ТЕМА ЗА ПРОЕКТ № 5

Хештаблица

Задача:

Напишете програма, която получава текст и може бързо да отговаря на множество заявки от вида “колко пъти се среща конкретна дума в текста”. **За реализирането имплементирате своя хеш таблица използвайки алгоритъма Linear Probing, като след реализирането и направете множество тестове които сравняват ефективността на алгоритъма с реализацията на хештаблица в c++. Намерете случаи, в които реализираният от вас алгоритъм е по-ефективен(или по-неефективен) и пробвайте да обясните защо мислите, че се получава така.**

Вход:

Нека M е просто число – въвежда се преди започване на програмата. Намира се в началото на файла – 8-ми ред.

Нека текстът, който се въвежда е на латиница, само с малки букви, без пунктуация.

Направени тестове и резултати:

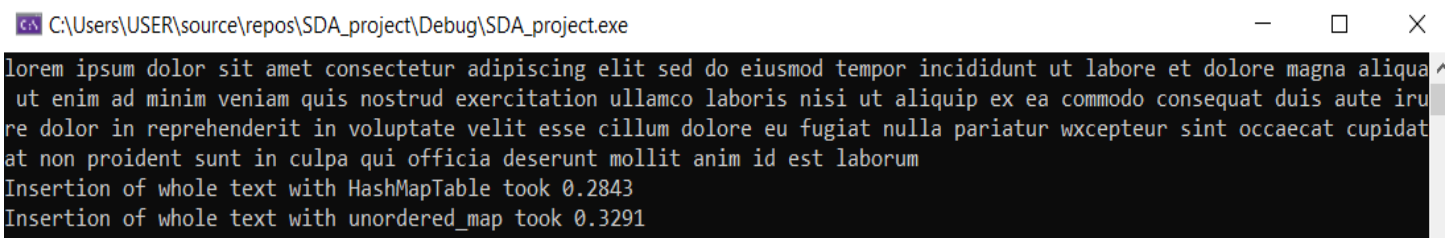
За провеждане на тестовете е използван *“lorem_ipsum.txt”* – извадка от главен текстов файл с намалено съдържание на думи, започващи само с малки букви, без пунктуация, като първоначално се задава по-голям размер на таблицата – в случая 10009:

lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur wxcepteur sint occaecat cupidatat non proident sunt in culpa qui officia deserunt mollit anim id est laborum

За скорост на работа на Hashtable и unordered_map:

- **Добавяне на всички думи от въведения текст:**

  Резултатите, които получаваме от тестовете са следните:



- **Добавяне на една дума към текста:**
  Резултатите, които получаваме от тестовете са следните:

```
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter the word to be inserted: we
Inserting of one element with HashMapTable took 0.0105
Inserting of one element with unordered_map took 0.0117
```

Разликата в милисекундите за функцията insert в полза на *HashMapTable* се дължи на следното: веднъж заделена необходимата памет за елементите на *HashMapTable*, в нея може много по-бързо да се добавят нови елементи чрез присвоявания към член-данните на вече съществуващ обект от масива, тъй като при *separate chaining* се налага заделяне на нова памет за всеки нов елемент, което забавя процеса.

Но когато зададем малък размер – например 5 елемент, след единично въвеждане на втора дума се случва следното:

```
we
Insertion of whole text with HashMapTable took 0.0396
Insertion of whole text with unordered_map took 0.008
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter the word to be inserted: are
Inserting of one element with HashMapTable took 12879.2
Inserting of one element with unordered_map took 2503.22
1.Insert element into the table
2.Search element from the key
3.Delete element at a key
4.Exit
Enter your choice: 1
Enter the word to be inserted: the
Inserting of one element with HashMapTable took 34482.8
Inserting of one element with unordered_map took 0.0145
```

Тази огромна разлика в милисекундите се дължи на преоразмеряването в HashMapTable заради копирането на всеки един обект от стария масив в новия, реализирано чрез *linear probing*. В *unordered_map* имплементацията на хештаблицата е базирана на *separate chaining* - динамичен масив от свързани списъци, което позволява по-бързо преоразмеряване на таблицата, тъй като при него не се извършва заделяне на нова памет за всеки неин елемент (двойка ключ-стойност), а може просто указателите към вече създадени обекти да бъдат поставени в нови списъци на съответните нови позиции след преоразмеряването.

<u>При размер на таблицата 10009:</u>

- **Търсене на дума в текста:**

  Резултатите, които получаваме от тестовете са следните:

```
Enter your choice: 2
Enter the word to be searched: id
The number of the word id searched with HashMapTable  is: 1
Searching with HashMapTable took 0.3408
The number of the word id searched with unordered_map  is: 1
Searching with unordered_map took 0.2241
```

- **Изтриване на дума от текста:**

  Резултатите, които получаваме от тестовете са следните:

```
Enter your choice: 3
Enter the word to be deleted: id
Deleting with HashMapTable took 0.3002
Deleting with unordered_map took 0.0065
```

Тази огромна разлика в милисекундите отново се дължи на преоразмеряването в HashMapTable.

Нека сега да променим размера от 10009 на <u>509</u>. Тогава се случва следното:

```
Enter your choice: 3
Enter the word to be deleted: id
Deleting with HashMapTable took 0.0106
Deleting with unordered_map took 0.0107
```

Пренебрежимо малка разлика, дължаща се на факта, че в този тест не се стига до преоразмеряване на таблицата.

ИЗВОД:

Когато знаем приблизително броя елементи, които ще бъдат добавени в хештаблицата, и можем предварително да я преоразмерим според минимално необходимите позиции, *linear probing* е подходяща имплементация за хештаблица. В този случай няма да бъде предизвикано преоразмеряване на хештаблицата, което е най-бавният процес в нея заради големия брой копирания на памет на елементи от един масив в друг.