Esempio di Programmazione Dinamica non-locale su linea/array (`prima_PD_non_locale_su_linea` [94 punti])

Sono dati due array A e C, entrambi di n celle che ospitano numeri interi. Per ogni indice $i \in [0, n)$, $A[i] \in [0, 100)$ e $C[i] \in [0, n)$. Volete selezionare un sottoinsieme S di $[0, n)$ in modo da massimizzare $\sum_{i \in S} A[i]$. Tuttavia, per nessuna coppia i, j con $0 \leq i < j < n$, vi è consentito riporre in S sia la celle $A[i]$ che la cella $A[j]$ se $j - i \leq C[i] + C[j]$.

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. Ogni istanza è descritta in tre righe: la prima riga contiene la lunghezza n dei due array, la seconda riga specifica nell'ordine i valori contenuti nelle celle di A (n numeri separati da spazi), la terza riga specifica nell'ordine i valori contenuti nelle celle di C (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 è un qualsiasi sottoinsieme S dell'insieme dei numeri interi nell'intervallo semi-aperto $[0, n)$, ed è considerata ammissibile quando per ogni due $i, j \in S$ valga che $C[i] + C[j] \leq |i - j|$. 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 la massima somma possibile dei valori in una soluzione ammissibile.

optsol la terza riga contiene una soluzione ottima, ossia una soluzione ammissibile di valore ottimo. Un modo non-ambiguo e pratico di fornire una soluzione è di specificare la lista ordinata degli indici delle celle che include.

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

Output su `stdout`

```
5
5
1
1
11
0

11
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 11 soluzioni ammissibili sono ottime, incluso quella che non seleziona alcun elemento di A. Oltre all'insieme vuoto, è ammissibile selezionare ciascun singolo elemento (6 ulteriori soluzioni), e sono infine ammissibili le seguenti selezioni: $\{0, 4\}, \{0, 5\}, \{1, 4\}, \{1, 5\}$. Nella terza istanza la soluzione ottima è di nuovo unica e totalizza $A[2] + A[5] = 3 + 6 = 9$. Delle 13 soluzioni ammissibili una è l'insieme vuoto, che ha valore nullo, 6 constano di un singolo elemento, e 6 constano di precisamente due elementi: $\{0, 3\}, \{0, 4\}, \{0, 5\}, \{1, 4\}, \{1, 5\}, \{2, 5\}$.

Subtask

Il tempo limite per istanza (ossia per ciascun testcase) è sempre di 3 secondi.

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. **[28 pts← 4 istanze da 1 + 2 + 2 + 2 punti] `big`:** $N \leq 200$
5. **[20 pts← 4 istanze da 1 + 2 + 1 + 1 punti] `large`:** $N \leq 1000$

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_non_locale_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 `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`