恶魔妹妹的算法模板。第几版已经不知道了。懒得维护版本号和更新内容了。反正想起来了传一份最新版。

最后，欢迎加群菜菜园子：730602769。

2024年7月25日。

编译

```cpp
#include<bits/stdc++.h>
using namespace std;
/*====================*/
#define endl "\n"
/*====================*/
typedef long long lnt;
/*====================*/
void Solve(void)
{

}
/*====================*/
int main()
{
#ifndef ONLINE_JUDGE
    freopen("IN.txt", "r+", stdin);
#endif
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);
    int T = 1; //cin >> T;
    while (T--)Solve();
    return 0;
}
```

表达式

获取优先级

```cpp
int GetPriority(char c)
{
    if (c == '*')return 0;
    if (c == '+')return -1;
    return -2;
}
```

中缀转后缀

```cpp
string PostfixExpression(string str)
{
    string res;
    stack<char>stk;
```

```
    for (auto c : str)
    {
        if (c == '_')
        {
            res.push_back(c);
        }
        if (c == '(' || c == ')')
        {
            if (c == '(')
            {
                stk.push('(');
            }
            if (c == ')')
            {
                while (!stk.empty() && stk.top() != '(')
                {
                    res.push_back(stk.top()); stk.pop();
                }
                stk.pop();
            }
        }
        if (c == '+' || c == '*')
        {
            while (!stk.empty() && GetPriority(stk.top()) >= GetPriority(c))
            {
                res.push_back(stk.top()); stk.pop();
            }
            stk.push(c);
        }
    }
    while (!stk.empty())
    {
        res.push_back(stk.top()); stk.pop();
    }
    return res;
}
```

建立表达式树

```
struct Node
{
    int val;
    char tag;
    Node* lch, * rch;
    Node(int _val = 0, char _tag = ' ', Node* _lch = NULL, Node* _rch = NULL)
    {
        val = _val, tag = _tag;
        lch = _lch, rch = _rch;
    }
};
Node* Build(string str)
{
    stack<Node*>stk;
```

```
    for (auto c : str)
    {
        if (c == '0' || c == '1')
        {
            stk.push(new Node(c - '0', ' ', NULL, NULL));
        }
        else
        {
            Node* rch = stk.top(); stk.pop();
            Node* lch = stk.top(); stk.pop();
            stk.push(new Node(((c == '&') ? (lch->val & rch->val) : (lch->val | rch->val)), c, lch, rch));
        }
    }
    return stk.top();
}
```

读写

__int128

```
ostream& operator<<(ostream& output, lnt integer)
{
    if (integer < 0)
    {
        output << "-"; integer *= -1;
    }
    string str;
    do
    {
        str.push_back('0' + char(integer % 10)); integer /= 10;
    } while (integer != 0);
    reverse(str.begin(), str.end());
    output << str;
    return output;
}
istream& operator>>(istream& input, lnt& integer)
{
    string str; input >> str; integer = 0;
    for (int i = 0; i < str.size(); ++i)
    {
        integer = integer * 10 + str[i] - '0';
    }
    return input;
}
```

快读快写

```
namespace FastIO
{
    template<int SIZE>
    class C_IOBUF
    {
```

```
    private:
        char p[SIZE], * p1, * p2, q[SIZE], * q1, * q2;
    public:
        C_IOBUF(void) { p1 = p2 = p, q1 = q, q2 = q + SIZE; }
        char GetChar(void)
        {
            return p1 == p2 && (p2 = (p1 = p) + fread(p, 1, SIZE, stdin)),
p1 == p2 ? EOF : *p1++;
        }
        void PutChar(char c)
        {
            if (q1 == q2) q2 = (q1 = q) + fwrite(q, 1, SIZE, stdout); *q1++
= c;
        }
        ~C_IOBUF(void) { fwrite(q, 1, q1 - q, stdout); }
    };
    C_IOBUF<1 << 20>BUF;
    class C_IStream
    {
    private:
        bool IsChar(const char& c)
        {
            return c != ' ' && c != '\n';
        }
        bool IsDigit(const char& c)
        {
            return '0' <= c && c <= '9';
        }
    public:
        C_IStream& operator >>(int& temp)
        {
            temp = 0; char c = BUF.GetChar(); bool flag = false;
            while (IsDigit(c) != true) { if (c == '-')flag = true; c =
BUF.GetChar(); }
            while (IsDigit(c) == true) { temp = temp * 10 + c - '0', c =
BUF.GetChar(); }
            temp = flag ? -temp : temp; return *this;
        }
        C_IStream& operator >>(lnt& temp)
        {
            temp = 0; char c = BUF.GetChar(); bool flag = false;
            while (IsDigit(c) != true) { if (c == '-')flag = true; c =
BUF.GetChar(); }
            while (IsDigit(c) == true) { temp = temp * 10 + c - '0', c =
BUF.GetChar(); }
            temp = flag ? -temp : temp; return *this;
        }
        C_IStream& operator >>(char& temp)
        {
            temp = ' '; char c = BUF.GetChar();
            while (IsChar(c) != true)c = BUF.GetChar();
            temp = c; return *this;
        }
        C_IStream& operator >>(string& temp)
        {
```

```
            temp.clear(); char c = BUF.GetChar();
            while (IsChar(c) != true)c = BUF.GetChar();
            while (IsChar(c) == true)temp += c, c = BUF.GetChar();
            return *this;
        }
    }cin;
    class C_OStream
    {
    public:
        C_OStream& operator <<(int temp)
        {
            int Top = 0; static int Stack[64];
            if (temp < 0) { BUF.PutChar('-'); temp = -temp; }
            do { Stack[Top++] = temp % 10; temp /= 10; } while (temp);
            while (Top) { BUF.PutChar(Stack[--Top] + '0'); } return *this;
        }
        C_OStream& operator <<(lnt temp)
        {
            int Top = 0; static int Stack[64];
            if (temp < 0) { BUF.PutChar('-'); temp = -temp; }
            do { Stack[Top++] = temp % 10; temp /= 10; } while (temp);
            while (Top) { BUF.PutChar(Stack[--Top] + '0'); } return *this;
        }
        C_OStream& operator <<(char temp)
        {
            BUF.PutChar(temp); return *this;
        }
        C_OStream& operator <<(string temp)
        {
            for (auto c : temp)BUF.PutChar(c); return *this;
        }
        C_OStream& operator <<(const char temp[])
        {
            int p = 0; while (temp[p] != '\0')BUF.PutChar(temp[p++]); return
*this;
        }
    }cout;
}
```

枚举

排列

全排列

```cpp
void AllPermutation(int l, int r)
{
    vector<int>p;
    for (int i = l; i <= r; ++i)
    {
        p.push_back(i);
    }
    do
    {
        //do something
    } while (next_permutation(p.begin(), p.end()));
}
```

子集

枚举全部子集

```cpp
void AllSubset(int s)
{
    for (int i = s; i; i = (i - 1) & s)
    {
        //do something
    }
}
```

枚举k大小子集

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

树论

LCT

```cpp
class C_LinkCutTree
{
public:
    int rev[N], fa[N], ch[N][2];
    /*====================*/
    bool Which(int x)
    {
        return ch[fa[x]][1] == x;
    }
```

```
    bool IsRoot(int x)
    {
        return ch[fa[x]][Which(x)] != x;
    }
    /*====================*/
    void PushUp(int x)
    {
        /*PushUp*/
    }
    void PushAll(int x)
    {
        if (!IsRoot(x))
        {
            PushAll(fa[x]);
        }
        PushDown(x);
    }
    void PushDown(int x)
    {
        if (rev[x])
        {
            swap(ch[x][0], ch[x][1]);
            rev[ch[x][0]] ^= 1;
            rev[ch[x][1]] ^= 1;
            rev[x] = 0;
        }
        /*PushDown*/
    }
    /*====================*/
    void Rotate(int x)
    {
        int y = fa[x], z = fa[y], w = Which(x);
        if (!IsRoot(y)) ch[z][Which(y)] = x; fa[x] = z;
        ch[y][w] = ch[x][w ^ 1];
        if (ch[y][w]) fa[ch[y][w]] = y;
        ch[x][w ^ 1] = y; fa[y] = x;
        PushUp(y); PushUp(x);
    }
    void Splay(int x)
    {
        PushAll(x);
        for (; !IsRoot(x); Rotate(x))
        {
            if (!IsRoot(fa[x]))
            {
                Rotate(Which(x) == Which(fa[x]) ? fa[x] : x);
            }
        }
    }
    /*====================*/
    void Access(int x)
    {
        for (int p = 0; x; p = x, x = fa[x])
        {
            Splay(x), ch[x][1] = p, PushUp(x);
```

```
        }
    }
    /*====================*/
    int FindRoot(int x)
    {
        Access(x); Splay(x);
        while (ch[x][0]) x = ch[x][0];
        Splay(x); return x;
    }
    void MakeRoot(int x)
    {
        Access(x); Splay(x); rev[x] ^= 1;
    }
    /*====================*/
    void Cut(int u, int v)
    {
        Split(u, v);
        if (fa[u] == v && !ch[u][1])
        {
            ch[v][0] = fa[u] = 0;
        }
    }
    void Link(int u, int v)
    {
        MakeRoot(u); fa[u] = v;
    }
    /*====================*/
    void Split(int u, int v)
    {
        MakeRoot(u); Access(v); Splay(v);
    }
    /*====================*/
    int LCA(int u, int v)
    {
        Access(u); int ans = 0;
        for (int child = 0; v; child = v, v = fa[v])
        {
            Splay(v); ch[v][1] = child; ans = v;
        }
        return ans;
    }
};
```

LCA

树剖法

```
class C_LCA
{
private:
    struct Node
    {
        int pre, dep, siz, son, top;
        Node(void)
```

```
        {
            pre = -1, dep = +0, siz = +1, son = -1, top = -1;
        }
    };
    /*====================*/
    int root;
    vector<int>* G;
    vector<Node>node;
    /*====================*/
    void DFS1(int pre, int cur)
    {
        node[cur].pre = pre;
        node[cur].dep = node[pre].dep + 1;
        for (auto nxt : G[cur])
        {
            if (nxt != pre)
            {
                DFS1(cur, nxt);
                node[cur].siz += node[nxt].siz;
                if (node[cur].son == -1)
                {
                    node[cur].son = nxt;
                }
                else if (node[nxt].siz > node[node[cur].son].siz)
                {
                    node[cur].son = nxt;
                }
            }
        }
    }
    void DFS2(int cur, int top)
    {
        node[cur].top = top;
        if (node[cur].son != -1)
        {
            DFS2(node[cur].son, top);
            for (auto nxt : G[cur])
            {
                if (nxt == node[cur].pre)continue;
                if (nxt == node[cur].son)continue;
                DFS2(nxt, nxt);
            }
        }
    }
public:
    void Init(int n, vector<int>G[], int root = 1)
    {
        this->G = G;
        this->root = root;
        node.assign(n + 1, Node());
        DFS1(root, root); DFS2(root, root);
    }
    int operator()(int u, int v)
    {
        while (node[u].top != node[v].top)
```

```
        {
            int topu = node[u].top;
            int topv = node[v].top;
            if (node[topu].dep > node[topv].dep)
            {
                u = node[topu].pre;
            }
            else
            {
                v = node[topv].pre;
            }
        }
        return node[u].dep > node[v].dep ? v : u;
    }
};
```

ST表法

```
class C_LCA
{
private:
    vector<int>* G;
    vector<int>dfn;
    vector<vector<int>>table;
    /*====================*/
    void DFS(int pre, int cur)
    {
        table[0][dfn[cur] = ++dfn[0]] = pre;
        for (auto nxt : G[cur])if (nxt != pre)DFS(cur, nxt);
    }
    int Get(int x, int y)
    {
        return dfn[x] < dfn[y] ? x : y;
    }
public:
    void Init(int n, vector<int>G[], int root = 1)
    {
        this->G = G; dfn.assign(n + 1, 0);
        table.assign(__lg(n) + 1, vector<int>(n + 1, 0));
        /*====================*/
        DFS(0, root);
        for (int d = 1; (1 << d) <= n; ++d)
        {
            for (int i = 1; i + (1 << d) - 1 <= n; ++i)
            {
                table[d][i] = Get(table[d - 1][i], table[d - 1][i + (1 << (d
- 1))]);
            }
        }
    }
    int operator()(int u, int v)
    {
        if (u == v)return u;
        if ((u = dfn[u]) > (v = dfn[v]))swap(u, v);
```

```
        int d = __lg(v - u++); return Get(table[d][u], table[d][v - (1 << d) + 1]);
    }
};
```

树哈希

```
class C_TreeHash
{
private:
    vector<int>* G;
    map<vector<int>, int>mp;
    /*====================*/
    int DFS(int pre, int cur)
    {
        vector<int>vec;
        for (auto nxt : G[cur])
        {
            if (nxt != pre)
            {
                vec.push_back(DFS(cur, nxt));
            }
        }
        sort(vec.begin(), vec.end());
        if (mp.find(vec) == mp.end())
        {
            mp[vec] = mp.size();
        }
        return mp[vec];
    }
public:
    int operator()(vector<int>G[], int root)
    {
        this->G = G;
        return DFS(root, root);
    }
};
```

树分治

点分治

```
class C_TCD
{
private:
    vector<int>* G;
    /*====================*/
    vector<int>siz;
    vector<bool>rooted;
    /*====================*/
    int centroid, ans_part;
    /*====================*/
    void DFS(int pre, int cur, int all)
```

```
    {
        siz[cur] = 1; int max_part = 0;
        for (auto nxt : G[cur])
        {
            if (nxt == pre)continue;
            if (rooted[nxt])continue;
            DFS(cur, nxt, all);
            siz[cur] += siz[nxt];
            max_part = max(max_part, siz[nxt]);
        }
        max_part = max(max_part, all - siz[cur]);
        if (max_part < ans_part)
        {
            ans_part = max_part, centroid = cur;
        }
    }
    int Centroid(int cur, int all)
    {
        centroid = -1; ans_part = all;
        DFS(cur, cur, all); return centroid;
    }
    /*====================*/
    void CalcSon(int pre, int cur)
    {
        /*
            添加cur到右树tree中
        */
        for (auto nxt : G[cur])
        {
            if (nxt == pre)continue;
            if (rooted[nxt])continue;
            CalcSon(cur, nxt);
        }
    }
    void CalcRoot(int root)
    {
        /*
            初始化左库lib
        */
        for (auto son : G[root])
        {
            if (!rooted[son])
            {
                /*
                    初始化右树tree
                */
                CalcSon(son, son);
                /*
                    遍历右树tree匹配左库lib
                */
                /*
                    添加右树tree到左库lib
                */
            }
        }
```

```
    }
    /*====================*/
    void DividTree(int root)
    {
        rooted[root] = true; CalcRoot(root);
        for (auto son : G[root])
        {
            if (!rooted[son])
            {
                DividTree(Centroid(son, siz[son]));
            }
        }
    }
public:
    void operator()(int n, vector<int>G[])
    {
        this->G = G;
        siz.assign(n + 1, 0);
        rooted.assign(n + 1, false);
        DividTree(Centroid(1, n));
    }
};
```

K级祖先

```
class C_KthAncestor
{
private:
    int n, root; vector<int>* G;
    /*====================*/
    struct Node
    {
        int pre, dep, siz, son;
        int top, dfn, idx;
        Node(void)
        {
            pre = -1; dep = +0;
            siz = +1; son = -1;
            dfn = +0, idx = +0;
            top = -1;
        }
    };
    /*====================*/
    vector<Node>node; int cnt;
    /*====================*/
    void DFS1(int pre, int cur)
    {
        node[cur].pre = pre;
        node[cur].dep = node[pre].dep + 1;
        for (auto nxt : G[cur])
        {
            if (nxt != pre)
            {
                DFS1(cur, nxt); node[cur].siz += node[nxt].siz;
```

```
                if (node[cur].son == -1)
                {
                    node[cur].son = nxt;
                }
                else if (node[nxt].siz > node[node[cur].son].siz)
                {
                    node[cur].son = nxt;
                }
            }
        }
    }
    void DFS2(int cur, int top)
    {
        node[cur].top = top;
        node[cur].dfn = ++cnt;
        node[cnt].idx = cur;
        if (node[cur].son != -1)
        {
            DFS2(node[cur].son, top);
            for (auto nxt : G[cur])
            {
                if (nxt == node[cur].pre)continue;
                if (nxt == node[cur].son)continue;
                DFS2(nxt, nxt);
            }
        }
    }
public:
    void Init(int n, vector<int>G[], int root = 1)
    {
        node.assign(n + 1, Node());
        this->n = n, this->root = root, this->G = G;
        cnt = 0; DFS1(root, root); DFS2(root, root);
    }
    int operator()(int x, int k)
    {
        int topx = node[x].top;
        while (k > 0)
        {
            topx = node[x].top;
            if (node[x].dep - node[topx].dep < k)
            {
                k -= node[x].dep - node[topx].dep + 1;
                x = node[topx].pre;
            }
            else
            {
                x = node[node[x].dfn - k].idx; k = 0;
            }
        }
        return x;
    }
};
```

树链剖分

重链剖分

```
class C_HLD
{
public:
    int n, root; vector<int>* G;
    /*====================*/
    struct Node
    {
        int pre, dep, siz, son;
        int top, dfn, idx;
        Node(void)
        {
            pre = -1; dep = +0;
            siz = +1; son = -1;
            dfn = +0, idx = +0;
            top = -1;
        }
    };
private:
    vector<Node>node; int cnt;
    /*====================*/
    void DFS1(int pre, int cur)
    {
        node[cur].pre = pre;
        node[cur].dep = node[pre].dep + 1;
        for (auto nxt : G[cur])
        {
            if (nxt != pre)
            {
                DFS1(cur, nxt); node[cur].siz += node[nxt].siz;
                if (node[cur].son == -1)
                {
                    node[cur].son = nxt;
                }
                else if (node[nxt].siz > node[node[cur].son].siz)
                {
                    node[cur].son = nxt;
                }
            }
        }
    }
    void DFS2(int cur, int top)
    {
        node[cur].top = top;
        node[cur].dfn = ++cnt;
        node[cnt].idx = cur;
        if (node[cur].son != -1)
        {
            DFS2(node[cur].son, top);
            for (auto nxt : G[cur])
            {
                if (nxt == node[cur].pre)continue;
                if (nxt == node[cur].son)continue;
                DFS2(nxt, nxt);
```

```
            }
        }
    }
public:
    void Init(int n, vector<int>G[], int root = 1)
    {
        node.assign(n + 1, Node());
        this->n = n, this->root = root, this->G = G;
        cnt = 0; DFS1(root, root); DFS2(root, root);
    }
    Node& operator[](int idx) { return node[idx]; }
};
```

树的重心

```
class C_TreeCentroid
{
private:
    vector<int>siz;
    vector<int>* G;
    int centroid, ans_part;
    /*====================*/
    void DFS(int pre, int cur, int all)
    {
        siz[cur] = 1; int max_part = 0;
        for (auto nxt : G[cur])
        {
            if (nxt != pre)
            {
                DFS(cur, nxt, all);
                siz[cur] += siz[nxt];
                max_part = max(max_part, siz[nxt]);
            }
        }
        max_part = max(max_part, all - siz[cur]);
        if (max_part < ans_part)
        {
            ans_part = max_part, centroid = cur;
        }
    }
public:
    int operator()(int n, vector<int>G[])
    {
        this->G = G; siz.assign(n + 1, 0);
        centroid = -1; ans_part = n;
        DFS(1, 1, n); return centroid;
    }
};
```

树上路径求交

假设当前要求路径 (a,b) 和 (c,d) 的交。
设 d_x 表示 x 的深度。
先求出 $p[4]=lca(a,c),lca(a,d),lca(b,c),lca(b,d)$。
将 p 数组按深度排序，取出深度较大的两个，记为 $p0,p1$。
若存在交，则 $(p0,p1)$ 即所求。
现在只需要判断路径是否有交。
若 $p0\neq p1$，则一定有交。
否则若 $d_{p0}=\max(d_{lca(a,b)},d_{lca(c,d)})$，也有交。
否则路径不相交。

树上启发式合并

对于以 u 为根的子树

①. 先统计它轻子树(轻儿子为根的子树)的答案，统计完后删除信息

②. 再统计它重子树(重儿子为根的子树)的答案，统计完后保留信息

③. 然后再将重子树的信息合并到 u上

④. 再去遍历 u 的轻子树，然后把轻子树的信息合并到 u 上

⑤. 判断 u 的信息是否需要传递给它的父节点（u 是否是它父节点的重儿子）

```
void DFS(int root, int cur)
{
    cnt[node[cur].val]++;
    if (cnt[node[cur].val] > maxcnt)
    {
        ans[root] = node[cur].val;
        maxcnt = cnt[node[cur].val];
    }
    else if (cnt[node[cur].val] == maxcnt)
    {
        ans[root] += node[cur].val;
    }
    for (auto nxt : G[cur])
    {
        if (nxt == node[cur].pre)continue;
        if (nxt == node[root].son)continue;
        DFS(root, nxt);
    }
}
void DSU(int cur, bool keep)
{
    for (auto nxt : G[cur])
    {
        if (nxt == node[cur].pre)continue;
        if (nxt == node[cur].son)continue;
        DSU(nxt, false);
    }
    if (node[cur].son != -1)DSU(node[cur].son, true);
    if (node[cur].son != -1)
    {
        ans[cur] = ans[node[cur].son];
    }
```

```
    DFS(cur, cur);
    if (keep == false)
    {
        maxcnt = 0;
        for (int i = node[cur].dfn; i < node[cur].dfn + node[cur].siz; ++i)
        {
            cnt[node[node[i].idx].val]--;
        }
    }
}
```

数据结构

莫队

```
int n, m;
/*====================*/
int S;
struct Query
{
    int l, r, idx;
    Query(int _l = 0, int _r = 0, int _idx = 0)
    {
        l = _l, r = _r, idx = _idx;
    }
    friend bool operator<(const Query& a, const Query& b)
    {
        return (a.l / S == b.l / S) ? (((a.l / S) & 1) ? (a.r > b.r) : (a.r < b.r)) : (a.l < b.l);
    }
}query[M];
/*====================*/
int ans[M];
/*====================*/
void Add(int pos)
{

}
void Del(int pos)
{

}
/*====================*/
void Solve(void)
{
    cin >> n >> m;
    S = n / sqrt(m) + 1;
    for (int i = 1; i <= m; ++i)
    {
        int l, r; cin >> l >> r;
        query[i] = Query(l, r, i);
    }
    sort(query + 1, query + 1 + m);
    int l = 1, r = 0;
```

```
    for (int i = 1; i <= m; ++i)
    {
        while (query[i].l < l)Add(--l);
        while (r < query[i].r)Add(++r);
        while (l < query[i].l)Del(l++);
        while (query[i].r < r)Del(r--);
        //获得ans[query[i].idx];
    }
}
```

猫树

```
#include<iostream>
using namespace std;

const int N = 1 << 20;
const int LEVEL = 20;

int a[N];
int lg[N];
int pos[N];
int MaoA[LEVEL][N];//最大子段和
int MaoB[LEVEL][N];//最大连续和

inline int ls(int p) { return p << 1; }
inline int rs(int p) { return (p << 1) | 1; }

void Build(int p, int l, int r, int level)
{
    if (l == r) { pos[l] = p; return; }
    int mid = (l + r) >> 1;
    int tempA, tempB;
    //the left
    MaoA[level][mid] = MaoB[level][mid] = a[mid];
    tempA = max(a[mid], 0), tempB = a[mid];
    for (int i = mid - 1; i >= l; --i)
    {
        tempA += a[i]; MaoA[level][i] = max(MaoA[level][i + 1], tempA);
tempA = max(tempA, 0);
        tempB += a[i]; MaoB[level][i] = max(MaoB[level][i + 1], tempB);
    }
    //the right
    MaoA[level][mid + 1] = MaoB[level][mid + 1] = a[mid + 1];
    tempA = max(a[mid + 1], 0), tempB = a[mid + 1];
    for (int i = mid + 2; i <= r; ++i)
    {
        tempA += a[i]; MaoA[level][i] = max(MaoA[level][i - 1], tempA);
tempA = max(tempA, 0);
        tempB += a[i]; MaoB[level][i] = max(MaoB[level][i - 1], tempB);
    }
    //
    Build(ls(p), l, mid, level + 1); Build(rs(p), mid + 1, r, level + 1);
}
int Ask(int l, int r)
```

```
{
    if (l == r)return a[l];
    int level = (lg[pos[l]] - lg[pos[l] ^ pos[r]]);
    return max(max(MaoA[level][l], MaoA[level][r]), MaoB[level][l] +
MaoB[level][r]);
}

int main()
{
    int n; cin >> n;
    int len = 1; while (len < n)len <<= 1;
    for (int i = 2; i < N; ++i)lg[i] = lg[i >> 1] + 1;
    for (int i = 1; i <= n; ++i)cin >> a[i];
    Build(1, 1, len, 1);
    int m; cin >> m;
    for (int i = 1; i <= m; ++i)
    {
        int l, r; cin >> l >> r;
        cout << Ask(l, r) << endl;
    }
    return 0;
}
```

ST表

```
template<class Type>
class C_ST
{
public:
    Type operator()(int l, int r)
    {
        int d = __lg(r - l + 1); return op(table[d][l], table[d][r - (1 <<
d) + 1]);
    }
    void Init(int n, Type arr[], Type(*op)(Type, Type))
    {
        this->op = op;
        /*====================*/
        table.assign(__lg(n) + 1, vector<Type>(n + 1, Type()));
        for (int i = 1; i <= n; ++i)table[0][i] = arr[i];
        /*====================*/
        for (int d = 1; (1 << d) <= n; ++d)
        {
            for (int i = 1; i + (1 << d) - 1 <= n; ++i)
            {
                table[d][i] = op(table[d - 1][i], table[d - 1][i + (1 << (d
- 1))]);
            }
        }
    }
private:
    Type(*op)(Type, Type);
    vector<vector<Type>>table;
};
```

扫描线

矩形面积并

```
namespace ScanLine
{
    const int N = 1e5 + 10;
    /*====================*/
    struct Rectangle
    {
        double x1, y1;
        double x2, y2;
    };
    Rectangle rectangle[N];
    /*====================*/
    vector<double>pos;
    /*====================*/
    struct Line
    {
        int val;
        int l, r;
        double h;
        Line(int _l = 0, int _r = 0, double _h = 0, int _val = 0)
        {
            l = _l, r = _r, h = _h, val = _val;
        }
        friend bool operator<(const Line& a, const Line& b)
        {
            if (a.h != b.h)
            {
                return a.h < b.h;
            }
            else
            {
                return  a.val > b.val;
            }
        }
    };
    vector<Line>line;
    /*====================*/
    struct Tree
    {
        int l, r;
        int cnt; double len;
    };
    Tree tree[N << 3];
    int ls(int p)
    {
        return p << 1;
    }
    int rs(int p)
    {
        return p << 1 | 1;
    }
```

```
51      void PushUp(int p)
52      {
53          if (tree[p].cnt >= 1)
54          {
55              tree[p].len = pos[tree[p].r] - pos[tree[p].l - 1];
56          }
57          else
58          {
59              if (tree[p].l != tree[p].r)
60              {
61                  tree[p].len = tree[ls(p)].len + tree[rs(p)].len;
62              }
63              else
64              {
65                  tree[p].len = 0;
66              }
67          }
68      }
69      void Build(int p, int l, int r)
70      {
71          tree[p].l = l, tree[p].r = r;
72          tree[p].cnt = 0; tree[p].len = 0;
73          if (tree[p].l != tree[p].r)
74          {
75              int mid = (tree[p].l + tree[p].r) >> 1;
76              Build(ls(p), l, mid + 0);
77              Build(rs(p), mid + 1, r);
78          }
79      }
80      void Change(int p, int l, int r, int d)
81      {
82          if (l <= tree[p].l && tree[p].r <= r)
83          {
84              tree[p].cnt += d; PushUp(p);
85          }
86          else
87          {
88              int mid = (tree[p].l + tree[p].r) >> 1;
89              if (l <= mid)Change(ls(p), l, r, d);
90              if (mid < r) Change(rs(p), l, r, d);
91              PushUp(p);
92          }
93      }
94      /*====================*/
95      double Init(void)
96      {
97          int n; cin >> n;
98          pos.clear(); line.clear();
99          for (int i = 1; i <= n; ++i)
100         {
101             double x1, y1; cin >> x1 >> y1;//左上
102             double x2, y2; cin >> x2 >> y2;//右下
103             pos.push_back(x1); pos.push_back(x2);
104             rectangle[i].x1 = x1; rectangle[i].y1 = y1;
105             rectangle[i].x2 = x2; rectangle[i].y2 = y2;
```

```
        }
        sort(pos.begin(), pos.end());
        pos.erase(unique(pos.begin(), pos.end()), pos.end());
        for (int i = 1; i <= n; ++i)
        {
            int l = lower_bound(pos.begin(), pos.end(), rectangle[i].x1) - pos.begin();
            int r = lower_bound(pos.begin(), pos.end(), rectangle[i].x2) - pos.begin();
            line.push_back(Line(l, r, rectangle[i].y1, +1));
            line.push_back(Line(l, r, rectangle[i].y2, -1));
        }
        sort(line.begin(), line.end());
        Build(1, 1, pos.size() - 1);
        bool flag = true;
        double ans = 0.0;
        double last = 0.0;
        auto it = line.begin();
        while (it != line.end())
        {
            double high = it->h;
            if (flag)last = high, flag = false;
            ans += (high - last) * tree[1].len;
            while (it != line.end() && it->h == high)
            {
                Change(1, it->l + 1, it->r, it->val); it++;
            }
            last = high;
        }
        return ans;
    }
}
```

矩形面积交

```
namespace ScanLine
{
    const int N = 1e5 + 10;
    /*====================*/
    struct Rectangle
    {
        double x1, y1;
        double x2, y2;
    };
    Rectangle rectangle[N];
    /*====================*/
    vector<double>pos;
    /*====================*/
    struct Line
    {
        int val;
        int l, r;
        double h;
        Line(int _l = 0, int _r = 0, double _h = 0, int _val = 0)
        {
```

```
            l = _l, r = _r, h = _h, val = _val;
        }
        friend bool operator<(const Line& a, const Line& b)
        {
            if (a.h != b.h)
            {
                return a.h < b.h;
            }
            else
            {
                return  a.val > b.val;
            }
        }
    };
    vector<Line>line;
    /*====================*/
    struct Tree
    {
        int cnt;
        int l, r;
        double len1;
        double len2;
    };
    Tree tree[N << 3];
    int ls(int p)
    {
        return p << 1;
    }
    int rs(int p)
    {
        return p << 1 | 1;
    }
    void PushUp(int p)
    {
        if (tree[p].cnt >= 1)
        {
            tree[p].len1 = pos[tree[p].r] - pos[tree[p].l - 1];
        }
        else
        {
            if (tree[p].l != tree[p].r)
            {
                tree[p].len1 = tree[ls(p)].len1 + tree[rs(p)].len1;
            }
            else
            {
                tree[p].len1 = 0;
            }
        }
        if (tree[p].cnt >= 2)
        {
            tree[p].len2 = pos[tree[p].r] - pos[tree[p].l - 1];
        }
        else
        {
```

```
            if (tree[p].l != tree[p].r)
            {
                if (tree[p].cnt == 1)
                {
                    tree[p].len2 = tree[ls(p)].len1 + tree[rs(p)].len1;
                }
                else
                {
                    tree[p].len2 = tree[ls(p)].len2 + tree[rs(p)].len2;
                }
            }
            else
            {
                tree[p].len2 = 0;
            }
        }
    }
    void Build(int p, int l, int r)
    {
        tree[p].cnt = 0;
        tree[p].l = l, tree[p].r = r;
        tree[p].len1 = 0; tree[p].len2 = 0;
        if (tree[p].l != tree[p].r)
        {
            int mid = (tree[p].l + tree[p].r) >> 1;
            Build(ls(p), l, mid + 0);
            Build(rs(p), mid + 1, r);
        }
    }
    void Change(int p, int l, int r, int d)
    {
        if (l <= tree[p].l && tree[p].r <= r)
        {
            tree[p].cnt += d; PushUp(p);
        }
        else
        {
            int mid = (tree[p].l + tree[p].r) >> 1;
            if (l <= mid)Change(ls(p), l, r, d);
            if (mid < r) Change(rs(p), l, r, d);
            PushUp(p);
        }
    }
    /*====================*/
    double Init(void)
    {
        int n; cin >> n;
        pos.clear(); line.clear();
        for (int i = 1; i <= n; ++i)
        {
            double x1, y1; cin >> x1 >> y1;//左上
            double x2, y2; cin >> x2 >> y2;//右下
            pos.push_back(x1); pos.push_back(x2);
            rectangle[i].x1 = x1; rectangle[i].y1 = y1;
            rectangle[i].x2 = x2; rectangle[i].y2 = y2;
```

```
        }
        sort(pos.begin(), pos.end());
        pos.erase(unique(pos.begin(), pos.end()), pos.end());
        for (int i = 1; i <= n; ++i)
        {
            int l = lower_bound(pos.begin(), pos.end(), rectangle[i].x1) -
pos.begin();
            int r = lower_bound(pos.begin(), pos.end(), rectangle[i].x2) -
pos.begin();
            line.push_back(Line(l, r, rectangle[i].y1, +1));
            line.push_back(Line(l, r, rectangle[i].y2, -1));
        }
        sort(line.begin(), line.end());
        Build(1, 1, pos.size() - 1);
        bool flag = true;
        double ans = 0.0;
        double last = 0.0;
        auto it = line.begin();
        while (it != line.end())
        {
            double high = it->h;
            if (flag)last = high, flag = false;
            ans += (high - last) * tree[1].len2;
            while (it != line.end() && it->h == high)
            {
                Change(1, it->l + 1, it->r, it->val); it++;
            }
            last = high;
        }
        return ans;
    }
}
```

矩形周长并

```
namespace ScanLine
{
    const int N = 1e5 + 10;
    /*====================*/
    struct Rectangle
    {
        double x1, y1;
        double x2, y2;
    };
    Rectangle rectangle[N];
    /*====================*/
    vector<double>pos;
    /*====================*/
    struct Line
    {
        int val;
        int l, r;
        double h;
        Line(int _l = 0, int _r = 0, double _h = 0, int _val = 0)
        {
```

```
            l = _l, r = _r, h = _h, val = _val;
        }
        friend bool operator<(const Line& a, const Line& b)
        {
            if (a.h != b.h)
            {
                return a.h < b.h;
            }
            else
            {
                return  a.val > b.val;
            }
        }
    };
    vector<Line>line;
    typedef vector<Line>::iterator iter;
    /*====================*/
    struct Tree
    {
        int l, r;
        int cnt; double len;
    };
    Tree tree[N << 3];
    int ls(int p)
    {
        return p << 1;
    }
    int rs(int p)
    {
        return p << 1 | 1;
    }
    void PushUp(int p)
    {
        if (tree[p].cnt >= 1)
        {
            tree[p].len = pos[tree[p].r] - pos[tree[p].l - 1];
        }
        else
        {
            if (tree[p].l != tree[p].r)
            {
                tree[p].len = tree[ls(p)].len + tree[rs(p)].len;
            }
            else
            {
                tree[p].len = 0;
            }
        }
    }
    void Build(int p, int l, int r)
    {
        tree[p].l = l, tree[p].r = r;
        tree[p].cnt = 0; tree[p].len = 0;
        if (tree[p].l != tree[p].r)
        {
```

```
            int mid = (tree[p].l + tree[p].r) >> 1;
            Build(ls(p), l, mid + 0);
            Build(rs(p), mid + 1, r);
        }
    }
    void Change(int p, int l, int r, int d)
    {
        if (l <= tree[p].l && tree[p].r <= r)
        {
            tree[p].cnt += d; PushUp(p);
        }
        else
        {
            int mid = (tree[p].l + tree[p].r) >> 1;
            if (l <= mid)Change(ls(p), l, r, d);
            if (mid < r) Change(rs(p), l, r, d);
            PushUp(p);
        }
    }
    /*====================*/
    double Init(void)
    {
        int n; cin >> n; double ans = 0;
        for (int i = 1; i <= n; ++i)
        {
            double x1, y1; cin >> x1 >> y1;//左上
            double x2, y2; cin >> x2 >> y2;//右下
            rectangle[i].x1 = x1; rectangle[i].y1 = y1;
            rectangle[i].x2 = x2; rectangle[i].y2 = y2;
        }
        /*====================*/
        pos.clear(); line.clear();
        for (int i = 1; i <= n; ++i)
        {
            pos.push_back(rectangle[i].x1);
            pos.push_back(rectangle[i].x2);
        }
        sort(pos.begin(), pos.end());
        pos.erase(unique(pos.begin(), pos.end()), pos.end());
        for (int i = 1; i <= n; ++i)
        {
            int l = lower_bound(pos.begin(), pos.end(), rectangle[i].x1) -
pos.begin();
            int r = lower_bound(pos.begin(), pos.end(), rectangle[i].x2) -
pos.begin();
            line.push_back(Line(l, r, rectangle[i].y1, +1));
            line.push_back(Line(l, r, rectangle[i].y2, -1));
        }
        sort(line.begin(), line.end());
        Build(1, 1, pos.size() - 1);
        double last1 = 0;
        for (iter it = line.begin(); it != line.end(); ++it)
        {
            Change(1, it->l + 1, it->r, it->val);
            ans += abs(tree[1].len - last1); last1 = tree[1].len;
```

```
        }
        /*====================*/
        pos.clear(); line.clear();
        for (int i = 1; i <= n; ++i)
        {
            pos.push_back(rectangle[i].y1);
            pos.push_back(rectangle[i].y2);
        }
        sort(pos.begin(), pos.end());
        pos.erase(unique(pos.begin(), pos.end()), pos.end());
        for (int i = 1; i <= n; ++i)
        {
            int l = lower_bound(pos.begin(), pos.end(), rectangle[i].y1) - pos.begin();
            int r = lower_bound(pos.begin(), pos.end(), rectangle[i].y2) - pos.begin();
            line.push_back(Line(l, r, rectangle[i].x1, +1));
            line.push_back(Line(l, r, rectangle[i].x2, -1));
        }
        sort(line.begin(), line.end());
        Build(1, 1, pos.size() - 1);
        double last2 = 0;
        for (iter it = line.begin(); it != line.end(); ++it)
        {
            Change(1, it->l + 1, it->r, it->val);
            ans += abs(tree[1].len - last2); last2 = tree[1].len;
        }
        /*====================*/
        return ans;
    }
}
```

可删堆

```
template<class Type, class Comp = greater<Type>>
class C_Heap
{
private:
    priority_queue<Type, vector<Type>, Comp>heap1;
    priority_queue<Type, vector<Type>, Comp>heap2;
public:
    Type Top(void)
    {
        while (!heap2.Empty() && heap1.Top() == heap2.Top())
        {
            heap1.Pop(); heap2.Pop();
        }
        return heap1.Top();
    }
    void Pop(void)
    {
        while (!heap2.Empty() && heap1.Top() == heap2.Top())
        {
            heap1.Pop(); heap2.Pop();
```

```
        }
        heap1.Pop();
    }
    int Size(void)
    {
        return heap1.Size() - heap2.Size();
    }
    void Clear(void)
    {
        while (!heap1.Empty())heap1.Pop();
        while (!heap2.Empty())heap2.Pop();
    }
    bool Empty(void)
    {
        return heap1.Size() == heap2.Size();
    }
    void Erase(Type val)
    {
        heap2.push(val);
    }
    void Insert(Type val)
    {
        heap1.push(val);
    }
};
```

并查集

```
class C_DSU
{
private:
    vector<int>pre, siz;
    /*====================*/
    int Find(int cur)
    {
        return cur == pre[cur] ? cur : pre[cur] = Find(pre[cur]);
    }
public:
    void Init(int n)
    {
        pre.assign(n + 1, 0);siz.assign(n + 1, 1);
        for (int i = 0; i <= n; ++i)pre[i] = i;
    }
    int operator[](int cur)
    {
        return Find(cur);
    }
    void operator()(int u, int v)
    {
        u = Find(u), v = Find(v);
        if (siz[u] < siz[v])
        {
            pre[u] = v, siz[v] += siz[u];
        }
```

```
        else
        {
            pre[v] = u, siz[u] += siz[v];
        }
    }
};
```

主席树

```
template<class Valu>
class C_ChairmanTree
{
private:
    struct Tree
    {
        int cnt;
        int ls, rs;
        Tree(void)
        {
            cnt = 0;
            ls = rs = -1;
        }
    };
    /*====================*/
    vector<int>root;
    vector<Tree>tree;
    /*====================*/
    int Creat(void)
    {
        tree.push_back(Tree());
        return tree.size() - 1;
    }
    void Build(int& cur, int treel, int treer)
    {
        cur = Creat();
        if (treel == treer)return;
        int mid = (treel + treer) >> 1;
        Build(tree[cur].ls, treel, mid + 0);
        Build(tree[cur].rs, mid + 1, treer);
    }
    void Insert(int pre, int& cur, int x, int treel, int treer)
    {
        cur = Creat();
        tree[cur] = tree[pre]; tree[cur].cnt++;
        if (treel == treer)return;
        int mid = (treel + treer) >> 1;
        if (x <= mid)Insert(tree[pre].ls, tree[cur].ls, x, treel, mid + 0);
        if (mid < x) Insert(tree[pre].rs, tree[cur].rs, x, mid + 1, treer);
    }
    int Ask(int pre, int cur, int k, int treel, int treer)
    {
        if (treel == treer)return treel;
        int mid = (treel + treer) >> 1;
        int cnt = tree[tree[cur].ls].cnt - tree[tree[pre].ls].cnt;
```

```
        if (cnt < k)return Ask(tree[pre].rs, tree[cur].rs, k - cnt, mid + 1,
treer);
        else return Ask(tree[pre].ls, tree[cur].ls, k, treel, mid + 0);
    }
    /*====================*/
    vector<Valu>lib;
public:
    void Init(int n, Valu arr[])
    {
        for (int i = 1; i <= n; ++i)
        {
            lib.push_back(arr[i]);
        }
        sort(lib.begin(), lib.end());
        lib.erase(unique(lib.begin(), lib.end()), lib.end());
        /*====================*/
        tree.reserve(2 * lib.size() + n * (ceil(log2(lib.size())) + 1));
        /*====================*/
        root.assign(n + 1, -1);
        Build(root[0], 1, lib.size());
        for (int i = 1; i <= n; ++i)
        {
            int x = lower_bound(lib.begin(), lib.end(), arr[i]) -
lib.begin() + 1;
            Insert(root[i - 1], root[i], x, 1, lib.size());
        }
    }
    Valu operator()(int l, int r, int k)
    {
        return lib[Ask(root[l - 1], root[r], k, 1, lib.size()) - 1];
    }
};
```

平衡树

带旋·Treap·权值树

```
template<class Valu>
class C_Treap
{
private:
    struct Tree
    {
        int siz;
        Valu valu;
        int priority;
        Tree* lch, * rch;
        Tree(void)
        {
            siz = 0;
            valu = Valu();
            lch = rch = NULL;
            priority = rand();
        }
```

```
    };
    /*====================*/
    Tree* null, * root;
    /*====================*/
    Tree* Creat(Valu valu)
    {
        Tree* node = new Tree;
        node->valu = valu;
        node->lch = null;
        node->rch = null;
        node->siz = 1;
        return node;
    }
    /*====================*/
    void PushUp(Tree* cur)
    {
        cur->siz = cur->lch->siz + cur->rch->siz + 1;
    }
    /*====================*/
    void LRotate(Tree*& cur)
    {
        Tree* son = cur->rch;
        cur->rch = son->lch; son->lch = cur; cur = son;
        PushUp(cur->lch); PushUp(cur);
    }
    void RRotate(Tree*& cur)
    {
        Tree* son = cur->lch;
        cur->lch = son->rch; son->rch = cur; cur = son;
        PushUp(cur->rch); PushUp(cur);
    }
    /*====================*/
    void Insert(Tree*& cur, Valu valu)
    {
        if (cur == null)
        {
            cur = Creat(valu);
        }
        else
        {
            if (valu < cur->valu)
            {
                Insert(cur->lch, valu);
                if (cur->priority < cur->lch->priority)
                {
                    RRotate(cur);
                }
            }
            else
            {
                Insert(cur->rch, valu);
                if (cur->priority < cur->rch->priority)
                {
                    LRotate(cur);
                }
```

```
            }
            PushUp(cur);
        }
    }
    void Delete(Tree*& cur, Valu valu)
    {
        if (cur == null)return;
        if (valu == cur->valu)
        {
            if (cur->lch != null && cur->rch != null)
            {
                if (cur->lch->priority < cur->rch->priority)
                {
                    LRotate(cur); Delete(cur->lch, valu); PushUp(cur);
                }
                else
                {
                    RRotate(cur); Delete(cur->rch, valu); PushUp(cur);
                }
            }
            else if (cur->lch != null)
            {
                RRotate(cur); Delete(cur->rch, valu); PushUp(cur);
            }
            else if (cur->rch != null)
            {
                LRotate(cur); Delete(cur->lch, valu); PushUp(cur);
            }
            else
            {
                delete cur; cur = null;
            }
        }
        else
        {
            if (valu < cur->valu)
            {
                Delete(cur->lch, valu);
            }
            else
            {
                Delete(cur->rch, valu);
            }
            PushUp(cur);
        }
    }
    /*====================*/
    Valu GetValuByRank(int rank)
    {
        Tree* cur = root;
        while (cur != null)
        {
            if (cur->lch->siz + 1 == rank)
            {
                return cur->valu;
```

```
            }
            else
            {
                if (cur->lch->siz < rank)
                {
                    rank -= cur->lch->siz + 1;
                    cur = cur->rch;
                }
                else
                {
                    cur = cur->lch;
                }
            }
        }
        return Valu();
    }
    int GetRankByValu(Valu valu)
    {
        int res = 1;
        Tree* cur = root;
        while (cur != null)
        {
            if (cur->valu < valu)
            {
                res += cur->lch->siz + 1;
                cur = cur->rch;
            }
            else
            {
                cur = cur->lch;
            }
        }
        return res;
    }
    /*====================*/
    void Clear(Tree* cur)
    {
        if (cur != null)
        {
            Clear(cur->lch);
            Clear(cur->rch);
            delete cur;
        }
    }
public:
    C_Treap(void)
    {
        root = null = new Tree;
    }
    ~C_Treap(void)
    {
        Clear(root); delete null;
    }
    /*====================*/
    int Size(void)
```

```
    {
        return root->siz;
    }
    void Clear(void)
    {
        Clear(root); root = null;
    }
    bool Empty(void)
    {
        return root->siz == 0;
    }
    void Erase(Valu valu)
    {
        Delete(root, valu);
    }
    void Insert(Valu valu)
    {
        Insert(root, valu);
    }
    int operator()(Valu valu)
    {
        return GetRankByValu(valu);
    }
    Valu operator[](int rank)
    {
        return GetValuByRank(rank);
    }
};
```

无旋·Treap·序列树

```
template<class Valu>
class C_Treap
{
public:
    struct Tree
    {
        int siz = 0;
        Valu valu = Valu();
        int priority = rand();
        Tree* lch = NULL, * rch = NULL;
    };
    /*====================*/
    Tree* null, * root;
    /*====================*/
    Tree* Creat(Valu valu)
    {
        Tree* node = new Tree;
        node->valu = valu;
        node->lch = null;
        node->rch = null;
        node->siz = 1;
        return node;
    }
    /*====================*/
```

```
    void PushUp(Tree* cur)
    {
        cur->siz = cur->lch->siz + cur->rch->siz + 1;
    }
    void PushDown(Tree* cur)
    {
        /*
        * 预留
        */
    }
    /*====================*/
    Tree* Build(int l, int r, Valu arr[])
    {
        if (l > r)return null;
        int mid = (l + r) >> 1;
        Tree* cur = Creat(arr[mid]);
        cur->lch = Build(l, mid - 1, arr);
        cur->rch = Build(mid + 1, r, arr);
        PushUp(cur); return cur;
    }
    Tree* Build(int l, int r, vector<Valu>& arr)
    {
        if (l > r)return null;
        int mid = (l + r) >> 1;
        Tree* cur = Creat(arr[mid]);
        cur->lch = Build(l, mid - 1, arr);
        cur->rch = Build(mid + 1, r, arr);
        PushUp(cur); return cur;
    }
    /*====================*/
    void Lower_Split(Tree* cur, int index, Tree*& ltree, Tree*&
rtree)//index留在rtree
    {
        if (cur == null)
        {
            ltree = rtree = null; return;
        }
        PushDown(cur);
        if (cur->lch->siz + 1 < index)
        {
            Lower_Split(cur->rch, index - cur->lch->siz - 1, ltree, rtree);
            cur->rch = ltree; PushUp(cur); ltree = cur;
        }
        else
        {
            Lower_Split(cur->lch, index, ltree, rtree);
            cur->lch = rtree; PushUp(cur); rtree = cur;
        }
    }
    void Upper_Split(Tree* cur, int index, Tree*& ltree, Tree*&
rtree)//index留在ltree
    {
        if (cur == null)
        {
            ltree = rtree = null; return;
```

```
        }
        PushDown(cur);
        if (cur->lch->siz < index)
        {
            Upper_Split(cur->rch, index - cur->lch->siz - 1, ltree, rtree);
            cur->rch = ltree; PushUp(cur); ltree = cur;
        }
        else
        {
            Upper_Split(cur->lch, index, ltree, rtree);
            cur->lch = rtree; PushUp(cur); rtree = cur;
        }
    }
    /*====================*/
    Tree* Merge(Tree* ltree, Tree* rtree)
    {
        if (ltree == null)return rtree;
        if (rtree == null)return ltree;
        PushDown(ltree); PushDown(rtree);
        if (ltree->priority < rtree->priority)
        {
            rtree->lch = Merge(ltree, rtree->lch);
            PushUp(rtree); return rtree;
        }
        else
        {
            ltree->rch = Merge(ltree->rch, rtree);
            PushUp(ltree); return ltree;
        }
    }
    /*====================*/
    void Split(int l, int r, Tree*& ltree, Tree*& mtree, Tree*& rtree)
    {
        Tree* o1, * o2, * o3, * o4;
        Upper_Split(root, r, o1, o2);
        Lower_Split(o1, l, o3, o4);
        ltree = o3, mtree = o4, rtree = o2;
    }
    void Merge(Tree* ltree, Tree* mtree, Tree* rtree)
    {
        root = Merge(Merge(ltree, mtree), rtree);
    }
    /*====================*/
    Valu operator[](int rank)
    {
        Tree* cur = root;
        while (cur != null)
        {
            PushDown(cur);
            if (cur->lch->siz + 1 == rank)
            {
                return cur->valu;
            }
            else
            {
```

```
                if (cur->lch->siz < rank)
                {
                    rank -= cur->lch->siz + 1;
                    cur = cur->rch;
                }
                else
                {
                    cur = cur->lch;
                }
            }
        }
        return Valu();
    }
    /*====================*/
    C_Treap(void)
    {
        root = null = new Tree;
    }
    void Clear(Tree* cur)
    {
        if (cur != null)
        {
            Clear(cur->lch);
            Clear(cur->rch);
            delete cur;
        }
    }
    ~C_Treap(void)
    {
        Clear(root); delete null;
    }
};
```

双旋·Splay·权值树

```
template<class Type>
class Splay
{
public:
    ~Splay(void)
    {
        Clear(root);
        delete null;
    }
    int size(void)
    {
        return count;
    }
    void clear(void)
    {
        Clear(root);
        root = null;
    }
    bool empty(void)
    {
```

```
        return count == 0;
    }
    Type pre(Type val)
    {
        root = splay(FindPre(root, val));
        return root->val;
    }
    Type nxt(Type val)
    {
        root = splay(FindNxt(root, val));
        return root->val;
    }
    void erase(Type val)
    {
        count--; root = Delete(FindByValu(root, val));
    }
    void insert(Type val)
    {
        count++; root = splay(Insert(root, val));
    }
    int operator()(Type val)
    {
        root = splay(FindByValu(root, val));
        return root->lch->siz + 1;
    }
    Type operator[](int rank)
    {
        root = splay(FindByRank(root, rank));
        return root->val;
    }
    Type lower_bound(Type val)
    {
        root = splay(FindLower(root, val));
        return root->val;
    }
    Type upper_bound(Type val)
    {
        root = splay(FindUpper(root, val));
        return root->val;
    }
private:
    struct Node
    {
        int siz = 0;
        Type val = Type();
        Node* fa = NULL;
        Node* lch = NULL;
        Node* rch = NULL;
    };
    /*====================*/
    typedef bool CHILD;
    const CHILD LCH = true;
    const CHILD RCH = false;
    /*====================*/
    int count = 0;
```

```
76      Node* null = new Node;
77      Node* root = null;
78      /*====================*/
79      CHILD Child(Node* cur)
80      {
81          Node* pre = cur->fa;
82          if (pre->lch == cur)
83          {
84              return LCH;
85          }
86          else
87          {
88              return RCH;
89          }
90      }
91
92      void PushUp(Node* cur)
93      {
94          cur->siz = cur->lch->siz + cur->rch->siz + 1;
95      }
96
97      void Del(Node* cur, Node* pre, CHILD WCH)
98      {
99          cur->fa = null;
100         if (WCH == LCH)pre->lch = null;
101         if (WCH == RCH)pre->rch = null;
102     }
103     void Add(Node* cur, Node* pre, CHILD WCH)
104     {
105         cur->fa = pre;
106         if (WCH == LCH)pre->lch = cur;
107         if (WCH == RCH)pre->rch = cur;
108     }
109
110     void LRotate(Node* cur)
111     {
112         Node* pre = cur->fa, * nxt = cur->lch, * anc = pre->fa;
113         CHILD WCH = Child(pre);
114         Del(nxt, cur, LCH); Del(cur, pre, RCH); Del(pre, anc, WCH);
115         Add(nxt, pre, RCH); Add(pre, cur, LCH); Add(cur, anc, WCH);
116         PushUp(pre); PushUp(cur);
117     }
118     void RRotate(Node* cur)
119     {
120         Node* pre = cur->fa, * nxt = cur->rch, * anc = pre->fa;
121         CHILD WCH = Child(pre);
122         Del(nxt, cur, RCH); Del(cur, pre, LCH); Del(pre, anc, WCH);
123         Add(nxt, pre, LCH); Add(pre, cur, RCH); Add(cur, anc, WCH);
124         PushUp(pre); PushUp(cur);
125     }
126
127     void Rotate(Node* cur)
128     {
129         if (Child(cur) == LCH)
130         {
```

```
            RRotate(cur);
        }
        else
        {
            LRotate(cur);
        }
    }

    void Clear(Node* cur)
    {
        if (cur != null)
        {
            Clear(cur->lch);
            Clear(cur->rch);
            delete cur;
        }
    }

    Node* Creat(Type val)
    {
        Node* cur = new Node;
        cur->lch = null;
        cur->rch = null;
        cur->fa = null;
        cur->val = val;
        cur->siz = 1;
        return cur;
    }

    Node* splay(Node* cur)
    {
        while (true)
        {
            Node* pre = cur->fa;
            if (cur->fa == null)break;
            if (pre->fa == null)break;
            CHILD CHpre = Child(pre);
            CHILD CHcur = Child(cur);
            if (CHpre == CHcur)
            {
                Rotate(pre); Rotate(cur); continue;
            }
            if (CHpre != CHcur)
            {
                Rotate(cur); Rotate(cur); continue;
            }
        }
        if (cur->fa != null)Rotate(cur); return cur;
    }

    Node* Insert(Node* cur, Type val)
    {
        CHILD WCH = LCH; Node* pre = null;
        while (cur != null)
        {
```

```
            if (val < cur->val)
            {
                pre = cur; cur = cur->lch; WCH = LCH;
            }
            else
            {
                pre = cur; cur = cur->rch; WCH = RCH;
            }
        }
        cur = Creat(val); Add(cur, pre, WCH); return cur;
    }

    Node* Delete(Node* cur)
    {
        splay(cur);
        Node* lch = cur->lch;
        Node* rch = cur->rch;
        delete cur; return Merge(lch, rch);
    }

    Node* Merge(Node* ltree, Node* rtree)
    {
        if (ltree == null)
        {
            rtree->fa = null; return rtree;
        }
        if (rtree == null)
        {
            ltree->fa = null; return ltree;
        }
        ltree->fa = null; rtree->fa = null;
        if (ltree->siz < rtree->siz)
        {
            Node* cur = FindMax(ltree); splay(cur);
            Add(rtree, cur, RCH); PushUp(cur); return cur;
        }
        else
        {
            Node* cur = FindMin(rtree); splay(cur);
            Add(ltree, cur, LCH); PushUp(cur); return cur;
        }
    }

    Node* FindByValu(Node* cur, Type val)
    {
        Node* res = null;
        while (cur != null)
        {
            if (val == cur->val)
            {
                res = cur, cur = cur->lch;
            }
            else
            {
                if (val < cur->val)
```

```
                {
                    cur = cur->lch;
                }
                else
                {
                    cur = cur->rch;
                }
            }
        }
        return res;
    }
    Node* FindByRank(Node* cur, int rank)
    {
        while (cur != null)
        {
            if (cur->lch->siz + 1 == rank)
            {
                return cur;
            }
            else
            {
                if (cur->lch->siz < rank)
                {
                    rank -= cur->lch->siz + 1;
                    cur = cur->rch;
                }
                else
                {
                    cur = cur->lch;
                }
            }
        }
        return null;
    }

    Node* FindLower(Node* cur, Type val)
    {
        Node* res = null;
        while (cur != null)
        {
            if (cur->val < val)
            {
                cur = cur->rch;
            }
            else
            {
                res = cur;
                cur = cur->lch;
            }
        }
        return res;
    }
    Node* FindUpper(Node* cur, Type val)
    {
        Node* res = null;
```

```
296         while (cur != null)
297         {
298             if (val < cur->val)
299             {
300                 res = cur;
301                 cur = cur->lch;
302             }
303             else
304             {
305                 cur = cur->rch;
306             }
307         }
308         return res;
309     }
310 
311     Node* FindMin(Node* cur)
312     {
313         while (cur->lch != null)
314         {
315             cur = cur->lch;
316         }
317         return cur;
318     }
319     Node* FindMax(Node* cur)
320     {
321         while (cur->rch != null)
322         {
323             cur = cur->rch;
324         }
325         return cur;
326     }
327 
328     Node* FindPre(Node* cur, Type val)
329     {
330         Node* res = null;
331         while (cur != null)
332         {
333             if (cur->val < val)
334             {
335                 res = cur;
336                 cur = cur->rch;
337             }
338             else
339             {
340                 cur = cur->lch;
341             }
342         }
343         return res;
344     }
345     Node* FindNxt(Node* cur, Type val)
346     {
347         Node* res = null;
348         while (cur != null)
349         {
350             if (val < cur->val)
```

```
            {
                res = cur;
                cur = cur->lch;
            }
            else
            {
                cur = cur->rch;
            }
        }
        return res;
    }
};
```

珂朵莉树

```
namespace Chtholly
{
    struct Tree
    {
        int l, r;
        mutable int val;
        Tree(int _l = 0, int _r = 0, int _val = int())
        {
            l = _l, r = _r, val = _val;
        }
        friend bool operator<(const Tree& a, const Tree& b)
        {
            return a.l < b.l;
        }
        int len(void)const { return r - l + 1; }
    };
    /*====================*/
    int n; set<Tree>tree;
    /*====================*/
    set<Tree>::iterator Find(int pos)
    {
        return --tree.upper_bound(Tree(pos));
    }
    set<Tree>::iterator Split(int pos)//lower
    {
        if (pos > n)return tree.end();
        auto it = Find(pos); if (it->l == pos)return it;
        int l = it->l, r = it->r; auto val = it->val; tree.erase(it);
        tree.insert(Tree(l, pos - 1, val)); return tree.insert(Tree(pos, r,
val)).first;
    }
    /*====================*/
    void GetRange(int l, int r, auto& begin, auto& end)
    {
        end = Split(r + 1), begin = Split(l);
    }
    void EraseRange(int l, int r)
    {
```

```
        set<Tree>::iterator begin, end; GetRange(l, r, begin, end);
tree.erase(begin, end);
    }
    void InsertRange(int l, int r, int val)
    {
        EraseRange(l, r); auto it = tree.insert(Tree(l, r, val)).first;
        {auto __it = it; __it--; if (it != tree.begin() && __it->val ==
val)l = __it->l; }
        {auto __it = it; __it++; if (__it != tree.end() && __it->val ==
val)r = __it->r; }
        EraseRange(l, r); tree.insert(Tree(l, r, val));
    }
    /*====================*/
    void Build(int _n)
    {
        n = _n; tree.insert(Tree(1, n, 1));
    }
}
```

树状数组

权值树状数组

```
class C_FenwickTree
{
private:
    int n, m; vector<int>tree;
    /*====================*/
    int lowbit(int x) { return x & (-x); }
public:
    int Size(void)
    {
        return tree[0];
    }
    void Init(int n)
    {
        this->n = n; m = 0;
        tree.assign(n + 1, 0);
        while ((1 << (m + 1)) <= n)m++;
    }
    bool Empty(void)
    {
        return tree[0] == 0;
    }
    void Erase(int x)
    {
        tree[0]--;
        while (x <= n)
        {
            tree[x] -= 1; x += lowbit(x);
        }
    }
    void Insert(int x)
    {
```

```
        tree[0]++;
        while (x <= n)
        {
            tree[x] += 1; x += lowbit(x);
        }
    }
    int operator()(int valu)
    {
        valu--;
        int rank = 0;
        while (valu)
        {
            rank += tree[valu];
            valu -= lowbit(valu);
        }
        return rank + 1;
    }
    int operator[](int rank)
    {
        int sum = 0, valu = 0;
        for (int i = m; i >= 0; --i)
        {
            int temp = valu + (1 << i);
            if (temp <= n && sum + tree[temp] < rank)
            {
                sum += tree[temp]; valu = temp;
            }
        }
        return valu + 1;
    }
};
```

一维树状数组

```
template<class Type>
class FenwickTree
{
public:
    Type ask(int pos)
    {
        Type res = Type();
        while (pos)
        {
            res += tree[pos];
            pos -= lowbit(pos);
        }
        return res;
    }
    void init(int n)
    {
        this->n = n;
        tree = new Type[n + 1];
        for (int i = 0; i <= n; ++i)
        {
            tree[i] = Type();
```

```
        }
    }
    ~FenwickTree(void)
    {
        delete[] tree;
    }
    void add(int pos, Type d)
    {
        while (pos <= n)
        {
            tree[pos] += d;
            pos += lowbit(pos);
        }
    }
    Type ask(int l, int r)
    {
        Type res = Type(); l--;
        while (r > l)res += tree[r], r -= lowbit(r);
        while (l > r)res -= tree[l], l -= lowbit(l);
        return res;
    }
private:
    int n = 0;
    Type* tree = NULL;
    /*====================*/
    int lowbit(int x) { return x & (-x); }
};
```

二维树状数组

```
template<class Type>
class FenwickTree
{
public:
    ~FenwickTree(void)
    {
        for (int i = 0; i <= n; ++i)
        {
            delete[] tree[i];
        }
        delete[] tree;
    }
    Type ask(int x, int y)
    {
        Type res = Type();
        while (x)
        {
            int tempy = y;
            while (tempy)
            {
                res += tree[x][tempy];
                tempy -= lowbit(tempy);
            }
            x -= lowbit(x);
        }
```

```
        return res;
    }
    void init(int n, int m)
    {
        this->n = n;
        this->m = m;
        tree = new Type * [n + 1];
        for (int i = 0; i <= n; ++i)
        {
            tree[i] = new Type[m + 1];
            for (int j = 0; j <= m; ++j)
            {
                tree[i][j] = Type();
            }
        }
    }
    void add(int x, int y, Type d)
    {
        while (x <= n)
        {
            int tempy = y;
            while (tempy <= m)
            {
                tree[x][tempy] += d;
                tempy += lowbit(tempy);
            }
            x += lowbit(x);
        }
    }
    Type ask(int x1, int y1, int x2, int y2)
    {
        return ask(x2, y2) - ask(x1 - 1, y2) - ask(x2, y1 - 1) + ask(x1 - 1, y1 - 1);
    }
private:
    int n = 0, m = 0;
    Type** tree = NULL;
    /*====================*/
    int lowbit(int x) { return x & (-x); }
};
```

区间mex

```
class Range_MEX
{
public:
    Range_MEX(int n, int arr[], int l, int r)
    {
        root = new int[n + 1];
        tree = new Tree[4200000 + 10];

        for (int i = 0; i <= n; ++i)root[i] = -1;

        BuildZero(root[0], l, r);
```

```
        for (int i = 1; i <= n; ++i)
        {
            BuildChain(root[i - 1], root[i], i, arr[i]);
        }
    }
    int operator()(int l, int r)
    {
        return Ask(root[r], l);
    }
    ~Range_MEX(void)
    {
        delete[] tree;
        delete[] root;
    }
private:
    struct Tree
    {
        int idx;
        int l, r;
        int ls, rs;
        Tree(void)
        {
            idx = 0;
            l = 0, r = 0;
            ls = -1, rs = -1;
        }
    };
    /*====================*/
    int * root;
    Tree* tree; int cnt = -1;
    /*====================*/
    void PushUp(int cur)
    {
        tree[cur].idx = min(tree[tree[cur].ls].idx, tree[tree[cur].rs].idx);
    }
    /*====================*/
    void BuildZero(int& cur, int l, int r)
    {
        if (cur == -1)cur = ++cnt;
        /*====================*/
        tree[cur].l = l, tree[cur].r = r;
        if (l != r)
        {
            int mid = (l + r) >> 1;
            BuildZero(tree[cur].ls, l, mid + 0);
            BuildZero(tree[cur].rs, mid + 1, r);
        }
    }
    void BuildChain(int& pre, int& cur, int idx, int val)
    {
        if (cur == -1)cur = ++cnt;
        /*====================*/
        tree[cur].l = tree[pre].l, tree[cur].r = tree[pre].r;
        tree[cur].ls = tree[pre].ls, tree[cur].rs = tree[pre].rs;
        if (tree[cur].l == tree[cur].r)
```

```
        {
            tree[cur].idx = idx;
        }
        else
        {
            int mid = (tree[cur].l + tree[cur].r) >> 1;
            if (val <= mid)BuildChain(tree[pre].ls, tree[cur].ls = -1, idx, val);
            if (mid < val) BuildChain(tree[pre].rs, tree[cur].rs = -1, idx, val);
            PushUp(cur);
        }
    }
    /*====================*/
    int Ask(int& cur, int l)
    {
        if (tree[cur].l == tree[cur].r)return tree[cur].l;
        if (tree[tree[cur].ls].idx < l)return Ask(tree[cur].ls, l);
        if (tree[tree[cur].rs].idx < l)return Ask(tree[cur].rs, l);
        return tree[cur].r + 1;
    }
};
```

Hash_MAP

```
const int Base = 19260817;
class Hash_Map
{
public:
    Hash_Map()
    {
        memset(head, -1, sizeof(head));
        nxt.reserve(1e7);
        key.reserve(1e7);
        val.reserve(1e7);
    }
    lnt& operator[](lnt k)
    {
        int h = hash(k);
        for (int i = head[h]; ~i; i = nxt[i])
        {
            if (key[i] == k)
            {
                return val[i];
            }
        }
        nxt.push_back(head[h]);
        key.push_back(k);
        val.push_back(0);
        head[h] = nxt.size() - 1;
        return val.back();
    }
    lnt has_key(lnt k)
    {
```

```
        int h = hash(k);
        for (int i = head[h]; ~i; i = nxt[i])
        {
            if (key[i] == k)
            {
                return true;
            }
        }
        return false;
    }
private:
    int head[Base];
    vector<int>nxt;
    vector<lnt>key;
    vector<lnt>val;
    int hash(lnt k)
    {
        return k % Base;
    }
};
```

线段树合并

```
/*
* 与动态开点权值线段树搭配使用
*/
Tree* Merge(Tree*& a, Tree*& b, int treel, int treer)
{
    Tree* cur = null;
    if (a == null)
    {
        cur = b; b = null; return cur;
    }
    if (b == null)
    {
        cur = a; a = null; return cur;
    }
    cur = Creat();
    if (treel == treer)
    {
        cur->siz = a->siz + b->siz;
    }
    else
    {
        int mid = (treel + treer) >> 1;
        cur->ls = Merge(a->ls, b->ls, treel, mid + 0);
        cur->rs = Merge(a->rs, b->rs, mid + 1, treer);
        cur->siz = cur->ls->siz + cur->rs->siz;
    }
    delete a; a = null; delete b; b = null; return cur;
}
```

李超线段树

```
class C_LiChaoTree
{
private:
    struct Function
    {
        int k, b;
        int operator()(int x)
        {
            return k * x + b;
        }
        Function(int _k = 0, int _b = +INF)
        {
            k = _k, b = _b;
        }
    };
    /*====================*/
    struct Tree
    {
        Function f;
        Tree* ls, * rs;
        Tree(void)
        {
            ls = rs = NULL;
        }
    };
    /*====================*/
    Tree* root; int treel, treer;
    /*====================*/
    void PushDown(Tree*& cur, Function f, int treel, int treer)
    {
        if (cur == NULL)cur = new Tree;
        int mid = (treel + treer) >> 1;
        if (treel != treer)
        {
            if (cur->f.k < f.k)
            {
                if (f(mid) < cur->f(mid))
                {
                    PushDown(cur->rs, cur->f, treel, mid + 0);
                }
                else
                {
                    PushDown(cur->ls, f, mid + 1, treer);
                }
            }
            else
            {
                if (f(mid) < cur->f(mid))
                {
                    PushDown(cur->ls, cur->f, treel, mid + 0);
                }
                else
                {
                    PushDown(cur->rs, f, mid + 1, treer);
                }
```

```
            }
        }
        if (f(mid) < cur->f(mid))cur->f = f;
    }
    void Add(Tree*& cur, int l, int r, Function f, int treel, int treer)
    {
        if (cur == NULL)cur = new Tree;
        if (l <= treel && treer <= r)
        {
            PushDown(cur, f, treel, treer);
        }
        else
        {
            int mid = (treel + treer) >> 1;
            if (l <= mid)Add(cur->ls, l, r, f, treel, mid + 0);
            if (mid < r) Add(cur->rs, l, r, f, mid + 1, treer);
        }
    }
    int Ask(Tree* cur, int x, int treel, int treer)
    {
        int res = +INF;
        if (cur != NULL)
        {
            res = cur->f(x);
            if (treel != treer)
            {
                int mid = (treel + treer) >> 1;
                if (x <= mid)res = min(res, Ask(cur->ls, x, treel, mid +
0));
                if (mid < x) res = min(res, Ask(cur->rs, x, mid + 1,
treer));
            }
        }
        return res;
    }
public:
    C_LiChaoTree(void)
    {
        root = NULL, treel = treer = 0;
    }
    ~C_LiChaoTree(void)
    {
        if (root != NULL)
        {
            queue<Tree*>q; q.push(root);
            while (!q.empty())
            {
                if (q.front()->ls != NULL)q.push(q.front()->ls);
                if (q.front()->rs != NULL)q.push(q.front()->rs);
                delete q.front(); q.pop();
            }
        }
    }
    /*====================*/
    void Init(int treel, int treer)
```

```
    {
        this->treel = treel;
        this->treer = treer;
    }
    void operator()(int l, int r, Function  f)
    {
        Add(root, l, r, f, treel, treer);
    }
    int operator[](int x)
    {
        return Ask(root, x, treel, treer);
    }
};
```

可持久化01Trie

```
int root[N], tot;
int siz[35 * N];
int trie[35 * N][2];
/*====================*/
void Insert(int pre, int cur, lnt num)
{
    for (int i = 32; i >= 0; --i)
    {
        siz[cur] = siz[pre] + 1;
        int bit = (num & (1ll << i)) ? 1 : 0;
        trie[cur][bit ^ 1] = trie[pre][bit ^ 1], trie[cur][bit] = ++tot;
        pre = trie[pre][bit]; cur = trie[cur][bit];
    }
    siz[cur] = siz[pre] + 1;
}
lnt Query(int cur, lnt num, int k)
{
    lnt ans = 0;
    for (int i = 32; i >= 0; --i)
    {
        int bit = (num & (1ll << i)) ? 1 : 0;
        if (siz[trie[cur][bit ^ 1]] >= k)
        {
            ans += 1ll << i;
            cur = trie[cur][bit ^ 1];
        }
        else
        {
            k -= siz[trie[cur][bit ^ 1]];
            cur = trie[cur][bit];
        }
    }
    return ans;
}
```

Treap维护珂朵莉树

```

```

```
namespace Treap
{
    struct Node
    {
        Range range;
        int priority;
        int lch, rch;
    }node[N * 4];
    /*====================*/
    int null = -1;
    int root = -1;
    /*====================*/
    int Creat(Range range)
    {
        static int cnt = 0; ++cnt;
        node[cnt].lch = null;
        node[cnt].rch = null;
        node[cnt].priority = rand();
        node[cnt].range = range;
        return cnt;
    }
    /*====================*/
    void PushUp(int cur)
    {
        node[cur].range.L = node[cur].range.l;
        node[cur].range.R = node[cur].range.r;
        node[cur].range.Sum = node[cur].range.sum();
        if (node[cur].lch != null)
        {
            node[cur].range.L = node[node[cur].lch].range.L;
            node[cur].range.Sum += node[node[cur].lch].range.Sum;
        }
        if (node[cur].rch != null)
        {
            node[cur].range.R = node[node[cur].rch].range.R;
            node[cur].range.Sum += node[node[cur].rch].range.Sum;
        }
    }
    void PushDown(int cur)
    {
        if (node[cur].range.lazy != 0)
        {
            if (node[cur].lch != null)
            {
                node[node[cur].lch].range.Maintain(node[cur].range.lazy);
            }
            if (node[cur].rch != null)
            {
                node[node[cur].rch].range.Maintain(node[cur].range.lazy);
            }
            node[cur].range.lazy = 0;
        }
    }
    /*====================*/
    int Merge(int ltree, int rtree)
```

```
    {
        if (ltree == null)return rtree;
        if (rtree == null)return ltree;
        PushDown(ltree); PushDown(rtree);
        if (node[ltree].priority < node[rtree].priority)
        {
            node[rtree].lch = Merge(ltree, node[rtree].lch);
            /*======*/PushUp(rtree); return rtree;
        }
        else
        {
            node[ltree].rch = Merge(node[ltree].rch, rtree);
            /*======*/PushUp(ltree); return ltree;
        }
    }
    /*====================*/
    void Lower_Split(int cur, unt index, int& ltree, int& rtree)//index留在
rtree
    {
        if (cur == null)
        {
            ltree = rtree = null; return;
        }
        PushDown(cur);
        if (node[cur].range.r < index)
        {
            Lower_Split(node[cur].rch, index, ltree, rtree);
            node[cur].rch = ltree; PushUp(cur); ltree = cur;
        }
        else
        {
            Lower_Split(node[cur].lch, index, ltree, rtree);
            node[cur].lch = rtree; PushUp(cur); rtree = cur;
        }
    }
    void Upper_Split(int cur, unt index, int& ltree, int& rtree)//index留在
ltree
    {
        if (cur == null)
        {
            ltree = rtree = null; return;
        }
        PushDown(cur);
        if (node[cur].range.l > index)
        {
            Upper_Split(node[cur].lch, index, ltree, rtree);
            node[cur].lch = rtree; PushUp(cur); rtree = cur;
        }
        else
        {

            Upper_Split(node[cur].rch, index, ltree, rtree);
            node[cur].rch = ltree; PushUp(cur); ltree = cur;
        }
    }
```

```
    /*====================*/
    void SplitL(int root, unt index, int& ltree, int& rtree)
    {
        int _temp, _ltree, _mtree, _rtree;
        Upper_Split(root, index, _temp, _rtree);
        Lower_Split(_temp, index, _ltree, _mtree);
        unt l = node[_mtree].range.l;
        unt r = node[_mtree].range.r;
        unt val = node[_mtree].range.val;
        if (l != index)
        {
            ltree = Merge(_ltree, Creat(Range(l, index - 1, val)));
            rtree = Merge(Creat(Range(index + 0, r, val)), _rtree);
        }
        else
        {
            ltree = _ltree;
            rtree = Merge(_mtree, _rtree);
        }
    }
    void SplitR(int root, unt index, int& ltree, int& rtree)
    {
        int _temp, _ltree, _mtree, _rtree;
        Upper_Split(root, index, _temp, _rtree);
        Lower_Split(_temp, index, _ltree, _mtree);
        unt l = node[_mtree].range.l;
        unt r = node[_mtree].range.r;
        unt val = node[_mtree].range.val;
        if (r != index)
        {
            ltree = Merge(_ltree, Creat(Range(l, index + 0, val)));
            rtree = Merge(Creat(Range(index + 1, r, val)), _rtree);
        }
        else
        {
            ltree = Merge(_ltree, _mtree);
            rtree = _rtree;
        }
    }
}
```

动态开点权值线段树

```
class C_SegmentTree
{
private:
    struct Tree
    {
        int siz;
        Tree* ls, * rs;
        Tree(void)
        {
            siz = 0; ls = rs = NULL;
        }
```

```
    };
    /*====================*/
    Tree* null;
    /*====================*/
    Tree* root; int treel, treer;
    /*====================*/
    Tree* Creat(int siz = 0)
    {
        Tree* cur = new Tree;
        cur->siz = siz; cur->ls = cur->rs = null;
        return cur;
    }
    /*====================*/
    void Change(Tree*& cur, int valu, int delta, int treel, int treer)
    {
        if (cur == null)cur = Creat();
        cur->siz += delta;
        if (treel != treer)
        {
            int mid = (treel + treer) >> 1;
            if (valu <= mid)Change(cur->ls, valu, delta, treel, mid + 0);
            if (mid < valu) Change(cur->rs, valu, delta, mid + 1, treer);
        }
    }
    /*====================*/
    int GetValuByRank(Tree* cur, int rank, int treel, int treer)
    {
        if (treel == treer)
        {
            return treel;
        }
        else
        {
            int mid = (treel + treer) >> 1;
            if (cur->ls->siz >= rank)
            {
                return GetValuByRank(cur->ls, rank, treel, mid + 0);
            }
            else
            {
                return GetValuByRank(cur->rs, rank - cur->ls->siz, mid + 1,
treer);
            }
        }
    }
    int GetRankByValu(Tree* cur, int valu, int treel, int treer)
    {
        if (cur == null)
        {
            return 0;
        }
        else
        {
            if (treer < valu)
            {
```

```
                return cur->siz;
            }
            else
            {
                int res = 0;
                int mid = (treel + treer) >> 1;
                res += GetRankByValu(cur->ls, valu, treel, mid + 0);
                if (mid + 1 < valu) res += GetRankByValu(cur->rs, valu, mid
+ 1, treer);
                return res;
            }
        }
    }
public:
    C_SegmentTree(void)
    {
        root = null = new Tree; treel = treer = 0;
    }
    ~C_SegmentTree(void)
    {
        if (root != null)
        {
            queue<Tree*>q; q.push(root);
            while (!q.empty())
            {
                if (q.front()->ls != null)q.push(q.front()->ls);
                if (q.front()->rs != null)q.push(q.front()->rs);
                delete q.front(); q.pop();
            }
        }
        delete null;
    }
    /*====================*/
    int Size(void)
    {
        return root->siz;
    }
    bool Empty(void)
    {
        return root->siz == 0;
    }
    void Init(int treel, int treer)
    {
        this->treel = treel; this->treer = treer;
    }
    void Erase(int valu)
    {
        Change(root, valu, -1, treel, treer);
    }
    void Insert(int valu)
    {
        Change(root, valu, +1, treel, treer);
    }
    int operator[](int rank)
    {
```

```
        return GetValuByRank(root, rank, treel, treer);
    }
    int operator()(int valu)
    {
        return GetRankByValu(root, valu, treel, treer) + 1;
    }
};
```

数据类型

大数类

```
struct Bigint
{
    int sign; string digits;
    /*====================*/
    Bigint(void) {}
    Bigint(string b) { (*this) = b; }
    Bigint(int b) { (*this) = to_string(b); }
    /*====================*/
    int size(void)
    {
        return digits.size();
    }
    Bigint inverseSign(void)
    {
        sign *= -1; return (*this);
    }
    Bigint normalize(int newSign)
    {
        for (int i = digits.size() - 1; i > 0 && digits[i] == '0'; i--)
        {
            digits.erase(digits.begin() + i);
        }
        sign = (digits.size() == 1 && digits[0] == '0') ? 1 : newSign;
return (*this);
    }
    /*====================*/
    void operator = (string b)
    {
        digits = b[0] == '-' ? b.substr(1) : b;
        reverse(digits.begin(), digits.end());
        this->normalize(b[0] == '-' ? -1 : 1);
    }
    /*====================*/
    bool operator < (const Bigint& b) const
    {
        if (sign != b.sign) return sign < b.sign;
        if (digits.size() != b.digits.size())
            return sign == 1 ? digits.size() < b.digits.size() :
digits.size() > b.digits.size();
        for (int i = digits.size() - 1; i >= 0; i--) if (digits[i] !=
b.digits[i])
```

```
            return sign == 1 ? digits[i] < b.digits[i] : digits[i] > b.digits[i];
        return false;
    }
    bool operator == (const Bigint& b) const
    {
        return digits == b.digits && sign == b.sign;
    }
    /*====================*/
    Bigint operator + (Bigint b)
    {
        if (sign != b.sign) return (*this) - b.inverseSign();
        Bigint c;
        for (int i = 0, carry = 0; i < digits.size() || i < b.size() || carry; i++) {
            carry += (i < digits.size() ? digits[i] - 48 : 0) + (i < b.digits.size() ? b.digits[i] - 48 : 0);
            c.digits += (carry % 10 + 48);
            carry /= 10;
        }
        return c.normalize(sign);
    }
    Bigint operator - (Bigint b)
    {
        if (sign != b.sign) return (*this) + b.inverseSign();
        int s = sign; sign = b.sign = 1;
        if ((*this) < b) return ((b - (*this)).inverseSign()).normalize(-s);
        Bigint c;
        for (int i = 0, borrow = 0; i < digits.size(); i++) {
            borrow = digits[i] - borrow - (i < b.size() ? b.digits[i] : 48);
            c.digits += borrow >= 0 ? borrow + 48 : borrow + 58;
            borrow = borrow >= 0 ? 0 : 1;
        }
        return c.normalize(s);
    }
    Bigint operator * (Bigint b)
    {
        Bigint c("0");
        for (int i = 0, k = digits[i] - 48; i < digits.size(); i++, k = digits[i] - 48) {
            while (k--) c = c + b;
            b.digits.insert(b.digits.begin(), '0');
        }
        return c.normalize(sign * b.sign);
    }
    Bigint operator / (Bigint b)
    {
        if (b.size() == 1 && b.digits[0] == '0') b.digits[0] /= (b.digits[0] - 48);
        Bigint c("0"), d;
        for (int j = 0; j < digits.size(); j++) d.digits += "0";
        int dSign = sign * b.sign; b.sign = 1;
        for (int i = digits.size() - 1; i >= 0; i--) {
```

```
            c.digits.insert(c.digits.begin(), '0');
            c = c + digits.substr(i, 1);
            while (!(c < b)) c = c - b, d.digits[i]++;
        }
        return d.normalize(dSign);
    }
    Bigint operator % (Bigint b)
    {
        if (b.size() == 1 && b.digits[0] == '0') b.digits[0] /=
(b.digits[0] - 48);
        Bigint c("0");
        b.sign = 1;
        for (int i = digits.size() - 1; i >= 0; i--) {
            c.digits.insert(c.digits.begin(), '0');
            c = c + digits.substr(i, 1);
            while (!(c < b)) c = c - b;
        }
        return c.normalize(sign);
    }
    /*====================*/
    friend ostream& operator<<(ostream& output, Bigint integer)
    {
        if (integer.sign == -1) output << "-";
        for (int i = integer.digits.size() - 1; i >= 0; i--)
        {
            output << integer.digits[i];
        }
        return output;
    }
    friend istream& operator>>(istream& input, Bigint& integer)
    {
        string str; input >> str; integer = str; return input;
    }
};
```

分数类

```
class Fraction
{
public:
    int up, dw;
private:
    int GCD(int a, int b)
    {
        return b == 0 ? a : GCD(b, a % b);
    }
    void Simplest(void)
    {
        int divisor = GCD(up, dw);
        if (divisor != 0)
        {
            up /= divisor, dw /= divisor;
            if (dw < 0)dw *= -1, up *= -1;
        }
```

```
    }
public:
    Fraction(const Fraction& temp)
    {
        up = temp.up, dw = temp.dw;
    }
    Fraction(int _up = 0, int _dw = 1)
    {
        up = _up, dw = _dw; Simplest();
    }
    /*====================*/
    double Val(void)
    {
        return double(up) / double(dw);
    }
    /*====================*/
    void operator+=(const Fraction& x)
    {
        up = up * x.dw + x.up * dw; dw = dw * x.dw; Simplest();
    }
    void operator-=(const Fraction& x)
    {
        up = up * x.dw - x.up * dw; dw = dw * x.dw; Simplest();
    }
    void operator*=(const Fraction& x)
    {
        up = up * x.up; dw = dw * x.dw; Simplest();
    }
    void operator/=(const Fraction& x)
    {
        up = up * x.dw; dw = dw * x.up; Simplest();
    }
    /*====================*/
    friend Fraction operator+(const Fraction& a, const Fraction& b)
    {
        return Fraction(a.up * b.dw + b.up * a.dw, a.dw * b.dw);
    }
    friend Fraction operator-(const Fraction& a, const Fraction& b)
    {
        return Fraction(a.up * b.dw - b.up * a.dw, a.dw * b.dw);
    }
    friend Fraction operator*(const Fraction& a, const Fraction& b)
    {
        return Fraction(a.up * b.up, a.dw * b.dw);
    }
    friend Fraction operator/(const Fraction& a, const Fraction& b)
    {
        return Fraction(a.up * b.dw, a.dw * b.up);
    }
    /*====================*/
    friend bool operator<(const Fraction& a, const Fraction& b)
    {
        return (a.up * b.dw) < (b.up * a.dw);
    }
    friend bool operator>(const Fraction& a, const Fraction& b)
```

```
    {
        return (a.up * b.dw) > (b.up * a.dw);
    }
    friend bool operator==(const Fraction& a, const Fraction& b)
    {
        return (a.up == b.up) && (a.dw == b.dw);
    }
    friend bool operator!=(const Fraction& a, const Fraction& b)
    {
        return !(a == b);
    }
    friend bool operator<=(const Fraction& a, const Fraction& b)
    {
        return !(a > b);
    }
    friend bool operator>=(const Fraction& a, const Fraction& b)
    {
        return !(a < b);
    }
};
```

模数类

```
template<int MOD = 998244353>
class Modulo
{
private:
    static int Pow(int a, int b)
    {
        int res = 1;
        while (b)
        {
            if (b & 1)
            {
                res = (1ll * res * a) % MOD;
            }
            b >>= 1, a = (1ll * a * a) % MOD;
        }
        return res;
    }
    static int Inv(int x)
    {
        return Pow(x, MOD - 2);
    }
public:
    int num;
    /*====================*/
    Modulo(int temp = 0)
    {
        num = temp % MOD;
    }
    Modulo(const Modulo& temp)
    {
        num = temp.num;
```

```
    }
    /*====================*/
    friend Modulo operator+(const Modulo& a, const Modulo& b)
    {
        return Modulo((a.num + b.num) >= MOD ? a.num + b.num - MOD : a.num + b.num);
    }
    friend Modulo operator-(const Modulo& a, const Modulo& b)
    {
        return Modulo((a.num - b.num + MOD) >= MOD ? a.num - b.num : a.num - b.num + MOD);
    }
    friend Modulo operator*(const Modulo& a, const Modulo& b)
    {
        return Modulo(a.num * b.num % MOD);
    }
    friend Modulo operator/(const Modulo& a, const Modulo& b)
    {
        return Modulo(a.num * Inv(b.num) % MOD);
    }
    /*====================*/
    friend bool operator< (const Modulo& a, const Modulo& b)
    {
        return a.num < b.num;
    }
    friend bool operator==(const Modulo& a, const Modulo& b)
    {
        return a.num == b.num;
    }
    friend bool operator> (const Modulo& a, const Modulo& b)
    {
        return a.num > b.num;
    }
    friend bool operator<=(const Modulo& a, const Modulo& b)
    {
        return a.num <= b.num;
    }
    friend bool operator!=(const Modulo& a, const Modulo& b)
    {
        return a.num != b.num;
    }
    friend bool operator>=(const Modulo& a, const Modulo& b)
    {
        return a.num >= b.num;
    }
    /*====================*/
    void operator+=(const Modulo& x)
    {
        num = num + x.num; if (num >= MOD)num -= MOD;
    }
    void operator-=(const Modulo& x)
    {
        num = num - x.num + MOD; if (num >= MOD)num -= MOD;
    }
    void operator*=(const Modulo& x)
```

```
    {
        num = num * x.num % MOD;
    }
    void operator/=(const Modulo& x)
    {
        num = num * Inv(x.num) % MOD;
    }
    /*====================*/
    friend ostream& operator<<(ostream& output, Modulo integer)
    {
        output << integer.num; return output;
    }
    friend istream& operator>>(istream& input, Modulo& integer)
    {
        int temp; input >> temp; integer = (temp % MOD + MOD) % MOD; return input;
    }
};
```

数学

逆元

离线法

```
class C_INV
{
private:
    int Pow(int a, lnt b, int P)
    {
        int res = 1;
        while (b)
        {
            if (b & 1)
            {
                res = (1ll * res * a) % P;
            }
            b >>= 1, a = (1ll * a * a) % P;
        }
        return res;
    }
public:
    vector<int> operator()(vector<int>& arr, int P)
    {
        vector<int>sum = arr;
        for (int i = 1; i < sum.size(); ++i)
        {
            sum[i] = 1ll * sum[i - 1] * arr[i] % P;
        }
        vector<int>inv = sum;
        inv.back() = Pow(inv.back(), P - 2, P);
        for (int i = inv.size() - 2; i >= 0; --i)
        {
            inv[i] = 1ll * inv[i + 1] * arr[i + 1] % P;
```

```
        }
        for (int i = 1; i < inv.size(); ++i)
        {
            inv[i] = 1ll * sum[i - 1] * inv[i] % P;
        }
        return inv;
    }
};
```

快速幂法

```
class C_INV
{
private:
    int P;
    /*====================*/
    int Pow(int a, lnt b, int P)
    {
        int res = 1;
        while (b)
        {
            if (b & 1)
            {
                res = (1ll * res * a) % P;
            }
            b >>= 1, a = (1ll * a * a) % P;
        }
        return res;
    }
public:
    void Init(int P)
    {
        this->P = P;
    }
    int operator[](int x)
    {
        return Pow(x % P, P - 2, P);
    }
};
```

Farey序列

```
class C_INV
{
private:
    struct Frac
    {
        int up, dw;
        Frac(int _up = 0, int _dw = 0)
        {
            up = _up, dw = _dw;
        }
    };
    /*====================*/
```

```
    int P, B1, B2;
    vector<Frac>s;
    vector<int>inv;
    vector<int>pre, suf;
    /*====================*/
    int Calc(int x, Frac frac)
    {
        lnt pos = 1ll * x * frac.dw;
        if (abs(pos - 1ll * P * frac.up) > P / B1) return -1;
        if ((pos %= P) <= P / B1) return 1ll * frac.dw * inv[pos] % P;
        return P - 1ll * frac.dw * inv[P - pos] % P;
    }
public:
    void Init(int P)
    {
        this->P = P;

        B1 = pow(P, 1.0 / 3); B2 = B1 * B1;

        s.assign(B2 + 1, Frac());
        inv.assign(P / B1 + 1, 0);
        pre.assign(B2 + 1, 0); suf.assign(B2 + 1, 0);

        inv[1] = 1;
        for (int i = 2; i <= P / B1; ++i)
        {
            inv[i] = 1ll * (P - P / i) * inv[P % i] % P;
        }

        s[0] = Frac(0, 1); s[B2] = Frac(1, 1);
        pre[B2] = suf[B2] = B2;
        for (int i = 2; i <= B1; ++i)
        {
            for (int j = 1; j < i; ++j)
            {
                int pos = 1ll * j * B2 / i;
                if (pre[pos]) continue;
                pre[pos] = suf[pos] = pos;
                s[pos] = Frac(j, i);
            }
        }

        for (int i = 1; i <= B2; ++i)if (!pre[i])pre[i] = pre[i - 1];
        for (int i = B2; i >= 0; --i)if (!suf[i])suf[i] = suf[i + 1];
    }
    int operator[](int x)
    {
        if (x <= P / B1) return inv[x];
        int pos = 1ll * x * B2 / P, res = Calc(x, s[pre[pos]]);
        if (res == -1) res = Calc(x, s[suf[pos]]);
        return res;
    }
};
```

线性递推法

```cpp
class C_INV
{
private:
    vector<int>inv;
public:
    void Init(int n, int P)
    {
        inv.assign(n + 1, 0); inv[1] = 1;
        for (int i = 2; i <= n; ++i)
        {
            inv[i] = 1ll * (P - P / i) * inv[P % i] % P;
        }
    }
    int operator[](int x)
    {
        return inv[x];
    }
};
```

扩展欧几里得法

```cpp
class C_INV
{
private:
    int P;
    /*====================*/
    void exgcd(int a, int b, int& x, int& y)
    {
        if (b == 0)
        {
            x = 1, y = 0;
        }
        else
        {
            exgcd(b, a % b, y, x);
            y -= a / b * x;
        }
    }
public:
    void Init(int P)
    {
        this->P = P;
    }
    int operator[](int x)
    {
        int a, b;
        exgcd(x, P, a, b);
        return (a % P + P) % P;
    }
};
```

质数

判定

六倍试除法

```
class C_Prime
{
public:
    bool operator()(int x)
    {
        if (x <= 1)return false;
        if (x == 2 || x == 3 || x == 5)return true;
        if (x % 2 == 0 || x % 3 == 0)return false;
        for (int i = 5; i * i <= x; i += 6)
        {
            if (x % i == 0 || x % (i + 2) == 0)
            {
                return false;
            }
        }
        return true;
    }
};
```

Miller-Rabin

```
class C_Prime
{
private:
    lnt Mul(lnt a, lnt b, lnt p)
    {
        return (__int128)a * b % p;
    }
    lnt Pow(lnt a, lnt b, lnt p)
    {
        lnt res = 1;
        while (b)
        {
            if (b % 2 == 1)
            {
                res = Mul(res, a, p);
            }
            b /= 2, a = Mul(a, a, p);
        }
        return res;
    }
public:
    bool operator()(lnt x)
    {
        if (x < 3 || x % 2 == 0) return x == 2;
        lnt a = x - 1, b = 0;
        while (a % 2 == 0) a /= 2, ++b;
        //lnt lib[] = { 2,7,61 };
```

```
        lnt lib[] = { 2,325,9375,28178,450775,9780504,1795265022 };
        for (lnt r : lib)
        {
            lnt v = Pow(r, a, x);
            if (v == 1 || v == x - 1 || v == 0) continue;
            for (lnt j = 1; j <= b; j++)
            {
                v = Mul(v, v, x);
                if (v == x - 1 && j != b) { v = 1; break; }
                if (v == 1) return false;
            }
            if (v != 1) return false;
        }
        return true;
    }
};
```

筛法

欧拉筛

```
class C_Prime
{
private:
    vector<int>prime;
    vector<bool>isprime;
public:
    void Init(int n)
    {
        isprime.assign(n + 1, true);
        isprime[0] = isprime[1] = false;
        for (int i = 2; i <= n; ++i)
        {
            if (isprime[i])prime.push_back(i);
            for (int j = 0; j < prime.size(); ++j)
            {
                if (i * prime[j] > n)break;
                isprime[i * prime[j]] = false;
                if (i % prime[j] == 0)break;
            }
        }
    }
    int Size(void)
    {
        return prime.size();
    }
    bool operator()(int x)
    {
        return isprime[x];
    }
    int operator[](int x)
    {
        return prime[x - 1];
    }
};
```

快速幂

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

龟速乘

```
int Mul(int a, int b, int p)
{
    int res = 0;
    while (b)
    {
        if (b & 1)
        {
            res = (res + a) % p;
        }
        b >>= 1, a = (a * 2) % p;
    }
    return res;
}
lnt Mul(lnt a, lnt b, lnt p)
{
    return (a * b - (lnt)(a / (long double)p * b + 1e-3) * p + p) % p;
}
lnt Mul(lnt a, lnt b, lnt p)
{
    return (__int128)a * b % p;
}
```

逆序对

```
template<class Type>
class _CountInversions
{
public:
    lnt operator()(int n, Type arr[])
    {
        res = 0;
        a = new Type[n + 1];
        temp = new Type[n + 1];
        for (int i = 1; i <= n; ++i)
```

```
        {
            a[i] = arr[i];
        }
        MergeSort(1, n);
        delete[] a; delete[] temp;
        return res;
    }
private:
    Type* a = NULL;
    lnt res = 0;
    Type* temp = NULL;
    /*====================*/
    void MergeSort(int l, int r)
    {
        if (l < r)
        {
            int i = l;
            int mid = (l + r) >> 1;
            int p = l, q = mid + 1;
            MergeSort(l, mid + 0);
            MergeSort(mid + 1, r);
            while (p <= mid && q <= r)
            {
                if (a[p] <= a[q])
                {
                    temp[i++] = a[p++];
                }
                else
                {
                    temp[i++] = a[q++];
                    res += (lnt)(mid - p + 1);
                }
            }
            while (p <= mid)temp[i++] = a[p++];
            while (q <= r)  temp[i++] = a[q++];
            for (i = l; i <= r; i++)a[i] = temp[i];
        }
    }
};
```

多项式

快速傅里叶变换

```
typedef complex<double> Comp;

const ll SIZE = 4000000 + 10;
const double PI = acos(-1.0);

Comp temp[SIZE];
void FFT(Comp* p, ll len, ll inv)//inv=+1 FFT;inv=-1 IFFT
{
    if (len == 1) return;
    const int E = 0, O = len / 2;
```

```
    for (ll i = 0; i < len; ++i) temp[i] = p[i];
    for (ll i = 0; i < len; ++i)
    {
        if ((i & 1) == 1)p[i / 2 + O] = temp[i];
        if ((i & 1) == 0)p[i / 2 + E] = temp[i];
    }
    Comp* pe = p + E; FFT(pe, len / 2, inv);
    Comp* po = p + O; FFT(po, len / 2, inv);
    Comp omega(1, 0);
    const double Angle = 2 * PI / len;
    const Comp step(cos(Angle), sin(inv * Angle));
    for (ll k = 0; k < len / 2; ++k, omega *= step)
    {
        temp[k + E] = pe[k] + omega * po[k];
        temp[k + O] = pe[k] - omega * po[k];
    }
    for (ll i = 0; i < len; ++i) p[i] = temp[i];
}
```

离散对数

```
ll BSGS(ll a, ll b, ll m)
{
    static unordered_map<ll, ll> hs;
    hs.clear();
    ll cur = 1, t = sqrt(m) + 1;
    for (int B = 1; B <= t; ++B)
    {
        (cur *= a) %= m;
        hs[b * cur % m] = B; // 哈希表中存B的值
    }
    ll now = cur; // 此时cur = a^t
    for (int A = 1; A <= t; ++A)
    {
        auto it = hs.find(now);
        if (it != hs.end())
            return A * t - it->second;
        (now *= cur) %= m;
    }
    return -1; // 没有找到，无解
}
```

基数排序

```
template <int B = 8, class Type>
void RadixSort(vector<Type>& a)
{
    const int mask = (1 << B) - 1, n = a.size();
    vector<Type> b(n); vector<int> cnt(1 << B);
    Type maxV = *max_element(a.begin(), a.end());
    for (int i = 0; maxV; i += B, maxV >>= B)
    {
        fill(cnt.begin(), cnt.end(), 0);
```

```
        for (int j = 0; j < n; j++)cnt[a[j] >> i & mask] += 1;
        for (int j = 1; j < (1 << B); j++)cnt[j] += cnt[j - 1];
        for (int j = n - 1; j >= 0; j--)b[--cnt[a[j] >> i & mask]] = a[j];
        swap(a, b);
    }
}
```

欧拉函数

试除法

```
class PHI
{
public:
    int operator[](int x)
    {
        return GetPhi(x);
    }
private:
    int GetPhi(int x)
    {
        int res = x;
        for (int i = 2; i * i <= x; ++i)
        {
            if (x % i == 0)
            {
                res = res / i * (i - 1);
                while (x % i == 0) x /= i;
            }
        }
        if (x > 1) res = res / x * (x - 1);
        return res;
    }
};
```

欧拉筛法

```
class PHI
{
public:
    ~PHI(void)
    {
        delete[] phi;
    }
    void init(int n)
    {
        GetPhi(n);
    }
    int operator[](int x)
    {
        return phi[x];
    }
private:
```

```
    int* phi = NULL;
    void GetPhi(int n)
    {
        phi = new int[n + 1];
        bool* vis = new bool[n + 1];
        int* table = new int[n + 1];
        for (int i = 0; i <= n; ++i)
        {
            vis[i] = true;
        }
        int cnt = 0; phi[1] = 1;
        for (int i = 2; i <= n; ++i)
        {
            if (vis[i])
            {
                phi[i] = i - 1;
                table[++cnt] = i;
            }
            for (int j = 1; j <= cnt; ++j)
            {
                if (i * table[j] > n)break;
                vis[i * table[j]] = false;
                if (i % table[j] == 0)
                {
                    phi[i * table[j]] = phi[i] * table[j]; break;
                }
                else
                {
                    phi[i * table[j]] = phi[i] * (table[j] - 1);
                }
            }
        }
        delete[] vis; delete[] table;
    }
};
```

欧拉降幂

```
class EX_Euler
{
public:
    int operator()(int a, string s, int p)
    {
        int b = 0;
        bool flag = false;
        int phi = GetPhi(p);
        for (auto c : s)
        {
            b = (b * 10 + c - '0');
            if (b >= phi)flag = true, b %= phi;
        }
        if (flag)b += phi; return Pow(a % p, b, p);
    }
private:
```

```
    int GetPhi(int x)
    {
        int res = x;
        for (int i = 2; i * i <= x; ++i)
        {
            if (x % i == 0)
            {
                res = res / i * (i - 1);
                while (x % i == 0) x /= i;
            }
        }
        if (x > 1) res = res / x * (x - 1);
        return res;
    }
    int Pow(int a, int b, int p)
    {
        int res = 1;
        while (b != 0)
        {
            if (b % 2 == 1)
            {
                res = (res * a) % p;
            }
            b /= 2, a = (a * a) % p;
        }
        return res;
    }
};
```

欧几里得

最大公因数

```
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}
```

最小公倍数

```
int lcm(int a, int b)
{
    return a / gcd(a, b) * b;
}
```

扩展欧几里得

```cpp
void exgcd(int a, int b, int& x, int& y)
{
    if (b == 0)
    {
        x = 1, y = 0;
    }
    else
    {
        exgcd(b, a % b, y, x);
        y -= a / b * x;
    }
}
```

分解质因数

Pollard-Rho

```cpp
class C_PrimeFactorization
{
private:
    lnt Mul(lnt a, lnt b, lnt p)
    {
        return (__int128)a * b % p;
    }
    lnt Pow(lnt a, lnt b, lnt p)
    {
        lnt res = 1;
        while (b)
        {
            if (b % 2 == 1)
            {
                res = Mul(res, a, p);
            }
            b /= 2, a = Mul(a, a, p);
        }
        return res;
    }
    bool MillerRabin(lnt x)
    {
        if (x < 3 || x % 2 == 0) return x == 2;
        lnt a = x - 1, b = 0;
        while (a % 2 == 0) a /= 2, ++b;
        //lnt lib[] = { 2,7,61 };
        lnt lib[] = { 2,325,9375,28178,450775,9780504,1795265022 };
        for (lnt r : lib)
        {
            lnt v = Pow(r, a, x);
            if (v == 1 || v == x - 1 || v == 0) continue;
            for (lnt j = 1; j <= b; j++)
            {
                v = Mul(v, v, x);
                if (v == x - 1 && j != b) { v = 1; break; }
                if (v == 1) return false;
            }
```

```
            if (v != 1) return false;
        }
        return true;
    }
    /*====================*/
    lnt GCD(lnt a, lnt b)
    {
        return b == 0 ? a : GCD(b, a % b);
    }
    lnt GetFactor(lnt X)
    {
        if (X == 4)return 2;
        if (MillerRabin(X))return X;
        mt19937 rng(time(NULL));
        while (1)
        {
            lnt c = rng() % (X - 1) + 1;
            auto f = [=](lnt x) { return ((__int128)x * x + c) % X; };
            lnt t = 0, r = 0, p = 1, q;
            do
            {
                for (int i = 0; i < 128; ++i)
                {
                    t = f(t), r = f(f(r));
                    if (t == r || (q = (__int128)p * abs(t - r) % X) ==
0)break;
                    p = q;
                }
                lnt d = GCD(p, X); if (d > 1)return d;
            } while (t != r);
        }
    }
    void GetAllFactor(lnt X, vector<lnt>& lib)
    {
        lnt fac = GetFactor(X);
        if (fac == X)lib.push_back(fac);
        else GetAllFactor(fac, lib), GetAllFactor(X / fac, lib);
    }
public:
    void operator()(lnt x, vector<lnt>& num, vector<lnt>& cnt)
    {
        num.clear(), cnt.clear();
        GetAllFactor(x, num);
        sort(num.begin(), num.end());
        for (int i = 0; i < num.size(); ++i)
        {
            if (i == 0 || num[i] != num[i - 1])
            {
                cnt.push_back(0);
            }
            cnt.back()++;
        }
        num.erase(unique(num.begin(), num.end()), num.end());
    }
};
```

预处理质数表法

```cpp
class C_PrimeFactorization
{
private:
    vector<int>prime;
public:
    void Init(int n)
    {
        vector<bool>isprime;
        isprime.assign(n + 1, true);
        isprime[0] = isprime[1] = false;
        for (int i = 2; i <= n; ++i)
        {
            if (isprime[i])prime.push_back(i);
            for (int j = 0; j < prime.size(); ++j)
            {
                if (i * prime[j] > n)break;
                isprime[i * prime[j]] = false;
                if (i % prime[j] == 0)break;
            }
        }
    }
    void operator()(int x, vector<int>& num, vector<int>& cnt)
    {
        num.clear(), cnt.clear();
        for (auto p : prime)
        {
            if (p * p > x)break;
            if (x % p == 0)
            {
                num.push_back(p), cnt.push_back(0);
                while (x % p == 0)cnt.back()++, x /= p;
            }
        }
        if (x != 1)
        {
            num.push_back(x), cnt.push_back(1);
        }
    }
};
```

预处理最小质因子法

```cpp
class C_PrimeFactorization
{
private:
    vector<int>mpf;
public:
    void Init(int n)
    {
        vector<int>prime;
        vector<bool>isprime;
        mpf.assign(n + 1, 0);
```

```
        isprime.assign(n + 1, true);
        isprime[0] = isprime[1] = false;
        for (int i = 2; i <= n; ++i)
        {
            if (isprime[i])prime.push_back(i), mpf[i] = i;
            for (int j = 0; j < prime.size(); ++j)
            {
                if (i * prime[j] > n)break;
                isprime[i * prime[j]] = false;
                mpf[i * prime[j]] = prime[j];
                if (i % prime[j] == 0)break;
            }
        }
    }
    void operator()(int x, vector<int>& num, vector<int>& cnt)
    {
        num.clear(), cnt.clear();
        while (x > 1)
        {
            if (num.empty() || num.back() != mpf[x])
            {
                num.push_back(mpf[x]); cnt.push_back(1);
            }
            else
            {
                cnt.back()++;
            }
            x /= mpf[x];
        }
    }
};
```

获得全部因数

```
void GetDivisor(int x, vector<int>& divisor)
{
    vector<int>num, cnt;
    PFF(x, num, cnt);
    divisor.push_back(1);
    for (int i = 0; i < num.size(); ++i)
    {
        int val = 1;
        int lim = divisor.size();
        for (int j = 1; j <= cnt[i]; ++j)
        {
            val *= num[i];
            for (int k = 0; k < lim; ++k)
            {
                divisor.push_back(divisor[k] * val);
            }
        }
    }
}
```

随机化

随机数生成器

```
mt19937 Rand(random_device{}());
```

模拟退火

```
#include<bits/stdc++.h>
using namespace std;
/*====================*/
#define endl "\n"
/*====================*/
typedef long long lnt;
/*====================*/
const int N = 1e3 + 10;
/*====================*/
double begintime = 0;
bool TLE(void)
{
    if ((clock() - begintime) / CLOCKS_PER_SEC > 0.9)
    {
        return true;
    }
    return false;
}
/*====================*/
int n;
/*====================*/
struct Node
{
    double w, x, y;
    Node(double _w = 0, double _x = 0, double _y = 0)
    {
        w = _w, x = _x, y = _y;
    }
};
Node node[N];
/*====================*/
double ansx, ansy, ansSigma = 1e18;
/*====================*/
double GetSigma(double x, double y)
{
    double res = 0;
    for (int i = 1; i <= n; ++i)
    {
        double dx = node[i].x - x;
        double dy = node[i].y - y;
        res += sqrt(dx * dx + dy * dy) * node[i].w * 100;
    }
    if (res < ansSigma)
    {
        ansx = x, ansy = y, ansSigma = res;
```

```
    }
    return res;
}
/*====================*/
double Rand() { return (double)rand() / RAND_MAX; }
/*====================*/
void SA(void)
{
    double curx = 0, cury = 0, curSigma = 0;
    for (int i = 1; i <= n; ++i)
    {
        curx += node[i].x, cury += node[i].y;
    }
    curx /= n, cury /= n; curSigma = GetSigma(curx, cury);

    double t = 1e4;
    while (t > 5e-4)
    {
        double nxtx = curx + t * (Rand() * 2.0 - 1.0);
        double nxty = cury + t * (Rand() * 2.0 - 1.0);
        double nxtSigma = GetSigma(nxtx, nxty);
        double delta = nxtSigma - curSigma;
        if (exp(-delta / t) > Rand())
        {
            curx = nxtx, curx = nxty, curSigma = nxtSigma;
        }
        t *= 0.9996;
    }
    for (int i = 1; i <= 5000; ++i)
    {
        double nxtx = ansx + t * (Rand() * 2 - 1);
        double nxty = ansy + t * (Rand() * 2 - 1);
        double nxtSigma = GetSigma(nxtx, nxty);
    }
}
/*====================*/
void Solve(void)
{
    cin >> n;
    for (int i = 1; i <= n; ++i)
    {
        double w, x, y;
        cin >> x >> y >> w;
        node[i] = Node(w, x, y);
    }
    while (!TLE())SA();
    printf("%.3f %.3f\n", ansx, ansy);
}
/*====================*/
int main()
{
#ifndef ONLINE_JUDGE
    freopen("IN.txt", "r+", stdin);
#endif
    srand(time(0)); begintime = clock();
```

```
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);
    int T = 1; //cin >> T;
    while (T--)Solve();
    return 0;
}
```

图论

存储

```
class C_Graph
{
public:
    struct Edge
    {
        int u, v, w;
        Edge(int _u = 0, int _v = 0, int _w = 0)
        {
            u = _u, v = _v, w = _w;
        }
        friend bool operator<(const Edge& a, const Edge& b)
        {
            return a.w < b.w;
        }
        int node(int x)const
        {
            return x == u ? v : u;
        }
    };
    /*====================*/
    int n;
    vector<Edge>edge;
    vector<vector<int>>G;
    /*====================*/
    void Init(int n)
    {
        this->n = n; G.assign(n + 1, vector<int>());
    }
    void AddEdge(int u, int v, int w)
    {
        edge.push_back(Edge(u, v, w));
        G[u].push_back(edge.size() - 1);
    }
};
```

2-SAT

```
namespace _TwoSAT
{
    using namespace _SCC;
    void Init(void)
    {
```

```
        for (int i = 1; i <= n; ++i)
        {
            int u = idx[i][1], v = idx[i][0];
            if (belong[u] == belong[v])
            {
                cout << "IMPOSSIBLE" << endl; return;
            }
        }
        cout << "POSSIBLE" << endl;
        for (int i = 1; i <= n; ++i)
        {
            int u = idx[i][1], v = idx[i][0];
            cout << ((belong[u] < belong[v]) ? 1 : 0) << " ";
        }
        cout << endl;
    }
}
```

欧拉图

无向图

```
namespace _Euler
{
    /*
    默认连通图，-1不存在，0存在欧拉路径，1存在欧拉回路
    */
    const int N = 1e5 + 10;
    /*====================*/
    int degree[N];
    /*====================*/
    int Init(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            degree[i] = G[i].size();
        }
        int cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            if (degree[i] % 2 == 1)cnt++;
        }
        return cnt == 0 ? 1 : (cnt == 2 ? 0 : -1);
    }
}
```

有向图

```
namespace _Euler
{
    /*
    默认连通图，-1不存在，0存在欧拉路径，1存在欧拉回路
    */
```

```
    const int N = 1e5 + 10;
    /*====================*/
    int degree[N];
    /*====================*/
    int Init(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            degree[i] = 0;
        }
        for (int i = 1; i <= m; ++i)
        {
            int u = edge[i].u;
            int v = edge[i].v;
            degree[u]++, degree[v]--;
        }
        int cnt1 = 0, cnt2 = 0, cnt3 = 0;
        for (int i = 1; i <= n; ++i)
        {
            if (degree[i] == -1)cnt1++;
            if (degree[i] == +0)cnt2++;
            if (degree[i] == +1)cnt3++;
        }
        if (cnt1 == 1 && cnt3 == 1 && cnt2 + 2 == n)
        {
            return +0;
        }
        else if (cnt2 == n)
        {
            return +1;
        }
        else
        {
            return -1;
        }
    }
}
```

最大团

```
namespace _MaxClique
{
    const int N = 5e1 + 10;
    /*====================*/
    int n; int G[N][N];
    int dp[N], stk[N][N], res;
    /*====================*/
    bool DFS(int ns, int dep)
    {
        if (ns == 0)
        {
            if (dep > res)
            {
                res = dep; return true;
```

```
            }
            return false;
        }
        for (int i = 0; i < ns; ++i)
        {
            int u = stk[dep][i], cnt = 0;
            if (dep + dp[u] <= res)return false;
            if (dep + ns - i <= res)return false;
            for (int j = i + 1; j < ns; ++j)
            {
                int v = stk[dep][j];
                if (G[u][v])stk[dep + 1][cnt++] = v;
            }
            if (DFS(cnt, dep + 1))return true;
        }
        return false;
    }
    /*====================*/
    int Init(void)
    {
        cin >> n; res = 0;
        memset(dp, 0, sizeof(dp));
        for (int i = 1; i <= n; ++i)
        {
            for (int j = 1; j <= n; ++j)
            {
                cin >> G[i][j];
            }
        }
        for (int i = n; i >= 1; --i)
        {
            int ns = 0;
            for (int j = i + 1; j <= n; ++j)
            {
                if (G[i][j])stk[1][ns++] = j;
            }
            DFS(ns, 1); dp[i] = res;
        }
        return res;
    }
}
```

最短路

SPFA

```
namespace _SPFA
{
    const int N = 1e5 + 10;
    /*====================*/
    int dis[N]; bool vis[N];
    /*====================*/
    void Init(int s)
    {
```

```
        memset(dis, 0X3F, sizeof(dis));
        memset(vis, false, sizeof(vis));
        queue<int>q; dis[s] = 0; q.push(s);
        while (!q.empty())
        {
            int cur = q.front(); q.pop(); vis[cur] = false;
            for (int i = 0; i < G[cur].size(); ++i)
            {
                int val = edge[G[cur][i]].w;
                int nxt = edge[G[cur][i]].node(cur);
                if (dis[nxt] > dis[cur] + val)
                {
                    dis[nxt] = dis[cur] + val;
                    if (!vis[nxt])
                    {
                        q.push(nxt); vis[nxt] = true;
                    }
                }
            }
        }
    }
}
```

Floyd

```
namespace _Floyd
{
    const int N = 2e2 + 10;
    /*====================*/
    int dp[N][N];
    /*====================*/
    void Init(void)
    {
        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]);
                }
            }
        }
    }
}
```

Dijkstra

```
namespace _Dijkstra
{
    const int N = 1e5 + 10;
    /*====================*/
    struct Unit
    {
```

```
        int v, w;
        Unit(int _v = 0, int _w = 0)
        {
            v = _v, w = _w;
        }
        friend bool operator<(const Unit& a, const Unit& b)
        {
            return a.w > b.w;
        }
    };
    /*====================*/
    int dis[N]; bool vis[N];
    /*====================*/
    void Init(int s)
    {
        memset(dis, 0X3F, sizeof(dis));
        memset(vis, false, sizeof(vis));
        priority_queue<Unit>q;
        q.push(Unit(s, dis[s] = 0));
        while (!q.empty())
        {
            int cur = q.top().v; q.pop();
            if (vis[cur])continue; vis[cur] = true;
            for (int i = 0; i < G[cur].size(); ++i)
            {
                int val = edge[G[cur][i]].w;
                int nxt = edge[G[cur][i]].node(cur);
                if (dis[nxt] > dis[cur] + val)
                {
                    dis[nxt] = dis[cur] + val;
                    q.push(Unit(nxt, dis[nxt]));
                }
            }
        }
    }
}
```

SPFA-SLF

```
namespace _SPFA
{
    const int N = 1e5 + 10;
    /*====================*/
    int dis[N]; bool vis[N];
    /*====================*/
    void Init(int s)
    {
        memset(dis, 0X3F, sizeof(dis));
        memset(vis, false, sizeof(vis));
        deque<int>q; dis[s] = 0; q.push_back(s);
        while (!q.empty())
        {
            int cur = q.front();
            q.pop_front(); vis[cur] = false;
            if (!q.empty() && dis[q.front()] > dis[q.back()])
```

```
            {
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[cur].size(); ++i)
            {
                int val = edge[G[cur][i]].w;
                int nxt = edge[G[cur][i]].node(cur);
                if (dis[nxt] > dis[cur] + val)
                {
                    dis[nxt] = dis[cur] + val;
                    if (!vis[nxt])
                    {
                        vis[nxt] = true;
                        if (!q.empty() && dis[nxt] < dis[q.front()])
                        {
                            q.push_front(nxt);
                        }
                        else
                        {
                            q.push_back(nxt);
                        }
                    }
                }
            }
        }
    }
}
```

环相关

判环

```
namespace _Loop
{
    const int N = 1e5 + 10;
    /*====================*/
    int indegree[N];
    /*====================*/
    bool Init(void)
    {
        queue<int>q; int cnt = 0;
        memset(indegree, 0, sizeof(indegree));
        for (int i = 1; i <= m; ++i)
        {
            indegree[edge[i].v]++;
        }
        for (int i = 1; i <= n; ++i)
        {
            if (indegree[i] == 0)q.push(i);
        }
        while (!q.empty())
        {
            int cur = q.front(); q.pop(); cnt++;
            for (int i = 0; i < G[cur].size(); ++i)
```

```
            {
                int nxt = edge[G[cur][i]].node(cur);
                if (--indegree[nxt] == 0)q.push(nxt);
            }
        }
        return cnt == n;
    }
}
```

判负环

```
namespace _Loop
{
    const int N = 1e5 + 10;
    /*====================*/
    int dis[N], cnt[N]; bool vis[N];
    /*====================*/
    bool Init(void)
    {
        queue<int>q;
        memset(cnt, 0, sizeof(cnt));
        memset(dis, 0, sizeof(dis));
        for (int i = 1; i <= n; ++i)
        {
            q.push(i); vis[i] = true;
        }
        while (!q.empty())
        {
            int cur = q.front();
            q.pop(); vis[cur] = false;
            for (int i = 0; i < G[cur].size(); ++i)
            {
                int val = edge[G[cur][i]].w;
                int nxt = edge[G[cur][i]].node(cur);
                if (dis[nxt] > dis[cur] + val)
                {
                    cnt[nxt] = cnt[cur] + 1;
                    dis[nxt] = dis[cur] + val;
                    if (cnt[nxt] > n)
                    {
                        return true;
                    }
                    if (!vis[nxt])
                    {
                        q.push(nxt); vis[nxt] = true;
                    }
                }
            }
        }
        return false;
    }
}
```

求最小环

```
namespace _Loop
{
    int Init(void)
    {
        int res = 0X3F3F3F3F;
        for (int i = 1; i <= m; ++i)
        {
            _Dijkstra::Init(edge[i].v, i);
            res = min(res, dis[edge[i].u] + edge[i].w);
        }
        return res;
    }
}
```

网络流

最大流

ISAP

```
namespace _ISAP
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int pre[N];//路径前驱
    int cur[N];//当前弧优化
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    int d[N], vis[N], num[N];//图分层
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
        G[v].push_back(cnt - 1);
    }
    /*====================*/
    void BFS(void)
```

```
    {
        for (int i = 0; i <= n; ++i)
        {
            d[i] = vis[i] = num[i] = 0;
        }
        queue<int>q; q.push(t);
        d[t] = 0; vis[t] = 1;
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (!vis[e.u] && e.c > e.f)
                {
                    vis[e.u] = 1; d[e.u] = d[x] + 1; q.push(e.u);
                }
            }
        }
        for (int i = 0; i < n; ++i)num[d[i]]++;
    }
    int Augumemt(void)
    {
        int x, k = INF;
        x = t; while (x != s)
        {
            Edge& e = edge[pre[x]];
            k = min(k, e.c - e.f);
            x = edge[pre[x]].u;
        }
        x = t; while (x != s)
        {
            edge[pre[x]].f += k;
            edge[pre[x] ^ 1].f -= k;
            x = edge[pre[x]].u;
        }
        return k;
    }
    int MaxFlow(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            pre[i] = cur[i] = 0;
        }

        BFS(); int x = s, flow = 0;

        while (d[s] < n)
        {
            if (x == t)
            {
                flow += Augumemt(); x = s;
            }
            int flag = 0;
            for (int& i = cur[x]; i < G[x].size(); ++i)
```

```
            {
                Edge& e = edge[G[x][i]];
                if (e.c > e.f && d[x] == d[e.v] + 1)
                {
                    flag = 1; pre[e.v] = G[x][i]; x = e.v; break;
                }
            }
            if (!flag)
            {
                int l = n - 1;
                for (int i = 0; i < G[x].size(); ++i)
                {
                    Edge& e = edge[G[x][i]];
                    if (e.c > e.f)l = min(l, d[e.v]);
                }
                if (--num[d[x]] == 0)break;
                num[d[x] = l + 1]++; cur[x] = 0;
                if (x != s)x = edge[pre[x]].u;
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        cin >> n >> m >> s >> t;
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c;
            cin >> u >> v >> c;
            AddEdge(u, v, c);
        }
        return MaxFlow();
    }
};
```

HLPP

```
namespace _HLPP
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int next, v, c;
        Edge(int _next = 0, int _v = 0, int _c = 0)
        {
```

```
            next = _next, v = _v, c = _c;
        }
    };
    /*====================*/
    int n, m, s, t;
    int d[N], num[N];
    stack<int> lib[N];
    int ex[N], level = 0;
    Edge edge[2 * M]; int head[N], cnt;
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt] = Edge(head[u], v, c), head[u] = cnt++;
        edge[cnt] = Edge(head[v], u, 0), head[v] = cnt++;
    }
    /*====================*/
    int Push(int u)
    {
        bool init = u == s;
        for (int i = head[u]; i != -1; i = edge[i].next)
        {
            const int& v = edge[i].v, & c = edge[i].c;
            if (!c || init == false && d[u] != d[v] + 1)continue;
            int k = init ? c : min(c, ex[u]);
            if (v != s && v != t && !ex[v]) lib[d[v]].push(v), level =
max(level, d[v]);
            ex[u] -= k, ex[v] += k, edge[i].c -= k, edge[i ^ 1].c += k;
            if (!ex[u]) return 0;
        }
        return 1;
    }
    void Relabel(int x)
    {
        d[x] = INF;
        for (int i = head[x]; i != -1; i = edge[i].next)
        {
            if (edge[i].c) d[x] = min(d[x], d[edge[i].v]);
        }
        if (++d[x] < n)
        {
            lib[d[x]].push(x); level = max(level, d[x]); ++num[d[x]];
        }
    }
    bool BFS(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            d[i] = INF; num[i] = 0;
        }
        queue<int>q; q.push(t), d[t] = 0;
        while (!q.empty())
        {
            int u = q.front(); q.pop(); num[d[u]]++;
            for (int i = head[u]; i!=-1; i = edge[i].next)
            {
```

```
                const int& v = edge[i].v;
                if (edge[i ^ 1].c && d[v] > d[u] + 1) d[v] = d[u] + 1,
q.push(v);
            }
        }
        return d[s] != INF;
    }
    int Select(void)
    {
        while (lib[level].size() == 0 && level > -1) level--;
        return level == -1 ? 0 : lib[level].top();
    }
    int MaxFlow(void)
    {
        if (!BFS()) return 0;
        d[s] = n; Push(s); int x;
        while (x = Select())
        {
            lib[level].pop();
            if (Push(x))
            {
                if (!--num[d[x]])
                {
                    for (int i = 1; i <= n; ++i)
                    {
                        if (i != s && i != t && d[i] > d[x] && d[i] < n +
1)
                        {
                            d[i] = n + 1;
                        }
                    }
                }
                Relabel(x);
            }
        }
        return ex[t];
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        cin >> n >> m >> s >> t;
        memset(head, -1, sizeof(head));
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c;
            cin >> u >> v >> c;
            AddEdge(u, v, c);
        }
        return MaxFlow();
    }
}
```

Dinic

```
/*
* 使用方法
* 1.创建对象
* 2.调用AddVertex()创建点
* 3.调用Init()初始化大小
* 4.调用AddEdge()创建边
* 5.调用MaxFlow()获取最大流
*/
class C_Dinic
{
public:
    static const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int n, s, t;
    /*====================*/
    vector<Edge>edge;
    vector<vector<int>>G;
    /*====================*/
    C_Dinic(void)
    {
        n = 2, s = 1, t = 2;
    }
private:
    vector<int>cur;//当前弧优化
    vector<int>d, vis;//图分层
    /*====================*/
    bool BFS(void)
    {
        fill(d.begin(), d.end(), 0);
        fill(vis.begin(), vis.end(), 0);
        d[s] = 0; vis[s] = 1;
        queue<int>q; q.push(s);
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (!vis[e.v] && e.c > e.f)
                {
                    vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
                }
            }
        }
```

```
        return vis[t];
    }
    int DFS(int x, int k)
    {
        int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int& i = cur[x]; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) >
0)
            {
                e.f += f; edge[G[x][i] ^ 1].f -= f;
                flow += f; k -= f; if (k == 0) break;
            }
        }
        return flow;
    }
public:
    int S(void) { return s; }
    int T(void) { return t; }
    /*====================*/
    int AddVertex(void)
    {
        return ++n;
    }
    int AddEdge(int u, int v, int c)
    {
        edge.push_back(Edge(u, v, c, 0));
        edge.push_back(Edge(v, u, 0, 0));
        G[u].push_back(edge.size() - 2);
        G[v].push_back(edge.size() - 1);
        return edge.size() - 2;
    }
    /*====================*/
    void Init(void)
    {
        d.assign(n + 1, int());
        vis.assign(n + 1, int());
        cur.assign(n + 1, int());
        G.assign(n + 1, vector<int>());
    }
    /*====================*/
    int MaxFlow(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            fill(cur.begin(), cur.end(), 0);
        }
        return flow;
    }
};
```

Dinic最后反悔

```
class C_MaxFlow
{
public:
    static const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
    };
    /*====================*/
    int n = 2, s = 1, t = 2;
    /*====================*/
    vector<Edge>edge;
    vector<vector<int>>G1;
    vector<vector<int>>G2;
private:
    vector<int>cur;//当前弧优化
    vector<int>d, vis;//图分层
    /*====================*/
    bool BFS(void)
    {
        fill(d.begin(), d.end(), 0);
        fill(vis.begin(), vis.end(), 0);
        d[s] = 0; vis[s] = 1;
        queue<int>q; q.push(s);
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G1[x].size(); ++i)
            {
                Edge& e = edge[G1[x][i]];
                if (!vis[e.v] && e.c > e.f)
                {
                    vis[e.v] = 1; d[e.v] = d[x] + 1; q.push(e.v);
                }
            }
        }
        return vis[t];
    }
    int DFS(int x, int k)
    {
        int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int& i = cur[x]; i < G1[x].size(); ++i)
        {
            Edge& e = edge[G1[x][i]];
            if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) >
0)
            {
```

```
                e.f += f; edge[G1[x][i] ^ 1].f -= f;
                flow += f; k -= f; if (k == 0) break;
            }
        }
        return flow;
    }
public:
    int S(void) { return s; }
    int T(void) { return t; }
    /*====================*/
    int AddVertex(void)
    {
        return ++n;
    }
    int AddEdge(int u, int v, int c)
    {
        edge.push_back(Edge(u, v, c, 0));
        edge.push_back(Edge(v, u, 0, 0));
        G1[u].push_back(edge.size() - 2);
        G2[v].push_back(edge.size() - 1);
        return edge.size() - 2;
    }
    /*====================*/
    void Init(void)
    {
        d.assign(n + 1, int());
        vis.assign(n + 1, int());
        cur.assign(n + 1, int());
        G1.assign(n + 1, vector<int>());
        G2.assign(n + 1, vector<int>());
    }
    /*====================*/
    int MaxFlow(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            fill(cur.begin(), cur.end(), 0);
        }
        for (int i = 1; i <= n; ++i)
        {
            for (auto idx : G2[i])
            {
                G1[i].push_back(idx);
            }
        }
        while (BFS())
        {
            flow += DFS(s, INF);
            fill(cur.begin(), cur.end(), 0);
        }
        return flow;
    }
};
```

Dinic-Scaling

```cpp
namespace _Dinic
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
        int u, v, c, f;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _f = 0)
        {
            u = _u, v = _v, c = _c, f = _f;
        }
        friend bool operator<(const Edge& a, const Edge& b)
        {
            return a.c > b.c;
        }
    };
    /*====================*/
    int d[N];//图分层
    int cur[N];//当前弧优化
    Edge _edge[M];//即将加入流网络的边
    int n, m, s, t;//点，边，源，汇
    vector<int>G[N];//邻接表
    Edge edge[2 * M]; int cnt;//边
    /*====================*/
    void AddEdge(int u, int v, int c)
    {
        edge[cnt++] = Edge(u, v, c, 0);
        edge[cnt++] = Edge(v, u, 0, 0);
        G[u].push_back(cnt - 2);
    }
    /*====================*/
    bool BFS(void)
    {
        for (int i = 0; i <= n; ++i)
        {
            d[i] = INF;
        }
        queue<int>q; q.push(s); d[s] = 0;
        while (!q.empty())
        {
            int x = q.front(); q.pop();
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e = edge[G[x][i]];
                if (d[e.v] >= INF && e.c > e.f)
                {
                    d[e.v] = d[x] + 1; q.push(e.v);
                }
            }
        }
```

```
        return d[t] < INF;
    }
    int DFS(int x, int k)
    {
        int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int& i = cur[x]; i < G[x].size(); ++i)
        {
            Edge& e = edge[G[x][i]];
            if (d[x] + 1 == d[e.v] && (f = DFS(e.v, min(k, e.c - e.f))) >
0)
            {
                e.f += f; edge[G[x][i] ^ 1].f -= f;
                flow += f; k -= f; if (k == 0) break;
            }
        }
        return flow;
    }
    int Dinic(void)
    {
        int flow = 0;
        while (BFS())
        {
            flow += DFS(s, INF);
            for (int i = 1; i <= n; ++i)
            {
                cur[i] = 0;
            }
        }
        return flow;
    }
    int MaxFlow(void)
    {
        int flow = 0;
        sort(_edge, _edge + m);
        for (int type : {0, 1})
        {
            for (int p = 1 << 30, i = 0; p; p /= 2)
            {
                while (i < m && _edge[i].c >= p)
                {
                    if (type == 0)AddEdge(_edge[i].u, _edge[i].v,
_edge[i].c);
                    if (type == 1)G[_edge[i].v].push_back(i * 2 + 1); i++;
                }
                flow += Dinic();
            }
        }
        return flow;
    }
    /*====================*/
    int Init(void)
    {
        cnt = 0;
        cin >> n >> m >> s >> t;
```

```
        for (int i = 1; i <= n; ++i)
        {
            G[i].clear();
        }
        for (int i = 0; i < m; ++i)
        {
            int u, v, c;
            cin >> u >> v >> c;
            _edge[i] = Edge(u, v, c);
        }
        return MaxFlow();
    }
}
```

费用流

EK

```
namespace _EK
{
    const int N = 1e5 + 10;
    const int M = 1e5 + 10;
    /*====================*/
    const int INF = 0X3F3F3F3F;
    /*====================*/
    struct Edge
    {
        int next, v, c, w;
        Edge(int _next = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            next = _next, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n, m, s, t;
    int maxflow, mincost;
    Edge edge[2 * M]; int head[N], cnt;
    int dis[N], pre[N], incf[N]; bool vis[N];
    /*====================*/
    void AddEdge(int u, int v, int c, int w)
    {
        edge[cnt] = Edge(head[u], v, c, +w); head[u] = cnt++;
        edge[cnt] = Edge(head[v], u, 0, -w); head[v] = cnt++;
    }
    /*====================*/
    bool SPFA(void)
    {
        memset(dis, 0X3F, sizeof(dis));
        queue<int> q; q.push(s);
        dis[s] = 0, incf[s] = INF, incf[t] = 0;
        while (!q.empty())
        {
            int u = q.front(); q.pop(); vis[u] = false;
            for (int i = head[u]; i != -1; i = edge[i].next)
            {
```

```
                int v = edge[i].v, c = edge[i].c, w = edge[i].w;
                if (!c || dis[v] <= dis[u] + w) continue;
                dis[v] = dis[u] + w, incf[v] = min(c, incf[u]), pre[v] = i;
                if (!vis[v])q.push(v), vis[v] = true;
            }
        }
        return incf[t];
    }
    int MinCost(void)
    {
        while (SPFA())
        {
            maxflow += incf[t];
            for (int u = t; u != s; u = edge[pre[u] ^ 1].v)
            {
                edge[pre[u]].c -= incf[t];
                edge[pre[u] ^ 1].c += incf[t];
                mincost += incf[t] * edge[pre[u]].w;
            }
        }
        return mincost;
    }
    /*====================*/
    int Init(void)
    {
        cin >> n >> m >> s >> t;
        mincost = maxflow = cnt = 0;
        memset(head, -1, sizeof(head));
        for (int i = 1; i <= m; ++i)
        {
            int u, v, c, w;
            cin >> u >> v >> c >> w;
            AddEdge(u, v, c, w);
        }
        return MinCost();
    }
}
```

ZKW费用流

```
/*
* 使用方法
* 1.创建对象
* 2.调用AddVertex()创建点
* 3.调用Init()初始化大小
* 4.调用AddEdge()创建边
* 5.调用MinCost()获取最小费用
*/
class Min_Cost
{
public:
    static const int INF = 0X7FFFFFFF;
    /*====================*/
    struct Edge
    {
```

```
        int u, v, c, w;
        Edge(int _u = 0, int _v = 0, int _c = 0, int _w = 0)
        {
            u = _u, v = _v, c = _c, w = _w;
        }
    };
    /*====================*/
    int n = 2, s = 1, t = 2;
    /*====================*/
    vector<Edge>edge;
    vector<vector<int>>G;
    int mincost, maxflow;
private:
    vector<int>dis;
    vector<bool>vis;
    /*====================*/
    bool SPFA(void)
    {
        fill(vis.begin(), vis.end(), false);
        fill(dis.begin(), dis.end(), INF);
        vis[t] = true, dis[t] = 0;
        deque<int> q; q.push_back(t);
        while (!q.empty())
        {
            int x = q.front(); q.pop_front(), vis[x] = false;
            if (!q.empty() && dis[q.front()] > dis[q.back()])
            {
                swap(q.front(), q.back());
            }
            for (int i = 0; i < G[x].size(); ++i)
            {
                Edge& e1 = edge[G[x][i] ^ 0];
                Edge& e2 = edge[G[x][i] ^ 1];
                if (e2.c != 0 && dis[e1.v] > dis[x] - e1.w)
                {
                    dis[e1.v] = dis[x] - e1.w;
                    if (!vis[e1.v])
                    {
                        vis[e1.v] = true;
                        if (!q.empty() && dis[e1.v] < dis[q.front()])
                        {
                            q.push_front(e1.v);
                        }
                        else
                        {
                            q.push_back(e1.v);
                        }
                    }
                }
            }
        }
        return dis[s] < INF;
    }
    int DFS(int x, int k)
    {
```

```
        vis[x] = true; int flow = 0, f;
        if (x == t || k == 0) return k;
        for (int i = 0; i < G[x].size(); ++i)
        {
            Edge& e1 = edge[G[x][i] ^ 0];
            Edge& e2 = edge[G[x][i] ^ 1];
            if (vis[e1.v] || e1.c == 0)continue;
            if (dis[x] - e1.w == dis[e1.v] && (f = DFS(e1.v, min(k, e1.c)))
> 0)
            {
                e1.c -= f, e2.c += f; flow += f, k -= f;
                mincost += f * e1.w; if (k == 0) break;
            }
        }
        return flow;
    }
public:
    /*====================*/
    int S(void) { return s; }
    int T(void) { return t; }
    /*====================*/
    int AddVertex(void)
    {
        return ++n;
    }
    int AddEdge(int u, int v, int c, int w)
    {
        edge.push_back(Edge(u, v, c, +w));
        edge.push_back(Edge(v, u, 0, -w));
        G[u].push_back(edge.size() - 2);
        G[v].push_back(edge.size() - 1);
        return edge.size() - 2;
    }
    /*====================*/
    void Init(void)
    {
        dis.assign(n + 1, int());
        vis.assign(n + 1, int());
        G.assign(n + 1, vector<int>());
    }
    /*====================*/
    int MinCost(void)
    {
        maxflow = mincost = 0;
        while (SPFA())
        {
            vis[t] = true;
            while (vis[t])
            {
                fill(vis.begin(), vis.end(), false);
                maxflow += DFS(s, INF);
            }
        }
        return mincost;
    }
```

```
};
```

支配树

```
namespace Lengauer_Tarjan
{
    struct Edge
    {
        int v, x;
        Edge(int _v = 0, int _x = 0)
        {
            v = _v, x = _x;
        }
    };
    /*====================*/
    int n, m;
    Edge edge[M * 3]; int head[3][N], tot;
    int idx[N], dfn[N], dfc;
    int fa[N], fth[N], mn[N], idm[N], sdm[N];
    /*====================*/
    void Add(int x, int u, int v)
    {
        edge[head[x][u] = ++tot] = Edge(v, head[x][u]);
    }
    void Add(int u, int v)
    {
        Add(0, u, v); Add(1, v, u);
    }
    void DFS(int u)
    {
        idx[dfn[u] = ++dfc] = u;
        for (int i = head[0][u]; i; i = edge[i].x)
        {
            int v = edge[i].v;
            if (!dfn[v])
            {
                DFS(v), fth[v] = u;
            }
        }
    }
    int Find(int x)
    {
        if (fa[x] == x)
        {
            return x;
        }
        int tmp = fa[x];
        fa[x] = Find(fa[x]);
        if (dfn[sdm[mn[tmp]]] < dfn[sdm[mn[x]]])
        {
            mn[x] = mn[tmp];
        }
        return fa[x];
    }
```

```
    void Tarjan(int st)
    {
        DFS(st);
        for (int i = 1; i <= n; ++i)
        {
            fa[i] = sdm[i] = mn[i] = i;
        }
        for (int i = dfc; i >= 2; --i)
        {
            int u = idx[i], res = INF;
            for (int j = head[1][u]; j; j = edge[j].x)
            {
                int v = edge[j].v; Find(v);
                if (dfn[v] < dfn[u])
                {
                    res = min(res, dfn[v]);
                }
                else
                {
                    res = min(res, dfn[sdm[mn[v]]]);
                }
            }
            sdm[u] = idx[res];
            fa[u] = fth[u];
            Add(2, sdm[u], u);
            u = fth[u];
            for (int j = head[2][u]; j; j = edge[j].x)
            {
                int v = edge[j].v; Find(v);
                if (sdm[mn[v]] == u)
                {
                    idm[v] = u;
                }
                else
                {
                    idm[v] = mn[v];
                }
            }
            head[2][u] = 0;
        }
        for (int i = 2; i <= dfc; ++i)
        {
            int u = idx[i];
            if (idm[u] != sdm[u])
            {
                idm[u] = idm[idm[u]];
            }
        }
    }
    /*====================*/
    void Init(int s)
    {
        Tarjan(s);
        tot = dfc = 0;
        for (int i = 1; i <= n; ++i)
```

```
        {
            dfn[i] = head[0][i] = head[1][i] = head[2][i] = 0;
        }
    }
    //树上连边idm[i] -> i;
}
```

拓扑排序

```
void TopSort(C_Graph& G)
{
    vector<int>indegree(G.n + 1, 0); queue<int>q;
    for (const auto& edge : G.edge)indegree[edge.v]++;
    for (int i = 1; i <= G.n; ++i)if (indegree[i] == 0)q.push(i);
    while (!q.empty())
    {
        int cur = q.front(); q.pop(); cout << cur << " ";
        for (int i = 0; i < G[cur].size(); ++i)
        {
            int nxt = G(G[cur][i]).node(cur);
            if (--indegree[nxt] == 0)q.push(nxt);
        }
    }
}
```

差分约束

```
namespace _SDC
{
    /*
    存在负环时无解
    记得建立一个超级源点
    A-B<=W的不等式，由B->A，边权为W
    跑最短路时为最大差值，跑最长路时为最小差值
    */
    const int N = 1e5 + 10;
    /*====================*/
    const int INF = 0X3F3F3F3F;
    /*====================*/
    int dis[N]; int cnt[N]; bool vis[N];
    /*====================*/
    bool Init(void)
    {
        G[0].clear(); cnt[0] = 0;
        for (int i = 1; i <= n; ++i)
        {
            G[0].push_back(++m);
            edge[m] = Edge(0, i, 0);
            dis[i] = INF, vis[i] = false, cnt[i] = 0;
        }
        queue<int>q; dis[0] = 0; q.push(0);
        while (!q.empty())
        {
```

```
            int cur = q.front(); q.pop(); vis[cur] = false;
            for (int i = 0; i < G[cur].size(); ++i)
            {
                int val = edge[G[cur][i]].w;
                int nxt = edge[G[cur][i]].node(cur);
                if (dis[nxt] > dis[cur] + val)
                {
                    cnt[nxt] = cnt[cur] + 1;
                    dis[nxt] = dis[cur] + val;
                    if (cnt[nxt] > n)return false;
                    if (!vis[nxt])
                    {
                        q.push(nxt); vis[nxt] = true;
                    }
                }
            }
        }
        return true;
    }
}
```

图的连通性

双连通分量

边双连通分量

```
namespace _E_DCC
{
    const int N = 1e5 + 10;
    /*====================*/
    int belong[N], cnt;
    int dfn[N], low[N], num;
    /*====================*/
    void Tarjan(int cur, int in_edge)
    {
        dfn[cur] = low[cur] = ++num;
        for (int i = 0; i < G[cur].size(); ++i)
        {
            int nxt = edge[G[cur][i]].node(cur);
            if (!dfn[nxt])
            {
                Tarjan(nxt, G[cur][i]);
                low[cur] = min(low[cur], low[nxt]);
                if (low[nxt] > dfn[cur])
                {
                    edge[G[cur][i]].bridge = true;
                }
            }
            else if (i != in_edge)
            {
                low[cur] = min(low[cur], dfn[nxt]);
            }
        }
```

```
    }
    void DFS(int cur)
    {
        belong[cur] = cnt;
        for (int i = 0; i < G[cur].size(); ++i)
        {
            int nxt = edge[G[cur][i]].node(cur);
            if (edge[G[cur][i]].bridge)continue;
            if (belong[nxt])continue; DFS(nxt);
        }
    }
    /*====================*/
    void Init(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            if (!dfn[i])Tarjan(i, 0);
        }
        for (int i = 1; i <= n; ++i)
        {
            if (!belong[i])cnt++, DFS(i);
        }
    }
}
```

点双连通分量

```
namespace _V_DCC
{
    const int N = 1e5 + 10;
    /*====================*/
    vector<int>dcc[N];
    bool cut[N]; int cnt;
    stack<int>lib; int root;
    int dfn[N], low[N], num;
    /*====================*/
    void Tarjan(int cur)
    {
        int flag = 0; lib.push(cur);
        dfn[cur] = low[cur] = ++num;
        if (cur == root && G[cur].size() == 0)
        {
            dcc[++cnt].push_back(cur); return;
        }
        for (int i = 0; i < G[cur].size(); ++i)
        {
            int nxt = edge[G[cur][i]].node(cur);
            if (!dfn[nxt])
            {
                Tarjan(nxt);
                low[cur] = min(low[cur], low[nxt]);
                if (low[nxt] >= dfn[cur])
                {
                    flag++; cnt++; int top;
                    if (cur != root || flag > 1)
```

```
                    {
                        cut[cur] = true;
                    }
                    do
                    {
                        top = lib.top(); lib.pop();
                        dcc[cnt].push_back(top);
                    } while (top != nxt);
                    dcc[cnt].push_back(cur);
                }
            }
            else
            {
                low[cur] = min(low[cur], dfn[nxt]);
            }
        }
    }
    /*====================*/
    void Init(void)
    {
        for (int i = 1; i <= n; ++i)
        {
            if (!dfn[i])Tarjan(root = i);
        }
    }
}
```

获取点双内部的边

```
void Tarjan(int cur, int e)
{
    dfn[cur] = low[cur] = ++num;
    if (cur != root || G[cur].size() != 0)
    {
        for (int i = 0; i < G[cur].size(); ++i)
        {
            int nxt = edge[G[cur][i]].node(cur);
            if (!dfn[nxt])
            {
                lib.push(G[cur][i]); Tarjan(nxt, G[cur][i]);
                low[cur] = min(low[cur], low[nxt]);
                if (low[nxt] >= dfn[cur])
                {
                    cnt++; int top;
                    do
                    {
                        top = lib.top(); lib.pop();
                        dcc[cnt].push_back(edge[top].w);
                    } while (top != G[cur][i]);
                }
            }
            else
            {
                if (dfn[nxt] < dfn[cur] && G[cur][i] != e)
                {
```

```
                    lib.push(G[cur][i]);
                }
                low[cur] = min(low[cur], dfn[nxt]);
            }
        }
    }
}
```

强连通分量

```
namespace _SCC
{
    const int N = 1e5 + 10;
    /*====================*/
    int belong[N];
    int dfn[N], low[N], num;
    stack<int>lib; int ins[N];
    vector<int>scc[N]; int cnt;
    /*====================*/
    void Tarjan(int cur)
    {
        lib.push(cur); ins[cur] = 1;
        dfn[cur] = low[cur] = ++num;
        for (int i = 0; i < G[cur].size(); ++i)
        {
            int nxt = edge[G[cur][i]].node(cur);
            if (!dfn[nxt])
            {
                Tarjan(nxt);
                low[cur] = min(low[cur], low[nxt]);
            }
            else if (ins[nxt])
            {
                low[cur] = min(low[cur], dfn[nxt]);
            }
        }
        if (dfn[cur] == low[cur])
        {
            cnt++; int top;
            do
            {
                top = lib.top(); lib.pop(); ins[top] = 0;
                belong[top] = cnt; scc[cnt].push_back(top);
            } while (top != cur);
        }
    }
    /*====================*/
    void Init(void)
    {
        num = cnt = 0;
        for (int i = 1; i <= n; ++i)
        {
            dfn[i] = low[i] = 0;
        }
        for (int i = 1; i <= n; ++i)
```

```
        {
            if (!dfn[i])Tarjan(i);
        }
    }
}
```

最小树形图

有向有环图

```
挖坑：朱刘算法
```

有向无环图

```
namespace _DMST
{
    const int N = 1e5 + 10;
    /*====================*/
    const int INF = 0X7FFFFFFF;
    /*====================*/
    int val[N], sum;
    /*====================*/
    void Init(void)
    {
        sum = 0;
        for (int i = 1; i <= n; ++i)
        {
            val[i] = INF;
        }
        for (int i = 1; i <= m; ++i)
        {
            int u = edge[i].u;
            int v = edge[i].v;
            int w = edge[i].w;
            val[v] = min(val[v], w);
        }
        for (int i = 1; i <= n; ++i)
        {
            if (val[i] != INF)
            {
                sum += val[i];
            }
        }
    }
}
```

三元环计数

```
const int N = 1e5 + 10;
const int M = 2e5 + 10;
/*====================*/
int n, m;
```

```cpp
struct Edge
{
    int u, v;
    Edge(int _u = 0, int _v = 0)
    {
        u = _u, v = _v;
    }
};
Edge edge[M];
int degree[N];
vector<int>Out[N];
/*====================*/
int tag[N];
/*====================*/
void Solve(void)
{
    cin >> n >> m;
    for (int i = 1; i <= m; ++i)
    {
        int u, v;
        cin >> u >> v;
        edge[i] = Edge(u, v);
        degree[u]++, degree[v]++;
    }
    for (int i = 1; i <= m; ++i)
    {
        int u = edge[i].u, v = edge[i].v;
        if (degree[u] == degree[v] && u > v)swap(u, v);
        if (degree[u] != degree[v] && degree[u] > degree[v])swap(u, v);
        Out[u].push_back(v);
    }
    int ans = 0;
    for (int u = 1; u <= n; ++u)
    {
        for (auto v : Out[u])
        {
            tag[v] = u;
        }
        for (auto v : Out[u])
        {
            for (auto w : Out[v])
            {
                if (tag[w] == u)
                {
                    ans++;
                }
            }
        }
    }
    cout << ans << endl;
}
```

字符串

Trie

01Trie

```
template<int BIT = 31>
class C_Trie
{
private:
    int root, cnt;
    vector<int>siz;
    vector<vector<int>>trie;
public:
    void Init(int sigma_n)
    {
        sigma_n *= BIT;
        cnt = -1; root = ++cnt;
        siz.assign(sigma_n + 1, 0);
        trie.assign(sigma_n + 1, vector<int>(2, -1));
    }
    void Insert(int val)
    {
        int cur = root; siz[cur]++;
        for (int i = BIT - 1; i >= 0; --i)
        {
            int bit = (val >> i) & 1;
            if (trie[cur][bit] == -1)
            {
                trie[cur][bit] = ++cnt;
            }
            siz[cur = trie[cur][bit]]++;
        }
    }
    int Query(int val)
    {
        int cur = root;
        for (int i = BIT - 1; i >= 0; --i)
        {
            int bit = (val >> i) & 1;
            if (trie[cur][bit] == -1)
            {
                return 0;
            }
            cur = trie[cur][bit];
        }
        return siz[cur];
    }
    int MaxXOR(int val)
    {
        int res = 0, cur = root;
        for (int i = BIT - 1; i >= 0; --i)
        {
            int bit = ((val >> i) & 1) ^ 1;
            if (trie[cur][bit] == -1)bit ^= 1;
            else res += 1 << i; cur = trie[cur][bit];
```

```
        }
        return res;
    }
};
```

字符Trie

```
template<int ASCII = 128>
class C_Trie
{
private:
    int root, cnt;
    vector<int>siz;
    vector<vector<int>>trie;
public:
    void Init(int sigma_s)
    {
        cnt = -1; root = ++cnt;
        siz.assign(sigma_s + 1, 0);
        trie.assign(sigma_s + 1, vector<int>(ASCII, -1));
    }
    void Insert(const string& str)
    {
        int cur = root; siz[cur]++;
        for (int i = 0; i < str.size(); ++i)
        {
            if (trie[cur][str[i]] == -1)
            {
                trie[cur][str[i]] = ++cnt;
            }
            siz[cur = trie[cur][str[i]]]++;
        }
    }
    int Query(const string& str)
    {
        int cur = root;
        for (int i = 0; i < str.size(); ++i)
        {
            if (trie[cur][str[i]] == -1)
            {
                return 0;
            }
            cur = trie[cur][str[i]];
        }
        return siz[cur];
    }
};
```

Hash

单Hash

```
class C_Hash
{
private:
    static const lnt MOD = 998244353;
    /*====================*/
    vector<lnt>powbase, invbase, sumhash;
public:
    void Init(const string& str, lnt base = 233)
    {
        powbase.assign(str.size(), 0);
        invbase.assign(str.size(), 0);
        sumhash.assign(str.size(), 0);
        /*====================*/
        for (int i = 0; i < str.size(); ++i)
        {
            if (i == 0)powbase[i] = 1;
            else powbase[i] = powbase[i - 1] * base % MOD;
        }
        base = Pow(base % MOD, MOD - 2, MOD);
        for (int i = 0; i < str.size(); ++i)
        {
            if (i == 0)invbase[i] = 1;
            else invbase[i] = invbase[i - 1] * base % MOD;
        }
        /*====================*/
        for (int i = 0; i < str.size(); ++i)
        {
            if (i == 0)sumhash[i] = str[i] * powbase[i] % MOD;
            else sumhash[i] = (sumhash[i - 1] + str[i] * powbase[i]) % MOD;
        }
    }
    lnt operator()(int l, int r)
    {
        return (sumhash[r] - (l > 0 ? sumhash[l - 1] : 0) + MOD) *
invbase[l] % MOD;
    }
};
```

双Hash

```
class C_DoubleHash
{
private:
    C_Hash Hash1, Hash2;
public:
    void Init(const string& str, lnt base1 = 233, lnt base2 = 19260817)
    {
        Hash1.Init(str, base1), Hash2.Init(str, base2);
    }
    pair<lnt, lnt> operator()(int l, int r)
    {
        return { Hash1(l,r),Hash2(l,r) };
```

```
    }
};
```

Split

```
class C_Split
{
public:
    vector<string> operator()(const string &str, char c)
    {
        string temp;
        vector<string>res;
        istringstream iss(str);
        while (getline(iss, temp, c))
        {
            if (temp != "")
            {
                res.push_back(temp);
            }
        }
        return res;
    }
};
```

Z函数

```
vector<int> Z_Function(const string& str)
{
    int n = str.size() - 1;
    vector<int>z(str.size());
    int l = 1, r = 1; z[1] = n;
    for (int i = 2; i <= n; ++i)
    {
        z[i] = (i <= r ? min(z[i - l + 1], r - i + 1) : 0);
        while (i + z[i] <= n && str[1 + z[i]] == str[i + z[i]])z[i]++;
        if (i + z[i] - 1 > r)r = i + z[i] - 1, l = i;
    }
    return z;
}
```