Dipingere i muri (`muro` [72 punti])

Questo problema è preso dalla fase territoriale delle Oii (Olimpiadi Italiane di Informatica).

Dobbiamo gestire un array A di n celle. Ci arriva una sequenza di q colori tutti diversi, e, per ogni colore $c = 1, ..., q$, ci viene anche fornito un numero intero L_i nell'intervallo $[0, n]$. Dobbiamo gestire il seguente processo:

per $i = 1, ..., q$ dobbiamo scegliere $L - i$ celle consecutive in A e colorarle col colore i (nota: il colore nuovo sovrascrive quello vecchio)

Il tuo obiettivo è massimizzare il numero di colori diversi presenti in A a fine processo.

Con quanta efficienza sai rispondere alle seguenti domande?

[**opt_unconstrained**]: quale è il massimo numero di colori che possono apparire in A a fine processo se fosse consentito di fuoriuscire da A?

[**opt_sort**]: quale è il massimo numero di colori che possono apparire in A a fine processo se fosse consentito di scegliere l'ordine dei colori?

[**opt**]: quale è il massimo numero di colori che possono apparire a fine processo?

[**opt_running**]: per $i = 1, ..., q$, quale è il massimo numero di colori che possono apparire se il processo si arresta dopo aver gestito il colore i-esimo?

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 è descritta in 5 righe, di cui solo le prime 3 vengono trasmesse immediatamente. La prima contiene i due numeri n e q nell'ordine e separati da uno spazio. La seconda riga contiene i q valori L_i, $i = 1, ..., q$, sempre separati da spazio e in questo ordine.

Output

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

[**goal 1 (opt_unconstrained)**]: la prima riga contiene il numero massimo di colori che possono apparire in A a fine processo qualora agli intervalli dei vari colori fosse consentito di fuoriuscire in tutto o in parte da A.

[**goal 2 (opt_sort)**]: la seconda riga contiene il numero massimo di colori che possono apparire in A a fine processo qualora fosse consentito di giocarsi i colori in un ordine qualsiasi.

[**goal 3 (opt)**]: la terza riga contiene il numero massimo di colori che possono apparire in A a fine processo (domanda originale alle Oii).

[**goal 4 (opt_running)**]: la quarta riga riporta, nell'ordine e saparati da spazi, q numeri $X_1, ..., X_q$ dove X_i è il massimo numero di colori che possano apparire in A dopo aver collocato il colore i-esimo.

Assunzioni

1. $1 \leq n \leq 100,000$, $1 \leq q \leq 100,000$

Esempio di Input/Output

Input da `stdin`

```
3
2 3
1 1 1
10 8
1 10 10 8 6 6 3 1
8 8
3 2 5 3 2 7 4 4
```

Output su `stdout`

```
2 2 2
1 2 2
8 7 6
1 1 1 2 3 4 5 6
8 7 4
1 2 3 4 5 2 3 4
```

Spiegazione: Nel primo caso d'esempio l'array A ha 2 celle e ci sono 3 colori con $L_1 = L_2 = L_3 = 1$. Si consideri il seguente processo:

1. colora 1 la secondo cella di A (al che si vedrà il colore 1)
2. colora 2 la prima cella di A (al che si vedranno i colori 1 e 2)
3. colora 3 la seconda cella di A (al che si vedranno i colori 2 e 3)

Tale processo non si è preso la libertà di fuoriuscire da A nè quella di applicare i colori in un ordine diverso dalla loro numerazione. E tuttavia è ottimo sia in quanto `opt_unconstrained=opt_sort=opt=2` sia in quanto la sequenza di valori da restituire per goal 4 è proprio 1, 2, 2.

Nel secondo e terzo caso d'esempio i colori sono 8, da spalmare su di un muro di lunghezza 10 od 8, rispettivamente. Si controlli che i valori riportati in output corrispondono alla propria comprensione del testo.

Subtask

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

I testcase sono raggruppati nei seguenti subtask.

1. **[0 pts← 3 istanze da 0 + 0 + 0 + 0 punti] `esempi_testo`:** i quattro esempi del testo
2. **[8 pts← 4 istanze da 0 + 0 + 1 + 1 punti] `tiny`:** $n \leq 10, q \leq 10$
3. **[8 pts← 4 istanze da 0 + 0 + 1 + 1 punti] `small_fixed_width`:** $n \leq 30, q \leq 30$, tutti i colori di una stessa lunghezza
4. **[8 pts← 4 istanze da 0 + 0 + 1 + 1 punti] `small_decreasing`:** $n \leq 30, q \leq 30$, monotonia $L_{i+1} \leq L_i$
5. **[8 pts← 4 istanze da 0 + 0 + 1 + 1 punti] `small`:** $n \leq 30, q \leq 30$
6. **[12 pts← 4 istanze da 1 + 0 + 1 + 1 punti] `medium`:** $n \leq 100, q \leq 100$
7. **[12 pts← 4 istanze da 0 + 1 + 1 + 1 punti] `big`:** $n \leq 1000, q \leq 1000$
8. **[16 pts← 4 istanze da 1 + 1 + 1 + 1 punti] `large`:** $n \leq 100,000, q \leq 100,000$

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 -a size=small_decreasing muro -- <MY_SOLUTION>
```

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `tiny`, `small_fixed_width`, `small_decreasing`.

Il valore di default per l'argomento `size` è `large` 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/algo2025`

Nel template di riga di comando quì sopra `<MY_SOLUTION>` è una qualsiasi scrittura che ove immessa anche da sola al prompt della CLI comporti l'avvio del solver da tè realizzato. Solo alcuni esempi:

- `./a.out` per un compilato da `C/c++`, eventualmente seguito dagli argomenti che prevede
- `./my_solution.py arg1 arg2 ...` se il tuo file `my_solution.py` col codice python ha i permessi di esecuzione e inizia con la riga di shebang
- `python my_solution.py` o `python3 my_solution.py` per far eseguire il tuo script da un interprete python

Se vuoi che una tua sottomissione venga conteggiata ai fini di un esame o homework devi allegare il sorgente della tua soluzione con `-fsource=<FILENAME>` (ad esempio `-fsource=my_solution.py` subito a valle del comando `connect`. Inoltre devi aver precedentemente affettuato il login tramite credenziali GIA (lancia prima `rtal -s <URL> login` e segui le istruzioni per impostare il sign-on).

Il comando `rtal` prevede diversi parametri, consigliamo di esplorarne e sperimentarne le potenzialità ed opzioni d'uso. Un tutorial all'uso di `rtal` è esposto alla pagina:

`https://github.com/romeorizzi/AlgoritmiUniVR/tree/main/strumenti`