SET1 Семестр 2. Задача A2. Кубическое пробирование

Фролов-Буканов Виктор Дмитриевич БПИ-228

16 февраля 2024

1 Условие задачи

Задание A2 (10 баллов) Кубическое пробирование

Хеш-таблицы с **открытой адресацией** используют различные методы пробирования для разрешения коллизий, к основным из которых можно отнести:

1. **Линейное** пробирование, при котором последовательно проверяются ячейки хеш-таблицы с индексами $hash(key), hash(key) + 1, hash(key) + 2, \ldots$
2. **Квадратичное** пробирование $hash(key, i) = hash(key) + c_1 \cdot i + c_2 \cdot i^2$, при котором:
   a. в простом варианте при $c_1 = c_2 = 1$ последовательно проверяются ячейки $hash(key), hash(key) + 1, hash(key) + 2, hash(key) + 6, \ldots$
   b. для хеш-таблицы размера $M = 2^m$ при $c_1 = c_2 = \frac{1}{2}$ последовательно проверяются ячейки $hash(key), hash(key) + 1, hash(key) + 3, hash(key) + 6, \ldots$

Мы решили пойти дальше и рассмотреть **кубическое** пробирование, при котором проверка ячеек в хеш-таблице выполняются по следующему правилу:

$hash(key, i) = hash(key) + c_1 \cdot i + c_2 \cdot i^2 + c_3 \cdot i^3$.

Оцените, будет ли кубическое пробирование выполнять распределение ключей по хеш-таблице лучше (более равномерно), чем квадратичное, с точки зрения образования кластеров и

SET 1 Хеширование и хеш-таблицы. Вероятностные структуры данных

Алгоритмы и структуры данных-2
2023/2024 учебный год

Дедлайн: 16.02.2024 23:00

возникновения коллизий. Подкрепите свои рассуждения программными экспериментами с хеш-таблицами различных размеров, а также приложите код.

Ограничений на используемые языки программирования в этом задании нет.

2 Аналитическое решение

В кубическом пробировании, в отличие от квадратичного, шаг, с которым будет идти проверка ячеек будет расти на порядок быстрее, но при этом вероятность образования кластеров будет примерно такой же, как и в квадратичном пробировании (по моему предположению), так как, повышение степени шага не будет давать сильных изменений, как оно давало в случае линейного и квадратичного пробирования, в силу того, что мы и раньше (в случае квадратичного пробирования) просматривали ячейки **непоследовательно**. Поэтому выдвинем *гипотезу* о том, что кубическое пробирование не будет давать существенных приростов с точки зрения равномерности распределения значений по хеш-таблице

3 Программное решение

main.cpp

```
#include <iostream>
#include <vector>

class SquareHashTable {
 public:
  explicit SquareHashTable(size_t size_) {
    size = size_;
    table = std::vector<int>(size_);
  }

  void insert(int key) {
    auto startInd = std::hash<int>{}(key) % size;
    size_t ind;
    int i = 0;

    do {
      ind = (startInd + i + i * i) % size;
      ++i;
    } while (table[ind] != 0);
    table[ind] = key;
  }

  void print() const {
    for (auto &el : table) {
      std::cout << el << ' ';
    }
    std::cout << '\n';
  }

 private:
  std::vector<int> table;
  size_t size;
};

class CubeHashTable {
 public:
  explicit CubeHashTable(size_t size_) {
    size = size_;
    table = std::vector<int>(size_);
  }

  void insert(int key) {
    auto startInd = std::hash<int>{}(key) % size;
    size_t ind;
    int i = 0;

    do {
      ind = (startInd + i + i * i + i * i * i) % size;
      ++i;
    } while (table[ind] != 0);
    table[ind] = key;
  }

  void print() const {
    for (auto &el : table) {
```

```
      std::cout << el << ' ';
    }
    std::cout << '\n';
  }

 private:
  std::vector<int> table;
  size_t size;
};

int main() {
  int size = 500;
  SquareHashTable square(size);
  CubeHashTable cube(size);

  for (auto i = 1; i <= size / 2; ++i) {
    auto el = rand() % (size / 2) + 1;
    square.insert(el);
    cube.insert(el);
  }

  square.print();
  std::cout << "\n##################################################################\n"
      ;
  cube.print();

  return 0;
}
```

4 Результаты работы программы на разных размерах хеш-таблицы

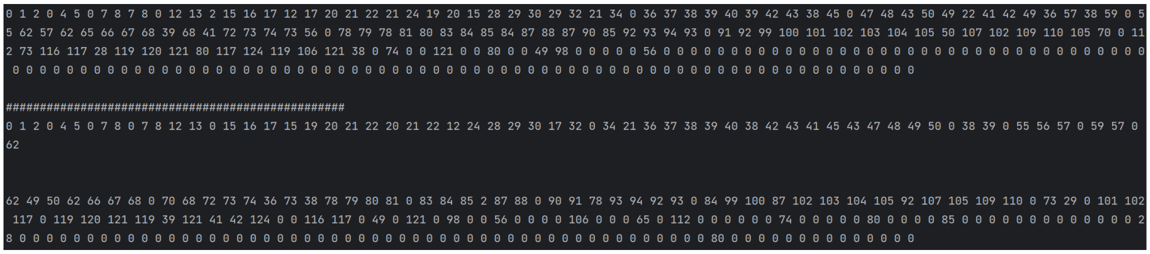
Хештэгами я разделяю вывод разных таблиц. Сначала выводится таблица на квадратичном пробировании ($c1 = c2 = 1$), потом на кубическом пробировании ($c1 = c2 = c3 = 1$). В каждую таблицу вставляю одни и те же псевдослучайные значения (функция *rand()* в C++). Вставляю ровно половину элементов от размера таблицы. Сами таблицы снабжаются неполноценным интерфейсом (нет методов remove, search), а также нет функционала перехеширования, так как такового не требует задание. Значения, вставляемые в таблицу, генерируются так, что они никогда не могут быть нулевыми, так что нулевое значение в итоговом выводе говорит лишь о том, что это ячейка пуста

```
0 1 0 3 4 3 6 7 0 9 10 5 12 13 12 15 4 17 18 17 20 21 0 21 18 25 0 0 0 17 18 0 0
 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


##################################################
0 1 18 3 4 5 6 7 0 9 10 0 12 13 0 15 0 17 18 0 20 21 17 0 21 25 12 0 0 0 0 17 18
 0 0 0 0 3 0 0 0 0 3 4 0 0 0 0 0 0
```

Figure 1: Размер 50

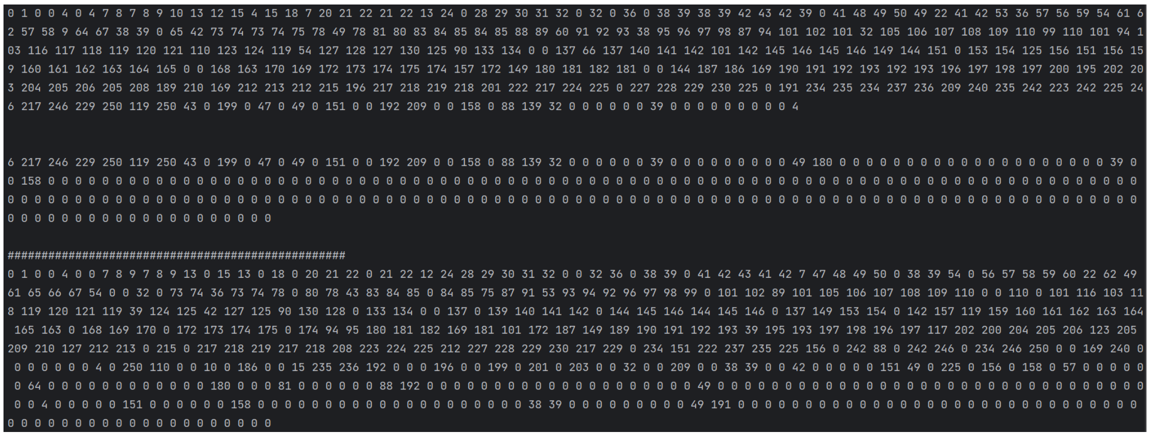
```
0 1 0 3 4 5 6 0 0 9 4 0 12 13 12 15 4 17 18 19 20 15 22 23 22 25 20 27 28 29 28 19 32 33 34 35 36 37 18 39 0 0 42 43 42 43 46 45 46 0 48 0 46 0 42 13 50 0 0 0 0 0 42 0
0 0 24 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


##############################################
0 1 0 3 4 5 6 4 0 9 0 0 12 13 0 15 13 17 18 19 20 18 22 20 24 25 12 27 28 29 0 28 32 33 34 35 22 37 0 39 0 0 42 43 0 42 46 0 48 46 50 12 0 0 15 0 42 43 19 45 46 0 23 0
0 0 0 0 0 0 0 0 0 0 0 36 0 0 0 0 0 42 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0
```

Figure 2: Размер 100



```
0 1 2 0 4 5 0 7 8 7 8 0 12 13 2 15 16 17 12 17 20 21 22 21 24 19 20 15 28 29 30 29 32 21 34 0 36 37 38 39 40 39 42 43 38 45 0 47 48 43 50 49 22 41 42 49 36 57 38 59 0 5
5 62 57 62 65 66 67 68 39 68 41 72 73 74 73 56 0 78 79 78 81 80 83 84 85 84 87 88 87 90 85 92 93 94 93 0 91 92 99 100 101 102 103 104 105 50 107 102 109 110 105 70 0 11
2 73 116 117 28 119 120 121 80 117 124 119 106 121 38 0 74 0 0 121 0 0 80 0 0 49 98 0 0 0 0 0 56 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

##################################################
0 1 2 0 4 5 0 7 8 0 7 8 12 13 0 15 16 17 15 19 20 21 22 20 21 22 12 24 28 29 30 17 32 0 34 21 36 37 38 39 40 38 42 43 41 45 43 47 48 49 50 0 38 39 0 55 56 57 0 59 57 0
62


62 49 50 62 66 67 68 0 70 68 72 73 74 36 73 38 78 79 80 81 0 83 84 85 2 87 88 0 90 91 78 93 94 92 93 0 84 99 100 87 102 103 104 105 92 107 105 109 110 0 73 29 0 101 102
 117 0 119 120 121 119 39 121 41 42 124 0 0 116 117 0 49 0 121 0 98 0 0 56 0 0 0 0 106 0 0 0 65 0 112 0 0 0 0 0 0 74 0 0 0 0 0 80 0 0 0 0 85 0 0 0 0 0 0 0 0 0 0 0 0 0 2
8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 80 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 3: Размер 250



```
0 1 0 0 4 0 4 7 8 7 8 9 10 13 12 15 4 15 18 7 20 21 22 21 22 13 24 0 28 29 30 31 32 0 32 0 36 0 38 39 38 39 42 43 42 39 0 41 48 49 50 49 22 41 42 53 36 57 56 59 54 61 6
2 57 58 9 64 67 38 39 0 65 42 73 74 73 74 75 78 49 78 81 80 83 84 85 84 85 88 89 60 91 92 93 38 95 96 97 98 87 94 101 102 101 32 105 106 107 108 109 110 99 110 101 94 1
03 116 117 118 119 120 121 110 123 124 119 54 127 128 127 130 125 90 133 134 0 0 137 66 137 140 141 142 101 142 145 146 145 146 149 144 151 0 153 154 125 156 151 156 15
9 160 161 162 163 164 165 0 0 168 163 170 169 172 173 174 175 174 157 172 149 180 181 182 181 0 0 144 187 186 169 190 191 192 193 192 193 196 197 198 197 200 195 202 20
3 204 205 206 205 208 189 210 169 212 213 212 215 196 217 218 219 218 201 222 217 224 225 0 227 228 229 230 225 0 191 234 235 234 237 236 209 240 235 242 223 242 225 24
6 217 246 229 250 119 250 43 0 199 0 47 0 49 0 151 0 0 192 209 0 0 158 0 88 139 32 0 0 0 0 0 0 39 0 0 0 0 0 0 0 0 0 4


6 217 246 229 250 119 250 43 0 199 0 47 0 49 0 151 0 0 192 209 0 0 158 0 88 139 32 0 0 0 0 0 0 39 0 0 0 0 0 0 0 0 0 49 180 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 39 0
0 158 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

##################################################
0 1 0 0 4 0 0 7 8 9 7 8 9 13 0 15 13 0 18 0 20 21 22 0 21 22 12 24 28 29 30 31 32 0 0 32 36 0 38 39 0 41 42 43 41 42 7 47 48 49 50 0 38 39 54 0 56 57 58 59 60 22 62 49
61 65 66 67 54 0 0 32 0 73 74 36 73 74 78 0 80 78 43 83 84 85 0 84 85 75 87 91 53 93 94 92 96 97 98 99 0 101 102 89 101 105 106 107 108 109 110 0 0 110 0 101 116 103 11
8 119 120 121 119 39 124 125 42 127 125 90 130 128 0 133 134 0 0 137 0 139 140 141 142 0 144 145 146 144 145 146 0 137 149 153 154 0 142 157 119 159 160 161 162 163 164
 165 163 0 168 169 170 0 172 173 174 175 0 174 94 95 180 181 182 169 181 101 172 187 149 189 190 191 192 193 39 195 193 197 198 196 197 117 202 200 204 205 206 123 205
209 210 127 212 213 0 215 0 217 218 219 217 218 208 223 224 225 212 227 228 229 230 217 229 0 234 151 222 237 235 225 156 0 242 88 0 242 246 0 234 246 250 0 0 169 240 0
 0 0 0 0 0 0 4 0 250 110 0 0 10 0 186 0 0 15 235 236 192 0 0 0 196 0 0 199 0 201 0 203 0 0 32 0 0 209 0 0 38 39 0 0 42 0 0 0 0 0 151 49 0 225 0 156 0 158 0 57 0 0 0 0 0
 0 64 0 0 0 0 0 0 0 0 0 0 0 0 180 0 0 0 81 0 0 0 0 0 0 88 192 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 49 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 4 0 0 0 0 0 151 0 0 0 0 0 0 158 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 38 39 0 0 0 0 0 0 0 0 0 49 191 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 4: Размер 500

5 Выводы

По скриншотам, представленным выше, видно, что более равномерного распределения кубическое пробирование не предоставляет, что подтверждает мою изначальную гипотезу