solo calare di singola coordinata (`down_to_the_origin` [96 punti])

Avatar è un drone che può essere posizionato solo sui punti a coordinate intere e non-negative della griglia n dimensionale. Oltre alla posizione iniziale $p = (p_1, p_2, ..., p_n) \in \mathbb{N}^n$ è noto un secondo vettore $\Delta = (\Delta_1, \Delta_2, ..., \Delta_n) \in \mathbb{N}^n$. E' consentito spostare Avatar solo modificando una singola coordinata alla volta, riducendola senza eccedere. Più precisamente, è consentito spostare Avatar da $p' \in \mathbb{N}^n$ a $p'' \in \mathbb{N}^n$ *se e solo se* esiste un $j \in \{1, 2, ..., n\}$ tale che $p''_i = p'_i$ per ogni $i \neq j$, e $0 < p'_j - p''_j \leq \Delta_j$.

Si consideri il gioco dove due giocatori si alternano nello spostare Avatar nel rispetto di tali regole, e dove il primo giocatore che non può più muovere perde la partita.

Ogni istanza di questo problema comincia con la proposta di una partitina da parte del server (l'intero n su una prima riga, le n coordinate di p su una seconda, le n componenti di Δ su una terza). Accetta la proposta scegliendo se vuoi essere il primo o il secondo a muovere (scrivi 1 o 2 sulla tua prima riga), e quindi gioca ogni tua mossa rispettando i turni, assicurandoti la vittoria!

Su ogni partita che vinci raggiungi il goal **victory**, e se ad ogni tuo turno riporti di seguito (su una stessa riga, separate da spazio) tutte le tue mosse vincenti, allora ottieni anche i punti del goal **boring_trainer**. La mossa che fa fede per il proseguio della partita sarà in ogni caso la prima di quelle riportate sulla riga.
Una mossa da una posizione p' trova codifica nella posizione p'' che essa produce: riporta le coordinate di p'' nell'ordine separandole con spazi, se riporti le varie mosse vincenti (quì l'ordine non conta) separale tra loro sempre con spazi.

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` l'intero n, su quella successiva il vettore $\Delta \in \mathbb{N}^n$, e sulla terza il punto $p \in \mathbb{N}^n$. Le n componenti di p e di Δ sono separate tra di loro da spazi; queste tre righe costituiscono l'inivito del server a giocare una partita. Devi rispondere dichiarando se preferisci giocare per primo (scrivere 1 su `stdout`) oppure per secondo (scrivere 2). La partita inizia con una mossa da parte del giocatore che tu hai stabilito muova per primo, dopodichè le mosse si alternano finché la configurazione $(0, ..., 0)$ non viene consegnata da un giocatore (il vincente) all'altro (il perdente). La codifica della mossa è quella spiegata sopra, per entrambi i giocatori: il server sarà sempre un boring_trainer ma si limiterà a rispondere con un'unica mossa quando si vede in svantaggio. Nel caso un giocatore elenchi più mosse, l'avversario prende a riferimento la prima di queste per proseguire la partita.

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. In ogni partita proposta dal server è garantito valere che $\Delta_i \geq \frac{p_i}{2}$ per ogni $i \in \{1, 2, ..., n\}$.

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 '#'.

```
> 3         # numero di testcase/istanze/partite
> 2         # la prima partita si svolge nel piano (n=2)
> 3 3       # Delta=(3,3)
> 8 9       # p=(8,9)
< 2         # il problem solver (o il suo programma) sceglie di muovere per secondo
(un errore!)
> 5 9  8 8  # il server passa all'avversario la configurazione p=(5,9) e riporta ogni
mossa vincente
< 5 6      # il problem solver passa all'avversario la configurazione p=(5,6), manca
di mosse vincenti
> 5 5  2 6  # il server passa all'avversario la configurazione p=(5,9) menzionando
anche l'alternativa che ha
< 2 5      # il problem solver passa all'avversario la configurazione p=(2,5), manca
di mosse vincenti
< 2 2 1 5   # il server passa all'avversario la configurazione p=(2,2), con l'unica
alternativa che ha per vincere
< 2 0      # il problem solver passa all'avversario la configurazione p=(2,0), manca
di mosse vincenti
< 0 0       # il server effettua l'unica mossa vincente e vince, nessun punto per i
problem solver!
> 2         # anche la seconda partita si svolgerà nel piano
> 2 2       # Delta=(2,2)
> 4 6       # p=(4,6)
< 2         # il problem solver sceglie di muovere per secondo (la scelta vincente!)
< 4 4 3 6   # e passa al server la configurazione p=(4,4), dice che l'unica sua
alternativa era passare p=(3,6)
> 2 4       # il server muove restituendo la configurazione p=(2,4)
< 1 4  2 2  # il problem solver passa all'avversario la configurazione p=(1,3), con
unica alternativa p=(2,2)
> 0 4       # il server muove restituendo la configurazione p=(0,4)
< 0 3       # il problem solver passa all'avversario la configurazione p=(0,3), dice
che non aveva alternative
> 0 1       # il server muove restituisce la configurazione p=(0,1)
< 0 0       # il problem solver effettua l'unica mossa vincente e chiude la partita
aggiudicandosi entrambi i goal
> 0 4       # il server passa all'avversario la configurazione p=(0,4)
> 2         # anche la terza partita si svolgerà nel piano
> 9 7       # Delta=(9,7)
> 4 6      # p=(4,6) come nella seconda partita ma ora si può ridurre arbitrariamente
una coordinata a scelta
< 1         # il problem solver sceglie di muovere per primo (la scelta vincente!)
< 4 4          # ed effettua l'unica mossa vincente passando all'avversario la
configurazione p=(4,4)
> 0 4       # il server passa all'avversario la configurazione p=(0,4)
< 0 0       # il problem solver effettua l'unica mossa vincente e chiude la partita
aggiudicandosi entrambi i goal
```

Subtask

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

I testcase sono raggruppati nei seguenti subtask.

1. **[3 pts← 3 istanze da 1 + 0 punti] `line_small`:** $n = 1, p_1 \leq 20$
2. **[3 pts← 3 istanze da 1 + 0 punti] `line_medium`:** $n = 1, p_1 \leq 300$
3. **[3 pts← 3 istanze da 1 + 0 punti] `line_big`:** $n = 1, p_1 \leq 200,000$
4. **[3 pts← 3 istanze da 1 + 0 punti] `line_huge`:** $n = 1, p_1 \leq 200,000,000$
5. **[9 pts← 3 istanze da 2 + 1 punti] `plane_small`:** $n = 2, p_i \leq 20$
6. **[3 pts← 3 istanze da 1 + 0 punti] `plane_medium`:** $n = 2, p_i \leq 100$
7. **[3 pts← 3 istanze da 1 + 0 punti] `plane_big`:** $n = 2, p_i \leq 100,000$
8. **[9 pts← 3 istanze da 2 + 1 punti] `plane_huge`:** $n = 2, p_i \leq 100,000,000$
9. **[6 pts← 3 istanze da 1 + 1 punti] `nim_small`:** $n = 10, \delta_i = p_i \leq 20$
10. **[6 pts← 3 istanze da 1 + 1 punti] `nim_medium`:** $n = 10, \delta_i = p_i \leq 300$
11. **[6 pts← 3 istanze da 1 + 1 punti] `nim_big`:** $n = 10, \delta_i = p_i \leq 200,000$
12. **[6 pts← 3 istanze da 1 + 1 punti] `nim_huge`:** $n = 10, \delta_i = p_i \leq 200,000,000$
13. **[9 pts← 3 istanze da 2 + 1 punti] `small`:** $n = 10, p_i \leq 20$
14. **[9 pts← 3 istanze da 2 + 1 punti] `medium`:** $n = 10, p_i \leq 300$
15. **[9 pts← 3 istanze da 2 + 1 punti] `big`:** $n = 10, p_i \leq 200,000$
16. **[9 pts← 3 istanze da 2 + 1 punti] `huge`:** $n = 10, p_i \leq 200,000,000$

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=plane_big
    down_to_the_origin  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`line_small`, `line_medium`, `line_big`, `line_huge`, `plane_small`, `plane_medium`, `plane_big`.

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