目录

[基础算法 3](#_Toc18135772)

[二分 3](#_Toc18135773)

[离散化 3](#_Toc18135774)

[拓扑排序 3](#_Toc18135775)

[排序 4](#_Toc18135776)

[计数排序 4](#_Toc18135777)

[求逆序对 5](#_Toc18135778)

[1.归并排序 5](#_Toc18135779)

[2.树状数组求逆序对 6](#_Toc18135780)

[马拉车（最长回文子串） 7](#_Toc18135781)

[数据结构 8](#_Toc18135782)

[STL使用 8](#_Toc18135783)

[优先队列 8](#_Toc18135784)

[并查集 9](#_Toc18135785)

[前缀和 10](#_Toc18135786)

[树状数组 10](#_Toc18135787)

[线段树（Segment Tree） 11](#_Toc18135788)

[1.区间查询、单点修改 11](#_Toc18135789)

[2.区间查询、区间修改 12](#_Toc18135790)

[线段树求区间最大连续字段和 14](#_Toc18135791)

[（线段树|树状数组）离线处理 15](#_Toc18135792)

[线段树+扫描线（矩形面积并） 17](#_Toc18135793)

[矩形周长（HDU-1828） 19](#_Toc18135794)

[单调栈 21](#_Toc18135795)

[单调队列 22](#_Toc18135796)

[单调队列优化 23](#_Toc18135797)

[优先队列优化 24](#_Toc18135798)

[数学 25](#_Toc18135799)

[基础数论 25](#_Toc18135800)

[整除 25](#_Toc18135801)

[求和公式 26](#_Toc18135802)

[整除分块 26](#_Toc18135803)

[完全数 27](#_Toc18135804)

[阶乘  27](#_Toc18135805)

[梅森数 28](#_Toc18135806)

[斐波那契数列 28](#_Toc18135807)

[矩阵快速幂 29](#_Toc18135808)

[杨辉三角的奇偶性 32](#_Toc18135809)

[三角形 33](#_Toc18135810)

[构造勾股数 33](#_Toc18135811)

[欧几里得算法 34](#_Toc18135812)

[扩展欧几里得 34](#_Toc18135813)

[线性同余方程 34](#_Toc18135814)

[高次同余方程（BSGS算法） 34](#_Toc18135815)

[扩展BSGS 36](#_Toc18135816)

[康拓展开&康拓逆展开 37](#_Toc18135817)

[组合数学 38](#_Toc18135818)

[Lucas定理 38](#_Toc18135819)

[卡特兰数列（Catlan） 39](#_Toc18135820)

[小球放盒模型 39](#_Toc18135821)

[Ferrers图像 40](#_Toc18135822)

[积性函数： 40](#_Toc18135823)

[欧拉函数 40](#_Toc18135824)

[莫比乌斯函数 41](#_Toc18135825)

[约数个数函数 41](#_Toc18135826)

[约数和函数 42](#_Toc18135827)

[元函数 42](#_Toc18135828)

[恒等函数： 42](#_Toc18135829)

[单位函数 42](#_Toc18135830)

[狄利克雷卷积 42](#_Toc18135831)

[线性筛约数个数 43](#_Toc18135832)

[杜教筛 44](#_Toc18135833)

[逆元 46](#_Toc18135834)

[中国剩余定理（除数互质） 46](#_Toc18135835)

[中国剩余定理（除数不互质） 47](#_Toc18135836)

[图论 49](#_Toc18135837)

[DFS序 49](#_Toc18135838)

[树链剖分 51](#_Toc18135839)

[LCA（最近公共祖先） 56](#_Toc18135840)

[树链剖分求LCA 56](#_Toc18135841)

[最短路： 57](#_Toc18135842)

[单源最短路（Single Source Shortest Path SSSP）无负权边 57](#_Toc18135843)

[Dijkstra 堆优化（优先队列 -- 大根堆）O（m log n） 58](#_Toc18135844)

[Bellman-Ford算法和SPFA算法 58](#_Toc18135845)

[Tarjan算法 61](#_Toc18135846)

[有向图强连通分量 61](#_Toc18135847)

[割边（桥）与割点 61](#_Toc18135848)

[二分图 63](#_Toc18135849)

[二分图判定 63](#_Toc18135850)

[二分图最大匹配 63](#_Toc18135851)

[一般图的最大匹配 64](#_Toc18135852)

[二分图带权匹配 64](#_Toc18135853)

[博弈论 65](#_Toc18135854)

[巴什博奕（Bash Game） 65](#_Toc18135855)

[威佐夫博奕（Wythoff Game） 65](#_Toc18135856)

[尼姆博弈(Nim Game) 65](#_Toc18135857)

[斐波那契博弈(Fibonacci Nim) 65](#_Toc18135858)

[JAVA大数 66](#_Toc18135859)

[瞎搞冷知识 67](#_Toc18135860)

[读写优化 68](#_Toc18135861)

# 基础算法

## 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(){

///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 int maxn = 5e5+10;

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

int n;

void discrete(){

///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;

}

}

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();

LL ans = build();

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

}

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;

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;

}

# 数据结构

## STL使用

### 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它的根**

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。

### 1.区间查询、单点修改

///HDU1166

const int maxn = 50000+10;

int a[maxn]; ///第 i 个营地的人数

struct SegmentTree{

int l, r; ///左右区间

int sum; ///区间内总人数

}Tree[4\*maxn];

int n, m;

///跟新区间和

void Push(int p){

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

}

///创建一棵线段树

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

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

if(l == r){

Tree[p].sum = a[l];

return ;

}

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

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

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

Push(p);

}

///单点修改, 将a[x] 增加 v

void Change(int p, int x, int v){

if(Tree[p].l == Tree[p].r){

Tree[p].sum += v;

return;

}

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

if(x <= mid) Change(p<<1, x, v);

else Change(p<<1|1, x, v);

Push(p);

}

///查询区间和

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

if(l <= Tree[p].l && r >= Tree[p].r) return Tree[p].sum;

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

int ans = 0;

if(mid >= l) ans += Query(p<<1, l, r);

if(mid < r) ans += Query(p<<1|1, l, r);

return ans;

}

int main()

{

int Case = 0;

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

while(T --){

printf("Case %d:\n", ++Case);

scanf("%d", &n);

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

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

}

Build(1, 1, n);

string s;

int x, y;

while(cin>>s){

if(s == "End")break;

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

if(s == "Add") Change(1, x, y);

else if(s == "Sub") Change(1, x, -y);

else if(s == "Query") printf("%d\n",Query(1, x, y));

}

}

return 0;

}

### 2.区间查询、区间修改

/\*\*

线段树，区间修改模板题 POJ3468

懒散标记法

\*\*/

const int maxn = 1e5 + 10;

struct SegmentTree{

int l, r; ///左右区间

LL sum, add; ///sum -> 区间和, add -> 修改标记

}tree[4\*maxn];

int a[maxn];

int n, m;

void push(int p){

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

}

///创建线段树

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

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

if(l == r){

tree[p].sum = a[l]; return ;

}

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

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

push(p);

}

///懒散标记法

void spread(int p){

if(tree[p].add){

tree[p<<1].sum += (tree[p].add \* (tree[p<<1].r - tree[p<<1].l + 1));

tree[p<<1|1].sum += (tree[p].add \* (tree[p<<1|1].r - tree[p<<1|1].l + 1));

tree[p<<1].add += tree[p].add;

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

tree[p].add = 0;

}

}

///区间修改

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

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

tree[p].sum += (LL)(c)\*(tree[p].r - tree[p].l + 1);

tree[p].add += c;

return;

}

spread(p);

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

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

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

push(p);

}

///区间查询

LL query(int p, int l, int r){

if(l <= tree[p].l && r >= tree[p].r)return tree[p].sum;

spread(p);

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

LL ans = 0;

if(l <= mid)ans += query(p<<1, l, r);

if(r > mid)ans += query(p<<1|1, l, r);

return ans;

}

int main(){

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

for(int i = 1; i <= n; i ++)scanf("%d", &a[i]);

build(1, 1, n);

char s[5];

int x, y, c;

while(m --){

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

// printf("%s %d %d\n", s, x, y);

if(s[0] == 'Q'){

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

printf("%lld\n", query(1, x, y));

}else{

scanf("%d", &c);

change(1, x, y, c);

}

}

return 0;

}

### 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

///sum表示区间和,dat表示区间最大连续字段和

ll lmax, rmax, sum, dat;

}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;

}

后缀数组

后缀数组(Suffix array)

## 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;

}

# 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

}

### 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)，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)，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

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define inf 0x3f3f3f3f

#define mes(a, val) memset(a, val, sizeof a)

#define mec(b, a) memcpy(b, a, sizeof a)

#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.\*;

**public** **class** Main {

**public** **static** BigInteger *Zero* = **new** BigInteger("0");

**public** **static** BigInteger *One* = **new** BigInteger("1");

**public** **static** BigInteger *Two* = **new** BigInteger("2");

**public** **static** BigInteger *Three* = **new** BigInteger("3");

**public** **static** BigInteger *Four* = **new** BigInteger("4");

**public** **static** BigInteger KSM(BigInteger a, **int** n) {

BigInteger res = *One*;

**while**(n != 0) {

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

a = a.multiply(a);

n /= 2;

}

**return** res;

}

///二分查找2的最大次幂

**public** **static** 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");

**int** k = *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*));

**return** res;

}

**public** **static** **void** 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;

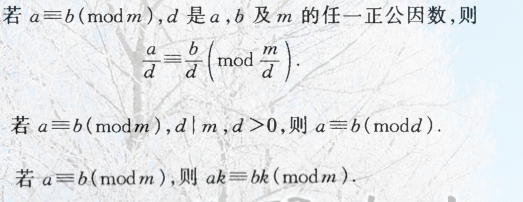
}

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];

int 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;

}

当n,m较小的时：

用动态规划求解dp[n,m] = dp[n-1,m]+dp[n-1,m-1]

### 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：

### 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。

### 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。

## 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.

求解：

NENU-500

1、简单方法

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：  。

### 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))

### 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：

#include <bits/stdc++.h>

#include <tr1/unordered\_map>

using namespace std;

#define ll long long

#define ull unsigned long long

#define inf 0x3f3f3f3f

#define mes(a, val) memset(a, val, sizeof a)

#define mec(b,a) memcpy(b,a, sizeof a)

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

1. 递推公式法：

int inv[mod+5];

void getInv(int mod){

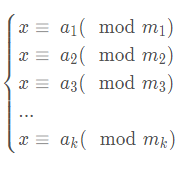
inv[1]=1;

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

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

}

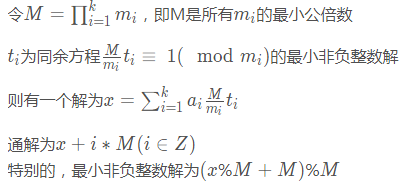
## 中国剩余定理（除数互质）

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

定理：



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);

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

return d;

}

ll china(){

ll M = 1, ans=0, mm;

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

ll x, y;

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

mm = M / m[i];

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

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

}

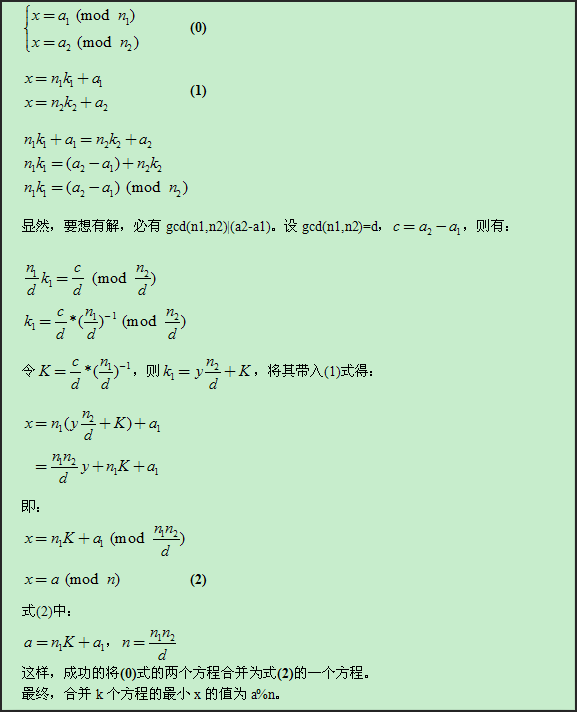
ans = (ans+M)%M;

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;

}

# 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 a[maxn][maxn], d[maxn];

int n, m;

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];

ll d[maxn];

bool vis[maxn];

int cnt[maxn];

int n, m, s, tot;

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;

}

## 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;

int 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]);

}

}

}

## 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)

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define inf 0x3f3f3f3f

#define mes(a, val) memset(a, val, sizeof a)

#define mec(b, a) memcpy(b, a, sizeof a)

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

}

}

///JAVA矩阵快速幂

//import java.util.\*;

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);

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;

}

}

# 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;

}