Zadanie: UKR
Ukryte drzewo

Olimpiada Informatyczna

XXXIII OI, etap III, dzień próbny. Plik źródłowy `ukr.*` **Dostępna pamięć: 128 MB.** *17.03.2026*

Bajtek i Bajtosia spędzają czas, grając w nową grę *Ukryte drzewo*. Rozgrywka odbywa się jednocześnie na wielu drzewach, z których każde składa się z n wierzchołków ponumerowanych od 1 do n. Reguły gry są następujące.

Najpierw Bajtek dostaje opis każdego drzewa w postaci listy $n-1$ krawędzi. Następnie, dla każdej krawędzi danego drzewa wybiera jeden bit: `0` lub `1`. Dla każdego wierzchołka danego drzewa definiuje to odpowiadający mu kod utworzony przez sklejenie bitów przypisanych do przyległych do niego krawędzi. Bity sklejamy zgodnie z kolejnością, w jakiej zostały podane krawędzie, do których zostały one przypisane. Ponadto Bajtek wybiera pewien parametr L wspólny dla wszystkich drzew.

Następnie Bajtosia otrzymuje serię zapytań. Każde zapytanie to kod odpowiadający wierzchołkowi jednego z drzew, które otrzymał Bajtek. Zagwarantowane jest, że zapytanie dotyczy wierzchołka o stopniu co najmniej L. Na podstawie otrzymanego kodu binarnego Bajtosia musi odgadnąć numer wierzchołka w drzewie. Zwróć uwagę, że w szczególności oznacza to, że Bajtek musi wybrać bity przypisane krawędziom tak, aby wierzchołki różnych drzew, którym odpowiadają takie same kody binarne, miały takie same numery.

Jeśli Bajtosia poprawnie odpowie na wszystkie zapytania to gracze otrzymują punkty zależne od parametru L wybranego przez Bajtka (im mniejszy, tym więcej punktów mogą uzyskać w całej rozgrywce, patrz sekcja *Ocenianie*). Twoim zadaniem jest napisanie programu, który będzie pełnił rolę Bajtka, jak i Bajtosi. Tę pierwszą nazywamy encoderem, a tę drugą decoderem.

Uwaga: we wszystkich testach poza jednym z testów przykładowych (o nazwie `0a`) zachodzi $n = 1000$. Twój program nie musi poprawnie rozwiązywać testów przykładowych.

Interakcja

Pierwszy wiersz wejścia zawiera jedno słowo `encoder` lub `decoder`, które określa rolę programu. Dalsza zawartość wejścia oraz zawartość wyjścia zależy od tej roli.

Rola Bajtka (encoder)

Wejście

Drugi wiersz wejścia zawiera jedną liczbę całkowitą t ($1 \le t \le 1000$) oznaczającą liczbę drzew. Następnie w kolejnych wierszach znajdują się opisy tych drzew. Każdy opis zaczyna się od liczby całkowitej n ($1 \le n \le 1000$) oznaczającej liczbę wierzchołków w danym drzewie. **We wszystkich testach poza jednym z testów przykładowych zachodzi** $n = 1000$. Następnie w każdym z kolejnych $n-1$ wierszy znajdują się dwie liczby całkowite a i b ($1 \le a, b \le n$) oznaczające krawędź łączącą wierzchołki a oraz b.

Wyjście

Twój program powinien wypisać najpierw t wierszy, z których i-ty odpowiada i-temu drzewu opisanemu na wejściu. Dla każdego drzewa należy wypisać ciąg $n-1$ znaków `0` oraz `1` (bez spacji), opisujący wartości przypisane kolejnym krawędziom zgodnie z kolejnością, w której były podane na wejściu. W kolejnym i ostatnim wierszu należy wypisać liczbę całkowitą L ($1 \le L \le n-1$).

Rola Bajtosi (decoder)

Wejście

W kolejnych wierszach wejścia znajdują się opisy kolejnych zapytań, a ostatni wiersz zawiera słowo `end` oznaczające koniec zapytań. Opis jednego zapytania to ciąg znaków `0` oraz `1` (bez spacji), który jest kodem binarnym pewnego wierzchołka stopnia przynajmniej L jednego z drzew. Kolejne zapytania mogą dotyczyć różnych z t drzew opisanych na wejściu. Możesz założyć, że pojawi się co najwyżej jedno zapytanie dotyczące tego samego wierzchołka danego drzewa.

Sfinansowano ze środków Ministra Edukacji

Ministerstwo Edukacji Narodowej

Wyjście

Dla każdego zapytania Twój program powinien wypisać wiersz zawierający jedną liczbę całkowitą oznaczającą numer wierzchołka odpowiadającego podanemu kodowi.

Przykład

Dla danych wejściowych:

```
encoder
1
5
3 4
5 3
1 4
2 4
```

możliwym wynikiem jest:

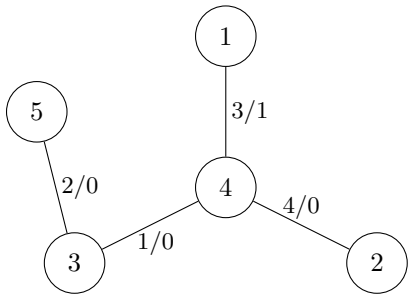
```
0010
2
```

Dla danych wejściowych:

```
decoder
00
010
end
```

poprawnym wynikiem jest:

```
3
4
```



Wyjaśnienie przykładu: Rysunek przedstawia drzewo z przykładu wraz z kolejnością krawędzi na wejściu (przed ukośnikiem) oraz przykładowym przypisaniem wartości przez Bajtka (po ukośniku). Dla takiego przypisania kody wierzchołków będą wyglądać następująco:

- $1 \mapsto 1$ (bity są brane kolejno z krawędzi: 3)
- $2 \mapsto 0$ (bity są brane kolejno z krawędzi: 4)
- $3 \mapsto 00$ (bity są brane kolejno z krawędzi: 1, 2)
- $4 \mapsto 010$ (bity są brane kolejno z krawędzi: 1, 3, 4)
- $5 \mapsto 0$ (bity są brane kolejno z krawędzi: 2)

Ponieważ Bajtek wybrał $L = 2$, Bajtosia będzie pytana tylko o wierzchołki o stopniu co najmniej 2, czyli o wierzchołki 3 i 4.

Testy przykładowe: Test `0a` to test z przykładu powyżej. Poza tym:

`0b:` $t = 1$, $n = 1000$, drzewo to gwiazda, której środek to wierzchołek 1.

`0c:` $t = 2$, $n = 1000$, każde drzewo to kompletne drzewo stopnia 10.

Ocenianie

Zestaw testów dzieli się na następujące podzadania. Testy do każdego podzadania składają się z kilku osobnych grup testów.

Podzadanie	Ograniczenia	Punkty
1	$t \leq 2$, każde drzewo jest gwiazdą	10
2	każde drzewo jest gwiazdą	10
3	$t \leq 2$, wierzchołki o stopniu co najmniej 10 nie sąsiadują ze sobą	10
4	wierzchołki o stopniu co najmniej 10 nie sąsiadują ze sobą	20
5	$t \leq 2$	20
6	brak dodatkowych ograniczeń	30

Sfinansowano ze środków Ministra Edukacji

Ministerstwo Edukacji Narodowej

Aby uzyskać punkty za dany przypadek testowy, musisz odpowiedzieć poprawnie na wszystkie zapytania w teście. Na Twój wynik dodatkowo wpływa wartość L podana przez Twój program działający w roli `encoder` dla danego testu. Poniżej znajduje się tabela przedstawiająca zależność punktacji za test od tej wartości.

L	$[1, 11]$	12	13	$[14, 17]$	$[18, 20]$	$[21, 30]$	$[31, 999]$
Procent punktów	100%	80%	70%	60%	40%	30%	10%

Testowanie lokalne

W folderze `dlazaw` dostępnym w zakładce "Pliki i testy" w SIO znajduje się skrypt `ukrrun` 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:

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

gdzie `[nazwa programu]` to nazwa pliku z kodem źródłowym Twojego programu (np. `ukr.cpp` lub `ukr.py`) lub skompilowanym programem (np. `ukr.e`), a `[nazwa testu]` to nazwa pliku z testem (np. `ukr0a.in`). Lokalne uruchomienie programu dla testu `ukr0a.in` może więc wyglądać następująco:

```
./ukrrun ukr.cpp < ukr0a.in
```

w przypadku programu w C++ lub

```
./ukrrun ukr.py < ukr0a.in
```

w przypadku programu w Pythonie.

Możesz też użyć opcji `-h`, by zobaczyć więcej możliwych ustawień skryptu `ukrrun`.