Ricerca binaria tra bugiardi incalliti (`ricerca_binaria` [84 punti])

In principio era il logos, e ti vennero comunicati due numeri interi positivi n e k. Miliardi di anni dopo, rinchiuso in un qualche laboratorio, credi che nell'intervallo chiuso $[1, n]$ sia ricompreso un misterioso numero intero x che devi assolutamente individuare. Puoi porre domande del tipo "l'è miga y?" con y un intero in $[1, n]$. Sai che alla tua i-esima domanda risponderà lo strumento $i\%k$, ossia lo strumento che ha per nome il resto della divisione di i per k. In altre parole, le domande vengono smistate in round robin sui k strumenti. Le risposte constano di un singolo carattere nell'alfabeto $\{<, >, =\}$. Otterrai '$=$' se e solo se $y = x$. Altrimenti, dipende ... Se davvero vuoi risolvere tutti i subtask allora devi sapere che, in alcuni di essi, alcuni strumenti dicono ancora sempre il vero, ma altri sono *bugiardi incalliti* e rispondono *consistentemente* '$<$' **quando invece** $x > y$ e '$>$' **quando invece** $x < y$.

In ogni caso, vedi di non sprecare domande (riferisciti al criterio del caso peggiore).

Protocollo di comunicazione

Il tuo programma interagirà col server leggendo dal proprio canale `stdin` e scrivendo sul proprio canale `stdout`. La prima riga di `stdin` contiene T, il numero di testcase da affrontare. All'inizio di ciascun testcase, trovi sulla prossima riga di `stdin` i tre numeri n, k e b separati da spazio. Il valore $b = 0$ indica che tutti gli strumenti dicono il vero, mentre $b = 1$ significa che ciascuno di loro (presi indipendentemente) potrebbe o dire sempre il vero o essere un mentitore seriale (deve ammettere il vero solo quando gli si chiede se $x = x$, come se lavorasse ad ordinamento invertito). Inizia ora un loop che durerà fino a quando tu non sia sicuro di conoscere il numero misterioso x. Non appena conoscerai con certezza l'identità di x puoi chiudere il testcase in corso scrivendo su `stdout` una stringa che inizi per "!" seguito dal numero x. Altrimenti puoi sottoporre la tua prossima domanda per il testcase corrente scrivendo su `stdout` una stringa che inizi per "?" seguito da un numero ricompreso nell'intervallo chiuso $[1, n]$. In questo secondo caso dovrai raccogliere da `stdin` la risposta del server: una riga di un singolo carattere nell'alfabeto $\{<, >, =\}$.

Nota 1: Ogni tua comunicazione verso il server deve essere collocata su una diversa riga di `stdout` e ricordati di forzarne l'invio immediato al server effettuando un flush del tuo output!

Nota 2: Il tuo dialogo diretto col server (ossia ancora prima di aver scritto una sola riga di codice) può aiutarti ad acquisire il corretto protocollo di comunicazione tra il server e il tuo programma. Verrai così anche a meglio conoscere il problema e sarai più pronto a progettare come condurre la fase di codifica. Verrai anche a meglio conoscere come gioca il server, il quale adotta una strategia adattiva per metterti alla prova secondo il criterio del caso peggiore. Il file `results.txt` verrà comunque scaricato nel folder `output` alla fine dell'interazione col server se la termini con `Ctrl-D`.

Nota 3: Per promuovere il dialogo diretto offriamo un servizio aggiuntivo che puoi chiamare con:

```
     rtal -s <URL>  connect -x <token> ricerca_binaria umano -a t=1 -a
n=10 <inserisci_qua_altri_argomenti>
```

Non solo ti darà agio di dialogare nei tuoi tempi ma il suo comportamento può essere meglio parametrizzato attraverso i parametri aggiuntivi opzionali:

```
  -a t=1      per settare ad 1 (o altri valori) il numero di testcase
  -a n=1      per settare ad 1 (o altri valori) l'estremo destro n
  -a k=1      per settare ad 1 (o altri valori) il numero di strumenti k
  -a extra=1  per settare ad 1 (o altri valori) l'eccesso di domande concesso
  -a bugiardi=b  con b=0: nessun bugiardo; b=1: tutti bugiardi; b=2: sceglie il server
con strategia adattiva
```

Esempio

Le righe che iniziano con '?' o '!' sono quelle inviate dallo studente o da suo programma, le altre sono quelle inviate dal server. Inoltre: di ciascuna riga è di mero commento quella parte che comincia col primo carattere di cancelletto '#'.

```
5      # numero di testcase/istanze/partite
1 7 1  #   il server comunica il setting del primo testcase (n=1, k=7, b=1)
!1     # il problem solver dà risposta x=1 e chiude il testcase 1 (fà punto).
1 1 0  #   avvio testcase 2: (n=1, k=1, b=0)
!2     # nessun punto: la risposta del problem solver è fuori range.
2 1 1  #   avvio testcase 3: (n=2, k=1, b=1)
?2     # il problem solver pone la query "x=2?"
<      # il server risponde "x<2"
!2     # nessun punto: l'unico strumento potrebbe essere un mentitore seriale.
2 1 1  #   avvio testcase 4: (n=2, k=1, b=1)
?2     # il problem solver pone la query "x=2?"
<      # il server risponde "x<2"
!1     # nessun punto: il server gioca adattivo e decide che lo strumento non mente.
7 1 0  #   avvio testcase 5: (n=7, k=1, b=0)
?4     # il problem solver (o il programma che gioca per lui) pone la query "x=4?"
<      # il server risponde "x<4"
?2     # il problem solver pone la query "x=2?"
>      # il server risponde "x>2"
!3     # ultimo testcase chiuso (la risposta x=3 fà punto dato che b=0.
*** LA COMUNICAZIONE E' ORA TERMINATA ***
```

Subtask

Il tempo limite per istanza (ossia per ciascun testcase) è sempre di 1 secondo.

I testcase sono raggruppati nei seguenti subtask.

1. **[5 pts← 5 istanze da 1 punto] `small0`:** $n \leq 15$, nessun strumento è bugiardo, ti è concessa una query extra
2. **[10 pts←10 istanze da 1 punto] `small1`:** $n \leq 15$, nessun strumento è bugiardo, non puoi regalare
3. **[5 pts← 5 istanze da 1 punto] `small2`:** $n \leq 15$, $k = 1$, non puoi regalare
4. **[10 pts←10 istanze da 1 punto] `small3`:** $n \leq 15$, nessuna assunzione, non puoi regalare
5. **[5 pts← 5 istanze da 1 punto] `medium0`:** $n \leq 100$, nessun strumento è bugiardo, ti è concessa una query extra
6. **[10 pts←10 istanze da 1 punto] `medium1`:** $n \leq 100$, nessun strumento è bugiardo, non puoi regalare
7. **[5 pts← 5 istanze da 1 punto] `medium2`:** $n \leq 100$, $k = 1$, non puoi regalare
8. **[10 pts←10 istanze da 1 punto] `medium3`:** $n \leq 100$, nessuna assunzione, non puoi regalare
9. **[6 pts← 6 istanze da 1 punto] `big0`:** $n \leq 10^9$, nessun strumento è bugiardo, ti è concessa una query extra
10. **[6 pts← 6 istanze da 1 punto] `big1`:** $n \leq 10^9$, nessun strumento è bugiardo, non puoi regalare
11. **[6 pts← 6 istanze da 1 punto] `big2`:** $n \leq 10^9$, $k = 1$, non puoi regalare
12. **[6 pts← 6 istanze da 1 punto] `big3`:** $n \leq 10^9$, nessuna assunzione, non puoi regalare

In generale, quando si richiede la valutazione di un subtask vengono valutati anche i subtask che li precedono, ma si evita di avventurarsi in subtask successivi fuori dalla portata del tuo programma

che potrebbe andare in crash o comportare tempi lunghi per ottenere la valutazione completa della sottomissione. Ad esempio, chiamando[1, 2] :

```
rtal -s <URL>  connect -x <token> -a size=medium0
     ricerca_binaria  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`small0`, `small1`, `small2`, `small3`, `medium0`.

Il valore di default per l'argomento `size` è `big3` che include tutti i testcase.

[1] <URL> server esame: `wss://ta.di.univr.it/esame`

[2] <URL> server esercitazioni e simula-prove: `wss://ta.di.univr.it/algo`