Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет
по лабораторной работе № 7

Дисциплина: Анализ алгоритмов

Тема: Поиск в словаре

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Волкова Л. Л.
Строганов Ю. В.

Москва, 2020 г.

Оглавление

Введение

В данной лабораторной работе требуется изучить и реализовать алгоритмы поиска слова в словаре. Так, требуется реализовать алгоритм полного перебора, двоичного поиска и оптимизированный алгоритм поиска на основе сегментов. Также необходимо выполнить анализ трудоемкости и сравнительный анализ времени работы для каждой из реализаций.

1. Аналитическая часть

В данном разделе будут рассмотрены понятия, связанные с поиском слова в словаре.

1.1. Структура словаря

Словарь (ассоциативный массив) — структура данных, позволяющая идентифицировать ее элементы как по числовому индексу, так и по произвольному.

Итак, каждый элемент словаря состоит из двух объектов: ключа и значения. В поставленной задаче ключом является порядковый номер слова, значением является само слово. Ключ идентифицирует элемент словаря, значение является данными, которые соответствуют данному ключу. Значения ключей — уникальны, двух одинаковых ключей и значений в словаре быть не может. [1]

Таким образом, задача поиска слова в словаре сводится к задаче поиска ключа. В данной работе будут рассмотрены следующие алгоритмы поиска.

1.2. Алгоритм полного перебора

Под полным перебором понимается методика разрешения задач путем рассмотрения всех возможных вариантов. Уровень сложности при полном переборе напрямую связан с количеством допустимых решений задачи. Так, в худшем случае (искомое слово последнее или отсутствует), требуется выполнить N сравнений в словаре размерности N.

Хотя алгоритм полного перебора при решении большинства конкретных проблем (исключая взлом шифрования) практически не используется, существует несколько исключений. Например, в случаях, когда алгоритм полного перебора всё-таки окажется оптимальным или он является первым шагом в создании общего алгоритма, его применение будет оправданным. [2]

1.3. Алгоритм двоичного поиска

Алгоритм двоичного поиска заключается в том, что на каждом шаге множество объектов делится на две части и в работе остаётся та часть множества, где находится искомый объект. Или же, в зависимости от постановки задачи, мы можем остановить процесс, когда мы

получим первый или последний индекс вхождения элемента. Последнее условие — это левосторонний/правосторонний двоичный поиск. [3]

Двоичный поиск производится в упорядоченном массиве. Двоичный или бинарный поиск также называют поиском методом деления отрезка пополам или дихотомии.

Обход словаря можно рассматривать, как обход бинарного дерева. Следовательно, трудоемкость этого алгоритма в худшем случае составляет $\log_2 N$ (проход по дереву от корня до листа).

1.4. Алгоритм поиска с сегментами

Алгоритм поиска в словаре с сегментами является оптимизированным алгоритмом поиска в словаре, предварительно разбитом на сегменты. Каждый сегмент представляет из себя множество слов, начинающихся с одинаковой буквы. Также можно рассматривать и дальнейшее разбиение сегмента – по второй букве. Тогда, в случае сегментирования, поиск слова состоит из двух частей: поиск нужного сегмента и поиск ключа в пределах сегмента. Задачу поиска требуемого сегмента можно оптимизировать, произведя сортировку сегментов в соответствии с результатами частотного анализа.

1.5. Вывод

В данном разделе были рассмотрены понятия, связанные с поиском слова в словаре и разобраны алгоритмы, реализующие поиск: классические алгоритмы (полный перебор, двоичный поиск) и оптимизированный (поиск с сегментами).

2. Конструкторская часть

В данном разделе будут рассмотрены схемы и оценена трудоемкость для каждой из реализаций алгоритма поиска слова в словаре.

2.1. Разработка алгоритмов

На рисунке 1 изображена схема *алгоритма полного перебора* слов в словаре.

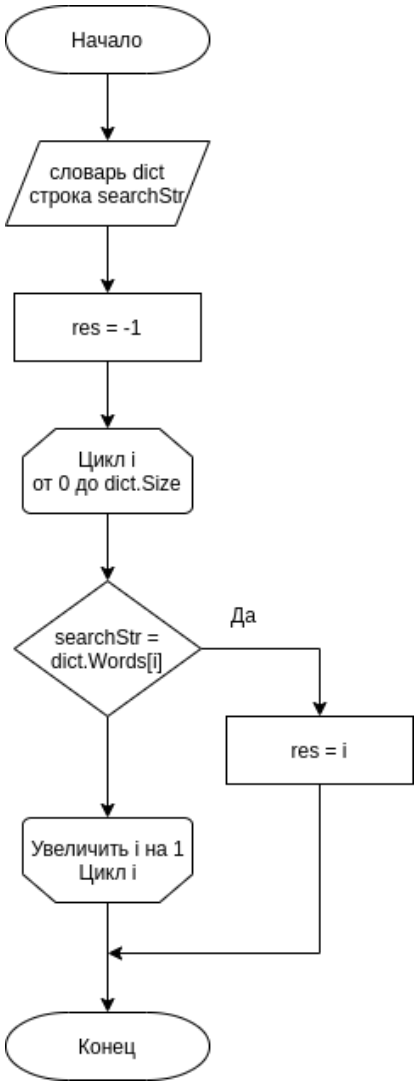

Рисунок 1. Схема алгоритма полного перебора.

На рисунке 2 изображена схема *алгоритма двоичного поиска* в словаре.

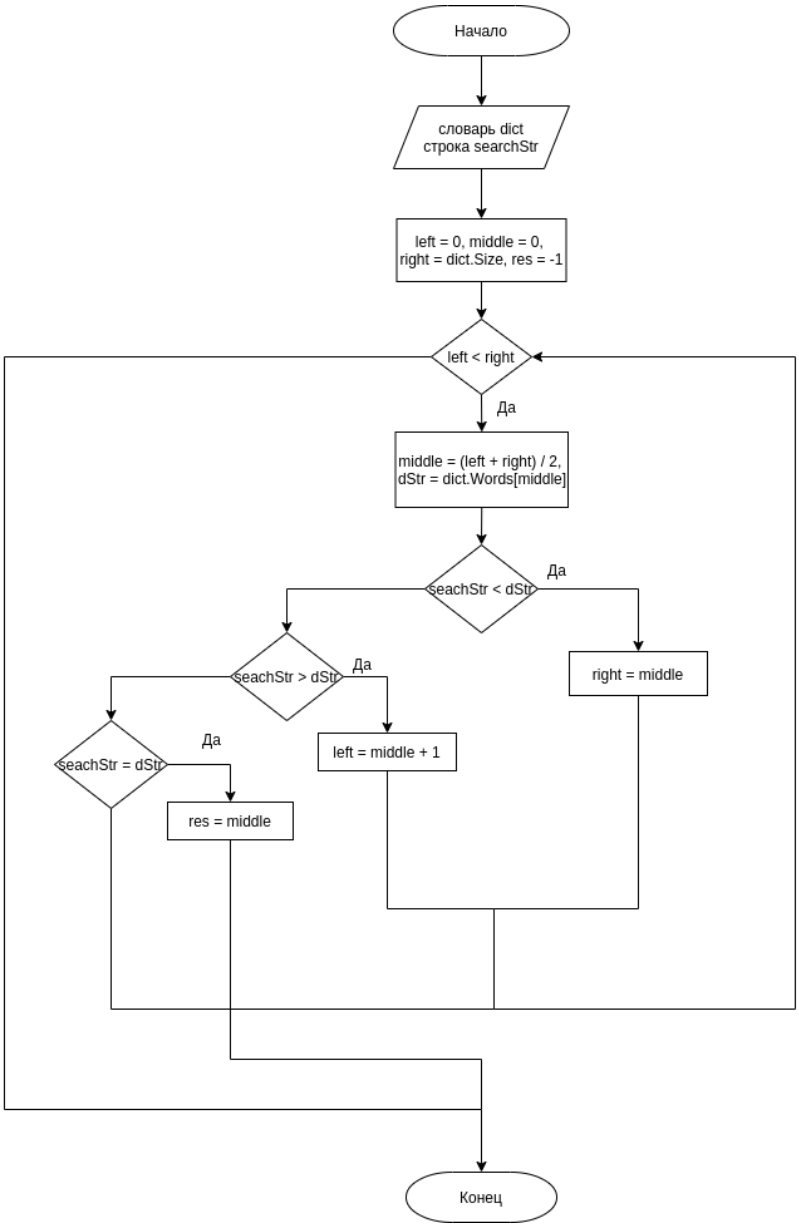

Рисунок 2. Схема алгоритма двоичного поиска.

На рисунке 3 изображена схема *алгоритма поиска с сегментами* в словаре.

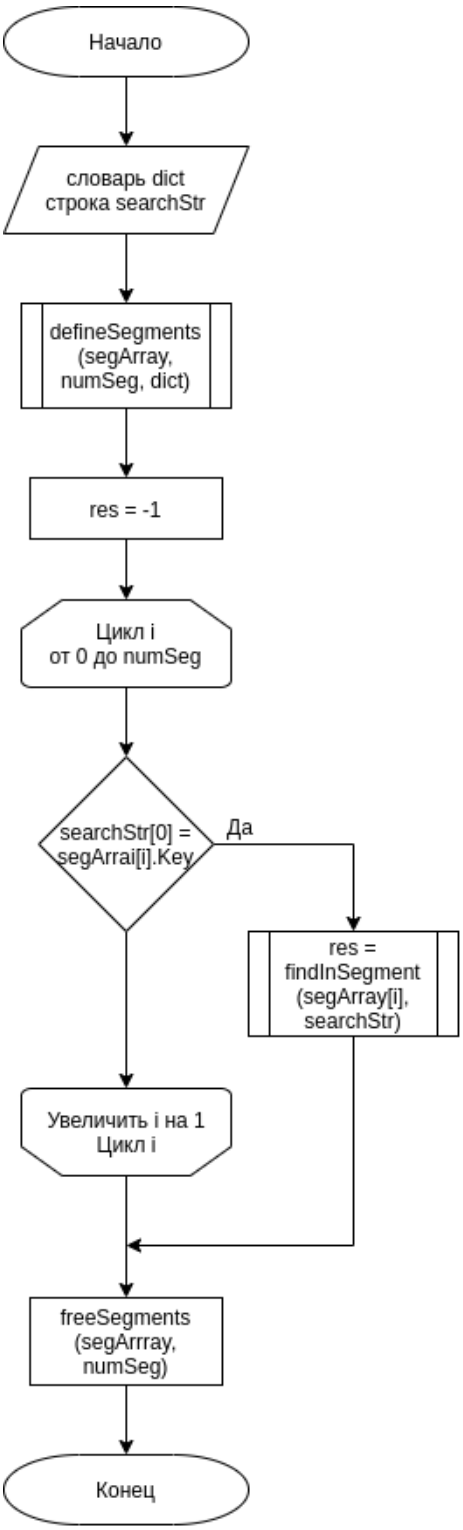

Рисунок 3. Схема алгоритма поиска сегмента.

Поиск в сегменте осуществляется с помощью алгоритма двоичного поиска (рис.2).

2.2. Анализ трудоемкости алгоритмов

Необходимо оценить трудоемкость для каждого алгоритма в трех случаях: лучший (слово найдено на первом сравнении), средний, худший (слово не найдено/найдено на последнем сравнении).

Пусть k_0 – количество операция на старте алгоритма, k_1 – количество операций при каждом сравнении, N – размерность словаря.

2.2.1. Трудоемкость алгоритма полного перебора

Лучший случай: $k_0 + k_1$

Худший случай: $k_0 + Nk_1$

Средний случай: $k_0 + k_1(1 + \frac{N}{2} - \frac{1}{N+1})$

2.2.2. Трудоемкость алгоритма двоичного поиска

Лучший случай: $k_0 + k_1$

Худший случай: $\log_2 N$

2.2.3. Трудоемкость алгоритма поиска с сегментами

Средний случай: $\sum_{i=1}^{N}(f_{\text{выбор сегмента}} + f_{\text{поиск в сегменте}})\, p_i$, где

p_i – вероятность того, что ключ окажется в i сегменте.

2.3. Вывод

В данном разделе были рассмотрены 3 схемы алгоритма поиска в словаре: полного перебора, двоичного поиска, поиска с сегментами. В случае сегментирования словаря задача поиска разбивается на две части: поиск сегмента и поиск в сегменте. Поиск в сегменте осуществляется с помощью алгоритма двоичного поиска.

3. Технологическая часть

В данном разделе будет рассмотрен язык программирования, среда разработки, требуемые инструменты для реализации. Также будет представлена реализация алгоритмов.

3.1. Требования к программному обеспечению

I. Программа должна предусматривать ввод слова произвольной длины.
II. Выбор применяемого алгоритма осуществляется пользователем из списка алгоритмов, предложенных в меню.
III. На выходе программа выводит индекс и слово, располагающееся по найденному индексу.
IV. Также необходимо предусмотреть выполнение замеров процессорного времени для каждой из реализаций.

3.2. Средства реализации

В данной работе используется язык программирования C++, из-за удобства хранения строк, представления словаря в виде объекта класса (сокращает количество аргументов для передачи в функцию) и опыта написания на нем. Среда разработки – Qt.

Для замеров процессорного времени использовалась функция clock(). [4]

3.3. Реализация алгоритмов

Реализация алгоритма полного перебора представлена в листинге 1.

Листинг 1. Алгоритм полного перебора.

```
#include "bruteForce.h"

int bruteForce(const Dictionary dict, const string searchStr)
{
   int res = -1;
   for (int i = 0; i < dict.Size; i++) {
      if (searchStr == dict.Words[i]) {
         res = i;
         break;
      }
   }
   return res;
}
```

Реализация алгоритма двоичного поиска представлена в листинге 2.

Листинг 2. Алгоритм двоичного поиска.

```
#include "binarySearch.h"

int binarySearch(const Dictionary dict, const string searchStr)
{
   int left = 0, middle = 0, right = dict.Size;
   int res = -1;

   while (left < right) {
      middle = (left + right) / 2;

      string dStr = dict.Words[middle];

      if (searchStr < dStr) {
         right = middle;
      } else {
         if (searchStr == dStr) {
            res = middle;
            break;
         } else {
            if (searchStr > dStr) {
               left = middle + 1;
            }
         }
      }
   }

   return res;
}
```

Реализация алгоритма поиска с сегментами представлена в листинге 3.

Листинг 3. Алгоритм поиска с сегментами.

```
#include "segmentSearch.h"

const string alphabet = "abcdefghijklmnopqrstuvwxyz";

int segmentSearch(const Dictionary dict, const string searchStr)
{
   const int numSegments = alphabet.size();
   Segment segmentArray[numSegments];
   defineSegments(segmentArray, numSegments, dict);

   int res = -1;

   for (int i = 0; i < numSegments; i++) {
      if (searchStr[0] == segmentArray[i].Key) {
         res = findInSegment(segmentArray[i], searchStr);
         break;
      }
   }

   freeSegemnts(segmentArray, numSegments);
```

```
    return res;
}


int findInSegment(const Segment segment, const string searchStr)

{
    int left = 0, middle = 0, right = segment.Size;
    int res = -1;

    while (left < right) {
        middle = (left + right) / 2;

        string dStr = segment.Words[middle];

        if (searchStr < dStr) {
            right = middle;
        } else {
            if (searchStr == dStr) {
                res = segment.OriginalIdx[middle];
                break;
            } else {
                if (searchStr > dStr) {
                    left = middle + 1;
                }
            }
        }
    }

    return res;
}
```

3.4. Описание тестирования

Тестирование осуществляется по принципу «черного ящика».

Для проверки корректности программы необходимо предусмотреть 4 случая расположения искомого слова: первое, произвольное, последнее и отсутствие искомого слова.

3.5. Вывод

В данном разделе были рассмотрены инструменты, необходимые для реализаций версий алгоритмов поиска в словаре, а также были представлены непосредственно реализации.

4. Экспериментальная часть

В данном разделе будут рассмотрены примеры работы программы, произведено тестирование, выполнены эксперименты по замеру времени, а также выполнен сравнительный анализ полученных данных.

4.1. Примеры работы программы

Взаимодействие с программой (меню, ввод слова, выбор алгоритма, вывод результата) представлено на рисунке 4.

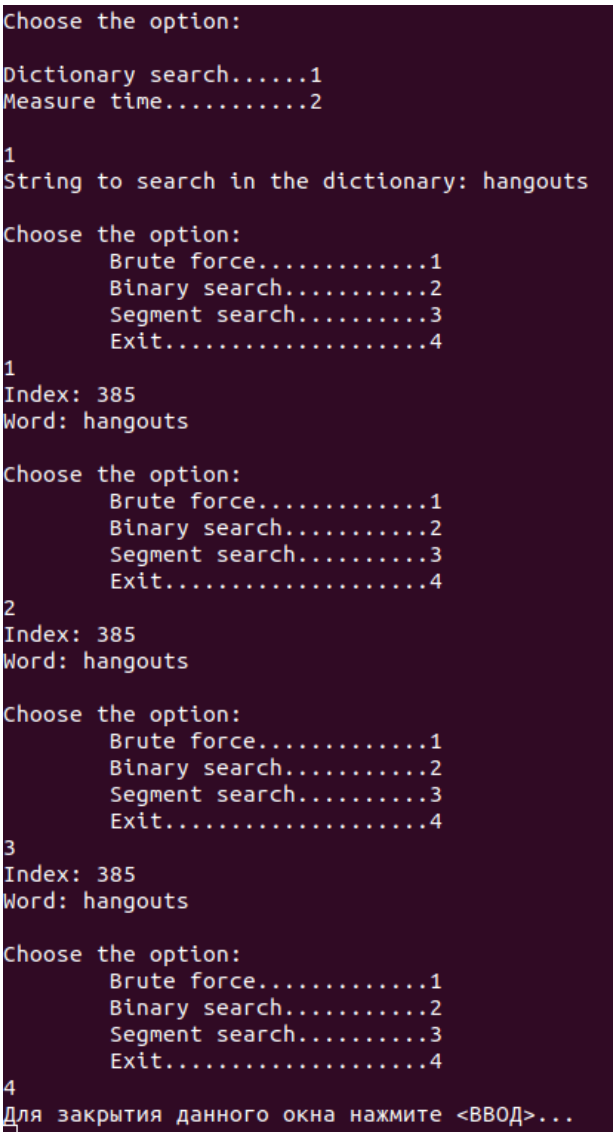

Рисунок 4. Пример взаимодействия с программой.

4.2. Технические характеристики устройства.

Операционная система – LinuxMint 64-bit;

Память – 8 ГБ;

Процессор – Intel™ Core™ i3-7100U CPU @ 2.40 ГГц;

Логических процессов – 4.

4.3. Результаты тестирования

Тестирование выполнялось на словаре размерность 958 строк, содержащем английские слова. Результаты тестирования приведены в таблице 4.

Таблица 4. Результаты тестирования.

Искомая строка	**Индекс**	**Полученное слово**
aadvark	0	aadvark
zonally	958	zonally
less	481	less
boat	-	“Word not found”

Поскольку результаты работы алгоритмов совпадают, в столбце «Полученный результат» приведено значение, одинаковое для всех реализаций.

Все тесты были пройдены успешно.

4.4. Сравнение реализаций по времени работы

Выполняется серия экспериментов по замеру времени. Эксперименты проводятся для N + 1 случаев, где N – размерность словаря. Эксперименты проводятся по 100 раз для усреднения результатов и сглаживания ошибок. Результаты замеров приведены на рисунке 4.1.

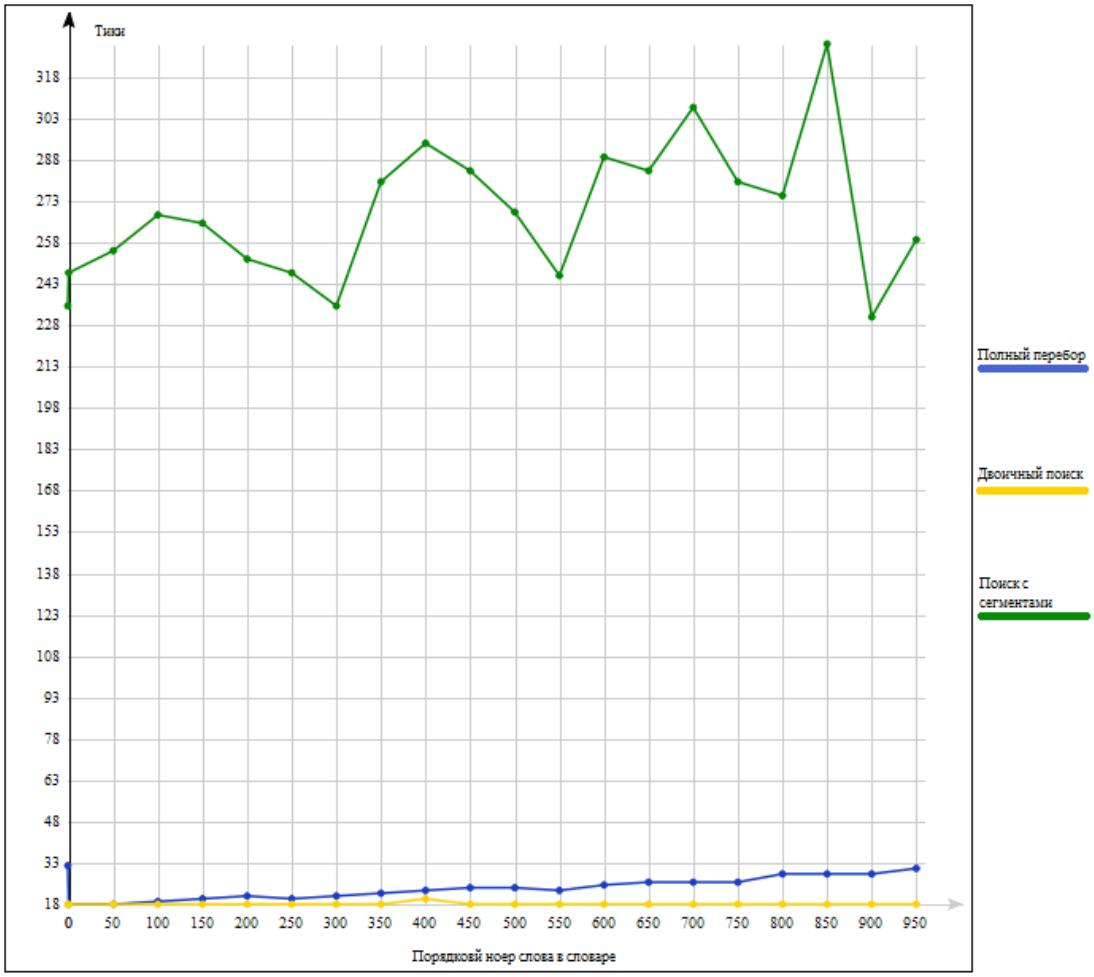

Рисунок 4.1. График зависимости времени работы алгоритмов от порядкового номера искомого слова.

Эксперименты проводились с учетом затрат на формирование и сортировку сегментов в алгоритме поиска с сегментами.

Следует отметить, что самым долгим является алгоритм сортировки с сегментами. Это связано с тем, что накладные расходы превышают выгоду по времени. Также стоит обратить внимание на нелинейность графика этого алгоритма, вызванную предварительной сортировкой сегментов на основе частотного анализа. Самым быстрым алгоритмом является алгоритм двоичного поиск. График алгоритма является константой. Что касается полного перебора, алгоритм работает дольше при среднем и худшем случаях. При поиске первого слова в словаре, он не уступает по времени двоичному поиску.

4.5. Вывод

В данном разделе были рассмотрены примеры работы программы, произведено тестирование, выполнены эксперименты по замеру времени, а также выполнен сравнительный анализ полученных данных.

Заключение

Цель работы достигнута, все поставленные задачи выполнены: были разработаны и реализованы три алгоритма поиска в словаре, а также проведен анализ времени работы для каждой реализации.

В результате анализа было установлено, что максимальную скорость работы имеет классический алгоритм поиска – двоичный поиск. Алгоритм полного переборы работает медленнее при худшем случае: в 2 раза замедляется время работы. При лучшем случае время работы алгоритмов разнится незначительно.

Самым долгим алгоритмом является алгоритм с сегментами, он работает более, чем в 10 раз медленнее, поскольку дополнительные расходы на сегментацию словаря превышают выгоду по времени. Для организации работы алгоритма с сегментами, требуется не только создать массив сегментов, но и выполнить частотный анализ, позволяющий выполнить сортировку сегментов.

Список литературы

1. Шуйкова И. А. Структуры данных. Словари. Множества. – М.: МЦНМО, 2016. Второе издание, исправленное и дополненное.
2. Воеводин В. В. и др.,. Параллельные вычисления. — СПб: БХВ-Петербург, 2002. — Р. 608. — ISBN 5-94157-160-7.
3. Левитин А. В. Глава 4. Метод декомпозиции: Бинарный поиск // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 180—183. — 576 с. — ISBN 978-5-8459-0987-9
4. Техническая документация.-URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019 (дата обращения: 20.09.2020). Текст: электронный.