# Łączenie superdrzew (supertrees)

Gardens by the Bay jest wielkim parkiem natury w Singapurze. W parku znajduje się $n$ wież, zwanych superdrzewami. Wieże te ponumerowane są liczbami naturalnymi od $0$ do $n-1$. Chcielibyśmy teraz stworzyć **zero lub więcej** mostów. Każdy most łączy parę różnych wież i jest **dwukierunkowy**. Żadne dwa mosty nie mogą łączyć tej samej pary wież.

Ścieżka z wieży $x$ do wieży $y$ to ciąg złożony z jednej lub więcej wież spełniający następujące warunki:

- Pierwszy element tego ciągu to $x$.
- Ostatni element tego ciągu to $y$.
- Wszystkie elementy tego ciągu są **różne**.
- Każde dwa kolejne elementy ciągu odpowiadają wieżom połączonym mostem.

Z definicji wynika, że istnieje dokładnie jedna ścieżka z dowolnej wieży do niej samej. Ponadto, liczba różnych ścieżek z wieży $i$ do wieży $j$ jest taka sama, jak liczba różnych ścieżek z wieży $j$ do wieży $i$.

Główny architekt odpowiedzialny za projekt chciałby zbudować mosty w taki sposób, żeby dla wszystkich par wież $i, j$, takich, że $0 \leq i, j \leq n-1$, istniało dokładnie $p[i][j]$ różnych ścieżek z wieży $i$ do wieży $j$, przy czym $0 \leq p[i][j] \leq 3$.

Stwórz układ mostów spełniający wymagania architekta albo stwierdź, że taki układ jest niemożliwy do utworzenia.

## Szczegóły implementacji

Twoim zadaniem jest zaimplementować poniższą funkcję:

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

- $p$: tablica $n \times n$ opisująca wymagania architekta.
- Jeśli konstrukcja mostów jest możliwa, ta funkcja powinna dokładnie raz wywołać procedurę `build` (opisaną poniżej), aby zaproponować konstrukcję, a następnie powinna zwrócić $1$.
- W przeciwnym przypadku, funkcja powinna zwrócić $0$ i nie powinna wywoływać procedury `build`.
- Funkcja zostanie wywołana dokładnie raz.

Procedura `build` jest zdefiniowana w następujący sposób:

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

- $b$: tablica $n \times n$, gdzie $b[i][j] = 1$, jeśli należy zbudować most łączący wieże $i$ oraz $j$ w proponowanym rozwiązaniu, zaś $b[i][j] = 0$ w przeciwnym przypadku.
- Zwróć uwagę na to, że tablica ta musi spełniać $b[i][j] = b[j][i]$ dla każdych $0 \leq i, j \leq n - 1$ oraz $b[i][i] = 0$ dla $0 \leq i \leq n - 1$.

# Przykłady

## Przykład 1

Rozważmy następujące wywołanie funkcji:

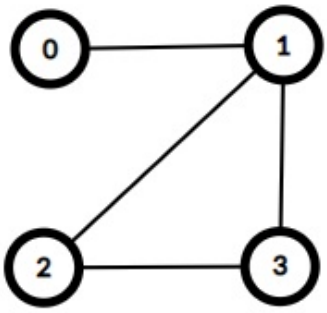
```
construct([[1, 1, 2, 2], [1, 1, 2, 2], [2, 2, 1, 2], [2, 2, 2, 1]])
```

To oznacza, że powinna istnieć dokładnie jedna ścieżka z wieży $0$ do wieży $1$. Dla każdej pozostałej pary wież $(x, y)$, takiej, że $0 \leq x < y \leq 3$, powinny istnieć dokładnie dwie ścieżki z wieży $x$ do wieży $y$.

Ten cel możemy osiągnąć za pomocą $4$ mostów, łączących następujące pary wież: $(0, 1)$, $(1, 2)$, $(1, 3)$ oraz $(2, 3)$.

Aby zaproponować to rozwiązanie, funkcja `construct` powinna wywołać procedurę `build` w następujący sposób:

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




Następnie, funkcja `construct` powinna zwrócić $1$.

W tym przypadku, istnieje wiele konstrukcji spełniających wszystkie wymagania. Każda z nich zostanie uznana za poprawną przez system sprawdzający.

## Przykład 2

Rozważmy następujące wywołanie funkcji:

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

To oznacza, że nie powinno być żadnej ścieżki z jednej wieży do drugiej. Ten cel możemy osiągnąć na jeden sposób: nie budując żadnych mostów.

Tak więc, funkcja `construct` powinna wywołać procedurę `build` w następujący sposób:

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

Następnie, funkcja `construct` powinna zwrócić $1$.

### Przykład 3

Rozważmy następujące wywołanie funkcji:

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

To oznacza, że powinny istnieć dokładnie $3$ ścieżki z wieży $0$ do wieży $1$. Tego zbioru wymagań nie możemy spełnić w żaden sposób. Dlatego też funkcja `construct` powinna zwrócić $0$ i nie powinna wywoływać procedury `build`.

## Ograniczenia

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

## Podzadania

1. (11 punktów) $p[i][j] = 1$ (dla $0 \leq i, j \leq n-1$)
2. (10 punktów) $p[i][j] = 0$ or $1$ (dla $0 \leq i, j \leq n-1$)
3. (19 punktów) $p[i][j] = 0$ or $2$ (dla $i \neq j$, $0 \leq i, j \leq n-1$)
4. (35 punktów) $0 \leq p[i][j] \leq 2$ (dla $0 \leq i, j \leq n-1$) oraz istnieje co najmniej jedna konstrukcja spełniająca wszystkie ograniczenia.
5. (21 punktów) $0 \leq p[i][j] \leq 2$ (dla $0 \leq i, j \leq n-1$)
6. (4 punkty) Brak dodatkowych ograniczeń.

## Przykładowy program oceniający

Przykładowy program oceniający wczytuje wejście w następującym formacie:

- wiersz $1$: $n$
- wiersze $2+i$ ($0 \leq i \leq n-1$): $p[i][0]$ $p[i][1]$ $\ldots$ $p[i][n-1]$

Przykładowy program oceniający wypisuje wyjście w następującym formacie:

- wiersz $1$: wartość zwrócona przez funkcję `construct`.

Jeśli funkcja `construct` zwróciła $1$, przykładowy program oceniający dodatkowo wypisuje poniższe wiersze:

- wiersze $2 + i$ ($0 \le i \le n - 1$): $b[i][0]$ $b[i][1]$ $\ldots$ $b[i][n - 1]$