# 2017ccpc秦皇岛

//ac自动机Trie树

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <cctype>

#include <queue>

using namespace std;

const int maxn = 300010;

int N;

struct Node

{

int fail[maxn];

int next[maxn][26];

int sta[maxn];

int root;

int CNT;

char s[100];

char ss[1000010];

int newnode()

{

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

next[CNT][i] = -1;

fail[CNT] = -1;

sta[CNT] = 0;

return CNT++;

}

void init()

{

CNT = 0;

root = newnode();

}

void ac\_insert()

{

int i, j, k;

int now = root;

int L = strlen(s);

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

{

if (next[now][s[i] - 'a'] == -1)

{

next[now][s[i] - 'a'] = newnode();

}

now = next[now][s[i] - 'a'];

}

sta[now]++;

}

void ac\_buildfail\_bfs()

{

int i, j;

int now = root;

queue<int>q;

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

{

if (next[root][i] != -1)

{

fail[next[root][i]] = root;

q.push(next[root][i]);

}

}

while (!q.empty())

{

now = q.front();

q.pop();

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

{

if (next[now][i] != -1)

{

int temp = fail[now];

while (temp != -1)

{

if (next[temp][i] != -1)

{

fail[next[now][i]] = next[temp][i];

break;

}

temp = fail[temp];

}

if (temp == -1)

{

fail[next[now][i]] = root;

}

q.push(next[now][i]);

}

}

}

}

int ac\_query()

{

int i, j;

int res = 0;

int now = root;

int L = strlen(ss);

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

{

while (next[now][ss[i] - 'a'] == -1 && now != root)

now = fail[now];

now = next[now][ss[i] - 'a'];

if (now == -1)

now = root;

int temp = now;

while (temp != root)

{

res += sta[temp];

sta[temp] = 0;

temp = fail[temp];

}

}

return res;

}

};

Node AC;

int main()

{

int i, j, k, u, n, m;

scanf("%d", &N);

AC.init();

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

{

scanf("%s", AC.s);

AC.ac\_insert();

}

AC.ac\_buildfail\_bfs();

scanf("%s", AC.ss);

printf("%d\n", AC.ac\_query());

return 0;

}

//ac自动机Trie图

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <cctype>

#include <queue>

using namespace std;

const int maxn = 300010;

int N;

struct Node

{

int fail[maxn];

int next[maxn][26];

int sta[maxn];

int root;

int CNT;

char s[100];

char ss[1000010];

int newnode()

{

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

next[CNT][i] = -1;

fail[CNT] = -1;

sta[CNT] = 0;

return CNT++;

}

void init()

{

CNT = 0;

root = newnode();

}

void ac\_insert()

{

int i, j, k;

int now = root;

int L = strlen(s);

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

{

if (next[now][s[i] - 'a'] == -1)

{

next[now][s[i] - 'a'] = newnode();

}

now = next[now][s[i] - 'a'];

}

sta[now]++;

}

void ac\_buildfail\_bfs()

{

int i, j;

int now = root;

queue<int>q;

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

{

if (next[root][i] == -1)

{

next[root][i] = root;

}

else

{

fail[next[root][i]] = root;

q.push(next[root][i]);

}

}

while (!q.empty())

{

now = q.front();

q.pop();

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

{

if (next[now][i] != -1)

{

int temp = fail[now];

while (temp != -1)

{

if (next[temp][i] != -1)

{

fail[next[now][i]] = next[temp][i];

break;

}

temp = fail[temp];

}

if (temp == -1)

{

fail[next[now][i]] = root;

}

q.push(next[now][i]);

}

else

{

next[now][i] = next[fail[now]][i];

}

}

}

}

int ac\_query()

{

int i, j;

int res = 0;

int now = root;

int L = strlen(ss);

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

{

now = next[now][ss[i] - 'a'];

int temp = now;

while (temp != -1)

{

res += sta[temp];

sta[temp] = 0;

temp = fail[temp];

}

}

return res;

}

};

Node AC;

//若把MAX直接改为数字，会快50ms

//next数组别忘了改范围，默认26

int main()

{

int i, j, k, u, n, m;

scanf("%d", &N);

AC.init();

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

{

scanf("%s", AC.s);

AC.ac\_insert();

}

AC.ac\_buildfail\_bfs();

scanf("%s", AC.ss);

printf("%d\n", AC.ac\_query());

return 0;

}

//Determinant

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

const double eps = 1.0e-9;

const int maxn = 110;

typedef long long ll;

ll mod;

ll zushu, var;

ll a[maxn\*maxn][maxn\*maxn], x[maxn\*maxn];

ll Determinant()

{

ll i, j, k, u;

ll res = 1;

ll sign = 1;

for (i = 1; i <= zushu; i++)

{

for (j = i + 1; j <= zushu; j++)

{

ll point1 = i;

ll point2 = j;

while (a[point2][i])

{

ll coe = a[point1][i] / a[point2][i];

for (u = i; u <= var; u++)

{

a[point1][u] = ((a[point1][u] - coe \* a[point2][u]) % mod + mod) % mod;

}

swap(point1, point2);

}

if (point1 != i)

{

sign \*= -1;

for (u = i; u <= var; u++)

swap(a[point1][u], a[point2][u]);

}

}

if (!a[i][i])

return 0;

}

for (i = 1; i <= zushu; i++)

res = ((res \* a[i][i]) % mod + mod) % mod;

res = ((res \* sign) % mod + mod) % mod;

return res;

}

//Gauss

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <vector>

#include <cmath>

#include <queue>

using namespace std;

const int maxn = 220;

const double eps = 1.0e-9;

int zushu, var;

double a[maxn][maxn], x[maxn];

int Gauss()

{

int i, j, k, u, col, max\_r;

    for (k = 1, col = 1; k <= zushu && col <= var; k++, col++)

    {

        max\_r = k;

        for (i = k + 1; i <= zushu; i++)

        {

            if (fabs(a[i][col]) > fabs(a[max\_r][col]))

                max\_r = i;

        }

        if (fabs(a[max\_r][col]) < eps)

            return 0;

        if (k != max\_r)

        {

            for (j = col; j <= var; j++)

                swap(a[k][j], a[max\_r][j]);

            swap(x[k], x[max\_r]);

        }

        x[k] /= a[k][col];

        for (j = col + 1; j <= var; j++)

            a[k][j] /= a[k][col];

        a[k][col] = 1;

        for (i = 1; i <= zushu; i++)

        {

            if (i != k)

            {

                x[i] -= x[k] \* a[i][col];

                for (j = col + 1; j <= var; j++)

                    a[i][j] -= a[k][j] \* a[i][col];

                a[i][col] = 0;

            }

        }

    }

    return 1;

}

//gcd:

//朴素欧几里得 非递归

int gcd(int x,int y)

{

int t;

while(y)

{

t=x%y;

x=y;

y=t;

}

return x;

}

最小公倍数 = x \* y / gcd(x, y);

//朴素欧几里得 递归

int gcd(int a,int b)

{

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

}

ll Extend\_Euclid(ll a, ll b, ll&x, ll& y)

{

if (b == 0)

{

x = 1, y = 0;

return a;

}

ll d = Extend\_Euclid(b, a%b, x, y);

ll t = x;

x = y;

y = t - a / b\*y;

return d;

}

//a在模n乘法下的逆元，没有则返回-1

ll inv(ll a, ll n)

{

ll x, y;

ll t = Extend\_Euclid(a, n, x, y);

if (t != 1)

return -1;

return (x%n + n) % n;

}

//费马定理

//a在模n乘法下的逆元

long long MUL(long long a, long long b, long long mod)

{

long long res = 1;

while (b)

{

if (b & 1)

{

res = (res \* a) % mod;

}

a = (a \* a) % mod;

b >>= 1;

}

return res;

}

long long inv(long long a, long long n)

{

return MUL(a, n-2, n);

}

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <vector>

using namespace std;

typedef long long ll;

const int INF = 2147483640;

const int maxn = 10000000;

bool check[maxn + 10];

int prime[maxn + 10];

int phi[maxn + 10];

int tot;

void phi\_and\_prime\_table(int N)

{

memset(check, false, sizeof(check));

phi[1] = 1;

tot = 0;

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

{

if(!check[i])

{

prime[tot++] = i;

phi[i] = i - 1;

}

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

{

if(i \* prime[j] > N)

break;

check[i \* prime[j]] = true;

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

{

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

break;

}

else

{

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

}

}

}

}

//质数个数 < tot

//欧拉定理 a在模n乘法下的逆元，n不用是质数

long long inv(long long a, long long n)

{

return MUL(a, (long long)phi[n]-1, n);

}

//逆元表1~n范围模p(奇质数)乘法下的逆元 o(n)

long long inv[maxn];

void inverse(long long n, long long p)

{

inv[1] = 1;

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

{

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

}

}

//扩展欧几里得

ll exGcd(ll a, ll b, ll &x, ll &y)

{

ll res, temp;

if(b==0)

{

x=1;

y=0;

return a;

}

res = exGcd(b, a%b, x, y);

temp = x;

x = y;

y = temp-a/b\*y;

return res;

}

/\*

int gcd(int a, int b, int &x, int &y)

{

if (b==0)

{

x=1；

y=0;

return a;

}

int q=gcd(b, a%b, y, x);

y -= a/b\*x;

return q;

}

\*/

//A[] 是余数， M[]是除数，MM是总乘数，Mi是去掉M[i]的乘数， x, y是扩展欧几里得的系数，ans是结果；

//中国剩余定理 互质模板；

#include<iostream>

#include<cstdio>

#include<cstring>

typedef long long ll;

const int maxn = 20;

using namespace std;

ll A[maxn], M[maxn];

void extend\_Euclid(ll a, ll b, ll &x, ll &y)

{

if (b == 0)

{

x = 1;

y = 0;

return;

}

extend\_Euclid(b, a % b, x, y);

ll tmp = x;

x = y;

y = tmp - (a / b) \* y;

}

ll CRT(ll n)

{

ll ans = 0;

ll MM = 1;

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

MM \*= M[i];

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

{

ll x, y;

ll Mi = MM / M[i];

extend\_Euclid(Mi, M[i], x, y);

ans = (ans + Mi \* x \* A[i]) % MM;

}

if (ans < 0) ans += MM;

return ans;

}

//A[] 是余数， M[]是除数，S是通解的差， x, y是扩展欧几里得的系数，ans是结果；

//中国剩余定理 非互质模板；

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

typedef long long int ll;

const int maxn = 15;

ll S;

ll A[maxn], M[maxn];

ll gcd(ll x, ll y)

{

ll t;

while (y)

{

t = x%y;

x = y;

y = t;

}

return x;

}

ll Extend\_Euclid(ll a, ll b, ll&x, ll& y)

{

if (b == 0)

{

x = 1, y = 0;

return a;

}

ll d = Extend\_Euclid(b, a%b, x, y);

ll t = x;

x = y;

y = t - a / b\*y;

return d;

}

//a在模n乘法下的逆元，没有则返回-1

ll inv(ll a, ll n)

{

ll x, y;

ll t = Extend\_Euclid(a, n, x, y);

if (t != 1)

return -1;

return (x%n + n) % n;

}

//将两个方程合并为一个

bool merge(ll a1, ll m1, ll a2, ll m2, ll& a3, ll& m3)

{

ll d = gcd(m1, m2);

ll c = a2 - a1;

if (c%d)

return false;

c = (c%m2 + m2) % m2;

c /= d;

m1 /= d;

m2 /= d;

c \*= inv(m1, m2);

c %= m2;

c \*= m1\*d;

c += a1;

m3 = m1\*m2\*d;

a3 = (c%m3 + m3) % m3;

return true;

}

//求模线性方程组x=ai(mod ni),ni可以不互质

//最小结果（特解）是(a1%n1 + n1) % n1，通解是(a1%n1 + n1) % n1 + K\*S，K是倍数

ll China\_Reminder2(ll len)

{

ll a1 = A[1], m1 = M[1];

ll a2, m2;

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

{

ll aa, mm;

a2 = A[i], m2 = M[i];

if (!merge(a1, m1, a2, m2, aa, mm))

return -1;

a1 = aa;

m1 = mm;

}

S = m1; //S 或者 n1 就是 通解的差

return (a1 % m1 + m1) % m1;

}

int main()

{

}

//Grundy函数

#include <cstdio>

#include <algorithm>

#include <cstring>

#include <cmath>

#include <set>

using namespace std;

const int maxn = 10000 + 10;

int grundy[maxn]; //记录sg值 初始化-1

int step[110]; //step记录的是可以取的数目

int N;

int getgrundy(int x)

{

if (grundy[x] != -1)

return grundy[x];

int i, j;

int visit[110];

memset(visit, 0, sizeof(visit));

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

{

if (step[i] <= x)

{

visit[getgrundy(x - step[i])] = 1;

}

// else

// break;

}

int res = 0;

while (visit[res])

res++;

grundy[x] = res;

return res;

}

int sa[maxn]; //每堆的个数

int step[maxn];

int grundy[maxn];

int N; //step数组的元素个数，即可以走的步长

void getgrundy()

{

int i, j;

int visit[maxn];

grundy[0] = 0;

int maxr = \*max\_element(sa+1, sa+1+cnt); //这个也可以在读入后使用写为 int maxr = \*max\_element(sa+1, sa+1+cnt);

for (i = 1; i <= maxr; i++)

{

memset(visit, 0, sizeof(visit));

for (j = 1; j <= N; j++)

{

if(step[j] <= i)

visit[grundy[i - step[j]]] = 1;

}

int res = 0;

while (visit[res])

res++;

grundy[i] = res;

}

}

//KMP

#include <cstdio>

#include <cstring>

#include <cctype>

using namespace std;

const int maxn = 10000;

char former[maxn], latter[maxn];

int Next[maxn];

void GETnext() //有优化

{

int i, j, k;

int L = strlen(former);

Next[0] = -1;

i = 0;

j = -1;

while (i < L && j < L)

{

if (j == -1 || former[i] == former[j])

{

i++;

j++;

//优化

if (former[i] == former[j])

Next[i] = Next[j];

else

Next[i] = j;

//如果不要优化，这样：

//Next[i] = j;

}

else

j = Next[j];

}

}

int KMP() //返回目标串在主串中出现的次数

{

int i, j;

GETnext();

int L1 = strlen(former);

int L2 = strlen(latter);

i = 0;

j = 0;

int res = 0;

while(i < L2)

{

while(-1 != j && latter[i] != former[j])

j = Next[j];

i++;

j++;

if(j >= L1)

{

res++;

j = Next[j];

}

}

return res;

}

//扩展KMP

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cctype>

using namespace std;

const int maxn = 100000 + 1000;

char former[maxn], latter[maxn];

int exnext[maxn], extend[maxn], Next[maxn];

//Next[i]是former的最长相同前缀，i后缀长度

//exnext[i]是former最长的前缀，和，i后缀的前缀的相同长度

//extend[i]是最长的former前缀，和，latter i后缀的前缀的相同长度

void GETnext()

{

int i, j, k;

int L = strlen(former);

Next[0] = -1;

i = 0;

j = -1;

while (i < L && j < L)

{

if (j == -1 || former[i] == former[j])

{

i++;

j++;

if (former[i] == former[j])

Next[i] = Next[j];

else

Next[i] = j;

}

else

j = Next[j];

}

}

void GETexnext()

{

int i, j, k;

int L = strlen(former);

exnext[0] = L;

i = 1;

j = 0;

while (former[i + j] == former[j] && i + j < L)

j++;

exnext[1] = j;

k = 1;

for (i = 2; i < L; i++)

{

int p = k + exnext[k] - 1;

int l = exnext[i - k];

if (i + l < p + 1)

{

exnext[i] = l;

}

else

{

j = max(0, p - i + 1);

while (i + j < L && former[j] == former[i + j])

j++;

exnext[i] = j;

k = i;

}

}

}

void GETextend()

{

int i, j, k;

int L1 = strlen(former);

int L2 = strlen(latter);

GETexnext();

i = 0;

j = 0;

while (j < L1 && j < L2 && former[j] == latter[j])

j++;

extend[0] = j;

k = 0;

for (i = 1; i < L2; i++)

{

int p = extend[k] + k - 1;

int l = exnext[i - k];

if (i + l < p + 1)

{

extend[i] = l;

}

else

{

j = max(0, p - i + 1);

while (j < L1 && i + j < L2 && former[j] == latter[i + j])

j++;

extend[i] = j;

k = i;

}

}

}

//MAX\_FLOW (Dinic)

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <vector>

#include <queue>

using namespace std;

const int maxn = 1010;

const int maxm = 1010;

const int INF = 2147483640;

struct DINIC

{

struct Edge

{

int to;

int cap;

int rev;

Edge() {}

Edge(int a, int b, int c) : to(a), cap(b), rev(c) {}

};

vector<Edge>G[maxm];

int level[maxn];

int iter[maxn];

void clear(int n)

{

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

G[i].clear();

}

void addedge(int u, int v, int cap)

{

G[u].push\_back(Edge(v, cap, G[v].size()));

G[v].push\_back(Edge(u, 0, G[u].size() - 1));

}

void bfs(int s)

{

int i, j;

queue<int>q;

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

level[s] = 0;

q.push(s);

while (!q.empty())

{

int u = q.front();

q.pop();

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

{

Edge &e = G[u][i];

if (e.cap > 0 && level[e.to] == -1)

{

level[e.to] = level[u] + 1;

q.push(e.to);

}

}

}

}

int dfs(int st, int end, int flow)

{

if (st == end)

return flow;

int i, j;

for (i = iter[st]; i < G[st].size(); i++)

{

Edge &e = G[st][i];

if (e.cap > 0 && level[e.to] == level[st] + 1)

{

int d = dfs(e.to, end, min(flow, e.cap));

if (d > 0)

{

e.cap -= d;

G[e.to][e.rev].cap += d;

iter[st] = i;

return d;

}

}

}

return 0;

}

int max\_flow(int st, int end)

{

int flow = 0;

while (1)

{

memset(iter, 0, sizeof(iter));

bfs(st);

if (level[end] == -1)

return flow;

int f;

while ((f = dfs(st, end, INF)) > 0)

flow += f;

}

}

};

DINIC dinic;

//不要忘了先clear()

//MAX\_FLOW (isap)

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <vector>

#include <queue>

using namespace std;

const int INF = 2147483640;

const int maxn = 90005;

const int maxm = 4000005;

struct ISAP

{

struct Edge

{

int to;

int next;

int cap;

int flow;

};

Edge edge[maxm];

int tol;

int head[maxn];

int gap[maxn];

int dep[maxn];

int cur[maxn];

int S[maxn];

int start;

int end;

int N;

int M;

void init()

{

tol = 0;

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

}

void addedge(int u, int v, int w, int rw = 0)

{

edge[tol].to = v;

edge[tol].cap = w;

edge[tol].flow = 0;

edge[tol].next = head[u];

head[u] = tol++;

edge[tol].to = u;

edge[tol].cap = rw;

edge[tol].flow = 0;

edge[tol].next = head[v];

head[v] = tol++;

}

void bfs()

{

int i, j;

queue<int>q;

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

memset(gap, 0, sizeof(gap));

gap[0] = 1;

dep[end] = 0;

q.push(end);

while(!q.empty())

{

int u = q.front();

q.pop();

for(i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if(dep[v] != -1)

continue;

dep[v] = dep[u] + 1;

gap[dep[v]]++;

q.push(v);

}

}

}

int max\_flow()

{

int i, j;

bfs();

memcpy(cur, head, sizeof(head));

int top = 0;

int u = start;

int ans = 0;

while(dep[start] < N)

{

if(u == end)

{

int Min = INF;

int inser;

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

{

if(Min > edge[S[i]].cap - edge[S[i]].flow)

{

Min = edge[S[i]].cap - edge[S[i]].flow;

inser = i;

}

}

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

{

edge[S[i]].flow += Min;

edge[S[i] ^ 1].flow -= Min;

}

ans += Min;

top = inser;

u = edge[S[top] ^ 1].to;

continue;

}

bool flag = false;

int v;

for(i = cur[u]; i != -1; i = edge[i].next)

{

v = edge[i].to;

if(edge[i].cap - edge[i].flow && dep[v] == dep[u] - 1)

{

flag = true;

cur[u] = i;

break;

}

}

if(flag)

{

S[top++] = cur[u];

u = v;

continue;

}

int Min = N;

for(i = head[u]; i != -1; i = edge[i].next)

{

if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)

{

Min = dep[edge[i].to];

cur[u] = i;

}

}

gap[dep[u]]--;

if(!gap[dep[u]])

return ans;

dep[u] = Min + 1;

gap[dep[u]]++;

if(u != start)

u = edge[S[--top] ^ 1].to;

}

return ans;

}

};

ISAP isap;

//注意要给 isap.start isap.end isap.N(包括源点和汇点) 赋值！和 初始化isap.init()

//双向边直接addedge两次就行了

//maxm是边数的2倍！ 数组开小了会TLE

//一些和数有关的杂七杂八

void init(\_\_int64 n)//求一个数的质因子

{

\_\_int64 i;

num=0;

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

{

if(n%i==0)

{

a[num++]=i;

while(n%i==0)

n=n/i;

}

}

if(n>1)//这里要记得

a[num++]=n;

}

//筛选法求出所有素因子

int prime[100005][20];

int cnt[100005]={0};

void Init(){

for(int i=2;i<=100000;i++){

if(cnt[i]) continue;

prime[i][0]=i;

cnt[i]=1;

for(int j=2;j\*i<=100000;j++)

prime[i\*j][cnt[i\*j]++]=i;

}

}

//二分匹配Hopcroft-Carp

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <vector>

#include <map>

#include <queue>

using namespace std;

const int INF = 2147483640;

typedef long long ll;

const int maxn1 = 3000;

const int maxn2 = 3000;

struct HC

{

vector<int>G[maxn1];

int uN;

int Mx[maxn1], My[maxn2];

int dx[maxn1], dy[maxn2];

int dist;

bool visit[maxn2];

void init()

{

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

G[i].clear();

}

void addedge(int u, int v)

{

G[u].push\_back(v);

}

bool SearchP()

{

queue<int>Q;

dist = INF;

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

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

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

{

if(Mx[i] == -1)

{

Q.push(i);

dx[i] = 0;

}

}

while(!Q.empty())

{

int u = Q.front();

Q.pop();

if(dx[u] > dist)

break;

int sz = G[u].size();

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

{

int v = G[u][i];

if(dy[v] == -1)

{

dy[v] = dx[u] + 1;

if(My[v] == -1)

dist = dy[v];

else

{

dx[My[v]] = dy[v] + 1;

Q.push(My[v]);

}

}

}

}

return dist != INF;

}

bool dfs(int u)

{

int sz = G[u].size();

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

{

int v = G[u][i];

if(!visit[v] && dy[v] == dx[u] + 1)

{

visit[v] = true;

if(My[v] != -1 && dy[v] == dist)

continue;

if(My[v] == -1 || dfs(My[v]))

{

My[v] = u;

Mx[u] = v;

return true;

}

}

}

return false;

}

int MaxMatch()

{

int res = 0;

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

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

while(SearchP())

{

memset(visit, false, sizeof(visit));

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

{

if(Mx[i] == -1 && dfs(i))

res++;

}

}

return res;

}

};

HC hop;

//要先给uN赋值

//初始化hop.init()

//二分匹配匈牙利（矩阵）

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <vector>

#include <map>

#include <queue>

using namespace std;

const int INF = 2147483640;

typedef long long ll;

const int maxn = 510;

struct HUNGARY

{

int uN;

int vN;

int g[maxn][maxn]; //邻接矩阵

int link[maxn];

bool visit[maxn];

void init()

{

memset(g, 0, sizeof(g));

}

bool dfs(int u)

{

for(int v = 1; v <= vN; v++)

{

if(g[u][v] && !visit[v])

{

visit[v] = true;

if(link[v] == -1 || dfs(link[v]))

{

link[v] = u;

return true;

}

}

}

return false;

}

int hungary()

{

int res = 0;

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

for(int u = 1; u <= uN; u++)

{

memset(visit, false, sizeof(visit));

if(dfs(u))

res++;

}

return res;

}

};

HUNGARY hungary;

//初始化hungary.init()

//要给 uN和vN 赋值才能算，uN表示左点，vN表示右点

//标号都是从1开始，1~uN， 1~vN

//二分匹配匈牙利（链表）

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <vector>

#include <map>

#include <queue>

using namespace std;

const int INF = 2147483640;

typedef long long ll;

const int maxn = 5010;

const int maxm = 50010;

struct HUNGARY

{

struct Edge

{

int to;

int next;

};

Edge edge[maxm];

int head[maxn];

int tol;

int link[maxn];

bool visit[maxn];

int uN;

void init()

{

tol = 0;

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

}

void addedge(int u, int v)

{

edge[tol].to = v;

edge[tol].next = head[u];

head[u] = tol++;

}

bool dfs(int u)

{

for(int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if(!visit[v])

{

visit[v] = true;

if(link[v] == -1 || dfs(link[v]))

{

link[v] = u;

return true;

}

}

}

return false;

}

int hungary()

{

int res = 0;

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

for(int u = 1; u <= uN; u++)

{

memset(visit, false, sizeof(visit));

if(dfs(u))

res++;

}

return res;

}

};

HUNGARY hungary;

//初始化 hungary.init()

//要给 uN 赋值才能算

//标号都是从1开始，1~uN

//二分图多重匹配(矩阵)

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <vector>

#include <map>

#include <queue>

using namespace std;

const int INF = 2147483640;

typedef long long ll;

const int maxn = 510; //左边点数

const int maxm = 1010; //右边点数

struct Max\_HUNGARY

{

int uN, vN;

int g[maxn][maxm];

int link[maxm][maxn]; //link[i][0]是右边点i已连接的左边点个数

bool visit[maxm];

int num[maxm];

void init()

{

memset(g, 0, sizeof(g));

}

bool dfs(int u)

{

for(int v = 1; v <= vN; v++)

{

if(g[u][v] && !visit[v])

{

visit[v] = true;

if(link[v][0] < num[v])

{

link[v][++link[v][0]] = u;

return true;

}

for(int i = 1; i <= link[v][0]; i++)

{

if(dfs(link[v][i]))

{

link[v][i] = u;

return true;

}

}

}

}

return false;

}

int Max\_hungary()

{

int res = 0;

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

link[i][0] = 0;

for(int u = 1; u <= uN; u++)

{

memset(visit, false, sizeof(visit));

if(dfs(u))

res++;

}

return res;

}

};

Max\_HUNGARY hungary;

//初始化hungary.init()

//要先给uN，vN赋值才能算

//二分最大权匹配KM

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <queue>

#include <vector>

using namespace std;

typedef long long ll;

const int INF = 2147483640;

const int maxn1 = 310;

const int maxn2 = 310;

struct KM

{

int nx, ny; //两边的点数

int g[maxn1][maxn2]; //二分图关系

int link[maxn2];

int lx[maxn1], ly[maxn2]; //对另一边点的最大要求值

int visitx[maxn1], visity[maxn2]; //在一次增广中是否被增广到

int slack[maxn2]; //右点还差多少就能满足左点的要求

void init()

{

memset(g, 0, sizeof(g));

}

bool dfs(int x)

{

visitx[x] = true;

for(int y = 1; y <= ny; y++)

{

if(visity[y])

continue;

int tmp = lx[x] + ly[y] - g[x][y];

if(tmp == 0)

{

visity[y] = true;

if(link[y] == -1 || dfs(link[y]))

{

link[y] = x;

return true;

}

}

else if(slack[y] > tmp)

{

slack[y] = tmp;

}

}

return false;

}

int km()

{

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

memset(ly, 0, sizeof(ly));

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

{

lx[i] = -INF;

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

{

if(g[i][j] > lx[i])

{

lx[i] = g[i][j];

}

}

}

for(int x = 1; x <= nx; x++)

{

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

slack[i] = INF;

while(true)

{

memset(visitx, false, sizeof(visitx));

memset(visity, false, sizeof(visity));

if(dfs(x))

break;

int d = INF;

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

{

if(!visity[i] && d > slack[i])

{

d = slack[i];

}

}

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

{

if(visitx[i])

lx[i] -= d;

}

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

{

if(visity[i])

ly[i] += d;

else

slack[i] -= d;

}

}

}

int res = 0;

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

{

if(link[i] != -1)

res += g[link[i]][i];

}

return res;

}

};

KM km;

//要先给nx，ny赋值才能算

//如果g图不是每个点都有权值，先全部赋值为0

//如果算最小权值，把权值都取负数，结果再取负数

//反素数

//反素数的定义：对于任何正整数n，其约数个数记为f[n]，例如f[6] = 4，如果某个正整数n满足：对任意的正整

// 数i(0 < i < n)，都有f[i] < f[n]，那么称n为反素数。

//在ACM竞赛中，最常见的问题如下：

//（1）给定一个数n，求一个最小的正整数x，使得x的约数个数为n (n > 0)

//（2）求出1~n中约数个数最多的这个数

//1、

#include <iostream>

#include <cstring>

#include <cstdio>

using namespace std;

typedef long long ll;

const ll INF = 9223372036854775807;

int p[16] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53};

ll RES;

void dfs(ll dept, ll tmp, ll num, ll limit)

{

if(num > limit) return;

if(num == limit && RES > tmp)

RES = tmp;

for(int i = 1; i <= 63; i++)

{

if(RES / p[dept] < tmp)

break;

dfs(dept + 1, tmp \*= p[dept], num\*(i+1), limit);

}

}

int main()

{

ll n;

while(cin>>n)

{

RES = INF;

dfs(0,1,1,n);

cout<<RES<<endl;

}

return 0;

}

//多项式除法

#include<iostream>

#include<sstream>

#include<fstream>

#include<vector>

#include<list>

#include<deque>

#include<queue>

#include<stack>

#include<map>

#include<set>

#include<bitset>

#include<algorithm>

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<cctype>

#include<cmath>

#include<ctime>

#include<iomanip>

using namespace std;

const double eps(1e-8);

typedef long long lint;

#define maxn 1110

vector <int> v[maxn];

struct polynomial

{

int coe[maxn];

int len;

polynomial()

{

memset(coe, 0, sizeof(coe));

len = 1;

}

void output()

{

printf("(");

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

{

if (coe[i] == 0) continue;

if (i == 0)

{

if (coe[i] > 0) printf("+");

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

continue;

}

if (coe[i] > 0 && i != len - 1) printf("+");

if (coe[i] == -1) printf("-");

if (abs(coe[i]) != 1) printf("%d", coe[i]);

if (i > 1)

printf("x^%d", i);

else printf("x");

}

printf(")");

return;

}

polynomial operator / (const polynomial pol);

};

polynomial polynomial :: operator / (const polynomial pol)

{

polynomial ret;

ret.len = len - pol.len + 1;

for (int i = len - 1; i >= pol.len - 1; i--)

{

int tim = coe[i] / pol.coe[pol.len - 1];

if (tim != 0)

{

for (int j = 0; j < pol.len; j++)

coe[i - j] -= tim\*pol.coe[pol.len - 1 - j];

ret.coe[i - pol.len + 1] = tim;

}

}

return ret;

}

polynomial p[maxn];

bool cmp(int i, int j)//return p[i] < p[j]

{

if (p[i].len != p[j].len) return p[i].len < p[j].len;

for (int k = p[i].len - 1; k >= 0; k--)

if (p[i].coe[k] != p[j].coe[k])

{

if (abs(p[i].coe[k]) == abs(p[j].coe[k]))

return p[i].coe[k] < 0;

else return abs(p[i].coe[k]) < abs(p[j].coe[k]);

}

return false;

}

//多项式出发算一个多项式的因子式

int main()

{

p[1].coe[0] = -1;

p[1].coe[1] = 1;

p[1].len = 2;//p[i] = (x - 1)

for (int i = 2; i <= 1100; i++)//计算p[2~1100]的特殊多项式

{

p[i].coe[0] = -1;

p[i].coe[i] = 1;

p[i].len = i + 1;

p[i] = p[i] / p[1];

v[i].push\_back(1);

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

{

if (i % j == 0)

{

p[i] = p[i] / p[j];

v[i].push\_back(j);

if (j\*j != i) p[i] = p[i] / p[i / j], v[i].push\_back(i / j);

}

}

v[i].push\_back(i);

}

int n;

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

{

if (n == 1)

{

printf("x-1\n");

continue;

}

sort(v[n].begin(), v[n].end(), cmp);

for (int i = 0, sz = v[n].size(); i < sz; i++)

p[v[n][i]].output();

printf("\n");

}

return 0;

}

//字典树模板

#include <cstdio>

#include <cstring>

#include <cctype>

#include <algorithm>

#include <cmath>

#include <vector>

using namespace std;

const int maxn = 100000;

struct Trie

{

int next[maxn][26];

int sta[maxn];

int root;

int CNT;

char s[100];

int newnode()

{

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

{

next[CNT][i] = -1;

}

sta[CNT] = 0;

return CNT++;

}

void init()

{

CNT = 0;

root = newnode();

}

void insert()

{

int i, j, k;

int now = root;

int L = strlen(s);

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

{

if(next[now][s[i] - 'a'] == -1)

{

next[now][s[i] - 'a'] = newnode();

}

now = next[now][s[i] - 'a'];

}

sta[now] = 1;

}

int search()

{

int i, j, k;

int now = root;

int L = strlen(s);

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

{

now = next[now][s[i] - 'a'];

if(now == -1)

break;

}

if(now != -1 && sta[now])

return 1;

else

return 0;

}

};

Trie trie;

//别忘了初始化trie.init()

//插入和查询都用s[]数组

//小室算法

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

#include <cmath>

#include <cctype>

#include <vector>

#include <queue>

#include <set>

#include <string>

#include <iostream>

using namespace std;

typedef long long ll;

const double eps = 1e-8;

const int INF = 100000000;

const ll MOD = 1e18;

const int maxn = 2000000 + 100;

const int M = 1000000 + 10;

int res[maxn];

int top;

struct Node

{

int id;

int val;

Node(int a, int b) : id(a), val(b) {}

bool operator < (const Node &p)

{

return id > p.id;

}

};

vector<Node>G[maxn];

int MAX;

void dfs(int n, int pos)

{

if (top > M)

return;

res[top++] = n;

int i, j;

for (i = 0; i < G[n].size(); i++)

{

if (G[n][i].id < pos)

break;

dfs(G[n][i].val, G[n][i].id);

}

}

int main()

{

int i, j, k, u, n, m, N, L, R, a, b;

scanf("%d %d %d", &N, &L, &R);

{

MAX = 0;

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

{

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

if (MAX < b)

MAX = b;

G[a].push\_back(Node(i, b));

}

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

{

if (G[i].size())

sort(G[i].begin(), G[i].end());

}

top = 0;

int count;

dfs(0, 0);

//for (i = 0; i < top; i++)

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

for (i = L % top, count = 1; count <= R - L; count++)

{

if (i == top)

{

i = 0;

}

if (count == 1)

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

else

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

i++;

}

putchar('\n');

}

return 0;

}

//快速幂快速乘

long long MUL(long long a, long long b, long long MOD)

{

long long res = 1;

while (b)

{

if (b & 1)

{

res = (res \* a) % MOD;

}

a = (a \* a) % MOD;

b >>= 1;

}

return res;

}

long long MUL(long long a, long long b, long long MOD)

{

long long res = 0;

while (b)

{

if (b & 1)

{

res = (res + a) % MOD;

}

a = (a + a) % MOD;

b >>= 1;

}

return res;

}

//快速筛素数

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

typedef long long ll;

const int maxn = 10000;

int prime[maxn + 1];

void getprime()

{

int i, j;

memset(prime, 0, sizeof(prime));

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

{

if (!prime[i])

prime[++prime[0]] = i;

for (j = 1; j <= prime[0] && i\*prime[j] <= maxn; j++)

{

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

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

break;

}

}

}

//找最少的数覆盖1~n区间

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <queue>

#include <vector>

#include <cctype>

using namespace std;

const int maxn = 1e5 + 10;

const int INF = 1e9;

typedef long long ll;

ll N, M;

ll ar[maxn];

ll stack[maxn];

int main()

{

ll i, n, m;

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

{

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

{

scanf("%lld %lld", &N, &M); //N为所给的数的个数， M为所要覆盖的区间最大值

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

{

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

}

sort(ar + 1, ar + 1 + N);

ll cur = 0;

ll cnt = 0;

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

{

while (cur < ar[i] - 1)

{

stack[++cnt] = cur + 1;

cur += cur + 1;

if (cur >= M)

break;

}

cur += ar[i];

if (cur >= M)

break;

}

while (cur < M)

{

cur += cur + 1;

cnt++;

}

printf("%lld\n", cnt); //cnt为覆盖区间需要的最小数字个数，stack数组存的是需要的数字

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

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

putchar('\n');

}

}

return 0;

}

//按顺序求最长上升区间和（离散化+线段树）

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

#include <cmath>

#include <cctype>

#include <vector>

#include <queue>

#include <set>

#include <string>

#include <iostream>

using namespace std;

typedef long long ll;

const double eps = 1e-8;

const int INF = 100000000;

const ll MOD = 1e18;

const int maxn = 100000 + 100;

struct Node

{

int l;

int r;

int mid;

int max;

int id;

};

Node ar[maxn];

Node G[maxn \* 4];

void pushup(int root)

{

G[root].max = max(G[root \* 2].max, G[root \* 2 + 1].max);

}

void build(int root, int begin, int end)

{

G[root].l = begin;

G[root].r = end;

G[root].mid = (begin + end) / 2;

G[root].max = 0;

if (begin == end)

return;

build(root \* 2, G[root].l, G[root].mid);

build(root \* 2 + 1, G[root].mid + 1, G[root].r);

}

void update(int root, int index, int val)

{

if (G[root].l == G[root].r && G[root].l == index)

{

G[root].max = val;

return;

}

if (index <= G[root].mid)

update(root \* 2, index, val);

else

update(root \* 2 + 1, index, val);

pushup(root);

}

int query(int root, int b, int e)

{

if (b <= G[root].l && G[root].r <= e)

{

return G[root].max;

}

int res = 0;

if (b <= G[root].mid)

res = max(res, query(root \* 2, b, e));

if (e > G[root].mid)

res = max(res, query(root \* 2 + 1, b, e));

return res;

}

bool comp1(const Node& a, const Node& b)

{

return a.l < b.l;

}

bool comp2(const Node& a, const Node& b)

{

return a.id < b.id;

}

int sa[maxn];

int res[maxn]; //res[i]是以i位置为结尾的最长上升区间和

int main()

{

int i, j, k, u, n, m, N;

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

{

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

{

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

ar[i].id = i;

}

sort(ar + 1, ar + 1 + N, comp1);

int cnt = 0;

ar[0].r = -INF;

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

{

if (ar[i].r != ar[i - 1].r)

{

sa[ar[i].id] = i + 1;

}

else //去重 此题输入区间不会有重叠 但是会重复

{

sa[ar[i].id] = sa[ar[i - 1].id];

}

}

sort(ar + 1, ar + 1 + N, comp2);

build(1, 1, N+1);

int Max = -INF;

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

{

res[i] = ar[i].r - ar[i].l + 1 + query(1, 1, sa[i] - 1);

update(1, sa[i], res[i]);

if (Max < res[i])

Max = res[i];

}

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

}

return 0;

}

//数组双向链表+前向星跳转

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

#include <cmath>

#include <cctype>

#include <vector>

#include <queue>

#include <set>

#include <string>

#include <iostream>

using namespace std;

typedef long long ll;

const double eps = 1e-8;

const int INF = 100000000;

const ll MOD = 1e18;

const int maxn = 2000000 + 100;

int ar[maxn];

int l[maxn], r[maxn], val[maxn], Next[maxn], head[maxn];

int cnt;

int main()

{

int i, j, k, u, n, m, N, L, R, a, b;

while (scanf("%d %d %d", &N, &L, &R) != EOF)

{

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

cnt = 1;

l[1] = 0;

r[1] = 0;

val[1] = 0;

head[0] = 1;

Next[0] = -1;

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

{

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

for (j = head[a]; j != -1; j = Next[j])

{

l[++cnt] = j;

r[cnt] = r[j];

l[r[j]] = cnt;

r[j] = cnt;

val[cnt] = b;

if (head[b] == -1)

{

Next[cnt] = -1;

head[b] = cnt;

}

else

{

Next[cnt] = head[b];

head[b] = cnt;

}

}

}

int tot = -1;

for (i = 1; r[i] != 0; i = r[i])

{

ar[++tot] = val[i];

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

}

ar[++tot] = val[i];

int l = L;

l %= (tot+1);

int count;

for (i = l, count = 1; count <= R - L; count++)

{

if (i == tot + 1)

{

i = 0;

}

if (count == 1)

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

else

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

i++;

}

putchar('\n');

}

return 0;

}

//最小生成树kruskal

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

#include <vector>

#include <set>

#include <deque>

using namespace std;

typedef long long ll;

const double eps = 1e-7;

const int INF = 2147483640;

const int maxn = 105;

const int maxm = 10000;

struct KRUSKAL

{

struct Edge

{

int u;

int v;

int w;

};

Edge edge[maxm];

int tol;

int F[maxn];

void init()

{

tol = 0;

}

struct comp

{

bool operator () (Edge& a, Edge& b)

{

return a.w < b.w;

}

};

void addedge(int a, int b, int c)

{

edge[tol].u = a;

edge[tol].v = b;

edge[tol++].w = c;

}

int find(int x)

{

if (F[x] == -1)

return x;

else

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

}

int kruskal(int n)

{

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

sort(edge, edge + tol, comp());

int cnt = 0;

int ans = 0;

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

{

int u = edge[i].u;

int v = edge[i].v;

int w = edge[i].w;

int t1 = find(u);

int t2 = find(v);

if (t1 != t2)

{

ans += w;

F[t1] = t2;

cnt++;

}

if (cnt == n - 1)

break;

}

if (cnt < n - 1)

return -1;

else

return ans;

}

};

KRUSKAL kruskal;

//kruskal.krusal(n), n为点数

//kruskal.init()

//最小生成树prime

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

#include <vector>

#include <set>

#include <deque>

using namespace std;

typedef long long ll;

const double eps = 1e-7;

const int INF = 2147483640;

const int maxn = 205;

struct PRIME

{

int N;

int cost[maxn][maxn];

int lowc[maxn];

bool visit[maxn];

int prime()

{

int ans = 0;

memset(visit, false, sizeof(visit));

visit[1] = true;

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

lowc[i] = cost[1][i];

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

{

int minc = INF;

int p = -1;

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

{

if(!visit[j] && minc > lowc[j])

{

minc = lowc[j];

p = j;

}

}

if(minc == INF)

return -1;

ans += minc;

visit[p] = true;

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

{

if(!visit[j] && lowc[j] > cost[p][j])

lowc[j] = cost[p][j];

}

}

return ans;

}

};

PRIME prime;

//prime.N

//1~N;

//prime.cost[][]

//最小费用最大流spfa

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <cctype>

#include <queue>

#include <vector>

using namespace std;

typedef long long ll;

const int INF = 2147483640;

const int maxn = 10000;

const int maxm = 200000;

struct COST\_SPFA

{

struct Edge

{

int to;

int next;

int cap; //该边的最大流量

int flow; //该边的流量

int cost; //单位流量，单价为1m/元的路程花费（即距离）

};

Edge edge[maxm];

int head[maxn];

int pre[maxn];

int dist[maxn];

bool vis[maxn];

int tol;

int start;

int end;

int N; //节点总个数，节点编号0~N-1

void init()

{

tol = 0;

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

}

void addedge(int u, int v, int cap, int cost)

{

edge[tol].to = v;

edge[tol].cap = cap;

edge[tol].cost = cost;

edge[tol].flow = 0;

edge[tol].next = head[u];

head[u] = tol++;

edge[tol].to = u;

edge[tol].cap = 0;

edge[tol].cost = -cost;

edge[tol].flow = 0;

edge[tol].next = head[v];

head[v] = tol++;

}

bool spfa()

{

queue<int>q;

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

{

dist[i] = INF;

vis[i] = false;

pre[i] = -1;

}

dist[start] = 0;

vis[start] = true;

q.push(start);

while(!q.empty())

{

int u = q.front();

q.pop();

vis[u] = false;

for(int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if(edge[i].cap > edge[i].flow && dist[v] > dist[u] + edge[i].cost)

{

dist[v] = dist[u] + edge[i].cost;

pre[v] = i;

if(!vis[v])

{

vis[v] = true;

q.push(v);

}

}

}

}

if(pre[end] == -1)

return false;

else

return true;

}

//返回的是最大流，&cost存的是最小费用

int minCostMaxFlow(int &cost)

{

int flow = 0;

cost = 0;

while(spfa())

{

int Min = INF;

for(int i = pre[end]; i != -1; i = pre[edge[i^1].to])

{

if(Min > edge[i].cap - edge[i].flow)

Min = edge[i].cap - edge[i].flow;

}

for(int i = pre[end]; i != -1; i = pre[edge[i^1].to])

{

edge[i].flow += Min;

edge[i^1].flow -= Min;

cost += edge[i].cost \* Min;

}

flow += Min;

}

return flow;

}

};

COST\_SPFA zkw;

//别忘了改maxn，maxm

//别忘了给 zkw.start, zkw.end, zkw.N 赋值 和 zkw.init() 初始化

//zkw.N为节点总个数，编号0~N-1

//cost为单位流量，单价为1m/元的路程花费（即为距离）

//要给minCostMaxFlow(int &cost) 那个cost的参数，带出来就是最小花费

//最小费用最大流ZKWstruct

#include <iostream>

#include <cstdio>

#include <cstring>

#include <queue>

#include <cmath>

#include <algorithm>

#include <deque>

using namespace std;

typedef long long ll;

const int INF = 2147483640;

const int maxn = 10000;

const int maxm = 200000;

struct ZKW

{

struct Edge

{

int to;

int next;

int cap; //该边的最大流量

int flow; //该边的流量

int cost; //单位流量，单价为1m/元的路程花费（即距离）

};

Edge edge[maxm];

int tol;

int N; //节点总个数，节点编号0~N-1

int maxflow;

int mincost;

int start, end;

int head[maxn];

bool visit[maxn];

int dist[maxn];

void init()

{

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

tol = 0;

}

void addedge(int u, int v, int cap, int cost)

{

edge[tol].to = v;

edge[tol].cap = cap;

edge[tol].cost = cost;

edge[tol].flow = 0;

edge[tol].next = head[u];

head[u] = tol++;

edge[tol].to = u;

edge[tol].cap = 0;

edge[tol].cost = -cost;

edge[tol].flow = 0;

edge[tol].next = head[v];

head[v] = tol++;

}

void SPFA()

{

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

dist[i] = INF;

priority\_queue<pair<int, int> > Q;

dist[start] = 0;

Q.push(make\_pair(0, start));

while(!Q.empty())

{

int u = Q.top().second;

int d = -Q.top().first;

Q.pop();

if(dist[u] != d)

continue;

for(int p = head[u]; p != -1; p = edge[p].next)

{

int &v = edge[p].to;

if(edge[p].cap && dist[v] > d + edge[p].cost)

{

dist[v] = d + edge[p].cost;

Q.push(make\_pair(-dist[v], v));

}

}

}

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

dist[i] = dist[end] - dist[i];

}

int add\_flow(int u, int flow)

{

if(u == end)

{

maxflow += flow;

mincost += dist[start] \* flow;

return flow;

}

visit[u] = true;

int now = flow;

for(int p = head[u]; p != -1; p = edge[p].next)

{

int &v = edge[p].to;

if(edge[p].cap && !visit[v] && dist[u] == dist[v] + edge[p].cost)

{

int tmp = add\_flow(v, min(now, edge[p].cap));

edge[p].cap -= tmp;

edge[p ^ 1].cap += tmp;

now -= tmp;

if(!now)

break;

}

}

return flow - now;

}

bool modify\_label()

{

int d = INF;

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

if(visit[u])

for(int p = head[u]; p != -1; p = edge[p].next)

{

int &v = edge[p].to;

if(edge[p].cap && !visit[v])

d = min(d, dist[v] + edge[p].cost - dist[u]);

}

if(d == INF)

return false;

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

if(visit[i])

dist[i] += d;

return true;

}

int minCostMaxFlow()

{

mincost = maxflow = 0;

SPFA();

while(true)

{

while(true)

{

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

visit[i] = 0;

if(!add\_flow(start, INF))

break;

}

if(!modify\_label())

break;

}

return mincost;

}

};

ZKW zkw;

//别忘了改maxn，maxm

//别忘了给 zkw.start, zkw.end, zkw.N 赋值 和 zkw.init() 初始化

//zkw.N为节点总个数，编号0~N-1

//cost为单位流量，单价为1m/元的路程花费（即为距离）

//minCostMaxFlow()返回的是最小费用，maxflow是最大流

#include <cstdio>

#include <queue>

#include <iostream>

#include <vector>

#include <cstring>

#include <cmath>

using namespace std;

const int maxn = 210;

const int maxm = 100010;

const int INF = 0x3f3f3f3f;

struct ZKW

{

int start, end;

int N;

int mincost;

int maxflow;

int tol;

int head[maxn];

int cap[maxm], cost[maxm], to[maxm], next[maxm]; //单位流量，单价为1m/元的路程花费（即距离）

int dist[maxn];

int visit[maxn];

void init()

{

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

tol = 0;

mincost = maxflow = 0;

}

void addedge(int u, int v, int capp, int costt)

{

to[tol] = v;

cap[tol] = capp;

cost[tol] = costt;

next[tol] = head[u];

head[u] = tol++;

to[tol] = u;

cap[tol] = 0;

cost[tol] = -costt;

next[tol] = head[v];

head[v] = tol++;

}

void SPFA()

{

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

dist[i] = INF;

priority\_queue<pair<int, int> > Q;

dist[start] = 0;

Q.push(make\_pair(0, start));

while(!Q.empty())

{

int u = Q.top().second;

int d = -Q.top().first;

Q.pop();

if(dist[u] != d)

continue;

for(int p = head[u]; p != -1; p = next[p])

{

int &v = to[p];

if(cap[p] && dist[v] > d + cost[p])

{

dist[v] = d + cost[p];

Q.push(make\_pair(-dist[v], v));

}

}

}

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

dist[i] = dist[end] - dist[i];

}

int add\_flow(int u, int flow)

{

if(u == end)

{

maxflow += flow;

mincost += dist[start] \* flow;

return flow;

}

visit[u] = true;

int now = flow;

for(int p = head[u]; p != -1; p = next[p])

{

int &v = to[p];

if(cap[p] && !visit[v] && dist[u] == dist[v] + cost[p])

{

int tmp = add\_flow(v, min(now, cap[p]));

cap[p] -= tmp;

cap[p ^ 1] += tmp;

now -= tmp;

if(!now)

break;

}

}

return flow - now;

}

bool modify\_label()

{

int d = INF;

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

if(visit[u])

for(int p = head[u]; p != -1; p = next[p])

{

int &v = to[p];

if(cap[p] && !visit[v])

d = min(d, dist[v] + cost[p] - dist[u]);

}

if(d == INF)

return false;

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

if(visit[i])

dist[i] += d;

return true;

}

int minCostMaxFlow()

{

SPFA();

while(true)

{

while(true)

{

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

visit[i] = 0;

if(!add\_flow(start, INF))

break;

}

if(!modify\_label())

break;

}

return mincost;

}

};

ZKW zkw;

//别忘了改maxn，maxm

//别忘了给 zkw.start, zkw.end, zkw.N 赋值 和 zkw.init() 初始化

//zkw.N为节点总个数，编号0~N-1

//cost为单位流量，单价为1m/元的路程花费（即为距离）

//minCostMaxFlow()返回的是最小费用，maxflow是最大流

//最短路dij(n2)

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

#include <vector>

#include <set>

using namespace std;

typedef long long ll;

const double eps = 1e-7;

const int INF = 1000000000;

const int maxn = 205;

struct DIJ

{

int g[maxn][maxn];

int dist[maxn];

int start;

int N;

bool visit[maxn];

int pre[maxn];

void dijkstra()

{

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

{

dist[i] = INF;

visit[i] = false;

pre[i] = -1;

}

dist[start] = 0;

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

{

int k = -1;

int Min = INF;

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

{

if(!visit[j] && dist[j] < Min)

{

Min = dist[j];

k = j;

}

}

if(k == -1)

break;

visit[k] = true;

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

{

if(!visit[j] && dist[j] > dist[k] + g[k][j])

{

dist[j] = dist[k] + g[k][j];

pre[j] = k;

}

}

}

}

};

DIJ dij;

//先给g[][], start, N 赋值

//最短路dij堆优化（ElogE）

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

#include <vector>

#include <set>

using namespace std;

typedef long long ll;

const double eps = 1e-7;

const int INF = 1000000000;

const int maxn = 205;

const int maxm = 1000;

struct P\_DIJ

{

struct Edge

{

int to;

int val;

int next;

};

Edge edge[maxm];

typedef pair<int, int> P;

struct comp

{

bool operator () (const P &a, const P &b)

{

return a.second > b.second;

}

};

int head[maxn];

int dist[maxn];

int tol;

int start;

int N;

void init()

{

tol = 0;

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

}

void addedge(int u, int v, int w)

{

edge[tol].to = v;

edge[tol].val = w;

edge[tol].next = head[u];

head[u] = tol++;

}

void P\_dijkstra()

{

int i, j, k;

priority\_queue<P, vector<P>, comp>Q;

P now, after;

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

dist[i] = INF;

now.first = start;

now.second = 0;

dist[start] = 0;

Q.push(now);

while (!Q.empty())

{

now = Q.top();

Q.pop();

if (dist[now.first] < now.second)

continue;

for (i = head[now.first]; i != -1; i = edge[i].next)

{

after.first = edge[i].to;

after.second = edge[i].val;

if (dist[after.first] > dist[now.first] + after.second)

{

dist[after.first] = dist[now.first] + after.second;

Q.push(after);

}

}

}

}

};

P\_DIJ dij;

//先给dij.start, dij.N赋值

//初始化dij.init()

//最短路spfa（kE）

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

#include <vector>

#include <set>

#include <deque>

using namespace std;

typedef long long ll;

const double eps = 1e-7;

const int INF = 1000000000;

const int maxn = 205;

const int maxm = 1000;

struct P\_SPFA

{

struct Edge

{

int to;

int val;

int next;

};

Edge edge[maxm];

int tol;

int N;

int start;

int head[maxn];

int dist[maxn];

int in[maxn];

int visit[maxn];

void init()

{

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

tol = 0;

}

void addedge(int u, int v, int w)

{

edge[tol].to = v;

edge[tol].val = w;

edge[tol].next = head[u];

head[u] = tol++;

}

bool P\_spfa()

{

int i, j, k;

deque<int>Q;

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

{

visit[i] = false;

in[i] = 0;

dist[i] = INF;

}

visit[start] = true;

dist[start] = 0;

Q.push\_back(start);

while(!Q.empty())

{

int u = Q.front();

Q.pop\_front();

visit[u] = false;

for(i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

int w = edge[i].val;

if(dist[v] > dist[u] + w)

{

dist[v] = dist[u] + w;

if(!visit[v])

{

visit[v] = true;

if(++in[v] > N)

return false;

if(!Q.empty() && dist[v] < dist[Q.front()])

Q.push\_front(v);

else

Q.push\_back(v);

}

}

}

}

}

};

P\_SPFA spfa;

//spfa.start, spfa.N, spfa.init()

//最长上升子序列

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <map>

#include <cctype>

#include <queue>

#include <vector>

using namespace std;

const int maxn = 500000 + 100;

typedef long long ll;

int LIS[maxn]; //记录长度为i的lis的最后一个元素的值

int ar[maxn];

int ld[maxn]; //记录以i结尾的lis的长度

int N;

int erfen(int x, int len) //找第一个 >= x的值

{

int l = 1;

int r = len;

int mid;

while (r >= l)

{

mid = (l + r) / 2;

if (LIS[mid] < x)

l = mid + 1;

else

r = mid - 1;

}

return l;

}

int lis()

{

int i, j, k;

LIS[1] = ar[1];

ld[1] = 1;

int len = 1;

for (i = 2; i <= N; i++)

{

if (ar[i] > LIS[len])

{

LIS[++len] = ar[i];

ld[i] = len;

}

else

{

int t = erfen(ar[i], len);

LIS[t] = ar[i];

ld[i] = t;

}

}

return len;

}

int main()

{

int i, j, k, u, n, m;

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

{

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

{

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

}

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

}

return 0;

}

//n^2的写法

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

ld[i] = 1;

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

{

for(j = 1; j < i; j++)

{

if(ar[i] > ar[j] && ld[i] < ld[j] + 1)

{

ld[i] = ld[j] + 1;

}

}

}

//有向图的强连通分量

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

#include <vector>

#include <set>

#include <deque>

using namespace std;

typedef long long ll;

const double eps = 1e-7;

const int INF = 2147483640;

const int maxn = 105;

const int maxm = 10000;

struct TARJAN

{

struct Edge

{

int to;

int next;

};

Edge edge[maxm];

int tol;

int Index;

int top;

int Sccc;

int head[maxn];

int Low[maxn];

int DFN[maxn];

int Stack[maxn];

bool Instack[maxn];

int Belong[maxn];

int num[maxn];

void init()

{

tol = 0;

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

}

void addedge(int u, int v)

{

edge[tol].to = v;

edge[tol].next = head[u];

head[u] = tol++;

}

void Tarjan(int u) //将一个环缩成一个点, 多个环相连只算一个环

{

int v;

Low[u] = DFN[u] = ++Index;

Stack[++top] = u;

Instack[u] = true;

for(int i = head[u]; i != -1; i = edge[i].next)

{

v = edge[i].to;

if(!DFN[v])

{

Tarjan(v);

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

}

else if(Instack[v] && Low[u] > DFN[v])

{

Low[u] = DFN[v];

}

}

if(Low[u] == DFN[u])

{

Sccc++;

do

{

v = Stack[top--];

Instack[v] = false;

Belong[v] = Sccc;

num[Sccc]++;

}while(v != u);

}

}

void solve(int N)

{

memset(DFN, 0, sizeof(DFN));

memset(Instack, false, sizeof(Instack));

memset(num, 0, sizeof(num));

Index = Sccc = top = 0;

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

{

if(!DFN[i])

Tarjan(i);

}

}

};

TARJAN tarjan;

//tarjan.init()

//tarjan.solve(N);

//N 1~N

//Sccc 1~Sccc;

//solve()变化很多

//树状数组

//单点更新，单点查询

void update(int x, int val)

{

int t = x;

while(t <= N)

{

c[t] += val;

t += t & -t;

}

}

int sum(int x)

{

int t = x;

int res = 0;

while(t >= 1)

{

res += c[t];

t -= t & -t;

}

return res;

}

update(x);

sum(x);

//单点更新，区间查询

void update(int x, int val)

{

int t = x;

while(t <= N)

{

c[t] += val;

t += t & -t;

}

}

int sum(int x)

{

int t = x;

int res = 0;

while(t >= 1)

{

res += c[t];

t -= t & -t;

}

return res;

}

update(x);

sum(y) - sum(x-1);

//区间更新， 单点查询

void update(int x, int val)

{

int t = x;

while(t <= N)

{

c[t] += val;

t += t & -t;

}

}

int sum(int x)

{

int t = x;

int res = 0;

while(t >= 1)

{

res += c[t];

t -= t & -t;

}

return res;

}

update(x, v);

update(y+1, -v);

sum(x);

//次小生成树

#include <cstdio>

#include <cstring>

#include <queue>

#include <algorithm>

#include <cmath>

#include <map>

#include <vector>

#include <set>

#include <deque>

using namespace std;

typedef long long ll;

const double eps = 1e-7;

const int INF = 2147483640;

const int maxn = 205;

struct S\_PRIME

{

int N;

int cost[maxn][maxn];

bool visit[maxn];

int lowc[maxn];

int pre[maxn];

int Max[maxn][maxn];

bool used[maxn][maxn];

int prime()

{

int ans = 0;

memset(visit, false, sizeof(visit));

memset(Max, 0, sizeof(Max));

memset(used, false, sizeof(used));

visit[1] = true;

pre[1] = -1;

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

{

lowc[i] = cost[1][i];

pre[i] = 1;

}

lowc[1] = 0;

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

{

int minc = INF;

int p = -1;

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

{

if(!visit[j] && minc > lowc[j])

{

minc = lowc[j];

p = j;

}

}

if(minc == INF)

return -1;

ans += minc;

visit[p] = true;

used[p][pre[p]] = used[pre[p]][p] = true;

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

{

if(visit[j])

Max[j][p] = Max[p][j] = max(Max[j][pre[p]], lowc[p]);

if(!visit[j] && lowc[j] > cost[p][j])

{

lowc[j] = cost[p][j];

pre[j] = p;

}

}

}

return ans;

}

};

S\_PRIME prime;

//prime.N;

//prime.cost[][]

//Max[i][j]表示MST中i到j的最大边权

//求完后，直接枚举所有不在MST中的边，替换掉最大边权的边，输出最优答案

//求1~n中与n互质的数的4次方

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

#include <cmath>

#include <cctype>

#include <vector>

#include <queue>

#include <set>

#include <string>

#include <iostream>

using namespace std;

typedef long long ll;

typedef unsigned long long ull;

const double eps = 1e-8;

const ll INF = 1e18;

const ll MOD = 1000000007;

const int maxn = 10010;

bool check[maxn + 10];

ll prime[maxn + 10];

ll tot;

void phi\_and\_prime\_table(ll N)

{

memset(check, false, sizeof(check));

tot = 0;

for (ll i = 2; i <= N; i++)

{

if (!check[i])

{

prime[tot++] = i;

}

for (ll j = 0; j < tot; j++)

{

if (i \* prime[j] > N)

break;

check[i \* prime[j]] = true;

}

}

}

//质数个数 < tot

vector<ll>G;

long long MUL2(long long a, long long b)

{

long long res = 0;

while (b)

{

if (b & 1)

{

res = (res + a) % MOD;

}

a = (a + a) % MOD;

b >>= 1;

}

return res;

}

ll Extend\_Euclid(ll a, ll b, ll&x, ll& y)

{

if (b == 0)

{

x = 1, y = 0;

return a;

}

ll d = Extend\_Euclid(b, a%b, x, y);

ll t = x;

x = y;

y = t - a / b\*y;

return d;

}

//a在模n乘法下的逆元，没有则返回-1

ll inv(ll a, ll n)

{

ll x, y;

ll t = Extend\_Euclid(a, n, x, y);

if (t != 1)

return -1;

return (x%n + n) % n;

}

ll Sum(ll n)

{

ll res = n;

res = (res \* (n + 1)) % MOD;

res = (res \* (2ll \* n + 1)) % MOD;

res = (res \* ((3ll \* n \* n % MOD + 3ll \* n % MOD - 1) % MOD)) % MOD;

res = (res \* inv(30ll, MOD)) % MOD;

return res;

}

ll Pow(ll n)

{

return (n \* n % MOD \* n % MOD \* n % MOD) % MOD;

}

ll dfs(ll n, int start, int count) //容斥原理

{

ll ans = 0;

int i;

for (i = start; i<count; i++)

{

ll temp = Pow(G[i]);

ans = (ans + (temp\*Sum(n / G[i])) % MOD -

(temp\*dfs(n / G[i], i + 1, count)) % MOD + MOD) % MOD;

}

return ans;

}

int main()

{

ll i, j, n, m, N;

phi\_and\_prime\_table(10001ll);

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

{

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

{

scanf("%lld", &N);

ll T = N;

G.clear();

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

{

if (N % prime[i] == 0)

{

G.push\_back(prime[i]);

while (N % prime[i] == 0)

{

N /= prime[i];

}

}

}

if (N != 1)

G.push\_back(N);

ll res = Sum(T);

// ll total = 1ll << G.size();

// for (i = 1; i < total; i++)

// {

// ll t = i;

// ll count = 0;

// ll temp = 1ll;

// for (j = 0; j < G.size(); j++)

// {

// if (t & (1ll << j))

// {

// count++;

// temp \*= G[j];

// }

// }

// if (count & 1)

// {

// ll tt = temp;

// ll u = T / temp;

// for (j = 1; j <= u; j++)

// {

// res = (res - Pow(tt)) % MOD;

// tt \*= temp;

// }

// }

// else

// {

// ll tt = temp;

// ll u = T / temp;

// for (j = 1; j <= u; j++)

// {

// res = (res + Pow(tt)) % MOD;

// tt \*= temp;

// }

// }

// }

printf("%lld\n", ((res - dfs(T, 0, G.size())) % MOD + MOD) % MOD);

}

}

return 0;

}

//求两圆环相交的面积

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

#include <cmath>

#include <cctype>

#include <vector>

#include <queue>

#include <set>

#include <string>

#include <iostream>

using namespace std;

typedef long long ll;

const double eps = 1e-8;

const double PI = acos(-1.0);

int sgn(double x)

{

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

if (x < 0) return -1;

else return 1;

}

struct Point

{

double x, y;

Point() {}

Point(double \_x, double \_y)

{

x = \_x; y = \_y;

}

Point operator + (const Point &b) const

{

return Point(x + b.x, y + b.y);

}

Point operator - (const Point &b) const

{

return Point(x - b.x, y - b.y);

}

double operator ^ (const Point &b) const

{

return x\*b.y - y\*b.x;

}

double operator \* (const Point &b) const

{

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

}

void transXY(double B)

{

double tx = x, ty = y;

x = tx\* cos(B) - ty\*sin(B);

y = tx\* sin(B) + ty\*cos(B);

}

};

double dist(Point a, Point b)

{

return sqrt((a - b)\*(a - b));

}

double getS(Point c1, double r1, Point c2, double r2)

{

double d = dist(c1, c2); //圆心距离

if (r1 + r2 < d + eps) return 0; //相离

if (d < fabs(r1 - r2) + eps) //内切或内含

{

double r = min(r1, r2);

return PI\*r\*r;

}

double x = (d\*d + r1\*r1 - r2\*r2) / (2 \* d); //余弦定理求 r1 \* r2

double t1 = acos(x / r1); // 1/2 \* 第一个扇形圆心角

double t2 = acos((d - x) / r2); // 1/2 \* 第二个扇形圆心角

return r1\*r1\*t1 + r2\*r2\*t2 - d\*r1\*sin(t1); //重叠面积 = 两扇形面积 - 两三角形面积

}

int main()

{

int i, j, n, m;

double r, R, xa, ya, xb, yb;

Point c1, c2;

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

{

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

{

scanf("%lf %lf", &r, &R);

scanf("%lf %lf %lf %lf", &xa, &ya, &xb, &yb);

c1.x = xa;

c1.y = ya;

c2.x = xb;

c2.y = yb;

printf("Case #%d: ", m);

printf("%.6f\n", getS(c1, R, c2, R) - 2.0\*getS(c1, r, c2, R) + getS(c1, r, c2, r));

}

}

return 0;

}

//直筛欧拉函数

#include <iostream>

#include <cstring>

#include <cstdio>

using namespace std;

typedef long long ll;

const ll maxn = 50000;

int euler[maxn+10];

void getEuler()

{

memset(euler, 0, sizeof(euler));

euler[1] = 1;

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

{

if(!euler[i])

{

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

{

if(!euler[j])

{

euler[j] = j;

}

euler[j] = euler[j] / i \* (i-1);

}

}

}

}

//线性筛

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <vector>

using namespace std;

typedef long long ll;

const int INF = 2147483640;

const int maxn = 10000000;

bool check[maxn + 10];

int prime[maxn + 10];

int phi[maxn + 10];

int tot;

void phi\_and\_prime\_table(int N)

{

memset(check, false, sizeof(check));

phi[1] = 1;

tot = 0;

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

{

if(!check[i])

{

prime[tot++] = i;

phi[i] = i - 1;

}

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

{

if(i \* prime[j] > N)

break;

check[i \* prime[j]] = true;

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

{

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

break;

}

else

{

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

}

}

}

}

//质数个数 < tot

//线性筛求莫比乌斯函数μ

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <cmath>

#include <vector>

using namespace std;

typedef long long ll;

const int INF = 2147483640;

const int maxn = 1000000;

bool check[maxn + 10];

int prime[maxn + 10];

int mu[maxn + 10];

void Moblus()

{

memset(check, false, sizeof(check));

mu[1] = 1;

int tot = 0;

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

{

if(!check[i])

{

prime[tot++] = i;

mu[i] = -1;

}

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

{

if(i \* prime[j] > maxn)

break;

check[i \* prime[j]] = true;

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

{

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

break;

}

else

{

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

}

}

}

}

//质数个数 < tot

//给若干区间求最少的不可接数

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

#include <cmath>

#include <cctype>

#include <vector>

#include <queue>

#include <set>

#include <string>

#include <iostream>

using namespace std;

typedef long long ll;

const double eps = 1e-8;

const int INF = 100000000;

const ll MOD = 1e9 + 7;

const int maxn = 600000 + 100;

multiset<ll>G;

struct Node

{

ll b;

ll e;

};

Node ar[maxn];

bool comp(const Node& a, const Node& b)

{

if (a.b != b.b)

return a.b < b.b;

else

return a.e < b.e;

}

int main()

{

ll i, j, k, u, n, m, a, b, cnt, N, M;

while (scanf("%lld %lld", &N, &M) != EOF)

{

cnt = 0;

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

{

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

ar[++cnt].b = a;

ar[cnt].e = a + b;

}

sort(ar + 1, ar + 1 + cnt, comp);

ll flag = -INF;

ll res = 0;

G.clear();

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

{

if (!G.empty())

{

multiset<ll>::iterator pos = G.lower\_bound(ar[i].b - M); //没有间隔M时间以内的限制 改为0即可

if (pos != G.end())

{

ll t = \*pos;

if (t <= ar[i].b)

{

G.erase(pos);

res++;

}

}

}

G.insert(ar[i].e);

}

printf("%lld\n", res); //res为可以合并的区间数

}

return 0;

}

//轻重边树链剖分

#include <cstdio>

#include <cstring>

#include <algorithm>

#include <map>

#include <cmath>

#include <cctype>

#include <vector>

#include <queue>

#include <set>

#include <string>

#include <iostream>

using namespace std;

typedef long long ll;

const double eps = 1e-8;

const int INF = 100000000;

const ll MOD = 1e9 + 7;

const int maxn = 10010 + 100;

//轻重树链剖分就是把树的边进行dfs序然后建成线段树进行操作

struct TREE\_P

{

int fa[maxn]; //fa[v]表示v的父亲结点

int sz[maxn]; //sz[v]表示以v为根的子树的结点个数

int son[maxn]; //son[v]表示v结点的重儿子

int pos[maxn]; //pos[v]表示v与其父亲结点所连的边在线段树中的位置

int deep[maxn]; //deep[v]表示结点v的深度

int top[maxn]; //top[v]表示v所在的重链的顶端节点

int P;

int tot;

int head[maxn];

struct Edge

{

int to;

int next;

};

Edge edge[maxn \* 2];

void init()

{

tot = 0;

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

P = 1;

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

}

void addedge(int u, int v)

{

edge[tot].to = v;

edge[tot].next = head[u];

head[u] = tot++;

}

//dfs序,找deep,fa,sz,son

void dfs1(int pre, int u, int dep)

{

deep[u] = dep;

fa[u] = pre;

sz[u] = 1;

for (int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if (v != pre)

{

dfs1(u, v, dep + 1);

sz[u] += sz[v];

if (son[u] == -1 || sz[son[u]] < sz[v])

{

son[u] = v;

}

}

}

}

//找top,pos;

void dfs2(int u, int first)

{

top[u] = first;

if (son[u] != -1)

{

pos[u] = P++;

dfs2(son[u], first);

}

else

{

pos[u] = P++;

return;

}

for (int i = head[u]; i != -1; i = edge[i].next)

{

int v = edge[i].to;

if (v != son[u] && v != fa[u])

{

dfs2(v, v);

}

}

}

//线段树

struct Node

{

int l;

int r;

int mid;

int Max;

};

Node G[maxn \* 4];

void build(int root, int begin, int end)

{

G[root].l = begin;

G[root].r = end;

G[root].mid = (begin + end) / 2;

G[root].Max = 0;

if (begin == end)

{

return;

}

build(root \* 2, G[root].l, G[root].mid);

build(root \* 2 + 1, G[root].mid + 1, G[root].r);

}

void pushup(int root)

{

G[root].Max = max(G[root \* 2].Max, G[root \* 2 + 1].Max);

}

void update(int root, int index, int val)

{

if (G[root].l == G[root].r && G[root].l == index)

{

G[root].Max = val;

return;

}

if (index <= G[root].mid)

update(root \* 2, index, val);

else

update(root \* 2 + 1, index, val);

pushup(root);

}

int query(int root, int b, int e)

{

if (b <= G[root].l && G[root].r <= e)

{

return G[root].Max;

}

int res = -INF;

if (b <= G[root].mid)

{

res = max(res, query(root \* 2, b, e));

}

if (e > G[root].mid)

{

res = max(res, query(root \* 2 + 1, b, e));

}

return res;

}

int Find(int u, int v)

{

int f1 = top[u];

int f2 = top[v];

int tmp = 0;

while (f1 != f2)

{

if (deep[f1] < deep[f2])

{

swap(f1, f2);

swap(u, v);

}

tmp = max(tmp, query(1, pos[f1], pos[u]));

u = fa[f1];

f1 = top[u];

}

if (u == v)

return tmp;

if (deep[u] > deep[v])

swap(u, v);

return max(tmp, query(1, pos[son[u]], pos[v]));

}

};

TREE\_P tree\_p;

//tree\_p.init()

//以P作为树的结点数

//加双向边

//模板是找区间最大值，单点修改区间查询

int e[maxn][3];

int main()

{

int i, j, n, m, N, M;

scanf("%d", &n);

{

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

{

tree\_p.init();

scanf("%d", &N);

for (i = 0; i < N-1; i++)

{

scanf("%d %d %d", &e[i][0], &e[i][1], &e[i][2]);

tree\_p.addedge(e[i][0], e[i][1]);

tree\_p.addedge(e[i][1], e[i][0]);

}

tree\_p.dfs1(0, 1, 0);

tree\_p.dfs2(1, 1);

tree\_p.build(1, 1, tree\_p.P - 1);

for (i = 0; i < N - 1; i++)

{

if (tree\_p.deep[e[i][0]] > tree\_p.deep[e[i][1]])

{

swap(e[i][0], e[i][1]);

}

tree\_p.update(1, tree\_p.pos[e[i][1]], e[i][2]);

}

char op[10];

int u, v;

while (scanf("%s", op) == 1)

{

if (op[0] == 'D')

break;

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

if (op[0] == 'Q')

printf("%d\n", tree\_p.Find(u, v));

else

tree\_p.update(1, tree\_p.pos[e[u - 1][1]], v);

}

}

}

return 0;

}

//高精度加法

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <cmath>

#include <cctype>

using namespace std;

const int LEN = 10000;

char t[LEN + 10];

char \* G\_plus(char \*a, char \*b);

int main()

{

int i, j, k, n, m, doudou;

char a[LEN], b[LEN];

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

{

doudou = 0;

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

{

scanf("%s %s", a, b);

if (doudou)

putchar('\n');

doudou = 1;

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

printf("%s + %s = %s\n", a, b, G\_plus(a, b));

}

}

return 0;

}

char \* G\_plus(char \*a, char \*b)

{

int L1, L2, L3, i, j, k, u;

char temp[LEN + 10], c[LEN + 10];

memset(c, 48, sizeof(c));

memset(t, 48, sizeof(t));

//puts(a);

//puts(b);

L1 = strlen(a);

L2 = strlen(b);

//不管谁长谁短一起搞，一个没了只加入另一个就行了，两个都没了结束；

for (i = L1 - 1, j = L2 - 1, k = 0; i >= 0 || j >= 0; i--, j--, k++)

{

if (j < 0)

{

u = c[k] + a[i] - '0' - '0';

if (u >= 10)

{

c[k + 1] += 1;

c[k] = u % 10 + '0';

}

else

c[k] = u + '0';

}

if (i < 0)

{

u = c[k] + b[j] - '0' - '0';

if (u >= 10)

{

c[k + 1] += 1;

c[k] = u % 10 + '0';

}

else

c[k] = u + '0';

}

if (i >= 0 && j >= 0)

{

u = c[k] + a[i] + b[j] - '0' - '0' - '0';

//printf("%d\n", u);

if (u >= 10)

{

c[k + 1] += 1;

c[k] = u % 10 + '0';

}

else

{

c[k] = u + '0';

}

}

}

c[k+1] = '\0';

i = strlen(c) - 1;

//去掉前导0；

while (c[i] == '0')

i--;

//如果结果是0，输出“0”

if (i == -1)

return "0";

//把反向改为正向输出

for (j = 0; i >= 0; i--, j++)

t[j] = c[i];

t[j] = '\0';

return t;

}

//高精度减法

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <cmath>

#include <cctype>

using namespace std;

const int LEN = 10010;

char p[LEN], q[LEN], c[LEN + 10];

char ta[LEN], tb[LEN];

char t[LEN + 10];

char \* G\_minus(char \*a, char \*b);

char \* G\_plus(char \*a, char \*b);

void Print\_G\_minus(char \*a, char \*b);

int main()

{

int i, j, k, n, m;

printf("请输入测试组数：\n");

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

{

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

{

printf("请输入两个数做减法：（例如 a b）\n");

scanf("%s %s", p, q);

Print\_G\_minus(p, q);

}

}

return 0;

}

void Print\_G\_minus(char \*a, char \*b)

{

bool flag1 = false, flag2 = false;

if (a[0] == '-')

flag1 = true;

if (b[0] == '-')

flag2 = true;

if (!flag1 && !flag2)

{

printf("%s - %s = %s\n", a, b, G\_minus(a, b));

}

else if (flag1 && !flag2)

{

printf("%s - %s = -%s\n", a, b, G\_plus(a + 1, b));

}

else if (!flag1 && flag2)

{

printf("%s - %s = %s\n", a, b, G\_plus(a, b + 1));

}

else

{

printf("%s - %s = %s\n", a, b, G\_minus(b + 1, a + 1));

}

}

char \* G\_minus(char \*a, char \*b)

{

int L1, L2, L3, i, j, k, u;

bool flag = false;

memset(c, 48, sizeof(c));

memset(t, 48, sizeof(t));

L1 = strlen(a);

L2 = strlen(b);

if (L1 < L2 || (L1 == L2 && strcmp(a, b) < 0))

{

strcpy(ta, b);

strcpy(tb, a);

flag = true;

}

else

{

strcpy(ta, a);

strcpy(tb, b);

}

L1 = strlen(ta);

L2 = strlen(tb);

for (i = L1 - 1, j = L2 - 1, k = 0; i >= 0 || j >= 0; i--, j--, k++)

{

if (j < 0)

{

u = c[k] + ta[i] - '0' - '0';

if (u < 0)

{

c[k + 1] -= 1;

c[k] = u + 10 + '0';

}

else

c[k] = u + '0';

}

if (i < 0)

{

u = c[k] + tb[j] - '0' - '0';

if (u < 0)

{

c[k + 1] -= 1;

c[k] = u + 10 + '0';

}

else

c[k] = u + '0';

}

if (i >= 0 && j >= 0)

{

u = c[k] + ta[i] - tb[j] + '0' - '0' - '0';

//printf("%d\n", u);

if (u < 0)

{

c[k + 1] -= 1;

c[k] = u + 10 + '0';

}

else

{

c[k] = u + '0';

}

}

}

c[k + 1] = '\0';

i = strlen(c) - 1;

while (c[i] == '0')

i--;

if (i == -1)

return "0";

if (flag)

{

t[0] = '-';

for (j = 1; i >= 0; i--, j++)

t[j] = c[i];

t[j] = '\0';

return t;

}

else

{

for (j = 0; i >= 0; i--, j++)

t[j] = c[i];

t[j] = '\0';

return t;

}

}

char \* G\_plus(char \*a, char \*b)

{

int L1, L2, L3, i, j, k, u;

char temp[LEN + 10], c[LEN + 10];

memset(c, 48, sizeof(c));

memset(t, 48, sizeof(t));

//puts(a);

//puts(b);

L1 = strlen(a);

L2 = strlen(b);

//不管谁长谁短一起搞，一个没了只加入另一个就行了，两个都没了结束；

for (i = L1 - 1, j = L2 - 1, k = 0; i >= 0 || j >= 0; i--, j--, k++)

{

if (j < 0)

{

u = c[k] + a[i] - '0' - '0';

if (u >= 10)

{

c[k + 1] += 1;

c[k] = u % 10 + '0';

}

else

c[k] = u + '0';

}

if (i < 0)

{

u = c[k] + b[j] - '0' - '0';

if (u >= 10)

{

c[k + 1] += 1;

c[k] = u % 10 + '0';

}

else

c[k] = u + '0';

}

if (i >= 0 && j >= 0)

{

u = c[k] + a[i] + b[j] - '0' - '0' - '0';

//printf("%d\n", u);

if (u >= 10)

{

c[k + 1] += 1;

c[k] = u % 10 + '0';

}

else

{

c[k] = u + '0';

}

}

}

c[k + 1] = '\0';

i = strlen(c) - 1;

//去掉前导0；

while (c[i] == '0')

i--;

//如果结果是0，输出“0”

if (i == -1)

return "0";

//把反向改为正向输出

for (j = 0; i >= 0; i--, j++)

t[j] = c[i];

t[j] = '\0';

return t;

}

//大数运算

#include <iostream>

#include <string>

using namespace std;

// 实现大数相加 结果存放在num中

void bigIntergerAdd(string &num, string add) {

int goBit = 0; // 存放进位

// 先交换下顺序 加数的位数要比较少

if (num.length() < add.length()) {

string tmp = num;

num = add;

add = tmp;

}

string tmp(num.length() - add.length(), '0');

add = tmp + add;

// 利用string的+号特性 不采用逆序相加法

int len1 = num.length(), len2 = add.length();

for (int i = len1 - 1; i >= 0; --i) {

int tmp = ((num[i] - '0') + (add[i] - '0') + goBit);

num[i] = tmp % 10 + '0';

goBit = tmp / 10;

}

// 特殊情况处理

if (goBit != 0)

num.insert(0, string(1, (char)goBit + '0'));

}

int main()

{

int i, j;

string s1, s2;

cin >> s1 >> s2;

bigIntergerAdd(s1, s2);

for (i = 0; i <= s1.size() - 1; i++)

{

if (s1[i] != '0')

{

break;

}

}

cout << &s1[i] << endl;

cout << s1 << endl;

return 0;

}

#include <iostream>

#include <string>

using namespace std;

// 大数相乘

string bigIntegerPlus(string res, string plusN) {

string ret;

if (res.length()< plusN.length()) {

string tmp = res;

res = plusN;

plusN = tmp;

}

int len1 = res.length(), len2 = plusN.length();

for (int i = len2-1; i>=0; --i ) {

string tmp(len1, '0'); // 存放相乘的中间结果

int goBit =0;

for (int j= len1-1; j >=0; --j) {

int mid = (res[j] -'0') \* (plusN[i] -'0') + goBit;

tmp[j] = mid%10 + '0';

goBit = mid /10;

}

if (goBit != 0)

tmp.insert(0, string(1,goBit +'0'));

for (int m=0; m< len2 -1-i; ++m)

tmp.push\_back('0'); // 补位

// 相乘后就相加 大数相加

if (i == len2-1)

ret = tmp;

else {

int goBit2 =0;

string s(tmp.length() - ret.length() ,'0');

ret = s + ret;

for (int m = tmp.length()-1; m>=0; --m) {

int mid = (tmp[m] -'0')+(ret[m] - '0') + goBit2;

ret[m] = mid %10 +'0';

goBit2 = mid/ 10;

}

if (goBit2 != 0)

ret.insert(0, string(1,goBit +'0'));

}

}

// 去掉前导0

while (ret.length() >1 && ret[0] == '0')

ret.erase(0,1);

return ret;

}

int main()

{

string s1, s2, res;

cin >> s1 >> s2;

res = bigIntegerPlus(s1, s2);

cout << res << endl;

return 0;

}

//00000000000000000001546834444444443412332131351313113131313

//00000000000000000000000000000546468468497978945613321515565561233

//845296249875477088332644014107880443737808280124537602414883530808871188929

#include <iostream>

#include <cstring>

using namespace std;

void bigDivision(char \*src, int num, char sign) {

int i, j;

long long rem = 0; // 存放新余数

long long prem = 0; // 原余数

char res[10000] = "";

bool flag = true;

int k = 0;

for (i = 0; i< strlen(src); ++i) {

rem = prem \* 10/\*向后退一位\*/ + src[i] - '0';

if (rem / num >0 || rem == 0) {

res[k++] = rem / num + '0';

prem = rem %num;

flag = false;

}

else {

prem = rem;

if (!flag)

res[k++] = '0';

}

}

if (sign == '%') {

cout << prem << endl;

return;

}

for (i = 0; i <= strlen(res) - 1; i++)

{

if (res[i] != '0')

{

break;

}

}

for (j = i; j< k; ++j)

cout << res[j];

cout << endl;

}

int main(int argc, char\*\* argv)

{

int i, j;

char src[10000] = "";

int num;

char sign;

while (scanf("%s%d", src, &num) != EOF)

{

bigDivision(src, num, '/');

bigDivision(src, num, '%');

}

return 0;

}

#include <iostream>

#include <string>

using namespace std;

// 求幂 思路: 先变成整数相乘 然后根据小数的位数 结合幂 算出小数点该结果字符串的位置 即可

string bigIntegerPlus(string src, string num) {

string tmp = src;

for (int i =num.length() -1; i >= 0 ; --i) {

string mid(tmp.length(),'0');

int goBit =0;

for (int j = tmp.length()-1; j >= 0; --j) {

int tm = goBit + (tmp[j] -'0')\* (num[i] - '0');

mid[j] = tm% 10 +'0';

goBit = tm /10;

}

for (int q = num.length()-1; q> i; --q)

mid.push\_back('0');

if (goBit != 0)

mid.insert(0, string(1, (char)goBit +'0'));

// 加法运算

if (i == num.length()-1)

src = mid;

else {

goBit =0;

string s(mid.length() - src.length(), '0');

src = s + src;

for (int j = mid.length()-1; j>=0; --j) {

int tm = (mid[j] - '0') +(src[j] - '0') + goBit;

src[j] = tm %10 + '0';

goBit = tm /10;

}

if (goBit !=0)

src.insert(0, string(1, (char)goBit +'0'));

}

}

return src;

}

int main(int argc, char\*\* argv) {

string str;

while ( getline(cin, str)) {

// 分割出待积数 和 幂 以及小数点位置

int i =0;

int index = 0;// 小数位置

int count = 0;//幂次数

string num;

while ( i< str.length()) {

if ( str[i] != ' ') {

if (str[i] == '.')

index = i;

else

num.push\_back(str[i]);

++i;

continue;

}

while ( !isdigit(str[i]))

++i;

if (i + 1 == str.length())

count = str[i] - '0';

else

count = (str[i] - '0') \* 10 + str[i+1] - '0';

break;

}

index = num.length() - index;

string res = num;

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

res = bigIntegerPlus( res, num);

}

index = index \* count;

res.insert(res.length() - index, string("."));

while (res.length() >1 && res[0] == '0')

res.erase(0, 1);

for (int i =res.length()-1; i>=0; --i) {

if (res[i] == '0' )

res.erase(i, i+1);

else

break;

}

cout<< res<< endl;

}

return 0;

}

//memmove

#include <string.h>

#include <stdlib.h>

//原型 void memmove(void \*s1, const void \*s2, size\_t n);

int \*p1, \*p2;

memmove(p1, p2, n\*sizeof(int));

//从p2指向的位置，拷贝n字节，到p1指向的位置； p1指向的那个值也被覆盖；

//s\_gets

#include <string.h>

char \* s\_gets(char \* st, int n);

char \* s\_gets(char \* st, int n)

{

char \* ret\_val;

char \* find;

ret\_val = fgets(st, n, stdin)

if(ret\_val)

{

find = strchr(st, '\n');

if(find)

\*find = '\0';

else

while (getchar() != '\n')

continue;

}

return ret\_val;

}