Fino all'ultimo quadretto (`nim2` [60 punti])

Flavonoidi e precursori di endorfine a parte, una tavola di cioccolato è una griglia di $m \times n$ quadretti. Considera due giocatori che si alternano nell'effettuare la seguente mossa: Spezza la tavoletta in due sotto-griglie, con taglio secco a ghigliottina (un taglio orizzontale, oppure verticale, ma comunque da parte a parte). Mangia una delle due tavolette più piccole così ottenute e riconsegna all'avversario l'altra (riconsegnerai quindi una tavola di $m' \times n$ quadretti con $1 \leq m' < m$, oppure di $m \times n'$ quadretti con $1 \leq n' < n$).

Chi si ritrova con una tavola 1×1 non può più muovere, quindi perde mangiandosi l'ultimo quadretto come premio di consolazione: il suo avversario ha vinto!

Ogni istanza di questo problema comincia con la proposta di una partitina da parte del server (una coppia di numeri m ed n). Accetta la proposta scegliendo se vuoi essere il primo o il secondo a muovere, e quindi gioca ogni tua mossa rispettando i turni, assicurandoti di non perdere la partita!

Interazione

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 partite che verrà giocato. All'inizio di ogni partita, trovi sulla prossima riga di `stdin` una coppia di numeri m ed n separati da spazio; essi costituiscono la proposta di una posizione di gioco avanzata dal server. A questo punto devi rispondere alla proposta dichiarando se preferisci giocare per primo (scrivere 1 su `stdout`) oppure per secondo (scrivere 2 su `stdout`). Dopo di chè inizia la partita vera e propria, che inizia con una mossa da parte del giocatore che tu hai stabilito muova per primo, e poi i giocatori si alternano a giocare finché la configurazione $(1, 1)$ non viene consegnata da un giocatore (il vincente) all'altro (il perdente). In generale, se la configurazione corrente è (m, n), la mossa consiste nel consegnare all'avversario una configurazione (m', n) con $1 \leq m' < m$, oppure una configurazione (m, n') con $1 \leq n' < n$. Per fare questo, il giocatore di turno scrive nella prossima riga del proprio `stdout`, separati da spazio, i due numeri che descrivono la configurazione prodotta dalla sua mossa e che egli intende consegnare all'avversario.

Nota 1: Potendo tu scegliere chi debba giocare per primo, potrai sempre vincere ogni partita, indipendentemente da quello che farà l'avversario.

Nota 2: Affinchè il server non possa tirare la partite per le lunghe per far scadere il time limit, il server si impegna a fare solo mosse in cui almeno una delle due coordinate venga almeno dimezzata in tutte le situazioni in cui questo non gli significa ragalare una partita altrimenti vinta.

Nota 3: 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 4: 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. Il file `results.txt` verrà comunque scaricato nel folder `output` alla fine dell'interazione col server se la termini con `Ctrl-D`.

Esempio

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

```
> 2     # numero di testcase/istanze/partite
> 2 2   # la prima partita inizia dalla configurazione (m=2, n=2)
< 2     # il problem solver (o il programma che gioca per lui) sceglie di muovere per secondo
> 1 2   # il server effettua una mossa consentita, consegnando all'avversario la configurazione (m=1, n=2)
< 1 1   # il problem solver consegna all'avversario la configurazione (m=1, n=1) e vince questa partita
> 4 6   # la seconda partita inizia dalla configurazione (m=4, n=6)
< 1     # il problem solver sceglie di muovere per primo
< 4 4   # con una mossa consentita, il problem solver consegna all'avversario la configurazione (m=4, n=4).
> 2 4   # con una mossa consentita, il server consegna all'avversario la configurazione (m=2, n=4)
< 2 2   # con una mossa consentita, il problem solver consegna all'avversario la configurazione (m=2, n=2).
> 2 1   # con una mossa consentita, il server consegna all'avversario la configurazione (m=2, n=1)
< 1 1   # il problem solver consegna all'avversario la configurazione (m=1, n=1) e vince questa partita
```

Subtask

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

I testcase sono raggruppati nei seguenti subtask.

1. **[12 pts←12 istanze da 1 punto] `tiny`:** $m, n \leq 6$
2. **[12 pts←12 istanze da 1 punto] `small`:** $m, n \leq 10$
3. **[12 pts←12 istanze da 1 punto] `medium`:** $m, n \leq 100$
4. **[12 pts←12 istanze da 1 punto] `skewed`:** $m \leq 3, n \leq 1,000,000,000$
5. **[12 pts←12 istanze da 1 punto] `big`:** $m, n \leq 1,000,000,000$

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=medium
         nim2  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`tiny`, `small`, `medium`.

Il valore di default per l'argomento `size` è `big` 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`