# System rezerwacji biletów

***Michał Dębski***

Po całym semestrze zmagań z algorytmami masz już dość; postanawisz rzucić to wszystko i wyjechać w Bieszczady. Pakujesz plecak, wychodzisz z domu i wesoło pogwizdując idzesz na dworzec. Kiedy jednak ustawiasz się w kolejce do kasy, żeby kupić bilet, masz niejasne przeczucie, że coś nie jest w porządku. Na poczatku kolejki robi się zamieszanie. Chwilę później z megafonów słyszysz komunikat: „Szanowni państwo, nastąpiła awaria systemu rezerwacji biletów. Do czasu usunięcia usterki sprzedaż biletów zostaje wstrzymana. Za utrudnienia uprzejmie przepraszamy. Programistów lub osoby znające programistów uprzejmie prosimy o zgłoszenie się do biura obsługi klieta". Wygląda na to, że zanim wyjedziesz na upragnione wakacje, raz jeszcze musisz użyć swoich umiejętności.

## Zadanie właściwe

Kluczowym elementem systemu rezerwacji jest klasa ReservationManager, która obsługuje rezerwacje miejsc w pojedynczym pociągu, zatrzymującym się na $N$ stacjach. Każda rezerwacja to para liczb całkowitych $(s, e)$, gdzie $0 \leq s < e < N$; $s$ oznacza numer stacji początkowej, a $e$ numer stacji końcowej. Klasa musi pozwalać na następujące operacje.

- Initialize – resetuje strukturę tak, aby reprezentowała pusty pociąg zatrzymujący się na zadanej liczbie stacji. Metoda spełnia rolę podobną do konstruktora, ale może być wywoływana wielokrotnie na tym samym obiekcie.
- AddReservation – dodaje rezerwację o zadanych parametrach $(s, e)$ do struktury. Można założyć, że dane są poprawne (tzn. $0 <= s < e < N$). Strukltura powinna umożliwiać przechowywanie wielu rezerwacji o tych samych parametrach.
- RemoveReservation – usuwa rezerwację o zadanych parametrach $(s, e)$ ze struktury. Można założyć, że dane są poprawne (tzn. $0 <= s < e < N$) oraz że rezerwacja $(s, e)$ zawiera się w strukturze.
- FilledSeats – zwraca informację o liczbie miejsc, które będą zajęte w momencie, kiedy pociąg będzie przejeżdżał przez zadaną stację $s$. Należy policzyć wszystkie rezerwacje, które zaczynają się wcześniej niż $s$ i kończą później niż $s$ (nie uwzględniamy rezerwacji, które zaczynają się lub kończą na zadanej stacji).
- FirstOverlappingReservation – zwraca pierwszą rezerwację, która przechodzi przez stację o zadanym numerze $k$; tzn. spośród rezerwacji $(s, e)$ takich, że $s <= k < e$, należy wybrać rezerwację o najmniejszej wartości $s$. Jeśli jest wiele rezerwacji o tej samej, minimalnej wartości $s$, należy spośród nich wybrać tę o największej wartości $e$. W przypadku, kiedy żadna rezerwacja nie spełnia żądanych wymagań, należy zwrócić krotkę $(-1, -1)$.

Każda z operacji poza Initialize powinna działać w czasie $O(\log N)$, gdzie $N$ jest liczbą stacji, na których zatrzymuje się pociąg; czas działania operacji Initialize nie powinien przekraczać czasu potrzebnego na zaalokowanie $O(N)$ pamięci.

## Uwagi i wskazówki

- Zadanie można rozwiązać inspirując się drzewami przedziałowymi.
- Czas działania $O(\log N)$ można uzyskać przy użyciu statycznej implementacji drzewa w tablicy, analogicznie do implementacji kopców binarnych (nie trzeba implementować drzew zrównoważonych).
- Można uzyskać 2.5 punktu za poprawną implementację operacji (Initialize, AddReservation, RemoveReservation, FilledSeats) oraz 2.5 punktu za poprawną implementację operacji (Initialize, AddReservation, RemoveReservation, FirstOverlappingReservation).