Un primo esempio di Programmazione Dinamica, su linea/array (`prima_PD_su_linea` [104 punti])

Di n oggetti in fila indiana, ognuno con un suo valore, scegli quali prendere volendo massimizzare il valore totale degli oggetti presi ma senza prendere due oggetti adiacenti (consecutivi nella fila) o che sono adiacenti ad uno stesso oggetto.

Formulazione: E' dato un vettore A di n celle con indici in $[0, n) := \{0, 1, ..., n-1\}$. Ogni sua cella $A_i, i \in [0, n)$, contiene un intero nell'intervallo $[0, 100)$. Tra i sottoinsiemi S di $[0, n)$ per cui $i, j \in S$ implichi $|j - i| \geq 3$ dovete produrne uno che massimizzi $\sum_{i \in S} A_i$.

Goals: Sai computare il valore ottimo per questo problema di ottimizzazione? Sai restituire una soluzione ottima (lista degli indici degli elementi da prendere)? Sai dire quante siano le soluzioni ottime e quelle ammissibili?

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 presenta un diverso array A da affrontare. Ogni istanza è descritta in due righe: la prima riga contiene la lunghezza n dell'array (il numero delle sue celle), la seconda riga specifica nell'ordine i valori contenuti nelle celle di A (n numeri separati da spazi).

Output

Per ciascuna istanza, prima di leggere l'istanza successiva, scrivi su `stdout` il tuo output strutturato anche in funzione dei seguenti 4 goals:

count_feas_sols la prima riga ritorna il numero $f(n)$ delle *soluzioni ammissibili*. Una *soluzione* S è un qualsiasi sottoinsieme di $[0, n)$, ed è considerata *ammissibile* quando per ogni $i, j \in S$ con $j > i$ valga $j \geq i + 3$ (ossia, tra ogni due celle di A che vengano scelte devono essere presenti almeno due celle non scelte). A dire il vero, per evitare problemi di overflow ($f(n)$ ha crescita esponenziale in n), chiediamo più precisamente di restituire $f(n)\%1,000,000,007$, ossia il resto della divisione del dividendo $f(n)$ per il divisore $1,000,000,007$. (Nel gergo dell'aritmetica modulare si è soliti chiamarlo "$f(n)$ modulo $1,000,000,007$".)

optval la seconda riga contiene il *valore ottimo* (=valore di una soluzione ottima), ossia il massimo valore possibile per la somma dei valori selezionati in una soluzione ammissibile.

optsol la terza riga contiene una *soluzione ottima*, ossia una soluzione ammissibile di valore ottimo.

count_opt_sols la quarta riga ritorna il numero delle soluzioni ottime modulo $1,000,000,007$. (Anche il numero delle sole soluzioni ottime potrebbe essere enorme, a seconda dell'istanza, raggiungendo ad esempio $f(n)$ quando tutti i valori in A fossero nulli.

Nella descrizione dei subtask è specificato quanti punti si acquisiscono su ciascuna istanza di quel subtask, vuoi per la correttezza del valore ottimo riportato nella prima riga, vuoi per la correttezza della soluzione ottima nella seconda riga, vuoi per la correttezza del valore $f(n)$ nella terza riga, o per quella del valore nella quarta ed ultima riga del tuo output per quell'istanza (testcase).

Esempio di Input/Output

Input da `stdin`

```
3
4
3 5 0 1
6
0 0 0 0 0 0
6
1 2 3 4 5 6
```

Output su `stdout`

```
6
5
1
1
13
0

13
13
9
2 5
1
```

Spiegazione: il valore ottimo per la prima istanza è 5, e l'unica soluzione di valore 5 consiste nel selezionare il solo elemento $A[1] = 5$. Nella seconda istanza tutte le 13 soluzioni ammissibili sono ottime, incluso quella che non seleziona alcun elemento di A. Nella terza istanza la soluzione ottima è di nuovo unica.

Subtask

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

I testcase sono raggruppati nei seguenti subtask.

1. **[6 pts← 3 istanze da 1 + 1 + 0 + 0 punti] `esempi_testo`:** i tre 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. **[21 pts← 3 istanze da 1 + 2 + 2 + 2 punti] `big`:** $N \leq 200$
5. **[15 pts← 3 istanze da 1 + 2 + 1 + 1 punti] `large`:** $N \leq 3000$
6. **[10 pts← 1 istanza da 1 + 4 + 4 + 1 punti] `all_equal`:** tutti gli oggetti hanno lo stesso valore, $N \leq 3000$
7. **[12 pts← 2 istanze da 2 + 2 + 1 + 1 punti] `extra_large`:** $N \leq 30000$

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
   prima_PD_su_linea  -- 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`