Zadanie: STE
Steganografia

Olimpiada Informatyczna

XXXIII OI, etap III, dzień drugi. Plik źródłowy `ste.*` Dostępna pamięć: 128 MB. *19.03.2026*

Bitoni jako szef bajtockiego wywiadu jest odpowiedzialny za komunikację z siatką szpiegów rozmieszczonych w Bitocji. Zadaniem szpiega jest przekazywanie komunikatów, które są napisami złożonymi z n małych liter alfabetu angielskiego (`a-z`). Każdy taki komunikat należy ukryć w tablicy rozmiaru $k \times k$, w której każde pole zawiera jeden bit. Szpieg używa programu kodującego, aby wygenerować taką tablicę na podstawie swojego komunikatu. Następnie przesyła uzyskaną tablicę do centrali. Bitocki kontrwywiad oczywiście próbuje utrudnić przesłanie komunikatu, ale na szczęście jest w stanie tylko dowolnie przemieszać wiersze i kolumny przesyłanej tablicy. Następnie Bitoni używa programu odkodowującego, aby na podstawie otrzymanej tablicy odtworzyć tę oryginalną.

Mówiąc formalnie, program kodujący otrzymuje napis s i wypisuje binarną tablicę rozmiaru $k \times k$, którą oznaczamy $M[1..k][1..k]$, gdzie $M[i][j]$ to wartość w i-tym wierszu i j-tej kolumnie tablicy. Następnie kontrwywiad wybiera dwie dowolne permutacje $\pi, \sigma : \{1, \ldots, k\} \to \{1, \ldots, k\}$ i na ich podstawie generuje kolejną binarną tablicę rozmiaru $k \times k$, którą oznaczamy $N[1..k][1..k]$, gdzie $N[i][j] = M[\pi(i)][\sigma(j)]$ dla każdych $i, j \in \{1, \ldots, k\}$. Program dekodujący otrzymuje tablicę N i ma na jej podstawie wypisać oryginalny napis s.

Twoim zadaniem jest zaimplementowanie obu programów tak, aby być w stanie ukryć jak najdłuższy napis. **Uwaga: we wszystkich testach poza jednym z testów przykładowych (o nazwie `0e`) zachodzi $k = 1000$.**

Wejście

W pierwszym wierszu wejścia znajduje się dodatnia liczba całkowita k ($1 \leq k \leq 1000$) oznaczająca rozmiar tablicy. W drugim wierszu wejścia znajduje się napis `encoder` lub `decoder` oznaczający, czy program ma zakodować, czy odkodować wiadomość. Jeśli jest to `encoder`, to w trzecim wierszu znajduje się dodatnia liczba całkowita n ($1 \leq n \leq 200\,000$) oznaczająca długość napisu do zakodowania, a w czwartym wierszu znajduje się napis s długości n. Jeśli jest to `decoder`, to w kolejnych k wierszach znajduje się opis otrzymanej tablicy. W i-tym z nich znajduje się opis i-tego wiersza, który składa się z k znaków `0` i `1`.

Wyjście

Jeśli w drugim wierszu wejścia znajdował się napis `encoder`, to program powinien wypisać tablicę $k \times k$ bitów. Wówczas w i-tym (dla $1 \leq i \leq k$) wierszu wyjścia powinien znaleźć się i-ty wiersz tej tablicy złożony z k znaków `0` i `1`. Jeśli w drugim wierszu wejścia znajdował się napis `decoder`, to program powinien wypisać oryginalny napis s.

Przykładowe uruchomienie programu

Załóżmy, że $k = 5$ i wiemy, że napis s jest długości 1. Wówczas program może wypisać tablicę skonstruowaną w następujący sposób. Zakodujemy jedyny znak s jako liczbę $b \in \{0, 1, \ldots, 25\}$ (0 oznacza `a`, 1 oznacza `b`, itd.). Konstruujemy tablicę 5×5, w której dokładnie b bitów jest ustawionych na 1, a pozostałe $25 - b$ bitów jest ustawionych na 0. Dowolne przepermutowanie wierszy i kolumn takiej tablicy nie zmienia liczby bitów ustawionych na 1, więc możemy łatwo odtworzyć napis s. Przykładowe wyniki uruchomienia takiego programu zostały przedstawione poniżej.

Dla danych wejściowych:

```
5
encoder
1
d
```

wyjście może wyglądać następująco:

```
10000
00000
00100
01000
00000
```


Sfinansowano ze środków Ministra Edukacji

Ministerstwo Edukacji Narodowej

a dla danych wejściowych:

```
5
decoder
00000
01000
00100
00000
10000
```

poprawnym wynikiem będzie:

```
d
```

Testy przykładowe:

`0a:` $n = 11$, s = `informatyka`, $\pi, \sigma = (1000, 999, 998, \ldots, 1)^*$.

`0b:` $n = 20\,000$, s = `ababab...`, $\pi = (2,1,4,3,6,5,\ldots)$ (zamienione sąsiednie elementy), $\sigma = (1, 2, \ldots, 1000)$.

`0c:` $n = 190\,000$, s = `abcdabcd...`, $\pi = (1000, 999, 998, \ldots, 1)$, $\sigma = (1, 2, \ldots, 1000)$.

`0d:` $n = 200\,000$, s = `yyyyy...yyyz` (199 999 `y`-ów i jedno `z`), $\pi, \sigma = (1, 2, \ldots, 1000)$ (tablica się nie zmienia).

`0e:` test z przykładu powyżej, $\pi = (5, 4, 3, 2, 1)$, $\sigma = (1, 2, 3, 4, 5)$.

Zwróć uwagę, że Twój program **nie musi** zaliczyć testów przykładowych, aby otrzymać pełną punktację w tym zadaniu, ponieważ we **wszystkich** testach właściwych mamy $k = 1000$.

Ocenianie

Zestaw testów dzieli się na następujące podzadania. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

Podzadanie	Ograniczenia	Punkty
1	$n \leq 190\,000$, $\sigma = (1, 2, 3 \ldots, 1000)$	7
2	$\sigma = (1, 2, 3, \ldots, 1000)$	11
3	$n \leq 4$	5
4	$n \leq 35$	14
5	$n \leq 20\,000$	19
6	$n \leq 40\,000$	13
7	$n \leq 190\,000$	20
8	brak dodatkowych ograniczeń	11

Dla każdego testu Twój program zostanie uruchomiony dwa razy. Za pierwszym razem na wejściu pojawi się liczba k oraz napis `encoder` (wraz z n oraz napisem s) oznaczający, że mamy wygenerować tablicę $M[1..k][1..k]$ na podstawie napisu s. W kolejnym uruchomieniu programu na wejściu pojawi się ta sama wartość k oraz napis `decoder`, a także opis tablicy $N[1..k][1..k]$ otrzymanej przez pewne przepermutowanie wierszy i kolumn tablicy M wygenerowanej w poprzednim uruchomieniu. Test zostanie zaliczony, jeśli odkodowany napis będzie taki sam jak oryginalny napis s.

Testowanie lokalne

W folderze `dlazaw` dostępnym w zakładce "Pliki i testy" w SIO znajduje się skrypt `sterun` pozwalający lokalnie przetestować swoje rozwiązanie. Znajdują się tam także przykładowe, błędne rozwiązania w C++ i Pythonie oraz testy przykładowe. Aby przetestować swoje rozwiązania należy użyć komendy:

```
./sterun [nazwa programu] < [nazwa testu]
```

gdzie `[nazwa programu]` to nazwa pliku z kodem źródłowym Twojego programu (np. `ste.cpp` lub `ste.py`) lub skompilowanym programem (np. `ste.e`), a `[nazwa testu]` to nazwa pliku z testem (np. `ste0a.in`). Lokalne uruchomienie programu dla testu `ste0a.in` może więc wyglądać następująco:

```
./sterun ste.cpp < ste0a.in
```

w przypadku programu w C++ lub

```
./sterun ste.py < ste0a.in
```

w przypadku programu w Pythonie.

*Napis $\pi = (a_1, a_2, a_3, \ldots, a_k)$ oznacza, że $\pi(1) = a_1, \pi(2) = a_2, \pi(3) = a_3, \ldots, \pi(k) = a_k$.