

## Zadanie G

### Bajto-Play

Twoja praca u operatora sieci komórkowej *BajtoPlay* ostatnimi czasy odbiera Ci sen z oczu. Sieć komórkowa *BajtoPlay* oferuje firmom specjalną atrakcyjną ofertę umożliwiającą wykupienie kilku kolejnych numerów telefonów. Musisz więc często sprawdzać czy zadane kolejne numery telefony są wolne. Tymczasem złe chochliki pomieszały bazę danych użytkowników. Do tego co chwila wpada jakiś nierozgarnięty Bajtocjanin i chce zrezygnować z posiadanego numer telefonu albo go zmienić.

Jakby jeszcze mało było pracy czasami wpada Twój szef i szuka dziury w całym. „Do kogo należy  $k$ -ty numer telefonu?” pyta na przykład. Skąd mu do głowy takie głupie pytania do głowy przychodzą to doprawdy nie wiadomo... Może bierze udział w konkursie „na najbardziej uciążliwego szefa”?...

Twoim zadaniem jest zaimplementowanie elektronicznej bazy użytkowanych numerów telefonów sieci komórkowej *BajtoPlay* za pomocą drzewa przeszukiwań binarnych (BST). Numer telefonu jest ciągiem 9 cyfr.

Należy zaimplementować następujące operacje:

- **INSERT nr nazwisko** - dodaje nowy numer telefonu do bazy. Jeśli numer jest już w bazie, operacja nie modyfikuje bazy, i wypisuje słowo **ERROR**,
- **FIND nr** - sprawdza, czy podany numer jest w bazie, jeśli jest, to wypisuje **YES** a następnie po spacji numer oraz po spacji nazwisko właściciela, w przeciwnym razie wypisuje **NO**,
- **DELETE nr** - usuwa numer telefonu **nr**, o ile jest w bazie. Wypisuje **OK** jeśli operacja się wykonała lub **ERROR**, jeżeli takiego numeru nie ma w bazie,
- **PRINT** - wypisuje listę wszystkich numerów w kolejności rosnących numerów. W każdej linii wypisany jest numer telefonu i po spacji nazwisko jego właściciela. Jeśli baza jest pusta, należy wypisać **EMPTY**.
- **TEST nr1 nr2** - wypisuje liczbę zajętych numerów pomiędzy dwoma wskazanymi numerami (również wówczas gdy zadanych numerów nie ma w bazie). Możesz założyć, że **nr1 < nr2**.
- **SELECT k** - wypisuje w pojedynczej linii  $k$ -ty numer telefonu w bazie oraz nazwisko jego właściciela. Inaczej mówiąc, wypisany zostaje numer telefonu, którego poprzedza dokładnie  $k-1$  numerów bazy. Jeżeli takiego numeru nie ma, należy wypisać **ERROR**.

Zadanie należy zrealizować przez zaimplementowanie szablonów dwóch klas: `node<T1, T2>` oraz `map<T1, T2>`. Szablon `map` realizuje słownik, czyli posortowany kontener asocjacyjny o zmiennej długości. Elementami słownika są unikatowe pary klucz i element (typu `T1` i `T2`). Należy go zaimplementować za pomocą **reprezentacji wskaźnikowej drzewa**

**przeszukiwań binarnych (BST)**, którego elementami są obiekty klasy `node<T1,T2>`. Nie jest wymagane równoważenie drzewa.

1. Klasa `node<T1,T2>` przechowuje pary o typach `T1` i `T2` oraz udostępnia następujące metody:

- `node(T1 k, T2 n)` – konstruktor, nowy obiekt przechowuje wartości `k` oraz `n`.
- `ostream& operator<<(ostream& st, node<T1,T2>& a)` – operator wypisuje wartości obiektu `a` w jednej linii oddzielając je spacją.
- `T2 GetName()` – zwraca przechowywane pole typu `T2`.

2. Klasa `map<T1,T2>` udostępnia następujące metody:

- konstruktor tworzący pusty słownik
- `T2& operator[] (T1 key)` – operator zwraca referencję do pola typu `T2` elementu o kluczu `key`. Jeśli takiego elementu nie ma w słowniku, zostaje dodany.
- `node<T1,T2>* find(T1 key)` – zwraca wskaźnik do elementu o kluczu `key`. Gdy elementu nie ma, metoda zwraca `nullptr`.
- `bool delet(T1 key)` – usuwa ze słownika element o kluczu `key`. Zwraca `true`, gdy usuwanie się powiodło, `false`, gdy nie ma w słowniku elementu o wskazanym kluczu.
- `void print()` – wypisuje wszystkie elementy słownika posortowane rosnąco według kluczy.
- `void clean()` – czyści słownik.
- `int test(T1 a, T1 b)` – zwraca liczbę elementów słownika, dla których pole `key` jest w przedziale  $(a, b)$ . Możesz założyć, że  $a < b$ .
- `node<T1,T2>* select(int k)` – zwraca wskaźnik do  $k$ -tego elementu słownika. Inaczej mówiąc, wskaźnik do elementu, którego poprzedza dokładnie  $k - 1$  elementów słownika. Jeżeli takiego numeru nie ma, funkcja zwraca `nullptr`.

Definicję wymienionych szablonów wraz z definicjami wszystkich metod i operatorów należy umieścić w pliku z rozszerzeniem `.h`. Tak przygotowany plik należy wysłać na Satori. Zostanie on skompilowany wraz z plikiem zawierającym funkcję `main`.

```
#include <iostream>
#include <string>
using namespace std;
#include "solution.h"

int main(){
    ios_base::sync_with_stdio(false);
    int z, n, k, number, number2;
```

```
string name;
map<int, string> T;

cin >> z;
while(z--) {
    cin >> n;
    while(n--) {
        string cmd;
        cin >> cmd;

        if (cmd == "INSERT")
        {
            cin >> number >> name;
            node<int, string>* a = T.find(number);
            if (a) cout << "ERROR" << endl; else T[number] = name;
        }
        if (cmd == "FIND") {
            cin >> number;
            node<int, string>* a = T.find(number);
            if (a) cout << "YES " << *a; else cout << "NO" << endl;
        }
        if (cmd == "DELETE") {
            cin >> number;
            if (T.delet(number)) cout << "OK" << endl;
            else cout << "ERROR" << endl;
        }
        if (cmd == "PRINT") T.print();

        if (cmd == "TEST") {
            cin >> number >> number2;
            cout << T.test(number, number2) << endl;
        }
        if (cmd == "SELECT") {
            cin >> k;
            node<int, string>* a = T.select(k);
            if (a) cout << *a; else cout << "ERROR" << endl;
        }
    }
}
T.clean();
return 0;
}
```

## Wejście

Pierwsza linia wejścia zawiera liczbę całkowitą  $z$  ( $1 \leq z \leq 2 \cdot 10^9$ ) – liczbę zestawów danych, których opisy występują kolejno po sobie. Opis jednego zestawu jest następujący:

Pierwsza linia zawiera liczbę naturalną  $n$  ( $1 \leq n \leq 2 \cdot 10^6$ ) oznaczającą ilość operacji do wykonania. Kolejne  $n$  linii zawiera: kod operacji oraz stosowne dla operacji argumenty oddzielone spacją. Argumentami są (w zależności od operacji): numer telefonu (9-znakowy napis składający się z cyfr) oraz nazwisko (maksymalnie 10 znakowy napis złożony z małych liter alfabetu angielskiego).

## Wyjście

Każdą wczytaną operację wykonaj zgodnie z jej opisem.

**Wersja G1** - nie obsługuje poleceń: **TEST** i **SELECT**, wersja za 0.7 pkt.

**Wersja G2\*** - obsługuje wszystkie polecenia, wersja za dodatkowe 0.3 pkt.

**Dostępna pamięć: w zależności od testu 2-8MB**

## Przykład

Dla danych wejściowych:

```
1
31
INSERT 123456789 kowalski
INSERT 321000000 iglinski
INSERT 220123456 bednarski
INSERT 220123456 kwaklinski
INSERT 000000000 nowak
INSERT 000667890 kowalski
INSERT 321000000 abacki
PRINT
INSERT 220123456 nowak
FIND 220123456
FIND 000067890
FIND 000000000
FIND 321000000
TEST 000667880 332100000
TEST 000000001 000000002
TEST 000000001 100000000
TEST 001000001 220123456
TEST 000667890 310123456
SELECT 1
SELECT 5
SELECT 8
DELETE 000000000
DELETE 321000000
DELETE 123456689
SELECT 2
SELECT 4
SELECT 6
PRINT
FIND 123456789
FIND 220123456
FIND 321000000
```

Poprawną odpowiedzią jest:

```
ERROR
ERROR
000000000 nowak
000667890 kowalski
123456789 kowalski
220123456 bednarski
321000000 iglinski
ERROR
YES 220123456 bednarski
NO
YES 000000000 nowak
YES 321000000 iglinski
4
0
1
1
2
000000000 nowak
321000000 iglinski
ERROR
OK
OK
ERROR
123456789 kowalski
ERROR
ERROR
000667890 kowalski
123456789 kowalski
220123456 bednarski
YES 123456789 kowalski
YES 220123456 bednarski
NO
```