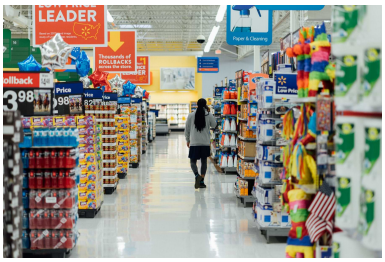
Kommunikationsprobleme beim Einkaufen



Bob geht einkaufen. Das Geschäft hat ein Sortiment von n Produkten, die von 0 bis $n-1$ nummeriert sind. Er möchte das günstigste Produkt mit Nummer zwischen l und r (beide inklusive) kaufen. Allerdings kennt er die Preise der Produkte nicht. Deshalb muss er sich mit der Verkäuferin Alice unterhalten. Sie kennt alle Preise $p_0, \dots p_{n-1}$, aber nicht l und r. Alle Preise sind unterschiedlich und zwischen 1 und n.

Implementiere zwei Programme, eines für Alices und eines für Bob, die einander Bits zur Kommunikation senden können. Am Ende soll Bob *die Nummer* (Index) des günstigsten Produkts zwischen l und r zurückgeben. Während der Kommunikation darf Bob *höchstens 18* Bits senden. Deine Punktezahl hängt von der Anzahl an Bits ab, die Alice sendet. Wenn sie mehr als 10000 Bits sendet, zählt die Lösung als falsch.

Implementierungsdetails

Implementiere folgenden zwei Funktionen:

```
void alice(std::vector<int> p);
int bob(int n, int l, int r);
```

Am Server werden zwei Instanzen deines Programms gestartet. Einmal für Bob und einmal für Alice. Um mit dem jeweils anderen Programm zu kommunizieren, stehen dir folgende Funktionen zur Verfügung:

```
void send(bool bit);
bool receive();
```

Die Funktion `receive` wartet bei Bedarf, bis das andere Programm auch wirklich ein Bit sendet. Wird `send` aufgerufen, bevor das andere Programm `receive` aufgerufen hat, werden die Daten zwischengespeichert (queue) und der `send`-Aufruf ist sofort fertig. Der nächste `receive`-Aufruf auf der Gegenseite kann die Daten dann entgegennehmen. Die mitgelieferte Implementierungsdatei enthält zur Veranschaulichung bereits eine Beispielkommunikation.

Beachte jedoch, dass der mitgelieferte Grader nur eine Instanz des Programms startet. Das soll euch das Debugging erleichtern. Allerdings heißt das auch, dass ihr globale Variablen jeweils nur von `alice` oder `bob` aus verwenden sollt, da ihr euch diese sonst gegenseitig überschreibt. Kommuniziert nur mit den gegebenen `send` und `receive` Funktionen. Zur Hilfe sind folgende Funktionen zum Senden/Empfangen von `bits`-Bit Integer vordefiniert:

```
void send(unsigned long long a, int bits) {
    for(int i = 0; i < bits; i++)
        send((a & (1ull << i)) != 0);
}
auto receive(int bits) {
    unsigned long long a = 0;
    for(int i = 0; i < bits; i++)
        if(receive())
            a |= (1ull << i);
```

```
    return a;
}
```

Um den mitgelieferten Grader unter Linux zu kompilieren verwende die Compileroption `-lpthread`.

Eingabe

Der Grader liest in der ersten Zeile n, l und r ein, gefolgt von einer Zeile mit den Preisen $p_0, \dots p_{n-1}$.

Ausgabe

Falls `bob` den richtigen Index zurückgibt, gibt der Grader die Anzahl an verwendeten Bits in beide Richtungen aus. Sonst wird der Fehler ausgegeben. Falls zu viele Bits verwendet werden, wird das Programm vorzeitig abgebrochen.

Subtasks

Allgemein gilt:

- $0 \leq l \leq r < n \leq 10^6$
- $p_0, \dots p_{n-1}$ ist eine Permutation von $1, \dots n$
- Alice darf höchstens 10000 Bits senden, Bob höchstens 18

Subtask 1 (3 Punkte): $n \leq 500$

Subtask 2 (5 Punkte): $n \leq 1000$

Subtask 3 (12 Punkte): $n \leq 10000$

Subtask 4 ($\frac{P(k)}{2}$ Punkte, bis zu 40): $l \leq \frac{n}{2} \leq r$

Subtask 5 ($\frac{P(k)}{2}$ Punkte, bis zu 40): Keine Einschränkungen

Wobei k die maximale Anzahl an Bits ist, die Alice in einem der Testfälle sendet, und

$$\mathrm{P}(k) = \begin{cases} \lfloor 20 \cdot \frac{10000-k}{5000} \rfloor & \text{wenn } 5000 < k \leq 10000 \\ 20 + \lfloor 35 \cdot \frac{5000-k}{4000} \rfloor & \text{wenn } 1000 < k \leq 5000 \\ 55 + \lfloor 25 \cdot \frac{1000-k}{700} \rfloor & \text{wenn } 300 < k \leq 1000 \\ 80 & \text{wenn } k \leq 300 \end{cases}$$

In anderen Worten: Die Punkte von Subtask 3 und 4 skalieren in der Anzahl an Bits, die Alice sendet. Pro Subtask sind bis zu 40 Punkte möglich, die bei höchstens 300 gesendeten Bits erreicht werden.