SET1 Семестр 2. Задача A4. Анализ производительности линейного пробирования

Фролов-Буканов Виктор Дмитриевич БПИ-228

14 февраля 2024

1 Условие задачи

Задание A1 (7 баллов) Анализ производительности 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)
  while table[ind] != NULL
    if table[ind] == key
      return
    ind = (ind + 1) mod M
  table[ind] = key
```

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

```
DELETE(key):
  ind = hash(key)
  while table[ind] != NULL
    if table[ind] == key
      return TRUE
    ind = (ind + 1) mod M
  return FALSE
```

1. (4 балла) Приведенные выше алгоритмы вставки, удаления и поиска ключа имеют проблему, которая приводит к долгому выполнению некоторой(-ых) последовательности(-ей) этих операций.
   a. Найдите такую(-ие) последовательность(-и) операций вставки, удаления и поиска.
   b. Охарактеризуйте соответствующее состояние хеш-таблицы. Приведите примеры.
2. (3 балла) Предложите доработки (кроме перехеширования) исходных алгоритмов вставки, удаления и поиска, которые помогут исправить обнаруженную вами проблему.

2 Решение пункта 1

Многократное повторение вставки, удаления и поиска одного и того же элемента существенно замедляет время работы таблицы. Приведем конкретный пример. Пусть мы 2 раза выполнили insert(10), delete(10), search(10). Пусть в начале таблица имела вид:

0	1	2	3	4
NULL	NULL	NULL	NULL	NULL

Состояние после insert(10):

0	1	2	3	4
10	NULL	NULL	NULL	NULL

Состояние после delete(10):

0	1	2	3	4
ERASED	NULL	NULL	NULL	NULL

search(10) посетит нулевую ячейку, а дальше вернет *False*
Состояние после insert(10):

0	1	2	3	4
ERASED	10	NULL	NULL	NULL

Состояние после delete(10):

0	1	2	3	4
ERASED	ERASED	NULL	NULL	NULL

search(10) посетит нулевую и первую ячейки, а дальше вернет *False*
Последовательно повторяя описанную операцию, операции вставки, удаления и поиска будут в таком случае работать за время, линейное от числа вставок в таблицу. В целом, такая ситуация повторяется с любыми элементами, имеющими равный хеш, которые удаляются сразу после вставки, а потом заново вставляются (возможно, с промежуточным поиском элемента)

Охарактеризовать данное состояние таблицы можно наличием кластера, состоящего из ERASED значений. В прошлом пункте я дал оценку времени выполнения такой последовательности действий как линейное от числа вставок в таблицу. В целом это же время можно оценить как линейное от числа образовавшегося кластера из ERASED значений

Заметим также, что такие кластеры могут образовываться в разных местах таблицы, и в целом такой кластер может быть не единственным. Это зависит от той последовательности ключей, которую мы вставляем в данную реализацию хеш-таблицы

3 Решение пункта 2

Из доработок, если мы оставляем механизм ленивого удаления, то можно только изменить метод insert, который будет идти в цикле while до тех пор, пока он не встретит либо NULL значение, либо ERASED значение. То есть условие цикла while будет выглядеть так:

while (table[ind] != NULL AND table[ind] != ERASED)

Такая доработка позволит вставлять значение в хеш-таблицу раньше, чем это было в предыдущем случае. При этом методы delete и search мы не меняем. В такой модификации работа метода insert существенно убыстряется, но скорость методов delete и search остается той же. Если же мы хотим убыстрить метод search, сохранив скорость insert, то нам надо избавиться от механизма ленивого удаления, полностью переписав метод delete, который будет сдвивать все следующие значения кластера после удаленного на 1 позицию влево. В таком случае мы отказываемся от состояния ERASED, которое может возникать в таблице, и тогда методы insert и search остаются в точности такими же, какими нам их привели в условии. При этом стоит отметить, что скорость работы метода delete увеличится, но не асимптотически, а на константу (асимптотика будет также O(длины кластера)). То есть если мы заранее знаем, что над таблицей будет произведено мало операций удаления, но много операций вставки и поиска, то имеет смысл рассмотреть эту модификацию, в противном случае следует попробовать первую модификацию, сохраняющую ленивое удаление