

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №7

По курсу: "АНАЛИЗ АЛГОРИТМОВ"

## Поиск в словаре

Работу выполнил: студент группы ИУ7-53Б

Наместник Анастасия

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

*Москва, 2020*

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Понятие словаря . . . . .	4
1.2 Поиск полным перебором . . . . .	5
1.3 Двоичный поиск в упорядоченном словаре . . . . .	5
1.4 Частичный анализ . . . . .	6
1.5 Используемые данные . . . . .	6
1.6 Вывод . . . . .	6
<b>2 Конструкторская часть</b>	<b>7</b>
2.1 Вывод . . . . .	12
<b>3 Технологическая часть</b>	<b>13</b>
3.1 Выбор языка программирования . . . . .	13
3.2 Сведения о модулях программы . . . . .	13
3.3 Тесты . . . . .	16
3.4 Вывод . . . . .	16
<b>4 Исследовательская часть</b>	<b>17</b>
4.1 Характеристики ЭВМ . . . . .	17
4.2 Временные характеристики . . . . .	17
4.2.1 Лучший случай . . . . .	17
4.2.2 Худший случай . . . . .	18
4.2.3 Произвольная позиция в словаре . . . . .	19
4.2.4 Ключ не найден . . . . .	19
4.3 Вывод . . . . .	20

Заключение	21
Список литературы	21

# Введение

В жизни широко распространены словари, например, привычные бумажные словари (толковые, орфографические, лингвистические). В них ключом является слово-заголовок статьи, а значением — сама статья. Для того, чтобы получить доступ к статье, необходимо указать слово-ключ. В программировании принцип устройства словаря тот же самый. В этой лабораторной буду рассмотрены алгоритмы поиска значения по ключу в словаре.

Целью данной лабораторной работы является изучение поиска значения по ключу в словаре с помощью алгоритмов поиска полным перебором, двоичного поиска в упорядоченном словаре и частичного анализа.

В данной лабораторной работе требуется решить пять задач:

- изучить алгоритм поиска в словаре полным перебором;
- изучить алгоритм двоичного поиска в упорядоченном словаре;
- изучить алгоритм частичного анализа для поиска в словаре;
- программно реализовать описанные выше алгоритмы;
- провести сравнительный анализ скорости работы реализованных алгоритмов.

# 1 | Аналитическая часть

В этом разделе будут представлены теоретические сведения, необходимые для программной реализации поиска значения в словаре по ключу.

## 1.1 Понятие словаря

**Словарь** - это ассоциативный массив, позволяющий хранить пары вида «(ключ, значение)» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу. Предполагается, что ассоциативный массив не может хранить две пары с одинаковыми ключами. В паре  $(k, v)$  значение  $v$  называется значением, ассоциированным с ключом  $k$ . Где  $k$  - это *key*, а  $v$  - это *value*. Семантика и названия вышеупомянутых операций в разных реализациях ассоциативного массива могут отличаться[4]. Ассоциативный массив с точки зрения интерфейса удобно рассматривать 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 интерпретируемых языках программирования высокого уровня, таких, как Perl[5], PHP[6], Python[2], Ruby[7], JavaScript[8] и других. Существует несколько вариантов поиска значения в словаре по ключу. Будут рассмотрены три из них:

- поиск в словаре полным перебором;
- двоичный поиск в упорядоченном словаре;
- частичный анализ для поиска в словаре.

## 1.2 Поиск полным перебором

Поиск *полным перебором* подразумевает последовательный проход по словарю со сравнением ключа искомого значения с  $i$ -ым ключом словаря на  $i$ -ом шаге цикла. Возможно  $(N + 1)$  случаев: ключ не найден и  $N$  возможных случаев расположения ключа в словаре. Лучший случай: за одно сравнение ключ найден в начале словаря. Худших случаев 2: за  $N$  сравнений либо элемент не найден, либо ключ найден на последнем сравнении. Пусть на старте алгоритм поиска затрагивает  $k_0$  операций, а при каждом сравнении  $k_1$  (2 константы). Тогда в лучшем случае будет затрачено  $k_0 + k_1$  операций. В случае, если ключ будет найден на 2ой позиции, будет затрачено  $k_0 + 2 * k_1$  операций, на последней позиции -  $k_0 + N * k_1$  операций, столько же, если ключ не будет найден вовсе.

## 1.3 Двоичный поиск в упорядоченном словаре

Если исходный массив уже отсортирован, то элемент в нем можно найти гораздо быстрее, если воспользоваться идеей *двоичного (бинарного) поиска*. Идея заключается в делении списка пополам, после чего в зависимости от значения медианного элемента в списке мы переходим либо к левой, либо к правой половине списка. Тем самым, длина части, в которой мы ищем элемент, сокращается в два раза на каждом шаге цикла, а, значит, общая сложность алгоритма двоичного поиска будет  $O(\log_2 n)$ .

На рис. 2.1 представлен пример решения задачи поиска элемента в массиве бинарным поиском.

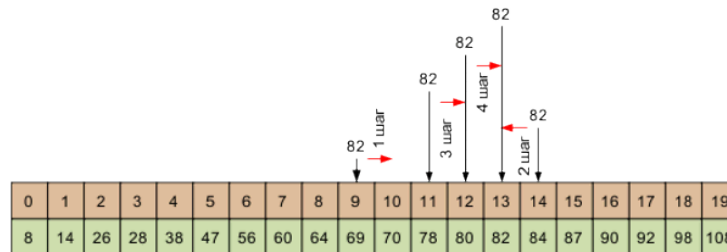


Рис 2.1: Двоичный (бинарный) поиск

## 1.4 Частичный анализ

*Частичный анализ* подразумевает, что словарь отсортирован по частоте встречаемости первого символа каждого ключа словаря. Ключами к нового словаря, сформированного на основе исходного, являются буквы алфавита, с которыми ассоциированы значения - словари, первая буква ключей которых совпадает с ключами к основного словаря. Такая организация напоминает толковый словарь. Изначально поиск осуществляется по первой букве ключа искомого значения, затем в найденном словаре поиск продолжается полным перебором.

## 1.5 Используемые данные

В этой лабораторной работе были использованы данные, представляющие таблицу пациентов, инфицированных COVID-19, в период от 01.01.2020 до 22.02.2020. Ключом является имя пациента, а значение - это структура со следующими полями:

- reporting\_date - дата регистрации пациента;
- gender - пол;
- age -возраст;
- symptom\_onset - дата начала проявления симптомов;
- hosp\_visit\_date - дата посещения больницы;
- id\_status - статус состояния здоровья.

## 1.6 Вывод

В данном разделе были рассмотрены теоретические сведения, необходимые для программной реализации поиска значения в словаре по ключу.

## 2 | Конструкторская часть

В данном разделе будут представлены схемы реализации алгоритма полного перебора, алгоритма двоичного поиска и алгоритма частичного анализа.

Схема реализации алгоритма полного перебора представлена на рис. 2.2.



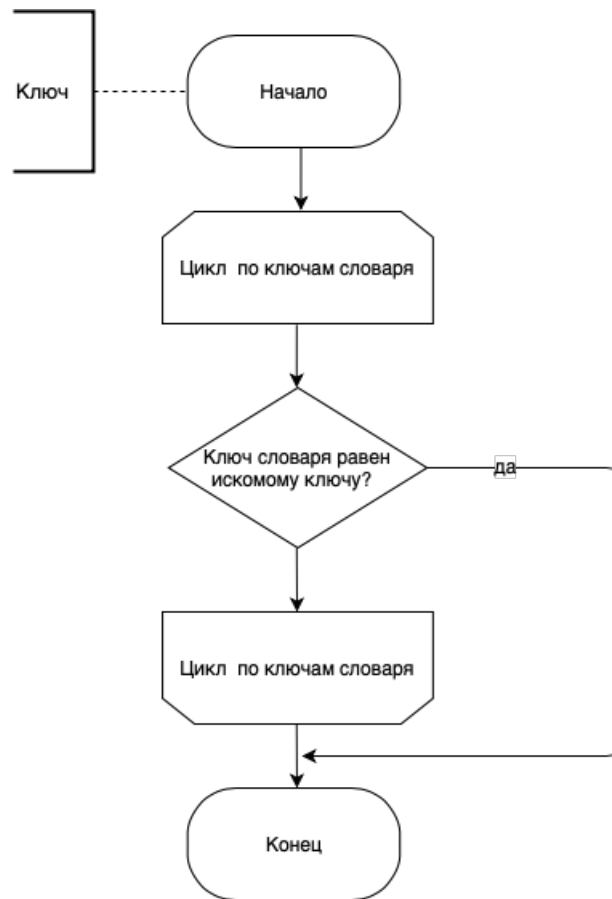


Рис 2.2: Схема реализации алгоритма полного перебора

Схема реализации алгоритма двоичного поиска представлена на рис. 2.3.

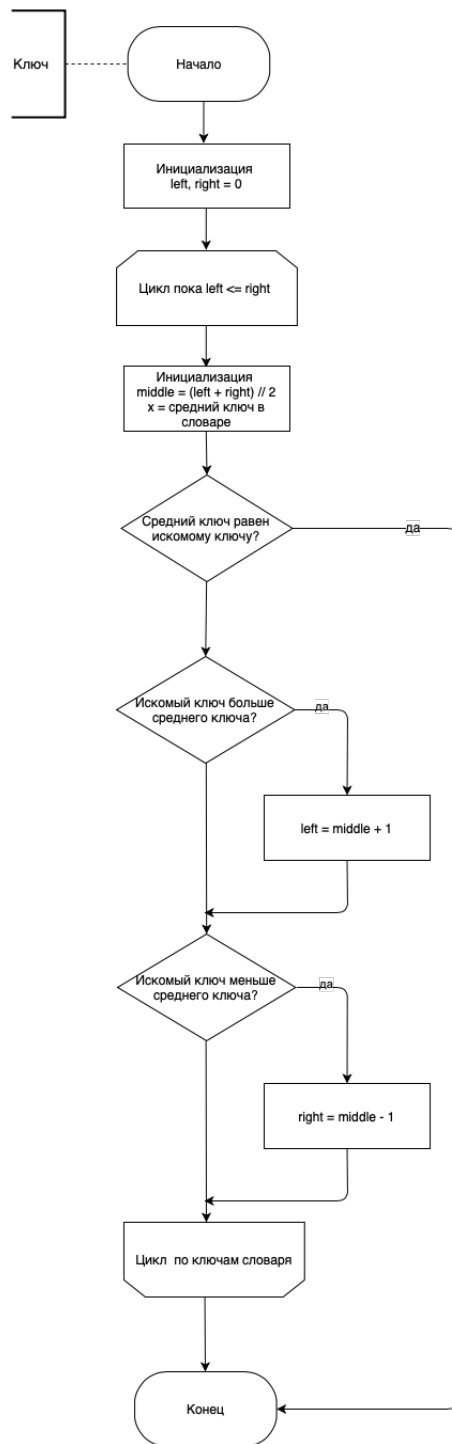


Рис 2.3: Схема реализации алгоритма двоичного поиска

Схема реализации алгоритма частичного анализа представлена на рис. 2.4.

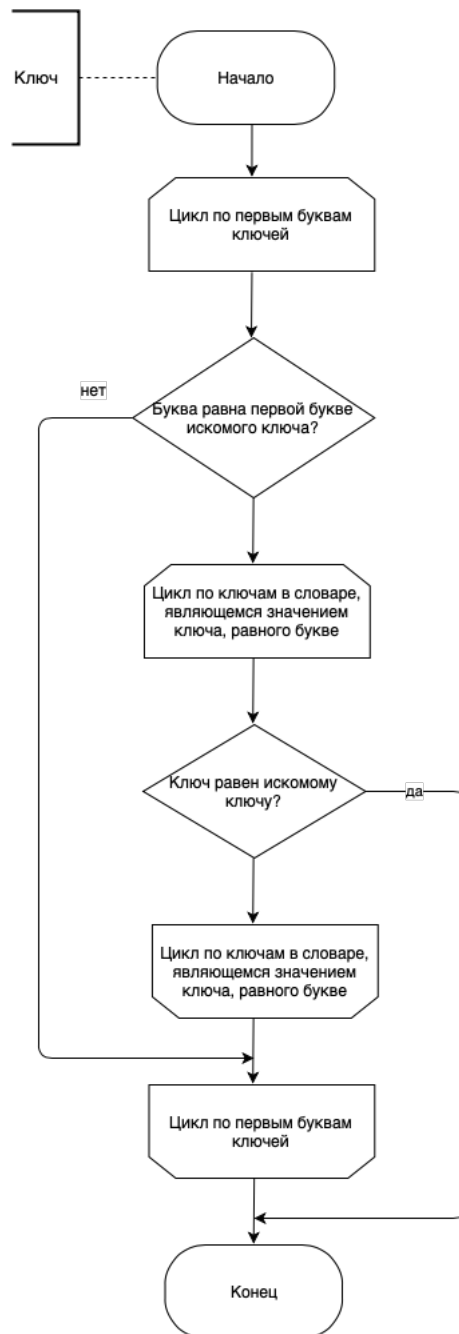


Рис 2.4: Схема реализации алгоритма частичного анализа

## 2.1 Вывод

В данном разделе были рассмотрены 3 схемы: схема реализации алгоритма полного перебора, схема алгоритма двоичного поиска и схема алгоритма частичного анализа

## 3 | Технологическая часть

### 3.1 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - Python [2], так как данный язык программирования предоставляет удобные библиотеки и инструменты для работы со структурами данных, в том числе со словарями. В качестве интегрированной среды разработки использовалась Visual studio [1].

### 3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - главный файл программы, в котором располагается точка входа в программу;
- PatientClass.py - организация хранения структуры данных;
- algorithms.py - реализация алгоритмов.

На листинге 3.1 представлена подпрограмма точки входа main().

Листинг 3.1: Код подпрограммы main()

```
1
2 def main():
3     data = Dictionary('person.csv')
4
5     try:
6         key = input("Enter the key: ")
7     except:
```

```

8         print("Input error!")
9         return
10
11     if key == '':
12         return
13
14     print("\nValue:\n{0}\n".format(data.BruteForceSearch(
15         key)))
16
17     print("Time characteristics: \n")
18     t1 = time()
19     for _ in range(COUNT):
20         data.BruteForceSearch(key)
21     t2 = time()
22     print("Brute force search: {0}".format(t2 - t1))
23
24     t1 = time()
25     for _ in range(COUNT):
26         data.BinarySearch(key, list_keys)
27     t2 = time()
28     print("Time binary search: {0}".format(t2 - t1))
29
30     new_dict = data.NewDictCreation()
31
32     t1 = time()
33     for _ in range(COUNT):
34         data.Search(key, new_dict)
35     t2 = time()
36     print("Time search: {0}".format(t2 - t1))

```

На листинге 3.2 представлена реализация алгоритма полного перебора.

Листинг 3.2: Реализация алгоритма полного перебора

```

1 def BruteForceSearch(self, key):
2     for x in self.data:
3         if key == x:
4             return self.data[x]
5     return

```

На листинге 3.3 представлена реализация алгоритма двоичного поиска.

Листинг 3.3: Реализация алгоритма двоичного поиска

```
1 def BinarySearch(self, key, list_keys):
2     left, right = 0, len(list_keys) - 1
3
4     while left <= right:
5         middle = (right + left) // 2
6         x = list_keys[middle]
7         if x == key:
8             return self.data[x]
9         elif x < key:
10            left = middle + 1
11        else:
12            right = middle - 1
13    return
```

На листинге 3.4 представлена реализация алгоритма частичного анализа.

Листинг 3.4: Реализация алгоритма частичного анализа

```
1 def Search(self, key, new_dict):
2     for letter in new_dict:
3         if key[0] == letter:
4             for x in new_dict[letter]:
5                 if x == key:
6                     return new_dict[letter][x]
7     return
8 return
```

На листинге 3.5 представлена подпрограммы организации словаря для частичного анализа.

Листинг 3.5: Подпрограммы организации словаря для частичного анализа

```
1 def NewDictCreation(self):
2     count_dict = {i: 0 for i in "
3         ABCDEFGHIJKLMNOPQRSTUVWXYZ"}
4     for key in self.data:
```



```

5         count_dict[key[0]] += 1
6
7     count_dict = self.sorting_by_values(count_dict)
8
9     new_dict = {i: dict() for i in count_dict}
10
11     for key in self.data:
12         new_dict[key[0]].update({key: self.data[key]})
13
14     return new_dict

```

### 3.3 Тесты

В этой лабораторной проводилось тестирование методом черного ящика. Результаты тестирования приведены в таблице 3.1.

Входные данные (комментарий)	Результат
Faeside (начало словаря)	Результат верный
Kantrius (конец словаря)	Результат верный
Bonn (середина словаря)	Результат верный
Kihn (произвольная позиция в словаре)	Результат верный
A (ключ, которого нет в словаре)	Результат верный

### 3.4 Вывод

В технологической части были представлены модули программы, листинги кода, а также обусловлен выбор языка программирования, приведены использовавшиеся в ходе работы инструменты, а также представлены результаты тестирования.

## 4 | Исследовательская часть

В этом разделе будет проведен сравнительный анализ характеристик полученного программного продукта.

### 4.1 Характеристики ЭВМ

- MacBook Pro (Retina, 15-inch, Mid 2014).
- 2,5 GHz Intel Core i7.
- Число логических ядер: 8.

### 4.2 Временные характеристики

Так как процедура поиска значения по ключу в словаре является достаточно быстрой, был применен метод массового усреднения эксперимента. Для этого поиск проводился заданное количество раз ( $\text{iter} = 1000$ ), затем измеренное время делится на  $\text{iter}$ , и таким образом результатом замеров будет среднее время выполнения каждого из алгоритмов.

#### 4.2.1 Лучший случай

Лучшим случаем считается ситуация, когда искомый ключ располагается в начале словаря. На тестовой выборке данных `person.csv` первым элементом является пациент с именем `Faeside`, который будет ключом в словаре. На рис. 4.1 представлен результат работы программы.

```

MacBook-Pro-Anastasia:lab7 anastasia$ python3 main.py
Введите ключ, по котрому будет осуществляться поиск: Faeside

Value:
gender: female
age: 66.0
symptom_onset: 01/03/20
hosp_visit_date: 01/11/20
id_status: 1

Time characteristics:

Brute force search: 0.0027849674224853516
Binary search time: 0.02169322967529297
Frequency Analysis search: 0.008713006973266602

```

Рис 4.1: Результат работы программы в лучшем случае

Результат свидетельствует о том, что в лучшем случае наиболее выигрышным по времени является алгоритм поиска полным перебором, так как искомый ключ найдется за 1 итерацию цикла поиска ключа в словаре.

На рис. 4.2 приведен графики зависимостей времени работы алгоритмов от размерности матрицы смежности.

### 4.2.2 Худший случай

Худшим случаем считается ситуация, когда искомый ключ располагается в конце словаря. На тестовой выборке данных person.csv последним элементом является пациент с именем Kantrius, который будет ключом в словаре. На рис. 4.2 представлен результат работы программы.

```

MacBook-Pro-Anastasia:lab7 anastasia$ python3 main.py
Введите ключ, по котрому будет осуществляться поиск: Kantrius

Value:
gender: male
age: 0.0
symptom_onset: 0
hosp_visit_date: 0
id_status: 3

Time characteristics:

Brute force search: 0.27983999252319336
Binary search time: 0.020900964736938477
Frequency Analysis search: 0.02790999412536621

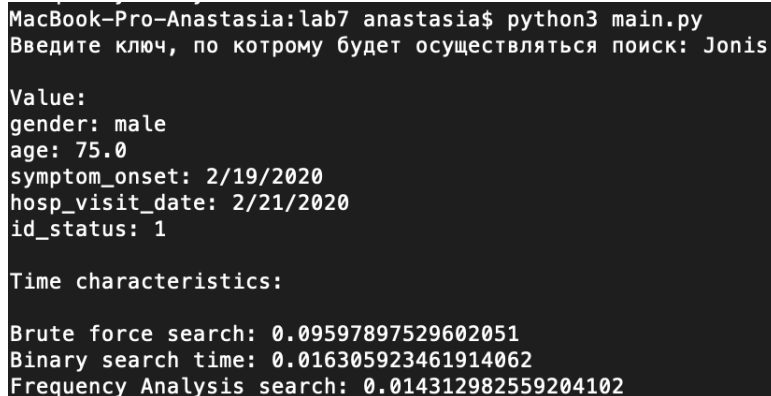
```

Рис 4.2: Результат работы программы в худшем случае

Результат свидетельствует о том, что в худшем случае наиболее выигрышным по времени является алгоритм двоичного поиска, так как искомый ключ найдется за  $N$  итераций цикла поиска ключа в словаре и при таком условии алгоритм полного перебора и частичного анализа работают медленнее за счет большого количества проходов цикла.

### 4.2.3 Произвольная позиция в словаре

На тестовой выборке данных person.csv произвольным элементом был взят пациент с именем Jonis, который будет ключом в словаре. На рис. 4.3 представлен результат работы программы.



```
MacBook-Pro-Anastasia:lab7 anastasia$ python3 main.py
Введите ключ, по которому будет осуществляться поиск: Jonis

Value:
gender: male
age: 75.0
symptom_onset: 2/19/2020
hosp_visit_date: 2/21/2020
id_status: 1

Time characteristics:
Brute force search: 0.09597897529602051
Binary search time: 0.016305923461914062
Frequency Analysis search: 0.014312982559204102
```

Рис 4.3: Результат работы в случае, когда ключ находится в произвольном месте словаря

Результат свидетельствует о том, что в случае произвольной позиции ключа в словаре алгоритм поиска частичным анализом работает быстрее всего.

### 4.2.4 Ключ не найден

Был взят ключ, значения которого нет в тестовой выборке данных person.csv: Nastya. На рис. 4.4 представлен результат работы программы.

```
MacBook-Pro-Anastasia:lab7 anastasia$ python3 main.py
Введите ключ, по котрому будет осуществляться поиск: Jonis

Value:
gender: male
age: 75.0
symptom_onset: 2/19/2020
hosp_visit_date: 2/21/2020
id_status: 1

Time characteristics:

Brute force search: 0.09597897529602051
Binary search time: 0.016305923461914062
Frequency Analysis search: 0.014312982559204102
```

Рис 4.4: Результат работы в случае, когда искомого ключа нет в словаре

Результат свидетельствует о том, что в случае, когда искомого ключа нет в словаре, алгоритм поиска частичным анализом работает быстрее всего.

## 4.3 Вывод

В результате сравнения алгоритма полного перебора, алгоритма двоичного поиска и алгоритма поиска частичным анализом было установлено, что в большинстве случаев алгоритм поиска полным перебором работает медленнее остальных, за исключением случая, когда искомый ключ находится в начале словаря. В худшем случае, когда ключ находится в конце словаря, по времени выигрышным оказывается алгоритм двоичного поиска. В остальных случаях выигрывает алгоритм поиска частичным анализом.

# Заключение

В ходе лабораторной работы был изучен поиск значения по ключу в словаре с помощью алгоритмов поиска полным перебором, двоичного поиска в упорядоченном словаре и частичного анализа.

В рамках выполнения работы решены следующие задачи:

- изучен алгоритм поиска в словаре полным перебором;
- изучен алгоритм двоичного поиска в упорядоченном словаре;
- изучен алгоритм частичного анализа для поиска в словаре;
- программно реализованы описанные выше алгоритмы;
- проведен сравнительный анализ скорости работы реализованных алгоритмов.

# Литература

- [1] Visual Studio [Электронный ресурс], режим доступа: <https://visualstudio.microsoft.com/ru/> (дата обращения: 01.10.2020)
- [2] Python [Электронный ресурс], режим доступа: <https://www.python.org> (дата обращения: 24.12.2020)
- [3] Муравьиные алгоритмы [Электронный ресурс], режим доступа: [http://www.machinelearning.ru/wiki/index.php?title=Муравьиные\\_алгоритмы](http://www.machinelearning.ru/wiki/index.php?title=Муравьиные_алгоритмы) (дата обращения: 20.12.2020)
- [4] NIST's Dictionary of Algorithms and Data Structures: Associative Array [Электронный ресурс], режим доступа: <https://xlinux.nist.gov/dads/HTML/assocarray.html> (дата обращения: 21.12.2020)
- [5] The Perl Programming Language [Электронный ресурс], режим доступа: <https://www.perl.org> (дата обращения: 24.12.2020)
- [6] Руководство по PHP [Электронный ресурс], режим доступа: <https://www.php.net/manual/ru/index.php> (дата обращения: 24.12.2020)
- [7] Язык программирования Ruby [Электронный ресурс], режим доступа: <https://www.ruby-lang.org/ru/> (дата обращения: 24.12.2020)
- [8] JavaScript [Электронный ресурс], режим доступа: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата обращения: 24.12.2020)