Count, rank e unrank delle piastrellature di un bagno 1xn (`piastrelle`)

Per ogni $n \in \mathbb{N}$, indichiamo con $\mathcal{P}_n$ l'insieme delle possibili piastrellature di una griglia 1xn con piastrelle 1x1 e 1x2. Graficamente, con `[]` indichiamo la piastrella 1x1 che copre una singola cella, mentre la piastrella 1x2, indicata con `[--]`, ricopre due celle affiancate. La prima piastrallatura della griglia 1x3 è `[][][]`, la seconda è `[][--]` e la terza ed ultima è `[--][]`. Su $\mathcal{P}_n$ vige un'ordinamento totale e chiamiamo *rango* di una piastrellatura $P \in \mathcal{P}_n$ la sua posizione in tale ordine. Ad esempio, per $n = 4$:

```
P           rango di P
[][][][]    0
[][][--]    1
[][--][]    2
[--][][]    3
[--][--]    4
```

Se ti restano dubbi sull'ordinamento inteso per le piastrellature in $\mathcal{P}_n$ per altri $n \in \mathbb{N}$ puoi interrogare il servizio `list` offerto per questo problema `TALight` fornendo in input il tuo valore per n.

Ti chiediamo di produrre del codice che implementi le seguenti competenze:

counting dato un $n \in \mathbb{N}$, calcolare il numero $f(n)$ di piastrellature in $\mathcal{P}_n$.

ranking dati $n \in \mathbb{N}$ e $P \in \mathcal{P}_n$, restituire il rango r di P in $\mathcal{P}_n$, ossia l'intero nell'intervallo $[0, f(n) - 1]$ che specifica la posizione di P nel nostro ordinamento (si parte da 0).

unranking dati i due numeri naturali n e r, con $r \in [0, f(n) - 1]$, restituire la piastrellatura P in $\mathcal{P}_n$ che appare in posizione r (partendo da 0) entro il nostro ordinamento su $\mathcal{P}_n$.

Nota: `rank(unrank(t, r)) = r`.

Assunzioni

1. quando ti chiediamo di computare $f(n)$, in realtà, se vorrai, potrai limitarti a restituire il resto della divisione di $f(n)$ per $1.000.000.007$. Con questo obiettivo puoi acquisire maggiore efficienza che potrebbe esserti necessaria per risolvere le istanze più grandi (subtask `huge`) dove l'elaborazione delle molte cifre di un numero aumenterebbe esponenzialmente i costi.
2. quando ti chiediamo di fare `rank`, è nostra cura fornirti in input una $P \in \mathcal{P}_n$ di rango inferiore a $1.000.000.007$.
3. quando ti chiediamo di fare `unrank`, è nostra cura fornirti in input un $r < 1.000.000.007$.

NB: per rank e unrank, anche se il rango coinvolto è inferiore $1.000.000.007$, $f(n)$ potrebbe essere maggiore, quindi, se utilizzi i numeri modulo $1.000.000.007$ dovresti ricordare quantomeno se il vero valore di $f(n)$ è maggiore di $1.000.000.007$, per effettuare correttamente tutti i confronti necessari.

Input

Si legga l'input da `stdin`. La prima riga contiene T, il numero di testcase (istanze) da risolvere. Seguono T istanze del problema, dove ogni istanza può porre diverse domande tutte relative ad una stesso valore di n. Per ogni istanza, la prima riga contiene 4 numeri interi separati da uno spazio: n, c, r ed u, dove $c \in \{0, 1\}$ indica se si richiede il valore di $f(n)$, r è il numero di richieste di ranking e u è il numero di richieste di unranking. Seguono r righe, l'i-esima delle quali contiene una piastrellatura $P_i \in \mathcal{P}_n$. (Si utilizza la rappresentazione a riga ottenuta concatenando le righe della piastrellatura come illustrato sopra.) Seguono infine u righe, l'i-esima delle quali contiene un numero intero $r_i \in [0, f(n) - 1]$.

Output

Per ciascuna istanza, prima di leggere l'istanza successiva, scrivi su `stdout` il tuo output strutturato in tre parti:

1. Se $c = 0$ questa prima parte è vuota, altrimenti $c = 1$ e devi stampare su `stdout` una singola riga che contenga $f(n)$ oppure $f(n)\%1.000.000.007$. Noi teniamo per buone entrambe queste risposte ma con la prima potresti non riuscire a risolvere nei tempi le istanze più grandi.
2. seguono $r \geq 0$ righe l'i-esima delle quali contiene un numero intero nell'intervallo $[0, f(n) - 1]$ che vuole essere il rango di P_i, ossia dell'i-esima piastrellatura di $\mathcal{P}_n$ ricevuta in input per questa istanza.
3. seguono $u \geq 0$ righe l'i-esima delle quali contiene la piastrellatura di rango r_i entro $\mathcal{P}_n$, dove r_i è il numero contenuto nell'i-esima riga del blocco delle ultime u righe di input per questa istanza.

Esempio

Input da `stdin`

```
3
 4 1 0 0
 4 0 4 0
  [][--][]
  [][][][]
  [][][--]
  [--][--]
 4 0 0 4
  2
  0
  1
  4
```

Output su `stdout`

```
5
2
0
1
4
[][--][]
[][][][]
[][][--]
[--][--]
```

In tutti e tre i testcase del'esempio si lavora sul caso $n = 4$. Nel primo/secondo/terzo testcase si chiede di risolvere solamente il problema di counting/ranking/unranking.

Subtask

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

I testcase sono raggruppati nei seguenti subtask.

1. **[3 istanze] `esempi_testo`:** i tre esempi del testo
2. **[6 istanze] `tiny_c`:** $n \leq 10, c = 1, r = 0, u = 0$
3. **[6 istanze] `tiny_r`:** $n \leq 10, c = 0, u = 0, r \leq 20$
4. **[6 istanze] `tiny_u`:** $n \leq 10, c = 0, r = 0, u \leq 20$
5. **[6 istanze] `small_c`:** $n \leq 50, c = 1, r = 0, u = 0$
6. **[6 istanze] `small_r`:** $n \leq 50, c = 0, u = 0, r \leq 50$
7. **[6 istanze] `small_u`:** $n \leq 50, c = 0, r = 0, u \leq 50$
8. **[6 istanze] `medium_c`:** $n \leq 100, c = 1, r = 0, u = 0$
9. **[7 istanze] `medium_r`:** $n \leq 100, c = 0, u = 0, r \leq 100$
10. **[7 istanze] `medium_u`:** $n \leq 100, c = 0, r = 0, u \leq 100$
11. **[7 istanze] `big_c`:** $n \leq 1000, c = 1, r = 0, u = 0$
12. **[7 istanze] `big_r`:** $n \leq 1000, c = 0, u = 0, r \leq 5$
13. **[7 istanze] `big_u`:** $n \leq 1000, c = 0, r = 0, u \leq 5$
14. **[28 istanze] `huge_c`:** $n \leq 100000, c = 1, r = 0, u = 0$
15. **[20 istanze] `colossal_c`:** $n \leq 10000000000, c = 1, r = 0, u = 0$

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=small_r
      piastrelle  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `tiny_c`, `tiny_r`, `tiny_u`, `small_c`, `small_r`.

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

Servizi TALight a tua disposizione

Se reputi ti serva disambiguare meglio l'ordine da noi stabilito su $\mathcal{P}_n$, puoi chiedere a TALight, ad esempio:

```
rtal -s <URL>  connect -x <token> piastrelle list  -a n=3
```

Ulteriori servizi di supporto alla tua esplorazione del problema (esempi di chiamate):

```
rtal -s <URL>  connect -x <token> piastrelle check_count  -a n=3 -a risp=5
```

```
rtal -s <URL>  connect -x <token> piastrelle check_rank  -a n=3 -a input_tiling="[--][]" -a right_rank=2
```

```
rtal -s <URL>  connect -x <token> piastrelle check_unrank -a n=3 -a input_rank=2 -a right_tiling="[--][]"
```

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

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