A1. Анализ линейного пробирования

Демченко Георгий Павлович , БПИ-235

M - **размер хэш-таблицы**

Insert

```
void Insert(key) {
    size_t ind = hash(key) % M;

    while (table[ind] != null) {
        if (table[ind] == key) {
            return;
        }
        ind = (ind + 1) % M;
    }

    table[ind] = key;
}
```

Delete

```
void Delete(key) {
    size_t ind = hash(key) % M;

    while (table[ind] != null) {
        if (table[ind] == key) {
            table[ind] = ERASED;
            return;
        }

        ind = (ind + 1) % M;
    }
}
```

Search

```
bool Search(key) {
    size_t ind = hash(key) % M;

    while (table[ind] != null) {
        if (table[ind] == key) {
            return true;
        }
        ind = (ind + 1) % M;
    }

    return false;
}
```

1. Проблема долгого выполнения последовательностей операций

Проблемы :

- Игнорирование значения ERASED при вставке нового элемента в таблицу
- Отсутствие смещения элементов в бакетах "справа-налево" при удалении элемента из таблицы

Анализ :

При продолжительной вставке элементов в таблицу образуются первичные кластеры (напрямую влияющие на оценку сложности основных операцией с таблицей), которые в силу реализации останутся до момента перехеширования, даже после удаления элемента из таблицы, и, более того, будут только увеличиваться в размерах.

Пример :

1. Вставка элементов в таблицу

=> Образование кластеров достаточной длинны (в том числе с смещенными-пробированными элементами из-за колизий)

2. Удаление элементов из хеш-таблицы, замена значений удаленных элементов в кластерах на ERASED

=> Размер кластеров не изменился, так как отсутвует смещение элементов при удалении

3. Вставка элементов в таблицу

=> Так как размер кластеров не изменился и при вставке элементов мы не учитываем статус ERASED элемента, то в случае необходимости смещения-пробирования элемента мы пройдем весь кластер из существующих и псевдо-удаленных элементов, пока не дойдем до конца кластера (NULL), куда и будет вставлен элемент. Таким образом вставка элементов, требующих пробирование, всегда будет происходить в конец кластера, увеличивая его размер.

4. Поиск (удаление) элементов в таблице

=> Если необходимо найти смещенный элемент для удаления или поиска (который находится не на своём индексе в таблице), то нам прийдется пройти большую часть кластера , так как новые элементы всегда вставляются в конец кластера, вне зависимости от ERASED элементов.

=> **В таблице не реализовано уменьшение размеров кластеров (физическое удаление элементов), операции удаления и вставки ничего не меняют, все операции деградируют по временными затратам по мере добавления элементов, в виду постоянного роста размеров класетров.**

Состояние хеш-таблицы: Вечные (до рехеширования), увеличивающиеся в размерах первичные кластеры. Бесконечно-увеличивающаяся в размерах таблица без возможности физического удаления элемента.

Конкретный пример :

Обоначим NULL за N, ERASED за E, значение елемента в таблице за T_i

1. Изначально имеем пустую таблицу

$$[N, \cdots, N]$$

2. Вставим элементы в таблицу, рассмотрим один из образованных кластеров (предположим, что такой имеется)

$$[\cdots N, T_1, T_2, \cdots T_{m-1}, T_m, N \cdots]$$

3. Удалим элементы из таблицы, пусть, для наглядности, удалятся все елементы кластера, кроме краевых

$[\cdots N, T_1, E, \cdots E, T_m, N \cdots]$

4. Вставим элемент T_{m+1}, хэш которого совпадает с позицией элемента T_1

=> Необходимо смещение, E элементы игнорируются, в поисках места проходим весь кластер длинны m и вставляем в позицию после T_m (конец кластера)

$[\cdots N, T_1, E, \cdots E, T_m, T_{m+1}, N \cdots]$

=> Выполнено (в худшем случае) m лишних операций поиска места (пройден весь кластер)

5. Последующий поиск и удаление елемента T_{m+1} также займет m лишних операций, так как снова прийдется проходить по всему "пустому" кластеру

=> Таблица и кластеры будут и дальше расти при вставке, не освобождая место E элементов, приводя к деградации операций.

2. Исправление операций

Варианты исправления:

- **Корректный учет ERASED элементов в алгоритме вставки.** Линейно ищем место во внутреннем массиве покуда не встретится NULL или ERASED. Получаем переиспользование места в кластерах.
- **Смещение элементов "справа-налево" в алгоритме удаления элемента.** После удаления элемента в цикле проходимся по следующим элементам в кластере для заполнения образующихся "пустот" при перекладывании подходящих элементов "справа-налево". Получаем уменьшение размеров кластеров путем их ссужения/разбиения.

Исправление путем смещения элементов при удалени более предпочтительно по сравнению с учетом ERASED элементов, так как при таком подходе поиск и удаление элементов в разрывах кластеров E элементами также потребует полного (большего, в виду существования ERASED элементов) прохода.