o目录

[基础算法 4](#_Toc23279246)

[输入输出 4](#_Toc23279247)

[二分 4](#_Toc23279248)

[离散化 4](#_Toc23279249)

[拓扑排序 5](#_Toc23279250)

[排序 6](#_Toc23279251)

[计数排序 6](#_Toc23279252)

[求逆序对 7](#_Toc23279253)

[1.归并排序 7](#_Toc23279254)

[2.树状数组求逆序对 8](#_Toc23279255)

[字符串 9](#_Toc23279256)

[字符串Hash 9](#_Toc23279257)

[KMP 10](#_Toc23279258)

[扩展KMP 11](#_Toc23279259)

[马拉车（最长回文子串） 13](#_Toc23279260)

[AC自动机 14](#_Toc23279261)

[最小表示法 15](#_Toc23279262)

[后缀数组 16](#_Toc23279263)

[数据结构 18](#_Toc23279264)

[STL使用 18](#_Toc23279265)

[bitset 18](#_Toc23279266)

[优先队列 19](#_Toc23279267)

[并查集 19](#_Toc23279268)

[前缀和 20](#_Toc23279269)

[树状数组 20](#_Toc23279270)

[线段树（Segment Tree） 21](#_Toc23279271)

[线段树求区间最大连续字段和 1](#_Toc23279272)

[线段树|树状数组 离线处理 2](#_Toc23279273)

[线段树+扫描线（矩形面积并） 4](#_Toc23279274)

[矩形周长（HDU-1828） 6](#_Toc23279275)

[莫对算法 8](#_Toc23279276)

[主席树 10](#_Toc23279277)

[单调栈 11](#_Toc23279278)

[单调队列 13](#_Toc23279279)

[单调队列优化 13](#_Toc23279280)

[优先队列优化 14](#_Toc23279281)

[平衡树 16](#_Toc23279282)

[Treap实现名词树 16](#_Toc23279283)

[数学 19](#_Toc23279284)

[基础数论 19](#_Toc23279285)

[整除 19](#_Toc23279286)

[求和公式 19](#_Toc23279287)

[整除分块 20](#_Toc23279288)

[完全数 20](#_Toc23279289)

[阶乘 21](#_Toc23279290)

[匹克定理 22](#_Toc23279291)

[梅森数 22](#_Toc23279292)

[斐波那契数列 23](#_Toc23279293)

[矩阵快速幂 23](#_Toc23279294)

[杨辉三角的奇偶性 26](#_Toc23279295)

[三角形 27](#_Toc23279296)

[构造勾股数 27](#_Toc23279297)

[欧几里得算法 28](#_Toc23279298)

[扩展欧几里得 28](#_Toc23279299)

[线性同余方程 28](#_Toc23279300)

[高次同余方程（BSGS算法） 29](#_Toc23279301)

[康拓展开&康拓逆展开 31](#_Toc23279302)

[组合数学 32](#_Toc23279303)

[Lucas定理 32](#_Toc23279304)

[Ferrers图像 33](#_Toc23279305)

[小球放盒模型 34](#_Toc23279306)

[约瑟夫环问题 35](#_Toc23279307)

[积性函数： 36](#_Toc23279308)

[欧拉函数 36](#_Toc23279309)

[莫比乌斯函数 37](#_Toc23279310)

[约数个数函数 37](#_Toc23279311)

[约数和函数 37](#_Toc23279312)

[反素数 38](#_Toc23279313)

[元函数 39](#_Toc23279314)

[恒等函数： 39](#_Toc23279315)

[单位函数 39](#_Toc23279316)

[狄利克雷卷积 39](#_Toc23279317)

[线性筛约数个数 40](#_Toc23279318)

[杜教筛 41](#_Toc23279319)

[逆元 43](#_Toc23279320)

[阶乘逆元(O(n)) 43](#_Toc23279321)

[原根 44](#_Toc23279322)

[中国剩余定理（除数互质） 45](#_Toc23279323)

[中国剩余定理（除数不互质） 45](#_Toc23279324)

[FFT 46](#_Toc23279325)

[计算几何 48](#_Toc23279326)

[二维基础 48](#_Toc23279327)

[基本运算 50](#_Toc23279328)

[点和直线 51](#_Toc23279329)

[圆 53](#_Toc23279330)

[凸包 57](#_Toc23279331)

[Graham Scan 57](#_Toc23279332)

[Andrew算法 59](#_Toc23279333)

[旋转卡壳求凸包直径 60](#_Toc23279334)

[三维坐标 61](#_Toc23279335)

[图论 62](#_Toc23279336)

[DFS序 62](#_Toc23279337)

[树链剖分 65](#_Toc23279338)

[LCA（最近公共祖先） 70](#_Toc23279339)

[树链剖分求LCA 70](#_Toc23279340)

[最短路： 71](#_Toc23279341)

[单源最短路（Single Source Shortest Path SSSP）无负权边 71](#_Toc23279342)

[Dijkstra 堆优化（优先队列 -- 大根堆）O（m log n） 72](#_Toc23279343)

[Bellman-Ford算法和SPFA算法 73](#_Toc23279344)

[Floyd 74](#_Toc23279345)

[最小生成树 75](#_Toc23279346)

[Kruskal(O(mlog m)) 75](#_Toc23279347)

[Tarjan算法 76](#_Toc23279348)

[有向图强连通分量 76](#_Toc23279349)

[割边（桥）与割点 77](#_Toc23279350)

[网络流 80](#_Toc23279351)

[EK 80](#_Toc23279352)

[Dinic 81](#_Toc23279353)

[二分图 82](#_Toc23279354)

[二分图判定 82](#_Toc23279355)

[二分图最大匹配 82](#_Toc23279356)

[一般图的最大匹配 83](#_Toc23279357)

[二分图带权匹配 83](#_Toc23279358)

[博弈论 84](#_Toc23279359)

[巴什博奕（Bash Game） 84](#_Toc23279360)

[威佐夫博奕（Wythoff Game） 84](#_Toc23279361)

[尼姆博弈(Nim Game) 84](#_Toc23279362)

[斐波那契博弈(Fibonacci Nim) 84](#_Toc23279363)

[JAVA大数 85](#_Toc23279364)

[瞎搞冷知识 87](#_Toc23279365)

[读写优化 88](#_Toc23279366)

# 基础算法

## 输入输出

一般忌用string类型(很容易TLE)，尽量用字符数组，

gets与 fgets, 二者都是按行读入

gets使用简单，gets(char \*s)，

fgets 使用安全， fgets(char \*s, maxn, stdin) , maxn表示输入字符串的最大长度，多出部分会截除，长度小的末尾不上’\0’字符，因此读入字符串的真正长度为strlen(s)-1

unsigned int 读入scanf("%u", &a);

unsigned long long 读入 scanf("%llu", &a);

long long double 读入”%llf”，输出”%lf”

## 二分

在单调递增序列a中查找>= x的数的最小的一个（即x或x的后继）

while(low < high){

int mid = (low + high) >> 1;

if(a[mid] >= x) high = mid;

else low = mid + 1;

}

在单调递增序列a中查找<=x 的数中最大的一个（即x或x的前驱）

while(low < high){

int mid = (low + high + 1) >> 1;

if(a[mid] <= x) l = mid;

else r = mid - 1;

}

## 离散化

void discrete(int \*a, int \*b, int n){

///对数组 a 离散化，数组 b 是临时数组

mec(b, a);

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

int p = unique(b+1, b+n+1)-b;

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

a[i] = lower\_bound(b+1, b+p, a[i])-b;

}

}

## 拓扑排序

bool bfs(){

while(!que.empty()){

que.pop();

}

int num = 0;

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

if(!indeg[i]){

que.push(i);

}

}

while(!que.empty()){

int x = que.front(); que.pop();

topu[++num] = x;

for(int i = 0; i < ve[x].size(); i ++){

int y = ve[x][i];

indeg[y] --;

if(!indeg[y]){

que.push(y);

}

}

}

if(num < n) return false;///表示存在环

return true;///不存在环

}

## 排序

### 计数排序

计数排序对一定量的整数排序时候的速度非常快，一般快于其他排序算法。是一种非常快捷的稳定性强的排序方法，时间复杂度O(n+k)其中n为要排序的数的个数，k为要排序的数的组大值。

限制条件：计数排序局限性比较大，只限于对整数进行排序。计数排序是消耗空间发杂度来获取快捷的排序方法，其空间发展度为O(k)同理k为要排序的最大值。

const int maxn = 1000000+10;

int a[maxn+10], b[maxn+10], c[maxn+10];

///b[i]记录元素i出现的次数,c表示完成的数组

int n, k;

void jishusort(){

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

b[a[i]] ++;

}

int num = 0;

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

while(b[i]){

b[i] --;

c[++num] = i;

}

}

for(int i = n; i >= n - k + 2; i --){

printf("%d ", c[i]);

}

printf("%d\n", c[n-k+1]);

}

## 求逆序对

### 1.归并排序

///POJ2299 归并排序求逆序对

const int maxn = 5e5 + 10;

int a[maxn], b[maxn]; LL cnt;

void Merge(int l, int mid, int r){

///合并 a[l~mid] 与 a[mid+1~r]

///a 是代派数列， b 是临时数组， cnt 是逆序对的个数

int i = l, j = mid+1;

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

if(j > r || (i <= mid && a[i] < a[j])){

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

}

else b[k] = a[j++]; cnt += (mid-i+1);

}

for(int k = l; k <= r; k ++) a[k] = b[k];

}

void Merge\_Sort(int l, int r){

int mid;

if(l < r){

mid = (l + r) >> 1;

Merge\_Sort(l, mid); Merge\_Sort(mid+1, r);

Merge(l, mid, r);

}

}

int main()

{

int n;

while(scanf("%d", &n) && n){

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

scanf("%d", &a[i]);

}

cnt = 0;

Merge\_Sort(1, n);

printf("%lld\n", cnt);

}

return 0;

}

### 2.树状数组求逆序对

const intmaxn = 5e5+10;

int c[maxn],a[maxn], b[maxn], int n;

int lowbit(int x){

return x&(-x);

}

void add(int x){

while(x<=n){

c[x] ++; x += lowbit(x);

}

}

int ask(int x){

int ans = 0;

while(x){

ans += c[x]; x -= lowbit(x);

}

return ans;

}

ll build(){

ll ans = 0;

mes(c, 0);

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

add(a[i]); ans += (i-ask(a[i]));

}

return ans;

}

int main()

{

int x;

while(cin>>n && n){

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

scanf("%d", &a[i]);

}

discrete(a, b, n);

ll ans = build();

printf("%lld\n", ans);

}

return 0;

}

# 字符串

## 字符串Hash

字符串Hash函数：把一个任意长度的字符串映射成一个非负整数，并且其冲突概率几乎为零。

选取一固定值P， 把字符串看做P进制数，并分配一个大于0的数值，代表每种字符，取以固定值M，求出该P进制数对M的余数，作为该字符串的Hash值。

一般来说，P取131，233， 13331，19260817 等的质数，此Hash值的冲突概率极低，M一般去， 即直接使用unsigned long long 类型，可以避免取模运算。

const int key = 137;

ull H[maxn], xp[maxn];

void init\_hash(char \*s){

int n = strlen(s);

H[n] = 0;

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

H[i] = H[i+1] \* key + (s[i]-'a' + 1);

}

xp[0] = 1;

for(int i = 1; i <= n; i ++) xp[i] = xp[i-1] \* key;

}

ull Hash(int pos, int len){

return H[pos]-H[pos+len]\*xp[len];

}

## KMP

解决字符串匹配问题：模式串str是否能在匹配串s中找到匹配。

const int maxn = 1e3 + 10;

int n, m;

char s[maxn], str[maxn];

int nextn[maxn];

/\*\*

串 str(模式串) 与串 s(匹配串) 匹配，

先获取串 str 的 nextn数组，

再与串 s 进行匹配

\*\*/

void getnext(int len){

mes(nextn, -1);

int i = 0, j = -1;

while(i < len){

if(j == -1 || str[i] == str[j]){

i ++; j++; nextn[i] = j;

}

else j = nextn[j];

}

}

int KMP(int n, int m){

int res = 0, i = 0, j = 0;

while(i < n){

if(j == -1 || s[i] == str[j]){i ++; j ++;}

else j = nextn[j];

if(j == m){res ++; j = 0;};

}

// return j == m ? i - j + 1 : -1; ///返回首次匹配成功的下标

return res ;///返回匹配成功的次数

}

int main()

{

while(scanf("%s", s)){

if(s[0] == '#')break;

scanf("%s", str);

int n = strlen(s), m = strlen(str);

getnext(m);

printf("%d\n", KMP(n, m));

}

return 0;

}

## 扩展KMP

解决问题：字符串S、T，能在线性时间解决T中每一个后缀对应S的最长前缀。

const int maxn = 1e5+100;

int nxt[maxn], ex[maxn];

char S[maxn], T[maxn];

void getnxt(char \*str) {

int len = strlen(str);

int i = 0, j;

nxt[0] = len;

while(str[i] == str[i+1] && i+1 < len) i++;

nxt[1] = i;//暴力匹配得到nxt[1]

int pos = 1;

for(i = 2; i < len; i++) {

if(nxt[i - pos] + i < nxt[pos] + pos) nxt[i] = nxt[i - pos];

else {

j = nxt[pos] + pos - i;

if(j < 0) j = 0;

while(i + j < len && str[j] == str[i + j]) j++;

nxt[i] = j;

pos = i;

}

}

}

void exkmp(char \*a, char \*b) {

int i = 0, j;

int n = strlen(a), m = strlen(b);

getnxt(b);

while(a[i] == b[i] && i < m && i < n) i++;

ex[0] = i;

int pos = 0;//从Pos开始匹配能匹配到的最远位置P

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

if(nxt[i - pos] + i < ex[pos] + pos) ex[i] = nxt[i-pos];

else {

j = ex[pos] + pos - i;

if(j < 0) j = 0;

while(i + j < n && j < m && a[i + j] == b[j]) j++;

ex[i] = j;

pos = i;

}

}

}

int main()

{

while(cin >> S >> T){

exkmp(T, S);

int slen = strlen(S), tlen = strlen(T);

int mx = 0;

for(int i = tlen - 1, j = 0; j < slen; j++, i--){

if(ex[i] == tlen - i){

mx = max(mx, ex[i]);

}

}

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

printf("%c", S[i]);

}

if(mx)printf(" ");

printf("%d\n", mx);

}

return 0;

}

## 马拉车（最长回文子串）

回文串(palindrome)：正读和反读都是一样的字符串，如：“noon”“abcba”。

暴力算法O(n^2)，马拉车算法O(n),在线性复杂度就能求出。

马拉车算法(Manacher Algorithm)

const int maxn = 2e5+100;

char str0[maxn],str[maxn<<1];

int p[maxn<<1],len; ///p[i]表示以i为中心的最长回文串的半径

void Pretreatment(){///预处理字符串str0

str[0] = '$', str[1] = '#';

len = strlen(str0);

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

str[i\*2+2] = str0[i];

str[i\*2+3] = '#';

}

str[len\*2+2] = '\0';

}

int Manacher(){//计算最长回文串长度

mes(p, 0);

int mx=0,id,ans=1;

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

if(mx>i)p[i]=min(p[2\*id-i],mx-i);

else p[i]=1;

for(;str[i-p[i]]==str[i+p[i]];p[i]++);

if(p[i]+i>mx)mx=p[i]+i,id=i;

ans = max(ans, p[i]);

}

return --ans;

}

如果求回文串的个数则将数组p遍历除以2，在求和即可

for(int i = 1; i < len2; i ++){

ans += p[i]/2;

}

## AC自动机

解决问题：解决多模式串匹配问题。

模板题：给定n个模式串s，1个匹配串t，，问有多少个模式串s是匹配串t的子串。

const int maxn = 1e6 + 100;

struct AC\_Machine{

int ch[maxn][26];

int num[maxn], f[maxn];

// f即为fail指针.

int tot;

void init(){

tot = 0; mes(num, 0); ///num 初始化为 0

mes(ch, 0);

}

void Insert(char \*s){///构建Trie树

int u = 0, len = strlen(s);/// 0 号节点是Trie树的根节点

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

if(!ch[u][s[i]-'a']){

ch[u][s[i]-'a'] = ++ tot;

}

u=ch[u][s[i]-'a'];

}

num[u] ++; ///num[u] 表示以 u 为结尾的字符串的个数

}

void Build(){///构建fail指针

queue<int>q; while(q.size())q.pop();

for(int i = 0; i < 26; i ++){

if(ch[0][i]){//第一层与其他单词不可能有公共前后缀,fail直接为根

f[ch[0][i]] = 0; q.push(ch[0][i]);

}

}

while(q.size()){

int u = q.front(); q.pop();

for(int i = 0; i < 26; i ++){

///fail[u]指针指u节点的父亲节点中与之相同的 u 子节点

if(ch[u][i]){

f[ch[u][i]] = ch[f[u]][i];

q.push(ch[u][i]);

}

else ch[u][i] = ch[f[u]][i];

}

}

}

int Query(char \*s){

int u = 0, len = strlen(s), ans = 0;

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

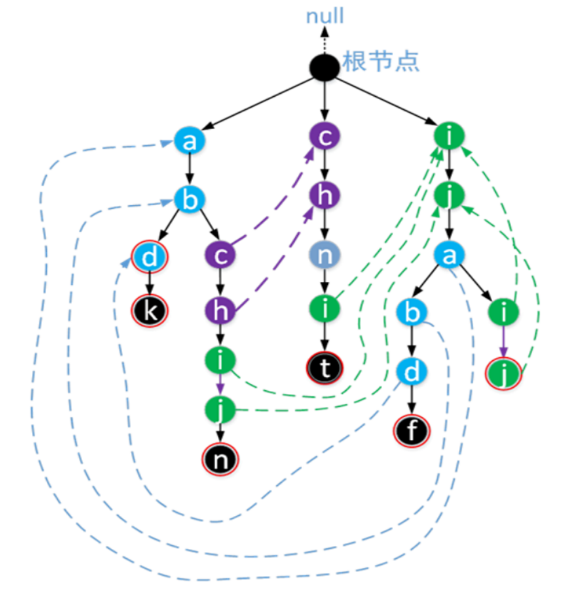
u = ch[u][s[i]-'a'];

for(int j = u; j && num[j] != -1; j = f[j]){

ans += num[j], num[j] = -1;//就用这个循环实现跳的过程

//因为直接已经在每个单词的最后面打了标记,所以直接加上即可.

}

 }

return ans;

}

}AC\_Ma;

char s[maxn];

int n;

int main()

{

AC\_Ma.init();

scanf("%d", &n);

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

scanf("%s", s);

AC\_Ma.Insert(s);

}

AC\_Ma.Build();

scanf("%s", s);

cout<<AC\_Ma.Query(s)<<endl;

}

## 最小表示法

给定字符串S[1…n]，不断地把最后一个字符放到开头，会得到n个字符串，称这n个字符串是循环同构的，这些字符串中字典序最小的一个成为字符串S的最小表示法。

int solve(char \*s){///注意数组开两倍模板题POJ 1509

int n = strlen(s+1);

for(int i = 1; i <= n; i ++) s[n + i] = s[i];

int i = 1, j = 2, k;

while(i <= n && j <= n){

for(k = 0; k <= n && s[i+k] == s[j+k]; k ++);

if(k == n) break;

if(s[i+k] > s[j+k]) i = i + k + 1;

else j = j + k + 1;

if(i == j) j ++;

}

return min(i, j);

}

## 后缀数组

struct Suffix\_Array{

char s[maxn];

int y[maxn], x[maxn], c[maxn], sa[maxn], rk[maxn], height[maxn], wt[30];

int MIN[maxn][30];

///sa对后缀排完序之后的序号，rk名次， height[i] = LCP(i, i-1)

int n,m;

void init(){

m = 122; mes(c, 0);

}

void Print(char \*s, int num){

for(int i = num; i <= n; i ++) printf("%c", s[i]);

}

void get\_SA(){

for(int i = 1; i <= n; i ++) ++c[x[i]=s[i]];

for(int i = 2; i <= m; i ++) c[i]+=c[i-1];

for(int i = n; i >= 1; i --) sa[c[x[i]]--] = i;

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

int num = 0;

for(int i = n - k + 1; i <= n; i ++) y[++ num] = i;

for(int i = 1; i <= n; i ++) if(sa[i]>k) y[++num] = sa[i]-k;

for(int i = 1; i <= m; i ++) c[i] = 0;

for(int i = 1; i <= n; i ++) ++c[x[i]];

for(int i = 2; i <= m; i ++) c[i] += c[i-1];

for(int i = n; i >= 1; i --) sa[c[x[y[i]]]--] = y[i], y[i] = 0;

swap(x, y);

x[sa[1]] = 1;

num = 1;

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

x[sa[i]] = (y[sa[i]] == y[sa[i-1]] && y[sa[i]+k] == y[sa[i-1]+k]) ? num : ++ num;

if(num == n) break;

m = num;

}

}

void get\_height(){

int k = 0;

for(int i = 1; i <= n; i ++) rk[sa[i]] = i;

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

if(rk[i] == 1) {height[rk[i]] = 0; continue;}

if(k) --k;

int j = sa[rk[i] - 1];

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

height[rk[i]] = k;

// printf("%d %d\n", rk[i], height[rk[i]]);

}

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

// printf("i = %d, sa = %d ", i, sa[i]);

// Print(s, sa[i]);

// printf("\n");

// }

// printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

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

// printf("i = %d, he = %d\n", i, height[i]);

// }

// printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

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

//

// printf("i = %d, rk = %d\n", i, rk[i]);

// }

}

void RMQ\_Init(){

for(int i = 1; i <= n; i ++)MIN[i][0] = height[i];

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

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

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

}

}

}

int RMQ\_Query(int l, int r){

int k = 0;

while((1<<(k+1)) <= (r - l + 1)) k ++;

return min(MIN[l][k], MIN[r -(1 << k) + 1][k]);

}

int LCP(int i, int j){

if(rk[i] > rk[j]) swap(i, j);

return RMQ\_Query(rk[i] + 1, rk[j]);

}

}SA;

# 数据结构

## STL使用

## bitset

注意：最低位下标为0，几最右边

bitset<length>bit;///无参构造，长度为 length

bitset<length>bit(x);///x转换成二进制，前面不足补零

string s = "110"; bitset<5>bit(s);///前面不足补零

若超过声明是长度，只取前面对应部分

bit = (~bit);///按位取反

bit = (bit<<1);///左移，右边补零

bit = (bit>>1);///右移，左边补零

bit1 == bit2///判断相同

bit1 != bit2///判断不同

bit1 & bit2///按位与运算

bit1 | bit2///按位或运算

bit1 ^ bit2///按位异或运算

bit.size();///统计bit的长度

bit.count();///统计bit中 1 的个数

bit.test(i)///测试第 i 位是否为1，是1返回true， 否则返回false

bit.any()///测试bit中是否存在1

bit.none()///测试bit中是否没有1

bit.all()///测试bit中是否都是1

bit.flip(i);///指定第i位取反，没有参数就默认所有位取反

bit.set(i, x);///指定第i为置为x

bit.set(i)///指定第i位置1

bit.set()///bit每一位置1

bit.reset();///置零

string s = bit.to\_string();///bit转化为字符串类型

int a = bit.to\_ulong();///转换为unsigned long 类型

int a = bit.to\_ullong();///转换为unsigned long long 类型

### 优先队列

priority\_queue<int>q;默认优先级，从大到小，大的优先级高

priority\_queue<int, vector<int>, greater<int>>q;从小到大，小的优先级高

自定义优先队列

struct node{

int val, id;

friend bool operator < (const node a, const node b){

return a.val < b.val;///从大到小排序

}

}; priority\_queue<node>qmax;

struct node{

int val, id;

}a[maxn];

struct cmp1{

bool operator () (const node &a, const node &b){

return a.val > b.val;///小的优先级高

}

};

struct cmp2{

bool operator () (const node &a, const node &b){

return a.val < b.val;///大的优先级高

}

};

priority\_queue<node, vector<node>, cmp1>qmin;

priority\_queue<node, vector<node>, cmp2>qmax;

## 并查集

并查集(Disjoint-set)：是一种可以动态维护若干个不重叠的集合，并支持合并和查询的数据结构。

用途：最小生成树（Kruskal）,合并集合，判断图是否存在环

并查集实际上是由若干棵树构成的森林，路径压缩后每个访问过的节点都会直接指向其根

int father[maxn];///father[i]->i的父节点

int find\_root(int x){

if(father[x] == x) return x;

return father[x] = find\_root(father[x]);///路径压缩

}

void merge\_v(int x, int y){///合并

int x\_root = find\_root(x);

int y\_root = find\_root(y);

father[x\_root] = y\_root;

}

## 前缀和

维护区间[1,i]的区间和

pre[i] = pre[i-1]+a[i];

方便区间求和 sum(l, r) = pre[r]-pre[l-1];

## 树状数组

树状数组（Binary Indexed Trees） --动态前缀和维护

const int maxn = 1e4 + 10;

int a[maxn], d[maxn];

int n;

int lowbit(int x){

return x & (-x);

}

///查询x的前缀和

int ask(int x){

int ans = 0;

while(x){

ans += d[x];

x -= lowbit(x);

}

return ans;

}

///单点修改

void add(int x, int v){

while(x <= n){

d[x] += v;

x += lowbit(x);

}

}

///构建树状数组

void build(){

memset(d, 0, sizeof a); ///初始化数组d为0

for(int i = 1; i <= n; i ++){ ///对每个节点进行单点修改

add(i, a[i]);

}

}

## 线段树（Segment Tree）

是一种基于分之思想的二叉树结构，用于在区间上进行信息统计。

#define lsn p<<1

#define rsn p<<1|1

#define len(p) (r(p) - l(p) + 1)

const int maxn = 1e5+100;

int n, m; ll a[maxn];

struct SegmentTree{

int l, r;

ll sum, add;

#define l(p) tree[p].l

#define r(p) tree[p].r

#define sum(p) tree[p].sum

#define add(p) tree[p].add

}tree[maxn<<2];

void push\_up(int p){

sum(p) = sum(lsn) + sum(rsn);

}

void push\_down(int p){

if(add(p)){

sum(lsn) += add(p) \* len(lsn);

sum(rsn) += add(p) \* len(rsn);

add(lsn) += add(p); add(rsn) += add(p);

add(p) = 0;

}

}

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

l(p) = l; r(p) = r; add(p) = 0;

if(l == r){

sum(p) = a[l]; return;

}

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

Build(l, mid, lsn); Build(mid+1, r, rsn);

push\_up(p);

}

void Change(int l, int r, int p, ll d){

if(l <= l(p) && r >= r(p)){

sum(p) += d \* len(p);

add(p) += d; return;

}

push\_down(p);

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

if(mid >= l) Change(l, r, lsn, d);

if(mid < r) Change(l, r, rsn, d);

push\_up(p);

}

ll Query(int l, int r, int p){

if(l <= l(p) && r >= r(p)){

return sum(p);

}

push\_down(p);

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

ll ans = 0;

if(mid >= l) ans += Query(l, r, lsn);

if(mid < r) ans += Query(l, r, rsn);

return ans;

}

int main()

{

scanf("%d %d", &n, &m);

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

scanf("%lld", &a[i]);

}

Build(1, n, 1);

while(m --){

int op; scanf("%d", &op);

int l, r ;

if(op == 1){

ll x; scanf("%d %d %lld", &l, &r, &x);

Change(l, r, 1, x);

}

else {

scanf("%d %d", &l, &r);

printf("%lld\n", Query(l, r, 1));

}

}

return 0;

}

/\*\*

5 5

1 5 4 2 3

2 2 4

1 2 3 2

2 3 4

1 1 5 1

2 1 4

\*\*/

/\*\*

Answer

11

8

20

\*\*/

### 线段树求区间最大连续字段和

struct SegmentTree{

int l, r;

//lmax表示从左区间向右的最大连续字段和，rmax表示从右区间向左的最大连续字段和

ll lmax, rmax, sum, dat; ///sum表示区间和,dat表示区间最大连续字段和

}tree[4\*maxn];

struct Data{

ll lmax, rmax, sum, dat;

};

void push(int p){///此处是关键

tree[p].lmax=max(tree[p<<1].lmax, tree[p<<1].sum+

tree[p<<1|1].lmax);

tree[p].rmax=max(tree[p<<1|1].rmax,tree[p<<1|1].sum+

tree[p<<1].rmax);

tree[p].sum = tree[p<<1].sum + tree[p<<1|1].sum;

tree[p].dat=max(max(tree[p<<1].dat,tree[p<<1|1].dat),

tree[p<<1].rmax+tree[p<<1|1].lmax);

}

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

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

if(l == r) {

tree[p].dat = tree[p].sum = tree[p].lmax = tree[p].rmax = -v[l].val; return ;

}

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

build(p<<1, l, mid);

build(p<<1|1, mid+1, r);

push(p);

}

Data query(int p, int l, int r){///注意查询函数返回的是结构体

if(l <= tree[p].l && r >= tree[p].r)return (Data){tree[p].lmax, tree[p].rmax, tree[p].sum, tree[p].dat};

int mid = (tree[p].l + tree[p].r) >> 1;

if(mid >= r) return query(p<<1, l, r);

if(mid < l) return query(p<<1|1, l, r);

Data a = query(p<<1, l, r), b = query(p<<1|1, l, r), c;

c.sum = a.sum + b.sum;

c.lmax = max(a.lmax, a.sum+b.lmax);

c.rmax = max(b.rmax, b.sum+a.rmax);

c.dat = max(max(a.dat, b.dat), a.rmax+b.lmax);

return c;

}

### 线段树|树状数组 离线处理

（HDU-3874）

const int maxn = 200000+10;

LL a[maxn], c[maxn], Ans[maxn];

struct node{

int l, r, id;

}no[200000+10];

int n, m;

map<int, int>mp;

bool cmp(const node &a, const node &b){

return a.r == b.r ? a.l < b.l : a.r < b.r;

}

int lowbit(int x){

return x&(-x);

}

void add(int x, LL d){

while(x <= n){

c[x] += d;

x += lowbit(x);

}

}

LL ask(int x){

LL ans = 0;

while(x){

ans += c[x];

x -= lowbit(x);

}

return ans;

}

int main()

{

int T; scanf("%d", &T);

while(T --){

mp.clear();

mes(c, 0); mes(Ans, 0);

scanf("%d", &n);

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

scanf("%lld", &a[i]);

}

scanf("%d", &m);

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

scanf("%d %d", &no[i].l, &no[i].r);

no[i].id = i;

}

sort(no+1, no+m+1, cmp);

int p = 1;

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

while(p <= no[i].r){

LL x = a[p];

if(mp[x]){

add(mp[x], -x);

}

add(p, x);

mp[x] = p; ///mp记录x上次出现的位置

p ++;

}

Ans[no[i].id] = ask(no[i].r)-ask(no[i].l-1);

}

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

printf("%lld\n", Ans[i]);

}

}

return 0;

}

### 线段树+扫描线（矩形面积并）

（HDU-1542）

const int maxn = 120+10;

int n;

double a[maxn\*4];

struct node{

double l, r, h; ///左右区间

int flag; ///上边1，下边-1

}no[maxn\*4];

bool cmp(const node &a, const node &b){

return a.h < b.h;

}

struct SegmentTree{

int l, r, s; ///左右区间、区间是否完整

double len; ///有效长度

}tree[maxn\*4];

void push(int p){

if(tree[p].s){

tree[p].len = a[tree[p].r+1] - a[tree[p].l];

}

else if(tree[p].l == tree[p].r){

tree[p].len = 0;

}

else {

tree[p].len = tree[p<<1].len + tree[p<<1|1].len;

}

}

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

tree[p].l = l; tree[p].r = r; tree[p].s = 0; tree[p].len = 0.0;

if(l == r){

return;

}

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

build(l, mid, p<<1);

build(mid+1, r, p<<1|1);

}

void change(int l, int r, int p, int x){

if(tree[p].l == l && tree[p].r == r){

tree[p].s += x;

push(p);

return;

}

// if(l==r)return ;

int mid = (tree[p].l + tree[p].r) >> 1;

if(r <= mid) change(l, r, p<<1, x);

else if(l > mid) change(l, r, p<<1|1, x);

else {

change(l, mid, p<<1, x);

change(mid+1, r, p<<1|1, x);

}

push(p);

}

int main(){

int Case = 0;

while(scanf("%d", &n) && n){

int num = 0;

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

double x1, x2, y1, y2;

cin>>x1>>y1>>x2>>y2;

no[++num].l = x1; no[num].r = x2; no[num].h = y1; no[num].flag = 1; a[num] = x1;

no[++num].l = x1; no[num].r = x2; no[num].h = y2; no[num].flag = -1; a[num] = x2;

}

sort(no + 1, no + num + 1, cmp);

sort(a + 1, a + num + 1);

double ans = 0.0;

int pos = unique(a + 1, a + num + 1) - a;

build(0, pos, 1);

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

int l = lower\_bound(a+1, a+pos, no[i].l)-a;

int r = lower\_bound(a+1, a+pos, no[i].r)-a-1;

change(l, r, 1, no[i].flag);

ans += (no[i+1].h - no[i].h)\*tree[1].len;

}

printf("Test case #%d\n", ++ Case);

printf("Total explored area: %.2f\n\n", ans);

}

return 0;

}

### 矩形周长（HDU-1828）

#include <bits/stdc++.h>

using namespace std;

const int maxn=100010;

struct Line{

int y, x1, x2, v;

Line(){}

Line(int y0, int x10, int x20, int v0){

y=y0, x1=x10, x2=x20, v=v0;

}

bool operator < (const Line &p) const{

return y<p.y;

}

}line[maxn];

struct node{

int l, r, v, len, s, ls, rs;

}tr[maxn<<3];

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

tr[m].l=l;

tr[m].r=r;

tr[m].len=tr[m].v=tr[m].s=tr[m].ls=tr[m].rs=0;

if(l==r) return;

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

build(m<<1, l, mid);

build(m<<1|1, mid+1, r);

}

void pushup(int m){

if(tr[m].v){

tr[m].len=tr[m].r+1-tr[m].l;

tr[m].s=1;

tr[m].ls=tr[m].rs=1;

}else if(tr[m].l==tr[m].r){

tr[m].len=tr[m].s=tr[m].ls=tr[m].rs=0;

}else{

tr[m].len=tr[m<<1].len+tr[m<<1|1].len;

tr[m].ls=tr[m<<1].ls;

tr[m].rs=tr[m<<1|1].rs;

tr[m].s=tr[m<<1].s+tr[m<<1|1].s-(tr[m<<1].rs&tr[m<<1|1].ls);

}

}

void updata(int m, int l, int r, int v){

if(tr[m].l==tr[m].r){

tr[m].v+=v;

pushup(m);

return;

}

int mid=(tr[m].r+tr[m].l)>>1;

if(r<=mid) updata(m<<1, l, r, v);

else if(l>mid) updata(m<<1|1, l, r, v);

else{

updata(m<<1, l, mid, v);

updata(m<<1|1, mid+1, r, v);

}

pushup(m);

}

int main(){

int n, x1, x2, y1, y2;

while(~scanf("%d", &n)){

if(n==0){

printf("0\n");

continue;

}

int L=maxn, R=-maxn, cnt=0;

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

scanf("%d%d%d%d", &x1, &y1, &x2, &y2);

line[++cnt]=Line(y1, x1, x2, 1);

line[++cnt]=Line(y2, x1, x2, -1);

L=min(L, x1);

R=max(R, x2);

}

build(1, L, R);

sort(line+1, line+1+cnt);

int ans=0, prelen=0;

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

int l=line[i].x1, r=line[i].x2-1, v=line[i].v;

updata(1, l, r, v);

ans+=abs(prelen-tr[1].len);

prelen=tr[1].len;

if(i!=cnt) ans+=2\*tr[1].s\*(line[i+1].y-line[i].y);

}

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

}

return 0;

}

## 莫对算法

题意: 求[L,R]内有多少对(i, j)满足|a[i] - a[j]| <= K，我们转换为求

[a[i] - k - 1, a[i] + k]的个数

const int maxn = 27005;

int a[maxn], b[maxn\*3], suml[maxn], sumr[maxn];

int n, m, K, Mx, block;

int c[maxn\*3];

struct Node{

int l, r, id;

bool operator < (const Node& rhs) const{

if(l/block == rhs.l/block)

return r < rhs.r;

return l/block < rhs.l/block;

}

}P[maxn];

int Ans[maxn];

int lowbit(int x){

return x&(-x);

}

void add(int x, int d){

while(x <= Mx){

c[x] += d; x += lowbit(x);

}

}

int ask(int x){

int ans = 0;

while(x){

ans += c[x]; x -= lowbit(x);

}return ans;

}

int Del(int x){

add(a[x], -1);

int y1 = suml[x], y2 = sumr[x];

return ask(y2) - ask(y1-1);

}

int Ins(int x){

int y1 = suml[x], y2 = sumr[x];

int ans = ask(y2) - ask(y1-1);

add(a[x], 1); return ans;

}

int main()

{

scanf("%d %d %d", &n, &m, &K);

int cnt = 0;

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

scanf("%d", &a[i]);

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

b[++ cnt] = a[i] + K;

b[++ cnt] = a[i] - K;

}Mx = 3 \* n;

sort(b + 1, b + cnt + 1);

int pos = unique(b + 1, b + cnt + 1) - b;

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

suml[i] = lower\_bound(b + 1, b + pos, a[i] - K) - b;

sumr[i] = lower\_bound(b + 1, b + pos, a[i] + K) - b;

a[i] = lower\_bound(b+1, b+pos, a[i]) - b;

}

block = (int)sqrt(n);

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

scanf("%d %d", &P[i].l, &P[i].r);

P[i].id = i;

}

sort(P + 1, P + m + 1);

int l = 1, r = 0;

int ans = 0;

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

while(P[i].l > l){ans -= Del(l); l ++; }

while(P[i].l < l){l --; ans += Ins(l); }

while(P[i].r > r){r ++; ans += Ins(r); }

while(P[i].r < r){ans -= Del(r); r --; }

Ans[P[i].id] = ans;

}

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

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

}

return 0;

}

## 主席树

求解区间第k大数

const int maxn = 2e5+10;

struct Node{

int l, r, sum;

}T[maxn<<5];

int n, m, q, tot;

int a[maxn], b[maxn], root[maxn];

void change(int l, int r, int &x, int y, int pos){

T[++tot] = T[y]; T[tot].sum = T[y].sum + 1; x = tot;

if(l == r) return;

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

if(mid >= pos) change(l, mid, T[x].l, T[y].l, pos);

else change(mid+1, r, T[x].r, T[y].r, pos);

}

int query(int l, int r, int x, int y, int k){

if(l == r) return l;

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

int len = T[T[y].l].sum - T[T[x].l].sum;

if(len >= k) query(l, mid, T[x].l, T[y].l, k);

else query(mid + 1, r, T[x].r, T[y].r, k - len);

}

int main()

{

tot = 0;

scanf("%d %d", &n, &q);

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

scanf("%d", &a[i]); b[i] = a[i];

}

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

int m = unique(b+1, b+n+1) - b - 1;

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

int x = lower\_bound(b+1, b+m+1, a[i]) - b;

change(1, n, root[i], root[i-1], x);

}

while(q --){

int l, r, k; scanf("%d %d %d", &l, &r, &k);

int ans = query(1, n, root[l-1], root[r], k);

printf("%d\n", b[ans]);

}

return 0;

}

## 单调栈

单调栈的维护是 O(n) 级的时间复杂度，因为所有元素只会进入栈一次，并且出栈后再也不会进栈了。

单调栈的性质：

1.单调栈里的元素具有单调性

2.元素加入栈前，会在栈顶端把破坏栈单调性的元素都删除

3.使用单调栈可以找到元素向左遍历第一个比他小的元素，也可以找到元素向左遍历第一个比他大的元素。

问题描述：求区间和乘以区间最小值中的最大值问题

n<5e5 0<=a[i]<1e5,数组元素都是非负数

思路：对于每一个元素，假定该元素就是他所在区间中的最小元素，那么只要找到满足要求的最大区间即可（因为元素都是非负的，所以区间长度越大越好），向左遍历直到出现小于该元素停止（即为左区间），向右遍历直到出现小于该元素停止。

使用单调栈正向、反向各遍历一次数组，即可找到对应的左右区间

const int maxn = 5e5+100;

int n;

ll pre[maxn];

struct node{

int val, s, e;

};

vector<node>v(maxn);

int main()

{

mes(pre, 0);

scanf("%d", &n);

pre[0] = 0;

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

scanf("%d", &v[i].val);

pre[i] = pre[i-1]+v[i].val;

v[i].s = i; v[i].e = i;

}

stack<int>s;

int i = 0;

while(i <= n){

if(s.empty() || v[i].val > v[s.top()].val){

s.push(i);

i ++;

}

else {

v[i].s = v[s.top()].s;

s.pop();

}

}

while(!s.empty()){

s.pop();

}

i = n;

while(i >= 1){

if(s.empty() || v[i].val > v[s.top()].val){

s.push(i);

i --;

}

else {

v[i].e = v[s.top()].e;

s.pop();

}

}

ll ans = -inf;

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

ans = max(ans, v[i].val\*(pre[v[i].e]-pre[v[i].s-1]));

}

cout<<ans<<endl;

return 0;

}

若果此题中数组元素有正有负，则先用单调栈预处理出每个元素对应的左右区间，再用线段树求解包含节点元素的最大连续字段和

tree[p].lmax=max(tree[p<<1].lmax,tree[p<<1].sum+tree[p<<1|1].lmax);

tree[p].rmax=max(tree[p<<1|1].rmax,tree[p<<1|1].sum+tree[p<<1].rmax);

tree[p].sum = tree[p<<1].sum+tree[p<<1|1].sum;

tree[p].dat=max(max(tree[p<<1].dat,tree[p<<1|1].dat),tree[p<<1].rmax+

tree[p<<1|1].lmax);

## 单调队列

两种：单调递增，单调递减。

单调队列来解决问题，一般都是需要得到当前的某个范围内的最小值或最大值。

POJ2823:给定长度为n(n<1e6)的数组，求其以k为长度的子区间的最大最小值

### 单调队列优化

O(n)，因为每一个元素只会入队和出队一次

const int maxn = 1e6+10;

int n, k;

struct node{

int val, id;

}v[maxn];

int a[maxn], minans[maxn], maxans[maxn];

void getmin(){///找最小值

int head = 1, tail = 0;

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

while(head <= tail && a[i] <= v[tail].val)tail --;

v[++tail].val = a[i]; v[tail].id = i;

}

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

while(head <= tail && a[i] <= v[tail].val)tail --;

v[++tail].val = a[i]; v[tail].id = i;

while(v[head].id < i-k+1)head ++;

minans[i] = v[head].val;

}

}

void getmax(){///找最大值

int head = 1, tail = 0;

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

while(head <= tail && a[i] >= v[tail].val)tail --;

v[++tail].val = a[i]; v[tail].id = i;

}

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

while(head <= tail && a[i] >= v[tail].val)tail --;

v[++tail].val = a[i]; v[tail].id = i;

while(v[head].id < i-k+1)head ++;

maxans[i] = v[head].val;

}

}

int main(){

scanf("%d %d", &n, &k);

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

scanf("%d", &a[i]);

}

getmin(); getmax();

printf("%d", minans[k]);

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

printf(" %d", minans[i]);

}

printf("\n");

printf("%d", maxans[k]);

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

printf(" %d", maxans[i]);

}

printf("\n");

return 0;

}

### 优先队列优化

O(nlog n)

const int maxn = 1e6+10;

int n, k;

struct node{

int val, id;

}a[maxn];

struct cmp1{

bool operator () (const node &a, const node &b){

return a.val > b.val;///小的优先级高

}

};

struct cmp2{

bool operator () (const node &a, const node &b){

return a.val < b.val;///大的优先级高

}

};

priority\_queue<node, vector<node>, cmp1>qmin;

priority\_queue<node, vector<node>, cmp2>qmax;

int ansa[maxn], ansb[maxn];

int main(){

scanf("%d %d", &n, &k);

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

scanf("%d", &a[i].val);

a[i].id = i;

}

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

qmin.push(a[i]);

qmax.push(a[i]);

}

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

qmin.push(a[i]);

node p = qmin.top();

while(p.id < i-k+1 || p.id > i){

qmin.pop();

p = qmin.top();

}

ansa[i] = p.val;

qmax.push(a[i]);

node q = qmax.top();

while(q.id < i-k+1 || q.id > i){

qmax.pop();

q = qmax.top();

}

ansb[i] = q.val;

}

printf("%d", ansa[k]);

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

printf(" %d", ansa[i]);

}

printf("\n");

printf("%d", ansb[k]);

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

printf(" %d", ansb[i]);

}

printf("\n");

cout<<endl;

}

## 平衡树

### Treap实现名词树

P3369 【模板】普通平衡树

1、插入xx数

2、删除xx数(若有多个相同的数，因只删除一个)

3、查询xx数的排名(排名定义为比当前数小的数的个数+1+1。若有多个相同的数，因输出最小的排名)

4、查询排名为xx的数

5、求xx的前驱(前驱定义为小于xx，且最大的数)

6、求xx的后继(后继定义为大于xx，且最小的数)

int ans = 0;

struct Treap{

Treap \*ch[2];

int r, v, s;///r->随机优先数, v->权值, s->节点总数

int w; ///值为 v 的个数

Treap(int v):v(v){ch[0] = NULL; ch[1] = NULL; r = rand(); s = 1; w = 1; }///构造函数

int cmp(int x) const {

if(x == v) return -1;

return x < v ? 0 : 1;

}

void maintain(){

s = w;

if(ch[0] != NULL) s += ch[0]->s;

if(ch[1] != NULL) s += ch[1]->s;

}

}\*Root;

void Rotate(Treap\* &root, int d){

Treap \*tmp = root->ch[d^1]; root->ch[d^1] = tmp->ch[d]; tmp->ch[d] = root;

root->maintain(); tmp->maintain(); root = tmp;

}///Rotate(root, d);

void Insert(Treap\* &root, int x){

if(root == NULL){

root = new Treap(x); //root->ch[0] = root->ch[1] = NULL;

}else {

int d = root->cmp(x);

if(d == -1){root->w ++; root->s ++; return;}

Insert(root->ch[d], x);

if(root->ch[d]->r > root->r) Rotate(root, d^1);

}

root->maintain();

}///Insert(root, x);

void Remove(Treap\* &root, int x){

int d = root->cmp(x);

if(d == -1){///找到对应元素

if(root->w > 1){root->w --; root->s --; return;}

Treap\* u = root;

if(root->ch[0] != NULL && root->ch[1] != NULL) {

int d2 = (root->ch[0]->r > root->ch[1]->r ? 1 : 0);

Rotate(root, d2); Remove(root->ch[d2], x);

}

else {

if(root->ch[0] == NULL) root = root->ch[1];

else root = root->ch[0];

delete u;

}

}

Else Remove(root->ch[d], x);

if(root != NULL) root->maintain();

}///Remove(root, x);

bool Find(Treap\* root, int x){

while(root != NULL){

int d = root->cmp(x);

if(d == -1) return true;

return Find(root->ch[d], x);

}return false;

}///Find(root, x);

void Print(Treap\* root){

if(root == NULL) return;

Print(root->ch[0]);

printf("%d ", root->v);

Print(root->ch[1]);

}

int getRank(Treap\* root, int x){

if(root == NULL) return 0;

if(root->v > x) return getRank(root->ch[0], x);

int cnt = root->ch[0] == NULL ? 0 : root->ch[0]->s;

if(root->v == x) return cnt + 1;

if(root->v < x){

if(root->ch[1] != NULL){

return root->w + cnt + getRank(root->ch[1], x);

}

} return 0;

}

int getKth(Treap\* root, int k){

if(root == NULL || k < 0 || k > root->s) return 0;

int cnt = root->ch[0] == NULL ? 0 : root->ch[0]->s;

if(cnt < k && cnt + root->w >= k) return root->v;

else if(cnt >= k) return getKth(root->ch[0], k);

else return getKth(root->ch[1], k - root->w - cnt);

}

void getPre(Treap\* root, int x){

if(root == NULL) return ;

if(root->v < x) ans = root->v, getPre(root->ch[1], x);

else getPre(root->ch[0], x);

}

void getSuff(Treap\* root, int x){

if(root == NULL) return ;

if(root->v > x) ans = root->v, getSuff(root->ch[0], x);

else getSuff(root->ch[1], x);

}

int main()

{

Root = NULL;

int n; scanf("%d", &n);

while(n --){

int op, x; scanf("%d %d", &op, &x);

if(op == 1){

Insert(Root, x);

}else if(op == 2){

Remove(Root, x);

} else if(op == 3){

printf("%d\n", getRank(Root, x));

}else if(op == 4){

printf("%d\n", getKth(Root, x));

}else if(op == 5){

getPre(Root, x); printf("%d\n", ans);

}else if(op == 6){

getSuff(Root, x); printf("%d\n", ans);

}

}

return 0;

}

# 数学

## 基础数论

### 整除

整数。如果存在，使得，那么就说b可被a整除，记作,且称b是a的倍数,a是b的约数(也可称为除数，因数)。b不能被a整除记作a\|b。

重要定理：

1.a|b且b|c，则a|c

1. a|b,且a|c，则对于任意

3.s设a!=0,，那么a整除b的充要条件是a整除c。

例题：a,b非零整数，且有整数x,y使得ax+by=1。

证明： 1)若a|n且b|n，则ab|n

1. 若a|bn,则a|n。

最大公约数:gcd(a,b) 最小公倍数:lcm(a,b)

lcm(a,b) = a\*b/gcd(a,b)

### 求和公式

两个求和公式：（数学归纳法可证）

平方和求和：

立方和求和：

调和级数

当n很大时,可以用此公式近似

(γ是欧拉常数γ=0.57721566490153286060651209...，

e自然数对数e = 2.7182818284)

求整数n的前k位，令,则d是x的整数部分，i是x的小数部分，，则n的前k位即为（例：123456取前3位，由公式求得d=5,,，取整数部分即123）

### 整除分块

对于，通过大表发现有很多具有相同的值，且他最后一次出现的位置就是n/(n/i)。

证明：

ll getans(ll n){

ll ans = 0;

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

r = n/(n/l);

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

}

return ans;

}

### 完全数

如果一个数恰好等于它的因子之和，则称该数为完全数。

前5位完全数6、28、496、8128、33550336

### 阶乘



关于阶乘的几个小性质：

阶乘位数：

斯特灵公式：，当n很大时可以用斯特灵公式近似

阶乘尾数零的个数：对于N!进行质因数分解，零的个数就素因子2和5的最小个数，由于2的个数总大于5的个数，实际上也就是5的个数，求法如下：



ll cnt = 0;

for(ll i = 5; i<= n; i\*= 5){

cnt += (n/i);

}

如果给定末尾零的个数，也能判断是否存在相应的的阶乘使其末尾的零的个数满足给定的数

用二分思想解决：

ll getcnt(ll n){

ll cnt = 0;

for(ll i = 5; i <= n; i \*= 5){

cnt += n / i;

}

return cnt;

}

ll solve(ll n){

ll low = 0, high = maxn;

ll mid = (low + high) >> 1;

while(low < high){

ll mid = (low + high) >> 1;

if(getcnt(mid) >= n){

high = mid;

}

else {

low = mid + 1;

}

}

if(getcnt(low) == n)return low; //求得low就是满足条件的最小的阶乘

return 0; //则表示不存在

}

### 匹克定理

2S = 2n + m – 2, 其中 S 是多边形面积，n 是多边形内部格点数目，m 是多边形边界上的格点数目。

### 梅森数

梅森数(Mersenne number)又称麦森数，是指形如2^p－1的[正整数](https://baike.baidu.com/item/%E6%AD%A3%E6%95%B4%E6%95%B0" \t "https://baike.baidu.com/item/%E6%A2%85%E6%A3%AE%E6%95%B0/_blank)，其中指数p是素数，常记为Mp 。若其是[素数](https://baike.baidu.com/item/%E7%B4%A0%E6%95%B0/115069" \t "https://baike.baidu.com/item/%E6%A2%85%E6%A3%AE%E6%95%B0/_blank)，则称为[梅森素数](https://baike.baidu.com/item/%E6%A2%85%E6%A3%AE%E7%B4%A0%E6%95%B0/816141)。

梅森素数表



## 斐波那契数列

定义：

快速求斐波那契数列方法：（矩阵快速幂）

令

则

故

令，则有递推公式的

有关斐波那契的公式：

1、前缀和：数学归纳法即可证明

2、

### 矩阵快速幂

void mul(ll f[2], ll a[2][2]){

ll c[2]; mes(c, 0);

for(int j = 0; j < 2; j ++){

for(int k = 0; k < 2; k ++){

c[j] = (c[j] + f[k]\*a[k][j]) % mod;

}

}

mec(f, c);

}

void mulself(ll a[2][2]){

ll c[2][2]; mes(c, 0);

for(int i = 0; i < 2; i ++){

for(int j = 0; j < 2; j ++){

for(int k = 0; k < 2; k ++){

c[i][j] = (c[i][j] + a[i][k] \* a[k][j]) % mod;

}

}

}

mec(a, c);

}

ll ksm(ll n){

ll f[2] = {0, 1};

ll a[2][2] = {{0, 1}, {1, 1}};

while(n){

if(n & 1) mul(f, a);

mulself(a);

n >>= 1;

}

return f[0];

}

较完整的写法

#define MOD (ll)(1000000007)

#define mod(x) (((x%MOD)+MOD)%MOD)

#define mod(x, p) (((x%p)+p)%p)

ll d;

int k;

typedef struct Matrix{

ll a[30][30];

};

Matrix init(){

Matrix A;

for(int i = 0; i < 30; i ++){

A.a[i][i] = 1;

}

return A;

}

ll ksm\_mul(ll a, ll b, ll p){

ll res = 0;

while(b){

if(b & 1) res = mod(res+a, p);

a = mod(a+a, p);

b >>= 1;

}

return res;

}

ll ksm(ll a, ll n, ll p){

ll res = 1;

while(n){

if(n & 1) res = ksm\_mul(res, a, p);

a = ksm\_mul(a, a, p);

n >>= 1;

}

return res;

}

Matrix operator \* (const Matrix& A, const Matrix& B){

Matrix C; mes(C.a, 0);

int n = 27;

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

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

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

C.a[i][j] = (C.a[i][j] + A.a[i][k]\*B.a[k][j]%MOD)%MOD;

}

}

}

return C;

}

Matrix ksm\_Matrix(Matrix A, ll n){

Matrix Res = init();

while(n){

if(n & 1) Res = Res \* A;

A = A \* A;

n >>= 1;

}

return Res;

}

ll solve(ll n, int k){

Matrix Res, A, Tmp;

mes(Res.a, 0); mes(A.a, 0);

ll p = ksm((ll)k, MOD-2, MOD);

for(int i = 1; i <= k; i ++)A.a[1][i] = p;

A.a[1][k+1] = 1;

for(int i = 2; i <= k; i ++)A.a[i][i-1] = 1;

A.a[k+1][k+1] = 1;

for(int i = 1; i <= k; i ++)Res.a[i][1] = k;

Res.a[k+1][1] = 1;

n -= k;

Tmp = ksm\_Matrix(A, n);

Res = Tmp \* Res;///一定注意矩阵相乘的顺序

return Res.a[1][1];

}

int main(){

scanf("%lld %d", &d, &k);

ll ans;

ans = solve(d, k);

printf("%lld\n", ans);

return 0;

}

## 杨辉三角的奇偶性

例题：求杨辉三角前n行中偶数的个数（n是10的50次方）

(1)可根据公式：

a[2n] = 3a[n] + n\*(n+1)/2

a[2n+1] = 2a[n] + a[n+1] + n\*(n+1)/2

(2)可递归求解，通过简单大表可以发现，当n = 2的k次幂时，这一行全是奇数，然后就可以递归求解剩下的行

JAVA完整代码

import java.math.\*;

import java.util.\*;

publicclass Main {

publicstatic BigInteger *Zero* = new BigInteger("0");

publicstatic BigInteger *One* = new BigInteger("1");

publicstatic BigInteger *Two* = new BigInteger("2");

publicstatic BigInteger *Three* = new BigInteger("3");

publicstatic BigInteger *Four* = new BigInteger("4");

publicstatic BigInteger KSM(BigInteger a, intn) {

BigInteger res = *One*;

while(n != 0) {

if(n % 2 == 1) res = res.multiply(a);

a = a.multiply(a);

n /= 2;

}

returnres;

}

///二分查找2的最大次幂

publicstatic BigInteger solve(BigInteger x) {

if(x.compareTo(*Two*) <= 0) return*Zero*;

if(x.compareTo(*Three*) == 0) return*One*;

if(x.compareTo(*Four*) == 0) return*One*;

BigInteger res = new BigInteger("1");

intk = *getpow*(x);

// System.out.println(k);

BigInteger pk = *KSM*(*Two*, k);

BigInteger pkk = x.subtract(pk);

BigInteger sumk = pk.multiply(pk.add(*One*)).divide(*Two*);

BigInteger sumk\_odd = *KSM*(*Three*, k);

BigInteger sumk\_even = sumk.subtract(sumk\_odd);

sumk\_even = sumk\_even.add(pk.multiply(*Two*).subtract(pkk.add(*One*)).multiply(pkk).divide(*Two*));

res = sumk\_even.add(*solve*(pkk).multiply(*Two*));

returnres;

}

publicstaticvoid main(String[] args) {

Scanner cin = new Scanner(System.*in*);

BigInteger n = new BigInteger("1");

BigInteger ans = new BigInteger("1");

///JAVA读入，直到文件结尾

while(cin.hasNextBigInteger()) {

n = cin.nextBigInteger();

n = n.add(*One*);

ans = *solve*(n);

System.*out*.println(ans);

}

}

}

## 三角形

三角形面积：

海伦公式：，

正弦公式：

三角形外接圆半径R求法，

三角形内切圆r半径求法，

## 构造勾股数

给定三角形一条边a，构造出的一个直角三角形

1.a=2k, 可以构造出边为,，的直角三角形

2.a=2k+1,可以构造出边为,，的直角三角形

## 欧几里得算法

///欧几里得求gcd

int gcd(int m, int n){

if(!n) return m;

return gcd(n, m%n);

}

## 扩展欧几里得

///求解线性方程 ax + by = gcd(a, b)

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

if(!b){

x = 1; y = 0;

return a;

}

int d = exgcd(b, a%b, x, y);

int z = x;

x = y; y = z - y\*(a/b);

return d;

}

因此对于任意线性方程：ax + by = c (令d = gcd(a, b))

当d | c时有解，并且其解的通项公式可表示为：

X = x0\*(c/d) + k\*(b/d)

Y = y0\*(c/d) + k\*(a/d) （k ∈ Z）

否则方程无解

### 线性同余方程

ax≡b(mod m) 可转换成 ax+my=b 可利用扩展欧几里得

### 高次同余方程（BSGS算法）

方程a^x≡b(mod p)，其中a与p互质

令 x = i\*t-j, t = ceil(sqrt(p)),0<=j<=t-1,则方程可转换成

(a^t)^i≡b\*a^j(mod p)

Baby Step, Giant Step算法

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

unordered\_map<int, int>mp; mp.clear();

int t = (int)sqrt(p)+1;

b %= p;

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

int val = (ll)b\*ksm(a, j, p)%p;

mp[val] = j;

}

a = ksm(a, t, p);

if(a == 0)return b == 0 ? 1 : -1;

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

int val = ksm(a, i, p);

int j = mp.find(val) == mp.end() ? -1 : mp[val];

if(j >= 0 && i\*t-j>=0)return i\*t-j;

}

return -1;

}

用hash表处理比map快

const int mod = 1048573;

int head[mod];

int cnt;

struct Hash{

int val, id, next;

}H[mod+10];

void init(){

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

H[i].val = -1;

H[i].id = H[i].next = 0;

}

mes(head, 0);

cnt = 0;

}

void Insert(int x, int pos){

int k = x % mod;

H[++cnt].val = x; H[cnt].id = pos;

H[cnt].next = head[k];

head[k] = cnt;

}

int Find(int x){

int k = x % mod;

for(int i = head[k]; i; i = H[i].next){

if(H[i].val == x)return H[i].id;

}

return -1;

}

LL ksm(int a, int n){

LL res = 1;

while(n){

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

a = (LL)a \* a % p;

n >>= 1;

}

return res;

}

LL BSGS(){

int m = (int)sqrt(p) + 1;

if(b == 1)return 0;

if(a == b)return 1;

if(!b){

if(a == 0) return 1;

return -1;

}

LL x=1;

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

x=x\*a%p;

Insert(x,i);

}

LL inv=1;

int inv2=ksm(a,p-m-1)%p;

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

int k=i\*m;

if(inv==-1) return -1;

int ans=inv\*b%p;

int jgy=Find(ans);

if(~jgy){

return k+jgy;

}

inv=1ll\*inv\*inv2%p;

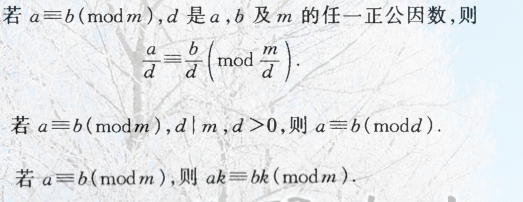
}

return -1;

}

扩展BSGS

方程a^x≡b(mod p)，其中a与p不互质



## 康拓展开&康拓逆展开

康拓展开：对于任意给定1~n的一个排列能O(n)求其排列是n的全排列中（按字典序）第几小的排列。

康拓逆展开：能O(n)的求出n的全排列中（按字典序）第i小的排列

int a[12], fac[12]; bool vis[12];

void getfac(int n){

fac[0] = 1;

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

}

int cantor(int n){///康拓展开

int ret = 0;

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

int cnt = 0;

for(int j = i + 1; j < n; ++j) if(a[i] > a[j]) ++cnt;

ret += cnt\*fac[n-1-i];

} return ret;

}

void decantor(int x, int n){///康拓逆展开

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

int tot = 0;

for(int i = n; i >= 1; --i){

int q = x/fac[i - 1], cnt = 0;

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

if(vis[j]) continue;

if(++cnt > q){

a[tot++] = j; vis[j] = true; break;

}

} x = x%fac[i - 1];

}

}

## 组合数学

加法原理：分类思想，a1+a2+...+an

乘法原理：分布思想，a1\*a2\*...\*a3

排列数：从n个元素依次取m个元素排成一列的所有排列的方法P(n, m)

组合数：从n个不同元素中取出m个组成一个集合的所有取法C(n, m)

性质：

1.C(n, m) = C(n, n-m);

2.C(n, m) = C(n-1, m) + C(n-1, m-1);

3.C(n, 0) + C(n, 1) + C(n, 2) + ... + C(n, n) = 2^n;

### Lucas定理

当n,m很大，p是较小的质数，例如：1007

ll multi(ll a, ll b, ll p){///求a\*b%p

ll res = 0;

while(b){

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

a = (a + a) % p;

b >>= 1;

} return res;

}

ll ksm(ll a, ll n, ll p){///求(a^n)%p

ll res = 1;

while(n){

if(n &1) res = multi(res, a, p);

a = multi(a, a, p);

n >>= 1;

} return res;

}

ll Comb(ll n, ll m, ll p){///求组合数C(n, m)

if(m == 0)return 1;

if(m == 1)return n % p;

if(m > n-m)m = n - m;

ll la = 1, lb = 1, res = 1;

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

la = la\*(n-i) % p; lb = lb\*(m-i) % p;

} return res = la\*ksm(lb, p-2, p) % p;

}

ll Lucas(ll n, ll m, ll p){///Lucas定理

if(m == 0) return 1;

return Lucas(n/p, m/p, p)\*Comb(n%p, m%p, p)%p;

}

卡特兰数列（Catlan）

定义：

计算公式：

实际应用：

例题：1、给定n个0和n个1,他们按照某种顺序排成长度为2n的序列，满足任意前缀中0的个数都不少于1的个数的序列的数量为：

例题2、给定n个0和m个1,(m <= n)他们按照某种顺序排成长度为n + m的序列，满足任意前缀中0的个数都不少于1的个数的序列的数量为：

### FerrersFerrers图像

一个从上而下的n层格子， m(i) 为第i 层的格子数，

当m(i) >= m(i+1)，其中（i=1,2,…，n-1）,即上层的

格子数不少于下层的格子数时(weakly decreasing )，

称之为Ferrers图像（Ferrers diagram）。

利用Ferrers图像可得关于整数拆分的几个结果。

(a)整数n拆分成最大数为k的拆分数，和数n拆分成k个数的和的拆分数相等。

解释：因整数n拆分成k个数的和的拆分可用一k行的图像表示。所得的Ferrers图像的共轭图像最上面一行有k个格子。

(b)整数n拆分成最多不超过m个数的和的拆分数，和n拆分成最大不超过m的拆分数相等。

(c)整数n拆分成互不相同的若干奇数的和的拆分数,和n拆分成自共轭的Ferrers图像的拆分数相等。

### 小球放盒模型

一种很有用的模型，很多技术问题都能转换成小球放盒模型

将n个小球放入m个盒子中，即小球放盒子问题，可分为以下的八种情况。

1.盒子相同，球相同，不允许空。

　　这个其实就相当于整数划分问题，就是把球看做数字，把盒子看做每一份。设f[i][j]为考虑了前i个，分成了j份，转移方程为:

f[i][j]=f[i−1][j−1]+f[i−j][j]

2.盒子相同，球相同，允许空。

这个东西就是刚刚求的那个整数划分的前缀和。

∑f[i][j](j = 0, 1, ...m)

3.盒子相同，球不同，不允许空。

　　第二类斯特林数，设f[i][j]表示处理了前i个球，用了j个盒子，转移方程为：

f[i][j]=f[i−1][j−1]+f[i−1][j]∗j  
　　这个东西的含义就是讨论当前新加入的球要放哪，第一种放法就是再开一个盒子：f[i][j]+=f[i−1][j−1]，第二种方法就是在前面的j个盒子里随便挑一个放进去，因为有j个盒子，所以f[i][j]+=f[i−1][j]∗j。

4.盒子相同，球不同，允许空。

　　第二类斯特林数的前缀和。其实就是看最后放了几个盒子，剩下的都是空的，加起来即可。

5.盒子不同，球相同，不允许空。

　　比较经典的隔板法，把m盒子看m−1个木板，然后要插入到n−1个空隙里，所以答案就是C(n-1, m-1)。

6.盒子不同，球相同，允许空。

　　这个就是上一个的变形，可以添加m个小球放到这m个盒子里，这样就保证了盒子非空，答案就是C(n+m-1, m-1)。

7.盒子不同，球不同，不允许空。

　　这个是第二类斯特林数f[n][m]\*m!，具体怎么证明不太会，好像就是把斯特林数换一种写法，然后消掉有序性。

8.盒子不同，球不同，允许空。

对于每一个小球都可以放到m个盒子里，答案就是m^n。

## 约瑟夫环问题

问题描述：N个人围成一圈，第一个人从1开始报数，报M的将被杀掉，下一个人接着从1开始报。如此反复，最后剩下一个，求最后的胜利者。

递推公式： f(N,M)=(f(N − 1, M) + M) % N

f(N, M) 表示，N个人报数，每报到M时杀掉那个人，最终胜利者的编号

f(N − 1, M)表示，N-1个人报数，每报到M时杀掉那个人，最终胜利者的编号

int ans = 0;

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

ans = (ans + m) % i;

}

printf("%d\n", ans + 1);

## 积性函数：

算术基本定理：任何一个正整数都可分解为

定义：如果当a, b互质时，有f(a\*b) = f(a)\*f(b),那么称函数f为积性函数。

若a,b为任意正整数时，满足f(a\*b) = f(a)\*f(b)，那么称函数f为完全积性函数

常见积性函数：

### **欧拉函数**

定义：1~N中与N互质的数的个数被称为欧拉函数，记为φ(N)。



性质： 1。任意 n>1, 1~n中与n互质的数的和为 n\*φ(n)/2。

2.若a ,b互质，则φ(a\*b) = φ(a) \*φ(b)。

3.若f是积性函数，且在算术基本定理中n = 

则f(n) = 。

4.若p|n且p^2|n, 则φ(n) = φ(n/p)\*p。

5.若p|n且p^2|\n,则φ(n) = φ(n/p)\*(p-1)。

6.

Fermat-Euler定理： ,

求解：

NENU-500

1、简单方法(o(sqrt(n))

int Phi(int n){

int ans = n;

for(int i = 2; i <= sqrt(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);

return ans;

}

2、O(n)线性筛

### **莫比乌斯函数**

定义：。

莫比乌斯函数求和

### 约数个数函数

定义：表示正整数n的所有整除数的个数，记为d(n),也记为。



### 约数和函数

定义：表示正整数n的所有整除数之和，。



函数前缀和，，整除分块O(sqrt(n)

线性筛约数和：

void sieze(int n){

m = 0; mes(v, 0); d[1] = 1;

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

if(v[i] == false) {

prime[++ m] = i; d[i] = i + 1; low[i] = i;

}

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

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

if(i % prime[j]) {

d[i \* prime[j]] = d[i] \* d[prime[j]];

low[i \* prime[j]] = prime[j];

}

else {

if(i == low[i]) d[i \* prime[j]] = d[i] + i \* prime[j];

else d[i \* prime[j]] = d[i / low[i]] \* d[low[i] \* prime[j]];

low[i \* prime[j]] = low[i] \* prime[j];

break;

}

}

}

}

### 反素数

函数表示n的约数个数， 若满足公式 ，则称n为反素数。

例如 1， 2， 4， 6 …, 令

引理1：不超过 2e9 的数的质因子分解中，最多有 10 个不同的质因子，且各个质因子的指数和不超过30。  
引理2：题目要求的最大反素数，实际上是求不超过 N 的数中因子数最多的数的集合中最小的那个数。

引理3：通过引理 2 以及交换证明法可以得出，各个质因子指数必须单调递减。

即

int p[] = {0, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29};

int ans, mx; ll n;

void dfs(int now, int mul, int cnt){

if(now >= 11) return;

if(cnt > mx){

mx = cnt; ans = mul;

}

if(cnt == mx && mul < ans) ans = mul;

for(int i = 1; i <= 32; i ++){

if(1ll \* mul \* p[now] > n) break;

mul \*= p[now];

dfs(now+1, mul, cnt \* (i + 1));

}

}//dfs(1, 1, 1)

### 元函数

定义：记为Ɛ(n), Ɛ(n) = [n = 1],当且仅当n = 1 时，Ɛ(n) = 1，否则均为0

### 恒等函数：

定义：I(n) ，对于任意 n 都是1, 即 I(n) = 1;

### 单位函数

定义: id(n) ， id(n) = n

### 狄利克雷卷积

定义：两个数论函数 f(n) 和 g(n)做卷积记为

卷积满足运算性质：

1. 交换律： f \* g = g \* f
2. 结合律： (f \* g) \* h = f \* (g \* h)
3. 分配率: (f + g) \* h = f \* h + g \* h

几个重要的卷积：

莫比乌斯反演定理证明：

若存在, 则有

证明：, 因此，等价于

即得

莫比乌斯函数与欧拉函数之间的关系

根据公式：

证明：，等价于

即得：，即

也可得

### 线性筛约数个数

根据公式：

设num(i)表示i的最小质因数的个数

容易的d(1)=1,d(2)=2,质数p有d(p) = 2

若质数p不能整除n/p,则d(n)=d(n/p)\*d(p)

若质数p能整除n/p,则表明n/p中存在质因子p，则d(n)=d(n/p)/(num(n/p)+1)\*(num(n/p)+2)

bool v[maxn];

int prime[maxn], d[maxn], num[maxn];

ll pre[maxn];

int m, n;

void sieze(int n){

mes(v, 0); m = 0 ; d[1] = 1;

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

if(!v[i]){

v[i]= 1; prime[++m] = i; d[i] = 2; num[i] = 1;

}

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

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

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

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

d[i\*prime[j]] = d[i]/(num[i]+1)\*(num[i]+2);

num[i\*prime[j]] = num[i]+1;

}

else {

d[i\*prime[j]] = d[i]\*d[prime[j]];

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

}

}

}

}

## 杜教筛

妙处：它可以在在时间复杂度内求出任意积性函数前缀和。

积性函数如何求解？

构造积性函数，令，

等价于

移项得：

构造出较好的函数使得能很方便求出，则S(n)就可以通过整除分块就得，时间复杂度为

几个简单的栗子：

1. ,

构造,

即得：

构造

即得：

实现代码：

const int mod = 1e9 + 7;

const int maxn = 5e6+100;

int prime[maxn];

ll phi[maxn]; int mu[maxn]; bool v[maxn];

int m;

tr1::unordered\_map<int, ll>sum\_phi;

tr1::unordered\_map<int, int>sum\_mu;

void sieze(int n){ ///线性筛欧拉函数、莫比乌斯函数

mes(v, 0); m = 0; phi[1] = mu[1] = 1;

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

if(!v[i]){

prime[++ m] = i; phi[i] = i - 1; mu[i] = -1;

}

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

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

if(i % prime[j]){

phi[i \* prime[j]] = phi[i] \* (prime[j] - 1);

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

}

else {

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

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

break;

}

}

} phi[0] = mu[0] = 0;

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

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

}

}

ll djs\_phi(int n){

if(n <= maxn - 10) return phi[n];

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

ll ans = 0;

for(int l = 2, r; l <= n; l = r + 1){

r = n / (n / l);

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

} sum\_phi[n] = (ull)n \* (n + 1ll) / 2ll - ans;

return sum\_phi[n];

}

int djs\_mu(int n){

if(n <= maxn - 10) return mu[n];

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

int ans = 0;

for(int l = 2, r; l <= n; l = r + 1){

r = n / (n / l);

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

} sum\_mu[n] = 1 - ans; return sum\_mu[n];

}

int main()

{

sieze(maxn-10);

int T; scanf("%d", &T);

while(T --){

int n; scanf("%d", &n);

printf("%lld %d\n", djs\_phi(n), djs\_mu(n));

}

return 0;

}

## 逆元

定义：对于整数a,b, 若a \* b ≡1 (mod p), 那么a 和 b互为模p意义下的逆元。

意义：(x/a)%p可以改写成x\*b%p

求逆元的三种方法：

1. 费马小定理：若p为质数时，有，所以则a^(p-2)是a在模p意义下的逆元。
2. 扩展欧几里得：a\*b ≡ 1(mod p) 等价于a\*b + k\*p=1(gcd(a, p)==1)

通过扩展欧几里得即可求出b

给定m, n, m, n互质 求最小正整数K满足 k\*m % n = 1

exgcd(m, n, x, y);

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

1. 递推公式法：

int inv[mod+5];

void getInv(int mod){

inv[1]=1;

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

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

}

## 阶乘逆元(O(n))

即得 , inv((n-1)!) = n \* inv(n!)

void init(int n){

f[0] = 1;

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

invf[n] = ksm(f[n], mod - 2, mod);

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

invf[i] = invf[i+1] \* (i + 1) % mod;

}

}

## 原根

定义：设m是正整数，a是整数，若a模m的阶等于φ(m)，则称a为模m的一个原根。即

定理1：可以证明，如果正整数(a,m) = 1和正整数 d 满足a^d≡1(mod m)，则 d 整除 φ(m)。因此Ordm(a)整除φ(m)。在例子中，当a= 3时，我们仅需要验证 3 的 1 、2、3 和 6 次方模 7 的[余数](http://baike.baidu.com/item/%E4%BD%99%E6%95%B0" \t "http://baike.baidu.com/_blank)即可。

定理2：记a的阶δ = Ordm(a)，则a^1，……a^(δ-1)模 m 两两不[同余](http://baike.baidu.com/item/%E5%90%8C%E4%BD%99" \t "http://baike.baidu.com/_blank)。因此当a是模m的原根时，a^1，……a^(δ-1)构成模 m 的[简化剩余系](http://baike.baidu.com/item/%E7%AE%80%E5%8C%96%E5%89%A9%E4%BD%99%E7%B3%BB" \t "http://baike.baidu.com/_blank)。

定理3：模m有原根的充要条件是m= 1,2,4,p,2p,p^n，其中p是奇质数，n是任意[正整数](http://baike.baidu.com/item/%E6%AD%A3%E6%95%B4%E6%95%B0" \t "http://baike.baidu.com/_blank)。

定理4：对正整数(a,m) = 1，如果 a 是模 m 的原根，那么 a 是整数模n乘法群（即加法群 Z/mZ的可逆元，也就是所有与 m [互素](http://baike.baidu.com/item/%E4%BA%92%E7%B4%A0)的正整数构成的[等价类](http://baike.baidu.com/item/%E7%AD%89%E4%BB%B7%E7%B1%BB)构成的乘法群）Zn的一个[生成元](http://baike.baidu.com/item/%E7%94%9F%E6%88%90%E5%85%83" \t "http://baike.baidu.com/_blank)。由于Zn有 φ(m)个元素，而它的生成元的个数就是它的可逆元个数，即 φ(φ(m))个，因此当模m有原根时，它有φ(φ(m))个原根。

vector<ll>v;

void fac(ll n){

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

if(n % i == 0){

v.push\_back(i);

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

}

} if(n > 1) v.push\_back(n);

}

ll ksm(ll a, ll n, ll p){

ll res = 1;

while(n){

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

a = a \* a % p; n >>= 1;

} return res;

}

int main()

{

ll n; scanf("%lld", &n);

fac(phi(n));

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

int flag = 1;

for(int j = 0; j < v.size(); j ++){

if(ksm(i, (phi(n))/v[j], n) == 1){flag = 0; break;}

}

if(flag){ printf("%lld\n", i); break;}

}

}

## 中国剩余定理（除数互质）

求解同余方程组，

其中m1, m2, ..., mk是两两互质的数。

求x的最小非负整数解

ll exgcd(ll a, ll b, ll &x, ll &y){

if(!b){x = 1; y = 0; return a;}

ll d = exgcd(b, a % b, x, y);

ll z = x; x = y; y = z - y \* (a / b);

return d;

}

ll china(){

ll M = 1, ans=0, mm, x, y;

for(int i = 1; i <= n; i ++) M \*= m[i];

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

mm = M / m[i];

exgcd(mm, m[i], x, y);

ans = (ans + a[i]\*mm\*x)%M;

} ans = (ans+M)%M;

return ans;

}

## 中国剩余定理（除数不互质）

   中国剩余定理求解同余方程要求模数两两互质，在非互质的时候其实也可以计算，这里采用的是合并方程的思想。

ll china(){

ll M, X;

M = m[1]; X = a[1];

ll A, B, C, x, y;

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

A = M; B = m[i]; C = a[i]-X;

ll gcd = exgcd(A, B, x, y);

///x即A相对于B的逆元

if(C%gcd)return -1;

ll mod = B / gcd;

x = ((x\*C/gcd)%mod+mod)%mod;

X = (X + M\*x)%(M/gcd\*m[i]);

M = M/gcd\*m[i];

} if(X)return X;

return M;

}

## FFT

高精度乘法，FFT模板题

const double pi = acos(-1.0);

const int maxn = 2e5+100;

struct Complex{

double real, image;/// real->实部, image->虚部

Complex(double real = 0.0, double image = 0.0):real(real), image(image){};

Complex operator + (const Complex& A){return Complex(real + A.real, image + A.image);}

Complex operator - (const Complex& A){return Complex(real - A.real, image - A.image);}

Complex operator \* (const Complex& A){

return Complex(real \* A.real - image \* A.image, real \* A.image + A.real \* image);

}

};

///交换完最终形式

void Change(Complex y[], int len){

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

if(i < t) swap(y[i], y[t]);

for(int j = len >> 1; (t ^= j) < j; j >>= 1);

}

}

void FFT(Complex y[], int len, int type){

Change(y, len);

for(int h = 2; h <= len; h <<= 1){

Complex wn = Complex(cos(pi\*2\*type/h), sin(pi\*2\*type/h));

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

Complex w = Complex(1.0, 0), t;

for(int j = 0; j < h >> 1; j ++, w = w \* wn){

t = y[i + j + (h >> 1)] \* w;

y[i + j + (h >> 1)] = y[i + j] - t;

y[i + j] = y[i + j] + t;

}

}

}

if(type == -1){

for(int i = 0; i < len; i ++) y[i].real /= len;

}

}

Complex a[maxn<<1], b[maxn<<1], c[maxn<<1];

char sa[maxn], sb[maxn];

int ans[maxn<<1];

int main()

{

int n; scanf("%d", &n);

scanf("%s %s", sa, sb);

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

a[n-1-i].real = 1.0 \* (sa[i] - '0');

b[n-1-i].real = 1.0 \* (sb[i] - '0');

}

int len ;

for(len = 1; len <= n + n; len <<= 1);

FFT(a, len, 1); FFT(b, len, 1);

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

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

}

FFT(c, len, -1);

mes(ans, 0);

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

ans[i] += (int)(c[i].real + 0.5);

if(ans[i] >= 10) {

ans[i+1] += ans[i] / 10;

ans[i] = ans[i] % 10;

}

}

int flag = 1;

for(int i = len; i >= 0; i --){///逆向输出

if(!ans[i] && flag) continue;

flag = 0;

printf("%d", ans[i]);

}

printf("\n");

return 0;

}

# 计算几何

## 二维基础

const double pi = acos(-1.0);

const double eps = 1e-6;

int dcmp(double x){

if(fabs(x) < eps) return 0; if(x > 0) return 1; return -1;

}

struct Point{

double x, y;

Point(double x = 0.0, double y = 0.0) : x(x), y(y){}

};

struct Line{

Point p; Vector v;

Line(Point p, Vector v) : p(p), v(v) {}

Point point(double t) {return p+v\*t;}

};

typedef Point Vector;

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

Vector operator - (Vector A, Vector 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 \* (double p, Vector A){return Vector(A.x\*p, A.y\*p);}

Vector operator / (Vector A, double p){return Vector(A.x/p, A.y/p);}

bool operator < (Vector A, Vector B){

return A.x < B.x || (A.x == B.x && A.y < B.y);

}

bool operator == (const Vector& A, const Vector& B){

return dcmp(A.x-B.x) == 0 && dcmp(A.y-B.y) == 0;

}

double AngleToRad(double x){///角度制转弧度制

return x \* pi / 180;

}

double RadToAngle(double x){///弧度制转角度制

return x \* 180 / pi;

}

double Dot(Vector A, Vector B){return A.x\*B.x + A.y\*B.y;}

double Length(Vector A){return sqrt(Dot(A, A));}

double Angle(Vector A, Vector B){

return acos(Dot(A, B) / (Length(A)\*Length(B)));

}

double Cross(Vector A, Vector B){return A.x\*B.y - A.y\*B.x;}

double Area(Point A, Point B, Point C){return 0.5\*fabs(Cross(B-A, C-A));}

Vector Rotate(Vector A, double rad){

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

}

Vector Normal(Vector A){///调用前确保A不是零向量

double L = Length(A);

return Vector(-A.y/L, A.x/L);

}

///使用前保证两直线有唯一交点， 即Cross(v, w)不为 0

Point GetLineIntersection(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 DistanceToLine(Line L, Point P){///点到直线的距离

return fabs(Cross(P-L.p, L.v)/Length(L.v));

}

double DistanceToSegment(Point P, Point A, Point B){

if(A == B) return Length(P-A);

Vector v1 = B - A, v2 = P - A, v3 = P - B;

if(dcmp(Dot(v1, v2)) < 0) return Length(v2);

if(dcmp(Dot(v1, v3)) > 0) return Length(v3);

return fabs(Cross(v1, v2) / Length(v1));

}

Point GetLineProjection(Line L, Point P){ ///求点P在直线L上的投影点

return L.p + L.v \* (Dot(L.v, P-L.p) / Dot(L.v, L.v));

}

Point GetMirrorPoint(Line L, Point P){ ///求点P关于L的对称点

Point Q = GetLineProjection(L, P);

return Q + (Q - P);

}

bool OnRays(Line L, Point P){ ///点P是否在射线上

return dcmp(Cross(P-L.p, L.v)) == 0 && dcmp(Dot(P-L.p, L.v)) > 0;

}

///点P是否在线段AB上

bool OnSegment(Point P, Point A, Point B){

return dcmp(Cross(P-A, P-B)) == 0 && dcmp(Dot(P-A, P-B)) < 0;

}

### 基本运算

定义向量

点积运算:

double Dot(Vector A, Vector B){return A.x\*B.x + A.y\*B.y;}

double Length(Vector A){return sqrt(Dot(A, A));}

double Angle(Vector A, Vector B){

return acos(Dot(A, B) / (Length(A)\*Length(B)));

}

叉积运算：

叉积大于0，表明向量 在 的左边

小于0， 表明向量 在 的又边

对于三角形的面积可以是 的一半

double Cross(Vector A, Vector B){return A.x\*B.y - A.y\*B.x;}

double Area(Point A, Point B, Point C){return 0.5\*fabs(Cross(B-A, C-A));}

对于三维向量 ,

向量旋转

A(x, y)逆时针旋转a 得B(x1, y1) = (x cosa – y sina, x sina + y cosa)

Vector Rotate(Vector A, double rad){///向量A逆时针旋转rad(弧度制)

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

}

特殊情况，计算向量的单位法线，即左转90°以后长度归一化

Vector Normal(Vector A){///调用前确保A不是零向量

double L = Length(A);

return Vector(-A.y/L, A.x/L);

}

### 点和直线

struct Point{///定义二维平面上一点 P(x, y)

double x, y;

Point(double x = 0.0, double y = 0.0) : x(x), y(y){}

};

struct Line{///定义直线,点向式， 点P和向量V

Point p;

Vector v;

Line(Point p, Vector v) : p(p), v(p) {}

Point point(double t) {return Point(p+v\*t);}

};

直线参数表示：直线可以用直线P0和方向向量v表示， 直线上任意一点P满足P0 + tv，t称之为参数

直线:参数t没有限制

射线:参数t>0

线段:参数 0 < t < 1

两直线交点：直线P + tv 和 Q + tw

///使用前保证两直线有唯一交点， 即Cross(v, w)不为 0

Point GetLineIntersection(Point P, Vector v, Point Q, Vector w){

Vector u = P - Q;

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

return P+v\*t;

}

点到直线的距离

直线外一点P, 直线上两点A, B, 点到直线的距离为

double DistanceToLine(Line L, Point P){///点到直线的距离

return fabs(Cross(P-L.p, L.v)/Length(L.v));

}

对于直线的一般形式Ax+By+C=0，点P(x0, y0)到直线的距离为

点到线段的距离：线段外点P, 线段两端点A,B

两种情况：1、点P在线段AB的投影点Q在线段AB上，可以转换为点P到直线AB的距离

2、否则，根据点Q的位置，若点Q在射线BA上，则为线段PA的距离，若在射线AB上，则为线段PB的距离

double DistanceToSegment(Point P, Point A, Point B){

if(A == B) return Length(P-A);

Vector v1 = B - A, v2 = P - A, v3 = P - B;

if(dcmp(Dot(v1, v2)) < 0) return Length(v2);

if(dcmp(Dot(v1, v3)) > 0) return Length(v3);

return fabs(Cross(v1, v2) / Length(v1));

}

点到直线的投影点：线段外点P, 线段两端点A,B

///求点P在直线L上的投影点

Point GetLineProjection(Line L, Point P){

return L.p + L.v \* (Dot(L.v, P-L.p) / Dot(L.v, L.v));

}

线段相交判定：线段a1a2, 选段b1b2,

///端点处相交不计为两线段相交，规范相交

bool SegmengProperIntersection(Point a1, Point a2, Point b1, Point b2){

double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);

double d1 = Cross(b2-b1, a1-b1), d2 = Cross(b2-b1, a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(d1) \* dcmp(d2) < 0;

}

若允许在端点处相交：

///非规范相交

bool SegmengIntersection(Point a1, Point a2, Point b1, Point b2){

if(OnSegment(a1, b1, b2) || OnSegment(a2, b1, b2)

|| OnSegment(b1, a1, a2) || OnSegment(b2, a1, a2)) return true;

double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);

double d1 = Cross(b2-b1, a1-b1), d2 = Cross(b2-b1, a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(d1) \* dcmp(d2) < 0;

}

## 圆

struct Circle{

Point c; double r;

Circle(Point c, double r):c(c), r(r){}

Point point(double a){///一点c为中心的极角

return Point(c.x+r\*cos(a), c.y+r\*sin(a));

}

};

直线和圆的交点

int GetLineCircleIntersection(Line L, Circle C, double& t1, double& t2, vector<Point>& sol){

double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;

double e = a\*a + c\*c, f = 2.0\*(a\*b + c\*d), g = b\*b+d\*d-C.r\*C.r;

double delta = f\*f - 4\*e\*g;

int tmp = dcmp(delta);

if(tmp < 0) return 0;

if(tmp == 0) {

t1 = t2 = -f/(2\*e); sol.push\_back(L.point(t1)); return 1;

}

t1 = (-f + sqrt(delta)) / (2\*e); t2 = (-f - sqrt(delta)) / (2\*e);

sol.push\_back(L.point(t1)); sol.push\_back(L.point(t2)); return 2;

}

两圆相交

int GetCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol){

double d = Length(C1.c-C2.c);

if(dcmp(d) == 0) {

if(dcmp(C1.r-C2.r) == 0) return -1; ///两圆重合

return 0; ///两圆没有交点

}

if(dcmp(C1.r+C2.r-d) < 0) return 0; ///两圆相离

if(dcmp(fabs(C1.r-C2.r) - d) > 0) return 0;///两圆内含

Vector V = C2.c - C1.c;

double a = atan2(V.y, V.x); ///圆心向量的极角

double da = acos((C1.r\*C1.r+d\*d-C2.r\*C2.r)/(2\*C1.r\*d));/余弦定理

Point p1 = C1.point(a-da), p2 = C1.point(a+da);

sol.push\_back(p1);

if(p1 == p2) return 1;///一个交点

sol.push\_back(p2); return 2;///两个交点

}

过定点作圆的切线

///过点p到圆C的切线，v[i]是第i条切点的向量，返回切线条数

int GetTangents(Point P, Circle C, vector<Vector>& sol){

Vector u = C.c - P;

double dist = Length(u);

int tmp = dcmp(dist - C.r);

if(tmp < 0) return 0;///点p在圆的里面

else if(tmp == 0) {///p点在圆上，一条切线

sol.push\_back(Rotate(u, pi/2)); return 1;

}

else {

double ang = asin(C.r / dist);

sol.push\_back(Rotate(u, ang));

sol.push\_back(Rotate(u, -ang));

return 2;

}

}

两圆的公切线

六种情况：

1. 两圆完全重合，有无数条公切线
2. 两圆内含，没有公共点，没有公切线

3、两圆内切，有一个公共点，一条公切线

4、两圆相交，有两条公切线

5、两圆外切，有三条公切线。

6、两圆相离，有四条公切线

///求两圆的公切线，返回切线的数目，切点

int GetCircleCircleTangents(Circle A, Circle B, vector<Point>& sola, vector<Point>& solb){

if(A.c == B.c && dcmp(A.r-B.r) == 0) return -1; ///两圆重合，if(dcmp(A.r-B.r) < 0) {swap(A, B); swap(sola, solb);}

double d = Length(A.c - B.c);

double rdiff = A.r - B.r, rsum = A.r + B.r;

double base = angle(B.c - A.c);

if(dcmp(d - rsum) < 0){

if(dcmp(d - rdiff) < 0) return 0; ///内含，无交点，无公切线

if(dcmp(d - rdiff) == 0) {///内切，一条公切线

sola.push\_back(A.point(base));

solb.push\_back(B.point(base));

return 1;

}

double ang = acos((A.r-B.r) / d);

sola.push\_back(A.point(base+ang));

solb.push\_back(B.point(base+ang));

sola.push\_back(A.point(base-ang));

solb.push\_back(B.point(base-ang));

return 2;

}

double ang = acos((A.r-B.r) / d);

if(dcmp(d - rsum) == 0) {///外切，三条公切线

sola.push\_back(A.point(base));

solb.push\_back(B.point(base));

sola.push\_back(A.point(base+ang));

solb.push\_back(B.point(base+ang));

sola.push\_back(A.point(base-ang));

solb.push\_back(B.point(base-ang));

return 3;

}

///相离，4条公切线

sola.push\_back(A.point(base+ang)); solb.push\_back(B.point(base+ang));

sola.push\_back(A.point(base-ang));

solb.push\_back(B.point(base-ang));

double ang1 = acos((A.r + B.r) / d);

sola.push\_back(A.point(base+ang1));

solb.push\_back(B.point(pi+base+ang1));

sola.push\_back(A.point(base-ang1));

solb.push\_back(B.point(pi+base-ang1));

return 4;

}

## 凸包

### Graham Scan

基于极角排序， 时间复杂度(nlogn)

const int maxn = 1e5+100;

const double pi = acos(-1.0);

const double eps = 1e-6;

struct Point{

double x, y;

Point(double x = 0.0, double y = 0.0) : x(x), y(y){}

};

int stk[maxn], top;

typedef Point Vector;

Point lst[maxn], p0;

Vector operator - (Vector A, Vector B){

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

}

bool operator < (Vector A, Vector B){

return A.x < B.x || (A.x == B.x && A.y < B.y);

}

int dcmp(double x){

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

if(x > 0) return 1;

return -1;

}

double Dis(Vector A, Vector B){

return sqrt((A.x-B.x)\*(A.x-B.x) + (A.y-B.y)\*(A.y-B.y));

}

double Cross(Vector A, Vector B){return A.x\*B.y - A.y\*B.x;}

bool cmp(const Point& A, const Point& B){///极角排序

int tmp = dcmp(Cross(A-p0, B-p0));

if(tmp > 0) return true;

if(tmp == 0 && (Dis(A, p0) < Dis(B, p0))) return true;

return false;

}

void Graham(int n){/// lst[0...n)， 计算最大空凸包

for(int i = 0; i <= n + 5; i ++) stk[i] = 0;

int k = 0; p0 = lst[0];

for(int i = 1; i < n; i ++){///寻找纵坐标最小的点，横坐标最小的点

if(p0.y > lst[i].y || (p0.y == lst[i].y && p0.x > lst[i].x)){

p0 = lst[i]; k = i;

}

}

lst[k] = lst[0]; lst[0] = p0;

sort(lst + 1, lst + n, cmp);

if(n == 1){top = 1; stk[0] = 0; return;}

if(n == 2){top = 2; stk[0] = 0; stk[1] = 1; return;}

stk[0] = 0; stk[1] = 1; top = 2;

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

while(top > 1 && Cross(lst[stk[top-1]] - lst[stk[top-2]], lst[i] - lst[stk[top-2]]) <= 0){

top --;

}stk[top ++] = i;

}return;

}

int main()

{

int n;

while(scanf("%d", &n) != EOF) {

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

scanf("%lf %lf", &lst[i].x, &lst[i].y);

}

Graham(n); stk[top] = stk[0];

double ans = 0;

for(int i = 0; i < top; i ++){///计算凸多边形面积

ans += Cross(lst[stk[i]], lst[stk[i+1]]);

}ans = fabs(ans)/2;///求凸多边形面积

printf("%.2f\n", ans);

}

return 0;

}

### Andrew算法

基于水平排序，时间复杂度O(nlogn)，首选Andrew算法

const int maxn = 1e5+100;

const double pi = acos(-1.0);

const double eps = 1e-6;

struct Point{

double x, y;

Point(double x = 0.0, double y = 0.0) : x(x), y(y){}

};

typedef Point Vector;

Point p[maxn], ch[maxn];

int dcmp(double x){

if(fabs(x) < eps) return 0; if(x < 0) return -1; return 1;

}

bool operator < (const Point& A, const Point& B){

return A.x < B.x || (A.x == B.x && A.y < B.y);

}

Vector operator - (Vector A, Vector 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;}

double Dis(Vector A, Vector B){

return sqrt((A.x-B.x)\*(A.x-B.x) + (A.y-B.y)\*(A.y-B.y));

}

int ConvexHull(Point \*p, int n, Point \*ch){

sort(p, p+n); int m = 0;

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

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 --){

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;

}///ConvexHull(p, n, ch);

### 旋转卡壳求凸包直径

double solve(Point \*ch, int n){///旋转卡壳求凸包直径

if(n == 1) return 0;

if(n == 2) return Dis(ch[0], ch[1]);

double ans = 0;

for(int u = 0, v = 1; u < n; u ++){

while(1){

double diff = Cross(ch[u+1]-ch[u], ch[v+1]-ch[v]);

if(dcmp(diff) <= 0){

ans = max(ans, Dis(ch[u], ch[v]));

if(dcmp(diff) == 0) ans = max(ans, Dis(ch[u], ch[v+1]));

break;

}v = (v + 1) % n;

}

}return ans;

}

## 三维坐标

定义三维点

struct Point3{

double x, y, z;

Point3(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z){}

};

typedef Point3 Vector3;

Vector3 operator + (Vector3 A, Vector3 B){

return Vector3(A.x+B.x, A.y+B.y, A.z+B.z);

}

Vector3 operator - (Vector3 A, Vector3 B){

return Vector3(A.x-B.x, A.y-B.y, A.z-B.z);

}

Vector3 operator \* (Vector3 A, double p){

return Vector3(A.x\*p, A.y\*p, A.z\*p);

}

Vector3 operator / (Vector3 A, double p){

return Vector3(A.x/p, A.y/p, A.z/p);

}

double Dot3(Vector3 A, Vector3 B){return A.x\*B.x+A.y\*B.y+A.z\*B.z;}

double Length3(Vector3 A){return sqrt(Dot3(A, A));}

double Angle3(Vector3 A, Vector3 B){

return acos(Dot3(A, B)/(Length3(A)\*Length3(B)));

}

Vector3 Cross3(Vector3 A, Vector3 B){

return Vector3(A.y\*B.z-A.z\*B.y, A.z\*B.x-A.x\*B.z, A.x\*B.y-A.y\*B.x);

}

double Area3(Point3 A, Point3 B, Point3 C){

return Length3(Cross3(B-A, C-A));

}

# 图论

## DFS序

定义：dfs序是指：每个节点在dfs深度优先遍历中的进出栈的时间序列

性质：对于一棵树的dfs序而言，同一棵子树所对应的一定是dfs序中连续的一段。

例题：HDU5692

题意：给定一棵n个点的有根苹果树，每个点上最多只有一个苹果，现在有Q次操作：

(1).修改一个点上的苹果个数。

(2).求某个子树内有几个苹果。

题解：求出树的DFS序，并记录当前节点到根的路径权值之和，用线段树维护最大值

const int maxn = 100000+10;

int n, m, tot, cnt;

ll a[maxn], v[maxn], p[maxn];

int le[maxn], ri[maxn];

vector<int>ve[maxn];///最好用链式前向星

void init(){

cnt = 0; tot = 0;

mes(a, 0); mes(v, 0);

for(int i = 0; i < maxn; i ++) ve[i].clear();

}

void dfs(int x, int fa, ll w){///求得DFS序

le[x] = ++ cnt;

p[x] = w;

for(int i = 0; i < ve[x].size(); i ++){

int y = ve[x][i];

if(y != fa){

dfs(y, x, w + v[y]);

}

}

ri[x] = cnt;

}

struct SegmentTree{

int l, r;

ll ans, add;

#define l(p) tree[p].l

#define r(p) tree[p].r

#define ans(p) tree[p].ans

#define add(p) tree[p].add

#define lson l, r, p<<1

#define rson l, r, p<<1|1

}tree[maxn\*8];

void push(int p){

ans(p) = max(ans(p<<1), ans(p<<1|1));

}

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

l(p) = l; r(p) = r; add(p) = 0;

if(l == r){

ans(p) = a[l]; return;

}

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

build(l, mid, p<<1);

build(mid+1, r, p<<1|1);

push(p);

}

void spread(int p){

if(add(p)){

ans(p<<1) += add(p); ans(p<<1|1) += add(p);

add(p<<1) += add(p); add(p<<1|1) += add(p);

add(p) = 0;

}

}

void change(int l, int r, int p, ll d){

if(l <= l(p) && r >= r(p)){

ans(p) += d; add(p) += d; return;

}

spread(p);

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

if(l <= mid) change(lson, d);

if(r > mid) change(rson, d);

push(p);

}

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

if(l <= l(p) && r >= r(p)){

return ans(p);

}

spread(p);

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

ll res = -inf;

if(l <= mid) res = max(res, query(lson));

if(r > mid) res = max(res, query(rson));

return res;

}

int main(){

int Case = 0;

int T; scanf("%d", &T);

while(T --){

init();

printf("Case #%d:\n", ++ Case);

scanf("%d %d", &n, &m);

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

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

ve[x].push\_back(y); ve[y].push\_back(x);

}

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

scanf("%lld", &v[i]);

}

dfs(0, -1, v[0]);

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

a[le[i]] = p[i];

}

build(1, n, 1);

while(m --){

int op, x, y;

ll d;

scanf("%d", &op);

if(op == 1){

scanf("%d", &x);

ll ans = query(le[x], ri[x], 1);

printf("%lld\n", ans);

}

else if(op == 0){

scanf("%d %lld", &x, &d);

change(le[x], ri[x], 1, (ll)(d - v[x]));

v[x] = d;

}

}

}

return 0;

}

## 树链剖分

描述：树剖是通过轻重边剖分将树分割成多条链，然后利用数据结构来维护这些链（本质上是一种优化暴力）

|  |  |
| --- | --- |
| 名称 | 解释 |
| fat[u] | 保存结点u的父亲节点 |
| dep[u] | 保存结点u的深度值 |
| Size[u] | 保存以u为根的子树节点个数 |
| son[u] | 保存重儿子 |
| rk[u] | 保存当前dfs标号在树中所对应的节点 |
| top[u] | 保存当前节点所在链的顶端节点 |
| id[u] | 保存树中每个节点剖分以后的新编号（DFS的执行顺序） |

模板题：洛谷P3384描述:

* 将树从x到y结点最短路径上所有节点的值都加上z
* 求树从x到y结点最短路径上所有节点的值之和
* 将以x为根节点的子树内所有节点值都加上z
* 求以x为根节点的子树内所有节点值之和

const int maxn = 1e5+10;

int n, m, root, mod;

int v[maxn];

int tot, ver[maxn<<1], Next[maxn<<1], head[2\*maxn<<1];

int fat[maxn], dep[maxn], Size[maxn], son[maxn], rk[maxn], top[maxn], id[maxn];

int cnt;

int vis[maxn];

void init(){///初始化

tot = 0; cnt = 0;

mes(head, 0); mes(Next, 0); mes(son, 0); mes(fat, 0);

}

void addedge(int x, int y){

ver[++tot] = y; Next[tot] = head[x]; head[x] = tot;

}

/\*\*

第一遍-> dfs1记录父亲节点f[u]，深度d[u], Size[u], son[u]，

\*\*/

void dfs1(int x, int fa, int depth){

fat[x] = fa; dep[x] = depth; Size[x] = 1;

for(int i = head[x]; i; i = Next[i]){

int y = ver[i];

if(y != fa){

dfs1(y, x, depth+1);

Size[x] += Size[y];

if(Size[y] > Size[son[x]]) son[x] = y; //选n[x]]) son[u]=v;

}

}

}

/\*\*

第二遍-> dfs2 获取id[u], rk[u], top[u]

\*\*/

void dfs2(int x, int t){///t->x的top节点

top[x] = t; id[x] = ++cnt; rk[cnt] = x;

if(!son[x]) return;

dfs2(son[x], t);

for(int i = head[x]; i; i = Next[i]){

int y = ver[i];

if(y != son[x] && y != fat[x]){

dfs2(y, y);

}

}

}

int LCA(int x, int y){

for(; top[x] != top[y]; ){

if(dep[top[x]] > dep[top[y]]) x = fat[top[x]];

else y = fat[top[y]];

}

if(dep[x] < dep[y]) return x;

return y;

}

struct SegmentTree{

int l, r;

int add, sum;

#define l(p) tree[p].l

#define r(p) tree[p].r

#define add(p) tree[p].add

#define sum(p) tree[p].sum

#define lson l, r, p<<1

#define rson l, r, p<<1|1

}tree[maxn<<2];

int getlen(int p){

return r(p) - l(p) + 1;

}

void push(int p){

sum(p) = (sum(p<<1) + sum(p<<1|1)) % mod;

}

void spread(int p){

if(add(p)){

sum(p<<1) = (sum(p<<1) + getlen(p<<1) \* add(p)) % mod;

sum(p<<1|1) = (sum(p<<1|1) + getlen(p<<1|1) \* add(p)) % mod;

add(p<<1) = (add(p<<1) + add(p)) % mod; add(p<<1|1) = (add(p<<1|1) + add(p)) % mod;

add(p) = 0;

}

}

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

l(p) = l; r(p) = r; add(p) = 0;

if(l == r){

sum(p) = v[rk[l]] % mod; return ;

}

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

build(l, mid, p<<1);

build(mid+1, r, p<<1|1);

push(p);

}

void change(int l, int r, int p, ll d){

if(l <= l(p) && r >= r(p)){

sum(p) = (sum(p) + getlen(p) \* d) % mod; add(p) = (add(p) + d) % mod;

return;

}

spread(p);

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

if(mid >= l) change(lson, d);

if(mid < r) change(rson, d);

push(p);

}

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

if(l <= l(p) && r >= r(p)){

return sum(p) % mod;

}

spread(p);

ll ans = 0;

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

if(mid >= l) ans = (ans + query(lson)) % mod;

if(mid < r) ans = (ans + query(rson)) % mod;

ans = (ans % mod + mod) % mod;

return ans;

}

void update(int x, int y, int d){

int l, r;

while(top[x] != top[y]){

if(dep[top[x]] < dep[top[y]]) swap(x, y);

l = id[top[x]], r = id[x];

change(l, r, 1, d);

x = fat[top[x]];

}

if(dep[x] > dep[y]) swap(x, y);

l = id[x]; r = id[y];

change(l, r, 1, d);

}

ll getval(int x, int y){

ll ans = 0;

int l, r;

while(top[x] != top[y]){

if(dep[top[x]] < dep[top[y]]) swap(x, y);

l = id[top[x]], r = id[x];

ans = (ans + query(l, r, 1)) % mod;

x = fat[top[x]];

}

if(dep[x] > dep[y]) swap(x, y);

l = id[x]; r = id[y];

ans = (ans + query(l, r, 1)) % mod;

ans = (ans % mod + mod) % mod;

return ans;

}

int main(){

scanf("%d %d %d %d", &n, &m, &root, &mod);

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

scanf("%d", &v[i]);

}

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

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

addedge(x, y); addedge(y, x);

}

dfs1(root, 0, 1); dfs2(root, root);

build(1, n, 1);

while(m --){

int op, x, y, d;

scanf("%d", &op);

if(op == 1){

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

d %= mod;

update(x, y, d);

}

else if(op == 2){

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

ll ans = getval(x, y);

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

}

else if(op == 3){

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

d %= mod;

change(id[x], id[x]+Size[x]-1, 1, d);

}

else if(op == 4){

scanf("%d", &x);

ll ans = query(id[x], id[x]+Size[x]-1, 1);

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

}

}

return 0;

}

## LCA（最近公共祖先）

定义：(Lowest Commond Ancestor)在一棵树中两个节点的LCA为这两个节点所有的公共祖先中深度最大的节点。

### 树链剖分求LCA

预处理(2\*(n+m))+LCA(log n)

int LCA(int x, int y){

for(; top[x] != top[y]; ){

if(dep[top[x]] > dep[top[y]]) x = fat[top[x]];

else y = fat[top[y]];

}

if(dep[x] < dep[y]) return x;

return y;

}

倍增

Tarjan

## 最短路：

### 单源最短路（Single Source Shortest Path SSSP）无负权边

Dijkstra 算法（O（n^2））

const int maxn = 1000;

int n, m, a[maxn][maxn], d[maxn];

bool vis[maxn];

void dijkstra(){

mes(d, INF); mes(vis, false); d[1] = 0;

while(1){

int v = -1;

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

if(!vis[u] && (v == -1 || d[u] < d[v])){

v = u; ///寻找最短路径

}

}

if(v == -1)break;

vis[v] = true;

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

d[u] = min(d[u], d[v]+a[v][u]);///跟新路径

}

}

}

int main()

{

scanf("%d %d", &n, &m);

mes(a, INF);

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

a[i][i] = 0;

}

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

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

a[x][y] = min(a[x][y], z);

}dijkstra();

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

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

}return 0;

}

### Dijkstra 堆优化（优先队列 -- 大根堆）O（m log n）

const int maxn = 1e5+10;

int head[maxn], ver[maxn], edge[maxn], Next[maxn], d[maxn];

bool vis[maxn];

int n, m, tot;

priority\_queue<pair<int, int >>q; /// pair的第一维为dist的相反数（变成小根堆），第二维是节点编号

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

ver[++tot] = y; edge[tot] = z;

Next[tot] = head[x]; head[x] = tot;

}

void dijkstra(){

mes(d, INF); mes(vis, false);

d[1] = 0;

q.push(make\_pair(0, 1));

while(q.size()){

int x = q.top().second; q.pop();

if(vis[x])continue;

vis[x] = true;

for(int i = head[x]; i ; i = Next[i]){

int y = ver[i], z = edge[i];

if(d[y] > d[x] + z){

d[y] = d[x] + z;

q.push(make\_pair(-d[y], y));

}

}

}

}

### Bellman-Ford算法和SPFA算法

Spfa算法（万能最短路算法，可跑负权边）

[Gym - 102072J](https://cn.vjudge.net/problem/2213242/origin#problem/_blank)(判断是否存在负权回路)

const int maxn = 2e5+10;

int head[maxn], ver[maxn], Next[maxn];

ll edge[maxn], d[maxn];

bool vis[maxn];

int n, m, s, tot, cnt[maxn];

queue<int>q;

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

ver[++tot] = y; edge[tot] = z; Next[tot] = head[x]; head[x] = tot;

}

bool spfa(int s, int n){

mes(vis, false); mes(cnt, 0);

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

d[i] = inf;

}

while(q.size()) q.pop();

q.push(s);

vis[s] = true; d[s] = 0; cnt[s] = 1;

while(q.size()){

int x = q.front(); q.pop();

vis[x] = false;

for(int i = head[x]; i; i = Next[i]){

int y = ver[i];

ll z = edge[i];

if(d[y] > d[x]+z){

d[y] = d[x]+z;

if(!vis[y]){

vis[y] = true; q.push(y);

if(++cnt[y] > n)return false; ///判断负权回路

}

}

}

} return true;

}

bool vs[maxn];

int main()

{

std::ios::sync\_with\_stdio(false);

cin.tie(0);

scanf("%d %d %d", &n, &m, &s);

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

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

add(x, y, z);

}

bool flag = spfa(s, n);

if(!flag){

printf("-1\n"); return 0;

}

while(1){

int id = 0, f = 0;

mes(vs, false);

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

if(!vs[i]){

if(d[i] == inf){

f = 1; id = i; vs[i] = true; break;

}

else vs[i] = true;

}

} flag = spfa(id, s);

if(!flag){

printf("-1\n"); return 0;

} if(f == 0)break;

f = 0;

} flag = spfa(s, n);

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

if(i == s)printf("0\n");

else if(d[i] == inf) printf("NoPath\n");

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

} return 0;

}

### Floyd

void Floyd(){

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

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

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

if(…)…

}

}

}

}

## 最小生成树

### Kruskal(O(mlog m))

const int maxn = 5e4+100;

int n, m, tot;

int father[maxn];

struct Edge{

int u, v, cost;

friend bool operator < (const Edge& A, const Edge& B){

return A.cost < B.cost;

}

}edge[maxn<<1];

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

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

}

int find\_root(int x){

if(father[x] == x) return father[x];

return father[x] = find\_root(father[x]);

}

void merge\_v(int x, int y){

int x\_root = find\_root(x);

int y\_root = find\_root(y);

father[x\_root] = y\_root;

}

int kruskal(){

for(int i = 0; i <= n + 10; i ++){

father[i] = i;

}

sort(edge+1, edge+tot+1);

int cnt = 0;

int ans = 0;

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

int x = edge[i].u, y = edge[i].v, z = edge[i].cost;

if(find\_root(x) == find\_root(y)) continue;

merge\_v(x, y);

cnt ++; ans += z;

if(cnt == n - 1) return ans;

}

}

## Tarjan算法

### 有向图强连通分量

void Tarjan(int x){

low[x] = dfn[x] = ++ cnt;

S.push(x); vis[x] = 1;

for(int i = head[x]; i; i = Next[i]){

int y = ver[i];

if(!dfn[y]){

Tarjan(y);

low[x] = min(low[x], low[y]);

}

else if(vis[y]){

low[x] = min(low[x], dfn[y]);

}

}

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

ans ++; ///表示强连通分量数

while(1){

int tmp = S.top();

vis[tmp] = 0; S.pop();

if(tmp == x) break;

}

}

}

### 割边（桥）与割点

注意:只有在无向图中才有割点和割边的定义！

割点：无向图中，去掉一个顶点及和它相邻的所有边，图中的连通分量数增加，则该顶点称为割点。

割边：无向图中，去掉一条边，图中的连通分量数增加，则这条边，称为桥或者割边。

割点与割边的关系：

有割点不一定有割边，有割边一定有割点

割边一定是割点依附的边

割边判定法则：无向图(x, y)是桥，当且仅当搜索树上存在x的一个子节点x满足：

dfn[x] < low[y]通过推理可得桥一定是搜索树中的边，并且一个简单环中的边一定都不是桥。

割点判定法则：若x不是搜索树的根节点，则x是割点当且仅当搜索树上存在x的一个子节点y，满足：dfn[x] <= low[y]。特别的，若x是搜索树的根节点，则x是割点当且仅当搜索树上存在至少两个子节点y1,y2,满足此条件。

/\*Tarjan算法求割点\*/

const int maxn = 100000 + 10;

int tot, ver[maxn<<1], head[maxn<<1], Next[maxn<<1];

int dfn[maxn], low[maxn];

bool cut[maxn];///表示节点是否是割点

int n, m, cnt, root, ans[maxn];

void init(){

mes(ver, 0); mes(head, 0); mes(Next, 0); mes(dfn, 0); mes(low, 0);

tot = 0; cnt = 0; mes(cut, false);

}

void addedge(int x, int y){

ver[++tot] = y; Next[tot] = head[x]; head[x] = tot;

}/\* root = x; Tarjan(x) \*/

void Tarjan(int x){

dfn[x] = low[x] = ++ cnt; int flag = 0;

for(int i = head[x]; i; i = Next[i]){

int y = ver[i];

if(!dfn[y]){

Tarjan(y);

low[x] = min(low[x], low[y]);

if(low[y] >= dfn[x]){

flag ++;

if(x != root || flag > 1) cut[x] = true;

}

}

else low[x] = min(low[x], dfn[y]);

}

}

/\*\*Tarjan算法求桥\*\*/

满足 low[u] > dfn[u]

const int maxn = 1e5+100;

int ver[maxn<<1], Next[maxn<<1], head[maxn<<1], dfn[maxn<<1], low[maxn];

int n, m, tot, num;

bool bridge[maxn<<1];

void addedge(int x, int y){

ver[++ tot] = y; Next[tot] = head[x]; head[x] = tot;

}

void Tarjan(int x, int in\_edge){

dfn[x] = low[x] = ++ num;

for(int i = head[x]; i; i = Next[i]){

int y = ver[i];

if(!dfn[y]){

Tarjan(y, i);

low[x] = min(low[x], low[y]);

if(low[y] > dfn[x]){

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

}

}

else if(i != (in\_edge ^ 1)){

low[x] = min(low[x], dfn[y]);

}

}

}

int main()

{

tot = 1; num = 0;///注意初始化为零

scanf("%d %d", &n, &m);

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

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

addedge(x, y); addedge(y, x);

}

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

if(!dfn[i]) Tarjan(i, 0);

}

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

if(bridge[i]){

printf("%d %d\n", ver[i], ver[i^1]);

}

}

return 0;

}

## 网络流

int n, m, k, tot;

int ver[maxn], head[maxn], Next[maxn], edge[maxn];

int d[maxn];

int s, t, maxflow;

queue<int>q;

void init(){

tot = 1; mes(head, 0);///tot初始化为0

maxflow = 0;

}

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

ver[++tot] = y; edge[tot] = z; Next[tot] = head[x]; head[x] = tot;

}

### EK

O(V\*E^2)

bool bfs(){

mes(vis, 0);

queue<int> q;

q.push(s); vis[s] = 1; incf[s] = inf;

while(q.size()){

int x = q.front(); q.pop();

for(int i = head[x]; i; i = Next[i]){

if(edge[i]){

int y = ver[i];

if(vis[y]) continue;

incf[y] = min(incf[x], edge[i]);

pre[y] = i; q.push(y), vis[y] = 1;

if(y == t) return 1;

}

}

}return 0;

}

void EK(){

int x = t;

while(x != s){

int i = pre[x];

edge[i] -= incf[t]; edge[i^1] += incf[t]; x = ver[i^1];

} maxflow += incf[t];

}

### Dinic

O(V^2\*E)

bool bfs(){

mes(d, 0);

while(q.size())q.pop();

q.push(s); d[s] = 1;

while(q.size()){

int x = q.front(); q.pop();

for(int i = head[x]; i; i = Next[i]){

int y = ver[i], z = edge[i];

if(z && !d[y]){

q.push(y); d[y] = d[x] + 1;

if(y == t) return 1;

}

}

}return 0;

}

int Dinic(int x, int flow){

if(x == t) return flow;

int rest = flow, k;

for(int i = head[x]; i; i = Next[i]){

int y = ver[i], z = edge[i];

if(edge[i] && d[y] == d[x] + 1){

k = Dinic(y, min(rest, z));

if(!k) d[y] = 0;

edge[i] -= k; edge[i^1] += k; rest -= k;

}

}return flow - rest;

}

## 二分图

定义：N个点的无向图，分成A、B两个非空集合，其中,并且在同一集合内的点没有边相连，则称此图为一张二分图。

### 二分图判定

定理：一张无向图G(V, E)是二分图，当且仅当图中不存在奇环（长度为奇数的环）。

也即，能用两种颜色染色此图，满足每条边的两端点的颜色不同。

相关概念：

匹配：任意两条边没有公共断点。

极大匹配：在一组匹配M中，对于任意一条边,都不是一组匹配。

最大匹配：二分图中，包含边数最多的一组匹配即为最大匹配。

注意：极大匹配不等同于最大匹配

边覆盖：图G中每一个顶点至少是F中一条边的某一端点

独立集：集合S中任意连个顶点没有边相连

点覆盖：图G中每一条边至少有一个端点在集合S中

最大匹配、最小边覆盖、最大独立集、最小顶点覆盖的关系：

二分图最大匹配

匈牙利算法O(VE)

const int maxn = 2e3+100;

int n, m, E;

int tot, ver[2000100], head[2000100], Next[2000100];

int match[maxn]; ///大小和定点个数相同

bool vis[maxn]; ///与一个集合的大小相同

void addedge(int x, int y){

ver[++ tot] = y; Next[tot] = head[x]; head[x] = tot;

}

bool dfs(int x){///找增广路

vis[x] = true;

for(int i = head[x]; i; i = Next[i]){

int y = ver[i], w = match[y];

if(w < 0 || (!vis[w] && dfs(w))){

match[x] = y; match[y] = x; return true;

}

}

return false;

}

int max\_match(){

mes(match, -1);

int res = 0;

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

if(match[i] < 0){

mes(vis, 0);

if(dfs(i)){

res ++;

}

}

}

return res;

}

int main()

{

scanf("%d %d %d", &n, &m, &E);

while(E --){

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

if(x > n || y > m) continue;

addedge(x, y + n); addedge(y + n, x);

}

printf("%d\n", max\_match());

return 0;

}

### 一般图的最大匹配

带花树算法：

### 二分图带权匹配

KM算法：

# 博弈论

## 巴什博奕（Bash Game）

描述：有n个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m个。最后取光者得胜。

如果n%(m+1) != 0先手必胜，否则先手必败！

## 威佐夫博奕（Wythoff Game）

描述：有两堆物品，一堆有a个物品，另一堆有b个物品，两个人轮流取走一些石子，有两种取法：

1.从某堆中取走任意多个物品

2.同时从两堆中取同样多的物品

规定每次至少取一个，多者不限，最后取光者得胜。

bool judge(int n, int m){

int a = min(n, m), b = max(n, m);

double r = (sqrt(5.0)+1)/2; ///黄金比例数

double c = (b - a);

if((int)(r\*c) == a) return false;///先手必败

return true;///先手必胜

}

## 尼姆博弈(Nim Game)

描述：有n堆若干个物品，两个人轮流从某一堆取任意多的物品，规定每次至少取一个，多者不限，最后取光着获胜。

a1^a2^···^an==0(奇异局势),先手必败，否则先手必胜！

## 斐波那契博弈(Fibonacci Nim)

描述：有一堆个数为 n 的石子，游戏双方轮流取石子，满足

1)先手不能在第一次把所有的石子取完；

2)之后每次可以取的石子数介于 1 到对手刚取的石子数的 2 倍之间（包含 1 和对手刚取的石子数的 2 倍）。

约定取走最后一个石子的人为赢家，求必败态。

若n是斐波那契数，则先手必败，否则先手必胜！

# JAVA大数

///基本输入输出

import java.math.\*;

import java.util.\*;

import java.math.BigDecimal;

import java.math.BigInteger;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner cin = new Scanner(System.in); ///输入类实例化

BigInteger a = new BigInteger("0");

BigInteger v[] = new BigInteger[100];

BigDecimal b = new BigDecimal(0);

BigDecimal w[] = new BigDecimal[100];

a = cin.nextBigInteger();

b = cin.nextBigDecimal();

cin.close();///关闭cin

//整数转大整数a = BigInteger.valueOf(10);

}

}

///JAVA矩阵快速幂

import java.math.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner cin = new Scanner(System.in);

BigInteger a = new BigInteger("0");

a = cin.nextBigInteger();

int n = cin.nextInt();

BigInteger ans = new BigInteger("0");

ans = KSM(a, n);

System.out.println(ans);

cin.close();

}

public static BigInteger KSM(BigInteger a, int n) {

BigInteger res = new BigInteger("1");

while(n != 0) {

if(n % 2 == 1) {

res = res.multiply(a);

}

a = a.multiply(a);

n /= 2;

} return res;

}

}

//JAVA威佐夫博弈

import java.math.\*;

import java.util.\*;

import java.math.BigInteger;

public class Main {

public static double five = 5.0;

public static BigDecimal FIVE = new BigDecimal("5.0");

public static BigDecimal ONE = new BigDecimal("1.0");

public static BigDecimal TWO = new BigDecimal("2.0");

public static BigDecimal ZERO = new BigDecimal("0.0");

public static BigDecimal eps = new BigDecimal("0.一百个01");

public static boolean check(BigDecimal m) {

BigDecimal ans = m.multiply(m).subtract(FIVE);

if(ans.compareTo(ZERO) >= 0) return true;

return false;

}

///威佐夫博弈

public static boolean judge(BigDecimal n, BigDecimal m, BigDecimal R) {

if(n.compareTo(m) == 0) return true;

BigDecimal a = new BigDecimal("0.0");

BigDecimal b = new BigDecimal("0.0");

if(n.compareTo(m) >= 0) {b = n; a = m;}

else {b = m; a = n;}

BigDecimal c = b.subtract(a);

BigDecimal Rc = R.multiply(c);

Rc = Rc.setScale(0, BigDecimal.ROUND\_FLOOR);///向下取整

// long b = a.setScale(0, BigDecimal.ROUND\_FLOOR).longValue();

if(Rc.compareTo(a) == 0) return true;

return false;

}

public static void main(String[] args) {

Scanner cin = new Scanner(System.in);

BigDecimal low = new BigDecimal("2.0");

BigDecimal high = new BigDecimal("3.0");

BigDecimal mid = new BigDecimal("0.0");

while(true) {//二分求根号5

if(high.subtract(low).compareTo(eps) <= 0) break;

mid = low.add(high).divide(TWO);

if(check(mid)) high = mid;

else low = mid;

}

BigDecimal R = mid.add(ONE).divide(TWO);

while(cin.hasNext()) {

BigDecimal n = cin.nextBigDecimal();

BigDecimal m = cin.nextBigDecimal();

if(judge(n, m, R)) {

System.out.println("0");

}

else System.out.println("1");

}

}

}

# 瞎搞冷知识

UVA-11827(输入冷知识)

没有给定输入数据大小，ungetc函数应用

int main()

{

int T; scanf("%d", &T);

while(getchar() != '\n');

while(T --){

int n = 0;

while((c = getchar()) != '\n'){

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

ungetc(c,stdin);

scanf("%d",&a[++n]);

}

}

int Max = 0;

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

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

Max = max(Max, \_\_gcd(a[i], a[j]));

}

}

cout<<Max<<endl;

}

return 0;

}

## 读写优化

int read(){

char x;

while((x = getchar()) > '9' || x < '0') ;

int u = x - '0';

while((x = getchar()) <= '9' && x >= '0')

u = (u << 3) + (u << 1) + x - '0';

return u;

}