## Задача A1. Анализ линейного пробирования

В хеш-таблице с открытой адресацией разрешение коллизий производится с помощью линейного пробирования. При удалении объекта из хеш-таблицы свободная ячейка получает значение ERASED, отличное от NULL, которое обозначает пустое значение.
Ниже приведены алгоритмы вставки, удаления и поиска, где M обозначает размер хеш-таблицы:

```
INSERT(key):
    ind = hash(key) mod M

    while (table[ind] != NULL)
        if (table[ind] == key) return
        ind = (ind + 1) mod M

    table[ind] = key
```

```
DELETE(key):
    ind = hash(key) mod M

    while (table[ind] != NULL)
        if (table[ind] == key)
            table[ind] = ERASED
            return
        ind = (ind + 1) mod M
```

```
SEARCH(key):
    ind = hash(key) mod M

    while (table[ind] != NULL)
        if (table[ind] == key)
            return true
        ind = (ind + 1) mod M
    return false
```

**1. Приведенные выше алгоритмы вставки, удаления и поиска ключа имеют проблему, которая приводит к долгому выполнению некоторой(-ых) последовательности(-ей) этих операций.**

- **Найдите такую(-ие) последовательность(-и) операций вставки, удаления и поиска.**
- **Охарактеризуйте соответствующее состояние хеш-таблицы. Приведите примеры.**

При использовании открытой адресации с линейным пробированием удалённые элементы не превращаются в свободные ячейки (NULL), а получают специальное значение ERASED. При этом алгоритмы вставки и поиска проверяют только на NULL, считая ячейки с ERASED занятыми, но пустыми с точки зрения содержимого. В результате, если в таблице накопилось много таких ячеек, то при выполнении операций вставки или поиска приходится проходить длинные последовательности ячеек, содержащих ERASED, прежде чем встретится настоящая пустая ячейка. Это может значительно замедлить работу алгоритмов.

**Рассмотрим таблицу размера $M = 10$:**

**1) Вставка:** Вставим несколько ключей, которые все хешируются в одну и ту же позицию, например в 0. При коллизиях они будут размещены в ячейках с индексами 0, 1, 2, . . . , 9

**2) Удаление:** Удалим все ключи, кроме первого. При удалении ячейка не становится NULL, а получает значение ERASED. После этого в таблице имеется один настоящий элемент (k1 в ячейке 0) и восемь ячеек с меткой ERASED (в ячейках 1–9).

**3) Поиск:** Теперь попробуем найти некоторый несуществующий ключ, который хешируется в 0. Алгоритм начинает с индекса 0, находит k1 (не совпадает с искомым ключом), затем идёт по индексам 1, 2, . . . , 9. Так как ячейки с 1 по 9 содержат ERASED, условие цикла while table[ind] != NULL не прерывается. Только после обхода всей таблицы операция завершается. Таким образом, поиск (а также вставка нового элемента с хешем 0) вынужден просматривать длинную цепочку ячеек, хотя многие из них уже устарели.

**Состояние таблицы:** Такая последовательность операций приводит к загрязнению таблицы большим числом ячеек со значением ERASED. В результате формируется длинный кластер, где почти все ячейки не содержат полезных данных, но и не считаются свободными для вставки, что ведёт к ухудшению времени работы операций поиска и вставки.

**2. Предложите доработки (кроме перехеширования) исходных алгоритмов вставки, удаления и поиска, которые помогут исправить обнаруженную вами проблему.**

При вставке нового элемента можно запоминать первую встреченную ячейку с ERASED вместо того, чтобы игнорировать её. Тогда, если при дальнейшей проверке не найдётся дублирующий элемент, новый элемент можно разместить именно в этой ячейке, а не ждать встречи с ячейкой NULL. Это позволяет очищать таблицу по ходу вставок, не давая кластерам ERASED накапливаться.

Другой вариант, back-shift deletion, заключается в том, чтобы после удаления элемента перемещать последующие элементы назад, заполняя пропуски.

1) После того как нашли и удалили элемент, заменив его на ERASED, переходим к следующей ячейке.
2) Если найденный элемент в последующей ячейке имеет хеш, который указывает на позицию до удалённой ячейки, его можно переместить назад.

3) Продолжаем до тех пор, пока не встретится NULL.

Однако этот метод может рассматриваться как локальное перехеширование части таблицы.

Алгоритм SEARCH в принципе корректен, так как он не прерывается на ERASED, т.е. продолжает искать, пока не встретится NULL. Если реализовать back-shift deletion, то цепочки для поиска будут короче.