目录

[动态规划 1](#_Toc113696881)

[数位dp 1](#_Toc113696882)

[基础 2](#_Toc113696883)

[二分推荐 2](#_Toc113696884)

[二分 2](#_Toc113696885)

[归并排序推荐 3](#_Toc113696886)

[归并排序 4](#_Toc113696887)

[快读快写 5](#_Toc113696888)

[快排推荐 6](#_Toc113696889)

[快排教材 6](#_Toc113696890)

[离散化 7](#_Toc113696891)

[双下表转存 7](#_Toc113696892)

[桶排序 8](#_Toc113696893)

[字符串 8](#_Toc113696894)

[bitset 9](#_Toc113696895)

[gcd\_lcm 9](#_Toc113696896)

[template\_head\_file 9](#_Toc113696897)

[template 10](#_Toc113696898)

[几何 11](#_Toc113696899)

[几何数据结构 11](#_Toc113696900)

[凸包andrew 13](#_Toc113696901)

[旋转卡壳 14](#_Toc113696902)

[geometric 15](#_Toc113696903)

[数据结构 17](#_Toc113696904)

[并查集 17](#_Toc113696905)

[带权并查集 18](#_Toc113696906)

[带修莫队 19](#_Toc113696907)

[分块 21](#_Toc113696908)

[回滚莫队 23](#_Toc113696909)

[莫队 26](#_Toc113696910)

[树状数组 29](#_Toc113696911)

[线段树 30](#_Toc113696912)

[lca推荐 31](#_Toc113696913)

[lca欧拉序版 32](#_Toc113696914)

[st表 33](#_Toc113696915)

[数学 34](#_Toc113696916)

[杜教筛 34](#_Toc113696917)

[分解质因数大数 36](#_Toc113696918)

[分解质因数小数 38](#_Toc113696919)

[高精度推荐 39](#_Toc113696920)

[高精度 40](#_Toc113696921)

[高精度我写版 43](#_Toc113696922)

[卡特兰数推荐 44](#_Toc113696923)

[卡特兰数1 45](#_Toc113696924)

[卡特兰数2 46](#_Toc113696925)

[卡特兰数3 47](#_Toc113696926)

[快速幂 47](#_Toc113696927)

[米勒拉宾素性检验 48](#_Toc113696928)

[莫比乌斯函数 49](#_Toc113696929)

[逆序数 50](#_Toc113696930)

[逆元费马小定理 51](#_Toc113696931)

[逆元拓展欧几里得 51](#_Toc113696932)

[逆元线性递推 52](#_Toc113696933)

[欧拉函数 52](#_Toc113696934)

[欧拉函数线性筛推荐 52](#_Toc113696935)

[欧拉函数线性筛 53](#_Toc113696936)

[欧拉筛 54](#_Toc113696937)

[整除分块 55](#_Toc113696938)

[组合数推荐 55](#_Toc113696939)

[组合数 56](#_Toc113696940)

[组合数对非质数取模 56](#_Toc113696941)

[组合数简易n≤61 58](#_Toc113696942)

[FFT\_example 58](#_Toc113696943)

[FFT 61](#_Toc113696944)

[图论 63](#_Toc113696945)

[双端队列优化SPFA 63](#_Toc113696946)

[2-sat 65](#_Toc113696947)

[二分图染色推荐 66](#_Toc113696948)

[二分图染色 67](#_Toc113696949)

[拓扑序 68](#_Toc113696950)

[匈牙利算法 69](#_Toc113696951)

[dijkstra推荐 70](#_Toc113696952)

[dijkstra 71](#_Toc113696953)

[floyd 71](#_Toc113696954)

[johnson 72](#_Toc113696955)

[kruskal 74](#_Toc113696956)

[kruskal重构树 75](#_Toc113696957)

[prim推荐 77](#_Toc113696958)

[prim 78](#_Toc113696959)

[spfa板子 79](#_Toc113696960)

[tarjan 80](#_Toc113696961)

[杂 82](#_Toc113696962)

[模拟退火 82](#_Toc113696963)

[玄学clock 83](#_Toc113696964)

[玄学vector 83](#_Toc113696965)

[cdq分治 83](#_Toc113696966)

[字符串 84](#_Toc113696967)

[hash 84](#_Toc113696968)

[kmp 84](#_Toc113696969)

[trie 85](#_Toc113696970)

# 动态规划

## 数位dp

#include <cstdio>

#include <cstring>

#include <iostream>

#include <string>

using namespace std;

typedef long long ll;

int a[20];

int dp[20][2];

int dfs(int pos, int pre, int sta, bool limit) {

if (pos == -1) return 1;

if (!limit && dp[pos][sta] != -1) return dp[pos][sta];

int up = limit ? a[pos] : 9;

int tmp = 0;

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

if (pre == 6 && i == 2) continue;

if (i == 4) continue; //都是保证枚举合法性

tmp += dfs(pos - 1, i, i == 6, limit && i == a[pos]);

}

if (!limit) dp[pos][sta] = tmp;

return tmp;

}

int solve(int x) {

int pos = 0;

while (x) {

a[pos++] = x % 10;

x /= 10;

}

return dfs(pos - 1, -1, 0, true);

}

signed use() {

int le, ri;

// memset(dp,-1,sizeof dp);可优化

while (~scanf("%d%d", &le, &ri) && le + ri) {

memset(dp, -1, sizeof dp);

printf("%d\n", solve(ri) - solve(le - 1));

}

return 0;

}

# 基础

## 二分推荐

const int n = 1000;

//

bool check(int x) {

;

}

// 带不带等号由check函数决定

// 必要条件在右区间的示例如下

// 必要条件在左区间将注释的两行交换

int binsearch(int l = 0, int r = n, int a[]) { //搜索范围l~r

int ans = -1; // 搜索不到结果时的返回值

while (l <= r) {

int mid = (l + r) / 2;

if (check(mid)) {

ans = mid;

l = mid + 1; //

} else {

r = mid - 1; //

}

}

return ans;

}

// 亲情推荐

// upper\_bound(a+1,a+n+1,value) - a 从左往右第一个大于

// lower\_bound(a+1,a+n+1,value) - a 从左往右第一个大于等于

// upper\_bound(a+1,a+n+1,value,greater<int>()) - a 返回单调非增序列从左往右第一个小于value的数的下标

// lower\_bound(a+1,a+n+1,value,greater<int>()) - a 返回单调非增序列从左往右第一个小于等于value的数的下标

## 二分

const int n = 1000;

//

bool check(int x) {

;

}

// 带不带等号由check函数决定，

// if中填什么由必要条件的区间决定

// 必要条件：（一般是一定满足第一个条件，可能满足第二个条件，一定存在答案的区间）

int binsearch(int l = 1, int r = n, int a[]) {

int mid = (l + r) / 2;

// int mid = (l + r + 1) / 2; //必要条件在左

while (l < r) {

if (check(mid)) {

r = mid; //必要条件在右

// l = mid; //必要条件在左

} else {

l = mid + 1; //必要条件在右

// r = mid - 1; //必要条件在左

}

mid = (l + r) / 2;

// mid = (l + r + 1) / 2;//必要条件在左

}

return l;

}

## 归并排序推荐

const int N = 100000;

int a[N], b[N];

int ans = 0;

void merger(int l, int r) {

if (l == r) return;

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

merger(l, mid);

merger(mid + 1, r);

int pl = l, pr = mid + 1;

for (int i = l; i <= r; i++) {

if (pl <= mid && a[pl] <= a[pr] || pr > r) { // 注意是三个条件

b[i] = a[pl++];

} else {

b[i] = a[pr++];

// ans += (mid - pl + 1);求逆序对

}

}

for (int i = l; i <= r; i++)

a[i] = b[i];

}

## 归并排序

// 归并排序（C-迭代版）

#include <bits/stdc++.h>

int min(int x, int y) {

return x < y ? x : y;

}

void merge\_sort(int arr[], int len) {

int\* a = arr;

int\* b = (int\*)malloc(len \* sizeof(int));

int seg, start;

for (seg = 1; seg < len; seg += seg) {

for (start = 0; start < len; start += seg + seg) {

int low = start, mid = min(start + seg, len), high = min(start + seg + seg, len);

int k = low;

int start1 = low, end1 = mid;

int start2 = mid, end2 = high;

while (start1 < end1 && start2 < end2)

b[k++] = a[start1] < a[start2] ? a[start1++] : a[start2++];

while (start1 < end1)

b[k++] = a[start1++];

while (start2 < end2)

b[k++] = a[start2++];

}

int\* temp = a;

a = b;

b = temp;

}

if (a != arr) {

int i;

for (i = 0; i < len; i++)

b[i] = a[i];

b = a;

}

free(b);

}

void test() {

int a[] = {9, 8, 7, 6, 5};

merge\_sort(a, 5);

for (int i = 0; i < 5; i++) {

std::cout << a[i];

}

}

//

//

// 归并排序（C-递归版）

void merge\_sort\_recursive(int arr[], int reg[], int start, int end) {

if (start >= end) return;

int len = end - start, mid = (len >> 1) + start;

int start1 = start, end1 = mid;

int start2 = mid + 1, end2 = end;

merge\_sort\_recursive(arr, reg, start1, end1);

merge\_sort\_recursive(arr, reg, start2, end2);

int k = start;

while (start1 <= end1 && start2 <= end2)

reg[k++] = arr[start1] < arr[start2] ? arr[start1++] : arr[start2++];

while (start1 <= end1)

reg[k++] = arr[start1++];

while (start2 <= end2)

reg[k++] = arr[start2++];

for (k = start; k <= end; k++)

arr[k] = reg[k];

}

void merge\_sort(int arr[], const int len) {

int reg[len];

merge\_sort\_recursive(arr, reg, 0, len - 1);

}

## 快读快写

#include <bits/stdc++.h>

//快读

inline int read() {

int x = 0, f = 1;

char c = getchar();

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

if (c == '-') {

f = -1;

c = getchar();

}

while (c >= '0' && c <= '9') {

x = x \* 10 + c - '0';

c = getchar();

}

return x \* f;

}

//快写

inline void write(int x) {

if (x < 0) {

putchar('-');

x = -x;

}

if (x > 9) write(x / 10);

putchar(x % 10 + '0');

}

## 快排推荐

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e5 + 10;

int a[maxn];

void qsort(int l, int r) //应用二分思想

{

int mid = a[(l + r) / 2]; //中间数

int i = l, j = r;

do {

while (a[i] < mid)

i++; //查找左半部分比中间数大的数

while (a[j] > mid)

j--; //查找右半部分比中间数小的数

if (i <= j) //如果有一组不满足排序条件（左小右大）的数

{

swap(a[i], a[j]); //交换

i++; // 可简写为swap(a[i++],a[j--]);

j--;

}

} while (i <= j); //这里注意要有=

if (l < j) qsort(l, j); //递归搜索左半部分

if (i < r) qsort(i, r); //递归搜索右半部分

}

## 快排教材

//快速排序（从小到大）

void quickSort(int left, int right, int arr[]) {

if (left >= right) return;

int i, j, base, temp;

i = left, j = right;

base = arr[left]; //取最左边的数为基准数

while (i < j) {

while (arr[j] >= base && i < j) //将>= 改成<= 变成从大到小

j--;

while (arr[i] <= base && i < j) //将<= 改成>= 变成从大到小

i++;

if (i < j) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

//基准数归位

arr[left] = arr[i];

arr[i] = base;

quickSort(left, i - 1, arr); //递归左边

quickSort(i + 1, right, arr); //递归右边

}

## 离散化

#include <bits/stdc++.h>

using namespace std;

const int N = 1000000;

int a[N];

int n;

// O(nlogn) a[i]直接被赋值为离散化后的值，原值为discre[a[i]]

int discre[N];

signed discretization() {

cin >> n;

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

cin >> a[i];

discre[i] = a[i];

}

sort(discre + 1, discre + n + 1);

int numnum = unique(discre + 1, discre + n + 1) - discre - 1;

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

a[i] = lower\_bound(discre + 1, discre + numnum + 1, a[i]) - discre;

}

}

## 双下表转存

int encode(int id1, int id2) {

return id1 \* 1000000 + id2;

}

int decode1(int code) {

return code / 1000000;

}

int decode2(int code) {

return code % 1000000;

}

## 桶排序

// #桶排序#

// 所谓的桶排，就是开一个桶cnt,其中cnt[i]?表示数字i的出现次数。排序变得非常简单：

// 时间复杂度O（n）

const int N = 1e7 + 10;

int a[N], cntt[N];

// a为待排序列，cnt为桶

//

void bucketsort(int n, int max\_num) { // max\_num可能出现的最大的数

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

cntt[a[i]]++;

}

int pos = 1;

for (int i = 0; i < max\_num; i++) {

for (int j = 0; j < cntt[i]; j++) {

a[pos++] = i;

}

}

}

## 字符串

// 提取子字符串

// substr() 函数用于从 string 字符串中提取子字符串，它的原型为：

// string substr(size\_t pos = 0, size\_t len = npos) const;

// pos 为要提取的子字符串的起始下标，len 为要提取的子字符串的长度。

//字符串查找

// 1) find() 函数

// find() 函数用于在 string 字符串中查找子字符串出现的位置，它其中的两种原型为：

// size\_t find (const string& str, size\_t pos = 0) const;

// size\_t find(const char\* s, size\_t pos = 0) const;

// 第一个参数为待查找的子字符串，它可以是 string 字符串，也可以是C风格的字符串。第二个参数

// 为开始查找的位置（下标）；如果不指明，则从第0个字符开始查找。

// find() 函数最终返回的是子字符串第一次出现在字符串中的起始下标。

// 2) rfind() 函数

// rfind() 和 find() 很类似，同样是在字符串中查找子字符串，不同的是 find() 函数从第二个参数开始往后查找，

// 而 rfind() 函数则最多查找到第二个参数处，如果到了第二个参数所指定的下标还没有找到子字符串，则返回一个

// 无穷大值4294967295。

## bitset

#include <bits/stdc++.h>

using namespace std;

bitset<100> bs1[40][30]; //相当于申请了一个 [40][30][100]的01数组注意bitset<>中是最后一维的长度

## gcd\_lcm

//最大公约数

int gcd(int a, int b) {

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

}

//最小公倍数

int lcm(int a, int b) {

return a \* b / gcd(a, b);

}

## template\_head\_file

#include <algorithm>

#include <bitset>

#include <cassert>

#include <cctype>

#include <cmath>

#include <cstdlib>

#include <cstring>

#include <ctime>

#include <deque>

#include <iomanip>

#include <iostream>

#include <list>

#include <map>

#include <queue>

#include <random>

#include <set>

#include <stack>

#include <vector>

// headfile

## template

#include <bits/stdc++.h>

#define int long long

// #define int \_\_int128

// #define int unsigned long long

#define double long double // %Lf

#define rep(a, b, c) for (register int a = b; a <= c; a++)

#define rrep(a, b, c) for (register int a = b; a >= c; a--)

#define pb push\_back

using namespace std;

typedef long long ll;

typedef pair<int, int> pii;

typedef unsigned long long ull;

// mt19937 mrand(time(0));

// mt19937\_64 mrand(time(0));

const ll INF = (ll)9e18;

const double PI = acosl(-1);

const int inf = 0x7fffffff;

const int maxn = 1e7 + 10;

const int mod = 1e9 + 7;

const int mod2 = 998244353;

const ull hashmod = 1e18 + 2049;

int n, m, x, y, z, k, t1, t2, op, ans, flagg, cnt, tot;

int a[maxn];

char ch;

struct Node {

int to, next, w, u;

} node[2 \* maxn];

int head[maxn], vis[maxn];

inline void add(int a, int b, int c = 0) {

node[++tot] = {b, head[a], c, a};

head[a] = tot;

}

inline int read() {

int x = 0, w = 1;

char ch = 0;

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

if (ch == '-') w = -1;

ch = getchar();

}

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

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

ch = getchar();

}

return x \* w;

}

inline void write(int x) {

if (x < 0) {

putchar('-');

x = -x;

}

if (x > 9) write(x / 10);

putchar(x % 10 + '0');

}

// cur ctrl+enter Insert line below

// CPH ctrl + Alt + B Run All Test On CPH

signed main() {

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

return 0;

}

# 几何

## 几何数据结构

#include <bits/stdc++.h>

using namespace std;

const int N = 5007, M = 50007, INF = 0x3f3f3f3f;

const double DINF = 1e18, eps = 1e-8;

struct Point {

double x, y;

Point(double x = 0, double y = 0) : x(x), y(y) {} //构造函数

};

//!注意区分点和向量

typedef Point Vector;

//向量平移之后还是那个向量，所以只需要原点和向量的终点即可

//!向量 + 向量 = 向量，点 + 向量 = 向量

Vector operator+(Vector A, Vector B) {

return Vector(A.x + B.x, A.y + B.y);

}

//!点 - 点 = 向量(向量BC = C - B)

Vector operator-(Point A, Point B) {

return Vector(A.x - B.x, A.y - B.y);

}

//!向量 \* 数 = 向量

Vector operator\*(Vector A, double p) {

return Vector(A.x \* p, A.y \* p);

}

//!向量 / 数= 向量

Vector operator/(Vector A, double p) {

return Vector(A.x / p, A.y / p);

}

//!点/向量的比较函数

bool operator<(const Point& a, const Point& b) {

return a.x < b.x || (a.x == b.x && a.y < b.y);

}

//!求极角//在极坐标系中，平面上任何一点到极点的连线和极轴的夹角叫做极角。

//单位弧度rad

double Polar\_angle(Vector A) {

return atan2(A.y, A.x);

}

//!三态函数sgn用于判断相等，减少精度误差问题

int sgn(double x) {

if (fabs(x) < eps) return 0;

if (x < 0) return -1;

return 1;

}

//重载等于运算符

bool operator==(const Point& a, const Point& b) {

return !sgn(a.x - b.x) && !sgn(a.y - b.y);

}

//!点积(满足交换律)

double Dot(Vector A, Vector B) {

return A.x \* B.x + A.y \* B.y;

}

//!向量的叉积(不满足交换律)

//等于两向量有向面积的二倍(从v的方向看,w在左边,叉积>0,w在右边,叉积<0,共线,叉积=0)

// cross(x, y) = -cross(y, x)

// cross(x, y) : xAyB - xByA

double Cross(Vector A, Vector B) {

return A.x \* B.y - B.x \* A.y;

}

## 凸包andrew

//计算凸包，输入点数组p，个数为p输出点数组ch，函数返回凸包顶点个数。

//输入不能有重复的点，函数执行完后的输入点的顺序将被破坏（因为要排序，可以加一个数组存原来的id）

//如果不希望在凸包边上有输入点，把两个<=改成<即可

#include <bits/stdc++.h>

using namespace std;

struct Point {

double x, y;

Point(double x = 0, double y = 0) : x(x), y(y) {} //构造函数

};

typedef Point Vector;

Vector operator-(Point A, Point B) {

return Vector(A.x - B.x, A.y - B.y);

}

double Cross(Vector A, Vector B) {

return A.x \* B.y - B.x \* A.y;

}

// Andrew算法

int ConvexHull(Point p[], int n, Point ch[]) {

sort(p, p + n);

int m = 0;

for (int i = 0; i < n; ++i) { //下凸包

//如果叉积<=0说明新边斜率小说明已经不是凸包边了，赶紧踢走

while (m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0)

m--;

ch[m++] = p[i];

}

int k = m;

for (int i = n - 2; i >= 0; --i) { //上凸包

while (m > k && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0)

m--;

ch[m++] = p[i];

}

if (n > 1) m--;

return m;

}

## 旋转卡壳

#include <bits/stdc++.h>

using namespace std;

const int N = 50007, M = 50007;

Point p[N], con[N];

struct Point {

int x, y;

Point(double x = 0, double y = 0) : x(x), y(y) {} //构造函数

};

typedef Point Vector;

Vector operator-(Point A, Point B) {

return Vector(A.x - B.x, A.y - B.y);

}

bool operator<(const Point& a, Point& b) {

return a.x < b.x || (a.x == b.x && a.y < b.y);

}

double Cross(Vector A, Vector B) {

return A.x \* B.y - B.x \* A.y;

}

//上为所需几何数据结构

//凸包

int ConvexHull(Point\* p /\*所有点的集合\*/, int n /\*全部点的个数\*/, Point\* ch /\*凸包存放\*/) {

sort(p, p + n);

int m = 0;

for (int i = 0; i < n; ++i) { //下凸包

while (m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0)

m--;

ch[m++] = p[i];

}

int k = m;

for (int i = n - 2; i >= 0; --i) { //上凸包

while (m > k && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) <= 0)

m--;

ch[m++] = p[i];

}

if (n > 1) m--;

// ch[0]是起点，最后一个点ch[m]也是起点

return m;

}

//点到原点的距离

int get\_dist(const Point& x) {

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

}

//旋转卡壳 返回直径

double Rotating\_calipers(int con\_num /\*点的个数\*/, Point con[] /\*凸包点集\*/) {

int op = 1, ans = 0;

for (int i = 0; i < con\_num; ++i) {

while (Cross((con[i] - con[op]), (con[i + 1] - con[i])) < Cross((con[i] - con[op + 1]), (con[i + 1] - con[i])))

//（写成<=会被两个点的数据卡掉，所以必须写成<）

op = (op + 1) % con\_num;

ans = max(ans, max(get\_dist(con[i] - con[op]), get\_dist(con[i + 1] - con[op])));

}

cout << ans;

return ans;

}

//返回直径

double use\_rotating(int n /\*全部点的个数\*/, Point p[] /\*所有点的集合\*/) {

int con\_num = ConvexHull(p, n, con);

double res = Rotating\_calipers(con\_num, con);

return res;

}

## geometric

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define double long double

//高精度

typedef long long ll;

const int inf = 0x7fffffff;

const int INF = (ll)9e18;

const double DINF = 12345678910, eps = 1e-10;

const double PI = acosl(-1);

// const long double PI = acosl(-1);

const int maxn = 50007, maxm = 50007;

const int mod = 1e9 + 7;

const int mod2 = 998244353;

struct Point {

int x, y;

Point(double x = 0, double y = 0) : x(x), y(y) {} //构造函数

};

typedef Point Vector;

Vector operator+(Vector A, Vector B) {

return Vector(A.x + B.x, A.y + B.y);

}

Vector operator-(Point A, Point B) {

return Vector(A.x - B.x, A.y - B.y);

}

Vector operator\*(Vector A, double p) {

return Vector(A.x \* p, A.y \* p);

}

Vector operator/(Vector A, double p) {

return Vector(A.x / p, A.y / p);

}

bool operator<(const Point& a, Point& b) {

return a.x < b.x || (a.x == b.x && a.y < b.y);

}

int dcmp(double x) {

if (fabs(x) < eps)

return 0;

else

return x < 0 ? -1 : 1;

}

bool operator==(const Point& a, const Point& b) {

return !dcmp(a.x - b.x) && !dcmp(a.y - b.y);

}

double Polar\_angle(Vector A) {

return atan2(A.y, A.x);

}

inline double D\_to\_R(double D) //角度转弧度

{

return PI / 180 \* D;

}

//右手定则，从第一个转到第二个向上为正向下为负

double Cross(Vector A, Vector B) {

return A.x \* B.y - B.x \* A.y;

}

Vector Rotate(Vector A, double rad) {

return Vector(A.x \* cos(rad) - A.y \* sin(rad), A.x \* sin(rad) + A.y \* cos(rad));

}

Point Get\_line\_intersection(Point P, Vector v, Point Q, Vector w) {

Vector u = P - Q;

double t = Cross(w, u) / Cross(v, w);

return P + v \* t;

}

double convex\_polygon\_area(Point\* p, int n) {

double area = 0;

for (int i = 1; i <= n - 2; ++i)

area += Cross(p[i] - p[0], p[i + 1] - p[0]);

return area / 2;

}

int read() {

int x = 0, w = 1;

char ch = 0;

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

if (ch == '-') w = -1;

ch = getchar();

}

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

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

ch = getchar();

}

return x \* w;

}

Point p[maxn], ch[maxn];

int n, m, x, y, z, k, t1, t2, op, ans, flagg, cnt, tot;

// !!!注意当数值达到一定程度时必须改用longlong而不是double，double含有效位

// 用快捷键，或全局替换将double 换成int (即long long )

signed main() {

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

return 0;

}

# 数据结构

## 并查集

const int N = 1e6 + 10;

int fa[N];

int found(int x) {

if (fa[x] == x)

return x;

else {

int oldFa = fa[x];

fa[x] = found(oldFa);

return fa[x];

}

}

void init(int n) {

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

fa[i] = i;

}

}

## 带权并查集

const int N = 100000;

//

int fa[N], d[N]; // d为到父节点的距离

int found(int x) {

if (fa[x] == x)

return x;

else {

int oldFa = fa[x];

fa[x] = found(oldFa);

d[x] = d[x] + d[oldFa];

return fa[x];

}

}

void merge(int x, int y, int w) { // y比x多w的权

int fax = found(x), fay = found(y);

if (fax == fay) return;

fa[fax] = fay;

d[fax] = -d[x] + d[y] + w;

}

int dist(int x, int y) { // y比x多多少权

int fax = found(x), fay = found(y);

if (fax != fay)

return -1;

else

return d[x] - d[y];

}

void init(int n) {

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

fa[i] = i;

}

}

## 带修莫队

#include <bits/stdc++.h>

using namespace std;

#define rep(a, b, c) for (int a = b; a <= c; a++)

int n, m, x, y, z, k, t1, t2, op, ans, flagg, cnt, tot;

int a[100000];

char ch;

int read() {

return 1;

}

void write(int num) {

;

}

const int N = 100000;

// O((1/4)\*n^(5/3)常数如下可被化为1/4)

int buc[N];

struct Block {

int l, r, id, pos1, pos2, tim, ans;

} block[N];

struct Update {

int id, pos, pre, aft;

} update[N];

// 实测比一般排序快近一倍

bool cmp(Block a, Block b) {

if (a.pos1 == b.pos1) {

if (a.pos2 == b.pos2) {

if ((a.pos2 & 1) ^ (a.pos1 & 1) ^ 1) { return a.tim > b.tim; } // 因为时间序最开始在最后

return a.tim < b.tim;

}

if (a.pos1 & 1) return a.pos2 < b.pos2;

return a.pos2 > b.pos2;

}

return a.pos1 < b.pos1;

}

bool cmp\_bak(Block a, Block b) {

if (a.pos1 == b.pos1) {

if (a.pos2 == b.pos2) { return a.tim < b.tim; }

return a.pos2 < b.pos2;

}

return a.pos1 < b.pos1;

}

bool cmp\_id(Block a, Block b) {

return a.id < b.id;

}

void updatef(int p, int op) {

if (op == 1) {

if (buc[p] == 0) { ans++; }

buc[p]++;

} else {

if (buc[p] == 1) { ans--; }

buc[p]--;

}

}

signed use\_example() {

int cnt1 = 0, cnt2 = 0;

int blen = pow(n, 2.0 / 3.0); // 实测2/3远快于1/2(至少4倍) 1/2还T了

rep(i, 1, m) {

ch = 0;

while (!isupper(ch)) {

scanf("%c", &ch);

}

x = read();

y = read();

if (ch == 'Q') {

block[++cnt1] = {x, y, i, (x - 1) / blen + 1, (y - 1) / blen + 1, cnt2};

} else {

update[++cnt2] = {i, x, a[x], y};

a[x] = y;

}

}

sort(block + 1, block + cnt1 + 1, cmp);

int pl = 1, pr = 0, pt = cnt2;

rep(i, 1, cnt1) {

while (block[i].tim > pt) {

++pt;

if (update[pt].pos >= pl && update[pt].pos <= pr) {

updatef(update[pt].pre, -1);

updatef(update[pt].aft, 1);

}

a[update[pt].pos] = update[pt].aft;

}

while (block[i].tim < pt) {

if (update[pt].pos >= pl && update[pt].pos <= pr) {

updatef(update[pt].aft, -1);

updatef(update[pt].pre, 1);

}

a[update[pt].pos] = update[pt].pre;

pt--;

}

while (block[i].r > pr) {

updatef(a[++pr], 1);

}

while (block[i].r < pr) {

updatef(a[pr--], -1);

}

while (block[i].l < pl) {

updatef(a[--pl], 1);

}

while (block[i].l > pl) {

updatef(a[pl++], -1);

}

block[i].ans = ans;

}

sort(block + 1, block + cnt1 + 1, cmp\_id);

rep(i, 1, cnt1) {

write(block[i].ans);

putchar('\n');

}

return 0;

}

## 分块

#include <bits/stdc++.h>

using namespace std;

int n;

const int N = 100000;

int a[N];

// O(m根n)

int bl[N], br[N], bt[N]; // bl(block left) br(block right) bt(belong to block)

int bval1[N]; // bval1(block value 1)

int btag1[N];

void init() {

int blen = sqrt(n);

int cnt1 = (n + blen - 1) / blen;

for (int i = 1; i <= cnt1; i++) {

bl[i] = (i - 1) \* blen + 1;

br[i] = (i)\*blen;

}

br[cnt1] = n;

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

int t1 = (i - 1) / blen + 1;

bt[i] = t1;

bval1[t1] += a[i];

}

}

void update(int l, int r, int val) {

int pl = bt[l], pr = bt[r];

if (pl == pr) {

for (int i = l; i <= r; i++) {

a[i] += val;

}

} else {

for (int i = pl + 1; i <= pr - 1; i++) {

btag1[i] += val;

}

for (int i = l; i <= br[pl]; i++) {

a[i] += val;

bval1[pl] += val;

}

for (int i = bl[pr]; i <= r; i++) {

a[i] += val;

bval1[pr] += val;

}

}

}

int query(int l, int r) {

int pl = bt[l], pr = bt[r];

int res = 0;

if (pl == pr) {

for (int i = l; i <= r; i++) {

res += a[i] + btag1[pl];

}

return res;

} else {

for (int i = pl + 1; i <= pr - 1; i++) {

res += (br[i] - bl[i] + 1) \* btag1[i] + bval1[i];

}

for (int i = l; i <= br[pl]; i++) {

res += (a[i] + btag1[pl]);

}

for (int i = bl[pr]; i <= r; i++) {

res += (a[i] + btag1[pr]);

}

}

return res;

}

## 回滚莫队

#include <bits/stdc++.h>

#define int long long

// #define int \_\_int128

// #define int unsigned long long

#define double long double // %Lf

#define rep(a, b, c) for (register int a = b; a <= c; a++)

#define rrep(a, b, c) for (register int a = b; a >= c; a--)

#define pb push\_back

using namespace std;

typedef long long ll;

typedef pair<int, int> pii;

typedef unsigned long long ull;

// mt19937 mrand(time(0));

// mt19937\_64 mrand(time(0));

const ll INF = (ll)9e18;

const double PI = acosl(-1);

const int inf = 0x7fffffff;

const int maxn = 1e7 + 10;

const int mod = 1e9 + 7;

const int mod2 = 998244353;

const ull hashmod = 1e18 + 2049;

int n, m, x, y, z, k, t1, t2, op, ans, flagg, cnt, tot;

int a[maxn];

char ch;

struct Node {

int to, next, w, u;

} node[maxn];

int head[maxn], vis[maxn];

inline void add(int a, int b, int c = 0) {

node[++tot] = {b, head[a], c, a};

head[a] = tot;

}

inline int read() {

int x = 0, w = 1;

char ch = 0;

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

if (ch == '-') w = -1;

ch = getchar();

}

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

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

ch = getchar();

}

return x \* w;

}

inline void write(int x) {

if (x < 0) {

putchar('-');

x = -x;

}

if (x > 9) write(x / 10);

putchar(x % 10 + '0');

}

// cur ctrl+enter Insert line below

struct Block {

int l, r, id, pos, ans;

} block[maxn];

bool cmp(Block a, Block b) {

if (a.pos == b.pos) return a.r < b.r;

return a.pos < b.pos;

}

bool cmp\_id(Block a, Block b) {

return a.id < b.id;

}

int buc1[maxn];

int buc2[maxn];

int clear[maxn];

void cal(Block& blocki) {

rep(i, blocki.l, blocki.r) {

buc1[a[i]] = 0;

}

rep(i, blocki.l, blocki.r) {

if (!buc1[a[i]]) {

buc1[a[i]] = i;

} else {

blocki.ans = max(blocki.ans, i - buc1[a[i]]);

}

}

rep(i, blocki.l, blocki.r) {

buc1[a[i]] = 0;

}

}

int discre[maxn];

// CPH ctrl + Alt + B Run All Test On CPH

signed use() {

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

n = read();

rep(i, 1, n) {

discre[i] = a[i] = read();

}

sort(discre + 1, discre + n + 1);

int num1 = unique(discre + 1, discre + n + 1) - discre - 1;

rep(i, 1, n) {

a[i] = lower\_bound(discre + 1, discre + num1 + 1, a[i]) - discre;

}

// rep(i, 1, n) {

// cout << a[i] << ' ';

// }

int blen = sqrt(n - 1) + 1;

m = read();

rep(i, 1, m) {

x = read();

y = read();

block[i] = {x, y, i, (x - 1) / blen + 1};

}

sort(block + 1, block + m + 1, cmp);

int pl = 1, pr = 0;

int cnt1 = 0;

for (int i = 1; i <= m;) {

int br = min(block[i].pos \* blen, n);

pr = br;

int ans = 0;

for (int j = block[i].pos; i <= m && j == block[i].pos; i++) {

pl = br + 1;

if (block[i].r <= br) {

cal(block[i]);

continue;

}

while (block[i].r > pr) {

++pr;

buc2[a[pr]] = pr;

if (buc1[a[pr]] == 0) {

buc1[a[pr]] = pr;

clear[++cnt1] = a[pr];

}

ans = max(ans, pr - buc1[a[pr]]);

}

int t1 = ans;

while (block[i].l < pl) {

--pl;

if (!buc2[a[pl]]) {

buc2[a[pl]] = pl;

} else {

ans = max(buc2[a[pl]] - pl, ans);

}

}

block[i].ans = ans;

while (pl <= br) {

if (buc2[a[pl]] == pl) buc2[a[pl]] = 0;

++pl;

}

ans = t1;

}

rep(i, 1, cnt1) {

buc1[clear[i]] = buc2[clear[i]] = 0;

}

cnt1 = 0;

}

sort(block + 1, block + m + 1, cmp\_id);

rep(i, 1, m) {

write(block[i].ans);

putchar('\n');

}

return 0;

}

## 莫队

#include <bits/stdc++.h>

using namespace std;

#define rep(a, b, c) for (register int a = b; a <= c; a++)

#define rrep(a, b, c) for (register int a = b; a >= c; a--)

int n, m, x, y, z, k, t1, t2, op, ans, flagg, cnt, tot;

const int N = 1000000;

int a[N];

char ch;

inline int read();

inline void write(int x);

// O(1/4\*(n^(3/2)))

int sum;

int buc[N];

struct Block {

int l, r, id, ansup, ansdown, pos;

} block[N];

bool cmp(Block a, Block b) {

if (a.pos == b.pos) {

if (a.pos & 1) return a.r < b.r;

return a.r > b.r;

}

return a.pos < b.pos;

}

bool cmp\_id(Block a, Block b) {

return a.id < b.id;

}

void update(int p, int op) {

p = a[p];

sum -= buc[p] \* buc[p];

buc[p] += op;

sum += buc[p] \* buc[p];

}

// CPH ctrl + Alt + B Run All Test On CPH

signed use() {

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

n = read();

m = read();

rep(i, 1, n) {

a[i] = read();

}

int blen = sqrt(n - 1) + 1;

rep(i, 1, m) {

block[i].l = read();

block[i].r = read();

block[i].id = i;

block[i].pos = (block[i].l - 1) / blen + 1;

}

sort(block + 1, block + m + 1, cmp);

int pl = 1, pr = 0;

rep(i, 1, m) {

while (block[i].r > pr) {

update(++pr, 1);

}

while (block[i].r < pr) {

update(pr--, -1);

}

while (block[i].l < pl) {

update(--pl, 1);

}

while (block[i].l > pl) {

update(pl++, -1);

}

if (block[i].l == block[i].r) {

block[i].ansdown = 1;

continue;

}

block[i].ansdown = (block[i].r - block[i].l + 1) \* (block[i].r - block[i].l);

block[i].ansup = sum - (block[i].r - block[i].l + 1);

int gcd = \_\_gcd(block[i].ansup, block[i].ansdown);

block[i].ansup /= gcd;

block[i].ansdown /= gcd;

}

sort(block + 1, block + m + 1, cmp\_id);

rep(i, 1, m) {

write(block[i].ansup);

putchar('/');

write(block[i].ansdown);

putchar('\n');

}

return 0;

}

## 树状数组

// # 树状数组 #

#include <bits/stdc++.h>

using namespace std;

#define lowbit(x) ((x) & (-x)) // 返回所属最小线段长度

const int N = 1e5 + 10;

int n;

int tree[N];

inline void update(int pos, int x) { //单点pos增加x pos!=0 否则会段错误

for (int i = pos; i <= n; i += lowbit(i)) {

tree[i] += x;

}

}

inline int query(int pos) { //求前pos项和 pos=0时返回0

int ans = 0;

for (int i = pos; i; i -= lowbit(i)) {

ans += tree[i];

}

return ans;

}

int queryi(int sum, int n = N) { // 相当于查询sum[pos]<=sum的最大位置，n为数组中数的个数;

// 替换注释修改为sum[pos]<=sum的第一次最大位置，即不计a[i]==0的位置

int pos = 0;

int now = 0;

for (int i = 31ll; i >= 0; --i) {

pos += 1ll << i;

// if (pos > n || tree[pos] + now >= sum)//第一次最大位置

if (pos > n || tree[pos] + now > sum) //最终最大位置

pos -= 1ll << i;

else

now += tree[pos];

}

// return pos + 1;//第一次最大位置

return pos; //最终最大位置

}

// 单点更新最大值 或最小值

void updatemax(int x, int val) {

while (x <= n) {

tree[x] = max(tree[x], val);

x += lowbit(x);

}

}

// 查询前缀最大值 或最小值

int querymax(int x) {

int t = 0;

while (x) {

t = max(t, tree[x]);

x -= lowbit(x);

}

return t;

}

## 线段树

#include <bits/stdc++.h>

using namespace std;

int a[1000000];

int n;

//

const int N = 1e7 + 10; // N建议为4\*n

struct Segm {

int value1, tag1;

} segm[N];

void push\_up(int p) {

segm[p].value1 = segm[p \* 2].value1 + segm[p \* 2 + 1].value1;

}

void push\_downf(int p, int l, int r, int tag1) {

segm[p].tag1 += tag1;

segm[p].value1 += tag1 \* (r - l + 1);

}

void push\_down(int p, int l, int r) {

int mid = (l + r) / 2;

push\_downf(p \* 2, l, mid, segm[p].tag1);

push\_downf(p \* 2 + 1, mid + 1, r, segm[p].tag1);

segm[p].tag1 = 0;

}

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

if (l == r) {

segm[p].value1 = a[l];

return;

}

int mid = (l + r) / 2;

build(p \* 2, l, mid);

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

push\_up(p);

}

void update(int ql, int qr, int p, int l, int r, int value1) {

if (ql <= l && qr >= r) {

segm[p].tag1 += value1;

segm[p].value1 += (r - l + 1) \* value1;

return;

}

int mid = (l + r) / 2;

push\_down(p, l, r);

if (ql <= mid) update(ql, qr, p \* 2, l, mid, value1);

if (qr > mid) update(ql, qr, p \* 2 + 1, mid + 1, r, value1);

push\_up(p);

}

int query(int ql, int qr, int p, int l, int r) {

int res = 0;

if (ql <= l && qr >= r) { return segm[p].value1; }

int mid = (l + r) / 2;

push\_down(p, l, r);

if (ql <= mid) { res += query(ql, qr, p \* 2, l, mid); }

if (qr > mid) { res += query(ql, qr, p \* 2 + 1, mid + 1, r); }

return res;

}

## lca推荐

#include <bits/stdc++.h>

using namespace std;

const int N = 1e7 + 5;

struct Node {

int to, next, w, u;

} node[N];

int head[N], tot;

void add(int u, int v, int w = 0) {

node[++tot] = {v, head[u], w, u};

head[u] = tot;

}

//

int st[N][\_\_lg(N \* 2)], depth[N];

void dfs(int u, int fa) {

depth[u] = depth[fa] + 1;

st[u][0] = fa;

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

int to = node[i].to;

if (to == fa) continue;

dfs(to, u);

}

}

int lca(int u, int v) {

if (depth[u] < depth[v]) // 确保 u 的深度更大

swap(u, v);

while (depth[u] > depth[v]) {

u = st[u][\_\_lg(depth[u] - depth[v])];

}

if (u == v) return u;

for (int i = \_\_lg(depth[u]); i >= 0; i--) {

if (st[u][i] != st[v][i]) {

u = st[u][i];

v = st[v][i];

}

}

return st[u][0];

}

int init(int n, int root) {

dfs(root, root);

for (int j = 1; j <= \_\_lg(n); j++)

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

st[i][j] = st[st[i][j - 1]][j - 1];

}

## lca欧拉序版

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

typedef pair<int, int> pii;

const int inf = 0x7fffffff;

struct Node {

int to, next, w, u;

} node[N];

int head[N], vis[N];

int n, m;

// O(nlogn) 预处理 O(1)查询

const int LOGN = \_\_lg(N \* 4);

int root, a[N], deepth[N];

int f[N \* 2], tot, pos[N];

struct LCA\_ST {

int id;

bool operator<(const LCA\_ST& t) const { return deepth[id] < deepth[t.id]; }

} st[N \* 2][LOGN];

inline int LCA(int u, int v) {

u = pos[u], v = pos[v];

if (u > v) swap(u, v);

int k = \_\_lg(v - u + 1);

return min(st[u][k], st[v - (1 << k) + 1][k]).id;

}

inline void dfs(int now, int fa) {

deepth[now] = deepth[fa] + 1;

f[++tot] = now;

pos[now] = tot;

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

int to = node[i].to;

if (to == fa) continue;

dfs(to, now);

f[++tot] = now;

}

}

void init() {

dfs(root, root);

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

st[i][0].id = f[i];

}

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

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

st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);

}

## st表

#include <bits/stdc++.h>

using namespace std;

const int N = 1000000;

// init时间复杂度（nlogn）查询时间复杂度O1

int st[N][\_\_lg(N) + 1]; // N为要查询的最大范围

// \_\_lg(n)返回log2(n)向向下取整,最高位的位数减一,可理解为将n最高位移到&1需要移动的位数

// 在不使用第0层时st表不会访问第0层，当然st表可以使用第0层

// 使用st(i, j) 来表示区间[i, i + 2 ^ j - 1] 中的最值

void init\_st(int a[], int n) {

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

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

}

for (int j = 1; j <= \_\_lg(n); j++) { //\_\_lg(len)len为要查询的范围

for (int i = 1; i + (1 << j - 1) <= n; i++) { // 防RE,多开30列可直接写n,标准写法i + (1 << j) - 1 <= n

st[i][j] = max(st[i][j - 1], st[i + (1 << j - 1)][j - 1]); // rmq(range max/min query)问题的st表

// st[i][j] = st[st[i][j - 1]][j - 1]; //路径倍增st表

}

}

}

int queryRMQst(int l, int r) {

int k = \_\_lg(r - l + 1);

return max(st[l][k], st[r - (1 << k) + 1][k]);

}

int queryRoadSt(int x, int k) {

int i = \_\_lg(k); // 也行, 加上 if (k != 0) 即可

if (k != 0)

for (int i = \_\_lg(k); i >= 0; i--)

if (x & (1 << i)) x = st[x][i];

return x;

}

# 数学

## 杜教筛

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

// const int N = 5e6 + 10; // n^2/3

const int N = 1665703; // n^2/3

int primes[N + 7];

ll mu[N + 7];

bool vis[N + 7];

ll phi[N + 7];

unordered\_map<ll, ll> sum\_mu; //数组开不了那么大，所以用哈希表

unordered\_map<ll, ll> sum\_phi;

inline void init(int n = N) {

vis[0] = vis[1] = 1;

mu[1] = phi[1] = 1;

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

if (!vis[i]) {

primes[++primes[0]] = i;

mu[i] = -1;

phi[i] = i - 1;

}

for (int j = 1; j <= primes[0] && i \* primes[j] <= n; ++j) {

vis[i \* primes[j]] = 1;

if (i % primes[j] == 0) {

mu[i \* primes[j]] = 0;

phi[i \* primes[j]] = phi[i] \* primes[j];

break;

} else {

mu[i \* primes[j]] = -mu[i];

phi[i \* primes[j]] = phi[i] \* phi[primes[j]];

}

}

}

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

mu[i] += mu[i - 1];

phi[i] += phi[i - 1];

}

}

inline int g\_sum(int x) // g的前缀和，这里的g = I(x)//常数函数

{

return x;

}

inline int get\_sum\_mu(int x) // 记忆化搜索

{

if (x <= N) return mu[x]; //预处理

// if (sum\_mu.find(x) != sum\_mu.end()) return sum\_mu[x]; //记忆化

if (sum\_mu[x]) return sum\_mu[x];

int ans = 1; // 杜教筛中推出的1

for (ll l = 2, r; l <= x; l = r + 1) { // 整除分块

r = x / (x / l);

//Σ\_i=2 {g(i)\*S(?n/i?)} g不一样, S一样，然后整除分块

ans -= (g\_sum(r) - g\_sum(l - 1)) \* get\_sum\_mu(x / l);

}

return sum\_mu[x] = ans / g\_sum(1); // 最后除以g(1)

}

inline ll get\_sum\_phi(int x) {

if (x <= N) return phi[x];

// if(sum\_phi.find(x) != sum\_phi.end()) return sum\_phi[x];

if (sum\_phi[x]) return sum\_phi[x];

ll ans = x \* ((ll)x + 1) / 2; //杜教筛中的 n(n + 1) / 2

for (ll l = 2, r; l <= x; l = r + 1) {

r = x / (x / l);

ans -= 1ll \* (g\_sum(r) - g\_sum(l - 1)) \* get\_sum\_phi(x / l);

}

return sum\_phi[x] = ans / g\_sum(1);

}

//使用前init();即可

## 分解质因数大数

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

map<ll, int> m\_pollard; // map中第一个键是因子，值是次数

const int mod = 10000019;

const int times = 50; //测试50次

ll mul(ll a, ll b, ll m) //求a\*b%m

{

ll ans = 0;

a %= m;

while (b) {

if (b & 1) ans = (ans + a) % m;

b /= 2;

a = (a + a) % m;

}

return ans;

}

ll pow(ll a, ll b, ll m) // a^b % m

{

ll ans = 1;

a %= m;

while (b) {

if (b & 1) ans = mul(a, ans, m);

b /= 2;

a = mul(a, a, m);

}

ans %= m;

return ans;

}

bool Miller\_Rabin(ll n, int repeat) // n是测试的大数，repeat是测试重复次数

{

if (n == 2 || n == 3) return true; //特判

if (n % 2 == 0 || n == 1) return false; //偶数和1

//将n-1分解成2^s\*d

ll d = n - 1;

int s = 0;

while (!(d & 1))

++s, d >>= 1;

// srand((unsigned)time(NULL));在最开始调用即可

for (int i = 0; i < repeat; i++) //重复repeat次

{

ll a = rand() % (n - 3) + 2; //取一个随机数,[2,n-1)

ll x = pow(a, d, n);

ll y = 0;

for (int j = 0; j < s; j++) {

y = mul(x, x, n);

if (y == 1 && x != 1 && x != (n - 1)) return false;

x = y;

}

if (y != 1) return false; //费马小定理

}

return true;

}

ll gcd(ll a, ll b) {

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

}

ll pollard\_rho(ll n, ll c) //找到n的一个因子

{

ll x = rand() % (n - 2) + 1;

ll y = x, i = 1, k = 2;

while (1) {

i++;

x = (mul(x, x, n) + c) + n; //不断调整x2

ll d = gcd(y - x, n);

if (1 < d && d < n) return d; //找到因子

if (y == x) return n; //找到循环，返回n，重新来

if (i == k) //一个优化

{

y = x;

k <<= 1;

}

}

}

void Find(ll n, ll c) {

if (n == 1) return; //递归出口

if (Miller\_Rabin(n, times)) //如果是素数，就加入

{

m\_pollard[n]++;

return;

}

ll p = n;

while (p >= n)

p = pollard\_rho(p, c--); //不断找因子，知道找到为止，返回n说明没找到

Find(p, c);

Find(n / p, c);

}

// O(n^(1/4))

int use\_find(ll n) { //将因子存如m（map）中

srand((unsigned)time(NULL));

m\_pollard.clear();

Find(n, rand() % (n - 1) + 1); //这是自己设置的一个数

return 0;

}

## 分解质因数小数

#include <bits/stdc++.h>

using namespace std;

int cnt[100000];

vector<int> prime\_factors;

void get\_prime\_factors(int x) {

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

if (x % i == 0) {

prime\_factors.emplace\_back(i);

while (x % i == 0) {

x /= i;

++cnt[i];

}

}

if (x > 1) {

prime\_factors.emplace\_back(x);

++cnt[x];

}

}

}

## 高精度推荐

#include <bits/stdc++.h>

using namespace std;

const int mlen = 100;

void show(int a[]) {

int onshow = 0;

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

if (a[i] != 0) onshow = 1;

if (onshow) cout << a[i];

}

if (onshow == 0) cout << 0;

cout << endl;

}

void equ(int a[], int x) {

memset(a, 0, (mlen + 1) \* sizeof(int));

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

a[i] = (x % 10);

x /= 10;

}

}

void add(int a[], int s[]) //高精求和s+=a

{

int g = 0;

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

s[i] = s[i] + a[i] + g;

g = s[i] / 10;

s[i] = s[i] % 10;

}

}

void mul(int a[], int x) //高精求积

{

int g = 0;

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

a[i] = a[i] \* x + g;

g = a[i] / 10;

a[i] = a[i] % 10;

}

}

## 高精度

#include <cstdio>

#include <cstring>

#include <iostream>

#include <string>

using namespace std;

const int maxn = 1000;

struct bign {

int d[maxn], len;

void clean() {

while (len > 1 && !d[len - 1])

len--;

}

bign() {

memset(d, 0, sizeof(d));

len = 1;

}

bign(int num) { \*this = num; }

bign(char\* num) { \*this = num; }

bign operator=(const char\* num) {

memset(d, 0, sizeof(d));

len = strlen(num);

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

d[i] = num[len - 1 - i] - '0';

clean();

return \*this;

}

bign operator=(int num) {

char s[20];

sprintf(s, "%d", num);

\*this = s;

return \*this;

}

bign operator+(const bign& b) {

bign c = \*this;

int i;

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

c.d[i] += b.d[i];

if (c.d[i] > 9) c.d[i] %= 10, c.d[i + 1]++;

}

while (c.d[i] > 9)

c.d[i++] %= 10, c.d[i]++;

c.len = max(len, b.len);

if (c.d[i] && c.len <= i) c.len = i + 1;

return c;

}

// 两个都可以用

// bign operator+(const bign& b) const {

// bign c;

// c.len = 0;

// for (int i = 0, g = 0; g || i < max(len, b.len); i++) {

// int x = g;

// if (i < len) x += d[i];

// if (i < b.len) x += b.d[i];

// c.d[c.len++] = x % 10;

// g = x / 10;

// }

// return c;

// }

bign operator-(const bign& b) {

bign c = \*this;

int i;

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

c.d[i] -= b.d[i];

if (c.d[i] < 0) c.d[i] += 10, c.d[i + 1]--;

}

while (c.d[i] < 0)

c.d[i++] += 10, c.d[i]--;

c.clean();

return c;

}

bign operator\*(const bign& b) const {

int i, j;

bign c;

c.len = len + b.len;

for (j = 0; j < b.len; j++)

for (i = 0; i < len; i++)

c.d[i + j] += d[i] \* b.d[j];

for (i = 0; i < c.len - 1; i++)

c.d[i + 1] += c.d[i] / 10, c.d[i] %= 10;

c.clean();

return c;

}

bign operator/(const bign& b) {

int i, j;

bign c = \*this, a = 0;

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

a = a \* 10 + d[i];

for (j = 0; j < 10; j++)

if (a < b \* (j + 1)) break;

c.d[i] = j;

a = a - b \* j;

}

c.clean();

return c;

}

bign operator%(const bign& b) {

int i, j;

bign a = 0;

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

a = a \* 10 + d[i];

for (j = 0; j < 10; j++)

if (a < b \* (j + 1)) break;

a = a - b \* j;

}

return a;

}

bign operator+=(const bign& b) {

\*this = \*this + b;

return \*this;

}

bool operator<(const bign& b) const {

if (len != b.len) return len < b.len;

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

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

return false;

}

bool operator>(const bign& b) const { return b < \*this; }

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

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

bool operator!=(const bign& b) const { return b < \*this || \*this < b; }

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

string str() const {

char s[maxn] = {};

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

s[len - 1 - i] = d[i] + '0';

return s;

}

};

istream& operator>>(istream& in, bign& x) {

string s;

in >> s;

x = s.c\_str();

return in;

}

ostream& operator<<(ostream& out, const bign& x) {

out << x.str();

return out;

}

int main() {

bign a = "34626234513452624", b = "34626234513452624";

cout << a + b;

printf("\nHello World");

}

## 高精度我写版

#include <bits/stdc++.h>

using namespace std;

#define int long long

const int N = 1e6 + 10;

int example\_a[N];

//

const int mlen = 100;

void show(int a[]) {

int onshow = 0;

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

if (a[i] != 0) onshow = 1;

if (onshow) cout << a[i];

}

if (onshow == 0) cout << 0;

cout << endl;

}

void mul(int a[], int x) //高精求积

{

int g = 0;

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

a[i] = a[i] \* x + g;

g = a[i] / 10;

a[i] = a[i] % 10;

}

}

void add(int a[], int s[]) //高精求和s+=a

{

int g = 0;

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

s[i] = s[i] + a[i] + g;

g = s[i] / 10;

s[i] = s[i] % 10;

}

}

void equ(int a[], int x) {

memset(a, 0, (mlen + 1) \* sizeof(int));

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

a[i] = (x % 10);

x /= 10;

}

}

void equ(int a[], string s1) {

memset(a, 0, (mlen + 1) \* sizeof(int));

int slen = s1.length();

for (int i = mlen; i > mlen - slen; i--) {

a[i] = s1[slen - (mlen - i) - 1] - '0';

}

}

void mul(int a[], int m[], int res[]) { // m\*a=res;

int temp[mlen + 1];

memset(res, 0, (mlen + 1) \* sizeof(int));

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

memset(temp, 0, (mlen + 1) \* sizeof(int));

int move = mlen - i;

for (int j = move; j <= mlen + move; j++) {

temp[j - move] = a[j];

}

mul(temp, m[i]);

add(temp, res);

}

}

## 卡特兰数推荐

#include <bits/stdc++.h>

using namespace std;

long long ans1, ans2;

// 最多30位卡特兰数含30

int C(int a, int b) { //简易版C

int mul = 1;

int dev = 1;

for (int i = a - b + 1, j = 1; i <= a; i++, j++) {

mul \*= i;

dev \*= j;

int dev1 = \_\_gcd(mul, dev);

mul /= dev1;

dev /= dev1;

}

return mul;

}

int catalan(int n) {

return C(2 \* n, n) - C(2 \* n, n - 1);

}

## 卡特兰数1

//记忆化搜索/递归 做法

#include <cstdio>

#define MAX\_N 20

#define ll long long

using namespace std;

//

int n;

ll f[MAX\_N][MAX\_N];

ll dfs(int i, int j) {

if (f[i][j]) return f[i][j];

if (i == 0) return 1; //边界

if (j > 0) f[i][j] += dfs(i, j - 1);

f[i][j] += dfs(i - 1, j + 1);

return f[i][j];

}

int catalan\_1(int n) {

return dfs(n, 0);

}

//递归转递推 递推做法

#include <cstdio>

#define MAX\_N 20

#define ll long long

using namespace std;

//

int n;

ll f[MAX\_N][MAX\_N];

int catalan\_2(int n) {

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

f[0][i] = 1;

}

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

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

if (i == j)

f[i][j] = f[i - 1][j];

else

f[i][j] = f[i][j - 1] + f[i - 1][j];

}

}

return f[n][n];

}

## 卡特兰数2

//记忆化搜索/递归 做法

#include <cstdio>

#define MAX\_N 20

#define ll long long

using namespace std;

//

int n;

ll f[MAX\_N][MAX\_N];

ll dfs(int i, int j) {

if (f[i][j]) return f[i][j];

if (i == 0) return 1; //边界

if (j > 0) f[i][j] += dfs(i, j - 1);

f[i][j] += dfs(i - 1, j + 1);

return f[i][j];

}

int catalan\_1(int n) {

return dfs(n, 0);

}

//递归转递推 递推做法

#include <cstdio>

#define MAX\_N 20

#define ll long long

using namespace std;

//

int n;

ll f[MAX\_N][MAX\_N];

int catalan\_2(int n) {

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

f[0][i] = 1;

}

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

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

if (i == j)

f[i][j] = f[i - 1][j];

else

f[i][j] = f[i][j - 1] + f[i - 1][j];

}

}

return f[n][n];

}

## 卡特兰数3

#include <cstdio>

#include <cstring> //为了NOIP不用万能头文件

#include <iostream>

using namespace std;

//

int f[20][20]; //数据就给到18，开个20算大方的

int n;

int catalan(int n) {

memset(f, 0, sizeof(f));

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

f[i][0] = 1; //边界一定要有

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

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

//我们要推f[0][n]，所以i要从零开始跑

{

if (i >= 1) f[i][j] = f[i - 1][j] + f[i + 1][j - 1];

if (i == 0) //栈内没有东西

f[i][j] = f[i + 1][j - 1];

}

return f[0][n];

}

## 快速幂

const int mod = 1e9 + 7;

long long qpow(long long a, long long b, long long p = mod) {

long long res = 1;

while (b) {

if (b & 1) res = res \* a % p;

a = a \* a % p;

b >>= 1;

}

return res;

}

## 米勒拉宾素性检验

#include <algorithm>

#include <cstdio>

#include <ctime>

#include <iostream>

#include <map>

using namespace std;

const int maxn = 1e5 + 10;

typedef long long ll;

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

ll ans = 0;

a %= m;

while (b) {

if (b & 1) ans = (ans + a) % m;

b /= 2;

a = (a + a) % m;

}

return ans;

}

ll pow(ll a, ll b, ll m) {

ll ans = 1;

a %= m;

while (b) {

if (b & 1) ans = mul(a, ans, m);

b /= 2;

a = mul(a, a, m);

}

ans %= m;

return ans;

}

bool Miller\_Rabin(ll n, int repeat = 100) {

if (n == 2 || n == 3) return true;

if (n % 2 == 0 || n == 1) return false;

ll d = n - 1;

int s = 0;

while (!(d & 1))

s++, d >>= 1;

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

ll a = rand() % (n - 3) + 2;

ll x = pow(a, d, n);

ll y = 0;

for (int j = 0; j < s; j++) {

y = mul(x, x, n);

if (y == 1 && x != 1 && x != (n - 1)) return false;

x = y;

}

if (y != 1) return false;

}

return true;

}

bool use\_miller(int n) { // k log^3(n)

srand((unsigned)time(NULL));

if (Miller\_Rabin(n, 100)) return true;

return false;

}

## 莫比乌斯函数

int const mu\_N = 1e7 + 10;

bool isprime[mu\_N] = {1, 1}; // 判断i是否为素数,i=0,i=1的时候都不是质数 ，所以直接标记

int prime[mu\_N]; //存质数

int mu[mu\_N];

int mu\_f[mu\_N];

void get\_mu(long long n) {

mu[1] = 1; // 存放 莫比乌斯函数；

// isprime[] 存放 是否是质数

// prime[] 存放 质数

int cnt = 0;

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

if (!isprime[i]) {

prime[++cnt] = i;

mu[i] = -1;

}

for (int j = 1; j <= cnt && i \* prime[j] <= n; j++) {

isprime[i \* prime[j]] = 1;

if (i % prime[j] == 0) {

mu[i \* prime[j]] = 0;

break;

} //也可以直接break 因为里面本来存的就是0

else

mu[i \* prime[j]] = -mu[i];

}

}

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

mu\_f[i] = mu\_f[i - 1] + mu[i];

}

}

## 逆序数

#include <bits/stdc++.h>

using namespace std;

const int N = 5e5 + 5;

int sum0[N], reflect[N];

int lowbit(int x) {

return x & (-x);

}

void update(int x, int n) {

while (x <= n) {

sum0[x] += 1;

x += lowbit(x);

}

}

int getsum(int x) {

int sum = 0;

while (x > 0) {

sum += sum0[x];

x -= lowbit(x);

}

return sum;

}

int nixu(int a[], int n) {

vector<pair<int, int>> v1;

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

v1.push\_back({a[i], i});

}

sort(v1.begin(), v1.end());

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

reflect[v1[i - 1].second] = i; //离散化

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

sum0[i] = 0; //初始化树状数组

long long ans = 0;

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

update(reflect[i], n);

ans += i - getsum(reflect[i]);

}

return ans;

}

## 逆元费马小定理

// O(log(n))

//就是快速幂且多数情况下比拓展欧几里得慢一个常数且a和mod互质

int mod;

// a\*x==1(mod p) --> x=a^(p-2)mod p

int qpow(int a, int b = mod - 2, int mod) {

int ans = 1;

a %= mod;

while (b) {

if (b & 1) ans = ans \* a % mod;

a = a \* a % mod;

b >>= 1;

}

return ans % mod;

}

## 逆元拓展欧几里得

// O(log(n))

//拓展欧几里得x为a的逆元b为模,x,y不用初始化什么值都可以

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

if (!b)

x = 1, y = 0;

else

Exgcd(b, a % b, y, x), y -= a / b \* x;

}

void use() {

int x, y, a, mod;

Exgcd(a, mod, x, y);

x = (x % mod + mod) % mod;

}

## 逆元线性递推

// O(n)

const int niyuan\_N = 3e6 + 20;

int inv[niyuan\_N] = {0, 1};

//线性递推

int niyuan(int n, int mod) {

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

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

return 0;

}

## 欧拉函数

// 1?N 中与 N 互质的数的个数，被称为欧拉函数

//时间复杂度为根号n

inline int euler\_one(int n) {

int ans = n;

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

if (n % i == 0) {

ans = ans / i \* (i - 1);

while (n % i == 0)

n /= i;

}

}

if (n > 1) ans = ans / n \* (n - 1); // n至多有一个比根号n大的质因子

return ans;

}

## 欧拉函数线性筛推荐

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

// 时间复杂度为两者之中的最大值max（根号r，64\*len）所求phi的区间长度的64倍

signed prime[N], is\_prime[N], tot;

void ola(int n) {

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

is\_prime[i] = 1;

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

if (is\_prime[i] == 1) prime[++tot] = i;

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

if (i \* prime[j] > n) break;

is\_prime[i \* prime[j]] = 0;

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

}

}

}

int nums[N], phi[N]; // num存数相当于n,phi存phi（id）值为了防MLE采用id=l+i存储

signed use(int l, int r) {

int sqrr = (int)(sqrt(r));

ola(sqrr);

for (int i = l; i <= r; i++) {

int id = i - l;

nums[id] = i;

phi[id] = i;

}

// 我们可以瞬间找到含有某个质数的数，却不能瞬间找到一个数含有哪些质数，于是我们用质数去找数

// 一个longlong范围内的数最多由64个质数相乘构成 于是我们的时间复杂度小于64\*区间长度

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

int p = prime[i];

int mint = (l + p - 1) / p; // 最小倍数min times

int maxt = r / p;

for (int j = mint;; maxt) {

int num = p \* j - l;

if (nums[num] % p == 0) {

phi[num] = phi[num] / p \* (p - 1);

while (nums[num] % p == 0) {

nums[num] /= p;

}

}

}

}

for (int i = l; i <= r; i++) {

int id = i - l;

if (nums[id] > 1) { phi[id] = phi[id] / nums[id] \* (nums[id] - 1); }

}

}

## 欧拉函数线性筛

const int N = 1000000;

int is\_prime[N], prime[N], tot, phi[N];

void euler\_phi(int n) {

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

is\_prime[i] = 1;

}

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

if (is\_prime[i]) {

prime[++tot] = i;

phi[i] = i - 1;

}

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

if (i \* prime[j] > n) break;

is\_prime[i \* prime[j]] = 0;

if (i % prime[j] == 0) {

phi[i \* prime[j]] = phi[i] \* prime[j]; //若p|m则mp与p质因子种类相同

break;

} else {

phi[i \* prime[j]] = phi[i] \* phi[prime[j]]; // 积性函数

}

}

}

}

## 欧拉筛

#include <bits/stdc++.h>

using namespace std;

#define int long long

int const ola\_N = 1e7 + 10;

int const ola\_maxn = 1e7 + 10;

bool is\_prime[ola\_N]; // 判断i是否为素数,i=0,i=1的时候都不是质数 ，所以直接标记

int prime[ola\_N]; // 存质数

int tot; // n>100000时tot<n/10

//欧拉筛时间复杂度O(n)每个合数只被其最小质因子筛去

void ola(int n) {

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

is\_prime[i] = 1;

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

if (is\_prime[i] == 1) prime[++tot] = i;

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

if (i \* prime[j] > n) break;

is\_prime[i \* prime[j]] = 0;

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

}

}

}

## 整除分块

//整除分块 复杂度O 根n

long long fenkuai(long long n) {

long long ans = 0;

for (long long l = 1, r; l <= n; l = r + 1) {

r = n / (n / l);

ans += (r - l + 1) \* (n / l);

}

return ans;

}

## 组合数推荐

#include <bits/stdc++.h>

using namespace std;

#define int long long

const int mod = 1e7 + 7;

int qpow(int a, int b, int p = mod) {

int res = 1;

while (b) {

if (b & 1) res = res \* a % p;

a = a \* a % p;

b >>= 1;

}

return res;

}

//求组合数

int C(int a, int b, int p) {

int res = 1;

for (int i = 1, j = a; i <= b; i++, j--)

res = res \* j % p \* qpow(i, p - 2, p) % p;

return res;

}

//计算对质数取模大范围组合数 /\*\*\*调用这个\*\*\*/

int lucas(int a, int b, int p = mod) {

if (a < p && b < p) return C(a, b, p);

return C(a % p, b % p, p) \* lucas(a / p, b / p, p) % p;

}

## 组合数

#include <bits/stdc++.h>

using namespace std;

#define int long long

const int mod = 1e7 + 7;

int qpow(int a, int b, int p = mod) {

int res = 1;

while (b) {

if (b & 1) res = res \* a % p;

a = a \* a % p;

b >>= 1;

}

return res;

}

//计算int范围组合数

int C(int a, int b, int p = mod) {

if (a < b) return 0;

int down = 1, up = 1;

for (int i = a, j = 1; j <= b; i--, ++j) {

up = up \* i % p;

down = down \* j % p;

}

return up \* qpow(down, p - 2, p) % p;

}

//计算对质数取模大范围组合数 /\*\*\*调用这个\*\*\*/

int lucas(int a, int b, int p = mod) {

if (a < p && b < p) return C(a, b, p);

return C(a % p, b % p, p) \* lucas(a / p, b / p, p) % p;

}

## 组合数对非质数取模

#include <bits/stdc++.h>

using namespace std;

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

if (!b) return (void)(x = 1, y = 0);

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

long long tmp = x;

x = y;

y = tmp - a / b \* y;

}

inline long long INV(long long a, long long p) {

long long x, y;

exgcd(a, p, x, y);

return (x + p) % p;

}

inline long long fast\_pow(long long a, long long b, long long p) {

long long t = 1;

a %= p;

while (b) {

if (b & 1LL) t = (t \* a) % p;

b >>= 1LL;

a = (a \* a) % p;

}

return t;

}

inline long long F(long long n, long long P, long long PK) {

if (n == 0) return 1;

long long rou = 1; //循环节

long long rem = 1; //余项

for (long long i = 1; i <= PK; i++) {

if (i % P) rou = rou \* i % PK;

}

rou = fast\_pow(rou, n / PK, PK);

for (long long i = PK \* (n / PK); i <= n; i++) {

if (i % P) rem = rem \* (i % PK) % PK;

}

return F(n / P, P, PK) \* rou % PK \* rem % PK;

}

inline long long G(long long n, long long P) {

if (n < P) return 0;

return G(n / P, P) + (n / P);

}

inline long long C\_PK(long long n, long long m, long long P, long long PK) {

long long fz = F(n, P, PK), fm1 = INV(F(m, P, PK), PK), fm2 = INV(F(n - m, P, PK), PK);

long long mi = fast\_pow(P, G(n, P) - G(m, P) - G(n - m, P), PK);

return fz \* fm1 % PK \* fm2 % PK \* mi % PK;

}

long long A[1001], B[1001];

// x=B(mod A)

//扩展卢卡斯定理 /\*\*\*调用这个\*\*\*/

inline long long C(long long n, long long m, long long P) {

long long ljc = P, tot = 0;

for (long long tmp = 2; tmp \* tmp <= P; tmp++) {

if (!(ljc % tmp)) {

long long PK = 1; // 分解为质数p的k次幂

while (!(ljc % tmp)) {

PK \*= tmp;

ljc /= tmp;

}

A[++tot] = PK;

B[tot] = C\_PK(n, m, tmp, PK);

}

}

if (ljc != 1) {

A[++tot] = ljc;

B[tot] = C\_PK(n, m, ljc, ljc);

}

long long ans = 0;

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

long long M = P / A[i], T = INV(M, A[i]);

ans = (ans + B[i] \* M % P \* T % P) % P;

}

return ans;

}

## 组合数简易n≤61

#include <bits/stdc++.h>

using namespace std;

#define int long long

int C(int a, int b) { //简易版C最大61即longlong范围内所有数

int mul = 1;

int dev = 1;

if (b >= a + 1 / 2) b = a - b;

for (int i = a - b + 1, j = 1; i <= a; i++, j++) {

mul \*= i;

dev \*= j;

int dev1 = \_\_gcd(mul, dev);

mul /= dev1;

dev /= dev1;

}

return mul;

}

## FFT\_example

#include <algorithm>

#include <cmath>

#include <cstdio>

#include <cstring>

#include <iostream>

#include <unordered\_map>

using namespace std;

typedef long long ll;

typedef unsigned long long ull;

const int N = 5000007;

const double PI = acos(-1);

int n, m;

int res, ans[N];

int limit = 1; //

int L; //二进制的位数

int R[N];

inline int read() {

register int x = 0, f = 1;

register 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;

}

struct Complex {

double x, y;

Complex(double x = 0, double y = 0) : x(x), y(y) {}

} a[N], b[N];

Complex operator\*(Complex J, Complex Q) {

//模长相乘，幅度相加

return Complex(J.x \* Q.x - J.y \* Q.y, J.x \* Q.y + J.y \* Q.x);

}

Complex operator-(Complex J, Complex Q) {

return Complex(J.x - Q.x, J.y - Q.y);

}

Complex operator+(Complex J, Complex Q) {

return Complex(J.x + Q.x, J.y + Q.y);

}

//自己集成的\_计算第一个比m+n大的2的整数幂limit,二进制翻转数数组R

void FFT\_init(int n, int m) {

while (limit <= n + m)

limit <<= 1, L++;

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

R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1));

}

void FFT(Complex\* A, int type) { // FFT板子

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

if (i < R[i]) swap(A[i], A[R[i]]);

// i小于R[i]时才交换，防止同一个元素交换两次，回到它原来的位置。

//从底层往上合并

for (int mid = 1; mid < limit; mid <<= 1) {

//待合并区间长度的一半，最开始是两个长度为1的序列合并,mid = 1;

Complex wn(cos(PI / mid), type \* sin(PI / mid)); //单位根w\_n^1;

for (int len = mid << 1, pos = 0; pos < limit; pos += len) {

// len是区间的长度，pos是当前的位置,也就是合并到了哪一位

Complex w(1, 0); //幂,一直乘，得到平方，三次方...

for (int k = 0; k < mid; ++k, w = w \* wn) {

//只扫左半部分，蝴蝶变换得到右半部分的答案,w 为 w\_n^k

Complex x = A[pos + k]; //左半部分

Complex y = w \* A[pos + mid + k]; //右半部分

A[pos + k] = x + y; //左边加

A[pos + mid + k] = x - y; //右边减

}

}

}

if (type == 1) return;

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

a[i].x /= limit;

//最后要除以limit也就是补成了2的整数幂的那个N，将点值转换为系数

//（前面推过了点值与系数之间相除是N）

}

//正常FFT

/\*

int main() {

n = read(), m = read();

//读入多项式的每一项，保存在复数的实部

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

a[i].x = read();

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

b[i].x = read();

while (limit <= n + m)

limit <<= 1, L++;

//也可以写成：limit = 1 << int(log2(n + m) + 1);

// 补成2的整次幂，也就是N

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

R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1));

FFT(a, 1); // FFT 把a的系数表示转化为点值表示

FFT(b, 1); // FFT 把b的系数表示转化为点值表示

//计算两个系数表示法的多项式相乘后的点值表示

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

a[i] = a[i] \* b[i];

//对应项相乘，O(n)得到点值表示的多项式的解C，利用逆变换完成插值得到答案C的点值表示

FFT(a, -1);

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

//这里的 x 和 y 是 double 的 hhh

printf("%d ", (int)(a[i].x + 0.5)); //注意要+0.5，否则精度会有问题

}

\*/

//三步变两步FFT

int main() {

n = read(), m = read();

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

a[i].x = read();

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

a[i].y = read(); //把b(x)放到a(x)的虚部上

while (limit <= n + m)

limit <<= 1, L++;

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

R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1));

FFT(a, 1);

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

a[i] = a[i] \* a[i]; //求出a(x)^2

FFT(a, -1);

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

printf("%d ", (int)(a[i].y / 2 + 0.5));

//虚部取出来除2，注意要+0.5，否则精度会有问题,这里的x和y都是double

}

## FFT

#include <bits/stdc++.h>

using namespace std;

//全局变量n，m分别存储两个多项式的最高次幂

const int N = 1e5 + 10; // N为多项式的最高次幂+1

const double PI = acos(-1);

int limit = 1; //刚比n+m大的2的整数幂

int L; //二进制的位数

int R[N]; //二进制翻转数数组

struct Complex {

double x, y;

Complex(double x = 0, double y = 0) : x(x), y(y) {}

};

Complex A[N];

Complex operator\*(Complex J, Complex Q) {

//模长相乘，幅度相加

return Complex(J.x \* Q.x - J.y \* Q.y, J.x \* Q.y + J.y \* Q.x);

}

Complex operator-(Complex J, Complex Q) {

return Complex(J.x - Q.x, J.y - Q.y);

}

Complex operator+(Complex J, Complex Q) {

return Complex(J.x + Q.x, J.y + Q.y);

}

void FFT\_init(int n, int m) {

while (limit <= n + m)

limit <<= 1, L++;

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

R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L - 1));

}

void FFT(Complex\* A, int type) // FFT板子 type=1表示傅里叶变换，-1表示逆变换

{

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

if (i < R[i]) swap(A[i], A[R[i]]);

for (int mid = 1; mid < limit; mid <<= 1) {

Complex wn(cos(PI / mid), type \* sin(PI / mid));

for (int len = mid << 1, pos = 0; pos < limit; pos += len) {

Complex w(1, 0);

for (int k = 0; k < mid; ++k, w = w \* wn) {

Complex x = A[pos + k];

Complex y = w \* A[pos + mid + k];

A[pos + k] = x + y;

A[pos + mid + k] = x - y;

}

}

}

if (type == 1) return;

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

A[i].x /= limit, A[i].y /= limit;

}

// FFT使用函数将多项式系数数组a与b的卷积放到c中，先init即可

// a[]的长度为n，b[]的长度为m

void use\_FFT(int a[], int b[], int c[], int n, int m) {

Complex A[N];

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

A[i].x = a[i];

A[i].y = b[i];

}

FFT(A, 1);

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

A[i] = A[i] \* A[i]; //求出a(x)^2

FFT(A, -1);

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

c[i] = (int)(A[i].y / 2 + 0.5);

}

}

# 图论

## 双端队列优化SPFA

#include <algorithm>

#include <cstdio>

#include <cstring>

#include <map>

#include <queue>

using namespace std;

const int maxn = 2.5e4 + 10, maxm = 2 \* (1e5 + 10);

typedef long long ll;

typedef pair<int, int> pii;

int T, R, P, S;

int dis[maxn], head[maxn], tot;

bool vis[maxn];

struct Edge {

int next, to, w;

} edge[maxm];

void spfa() {

deque<int> dq;

dq.push\_back(S);

vis[S] = 1;

memset(dis, 0x7f, sizeof(dis));

dis[S] = 0;

while (!dq.empty()) {

int now = dq.front();

dq.pop\_front();

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

int to = edge[i].to;

if (dis[to] > dis[now] + edge[i].w) {

dis[to] = dis[now] + edge[i].w;

if (!vis[to]) {

if (!dq.empty() && dis[to] <= dis[dq.front()])

dq.push\_front(to);

else

dq.push\_back(to);

vis[to] = 1;

}

}

}

vis[now] = 0;

}

}

void add(int x, int y, int z) {

edge[++tot] = Edge{head[x], y, z};

head[x] = tot;

}

int main() {

int i, j, k;

int x, y, z;

scanf("%d%d%d%d", &T, &R, &P, &S);

for (i = 1; i <= R; i++) {

scanf("%d%d%d", &x, &y, &z);

add(x, y, z);

add(y, x, z);

}

for (i = 1; i <= P; i++) {

scanf("%d%d%d", &x, &y, &z);

add(x, y, z);

}

spfa();

for (i = 1; i <= T; i++) {

if (dis[i] != 0x7f7f7f7f) {

printf("%d\n", dis[i]);

} else {

printf("NO PATH\n");

}

}

}

## 2-sat

#include <bits/stdc++.h>

#define int long long

using namespace std;

const int N = 100000;

struct Node {

int to, next;

} node[N];

int head[N], tot;

int dfn[N], vis[N], low[N], tim;

int scc[N], scc\_sum;

void add(int a, int b) {

node[++tot] = {b, head[a]};

head[a] = tot;

}

stack<int> st1;

void tarjan(int u, int fa) {

dfn[u] = low[u] = ++tim;

st1.push(u);

vis[u] = 1;

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

int to = node[i].to;

if (!dfn[to]) {

tarjan(to, u);

low[u] = min(low[to], low[u]);

} else if (vis[to]) {

low[u] = min(low[u], dfn[to]);

}

}

if (dfn[u] == low[u]) {

++scc\_sum;

while (1) {

int x = st1.top();

st1.pop();

scc[x] = scc\_sum;

vis[x] = 0;

if (x == u) break;

}

}

}

signed use(int n, int m) {

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

int a, va, b, vb, op;

if (op == 1) { // a=va或b=vb

add(a + n \* (va ^ 1), b + n \* vb);

add(b + n \* (vb ^ 1), a + n \* va);

}

if (op == 2) { // a=va时b=vb b=vb时a=va

add(a + n \* va, b + n \* vb);

add(a + n \* (va ^ 1), b + n \* (vb ^ 1));

}

if (op == 3) { // a=va必成立

add(a + n \* (va ^ 1), a + n \* (va));

}

}

// node[i]=1表示i为真 node[i+n]=1表示i为假

for (int i = 1; i <= 2 \* n; i++) {

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

}

// 一个变量的真和假存在同一个强连通分量中就不可能实现

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

if (scc[i] == scc[i + n]) {

cout << "IMPOSSIBLE";

return 0;

}

}

cout << "POSSIBLE" << endl;

// 每个变量取拓扑序大的值（0或1）即可

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

printf("%lld ", scc[i] > scc[i + n]);

}

return 0;

}

## 二分图染色推荐

const int N = 100000;

// 二分图本质是无向图

struct Node {

int to, next;

} node[N];

// O(n+m)

int head[N];

int color[N];

// 染色1表示白色-1表示黑色0表示没染

bool dfs(int v, int c) {

color[v] = c;

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

int to = node[i].to;

if (color[to] == c) return false;

if (color[to] == 0 && !dfs(to, -c)) return false;

}

return true;

}

bool use(int n) {

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

if (!color[i]) {

if (!dfs(i, 1)) return false;

}

}

return true;

}

## 二分图染色

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1e6 + 10;

// O(n+m)

vector<int> mp[maxn];

int color[maxn];

int n, m;

bool bfs\_tu(int u) {

queue<int> que;

que.push(u);

color[u] = 1;

while (que.size()) {

int x = que.front();

que.pop();

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

if (mp[x][i]) {

if (color[i] == -1) //如果未能染色

{

color[i] = color[x] ^ 1; //与上一个相连结点颜色要不同

que.push(i);

} else if (color[i] == color[x]) //如果有冲突

{

return false;

}

}

}

}

return true;

}

bool match() {

memset(color, -1, sizeof(color));

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

if (color[i] == -1) //如果没有染色，就说明这不在上一个连通块里

{

if (!bfs\_tu(i)) return false;

}

}

return true;

}

## 拓扑序

// 邻接表建图版

#include <bits/stdc++.h>

using namespace std;

//

const int N = 1e6 + 10;

struct Node {

int to, next;

} node[N];

int n; //节点数

int head[N], tot;

int ind[N]; //入度 in degree 直译入度

int out[N]; //出度

int topolist[N];

queue<int> q;

void toposort() { // 即bfs

int pos = 1;

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

if (!ind[i]) {

q.push(i);

topolist[pos++] = i;

}

}

while (!q.empty()) {

int u = q.front();

q.pop();

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

int x = node[i].to;

ind[x]--;

if (!ind[x]) {

q.push(x);

topolist[pos++] = x;

}

}

}

return;

}

void add(int a, int b) {

ind[b]++;

out[a]++;

node[++tot] = {b, head[a]};

head[a] = tot;

}

## 匈牙利算法

const int N = 100000;

struct Node {

int to, next;

} node[N];

int head[N], color[N];

int vistime[N];

int mch[N]; // 与之配对的点

bool match(int u, int tag) {

if (vistime[u] == tag) return false;

vistime[u] = tag;

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

int to = node[i].to;

if (mch[to] == 0 || match(mch[to], tag)) {

mch[to] = u;

return true;

}

}

return false;

}

int use(int n) {

int max\_matches;

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

if (color[i] == 1 && match(i, i)) { ++max\_matches; }

}

return max\_matches;

}

## dijkstra推荐

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

typedef pair<int, int> pii;

const int inf = 0x7fffffff;

struct Node {

int to, next, w, u;

} node[N];

int head[N], vis[N];

int n, m;

// O(mlogm)

int dis[N];

priority\_queue<pii, vector<pii>, greater<pii>> pq1;

void dijkstra(int start, int n) {

pq1.push({0, start});

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

dis[i] = inf;

}

dis[start] = 0;

while (!pq1.empty()) {

int u = pq1.top().second;

pq1.pop();

if (vis[u]) continue;

vis[u] = 1;

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

int to = node[i].to;

int w = node[i].w;

int t1 = dis[u] + w;

if (t1 < dis[to]) {

pq1.push({t1, to});

dis[to] = t1;

}

}

}

}

## dijkstra

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

typedef pair<int, int> pii;

const int inf = 0x7fffffff;

struct Node {

int to, next, w;

} node[N];

int head[N], vis[N];

// O(n^2)

int dis[N];

void dijkstra(int start, int n) {

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

dis[i] = inf;

}

dis[start] = 0;

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

int u = 0;

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

if (vis[i]) continue;

if (u == 0 || dis[u] > dis[i]) u = i;

}

vis[u] = 1;

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

int to = node[i].to;

int w = node[i].w;

int t1 = dis[u] + w;

if (dis[to] > t1) { dis[to] = t1; }

}

}

}

## floyd

#include <bits/stdc++.h>

using namespace std;

const int N = 500;

const int inf = 0x7fffffff;

int dis[N][N];

int floyd(int n) {

for (int k = 1; k <= n; k++) { // 整体上看k表示以前k个点作为中转点i能否到达j到达j的最短距离是多少

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

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

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

}

}

}

}

## johnson

#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> pii;

const int N = 3e3 + 10;

const int inf = 0x7fffffff;

int tot;

struct Node {

int to, next, w;

} node[N];

int head[N], vis[N];

inline void add(int a, int b, int c = 0) {

node[++tot] = {b, head[a], c};

head[a] = tot;

}

// O(nmlogm)

//

int h[N], in[N];

int dis[N];

int disMatrix[N][N]; // 距离矩阵 存从i出发到j的最短路距离

queue<int> q1;

bool spfa(int start, int n) {

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

h[i] = inf;

}

h[start] = 0;

q1.push(start);

while (!q1.empty()) {

int u = q1.front();

q1.pop();

vis[u] = 0; // vis表示是否在队列中

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

int to = node[i].to;

int w = node[i].w;

int t1 = h[u] + w;

if (t1 < h[to]) {

h[to] = t1;

if (!vis[to]) {

q1.push(to);

vis[to] = 1;

in[to]++;

if (in[to] > n) return false; // 若一个点入队超过n次则存在负权环

}

}

}

}

return true;

}

priority\_queue<pii, vector<pii>, greater<pii>> pq1;

void dijkstra(int start, int n) {

pq1.push({0, start});

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

dis[i] = inf;

}

dis[start] = 0;

while (!pq1.empty()) {

int u = pq1.top().second;

pq1.pop();

if (vis[u]) continue;

vis[u] = 1;

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

int to = node[i].to;

int w = node[i].w;

int t1 = dis[u] + w;

if (t1 < dis[to]) {

pq1.push({t1, to});

dis[to] = t1;

}

}

}

}

// johnson全源最短路 若存在负环返回false

bool johnson(int n) {

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

add(0, i, 0);

}

if (!spfa(0, n)) return false;

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

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

int to = node[i].to;

node[i].w = node[i].w + h[u] - h[to];

}

}

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

dijkstra(i, n);

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

vis[j] = 0; // spfa跑完vis是全空 dijkstra跑完能到的vis全满

if (dis[j] != inf) {

disMatrix[i][j] = dis[j] - h[i] + h[j];

} else {

disMatrix[i][j] = inf;

}

}

}

return true;

}

## kruskal

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

typedef pair<int, int> pii;

const int inf = 0x7fffffff;

struct Node {

int to, next, w, u;

} node[N];

int head[N], vis[N];

int n, m;

//

// O(mlogm)

int dis[N];

int fa[N];

int found(int x) {

if (x == fa[x]) return x;

return fa[x] = found(fa[x]);

}

bool cmp(Node a, Node b) {

return a.w < b.w;

}

void init() {

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

fa[i] = i;

}

}

void kruskal() { // 注意kruskal其实可以只用m而经常被用成2\*m用成2m了记得乘2

sort(node, node + m, cmp);

//将边的权值排序

int cnt = 0;

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

int fau = found(node[i].u), fav = found(node[i].to);

if (fau == fav) { continue; }

//若出现两个点已经联通了，则说明这一条边不需要了

fa[fav] = fau;

//将eu、ev合并

if (++cnt == n - 1) { break; }

//循环结束条件，及边数为点数减一时

}

}

## kruskal重构树

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define rep(i, a, b) for (int i = a; i <= b; i++)

#define per(i, a, b) for (int i = a; i >= b; i--)

#define cf \

int \_; \

cin >> \_; \

while (\_--)

typedef pair<int, int> par;

int lowbit(int n) {

return (n & (-n));

}

int read() {

int x = 0, w = 1;

char ch = 0;

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

if (ch == '-') w = -1;

ch = getchar();

}

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

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

ch = getchar();

}

return x \* w;

}

const int mod = 998244353;

const int INF = 0x3f3f3f3f;

const int N = 2001000;

const int M = 2000010;

int n, m, k, ans, a, b, T, cnt, tot, top, num, sum, root, mas, Q, x, y, t;

char ch;

int f[N][30];

int dpth[N], dist[N][30];

int smm[N];

struct node {

int f, t, w;

} arr[M];

bool cmp(node a, node b) {

return a.w > b.w;

}

vector<int> v[N];

int bcj[N];

int find(int a) {

if (a == bcj[a]) return a;

return bcj[a] = find(bcj[a]);

}

void dfs(int u, int father) {

dpth[u] = dpth[father] + 1;

f[u][0] = father;

for (int i = 1; (1 << i) <= dpth[u]; i++) {

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

}

for (auto ed : v[u]) {

if (ed != father) { dfs(ed, u); }

}

}

int lca(int x, int y) {

if (dpth[x] > dpth[y]) swap(x, y);

per(i, t, 0) {

if (dpth[x] <= dpth[y] - (1 << i)) y = f[y][i];

}

if (x == y) return x;

per(i, t, 0) {

if (f[y][i] != f[x][i]) y = f[y][i], x = f[x][i];

}

return f[x][0];

}

signed main() {

n = read();

m = read();

rep(i, 1, m) {

x = read();

y = read();

t = read();

arr[i] = {x, y, t};

}

int lim = (n << 1) - 1;

rep(i, 1, n) bcj[i] = i;

sort(arr + 1, arr + 1 + m, cmp);

tot = n;

rep(i, 1, m) {

int x = arr[i].f, y = arr[i].t;

a = find(x);

b = find(y);

if (a == b) continue;

++tot;

bcj[a] = tot;

bcj[b] = tot;

bcj[tot] = tot;

smm[tot] = arr[i].w;

v[tot].push\_back(a);

v[tot].push\_back(b);

if (tot == lim) break;

}

t = log(tot) / log(2) + 1;

for (int i = tot; i > n; --i) //预处理lca,注意原图可能不联通，所以我们可能构出了一个森林

if (!dpth[i]) dpth[i] = 1, f[i][0] = i, dfs(i, 0);

cf {

a = read();

b = read();

// lca 的板子

if (find(a) != find(b)) {

cout << -1 << "\n";

continue;

}

int now = lca(a, b);

printf("%d\n", smm[now]);

}

return 0;

}

## prim推荐

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

typedef pair<int, int> pii;

const int inf = 0x7fffffff;

struct Node {

int to, next, w;

} node[N];

int head[N], vis[N];

// O(n^2)

int dis[N];

int prim(int n) {

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

dis[i] = inf;

}

int cnt = 0, u = 1;

while (++cnt <= n) //最小生成树边数等于点数-1但为了让连通图所有点vis都等于1

{

int minw = inf;

vis[u] = 1;

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

int to = node[i].to;

if (dis[to] > node[i].w && !vis[to]) { dis[to] = node[i].w; }

}

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

if (!vis[i] && minw > dis[i]) {

minw = dis[i];

u = i;

}

}

}

return 0;

}

## prim

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

typedef pair<int, int> pii;

const int inf = 0x7fffffff;

struct Node {

int to, next, w;

} node[N];

int head[N], vis[N];

// O(mlogm)

int dis[N];

priority\_queue<pii, vector<pii>, greater<pii> > q;

void prim(int n) {

dis[1] = 0;

int cnt = 0;

q.push(make\_pair(0, 1));

while (!q.empty() && cnt < n) {

int d = q.top().first, u = q.top().second;

q.pop();

if (vis[u]) continue;

cnt++;

vis[u] = 1;

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

if (node[i].w < dis[node[i].to]) {

dis[node[i].to] = node[i].w;

q.push(make\_pair(dis[node[i].to], node[i].to));

}

}

}

}

## spfa板子

#include <bits/stdc++.h>

using namespace std;

const int N = 100000;

typedef pair<int, int> pii;

const int inf = 0x7fffffff;

struct Node {

int to, next, w;

} node[N];

int head[N], vis[N];

/\*\*\* SPFA(shortest path faster algorithm) \*\*\*/

// 玄学：O(n ~ n\*m)不等

//

int dis[N], in[N];

queue<int> q1;

bool spfa(int start, int n) {

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

dis[i] = inf;

}

dis[start] = 0;

q1.push(start);

while (!q1.empty()) {

int u = q1.front();

q1.pop();

vis[u] = 0; // vis表示是否在队列中

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

int to = node[i].to;

int w = node[i].w;

int t1 = dis[u] + w;

if (t1 < dis[to]) {

dis[to] = t1;

if (!vis[to]) {

q1.push(to);

vis[to] = 1;

in[to]++;

if (in[to] > n) return false; // 若一个点入队超过n次则存在负权环

}

}

}

}

return true;

}

## tarjan

#include <bits/stdc++.h>

using namespace std;

const int M = 1001, N = 1001;

struct Node {

int to, next;

} node[2 \* M + 10];

int head[N], vis[N];

// O(N+M)

//

int dfn[N], low[N]; // dfn是严格单调递增的只有low可能变化

int Stack[N], tim, idx; // tim时间戳

int scc[N], p[N]; // s表示属于哪一个强连通分量(strongly connected components)，p表示点权

int cp[N]; // 割点cut\_point的缩写

int scc\_cnt; // 强连通分量序即逆拓扑序

set<pair<int, int>> brige;

// tarjan 算法在有向图中用于求强连通分量在无向图中用于求割点和割边（桥）

void tarjan(int u, int fa) {

dfn[u] = low[u] = ++tim;

Stack[++idx] = u;

vis[u] = 1; // 表示在栈里，最终栈一定是空的

int child = 0; //代表子树的数量在求割点与桥时在根节点判断是否为割点或桥时才用

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

int to = node[i].to;

if (!dfn[to]) {

child++; // 求割点或桥时这个计数只在判断根节点时才使用

tarjan(to, u);

low[u] = min(low[u], low[to]);

if (low[to] >= dfn[u] && u != fa) { // 求割点用>=求桥用>

cp[u] = true;

// brige.insert({u, to});

}

}

// 求强连通分量

else if (vis[to]) {

low[u] = min(low[u], dfn[to]);

// 在求强连通分量时写 dfn和low都是对的但在求割点与桥时只能写dfn也就是现在这种写法

// 因为在求强连通分量时只要low!=dfn这个点就会被缩点

}

// 求割点或桥

// else if (dfn[to] < dfn[u] && to != fa) {

// low[u] = min(low[u], dfn[to]);

// }

}

if (low[u] == dfn[u]) {

// scc\_cnt++;

while (1) {

int y = Stack[idx--];

scc[y] = u; // 缩点

// scc[y] = scc\_cnt; // 顺便求逆拓扑序

vis[y] = false;

if (u == y) break;

p[u] += p[y]; // 将点上的信息全部加到代表元（点）上（点权等）

}

}

if (fa == u && child > 1) { cp[u] = true; } // 判断根节点是否为割点

}

int use(int n) {

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

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

return 0;

}

# 杂

## 模拟退火

#include <bits/stdc++.h>

using namespace std;

int n, sx, sy; // sx=sumx

double ansx, ansy; //全局最优解的坐标

double ans = 1e18, t; //全局最优解、温度

const double delta = 0.993; // 0.993; //降温系数

inline double cal(double x, double y) { //计算

return 0.0;

}

inline void simulate\_anneal() { // SA主过程

double x = ansx, y = ansy;

t = 2000; //初始温度

while (t > 1e-14) {

double X = x + ((rand() << 1) - RAND\_MAX) \* t;

double Y = y + ((rand() << 1) - RAND\_MAX) \* t; //得出一个新的坐标

double now = cal(X, Y);

double Delta = now - ans;

if (Delta < 0) { //接受

x = X, y = Y;

ansx = x, ansy = y, ans = now;

} else if (exp(-Delta / t) \* RAND\_MAX > rand())

x = X, y = Y; //以一个概率接受

t \*= delta;

}

}

inline void use() { //多跑几遍SA，减小误差

ansx = (double)sx / n, ansy = (double)sy / n; //从平均值开始更容易接近最优解

simulate\_anneal();

simulate\_anneal();

simulate\_anneal();

simulate\_anneal();

simulate\_anneal();

}

void init() {

srand(18253517);

srand(rand());

srand(rand()); //玄学srand

// cout << RAND\_MAX;//32767

}

## 玄学clock

#include <bits/stdc++.h>

using namespace std;

void clock\_break(int ans) { // 在你觉得可能超时的循环中加入它

if ((double)clock() / CLOCKS\_PER\_SEC > 0.987) {

cout << ans;

exit(0);

}

}

## 玄学vector

#include <bits/stdc++.h>

using namespace std;

// vector能在1s内模拟10^5的随机插入

vector<int> v1;

void use(int a[], int b[], int n) {

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

v1.insert(v1.begin() + a[i], b[i]);

}

}

## cdq分治

const int N = 100000;

int a[N], b[N];

int cdq\_merger(int l, int r) {

if (l == r) {

// return;

return a[l] >= 0; // 求顺序对

}

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

int ans = cdq\_merger(l, mid) + cdq\_merger(mid + 1, r);

int pl = l, pr = mid + 1;

for (int i = l; i <= r; i++) {

if (pl <= mid && a[pl] <= a[pr] || pr > r) { // 注意是三个条件

b[i] = a[pl++];

} else {

b[i] = a[pr++];

// ans += (mid - pl + 1);//求逆序对

ans += pl - mid; //求顺序对，即求二维偏序

}

}

for (int i = l; i <= r; i++)

a[i] = b[i];

return ans;

}

# 字符串

## hash

#include <bits/stdc++.h>

using namespace std;

#define int unsigned long long

typedef unsigned long long ull;

const int hashmod = 1e18 + 2049;

//

int Hash(char s[]) { //用scanf加快读入

int base = 131;

int val = 0, len = strlen(s);

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

val = (base \* val + (ull)s[i]) % hashmod;

}

return val;

//枚举该字符串的每一位，与base相乘，转化为base进制，加(ull)是为了防止爆栈搞出一个负数，(ull)是无符号的，但其实加了一个ull是可以不用mod的，加个mod更保险

//然而加了mod会很玄学，莫名比不加mod慢了300多ms

}

## kmp

#include <bits/stdc++.h>

using namespace std;

int kmp[1000];

void kmp\_match(char a[], char b[]) { // a为主串b为模式串 （简单点说a长，b短）

int j = -1;

int la = strlen(a);

int lb = strlen(b);

kmp[0] = -1;

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

while (j != -1 && b[i] != b[j + 1])

j = kmp[j];

//此处判断j是否为0的原因在于，如果回跳到第一个字符就不 用再回跳了

//通过自己匹配自己来得出每一个点的kmp值

if (b[j + 1] == b[i]) j++;

kmp[i] = j;

// i+1失配后应该如何跳

}

j = -1; // j可以看做表示当前已经匹配完的模式串的最后一位的位置

//如果楼上看不懂，你也可以理解为j表示模式串匹配到第几位了

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

while (j != -1 && b[j + 1] != a[i])

j = kmp[j];

//如果失配 ，那么就不断向回跳，直到可以继续匹配

if (b[j + 1] == a[i]) j++;

//如果匹配成功，那么对应的模式串位置++

if (j == lb - 1) {

j = kmp[j];

//继续匹配

}

}

}

## trie

// trie tree的储存方式：将字母储存在边上，边的节点连接与它相连的字母

// trie[rt][x]=tot:rt是上个节点编号，x是字母，tot是下个节点编号

#include <bits/stdc++.h>

using namespace std;

const int N = 2000010;

int tot = 1;

//

int trie[N][26];

int isw[N];

int sum[N];

int search(char s[], int op = 1) {

int root = 0;

int len = strlen(s);

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

int id = s[i] - 'a';

if (!trie[root][id]) return 0;

root = trie[root][id];

} // root经过此循环后变成前缀最后一个字母所在位置

if (op == 1) return isw[root]; // op=1查询是否为单词或单词出现的次数

if (op == 2) return sum[root]; // op=2查询前缀和出现的次数

return 1; // op=0是否为前缀或单词

}

void insert(char s[]) {

int len = strlen(s);

int root = 0;

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

int id = s[i] - 'a';

if (!trie[root][id]) trie[root][id] = ++tot;

root = trie[root][id];

sum[root]++; //前缀保存

}

isw[root]++; //标志该单词末位字母的尾结点，在查询整个单词时用到

}