Menu

0. CPP_HEAD

```
#include<bits/stdc++.h>
using namespace std;

/*----------Consts----------*/
const long MOD=1e9+7;
const double eps=1e-6;

const double pi = acos(-1.0);
const long long INF=0x3fffffffffffffff;
/*----------Consts----------*/

typedef long long ll;
typedef long double ld;
typedef pair<ll,ll> pll;

#define lowbit(x) ((x)&(-(x)))
#define ALL(A) (A).begin(),(A).end()
#define SORT(A) sort(ALL(A))
#define SORT_REV(A) sort((A).rbegin(),(A).rend())
#define FORLL(i,l,r) for(ll i=l;i<=r;i++)
#define FORLL_rev(i,r,l) for(ll i=r;i>=l;i--)
#define Presentation(i,r) " \n"[i==r]
#define endl '\n'

#define ONLINE_JUDGE
#define FAST_IO
#define MUTIPLE_JUDGE

/*----------Code Area----------*/
const ll N = 200005;
void solve()
{

}
/*----------Code Area----------*/

unsigned main(){
    #ifndef ONLINE_JUDGE
        if(freopen("in.txt", "r", stdin)==NULL) {cout << "Fail opening in.txt!" <<
endl;return 0;}
        if(freopen("out.txt", "w", stdout)==NULL) {cout << "Fail opening out.txt!"
<< endl;return 0;}
    #endif

    #ifdef FAST_IO
        ios::sync_with_stdio(false);
        cin.tie(nullptr); cout.tie(nullptr);
    #endif
```

```
    #ifdef MUTIPLE_JUDGE
        long T; cin >> T;
        while(T--) solve();
    #else
        solve();
    #endif

    return 0;
}
```

1. 模下运算

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

namespace MOLDULE
{
    const ll MOD = 1e9 + 7;
    inline ll Get_Mod(ll x) {return (x-x/MOD*MOD+MOD)%MOD;}
    ll qcpow(ll x, ll y) {
        ll res = 1; x = Get_Mod(x);
        for (; y; y >>= 1, x = x * x % MOD)
            if (y & 1) res = res * x % MOD;
        return res;}
    inline ll inv(ll x) {return qcpow(x,MOD-2);}
    inline ll add(ll x, ll y) {return Get_Mod(x + y);}
    inline ll addto(ll &x, ll y) {return x = add(x, y);}
    inline ll sub(ll x, ll y) {return Get_Mod(x - y);}
    inline ll subto(ll &x, ll y) {return x = sub(x, y);}
    inline ll mul(ll x, ll y) {return Get_Mod(1ll*x * y);}
    inline ll multo(ll &x, ll y) {return x = mul(x, y);}
    inline ll mdiv(ll x, ll y) {return Get_Mod(1ll*x*inv(y));}
    inline ll mdivto(ll &x, ll y) {return x = mdiv(x, y);}
}
```

2. int128

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

struct int128{
    __int128_t value;

    int128():value(0){}
    int128(ll _val):value(_val){}
    int128(__int128_t _val):value(_val){}

    static int128 trans(string input) {
        bool isNegative = false;
        if (input[0] == '-') {isNegative = true;input = input.substr(1);}
        __int128_t result=0;
        for (char c : input) {result = result * 10 + (c - '0');}
        if (isNegative) {result = -result;}
        return int128(result);
    }

    void print() const {
        __int128_t x = value;
        if (x < 0) {putchar('-');x = -x;}
        if (x > 9) {int128(x / 10).print();}
        putchar(x % 10 + '0');
    }

    int128 operator + (const int128 &b)const{return value+b.value;}
    int128 operator - (const int128 &b)const{return value-b.value;}
    int128 operator * (const int128 &b)const{return value*b.value;}
    int128 operator / (const int128 &b)const{return value/b.value;}
};

istream& operator>>(istream& in,int128& x){
    string Input;
    in >> Input;
    x.trans(Input);
    return in;
}

ostream& operator<<(ostream& out,const int128& x){
    x.print();
    return out;
}

int main(){
    int128 i;
    i=0x3fffffffffffffff;
```

```
    ll t;cin >> t;
    ll x;
    while(t--){
        cin >> x;i=i/x;
        i.print();putchar('\n');
    }
    return 0;
}
```

3. 树状数组

```
#include <bits/stdc++.h>
using namespace std;

#define N 100000
typedef long long ll;

#define lowbit(x) ((x) & (-(x))) // 取最后一个1所在位置的权值
struct BITree
{ // 树状数组，下标i从1开始
    vector<ll> Data;

    explicit BITree(ll n) : Data(n * 2 + 5, 0) {}

    void update(ll i, ll dif)
    { // 给予i增量dif,维护树状数组，O(logn)
        while (i < Data.size())
        {
            Data[i] += dif;
            i += lowbit(i);
        }
    }

    ll presum(ll i)
    { // 查询前缀和sum[i]，O(logn)
        ll sum = 0;
        while (i)
        {
            sum += Data[i];
            i -= lowbit(i);
        }
        return sum;
    }

    ll query(ll l, ll r)
    { // 查询区间和
        return presum(r) - presum(l - 1);
    }

    ll operator[](ll index)
    { // 下标调用元素（只读）
        return query(index, index);
    }
};

int main()
{
    ll n, t, tt = 0;
    cin >> n;
```

```
    BITree bt(n);
    for (ll i = 1; i <= n; i++)
    {
        cin >> t;
        bt.update(i, t); // 维护原数组，实现单点修改，区间查询
        /*
        bt.update(i,t-tt);tt=t;
        维护差分数组，实现区间修改，单点查询
        对区间[l,r]的修改变为update(l,dif);update(r+1,-dif);
        对元素a[i]的查询变为presum(i);
        */
    } // 建树O(nlogn)
    ll l, r;
    cin >> l >> r;
    cout << bt.query(l, r) << endl;
    ll i, x;
    cin >> i >> x;
    bt.update(i, x);
    cin >> l >> r;
    cout << bt.query(l, r) << endl;
    return 0;
}
```

4. 并查集

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

struct DSU
{
    vector<ll> parents, size;

    explicit DSU(ll n) : parents(n + 1), size(n + 1, 1) { iota(parents.begin(),
parents.end(), 0); }

    ll find(ll x) { return (parents[x] == x) ? x : (parents[x] =
find(parents[x])); }

    void merge(ll a, ll b)
    { // merge a into b
        a = find(a);
        b = find(b);
        if (a == b)
            return;
        if (size[a] > size[b])
            swap(a, b);
        parents[a] = b;
        size[b] += size[a];
    }
};
```

5. 字符串Hash

```
#include <bits/stdc++.h>
using namespace std;

#define N 200005
struct strHash
{ // 字符串哈希
    typedef long long ll;
    typedef pair<ll, ll> pll;
    const ll P1 = 57751, mod1 = 1e9 + 7, P2 = 43331, mod2 = 1e9 + 9;
    size_t length, size;
    vector<ll> hz1, hf1, pz1, pf1, hz2, hf2, pz2, pf2;
    // h:Hash;p:Pow;
    // z:正向;f:反向;
    // 1/2:双Hash;下标从1开始
    strHash(string str)
    {
        length = size = str.length();
        str = ' ' + str;
        hz1.resize(size + 2);
        pz1.resize(size + 2);
        hf1.resize(size + 2);
        pf1.resize(size + 2);
        hz2.resize(size + 2);
        pz2.resize(size + 2);
        hf2.resize(size + 2);
        pf2.resize(size + 2);
        pz1[0] = 1;
        for (int i = 1; i <= size; i++)
        {
            hz1[i] = (hz1[i - 1] * P1 + str[i]) % mod1;
            pz1[i] = pz1[i - 1] * P1 % mod1;
        }
        pf1[size + 1] = 1;
        for (int i = size; i >= 1; i--)
        {
            hf1[i] = (hf1[i + 1] * P1 + str[i]) % mod1;
            pf1[i] = pf1[i + 1] * P1 % mod1;
        }
        pz2[0] = 1;
        for (int i = 1; i <= size; i++)
        {
            hz2[i] = (hz2[i - 1] * P2 + str[i]) % mod2;
            pz2[i] = pz2[i - 1] * P2 % mod2;
        }
        pf2[size + 1] = 1;
        for (int i = size; i >= 1; i--)
        {
            hf2[i] = (hf2[i + 1] * P2 + str[i]) % mod2;
```

```
            pf2[i] = pf2[i + 1] * P2 % mod2;
        }
    }
    pll findz(int l, int r)
    { // 返回[l,r]的正向双Hash
        return {((hz1[r] - hz1[l - 1] * pz1[r - l + 1]) % mod1 + mod1) % mod1,
((hz2[r] - hz2[l - 1] * pz2[r - l + 1]) % mod2 + mod2) % mod2};
    }
    pll findf(int l, int r)
    { // 返回[l,r]的反向双Hash
        return {((hf1[l] - hf1[r + 1] * pf1[length - r + l]) % mod1 + mod1) %
mod1, ((hf2[l] - hf2[r + 1] * pf2[length - r + l]) % mod2 + mod2) % mod2};
    }
    bool isPalin(ll l, ll r)
    { // 判断[l,r]是否为回文串
        return findz(l, r) == findf(l, r);
    }
};
```

6. 线性筛+欧拉函数

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

const ll N = 1000;
bool check[N + 10]; // check=1表示合数，被筛除
ll phi[N + 10]; // 欧拉函数，phi[i]表示1~i中与i互质的数的个数
ll prime[N + 10]; // 素数表，下标从0开始
ll tot; // 素数的个数
void Phi_and_Prime_Table(ll N)
{
    memset(check, false, sizeof(check));
    phi[1] = 1;
    tot = 0;
    for (ll i = 2; i <= N; i++)
    {
        if (!check[i])
        {
            prime[tot++] = i;
            phi[i] = i - 1;
        }
        for (ll j = 0; j < tot; j++)
        {
            if (i * prime[j] > N)
                break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else
            {
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            }
        }
    }
}
```

7. 归并排序与逆序对计数

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll mergeAndCount(vector<ll> &arr, ll l, ll m, ll r)
{
    vector<ll> temp(r - l + 1);
    ll invCount = 0;
    ll i = l, j = m + 1, k = 0;
    while (i <= m && j <= r)
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
            invCount += m - i + 1;
        }
    }
    while (i <= m)
        temp[k++] = arr[i++];
    while (j <= r)
        temp[k++] = arr[j++];
    for (ll p = 0; p < temp.size(); p++)
        arr[l + p] = temp[p];

    return invCount;
}

ll mergeSortAndCount(vector<ll> &arr, ll l, ll r)
{
    ll invCount = 0;
    if (l < r)
    {
        ll m = l + (r - l) / 2;
        invCount += mergeSortAndCount(arr, l, m);
        invCount += mergeSortAndCount(arr, m + 1, r);
        invCount += mergeAndCount(arr, l, m, r);
    }
    return invCount;
}
```

8. 大数因子快速随机

```
#include <bits/stdc++.h>
// Pollard_rho 大数因子分解快速随机算法
// Miller-Rabin 素数性测试算法
using namespace std;
typedef long long ll;
ll qcpow_p(ll a, ll b, ll p)
{
    ll ret = 1;
    for (; b; b >>= 1, a = (__int128)a * a % p)
        if (b & 1)
            ret = (__int128)ret * a % p;
    return ret;
}
bool Miller_Rabin(ll p)
{
    if (p < 2)
        return 0;
    if (p == 2 || p == 3)
        return 1;
    ll d = p - 1, r = 0;
    while (!(d & 1))
        ++r, d >>= 1;
    for (ll k = 0; k < 10; ++k)
    {
        ll a = rand() % (p - 2) + 2;
        ll x = qcpow_p(a, d, p);
        if (x == 1 || x == p - 1)
            continue;
        for (int i = 0; i < r - 1; ++i)
        {
            x = (__int128)x * x % p;
            if (x == p - 1)
                break;
        }
        if (x != p - 1)
            return 0;
    }
    return 1;
}
ll Pollard_Rho(ll x)//随机返回x的一个因子
{
    ll s = 0, t = 0;
    ll c = (ll)rand() % (x - 1) + 1;
    int step = 0, goal = 1;
    ll val = 1;
    for (goal = 1;; goal <<= 1, s = t, val = 1)
    {
        for (step = 1; step <= goal; step++)
```

```
        {
            t = ((__int128)t * t + c) % x;
            val = (__int128)val * abs(t - s) % x;
            if (step % 127 == 0)
            {
                ll d = __gcd(val, x);
                if (d > 1)
                    return d;
            }
        }
        ll d = __gcd(val, x);
        if (d > 1)
            return d;
    }
}
int main()
{
    ll t, n;
    cin >> t;
    while (t--)
    {
        cin >> n;
        cout << Pollard_Rho(n) << endl;
    }
    return 0;
}
```

9. 二部图最大匹配

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
struct augment_path
{
    vector<vector<ll>> g;
    vector<ll> pa; // 匹配
    vector<ll> pb;
    vector<ll> vis; // 访问
    ll n, m;        // 两个点集中的顶点数量
    ll dfn;         // 时间戳记
    ll res;         // 匹配数
    augment_path(ll _n, ll _m) : n(_n), m(_m)
    {
        assert(0 <= n && 0 <= m);
        pa = vector<ll>(n, -1);
        pb = vector<ll>(m, -1);
        vis = vector<ll>(n);
        g.resize(n);
        res = 0;
        dfn = 0;
    }
    void add(ll from, ll to)
    {
        assert(0 <= from && from < n && 0 <= to && to < m);
        g[from].push_back(to);
    }
    bool dfs(ll v)
    {
        vis[v] = dfn;
        for (ll u : g[v])
        {
            if (pb[u] == -1)
            {
                pb[u] = v;
                pa[v] = u;
                return true;
            }
        }
        for (ll u : g[v])
        {
            if (vis[pb[u]] != dfn && dfs(pb[u]))
            {
                pa[v] = u;
                pb[u] = v;
                return true;
            }
```

```
        }
        return false;
    }
    ll solve()
    {
        while (true)
        {
            dfn++;
            ll cnt = 0;
            for (ll i = 0; i < n; i++)
            {
                if (pa[i] == -1 && dfs(i))
                {
                    cnt++;
                }
            }
            if (cnt == 0)
            {
                break;
            }
            res += cnt;
        }
        return res;
    } // 返回最大匹配数
};
int main()
{
    ll n, m;
    cin >> n >> m;
    augment_path G(n, n);
    ll u, v;
    for (ll i = 0; i < m; i++)
    {
        cin >> u >> v;
        G.add(u, v);
    }
    cout << G.solve() << endl;
    return 0;
}
```

10. 高精度加乘

```
#include<bits/stdc++.h>
using namespace std;

//高精度正整数计算
namespace CCHA
{
    string HAintadd(const string& num1, const string& num2)
    {
        int len1 = num1.length();
        int len2 = num2.length();
        int diff = len1 - len2;
        int carry = 0;

        if (diff < 0) return HAintadd(num2, num1);

        string result(len1 + 1 , '0');

        for (int i = len1 - 1; i >= 0; i--)
        {
            int digitSum = (num1[i] - '0') + (i - diff >= 0 ? num2[i - diff] - '0'
: 0) + carry;
            carry = digitSum / 10;
            result[i + 1] = (digitSum % 10) + '0';
        }

        if (carry)
            result[0] = carry + '0';
        else
            result.erase(result.begin());

        return result;
    }

    string HAintmul(const string& num1, const string& num2)
    {
        int len1 = num1.length();
        int len2 = num2.length();
        string result(len1 + len2, '0');

        for (int i = len1 - 1; i >= 0; i--)
        {
            int carry = 0;
            for (int j = len2 - 1; j >= 0; j--)
            {
                int digit = (num1[i] - '0') * (num2[j] - '0') + (result[i + j + 1]
- '0') + carry;
                carry = digit / 10;
                result[i + j + 1] = (digit % 10) + '0';
```

```
            }
            result[i] = carry + '0';
        }

        size_t pos = result.find_first_not_of('0');
        if (pos != string::npos) result.erase(0, pos);
        else result = "0";

        return result;
    }
};
```

11. 二维计算几何+凸包

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<ll, ll> pll;

namespace DEFINITION
{
#define scanfll(a) scanf("%lld", &a)
#define lowbit(x) ((x) & (-(x)))
#define RESET(A) memset(A, 0, sizeof(A))
#define ALL(A) A.begin(), A.end()
#define SORT(A) sort(ALL(A))
#define Presentation(i, r) " \n"[i == r]
#define FORLL(i, l, r) for (ll i = l; i <= r; i++)
#define FORLL_rev(i, r, l) for (ll i = r; i >= l; i--)
#define Get_Mod(a) (((a) + MOD) % MOD)
#define NO "NO\n"
#define YES "YES\n"
}
using namespace DEFINITION;

/*----------Consts----------*/
const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
/*----------Consts----------*/
int sgn(double x)
{
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
inline double sqr(double x) { return x * x; }

struct Point
{
    double x, y;
    Point() {} // Empty Point
    Point(double _x, double _y) { x = _x; y = _y; } // Point

    void input() { cin >> x >> y; }

    bool operator==(Point b) const { return sgn(x - b.x) == 0 && sgn(y - b.y) ==
0; }
    bool operator<(Point b) const { return sgn(x - b.x) == 0 ? sgn(y - b.y) < 0 :
x < b.x; }
    bool operator>(Point b) const { return sgn(x - b.x) == 0 ? sgn(y - b.y) > 0 :
```

```
x > b.x; }

    Point operator-(const Point &b) const { return Point(x - b.x, y - b.y); } //
相减(转向量)：A-B=BA
    Point operator+(const Point &b) const { return Point(x + b.x, y + b.y); } //
向量和

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

    double len() { return hypot(x, y); }    // 向量长度
    double len2() { return x * x + y * y; } // 向量长度平方

    double distance(Point p) { return hypot(x - p.x, y - p.y); } // 与另一点的距离

    Point operator*(const double &k) const { return Point(x * k, y * k); }
    Point operator/(const double &k) const { return Point(x / k, y / k); }

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

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

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

    // 绕着 p 点逆时针旋转angle(弧度制)
    Point rotate(Point p, double angle)
    {
        Point v = (*this) - p;
        double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x * c - v.y * s, p.y + v.x * s + v.y * c);
    }
};

// 计算凸包
vector<Point> Convex_Hull(vector<Point> pvec)
{
    vector<Point> ch;
    ll n = pvec.size();
    SORT(pvec);
    vector<ll> stk(n + 1);
    ll top = 0;
    stk[++top] = 0;
```

```
    vector<bool> used(n + 1, false);
    FORLL(i, 1, n - 1)
    {
        while (top > 1 && (pvec[stk[top]] - pvec[stk[top - 1]]).operator^(pvec[i]
- pvec[stk[top]]) <= 0)
            used[stk[top--]] = false;
        stk[++top] = i;
        used[i] = true;
    }
    ll tmp = top;
    FORLL_rev(i, n - 2, 0) if (!used[i])
    {
        while (top > tmp && (pvec[stk[top]] - pvec[stk[top -
1]]).operator^(pvec[i] - pvec[stk[top]]) <= 0)
            used[stk[top--]] = false;
        stk[++top] = i;
        used[i] = true;
    }
    FORLL(i, 1, top)
    ch.emplace_back(pvec[stk[i]]);
    return ch;
}
```

12. 字典树

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
//字典树
struct TRIE{
    vector<array<int,62>> nxt;
    vector<int> cnt;
    ll n,curn;
    TRIE(ll n_):nxt(n_+1,{0}),cnt(n_+1,0),n(n_),curn(0){} //n是字符串总长度
    void insert(string s){
        ll p=0;
        for(auto c:s){
            if(c>='a'&&c<='z') c-='a';else if(c>='A'&&c<='Z') c-='A'-26; else c-
='0'-52;
            if(nxt[p][c]==0) nxt[p][c]=++curn;
            p=nxt[p][c];
            cnt[p]++; //统计前缀出现次数
        }
        // cnt[p]++; //统计单词出现次数
    }
    ll count(string s){
        ll p=0;
        for(auto c:s){
            if(c>='a'&&c<='z') c-='a';else if(c>='A'&&c<='Z') c-='A'-26; else c-
='0'-52;
            if(nxt[p][c]==0) return 0;
            p=nxt[p][c];
        }
        return cnt[p];
    }
    void clear(){
        FORLL(i,0,curn+1){
            FORLL(j,0,61) nxt[i][j]=0;
            cnt[i]=0;
        }
        curn=0;
    }
};
const ll N=3e6+6;
TRIE trie(N);
```

13. 快速幂

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll MOD = 1e9 + 7;
#define Get_Mod(x) ((x) % MOD + MOD) % MOD

ll qcpow(ll x, ll b)
{
    ll ret = 1;
    x = Get_Mod(x);
    for (; b; b >>= 1, x = 1ll * x * x % MOD)
        if (b & 1)
            ret = Get_Mod(1ll * ret * x);
    return ret;
}
```

14. 扩展欧几里得

```
    // 扩欧返回d=gcd(a,b);x,y对应ax+by=d的解
    ll Exgcd(ll a, ll b, ll &x, ll &y)
    {
        if (a == 0 && b == 0)
            return -1;
        if (b == 0)
        {
            x = 1;
            y = 0;
            return a;
        }
        ll d = Exgcd(b, a % b, y, x);
        y -= a / b * x;
        return d;
    }
```

15. 预处理阶乘+第二类Stirling数

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define FORLL(i,l,r) for(ll i=l;i<=r;i++)

using namespace MOLDULE;

vector<ll> Fac, Fac_inv;
void Prepare_Factorium(ll n)
{
    Fac.clear();
    Fac.resize(n + 1);
    Fac[0] = Fac[1] = 1;
    Fac_inv.clear();
    Fac_inv.resize(n + 1);
    Fac_inv[0] = Fac_inv[1] = 1;
    FORLL(i,2,n)
    {
        Fac[i] = Get_Mod(Fac[i - 1] * i);
        Fac_inv[i] = qcpow(Fac[i], MOD - 2);
    }
}
void Prepare_Combination(ll n) { Prepare_Factorium(n); }
ll Get_Combination(ll m, ll n) { return Get_Mod(Get_Mod(Fac[m] * Fac_inv[m - n]) *
Fac_inv[n]); }

//第二类Stirling数，n个不同球放入m个不同盒子的方案数 复杂度O(mlogn)
ll Get_Stirling(ll n,ll m){
    ll ans=0;
    if(n<m) return 0;
    FORLL(i,0,m){
        ll t=mul(Get_Combination(m,i),qcpow(m-i,n));
        if((m-i)&1) t=MOD-t;
        addto(ans,t);
    }return ans;
}
```

16. 非负权值单源最短路Dijkstra

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF=0x3fffffffffffffff;
#define FORLL(i,l,r) for(ll i=l;i<=r;i++)

//非负权值单源最短路Dijkstra O(mlogm)
class Dijkstra{
private:
    struct Edge{ ll v, w; };
    struct Node{
        ll dis, u;
        bool operator>(const Node &a) const { return dis > a.dis; }
    };
    vector<vector<Edge>> G;
    vector<int> vis;
    vector<ll> dis;
    priority_queue<Node,vector<Node>,greater<Node>> Q;
    ll n=0;
    //换源前初始化
    void Init(){
        while(!Q.empty()) Q.pop();
        FORLL(i,1,n){
            vis[i]=0;
            dis[i]=INF;
        }
    }
public:
    Dijkstra(ll _n):n(_n),G(_n+1),vis(_n+1,0),dis(_n+1,INF){}
    void AddEdge(ll u,ll v,ll w){//加边
        G[u].push_back({v,w});
    }
    //s为源点的单源最短路
    void Solve(ll s){
        Init();
        dis[s] = 0;
        Q.push({0, s});
        while (!Q.empty()) {
            ll u = Q.top().u;
            Q.pop();
            if (vis[u]) continue;
            vis[u] = 1;
            for (auto e : G[u]) {
                ll v = e.v, w = e.w;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    Q.push({dis[v], v});
                }
```

```
                }
            }
        }
        //访问dis数组：到t的最短路
        ll getDis(ll t){
            return dis[t];
        }
        //访问dis数组
        ll operator[](ll i){
            return dis[i];
        }
    };
```

17. 最小生成树Kruskal

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<ll,ll> pll;
#define FORLL(i,l,r) for(ll i=l;i<=r;i++)

struct DSU{};//并查集

//最小生成树
struct MSTree{
private:
    struct Edge{
        ll u,v,w;
        bool operator>(const Edge &e) const{
            return w > e.w;
        }
    };
    ll n;
    DSU dsu;
    priority_queue<Edge,vector<Edge>,greater<Edge>> Q;
public:
    ll ans=0;
    vector<vector<pll>> G;
    vector<ll> fa;
    MSTree(ll _n):n(_n),dsu(_n),G(_n+1),fa(n+1,0){}
    void add_edge(ll u,ll v,ll w){
        Q.push({u,v,w});
    }
    void solve(){
        ans = 0;
        while(!Q.empty()){
            auto e = Q.top();Q.pop();
            if(dsu.find(e.u) != dsu.find(e.v)){
                dsu.merge(e.u,e.v);
                G[e.u].emplace_back(e.v,e.w);
                G[e.v].emplace_back(e.u,e.w);
                ans += e.w;
            }
        }
        //dfs求fa
        auto DFS = [&](auto &&self, ll u=1,ll f=0) -> void{
            fa[u] = f;
            for(auto &p:G[u]){
                ll v = p.first;
                if(v == f) continue;
                self(self,v,u);
            }
        };
```

```
        DFS(DFS);
        // DFS(); //求fa
    }
    //判断是否连通
    bool connected(){
        return dsu.size[dsu.find(1)] == n;
    }
};
void solve(){
    ll n,m;cin >> n >> m;
    MSTree mst(n);
    ll u,v,w;
    FORLL(i,1,m){
        cin >> u >> v >> w;
        mst.add_edge(u,v,w);
    }
    mst.solve();
    if(mst.connected()) cout << mst.ans << endl;
    else cout << "orz\n";
}
```

18. 线段树

```
// 线段树 实现logn的区间修改和查询，下标从0开始
// 使用前确认采用的运算方法
template <typename T>
class SegTree{
    vector<T> tree, lazyadd, lazymul, lazyset;
    vector<T> *arr;
    int n, root, n4, end;

    void maintain(int cl, int cr, int p) {
        int cm = cl + (cr - cl) / 2;
        if (cl != cr) {//cl==cr：叶子节点

            if(lazymul[p]!=1){ //区间乘法
                multo(lazymul[p * 2],lazymul[p]); //lazymul[p * 2] *= lazymul[p];
                multo(lazymul[p * 2 + 1],lazymul[p]); //lazymul[p * 2 + 1] *=
lazymul[p];
                multo(lazyadd[p * 2],lazymul[p]); //lazyadd[p * 2] *= lazymul[p];
                multo(lazyadd[p * 2 + 1],lazymul[p]); //lazyadd[p * 2 + 1] *=
lazymul[p];
                multo(tree[p * 2],lazymul[p]); //tree[p * 2] *= lazymul[p];
                multo(tree[p * 2 + 1],lazymul[p]); //tree[p * 2 + 1] *=
lazymul[p];
                lazymul[p] = 1;
            }

            if(lazyadd[p]){ //区间增量
                addto(lazyadd[p * 2],lazyadd[p]); //lazyadd[p * 2] += lazyadd[p];
                addto(lazyadd[p * 2 + 1],lazyadd[p]); //lazyadd[p * 2 + 1] +=
lazyadd[p];
                addto(tree[p * 2],lazyadd[p] * (cm - cl + 1)); //tree[p * 2] +=
lazyadd[p] * (cm - cl + 1);
                addto(tree[p * 2 + 1],lazyadd[p] * (cr - cm)); //tree[p * 2 + 1]
+= lazyadd[p] * (cr - cm);
                lazyadd[p] = 0;
            }

            if(lazyset[p]){ //区间直接修改
                lazyset[p * 2] = Get_Mod(lazyset[p]); //lazyset[p * 2] =
lazyset[p];
                lazyset[p * 2 + 1] = Get_Mod(lazyset[p]); //lazyset[p * 2 + 1] =
lazyset[p];
                tree[p * 2] = Get_Mod(lazyset[p] * (cm - cl + 1)); //tree[p * 2] =
lazyset[p] * (cm - cl + 1);
                tree[p * 2 + 1] = Get_Mod(lazyset[p] * (cr - cm)); //tree[p * 2 +
1] = lazyset[p] * (cr - cm);
                lazyset[p] = 0;
            }
        }
```

```
    }
    void range_mul(int l, int r, T val, int cl, int cr, int p) {
    // [l, r] 为查询区间，[cl, cr] 为当前节点包含的区间，p 为当前节点的编号
        if (l <= cl && cr <= r) {
            multo(lazyadd[p],val); //lazyadd[p] *= val;
            multo(tree[p],val); //tree[p] *= val;
            multo(lazymul[p],val); //lazymul[p] *= val;
            return;
        }
        int m = cl + (cr - cl) / 2;
        maintain(cl, cr, p);
        if (l <= m) range_mul(l, r, val, cl, m, p * 2);
        if (r > m) range_mul(l, r, val, m + 1, cr, p * 2 + 1);
        tree[p] = add(tree[p * 2],tree[p * 2 + 1]); //tree[p] = tree[p * 2] +
tree[p * 2 + 1];
    }
    void range_add(int l, int r, T val, int cl, int cr, int p) {
        if (l <= cl && cr <= r) {
            addto(lazyadd[p],val); //lazyadd[p] += val;
            addto(tree[p],(cr - cl + 1) * val); //tree[p] += (cr - cl + 1) * val;
            return;
        }
        int m = cl + (cr - cl) / 2;
        maintain(cl, cr, p);
        if (l <= m) range_add(l, r, val, cl, m, p * 2);
        if (r > m) range_add(l, r, val, m + 1, cr, p * 2 + 1);
        tree[p] = add(tree[p * 2],tree[p * 2 + 1]); //tree[p] = tree[p * 2] +
tree[p * 2 + 1];
    }
    void range_set(int l, int r, T val, int cl, int cr, int p) {
        if (l <= cl && cr <= r) {
            lazyset[p] = Get_Mod(val); //lazyset[p] = val;
            tree[p] = Get_Mod(val * (cr - cl + 1)); //tree[p] = val * (cr - cl +
1);
            return;
        }
        int m = cl + (cr - cl) / 2;
        maintain(cl, cr, p);
        if (l <= m) range_set(l, r, val, cl, m, p * 2);
        if (r > m) range_set(l, r, val, m + 1, cr, p * 2 + 1);
        tree[p] = add(tree[p * 2],tree[p * 2 + 1]); //tree[p] = tree[p * 2] +
tree[p * 2 + 1];
    }

    T range_sum(int l, int r, int cl, int cr, int p) {
        if (l <= cl && cr <= r) return tree[p];
        int m = cl + (cr - cl) / 2;
        T sum = 0;
        maintain(cl, cr, p);
        if (l <= m) //sum += range_sum(l, r, cl, m, p * 2);
            sum = add(sum,range_sum(l, r, cl, m, p * 2));
        if (r > m) //sum += range_sum(l, r, m + 1, cr, p * 2 + 1);
            sum = add(sum,range_sum(l, r, m + 1, cr, p * 2 + 1));
        return sum;
```

```
    }

    void build(int s, int t, int p) {
        if (s == t) {
            tree[p] = Get_Mod((*arr)[s]);
            return;
        }
        int m = s + (t - s) / 2;
        build(s, m, p * 2);
        build(m + 1, t, p * 2 + 1);
        tree[p] = add(tree[p * 2],tree[p * 2 + 1]); //tree[p] = tree[p * 2] +
tree[p * 2 + 1];
    }

public:
    explicit SegTree<T>(vector<T> v) {
        n = v.size();
        n4 = n * 4;
        tree = vector<T>(n4, 0);
        lazyadd = vector<T>(n4, 0);
        lazymul = vector<T>(n4, 1);
        lazyset = vector<T>(n4, 0);
        arr = &v;
        end = n - 1;
        root = 1;
        build(0, end, 1);
        arr = nullptr;
    }

    void show(int p, int depth = 0) {
        if (p > n4 || tree[p] == 0) return;
        show(p * 2, depth + 1);
        for (int i = 0; i < depth; ++i) putchar('\t');
        printf("%d:%d\n", tree[p], lazyadd[p]);
        show(p * 2 + 1, depth + 1);
    }

    T range_sum(int l, int r) { return range_sum(l, r, 0, end, root); }

    void range_mul(int l, int r, T val) { range_mul(l, r, val, 0, end, root); }
    void range_add(int l, int r, T val) { range_add(l, r, val, 0, end, root); }
    void range_set(int l, int r, T val) { range_set(l, r, val, 0, end, root); }
};
```

19. 最值线段树

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF = 0x7fffffffffffffff;

// 最值线段树 实现log^2n的区间增加、区间最值、区间查询
class MxSegTree{
    struct data{
        ll mx,mx2,cmx,tmx;
        ll mn,mn2,cmn,tmn;
        ll tad,sum;
        data(){mx=mx2=-INF;mn=mn2=INF;cmx=cmn=0;tmx=-INF;tmn=INF;tad=0;sum=0;}
    };
    vector<data> tree;
    vector<ll> *arr;
    long n,n5,root,end;

    void push_add(ll cl, ll cr, ll p,ll v){
        tree[p].sum += (cr - cl + 1) * v;
        tree[p].mx += v, tree[p].mn += v;
        if (tree[p].mx2 != -INF) tree[p].mx2 += v;
        if (tree[p].mn2 != INF) tree[p].mn2 += v;
        if (tree[p].tmx != -INF) tree[p].tmx += v;
        if (tree[p].tmn != INF) tree[p].tmn += v;
        tree[p].tad += v;
    }
    void push_min(ll p, ll tg){
        if (tree[p].mx <= tg) return;
        tree[p].sum += (tg * 1ll - tree[p].mx) * tree[p].cmx;
        if (tree[p].mn2 == tree[p].mx) tree[p].mn2 = tg;
        if (tree[p].mn == tree[p].mx) tree[p].mn = tg;
        if (tree[p].tmx > tg) tree[p].tmx = tg;
        tree[p].mx = tg, tree[p].tmn = tg;
    }
    void push_max(ll p, ll tg){
        if (tree[p].mn > tg) return;
        tree[p].sum += (tg * 1ll - tree[p].mn) * tree[p].cmn;
        if (tree[p].mx2 == tree[p].mn) tree[p].mx2 = tg;
        if (tree[p].mx == tree[p].mn) tree[p].mx = tg;
        if (tree[p].tmn < tg) tree[p].tmn = tg;
        tree[p].mn = tg, tree[p].tmx = tg;
    }
    void push_up(ll p) { //用子节点维护当前节点信息
        tree[p].sum = tree[p * 2].sum + tree[p * 2 + 1].sum;
        if (tree[p * 2].mx == tree[p * 2 + 1].mx) {
            tree[p].mx = tree[p * 2].mx, tree[p].cmx = tree[p * 2].cmx + tree[p *
2 + 1].cmx;
            tree[p].mx2 = max(tree[p * 2].mx2, tree[p * 2 + 1].mx2);
```

```
        } else if (tree[p * 2].mx > tree[p * 2 + 1].mx) {
            tree[p].mx = tree[p * 2].mx, tree[p].cmx = tree[p * 2].cmx;
            tree[p].mx2 = max(tree[p * 2].mx2, tree[p * 2 + 1].mx);
        } else {
            tree[p].mx = tree[p * 2 + 1].mx, tree[p].cmx = tree[p * 2 + 1].cmx;
            tree[p].mx2 = max(tree[p * 2].mx, tree[p * 2 + 1].mx2);
        }
        if (tree[p * 2].mn == tree[p * 2 + 1].mn) {
            tree[p].mn = tree[p * 2].mn, tree[p].cmn = tree[p * 2].cmn + tree[p *
2 + 1].cmn;
            tree[p].mn2 = min(tree[p * 2].mn2, tree[p * 2 + 1].mn2);
        } else if (tree[p * 2].mn < tree[p * 2 + 1].mn) {
            tree[p].mn = tree[p * 2].mn, tree[p].cmn = tree[p * 2].cmn;
            tree[p].mn2 = min(tree[p * 2].mn2, tree[p * 2 + 1].mn);
        } else {
            tree[p].mn = tree[p * 2 + 1].mn, tree[p].cmn = tree[p * 2 + 1].cmn;
            tree[p].mn2 = min(tree[p * 2].mn, tree[p * 2 + 1].mn2);
        }
    }
    void push_down(ll cl, ll cr, ll p) { //下放标记
        ll cm = cl + (cr - cl) / 2;
        if(tree[p].tad) push_add(cl, cm, p * 2, tree[p].tad), push_add(cm + 1, cr,
p * 2 + 1, tree[p].tad);
        if(tree[p].tmx != -INF) push_max(p * 2, tree[p].tmx), push_max(p * 2 + 1,
tree[p].tmx);
        if(tree[p].tmn != INF) push_min(p * 2, tree[p].tmn), push_min(p * 2 + 1,
tree[p].tmn);
        tree[p].tad = 0, tree[p].tmx = -INF, tree[p].tmn = INF;
    }

    void build(ll s, ll t, ll p) {
        tree[p].tmx = -INF, tree[p].tmn = INF;
        if (s == t) {
            tree[p].sum = tree[p].mx = tree[p].mn = (*arr)[s];
            tree[p].mx2 = -INF, tree[p].mn2 = INF;
            tree[p].cmx = tree[p].cmn = 1;
            return;
        }
        ll m = s + (t - s) / 2;
        build(s, m, p * 2);
        build(m + 1, t, p * 2 + 1);
        push_up(p);
    }

    void range_add(ll l, ll r, ll v, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return;
        if (l <= cl && cr <= r) return push_add(cl, cr, p, v);
        ll cm = cl + (cr - cl) / 2;
        push_down(cl, cr, p);
        if (l <= cm) range_add(l, r, v, cl, cm, p * 2);
        if (r > cm) range_add(l, r, v, cm + 1, cr, p * 2 + 1);
        push_up(p);
    }
    void range_min(ll l, ll r, ll v, ll cl, ll cr, ll p) {
```

```
        if(cl > r || cr < l || tree[p].mx <= v) return;
        if (l <= cl && cr <= r && tree[p].mx2 < v) return push_min(p, v);
        ll cm = cl + (cr - cl) / 2;
        push_down(cl, cr, p);
        if (l <= cm) range_min(l, r, v, cl, cm, p * 2);
        if (r > cm) range_min(l, r, v, cm + 1, cr, p * 2 + 1);
        push_up(p);
    }
    void range_max(ll l, ll r, ll v, ll cl, ll cr, ll p) {
        if(cl > r || cr < l || tree[p].mn >= v) return;
        if (l <= cl && cr <= r && tree[p].mn2 > v) return push_max(p, v);
        ll cm = cl + (cr - cl) / 2;
        push_down(cl, cr, p);
        if (l <= cm) range_max(l, r, v, cl, cm, p * 2);
        if (r > cm) range_max(l, r, v, cm + 1, cr, p * 2 + 1);
        push_up(p);
    }

    ll query_sum(ll l, ll r, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return 0;
        if (l <= cl && cr <= r) return tree[p].sum;
        ll cm = cl + (cr - cl) / 2;
        ll sum = 0;
        push_down(cl, cr, p);
        if (l <= cm) sum += query_sum(l, r, cl, cm, p * 2);
        if (r > cm) sum += query_sum(l, r, cm + 1, cr, p * 2 + 1);
        return sum;
    }
    ll query_max(ll l, ll r, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return -INF;
        if (l <= cl && cr <= r) return tree[p].mx;
        ll cm = cl + (cr - cl) / 2;
        ll mx = -INF;
        push_down(cl, cr, p);
        if (l <= cm) mx = max(mx, query_max(l, r, cl, cm, p * 2));
        if (r > cm) mx = max(mx, query_max(l, r, cm + 1, cr, p * 2 + 1));
        return mx;
    }
    ll query_min(ll l, ll r, ll cl, ll cr, ll p) {
        if(cl > r || cr < l) return INF;
        if (l <= cl && cr <= r) return tree[p].mn;
        ll cm = cl + (cr - cl) / 2;
        ll mn = INF;
        push_down(cl, cr, p);
        if (l <= cm) mn = min(mn, query_min(l, r, cl, cm, p * 2));
        if (r > cm) mn = min(mn, query_min(l, r, cm + 1, cr, p * 2 + 1));
        return mn;
    }

public:
    explicit MxSegTree(vector<ll> v) {
        n = v.size();
        n5 = n * 5;
        end = n - 1;
```

```
        root = 1;
        arr = &v;
        tree.resize(n5);
        build(0, end, 1);
        arr = nullptr;
    }
    void range_add(ll l, ll r, ll v) { range_add(l, r, v, 0, end, root); }
    void range_min(ll l, ll r, ll v) { range_min(l, r, v, 0, end, root); }
    void range_max(ll l, ll r, ll v) { range_max(l, r, v, 0, end, root); }
    ll query_sum(ll l, ll r) { return query_sum(l, r, 0, end, root); }
    ll query_max(ll l, ll r) { return query_max(l, r, 0, end, root); }
    ll query_min(ll l, ll r) { return query_min(l, r, 0, end, root); }
};
```

20. 数论分块

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

vector<ll> vr; //数论分块，vr存储\floor{x/i}相同的分块的右端点
void Sqrt_dec(ll x){
    vr.clear(); ll r=1;
    for(ll l=1;l<=x;l=vr.back()+1){
        r=x/(x/l);
        vr.emplace_back(r);
    }
}

vector<ll> pref; //函数f前缀和
ll Get_G(ll x){ //数论分块，求函数g(x)=\sum_{i=1}^{x} f(i)*\floor{x/i}
    ll res=0,r=1;
    for(ll l=1;l<=x;l=vr.back()+1){
        r=x/(x/l);
        res+=(pref[r]-pref[l-1])*(x/l)*(r-l+1);
    }
    return res;
}
```