**【GDOI2014】吃**

Time Limits: 2000 ms  Memory Limits: 262144 KB

**Description**

W师兄计划了很久，终于成功的在BG开了一家寿司店。

正当W师兄还在兴奋的时候，这时一个噩耗传来，吃货L师姐居然知道了这件事，而且正赶过来，W师兄瞬间心就冷了下去，但是机智的W师兄也瞬间想到了应付L师姐的策略.......

这时，L师姐到了寿司店，先四处望了望风景，发现现在只有L师姐一个顾客，下面是L师姐的选餐说明：

1.寿司店内的寿司被排在一行共N个盘子里，按从左到右编号为1~N。

2.每个位置上寿司的数量是确定的并且有玻璃窗保护。

3.每隔一段时间就会有一个选餐时间，L师姐可以在一个连续的区间[l, r]中选择其中一盘，然后在该区间之外选择另一盘（如果区间外有盘子）。

L师姐发现这家寿司店厨师的制作速度很快，总能在下一次选餐时间前将寿司数量恢复原样。

作为有尊严有追求的吃货，L师姐也有自己的规则，L师姐在选完两盘寿司后，会决定每口恰好吃D个寿司，且使得两盘寿司刚好可以分别吃完，不剩余任何寿司。比如两盘寿司数量为2和4,那么D=1或者D=2都可以恰好将两盘寿司分别吃干净，而两盘寿司数量为3和5时，那么只能D=1才行。

作为有特殊追求的L师姐才不在乎吃的数量，L师姐在乎的是一口吃多个寿司的感觉。于是，如果L师姐可以一口吃D个寿司，那么L师姐的愉悦值为D，但是L师姐没有选到两盘寿司，那么她的愉悦值为0。

现在L师姐知道每个盘子所放着的寿司数量，L师姐想知道每次选择时间过后她可以获得的最大愉悦值是多少？

**Input**

第一行输入一个整数N，表示寿司的盘子数量。

第二行输入N个整数a1,a2,…,aN，ai表示第i个盘子内的寿司数量。

第三行输入一个整数M，表示有多少个选餐时间。

接下来M行，每行两个整数li, ri (1 <= li <= ri <= N)，含义如题面所示。

**Output**

输出M行，第i行表示第i个选择时间师姐可能达到的最大愉悦值D。

**Sample Input**

输入1：

5

1 2 3 4 5

2

2 3

2 4

输入2：

5

2 4 8 16 32

2

3 4

2 3

**Sample Output**

输出1：

2

1

输出2：

16

8

样例解释：

样例1里的第一个选餐时间，可以选择2和4，这样L师姐就可以每次吃两个寿司，使得两个盘子都可以吃干净，第二个选餐时间，师姐不管选哪两个盘子，都只能每次吃一个。

样例2 里的第一个选餐时间，可以选择16和32,而第二个选餐时间，L师姐可以选择8和16或者8和32。

**Data Constraint**

对于20%的数据，N <= 100, M <= 100, max(a1,a2,…,aN) <= 100。

对于50%的数据，N <= 10000, M <= 10000, max(a1,a2,…,aN) <= 10000。

对于100%的数据，N <= 100000, M <= 100000, max(a1,a2,…,aN) <= 100000。

用pre[][i][j]记录每个数离a[i]最近的(前/后)含有它的第j个因子的数。然后把询问排序离线处理，用线段树f[i]记录在范围之前/之后的数中与i的最大gcd

正着做一次统计(1-l-1,l-r)的maxgcd，反着做一次统计(l-r,r+1-n)的maxgcd即可



#include<cstdio>

#include<cstdlib>

#include<algorithm>

#include<cmath>

#include<cstring>

using namespace std;

int f[800111];

int pre[2][100111][101],can[100111][101];

int hash[100111];

int n,m,tot,i,j,k,last;

int ans[100111],a[100111];

struct data{

int l,r,id;

}q[100111];

bool cmp(data a,data b)

{

return a.l<b.l;

}

bool cmp2(data a,data b)

{

return a.r>b.r;

}

void insert(int l,int r,int x,int y,int t)

{

int mid;

if(l==r){

f[t]=max(f[t],y);

return;

}

mid=(l+r)/2;

if(x<=mid)insert(l,mid,x,y,t+t);

if(x>mid)insert(mid+1,r,x,y,t+t+1);

f[t]=max(f[t+t],f[t+t+1]);

}

int ask(int l,int r,int x,int y,int t)

{

int mid;

if(l==x&&r==y)return f[t];

mid=(l+r)/2;

if(y<=mid)return ask(l,mid,x,y,t+t);

if(x>mid)return ask(mid+1,r,x,y,t+t+1);

if(x<=mid&&y>mid)return max(ask(l,mid,x,mid,t+t),ask(mid+1,r,mid+1,y,t+t+1));

}

int main()

{

scanf("%d",&n);

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

memset(hash,0,sizeof(hash));

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

tot=0;

for(j=1;j<=(int)sqrt(a[i]);j++)if(a[i]%j==0){

tot++;

pre[0][i][tot]=hash[j];

can[i][tot]=j;

hash[j]=i;

k=a[i]/j;

if(j!=k){

tot++;

can[i][tot]=k;

pre[0][i][tot]=hash[k];

hash[k]=i;

}

}

can[i][0]=tot;

}

memset(hash,0,sizeof(hash));

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

tot=0;

for(j=1;j<=(int)sqrt(a[i]);j++)if(a[i]%j==0){

tot++;

can[i][tot]=j;

pre[1][i][tot]=hash[j];

hash[j]=i;

k=a[i]/j;

if(k!=j){

tot++;

can[i][tot]=k;

pre[1][i][tot]=hash[k];

hash[k]=i;

}

}

can[i][0]=tot;

}

scanf("%d",&m);

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

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

q[i].id=i;

}

sort(q+1,q+1+m,cmp);

last=1;

memset(f,0,sizeof(f));

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

for(j=last;j<q[i].l;j++)

for(k=1;k<=can[j][0];k++)if(pre[1][j][k])insert(1,n,pre[1][j][k],can[j][k],1);

ans[q[i].id]=ask(1,n,q[i].l,q[i].r,1);

last=q[i].l;

}

memset(f,0,sizeof(f));

sort(q+1,q+1+m,cmp2);

last=n;

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

for(j=last;j>q[i].r;j--)

for(k=1;k<=can[j][0];k++)if(pre[0][j][k])insert(1,n,pre[0][j][k],can[j][k],1);

ans[q[i].id]=max(ans[q[i].id],ask(1,n,q[i].l,q[i].r,1));

last=q[i].r;

}

for(i=1;i<=m;i++)printf("%d\n",ans[i]);

}

简单题意

* 给定长度为N的数列a1,a2,...,aN，给出M次询问，每次询问一个区间[l, r]，在区间内选一个数X，在[1,l)或(r,N]选一个数Y，使得D = gcd(X,Y)最大。
* 20%数据N<=100,M<=100,max(ai)<=100。
* 50%数据N<=10000,M<=10000,max(ai)<=10000。
* 100%数据N<=100000,M<=100000,max(ai)<=100000。

方法一：

* 对于每次询问暴力枚举所有可能性。
* 时间复杂度O(M\*N\*N)，期望得分20分。

方法二

* D=gcd(X,Y)最大，D同时表示X和Y的最大的公因数。
* 于是对于每次询问，先枚举区间内的数，给其所有因数打上标记1，再枚举区间外的数，给其所有因数打上标记2。
* 此时同时具有标记1和标记2的最大数即为答案。
* 时间复杂度O(M\*N\*√D)。
* 期望得分20～30。

方法三

* 方法二的缺陷在于每次询问都会重新计算所有因数，显然可以通过尽量重用已经计算出来的结果来进行优化。
* 于是我们可以离线处理询问，将询问区间用分块思想排序，每次转移可以通过枚举因子和用堆维护来完成。
* 时间复杂度O(N\*√N\*√D\*logN)。
* 期望得分：50～80。

方法四

* 同样在方法二的基础上，可以通过重用已经枚举出来的因数来进行优化。
* 通过建立函数式值域线段树进行优化，结点每次询问区间[l,r]可以通过函数式线段树用类似搜索剪枝的方法得到。
* 在随机数据下该算法表现良好。
* 时间复杂度不好估计。
* 期望得分:80~100。

方法五

* D=gcd(X,Y)最大，可以理解为区间内存在数X与区间外存在数Y，使得D|X且D|Y。
* 假定整数X，Y所在下标为px，py，则可以建立权值为D的线段[px,py]，此时若询问区间[l,r]与[px,py]为相交关系（不包括包含），则询问[l,r]的答案至少为D。

方法五（2）

* 定义函数V[l,r]=gcd(al, ar)，则询问Q[l,r]的答案为max{
  + V[l1,r1](l1 < l && r1 >= l && r1 <= r),
  + V[l2,r2](l2 >= l && l2 <= r && r2 > r)
  + }

显然V[l,r]的数量为O(N\*N)，不可以先预处理，但是我们关心的是V的值，而对于给定的数X（下标为pos），V[i,pos]的所有不同取值个数为X的因数个数。

方法五（3）

* 故我们可以通过读入所有询问，离线处理（左右分别循环一次），下面只讨论一个方向，另一个方向类似。
* pre[x]表示处理到当前位置pos的数Y时，上一个包含有因数x的数的位置。
* 此时枚举Y的因数d，则V[pre[d],pos]=d。
* 可以通过线段树来快速进行插入和查询。
* 时间复杂度O(N\*√D\*logN)。
* 期望得分：100。