

Genera il prossimo (prev_and_next [78 punti])

La figura intende precisare un ordine inteso per tre diverse famiglie parametriche di oggetti:

$C(k, B)$ le B^k configurazioni di un contatore di k cifre e base B (per $k = 3$ e $B = 2$)

$T(n)$ i possibili tiling di una griglia $1 \times n$ tramite tiles 1×1 e 1×2 (per $n = 5$)

$F(n)$ le formule ben formate a n coppie di parentesi (esemplificato per valore del parametro $n = 4$)

0.==>	0 0 0	[] [] [] [] []	() () () ()
1.==>	0 0 1	[] [] [] [- -]	() () (())
2.==>	0 1 0	[] [] [- -] []	() (()) ()
3.==>	0 1 1	[] [- -] [] []	() (()) ()
4.==>	1 0 0	[] [- -] [- -]	() ((()))
5.==>	1 0 1	[- -] [] [] []	(()) () ()
6.==>	1 1 0	[- -] [] [- -]	(()) (())
7.==>	1 1 1	[- -] [- -] []	(()) (())
8.==>			((())) ()
9.==>			(()) (())
10.==>			(()) (())
11.==>			((())) ()
12.==>			((()))
13.==>			(((())))

Ricevi in input un oggetto di una delle tre famiglie e devi restituire l'oggetto che lo segue e quello che lo precede, secondo l'ordine inteso, nella sua stessa famiglia di pari parametri.

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 un singolo oggetto di $F(n)$, oppure di $C(k, B)$, oppure di $T(n)$ codificato in una singola riga di testo come visto sopra.

Nota: nel caso di un counter, assumi che il valore inespresso del parametro B sia dato dal valore della massima cifra che appare nel counter assegnato, incrementata di 1.

Output

Per ciascuna istanza, prima di leggere l'istanza successiva, scrivi su `stdout` due righe di testo:

- + la prima riga contiene l'oggetto successivo, oppure è vuota se l'oggetto ricevuto in input era il primo tra i suoi simili (stessa famiglia e stesso valore dei parametri)
- + la seconda riga contiene l'oggetto precedente, oppure è vuota se l'oggetto ricevuto in input era l'ultimo tra i suoi simili

Esempio di Input/Output

Input da `stdin`

```
7
[][][][]
((( )))
0 1 1
5 0 1 1
[][]
((( )))(( ))()
[] [--][][] [--][][][] [--][][][]
```

Output su `stdout`

```
[] [] [] [--]
((( )))

0 1 0
1 0 0
5 0 1 0
5 0 1 2

[ --]
((( )))(( ))()
((( )))(( ))()
[] [--][][] [--][][][] [--] [--]
[] [--][][] [--][][][] [--] [] [--]
```

Subtask

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

I testcase sono raggruppati nei seguenti subtask.

1. [14 pts ← 7 istanze da 1 + 1 punti] **esempi_testo**: i sette esempi del testo
2. [6 pts ← 3 istanze da 1 + 1 punti] **smallC**: k-digits counter with $k \leq 10$
3. [8 pts ← 4 istanze da 1 + 1 punti] **mediumC**: k-digits counter with $k \leq 500$
4. [6 pts ← 3 istanze da 1 + 1 punti] **bigC**: k-digits counter with $k \leq 100.000$
5. [8 pts ← 4 istanze da 1 + 1 punti] **smallT**: tiling of a $1 \times n$ grid with $n \leq 10$
6. [6 pts ← 3 istanze da 1 + 1 punti] **mediumT**: tiling of a $1 \times n$ grid with $n \leq 500$
7. [8 pts ← 4 istanze da 1 + 1 punti] **bigT**: tiling of a $1 \times n$ grid with $n \leq 100.000$
8. [8 pts ← 4 istanze da 1 + 1 punti] **smallF**: parenthesis formula with $n \leq 10$
9. [6 pts ← 3 istanze da 1 + 1 punti] **mediumF**: parenthesis formula with $n \leq 500$
10. [8 pts ← 4 istanze da 1 + 1 punti] **bigF**: parenthesis formula with $n \leq 10.000$

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=smallT
prev_and_next -- python my_solution.py
```

vengono valutati, nell'ordine, i subtask:

esempi_testo, smallC, mediumC, bigC, smallT.

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

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

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