

Aufzug

Zeit: 10 Sekunden

Speicherplatz: 64 MiB (pro Prozess)

Dies ist eine interaktive Aufgabe.

Es gibt ein Gebäude mit von 0 bis N durchnummerierten Stockwerken. Stockwerk 0 ist das Erdgeschoss und über Stockwerk N befindet sich das Dach.

In jedem Stockwerk f ($0 \leq f \leq N$) lebt genau ein Bewohner und es gibt eine geheime Ganzzahl v_f ($0 \leq v_f \leq 3$), die nur dem Bewohner bekannt ist. Außerdem wohnt Karlsson auf dem Dach. Karlsson möchte so viele der Werte $v_0, v_1, \dots, v_N$ herausfinden, wie er nur kann. Alle Bewohner **arbeiten zusammen**, damit er möglichst viele Zahlen herausfinden kann.

Um miteinander zu kommunizieren, verwenden die Bewohner den Aufzug des Gebäudes auf die folgende Art und Weise:

Der Aufzug startet im Stockwerk 0 und bewegt sich ausschließlich nach oben. Im Aufzug gibt es Knöpfe, die mit den Zahlen von 1 bis N beschriftet sind. Zu Beginn ist kein Knopf gedrückt. Sobald ein Knopf gedrückt wurde, bleibt er dauerhaft gedrückt. Der Aufzug hält im Stockwerk f an und öffnet seine Türen, wenn entweder $f = 0$ gilt oder der zu diesem Stockwerk gehörende Knopf bei einem früheren Halt gedrückt wurde. Sobald er das höchste Stockwerk erreicht hat, dessen Knopf gedrückt wurde (oder Stockwerk 0, wenn kein Knopf je gedrückt wurde), fährt der Aufzug direkt bis zum Dach, ohne bei den verbleibenden Stockwerken anzuhalten.

Wenn der Aufzug im Stockwerk f ankommt, passiert das folgende:

1. Der Bewohner sieht alle Knöpfe, die bisher gedrückt wurden.
2. Basierend auf dieser Information und dem Wert v_f kann er eine Teilmenge aller bisher nicht gedrückten Knöpfe auswählen, die zu Stockwerken mit **strikt größeren** Nummern als sein eigenes gehören. Der Bewohner drückt diese Knöpfe.
3. Der Aufzug bewegt sich zum nächsthöheren Stockwerk, dessen Knopf gedrückt wurde (oder zum Dach, wenn bereits alle Stockwerke mit gedrückten Knöpfen besucht wurden).

Hält der Aufzug in einem Stockwerk nicht, kann der Bewohner dieses Stockwerks keine Knöpfe drücken.

Wenn der Aufzug das Dach erreicht, sieht Karlsson lediglich die finale Menge der gedrückten Knöpfe. Mit diesen Informationen versucht er, so viele der Werte $v_0, v_1, \dots, v_N$ wie möglich zu

erraten.

Die Werte von v werden festgelegt, bevor das Programm startet, und ändern sich während des Prozesses nicht.

Implementierungsdetails

Es gibt T Testfälle.

Du musst eine Datei einsenden, die die folgenden zwei Funktionen implementiert.

Die erste Funktion, die du implementieren sollst, ist die folgende:

```
std::vector<int> press_buttons(int subtask, int N,
                              int f, int v, std::vector<int> p)
```

- $subtask$: die Teilaufgabe, für die diese Funktion aufgerufen wird. Dabei ist $0 \leq subtask \leq 4$;
- N : die Nummer des höchsten Stockwerks;
- f : das aktuelle Stockwerk. Dabei ist $0 \leq f \leq N$;
- v : der zum Stockwerk f gehörende Wert v_f ;
- p : die aktuell gedrückten Knöpfe in aufsteigender Reihenfolge.

Diese Funktion wird aufgerufen, wenn der Aufzug im Stockwerk f anhält (wenn entweder $f = 0$ gilt oder der zum Stockwerk f gehörende Knopf bei einem früheren Stop gedrückt wurde). Diese Funktion wird für ein bestimmtes Stockwerk nicht aufgerufen, wenn der dazugehörige Knopf vorher nicht gedrückt wurde.

Die Rückgabe ist die Menge der neuen Knöpfe, die gedrückt werden sollen. Die Reihenfolge dieser Werte im zurückgegebenen Array ist nicht wichtig. Jeder gedrückte Knopf x muss die folgenden Bedingungen erfüllen:

- $f < x \leq N$;
- x ist noch nicht in p enthalten und x kommt genau einmal im zurückgelieferten Array vor.

Die zweite Funktion, die du implementieren sollst, ist die folgende:

```
std::vector<int> answer(int subtask, int N, std::vector<int> p)
```

- $subtask$: die Teilaufgabe, für die diese Funktion aufgerufen wird. Dabei ist $0 \leq subtask \leq 4$;
- N : die Nummer des höchsten Stockwerks;
- p : die final gedrückten Knöpfe in aufsteigender Reihenfolge.

Diese Funktion wird für jeden Testfall genau einmal aufgerufen, wenn der Aufzug das Dach erreicht.

Das zurückgegebene Array muss Länge $N + 1$ haben. Dabei muss das i -te Element gleich v_i sein. Die Werte, die Karlsson **nicht ermitteln** kann, müssen durch -1 ersetzt werden.

Wichtig: Die Menschen im Gebäude können nur durch die gedrückten Knöpfe, und auf keinem anderen Weg miteinander kommunizieren. Um dies sicherzustellen, wird dein Programm in $N + 2$ verschiedenen Prozessen aufgerufen: Einer für jedes Stockwerk und einer für das Dach. In jedem Prozess für ein Stockwerk wird die Funktion `press_buttons` maximal T Mal aufgerufen. Im Prozess für das Dach wird `answer` genau T mal aufgerufen. Dein Programm sollte also von keinem gemeinsamen Speicher (wie beispielsweise globalen Variablen) ausgehen. Jeder Prozess verfügt über 64 MiB an Speicher, der vom Speicher der anderen Prozesse getrennt ist. All diese Aufrufe (bis zu $(N + 1) \times T$ Mal `press_buttons` und genau T Mal `answer`) müssen in 10 Sekunden abgeschlossen sein.

Beschränkungen

- $N = 60$
- $T \leq 10\,000$
- $0 \leq v_i \leq 3$ für alle $0 \leq i \leq N$

Beispiel

Das Beispiel hat 1 Testfall mit den folgenden Werten für die Stockwerke:

```
1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1
1 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1
```

Dies ist eine beispielhafte Interaktion:

Dein Programm	Programm der Jury
	<code>press_buttons(0, 60, 0, 1, {})</code>
<code>return {2}</code>	
	<code>press_buttons(0, 60, 2, 0, {2})</code>
<code>return {13, 42}</code>	
	<code>press_buttons(0, 60, 13, 0, {2, 13, 42})</code>
<code>return {}</code>	
	<code>press_buttons(0, 60, 42, 1, {2, 13, 42})</code>
<code>return {}</code>	
	<code>answer(0, 60, {2, 13, 42})</code>
<code>return {1, -1, 0, -1, -1, ..., -1}</code>	

Diese Interaktion ermittelt die korrekten Werte für das 0-te und das 2-te Stockwerk. Alle anderen zurückgegebenen Werte sind gleich -1 , da Karlsson sie nicht ermitteln konnte.

Teilaufgaben

Teilaufgabe	Punkte	Zusätzliche Beschränkungen	Anzahl der Werte, die für die volle Punktzahl ermittelt werden müssen
0	0	Das Beispiel.	–
1	15	$0 \leq v_i \leq 1$ $v_0 = v_N = 1$ Wenn $v_i = 0$, dann $v_{i+1} = 1$ für alle $0 \leq i < N$	60
2	35	$0 \leq v_i \leq 1$	40
3	15	$0 \leq v_i \leq 2$	30
4	35	$0 \leq v_i \leq 3$	25

Punkte

Wenn dein Programm für irgendeinen Testfall einer Teilaufgabe einen falschen Wert zurückgibt oder eine ungültige Operation ausführt (also beispielsweise einen `vector` mit der falschen Länge zurückgibt), erhält deine Einsendung keine Punkte für diese Teilaufgabe.

Für einen Testfall ist die *ermittelte Anzahl* deiner Einsendung die Anzahl der Stockwerke, deren zurückgegebener Wert nicht -1 ist. Es sei K die minimale ermittelte Anzahl unter allen Testfällen einer Teilaufgabe (jede Teilaufgabe hat nur einen Test, der bis zu 10 000 Testfälle enthält). Die Punkte für jede Teilaufgabe werden gemäß der folgenden Tabelle ermittelt.

Teilaufgabe	Wertebereich von K	Punkte
0	-	0
1	$K < 60$	$0.25K$
	$K \geq 60$	15
2	$K < 30$	$0.5K$
	$30 \leq K < 40$	$2K - 45$
	$K \geq 40$	35
3	$K < 30$	$0.5K$
	$K \geq 30$	15
4	$K < 20$	$0.7K$
	$20 \leq K < 25$	$4K - 66$
	$K \geq 25$	35

Sample Grader

Der Sample Grader führt alle Funktionsaufrufe für alle Testfälle in einem einzigen Durchlauf aus.

Das Eingabeformat ist das folgende:

- Zeile 1: drei Ganzzahlen – die Anzahl der Testfälle T , die Nummer des höchsten Stockwerks N und die Zahl der Teilaufgabe S ;
- Zeile $1 + i$: $N + 1$ Ganzzahlen $v_0, v_1, \dots, v_N$, wobei v_f der Wert des Stockwerks f für diesen Testfall ist.

Das Ausgabeformat ist das folgende:

- Zeile i : eine Ganzzahl – die Anzahl der korrekt ermittelten Werte von v im Testfall i ;
- Zeile $T + 1$: finale Punktzahl.

Um detaillierteres Feedback zu erhalten, kannst du das macro `DETAILED` in der ersten Zeile des Graders von `false` zu `true` ändern.

Du kannst den Grader Testfälle für dich generieren lassen (also Werte von v_f), indem du das Macro `AUTO_GENERATE` in der zweiten Zeile des Graders von `false` zu `true` änderst.