

# Conectando superárboles (supertrees)

Los Jardines de la Bahía es un gran parque natural en Singapur. En el parque se encuentran  $n$  torres, también conocidas como superárboles. Estas torres están numeradas  $0$  a  $n - 1$ . Quisiéramos construir un conjunto de **cero o más** puentes. Cada puente conecta a un par de torres distintas y puede ser recorrido en **cualquier** dirección. Dos puentes no conectarán al mismo par de torres.

Un camino desde la torre  $x$  a la torre  $y$  es una secuencia de una o más torres tal que:

- el primer elemento de la secuencia es  $x$ ,
- el último elemento de la secuencia es  $y$ ,
- todos los elementos de la secuencia son **distintos**, y
- cada par de elementos (torres) consecutivos en la secuencia está conectado entre sí por un puente.

Fíjate que, por definición, hay exactamente un camino desde una torre a ella misma y que el número de caminos distintos desde la torre  $i$  a la torre  $j$  es el mismo que desde la torre  $j$  a la torre  $i$ .

El arquitecto encargado del diseño desea que los puentes se construyan de forma tal que para todos los  $0 \leq i, j \leq n - 1$ , hayan exactamente  $p[i][j]$  caminos distintos desde la torre  $i$  a la torre  $j$ , cumpliéndose que  $0 \leq p[i][j] \leq 3$ .

Construye un conjunto de puentes que satisfagan los requerimientos del arquitecto, o determina que es imposible.

## Detalles de implementación

Implementa el siguiente procedimiento:

```
int construct(int[][] p)
```

- $p$ : un arreglo  $n \times n$  que representa los requerimientos del arquitecto.
- Si una construcción es posible, este procedimiento deberá hacer exactamente una llamada a `build` (leer debajo) para reportar la construcción, y luego deberá retornar `1`.
- De lo contrario, el procedimiento deberá retornar `0`, sin hacer ninguna llamada a `build`.
- Este procedimiento será llamado exactamente una vez.

El procedimiento `build` se define de la siguiente manera:

```
void build(int[][] b)
```

- $b$ : un arreglo  $n \times n$ , donde  $b[i][j] = 1$  si existe un puente que conecte a las torres  $i$  y  $j$ , o  $b[i][j] = 0$  de lo contrario.
- Fíjate que el arreglo debe satisfacer  $b[i][j] = b[j][i]$  para todas las  $0 \leq i, j \leq n - 1$  y  $b[i][i] = 0$  para todas las  $0 \leq i \leq n - 1$ .

## Ejemplos

### Ejemplo 1

Considera la siguiente llamada:

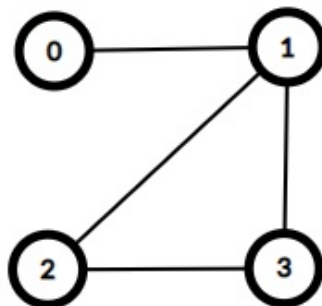
```
construct([[1, 1, 2, 2], [1, 1, 2, 2], [2, 2, 1, 2], [2, 2, 2, 1]])
```

Esto significa que debe haber exactamente un camino desde la torre 0 a la torre 1. Para todos los otros pares de torres  $(x, y)$ , tal que  $0 \leq x < y \leq 3$ , deben haber exactamente dos caminos desde la torre  $x$  a la torre  $y$ .

Esto se puede conseguir con 4 puentes, conectando los siguientes pares de torres:  $(0, 1)$ ,  $(1, 2)$ ,  $(1, 3)$  y  $(2, 3)$ .

Para reportar esta solución, el procedimiento `construct` debe hacer la siguiente llamada:

- `build([[0, 1, 0, 0], [1, 0, 1, 1], [0, 1, 0, 1], [0, 1, 1, 0]])`



Por último, deberá retornar 1.

En este caso, hay múltiples construcciones que cumplen con los requerimientos y todas se considerarían correctas.

### Ejemplo 2

Considera la siguiente llamada:

```
construct([[1, 0], [0, 1]])
```

Esto significa que no debería haber forma de viajar entre las dos torres. La única manera de satisfacer esto es no teniendo puentes.

Por tanto, el procedimiento `construct` debería hacer la siguiente llamada:

- `build([[0, 0], [0, 0]])`

Luego de esto, `construct` debería retornar 1.

### Ejemplo 3

Considere la siguiente llamada:

```
construct([[1, 3], [3, 1]])
```

Esto significa que deben haber exactamente 3 caminos desde la torre 0 a la torre 1. Este conjunto de requisitos no se puede satisfacer. Como tal, el procedimiento `construct` debería devolver 0 sin hacer ninguna llamada a `build`.

## Restricciones

- $1 \leq n \leq 1000$
- $p[i][i] = 1$  (para toda  $0 \leq i \leq n - 1$ )
- $p[i][j] = p[j][i]$  (para toda  $0 \leq i, j \leq n - 1$ )
- $0 \leq p[i][j] \leq 3$  (para toda  $0 \leq i, j \leq n - 1$ )

## Sub-tareas

1. (11 points)  $p[i][j] = 1$  (para todas las  $0 \leq i, j \leq n - 1$ )
2. (10 points)  $p[i][j] = 0$  or 1 (para todas las  $0 \leq i, j \leq n - 1$ )
3. (19 points)  $p[i][j] = 0$  or 2 (para todas las  $i \neq j, 0 \leq i, j \leq n - 1$ )
4. (35 points)  $0 \leq p[i][j] \leq 2$  (para todas las  $0 \leq i, j \leq n - 1$ ) y hay al menos una construcción que satisface los requisitos.
5. (21 points)  $0 \leq p[i][j] \leq 2$  (para todas las  $0 \leq i, j \leq n - 1$ )
6. (4 points) Sin restricciones adicionales.

## Evaludor Local

El evaluador local lee la entrada en el siguiente formato:

- línea 1:  $n$
- línea  $2 + i$  ( $0 \leq i \leq n - 1$ ):  $p[i][0] \ p[i][1] \ \dots \ p[i][n - 1]$

La salida del evaluador local se presenta en el siguiente formato:

- línea 1: el valor que `construct` retorna.

Si el valor de retorno de `construct` es 1, el evaluador local además imprime:

- línea  $2 + i$  ( $0 \leq i \leq n - 1$ ):  $b[i][0] \ b[i][1] \ \dots \ b[i][n - 1]$