Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

Поиск

Вариант № 7

Преподаватель \_\_\_\_\_\_\_\_\_\_ И. В. Матковский

подпись, дата

Студент КИ20-08Б 032049025 \_\_\_\_\_\_\_\_\_\_ А. Н. Головань

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2021

**1 Цель работы**

Изучить алгоритмы поиска.

**2 Порядок выполнения работы**

Разработать для решения поставленной задачи алгоритм; решить поставленную задачу с помощью последовательного поиска и поиска, предусмотренного вариантом. Оценить алгоритмы.

**3 Задание**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *№* | *Задание* | *Тип поиска* | *Содержимое массива* | *Массив отсортирован* |
| 7 | Найти заданный элемент | Интерполирующий | Символьные строки | Да |

**4 Код программы**

Program.cpp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120 | from random import randint  import os  from time import time  os.system("cls")  count = 10  word = "hello my name is andrey"  word\_massive=word.split(" ")  symb = "is"  def interSearch(word\_massive, check):  dict\_words = []  val\_massive = []  for word in word\_massive:  dict\_words.append({})  dict\_words[len(dict\_words)-1]["word"] = word  dict\_words[len(dict\_words)-1]["val"] = 0  count = 0  for symb in word:  count += 1  dict\_words[len(dict\_words)-1]["val"]+= ord(symb) + ord(symb) \* count  val\_massive.append(dict\_words[len(dict\_words)-1]["val"])  count = 0  val = 0  for symb in check:  count += 1  val += ord(symb) + ord(symb) \* count  val\_massive = sorted(val\_massive)  beg = 0  end = (len(dict\_words) - 1)  while beg <= end and val >= val\_massive[beg] and val <= val\_massive[end]:  middle = beg + int((float((end - beg) / ( val\_massive[end] - val\_massive[beg])) \* ( val - val\_massive[beg])))  if val\_massive[middle] == val:  for node in range(len(dict\_words)):  if dict\_words[node]["val"] == val:  return node  if val\_massive[middle] < val:  beg = middle + 1;  else:  end = middle - 1;  def lineSearch(word\_massive,symb):  for i in range(len(word\_massive)):  if word\_massive[i] == symb:  return i  def binSearch(word\_massive,check):  dict\_words = []  val\_massive = []  for word in word\_massive:  dict\_words.append({})  dict\_words[len(dict\_words)-1]["word"] = word  dict\_words[len(dict\_words)-1]["val"] = 0  count = 0  for symb in word:  count += 1  dict\_words[len(dict\_words)-1]["val"]+= ord(symb) + ord(symb) \* count  val\_massive.append(dict\_words[len(dict\_words)-1]["val"])  count = 0  val = 0  for symb in check:  count += 1  val += ord(symb) + ord(symb) \* count  val\_massive = sorted(val\_massive)  mid = len(val\_massive) // 2  low = 0  high = len(val\_massive) - 1  while val\_massive[mid] != val and low <= high:  if val > val\_massive[mid]:  low = mid + 1  else:  high = mid - 1  mid = (low + high) // 2  if low <= high:  for node in range(len(dict\_words)):  if dict\_words[node]["val"] == val:  return node  print(word\_massive)  print(symb)  start\_time = time()  index = interSearch(word\_massive,symb)  end\_time = time()  print("interSearch -- > {} for {} sec".format(index,end\_time-start\_time))  start\_time = time()  index = lineSearch(word\_massive,symb)  end\_time = time()  print("lineSearch -- > {} for {} sec".format(index,end\_time-start\_time))  start\_time = time()  index = binSearch(word\_massive,symb)  end\_time = time()  print("binSearch -- > {} for {} sec".format(index,end\_time-start\_time)) |

**5 Теоретические оценки сложности алгоритмов**

В программе используются два алгоритма поиска. Первый из них – последовательный, его сложность составляет . В качестве второго алгоритма используется интерполирующий поиск, в среднем имеет сложность , в худшем .

**6 Экспериментальные оценки сложности алгоритмов**

Таблица 1 - зависимость времени от объема входных данных

|  |  |  |  |
| --- | --- | --- | --- |
| Количество слов | 150 | 500 | 1000 |
| Последовательный  Время с | 0 с | 0.005 с | 0.009 с |
| Интерполирующий  Время с | 0 с | 0 с | 0 с |