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»
ДИСЦИПЛИНА «Анализ алгоритмов»

Лабораторная работа № 7

Дисциплина Анализ алгоритмов
Тема Поиск в словаре
Студент Боренко А. Д.
Группа ИУ7-52Б
Оценка (баллы) _____
Преподаватель Волкова Л.Л.

Москва — 2022г.

Содержание

Введение

Ассоциативный массив - абстрактный тип данных (интерфейс к хра- нилищу данных), позволяющий хранить пары вида $\frac{3}{4}$(ключ, значение)¿ и под- держивающий операции добавления пары, а также поиска и удаления пары по ключу:

1. INSERT(ключ, значение)
2. FIND(ключ)
3. REMOVE(ключ)

Предполагается, что ассоциативный массив не может хранить две па- ры с одинаковыми ключами. В паре (k, v) значение v называется значением, ассоциированным с ключом k. Где k - это key, а v - value. Операция FIND(ключ) возвращает значение, ассоциированное с за- данным ключом, или некоторый специальный объект UNDEF, означающий, что значения, ассоциированного с заданным ключом, нет. Две другие опера- ции ничего не возвращают (за исключением, возможно, информации о том, успешно ли была выполнена данная операция). Ассоциативный массив с точки зрения интерфейса удобно рассматри- вать как обычный массив, в котором в качестве индексов можно использовать не только целые числа, но и значения других типов - например, строки. Поддержка ассоциативных массивов есть во многих интерпретируе- мых языках программирования высокого уровня, таких, как Perl, PHP, Python, Ruby, Tcl, JavaScript и других. Для языков, которые не имеют встроенных средств работы с ассоциативными массивами, существует множество реализаций в виде библиотек. Примером ассоциативного массива является телефонный справочник: значением в данном случае является совокупность "Ф. И. О. + адрес а ключом - номер телефона, один номер телефона имеет одного владельца, но один человек может иметь несколько номеров. Три основных операции часто дополняются другими, наиболее попу- лярные расширения:

1. CLEAR удалить все записи,

2. EACH "пробежаться"по всем хранимым парам,

3. MIN найти пару с минимальным значением ключа,

4. MAX найти пару с максимальным значением ключа.

В последних двух случаях необходимо, чтобы на ключах была опре- делена операция сравнения.

Целью данной лабороторной работы является изучение способа эф- фекливного по времени и памяти поиска по словарю. Задачами данной лабораторной являются:

1. исследование алгоритмы поиска по словарю;

2. проведение тестирования работы алгоритмов в лучшем, худшем и произвольном случае;

3. замеры процессорного времени работы алгоритмов поиска пословарю для каждого ключа и для отсутствующего ключа, вывести минимальное, максимальное и среднее время поиска ключа;

4. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1. Аналитическая часть

В данном разделе будут рассмотрены алгоритмы поиска в словаре: полный перебор, бинарный поиск, частотный анализ.

1.1 Алгоритм полного перебора

Идея алгоритма заключается в том, что поиск заданного элемента из множества происходит непосредственно сравниванием каждого элемента этого множества с искомым, до тех пор, пока искомый не найдётся или множество не закончится. Сложность алгоритма линейно зависит от объёма словаря, а время может стремиться к экспоненциальному времени работы.

1.2 Алгоритм бинарного поиска

Данный алгоритм содержит в себе идею, которая заключается в том, что берётся значение ключа из середины словаря и сравнивается с данным. Если значение меньше (в контексте типа данных) данного, то продолжается поиск в левой части словаря, при обратном случае - в правой. На новом интервале также берётся значение ключа из середины и сравнивается с данным. Так продолжается до тех пор, пока найденное значение ключа не будет равно данному.

Поиск в словаре с использованием данного алгоритма в худшем случае будет иметь трудоёмкость O(log2N), что быстрее поиска при помощи алгоритма полного перебора. Но стоит учитывать, что алгоритм бинарного поиска работает только для заранее отсортированного словаря. В случае большого объёма словаря и обратного порядка сортировки, может произойти так, что алгоритм полного перебора будет эффективнее по времени.

1.3 Алгоритм частного анализа

Идея алгоритма заключается в составлении частотного анализа. Чтобы провести частотный анализ, необходимо взять первый элемент каждого значения в словаре по ключу и подсчитать частотную характеристику, т.е. сколько раз этот элемент встречается в качестве первого элемента. По полученным значениям словарь разбивается на сегменты так, что все элементы с одинаковым первым элементом оказываются в одном сегменте. Далее сегменты упорядочиваются по значению частотной характеристики таким образом, чтобы элементы с наибольшей частотной характеристикой был самый быстрый доступ. Далее каждый сегмент упорядочивается по значению. Это необходимо для реализации бинарного поиска, который обеспечит эффективный поиск в сегменте при сложности O(N logN)

1.4 Вывод аналитической части

В данной работе стоит задача реализации следующих алгоритмов поиска по словарю: полный перебор, бинарный поиск, частотный анализ. Необходимо сравнить алгоритмы по эффективности по времени.

2. Конструкторская часть

В данном разделе представлены схемы алгоритмов. Так же будут описаны пользовательские структуры данных, приведены классы эквивалентности для тестирования реализуемого ПО.

2.1 Схемы алгоритмов

2.1.1 Схема алгоритма полного перебора

На рисунке 2.1 показана схема алгоритма полного перебора.

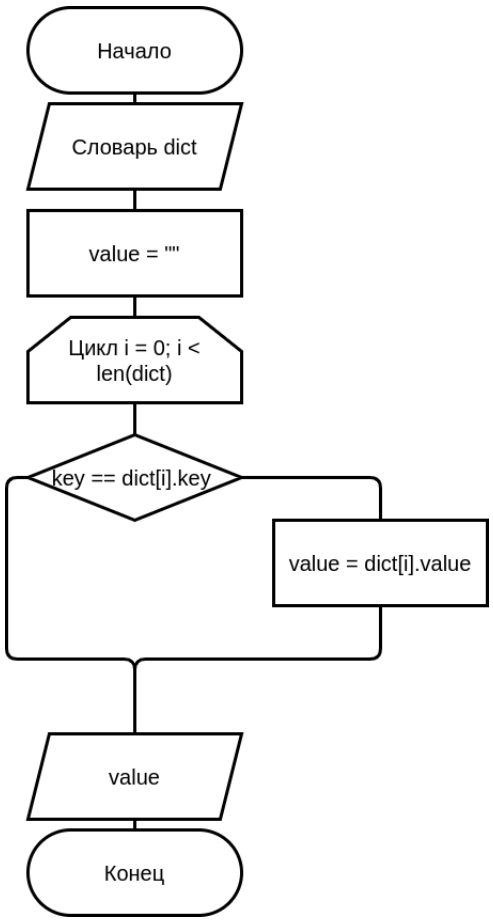

Рис. 2.1: Схема алгоритма полного перебора

2.1.2 Схема алгоритма бинарного поиска

На рисунке 2.2 показана схема алгоритма бинарного поиска.

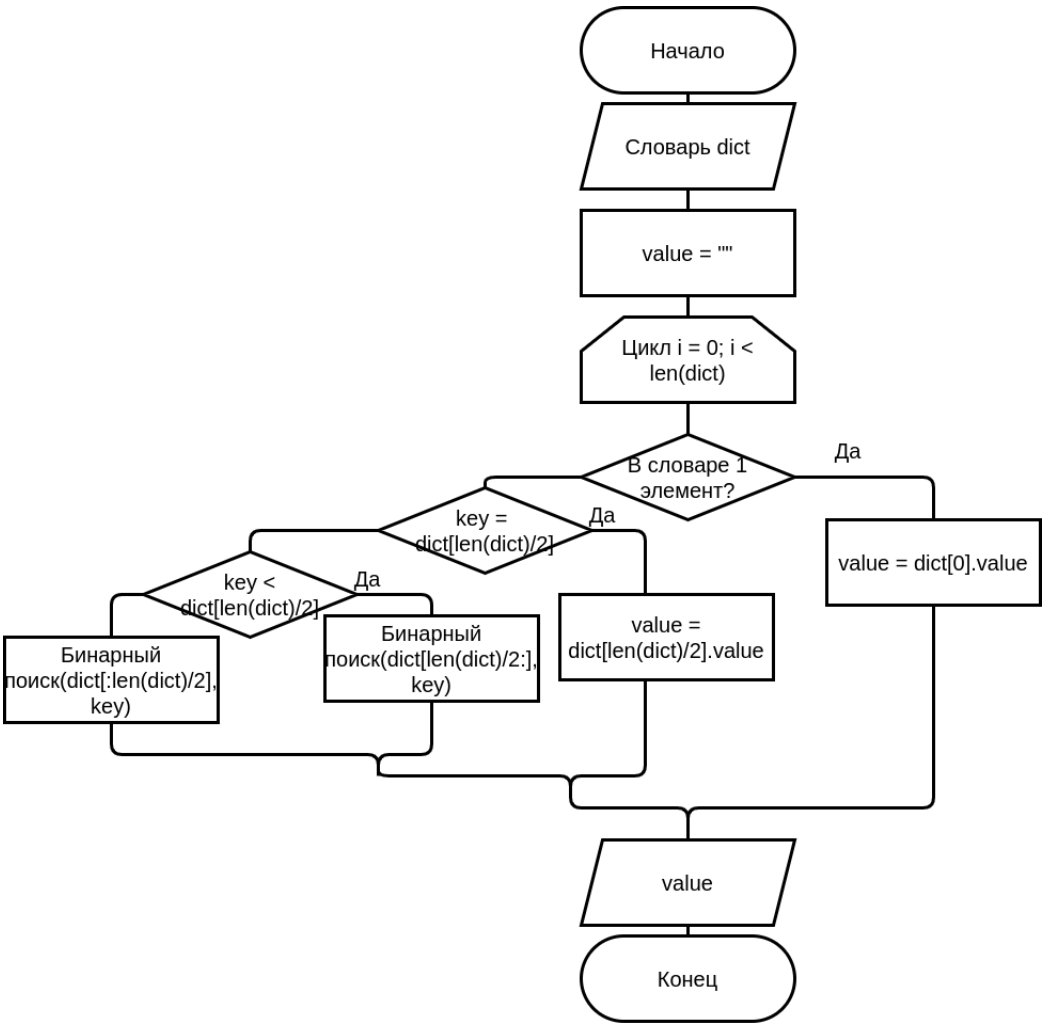

Рис. 2.2: Схема алгоритма бинарного поиска

2.1.3 Схема алгоритма частотного анализа

На рисунке 2.3 показана схема алгоритма частотного анализа.

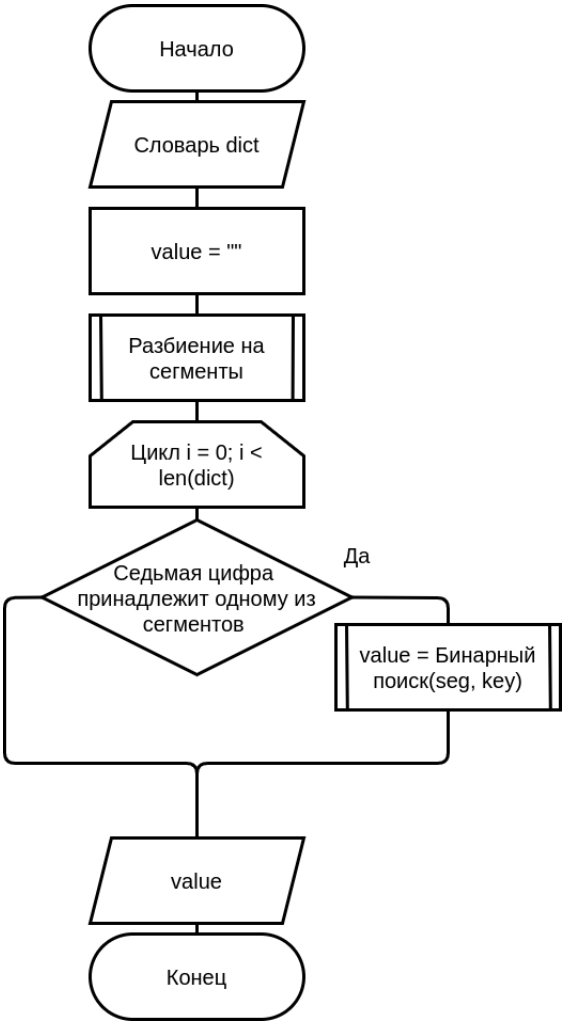

Рис. 2.3: Схема алгоритма частотного анализа

2.2 Структуры данных

При реализации приведенных алгоритмов потребуется тип данных: словарь. Словарь содержит два поля:

1. ключ

2. значение

2.3 Тестирование

Для алгоритмов поиска по словарю можно выделить следующие классы эквивалентности:

1. ключа нет в словаре;
2. первый ключ;
3. последний ключ;
4. случайный ключ:

2.4 Вывод конструкторской части

На основе данных, полученных в аналитическом разделе, были построены схемы используемых алгоритмов, выделены необходимые для реализации структуры данных и методы тестирования.

3. Технологическая часть

3.1 Требования к ПО

Требования к программному обеспечению:

1. на вход подается ключ;

2. результат: значение, соответствующее ключу.

3.2 Выбор языка программирования

Был выбран язык go, поскольку он удовлетворяет требованиям лабораторной работы. Средой разработки выбрана Visual Studio Code.

3.3 Структуры данных

На листинге 3.1 представлено описание структуры словаря.

Листинг 3.1: Структура словаря

```
type person_t struct{
  name string
    snils int
}
```

3.4 Реализация алгоритмов

На листинге 3.2 представлена реализация алгоритма полного перебора.

Листинг 3.2: Реализация последовательного алгоритма нахождения среднего арифметического матрицы

```
func all_search(d dict_t, s int)person_t{
  var p person_t
  p.snils = s

  for i:=0;i<len(d);i++{
    if (s == d[i].snils){
      p.name = d[i].name
      break
    }
  }
  return p
}
```

На листинге 3.3 представлена реализация алгоритма бинарного поиска.

Листинг 3.3: Реализация алгоритма бинарного поиска

```
func bin_search(d dict_t, s int)person_t{
  if (len(d) < 1){
    return person_t{name: "", snils: s}
  }
  var middle = len(d)/2
  if (d[middle].snils < s){
    return bin_search(d[middle:], s)
  } else  if (d[middle].snils > s){
    return bin_search(d[:middle], s)
  }
  return d[middle]
}
```

На листинге 3.5 представлена реализация алгоритма формирования сегментов.

Листинг 3.4: Реализация алгоритма формирования сегментов

```
func make_seg(d dict_t, seg_count int)[][]person_t{
    var seg [][]person_t = make([][]person_t, 10)

  for i:=0;i<10;i++{
```

```
    for j:=0;j<len(d);j++{
      if (d[j].snils / 10000 %10 == i){
        seg[i] = append(seg[i], d[j])
      }
  }}
  return seg
}
```

На листинге 3.5 представлена реализация алгоритма частотного анализа.

Листинг 3.5: Реализация алгоритма частотного анализа

```
func get_seg_search(d dict_t, seg_count int) (func(dict_t, int)person_t){
  seg := make_seg(d, seg_count)
  return func(d dict_t, s int)person_t{
        var p person_t
    var index = s / 10000 %10
    if (index >= len(seg) || index <0){
      p = person_t{name:"", snils:s}
    } else {
      p = bin_search(seg[index], s)
    }
    return p
  }
}
/*
```

3.5 Тестирование

Модульные тесты

В таблице представлены тестовые данные 3.1.

Тесты пройдены.

Таблица 3.1 – Тесты

№	Ввод	Вывод
1	0	Нет элемента.
2	23400005678	Kathleen Johnson
3	23400005678	2
4	23401515678	Donna Weaver
5	23420735678	Kenneth Hunter

3.6 Вывод технологической части

Были реализованы исследуемые алгоритмы, программа прошла тесты и удовлетворяет требованиям.

4. Экспериментальная часть

Оценка качества работы алгоритмов. Экспериментальное сравнение работы различных алгоритмов нахождения среднего арифметического матрицы (зависимость времени выполнения от размерности матриц).

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

1. процессор: Intel® Core™ i3-7100U CPU @ 2.40GHz × 4;
2. память: 11,6 GiB;
3. операционная система: Ubuntu 20.04.1 LTS.

4.2 Примеры работы

На рисунке 4.1 показан пример работы.

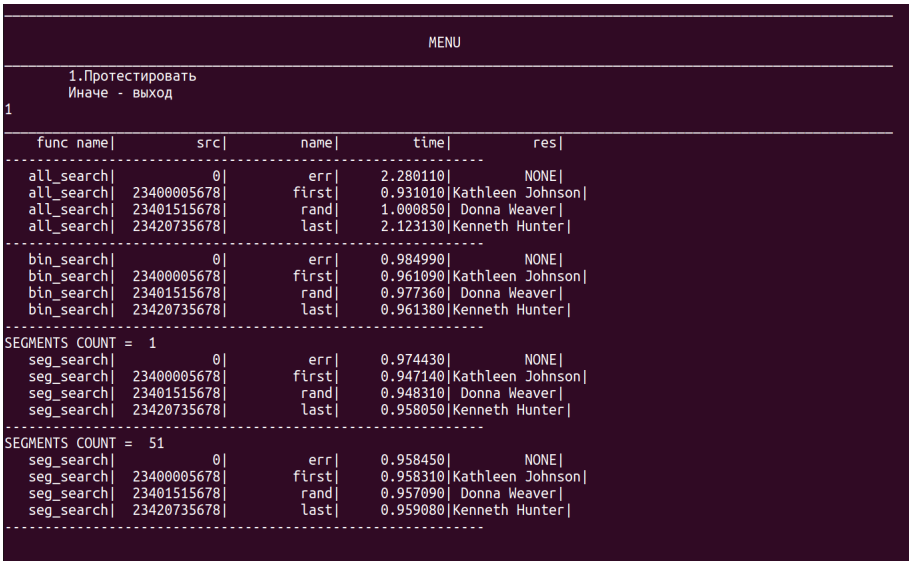

Рис. 4.1: Пример работы 1

4.3 Замеры времени

Так как поиск в словаре считается короткой задачей, воспользуемся усреднением массового эксперимента. Для этого сложим результат работы алгоритма n раз (n >= 10), после чего поделим на n. Тем самым получим достаточно точные характеристики времени. Сравнение произведем при n = 10000

Таблица 4.1 содержит результаты замеров времени. Используются следующие обозначения all_search - алгоритм полного перебора, bin_search - алгоритм бинарного поиска, seg_search-алгоритм поиска по сегментам.

4.4 Сравнительный анализ алгоритмов

По результатам экспериментов можно заключить следующее:

1. представлен результат поиска первого значения. Выигрыш алгоритма поиска полным перебором обосновывается тем, что он тратит лишь одно сравнение для того, чтобы найти первый ключ, в том время, когда бинарный поиск затрачивает гораздо больше сравнений. Частичный анализ работает чуть медленнее, так как ему нужно произвести дополни- тельное сравнение первых букв;

Таблица 4.1 – Замеры времени (в нсек)

Название функции	ключ	Время	Результат
all_search	0	2.280110	NONE
all_search	23400005678	0.931010	Kathleen Johnson
all_search	23401515678	1.000850	Donna Weaver
all_search	23420735678	2.123130	Kenneth Hunter
bin_search	0	0.984990	NONE
bin_search	23400005678	0.961090	Kathleen Johnson
bin_search	23401515678	0.977360	Donna Weaver
bin_search	23420735678	0.961380	Kenneth Hunter
SEGMENTS COUNT = 1			
seg_search	0	0.974430	NONE
seg_search	23400005678	0.947140	Kathleen Johnson
seg_search	23401515678	0.948310	Donna Weaver
seg_search	23420735678	0.958050	Kenneth Hunter
SEGMENTS COUNT = 51			
seg_search	0	0.958450	NONE
seg_search	23400005678	0.958310	Kathleen Johnson
seg_search	23401515678	0.957090	Donna Weaver
seg_search	23420735678	0.959080	Kenneth Hunter

2. представлен результат поиска среднего значения: выигрыш бинарного поиска в том что он находит середину за одну итерацию;

3. представлен результат поиска произвольного ключа: алгоритм полного перебора работает медленнее всех;

4. произведено аналогичное сравнение, только в качестве искомого ключа взят несуществующий. Аналогично поиск полным перебором затрачивает больше всего времени по вышеописанной причине.

4.5 Вывод экспериментальной части

В данном разделе было произведено сравнение трех алгоритмов. По результатам исследования было доказано, что алгоритм полного перебора в основном работает медленнее всех, за исключением, когда ключ лежит достаточно близко к началу

Заключение

В данной работе были изучены алгоритмы поиска по словарю. Получены практические навыки реализации приведенных алгоритмов. Проведён сравнительный анализ алгоритмов по времени. Экспериментально подтверждены различия в эффективности алгоритмов с указанием лучших и худших случаев. Цель работы достигнута, решены поставленные задачи.

Список литературы

[1] Gratzer George A. More Math Into LaTeX. 4th изд. Boston: Birkhauser, 2007.

[2] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: https://www.debian.org/. Дата обращения: 20.12.2021.

[3] Linux – Getting Started [Электронный ресурс]. Режим доступа: https://linux.org. Дата обращения: 20.12.2021.

[4] Макконнел. Дж. Анализ алгоритмов. Активный обучающий подход. – М. Режим доступа:https://www.linux.org.ru/. 2017. 267 с.

[5] Документация языка C++ 98 [Электронный ресурс]. Режим доступа:http://www.open-std.org/JTC1/SC22/WG21/. Дата обращения: 22.12.2021.