Count, rank e unrank di matrici binarie monotone (`monotone_01_matrix`)

Guardiamo alle *matrici binarie monotone sia sulle righe che sulle colonne*, come questa:

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

Questa matrice binaria appartiene a $\mathcal{M}_{(4,5,11)}$ in quanto ha $m = 4$ righe, $n = 5$ colonne, somma $s = 11$, e rispetta le seguenti due monotonie:

monotonia di riga, se $M[i, j] = 1$ e $j > 0$ allora $M[i, j-1] = 1$;
monotonia di colonna, se $M[i, j] = 1$ e $i > 0$ allora $M[i-1, j] = 1$.

Più in generale, per ogni tripla $t = (m, n, s) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ con $m, n > 0$, indichiamo con $\mathcal{M}_t$ l'insieme delle matrici binarie $m \times n$, con s valori '1', che rispettano le monotonie di riga e di colonna.

Su $\mathcal{M}_t$ vige un'ordinamento totale e chiamiamo *rango* di una matrice $M \in \mathcal{M}_t$ la sua posizione in tale ordine. Ad esempio, per $t = (3, 4, 5)$:

```
M           rango    rappresentazione a riga
1 1 1 1
1 0 0 0     0        1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0

1 1 1 0
1 1 0 0     1        1 1 1 0 1 1 0 0 0 0 0 0
0 0 0 0

1 1 1 0
1 0 0 0     2        1 1 1 0 1 0 0 0 1 0 0 0
1 0 0 0

1 1 0 0
1 1 0 0     3        1 1 0 0 1 1 0 0 1 0 0 0
1 0 0 0
```

Se ti restano dubbi sull'ordinamento inteso per le matrici in $\mathcal{M}_t$ per altre triple (m, n, s) puoi interrogare il servizio `list` offerto per questo problema `TALight` fornendo in input una tua tripla (m, n, s) valida.

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

- **counting** data una tripla $t = (m, n, s) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ con $m, n > 0$, calcolare il numero $f(t) = f(m, n, s)$ di matrici monotone in $\mathcal{M}_t$.
- **ranking** date t e $M \in \mathcal{M}_t$, restituire il rango r di M in $\mathcal{M}_t$, ossia l'intero nell'intervallo $[0, f(t) - 1]$ che specifica la posizione di M nel nostro ordinamento (si parte da 0).
- **unranking** date la tripla t e l'intero r, con $r \in [0, f(t) - 1]$, restituire la matrice M in $\mathcal{M}_t$ che, partendo da 0, appare in posizione r entro il nostro ordinamento su $\mathcal{M}_t$.

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

Assunzioni

1. quando ti chiediamo di computare $f(t)$, in realtà, se vorrai, potrai limitarti a restituire il resto della divisione di $f(t)$ per $1.000.000.007$. Con questo obiettivo puoi acquisire maggiore efficienza che,

anche in dipendenza del linguaggio di programmazione utilizzato, potrebbe esserti necessaria per risolvere le istanze più grandi.

2. quando ti chiediamo di fare `rank`, è nostra cura fornirti in input una $M \in \mathcal{M}_t$ 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 $1.000.000.007$, $f(t)$ potrebbe essere maggiore, quindi, se utilizzi i numeri modulo $1.000.000.007$ dovresti ricordare quantomeno se il vero valore di $f(t)$ 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 stessa tripla $t = (m, n, s)$. Per ogni istanza, la prima riga contiene sei numeri interi separati da uno spazio: m, n, s, c, r ed u, dove i primi tre numeri specificano la tripla t, il numero $c \in \{0, 1\}$ indica se si richiede il valore di $f(t)$, 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 matrice M_i di $\mathcal{M}_t$. (Si utilizza la rappresentazione a riga ottenuta concatenando le righe della matrice come illustrato sopra.) Seguono infine u righe, l'i-esima delle quali contiene un numero intero $r_i \in [0, f(t) - 1]$.

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 $f(t)$ oppure $f(t) \% \ 1.000.000.007$. 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 $[0, f(t) - 1]$ che vuole essere il rango di M_i, ossia dell'i-esima matrice di $\mathcal{M}_t$ ricevuta in input per questa istanza.
3. seguono $u \geq 0$ righe l'i-esima delle quali contiene la matrice di rango r_i entro $\mathcal{M}_t$, 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
 3 4 5 1 0 0
 3 4 5 0 4 0
  1 1 1 0 1 0 0 0 1 0 0 0
  1 1 1 1 1 0 0 0 0 0 0 0
  1 1 1 1 1 0 0 0 0 0 0 0
  1 1 0 0 1 1 0 0 1 0 0 0
 3 4 5 0 0 4
  2
  0
  1
  3
```

Output su `stdout`

```
4
2
0
1
3
1 1 1 0 1 0 0 0 1 0 0 0
1 1 1 1 1 0 0 0 0 0 0 0
1 1 1 0 1 0 0 0 1 0 0 0
1 1 0 0 1 1 0 0 1 0 0 0
```

In tutti e tre i testcase si lavora con la tripla $(m, n, s) = (3, 4, 5)$. 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. **[10 istanze] `small_c`:** $m \leq 5, n \leq 5, c = 1, r = 0, u = 0$
3. **[10 istanze] `small_r`:** $m \leq 5, n \leq 5, c = 0, u = 0, r \leq 20$
4. **[10 istanze] `small_u`:** $m \leq 5, n \leq 5, c = 0, r = 0, u \leq 20$
5. **[10 istanze] `medium_c`:** $m \leq 10, n \leq 10, c = 1, r = 0, u = 0$
6. **[10 istanze] `medium_r`:** $m \leq 10, n \leq 10, c = 0, u = 0, r \leq 50$
7. **[10 istanze] `medium_u`:** $m \leq 10, n \leq 10, c = 0, r = 0, u \leq 50$
8. **[10 istanze] `big_c`:** $m \leq 50, n \leq 50, c = 1, r = 0, u = 0$
9. **[10 istanze] `big_r`:** $m \leq 50, n \leq 50, c = 0, u = 0, r \leq 200$
10. **[10 istanze] `big_u`:** $m \leq 50, n \leq 50, c = 0, r = 0, u \leq 200$

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_c
   monotone_01_matrix  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `small_c`, `small_r`, `small_u`, `medium_c`.

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

Servizi TALight a tua disposizione

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

```
rtal -s <URL>  connect -x <token> monotone_01_matrix list  -a m=3 -a n=3 -a s=6
```

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

```
rtal -s <URL>  connect -x <token> monotone_01_matrix check_count  -a m=3 -a n=3 -a
s=6 -a risp=5
```

```
rtal -s <URL>  connect -x <token> monotone_01_matrix check_rank  -a m=3 -a n=3 -a
s=6 -a input_matrix="1 3" -a right_rank=8
```

```
rtal -s <URL>  connect -x <token> monotone_01_matrix check_unrank -a input_rank=3
-a m=3 -a n=3 -a s=6 -a input_rank=2 -a right_matrix="2 4 5"
```

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

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