# 基 本框架

#include <iostream> #include <cstdio> #include <algorithm> #include <cmath> #include <cstring> #include <string> #include <cstdlib> #include <sstream> #include <queue> using namespace std; typedef long long ll;

typedef unsigned int ull; typedef pair<ll,int> pil;

const ll ll\_INF = 0x3f3f3f3f3f3f3f3f; const int int\_INF = 0x3f3f3f3f; const double EPS = 1e-7;

#define R register

#define \_max(a,b) ((a)>(b)?(a):(b))

#define \_min(a,b) ((a)<(b)?(a):(b))

#define \_abs(a) ((a)> 0 ?(a):-(a))

#define \_swap(a,b) ((a)^=(b)^=(a)^=(b))

#define \_eql(x,y) (\_abs((x) - (y))<EPS)

const int MX = 1000010; int \_r() {

int x = 0, f = 1; char c = getchar();

while (c > '9' || c < '0') { if (c == '-')f = -1; c = getchar(); } while (c >= '0' && c <= '9')x = (x << 3) + (x << 1) + (c ^ 48), c =

getchar();

return f==-1?-x:x;

}

void Init() {

}

void solve() {

}

int main() {

//freopen("data.in", "r", stdin);

//freopen("data.out", "w", stdout); int t = \_r();

while (t--) {

Init();

solve();

}

return 0;

}

**S TL**

**数 组 / vector**

### **unique()**

去除相邻重复元素

n=unique(a,a+n)-a;//n为删除后元素个数

auto last=unique(v.begin(),v.end());//last为删除后的尾迭代器

v.erase(last, v.end());

**next\_permutation() & prev\_permutation()**

返回值为bool：是否存在下一个排列 

while(next\_permutation(v.begin(),v.end()){ ... }//遍历全排列

**upper\_bound( ) & lower\_bound()**

二分第一个 (upper) \ (lower) x 的值，在不存在对应值时都为右端点 

int pre=a[upper\_bound(a,a+n,x)-a];//启， 止， 值

**b itset**

**m ap / multi\_map**

基于红黑树，必须是 multi\_map 才允许重复的  值

**operator []**

在访问不存在的  时会新插入一个 并设为默认值（比如说 int 默认 0, string 默认 “”

）

**mp.upper\_bound() & mp.lower\_bound()**

返回以  排序的 upper\_bound 和 lower\_bound，在不存在对应值时都为右端点

map<int,string>::iterator prev=mp.upper\_bound(10);//值

**mp.count()**

返回满足  的个数 

**mp.find()**

返回满足  的迭代器位置，若不存在则为 mp.end()  整体上看效率比

*count()* 更优

**u norder\_map**

基于哈希表

**高精**

**手动扩栈**

#pragma comment(linker, "/STACK:10240000,10240000")

**无负数**

const int BASE = 10000; const int MX\_LEN = 1010; struct Bign {

ll num[MX\_LEN]; int len; Bign() {

memset(num, 0, sizeof(num));

len = 1;

}

Bign(const ll x) { \*this = x; } Bign(const string x) { \*this = x; } Bign(const Bign& x) {

memset(num, 0, sizeof(num)); len = x.len;

for (int i = 0; i < len; i++) num[i] = x.num[i];

}

void clean() {

while (num[len - 1] == 0 && len != 1) len--;

}

Bign operator = (const ll x) { stringstream ss;

ss << x; string temp; ss >> temp;

return \*this = temp;

}

Bign operator = (const string x) { len = 0;

memset(num, 0, sizeof(num)); ll temp = 0;

ll base = 1;

for (int i = x.length() - 1; i >= 0; i--) {

temp += (x[i] - '0') \* base; base \*= 10;

if (base == BASE) { num[len++] = temp; temp = 0;

base = 1;

}

}

num[len++] = temp; clean();

return \*this;

}

Bign operator + (const Bign& b) const { Bign c;

c.len = \_max(len, b.len) + 1; for (int i = 0; i < c.len; i++) {

c.num[i] += num[i] + b.num[i]; c.num[i + 1] += c.num[i] / BASE; c.num[i] %= BASE;

}

c.clean(); return c;

}

Bign operator - (const Bign& b) const {//a-b保证a>b Bign c;

c.len = \_max(len, b.len);

for (int i = 0; i < c.len; ++i) { c.num[i] += num[i] - b.num[i]; if (c.num[i] < 0) {

c.num[i] += BASE; c.num[i + 1] -= 1;

}

}

c.clean(); return c;

}

Bign operator << (const int& num) const { Bign c = \*this;

c.len += 10;

for (R int i = 0; i < c.len; ++i) { c.num[i] <<= num;

if (i && c.num[i - 1] >= BASE)

++c.num[i], c.num[i - 1] -= BASE;

}

c.clean(); return c;

}

Bign operator >> (const int& num) const { Bign c = \*this;

for (R int i = len - 1; i >= 0; --i) {

if ((c.num[i] & 1) && i) c.num[i - 1] += BASE;

c.num[i] >>= num;

}

c.clean(); return c;

}

Bign operator \* (const Bign& b) const { Bign c;

c.len = len + b.len + 5;

for (int i = 0; i < c.len; ++i) { for (int j = 0; j < b.len; ++j) {

c.num[i + j] += num[i] \* b.num[j]; c.num[i + j + 1] += c.num[i + j] / BASE; c.num[i + j] %= BASE;

}

}

c.clean(); return c;

}

Bign operator / (const ll& b) const { //大数除以long long Bign c;

c.len = len; ll rest = 0;

for (int i = len - 1; i >= 0; --i) { rest = rest \* BASE + num[i]; c.num[i] = rest / b;

rest %= b;

}

c.clean(); return c;

}

Bign operator / (const Bign& b) const { Bign c, rest, now, \_base;

now = \*this; rest = b;

\_base = 1;

while (now >= rest) { rest = rest << 1;

\_base = \_base << 1;

}

while (\_base.len > 1 || \_base.num[0]) { if (now >= rest) {

now -= rest; c += \_base;

}

rest = rest >> 1;

\_base = \_base >> 1;

}

c.clean(); return c;

}

Bign operator % (const ll& b) const { return (\*this) - ((\*this) / b) \* b;

}

Bign operator % (const Bign& b) const { return (\*this) - ((\*this) / b) \* b;

}

Bign operator += (const Bign& b) { return (\*this) = (\*this) + b;

}

Bign operator -= (const Bign& b) { return (\*this) = (\*this) - b;

}

Bign operator \*= (const Bign& b) { return (\*this) = (\*this) \* b;

}

Bign operator /= (const ll& b) { return (\*this) = (\*this) / b;

}

Bign operator /= (const Bign& b) { return (\*this) = (\*this) / b;

}

Bign operator %= (const ll& b) { return (\*this) = (\*this) % b;

}

Bign operator %= (const Bign& b) { return (\*this) = (\*this) % b;

}

bool operator < (const Bign& b) const { if (len == b.len) {

for (int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return num[i] < b.num[i];

}

return 0;

}

return len < b.len;

}

bool operator > (const Bign& b) const { if (len == b.len) {

for (int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return num[i] > b.num[i];

}

return 0;

}

return len > b.len;

}

bool operator == (const Bign& b) const { if (len == b.len) {

for (int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return 0;

}

return 1;

}

return 0;

}

bool operator != (const Bign& b) const { return !((\*this) == b);

}

bool operator <= (const Bign& b) const {

return !((\*this) > b);

}

bool operator >= (const Bign& b) const { return !((\*this) < b);

}

friend ostream& operator << (ostream& out, const Bign& x) { out << x.num[x.len - 1];

for (int i = x.len - 2; i >= 0; i--) { int t = BASE / 10;

while (x.num[i] < t && t>1) { out << 0;

t /= 10;

}

out << x.num[i];

}

return out;

}

friend istream& operator >> (istream& in, Bign& x) { string temp;

in >> temp; x = temp; return in;

}

};

**有 负数**

struct Bign {

ll num[MX\_LEN]; int len;

int neg; Bign() {

memset(num, 0, sizeof(num));

len = 1;

neg = 0;

}

Bign(const ll x) { \*this = x; } Bign(const string x) { \*this = x; } Bign(const Bign& x) {

memset(num, 0, sizeof(num)); len = x.len;

neg = x.neg;

for (R int i = 0; i < len; ++i) num[i] = x.num[i];

}

void clean() {

while (num[len - 1] == 0 && len != 1)

--len;

if (len == 1 && num[0] == 0) neg = 0;

}

Bign operator = (const ll& x) { stringstream ss;

ss << x; string temp; ss >> temp;

return \*this = temp;

}

Bign operator = (const string& x) { len = 0;

memset(num, 0, sizeof(num)); if (x[0] == '-')

neg = 1; else

neg = 0; ll temp = 0; ll base = 1;

for (R int i = x.length() - 1; i >= neg; --i) {

temp += (x[i] - '0') \* base; base \*= 10;

if (base == BASE) { num[len++] = temp; temp = 0;

base = 1;

}

}

num[len++] = temp; clean();

return \*this;

}

Bign operator + (const Bign& b) const { if (neg ^ b.neg) {

if (neg)

return b - (-\*this);

else

}

return (\*this) - (-b);

else {

Bign c; if (neg)

c.neg = 1;

c.len = \_max(len, b.len) + 1;

for (R int i = 0; i < c.len; ++i) { c.num[i] += num[i] + b.num[i]; c.num[i + 1] += c.num[i] / BASE; c.num[i] %= BASE;

}

c.clean(); return c;

}

}

Bign operator - (const Bign& b) const { if (neg ^ b.neg) {

if (neg)

return -((-\*this) + b);

else

}

return (\*this) + (-b);

else {

Bign c; if (neg)

c.neg = 1;

if (abs(\*this) < abs(b)) { c.neg ^= 1;

c.len = \_max(len, b.len);

for (R int i = 0; i < c.len; ++i) { c.num[i] += b.num[i] - num[i]; if (c.num[i] < 0) {

c.num[i] += BASE; c.num[i + 1] -= 1;

}

}

c.clean(); return c;

}

else {

c.len = \_max(len, b.len);

for (R int i = 0; i < c.len; ++i) { c.num[i] += num[i] - b.num[i]; if (c.num[i] < 0) {

c.num[i] += BASE; c.num[i + 1] -= 1;

}

}

c.clean(); return c;

}

}

}

Bign operator - () const { Bign c = \*this;

c.neg ^= 1; return c;

}

Bign operator \* (const Bign& b) const { Bign c;

c.len = len + b.len + 1;

for (R int i = 0; i < len; ++i) {

for (R int j = 0; j < b.len; ++j) { c.num[i + j] += num[i] \* b.num[j];

}

}

for (R int i = 0; i < c.len; ++i) { c.num[i + 1] += c.num[i] / BASE;

c.num[i] %= BASE;

}

if (neg ^ b.neg) c.neg = 1;

c.clean(); return c;

}

Bign operator / (const ll& b) const { Bign c;

c.len = len; ll rest = 0;

for (R int i = len - 1; i >= 0; --i) { rest = rest \* BASE + num[i]; c.num[i] = rest / b;

rest %= b;

}

if (neg ^ (b < 0)) c.neg = 1;

c.clean(); return c;

}

Bign operator / (const Bign& b) const { Bign c, rest, now, \_base;

now = abs(\*this); rest = abs(b);

\_base = 1;

while (now >= rest) { rest <<= 1;

\_base <<= 1;

}

while (\_base.len > 1 || \_base.num[0]) { if (now >= rest) {

now -= rest; c += \_base;

}

rest >>= 1;

\_base >>= 1;

}

if (neg ^ b.neg) c.neg = 1;

c.clean(); return c;

}

Bign operator << (const int& num) const { Bign c = \*this;

c.len += 10;

for (R int i = 0; i < c.len; ++i) { c.num[i] <<= num;

if (i && c.num[i - 1] >= BASE)

++c.num[i], c.num[i - 1] -= BASE;

}

c.clean();

return c;

}

Bign operator >> (const int& num) const { Bign c = \*this;

for (R int i = len - 1; i >= 0; --i) {

if ((c.num[i] & 1) && i) c.num[i - 1] += BASE;

c.num[i] >>= num;

}

c.clean(); return c;

}

Bign operator % (const ll& b) const { return (\*this) - ((\*this) / b) \* b;

}

Bign operator % (const Bign& b) const { return (\*this) - ((\*this) / b) \* b;

}

Bign operator += (const Bign& b) { return (\*this) = (\*this) + b;

}

Bign operator -= (const Bign& b) { return (\*this) = (\*this) - b;

}

Bign operator \*= (const Bign& b) { return (\*this) = (\*this) \* b;

}

Bign operator /= (const ll& b) { return (\*this) = (\*this) / b;

}

Bign operator /= (const Bign& b) { return (\*this) = (\*this) / b;

}

Bign operator %= (const ll& b) { return (\*this) = (\*this) % b;

}

Bign operator %= (const Bign& b) { return (\*this) = (\*this) % b;

}

Bign operator <<= (const int& num) { return (\*this) = (\*this) << num;

}

Bign operator >>= (const int& num) { return (\*this) = (\*this) >> num;

}

bool operator < (const Bign& b) const { if (neg && !b.neg)

return 1;

if (!neg && b.neg) return 0;

if (neg) {

if (len == b.len) {

for (R int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return num[i] > b.num[i];

}

return 0;

}

return len > b.len;

}

else {

if (len == b.len) {

for (R int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return num[i] < b.num[i];

}

return 0;

}

return len < b.len;

}

}

bool operator > (const Bign& b) const { if (neg && !b.neg)

return 0;

if (!neg && b.neg) return 1;

if (neg) {

if (len == b.len) {

for (R int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return num[i] < b.num[i];

}

return 0;

}

return len < b.len;

}

else {

if (len == b.len) {

for (R int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return num[i] > b.num[i];

}

return 0;

}

return len > b.len;

}

}

bool operator == (const Bign& b) const { if (neg != b.neg)

return 0;

if (len == b.len) {

for (R int i = len - 1; i >= 0; --i) {

if (num[i] != b.num[i]) return 0;

}

return 1;

}

return 0;

}

bool operator != (const Bign& b) const { return !((\*this) == b);

}

bool operator <= (const Bign& b) const { return !((\*this) > b);

}

bool operator >= (const Bign& b) const { return !((\*this) < b);

}

friend Bign abs(const Bign& x) { Bign c = x;

c.neg = 0; return c;

}

friend ostream& operator << (ostream& out, const Bign& x) { if (x.neg)

out << '-';

out << x.num[x.len - 1];

for (R int i = x.len - 2; i >= 0; --i) { int t = BASE / 10;

while (x.num[i] < t && t>1) { out << 0;

t /= 10;

}

out << x.num[i];

}

return out;

}

friend istream& operator >> (istream& in, Bign& x) { string temp;

in >> temp; x = temp; return in;

}

};

# 数 据结构

## 单 调队列

struct Node {

int num; int id;

}a[MX];

deque<Node> q; void solve() {

Node now;

for (int i = 1; i <= n; i++) { if (q.empty()) {

q.push\_back(a[i]);

}

else {

now = q.back();

while (now.num > a[i].num) { q.pop\_back();

if (q.empty())break; now = q.back();

}

q.push\_back(a[i]); now = q.front();

if (i - now.id >= k) q.pop\_front();

}

if (i >= k)

cout<<q.front().num<<" ";//当前区间最大为队首

}

}

**S T表**

ll ST[MX][30] = { 0 };

int Log2[MX] = { 0 }; void creat\_ST() {

Log2[1] = 0, Log2[2] = 1;

for (int i = 3; i <= n; i++) Log2[i] = Log2[i >> 1] + 1;

for (int i = 1; i <= n; i++) ST[i][0] = a[i];

for (int i = 1; i < 25; i++) {

for (int j = 1; j <= n - (1 << (i-1)) + 1; j++) {

ST[j][i] = \_max(ST[j][i - 1], ST[j + (1 << (i - 1))][i -

1]);//RMQ

}

}

}

ll query\_ST(int l, int r) { int k = Log2[r - l + 1];

return gcd(ST[l][k], ST[r - (1 << k) + 1][k]);

}

## 树 状数组

int low\_bit(int x) { return x & (-x);

}

void BIT\_add(int i, ll val) { while (i <= n) {

BIT[i] += val;

i += low\_bit(i);

}

}

ll BIT\_sum(int i) { ll s = 0;

while (i > 0) { s += BIT[i];

i -= low\_bit(i);

}

return s;

}

**线 段树**

struct Node {

int l, r; ll val; ll add;

}tr[MX << 2] = { 0 };

ll a[MX] = { 0 };

void pushdown(int i) { if (tr[i].add == 0)

return;

if (tr[i].l == tr[i].r) { tr[i].add = 0; return;

}

tr[i << 1].val += tr[i].add \* (tr[i << 1].r - tr[i << 1].l + 1); tr[i << 1].add += tr[i].add;

tr[i << 1 | 1].val += tr[i].add \* (tr[i << 1 | 1].r - tr[i << 1 | 1].l

+ 1);

tr[i << 1 | 1].add += tr[i].add; tr[i].add = 0;

}

void pushup(int i) {

tr[i].val = tr[i << 1].val + tr[i << 1 | 1].val;

}

void build(int i, int l, int r) {

tr[i].l = l, tr[i].r = r, tr[i].val = tr[i].add = 0; if (l == r) {

tr[i].val = a[l]; return;

}

build(i << 1, l, (l + r) >> 1);

build(i << 1 | 1, ((l + r) >> 1) + 1, r); pushup(i);

}

ll query(int i, int l, int r) {

if (l > tr[i].r || r < tr[i].l) return 0;

if (l <= tr[i].l && r >= tr[i].r) return tr[i].val;

pushdown(i);

return (query(i << 1, l, r) + query(i << 1 | 1, l, r));

}

void update(int i, int l, int r, ll val) { if (r<tr[i].l || l>tr[i].r)

return;

if (l <= tr[i].l && r >= tr[i].r) {

tr[i].val += (tr[i].r - tr[i].l + 1) \* val; tr[i].add += val;

return;

}

pushdown(i);

update(i << 1, l, r, val); update(i << 1 | 1, l, r, val); pushup(i);

return;

}

**数 学**

**快 速幂**

计算

ll \_pow(ll base, ll n){ ll pow\_res = 1; while (n){

if (n & 1)

pow\_res = pow\_res \* base % MOD; base = base \* base % MOD;

n >>= 1;

}

return pow\_res;

}

**龟速乘**

用于 时,乘法可能爆出ll的问题,通常与快速幂搭配使用

ll \_mul(ll x, ll y) { ll res = 0; while (y){

if (y & 1)

res = (res + x) % MOD; x = (x + x) % MOD;

y >>= 1;

}

return res;

}

**快速乘**

另一种用于处理爆出 ll 的方法

ll \_mul(ll x, ll y) { a %= MOD, b %= MOD;

return (a \* b - (ll)(((long double)a \* b + 0.5) / MOD) \* MOD) % MOD;

}

**矩阵快速幂**

计算递推式 . 手推前 项后使用矩阵



快速幂求解. 求解过程与转移矩阵构造如下:(图中所使用的矩阵都是 阶方阵, 和行\列为求 的部分, 行为求 的部分,不需要的话可以对应减少阶数)





const int MAT\_SIZ = 3; struct mat {

ll num[MAT\_SIZ][MAT\_SIZ];

mat() {

memset(num, 0, sizeof(num));

}

void reset() {

for (int i = 0; i < MAT\_SIZ; i++) num[i][i] = 1;

}

mat operator \* (const mat& b) { mat c;

for (int i = 0; i < MAT\_SIZ; i++) { for (int j = 0; j < MAT\_SIZ; j++) {

for (int k = 0; k < MAT\_SIZ; k++) {

c.num[i][j] += (num[i][k] \* b.num[k][j]) % MOD; c.num[i][j] %= MOD;

}

}

}

return c;

}

};

mat mat\_pow(mat a, ll x) { mat res, base; res.reset(); base = a; while (x) {

if (x & 1)

res = res \* base; base = base \* base; x >>= 1;

}

return res;

}

**质 因数**

**质数判断**

ll prime[6] = {2, 3, 5, 233, 331};

bool Miller\_Rabin(ll p) { if(p < 2) return 0;

if(p != 2 && p % 2 == 0) return 0; ll s = p - 1;

while(! (s & 1)) s >>= 1; for(int i = 0; i < 5; ++i) {

if(p == prime[i]) return 1;

ll t = s, m = \_pow(prime[i], s, p); while(t != p - 1 && m != 1 && m != p - 1) {

m = \_mul(m, m, p); t <<= 1;

}

if(m != p - 1 && !(t & 1)) return 0;

}

return 1;

}

**质数筛法**

void EularSieve(int range) {

for (int i = 2; i < range; i++) { if (not\_pri[i] == 0)

pri[++pri[0]] = i;

for (int j = 1; j <= pri[0]; j++) { if (pri[j] \* i > range)break; not\_pri[pri[j] \* i] = 1;

if (i % pri[j] == 0)break;

}

}

}

**质因数分解**

p[i]:质因数表

a[i]:对应质因数的次数

void PrimeDevide(int num) {

for (int i = 1; i < pri[0] && pri[i] \* pri[i] <= num; i++) { if (num % pri[i] == 0) {

p[++p[0]] = pri[i];

while (num % pri[i] == 0) { a[p[0]]++;

num /= pri[i];

}

}

}

if (num != 1) {

p[++p[0]] = num;

a[p[0]] = 1;

}

}

**g cd与lcm**

**普通欧几里得**

ll gcd(ll a, ll b) {

return b == 0 ? a : gcd(b, a % b);

}

**拓展欧几里得**

求解 的一组可行解并返回

ll exgcd(ll a, ll b, ll& x, ll& y) { if (b == 0) {

x = 1;

y = 0;

return a;

}

ll t = x; x = y;

y = t - a / b \* y;

return exgcd(b, a % b, x, y);

}

**逆 元**

**拓欧求法**

ll get\_inv(ll a) { ll x, y, d;

d = exgcd(a, MOD, x, y);

return d == 1 ? (x % MOD + MOD) % MOD : -1;

}

**快速幂求法**

 需要为质数

ll get\_inv(ll a){

return \_pow(a, MOD - 2);

}

**递推求法**

**所有数的逆元**

void get\_inv(int range) { inv[1] = 1;

for (int i = 2; i <= range; i++)

inv[i] = MOD - ((MOD / i) \* inv[MOD % i]) % MOD;

}

**任意 个 数 的 逆 元** a[i]:原数组下标从1开始s[i]:前缀积

s\_inv[i]:前缀积的逆元

void get\_set\_inv(int range) { s[0] = 1;

for (int i = 1; i <= range; i++) s[i] = s[i - 1] \* a[i] % MOD;

s\_inv[n] = \_pow(s[n], MOD - 2); for (int i = range; i >= 1; i--)

s\_inv[i - 1] = s\_inv[i] \* a[i] % MOD; for (int i = 1; i <= range; i++)

inv[i] = s\_inv[i] \* s[i - 1] % MOD;

}

**阶乘逆元**

void get\_fac\_inv(int range) { fac[0] = 1;

for (int i = 1; i <= range; i++) fac[i] = (fac[i - 1] \* i) % MOD;

fac\_inv[range] = \_pow(fac[range], MOD - 2); for (int i = range - 1; i >= 0; i--)

fac\_inv[i] = (fac\_inv[i + 1] \* (i + 1)) % MOD;

}

也可以顺势计算普通逆元

void get\_fac\_inv(int range) { fac[0] = 1;

for (int i = 1; i <= range; i++) fac[i] = (fac[i - 1] \* i) % MOD;

inv[1] = 1;

for (int i = 2; i <= range; i++)

inv[i] = MOD - ((MOD / i) \* inv[MOD % i]) % MOD; fac\_inv[0] = fac\_inv[1] = 1;

for (int i = 2; i <= range; i++)

fac\_inv[i] = (fac\_inv[i - 1] \* inv[i]) % MOD;

}

**组 合数**

计算 和 ，需要先进行阶乘逆元的计算

ll \_A(int n, int m) { if (n < m || m < 0)

return 0;

return fac[n] \* fac\_inv[n - m] % MOD;

}

ll \_C(int n, int m) { if (n < m || m < 0)

return 0;

return (fac[n] \* fac\_inv[m] % MOD) \* fac\_inv[n - m] % MOD;

}

**计 算几何**

**基 本框架**

const double EPS = 1e-8;

#define \_eql(x,y) (\_abs((x) - (y))<EPS) #define Node Vector2

struct Vector2 { double x, y; double get\_len() {

return sqrt(x \* x + y \* y);

}

double operator \* (const Vector2 &b )const {//点乘

return x \* b.x + y \* b.y;

}

double operator % (const Vector2& b)const {//叉乘

return x \* b.y - y \* b.x;

}

};

**面 积**

**三角形面积**

double get\_S(Vector2 a, Vector2 b) { return \_abs(a % b);

}

**多边形面积**

double get\_S() {

double res = poi[0] % poi[n - 1]; for (int i = 1; i < n; i++) {

res += poi[i] % poi[i - 1] ;

}

return \_abs(res / 2);

}

**角 度**

**字 符串**

**h ash**

**乘法哈希**

ull hash(string x){ ull res=0;

int hash\_base=33;

//int hash\_mod=402653189; for(int i=0;i<x.length();i++){

res=res\*base+x[i];

//res%=hash\_mod;

}

return hash;

}

**位运算哈希**

ull hash(string x){ ull res=0;

for(int i=0;i<x.length();i++){ res=(res<<4)^(res>>28)^x[i];

}

return res;

}

**FNV哈希**

int: res=2166136261 , FNV\_prime=16777619

ull: res=14695981039346656037 , FNV\_prime=1099511628211

ull hash(string x){ ull res=2166136261;

int FNV\_prime=16777619; for(int i=0;i<x.length();i++){

hash^=x[i]; hash\*=FNV\_prime;

}

return hash;

}

**字 典树**

struct Node {

bool end; int son[26];

Node() {

end = 0;

for (int i = 0; i < 26; i++) son[i] = 0;

}

}tr[50010];

int tr\_cnt=0;

void insert(string& x) {

int u=0, v, lenx = x.length(); for (int i = 0; i < lenx; i++) {

v = x[i] - 'a';

if (!tr[u].son[v]) tr[u].son[v] = ++tr\_cnt;

u = tr[u].son[v];

}

tr[u].end = 1;

}

bool check(string& x) { int u = 0, v;

for (int i = 0; i < x.length(); i++) { v = x[i] - 'a';

if (!tr[u].son[v]) return 0;

u = tr[u].son[v];

}

return tr[u].end;

}

**K MP**

int KMP\_next[MX] = { 0 }; void get\_next(string& b) {

ll l = -1, r = 0; KMP\_next[0] = -1;

while (r < b.length()) {

if (l == -1 || b[l] == b[r]) KMP\_next[++r] = ++l;

else

l = KMP\_next[l];

}

}

void KMP(string& a, string& b) { ll l = 0, r = 0, cnt = 0; while (r < a.length()) {

if (l == -1 || a[r] == b[l]) l++, r++;

else

l = KMP\_next[l];

if (l == b.length()) {//found cnt++;

l = KMP\_next[l];

}

}

}

**M anacher**

int p[MX<<1];

int Manacher(string s) {

int pos = 0, r = 0, len = s.length(); string new\_s = "##";

for (int i = 0; i < len; i++) new\_s += s[i], new\_s += '#';

len = new\_s.length();

for (int i = 0; i < len; i++) { if (i < r)

p[i] = \_min(p[(pos << 1) - i], p[pos] + pos - i); else

p[i] = 1;

while (i - p[i] >= 1 && i + p[i] < len && new\_s[i - p[i]] == new\_s[i + p[i]])

++p[i];

if (i + p[i] > r) { pos = i;

r = i + p[i];

}

}

int max\_len = 0;

for (int i = 1; i <= len; ++i)

max\_len = \_max(max\_len, p[i] - 1); return max\_len;

}

**图 论**

**基 本框架**

struct Node {

ll val; int v;

int \_next;

}e[MX<<1];

int head[MX], cnt = 0;

void addedge(int u, int v, ll val) { e[++cnt].v = v;

e[cnt].val = val; e[cnt].\_next = head[u]; head[u] = cnt;

}

**最 短路**

**Dijkstra + 堆优化**

typedef pair<ll, int> pil;

priority\_queue<pil, vector<pil>, greater<pil> > q; int vis[MX] = { 0 };

void dijkstra(int ) { q.push(make\_pair(0, s)); dis[s] = 0;

int u;

while (!q.empty()) {

pil a = q.top(); q.pop(); u = a.second;

if (vis[u])continue; vis[u] = 1;

for (int i = head[u]; i; i = e[i].\_next) {

if (dis[u] + e[i].val < dis[e[i].v]) {

dis[e[i].v] = dis[u] + e[i].val;

q.push(make\_pair(dis[e[i].v], e[i].v));

}

}

}

}

**SPFA**

**SPFA + SLF**

deque<int> q;

int inq[MX], dis[MX], len[MX]; int spfa\_SLF(int u) {

while (!q.empty())q.pop\_back(); q.push\_back(u);

while (!q.empty()) {

u = q.front(); q.pop\_front(); inq[u] = 0;

for (int i = head[u]; i; i = e[i].\_next) { int v = e[i].v;

if (dis[v] > dis[u] + e[i].val) {

dis[v] = dis[u] + e[i].val;

//判定负环（ 不保证出现时用 e.g.差分约束）

len[v] = len[u] + 1;

if (len[v] > n) { flag = 1;

return 1;

}

if (!inq[v]) { inq[v] = 1;

if (q.empty() || dis[v] < dis[q.front()]) q.push\_front(v);

else

q.push\_back(v);

}

}

}

}

return 0;

}

**DFS\_SPFA**

int DFS\_SPFA(int u, int fa) { inq[u] = 1;

int v;

for (int i = head[u]; i; i = e[i].\_next) { v = e[i].v;

if (v == fa)//spfa前预处理自环边权为负

continue;

if (dis[v] > dis[u] + e[i].val) { if (inq[v]) {

return 1;

}

dis[v] = dis[u] + e[i].val; if (DFS\_SPFA(v, u))

return 1;

}

}

inq[u] = 0;

return 0;

}

**差分约束**

在使用最短路(找最小上界)时表示 ，使用最长路(找最大



下界)时表示 ,以最短路为例,常用逻辑关系转换：( ; 号节点为超级原点)



|  |  |  |  |
| --- | --- | --- | --- |
| **文字表达** | **加边方式** | **文字表达** | **加边方式** |
|  |  |  |  |
|  |  |  |  |
|  | , |  | , |

剩下的就是SPFA+SLF判定负环是否存在解并计算上下界

**Floyd**

void floyd() {

for (int k = 1; k <= n; k++) { for (int i = 1; i <= n; i++) {

if (dis[i][k] ^ ll\_INF) {

for (int j = 1; j <= i; j++) { if (dis[k][j] ^ ll\_INF) {

dis[i][j] = \_min(dis[i][j], dis[i][k]+dis[k][j]); dis[j][i]=dis[i][j];

}

}

}

}

}

}

**K短路(A\*)**

struct Ori {

int u, v, val;

}ori[100010];

struct Node {

int v, val; int next;

}e[100010<<1];

int head[MX], vis[MX], cnt = 0; int n, m, S, T, K;

void addedge(int u, int v, int val) { e[++cnt].v = v, e[cnt].val = val; e[cnt].next = head[u];

head[u] = cnt;

}

ll pred[MX] = { 0 }; priority\_queue<pil,vector<pil>,greater<pil> > q; void dijkstra() {

for (int i = 1; i <= n; i++) { pred[i] = INF;

vis[i] = 0;

}

pred[T] = 0; q.push(make\_pair(0, T));

while (!q.empty()) {

pil now = q.top(); q.pop(); int v, u = now.second;

if (vis[u])

continue; vis[u] = 1;

for (int i = head[u]; i; i = e[i].next) { v = e[i].v;

if (pred[v] > pred[u] + e[i].val) {

pred[v] = pred[u] + e[i].val;

q.push(make\_pair(pred[v], v));

}

}

}

}

int cc; ll ans;

struct Road {

ll val; int u;

bool operator < (const Road& a) const { return val+pred[u] > a.val+ pred[a.u];

}

};

priority\_queue<Road> q2; int AStar() {

while (!q2.empty())q2.pop(); Road now, nex;

now.u = S; now.val = 0; q2.push(now); int v, check;

while (!q2.empty()) {

now = q2.top(); q2.pop(); if (vis[now.u] > K)

continue; vis[now.u]++;

if (now.u == T && vis[now.u] == K) { ans = now.val;

return 1;

}

for (int i = head[now.u]; i; i = e[i].next) {

v = e[i].v; nex.u = v;

nex.val = now.val + e[i].val; q2.push(nex);

}

}

return 0;

}

void Init() {

n = \_r(), m = \_r();

for (int i = 0; i < m; i++) {

ori[i].u = \_r(), ori[i].v = \_r(), ori[i].val = \_r();

addedge(ori[i].v, ori[i].u, ori[i].val);

}

S = \_r(), T = \_r(), K = \_r(); if (S == T)

K++;

}

void solve() {

//反图

dijkstra();

//重建

cnt = 0;

for (int i = 1; i <= n; i++) vis[i] = head[i] = 0;

for (int i = 0; i < m; i++)

addedge(ori[i].u, ori[i].v, ori[i].val);

if (AStar())

printf("%d\n", ans); else

printf("-1\n");

}

## 最 小环

void floyd() {

ll min\_loop = ll\_INF;

for (int k = 1; k <= n; k++) { for (int i = 1; i < k; i++) {

if (e[i][k] ^ ll\_INF) {

for (int j = i + 1; j < k; j++) {

if (e[k][j] ^ ll\_INF && dis[j][i] ^ ll\_INF) { min\_loop = \_min(min\_loop, e[i][k] + e[k][j] +

dis[j][i]);

}

}

}

}

for (int i = 1; i <= n; i++) { if (dis[i][k] ^ ll\_INF) {

for (int j = 1; j <= i; j++) { if (dis[k][j] ^ ll\_INF) {

dis[i][j] = \_min(dis[i][j], dis[i][k]+dis[k][j]); dis[j][i]=dis[i][j];

}

}

}

}

}

}

**强 连通分量**

### **Tarjan**

int dfn[MX], low[MX],col[MX],col\_cnt, DFS\_clock; int \_stack[MX], ins[MX];

void tarjan(int u) {

\_stack[++\_stack[0]] = u; ins[u] = 1;

dfn[u] = low[u] = ++DFS\_clock;

for (int i = head[u]; i; i = e[i].\_next) { int v = e[i].v;

if (!dfn[v]) { tarjan(v);

low[u] = \_min(low[u], low[v]);

}

else if (ins[v])

low[u] = \_min(low[u], dfn[v]);

}

if (low[u] == dfn[u]) { col\_cnt++;

do {

col[\_stack[\_stack[0]]] = col\_cnt; ins[\_stack[\_stack[0]]] = 0;

} while (\_stack[\_stack[0]--] != u);

}

}

**割点与桥**

注意都是**无向图**概念

**割点**

int dfn[MX], low[MX], DFS\_clock = 0; int is\_cut[MX];

void tarjan(int u, int fa, int root) { dfn[u] = low[u] = ++DFS\_clock;

int now\_size = 0;

for (int i = head[u]; i; i = e[i].\_next) { int v = e[i].v;

if (!dfn[v]) { tarjan(v, u, root);

low[u] = \_min(low[u], low[v]);

if (low[v] >= dfn[u]) {//割点的判定

now\_size++;

if (u != root || now\_size > 1) { is\_cut[u] = 1;

}

}

}

else if (v != fa)

low[u] = \_min(low[u], dfn[v]);

}

}

**桥**

int is\_bridge[MX];

void tarjan(int u, int edge\_id){ dfn[u] = low[u] = ++DFS\_clock;

for (int i = head[x]; i; i = e[i].\_next) { int v = e[i].v;

if (!dfn[v]) { tarjan(v, i);

low[x] = \_min(low[x], low[v]); if (low[v] > dfn[x])//桥的判定

bridge[i] = bridge[i ^ 1] = 1;

b

}

else if (i != (edge\_id ^ 1)) low[x] = \_min(low[x], dfn[v]);

}

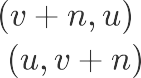
}

**2-SAT**



表示选取 后必选取 ，常用逻辑关系转换：





|  |  |  |  |
| --- | --- | --- | --- |
| **文字表达** | **加边方式** | **文字表达** | **加边方式** |
| 取 和 | , | 必不取 |  |
| 取 或 （ |  ） | , | 必取 |  |
| 取 或 （ ^  ） | , ，  , |  |  |

void t\_SAT() {

for (int i = 1; i <= (n << 1); i++) { if (dfn[i] == 0) {

tarjan(i);

}

}

for (int i = 1; i <= n; i++) { if (col[i] == col[i + n]) {

printf("IMPOSSIBLE\n"); return;

}

}

printf("POSSIBLE\n");

for (int i = 1; i <= n; i++) {

if (col[i] < col[i + n]) //拓扑逆序选用

printf("1 "); else

printf("0 ");

}

printf("\n");

}

**生 成树**

**并查集**

int \_fa[MX] = { 0 }; int \_find(int x) {

return x == \_fa[x] ? x : \_fa[x] = \_find(\_fa[x]);

}

void \_union(int x, int y) {

int xx = \_find(x), yy = \_find(y); if (xx == yy)

return; if (xx > yy)

\_swap(xx, yy);

\_fa[yy] = xx;

}

**最小生成树**

bool cmp(Node a, Node b) { return a.val < b.val;

}

ll kruskal() { ll MST = 0;

for (int i = 1; i <= n; i++)

\_fa[i] = i; sort(ori, ori + m, cmp); int u, v, ok = 1;

for (int i = 0; i < m; i++) { u = ori[i].u; v = ori[i].v; if (\_find(u) == \_find(v))

continue;

\_union(u, v);

MST += ori[i].val; if (++ok == n)

return MST;

}

return -1;

}

**次小生成树**

int n, m, cc = 0; struct Node {

int u, v, val; bool use;

}ori[MX]; struct Edge {

int v, \_next; ll val;

}e[MX];

int head[MX], cnt = 0;

void addedge(int u, int v, ll val) { e[++cnt].v = v;

e[cnt].val = val; e[cnt].\_next = head[u]; head[u] = cnt;

}

int \_fa[MX] = { 0 }; int \_find(int x) {

return x == \_fa[x] ? x : \_fa[x] = \_find(\_fa[x]);

}

void \_union(int x, int y) {

int xx = \_find(x), yy = \_find(y); if (xx == yy)

return; if (xx > yy)

\_swap(xx, yy);

\_fa[yy] = xx;

}

void Init() {

n = \_r(), m = \_r();

for (int i = 0; i < m; i++) {

ori[i].u = \_r(); ori[i].v = \_r(); ori[i].val = \_r();

}

}

bool cmp(Node a, Node b) { return a.val < b.val;

}

ll kruskal() { ll MST = 0;

for (int i = 1; i <= n; i++)

\_fa[i] = i; sort(ori, ori + m, cmp); int u, v, ok = 1;

for (int i = 0; i < m; i++) { u = ori[i].u; v = ori[i].v; if (\_find(u) == \_find(v))

continue;

\_union(u, v);

addedge(u, v, ori[i].val);

addedge(v, u, ori[i].val); ori[i].use = 1;

MST += ori[i].val; if (++ok == n)

return MST;

}

return -1;

}

int anc[MX][21] = { 0 }, max\_e[2][MX][21] = { 0 }, deep[MX] = { 0 };

void DFS(int u, int fa) { anc[u][0] = fa;

for (int i = 1; i <= 20; i++) {

anc[u][i] = anc[anc[u][i - 1]][i - 1];

max\_e[0][u][i] = max(max\_e[0][u][i - 1], max\_e[0][anc[u][i - 1]][i

- 1]);

- 1]);

max\_e[1][u][i] = max(max\_e[1][u][i - 1], max\_e[1][anc[u][i - 1]][i

if (max\_e[0][u][i - 1] != max\_e[0][anc[u][i - 1]][i - 1]) {

max\_e[1][u][i] = max(max\_e[1][u][i], min(max\_e[0][u][i - 1],

max\_e[0][anc[u][i - 1]][i - 1]));

}

}

for (int i = head[u]; i; i = e[i].\_next) { if (e[i].v ^ fa) {

deep[e[i].v] = deep[u] + 1; max\_e[0][e[i].v][0] = e[i].val; DFS(e[i].v, u);

}

}

}

int lca(int x, int y) {

if (deep[x] < deep[y])swap(x, y); for (int i = 20; i >= 0; i--) {

if (deep[anc[x][i]] >= deep[y]) { x = anc[x][i];

}

}

if (x == y)return x;

for (int i = 20; i >= 0; i--) { if (anc[x][i] ^ anc[y][i]) {

x = anc[x][i];

y = anc[y][i];

}

}

return anc[x][0];

}

ll change(int u, int v, ll try\_val) { int l = lca(u, v);

int loopmax = -1, loopless = -1; for (int i = 20; i >= 0; i--) {

if (deep[anc[u][i]] >= deep[l]) {

if (try\_val != max\_e[0][u][i]) {

if (loopmax < max\_e[0][u][i]) { loopless = loopmax; loopmax = max\_e[0][u][i];

}

}

if (max\_e[1][u][i] && try\_val != max\_e[1][u][i]) loopless = max(loopless, max\_e[1][u][i]);

u = anc[u][i];

}

}

for (int i = 20; i >= 0; i--) {

if (deep[anc[v][i]] >= deep[l]) {

if (try\_val != max\_e[0][v][i]) {

if (loopmax < max\_e[0][v][i]) { loopless = loopmax;

loopmax = max\_e[0][v][i];

}

}

if (max\_e[1][v][i] && try\_val != max\_e[1][v][i]) loopless = max(loopless, max\_e[1][v][i]);

u = anc[v][i];

}

}

return max(loopmax, loopless);

}

void solve() {

ll MST = kruskal(); deep[1] = 1;

DFS(1, 0);

ll ans = ll\_INF, t;

for (int i = 0; i < m; i++) { if (!ori[i].use) {

t = change(ori[i].u, ori[i].v, ori[i].val); if (t != -1)

ans = \_min(ans, MST - t + ori[i].val);

}

}

printf("%lld", ans);

}