Прости числа 1

Простите числа представляват интерес за много хора, от много дълго време. Освен това се появяват сравнително често на състезания по програмиране. Именно за това в настоящата задача ще трябва да напишете програма, която да отговаря на серия от различни типове заявки върху простите числа в интервала [2, 10^8].

Жокер: В задачи, подобни на тази се използва подхода precomputing. Идеята е веднъж, в началото на програмата да генерираме и запазим всички възможни отговори, за които очакваме да бъдем питани, след което за всяка заявка да извеждаме отговора с константна или логаритмична сложност. За да можем да отговаряме на заявките, предварително трябва да намерим всички прости числа в интервала [2; 10^8]. След което за повечето от тях ще прилагаме различни модификации на двоично търсене върху вече намерените прости числа (с логаритмична сложност), а за заявката за най-малък прост делител ще отговаряме с константна сложност. Важно е да се спомене, че в процеса на търсенето на простите числа и запазването им в масив те са сортирани в нарастващ ред, което ни позволява да използваме двоичното търсене.
Оптимизиран алгоритъм за намиране на прости числа:

Ще разгледаме сравнително прост алгоритъм за намиране на всички прости числа (по-големи от едно цели числа, които се делят само на себе си и на едно) в интервала [2;N]. Най-известния алгоритъм за тези цел е не без известния алгоритъм "Решето на Ератостен". ЗАДЪЛЖИТЕЛНО ПРОЧЕТЕТЕ ЗА "Решето на Ератостен" (Eratosten's Sieve) В ИНТЕРНЕТ, ИНАЧЕ ЩЕ ВИ Е ТРУДНО ДА РАЗБЕРЕТЕ ДОЛУОПИСАНИЯ АЛГОРИТЪМ. Той обаче има сложност O(NloglogN). Съществува линеен алгоритъм, който е над два пъти по-бърз, но негов недостатък е че заема памет от порядъка на N (както и "Решето на Ератостен"), което го прави неприложим за N > 10^8. Въпреки това той е особено полезен и поради един свой „страничен ефект“ – факторизация (представяне на дадено число, като произведение само от прости числа) на всички числа в интервала [2;N], което може да бъде полезно в някои задачи.

Описание на алгоритъма: Алгоритъма се базира на теоремата, че всяко цяло число X, може да бъде представено по единствен уникален начин:
$X = P_{min} . I_{unique}$
където P_{min} е най-малкото, просто число делящо X, а I_{unique} е единствено уникално

число. Доказателството е очевидно: Ясно е че най-малкото, просто число делящо X е конкретно уникално число. $I_{unique} = X / P_{min}$. Щом X и P_{min} са уникални, то следва, че и частното им е уникално.
Нека lp[] е масив, инициализиран с нули, в който за всяко i в интервала [2;N] ще пазим неговия най-малък прост делител, а намерените до момента прости числа ще пазим в масива pr[].
За всяко i от 2 до N разглеждаме следните два случая:
lp[i] == 0 – това означава, че i е просто, следователно няма други делители, а оттук lp[i] = i, след което добавяме i в края на pr[].
lp[i] ≠ 0 – това означава, че i е съставно и неговия най-малък прост делител се явява lp[i].
За всяко просто число pr[j] <= lp[i] е вярно, че съществува съставно число x_j = i * pr[j], за което pr[j] е най-малкия му прост делител, а i е уникално число, т.е. lp[x_j] = pr[j]. Ако pr[j] > lp[i], то уникалното представяне на x_j според горната теорема щеше да е x_j = i * pr[j] = lp[i] * (i * pr[j] / lp[i]) и i нямаше да е I_{unique} = (i * pr[j] / lp[i]) за това x_j, при което pr[j] > lp[i]. Използвайки това, запълваме всички възможни кратни на i lp[x_j] елементи на lp[], за които pr[j] <= lp[i], по следния начин: Въртим цикъл по j, докато j < pr.size() (броя на намерените прости числа до момента) и pr[j] <= lp[i] и i * pr[j] <= N. В цикъла маркираме, че числото x_j = i * pr[j] не е просто и има най-малък прост делител pr[j], така: lp[i * pr[j]] = pr[j]. След края на изпълнението на гореописания алгоритъм масива pr[] ще съдържа всички прости числа в интервала [2;N]

При изпълнение на заявките използувайте двоично търсене в масива pr[], чрез std::lower_bound(pr.begin(), pr.end(), SEARCHED_VALUE) или std::lower_bound(pr, pr + 100000000, SEARCHED_VALUE), ako pr[] е стандартен C/C++ масив.

Input Format

Разгледайте следващата таблица, описваща типовете заявки и очаквания отговор за всяка от тях:

Тип заявка	Очакван отговор
`A a b`	Броя на простите числа в интервала [a, b].
`B k`	Най-малкият прост делител на k.
`C k`	1 - ако k е просто число и 0 в противен случай.
`D k`	1 - ако k обърнато на обратно (след премахването на водещите нули, ако има такива) е просто число и 0 - в противен случай.
`E k`	Най-близкото просто число до k. Ако две числа са еднакво близо до k, да се изведат и двете в нарастващ ред.
`F k`	Броят на простите числа, по-малки от k.
`quit`	Прекратяване изпълнението на програмата.

Първият символ за всяка от командите може да е малка или главна латинска буква.

Constraints

Числата в заявките са в интервала [2, 10^8]

Output Format

За всяка заявка от поредния ред от входните данни, изпечатайте отговорът за нея на нов ред.

Sample Input 0

```
A 3 7
B 176
C 95
D 11
E 5
F 20
E 6
quit
```

Sample Output 0

```
3
2
0
1
5
8
5 7
```