Rovescia prefisso come comanda (solitario di Napoleone) (`prefixflip_solitaire` [75 punti])

Vuoi ammazzare il tempo? Eccoti un solitario senza-pensieri della tipologia "solitari di Napoleone" (che per fare fronte alle sue lunghe notti di insonnia ne aveva inventati diversi, alcuni tuttora in voga). Si gioca come segue: prendi una permutazione random (come un mazzo di carte da riordinare), ad esempio:

4	8	7	3	2	1	5	6	9

L'elemento in testa (il 4) comanda di voler andare a casa sua (in posizione 4), e per farlo rovescia l'intero prefisso di quelli sulla sua strada. Si ottiene così la seguente permutazione che diviene lo stato attuale dopo il primo passo:

3	7	8	4	2	1	5	6	9

Nella prossima mossa si rovescia il prefisso dei primi 3 elementi (comanda sempre l'elemento attualmente in testa, e si fà portare a casa col rovesciamento di un prefisso), pervenendo al prossimo stato, e da lì così via di seguito di stato in stato:

8	7	3	4	2	1	5	$\underline{6}$	9
6	5	1	2	4	$\underline{3}$	7	8	9
3	4	$\underline{2}$	1	5	6	7	8	9
2	$\underline{4}$	3	1	5	6	7	8	9
4	2	3	$\underline{1}$	5	6	7	8	9
$\underline{1}$	3	2	4	5	6	7	8	9

A questo punto il gioco termina, bloccato dal fatto che quando l'1 è in testa il capo è contento. Questa volta abbiamo perso perchè l'obiettivo di riordinare l'intero mazzo di carte durante il passatempo è fallito (il 2 e il 3 non sono ancora a casa)! Pazienza, sarà per un'altra volta e i solitari di Napoleone più apprezzati sono proprio quelli più rari a riuscire, meglio poi se imprevedibili fino quasi alla fine.

Ad ogni solitario di Napoleone che si rispetti è però richiesto di terminare *sempre* entro un tempo finito, perchè domani all'alba ci attende un'altra battaglia, chissà come andrà anche quella coi suoi morti e i suoi feriti e invalidi.

Competenza da esibire: dimostrare che il solitario ha sempre termine, da qualsiasi permutazione prenda avvio. La tecnica è di progettare una monovariante, ossia una funzione che associ un numero naturale ad ogni stato e con la proprietà che se lo stato s' è lo stato che viene prodotto con una singola mossa dallo stato s allora $f(s') < f(s)$. Implementa una tale funzione f dentro al template di soluzione (fornito per i linguaggi `python` e `C`/`c++`).

Goals: oltre alla velocità con cui ti è possibile computare la f (sulle istanze non piccole non farai in tempo a simulare la partita che parte dallo stato dato), la qualità di una monovariante stà anche nell'approssimare da vicino, se vista come upperbound, il numero delle mosse restanti. Abbiamo previsto i seguenti quattro *goals*:

`any` una qualsiasi monovariante f va bene (ovviamente il codominio di una mono-variante deve essere sui naturali altrimenti non dimostra nulla).

$n!$ è noto che $n!$ è il numero di permutazioni di n oggetti. Forse è persino difficile inventarsi una mono-variante che produca numeri oltre $n!$, ma preferiamo comunque distinguere e premiare mono-varianti con questa qualità.

2^n riesce ancora naturale produrre una monovariante f tale che $f(s) \leq 2^n$ per ogni permutazione s dei numeri in $\{1, 2, ..., n\}$.

`fibonacci` entrando in argomentazioni combinatoriche, Knuth dimostrò che su nessuna permutazione di $\{1, 2, ..., n\}$ il solitario esegue più di $F(n)$ passi, dove $F(n)$ è l'n-esimo numero di Fibonacci. Pose come problema (credo tuttora aperto) di fornire un'upper bound polinomiale (lavorando sperimentalmente, aveva avuto l'impressione che l'ordine asintotico di crescita della partita più lunga fosse $O(n^2)$). Non credo nemmeno che sia stata proposta una monovariante che cresca al più come $F(n)$, ma è presumibile ci possa essere.

Input e Output

In realtà, se è fornito un template per il tuo linguaggio, il tuo lavoro potrà limitarsi alla sola implementazione della funzione $f()$ e non devi davvero preoccuparti dei dettagli dei formati di input e output e del protocollo di comunicazione. Ma, se già non lo conosci, incontri quì sotto il concetto di *batch*.

Input

Si legga l'input da `stdin`. La prima riga contiene B, il numero totale di batch come presi su tutti i subtask selezionati. Un batch è un blocco di testcase (istanze) consecutive. Il subtask è superato se le risposte a tutte le sue istanze incontrano l'approvazine del valutatore e ogni suo batch è evaso entro il timelimit inteso come compessivo per l'intero batch. Ogni batch comincia scrivendo il numero T delle proprie istanze su una prima riga, sequita da altre T righe ciscuna contenete una diversa istanza del batch. Ogni istanza è codificata in due righe: la prima contiene un numero naturale n e la seconda una permutazione π dei numeri 1, 2, ..., n, ossia gli stessi, scritti in un qualche ordine, e separati da spazi.

Output

Per ciascuna istanza, prima di leggere l'istanza successiva, scrivi su `stdout` una riga contenente il numero naturale $f(\pi)$.

Subtask e batch del subtask

I testcase sono raggruppati nei seguenti subtask. Nella descrizione dei subtask è specificato quanti punti vengano ottenuti rispondendo coerentemente a tutte le istanze di quel subtask, come anche in funzione di quali goals sono stati raggiunti. Tali punti vengono assegnati purchè ogni intero batch di istanze del subtask venga gestito entro il timelimit. Il timelimit è sempre di 1 secondo.

1. **[3 pts← 3 istanze da 1 + 0 + 0 + 0 punti]** **`esempi_testo`:** i cinque esempi del testo
2. **[20 pts← 5 istanze da 1 + 1 + 1 + 1 punti]** **`small`:** $N \leq 10$
3. **[20 pts← 5 istanze da 1 + 1 + 1 + 1 punti]** **`medium`:** $N \leq 100$
4. **[12 pts← 3 istanze da 1 + 1 + 1 + 1 punti]** **`big`:** $N \leq 1000$
5. **[12 pts← 3 istanze da 1 + 1 + 1 + 1 punti]** **`large`:** $N \leq 10.000$
6. **[8 pts← 2 istanze da 1 + 1 + 1 + 1 punti]** **`extra_large`:** $N \leq 100.000$

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 -x <token> -a size=medium
 prefixflip_solitaire  -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

`esempi_testo`, `small`, `medium`.

Il valore di default per l'argomento `size` è `extra_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/algo`