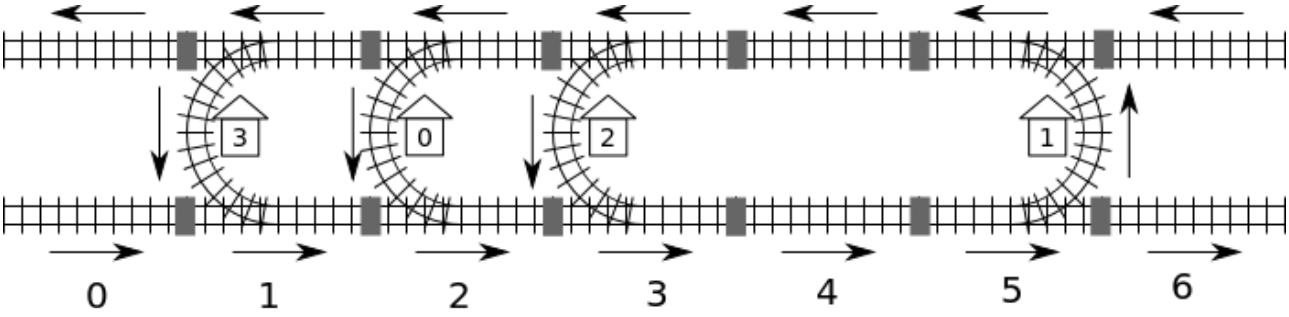
Kolej (Rail)

Na Tajwanie znajduje się długa linia kolejowa łącząca zachodnie i wschodnie wybrzeże wyspy. Linia ta składa się z m bloków. Kolejne bloki są ponumerowane $0, \ldots, m-1$, począwszy od zachodniego końca linii. W każdym bloku znajduje się jednokierunkowy tor biegnący na zachód położony na północy bloku oraz jednokierunkowy tor biegnący na wschód położony na południu bloku. Między torami bloku może znajdować się stacja kolejowa.

Możliwe są trzy typy bloków. Blok typu *C* zawiera stację kolejową, na którą wjeżdża się z toru północnego i którą opuszcza się torem południowym. Blok typu *D* zawiera stację kolejową, na którą wjeżdża się z toru południowego i którą opuszcza się torem północnym. Blok *pusty* nie zawiera stacji kolejowej. Dla przykładu, na poniższym rysunku bloki 0, 4 i 6 są puste, bloki 1, 2 i 3 są typu C, a blok 5 jest typu D. Tory w kolejnych blokach są połączone za pomocą *łączników*; są one przedstawione na rysunku jako zaciemnione prostokąciki.



System kolejowy zawiera n stacji ponumerowanych od 0 do $n-1$. Należy założyć, że za pomocą torów kolejowych *można dotrzeć z każdej stacji do każdej innej stacji*. Przykładowo, aby dotrzeć ze stacji 0 do stacji 2, należy zacząć w bloku 2, następnie przejechać przez bloki 3 i 4, używając torów południowych, później przejechać przez blok 5, a tym samym stację 1, następnie przejechać przez blok 4, używając toru północnego, by na końcu dojechać do stacji 2 w bloku 3.

Jako że może być więcej niż jedna trasa, odległość między dwiema stacjami definiujemy jako *minimalną* liczbę łączników na ścieżce między tymi stacjami. Dla przykładu, najkrótsza ścieżka ze stacji 0 do stacji 2 biegnie blokami 2-3-4-5-4-3 i przechodzi przez 5 łączników, więc odległość między tymi dwiema stacjami jest równa 5.

Struktura systemu kolejowego była przechowywana z użyciem specjalnego oprogramowania. Niestety usterka zasilania spowodowała uszkodzenie danych programu, przez co informacje o położeniach stacji i typach bloków zostały utracone. W pamięci pozostała tylko informacja o numerze bloku, w którym znajduje się stacja 0, oraz informacja, że jest to blok typu C. Szczęśliwie, program wciąż pozwala odpowiadać na zapytania o odległość między dowolną parą stacji. Przykładowo, w programie możemy zapytać: „Jaka jest odległość od stacji 0 do stacji 2" i uzyskać odpowiedź: 5.

Zadanie

Napisz funkcję `findLocation`, która określi dla każdej stacji numer oraz typ bloku, w którym ta

stacja się znajduje.

- `findLocation(n, first, location, stype)`
  - `n`: liczba stacji.
  - `first`: numer bloku, w którym znajduje się stacja 0.
  - `location`: tablica rozmiaru n; `location[i]` powinno na końcu działania funkcji zawierać numer bloku, w którym znajduje się stacja i.
  - `stype`: tablica rozmiaru n; `stype[i]` powinno na końcu działania funkcji zawierać typ bloku, w którym znajduje się stacja i: 1 dla typu C, a 2 dla typu D.

Przy ustalaniu położeń i typów stacji Twoja funkcja może używać funkcji `getDistance`:

- `getDistance(i, j)` daje w wyniku odległość od stacji `i` do stacji `j`. `getDistance(i, i)` daje wynik 0. `getDistance(i, j)` daje wynik -1, jeśli `i` lub `j` znajduje się poza zakresem $0 \leq i, j \leq n-1$.

Podzadania

We wszystkich podzadaniach liczba bloków m nie przekracza 1,000,000. W niektórych podzadaniach liczba wywołań funkcji `getDistance` jest ograniczona. Ograniczenie to zależy od konkretnego podzadania. Jeśli Twój program przekroczy to ograniczenie, otrzyma wynik 'wrong answer', .

podzadanie	liczba punktów	n	max liczba wywołań getDistance	dodatkowe ograniczenia
1	8	$1 \leq n \leq 100$	brak ograniczeń	Wszystkie stacje poza stacją 0 znajdują się w blokach typu D.
2	22	$1 \leq n \leq 100$	brak ograniczeń	Wszystkie stacje położone na wschód od stacji 0 znajdują się w blokach typu D, a wszystkie stacje położone na zachód od stacji 0 znajdują się w blokach typu C.
3	26	$1 \leq n \leq 5,000$	$n(n-1)/2$	brak
4	44	$1 \leq n \leq 5,000$	$3(n-1)$	brak

Implementacja

Powinieneś zgłosić dokładnie jeden plik o nazwie `rail.c`, `rail.cpp` lub `rail.pas`. W pliku powinna znaleźć się implementacja funkcji `findLocation` opisanej powyżej, o następującej sygnaturze. W przypadku programu w C/C++ powinieneś także załączyć (*include*) plik nagłówkowy `rail.h`.

Programy w C/C++

```
void findLocation(int n, int first, int location[], int stype[]);
```

Programy w Pascalu

```
procedure findLocation(n, first : longint; var location,
stype : array of longint);
```

Sygnatury funkcji `getDistance` są następujące.

Programy w C/C++

```
int getDistance(int i, int j);
```

Programy w Pascalu

```
function getDistance(i, j: longint): longint;
```

Przykładowy program sprawdzający

Przykładowy program sprawdzający wczytuje dane w następującym formacie:

- wiersz 1: numer podzadania
- wiersz 2: `n`
- wiersz $3+i$, ($0 \leq i \leq n-1$): `stype[i]` (1 dla typu C i 2 dla typu D), `location[i]`.

Przykładowy program sprawdzający wypisze komunikat `Correct`, jeśli `location[0]` ... `location[n-1]` oraz `stype[0]` ... `stype[n-1]` na końcu wywołania Twojej funkcji `findLocation` będą identyczne z tymi podanymi na wejściu, a `Incorrect`, jeśli nie są one identyczne.