---------------------------------------------------------------------------------------------------------------------------------

**XCSZBDNL**

-----代码模板

---------------------------------------------------------------------------------------------------------------------------------

1. //                            \_ooOoo\_
2. //                           o8888888o
3. //                           88" . "88
4. //                           (| -\_- |)
5. //                            O\ = /O
6. //                        \_\_\_\_/`---'\\_\_\_\_
7. //                      .   ' \\| |// `.
8. //                       / \\||| : |||// \
9. //                     / \_||||| -:- |||||- \
10. //                       | | \\\ - /// | |
11. //                     | \\_| ''\---/'' | |
12. //                      \ .-\\_\_ `-` \_\_\_/-. /
13. //                   \_\_\_`. .' /--.--\ `. . \_\_
14. //                ."" '< `.\_\_\_\\_<|>\_/\_\_\_.' >'"".
15. //               | | : `- \`.;`\ \_ /`;.`/ - ` : | |
16. //                 \ \ `-. \\_ \_\_\ /\_\_ \_/ .-` / /
17. //         ======`-.\_\_\_\_`-.\_\_\_\\_\_\_\_\_/\_\_\_.-`\_\_\_\_.-'======
18. //                            `=---='
19. //
20. //         .............................................
21. //                  佛祖保佑             永无BUG

**此模板仅为向昌盛的个人模板，与向昌盛本人无关，如有雷同，肯定是抄他的^o^ ←\_←!**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**字符串：**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <stdlib.h>

#define MAX\_CHARACTER 26

#define MAX\_LENTH 1000005

#define MAX\_INPUT 60

#define MAX\_NUMBER 500000

struct Trei {

int flag;

struct Trei \*fail;

struct Trei \*next[MAX\_CHARACTER];

};

struct Trei \*queue[MAX\_NUMBER];

struct Trei tempNode[MAX\_NUMBER]

char description[MAX\_LENTH];

char key\_word[MAX\_INPUT];

struct Trei \*newNode() {

for (int i = 0; i < 36; i++) {

tempNode[cnt\_node].next[i] = NULL;

}

tempNode[cnt\_node].fail = NULL;

tempNode[cnt\_node].flag = 0;

return &tempNode[cnt\_node++];

};

void insertWord(struct Trei \*root, char \*str) {

int i, index;

struct Trei \*temp;

temp = root;

i = 0;

while (str[i]) {

index = str[i] - 'a';

if (temp->next[index] == NULL) {

temp->next[index] = createNewNode();

}

temp = temp->next[index];

i++;

}

temp->flag++;

}

void buildAc(struct Trei \*root) {

struct Trei \*cnt\_node, \*previous\_fail;

int i;

int front, rear;

root->fail = NULL;

front = 0, rear = 1;

queue[front] = root;

while (front < rear) {

cnt\_node = queue[front++];

for (i = 0; i < MAX\_CHARACTER; i++) {

if (cnt\_node->next[i] != NULL) {

if (cnt\_node == root) {

cnt\_node->next[i]->fail = root;

}

else {

previous\_fail = cnt\_node->fail;

while (previous\_fail != NULL) {

if (previous\_fail->next[i] != NULL) {

cnt\_node->next[i]->fail = previous\_fail->next[i];

break;

}

previous\_fail = previous\_fail->fail;

}

if (previous\_fail == NULL) {

cnt\_node->next[i]->fail = root;

}

}

queue[rear++] = cnt\_node->next[i];

}

}

}

}

int query(struct Trei \*root, char \*str) {

int index, i, ans;

struct Trei \*node, \*temp;

ans = 0;

node = root;

i = 0;

while (str[i]) {

index = str[i] - 'a';

while (node->next[index] == NULL && node != root) {

node = node->fail;

}

node = node->next[index];

if (node == NULL) {

node = root;

}

temp = node;

while (temp != root && temp->flag != -1) {

ans += temp->flag;

temp->flag = -1;

temp = temp->fail;

}

i++;

}

return ans;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**图论：**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*tarjan 寻找强连通分量\*/

#include <cstring>

#include <cstdio>

#include <stack>

using namespace std;

const int maxn = 10006;

int pre[maxn], sccno[maxn], lowlink[maxn];

vector<int> path[maxn];

stack<int> S;

int n, dfs\_clock, scc\_cnt;

void dfs(int u) {

pre[u] = lowlink[u] = ++dfs\_clock;

S.push(u);

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

int v = path[u][i];

if (!pre[v]) {

dfs(v);

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

}

else if (!sccno[v]) {

lowlink[u] = min(lowlink[u], pre[v]);

}

}

if (lowlink[u] == pre[u]) {

scc\_cnt++;

for (;;) {

int x = S.top();

S.pop();

sccno[x] = scc\_cnt;

if (x == u) {

break;

}

}

}

}

void findScc() {

scc\_cnt = dfs\_clock = 0;

memset(sccno, 0, sizeof(sccno));

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

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

if (!pre[i]) {

dfs(i);

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*TwoSat\*/

xi V xj对应为addClause(i\_no, 0, j\_no, 0)

!xiV!xj对应为addClause(i\_no, 1, j\_no, 1)

不为非的时候 +0,为非的时候+1

struct TwoSat {

int n;

vector<int> path[maxn];

bool mark[maxn \* 2];

int S[maxn \* 2], c;

bool dfs(int x) {

if (mark[x ^ 1]) {

return false;

}

if (mark[x]) {

return true;

}

mark[x] = true;

S[c++] = x;

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

if (!dfs(path[x][i])) {

return false;

}

}

return true;

}

void init(int n) {

this->n = n;

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

path[i].clear();

}

memset(mark, 0, sizeof(mark));

}

void addClause(int x, int xval, int y, int yval) {

x = x \* 2 + xval;

y = y \* 2 + yval;

path[x ^ 1].push\_back(y);

path[y ^ 1].push\_back(x);

}

bool solve() {

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

if (!mark[i] && !mark[i + 1]) {

c = 0;

if (!dfs(i)) {

while (c > 0) {

mark[S[--c]] = false;

}

if (!dfs(i + 1)) {

return false;

}

}

}

}

return true;

}

};

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\***倍增查找最近公共祖先**\*/

#include <cstdio>

#include <cstring>

#include <vector>

using namespace std;

const int maxn = 50007;

vector<int> path[maxn];

int vis[maxn], depth[maxn], max\_depth, root[6], father[maxn][20];

int n, q, root\_num;

void dfs(int u, int cnt\_depth) {

depth[u] = cnt\_depth;

if (max\_depth < cnt\_depth) {

max\_depth = cnt\_depth;

root[root\_num] = u;

}

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

int v = path[u][i];

if (!vis[v]) {

vis[v] = 1;

father[v][0] = u;

dfs(v, cnt\_depth + 1);

}

}

}

void getAnc() {

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

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

if (father[i][j - 1] != -1) {

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

}

}

}

}

int lca(int x, int y) {

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

int temp = x;

x = y;

y = temp;

}

int k = depth[x] - depth[y];

for (int i = 0; i <= 16; i++) {

if ((1 << i) & k) {

x = father[x][i];

}

}

if (x == y) {

return x;

}

for (int i = 16; i >= 0; i--) {

if (father[x][i] != father[y][i]) {

x = father[x][i];

y = father[y][i];

}

}

return father[x][0];

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\***Dinic最大流**\*/

struct Edge {

int start;

int end;

int cap;

int next;

};

int bfs() {

int front = 0;

int rear = 1;

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

dist[source] = 0;

que[front] = source;

while (front < rear) {

int cnt = que[front];

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

int v = edge[i].v;

if (edge[i].cap > 0 && dist[v] == -1) {

dist[v] = dist[cnt] + 1;

que[rear++] = v;

if (v == dest) {

return 1;

}

}

}

front++;

}

return 0;

}

int dinic() {

int top, cnt, min\_flow, min\_flow\_num, flow;

flow = 0;

while (bfs()) {

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

top = 0;

cnt = source;

while (1) {

if (cnt == dest) {

min\_flow = INF;

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

if (edge[s[i]].cap < min\_flow) {

min\_flow = edge[s[i]].cap;

min\_flow\_num = i;

}

}

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

edge[s[i]].cap -= min\_flow;

edge[s[i] ^ 1].cap += min\_flow;

}

flow += min\_flow;

top = min\_flow\_num;

cnt = edge[s[top]].u;

}

for (int i = cur[cnt]; i != -1; cur[cnt] = i = edge[i].next) {

if (edge[i].cap > 0 && dist[edge[i].v] == dist[cnt] + 1) {

break;

}

}

if (cur[cnt] != -1) {

s[top++] = cur[cnt];

cnt = edge[cur[cnt]].v;

}

else {

if (top == 0) {

break;

}

dist[cnt] = -1;

top--;

cnt = edge[s[top]].u;

}

}

}

return flow;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\***spfa最小费用流**\*/

struct Edge {

int start;

int end;

int cap;

int cost;

int next;

};

struct Edge edge[MAX\_EDGE];

int head[MAX\_POINT], queue[MAX\_POINT], pre\_edge[MAX\_POINT], distance[MAX\_POINT], vis[MAX\_POINT];

int spfa(int &flow, int &cost) {

int i, cnt\_point, next\_point, next\_edge, front, rear;

for (i = 0; i <= point\_number; i++) {

distance[i] = INF;

}

memset(vis, 0, sizeof(vis));

/\*入队\*/

vis[source] = 1;

distance[source] = 0;

front = 0;

rear = 1;

queue[front] = source;

/\*找到最小费用\*/

while (front < rear) {

cnt\_point = queue[front];

vis[cnt\_point] = 1;

for (next\_edge = head[cnt\_point]; next\_edge != -1; next\_edge = edge[next\_edge].next) {

next\_point = edge[next\_edge].end;

if (edge[next\_edge].cap > 0 && distance[next\_point] > distance[cnt\_point] + edge[next\_edge].cost) {

distance[next\_point] = distance[cnt\_point] + edge[next\_edge];

/\*记录更新此费用的边号\*/

pre\_edge[next\_point] = next\_edge;

if (!vis[next\_point]) {

vis[next\_point] = 1;

queue[rear] = next\_point;

rear++;

rear = rear % QUEUE\_LENTH;

}

}

}

front++;

front = front % QUEUE\_LENTH;

vis[cnt\_point] = 0;

}

/\*找不到最小费用了\*/

if (distance[destination] == INF) {

return 0;

}

min\_flow = INF;

/\*找到最小费用之后找到可以通过的最大流\*/

for (i = destination; i != source; i = edge[pre\_edge\_number].start) {

pre\_edge\_number = pre\_edge[i];

if (min\_flow > edge[pre\_edge\_number].cap) {

min\_flow = edge[pre\_edge\_number].cap;

}

}

for (i = destination; i != source; i = edge[pre\_edge\_number].start) {

pre\_edge\_number = pre\_edge[i];

edge[pre\_edge\_number].cap -= min\_flow;

edge[pre\_edge\_number ^ 1].cap += min\_flow;

costs += edge[pre\_edge\_number].cost \* min\_flow;

}

flow += min\_flow;

return 1;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\***树链剖分** \*/

int findMax(int a, int b) {

int f1 = top[a];

int f2 = top[b];

int temp = -INF;

while (f1 != f2) {

if (depth[f1] < depth[f2]) {

swap(f1, f2);

swap(a, b);

}

temp = max(temp, query(1, w[f1], w[a]));

a = father[f1];

f1 = top[a];

}

if (a == b) {

return temp;

}

if (depth[a] > depth[b]) {

swap(a, b);

}

return max(temp, query(1, w[son[a]], w[b]));

}

void dfs(int cnt) {

siz[cnt] = 1;

son[cnt] = 0;

for (int i = 0; i < path[cnt].size(); i++) {

int v = path[cnt][i].first;

if (!vis[v]) {

vis[v] = 1;

father[v] = cnt;

depth[v] = depth[cnt] + 1;

dfs(v);

siz[cnt] += siz[v];

if (siz[v] > siz[son[cnt]]) {

son[cnt] = v;

}

}

}

}

void buildTree(int cnt, int anc) {

w[cnt] = ++idx;

top[cnt] = anc;

if (son[cnt]) {

buildTree(son[cnt], anc);

}

for (int i = 0; i < path[cnt].size(); i++) {

int v = path[cnt][i].first;

if (v != son[cnt] && v != father[cnt]) {

buildTree(v, v);

}

}

}

最小路径覆盖 + 最大匹配数 = 顶点数

最大独立集 + 最小顶点覆盖集 = 顶点数

最大团 = 补图的最大独立集

最小K路径覆盖：构造二部图。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**数据结构**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*单调队列\*/

int main() {

//把val[i]插入到单调队列中

while (rear >= front && val[que[rear]] < val[i]) {

rear--;

}

rear++;

que[rear] = i;

//取出单调队列最大值，len为满足要求的长度

while (rear >= front && i - que[front] + 1 > len) {

front++;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*SplayTree\*/

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int N = (int)1e6+10;

const int inf=~0u>>2;

int s[N];

struct splaytree{

int cnt,root;

int sz[N],ch[N][2],pre[N];

int add[N],rev[N],minn[N],val[N];

inline void newnode(int &t,int v){

t=++cnt;

ch[t][0]=ch[t][1]=0;

add[t]=rev[t]=0;

sz[t]=1;

minn[t]=v;

val[t]=v;

}

inline void pushup(int x){

sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+1;

minn[x]=min(min(minn[ch[x][0]],minn[ch[x][1]]),val[x]);

}

inline void pushdown(int x){

if(rev[x]){

if(ch[x][0])rev[ch[x][0]]^=1;

if(ch[x][1])rev[ch[x][1]]^=1;

swap(ch[x][0],ch[x][1]);

rev[x]^=1;

}

if(add[x]){

if(ch[x][0])add[ch[x][0]]+=add[x],minn[ch[x][0]]+=add[x],val[ch[x][0]]+=add[x];

if(ch[x][1])add[ch[x][1]]+=add[x],minn[ch[x][1]]+=add[x],val[ch[x][1]]+=add[x];

add[x]=0;

}

}

inline bool dir(int x){return ch[pre[x]][1]==x;}

inline void link(int x,int y,int d){

if(y)ch[y][d]=x;

if(x)pre[x]=y;

}

inline void rotate(int x){

int y=pre[x];bool d=dir(x);

pushdown(y);pushdown(x);

link(x,pre[y],dir(y));

link(ch[x][!d],y,d);

link(y,x,!d);

pushup(y);

}

inline void splay(int x,int goal){

pushdown(x);

while(pre[x]!=goal){

if(pre[pre[x]]==goal)rotate(x);

else{

if(dir(x)==dir(pre[x]))rotate(pre[x]);

else rotate(x);

rotate(x);

}

}

pushup(x);

if(goal==0)root=x;

}

inline void select(int k,int goal){

int x=root;

pushdown(x);

while(sz[ch[x][0]]!=k){

if(sz[ch[x][0]] > k ) x=ch[x][0];

else{

k-=(sz[ch[x][0]]+1);x=ch[x][1];

}

pushdown(x);

}

splay(x,goal);

}

inline int get(int L,int R){

select(L-1,0);

select(R+1,root);

return ch[root][1];

}

inline void build(int l,int r,int &t,int p){

if(l>r)return;

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

newnode(t,s[m]);

build(l,m-1,ch[t][0],t);

build(m+1,r,ch[t][1],t);

pre[t]=p;

pushup(t);

}

void init(int n){

cnt=root=0;

sz[0]=ch[0][0]=ch[0][1]=pre[0]=0;

add[0]=rev[0]=0;

minn[0]=inf;

val[0]=inf;

newnode(root,inf);

newnode(ch[root][1],inf);

pre[cnt]=root;

sz[root]=2;

build(1,n,ch[ch[root][1]][0],ch[root][1]);

pushup(ch[root][1]);

pushup(root);

}

inline void ADD(int L,int R,int d){

int x=get(L,R);

add[ch[x][0]]+=d;

minn[ch[x][0]]+=d;

val[ch[x][0]]+=d;

pushup(x);

pushup(root);

}

inline void REVERSE(int L,int R){

int x=get(L,R);

rev[ch[x][0]]^=1;

}

inline void INSERT(int x,int val){

select(x,0);

select(x+1,root);

newnode(ch[ch[root][1]][0],val);

pre[ch[ch[root][1]][0]]=ch[root][1];

pushup(ch[root][1]);

pushup(root);

}

inline void REVOLVE(int L,int R,int t){

int len=(R-L+1);t=(t%len+len)%len;

if(t==0)return;

REVERSE(L,R-t);

REVERSE(R-t+1,R);

REVERSE(L,R);

}

inline void DELETE(int P){

int x=get(P,P);

ch[x][0]=0;

pushup(x);

pushup(root);

}

inline int MIN(int L,int R){

int x=get(L,R);

return minn[ch[x][0]];

}

};

splaytree a;

int main(){

int n,m,x,y,t;

char op[10];

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

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

a.init(n);

scanf("%d",&m);

while(m--){

scanf("%s",op);

if(op[0]=='A'){

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

a.ADD(x,y,t);

}else if(op[0]=='R'){

if(op[3]=='E'){

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

a.REVERSE(x,y);

}else{

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

a.REVOLVE(x,y,t);

}

}else if(op[0]=='I'){

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

a.INSERT(x,t);

}else if(op[0]=='D'){

scanf("%d",&x);

a.DELETE(x);

}else{

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

printf("%d\n",a.MIN(x,y));

}

}

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**数论**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*GCD求整点\*/

//d为求出的最大公因数，如果ax + by = c,c不能整除d的话则说明无解

//求出x后，如果要求正整数点的话：x = (x % b + b) % b，然后再求出y

void gcd(long long a, long long b, long long &d, long long &x, long long &y) {

if (!b) {

d = a;

x = 1;

y = 0;

}

else {

gcd(b, a % b, d, y, x);

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

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*PollardRho找素因子\*/

#include <cstdio>

#include <cstring>

#include <ctime>

#include <algorithm>

using namespace std;

const int MAX\_NUMBER = 1007;

long long n;

long long value[MAX\_NUMBER];

int prim\_number;

long long multi(long long a, long long b, long long n) {

long long res = a % n;

long long ans = 0;

while (b > 0) {

if (b & 1) {

ans = (ans + res) % n;

}

res = (res \* 2) % n;

b = b / 2;

}

return ans;

}

long long getMod(long long a, long long b, long long n) {

long long ans = 1;

while (b > 0) {

if (b & 1) {

ans = multi(ans, a, n);

}

a = multi(a, a, n);

b = b / 2;

}

return ans;

}

long long gcd(long long a, long long b) {

if (b == 0) {

return a;

}

long long c;

while (b != 0) {

c = b;

b = a % b;

a = c;

}

return a;

}

int millerRabin(long long n, long long times) {

if (n == 2) {

return 0;

}

if (n == 1 || n % 2 == 0) {

return 1;

}

long long u = n - 1;

long long a, x, y;

int t = 0;

while (u % 2 == 0) {

t++;

u /= 2;

}

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

a = rand() % (n - 1) + 1;

x = getMod(a, u, n);

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

y = multi(x, x, n);

if (y == 1 && x != 1 && x != n - 1) {

return 1;

}

x = y;

}

if (y != 1) {

return 1;

}

}

return 0;

}

long long pollardRho(long long n) {

long long x, y, d, k = 2, i = 1;

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

y = x;

while (1) {

i++;

x = multi(x, x, n) % n;

d = gcd(y - x, n);

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

return d;

}

if (y == x) {

return n;

}

if (i == k) {

k = k \* 2;

y = x;

}

}

}

void findFactor(long long n) {

if (n == 1) {

return ;

}

if (!millerRabin(n, 6)) {

value[++prim\_number] = n;

return ;

}

long long p = n;

while (p >= n) {

p = pollardRho(p);

}

findFactor(p);

findFactor(n / p);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*Gauss消元\*/

返回为自由元个数，如果不存在解则返回-1

int gauss(int equ, int var) {

int i = 0, j = 0;

while (j < equ && i < var) {

int r = i;

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

if (fabs(matrix[k][j]) > fabs(matrix[r][j])) {

r = k;

}

}

if (r != i) {

for (int k = j; k <= var; k++) {

swap(matrix[i][k], matrix[r][k]);

}

}

if (fabs(matrix[i][j]) < eps) {

j++;

continue;

}

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

if (fabs(matrix[k][j]) > eps) {

double f = matrix[k][j] / matrix[i][j];

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

matrix[k][s] -= f \* matrix[i][s];

}

}

}

i++;

j++;

}

int ret = equ - i;

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

if (fabs(matrix[k][var]) > eps) {

return -1;

}

}

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

if (fabs(matrix[i][i]) < eps) {

continue;

}

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

matrix[i][loc] -= matrix[j][var] \* matrix[i][j];

}

matrix[i][var] /= matrix[i][i];

}

return ret;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*FFT傅里叶快速变换\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <stack>

#include <algorithm>

using namespace std;

const double pi = acos(-1.0);

const double eps = 1e-6;

const int maxn = 300007;

struct Complex {

double a, b;

Complex() {

}

Complex(double a, double b): a(a), b(b) {

}

Complex operator +(const Complex &x) const {

return Complex(a + x.a, b + x.b);

}

Complex operator -(const Complex &x) const {

return Complex(a - x.a, b - x.b);

}

Complex operator \*(const Complex &x) const {

return Complex(a \* x.a - b \* x.b, a \* x.b + b \* x.a);

}

};

char str\_1[maxn], str\_2[maxn];

Complex x\_1[maxn], x\_2[maxn];

stack<int> ans;

//二进制反转

void brc(Complex \*x, int n) {

int i, j, k;

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

if (i < j) {

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

}

k = n >> 1;

while (j >= k) {

j -= k;

k >>= 1;

}

if (j < k) {

j += k;

}

}

}

//on == 1是正变换，on == -1是逆变换

void FFT(Complex \*x, int n, int on) {

int h, i, j, k, p;

double r;

Complex u, t;

brc(x, n);

for (h = 2; h <= n; h <<= 1) {

r = on \* 2.0 \* pi / h;

Complex wn(cos(r), sin(r));

p = h >> 1;

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

Complex w(1, 0);

for (k = j; k < j + p; k++) {

u = x[k];

t = w \* x[k + p];

x[k] = u + t;

x[k + p] = u - t;

w = w \* wn;

}

}

}

if (on == -1) {

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

x[i].a = x[i].a / n;

}

}

}

int main() {

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

while (scanf("%s%s", str\_1, str\_2) != EOF) {

int len\_1 = strlen(str\_1);

int len\_2 = strlen(str\_2);

int tmp = max(len\_1, len\_2) << 1;

int n = 1;

while (n < tmp) {

n <<= 1;

}

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

x\_1[i].a = x\_1[i].b = x\_2[i].a = x\_2[i].b = 0;

}

for (int i = 0; i < len\_1; i++) {

x\_1[i].a = str\_1[len\_1 - i - 1] - '0';

}

for (int i = 0; i < len\_2; i++) {

x\_2[i].a = str\_2[len\_2 - i - 1] - '0';

}

FFT(x\_1, n, 1);

FFT(x\_2, n, 1);

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

x\_1[i] = x\_1[i] \* x\_2[i];

}

FFT(x\_1, n, -1);

while (!ans.empty()) {

ans.pop();

}

int pre = 0;

for (int i = 0; i < len\_1 + len\_2 || pre; i++) {

int cnt = (int)(x\_1[i].a + 0.5);

cnt = cnt + pre;

ans.push(cnt % 10);

pre = cnt / 10;

}

while (!ans.empty() && ans.top() == 0) {

ans.pop();

}

if (ans.empty()) {

puts("0");

}

else {

while (!ans.empty()) {

printf("%d", ans.top());

ans.pop();

}

printf("\n");

}

}

return 0;

}

void getMobius() {

memset(vis, 0, sizeof(vis));

memset(mobius, 0, sizeof(mobius));

mobius[1] = 1;

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

if (!vis[i]) {

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

vis[j] = 1;

if (mobius[j] == -1) {

continue;

}

if ((j / i) % i == 0) {

mobius[j] = -1;

}

else {

mobius[j]++;

}

}

}

}

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

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

mobius[i] = 0;

}

else if (mobius[i] % 2) {

mobius[i] = -1;

}

else {

mobius[i] = 1;

}

}

}

G(n)=sigma(F(d)) (其中d|n)

F(n)=sigma(U(n/d)\*G(d))

G(n)=sigma(F(d))

F(n)=sigma(U(d/n)\*G(d))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

m的原根a,a ^ 0, a ^ 1….a ^ k – 1组成0..m – 1的一个排列

m有phi[phi[m]]个原根

求三维立方和：x1,y1,z1-🡪x2,y2,z2

sum(x2, y2, z2)-sum(x2, y1-1, z2)-sum(x1-1, y2, z2)+sum(x1-1, y1-1, z2)-(sum(x2, y2, z1-1)-sum(x2, y1-1, z1-1)-sum(x1-1, y2, z1-1)+sum(x1-1, y1-1, z1-1)

梅森素数：

Mersenne[maxn]= {0,2,3,5,7,13,17,19,31,61,89,107,127,521,607,1279,2203,2281,3217,4253,

4423,9689,9941,11213,19937,21701,23209,44497,86243,110503,132049,216091,756839,859433,1257787,1398269

,2976221,3021377,6972593,13466917,20996011,24036583,25964951,30402457};