur] = d;
        for (auto v : e[cur])
        {
            if (v == last)
                continue;
            dfs1(v, cur, d + 1);
            fa[v] = cur;
            sz[cur] += sz[v];
            if (sz[v] > maxi)
                hson[cur] = v, maxi = sz[v];
        }
    }
    // initially t=root
    void dfs2(int cur, int last, int t)
    {
        top[cur] = t;
        dfn[cur] = ++tot;
        rank[tot] = cur;
        if (hson[cur])
            dfs2(hson[cur], cur, t);
        for (auto v : e[cur])
        {
            if (v == last || v == hson[cur])
                continue;
            dfs2(v, cur, v);
        }
        rdfn[cur] = tot;
    }
    int lca(int x, int y)
    {
        while (top[x] != top[y])
        {
            if (dep[top[x]] > dep[top[y]])
                x = fa[top[x]];
            else
                y = fa[top[y]];
        }
```

```
        return dep[x] > dep[y] ? y : x;
    }
    // build after e and val is filled
    void build(int root, int _n)
    {
        n = _n;
        dfs1(root, 0);
        dfs2(root, 0, root);
        for (int i = 1; i <= n; ++i)
            tree.arr[dfn[i]] = val[i];
        tree.build(1, n);
    }
    void update_path(int x, int y, ll d)
    {
        while (top[x] != top[y])
        {
            if (dep[top[x]] > dep[top[y]])
            {
                tree.update(dfn[top[x]], dfn[x], 1, n, d);
                x = fa[top[x]];
            }
            else
            {
                tree.update(dfn[top[y]], dfn[y], 1, n, d);
                y = fa[top[y]];
            }
        }
        if (dep[x] > dep[y])
            tree.update(dfn[y], dfn[x], 1, n, d);
        else
            tree.update(dfn[x], dfn[y], 1, n, d);
    }
    void update_tree(int x, ll d)
    {
        tree.update(dfn[x], rdfn[x], 1, n, d);
    }
    ll query_path(int x, int y)
    {
        ll ans = 0;
        while (top[x] != top[y])
        {
            if (dep[top[x]] > dep[top[y]])
            {
                ans += tree.query(dfn[top[x]], dfn[x], 1, n);
                x = fa[top[x]];
            }
            else
            {
                ans += tree.query(dfn[top[y]], dfn[y], 1, n);
                y = fa[top[y]];
            }
        }
        if (dep[x] > dep[y])
            ans += tree.query(dfn[y], dfn[x], 1, n);
        else
```

```
            ans += tree.query(dfn[x], dfn[y], 1, n);
        return ans;
    }
    ll query_tree(int x)
    {
        return tree.query(dfn[x], rdfn[x], 1, n);
    }
}
```

Heuristic_Merge

> 启发式合并：将小的集合合并到大的集合中，以降低平均时间复杂度，一般为 nlogn。

1. 树上启发式合并 (dsu on tree)

> 给出一棵 n 个节点以 1 为根的树，节点 u 的颜色为 c_u，现在对于每个结点 u 询问 u 子树里一共出现了多少种不同的颜色。(n <= 10^5)

```
namespace solve
{
    const int N = 1e5 + 5;
    int n;
    // g[u]: 存储与 u 相邻的结点
    vector<int> g[N];
    // sz: 子树大小
    // big: 重儿子
    // col: 结点颜色
    // L[u]: 结点 u 的 DFS 序
    // R[u]: 结点 u 子树中结点的 DFS 序的最大值
    // Node[i]: DFS 序为 i 的结点
    // ans: 存答案
    // cnt[i]: 颜色为 i 的结点个数
    // totColor: 目前出现过的颜色个数
    int sz[N], big[N], col[N], L[N], R[N], Node[N], totdfn;
    int ans[N], cnt[N], totColor;
    void add(int u)
    {
        if (cnt[col[u]] == 0)
            ++totColor;
        cnt[col[u]]++;
    }
    void del(int u)
    {
        cnt[col[u]]--;
        if (cnt[col[u]] == 0)
            --totColor;
    }
    int getAns() { return totColor; }
    void dfs0(int u, int p)
    {
        L[u] = ++totdfn;
        Node[totdfn] = u;
        sz[u] = 1;
        for (int v : g[u])
            if (v != p)
```

```
        {
            dfs0(v, u);
            sz[u] += sz[v];
            if (!big[u] || sz[big[u]] < sz[v])
                big[u] = v;
        }
    R[u] = totdfn;
}
void dfs1(int u, int p, bool keep)
{
    // 计算轻儿子的答案
    for (int v : g[u])
        if (v != p && v != big[u])
        {
            dfs1(v, u, false);
        }
    // 计算重儿子答案并保留计算过程中的数据（用于继承）
    if (big[u])
    {
        dfs1(big[u], u, true);
    }
    for (int v : g[u])
        if (v != p && v != big[u])
        {
            // 子树结点的 DFS 序构成一段连续区间，可以直接遍历
            for (int i = L[v]; i <= R[v]; i++)
            {
                add(Node[i]);
            }
        }
    add(u);
    ans[u] = getAns();
    if (keep == false)
    {
        for (int i = L[u]; i <= R[u]; i++)
        {
            del(Node[i]);
        }
    }
}
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &col[i]);
    for (int i = 1; i < n; i++)
    {
        int u, v;
        scanf("%d%d", &u, &v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs0(1, 0);
    dfs1(1, 0, false);
    for (int i = 1; i <= n; i++)
```

```
            printf("%d%c", ans[i], " \n"[i == n]);
        return 0;
    }
}
```

Hungarian_Algorithm

> Hungarian Algorithm or Kuhn–Munkres Algorithm
>
> index starts from 0
>
> if perfect is true and output is about -1e14, there is no solution

```
template <typename T>
class Hungarian
{
private:
    bool perfect;
    int n;
    int org_n;
    int org_m;
    T inf;
    vector<int> pre;
    vector<bool> visx;
    vector<bool> visy;
    vector<T> slack;
    vector<T> lx;
    vector<T> ly;
    queue<int> q;
    vector<vector<T>> g;

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

    void bfs(int i)
    {
        while (!q.empty())
            q.pop();
        q.push(i);
        visx[i] = true;
        while (true)
```

```
        {
            while (!q.empty())
            {
                int u = q.front();
                q.pop();
                for (int v = 0; v < n; v++)
                {
                    if (visy[v])
                        continue;
                    T delta = lx[u] + ly[v] - g[u][v];
                    if (slack[v] >= delta)
                    {
                        pre[v] = u;
                        if (delta)
                            slack[v] = delta;
                        else if (check(v))
                            return;
                    }
                }
            }
            T a = inf;
            for (int j = 0; j < n; j++)
            {
                if (!visy[j])
                    a = min(a, slack[j]);
            }
            for (int j = 0; j < n; j++)
            {
                if (visx[j])
                    lx[j] -= a;
                if (visy[j])
                    ly[j] += a;
                else
                    slack[j] -= a;
            }
            for (int j = 0; j < n; j++)
            {
                if (!visy[j] && slack[j] == 0 && check(j))
                    return;
            }
        }
    }

public:
    T res;
    vector<int> matchx;
    vector<int> matchy;

    Hungarian(int _n, int _m, bool _perfect = false)
    {
        org_n = _n;
        org_m = _m;
        perfect = _perfect;
        n = max(_n, _m);
        inf = numeric_limits<T>::max();
```

```
        res = 0;
        if (perfect)
            g = vector<vector<T>>(n, vector<T>(n, -1e14));
        else
            g = vector<vector<T>>(n, vector<T>(n, 0));
        matchx = vector<int>(n, -1);
        matchy = vector<int>(n, -1);
        pre = vector<int>(n);
        visx = vector<bool>(n);
        visy = vector<bool>(n);
        lx = vector<T>(n, -inf);
        ly = vector<T>(n);
        slack = vector<T>(n);
    }

    void addEdge(int u, int v, int w)
    {
        if (perfect)
            g[u][v] = w;
        else
            g[u][v] = max(w, 0);
    }

    T solve()
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                lx[i] = max(lx[i], g[i][j]);
        for (int i = 0; i < n; i++)
        {
            fill(slack.begin(), slack.end(), inf);
            fill(visx.begin(), visx.end(), false);
            fill(visy.begin(), visy.end(), false);
            bfs(i);
        }
        if (perfect)
        {
            for (int i = 0; i < n; i++)
            {
                res += g[i][matchx[i]];
            }
        }
        else
        {
            for (int i = 0; i < n; i++)
            {
                if (g[i][matchx[i]] > 0)
                    res += g[i][matchx[i]];
                else
                    matchx[i] = -1;
            }
        }
        return res;
    }
};
```

Johnson

```
#define MAXN 10005
const ll inf = 1e9;

struct edge
{
    ll v, w, next;
};

struct node
{
    ll dis, id;
    bool operator>(const node &b) const { return dis > b.dis; }
};

vector<edge> e(MAXN);
ll head[5005], vis[5005], t[5005];
ll cnt;
ll h[5005], dis[5005];

void addedge(ll u, ll v, ll w)
{
    e[++cnt].v = v;
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt;
}

bool spfa(ll n, ll s)
{
    queue<ll> q;
    memset(h, 63, sizeof(h));
    h[s] = 0, vis[s] = 1;
    q.push(s);
    while (!q.empty())
    {
        ll u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i; i = e[i].next)
        {
            ll v = e[i].v;
            if (h[v] > h[u] + e[i].w)
            {
                h[v] = h[u] + e[i].w;
                if (!vis[v])
                {
                    vis[v] = 1;
                    q.push(v);
                    t[v]++;
                    if (t[v] == n + 1)
                        return false;
                }
            }
```

```
        }
    }
    return true;
}

void dijkstra(ll n, ll s)
{
    priority_queue<node, vector<node>, greater<node>> q;
    for (int i = 1; i <= n; i++)
        dis[i] = inf;
    memset(vis, 0, sizeof(vis));
    dis[s] = 0;
    q.push({0, s});
    while (!q.empty())
    {
        ll u = q.top().id;
        q.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        for (int i = head[u]; i; i = e[i].next)
        {
            ll v = e[i].v;
            if (dis[v] > dis[u] + e[i].w)
            {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v])
                    q.push({dis[v], v});
            }
        }
    }
}

int main()
{
    // clock_t st = clock(), ed;
    ios::sync_with_stdio(0);
    cin.tie(0);
    // cout << setprecision(15) << fixed;
    ll n, m;
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        ll u, v, w;
        cin >> u >> v >> w;
        addedge(u, v, w);
    }
    for (int i = 1; i <= n; i++)
        addedge(0, i, 0);
    if (!spfa(n, 0))
    {
        cout << -1 << endl;
        return 0;
    }
    for (int u = 1; u <= n; u++)
```

```
        for (int i = head[u]; i; i = e[i].next)
            e[i].w += h[u] - h[e[i].v];
    for (int i = 1; i <= n; i++)
    {
        dijkstra(n, i);
        ll ans = 0;
        for (int j = 1; j <= n; j++)
        {
            if (dis[j] == inf)
                ans += j * inf;
            else
                ans += j * (dis[j] + h[j] - h[i]);
        }
        cout << ans << endl;
    }
    // ed = clock();
    // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
    // cout << "Total time: " << endtime << endl;
    return 0;
}
```

Kosaraju

> g 是原图，g2 是反图

```
#define MAXN 100005

vector<bool> vis(MAXN);
vector<int> color(MAXN), s;
int n, sccCnt;

vector<vector<int>> g(MAXN), g2(MAXN);

void dfs1(int u)
{
    vis[u] = true;
    for (int v : g[u])
        if (!vis[v])
            dfs1(v);
    s.push_back(u);
}

void dfs2(int u)
{
    color[u] = sccCnt;
    for (int v : g2[u])
        if (!color[v])
            dfs2(v);
}

void kosaraju()
{
    sccCnt = 0;
    for (int i = 1; i <= n; ++i)
        if (!vis[i])
```

```
            dfs1(i);
    for (int i = n; i >= 1; --i)
        if (!color[s[i]])
        {
            ++sccCnt;
            dfs2(s[i]);
        }
}
```

Kruskal

```
class DSU
{
private:
    int n;
    vector<int> fa, sz;

public:
    DSU(int _n) : n(_n)
    {
        fa.resize(n + 1);
        sz.resize(n + 1);
        for (int i = 1; i <= n; ++i)
        {
            fa[i] = i;
            sz[i] = 1;
        }
    }
    int find(int x)
    {
        if (fa[x] == x)
            return x;
        else
        {
            fa[x] = find(fa[x]);
            return fa[x];
        }
    }
    void merge(int x, int y)
    {
        int xfa = find(x), yfa = find(y);
        if (xfa == yfa)
            return;
        if (sz[xfa] <= sz[yfa])
        {
            sz[yfa] += sz[xfa];
            fa[xfa] = yfa;
        }
        else
        {
            sz[xfa] += sz[yfa];
            fa[yfa] = xfa;
        }
    }
    int get_size(int x)
```

```
    {
        return sz[find(x)];
    }
};

class Kruskal
{
private:
    int n, m;
    DSU dsu;

public:
    struct edge
    {
        int u, v, w;
    };
    static bool cmp(edge a, edge b)
    {
        return a.w < b.w;
    }
    vector<edge> e;
    Kruskal(int _n, int _m) : n(_n), m(_m), dsu(_n)
    {
        e.resize(m + 5);
    }
    void solve()
    {
        sort(e.begin() + 1, e.begin() + 1 + m, cmp);
        for (int i = 1; i <= m; ++i)
        {
            if (dsu.find(e[i].u) == dsu.find(e[i].v))
                continue;
            else
            {
                dsu.merge(e[i].u, e[i].v);
                cout << e[i].u << " " << e[i].v << " " << e[i].w << endl;
            }
        }
    }
};
```

LCA

> Binary Lifting

```
class LCA
{
private:
    int n;
    vector<int> dep;
    vector<vector<int>> fa;

public:
    vector<vector<int>> v;
    LCA(int _n) : n(_n)
```

```
    {
        v.resize(n + 5);
        dep.resize(n + 5, 0);
        fa = vector<vector<int>>(n + 5, vector<int>(31, 0));
    }
    void init(int cur = 1, int fanode = 0)
    {
        fa[cur][0] = fanode;
        dep[cur] = dep[fanode] + 1;
        for (int i = 1; i < 31; ++i)
            fa[cur][i] = fa[fa[cur][i - 1]][i - 1];
        for (int i = 0; i < v[cur].size(); ++i)
        {
            if (v[cur][i] == fanode)
                continue;
            init(v[cur][i], cur);
        }
    }
    int lca(int x, int y)
    {
        if (dep[x] > dep[y])
            swap(x, y);
        int tmp = dep[y] - dep[x];
        for (int j = 0; tmp; ++j, tmp >>= 1)
            if (tmp & 1)
                y = fa[y][j];
        if (y == x)
            return x;
        for (int j = 30; j >= 0 && y != x; --j)
        {
            if (fa[x][j] != fa[y][j])
            {
                x = fa[x][j];
                y = fa[y][j];
            }
        }
        return fa[x][0];
    }
};
```

MaxFlow

1. EK

> O(V*E^2)
>
> 未处理重边

```
struct Edge
{
    int to;
    int w;
    int nxt;
};
```

```
void solve()
{
    int m, n, u, v, w;
    cin >> m >> n;
    int s = 1, t = n;
    vector<Edge> e(m * 2 + 10);
    vector<int> head(n + 1, 0), last(n + 1), flow(n + 1);
    int cnt = 1;
    function<void(int u, int v, int w)> add = [&](int u, int v, int w)
    {
        ++cnt;
        e[cnt].to = v;
        e[cnt].w = w;
        e[cnt].nxt = head[u];
        head[u] = cnt;
    };
    for (int i = 1; i <= m; ++i)
    {
        cin >> u >> v >> w;
        add(u, v, w);
        add(v, u, 0);
    }
    function<bool()> bfs = [&]()->bool
    {
        queue<int> q;
        flow[s] = 1e9;
        fill(all(last), -1);
        q.push(s);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int eg = head[u]; eg; eg = e[eg].nxt)
            {
                int to = e[eg].to;
                int w = e[eg].w;
                if (last[to] == -1 && w > 0)
                {
                    last[to] = eg;
                    flow[to] = min(flow[u], w);
                    q.push(to);
                    if (to == t)
                        return true;
                }
            }
        }
        return last[t] != -1;
    };
    int maxflow = 0;
    while (bfs())
    {
        maxflow += flow[t];
        for (int x = t; x != s; x = e[last[x] ^ 1].to)
        {
```

```
            e[last[x]].w -= flow[t];
            e[last[x] ^ 1].w += flow[t];
        }
    }
    cout << maxflow << endl;
}
```

2. Dinic

> O(V^2*E) 跑不满
>
> 在单位容量的网络上，Dinic 算法的总时间复杂度是 O(E*min(E^0.5, V^0.66))
>
> 在单位容量的网络上，如果除源汇点外每个结点 u 都满入度出度为 1，Dinic 算法的总时间复杂度是 O(E*V^0.5)

```
namespace Dinic
{
    const ll N = 1e4, INF = 1e9;

    struct MF {
        struct edge {
            int v, nxt, cap, flow;
        } e[N];

        int fir[N], cnt = 0;

        int n, S, T;
        ll maxflow = 0;
        int dep[N], cur[N];

        void init() {
            memset(fir, -1, sizeof fir);
            cnt = 0;
        }

        void addedge(int u, int v, int w) {
            e[cnt] = {v, fir[u], w, 0};
            fir[u] = cnt++;
            e[cnt] = {u, fir[v], 0, 0};
            fir[v] = cnt++;
        }

        bool bfs() {
            queue<int> q;
            memset(dep, 0, sizeof(int) * (n + 1));

            dep[S] = 1;
            q.push(S);
            while (q.size()) {
                int u = q.front();
                q.pop();
                for (int i = fir[u]; ~i; i = e[i].nxt) {
                    int v = e[i].v;
                    if ((!dep[v]) && (e[i].cap > e[i].flow)) {
                        dep[v] = dep[u] + 1;
```

```
                    q.push(v);
                }
            }
        }
        return dep[T];
    }

    int dfs(int u, int flow) {
        if ((u == T) || (!flow)) return flow;

        int ret = 0;
        for (int& i = cur[u]; ~i; i = e[i].nxt) {
            int v = e[i].v, d;
            if ((dep[v] == dep[u] + 1) &&
                (d = dfs(v, min(flow - ret, e[i].cap - e[i].flow)))) {
                ret += d;
                e[i].flow += d;
                e[i ^ 1].flow -= d;
                if (ret == flow) return ret;
            }
        }
        return ret;
    }

    void dinic() {
        while (bfs()) {
            memcpy(cur, fir, sizeof(int) * (n + 1));
            maxflow += dfs(S, INF);
        }
    }
} mf;

void solve()
{
    int n, m, s, t, x, y, w;
    cin >> n >> m >> s >> t;
    mf.init();
    mf.n = n;
    mf.S = s;
    mf.T = t;
    for (int i = 1; i <= m; ++i)
    {
        cin >> x >> y >> w;
        mf.addedge(x, y, w);
    }
    mf.dinic();
    cout << mf.maxflow << endl;
}
}
```

Prim

```
#define MAXN 100005

int k, n, m, cnt, sum, ai, bi, ci, head[MAXN], dis[MAXN], vis[MAXN];

struct Edge
{
    int v, w, next;
} e[MAXN];

void add(int u, int v, int w)
{
    e[++k].v = v;
    e[k].w = w;
    e[k].next = head[u];
    head[u] = k;
}

priority_queue<pii, vector<pii>, greater<pii>> q;

void prim()
{
    memset(dis, 127, sizeof(dis));
    memset(head, -1, sizeof(head));
    cin >> n >> m;
    for (int i = 1; i <= m; ++i)
    {
        cin >> ai >> bi >> ci;
        add(ai, bi, ci);
        add(bi, ai, ci);
    }
    dis[1] = 0;
    q.push(make_pair(0, 1));
    while (!q.empty() && cnt < n)
    {
        int d = q.top().first, u = q.top().second;
        q.pop();
        if (vis[u])
            continue;
        ++cnt;
        sum += d;
        vis[u] = 1;
        for (int i = head[u]; i != -1; i = e[i].next)
            if (e[i].w < dis[e[i].v])
                dis[e[i].v] = e[i].w, q.push(make_pair(dis[e[i].v], e[i].v));
    }
    cout << sum << endl;
}
```

Tarjan

```
#define N 100005

vector<vector<int>> e(N);

int dfn[N], low[N], dfncnt, s[N], in_stack[N], tp;
int scc[N], sc;
int sz[N];

void tarjan(int u)
{
    low[u] = dfn[u] = ++dfncnt, s[++tp] = u, in_stack[u] = 1;
    for (auto v : e[u])
    {
        if (!dfn[v])
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (in_stack[v])
            low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u])
    {
        ++sc;
        while (s[tp] != u)
        {
            scc[s[tp]] = sc;
            ++sz[sc];
            in_stack[s[tp]] = 0;
            --tp;
        }
        scc[s[tp]] = sc;
        ++sz[sc];
        in_stack[s[tp]] = 0;
        --tp;
    }
}
```

Topology_Sort

> Kahn
>
> use priority_queue for lexicographical order

```
#define MAXN 0x3f3f3f
int deg[MAXN], A[MAXN];
vector<int> edges[MAXN];
bool toposort(int n)
{
    int cnt = 0;
    queue<int> q;
    for (int i = 1; i <= n; ++i)
```

```
        if (deg[i] == 0)
            q.push(i);
    while (!q.empty())
    {
        int t = q.front();
        q.pop();
        A[cnt++] = t;
        for (auto to : edges[t])
        {
            deg[to]--;
            if (deg[to] == 0) // in is zero
                q.push(to);
        }
    }
    return cnt == n;
}
```

7. Math

Cantor_Expansion

1. 康托展开

> 用于求某排列的排名
>
> $rk=1+\sum_{i=1}^{n}{A[i]×(n−i)!}$
>
> 其中 $A[i]$ 为 $\sum_{j=i}^{n}{[a[j]<a[i]]}$，朴素求 A 是 $O(n^2)$，可以考虑树状数组优化。

```
class biTree
{
private:
    int MAXN;
    vector<ll> tree;
    ll lowbit(ll x)
    {
        return x & (-x);
    }

public:
    biTree(int _MAXN = 100005) : MAXN(_MAXN)
    {
        tree.resize(MAXN);
    }
    void update(ll index, ll x)
    {
        for (ll pos = index; pos < MAXN; pos += lowbit(pos))
            tree[pos] += x;
    }
    ll query(ll n)
    {
        ll sum = 0;
        for (ll i = n; i; i -= lowbit(i))
            sum += tree[i];
```

```
        return sum;
    }
    // (a, b]
    ll query(ll a, ll b)
    {
        return query(b) - query(a);
    }
};

ll cantor()
{
    ll ans = 1, MOD = 998244353;

    int n;
    cin >> n;
    vector<ll> arr(n + 1), A(n + 1), fac(n + 1);
    biTree tree(n);
    fac[0] = 1;
    for (int i = 1; i <= n; ++i)
        cin >> arr[i], fac[i] = fac[i - 1] * i % MOD;

    for (int i = n; i >= 1; --i)
    {
        tree.update(arr[i], 1);
        A[i] = tree.query(arr[i] - 1);
    }
    for (int i = 1; i <= n; ++i)
    {
        ans += A[i] * fac[n - i];
        ans %= MOD;
    }

    return ans;
}
```

2. 逆康托展开

> 类似于进制转换，不断 $\mod (n-i)!$, $/(n-1)!$ 就可以得到 A 数组，然后就可以还原出原排列。

```
vector<int> incantor(int x, int n) {
    x--;
    vector<int> res(n), fac(n);
    fac[0] = 1;
    for (int i = 1; i < n; ++i)
        fac[i] = fac[i - 1] * i;
    bool st[10] = {0};
    for (int i = 0; i < n; ++i)
    {
        int cnt = x / fac[n - i - 1]; // 比a[i]小且没有出现过的数的个数
        x %= fac[n - i - 1];
        for (int j = 1;j <= n; ++j)
        {
            if (st[j])
                continue;
```

```
            if (!cnt)
            {
                st[j] = 1;
                res[i] = j;
                break;
            }
            cnt--;
        }
    }
    return res;
}
```

Chinese_Remainder_Theorem

> x == m1 (mod a1)
>
> x == m2 (mod a2)
>
> ...

1. a1, a2, a3 ... 保证互质

```
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if (b == 0)
    {
        x = 1, y = 0;
        return a;
    }
    ll d= exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

void CRT()
{
    ll n, M = 1, ans = 0;
    cin >> n;
    vector<ll> arr(n + 1), mrr(n + 1);
    for (int i = 1; i <= n; ++i)
    {
        cin >> arr[i] >> mrr[i];
        M *= arr[i];
    }
    for (int i = 1; i <= n; ++i)
    {
        ll Mi = M / arr[i], x, y;
        ll d = exgcd(Mi, arr[i], x, y);
        ans = (ans + mrr[i] * Mi % M * x) % M;
    }
    cout << (ans % M + M) % M << endl;
}
```

2. a1, a2, a3 ... 不保证互质

```
void exCRT()
{
    int n;
    cin >> n;
    ll a1, m1, a2, m2, k1, k2;
    cin >> a1 >> m1;
    for (int i = 1; i < n; ++i)
    {
        cin >> a2 >> m2;
        ll d = exgcd(a1, a2, k1, k2);
        if ((m2 - m1) % d)
        {
            cout << -1 << endl;
            return;
        }
        k1 *= (m2 - m1) / d;
        ll t = a2 / d;
        k1 = (k1 % t + t) % t;
        m1 = a1 * k1 + m1;
        a1 = a1 / d * a2;
    }
    cout << (m1 % a1 + a1) % a1 << endl;
}
```

Combination

1. 朴素版本

```
ll comb(ll n, ll k)
{
    ll res = 1;
    for (ll i = 1; i <= k; ++i)
    {
        res *= n - i + 1;
        res /= i;
        // use inverse if you need modulo
    }
    return res;
}
```

2. 线性逆元版本

> C_{x}^{y}

```
namespace comb
{
    int n, p;
    vector<int> inv, fac, finv;
    void Preprocess(int _n, int _p)
    {
        n = _n;
        p = _p;
        inv.resize(n + 5);
        fac.resize(n + 5);
```

```
        finv.resize(n + 5);
        inv[1] = 1;
        for (int i = 2; i <= n + 1; ++i)
            inv[i] = (ll)(p - p / i) * inv[p % i] % p;
        fac[0] = 1;
        for (int i = 1; i <= n + 1; ++i)
            fac[i] = (ll)fac[i - 1] * i % p;
        finv[0] = 1;
        for (int i = 1; i <= n + 1; ++i)
            finv[i] = (ll)finv[i - 1] * inv[i] % p;
    }
    int C(int x, int y) { return (ll)fac[x] * finv[y] % p * finv[x - y] % p; }
}
```

3. 杨辉三角形

```
int c[2005][2005];

void build(int p)
{
    c[1][1] = 1;
    for (int i = 0; i <= 2000; i++)
        c[i][0] = 1;
    for (int i = 2; i <= 2000; i++)
        for (int j = 1; j <= i; j++)
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % p;
}
```

Euler's_Totient_Function

1. 朴素版本

```
int Euler_phi(int n)
{
    ll ans = n;
    for (ll i = 2; i * i <= n; ++i)
        if (n % i == 0)
        {
            ans = ans / i * (i - 1);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        ans = ans / n * (n - 1);
    return ans;
}
```

2. 筛

```
vector<int> primes, phi;
vector<bool> nop;
void Euler(int n)
{
    phi.resize(n + 10, 0);
```

```
    nop.resize(n + 10, 0);
    phi[1] = 1;
    for (int i = 2; i <= n; ++i)
    {
        if (!nop[i])
        {
            primes.pb(i);
            phi[i] = i - 1;
        }
        for (auto x: primes)
        {
            if (x * i > n)
                break;
            nop[i * x] = true;
            if (i % x == 0)
            {
                phi[i * x] = phi[i] * x;
                break;
            }
            phi[i * x] = phi[i] * (x - 1);
        }
    }
}
```

Exgcd

> ax + by = gcd(a, b)

1. version 1

```
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if (b == 0)
    {
        x = 1, y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

2. version 2

```
int exgcd2(int a, int b, int &x, int &y)
{
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1)
    {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
```

GCD

1. 减法

```
ll gcd1(ll a, ll b)
{
    // if (a == 1 || b == 1)
    //     return 1;
    // if (a == b)
    //     return a;
    // if (a > b)
    //     return gcd(a - b, b);
    // return gcd(a, b - a);
    while (a != b)
    {
        if (a == 1 || b == 1)
            return 1;
        if (a > b)
            a -= b;
        else
            b -= a;
    }
    return a;
}
```

2. Euclid

```
ll gcd2(ll a, ll b)
{
    while (b != 0)
    {
        ll tmp = a;
        a = b;
        b = tmp % b;
    }
    return a;
}
```

3. multiple numbers

```
ll gcd3(vector<ll> &v)
{
    ll res = v[0];
    for (int i = 1; i < v.size(); i++)
        res = gcd2(res, v[i]);
    return res;
}
```

Gauss_Jordan_Elimination

> 0: OK
>
> 1: Infinite group solutions
>
> 2: No solution

```
const double eps = 1e-6;

int Guass(int n, vector<vector<double>> &mat)
{
    int r = 1, c = 1;
    for (; c <= n; ++c)
    {
        int maxi = r;
        for (int i = r; i <= n; ++i)
            if (fabs(mat[i][c]) > fabs(mat[maxi][c]))
                maxi = i;
        if (fabs(mat[maxi][c]) < eps)
            continue;
        for (int j = c; j <= n + 1; ++j)
            swap(mat[r][j], mat[maxi][j]);
        for (int j = n + 1; j >= c; --j)
            mat[r][j] /= mat[r][c];
        for (int i = r + 1; i <= n; ++i)
            if (fabs(mat[i][c]) > eps)
                for (int j = n + 1; j >= c; --j)
                    mat[i][j] -= mat[i][c] * mat[r][j];
        ++r;
    }
    if (r <= n)
    {
        for (int i = r; i <= n; ++i)
            if (fabs(mat[i][n + 1]) > eps)
                return 2;
        return 1;
    }
    for (int i = n - 1; i >= 1; --i)
        for (int j = i + 1; j <= n; ++j)
            mat[i][n + 1] -= mat[i][j] * mat[j][n + 1];
    return 0;
}
```

LCM

1. 两个数

```
ll lcm1(ll a, ll b)
{
    return a / gcd(a, b) * b;
}
```

2. multiple numbers

```
ll lcm2(vector<ll> a)
{
    ll res = 1;
    for (auto x : a)
        res = lcm1(res, x);
    return res;
}
```

Lucas

```
namespace comb
{
    int n, p;
    vector<int> inv, fac, finv;
    void Preprocess(int _n, int _p)
    {
        n = _n;
        p = _p;
        inv.resize(n + 5);
        fac.resize(n + 5);
        finv.resize(n + 5);
        inv[1] = 1;
        for (int i = 2; i <= n + 1; ++i)
            inv[i] = (ll)(p - p / i) * inv[p % i] % p;
        fac[0] = 1;
        for (int i = 1; i <= n + 1; ++i)
            fac[i] = (ll)fac[i - 1] * i % p;
        finv[0] = 1;
        for (int i = 1; i <= n + 1; ++i)
            finv[i] = (ll)finv[i - 1] * inv[i] % p;
    }
    int C(int x, int y) { return (ll)fac[x] * finv[y] % p * finv[x - y] % p; }
}

ll Lucas(ll n, ll m, ll p)
{
    if (m == 0)
        return 1;
    if (n % p < m % p)
        return 0;
    return (comb::C(n % p, m % p) * Lucas(n / p, m / p, p)) % p;
}
```

Modular_Multiplicative_Inverse

1. with exgcd

> a * x === 1 (mod b) => a * x + b * y == 1
>
> ans = (a * b * c) % p;
>
> to get rid of a
>
> ans = ans * x % p;

```
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if (!b)
    {
        x = 1;
        y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - (a / b) * y;
    return d;
}

ll inv(ll a, ll b)
{
    ll x, y;
    ll d = exgcd(a, b, x, y);
    // smallest x
    if (d == 1)
        return (x % b + b) % b;
    return -1;
}
```

2. if b is prime

> according to Fermat's little theorem

```
ll inv(ll a, ll b)
{
    ll ans = 1, M = b;
    b -= 2;
    while (b)
    {
        if (b & 1)
            ans = (a * ans) % M;
        a = (a * a) % M;
        b >>= 1;
    }
    return ans;
}
```

Modulo_For_Rational_Numbers

> a / b

```
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll t = x;
    x = y;
    y = t - a / b * y;
    return d;
}

ll mod(ll a, ll b, ll p)
{
    ll x, y;
    ll d = exgcd(b, p, x, y);
    x = (x % p + p) % p;
    return x * a % p;
}

// for big numbers

ll getint(ll p)
{
    ll res = 0, ch = getchar();
    while (!isdigit(ch) && ch != EOF)
        ch = getchar();
    while (isdigit(ch))
    {
        res = (res << 3) + (res << 1) + (ch - '0');
        res %= p;
        ch = getchar();
    }
    return res;
}
```

Numbers

1. 卡特兰数（Catalan）

> H_n（下标从 0 开始）：1,1,2,5,14,42,132 ...
>
> $H_n = H_{n-1}\times(4\times n-2)/(n+1)$

> 在圆上选择 2n 个点，将这些点成对连接起来使得所得到的 n 条线段不相交的方法数?
>
> 对角线不相交的情况下，将一个凸多边形区域分成三角形区域的方法数?
>
> n 个结点可构造多少个不同的二叉树?

> 有一个大小为 n * n 的方格图左下角为 (0, 0) 右上角为 (n, n)，从左下角开始每次都只能向右或者向上走一单位，不走到对角线 y=x 上方（但可以触碰）的情况下到达右上角有多少可能的路径?

2. 第二类斯特林数（Stirling Number）

> S(n,k)，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数。

```
/*
n/k 0   1   2   3   4   5
0   1
1   0   1
2   0   1   1
3   0   2   3   1
4   0   6   11  6   1
5   0   24  50  35  10  1
*/
```

$S(n, k) = S(n-1,k-1)+k\times S(n-1,k)$

3. 错位排列（derangement）

> D_n（下标从 0 开始）: 0,1,2,9,44,265 ...
>
> $D_n=(n-1)\times(D{*n-1*}+D{n-2})$

4. 斐波那契数列

> F_n（下标从 0 开始）: 0,1,1,2,3,5,8,13 ...
>
> $F_n=F{*n-1*}+F{n-2}$
>
> 使用矩阵快速幂计算:
>
> 由 $[F{*n-1*}, F_n]=[F{n-2}, F_{n-1}]\times((0,1)^T,(1,1)^T)$
>
> 得 $[F_n, F_{n+1}]=[F_0, F_1]*P^n$

Pigeonhole_Principle

> 将 n+1 个物体，划分为 n 组，那么有至少一组有两个（或以上）的物体。
>
> 把 mn - 1 个物体放入 n 个抽屉中，其中必有一个抽屉中至多有 m—1 个物体

> https://ac.nowcoder.com/acm/problem/257495
>
> 自动计算机
>
> 给定一个长度为 m 的序列 a，分别为 a_0, a_1, ... , a_m-1，一开始你拥有一个初始数为 x，
>
> 你的第 i 次操作会使得 x=(x+a_((i−1)%m))%n，请问至少多少次操作才能使 x 变成 0。
>
> 如果无论多少次操作均无法使 x=0，那么输出 −1。
>
> 由抽屉原理，n 轮操作后 x 一定循环。
>
> 而每一轮中的 m 次操作可以预处理出进行 i 次操作后在模意义下的偏差值。

```
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
```

```
    // cout << setprecision(15) << fixed;
    ll n, m, x;
    cin >> n >> m >> x;
    vector<ll> pre(m + 1, 0), arr(m + 1);
    map<ll, ll> book;
    for (int i = 1; i <= m; ++i)
    {
        cin >> arr[i];
        pre[i] = (pre[i - 1] + arr[i]) % n;
        if (book.find(pre[i]) == book.end())
            book[pre[i]] = i;
    }
    ll ans = -1;
    for (int i = 1; i <= n; ++i)
    {
        ll goal = n - x;
        if (x == 0)
        {
            ans = (i - 1) * m;
            break;
        }
        else
        {
            if (book.find(goal) != book.end())
            {
                ans = (i - 1) * m + book[goal];
                break;
            }
        }
        x = (x + pre[m]) % n;
    }
    cout << ans << endl;
    return 0;
}
```

Pollard_Rho

> for 1 ~ (1 << 64)
>
> max fac

```
namespace millerRabin
{
    ll qpow(ll a, ll t, ll mod)
    {
        a %= mod;
        ll res = 1;
        while (t)
        {
            if (t & 1)
                res = (__int128)res * a % mod;
            a = (__int128)a * a % mod;
            t >>= 1;
        }
        return res;
```

```
    }

    const int primebook[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

    bool isPrime(ll n)
    {
        if (n < 3 || n % 2 == 0)
            return n == 2;
        if (n <= 40)
        {
            for (ll a : primebook)
                if (a == n)
                    return 1;
            return 0;
        }
        ll u = n - 1, t = 0;
        while (u % 2 == 0)
            u /= 2, ++t;
        for (ll a : primebook)
        {
            ll v = qpow(a, u, n);
            if (v == 1)
                continue;
            ll s;
            for (s = 0; s < t; ++s)
            {
                if (v == n - 1)
                    break;
                v = (__int128)v * v % n;
            }
            if (s == t)
                return 0;
        }
        return 1;
    }
}

namespace pollardRho
{
    ll max_factor;
    ll Pollard_Rho(ll x)
    {
        ll s = 0, t = 0;
        ll c = (ll)rand() % (x - 1) + 1;
        int step = 0, goal = 1;
        ll val = 1;
        for (goal = 1;; goal *= 2, s = t, val = 1)
        {
            for (step = 1; step <= goal; ++step)
            {
                t = ((__int128)t * t + c) % x;
                val = (__int128)val * abs(t - s) % x;
                if ((step % 127) == 0)
                {
                    ll d = gcd(val, x);
```

```
                    if (d > 1)
                        return d;
                }
            }
            ll d = gcd(val, x);
            if (d > 1)
                return d;
        }
    }

    void func(ll x)
    {
        if (x <= max_factor || x < 2)
            return;
        if (millerRabin::isPrime(x))
        {
            max_factor = max(max_factor, x);
            return;
        }
        ll p = x;
        while (p >= x)
            p = Pollard_Rho(x);
        while ((x % p) == 0)
            x /= p;
        func(x), func(p);
    }

    ll fac(ll x)
    {
        srand((unsigned)time(0));
        max_factor = 0;
        func(x);
        return max_factor;
    }
}
```

Poly

1. 求原根

```
namespace Root
{
    ll pow(const ll x, const ll n, const ll p)
    {
        ll ans = 1;
        for (ll num = x, tmp = n; tmp; tmp >>= 1, num = num * num % p)
            if (tmp & 1)
                ans = ans * num % p;
        return ans;
    }
    ll root(const ll p)
    {
        for (int i = 2; i <= p; i++)
        {
            int x = p - 1;
```

```
            bool flag = false;
            for (int k = 2; k * k <= p - 1; k++)
                if (x % k == 0)
                {
                    if (pow(i, (p - 1) / k, p) == 1)
                    {
                        flag = true;
                        break;
                    }
                    while (x % k == 0)
                        x /= k;
                }
            if (!flag && (x == 1 || pow(i, (p - 1) / x, p) != 1))
            {
                return i;
            }
        }
        throw;
    }
}
```

2. Poly

```
typedef long long ll;
typedef unsigned long long ull;
typedef vector<int> poly;

const int FFTN = 1 << 22, mod = 998244353; // FFTN最大值(1 << 23), mod-1 = 119 *
(1 << 23)

int qpow(int a, int t, int p = mod)
{
    int s = 1;
    for (; t; t >>= 1, a = (ll)a * a % p)
        if (t & 1)
            s = (ll)s * a % p;
    return s;
}

// FNTT

namespace FFT
{
    int w[FFTN + 5], W[FFTN + 5], R[FFTN + 5];

    // 初始化单位根的幂
    void FFTinit()
    {
        W[0] = 1;
        W[1] = qpow(3, (mod - 1) / FFTN); // 3为原根，若更换模数需要注意
        for (int i = 2; i <= FFTN; ++i)
            W[i] = (ll)W[i - 1] * W[1] % mod;
    }
    int FFTinit(int n)
    {
```

```
        int L = 1;
        for (; L <= n; L <<= 1)
            ;
        for (int i = 0; i <= L - 1; ++i)
            R[i] = (R[i >> 1] >> 1) | ((i & 1) ? (L >> 1) : 0);
        return L;
    }

    ull p[FFTN + 5];
    int A[FFTN + 5], B[FFTN + 5];

    // DFT和IDFT过程
    void DFT(int *a, int n)
    {
        for (int i = 0; i < n; ++i)
            p[R[i]] = a[i];
        for (int d = 1; d < n; d <<= 1)
        {
            int len = FFTN / (d << 1);
            for (int i = 0, j = 0; i < d; ++i, j += len)
                w[i] = W[j];
            for (int i = 0; i < n; i += (d << 1))
                for (int j = 0; j < d; ++j)
                {
                    int y = p[i + j + d] * w[j] % mod;
                    p[i + j + d] = p[i + j] + mod - y;
                    p[i + j] += y;
                }
            if (d == (1 << 15))
                for (int i = 0; i < n; ++i)
                    p[i] %= mod;
        }
        for (int i = 0; i < n; ++i)
            a[i] = p[i] % mod;
    }
    void IDFT(int *a, int n)
    {
        for (int i = 0; i < n; ++i)
            p[R[i]] = a[i];
        for (int d = 1; d < n; d <<= 1)
        {
            int len = FFTN / (d << 1);
            for (int i = 0, j = FFTN; i < d; ++i, j -= len)
                w[i] = W[j];
            for (int i = 0; i < n; i += (d << 1))
                for (int j = 0; j < d; ++j)
                {
                    int y = p[i + j + d] * w[j] % mod;
                    p[i + j + d] = p[i + j] + mod - y;
                    p[i + j] += y;
                }
            if (d == (1 << 15))
                for (int i = 0; i < n; ++i)
                    p[i] %= mod;
        }
```

```
        int val = qpow(n, mod - 2);
        for (int i = 0; i < n; ++i)
            a[i] = p[i] * val % mod;
    }

    // 多项式加法, Plus(poly a , poly b)返回a+b
    poly Plus(const poly &a, const poly &b)
    {
        int sza = a.size() - 1, szb = b.size() - 1;
        poly ans(std::max(sza, szb) + 1);
        for (int i = 0; i <= sza; ++i)
            ans[i] = a[i];
        for (int i = 0; i <= szb; ++i)
            ans[i] = (ans[i] + b[i]) % mod;
        return ans;
    }

    // 多项式减法, Minus(poly a , poly b)返回a-b
    poly Minus(const poly &a, const poly &b)
    {
        int sza = a.size() - 1, szb = b.size() - 1;
        poly ans(std::max(sza, szb) + 1);
        for (int i = 0; i <= sza; ++i)
            ans[i] = a[i];
        for (int i = 0; i <= szb; ++i)
            ans[i] = (ans[i] + mod - b[i]) % mod;
        return ans;
    }

    // 多项式乘法, Mul(poly a , poly b)返回a*b
    poly Mul(const poly &a, const poly &b)
    {
        int sza = a.size() - 1, szb = b.size() - 1;
        poly ans(sza + szb + 1);
        if (sza <= 30 || szb <= 30)
        {
            for (int i = 0; i <= sza; ++i)
                for (int j = 0; j <= szb; ++j)
                    ans[i + j] = (ans[i + j] + (ll)a[i] * b[j]) % mod;
            return ans;
        }
        int L = FFTinit(sza + szb);
        for (int i = 0; i < L; ++i)
            A[i] = (i <= sza ? a[i] : 0);
        for (int i = 0; i < L; ++i)
            B[i] = (i <= szb ? b[i] : 0);
        DFT(A, L), DFT(B, L);
        for (int i = 0; i < L; ++i)
            A[i] = (ll)A[i] * B[i] % mod;
        IDFT(A, L);
        for (int i = 0; i <= sza + szb; ++i)
            ans[i] = A[i];

        return ans;
    }
```

```
// 多项式快速幂, PolyPow(poly a , b)返回a^b
// 注意次数为(a.size()-1)*b
poly PolyPow(const poly &a, int b)
{
    int sza = a.size() - 1;
    poly ans(sza * b + 1);
    int L = FFTinit(sza * b + 1);
    for (int i = 0; i < L; ++i)
        A[i] = (i <= sza ? a[i] : 0);
    DFT(A, L);
    for (int i = 0; i < L; ++i)
        A[i] = qpow(A[i], b);
    IDFT(A, L);
    for (int i = 0; i <= sza * b; ++i)
        ans[i] = A[i];

    return ans;
}

// % x^k
// b 可先对 mod 取模
// 当 a0 = 1 时， 也可转为求 e^{k*ln(a)}
poly PolyPow(const poly &a, int b, int k)
{
    poly res(1, 1);
    poly aa(a);
    while (b)
    {
        aa.resize(k);
        if (b & 1)
        {
            res = Mul(res, aa);
            res.resize(k);
        }
        aa = Mul(aa, aa);
        b >>= 1;
    }
    return res;
}

// 多项式求逆, getinv(poly a , n)返回a^(-1) mod x^n
void Getinv(int *a, int *b, int n)
{
    if (n == 1)
    {
        b[0] = qpow(a[0], mod - 2);
        return;
    }
    int nn = (n + 1) >> 1;
    Getinv(a, b, nn);
    int L = FFTinit(n << 1);
    for (int i = 0; i < L; ++i)
        A[i] = (i < n ? a[i] : 0);
    for (int i = 0; i < L; ++i)
```

```
            B[i] = (i < nn ? b[i] : 0);
        DFT(A, L);
        DFT(B, L);
        for (int i = 0; i < L; ++i)
            A[i] = (ll)B[i] * (2 + mod - (ll)A[i] * B[i] % mod) % mod;
        IDFT(A, L);
        for (int i = 0; i < n; ++i)
            b[i] = A[i];
    }
    poly getinv(const poly &a, int n)
    {
        int *b = new int[n], *c = new int[n], sz = a.size();
        for (int i = 0; i < n; ++i)
            b[i] = (i < sz ? a[i] : 0);
        for (int i = 0; i < n; ++i)
            c[i] = 0;

        poly ans(n);
        Getinv(b, c, n);
        for (int i = 0; i < n; ++i)
            ans[i] = c[i];
        delete[] b;
        delete[] c;
        return ans;
    }

    // 结果 %x^len
    poly derivation(const poly &a, int len)
    {
        poly ans(len, 0);
        for (int i = 0; i < len && i + 1 < a.size(); ++i)
            ans[i] = (ll)a[i + 1] * (i + 1) % mod;
        return ans;
    }

    // 注意常数项这里默认为 0
    // 结果 %x^len
    poly integral(const poly &a, int len)
    {
        poly ans(len, 0);
        for (int i = 1; i < len && i - 1 < a.size(); ++i)
            ans[i] = (ll)a[i - 1] * qpow(i, mod - 2) % mod;
        return ans;
    }

    // 模意义下当且仅当 a0 = 1 时，a 有对数多项式
    // 结果 %x^len
    poly getln(const poly &a, int len)
    {
        poly dB = Mul(derivation(a, len), getinv(a, len));
        dB.resize(len);
        return integral(dB, len);
    }

    // a0 = 0 时有意义
```

```
    // exp 结果 %x^len
    void Exp(const poly &a, poly &res, int len)
    {
        if (len == 1)
        {
            res[0] = 1;
            return;
        }
        Exp(a, res, len + 1 >> 1);
        poly F = getln(res, len);
        F[0] = (a[0] + 1 - F[0] + mod) % mod;
        for (int i = 1; i < len; ++i)
            F[i] = (a[i] - F[i] + mod) % mod;
        poly t(res.begin(), res.begin() + len);
        t = Mul(t, F);
        for (int i = 0; i < len; ++i)
            res[i] = t[i];
    }
    poly exp(const poly &a, int len)
    {
        poly res(len, 0);
        Exp(a, res, len);
        return res;
    }
}

// example

int main()
{
    // clock_t st = clock(), ed;
    ios::sync_with_stdio(0);
    cin.tie(0);
    // cout << setprecision(15) << fixed;

    FFT::FFTinit(); // 必需
    int n, m, k;
    cin >> n >> m >> k;
    poly f(k + 1);
    ll C = 1;
    for (int i = 0; i <= k; ++i)
    {
        // debug(C);
        if (i % 2 == 0)
            f[i] = C;
        C = C * (n - i) % mod;
        C = C * qpow(i + 1, mod - 2) % mod;
    }
    poly g(1, 1);
    int ans = qpow(2, m);
    while (m)
    {
        f.resize(k + 1);
        if (m & 1)
        {
```

```
            g = FFT::Mul(g, f);
            g.resize(k + 1);
        }
        f = FFT::Mul(f, f);
        m >>= 1;
    }
    ans = 1ll * g[k] * ans % mod;
    cout << ans << endl;

    // ed = clock();
    // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
    // cout << "Total time: " << endtime << endl;
    return 0;
}
```

Prime

1. the Sieve of Eratosthenes

```
#define MAXN 100005

vector<bool> nop(MAXN, false);

void init1()
{
    nop[0] = true;
    nop[1] = true;
    for (int i = 2; i * i < MAXN; ++i)
    {
        if (!nop[i])
        {
            for (int j = i * i; j < MAXN; j += i)
                nop[j] = true;
        }
    }
}
```

2. Euler sieve

```
bool isnp[MAXN];
vector<int> primes;
void init2()
{
    isnp[0] = isnp[1] = 1;
    for (int i = 2; i < MAXN; i++)
    {
        if (!isnp[i])
            primes.push_back(i);
        for (int p : primes)
        {
            if (p * i > MAXN)
                break;
            isnp[p * i] = 1;
            if (i % p == 0)
```

```
                break;
        }
    }
}
```

3. Deterministic Miller–Rabin primality test

> for 1 ~ (1 << 64)

```
namespace millerRabin
{
    ll qpow(ll a, ll t, ll mod)
    {
        a %= mod;
        ll res = 1;
        while (t)
        {
            if (t & 1)
                res = (__int128)res * a % mod;
            a = (__int128)a * a % mod;
            t >>= 1;
        }
        return res;
    }

    const int primebook[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

    bool isPrime(ll n)
    {
        if (n < 3 || n % 2 == 0)
            return n == 2;
        if (n <= 40)
        {
            for (ll a : primebook)
                if (a == n)
                    return 1;
            return 0;
        }
        ll u = n - 1, t = 0;
        while (u % 2 == 0)
            u /= 2, ++t;
        for (ll a : primebook)
        {
            ll v = qpow(a, u, n);
            if (v == 1)
                continue;
            ll s;
            for (s = 0; s < t; ++s)
            {
                if (v == n - 1)
                    break;
                v = (__int128)v * v % n;
            }
            if (s == t)
                return 0;
```

```
        }
        return 1;
    }
}
```

4. prime factorization

```
void prime_factor(int x)
{
    for (int i = 2; i <= x / i; ++i)
    {
        while (x % i == 0)
        {
            // do sth.
            x /= i;
        }
    }
    if (x > 1)
    {
        // do sth.
    }
}
```

8. Others

Big_Integer

1. __int128

```
__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;
}

void print(__int128 x)
{
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    string op = "";
    while (x)
```

```
    {
        op.pb(x % 10 + '0');
        x /= 10;
    }
    reverse(all(op));
    cout << op;
}
```

2. BigIntTiny

> + - * / = compare cout
>
> link: https://github.com/Baobaobear/MiniBigInteger

```
struct BigIntTiny
{
    int sign;
    std::vector<int> v;

    BigIntTiny() : sign(1) {}
    BigIntTiny(const std::string &s) { *this = s; }
    BigIntTiny(int v)
    {
        char buf[21];
        sprintf(buf, "%d", v);
        *this = buf;
    }
    void zip(int unzip)
    {
        if (unzip == 0)
        {
            for (int i = 0; i < (int)v.size(); i++)
                v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 + get_pos(i * 4
+ 2) * 100 + get_pos(i * 4 + 3) * 1000;
        }
        else
            for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a; i >= 0;
i--)
                a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100, v[i] = (i &
1) ? a / 10 : a % 10;
        setsign(1, 1);
    }
    int get_pos(unsigned pos) const { return pos >= v.size() ? 0 : v[pos]; }
    BigIntTiny &setsign(int newsign, int rev)
    {
        for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
            v.erase(v.begin() + i);
        sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0)) ? 1 : (rev ?
newsign * sign : newsign);
        return *this;
    }
    std::string to_str() const
    {
        BigIntTiny b = *this;
        std::string s;
        for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
```

```
            s += char(*(b.v.rbegin() + i) + '0');
        return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") : s);
    }
    bool absless(const BigIntTiny &b) const
    {
        if (v.size() != b.v.size())
            return v.size() < b.v.size();
        for (int i = (int)v.size() - 1; i >= 0; i--)
            if (v[i] != b.v[i])
                return v[i] < b.v[i];
        return false;
    }
    BigIntTiny operator-() const
    {
        BigIntTiny c = *this;
        c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
        return c;
    }
    BigIntTiny &operator=(const std::string &s)
    {
        if (s[0] == '-')
            *this = s.substr(1);
        else
        {
            for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
                v.push_back(*(s.rbegin() + i) - '0');
            zip(0);
        }
        return setsign(s[0] == '-' ? -1 : 1, sign = 1);
    }
    bool operator<(const BigIntTiny &b) const
    {
        return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) :
b.absless(*this));
    }
    bool operator==(const BigIntTiny &b) const { return v == b.v && sign ==
b.sign; }
    BigIntTiny &operator+=(const BigIntTiny &b)
    {
        if (sign != b.sign)
            return *this = (*this) - -b;
        v.resize(std::max(v.size(), b.v.size()) + 1);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++)
        {
            carry += v[i] + b.get_pos(i);
            v[i] = carry % 10000, carry /= 10000;
        }
        return setsign(sign, 0);
    }
    BigIntTiny operator+(const BigIntTiny &b) const
    {
        BigIntTiny c = *this;
        return c += b;
    }
    void add_mul(const BigIntTiny &b, int mul)
```

```
    {
        v.resize(std::max(v.size(), b.v.size()) + 2);
        for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++)
        {
            carry += v[i] + b.get_pos(i) * mul;
            v[i] = carry % 10000, carry /= 10000;
        }
    }
    BigIntTiny operator-(const BigIntTiny &b) const
    {
        if (b.v.empty() || b.v.size() == 1 && b.v[0] == 0)
            return *this;
        if (sign != b.sign)
            return (*this) + -b;
        if (absless(b))
            return -(b - *this);
        BigIntTiny c;
        for (int i = 0, borrow = 0; i < (int)v.size(); i++)
        {
            borrow += v[i] - b.get_pos(i);
            c.v.push_back(borrow);
            c.v.back() -= 10000 * (borrow >>= 31);
        }
        return c.setsign(sign, 0);
    }
    BigIntTiny operator*(const BigIntTiny &b) const
    {
        if (b < *this)
            return b * *this;
        BigIntTiny c, d = b;
        for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(), 0))
            c.add_mul(d, v[i]);
        return c.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator/(const BigIntTiny &b) const
    {
        BigIntTiny c, d;
        d.v.resize(v.size());
        double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() - 2) /
1e4) +
                           (b.get_pos((unsigned)b.v.size() - 3) + 1) / 1e8);
        for (int i = (int)v.size() - 1; i >= 0; i--)
        {
            c.v.insert(c.v.begin(), v[i]);
            int m = (int)((c.get_pos((int)b.v.size()) * 10000 +
c.get_pos((int)b.v.size() - 1)) * db);
            c = c - b * m, c.setsign(c.sign, 0), d.v[i] += m;
            while (!(c < b))
                c = c - b, d.v[i] += 1;
        }
        return d.setsign(sign * b.sign, 0);
    }
    BigIntTiny operator%(const BigIntTiny &b) const { return *this - *this / b *
b; }
    bool operator>(const BigIntTiny &b) const { return b < *this; }
```

```
    bool operator<=(const BigIntTiny &b) const { return !(b < *this); }
    bool operator>=(const BigIntTiny &b) const { return !(*this < b); }
    bool operator!=(const BigIntTiny &b) const { return !(*this == b); }
};
```

Data

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

using namespace std;

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

#define pb push_back
#define all(x) (x).begin(), (x).end()
#define fi first
#define se second
#define endl '\n'
#define debug(x) { cerr << #x << " = " << x << endl; }

/*-------------------------------------------*/

#include <sys/time.h>
// millisecond
// you should include <sys/time.h> before use it

int getSeed()
{
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec * 1000 + tv.tv_usec / 1000;
}

// better randint

int randint(int l, int r = 0)
{
    // static mt19937 eng(time(0));
    static mt19937 eng(getSeed());
    if (l > r)
        swap(l, r);
    uniform_int_distribution<int> dis(l, r);
    return dis(eng);
}

// Example

int main()
{
    // clock_t st = clock(), ed;
    ios::sync_with_stdio(0);
    cin.tie(0);
    // cout << setprecision(15) << fixed;
```

```
    int N = randint(1, 10);
    cout << N << endl;
    for (int i = 1; i <= N; ++i)
    {
        int x = randint(1, 10);
        cout << x << ' ';
    }
    cout << endl;
    // ed = clock();
    // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
    // cout << "Total time: " << endtime << endl;
    return 0;
}
```

Fast_IO

1. fread version

```
namespace IO
{
    char buf[1 << 15], *S, *T;
    inline char gc()
    {
        if (S == T)
        {
            T = (S = buf) + fread(buf, 1, 1 << 15, stdin);
            if (S == T)
                return EOF;
        }
        return *S++;
    }
    inline int read()
    {
        int x;
        bool f;
        char c;
        for (f = 0; (c = gc()) < '0' || c > '9'; f = c == '-')
            ;
        for (x = c ^ '0'; (c = gc()) >= '0' && c <= '9'; x = x * 10 + (c ^ '0'))
            ;
        return f ? -x : x;
    }
    const int olim = 1 << 20;
    char obuf[olim + 5], *OS;
    inline void print()
    {
        fwrite(obuf, 1, OS - obuf, stdout);
        OS = obuf;
    }
    inline void pc(char c)
    {
        *OS++ = c;
        if (OS == obuf + olim)
            print();
    }
```

```
    inline void write(const char *s, char c = '\n')
    {
        for (const char *i = s; *i != ' '; ++i)
            pc(*i);
        pc(c);
    }
    inline void write(int x, char c = '\n')
    {
        if (x < 0)
            pc('-'), x = -x;
        char temp[20], cnt;
        for (temp[cnt = 0] = x % 10; x /= 10; temp[++cnt] = x % 10)
            ;
        for (; ~cnt; --cnt)
            pc(temp[cnt] ^ '0');
        pc(c);
    }
    struct flusher
    {
        flusher() { OS = obuf; }
        ~flusher() { print(); }
    } __flusher__;
} // namespace IO
```

2. old version

```
namespace IO_old // pay attention to overflowing case
{
    inline void read(int &x)
    {
        x = 0;
        bool flag(0);
        char ch = getchar();
        while (!isdigit(ch))
            flag = ch == '-', ch = getchar();
        while (isdigit(ch))
            x = (x << 1) + (x << 3) + (ch ^ 48), ch = getchar();
        flag ? x = -x : 0;
    }

    inline void write(int x, char c = '\n')
    {
        x < 0 ? x = -x, putchar('-') : 0;
        static short Stack[50], top(0);
        do
            Stack[++top] = x % 10, x /= 10;
        while (x);
        while (top)
            putchar(Stack[top--] | 48);
        putchar(c);
    }
}
```

Functions

> A collection of ~~(unusual but)~~ useful functions

---

algorithm

- stable_sort
- partial_sort
- nth_element
- upper_bound
- lower_bound
- unique

math

- __lg
- ceil
- floor
- round
- exp
- next_permutation
- prev_permutation

method

- string.c_str // char *p = string.c_str();
- string.substr
- string.replace // string.replace(find(old), len, new);
- string.compare
- set.lowerbound

binary

- __builtin_ffs
- __builtin_clz
- __builtin_ctz
- __builtin_popcount

others

- memset
- fill
- iota
- shuffle
- sprintf
- atoi
- stoi

Gospers_Hack

```
void GospersHack(int k, int n)
{
    int cur = (1 << k) - 1;
    int limit = (1 << n);
    while (cur < limit)
    {
        // do something
        int lb = cur & -cur;
        int r = cur + lb;
        cur = ((r ^ cur) >> __builtin_ctz(lb) + 2) | r;
        // or: cur = (((r ^ cur) >> 2) / lb) | r;
    }
}

// GospersHack(3, 5)

// 00111
// 01011
// 01101
// 01110
// 10011
// 10101
// 10110
// 11001
// 11010
// 11100
```

Matching

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

using namespace std;

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

#define pb push_back
#define all(x) (x).begin(), (x).end()
#define fi first
#define se second
#define endl '\n'
#define debug(x) { cerr << #x << " = " << x << endl; }

/*-----------------------------------------*/

class Matching_Windows
{
public:
    static void match()
    {
        int T = 1;
        while (114 != 514)
```

```
        {
            system("data.exe > in.txt");
            system("baoli.exe < in.txt > baoli.txt");
            system("std.exe < in.txt > std.txt");

            if (system("fc std.txt baoli.txt"))
            {
                cout << "#" << T << " Wrong Answer" << endl;
                break;
            }
            cout << "#" << T << " Accepted" << endl;
            ++T;
        }
    }
};

class Matching_Linux
{
public:
    static void match()
    {
        int T = 1;
        while (114 != 514)
        {
            system("./data > in.txt");
            system("cat in.txt | ./std > std.txt");
            system("cat in.txt | ./baoli > baoli.txt");

            if (system("diff std.txt baoli.txt"))
            {
                cout << "#" << T << " Wrong Answer" << endl;
                break;
            }
            cout << "#" << T << " Accepted" << endl;
            ++T;
        }
    }
};
```

Mo's_Algorithm

```
namespace normalMo
{
    const int MAXN = 30005, MAXQ = 200005, MAXM = 1000005;
    int sq;
    struct query // 把询问以结构体方式保存
    {
        int l, r, id;
        bool operator<(const query &o) const // 重载<运算符，奇偶化排序
        {
            // 这里只需要知道每个元素归属哪个块，而块的大小都是sqrt(n)，所以可以直接用l/sq
            if (l / sq != o.l / sq)
                return l < o.l;
            if (l / sq & 1)
                return r < o.r;
```

```
            return r > o.r;
        }
    } Q[MAXQ];
    int A[MAXN], ans[MAXQ], cnt[MAXM], cur = 0, l = 1, r = 0;
    void add(int p)
    {
        if (cnt[A[p]] == 0)
            cur++;
        cnt[A[p]]++;
    }
    void del(int p)
    {
        cnt[A[p]]--;
        if (cnt[A[p]] == 0)
            cur--;
    }
    void solve()
    {
        int n;
        cin >> n;
        sq = sqrt(n);
        for (int i = 1; i <= n; ++i)
            cin >> A[i];
        int q;
        cin >> q;
        for (int i = 0; i < q; ++i)
            cin >> Q[i].l >> Q[i].r, Q[i].id = i; // 把询问离线下来
        sort(Q, Q + q);                           // 排序
        for (int i = 0; i < q; ++i)
        {
            while (l > Q[i].l)
                add(--l);
            while (r < Q[i].r)
                add(++r);
            while (l < Q[i].l)
                del(l++);
            while (r > Q[i].r)
                del(r--);
            ans[Q[i].id] = cur; // 储存答案
        }
        for (int i = 0; i < q; ++i)
            cout << ans[i] << endl; // 按编号顺序输出
    }
}
```

ModInt

```
template<const int T>
class ModInt {
    const static int mod = T;
    int x;

public:
    ModInt(int x = 0) : x(x % mod) {}
    ModInt(long long x) : x(int(x % mod)) {}
```

```
    int val() { return x; }
    ModInt operator + (const ModInt &a) const { int x0 = x + a.x; return
ModInt(x0 < mod ? x0 : x0 - mod); }
    ModInt operator - (const ModInt &a) const { int x0 = x - a.x; return
ModInt(x0 < 0 ? x0 + mod : x0); }
    ModInt operator * (const ModInt &a) const { return ModInt(1LL * x * a.x %
mod); }
    ModInt operator / (const ModInt &a) const { return *this * a.inv(); }
    bool operator == (const ModInt &a) const { return x == a.x; };
    bool operator != (const ModInt &a) const { return x != a.x; };
    void operator += (const ModInt &a) { x += a.x; if (x >= mod) x -= mod; }
    void operator -= (const ModInt &a) { x -= a.x; if (x < 0) x += mod; }
    void operator *= (const ModInt &a) { x = 1LL * x * a.x % mod; }
    void operator /= (const ModInt &a) { *this = *this / a; }
    friend ModInt operator + (int y, const ModInt &a){ int x0 = y + a.x; return
ModInt(x0 < mod ? x0 : x0 - mod); }
    friend ModInt operator - (int y, const ModInt &a){ int x0 = y - a.x; return
ModInt(x0 < 0 ? x0 + mod : x0); }
    friend ModInt operator * (int y, const ModInt &a){ return ModInt(1LL * y *
a.x % mod);}
    friend ModInt operator / (int y, const ModInt &a){ return ModInt(y) / a;}
    friend ostream &operator<<(ostream &os, const ModInt &a) { return os <<
a.x;}
    friend istream &operator>>(istream& is, ModInt& t){return is >> t.x;}

    ModInt pow(int64_t n) const {
        ModInt res(1), mul(x);
        while(n){
            if (n & 1) res *= mul;
            mul *= mul;
            n >>= 1;
        }
        return res;
    }
    ModInt inv() const {
        int a = x, b = mod, u = 1, v = 0;
        while (b) {
            int t = a / b;
            a -= t * b; swap(a, b);
            u -= t * v; swap(u, v);
        }
        if (u < 0) u += mod;
        return u;
    }
};

using mint = ModInt<998244353>;
```

Tricks

> A collection of useful tricks

- integer floor

```
a / b
```

- integer ceil

```
(a + b - 1) / b
```

- integer sqrt

```
ll r = sqrtl(x) + 1;
while (r * r > x)
    r--;
```

- new line

```
for(i = 1; i <= n; i++)
    for(j = 1; j <= m; j++)
    cout << a[i][j] << " \n"[j == m];
```

- rm the same element

```
sort(all(vec));
vec.resize(unique(all(vec)) - vec.begin());
```

- count how many segments covering the range

```
vector<pii> arr(n);
vector<int> node(2 * n), cnt(2 * n, 0);
for (int i = 0; i < n; ++i)
{
    cin >> arr[i].fi >> arr[i].se;
    node[k++] = arr[i].fi;
    node[k++] = arr[i].se;
}
sort(all(node));
int N = unique(all(node)) - node.begin();
for (int i = 0; i < n; ++i)
{
    cnt[lower_bound(node.begin(), node.begin() + N, arr[i].fi) -
node.begin()]++;
    cnt[lower_bound(node.begin(), node.begin() + N, arr[i].se) -
node.begin()]--;
}
int cur = 0;
for (int i = 0; i < N - 1; ++i)
{
    cur += cnt[i];
    // cur is the number
    int len = node[i + 1] - node[i];
}
```

- two rectangles cross

```
bool cross(Rec A, Rec B)
{
    return max(A.x1, B.x1) <= min(A.x2, B.x2) && max(A.y1, B.y1) <=
min(A.y2, B.y2);
}
```

- tie and tuple

```
tuple<int, int, int, char> t(3, 4, 5, 'g');
int a, b;
tie(b, ignore, a, ignore) = t;
```

- raw string

```
string r_str = R"(Hello\tWorld\n)";
// output: Hello\tWorld\n
```

- regex

```
regex email_pattern(R"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)");
string valid_email("swift@codeforces.com"), invalid_email("hello world");

if (regex_match(valid_email, email_pattern))
    cout << valid_email << " is valid" << endl;
if (!regex_match(invalid_email, email_pattern))
    cout << invalid_email << " is invalid" << endl;
```

- hash

```
// for unordered_X, e.g. unordered_map<int, int, my_hash>
// link: https://codeforces.com/blog/entry/62393
struct my_hash
{
    static uint64_t splitmix64(uint64_t x)
    {
        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);
    }

    size_t operator()(pair<uint64_t, uint64_t> x) const
    {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x.first + FIXED_RANDOM) ^
```

```
            (splitmix64(x.second + FIXED_RANDOM) >> 1);
    }
};
```

9. Snippets

Snippets_C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define ll long long

int main()
{
    // clock_t st = clock(), ed;



    // ed = clock();
    // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
    // printf("Total time: %lf\n", endtime);
    return 0;
}
```

Snippets_Cpp

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

using namespace std;

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

#define pb push_back
#define all(x) (x).begin(), (x).end()
#define fi first
#define se second
#define endl '\n'
#define debug(x) { cerr << #x << " = " << x << endl; }

/*-------------------------------------------*/

void solve()
{

}

int main()
{
```

```
    cin.tie(0)->sync_with_stdio(0);
    // cout << setprecision(15) << fixed;
    int T = 1;
    // cin >> T;
    while (T--)
    {
        solve();
    }

    return 0;
}
```

Snippets_Cpp_full

```
#include <algorithm>
#include <array>
#include <bitset>
#include <cassert>
#include <chrono>
#include <climits>
#include <cmath>
#include <complex>
#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <ctime>
#include <functional>
#include <iomanip>
#include <iostream>
#include <map>
#include <numeric>
#include <queue>
#include <random>
#include <set>
#include <string>
#include <unordered_set>
#include <unordered_map>
#include <vector>

using namespace std;

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

#define pb push_back
#define all(x) (x).begin(), (x).end()
#define fi first
#define se second
#define endl '\n'

#define debug(x)                          \
    {                                     \
        cerr << #x << " = " << x << endl; \
    }
#define debugfull(x)                                                    \
```

```
    {                                                                       \
        cerr << #x << " = " << x << " (line " << __LINE__ << ")" << endl; \
    }

/*-----------------------------------------*/

int main()
{
    // clock_t st = clock(), ed;
    ios::sync_with_stdio(0);
    cin.tie(0);
    // cout << setprecision(15) << fixed;



    // ed = clock();
    // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
    // cout << "Total time: " << endtime << endl;
    return 0;
}
```

Snippets_others

Node.js

```
"use strict";

process.stdin.resume();
process.stdin.setEncoding("utf-8");

let strInput = "";
let curLine = 0;

process.stdin.on("data", (inputStdin) => {
  strInput += inputStdin;
});

process.stdin.on("end", () => {
  strInput = strInput
    .trim()
    .split("\n")
    .map((_) => {
      return _.trim();
    });
  main();
});

function readline() {
  return strInput[curLine++];
}

function main() {
  let t = parseInt(readline());
  while (t--) {
```

```
  }
}
```

Python

```python
import sys
import bisect
import collections
import heapq
import math
from functools import cmp_to_key, lru_cache
from itertools import permutations, combinations
from random import getrandbits

input = lambda: sys.stdin.readline().strip()
# sys.setrecursionlimit(10**5)
T = 1
# T = int(input())


def solve():
    pass


for _ in range(T):
    solve()
```

Java

```java
// TODO
```

10. String

Aho-Corasick_Automaton

1. normal version

```cpp
class AC
{
private:
    const static int N = 500005;
    int tr[N][26], val[N], fail[N], cnt;
    queue<int> q;

public:
    void insert(const string &s)
    {
        int cur = 0;
        for (int i = 0; i < s.size(); ++i)
        {
            int v = s[i] - 'a';
```

```
            if (!tr[cur][v])
                tr[cur][v] = ++cnt;
            cur = tr[cur][v];
        }
        ++val[cur];
    }
    void build()
    {
        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)
                if (tr[u][i])
                    fail[tr[u][i]] = tr[fail[u]][i], q.push(tr[u][i]);
                else
                    tr[u][i] = tr[fail[u]][i];
        }
    }
    int query(const string &s)
    {
        int cur = 0, ans = 0;
        for (int i = 0; i < s.size(); ++i)
        {
            cur = tr[cur][s[i] - 'a'];
            for (int t = cur; t && ~val[t]; t = fail[t])
                ans += val[t], val[t] = -1;
        }
        return ans;
    }
};
```

2. improved with topo

```
class AC_topo
{
private:
    const static int maxn = 5000005;
    int cnt;
    int indeg[maxn];
    queue<int> q;
    struct trie_node
    {
        int son[27];
        int fail;
        int flag;
        int ans;
        void init()
        {
            memset(son, 0, sizeof(son));
            ans = fail = flag = 0;
        }
```

```
    } trie[maxn];

public:
    int vis[maxn], rev[maxn];
    void init(int n)
    {
        for (int i = 0; i <= cnt; ++i)
            trie[i].init();
        for (int i = 1; i <= n; ++i)
            vis[i] = 0;
        cnt = 1;
    }
    void insert(const string &s, int num)
    {
        int u = 1;
        for (int i = 0; i < s.size(); ++i)
        {
            int v = s[i] - 'a';
            if (!trie[u].son[v])
                trie[u].son[v] = ++cnt;
            u = trie[u].son[v];
        }
        if (!trie[u].flag)
            trie[u].flag = num;
        rev[num] = trie[u].flag;
        return;
    }
    void build()
    {
        for (int i = 0; i < 26; ++i)
            trie[0].son[i] = 1;
        q.push(1);
        trie[1].fail = 0;
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            int Fail = trie[u].fail;
            for (int i = 0; i < 26; ++i)
            {
                int v = trie[u].son[i];
                if (!v)
                {
                    trie[u].son[i] = trie[Fail].son[i];
                    continue;
                }
                trie[v].fail = trie[Fail].son[i];
                ++indeg[trie[Fail].son[i]];
                q.push(v);
            }
        }
    }
    void topo()
    {
        for (int i = 1; i <= cnt; ++i)
```

```
            if (!indeg[i])
                q.push(i);
        while (!q.empty())
        {
            int fr = q.front();
            q.pop();
            vis[trie[fr].flag] = trie[fr].ans;
            int u = trie[fr].fail;
            trie[u].ans += trie[fr].ans;
            if (!(--indeg[u]))
                q.push(u);
        }
    }
    void query(const string &s)
    {
        int u = 1;
        for (int i = 0; i < s.size(); ++i)
            u = trie[u].son[s[i] - 'a'], ++trie[u].ans;
    }
};

// example

AC_topo ac;

int main()
{
    // clock_t st = clock(), ed;
    ios::sync_with_stdio(0);
    cin.tie(0);
    // cout << setprecision(15) << fixed;
    int n;
    string s;
    cin >> n;
    ac.init(n);
    for (int i = 1; i <= n; ++i)
        cin >> s, ac.insert(s, i);
    ac.build();
    cin >> s;
    ac.query(s);
    ac.topo();
    for (int i = 1; i <= n; ++i)
        cout << ac.vis[ac.rev[i]] << endl;
    // ed = clock();
    // double endtime = (double)(ed - st) / CLOCKS_PER_SEC;
    // cout << "Total time: " << endtime << endl;
    return 0;
}
```

Hash

1. string hash

```
const int M = 1e9 + 7;
const int B = 233;

int get_hash(const string &s)
{
    int res = s[0], sz = s.size();
    for (int i = 1; i < sz; ++i)
        res = (ll(res) * B + s[i]) % M;
    return res;
}
```

2. double hash

```
const int mod1 = 19260817;
const int mod2 = 1e9 + 7;

pii get_hash(string &s)
{
    ll r1 = 0, r2 = 0;
    for (int i = 0; i < s.size(); ++i)
    {
        r1 = (r1 * 131 + s[i]) % mod1;
        r2 = (r2 * 233 + s[i]) % mod2;
    }
    return {r1, r2};
}
```

3. substring hash(with prefix)

```
const int N = 1e5 + 5, P = 133;
unsigned long long h[N], p[N];

unsigned long long query(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}

void prepro(string &str)
{
    // index start from 1
    p[0] = 1;
    h[0] = 0;
    int sz = str.size();
    for (int i = 0; i < sz; i++)
    {
        p[i + 1] = p[i] * P;
        h[i + 1] = h[i] * P + str[i];
    }
}
```

4. double substring hash(with prefix)

```
namespace Double_Pre_Hash
{
    using pll = pair<ll, ll>;
    static const int N = 1048576 * 2 + 5, P = 133;
    const ll mod = 1000000033;
    const ll mod2 = 1000000007;
    ll h[N], p[N];
    ll h2[N], p2[N];

    map<pll, ll> ID;
    ll cnt = 0;

    ll query(int l, int r)
    {
        ll res = ((h[r] - h[l - 1] * p[r - l + 1]) % mod + mod) % mod;
        ll res2 = ((h2[r] - h2[l - 1] * p2[r - l + 1]) % mod2 + mod2) % mod2;
        if (ID.find({res, res2}) == ID.end())
            ID[{res, res2}] = ++cnt;
        return ID[{res, res2}];
    }

    void prepro(string &str)
    {
        // index start from 1
        p[0] = 1;
        h[0] = 0;
        p2[0] = 1;
        h2[0] = 0;
        int sz = str.size();
        for (int i = 0; i < sz; i++)
        {
            p[i + 1] = (p[i] * P) % mod;
            h[i + 1] = (h[i] * P + str[i]) % mod;
            p2[i + 1] = (p2[i] * P) % mod2;
            h2[i + 1] = (h2[i] * P + str[i]) % mod2;
        }
    }
}
```

KMP

> index starts from 1: s, p

```
namespace KMP
{
    const int N = 100010, M = 1000010;

    int n, m;
    int ne[N];
    char s[M], p[N];

    void get_next()
```

```
    {
        for (int i = 2, j = 0; i <= n; ++i)
        {
            while (j && p[i] != p[j + 1])
                j = ne[j];
            if (p[i] == p[j + 1])
                ++j;
            ne[i] = j;
        }
    }

    void match()
    {
        for (int i = 1, j = 0; i <= m; ++i)
        {
            while (j && s[i] != p[j + 1])
                j = ne[j];
            if (s[i] == p[j + 1])
                ++j;
            if (j == n)
            {
                printf("%d ", i - n);
                j = ne[j];
            }
        }
    }
}
```

Trie

1. dynamic

```
class Trie
{
private:
    Trie* son[26];
    bool exist;
public:
    Trie()
    {
        exist = false;
        for (int i = 0; i < 26; i++)
            son[i] = nullptr;
    }
    ~Trie()
    {
        for (int i = 0; i < 26; i++)
            if (son[i] != nullptr)
                delete son[i];
    }
    void insert(string &word)
    {
        Trie* root = this;
        for (char x: word)
        {
```

```
            int cur = x - 'a';
            if (root->son[cur] == nullptr)
                root->son[cur] = new Trie();
            root = root->son[cur];
        }
        root->exist = true;
    }
    bool find(string &word)
    {
        Trie* root = this;
        for (char x : word)
        {
            int cur = x - 'a';
            if (root->son[cur] == nullptr)
                return false;
            root = root->son[cur];
        }
        return root->exist;
    }
};
```

2. static

> MAXN = N * len

```
#define MAXN 100005

struct Trie
{
    int nex[MAXN][26], cnt;
    bool exist[MAXN];

    void insert(string &s)
    {
        int p = 0;
        for (int i = 0; i < s.size(); ++i)
        {
            int c = s[i] - 'a';
            if (!nex[p][c])
                nex[p][c] = ++cnt;
            p = nex[p][c];
        }
        exist[p] = 1;
    }
    bool find(string &s)
    {
        int p = 0;
        for (int i = 0; i < s.size(); ++i)
        {
            int c = s[i] - 'a';
            if (!nex[p][c])
                return 0;
            p = nex[p][c];
        }
        return exist[p];
```

```
    }
};
```

3. for 01

```
struct Trie_01
{
    int nex[MAXN][2], cnt;
    bool exist[MAXN];

    void insert(int x)
    {
        int p = 0;
        for (int i = 30; i >= 0; --i)
        {
            int cur = (x >> i) & 1;
            if (!nex[p][cur])
                nex[p][cur] = ++cnt;
            p = nex[p][cur];
        }
        exist[p] = 1;
    }
    bool find(int x)
    {
        int p = 0;
        for (int i = 30; i >= 0; --i)
        {
            int cur = (x >> i) & 1;
            if (!nex[p][cur])
                return 0;
            p = nex[p][cur];
        }
        return exist[p];
    }
};
```

END of ACM-Template