Jakub Pachocki	Adam Polak, Jakub Radoszewski	Adam Polak
Treść zadania	Opracowanie	Program

Dostępna pamięć: 128 MB. *OI, etap III, dzień drugi, 14.03.2013*

Gdzie jest jedynka?

Bajtazar wymyślił sobie pewną permutację[1] *P liczb od 1 do n. Bajtazar chce, abyś zgadł(a), na którym miejscu tej permutacji znajduje się liczba 1. Abyś nie musiał(a) zgadywać w ciemno, Bajtazar będzie udzielać Ci odpowiedzi na pytania postaci:*

- *f(i, j, d): czy różnica między i-tym a j-tym elementem permutacji P jest podzielna przez d, tzn. czy d | P[i] − P[j]?*
- *g(i, j): czy i-ty element permutacji P jest większy niż j-ty element permutacji P?*

W powyższych pytaniach i, j są dowolnymi indeksami ze zbioru $\{1, 2, \ldots, n\}$, zaś d jest dowolną liczbą całkowitą dodatnią.

Podczas zgadywanki możesz dowolnie wiele razy użyć pytania typu f, natomiast liczba wykonanych pytań typu g musi być jak najmniejsza.

Napisz program komunikujący się z biblioteką dostarczoną przez Bajtazara, który rozwiąże tę zagadkę.

Ocenianie

Niech $M(n)$ będzie najmniejszą możliwą liczbą pytań typu g, za pomocą której da się znaleźć jedynkę w permutacji o ustalonej długości n, niezależnie od tego, jaka jest ta permutacja. Twój program otrzyma punkty za dany test, tylko jeśli liczba pytań typu g, jaką wykona, nie przekroczy $M(n)$. Poza tym program musi zmieścić się w limicie czasowym, a więc wykonać rozsądną liczbę pytań typu f (choć niekoniecznie minimalną).

We wszystkich testach zachodzi warunek $1 \leqslant n \leqslant 500\,000$. W testach wartych 28% punktów zachodzi dodatkowy warunek $n \leqslant 5000$.

Komunikacja

Aby użyć biblioteki, należy wpisać na początku programu:

- **C/C++:** `#include "cgdzlib.h"`
- **Pascal:** `uses pgdzlib;`

Biblioteka udostępnia następujące funkcje i procedury:

- `inicjuj` – *zwraca liczbę n. Powinna zostać użyta dokładnie raz, na samym początku działania programu.*
  - **C/C++:** `int inicjuj();`
  - **Pascal:** `function inicjuj : LongInt;`

[1] Permutacja liczb od 1 do n to taki ciąg o długości n, w którym każda liczba od 1 do n występuje dokładnie raz.

- `f(i, j, d)` – *zadaje bibliotece Bajtazara pytanie typu f. Jej wynikiem jest jedna liczba: 1, jeśli $d \mid P[i] - P[j]$, a 0 w przeciwnym przypadku.*
  - **C/C++:** `int f(int i, int j, int d);`
  - **Pascal:** `function f(i, j, d : LongInt) : LongInt;`
- `g(i, j)` – *zadaje bibliotece Bajtazara pytanie typu g. Jej wynikiem jest jedna liczba: 1, jeśli $P[i] > P[j]$, a 0 w przeciwnym przypadku.*
  - **C/C++:** `int g(int i, int j);`
  - **Pascal:** `function g(i, j : LongInt) : LongInt;`
- `odpowiedz(k)` – *odpowiada bibliotece Bajtazara, że jedynka znajduje się na k-tej pozycji w permutacji (tzn. że $P[k] = 1$). Uruchomienie tej procedury/funkcji* **kończy działanie Twojego programu**.
  - **C/C++:** `void odpowiedz(int k);`
  - **Pascal:** `procedure odpowiedz(k : LongInt);`

Twój program **nie może** *czytać żadnych danych (ani ze standardowego wejścia, ani z plików).* **Nie może** *również nic wypisywać do plików ani na standardowe wyjście. Może pisać na standardowe wyjście diagnostyczne (*`stderr`*) – pamiętaj jednak, że zużywa to cenny czas.*

Przykładowe wykonanie

Poniższa tabela zawiera przykładowy ciąg pytań do biblioteki Bajtazara prowadzący do odgadnięcia pozycji, na której znajduje się jedynka.

Numer pytania	Wywołanie	Wynik	Wyjaśnienie
	`inicjuj`	*5*	$n = 5$
1	`f(1, 2, 2)`	*0*	$2 \nmid P[1] - P[2]$
2	`g(1, 2)`	*0*	$P[1] < P[2]$
3	`f(3, 2, 3)`	*1*	$3 \mid P[3] - P[2]$
4	`g(2, 5)`	*1*	$P[2] > P[5]$
5	`f(1, 3, 2)`	*1*	$2 \mid P[1] - P[3]$
6	`f(1, 4, 3)`	*1*	$3 \mid P[1] - P[4]$
	`odpowiedz(4)`		*Odpowiadamy, że $k = 4$.*

Po drugim pytaniu wiemy, że $P[2] \neq 1$. Stąd po trzecim pytaniu zachodzi jedna z możliwości: ($P[2] = 2$, $P[3] = 5$) lub ($P[2] = 4$, $P[3] = 1$), lub ($P[2] = 5$, $P[3] = 2$). Czwarte pytanie eliminuje pierwszą z tych możliwości. Piąte pytanie pozwala teraz stwierdzić, że $P[1] \in \{3, 4\}$. Skoro więc w szóstym pytaniu mamy $3 \mid P[1] - P[4]$, to $P[1] = 4$, $P[4] = 1$. Szukaną pozycją w permutacji jest $k = 4$.

Podana sekwencja pytań poprawnie znajduje pozycję jedynki w permutacji, jednak nie uzyskałaby żadnych punktów za ten test, gdyż w dowolnej permutacji pięciu elementów jedynkę da się znaleźć za pomocą co najwyżej jednego pytania typu g (tj. $M(5) = 1$). Zauważ, że liczba wykonanych pytań typu f nie gra tu roli.

Eksperymenty

W katalogu **/home/zawodnik/rozw/gdz** *na stacji roboczej dostępna jest przykładowa biblioteka, która pozwoli Ci przetestować poprawność formalną rozwiązania. Biblioteka wczytuje opis permutacji ze standardowego wejścia w następującym formacie:*

- *w pierwszym wierszu liczba całkowita n – długość permutacji;*
- *w drugim wierszu n pooddzielanych pojedynczymi odstępami liczb od 1 do n – kolejne elementy permutacji.*

Przykładowe wejście dla biblioteki znajduje się w pliku `gdz0.in`*. Po zakończeniu działania programu biblioteka wypisuje na standardowe wyjście informację, czy odpowiedź udzielona przez Twoje rozwiązanie była poprawna, oraz liczbę zadanych pytań typu g.*

W tym samym katalogu znajdują się przykładowe rozwiązania `gdz.c`*,* `gdz.cpp` *i* `gdz.pas` *korzystające z biblioteki. Rozwiązania te nie minimalizują liczby pytań typu g.*

Do kompilacji rozwiązania wraz z biblioteką służą polecenia:

- **C:** `gcc -O2 -static cgdzlib.c gdz.c -lm -o gdz`
- **C++:** `g++ -O2 -static cgdzlib.c gdz.cpp -lm -o gdz`
- **Pascal:** `ppc386 -O2 -XS -Xt gdz.pas`

Plik z rozwiązaniem i biblioteka powinny znajdować się w tym samym katalogu.

Rozwiązanie

W zadaniu mamy do czynienia z permutacją o długości n. Możemy zadawać pytania dwóch typów: „czy i-ty i j-ty wyraz dają taką samą resztę z dzielenia przez d?" oraz „czy i-ty wyraz jest większy od j-tego?". Naszym celem jest stwierdzenie, na której pozycji w permutacji znajduje się jedynka. Powinniśmy przy tym zadać minimalną możliwą (dla danej długości permutacji) liczbę pytań drugiego typu oraz niezbyt dużą liczbę pytań pierwszego typu (żeby nie przekroczyć limitów czasowych).

Warto odnotować, że gdyby nie ograniczenie na pytania drugiego typu, zadanie byłoby bardzo proste. Wystarczyłoby zastosować najprostszy algorytm wyznaczania minimum w tablicy, który wykonuje dokładnie $n-1$ porównań elementów (czyli pytań drugiego typu).

Rozwiązanie wolne

Spróbujmy najpierw poszukać rozwiązania, które wykonuje możliwie mało pytań drugiego typu, natomiast nie przejmujmy się na razie liczbą pytań pierwszego typu.

Zauważmy, że jedynka jest jedynym elementem permutacji, dla którego istnieje element większy o $n-1$. Najprostsze rozwiązanie może więc polegać na przejrzeniu wszystkich par różnych indeksów $i < j$ w permutacji i zadaniu dla każdej z nich pytania pierwszego typu z parametrem $d = n-1$, tzn. sprawdzeniu, czy $P[i] - P[j]$ dzieli się przez $n-1$. To pozwoli zidentyfikować dwa indeksy x i y, dla których $n-1$

dzieli $P[x] - P[y]$. Niestety nie będziemy wciąż wiedzieli, czy $P[x] = 1$ i $P[y] = n$, czy jest odwrotnie. W tym miejscu zastosujemy więc pytanie drugiego typu i sprawa stanie się jasna.

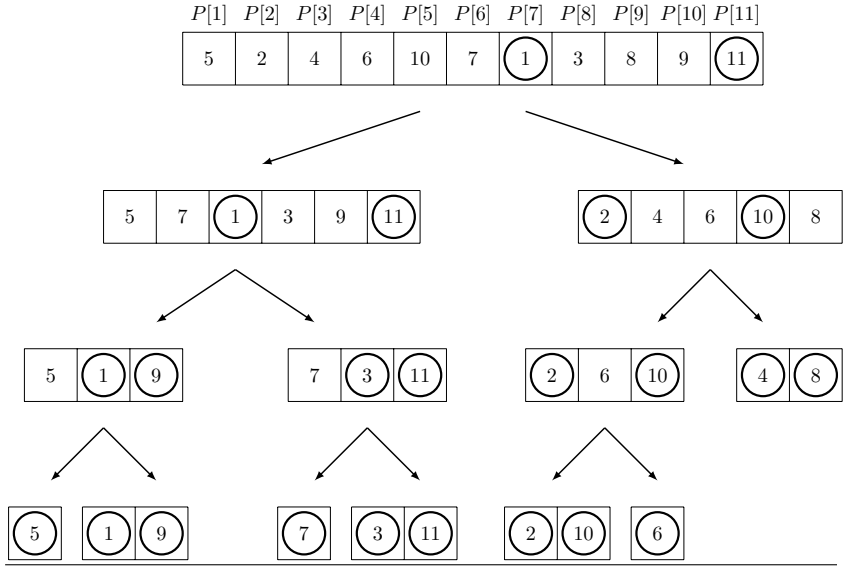
Czy to rozwiązanie rzeczywiście zadaje minimalną liczbę pytań drugiego typu? Okazuje się, że tak: nie licząc przypadku $n = 1$, konieczne jest zadanie przynajmniej jednego pytania drugiego typu. Dzieje się tak, gdyż pytania pierwszego typu nie są w stanie odróżnić permutacji $1, 2, \ldots, n$ od permutacji $n, n-1, \ldots, 1$ i, ogólnie, permutacji $P[i]$ od permutacji $P'[i] = n + 1 - P[i]$.

Opisane rozwiązanie zadaje $O(n^2)$ pytań pierwszego typu i optymalną liczbę pytań drugiego typu. Przykładowa implementacja znajduje się w plikach `gdzs0.c` oraz `gdzs1.pas`. Zgodnie z treścią zadania, takie rozwiązanie uzyskiwało na zawodach 28% punktów.

Rozwiązanie wzorcowe

Poprzednie rozwiązanie sprowadzało się do wyznaczenia pary indeksów x, y takiej, że $\{P[x], P[y]\} = \{1, n\}$, wyłącznie przy pomocy pytań pierwszego typu. Znając taką parę, mogliśmy łatwo znaleźć jedynkę, używając jednego pytania drugiego typu. W rozwiązaniu wzorcowym osiągniemy to samo, wykonując tylko $O(n \log n)$ pytań pierwszego typu. W tym celu skorzystamy z metody „dziel i zwyciężaj”.

W fazie „dziel” wybierzemy dowolny element permutacji (np. pierwszy) i podzielimy pozostałe elementy na te o takiej samej parzystości co wybrany i te o przeciwnej parzystości. Wystarczy do tego $n - 1$ pytań pierwszego typu z parametrem $d = 2$. Następnie wywołamy się rekurencyjnie na obu połówkach, szukając ich minimów



Rys. 1: Przykład działania rozwiązania wzorcowego dla permutacji o $n = 11$ elementach. Kółkami zaznaczono minima i maksima.

i maksimów. Wiemy, że minimum i maksimum całej permutacji znajdują się wśród minimów i maksimów tych dwóch połówek. Aby je wyznaczyć, w fazie „zwyciężaj" sprawdzimy cztery pary elementów, zadając pytanie, czy ich różnica dzieli się przez $n-1$. Na końcu nie będziemy wiedzieli, który z wyznaczonych elementów jest minimum, a który maksimum permutacji; stwierdzimy to, zadając jedyne pytanie drugiego typu.

Kolejne poziomy wywołania rekurencyjnego obsługujemy podobnie, z niewielkimi zmianami: nie sprawdzamy parzystości tylko podzielność przez kolejne potęgi dwójki, a szukając pary elementów stanowiącej minimum i maksimum, nie dzielimy przez $n-1$, tylko liczbę odpowiednio mniejszą. Dokładniej, elementy, z którymi mamy do czynienia na i-tym poziomie rekursji, stanowią ciąg arytmetyczny o różnicy 2^i, więc znając jego długość, możemy wywnioskować, jaka jest wartość bezwzględna różnicy minimum i maksimum ciągu. Wywołania rekurencyjne przerywamy, gdy pozostały nam do rozważenia co najwyżej dwa elementy. Przykład pełnego drzewa rekursji znajduje się na rys. 1.

Czas działania rozwiązania wzorcowego opisuje równanie

$$T(n) = 2T(n/2) + O(n),$$

którego rozwiązaniem jest $T(n) = O(n \log n)$. Implementacje można znaleźć w plikach `gdz.cpp` oraz `gdz1.pas`.

Możemy poczynić jeszcze jedną obserwację, która przyspiesza program dla niektórych wartości n. A mianowicie, jeśli na którymś poziomie rekursji dostajemy fragment nieparzystej długości, to po podziale na dwie części nie musimy się wywoływać na krótszej połówce. Wynika to z tego, że minimum i maksimum przy podziale na pewno wpadną do dłuższej połówki.

Biblioteka

Biblioteka używana do sprawdzania rozwiązań wczytuje z wejścia opis permutacji i odpowiada zgodnie z tym opisem. Jedyna jej „złośliwość" polega na wykorzystaniu faktu, że dla $n > 1$ nie da się obejść bez pytań drugiego typu – jeśli zawodnik próbuje odpowiedzieć przed zadaniem jakiegokolwiek tego typu pytania, biblioteka sygnalizuje błędną odpowiedź.

Testy

Przygotowano 13 losowych testów, z czego pierwsze dwa (jeden z nich zawiera przypadek $n = 1$) są zgrupowane.

Istnieje pewien heurystyczny argument, że w przypadku tego zadania testy losowe są równie dobre co każde inne: gdyby było inaczej i testy losowe miałyby być w jakimś sensie prostsze, rozwiązanie mogłoby zaczynać od wylosowania permutacji i złożenia jej z permutacją wejściową, co w wyniku dałoby również losową permutację, na której można by dalej operować.

W niektórych testach n jest potęgą dwójki. Dzięki temu implementacje rozwiązania wzorcowego muszą wykonać wszystkie wywołania rekurencyjne (za każdym razem fragment ma długość parzystą).