Chytrzy studenci **(chytrzy-studenci)**

Limit pamięci: 32 MB Limit czasu: 2.00 s

Gdy zaczyna się lato, a z latem wakacje i sparingów nie ma już tyle, student Igor musi znaleźć sobie inne źródło utrzymania. Zagląda wtedy do swojego magicznego kuferka, w którym codziennie znajduje N dodatnich liczb naturalnych. Niegdyś Igor po prostu oddawał znalezione przez siebie liczby innemu studentowi, Krzysztofowi, który sprzedawał je na bazarku. Krzysztof zawsze wyznawał ideologię, zgodnie z którą liczbę x można sprzedać tylko i wyłącznie za dokładnie x student coinów. Szybko się jednak okazało, że najlepsze liczby, to liczby nieparzyste, a że konkurencja na bazarku nigdy nie należała do najmniejszych, to liczby parzyste w ogóle się nie sprzedawały. Z drugiej strony liczby nieparzyste zawsze sprzedawały się w mgnieniu oka. Igor wpadł więc na pomysł, by rozłożyć każdą z liczb przed wypuszczeniem ich na rynek. Narzędzia którymi dysponuje pozwalają mu rozłożyć świeżo wyciągniętą z kuferka liczbę na co najwyżej dwie liczby, których iloczyn równy jest oryginalnej liczbie. Czyli na przykład liczbę 6 można rozłożyć na liczby 2 i 3, oraz na liczby 1 i 6. Dalsze rozkładanie wpłynęłoby niekorzystnie na pożądane właściwości liczb, wobec czego stałyby bezwartościowe. Chytrzy studenci chcieliby oczywiście zarobić jak najwięcej, do tej pory nie odkryli jednak algorytmu optymalnego rozkładania liczb. Poprosili więc Ciebie o napisanie programu, który dla każdej z liczb policzy maksymalny zysk, który można dzięki niej uzyskać.

Wejście

W pierwszym wierszu wejścia znajduje się liczba N oznaczająca ilość liczb w magicznym kuferku. W drugim wierszu znajduje się N liczb, i-ta z nich oznaczana jest niżej jako a_i.

Wyjście

Na wyjście należy wypisać N liczb, i-ta z nich powinna być maksymalnym zyskiem możliwym do uzyskania dzięki liczbie a_i

Ograniczenia

$1 \leq N \leq 10^6$, $1 \leq a_i \leq 10^{18}$

Podzadania

Podzadanie	*Warunki*	*Punkty*
1	$N = 1$, $a_i \leq 1\ 000\ 000$	20
2	$N = 1$, $a_i \leq 10^{12}$	40
3	brak dodatkowych ograniczeń	40

Przykład

Wejście	**Wyjście**	**Wyjaśnienie**
`3` `2 6 10`	`1 3 5`	Przykładowo, liczbę 2 można rozłożyć tylko na liczby 1 i 2, zysk będzie tylko z pierwszej

Hakowanie kłódki **(klodka)**

Limit pamięci: 8 MB Limit czasu: 5.00 s

Po wielu latach ciężkich zmagań olimpijskich w końcu udało Ci się znaleźć pracę jako pentester kłódek w firmie Kłódexpol. Kłódki, których solidność przyszło Ci testować, zabezpieczone są szyfrem składającym się z N cyfr dziesiętnych (od 0 do 9). Mechanizm ten bardzo przypomina popularne rozwiązanie w zapięciach rowerowych. Na kłódkach znajduje się N pierścieni, które można obracać, a na każdym pierścieniu znajdują się wszystkie cyfry dziesiętne wypisane cyklicznie po kolei. Kłódka otwiera się, gdy cyfry wskazywane przez aktualną konfigurację pierścieni tworzą ustalony szyfr. Jak powszechnie wiadomo, im kłódka jest solidniejsza, tym więcej czasu potrzeba na jej otworzenie. Oczywistym jest też, że otwieranie kłódki trwa tym dłużej, im więcej razy należy obrócić pierścienie. Dobrym pomysłem byłoby więc przemieszanie cyferek na kłódce po jej zamknięciu w taki sposób, by otwieranie jej zajęło więcej czasu. Nic więc dziwnego, że firma Kłódexpol zamierza umieścić w swoich produktach sugestie takich konfiguracji. Eksperci od szyfrów mają już swoją propozycję, ale zbadanie jej solidności zostało powierzone Tobie. Mając podaną pewną konfigurację kłódki, oraz otwierający ją szyfr, musisz podać minimalną liczbę obrotów pierścieni potrzebną doskonałemu hakerowi kłódek do otworzenia tej kłódki.

Wejście

W pierwszym wierszu wejścia znajduje się liczba N, będąca liczbą pierścieni w kłódce. W drugim wierszu znajduje się N cyfr, i-ta z nich, oznaczana niżej jako a_i, jest cyfrą na którą początkowo ustawiony jest i-ty pierścień kłódki. W trzecim wierszu znajduje się N cyfr, i-ta z nich, oznaczana niżej jako b_i, jest i-tą cyfrą szyfru otwierającego kłódkę.

Wyjście

W pierwszym i jedynym wierszu wyjścia należy wypisać jedną liczbę, będącą najmniejszą liczbą obrotów pierścieni potrzebną do otworzenia kłódki.

Ograniczenia

$1 \leq N \leq 4\,000\,000$, $0 \leq a_i, b_i \leq 9$

Podzadania

Podzadanie	*Warunki*	*Punkty*
1	$N = 5$, $a_i, b_i \in \{0, 1\}$	30
2	$N = 5$	20
3	$N \leq 5$	10
4	$N \leq 200\,000$	20
5	brak dodatkowych ograniczeń	20

Przykład

Wejście

```
3
3 4 1
5 8 9
```

Wyjście

```
8
```

Wyjaśnienie

Pierwszy pierścień można przekręcić o dwie pozycje do góry, drugi o cztery do góry, a trzeci o dwa w dół. Można dowieść, że jest to optymalne otworzenie kłódki.

Mini Saper **(mini-saper)**

Limit pamięci: 256 MB Limit czasu: 2.00 s

To zadanie jest interaktywne.

Pewnie wielu z was słyszało o grze Saper. W zadaniu mamy doczynienia z uproszczoną wersją tej gry. Twoim zadaniem jest oznakowanie pojedynczej bomby, która znajduje się na planszy o wymiarach 3 na 3. Aby to uczynić należy odkrywać pola niezawierające bomby. Każde takie pole zawiera informację o tym czy pole zawierające bombę znajduje się w otoczeniu odkrytego pola. Powiemy, że pole jest w otoczeniu innego pola, jeżeli stykają się one bokiem lub wierzchołkiem. Odkrycie bomby skutkuje porażką. Aby umożliwić Tobie skuteczne oznakowanie bomby, masz zagwarantowane, że bomba nie znajduje się w polu $(1, 1)$.

Protokół interakcji

Do komunikacji z programem sprawdzającym należy używać poniższych zapytań:

- `odkryj i j` – odkrywa pole w i–tym wierszu i j–tej kolumnie. Wiersze i kolumny numerujemy od jedynki. W przypadku próby odkrycia pola poza planszą; pola, które zostało już wcześniej odkryte lub pola, które zawiera bombę zostanie wypisane na wejście `-1`. W tym przypadku należy zakończyć działanie programu. W przeciwnym wypadku na wejście zostanie wypisane `1`, jeżeli w otoczeniu pola (i, j) znajduje się bomba lub `0` w przeciwnym wypadku.
- `bomba i j` – oznakowuje pole w i–tym wierszu i j–tej kolumnie. W przypadku poprawnego oznakowania test zostanie zaliczony, a w przeciwnym, zostanie zwrócony odpowiedni werdykt informujący o niepoprawnym oznaczeniu pola. W obu przypadkach należy zakończyć dalszą interakcję z programem.

Należy pamiętać o opróżnianiu bufora wypisywania po każdym zapytaniu. Aby to uczynić należy wykonać `cout.flush();` lub `cout << endl` jeżeli używamy cin/cout w C++, `fflush(stdout)` dla printf/scanf w C++, `sys.stdout.flush()` w Pythonie oraz `System.out.flush()` w Javie.

Przykładowa interakcja

Wejście	Wyjście
	`odkryj 1 1`
`1`	
	`odkryj 3 3`
`1`	
	`bomba 2 2`

Wyjaśnienie przykładu: Zapytanie o odkrycie pola $(1, 1)$, dało nam informację, że bomba znajduje się na którymś z pól: $(1, 2), (2, 2), (2, 1)$. Następne zapytanie ujawnia, że bomba musi się na którymś z pól: $(3, 2), (2, 2), (2, 3)$. Zatem wiemy, że bomba znajduje się na polu $(2, 2)$. Po zapytaniu `bomba 2 2` należy zakończyć interakcję z programem.