**组合数 2 字符串哈希 37**

**前缀和/差分 2 序列自动机 38**

**并查集 4 trie 40**

**exkmp/z函数 5 三分 42**

**分数类 6 逆序对 42**

**hash 8 筛法/数论 43**

**逆元 8 线性基 49**

**快读快写 10 OJ版本测试 56**

**矩阵类 11**

**取模类 14**

**质因数分解相关 18**

**pbds::tree 21**

**质数筛 23**

**快速幂 24**

**后缀自动机 25**

**线段树 32**

**st表 36**

**组合数**

using i64 = long long;

const int mod = 1e9 + 7;

struct Comb {

const int N;

vector<i64> fac, invfac;

Comb(int n) : N(n), fac(n + 2), invfac(n + 2) { init(); };

i64 qpow(i64 x, i64 p) {

i64 res = 1 % mod; x %= mod;

for (; p; p >>= 1, x = x \* x % mod)

if (p & 1) res = res \* x % mod;

return res;

}

i64 inv(i64 x) { return qpow(x, mod - 2); };

void init() {

fac[0] = 1;

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

invfac[N] = inv(fac[N]);

for (int i = N - 1; i >= 0; --i) invfac[i] = (invfac[i + 1] \* (i + 1)) % mod;

}

i64 C(int n, int m) {

if (n < m || m < 0) return 0;

return fac[n] \* invfac[m] % mod \* invfac[n - m] % mod;

}

i64 A(int n, int m) {

if (n < m || m < 0) return 0;

return fac[n] \* invfac[n - m] % mod;

}

};

**前缀和/差分**

//use 1-n

//diff

vector<int> diff(maxn);

void diff\_init(vector<int> &ori)

{

diff.resize(ori.size()+10);

diff[1]=ori[1];

for(int i=2;i<ori.size();i++)

{

diff[i]=ori[i]-ori[i-1];

}

return;

}

//[l,r]

void add(int l,int r,int val)

{

diff[l]+=val;

diff[r+1]-=val;

}

void find(int target)

{

int sum=0;

for(int i=1;i<diff.size()&&i<=target;i++)

{

sum+=diff[i];

//now sum is the current value of ori[i]

//you can do anything on it

}

return;

}

//sum

vector<int> summ(maxn);

void summ\_init(vector<int> &ori)

{

summ.resize(ori.size()+10);

for(int i=1;i<ori.size();i++)

{

summ[i]=summ[i-1]+ori[i];

}

return;

}

long long search(int l,int r)

{

return summ[r]-summ[l-1];

}

**并查集**

struct DSU {

std::vector<int> f, siz;

DSU() {}

DSU(int n) {

init(n);

}

void init(int n) {

f.resize(n);

std::iota(f.begin(), f.end(), 0);

siz.assign(n, 1);

}

int find(int x) {

while (x != f[x]) {

x = f[x] = f[f[x]];

}

return x;

}

bool same(int x, int y) {

return find(x) == find(y);

}

bool merge(int x, int y) {

x = find(x);

y = find(y);

if (x == y) {

return false;

}

if(x>y) std::swap(x,y);

siz[x] += siz[y];

f[y] = x;

return true;

}

int size(int x) {

return siz[find(x)];

}

};

**Exkmp**

vector<int> exkmp(string s) {

int n = (int)s.length();

vector<int> z(n);

for (int i = 1, l = 0, r = 0; i < n; ++i) {

if (i <= r && z[i - l] < r - i + 1) {

z[i] = z[i - l];

} else {

z[i] = max(0ll, r - i + 1);

while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];

}

if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;

}

return z;

}

**分数**

template<class T>

struct Frac {

T num;

T den;

Frac(T num\_, T den\_) : num(num\_), den(den\_) {

if (den < 0) {

den = -den;

num = -num;

}

}

Frac() : Frac(0, 1) {}

Frac(T num\_) : Frac(num\_, 1) {}

explicit operator double() const {

return 1. \* num / den;

}

Frac &operator+=(const Frac &rhs) {

num = num \* rhs.den + rhs.num \* den;

den \*= rhs.den;

return \*this;

}

Frac &operator-=(const Frac &rhs) {

num = num \* rhs.den - rhs.num \* den;

den \*= rhs.den;

return \*this;

}

Frac &operator\*=(const Frac &rhs) {

num \*= rhs.num;

den \*= rhs.den;

return \*this;

}

Frac &operator/=(const Frac &rhs) {

num \*= rhs.den;

den \*= rhs.num;

if (den < 0) {

num = -num;

den = -den;

}

return \*this;

}

friend Frac operator+(Frac lhs, const Frac &rhs) {

return lhs += rhs;

}

friend Frac operator-(Frac lhs, const Frac &rhs) {

return lhs -= rhs;

}

friend Frac operator\*(Frac lhs, const Frac &rhs) {

return lhs \*= rhs;

}

friend Frac operator/(Frac lhs, const Frac &rhs) {

return lhs /= rhs;

}

friend Frac operator-(const Frac &a) {

return Frac(-a.num, a.den);

}

friend bool operator==(const Frac &lhs, const Frac &rhs) {

return lhs.num \* rhs.den == rhs.num \* lhs.den;

}

friend bool operator!=(const Frac &lhs, const Frac &rhs) {

return lhs.num \* rhs.den != rhs.num \* lhs.den;

}

friend bool operator<(const Frac &lhs, const Frac &rhs) {

return lhs.num \* rhs.den < rhs.num \* lhs.den;

}

friend bool operator>(const Frac &lhs, const Frac &rhs) {

return lhs.num \* rhs.den > rhs.num \* lhs.den;

}

friend bool operator<=(const Frac &lhs, const Frac &rhs) {

return lhs.num \* rhs.den <= rhs.num \* lhs.den;

}

friend bool operator>=(const Frac &lhs, const Frac &rhs) {

return lhs.num \* rhs.den >= rhs.num \* lhs.den;

}

friend std::ostream &operator<<(std::ostream &os, Frac x) {

T g = std::gcd(x.num, x.den);

if (x.den == g) {

return os << x.num / g;

} else {

return os << x.num / g << "/" << x.den / g;

}

}

};

**Hash**

struct custom\_hash {

static uint64\_t splitmix64(uint64\_t x) {

x += 0x9e3779b97f4a7c15;

x = (x ^ (x >> 30)) \* 0xbf58476d1ce4e5b9;

x = (x ^ (x >> 27)) \* 0x94d049bb133111eb;

return x ^ (x >> 31);

}

size\_t operator()(uint64\_t x) const {

static const uint64\_t FIXED\_RANDOM = chrono::steady\_clock::now().time\_since\_epoch().count();

return splitmix64(x + FIXED\_RANDOM);

}

};

**逆元**

const int p = 998244353;

int m[maxn];

int mod(int a,int p)

{

return (a%p+p)%p;

}

int inv(int n,int p)

{

if(m[n])

return m[n];

m[n]=mod(-p/n\*inv(p%n,p),p);

return m[n];

}

//线性需要for(i-n)

void exgcd(int a, int b, int& x, int& y) {

if (b == 0) {

x = 1, y = 0;

return;

}

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

y -= a / b \* x;

}

int inv2(int a, int n) {

int x, y;

exgcd(a, n, x, y);

x = (x % n + n) % n;

return x;

}

//ax==1 (mod b)

//gcd(a,n)=1

//ax+by=1

template <typename T,typename T2>

T qpow(T base, T power, T2 mod)

{

T temp = 1;

while (power!=1)

{

if (power & 1)

{

temp \*= base;

temp %= mod;

power--;

continue;

}

base \*= base;

base %= mod;

power >>= 1;

}

return (base \* temp) % mod;

}

int inv3(int a,int p)

{

return qpow(a,p-2,p);

}

//x==a^(p-2) (mod p)

//p is prime

**快读快写**

template<typename T>

void write(T x)

{

if(x<0)

putchar('-'),x=-x;

if(x>9)

write(x/10);

putchar(x%10+'0');

return;

}

inline \_\_int128\_t read()

{

\_\_int128\_t x = 0, f = 1;

char ch = getchar();

while (ch < '0' || ch > '9')

{

if (ch == '-')

f = -1;

ch = getchar();

}

while (ch >= '0' && ch <= '9')

x = x \* 10 + ch - '0', ch = getchar();

return x \* f;

}

ios::sync\_with\_stdio(0);

cin.tie(0);

std::ostream &operator<<(std::ostream &os, lll n) {

std::string s;

while (n) {

s += '0' + n % 10;

n /= 10;

}

std::reverse(s.begin(), s.end());

return os << s;

}

**矩阵类**

struct matrix

{

vector<vector<int>> data;

int row, column; // 行，列

long long MOD = \_\_INT64\_MAX\_\_;

matrix(int n) // 构造n行n列矩阵

{

data.resize(n + 1);

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

{

data[i].resize(n + 1);

}

row = n;

column = n;

}

matrix(int n, int m) // 构造n行m列矩阵

{

data.resize(n + 1);

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

{

data[i].resize(m + 1);

}

row = n;

column = m;

}

void setmod(int modd)

{

MOD=modd;

}

matrix operator-(const matrix &T) const

{

if (this->column != T.column || this->row != T.row)

{

cerr << "minus error!" << endl;

return matrix(0);

}

matrix res(\*this);

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

for (int j = 1; j <= column; ++j)

{

res.data[i][j] -= T.data[i][j];

res.data[i][j] = (res.data[i][j] + MOD) % MOD;

}

return res;

}

matrix operator+(const matrix &T) const

{

if (this->column != T.column || this->row != T.row)

{

cerr << "add error!" << endl;

return matrix(0);

}

matrix res(\*this);

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

for (int j = 1; j <= column; ++j)

{

res.data[i][j] += T.data[i][j];

res.data[i][j] %= MOD;

}

return res;

}

matrix operator\*(const matrix &T) const

{

if (this->column != T.row)

{

cerr << "mul error" << endl;

return matrix(0);

}

matrix res(row, T.column);

int r;

for (int i = 1; i <= res.row; ++i)

for (int k = 1; k <= this->column; ++k)

{

r = data[i][k];

for (int j = 1; j <= res.column; ++j)

res.data[i][j] += T.data[k][j] \* r, res.data[i][j] %= MOD;

}

return res;

}

matrix operator^(long long x) const

{

if (column != row)

{

cerr << "pow error" << Endl;

return matrix(0);

}

if (x == 1)

return \*this;

matrix res(column), bas(column);

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

res.data[i][i] = 1;

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

for (int j = 1; j <= column; ++j)

bas.data[i][j] = data[i][j] % MOD;

while (x)

{

if (x & 1)

res = res \* bas;

bas = bas \* bas;

x >>= 1;

}

return res;

}

};

**取模类**

template<class T>

constexpr T qpow(T a, long long b) {

T res = 1;

for (; b; b /= 2, a \*= a) {

if (b % 2) {

res \*= a;

}

}

return res;

}

constexpr long long mul(long long a, long long b, long long p) {

long long res = a \* b - ((long long)(1.L \* a \* b / p) \* p);

res %= p;

if (res < 0) {

res += p;

}

return res;

}

template<long long P>

struct modint {

long long x;

constexpr modint() : x{} {}

constexpr modint(long long x) : x{norm(x % getMod())} {}

static long long Mod;

constexpr static long long getMod() {

if (P > 0) {

return P;

} else {

return Mod;

}

}

constexpr static void setMod(long long Mod\_) {

Mod = Mod\_;

}

constexpr long long norm(long long x) const {

if (x < 0) {

x += getMod();

}

if (x >= getMod()) {

x -= getMod();

}

return x;

}

constexpr long long val() const {

return x;

}

explicit constexpr operator long long() const {

return x;

}

constexpr modint operator-() const {

modint res;

res.x = norm(getMod() - x);

return res;

}

constexpr modint inv() const {

assert(x != 0);

return qpow(\*this, getMod() - 2);

}

constexpr modint &operator\*=(modint rhs) & {

x = mul(x, rhs.x, getMod());

return \*this;

}

constexpr modint &operator+=(modint rhs) & {

x = norm(x + rhs.x);

return \*this;

}

constexpr modint &operator-=(modint rhs) & {

x = norm(x - rhs.x);

return \*this;

}

constexpr modint &operator/=(modint rhs) & {

return \*this \*= rhs.inv();

}

friend constexpr modint operator\*(modint lhs, modint rhs) {

modint res = lhs;

res \*= rhs;

return res;

}

friend constexpr modint operator+(modint lhs, modint rhs) {

modint res = lhs;

res += rhs;

return res;

}

friend constexpr modint operator-(modint lhs, modint rhs) {

modint res = lhs;

res -= rhs;

return res;

}

friend constexpr modint operator/(modint lhs, modint rhs) {

modint res = lhs;

res /= rhs;

return res;

}

friend constexpr std::istream &operator>>(std::istream &is, modint &a) {

long long v;

is >> v;

a = modint(v);

return is;

}

friend constexpr std::ostream &operator<<(std::ostream &os, const modint &a) {

return os << a.val();

}

friend constexpr bool operator==(modint lhs, modint rhs) {

return lhs.val() == rhs.val();

}

friend constexpr bool operator!=(modint lhs, modint rhs) {

return lhs.val() != rhs.val();

}

};

using mint=modint<998244353>;

**质因数分解相关**

i64 mul(i64 a, i64 b, i64 m) {

return static\_cast<\_\_int128>(a) \* b % m;

}

i64 power(i64 a, i64 b, i64 m) {

i64 res = 1 % m;

for (; b; b >>= 1, a = mul(a, a, m))

if (b & 1)

res = mul(res, a, m);

return res;

}

bool isprime(i64 n) {

if (n < 2)

return false;

static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};

int s = \_\_builtin\_ctzll(n - 1);

i64 d = (n - 1) >> s;

for (auto a : A) {

if (a == n)

return true;

i64 x = power(a, d, n);

if (x == 1 || x == n - 1)

continue;

bool ok = false;

for (int i = 0; i < s - 1; ++i) {

x = mul(x, x, n);

if (x == n - 1) {

ok = true;

break;

}

}

if (!ok)

return false;

}

return true;

}

std::vector<i64> factorize(i64 n) {

std::vector<i64> p;

std::function<void(i64)> f = [&](i64 n) {

if (n <= 10000) {

for (int i = 2; i \* i <= n; ++i)

for (; n % i == 0; n /= i)

p.push\_back(i);

if (n > 1)

p.push\_back(n);

return;

}

if (isprime(n)) {

p.push\_back(n);

return;

}

auto g = [&](i64 x) {

return (mul(x, x, n) + 1) % n;

};

i64 x0 = 2;

while (true) {

i64 x = x0;

i64 y = x0;

i64 d = 1;

i64 power = 1, lam = 0;

i64 v = 1;

while (d == 1) {

y = g(y);

++lam;

v = mul(v, std::abs(x - y), n);

if (lam % 127 == 0) {

d = std::gcd(v, n);

v = 1;

}

if (power == lam) {

x = y;

power \*= 2;

lam = 0;

d = std::gcd(v, n);

v = 1;

}

}

if (d != n) {

f(d);

f(n / d);

return;

}

++x0;

}

};

f(n);

std::sort(p.begin(), p.end());

return p;

}

**Pbds::tree**

\_\_gnu\_pbds :: tree

#include <ext/pb\_ds/assoc\_container.hpp> // 因为tree定义在这里 所以需要包含这个头文件

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

\_\_gnu\_pbds ::tree<Key, Mapped, Cmp\_Fn = std::less<Key>, Tag = rb\_tree\_tag,

Node\_Update = null\_tree\_node\_update,

Allocator = std::allocator<char> >

模板形参

Key: 储存的元素类型，如果想要存储多个相同的 Key 元素，则需要使用类似于 std::pair 和 struct 的方法，并配合使用 lower\_bound 和 upper\_bound 成员函数进行查找

Mapped: 映射规则（Mapped-Policy）类型，如果要指示关联容器是 集合，类似于存储元素在 std::set 中，此处填入 null\_type，低版本 g++ 此处为 null\_mapped\_type；如果要指示关联容器是 带值的集合，类似于存储元素在 std::map 中，此处填入类似于 std::map<Key, Value> 的 Value 类型

Cmp\_Fn: 关键字比较函子，例如 std::less<Key>

Tag: 选择使用何种底层数据结构类型，默认是 rb\_tree\_tag。\_\_gnu\_pbds 提供不同的三种平衡树，分别是：

rb\_tree\_tag：红黑树，一般使用这个，后两者的性能一般不如红黑树

splay\_tree\_tag：splay 树

ov\_tree\_tag：有序向量树，只是一个由 vector 实现的有序结构，类似于排序的 vector 来实现平衡树，性能取决于数据想不想卡你

Node\_Update：用于更新节点的策略，默认使用 null\_node\_update，若要使用 order\_of\_key 和 find\_by\_order 方法，需要使用 tree\_order\_statistics\_node\_update

Allocator：空间分配器类型

构造方式

\_\_gnu\_pbds::tree<std::pair<int, int>, \_\_gnu\_pbds::null\_type,

std::less<std::pair<int, int> >, \_\_gnu\_pbds::rb\_tree\_tag,

\_\_gnu\_pbds::tree\_order\_statistics\_node\_update>

trr;

成员函数

insert(x)：向树中插入一个元素 x，返回 std::pair<point\_iterator, bool>。

erase(x)：从树中删除一个元素/迭代器 x，返回一个 bool 表明是否删除成功。

order\_of\_key(x)：返回 x 以 Cmp\_Fn 比较的排名。

find\_by\_order(x)：返回 Cmp\_Fn 比较的排名所对应元素的迭代器。

lower\_bound(x)：以 Cmp\_Fn 比较做 lower\_bound，返回迭代器。

upper\_bound(x)：以 Cmp\_Fn 比较做 upper\_bound，返回迭代器。

join(x)：将 x 树并入当前树，前提是两棵树的类型一样，x 树被删除。

split(x,b)：以 Cmp\_Fn 比较，小于等于 x 的属于当前树，其余的属于 b 树。

empty()：返回是否为空。

size()：返回大小。

示例

// Common Header Simple over C++11

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef unsigned long long ull;

typedef long double ld;

typedef pair<int, int> pii;

#define pb push\_back

#define mp make\_pair

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

\_\_gnu\_pbds ::tree<pair<int, int>, \_\_gnu\_pbds::null\_type, less<pair<int, int> >,

\_\_gnu\_pbds::rb\_tree\_tag,

\_\_gnu\_pbds::tree\_order\_statistics\_node\_update>

trr;

int main() {

int cnt = 0;

trr.insert(mp(1, cnt++));

trr.insert(mp(5, cnt++));

trr.insert(mp(4, cnt++));

trr.insert(mp(3, cnt++));

trr.insert(mp(2, cnt++));

// 树上元素 {{1,0},{2,4},{3,3},{4,2},{5,1}}

auto it = trr.lower\_bound(mp(2, 0));

trr.erase(it);

// 树上元素 {{1,0},{3,3},{4,2},{5,1}}

auto it2 = trr.find\_by\_order(1);

cout << (\*it2).first << endl;

// 输出排名 0 1 2 3 中的排名 1 的元素的 first:1

int pos = trr.order\_of\_key(\*it2);

cout << pos << endl;

// 输出排名

decltype(trr) newtr;

trr.split(\*it2, newtr);

for (auto i = newtr.begin(); i != newtr.end(); ++i) {

cout << (\*i).first << ' ';

}

cout << endl;

// {4,2},{5,1} 被放入新树

trr.join(newtr);

for (auto i = trr.begin(); i != trr.end(); ++i) {

cout << (\*i).first << ' ';

}

cout << endl;

cout << newtr.size() << endl;

// 将 newtr 树并入 trr 树，newtr 树被删除。

return 0;

}

**质数筛**

vector<int> pri;

bool not\_prime[maxn];

void pre(int n) {

for (int i = 2; i <= n; ++i) {

if (!not\_prime[i]) {

pri.push\_back(i);

}

for (int pri\_j : pri) {

if (i \* pri\_j > n) break;

not\_prime[i \* pri\_j] = true;

if (i % pri\_j == 0) {

break;

}

}

}

}

vector<int> prime;

bool is\_prime[maxn];

void Eratosthenes(int n)

{

is\_prime[0] = is\_prime[1] = false;

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

{is\_prime[i] = true;}

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

{

if (is\_prime[i])

{

prime.push\_back(i);

if ((long long)i \* i > n)

continue;

for (int j = i \* i; j <= n; j += i)

{

is\_prime[j] = false;

}

}

}

}

**快速幂**

template <typename T>

T qpow(T a, T b, long long p)

{

int res = 1;

for (; b; b /= 2, a = (1ll \* a \* a) % p)

{

if (b % 2)

{

res = 1LL \* res \* a % p;

}

}

return res;

}

template <typename T>

T qpow(T a, T b)

{

int res = 1;

for (; b; b /= 2, a = (1ll \* a \* a))

{

if (b % 2)

{

res = 1LL \* res \* a;

}

}

return res;

}

**后缀自动机**

struct SAM {

//edited by piaoyun from some other's code

//必须#define int long long

static const int MAXN=1000010,MAXS=28;

int tot=1,last=1,link[MAXN << 1],ch[MAXN << 1][MAXS],len[MAXN << 1],endpos[MAXN << 1];

//总点数tot，点的index属于[1-tot]，空串/根为1

//last为上一次插入的点

//link为点的parent树父节点 / 最长 出现位置与自己不同 的后缀

//ch[n][s] 指节点n末尾加字符s所转移到的点

//len指该节点的串的 最长长度，注意到 最短长度 等于 len[link[n]] + 1 即父节点最长 + 1

//endpos[n] 参考get\_endpos()的注释

//可以不用define int ll 注意空间复杂度

void clear(){

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

link[i] = len[i] = endpos[i] = 0;

for(int k = 0; k < MAXS; k++) ch[i][k] = 0;

}

tot=1;last=1;

}

//延长一个字符，通常为[1-26]

void extend(int w){

int p=++tot,x=last,r,q;

endpos[p]=1;

for(len[last=p]=len[x]+1; x&&!ch[x][w]; x=link[x]) ch[x][w]=p;

if(!x)link[p]=1;

else if(len[x]+1==len[q=ch[x][w]]) link[p]=q;

else {

link[r=++tot]=link[q];

memcpy(ch[r],ch[q],sizeof ch[r]);

//ch[r] = ch[q]; 修改成map时使用

len[r]=len[x]+1;

link[p]=link[q]=r;

for(; x&&ch[x][w]==q; x=link[x])ch[x][w]=r;

}

}

//\*注意vector占用的空间

vector<int> p[MAXN << 1]; //建立parent树，以便从上到下dfs

void dfs(int u){

int v;

for(int i=0;i<p[u].size();i++){

v=p[u][i];

dfs(v);

endpos[u]+=endpos[v];

}

}

//注意！在使用该方法前，endpos[]代表每个点作为“终结点”的次数

//使用该方法后，endpos[]指在串中出现总次数，即原数组的子树求和

void get\_endpos(){

for(int i = 1;i <= tot; i++) p[i].clear();

for(int i = 2;i <= tot; i++){

p[link[i]].push\_back(i); //建立parent树，以便从上到下dfs

}

dfs(1);

for(int i = 1;i <= tot; i++) p[i].clear();

}

//\*在您不确定是否有抄写错误时再使用该方法

//\*必须在输入任何数据前自检，此前的数据会被清空

static const int STC = 998244353;

void self\_test(){

clear();

for(int i = 1;i <= 1000; i++) extend(i \* i % 26 + 1);

int tmp = 107 \* last + 301 \* tot;

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

tmp = (tmp \* 33 + link[i] \* 101 + len[i] \* 97) % STC;

for(int k = 1; k < MAXS; k++) tmp = (tmp + k \* ch[i][k]) % STC;

}

assert("stage 1" && tmp == 393281314); // stage1 : 检查建树是否正确

tmp = 0;

get\_endpos();

for(int i = 1;i <= tot; i++) tmp = (tmp \* 33 + endpos[i]) % STC;

assert("stage 2" && tmp == 178417668); // stage2 : 检查endpos计算是否正确，如果您修改了endpos[]的含义则会报错

cout<<"Self Test Passed.Remember to delete this function's use."<<endl;

clear();

}

//调试时可调用

void debug\_print(){

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

cout<<"node:"<<i<<" father:"<<link[i]<<" endpos:"<<endpos[i]<<" len:"<<len[i]<<endl;

}

}

ll solve(){

//在这里输入你自己的解题逻辑

ll ans = 0;

get\_endpos();

for(int i = 1;i <= tot; i++) if(endpos[i] >= 2) ans = max(ans,(ll)endpos[i] \* len[i]);

return ans;

}

}sam;

struct SAM2

//map版

{

// edited by piaoyun from some other's code

// 必须#define int long long

static const int MAXN = 100000, MAXS = 28;

int tot = 1, last = 1, link[MAXN << 1], len[MAXN << 1], endpos[MAXN << 1];

map<int, int> ch[MAXN << 1];

// 总点数tot，点的index属于[1-tot]，空串/根为1

// last为上一次插入的点

// link为点的parent树父节点 / 最长 出现位置与自己不同 的后缀

// ch[n][s] 指节点n末尾加字符s所转移到的点

// len指该节点的串的 最长长度，注意到 最短长度 等于 len[link[n]] + 1 即父节点最长 + 1

// endpos[n] 参考get\_endpos()的注释

void clear()

{

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

{

link[i] = len[i] = endpos[i] = 0;

ch[i].clear();

}

tot = 1;

last = 1;

}

// 延长一个字符，通常为[1-26]

void extend(int w)

{

int p = ++tot, x = last, r, q;

endpos[p] = 1;

for (len[last = p] = len[x] + 1; x && !ch[x].contains(w); x = link[x])

ch[x][w] = p;

if (!x)

link[p] = 1;

else if (len[x] + 1 == len[q = ch[x][w]])

link[p] = q;

else

{

link[r = ++tot] = link[q];

ch[r] = ch[q];

len[r] = len[x] + 1;

link[p] = link[q] = r;

for (; x && ch[x][w] == q; x = link[x])

ch[x][w] = r;

}

}

//\*注意vector占用的空间

vector<int> p[MAXN << 1]; // 建立parent树，以便从上到下dfs

void dfs(int u)

{

int v;

for (int i = 0; i < p[u].size(); i++)

{

v = p[u][i];

dfs(v);

endpos[u] += endpos[v];

}

}

// 注意！在使用该方法前，endpos[]代表每个点作为“终结点”的次数

// 使用该方法后，endpos[]指在串中出现总次数，即原数组的子树求和

void get\_endpos()

{

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

p[i].clear();

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

{

p[link[i]].push\_back(i); // 建立parent树，以便从上到下dfs

}

dfs(1);

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

p[i].clear();

}

//\*在您不确定是否有抄写错误时再使用该方法

//\*必须在输入任何数据前自检，此前的数据会被清空

static const int STC = 998244353;

void self\_test()

{

clear();

for (int i = 1; i <= 1000; i++)

extend(i \* i % 26 + 1);

int tmp = 107 \* last + 301 \* tot;

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

{

tmp = (tmp \* 33 + link[i] \* 101 + len[i] \* 97) % STC;

for (int k = 1; k < MAXS; k++)

tmp = (tmp + k \* ch[i][k]) % STC;

}

assert("stage 1" && tmp == 393281314); // stage1 : 检查建树是否正确

tmp = 0;

get\_endpos();

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

tmp = (tmp \* 33 + endpos[i]) % STC;

assert("stage 2" && tmp == 178417668); // stage2 : 检查endpos计算是否正确，如果您修改了endpos[]的含义则会报错

cout << "Self Test Passed.Remember to delete this function's use." << endl;

clear();

}

// 调试时可调用

void debug\_print()

{

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

{

cout << "node:" << i << " father:" << link[i] << " endpos:" << endpos[i] << " len:" << len[i] << endl;

}

}

ll solve()

{

// 在这里输入你自己的解题逻辑

ll ans = 0;

get\_endpos();

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

if (endpos[i] >= 2)

ans = max(ans, (ll)endpos[i] \* len[i]);

return ans;

}

} sam;

string tmp;

void prepare() {

//sam.self\_test();

cin>>tmp;

sam.clear();

for(int i = 0; i < tmp.size(); i++) sam.extend(tmp[i]-'a'+1);

cout<<sam.solve();

}

**线段树**

struct segment\_tree\_node\_lazytag

{

    long long add;

};

struct segment\_tree\_node

{

    long long val;

    long long length;

    int l, r;

    long long min;

    int minidx; // 这个模版维护的是最小idx

    segment\_tree\_node\_lazytag tag;

};

struct segment\_tree

{

    std::vector<segment\_tree\_node> tree;

    segment\_tree(int max\_size)

    {

        tree.resize(max\_size \* 4 + 100);

    }

    void pushup(int idx)

    {

        tree[idx].val = tree[idx \* 2].val + tree[idx \* 2 + 1].val;

        if (tree[idx \* 2].min <= tree[idx \* 2 + 1].min)

        {

            tree[idx].minidx = tree[idx \* 2].minidx;

        }

        else

        {

            tree[idx].minidx = tree[idx \* 2 + 1].minidx;

        }

        tree[idx].min = std::min(tree[idx \* 2].min, tree[idx \* 2 + 1].min);

    }

    void build(int l, int r, std::vector<int> &a, int idx = 1)

    {

        tree[idx].l = l;

        tree[idx].r = r;

        tree[idx].length = r - l + 1;

        if (l == r)

        {

            tree[idx].val = a[l];

            tree[idx].length = 1;

            tree[idx].l = tree[idx].r = l;

            tree[idx].min = a[l];

            tree[idx].minidx = l;

        }

        else

        {

            int mid = (l + r) >> 1;

            build(l, mid, a, idx \* 2);

            build(mid + 1, r, a, idx \* 2 + 1);

            pushup(idx);

        }

    }

    void pushdown(int idx)

    {

        tree[idx \* 2].tag.add += tree[idx].tag.add;

        tree[idx \* 2 + 1].tag.add += tree[idx].tag.add;

        tree[idx \* 2].val += tree[idx \* 2].length \* tree[idx].tag.add;

        tree[idx \* 2 + 1].val += tree[idx \* 2 + 1].length \* tree[idx].tag.add;

        tree[idx \* 2].min += tree[idx].tag.add;

        tree[idx \* 2 + 1].min += tree[idx].tag.add;

        tree[idx].tag.add = 0;

    }

    ll find\_sum(int l, int r, int idx = 1)

    {

        if (tree[idx].l > r || tree[idx].r < l)

            return 0;

        if (tree[idx].l >= l && tree[idx].r <= r)

            return tree[idx].val;

        pushdown(idx);

        pushup(idx);

        return find\_sum(l,r,idx\*2)+find\_sum(l,r,idx\*2+1);

    }

    pair<ll,ll> find\_min(int l, int r, int idx = 1)//minvalue minidx

    {

        if (tree[idx].l > r || tree[idx].r < l)

            return {INT64\_MAX,-1};

        if (tree[idx].l >= l && tree[idx].r <= r)

            return {tree[idx].min,tree[idx].minidx};

        pushdown(idx);

        pushup(idx);

        auto a=find\_min(l, r, idx \* 2), b=find\_min(l, r, idx \* 2 + 1);

        if(a.first<=b.first) rt a;

        else rt b;

    }

    void add(int l, int r, int add\_val, int idx = 1)

    {

        if (tree[idx].l > r || tree[idx].r < l)

            return;

        if (tree[idx].l >= l && tree[idx].r <= r)

        {

            tree[idx].tag.add += add\_val;

            tree[idx].val += add\_val \* tree[idx].length;

            tree[idx].min += add\_val;

            return;

        }

        pushdown(idx);

        add(l, r, add\_val, idx \* 2);

        add(l, r, add\_val, idx \* 2 + 1);

        pushup(idx);

    } // 区间加上x

    void set(int l, int val, int idx = 1)

    {

        if (tree[idx].l > l || tree[idx].r < l)

            return;

        if (tree[idx].l == l && tree[idx].r == l)

        {

            tree[idx].val = val;

            tree[idx].min = val;

            return;

        }

        pushdown(idx);

        set(l, val, idx \* 2);

        set(l, val, idx \* 2 + 1);

        pushup(idx);

    }

};

**ST表**

int LOG2[maxn];

int st\_table\_init\_tag = 0;

vector<int> a(maxn);

int st\_table[maxn][30];

template <typename T>

T op(T a, T b)

{

return std::max(a, b);

}

void st\_table\_init()

{

LOG2[0] = 0;

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

LOG2[i] = log2(i);

st\_table\_init\_tag = 1;

}

void build()

{

if (!st\_table\_init\_tag)

st\_table\_init();

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

st\_table[i][0]=a[i];

for (int j = 1; j <= LOG2[n]; j++)

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

{

st\_table[i][j] = op(st\_table[i][j - 1], st\_table[i + (1 << (j - 1))][j - 1]);

// op[i,i+2^j-1]

}

}

int srh(int l, int r)

{

if (l == r)

return st\_table[l][0];

int tp = (LOG2[r - l + 1]);

return op(st\_table[l][tp], st\_table[r + 1 - (1 << tp)][tp]);

}

**字符串哈希**

using ui64 = unsigned long long;

using PUU = pair<ui64, ui64>;

using i64 = long long;

// N 为字符串最长长度 p为seed mod1/mod2 为两个模数

const ui64 N = 1e5 + 10, p = 131, mod1 = 998244853, mod2 = 1e9 + 7;

ui64 a1[N], a2[N], hs1[N], hs2[N];

void init() {//初始化 a 数组

a1[0] = a2[0] = 1;

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

a1[i] = a1[i - 1] \* p % mod1;

a2[i] = a2[i - 1] \* p % mod2;

}

}

void hashstr(string& str) {//将str哈希化

int n = str.size();

//默认str下标从0开始 如果从1开始则需要修改str[i - 1]为str[i]

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

hs1[i] = (hs1[i - 1] \* p % mod1 + str[i - 1]) % mod1;

hs2[i] = (hs2[i - 1] \* p % mod2 + str[i - 1]) % mod2;

}

}

ui64 geths1(int l, int r) {//得到str[l -- r]的第一哈希值 定义域[1, n]

return (hs1[r] - hs1[l - 1] \* a1[r - l + 1] % mod1 + mod1) % mod1;

}

ui64 geths2(int l, int r) {//得到str[l -- r]的第二哈希值 定义域[1, n]

return (hs2[r] - hs2[l - 1] \* a2[r - l + 1] % mod2 + mod2) % mod2;

}

使用: 给出一个字符串 str，且 len 为 str 的长度

\*init() <初始化预处理>

\*hashstr(str) <将字符串str哈希化预处理>

-hs1[len] / hs2[len] <str整串的第一/二哈希值>

-geths1(l, r) / geths2(l, r) <得到str[l--r]的第一/二哈希值>

<将哈希值存入 map<PUU, int>s, set<PUU> s 进一步处理>

例如:

map[{ geths1(l, r), geths2(l, r) }];

set,insert({ geths1(l, r), geths2(l, r) });

(注意: l，r严格在闭区间[1, n]内)

**序列自动机**

struct SUBM

{

// 1-idx

static const int maxn = 2050, maxs = 26;

int ch[maxn][maxs], la[maxs];

int idx = maxn;

void clear()

{

memset(la, 0, sizeof la);

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

for (int j = 0; j < maxs; j++)

{

ch[i][j] = 0;

}

idx=1;

}

void clear(int n)

{

memset(la, 0, sizeof la);

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

for (int j = 0; j < maxs; j++)

{

ch[i][j] = 0;

}

idx=1;

}

void init(string &s)

{

clear(s.size()+10);

for (int i = s.size(); i >= 2; i--)

{

for (int j = 0; j < maxs; j++)

{

ch[i][j] = la[j];

}

la[s[i - 1] - 'a'] = i;

}

for (int j = 0; j < maxs; j++)

{

ch[1][j] = la[j];

}

idx = s.size() + 3;

}

void init(vector<int> &a)

{

clear(s.size()+10);

for (int i = a.size(); i >= 2; i--)

{

for (int j = 0; j < maxs; j++)

{

ch[i][j] = la[j];

}

la[a[i - 1]] = i;

}

for (int j = 0; j < maxs; j++)

{

ch[1][j] = la[j];

}

idx = a.size() + 3;

}

}

**Trie**

const int N = 3e6 + 10, M = 26 + 26 + 10 + 10;//N 字符串总长度 M 字符种类数目

int trie[N][M], cnt[N], idx = 0;

bool exist[N];

void clear()

{

fill(trie[0], trie[0] + idx \* M, 0);

fill(exist, exist + idx + 1, false);

fill(cnt, cnt + idx + 1, 0);

idx = 0;

}

int pos(char x)

{

if (x >= 'a' && x <= 'z') return x - 'a' + 1;

if (x >= 'A' && x <= 'Z') return x -'A' + 27;

if (x >= '0' && x <= '9') return x -'0' + 53;

}

void insert(string & str)//插入字符串 str

{

int now = 0;

for (auto& x : str)

{

if (!trie[now][pos(x)]) trie[now][pos(x)] = ++idx;

now = trie[now][pos(x)];

++cnt[now];

}

exist[now] = true;

}

int findpre(string & str)//查询有多少个相同前缀 str

{

int now = 0;

for (auto& x : str)

{

if (!trie[now][pos(x)]) return 0;

now = trie[now][pos(x)];

}

return cnt[now];

}

bool findstr(string & str)//查询是否存在字符串 str

{

int now = 0;

for (auto& x : str)

{

if (!trie[now][pos(x)]) return false;

now = trie[now][pos(x)];

}

return exist[now];

}

使用: 给出字符串 <str>

-clear() <清空字典树>

-insert(str) <向树上插入str字符串>

-findpre(str) <查询是否有前缀串str>

-findstr(str) <查询是否存在字符串str>

**三分**

int l = 1, r = \*max\_element(all(c));

while (l < r - 1)

{

int mid1 = (l + r) >> 1;

int mid2 = (mid1 - 1);

int mid3 = (mid1 + 1);

int sum1 = 0, sum2 = 0;

for (auto i : c)

{

sum1 += cnt(i, mid2);

sum2 += cnt(i, mid3);

}

sum1 \*= b;

sum2 \*= b;

sum1 -= (mid2 - 1) \* x;

sum2 -= (mid3 - 1) \* x;

if (sum1 > sum2)

{

r = mid1;

}

else

l = mid1;

}

**逆序对**

int t[maxn];

int srt(vector<int> &a, int l ,int r){

if (l == r) return 0;

int mid = (l + r )>> 1;

int res = srt(a, l , mid) + srt(a, mid + 1, r);

int i = l , j = mid + 1, k = 0;

while (i <= mid && j <= r){

if (a[i] <= a[j]) t[k++] = a[i++];

else{

t[k++] = a[j++];

res += mid - i + 1;

}

}

while (i <= mid) t[k++] = a[i++];

while (j <= r) t[k++] = a[j++];

for (int i = 0 , j = l ; j <= r ; i ++ , j ++) a[j] = t[i];

return res;

}

//1-n index

//如果直接引用调用，会排序数组a

**筛法相关**

// 线性筛求欧拉函数

vector<int> pri;

bool not\_prime[N];

int phi[N];

void pre(int n)

{

phi[1] = 1;

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

{

if (!not\_prime[i])

{

pri.push\_back(i);

phi[i] = i - 1;

}

for (int pri\_j : pri)

{

if (i \* pri\_j > n)

break;

not\_prime[i \* pri\_j] = true;

if (i % pri\_j == 0)

{

phi[i \* pri\_j] = phi[i] \* pri\_j;

break;

}

phi[i \* pri\_j] = phi[i] \* phi[pri\_j];

}

}

}

// 筛法求莫比乌斯函数

vector<int> pri;

bool not\_prime[N];

int mu[N];

void pre(int n)

{

mu[1] = 1;

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

{

if (!not\_prime[i])

{

mu[i] = -1;

pri.push\_back(i);

}

for (int pri\_j : pri)

{

if (i \* pri\_j > n)

break;

not\_prime[i \* pri\_j] = true;

if (i % pri\_j == 0)

{

mu[i \* pri\_j] = 0;

break;

}

mu[i \* pri\_j] = -mu[i];

}

}

}

// 筛法求因数个数

vector<int> pri;

bool not\_prime[N];

int d[N], num[N];

void pre(int n)

{

d[1] = 1;

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

{

if (!not\_prime[i])

{

pri.push\_back(i);

d[i] = 2;

num[i] = 1;

}

for (int pri\_j : pri)

{

if (i \* pri\_j > n)

break;

not\_prime[i \* pri\_j] = true;

if (i % pri\_j == 0)

{

num[i \* pri\_j] = num[i] + 1;

d[i \* pri\_j] = d[i] / num[i \* pri\_j] \* (num[i \* pri\_j] + 1);

break;

}

num[i \* pri\_j] = 1;

d[i \* pri\_j] = d[i] \* 2;

}

}

}

// 因数个数前缀和 n<2^63(SP26073)

// O(n^(1/3)logn)

void myw(lll x)

{

if (!x)

return;

myw(x / 10);

printf("%d", (int)(x % 10));

}

struct vec

{

ll x, y;

vec(ll x0 = 0, ll y0 = 0) { x = x0, y = y0; }

vec operator+(const vec b) { return vec(x + b.x, y + b.y); }

};

ll N;

vec stk[1000005];

int len;

vec P;

vec L, R;

bool ninR(vec a) { return N < (lll)a.x \* a.y; }

bool steep(ll x, vec a) { return (lll)N \* a.x <= (lll)x \* x \* a.y; }

lll Solve()

{

len = 0;

ll cbr = cbrt(N), sqr = sqrt(N);

P.x = N / sqr, P.y = sqr + 1;

lll ans = 0;

stk[++len] = vec(1, 0);

stk[++len] = vec(1, 1);

while (1)

{

L = stk[len--];

while (ninR(vec(P.x + L.x, P.y - L.y)))

ans += (lll)P.x \* L.y + (lll)(L.y + 1) \* (L.x - 1) / 2,

P.x += L.x, P.y -= L.y;

if (P.y <= cbr)

break;

R = stk[len];

while (!ninR(vec(P.x + R.x, P.y - R.y)))

L = R, R = stk[--len];

while (1)

{

vec mid = L + R;

if (ninR(vec(P.x + mid.x, P.y - mid.y)))

R = stk[++len] = mid;

else if (steep(P.x + mid.x, R))

break;

else

L = mid;

}

}

for (int i = 1; i < P.y; i++)

ans += N / i;

return ans \* 2 - sqr \* sqr;

}

int T;

int main()

{

scanf("%d", &T);

while (T--)

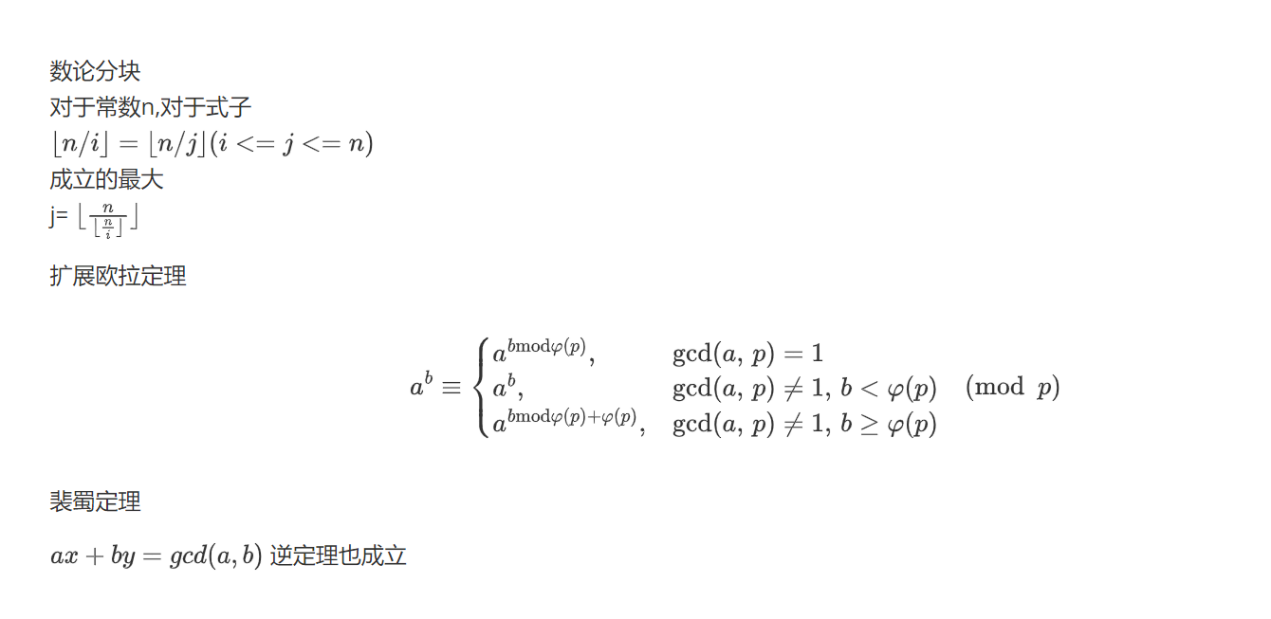
{

scanf("%lld", &N);

myw(Solve());

printf("\n");

}

}

**线性基**

struct LinearBase

{

    // 高斯消元前 idx从0开始

    // 高斯消元后 idx从0开始

    // 前缀线性基：维护一个当前基底的右端点，插入如果靠右，就交换

    int N;

    vector<ull> base;

    int iffzero = 0;

    LinearBase()

    {

        N = 100;

        base.resize(102);

    }

    LinearBase(int n)

    {

        N = n + 5;

        base.resize(n + 20);

    }

    void clear()

    {

        N = 0;

        base.clear();

    }

    bool insert(ull x)

    {

        assert(N != 0);

        for (int i = 62; i >= 0; i--)

        {

            if (!(x & (1ull << i)))

                continue;

            if (!base[i])

            {

                base[i] = x;

                return 1;

            }

            x ^= base[i];

            if (x == 0)

            {

                iffzero = 1;

                return false;

            }

        }

        return 1;

    }

    void gauss()

    {

        assert(N != 0);

        sort(base.begin(), base.end(), greater<ull>());

        int row = 0;

        for (int i = 62; i >= 0; i--)

        {

            for (int j = row; j < N; j++)

            {

                if ((1ull << i) & base[j])

                {

                    swap(base[j], base[row]);

                }

            }

            if (!((1ull << i) & base[row]))

                continue;

            for (int j = 0; j < N; j++)

            {

                if (j == row)

                    continue;

                if ((1ull << i) & base[j])

                {

                    base[j] ^= base[row];

                }

            }

            row++;

        }

    }

    void printbi(ull x)

    {

        for (int j = 62; j >= 0; j--)

        {

            if ((1ull << j) & x)

            {

                cerr << 1;

            }

            else

                cerr << 0;

        }

        cerr << endl;

    }

    void print()

    {

        assert(N != 0);

        for (int i = 0; i < N; i++)

        {

            for (int j = 62; j >= 0; j--)

            {

                if ((1ull << j) & base[i])

                {

                    cerr << 1;

                }

                else

                    cerr << 0;

            }

            cerr << endl;

        }

        cerr << endl;

    }

    LinearBase merge(LinearBase &lb1, LinearBase &lb2)

    {

        LinearBase ans;

        for (int i = 0; i < 65; i++)

        {

            ans.insert(lb1.base[i]);

            ans.insert(lb2.base[i]);

        }

        return ans;

    };

};

struct LinearBase  //前缀线性基模版参考，注意！开了unsigned int

{

    // 高斯消元前 idx从1开始

    // 高斯消元后 idx从0开始

    int N;

    vector<unsigned int> base;

    vector<int> idx;

    int iffzero = 0;

    LinearBase()

    {

        N = 20;

        base.resize(21);

        idx.resize(21);

    }

    LinearBase(int n)

    {

        N = n + 5;

        base.resize(n + 20);

    }

    void clear()

    {

        N = 0;

        base.clear();

    }

    bool insert(ull x,int dx)

    {

        assert(N != 0);

        for (int i = 20; i >= 0; i--)

        {

            if (!(x & (1ull << i)))

                continue;

            if (!base[i])

            {

                base[i] = x;

                idx[i]=dx;

                return 1;

            }

            if(idx[i]<dx)

            {

                swap(base[i],x);

                swap(idx[i],dx);

            }

            x ^= base[i];

            if (x == 0)

            {

                iffzero = 1;

                return false;

            }

        }

        return 1;

    }

    void gauss()

    {

        assert(N != 0);

        sort(base.begin(), base.end(), greater<ull>());

        int row = 0;

        for (int i = 20; i >= 0; i--)

        {

            for (int j = row; j < N; j++)

            {

                if ((1ull << i) & base[j])

                {

                    swap(base[j], base[row]);

                    swap(idx[j],idx[row]);

                }

            }

            if (!((1ull << i) & base[row]))

                continue;

            for (int j = 0; j < N; j++)

            {

                if (j == row)

                    continue;

                if ((1ull << i) & base[j])

                {

                    base[j] ^= base[row];

                }

            }

            row++;

        }

    }

    void printbi(ull x)

    {

        for (int j = 20; j >= 0; j--)

        {

            if ((1ull << j) & x)

            {

                cerr << 1;

            }

            else

                cerr << 0;

        }

        cerr << endl;

    }

    void print()

    {

        assert(N != 0);

        for (int i = 0; i < N; i++)

        {

            for (int j = 20; j >= 0; j--)

            {

                if ((1ull << j) & base[i])

                {

                    cerr << 1;

                }

                else

                    cerr << 0;

            }

            cerr << endl;

        }

        cerr << endl;

    }

};

**OJ测试**

for (int i : {1, 2}) {} // GNU C++11 支持范围表达式

auto cc = [&](int x) { x++; }; // GNU C++11 支持 auto 与 lambda 表达式

cc(2);

tuple<string, int, int> V; // GNU C++11 引入

array<int, 3> C; // GNU C++11 引入

auto dfs = [&](auto self, int x) -> void { // GNU C++14 支持 auto 自递归

    if (x > 10) return;

    self(self, x + 1);

};

dfs(dfs, 1);

vector in(1, vector<int>(1)); // GNU C++17 支持 vector 模板类型缺失

map<int, int> dic;

for (auto [u, v] : dic) {} // GNU C++17 支持 auto 解绑

dic.contains(12); // GNU C++20 支持 contains 函数

auto dfs(this auto self)->void

{};//C++23

using i64 = \_\_int128; // 64 位 GNU C++11 支持

#define int long long

map<int, int> dic;

int x = dic.size() - 1;

cout << x << endl;//windows ：-1 |  其他：随机数