# 算法模板-XMH

1,线段树color the ball

2,树状数组，color the ball

3,DFS求连通块

4,FFT-kuangbin模板-两数相乘

5,FFT-kuangbin模板-机器人打高尔夫

6,KMP 模板

7,KMP-Cyclic Nacklace -求循环节的个数

8,KMP-Oulipo-可重叠子串

9, KMP剪花布条 -不可重复子串的个数

10, RMQ模板

11，tarjan-割边

12，Tarjan-割边+割点-self

13，Tarjan-求割边的数量

14，Tarjan-有向图缩点-POJ2186

15，并查集-模板

16，博弈论-Nim博弈-Being a Good Boy in Spring Festival

17，博弈论-巴什博奕-Brave Game

18，尺取法-Subsequence-O（n)

19，尺取法-最大区间和

20，动态规划-（经典）钱币兑换问题

21，动态规划-（经典）整数拆分-Ignatius and the Princess III

22，动态规划-01背包

23，动态规划-多重背包(优化版-k)-Coins

24，动态规划-多重背包-Fruit

25，动态规划-树形DP-没有上司的舞会

26，动态规划-完全背包-Sumsets

27，二分搜索-边界问题总结

28，二分图匹配-匈牙利算法-增广路

29，二分图-最大匹配+二分图判断-The Accomodation of Students

30，二分图-最大匹配-最小覆盖数的dfs实现

31，回溯法-UVA140-Bandwidth-压缩-双剪枝

32，马拉车算法模板

33，马拉车-最长回文

34，散列法-哈希函数

35，骚操作-自定义二分查找-Coffe Break

36，骚操作-自定义优先队列比较方式-Silver Cow Party

37，树状数组（单点更新）-Cows

38，树状数组(区间更新模板）-color the ball

39，数论-Miller\_rbin-素数判定

40，数论-埃式筛选法（改进版）

41，数论-大数因数分解Pollard\_rho算法-POJ2429 GCD & LCM Inverse-self

42，数论-大数因数分解Pollard\_rho算法-self

43，数论-欧拉筛选法

44，数论-欧拉筛选法-Enlarge GCD-好题-self

45，数论-区间内素数的个数-模板

46，数论-无穷级数-Harmonic Number

47，数论-质因数分解-全排列-3421-X-factor Chains

48，数位DP-不要62-DFS模板

49，图论大师-强连通分量分解-Popular Cows-POJ2186

50，线段树+RMQ-2019南昌网络邀请赛-I-Max Answer

51，折半搜索-自定义二分搜索-Sumsets-最终版

52，字典树-链表实现-单词数

53，字典树-模板-统计难题

54，组合数学-鸽巢原理- Find a multiple

55，最短路-Bellman\_Ford模板

56，最短路-Dijikstra-队列优化模板

57，最短路-DJ-次短路-好题

58，最短路-优先队列-Edge Deletion

59, 最小表示法-Hidden Password

60, 最小生成树-Kruskal-并查集-Conscription-

61, 最小生成树-prime（优先队列优化)-Agri-Net

62, 坐标离散化-Paint Color

63，最大团-Maximum Clique-HDU1530DFS+DP剪枝（输出目标点）

64，动态规划-计数DP-666RPG

1,线段树color the ball

#define INF 0x7fffffff

using namespace std;

const int maxn = 400000 + 10;//开1e5小了，线段树一般开原来的4倍

struct SegTree{

int l,r;

int sum;

int add\_mark;

};

SegTree seg\_tree[maxn];

int arr[maxn];

int n = 0;

void build(int root,int is,int ie){

seg\_tree[root].add\_mark = 0;

seg\_tree[root].l = is;

seg\_tree[root].r = ie;

if(is == ie){

seg\_tree[root].sum = arr[is];

return ;

}else{

int mid = (ie - is)/2 + is;

build(root\*2,is,mid);

build(root\*2+1,mid+1,ie);

seg\_tree[root].sum = seg\_tree[root\*2].sum + seg\_tree[root\*2+1].sum;

}

}

void push\_down(int root){

if(seg\_tree[root].add\_mark != 0){

seg\_tree[root\*2].add\_mark += seg\_tree[root].add\_mark;

seg\_tree[root\*2+1].add\_mark += seg\_tree[root].add\_mark;

seg\_tree[root\*2].sum += seg\_tree[root].add\_mark;

seg\_tree[root\*2+1].sum += seg\_tree[root].add\_mark;

seg\_tree[root].add\_mark = 0;

}

}

//void update(int root,int is,int ie,int us,int ue,int add\_val){

void update(int root,int us,int ue,int add\_val){

int is = seg\_tree[root].l,ie = seg\_tree[root].r ;

if(is > ue || ie < us){

return ;

}else if(is >= us && ie <= ue){

seg\_tree[root].add\_mark += add\_val;

seg\_tree[root].sum += add\_val;

return ;

}else{

push\_down(root);

//int mid = (ie - is)/2 + is;

//update(root\*2,is,mid,us,ue,add\_val);

//update(root\*2+1,mid+1,ie,us,ue,add\_val);

update(root\*2,us,ue,add\_val);

update(root\*2+1,us,ue,add\_val);

seg\_tree[root].sum = seg\_tree[root\*2].sum + seg\_tree[root\*2+1].sum;

}

}

int query(int root,int us,int ue){

int is = seg\_tree[root].l,ie = seg\_tree[root].r;

if(is > ue || ie < us){

return 0;

}else if(is >= us && ie <= ue){

return seg\_tree[root].sum;

}else{

push\_down(root);

//int mid = (ie - is)/2 + is;

//return query(root\*2,is,mid) + query(root\*2+1,mid+1,ie);

return query(root\*2,us,ue) + query(root\*2+1,us,ue);

}

}

int main(){

int a = 0,b = 0;

while(scanf("%d",&n) != EOF && n){

memset(arr,0,sizeof(arr));

memset(seg\_tree,0,sizeof(seg\_tree));

build(1,1,n);

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

scanf("%d %d",&a,&b);

update(1,a,b,1);

}

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

if(i > 1){

printf(" ");

}

printf("%d",query(1,i,i));

}

printf("\n");

}

return 0;

}

2,树状数组，color the ball

const int maxn = 1e5 + 10;

int tree[maxn];

int n = 0;

//输入的是对区间的操作，如果最终要求的是单点的值，

//那么updata区间1 - m更新操作 - 从上往下，单点求和操作 - 从下往上。

int lowbit(int x){

return x & (-x);

}

void add(int loc,int val){//在1 - loc位置增加val

while(loc > 0){

tree[loc] += val;

loc -= lowbit(loc);

}

}

int getsum(int num){

int sum = 0;

while(num <= n){

sum += tree[num];

num += lowbit(num);

}

return sum;

}

/\*

void add(int loc,int val){//在loc位置增加val

if(loc < 1){

return ;

}

for(int i = loc;i <= n;i += lowbit(i)){//单点更新，从叶子节点向上更新tree[]数组，是查询的逆过程

tree[i] += val;

}

}

\*/

/\*

int getsum(int num){//从上到下

int ans = 0;

for(int i = num;i > 0;i -= lowbit(i)){

ans += tree[i];

}

return ans;

}

\*/

int main(){

//std::ios::sync\_with\_stdio(false);

int a = 0,b = 0;

//while(cin >> n && n){

while(scanf("%d",&n) != EOF && n){

memset(tree,0,sizeof(tree));

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

scanf("%d %d",&a,&b);

add(b,1);

add(a - 1,-1);

}

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

if(j != 1){

printf(" ");

}

printf("%d",getsum(j));

}

printf("\n");

}

return 0;

}

3,DFS求连通块

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

const int maxn = 100 + 5;

int mov[2][3] = { {0,1,-1} , { 0,1,-1} }; //移动数组

char pic[maxn][maxn];

int m,n,idx[maxn][maxn];

void dfs(int r,int c,int id){

if(r < 0 || r >= m || c < 0 ||c >= n) return ; //出界的格子

if(pic[r][c] != 'W' || idx[r][c] > 0 ) return;//不是"@"或者已经访问过的格子

idx[r][c] = id;//连通分量编号

for(int i = 0;i < 3;i++){

for(int j = 0;j < 3;j++){

if(i != 0 || j != 0){

//r += mov[i][0];

//c += mov[0][j];

dfs(r + mov[0][i],c + mov[1][j],id); //移动方式

}

}

}

/\*

for(int dr = -1;dr <= 1;dr++){

for(int dc = -1;dc <= 1;dc++){

if(dr != 0 || dc != 0){

dfs(r+dr,c+dc,id);

}

}

}

\*/

}

int main(){

while(scanf("%d %d",&m,&n) == 2 && m && n){

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

scanf("%s",pic[i]);

}

int cnt = 0;

memset(idx,0,sizeof(idx));//初始化

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

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

if(pic[i][j] == 'W' && idx[i][j] == 0){

++cnt;

dfs(i,j,cnt);

}

}

}

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

}

return 0;

}

4,FFT-kuangbin模板-两数相乘

#include <stdio.h>

#include <string.h>

#include <iostream>

#include <algorithm>

#include <math.h>

using namespace std;

const double PI = acos(-1.0);

//复数结构体 //HDU1402

struct complex{

double r,i;

complex(double \_r = 0.0,double \_i = 0.0){

r = \_r; i = \_i;

}

complex operator +(const complex &b){

return complex(r+b.r,i+b.i);

}

complex operator -(const complex &b){

return complex(r-b.r,i-b.i);

}

complex operator \*(const complex &b){

return complex(r\*b.r-i\*b.i,r\*b.i+i\*b.r);

}

};

/\*

\* 进行FFT和IFFT前的反转变换。

\* 位置i和 （i二进制反转后位置）互换

\* len必须去2的幂

\*/

void change(complex y[],int len){

int i,j,k;

for(i = 1, j = len/2;i < len-1; i++){

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

//交换互为小标反转的元素，i<j保证交换一次

//i做正常的+1，j左反转类型的+1,始终保持i和j是反转的

k = len/2;

while( j >= k){

j -= k;

k /= 2;

}

if(j < k) j += k;

}

}

/\*

\* 做FFT

\* len必须为2^k形式，

\* on==1时是DFT，on==-1时是IDFT

\*/

void fft(complex y[],int len,int on){

change(y,len);

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

complex wn(cos(-on\*2\*PI/h),sin(-on\*2\*PI/h));

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

complex w(1,0);

for(int k = j;k < j+h/2;k++){

complex u = y[k];

complex t = w\*y[k+h/2];

y[k] = u+t;

y[k+h/2] = u-t;

w = w\*wn;

}

}

}

if(on == -1)

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

y[i].r /= len;

}

const int MAXN = 200010;

complex x1[MAXN],x2[MAXN];

char str1[MAXN/2],str2[MAXN/2];

int sum[MAXN];

int main(){

while(scanf("%s%s",str1,str2)==2){

int len1 = strlen(str1);

int len2 = strlen(str2);

int len = 1;

while(len < len1\*2 || len < len2\*2)len<<=1;

for(int i = 0;i < len1;i++)

x1[i] = complex(str1[len1-1-i]-'0',0);//将序列倒置

for(int i = len1;i < len;i++)

x1[i] = complex(0,0);

for(int i = 0;i < len2;i++)

x2[i] = complex(str2[len2-1-i]-'0',0);

for(int i = len2;i < len;i++)

x2[i] = complex(0,0);

//求DFT

fft(x1,len,1);

fft(x2,len,1);

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

x1[i] = x1[i]\*x2[i];

fft(x1,len,-1);

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

sum[i] = (int)(x1[i].r+0.5);

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

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

sum[i]%=10;

}

len = len1+len2-1;

while(sum[len] <= 0 && len > 0)len--;

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

printf("%c",sum[i]+'0');

printf("\n");

}

return 0;

}

5,FFT-kuangbin模板-机器人打高尔夫

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <algorithm>

#include <math.h>

using namespace std;

typedef long long LL;

/\*

这里省略了fft

\*/

const int MAXN = 400002;//注意数组大小

LL num[MAXN\*2];

int a[MAXN/2];

Complex x1[MAXN\*2];

int m, n;

/\*题意：一个机器人打高尔夫只往一个方向打，每次只能打固定的距离，

求最多两杆能打进的洞的个数；差不多可以理解为一个数列相加；

题解：利用fft算法的特点，

ax \* by 放在 c的第 x+y位； x，y就是那些固定的距离，x+y就是可以打进的洞；还不懂的话具体看代码；

\*/

//题目来源：Gym - 100783C https://vjudge.net/problem/Gym-100783C

int main() {

scanf("%d", &n);

int len1 = 0;

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

scanf("%d", a+i);

num[a[i]] = 1; //这里赋值为一就好了

len1 = max(len1, a[i]+1);//找到最大值就是len1

}

num[0] = 1;//这里要赋值为1，求出1杆可以打进的洞

//下面就是套路了

int len = 1;

while (len < 2\*len1) len <<= 1;

for(int i = 0; i < len1; i++)

x1[i] = Complex(num[i], 0);

for(int i = len1; i < len; i++)

x1[i] = Complex(0, 0);

fft(x1, len, 1);

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

x1[i] = x1[i]\*x1[i];

fft(x1, len, -1);

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

num[i] = (LL)(x1[i].x+0.5);

}

scanf("%d", &m);

int res = 0;

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

int t;

scanf("%d", &t);

if (num[t])//这里写起来就比较简单了

res++;

}

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

return 0;

}

6,KMP 模板

using namespace std;

const int maxn = 1020;

string str1,str2;

int nex[maxn];

void make\_next(){

nex[0] = -1;

int j = 0,k = -1;

while(j < str2.size()){

if(k == -1 || str2[k] == str2[j]){

if(str2[k+1] == str2[j+1]){ //当两个字符相等时要跳过

nex[j+1] = nex[k+1]; //因为回退到k时，因为相等，就是多余的

++j,++k;

}else{

nex[j+1] = k+1;//不相等时，加1

++j,++k;

}

}else{

k = nex[k];

}

}

}

int kmp(){

int i = 0,j = 0;

while(i < str1.size() && j < str2.size()){

if( j == -1 || str1[i] == str2[j] ){

++i;

++j;

}else{

j = nex[j];

}

}

if(j == str2.size()){

return i - j;

}else{

return -1;

}

}

int main(){

make\_next();

return 0;

}

7,KMP-Cyclic Nacklace -求循环节的个数

/\*

思路：kmp+字符串的最小循环节问题

分析：

1 题目要求的是给定一个字符串，问我们还需要添加几个字符可以构成一个由n个循环节组成的字符串。

2 可知我们应该先求出字符串的最小循环节的长度：假设字符串的长度为len，那么最小的循环节就是cir = len-next[len] ;

如果有len%cir == 0,那么这个字符串就是已经是完美的字符串，不用添加任何字符；

如果不是完美的那么需要添加的字符数就是cir - (len-(len/cir)\*cir))，相当与需要在最后一个循环节上面添加几个。

3 如果cir = 1，说明字符串只有一种字符例如“aaa” ;

如果cir = m说明最小的循环节长度为m，那么至少还需m个；

如果m%cir == 0，说明已经不用添加了。

\*/

const int maxn = 1e5 + 10;

int nex[maxn];

string str;

void make\_nex(){

int j = 0,k = -1;

nex[0] = -1;

while(j < str.size()){//是j < m而不是m-1,j <m-1也WA了，j < m不用担心越界

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

if(str[k+1] == str[j+1]){

nex[j+1] = nex[k+1];

++k,++j;

}else{

nex[j+1] = k+1;

++j,++k;

}

}else{

k = nex[k];

}

}

}

int main(){

int T = 0;

cin >> T;

while(T--){

cin >> str;

memset(nex,0,sizeof(nex));

make\_nex();

int len = str.size();//循环节必须用个数n（也就是从1开始计算的位数）

//nex[i]表示的是第nex[i] 位，从1开始，而不是0

int ans = len - nex[len];//循环节的大小，这个很重要，很多题目多要用到

if(ans != len && (len % ans == 0) ){

cout << "0" << endl;

}else{

cout << ans - len%ans << endl;

}

}

}

8,KMP-Oulipo-可重叠子串

const int maxn = 1e5 + 10;

int nex[maxn];

string str1,str2;

int ans = 0;

void make\_nex(){

int j = 0,k = -1;

nex[0] = -1;

int len2 = str2.size();

while(j < len2){//是j < m而不是m-1,虽然j <m-1也AC了，j < m不用担心越界

if(k == -1 || str2[k] == str2[j]){

if(str2[k+1] == str2[j+1]){

nex[j+1] = nex[k+1];

++j,++k;

}else{

nex[j+1] = k+1;

++j,++k;

}

}else{

k = nex[k];

}

}

}

void kmp(){

make\_nex();

int i = 0,j = 0;

int len1 = str1.size(),len2 = str2.size();

while(i < len1){

while(i < len1 && j < len2){

if(str1[i] == str2[j] || j == -1){

++i,++j;

}else{

j = nex[j];

}

}

if(j == len2){

++ans; //返回上一次重复的地方

j = nex[j]; //相对于KMP-减布花，稍微调一下位置，就对了

/\*比如样例： 利用对称性，深入理解那个nex[k]的含义便知道了

AZA

AZAZAZA

\*/

}else{

return ;

}

}

}

int main(){

int T = 0;

cin >> T;

while(T--){

cin >> str2 >> str1;

ans = 0;

memset(nex,0,sizeof(nex));

make\_nex();

kmp();

cout << ans << endl;

}

}

9, KMP剪花布条 -不可重复子串的个数

const int maxn = 1e5 + 10;

int nex[maxn];

string str1,str2;

int ans = 0;

void make\_nex(){

int j = 0,k = -1;

nex[0] = -1;

int len2 = str2.size();

while(j < len2){//是j < m而不是m-1,虽然j <m-1也AC了，j < m不用担心越界

if(k == -1 || str2[k] == str2[j]){

if(str2[k+1] == str2[j+1]){

nex[j+1] = nex[k+1];

++j,++k;

}else{

nex[j+1] = k+1;

++j,++k;

}

}else{

k = nex[k];

}

}

}

void kmp(){

make\_nex();

int i = 0,j = 0;

int len1 = str1.size(),len2 = str2.size();

while(i < len1){

j = 0; //每次都要把j拨回到0，重新开始匹配，因为i已经一步一步移动的，所以不会重复

//匹配成功一次后，每次重新匹配，这是求有多少个循环节，不重叠

while(i < len1 && j < len2){

if(str1[i] == str2[j] || j == -1){

++i,++j;

}else{

j = nex[j];

}

}

if(j == len2){

++ans;

}else{

return ;

}

}

}

int main(){

while(cin >> str1 && str1[0] != '#'){

cin >> str2;

ans = 0;

memset(nex,0,sizeof(nex));

make\_nex();

kmp();

cout << ans << endl;

}

}

10,RMQ模板

const int maxn = 5e4;

int n = 0,q = 0;

int a[maxn+10];

int dp[maxn+10][20];

void RMQ(){

per(i,1,n){

dp[i][0] = a[i];

}

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

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

dp[i][j] = max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);

}

}

}

int query(int l,int r){

if(l > r){

return 0;

}

int k = 0;

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

++k;

}

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

}

int main(){

//std::ios::sync\_with\_stdio(false);

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

per(i,1,n){

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

}

RMQ();

per(i,1,q){

int l = 0,r = 0;

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

int ans = query(l,r);

printf("%d\n");

}

}

return 0;

}

11，tarjan-割边

#include<bits/stdc++.h>

using namespace std;

const int maxn = 2e5;

int m = 0,n = 0;

vector<int> g[maxn+10];

bool vis[maxn+10];

int low[maxn+10],dfn[maxn+10];

int cnt = 0,dig = 0;

//这里是tarjan代码模板核心

/\*

dfn[i]:节点 i dfs的次序//时间戳

low[i]: 节点 i 所在环中 所有节点中最小的 dfn[i]。

\*/

void tarjan(int u,int w){

vis[u] = true;

dfn[u] = low[u] = (++cnt);

int size = g[u].size();

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

int v = g[u][i];

if(v == w){//因为是无向图，防止向后返回去遍历

continue;

}

if(!vis[v]){//没有遍历过，就接着向下遍历，直到找到后，再执行下面的代码，也就是回溯

tarjan(v,u);

}

if(vis[v]){//回溯

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

}

}

if(low[u] == dfn[u]){//判断当前的节点

++dig;

}

}

void init(){//初始化函数

memset(vis,false,sizeof(vis));

memset(low,false,sizeof(low));

memset(dfn,false,sizeof(dfn));

}

int main(){

std::ios::sync\_with\_stdio(false);

cin.tie(0);cout.tie(0);

cin >> n >> m;

init();

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

int x = 0,y = 0;

cin >> x >> y;

g[x].push\_back(y);g[y].push\_back(x);

}

tarjan(1,-1);

//cout << m - (dig-1) << endl;

cout << dig - 1 << endl;//割边的数量

return 0;

}

12，Tarjan-割边+割点-self

#include<bits/stdc++.h>

using namespace std;

const int maxn = 2e5;

int n = 0,m = 0;

vector<int> g[maxn+10];

int fa[maxn+10];

int dfn[maxn+10],low[maxn+10];

bool is\_cut[maxn+10];

int tim = 0;

void init(){

tim = 0;

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

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

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

memset(is\_cut,false,sizeof(is\_cut));

}

void Tarjan(int u,int father){

fa[u] = father;

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

int size = g[u].size();

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

int v = g[u][i];

if(dfn[v] == -1){

Tarjan(v,u);

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

}else if(v != father){/\*假如v是u的父亲的话，那么这就是无向边中的重边，有重边那么一定不是桥\*/

low[u] = min(low[u],dfn[v]);//dfn[v]可能！=low[v]，所以不能用low[v]代替dfn[v],否则会上翻过头了。

}

}

}

void count(){

int rootson = 0;

Tarjan(1,0);

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

int v = fa[i];

if(v == 1){/\*统计根节点子树的个数，根节点的子树个数>=2,就是割点\*/

++rootson;

}else{

if(low[i] >= dfn[v]){//割点的条件

is\_cut[v] = true;

}

}

}

if(rootson > 1){

is\_cut[1] = true;

}

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

if(is\_cut[i]){

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

}

}

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

int v = fa[i];

if(v > 0 && low[i] > dfn[v]){//桥（割边）的条件

printf("%d %d\n",v,i);

}

}

}

int main(){

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

init();

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

int x = 0,y = 0;

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

g[x].push\_back(y);g[y].push\_back(x);

}

count();

}

return 0;

}

13，Tarjan-求割边的数量

#include<bits/stdc++.h>

using namespace std;

const int maxn = 2e5;

int m = 0,n = 0;

vector<int> g[maxn+10];

bool vis[maxn+10];

int low[maxn+10],dfn[maxn+10];

int cnt = 0,dig = 0;

//这里是tarjan代码模板核心

void tarjan(int u,int w){

vis[u] = true;

dfn[u] = low[u] = (++cnt);

int size = g[u].size();

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

int v = g[u][i];

if(v == w){//因为是无向图，防止向后返回去遍历

continue;

}

if(dfn[v] == -1){//没有遍历过，就接着向下遍历，直到找到后，再执行下面的代码，也就是回溯

tarjan(v,u);

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

}else{

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

}

}

if(low[u] == dfn[u]){//这个用来求割边的个数，但是无法求割边具体是什么

++dig;//求有几个强连通分支，那么割边的数量等于强连通分支数量-1

}

}

void init(){//初始化函数

memset(vis,false,sizeof(vis));

memset(low,0,sizeof(low));

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

}

int main(){

std::ios::sync\_with\_stdio(false);

cin.tie(0);cout.tie(0);

cin >> n >> m;

init();

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

int x = 0,y = 0;

cin >> x >> y;

g[x].push\_back(y);g[y].push\_back(x);

}

tarjan(1,-1);

cout << dig-1 << endl;//割边的数量

return 0;

}

14，Tarjan-有向图缩点-POJ2186

const int maxn=1e4;

const LL mod=1e9+7;

int n = 0,m = 0;

vector<int> g[maxn+10];

int dfn[maxn+10],low[maxn+10];//dfn表示时间戳，low[i]表示节点i可以向上连接的最小时间戳的点，把遍历顺序看做一棵树

int fa[maxn+10],sum[maxn+10],out[maxn+10],stac[maxn+10];

int tim = 0,loc = 0,cnt = 0;

void init(){

tim = 0,loc = 0,cnt = 0;

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

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

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

memset(sum,0,sizeof(sum));

memset(out,0,sizeof(out));//初始化为0，不能是-1

}

void Tarjan(int u){//有向图，无向图在输入时，两边pusu\_back,还有就是判断条件（父节点）不能倒退

/\*

相对于另外一个没有缩点的template，这里的fa存储的是该所处强连通分量的编号.

如果要求具体的割点和割边，则还要一个数组记录每个点的父节点 ,然后利用割点和割边的条件判定

割点：low[v] >= dfn[u]:v是u的子所有节点，可以相反考虑，也就是low[v] < dfn[u]则u不是割点

割边：low[v] > dfn[u]:v是u的子所有节点，可以相反考虑，也就是low[v] < dfn[u]则u不是割边，

当Low[v] = dfn[u]表示v是在以u为根节点的强联通分量，删去强连通分量的一条边，无影响

\*/

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

int size = g[u].size();

stac[++loc] = u;//模拟栈

per(i,0,size-1){

int v = g[u][i];

if(dfn[v] == -1){//该点没有访问

Tarjan(v);

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

}else if(fa[v] == -1){//该点还不属于一个已知的（最大）强连通分量 ，无向图判断不一样

low[u] = min(low[u],dfn[v]);//dfn[v]不是low[v] ,防止上翻过度

}

}

if(dfn[u] == low[u]){//将同一个强连通分量的点缩点，最大强连通子图都会被压缩成一个点

int len = loc;// 强连通分量的点的编号不是统一的，而是一个小的强连通图一个标号

++cnt;//强连通分量的编号

while(stac[loc] != u){

fa[stac[loc--]] = cnt;//出栈

}

fa[stac[loc--]] = cnt;

sum[cnt] = len - loc;

}

}

/\*

首先跑一遍有向图的tarjan,边跑边压缩。（将每一个最大强连通子图压缩成一个点）

压缩之后，统计各点的出度，有且只有一个点出度为0时，解是这个压缩点所包含的原节点的个数，其余情况都是0

\*/

int main(){

//std::ios::sync\_with\_stdio(false);

//cin.tie(0);cout.tie(0);

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

init();

per(i,1,m){

int x = 0,y = 0;

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

g[x].push\_back(y);

}

per(i,1,n){

if(dfn[i] == -1){

Tarjan(i);

}

}

per(i,1,n){//计算出度

int size = g[i].size();

per(j,0,size-1){

int v = g[i][j];

if(fa[i] != fa[v]){

++out[fa[i]];

}

}

}

int x = 0;

per(i,1,cnt){//只能有一个出度为0的点

if(out[i] == 0){

if(x > 0){

printf("0\n");

return 0;

}

x = sum[i];

}

}

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

}

return 0;

}

15，并查集-模板

const int maxn = 10000 + 10;

int par[maxn];//父亲

int rak[maxn];//树的高度

int n = 0;

//初始化n个元素

void init(){

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

par[i] = i;

rak[i] = 0;

}

}

//查询树的根

int find(int x){

if(par[x] == x){

return x;

}else{

return par[x] = find(par[x]);

}

}

//合并x和y所属的集合

void unite(int x,int y){

x = find(x),y = find(y);

if(x == y){

return ;

}

if(rak[x] < rak[y]){

par[x] = y;

}else{

par[y] = x;

if(rak[x] == rak[y]){

++rak[x];

}

}

}

//判断x和y是否属于同一个集合

bool same(int x,int y){

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

}

int main(){

return 0;

}

16，博弈论-Nim博弈-Being a Good Boy in Spring Festival

const int maxn = 1e2;

int n = 0,m = 0;

int a[maxn+10];

int main(){

while(~scanf("%d",&n) && (n != 0)){

int sum = 0,cnt = 0;

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

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

sum ^= a[i];

}

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

if((sum ^ a[i]) < a[i]){

cnt++;

}

}

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

}

return 0;

}

17，博弈论-巴什博奕-Brave Game

const int maxn = 3e5;

int n = 0,m = 0;

/\*

如果n=m+1,因为最多取m个，所以先拿的人拿多少个，后拿的人能全拿走。

由此可以总结出 取胜秘籍： n=(m+1)\*r+s ?(r为任意自然数，s为小于等于m）。

那么先取者拿走s个物品，如果后取者拿走K（≤m）个，那么先取者肯定获胜。

总之，要保持给对手留下（m+1）的倍数，就能最后获胜。

于是，这道题解法就出来了，只要判断 n%(m+1)是否等于等于0。

等于0代表，无论怎么取，最后都会剩下<=m个，因此后取者将会获胜。

\*/

int main(){

int T = 0;

scanf("%d",&T);

while(T--){

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

if((n % (m+1)) != 0){

printf("first\n");

}else{

printf("second\n");

}

}

return 0;

}

18，尺取法-Subsequence-O（n)

const int maxn=2e5 + 10;

int cnt1=0,cnt2=0,cnt3=0,n = 0,tmp = 0,a[maxn],b[maxn],c[maxn],ma=-inf,flag;

int s = 0;

int solve(){

int minv = n + 1;

int sum = 0 ,i = 1,e = 1;

while(true){

while(e <= n && sum < s){

sum += a[e++];

//sum += a[++e];int e = 0;while(e < n)

}

if(sum < s){

break;

}

minv = min(minv,e - i);

sum -= a[i++];

}

if(minv > n){

return 0;

}

return minv;

}

int main(){

int T = 0;

scanf("%d",&T);

while(T--){

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

per(i,1,n){

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

}

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

}

return 0;

}

19，尺取法-最大区间和

const int maxn = 100000 + 10;

int a[maxn];

int main(){

int T = 0,n = 0;

cin >> T;

int kase = 0;

while(T--){

cin >> n;

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

cin >> a[i];

}

int start = 1,end = 1;//最终答案

int s = 1,e = 1;//记录当前位置

//start起始位置，end结尾位置,sum为连续数字的和，s为起始位置变量，

//和start有关，e为结尾位置变量,max为所求的最大值

int max = a[1],sum = 0;

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

sum += a[j];

e = j; //记录运算到的位置，也是末位置

if(sum > max){ //更新最大值

max = sum;

start = s,end = e;

}

if(sum < 0){ //当目前的和小于0时，从下一位重新开始

s = j + 1; //记录新运算的起始位置

sum = 0;

}

}

if(kase > 0){

cout << endl;

}

cout << "Case " << ++kase << ":" << endl;

cout << max << " " << start << " " << end << endl;

}

return 0;

}

20，动态规划-（经典）钱币兑换问题

#include<bits/stdc++.h>

using namespace std;

const int maxn = 32768;

int dp[4][maxn + 100];//dp[i][j]表示使用前i种货币，凑出j元的方法数

int n = 0;

void solve(){

fill(dp[0],dp[0]+4\*(maxn+100),0);

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

dp[1][i] = 1;

}

for(int i = 1;i <= 3;++i){

dp[i][0] = 1;

}

for(int i = 2;i <= 3;++i){

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

dp[i][j] = dp[i-1][j] + (j >= i ? dp[i][j-i]:0);

}

}

}

int main(){

solve();

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

printf("%d\n",dp[3][n]);

}

return 0;

}

21，动态规划-（经典）整数拆分-Ignatius and the Princess III

#include<bits/stdc++.h>

using namespace std;

const int maxn = 125;

int dp[maxn+5][maxn + 5];//dp[i][j]表示使用前i种货币，凑出j元的方法数

int n = 0;

void solve(){

fill(dp[0],dp[0]+(maxn+5)\*(maxn+5),0);

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

dp[1][i] = 1;

}

for(int i = 1;i <= maxn;++i){//要初始化为1

dp[i][0] = 1;

}

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

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

dp[i][j] = dp[i-1][j] + (j >= i ? dp[i][j-i]:0);//i,j状态由只是用前i-1种货币

//凑成j元的方法数+使用前i种货币j-i元（在加上i元货币）的方法数

}

}

}

int main(){

solve();

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

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

}

return 0;

}

22，动态规划-01背包

const int maxn = 3e5;

int n = 0,m = 0;

int dp[maxn+10];

struct node{

int v,w;

};

node a[maxn+10];

void solve(){

memset(dp,0,sizeof(dp));

dp[0] = 0;

per(i,1,n){

rep(j,m,0){

dp[j] = (j >= a[i].w ? max(dp[j-a[i].w] + a[i].v,dp[j]): dp[j]);

}

}

printf("%d\n",dp[m]);

}

int main(){

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

per(i,1,n){

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

}

solve();

}

return 0;

}

23，动态规划-多重背包(优化版-k)-Coins

const int maxn = 1e5;

int n = 0,m = 0;

struct node{

int val,num;

};

node coin[maxn+10];

bool dp[maxn+10];//不能用二维数组，那样会MLE,要滚动降维

int cnt[maxn+10];

bool cmp(node a,node b){

return a.val != b.val ? a.val < b.val : a.num < b.num;

}

void solve(){

fill(dp,dp+(maxn+10),false);

dp[0] = true;

per(i,1,n){

fill(cnt,cnt+maxn+10,0);

per(j,0,m){//正序，因为是从更新后的状态退出后面的j的状态的

if(dp[j] == true){//不要跳过，否则cnt会+1,导致过快选择的i种卡超过数量限制

continue;//为了保证使用最少的第i种卡，凑出j

}

if(j >= coin[i].val && dp[j-coin[i].val] &&

cnt[j - coin[i].val] < coin[i].num){

dp[j] = dp[j - coin[i].val];

cnt[j] = cnt[j-coin[i].val] + 1;

}

}

}

int ans = 0;

per(i,1,m){

ans += dp[i] ? 1 : 0;

}

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

}

int main(){

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

per(i,1,n){

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

}

per(i,1,n){

scanf("%d",&coin[i].num);

}

//sort(coin+1,coin+1+n,cmp);//排序是多余的

solve();

}

return 0;

}

24，动态规划-多重背包-Fruit

#include<bits/stdc++.h>

using namespace std;

const int maxn = 100 + 10;

struct node{

int mi,ma;

};

node num[maxn];

int n = 0,m = 0;

int dp[maxn][maxn];

void solve(){

fill(dp[0],dp[0]+maxn\*maxn,0);

/\*

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

dp[i][0] = 1;

}

\*/

dp[0][0] = 1;

int min\_sum = 0,max\_sum = 0;

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

for(int j = min\_sum + num[i].mi;j <= m && j <= max\_sum + num[i].ma;++j){//j <= max\_sum + num[i].ma是错的

for(int k = num[i].mi;k <= num[i].ma && k <= j;++k){

dp[i][j] += dp[i-1][j-k];

}

}

min\_sum += num[i].mi;max\_sum += num[i].ma;

}

printf("%d\n",dp[n][m]);

}

int main(){

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

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

scanf("%d %d",&num[i].mi,&num[i].ma);

}

solve();

}

return 0;

}

25，动态规划-树形DP-没有上司的舞会

const int maxn = 6000 + 10;

int val[maxn];

vector<int> vt[maxn];

int dp[maxn][2];

int n = 0;

bool is\_root[maxn];

void dfs(int u){

dp[u][0] = 0;

dp[u][1] = val[u];

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

int v = vt[u][i];

dfs(v);

dp[u][1] += dp[v][0];

dp[u][0] += max(dp[v][1],dp[v][0]);

}

}

int main(){

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

fill(is\_root,is\_root+n+1,true);

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

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

}

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

int e = 0,s = 0;

scanf("%d %d",&e,&s);

vt[s].push\_back(e);

is\_root[e] = false;

}

int root = -1;

for(int i = 1;i <= n;++i){//根结点是唯一入度为0的节点

if(is\_root[i]){

root = i;

break;

}

}

dfs(root);

printf("%d\n",max(dp[root][0],dp[root][1]));

}

return 0;

}

26，动态规划-完全背包-Sumsets

const int maxn = 1e6;

int n = 0,m = 0;

int dp[maxn + 10];

const LL mod = 1e9;//取后面的9位数是mod1e9而不是1e10，这里我WA了一上午

void solve(){//完全背包

fill(dp,dp+maxn+10,0);

dp[0] = 1;

int tmp = 1;

per(i,1,21){

if(n < tmp){

break;

}

per(j,tmp,n){

dp[j] = (dp[j] + dp[j-tmp]) % mod;

}

tmp <<= 1;

}

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

}

int main(){

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

solve();

}

return 0;

}

27，二分搜索-边界问题总结

const int maxn = 1e5;

int n = 0,u = 0;

int a[maxn+10];

/\*

int solve(int l,int x){

int r = n;

int ans = -1;

while(l <= r){

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

if(a[mid] > x){

r = mid - 1;

}else{

ans = mid;

l = mid + 1;

}

}

return ans;

}

\*/

/\*

int solve(int l,int x){

int r = n+1;

int ans = -1;

while(l < r){

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

if(a[mid] > x){

r = mid;

}else{

ans = mid;

l = mid + 1;

}

}

return r-1;//这里返回的是r-1,因为是[l,r)

}

\*/

int solve(int l,int x){

int r = n+1;

--l;

int ans = -1;

while(l + 1 < r){

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

if(a[mid] > x){

r = mid;

}else{

l = mid;

}

}

//return l;//

return r-1;

}

/\*根据上面3种情况，可以得到规律：

[l,r]返回mid; [l,r)返回r-1; (l,r)返回r-1或者l

1,如果是闭区间则要返回mid

2,如果半开半闭区间，r开时，返回r-1;l,r都开时，返回 r-1,或者l都可以

\*/

int main(){

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

per(i,1,n){

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

}

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

double ans = 0;

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

//int k = lower\_bound(a+i,a+1+n,a[i]+u) - a;

int k = solve(i,a[i]+u);

//cout << k << endl;

if(k <= i+1){

continue;

}

//if(a[k] != a[i] + u){//不用lower\_bound后，就不需要了，因为自己写的二分

//函数一定是<=a[i]+u的，而lower\_bound函数是>=

// --k;

//}

ans = max(ans,(a[k] - a[i+1])\*1.0 / (1.0 \* (a[k] - a[i])) );

}

if(ans == 0){

printf("-1\n");

}else{

printf("%.10lf\n",ans);

}

}

return 0;

}

28，二分图匹配-匈牙利算法-增广路

#include<bits/stdc++.h>

using namespace std;

typedef long long LL;

const int maxn = 1e4;

int edge[maxn+10][maxn+10];//边的邻接矩阵

bool vis[maxn+10];//访问数组

int cx[maxn+10],cy[maxn+10];//记录x集合中匹配的y集合的元素

int nx = 0,ny = 0,t = 0;//x集合中的元素

int path(int u){

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

if(vis[i] == false && edge[u][i] == 1){//没有访问，并且有边相连

vis[i] = true;

if(cy[i] == -1 || path(cy[i])){//如果y集合中的i没有匹配，或者i已经匹配

//但是可以从cy[i]中可以找到一条增广路

cx[u] = i;cy[i] = u;//找到增广路，修改匹配

return 1;

}

}

}

return 0;

}

int max\_match(){

int sum = 0;

memset(cx,-1,sizeof(cx));memset(cy,-1,sizeof(cy));//初始化-1，表示都没有匹配

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

if(cx[i] == -1){//没有匹配时，执行path代码

memset(vis,false,sizeof(vis));//重置标记为未访问

sum += path(i);

}

}

return sum;

}

int main(){

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

memset(edge,0,sizeof(edge));

scanf("%d %d",&nx,&ny);

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

int x = 0,y = 0;

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

edge[x][y] = edge[y][x] = 1;

}

int sum = max\_match();

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

}

return 0;

}

29，二分图-最大匹配+二分图判断-The Accomodation of Students

const int maxn = 2e2;

int n = 0,m = 0;

vector<int> mp[maxn+10];

bool vis[maxn+10];

int color[maxn+10];

int cntx = 0,cnty = 0;

int cx[maxn+10],cy[maxn+10];

int px[maxn+10],py[maxn+10];

bool bfs(int s){

memset(vis,false,sizeof(vis));

queue<int> que;

que.push(s);

vis[s] = true;

color[s] = 1;

while(!que.empty()){

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

int size = mp[u].size();

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

if(!vis[mp[u][i]]){

vis[mp[u][i]] = true;

color[mp[u][i]] = !color[u];

que.push(mp[u][i]);

}

if(color[mp[u][i]] == color[u]){

return false;

}

}

}

return true;

}

int path(int u){

int size = mp[u].size();

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

if(!vis[mp[u][i]]){

vis[mp[u][i]] = true;

if(py[mp[u][i]] == -1 || path(py[mp[u][i]])){

px[u] = mp[u][i];py[mp[u][i]] = u;

return 1;

}

}

}

return 0;

}

int max\_match(){

memset(px,-1,sizeof(px));memset(py,-1,sizeof(py));

int sum = 0;

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

if(px[cx[i]] == -1){

memset(vis,false,sizeof(vis));

sum += path(cx[i]);

}

}

return sum;

}

int main(){

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

per(i,1,n){

mp[i].clear();

}

per(i,1,m){

int x = 0,y = 0;

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

mp[x].push\_back(y);mp[y].push\_back(x);

}

memset(color,-1,sizeof(color));//color如果只是bool类型，则无法区分未访问的

//连通分支和颜色相反的，有3种状态：未访问，1，0

bool flag = true;

per(i,1,n){

if(color[i] == -1){

if(bfs(i) == false){

flag = false;

break;

}

}

}

if(flag == false){

printf("No\n");

continue;

}

//cout << "jaja "<< endl;

cntx = 0,cnty = 0;

per(i,1,n){

if(color[i] == 1){

cx[++cntx] = i;

}else{

cy[++cnty] = i;

}

}

int sum = max\_match();

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

}

return 0;

}

30，二分图-最大匹配-最小覆盖数的dfs实现

#include<bits/stdc++.h>

using namespace std;

typedef long long LL;

const int maxn = 1e4;

int edge[maxn+10][maxn+10];//边的邻接矩阵

bool vis[maxn+10];//访问数组

int cx[maxn+10],cy[maxn+10];//记录x集合中匹配的y集合的元素

int nx = 0,ny = 0,t = 0;//x集合中的元素

int path(int u){

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

if(vis[i] == false && edge[u][i] == 1){//没有访问，并且有边相连

vis[i] = true;

if(cy[i] == -1 || path(cy[i])){//如果y集合中的i没有匹配，或者i已经匹配

//但是可以从cy[i]中可以找到一条增广路

cx[u] = i;cy[i] = u;//找到增广路，修改匹配

return 1;

}

}

}

return 0;

}

int max\_match(){

int sum = 0;

memset(cx,-1,sizeof(cx));memset(cy,-1,sizeof(cy));//初始化-1，表示都没有匹配

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

if(cx[i] == -1){//没有匹配时，执行path代码

memset(vis,false,sizeof(vis));//重置标记为未访问

sum += path(i);

}

}

return sum;

}

int main(){//最大匹配 == 最小顶点覆盖数，最小路径覆盖数 = V - 最大匹配

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

memset(edge,0,sizeof(edge));

scanf("%d %d",&nx,&ny);

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

int x = 0,y = 0;

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

//edge[x][y] = 1;//有向边

edge[x][y] = edge[y][x] = 1;

}

int sum = max\_match();

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

}

return 0;

}

31，回溯法-UVA140-Bandwidth-压缩-双剪枝

const int maxn = 10;

int n = 0,m = 0;

//string str;

char str[1000];

//vector<int> vt[maxn];

int s[maxn+10],sy[maxn+10];

int mp[maxn+2][maxn+2];

int ans = 20;

bool vis[maxn+10];

int id[30],back[maxn+10];//id的内存要够

void dfs(int cur,int bw){

if(cur == n){

//if(bw != 0 && bw < ans){//多余的判断，能进入到这一层，一定会满足这些条件（能进行下一层地柜的条件）

memcpy(sy,s,sizeof(s));

ans = bw;

//}

return ;

}

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

if(vis[i]){

continue;

}

int tmp = bw;//bw也要在不同的选择还原

s[cur] = i;

int mi = 0;

per(j,0,cur-1){

if(mp[i][s[j]] == 1 || mp[s[j]][i] == 1){

tmp = max(bw,cur-j);

break;

}

}

//最理想化剪枝

per(j,0,n-1){

if(mp[i][j] == 1 && !vis[j]){

++mi;

}

}

if(mi >= ans){

continue;

}

if(tmp < ans){//最优性剪枝

vis[i] = true;

dfs(cur+1,tmp);

vis[i] = false;

}

//bw = tmp;//同一层的选择之间互不影响

}

}

int main(){

//std::ios::sync\_with\_stdio(false);

while(~scanf("%s",str) && str[0] != '#'){

memset(vis,false,sizeof(vis));

memset(mp,0,sizeof(mp));

ans = 20;

set<int> st;

//if(strcmp(str,"#") == 0){

// break;

//}

int size = strlen(str);

per(j,0,size-1){

if(isalpha(str[j])){

st.insert(str[j] - 'A');

}

}

int cnt = 0;

set<int>::iterator it = st.begin();

for(it = it;it != st.end();++it){//映射，一直WA的原因是题目所给的字母不一定是相邻的，所以字母范围不是A-H

//有可能是A和Z两个字母

id[\*it] = cnt;//压缩，减少mp的空间

back[cnt++] = \*it;//还原答案

}

int i = 0;

while(i < size){

int node = str[i] - 'A';

i += 2;

while(i < size && str[i] != ';'){

mp[id[str[i] - 'A']][id[node]] = 1;

mp[id[node]][id[str[i] - 'A']] = 1;

++i;

}

++i;

}

n = st.size();

dfs(0,0);

per(j,0,n-1){

printf("%c ",'A' + back[sy[j]]);

}

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

}

return 0;

}

32，马拉车算法模板

string Manacher(string s){

// Insert '#'

string t = "$#";

for (int i = 0; i < s.size(); ++i){

t += s[i];

t += "#";

}

// Process t

vector<int> p(t.size(),0);

int mx = 0, id = 0, resLen = 0, resCenter = 0;

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

p[i] = mx > i ? min(p[2 \* id - i], mx - i) : 1;

while (t[i + p[i]] == t[i - p[i]]) ++p[i];

if (mx < i + p[i]){

mx = i + p[i];

id = i;

}

if (resLen < p[i]){

resLen = p[i];

resCenter = i;

}

}

cout << resCenter << " " << resLen << endl;

return s.substr((resCenter - resLen) / 2, resLen - 1);

}

int main(){

string s1 = "12212";

cout << Manacher(s1) << endl;

/\*

string s2 = "122122";

cout << Manacher(s2) << endl;

string s = "waabwswfd";

cout << Manacher(s) << endl;

\*/

return 0;

}

33，马拉车-最长回文

int Manacher(string s){

// Insert '#'

string t = "$#";//开始处，加一个$

for (int i = 0; i < s.size(); ++i){//预处理

t += s[i];

t += "#";

}

// Process t

vector<int> p(t.size(), 0);

int mx = 0, id = 0, resLen = 0, resCenter = 0;

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

p[i] = mx > i ? min(p[2 \* id - i], mx - i) : 1;//关键代码

while (t[i + p[i]] == t[i - p[i]]) ++p[i];//不断遍历，更新p[i]

if (mx < i + p[i]){ //更新右边的最大值

mx = i + p[i];

id = i; //更新中点值

}

if (resLen < p[i]) {//更新字符串长度

resLen = p[i];

resCenter = i; //更新中点值

}

}

//return s.substr((resCenter - resLen) / 2, resLen - 1);

//从第(resCenter-resLen)开始截取resLen-1长度的字符串

return resLen - 1 ;//resLen是以中心为原点，向两边拓展的值，其中中心算了两次，

//所以resLen-1就是回文串的长度

}

int main(){

string str;

while(cin >> str){

cout << Manacher(str) << endl;

}

return 0;

}

34，散列法-哈希函数

const int M = 1000010 ;

const int L = 14;

char H[M][L];

//将字符转化为数值

int getChar(char ch)

{

if(ch == 'A') return 1;

else if(ch == 'C') return 2;

else if(ch == 'G') return 3;

else if(ch == 'T') return 4;

else return 0;

}

//将字符串转化为数值并生成key

long long getKey(char str[])

{

long long sum = 0, p = 1;

for(size\_t i = 0;i < strlen(str);i++){

sum = p\*(getChar(str[i]));

p \*= 5;

}

return sum;

}

int h1(long long key)

{

return key % M;

}

int h2(long long key)

{

return 1 + (key % (M-1));

}

int hf(long long key,int i)//开放地址法

{

return ( h1(key) + i \* h2(key)) % M;

}

int find(char str[])

{

long long key, i ,h;

key = getKey(str);

for(i = 0; ;i++){

h = hf(key,i);

if(strcmp(H[h] , str) == 0) return 1;

else if(strlen(H[h]) == 0) return 0;

}

return 0;

}

int insert(char str[])

{

long long key,i,h;

key = getKey(str);

for(i = 0; ; i++){

h = hf(key,i);

if(strcmp(H[h],str) == 0) return 1;//原来就有该元素

else if (strlen(H[h]) == 0){ //找到新的空位置

strcpy(H[h],str);

return 0;

}

}

return 0;

}

int main()

{

int n;

char str[L],com[9];

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

H[i][0] = '\0'; //初始化，清空

}

scanf("%d",&n);

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

scanf("%s %s",com,str);

if(com[0] == 'i'){

insert(str);

}

else{

if(find(str)){

printf("yes\n");

}

else{

printf("no\n");

}

}

}

return 0;

}

35，骚操作-自定义二分查找-Coffe Break

#include <bits/stdc++.h>

using namespace std;

const int N = 2e5 + 10;

struct Node{

int v, id;

};

Node a[N];

int rec[N];

set <Node> s;

bool operator <(Node x, Node y){

if (x.v == y.v) return x.id < y.id;

return x.v < y.v;

}

bool cmp(const Node& x, const Node& y){

return x.v < y.v;

}

int main(){

int n, m, d, cnt;

set <Node>::iterator it;

Node ntemp;

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

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

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

a[i].id = i;

}

sort(a + 1, a + n + 1, cmp);

cnt = 0;

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

ntemp.v = a[i].v;

ntemp.id = 0;

it = s.upper\_bound(ntemp);

if (it == s.begin()){

ntemp.v = a[i].v + d;

ntemp.id = ++cnt;

s.insert(ntemp);

rec[a[i].id] = cnt;

}

else{

it--;

ntemp = \*it;

rec[a[i].id] = ntemp.id;

ntemp.v = a[i].v + d;

s.erase(it);

s.insert(ntemp);

}

}

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

for (int i = 1; i <= n; i++) printf("%d ",rec[i]);

printf("\n");

return 0;

}

36，骚操作-自定义优先队列比较方式-Silver Cow Party

const int maxn=1e3 + 10;

int dx[4] = {0,0,-1,1};

int dy[4] = {1,-1,0,0};

int n = 0,m = 0,x = 0;

int adj[maxn][maxn];

int d1[maxn],d2[maxn];

bool used[maxn];

struct cmp{//cmp类型，然后重新定义比较方式

bool operator()(const pii a,const pii b){

return a.second > b.second;//second小值优先

}

};

void dijikstra(int s,int\* dist){

fill(used,used+maxn,false);

fill(dist,dist+maxn,INF);

priority\_queue<pii,vector<pii>,cmp > que;

dist[s] = 0;//初始值设置

que.push(pii(s,0));

while(!que.empty()){

pii p = que.top();que.pop();//从未访问过的节点中选取距离远点最短的(更新距离的那些点）

int u = p.first,d = p.second;

if(dist[u] < d || used[u] == true){//如果一直最小值比d小，或者已经访问过，就直接跳过

continue;

}

used[u] = true;

for(int i = 1;i <= n;++i){//更新周围点的最短路径

if(dist[i] > dist[u] + adj[u][i] && used[i] == false){

dist[i] = dist[u] + adj[u][i];

que.push(pii(i,dist[i]));

}

}

}

}

int main(){

#ifndef ONLINE\_JUDGE

//freopen("a.txt","r",stdin);

#endif

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

fill(adj[0],adj[0] + maxn\*maxn,INF);

per(i,1,m){

int a = 0,b = 0,t = 0;

scanf("%d %d %d",&a,&b,&t);

adj[a][b] = t;

}

dijikstra(x,d1);

per(i,1,n){//路径反转

per(j,i+1,n){//j要从i+1开始，否则反转两次，相当于没变化

swap(adj[i][j],adj[j][i]);

}

}

dijikstra(x,d2);

int maxv = 0;

per(i,1,n){

if(d1[i] != INF && d2[i] != INF){//必须加上这个判断条件

maxv = max(maxv , d1[i] + d2[i]);

}

}

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

}

return 0;

}

37，树状数组（单点更新）-Cows

const int maxn = 1e5 + 10;

struct Node{

int x,y;

int id;

};

Node a[maxn];

int tree[maxn],level[maxn];

int n = 0;

bool cmp(Node a,Node b){

if(a.y != b.y){

return a.y > b.y;

}else{

return a.x < b.x;

}

}

int lowbit(int x){

return x & (-x);

}

void add(int loc,int val){

for(int i = loc;i <= maxn;i += lowbit(i)){

tree[i] += val;

}

/\*

int i = loc;

while(i <= loc){

tree[i] += val;

i += lowbit(i);

}

\*/

}

int getsum(int loc){

int ans = 0;

for(int i = loc;i > 0;i -= lowbit(i)){

ans += tree[i];

}

/\*

int i = loc;

while(i > 0){

ans += tree[i];

i -= lowbit(i);

}

\*/

return ans;

}

int main(){

while(scanf("%d",&n) != EOF && n){

memset(tree,0,sizeof(tree));

memset(level,0,sizeof(level));

for(int i = 0;i < n;++i){//因为y是递增的，所以可以不用考虑y

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

++a[i].x,++a[i].y; //避免0的出现

a[i].id = i; //当前从区间1-x的和（闭区间），即是水平数

//但是，因为是递增的，所以 只能看当前1-x区间的总和

//然后再单点更新（不包括本身）

}

sort(a,a + n,cmp);//

level[a[0].id] = getsum(a[0].x);

add(a[0].x,1);

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

if(a[i].x == a[i-1].x && a[i].y == a[i-1].y){

level[a[i].id] = level[a[i-1].id];

}else{

level[a[i].id] = getsum(a[i].x);

}

add(a[i].x,1);

}

printf("%d",level[0]);

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

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

}

printf("\n");

}

return 0;

}

38，树状数组(区间更新模板）-color the ball

const int maxn = 1e5 + 10;

int tree[maxn];

int n = 0;

//输入的是对区间的操作，如果最终要求的是单点的值，

//那么updata区间1 - m更新操作 - 从上往下，单点求和操作 - 从下往上。

int lowbit(int x){

return x & (-x);

}

void add(int loc,int val){//在1 - loc位置增加val

while(loc > 0){

tree[loc] += val;

loc -= lowbit(loc);

}

}

int getsum(int num){

int sum = 0;

while(num <= n){

sum += tree[num];

num += lowbit(num);

}

return sum;

}

/\*

void add(int loc,int val){//在loc位置增加val

if(loc < 1){

return ;

}

for(int i = loc;i <= n;i += lowbit(i)){//单点更新，从叶子节点向上更新tree[]数组，是查询的逆过程

tree[i] += val;

}

}

\*/

/\*

int getsum(int num){//从上到下

int ans = 0;

for(int i = num;i > 0;i -= lowbit(i)){

ans += tree[i];

}

return ans;

}

\*/

int main(){

std::ios::sync\_with\_stdio(false);

int a = 0,b = 0;

while(cin >> n && n){

//while(scanf("%d",&n) != EOF && n){

memset(tree,0,sizeof(tree));

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

cin >> a >> b;

add(b,1);

add(a - 1,-1);

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

if(j > 1){

cout << " ";

}

cout << tree[j] ;//<< endl;

}

cout << endl;

}

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

if(j != 1){

cout << " ";

}

cout << getsum(j);

}

cout << endl;

}

return 0;

}

39，数论-Miller\_rbin-素数判定

const int times = 20;

int number = 0;

map<long long, int>m;

long long Random( long long n ) //生成[ 0 , n ]的随机数

{

return ((double)rand( ) / RAND\_MAX\*n + 0.5);

}

long long q\_mul( long long a, long long b, long long mod ) //快速计算 (a\*b) % mod

{

long long ans = 0;

while(b)

{

if(b & 1)

{

//b--;

ans =(ans+ a)%mod;

}

b /= 2;

a = (a + a) % mod;

}

return ans;

}

long long q\_pow( long long a, long long b, long long mod ) //快速计算 (a^b) % mod

{

long long ans = 1;

while(b)

{

if(b & 1)

{

ans = q\_mul( ans, a, mod );

}

b /= 2;

a = q\_mul( a, a, mod );

}

return ans;

}

bool witness( long long a, long long n )//miller\_rabin算法的精华

{//用检验算子a来检验n是不是素数

long long tem = n - 1;

int j = 0;

while(tem % 2 == 0)

{

tem /= 2;

j++;

}

//将n-1拆分为a^r \* s

long long x = q\_pow( a, tem, n ); //得到a^r mod n

if(x == 1 || x == n - 1) return true; //余数为1则为素数

while(j--) //否则试验条件2看是否有满足的 j

{

x = q\_mul( x, x, n );

if(x == n - 1) return true;

}

return false;

}

bool miller\_rabin( long long n ) //检验n是否是素数

{

if(n == 2)

return true;

if(n < 2 || n % 2 == 0)

return false; //如果是2则是素数，如果<2或者是>2的偶数则不是素数

for(int i = 1; i <= times; i++) //做times次随机检验

{

long long a = Random( n - 2 ) + 1; //得到随机检验算子 a

if(!witness( a, n )) //用a检验n是否是素数

return false;

}

return true;

}

int main(){

long long tar;

while(cin >> tar){

if(miller\_rabin( tar )) //检验tar是不是素数

cout << "Yes, Prime!" << endl;

else

cout << "No, not prime.." << endl;

}

return 0;

}

40，数论-埃式筛选法（改进版）

const double eps=1.0e-5;

const int maxn=1e6 + 10;

int dx[4] = {0,0,-1,1};

int dy[4] = {1,-1,0,0};

int cnt1=0,cnt2=0,cnt3=0,tmp = 0,a[maxn],b[maxn],c[maxn],ma=-inf,flag;

int n = 0;

bool is\_prime[maxn];

int prime[maxn];

int eh\_ais(){

fill(is\_prime,is\_prime+n,true);

fill(prime,prime+n,0);

int cnt = 0,m = sqrt(n + 0.5);

per(i,2,m){

if(is\_prime[i]){

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

is\_prime[j] = false;

}

}

}

per(i,2,n){//遍历

if(is\_prime[i]){

prime[cnt++] = i;

}

}

return cnt;

}

int main(){

#ifndef ONLINE\_JUDGE

//freopen("a.txt","r",stdin);

#endif

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

int cnt = eh\_ais();

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

per(i,0,cnt-1){

printf("%d%c",prime[i],(i == cnt-1)?'\n':' ');

}

}

return 0;

}

41，数论-大数因数分解Pollard\_rho算法-POJ2429 GCD & LCM Inverse-self

//#define INF 0x3f3f3f//使用这个INF会WA

const LL INF=(LL)1<<61;

const int maxn = 2e5;

const int times = 100;

map<LL,int> mp;

int cnt = 0;

LL gc = 0,lc = 0,c = 234237,k = 0;

LL minv = INF;

LL resa = 0,resb = 0;

LL gcd(LL a,LL b){

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

}

LL q\_mul(LL a,LL b,LL mod){

LL res = 0;

while(b > 0){

if(b & 1){

res = (res + a) % mod;

}

a = (a + a) % mod;

b >>= 1;

}

return res;

}

LL q\_pow(LL a,LL b,LL mod){

LL res = 1;

while(b > 0){

if(b & 1){

res = q\_mul(res,a,mod);

}

a = q\_mul(a,a,mod);

b >>= 1;

}

return res;

}

bool witness(LL a,LL n){

LL temp = n - 1;

int j = 0;

while(temp % 2 == 0){

temp /= 2;

j++;

}

LL x = q\_pow(a,temp,n);

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

return true;

}

while(j--){

x = q\_mul(x,x,n);

if(x == n-1){

return true;

}

}

return false;

}

bool Miller\_rabin(LL n){

if(n == 2){

return true;

}

if(n < 2 || (n % 2 == 0 && n != 2)){

return false;

}

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

LL r = rand() % (n - 2) + 1;//n-1和n-2都可以

if(witness(r,n) == false){

return false;

}

}

return true;

}

LL pollard\_rho(LL n,LL c){

LL i = 1,k = 2;

LL x = rand() % (n-1) + 1;

LL y = x;

while(true){

i++;

x = (q\_mul(x,x,n) + c) % n;

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

if(1 < d && d < n){

return d;

}

if(y == x){

return n;

}

if(i == k){

y = x;

k <<= 1;

}

}

}

void find(LL n,LL c){

if(n == 1){

return ;

}

if(Miller\_rabin(n) == true){

mp[n]++;

return ;

}

LL p = n;

while(p >= n){

p = pollard\_rho(p,c--);

}

find(p,c);find(n/p,c);

}

void dfs(int depth,LL a){

if(depth == cnt){

LL b = k / a;

if(gcd(a,b) == 1){

if(a + b < minv){

minv = a + b;

resa = a;resb = b;

}

}

return ;

}

map<LL,int>::iterator it = mp.begin();

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

it++;

}

//cout << "hah " << it->first << " " << it->second << endl;

LL temp = q\_pow(it->first,it->second,INF);

//cout << temp << endl;

if(a \* temp < minv){

dfs(depth + 1,a \* temp);

}

if(a < minv){

dfs(depth + 1,a);

}

}

void solve(LL n,LL c){

find(k,c);

cnt = mp.size();

/\*

map<LL,int>::iterator it = mp.begin();

for(it = mp.begin();it != mp.end();it++){

cout << "em " << it->first << " " << it->second << endl;

}

\*/

dfs(0,1);

if(resa > resb){

swap(resa,resb);

}

printf("%I64d %I64d\n",resa \* gc,resb \* gc);

}

int main(){

while(~scanf("%I64d %I64d",&gc,&lc)){

minv = INF;

c = 2137342;

mp.clear();

k = lc / gc;

solve(k,c);

}

return 0;

}

42，数论-大数因数分解Pollard\_rho算法-self

#define per(i,a,b) for(int i=a;i<=b;i++)

typedef long long LL;

const int times = 20;

map<LL,int> mp;

int number = 0;

LL Random(LL n){

return ((double)rand() / RAND\_MAX \* n + 0.5);

}

LL q\_mul(LL a,LL b,LL mod){

LL ans = 0;

while(b > 0){

if(b & 1){

ans = (ans + a) % mod;

}

b >>= 1;

//b /= 2;

a = (a + a) % mod;

}

return ans;

}

LL q\_pow(LL a,LL b,LL mod){

LL ans = 1;

while(b > 0){

if(b & 1){

ans = q\_mul(ans,a,mod);

}

b >>= 1;

//b /= 2;

a = q\_mul(a,a,mod);

}

return ans;

}

LL gcd(LL a,LL b){

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

}

bool witness(LL a,LL n){//Miller\_rbin()算法的精华所在

LL temp = n - 1;//是n-1

int j = 0;

while(temp % 2 == 0){

temp /= 2;

j++;

}

LL x = q\_pow(a,temp,n);//计算a^r

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

return true;

}

while(j--){

x = q\_mul(x,x,n);

if(x == n-1){

return true;

}

}

return false;

}

bool Miller\_rbin(LL n){

if(n == 2){//这两个顺序不能变 ，因为这个条件，会被包含在第二个判断里面但是要先判断

return true;

}

if(n < 2 || n % 2 == 0){//这个判断必须在后面，想哭

return false;

}

per(i,1,times){

LL a = Random(n - 2) + 1;

if(witness(a,n) == false){

return false;

}

}

return true;

}

LL pollard\_rho(LL n,LL c){

LL i = 1,k = 2;

LL x = Random(n-1) + 1;

LL y = x;

while(true){

i++;

x = (q\_mul(x,x,n) + c) % n ;

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

if(1 < d && d < n){

return d;

}

if(y == x){

return n;

}

if(i == k){//一种不清楚的优化

y = x;

k <<= 1;

}

}

}

void find(LL n,LL c){//c是随机因子

if(n == 1){

return ;

}

if(Miller\_rbin(n) == true){

number++;

mp[n]++;

return ;

}

LL p = n;

while(p >= n){

p = pollard\_rho(p,c--);

}

find(p,c);

find(n/p,c);

}

int main(){

#ifndef ONLINE\_JUDGE

//freopen("a.txt","r",stdin);

#endif

LL n = 0;

while(~scanf("%I64d",&n)){

number = 0;

mp.clear();

LL mi = 2137342;

find(n, mi);

printf("%I64d = ",n);

if(mp.empty()){

printf("%I64d\n",n);

}

map<LL,int>::iterator it ;

for(it = mp.begin();it != mp.end();){

printf("%I64d^%d",it->first,it->second);

if((++it) != mp.end()){

printf(" \* ");

}

}

printf("\n");

}

return 0;

}

43，数论-欧拉筛选法

const int maxn=1e6 + 10;

int dx[4] = {0,0,-1,1};

int dy[4] = {1,-1,0,0};

int cnt1=0,cnt2=0,cnt3=0,tmp = 0,a[maxn],b[maxn],c[maxn],ma=-inf,flag;

int n = 0;

bool is\_prime[maxn];

int prime[maxn];

int ouler(){

fill(is\_prime,is\_prime+n,true);

fill(prime,prime+n,0);

int cnt = 0;

//int m = sqrt(n+0.5);

per(i,2,n){

if(is\_prime[i]){

prime[cnt++] = i;

}

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

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

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

break;

}

}

}

return cnt;

}

int main(){

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

int cnt = ouler();

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

per(i,0,cnt-1){

printf("%d%c",prime[i],(i == cnt-1)?'\n':' ');

}

}

return 0;

}

44，数论-欧拉筛选法-Enlarge GCD-好题-self

int n = 0;

int prime[maxn];

int a[maxn];

int u[maxn],s[maxn];

int gcd(int x,int y){

return y == 0 ? x : gcd(y,x%y);

}

void ouler(){

fill(u,u+maxn,0);

fill(prime,prime+maxn,0);

int cnt = 0;

per(i,2,maxn-1){

if(u[i] == 0){

u[i] = prime[cnt++] = i;

}

for(int j = 0;j < cnt && i \* prime[j] < maxn;++j){

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

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

break;

}

}

}

}

int main(){

#ifndef ONLINE\_JUDGE

//freopen("a.txt","r",stdin);

#endif

ouler();

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

fill(s,s+maxn,0);

int g = 0,x = 0;

per(i,1,n){

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

g = gcd(max(g,a[i]),min(g,a[i]));//求所有数的最大公因数

}

per(i,1,n){

for(int j = a[i]/g;j > 1;){//先除以最大公约数

s[x = u[j]]++; //除以相应的素数，+1，同一个数对于同一个素数只加一次

while(u[j] == x){//将这个素数除尽

j /= u[j];

}

}

}

/\*

per(i,1,n){

for(int j = a[i]/g;j > 1;){//先除以最大公约数

//除以相应的素数，+1，同一个数对于同一个素数只加一次

//将这个素数除尽

for(++s[x = u[j]];u[j] == x;){

j /= u[j];

}

}

}

\*/

int ans = 0;

per(i,1,maxn){

ans = max(ans,s[i]);//取一个素数能整除最多的数的数

}

printf("%d\n",ans == 0 ? -1 : n - ans);

}

return 0;

}

45，数论-区间内素数的个数-模板

#include<bits/stdc++.h>

using namespace std;

typedef long long LL;

const int maxn = 1e6 + 100;

bool is\_prime[maxn];

bool is\_prime\_small[maxn];

LL prime[maxn];

int segment\_sieve(LL a,LL b){

for(int i = 0;(LL)i \* i < b;++i){

is\_prime\_small[i] = true;

}

for(int i = 0;i < b - a;++i){//左闭右开区间

is\_prime[i] = true;

}

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

if(is\_prime\_small[i]){

for(int j = 2 \* i;(LL)j \*j <= b;j += i){//筛选(0,根号b)的素数

is\_prime\_small[j] = false;

}

for(LL j = max(2LL , ((a + i - 1)/i) \* i);j < b;j += i){//筛选[a,b)的素数

if(j == i){//避免把本身也

continue;

}

is\_prime[j-a] = false;

}

}

}

int cnt = 0;

for(int i = 0;i < b-a;++i){

if(is\_prime[i] == true){

cnt++;

}

}

return cnt;

}

int main(){

LL a = 0,b = 0;

while(cin >> a >> b){

cout << segment\_sieve(a,b) << endl;

}

return 0;

}

46，数论-无穷级数-Harmonic Number

#include<bits/stdc++.h>

#define r 0.57721566490153286060651209//欧拉常数

using namespace std;

int n = 0;

const int maxn = 1e4;

double ans[maxn+10];

void pre\_solve(){

ans[0] = 0;

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

ans[i] = ans[i-1] + 1.0/i;

}

}

int main(){

int T = 0;

scanf("%d",&T);

int kase = 0;

pre\_solve();

while(T--){

scanf("%d",&n);

if(n <= maxn){

printf("Case %d: %.10lf\n",++kase,ans[n]);

}else{

printf("Case %d: %.10lf\n",++kase,log(n+1) + r - 1.0/(2\*n));

//调和级数的公式log(n+1) + r(欧拉常数)

//这道题再进行修正，减去1.0/(2\*n)刚好，尝试加上一些可行项

}

}

return 0;

}

47，数论-质因数分解-全排列-3421-X-factor Chains

const int maxn=3e5 + 10;

int dx[4] = {0,0,-1,1};

int dy[4] = {1,-1,0,0};

int n = 0;

int a[maxn];

map<int,int> mp;

void get\_prime(int x){

int temp = x;

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

while(temp % i == 0){

temp /= i;

mp[i]++;

}

}

if(temp > 1){

mp[temp]++;

}

}

LL mul\_fac(int x){

LL res = 1;

per(i,2,x){

res \*= i;

}

return res;

}

int main(){

#ifndef ONLINE\_JUDGE

//freopen("a.txt","r",stdin);

#endif

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

mp.clear();

get\_prime(n);

map<int,int>::iterator it = mp.begin();

int num = 0;//记录所有质因数指数之和

LL mul = 1;

for(it = mp.begin();it != mp.end();it++){

mul \*= mul\_fac(it->second);

num += it->second;

}

LL ans = mul\_fac(num) / mul;

printf("%d %I64d\n",num,ans);

}

return 0;

}

48，数位DP-不要62-DFS模板

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

int d[20];

int dp[20][2];

//pre适用于有前导0的情况 ,sta是题目中的合法状态的记录

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

if(pos == 0){//枚举完了数位，说明合法

return 1;

}

if(!lim && dp[pos][sta] != -1){//如果没有限制，返回全部

return dp[pos][sta];

}

int up = lim ? d[pos] : 9;//枚举的上限

int temp = 0;

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

if(pre == 6 && i == 2){

continue;

}

if(i == 4){

continue;

}

temp += dfs(pos-1,i,i==6,lim && i==d[pos]);

}

if(!lim){//如果没有限制，返回全部

dp[pos][sta] = temp;

}

return temp;

}

int solve(int x){//数字的拆分

int temp = x;

int cnt = 0;

while(temp > 0){

d[++cnt] = temp % 10;

temp /= 10;

}

d[cnt+1] = 0;

return dfs(cnt,-1,0,true);//初始化状态：pos:cnt,前面的数字-1，

}

void init(){

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

}

int main(){

int n = 0,m = 0;

while(scanf("%d %d",&n ,&m) != EOF && (m >= n) && n){

init();

printf("%d\n",solve(m) - solve(n-1));//闭区间[0,m]-[0,n-1] = [n,m]

}

return 0;

}

49，图论大师-强连通分量分解-Popular Cows-POJ2186

const int maxn = 2e4;

int n = 0,m = 0;

vector<int> g[maxn+10],rg[maxn+10];

vector<int> post;

bool vis[maxn+10];

int dfn[maxn+10];

void init(){

memset(vis,false,sizeof(vis));

per(i,1,n){

g[i].clear(); rg[i].clear();

}

post.clear();

}

void dfs(int s){

vis[s] = true;

int size = g[s].size();

per(i,0,size-1){

int v = g[s][i];

if(!vis[v]){

dfs(v);

}

}

post.push\_back(s);

}

void rdfs(int s,int id){

vis[s] = true;

dfn[s] = id;

int size = rg[s].size();

per(i,0,size-1){

int v = rg[s][i];

if(!vis[v]){

rdfs(v,id);

}

}

}

int scc(){

per(i,1,n){

if(!vis[i]){

dfs(i);

}

}

memset(vis,false,sizeof(vis));

int size = post.size();

int k = 1;

rep(i,size-1,0){

int v = post[i];

if(!vis[v]){

rdfs(v,k++);

}

}

return k;

}

int main(){

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

init();

per(i,1,m){

int u = 0,v = 0;

scanf("%d %d",&u,&v);

g[v].push\_back(u); rg[u].push\_back(v);

}

int k = scc() - 1;

int u = 0,num = 0;

per(i,1,n){

if(dfn[i] == 1){//这里是1，而不是k,因为红牛是能到达其他所有牛的（我是这样建图的）

++num;

u = i;

}

}

memset(vis,false,sizeof(vis));

dfs(u);

per(i,1,n){

if(!vis[i]){

num = 0;

break;

}

}

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

}

return 0;

}

50，线段树+RMQ-2019南昌网络邀请赛-I-Max Answer

#include<bits/stdc++.h>

#define per(i,a,b) for(int i = (a);i <= (b);++i)

#define rep(i,a,b) for(int i = (a);i >= (b);--i)

#define INF 0x3f3f3f

using namespace std;

typedef long long LL;

const int maxn = 5e5;

int n = 0,m = 0;

LL a[maxn+10];

LL pre[maxn+10],suf[maxn+10],sum = 0;

int L[maxn+10],R[maxn+10];

struct node{

LL ma,mi;

int l,r;

};

node tree[2][4\*maxn+10];//0是往左边的，1是往右边的

void build\_tree(int l,int r,int root){

tree[0][root].l = l,tree[0][root].r = r;

tree[1][root].l = l,tree[1][root].r = r;

if(l == r){

tree[0][root].mi = tree[0][root].ma = sum - pre[l-1];

tree[1][root].mi = tree[1][root].ma = sum - suf[l+1];

}else{

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

build\_tree(l,mid,root<<1);

build\_tree(mid+1,r,root<<1|1);

tree[0][root].mi = min(tree[0][root<<1].mi,tree[0][root<<1|1].mi);

tree[0][root].ma = max(tree[0][root<<1].ma,tree[0][root<<1|1].ma);

tree[1][root].mi = min(tree[1][root<<1].mi,tree[1][root<<1|1].mi);

tree[1][root].ma = max(tree[1][root<<1].ma,tree[1][root<<1|1].ma);

}

return ;

}

LL query\_max(int l,int r,int root,int id){

if(tree[id][root].l > r || tree[id][root].r < l){

return -1e18;

}else if(l <= tree[id][root].l && r >= tree[id][root].r){

return tree[id][root].ma;

}else{

return max(query\_max(l,r,root<<1,id),query\_max(l,r,root<<1|1,id));

}

}

LL query\_min(int l,int r,int root,int id){

//if(l > r){

// return 0;

//}

if(tree[id][root].l > r || tree[id][root].r < l){

return 1e18;

}else if(l <= tree[id][root].l && r >= tree[id][root].r){

return tree[id][root].mi;

}else{

return min(query\_min(l,r,root<<1,id),query\_min(l,r,root<<1|1,id));

}

}

void Mo\_stack(){

stack<int> st;

per(i,1,n){

while(!st.empty() && a[i] <= a[st.top()]){

st.pop();

}

if(st.empty()){

L[i] = 0;

}else{

L[i] = st.top();

}

st.push(i);

}

while(!st.empty()){

st.pop();

}

rep(i,n,1){

while(!st.empty() && a[i] <= a[st.top()]){

st.pop();

}

if(st.empty()){

R[i] = n+1;

}else{

R[i] = st.top();

}

st.push(i);

}

}

void init(){

memset(pre,0,sizeof(pre)); memset(suf,0,sizeof(suf));

sum = 0;

per(i,0,1){

per(j,0,5e5\*4){

tree[i][j].mi = 1e18; tree[i][j].ma = -1e18;

}

}

}

int main(){

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

init();

per(i,1,n){

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

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

sum += a[i];

}

rep(i,n,1){

suf[i] = suf[i+1] + a[i];

}

Mo\_stack();

build\_tree(1,n,1);

LL maxv = -1e18;

per(i,1,n){

if(a[i] > 0){

//LL lm = max(query\_max(L[i]+1,i-1,1,0),1ll\*0),rm = max(query\_max(i+1,R[i]-1,1,1),1ll\*0);

//maxv = max(maxv,1ll\*a[i]\*(lm+a[i]+rm));

maxv = max(maxv,1ll\*a[i]\*(pre[R[i]-1] - pre[L[i]]));//最小值为正数的区间的数全>0,所以区间内的数全要

}else if(a[i] < 0){

//左边-suf[i],右边-pre[i]。

LL lm = min(query\_min(L[i]+1,i-1,1,0) - suf[i],1ll\*0),rm = min(query\_min(i+1,R[i]-1,1,1) - pre[i],1ll\*0);

maxv = max(maxv,1ll\*a[i]\*(lm+a[i]+rm));

}else{

maxv = max(maxv,1ll\*0);

}

}

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

}

return 0;

}

51，折半搜索-自定义二分搜索-Sumsets-最终版

const int maxn=1e6 + 10;

int cnt1=0,cnt2=0,cnt3=0,n = 0,tmp = 0,a[maxn],b[maxn],c[maxn],ma=-inf,flag;

struct node{

int sum;

int x,y;

};

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

return a.sum < b.sum;

}

struct cmp1{

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

return a.sum < b.sum;

}

};

void solve(){

vector<node> nod;

int cnt = 0;

sort(a,a+n);

per(i,0,n-1){

per(j,i+1,n-1){

node tmp;

tmp.sum = a[i] + a[j];

tmp.x = a[i];tmp.y = a[j];

nod.push\_back(tmp);

}

}

sort(nod.begin(),nod.end(),cmp);

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

for(int j = n-1;j >= 0;--j){//之前WA是没有考虑负数的情况，不过这个j循环改一下就行了

if(j == i){

continue;

}

int cd = a[i] - a[j];

node tmp;tmp.sum = cd;

vector<node>::iterator s = upper\_bound(nod.begin(),nod.end(),tmp,cmp1());

vector<node>::iterator e = lower\_bound(nod.begin(),nod.end(),tmp,cmp1());

if(s != e){

vector<node>::iterator k = e;

for(k = e;k != s;k++){

if((\*k).x != a[j] && (\*k).y != a[j] && (\*k).x != a[i] && (\*k).y != a[i]){

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

return ;

}

}

}

}

}

printf("no solution\n");

return ;

}

int main(){

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

per(i,0,n-1){

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

}

solve();

}

return 0;

}

52，字典树-链表实现-单词数

using namespace std;//统计文本中不同单词的个数

const int maxn = 1e4 + 10;//指针操作迷得一批，不过弄对了，说明你掌握了指针这个难点的原理

struct node{

int cnt;//cnt可以表示一个字典树到此有多少相同前缀的数目，这里根据需要应当学会自由变化

node\* nex[26];//next是表示每层有多少种类的数，如果只是小写字母，则26即可，

//若改为大小写字母，则是52，若再加上数字，则是62了，这里根

bool used;

node(){//类似于构造函数

cnt = 0;

memset(nex,0,sizeof(nex));

used = false;//标记是否用过

}

};

node \*root = new node;//新建立一个空间，并把地址给root，作为根结点

void insert(string s){

node\* p = root;

int len = s.size();

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

if(p->nex[s[i] - 'a'] == NULL){

//cout << "qq: " << s << endl;

node\* temp = new node;

p->nex[s[i] - 'a'] = temp;

}

p = p->nex[s[i] - 'a'];

++p->cnt;

}

}

int search(string s){

node\* p = root;

//int len = strlen(s);

int len = s.size();

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

// cout << "pp: " << s << endl;

if(p->nex[s[i] - 'a'] == NULL){

return 0;

}

p = p->nex[s[i] - 'a'];

if(i == len -1){//最后一个要特判

for(int j = 0;j < 26;++j){//遍历他的子节点，如果还有子节点，说明只是前缀，还要++ans

if(p->nex[j] && p->used == false){//用来处理前缀，比如ji,j

p->used = true;//used的标记是为了看前缀出现的次数，出现了的不能再计算

return 0;

}

}

}

}

return p->cnt;//

}

void del(node\* root){

/\*

for(int i = 0;i < 10;++i){//这个操作是失败的

if(root->nex[i]){

del(root->nex[i]);

}

}

del(root);

\*/

//delete root;//原来只是释放了根结点的空间，其他的没有释放

if(root == NULL){//释放所有空间，防止内存泄露

return ;

}else{

for(int i = 0;i < 26;++i){//这个操作是失败的

if(root->nex[i]){

del(root->nex[i]);

}

}

}

delete root;

}

int main(){

string s,str;

int flag = 0;

while(getline(cin , s)){

if(s == "#"){

break;

}

int ans = 0;

for(int i = 0;i < s.size();++i){

str = "";

flag = 0;

while(s[i] >= 'a' && s[i] <= 'z'){

str += s[i];

flag = 1;

++i;

}

//cout << "em " << str << endl;

if(flag == 1){

if(ans == 0){

insert(str);

//first = 0;

++ans;

}else

if(search(str) == 0){//原来不存在

insert(str);

++ans;

}

}

//cout << "f: " << ans << endl;

}

cout << ans << endl;

del(root);//记得销毁前面的，因为根结点是用一个指针表示的，所以只要销毁指针即可

root = new node;//记得还要新建一个，与后面的连续

}

return 0;

}

53，字典树-模板-统计难题

const int maxn = 1e4 + 10;

struct node{

int cnt;

node\* nex[26];

node(){

cnt = 0;

memset(nex,0,sizeof(nex));

}

}root;

void insert(char \*s){

node\* p = &root;

int len = strlen(s);

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

if(p->nex[s[i] - 'a'] == NULL){

node\* tmp = new node;

p->nex[s[i] - 'a'] = tmp;

}

p = p->nex[s[i] - 'a'];

++p->cnt;

}

}

int search(char \*s){

node\* p = &root;

int len = strlen(s);

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

if(p->nex[s[i] - 'a'] == NULL){

//printf("9\n");

return 0;

}

p = p->nex[s[i] - 'a'];

}

//printf("%d\n",p->cnt);

return p->cnt;

}

void del(node\* root){

for(int i = 0;i < 10;++i){

if(root->nex[i]){

del(root->nex[i]);

}

}

del(root);

}

int main(){

char str[12];

//while(gets(str) && strcmp(str,"") != 0){

while(gets(str) && str[0] != '\0'){//使用gets读取空行

insert(str);

}

while(scanf("%s",str) != EOF){

printf("%d\n",search(str));

}

//del(&root);//不注释掉会超时

return 0;

}

54，组合数学-鸽巢原理- Find a multiple

#include<bits/stdc++.h>

using namespace std;

typedef long long LL;

#define INF 0x3f3f3f3f

#define per(i,a,b) for(int i = a;i <= b;++i)

#define rep(i,a,b) for(int i = a;i >= b;--i)

const int maxn = 1e4;

int n = 0;

int a[maxn+10],pre[maxn+10];

int mod[maxn+10];

/\*

鸽巢原理的妙用:

求前缀和，pre%n有n中可能，如果mod == 0，那么1->i就是我们要的结果。

否则%n有余数，1->n-1（排除余数为0的情况），中共n-1中状态，而有n个余数，所以必然有两个余数相等，那么余数相等的

两个区间差就是答案。

\*/

int main(){

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

memset(pre,0,sizeof(pre)); memset(mod,0,sizeof(mod));

bool flag = false;

per(i,1,n){

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

if(flag == true){//已经输出了答案，只要读取数据就可以了

continue;

}

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

if(pre[i] % n == 0){

flag = true;

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

per(j,1,i){

printf("%d\n",a[j]);

}

}else{

if(mod[pre[i] % n] != 0){

flag = true;

printf("%d\n",i - mod[pre[i] % n]);

per(j,mod[pre[i]%n]+1,i){

printf("%d\n",a[j]);

}

}else{

mod[pre[i] % n] = i;

}

}

}

}

return 0;

}

55，最短路-Bellman\_Ford模板

#include<iostream>

#include<cstdio>

#include<cstring>

#define INF 1e5

using namespace std;

typedef long long LL;

const int maxn = 1e4;

const int maxv = 1e3;

int V = 0,E = 0;

struct Edge{

int u,v;

int w;

};

Edge edge[maxn];

int dist[maxv];

//求解从顶点S出发到所有点的最短距离

void B\_F(int s){//挑战的模板 ，只能用于没有负边的图

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

dist[i] = INF;

}

dist[s] = 0;

while(true){ //如果有负边，会陷入while死循环

bool update = false;

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

Edge e = edge[i];

if(dist[e.u] != INF && dist[e.v] > dist[e.u] + e.w){

dist[e.v] = dist[e.u] + e.w;

update = true;

}

}

if(update == false){

break;

}

}

}

//时间复杂度:O(V\*E)

bool B\_F(int s){ //Bellman\_Ford算法,适用于负边的情况

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

dist[i] = INF;

}

dist[s] = 0;//起点设置为0

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

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

int u = edge[j].u,v = edge[j].v;

if(dist[u] != INF && dist[v] > dist[u] + edge[j].w){//relax,松弛操作

dist[v] = dist[u] + edge[j].w;

}

}

}

bool flag = true;

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

if(dist[edge[i].v] > dist[edge[i].u] + edge[i].w){//检查到负边

flag = false;

break;

}

}

return flag;

} //改错，因为存在负圈的话，根本就不能求出最短路

//不好意思，这个版本不能求最短路，只能找出负圈（此时无法求解最短路）

//bool B\_F(){ //Bellman\_Ford算法,适用于负边的情况

bool find\_negative\_loop(){

memset(dist,0,sizeof(dist));

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

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

int u = edge[j].u,v = edge[j].v;

if(dist[v] > dist[u] + edge[j].w){//relax,松弛操作

dist[v] = dist[u] + edge[j].w;

if(i == V){//如果第n次仍然更新了，则存在负边

return false;

}

}

}

}

return true;

}

int main(){

return 0;

}

56，最短路-Dijikstra-队列优化模板

int n = 0,m = 0,x = 0;

int adj[maxn][maxn];

int d1[maxn],d2[maxn];

bool used[maxn];

struct cmp{//cmp类型，然后重新定义比较方式

bool operator()(const pii a,const pii b){

return a.second > b.second;//second小值优先

}

};

void dijikstra(int s,int\* dist){

fill(used,used+maxn,false);

fill(dist,dist+maxn,INF);

priority\_queue<pii,vector<pii>,cmp > que;

dist[s] = 0;//初始值设置

que.push(pii(s,0));

while(!que.empty()){

pii p = que.top();que.pop();//从未访问过的节点中选取距离源点最短的(更新距离的那些点）

int u = p.first,d = p.second;

if(dist[u] < d || used[u] == true){//如果已知最小值比d小，或者已经访问过，就直接跳过

continue;

}

used[u] = true;

for(int i = 1;i <= n;++i){//更新周围点的最短路径

if(dist[i] > dist[u] + adj[u][i] && used[i] == false){

dist[i] = dist[u] + adj[u][i];

que.push(pii(i,dist[i]));

}

}

}

}

int main(){

#ifndef ONLINE\_JUDGE

//freopen("a.txt","r",stdin);

#endif

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

fill(adj[0],adj[0] + maxn\*maxn,INF);

per(i,1,m){

int a = 0,b = 0,t = 0;

scanf("%d %d %d",&a,&b,&t);

adj[a][b] = t;

}

dijikstra(x,d1);

per(i,1,n){//路径反转

per(j,i+1,n){//j要从i+1开始，否则反转两次，相当于没变化

swap(adj[i][j],adj[j][i]);

}

}

dijikstra(x,d2);

int maxv = 0;

per(i,1,n){

if(d1[i] != INF && d2[i] != INF){//必须加上这个判断条件

maxv = max(maxv , d1[i] + d2[i]);

}

}

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

}

return 0;

}

57，最短路-DJ-次短路-好题

using namespace std;//POJ 3255

const int INF = 5e8;

const int maxn = 5000 + 100;

int n = 0,r = 0;

struct edge{

int to;

int cost;

};

typedef pair<int,int> P;

vector<edge> G[maxn];//邻接表的正确姿势，结构体存储额外信息，存储边而不是点（这样便于写结构体）

int dist[maxn],dist2[maxn];//最短距离与次短距离

void solve(){

fill(dist,dist + maxn,INF);

fill(dist2,dist2 + maxn,INF);

dist[1] = 0;

priority\_queue<P,vector<P>,greater<P> >que;

que.push(P(1,0));

while(!que.empty()){

P p = que.top();

que.pop();

int v = p.first,d = p.second;//first是节点,second是对应的最短距离

if(dist2[v] < d){//如果条件成立，说明该点已经取到最短距离,d表示的是标记的距离，则不用更新了

continue;

//关键的终止条件，遍历相邻的点，更新相邻点的dist和dist2,并且这些点会遍历到以前遍历过

//的点并且会将他放入优先队列中，再次出对时就会判断的，就会将它排除

//这里比较难以理解，最好自己举例理解

//比如从1->2->3->1

//这是比较两者的dist，就是上面的那个条件，如果成立，

}

for(int i = 0;i < G[v].size();++i){//遍历该节点的邻接边

edge e = G[v][i];

int d2 = e.cost + d;

if(dist[e.to] > d2){//小于最短距离

swap(dist[e.to] , d2);

que.push(P(e.to,dist[e.to]));

dist2[e.to] = d2;

que.push(P(e.to,dist2[e.to]));

continue;

}

if(dist2[e.to] > d2 && dist[e.to] < d2){//处于最短距离和次短距离之间

dist2[e.to] = d2;

que.push(P(e.to,dist2[e.to]));

continue;

}

}

}

printf("%d\n",dist2[n]);

}

int main(){

while(scanf("%d %d",&n,&r) != EOF){

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

int a = 0,b = 0,c = 0;

edge e;

scanf("%d %d %d",&a,&b,&c);

e.cost = c;

e.to = b;G[a].push\_back(e);

e.to = a;G[b].push\_back(e);

}

solve();

}

return 0;

}

58，最短路-优先队列-Edge Deletion

//#define INF 0x3f3f3f3f //对long long来说太小了

#define INF LONG\_LONG\_MAX //这才是正确的姿势

#define PI acos(-1.0)

#define pii pair<int,LL> //pair也要改成long long,这里不改成LL,会WA53

#define all(x) x.begin(),x.end()

#define mem(a,b) memset(a,b,sizeof(a))

#define per(i,a,b) for(int i = a;i <= b;++i)

#define rep(i,a,b) for(int i = a;i >= b;--i)

const int maxn = 3e5;

int n = 0,m = 0,k = 0;

struct edge{

int id,to,w;

edge(int x,int y,int w1){

id = x;to = y;w = w1;

}

};

vector<edge> G[maxn+10];

bool vis[maxn+10];

LL d[maxn+10];

int father[maxn+10],faedge[maxn+10];

struct cmp{

bool operator () (const pii a,const pii b){

return a.second > b.second;

}

};

void dijikstra(int s){

fill(d,d+maxn+10,INF);

fill(vis,vis+maxn+10,false);

priority\_queue<pii,vector<pii>,cmp> pq;

d[s] = 0;

father[s] = 1;

pq.push(pii(s,0));

while(!pq.empty()){

pii p = pq.top();

pq.pop();//别忘记删除节点

int u = p.first;

if(vis[u] == true || d[u] < p.second){

continue;

}

vis[u] = true;

//per(i,0,G[u].size()-1){//这里面有宏定义错误，per(1,a,b)a <= b

//而这里a > b ，就会导致程序崩掉

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

edge v = G[u][i];

if(d[v.to] > d[u] + v.w && vis[v.to] == false){

d[v.to] = d[u] + v.w;

father[v.to] = u;//记录父亲节点

faedge[v.to] = v.id;//记录父亲边

pq.push(pii(v.to,d[v.to]));

}

}

}

}

vector<int> son[maxn+10];

vector<int> ans;

void bfs(int s){

queue<int> que;

int cnt = 0;

que.push(s);

while(!que.empty() && cnt <= k){

int u = que.front();

que.pop();

//per(i,0,son[u].size()-1){

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

if(cnt >= k){

break;

}

int v = son[u][i];

ans.push\_back(faedge[v]);

que.push(v);

cnt++;

}

}

}

int main(){

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

//ans.clear();

int x,y,w;

per(i,1,m){

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

G[x].push\_back(edge(i,y,w));

G[y].push\_back(edge(i,x,w));

}

dijikstra(1);

per(i,2,n){//将最短路径树的父亲关系转化为儿子关系

son[father[i]].push\_back(i);

}

bfs(1);

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

//per(i,0,ans.size()-1){//这里面有宏定义错误，per(1,a,b)a <= b

//而这里a > b ，就会导致程序崩掉

for(int i = 0;i < ans.size();++i){

printf("%d%c",ans[i],i == (ans.size()-1) ? '\n' : ' ');

}

}

return 0;

}

59, 最小表示法-Hidden Password

const int maxn = 1e5 + 10;

int n = 0;

string str;

int min\_rep(){

int i = 1,j = 0,k = 0;

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

k = 0;

while(str[(i+k) % n] == str[(j+k) % n] && k < n){

++k;

}

if(k == n){

return i;

}

if(str[i+k] > str[j+k]){

if(i +k +1 > j){

i = i+k+1;

}else{

i = j+1;

}

//i = max(i+k+1 ,j + 1);

}else{

if(j+k+1 > i){

j = j+k+1;

}else{

j = i+1;

}

//j = max(j+k+1, i+1);

}

}

return i < j? i :j;

}

int main(){

int T = 0;

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

cin >> T;

while(T--){

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

//scanf("%d",&)

cin >> n;

cin >> str;

cout << min\_rep() << endl;

//cout << Mini() << endl;

}

return 0;

}

60, 最小生成树-Kruskal-并查集-Conscription

const int maxn = 5e4 + 10;

int n = 0,m = 0,r = 0;

struct edge{

int u,v,d;

};

int rak[maxn],par[maxn];

edge es[maxn];

bool cmp(edge a,edge b){

return a.d < b.d;

}

void init(int n){

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

par[i] = i;

rak[i] = 0;

}

}

int find(int x){

if(par[x] == x){//找到根了

return x;

}else{

return find(par[x]);

}

}

void unite(int x,int y){

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

if(x == y){

return ;

}

if(rak[x] < rak[y]){

par[x] = y;

}else{

par[y] = x;

if(rak[x] == rak[y]){

rak[x] += 1;

}

}

}

bool same(int x,int y){

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

}

void solve(){

//int V = n + m;

int E = r;

init(n + m);

int res = 0;

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

edge e = es[i];

if(!same(e.u,e.v)){

unite(e.u,e.v);

res += e.d;

}

}

int ans = 1e4 \* (n + m) + res;

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

}

int main(){

int T = 0;

scanf("%d",&T);

while(T--){

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

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

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

es[i].d = -es[i].d;

es[i].v += n;//

}

sort(es,es + r,cmp);

solve();

}

return 0;

}

61, 最小生成树-prime（优先队列优化)-Agri-Net

#define PI acos(-1.0)

#define INF 0x3f3f3f3f

typedef long long LL;

typedef pair<int,int> pii;

const double eps=1.0e-5;

const int maxn=100 + 10;

int dx[4] = {0,0,-1,1};

int dy[4] = {1,-1,0,0};

int n = 0;

bool vis[maxn];

int adj[maxn][maxn];

int d[maxn];

struct cmp{

bool operator()(const pii a,const pii b){

return a.second > b.second;

}

};

void prime(int s){

fill(vis,vis+maxn,false);

fill(d,d+maxn,INF);

priority\_queue<pii,vector<pii>,cmp> que;

que.push(pii(s,0));

int res = 0;

while(!que.empty()){

pii p = que.top();

que.pop();

int u = p.first,v = p.second;

if(vis[u] == true || d[u] < v){//后面一个条件是为了去掉大于已知最小值的点

continue;//因为与已经访问过的点相邻的点都进入了优先队列，但是我们只访问

}//了其中距离已知点最近的那个，所以其他的可能是不行的

res += v;

vis[u] = true;

for(int i = 1;i <= n;++i){//除了这里的比较和更新，其他和DJ算法的写法是一样的

if(d[i] > adj[i][u] && vis[i] == false){

d[i] = adj[i][u];//此处与DJ算法的区别：DJ是选取距离原点最近的点

//而prime算法选取的是距离已访问节点到未访问节点距离最短的节点

que.push(pii(i,d[i]));

}

}

}

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

}

int main(){

#ifndef ONLINE\_JUDGE

//freopen("a.txt","r",stdin);

#endif

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

fill(adj[0],adj[0] + maxn\*maxn,INF);

per(i,1,n){

per(j,1,n){

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

}

}

prime(1);

}

return 0;

}

62, 坐标离散化-Paint Color

typedef long long LL;

const double eps=1.0e-5;

const int maxn=500 + 10;

int dx[4] = {0,0,-1,1};

int dy[4] = {1,-1,0,0};

int cnt1=0,cnt2=0,cnt3=0,tmp = 0,a[maxn],b[maxn],c[maxn],ma=-inf,flag;

int n = 0,w = 0,h = 0;

int xx1[maxn],xx2[maxn],yy1[maxn],yy2[maxn];//存储坐标

//对x1,x2进行坐标离散化，并返回离散化后的宽度

bool fld[3\*maxn][3\*maxn];

int compress(int\* xx1,int\* xx2,int w){

vector<int> xs;

per(i,0,n-1){

per(d,-1,1){

int tx1 = xx1[i] + d,tx2 = xx2[i] + d;

if(1 <= tx1 && tx1 <= w ){//不是tx1>=0,而是>=1,因为坐标是从1开始的，

//1是不行的,1是坐标的边界

xs.push\_back(tx1);

}

if(1 <= tx2 && tx2 <= w){

xs.push\_back(tx2);

}

}

}

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

xs.erase(unique(xs.begin(),xs.end()),xs.end());//删除去重数列的后面所有多余元素

//因为unique并没有删除重复元素，只是用后面的不重复元素填充重复元素

per(i,0,n-1){ //直线坐标压缩

xx1[i] = find(xs.begin(),xs.end(),xx1[i]) - xs.begin() + 1;

xx2[i] = find(xs.begin(),xs.end(),xx2[i]) - xs.begin() + 1;

}

return xs.size();

}

//压缩前的边界是1-w，压缩后是0-w-1，但是+1后就是1-w-1(而不是1-w),由于这道题加一

//之后后面的边界要减一，模板的左边是方块的坐标，而这道题是点的坐标，所以需要一些小转化

void solve(){

w = compress(xx1,xx2,w);

h = compress(yy1,yy2,h);

//fill(fld[0],fld[0]+9\*maxn\*maxn,false);

mem(fld,false);

per(i,0,n-1){//设置墙

per(x,xx1[i],xx2[i]-1){

per(y,yy1[i],yy2[i]-1){

fld[x][y] = true;

}

}

}

int ans = 0;

per(x,1,w-1){

per(y,1,h-1){//之前这里少见了1导致结果是27，而答案是6

if(fld[x][y] == true){

continue;

}

ans++;

queue<pair<int,int> > que;

que.push(make\_pair(x,y));

fld[x][y] = true;

while(!que.empty()){

int sx = que.front().first,sy = que.front().second;

que.pop();

per(i,0,3){

int tx = sx + dx[i],ty = sy + dy[i];

if(1 <= tx && tx < w && 1 <= ty && ty < h && fld[tx][ty] != true){

que.push(make\_pair(tx,ty));

fld[tx][ty] = true;

}

}

}

}

}

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

}

int main(){

#ifndef ONLINE\_JUDGE

freopen("a.txt","r",stdin);

#endif

while(~scanf("%d %d",&w,&h) && (w != 0 || h != 0)){

scanf("%d",&n);

w++;h++;//全部增加一

per(i,0,n-1){

scanf("%d %d %d %d",&xx1[i],&yy1[i],&xx2[i],&yy2[i]);

xx1[i]++;xx2[i]++;yy1[i]++;yy2[i]++;//全部增加一

}

solve();

}

return 0;

}

63，最大团-Maximum Clique-HDU1530DFS+DP剪枝（输出目标点）

const int maxn = 50;

int n = 0,m = 0;

int mp[maxn+10][maxn+10];

int dp[maxn+10];

int clique[maxn+10],cli[maxn+10],tmp[maxn+10];//clique存储最终结果，cli存储当前元素与所有点的关系

int ans = 0;

/\*

这个优化如何保证加入的点与之前的点都相连？

\*/

void dfs(int depth,int now){

if(depth + (n - now) + 1 <= ans || dp[now] + depth <= ans){

return ;

}

int t\_cli[maxn+10];

int able = 0;//最多有多少个点（即相邻的点的个数）

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

t\_cli[i] = cli[i];

if(cli[i] == 1){

++able;

}

}

if(able + depth <= ans){

return ;

}

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

if(t\_cli[i] == 0){//排除了那些与已知点不相邻的点 ，相当于之前模板的is\_clique函数

continue;

}

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

cli[j] = t\_cli[j] & mp[i][j];//cli[j]表示当与前节点now相邻的节点相邻节点

//便于下次递归计算able的值

}

tmp[depth+1] = i;

//cout << i << endl;

dfs(depth+1,i);

}

if(depth > ans){

ans = depth;

memcpy(clique,tmp,sizeof(tmp));

}

}

int max\_tuan(){

dp[n] = 1;ans = 1;

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

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

cli[j] = mp[i][j];

}

tmp[1] = i;

dfs(1,i+1);

dp[i] = ans;

}

return ans;

}

int main(){

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

memset(dp,sizeof(dp),0);

per(i,1,n){

per(j,1,n){

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

}

}

printf("%d\n",max\_tuan());

int num = max\_tuan();//输出最大团的点

per(i,1,num){

printf("%d%c",clique[i],i == num ? '\n':' ');

}

}

return 0;

}

64，动态规划-计数DP-666RPG

const int ma = 2e5;

const int mod = 1e8 + 7;

int n = 0,m = 0;

LL a[300+10];

LL dp[2][2\*ma+1000];

/\*

dp[i][j]，前面的i只有0,1两种取值，使用了滚动数组，保存了当前状态，和上一层状态

j表示当前的分数。

这里要处负数的情况，将原来的区间[-ma,ma]变成[0,2\*ma],其中如果把正数映射到[0,ma]那么j-a[i]，

可能会出现负数，所以把原来的正数映射到[ma,2\*ma],这样就处理了负数的情况

\*/

int main(){

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

per(i,1,n){

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

}

memset(dp,0,sizeof(dp));

dp[0][ma] = 1;

per(i,1,n){

int num = i&1;

memset(dp[num],0,sizeof(dp[num]));

per(j,2000,2\*ma){

if(j == ma + 666){

dp[num][j] = 0;

}else{

dp[num][j] = (dp[num^1][2\*ma-j] +dp[num^1][j-a[i]]) % mod;

}

}

}

printf("%lld\n",dp[n&1][ma-666]);

}

return 0;

}