

**Сдать задание нужно до 9:00 20 мая 2017г. включительно.**

Ссылка на констест: <https://contest.yandex.ru/contest/4463/enter/>

### **Задача 1. Хеш-таблица (7 баллов)**

Реализуйте структуру данных типа “множество строк” на основе динамической хеш-таблицы с открытой адресацией. Хранимые строки непустые и состоят из строчных латинских букв.

Хеш-функция строки должна быть реализована с помощью вычисления значения многочлена методом Горнера.

Начальный размер таблицы должен быть равным 8-ми. Перехеширование выполняйте при добавлении элементов в случае, когда коэффициент заполнения таблицы достигает 3/4.

Структура данных должна поддерживать операции добавления строки в множество, удаления строки из множества и проверки принадлежности данной строки множеству.

**1\_1.** Для разрешения коллизий используйте квадратичное пробирование.  $i$ -ая проба  $g(k, i) = g(k, i-1) + i \pmod{m}$ .  $m$  - степень двойки.

**1\_2.** Для разрешения коллизий используйте двойное хеширование.

#### Формат входных данных

Каждая строка входных данных задает одну операцию над множеством. Запись операции состоит из типа операции и следующей за ним через пробел строки, над которой проводится операция.

Тип операции – один из трех символов:

- + означает добавление данной строки в множество;
- означает удаление строки из множества;
- ? означает проверку принадлежности данной строки множеству.

При добавлении элемента в множество НЕ ГАРАНТИРУЕТСЯ, что он отсутствует в этом множестве. При удалении элемента из множества НЕ ГАРАНТИРУЕТСЯ, что он присутствует в этом множестве.

#### Формат выходных данных

Программа должна вывести для каждой операции одну из двух строк OK или FAIL, в зависимости от того, встречается ли данное слово в нашем множестве.

stdin	stdout
+ hello	OK
+ bye	OK
? bye	OK
+ bye	FAIL
- bye	OK

? bye	FAIL
? hello	OK

## Задача 2. Порядок обхода (3 балла)

Дано число  $N < 10^6$  и последовательность целых чисел из  $[-2^{31}..2^{31}]$  длиной  $N$ .

Требуется построить бинарное дерево, заданное наивным порядком вставки.

Т.е., при добавлении очередного числа  $K$  в дерево с корнем  $root$ , если  $root \rightarrow Key \leq K$ , то узел  $K$  добавляется в правое поддерево  $root$ ; иначе в левое поддерево  $root$ .

Рекурсия запрещена.

### 2\_1. Выведите элементы в порядке in-order (слева направо).

in	out
3 2 1 3	1 2 3
3 1 2 3	1 2 3
3 3 1 2	1 2 3

### 2\_2. Выведите элементы в порядке pre-order (сверху вниз).

in	out
3 2 1 3	2 1 3
3 1 2 3	1 2 3
3 3 1 2	3 1 2
4 3 1 4 2	3 1 2 4

### 2\_3. Выведите элементы в порядке post-order (снизу вверх).

in	out
3 2 1 3	1 3 2
3 1 2 3	3 2 1
3 3 1 2	2 1 3

**2\_4. Выведите элементы в порядке level-order (по слоям, “в ширину”).**

in	out
3 2 1 3	2 1 3
3 1 2 3	1 2 3
3 3 1 2	3 1 2
4 3 1 4 2	3 1 4 2

**Задача 3. Декартово дерево (4 балла)**

Дано число  $N < 10^6$  и последовательность пар целых чисел из  $[-2^{31}..2^{31}]$  длиной  $N$ .

Построить декартово дерево из  $N$  узлов, характеризующихся парами чисел  $\{X_i, Y_i\}$ .

Каждая пара чисел  $\{X_i, Y_i\}$  определяет ключ  $X_i$  и приоритет  $Y_i$  в декартовом дереве.

Добавление узла в декартово дерево выполняйте второй версией алгоритма, рассказанного на лекции:

- При добавлении узла  $(x, y)$  выполняйте спуск по ключу до узла  $P$  с меньшим приоритетом. Затем разбейте найденное поддерево по ключу  $x$  так, чтобы в первом поддереве все ключи меньше  $x$ , а во втором больше или равны  $x$ . Получившиеся два дерева сделайте дочерними для нового узла  $(x, y)$ . Новый узел вставьте на место узла  $P$ .

Построить также наивное дерево поиска по ключам  $X_i$  методом из задачи 2.

**3\_1.** Вычислить разницу глубин наивного дерева поиска и декартового дерева. Разница может быть отрицательна.

in	out
10 5 11 18 8 25 7 50 12 30 30 15 15 20 10 22 5 40 20 45 9	2
10 38 19 37 5 47 15	2

35 0 12 3 0 42 31 37 21 45 30 26 41 6	
---	--

**3\_2.** Вычислить количество узлов в самом широком слое декартового дерева и количество узлов в самом широком слое наивного дерева поиска. Вывести их разницу. Разница может быть отрицательна.

in	out
10 5 11 18 8 25 7 50 12 30 30 15 15 20 10 22 5 40 20 45 9	1
10 38 19 37 5 47 15 35 0 12 3 0 42 31 37 21 45 30 26 41 6	1

#### Задача 4. Использование AVL-дерева (6 баллов)

**4\_1. Солдаты.** В одной военной части решили построить в одну шеренгу по росту. Т.к. часть была далеко не образцовая, то солдаты часто приходили не вовремя, а то их и вовсе приходилось выгонять из шеренги за плохо начищенные сапоги. Однако солдаты в процессе прихода и ухода должны были всегда быть выстроены по росту – сначала самые высокие, а в конце – самые низкие. За расстановку солдат отвечал прапорщик, который заметил интересную особенность – все солдаты в части разного роста. Ваша задача состоит в том, чтобы помочь прапорщику правильно расставлять солдат, а именно для каждого приходящего солдата указывать, перед каким солдатом в строе он должен становится. Требуемая скорость выполнения команды -  $O(\log n)$ .

Формат входных данных.

Первая строка содержит число  $N$  – количество команд ( $1 \leq N \leq 30\,000$ ). В каждой

следующей строке содержится описание команды: число 1 и X если солдат приходит в строй (X – рост солдата, натуральное число до 100 000 включительно) и число 2 и Y если солдата, стоящим в строю на месте Y надо удалить из строя. Солдаты в строю нумеруются с нуля.

#### Формат выходных данных.

На каждую команду 1 (добавление в строй) вы должны выводить число K – номер позиции, на которую должен встать этот солдат (все стоящие за ним двигаются назад).

in	out
5	0
1 100	0
1 200	2
1 50	1
2 1	
1 150	

**4\_2. Порядковые статистики.** Дано число N и N строк. Каждая строка содержит команду добавления или удаления натуральных чисел, а также запрос на получение k-ой порядковой статистики. Команда добавления числа A задается положительным числом A, команда удаления числа A задается отрицательным числом “-A”. Запрос на получение k-ой порядковой статистики задается числом k. Требуемая скорость выполнения запроса -  $O(\log n)$ .

in	out
5	40
40 0	40
10 1	10
4 1	4
-10 0	50
50 2	

## Задача 5. Алгоритм сжатия данных Хаффмана (6 баллов и более)

Напишите две функции для создания архива из одного файла и извлечения файла из архива.

```
// Метод архивирует данные из потока original
void Encode(IInputStream& original, IOutputStream& compressed);
// Метод восстанавливает оригинальные данные
void Decode(IInputStream& compressed, IOutputStream& original);
где:
Type
def char byte;

interface IInputStream {
```

```

        // Возвращает false, если поток закончился
        virtual bool Read(byte& value) = 0;
};

```

```

interface IOutputStream {
    virtual void Write(byte value) = 0;
};

```

В архиве сохраняйте дерево Хаффмана и код Хаффмана от исходных данных. Дерево Хаффмана требуется хранить эффективно - не более 10 бит на каждый 8-битный символ.

В контекст необходимо отправить .cpp файл содержащий функции Encode, Decode, а также включающий файл Huffman.h. Тестирующая программа выводит размер сжатого файла в процентах от исходного.

Лучшие 3 решения с  $K_s \leq 50\%$  из каждой группы оцениваются в 15, 10 и 5 баллов соответственно.

Пример минимального решения:

```

#include "Huffman.h"

```

```

static void copyStream(IInputStream& input, IOutputStream& output)
{
    byte value;
    while (input.Read(value))
    {
        output.Write(value);
    }
}

```

```

void Encode(IInputStream& original, IOutputStream& compressed)
{
    copyStream(original, compressed);
}

```

```

void Decode(IInputStream& compressed, IOutputStream& original)
{
    copyStream(compressed, original);
}

```