ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Работа с таблицей символов

Выполнил: Петров Е. Д.

Вариант - 2

**Цель работы:** изучить основные методы организации таблиц идентификаторов, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц символов (идентификаторов). Для выполнения лабораторной работы требуется написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить многократный поиск идентификатора в этой таблице. Список идентификаторов считать заданным в виде текстового файла. Длина идентификаторов ограничена 32 символами.

В этой лабораторной работе нужно создать программу, которая будет работать с таблицами идентификаторов с помощью хеширования. Основная задача — сделать хеш-таблицу, в которой каждый идентификатор будет превращаться в числовой ключ с помощью простой формулы: складываются коды первых двух букв. Если два идентификатора получат одинаковый ключ (коллизия), их нужно будет хранить в списке и искать их вручную.

Программа должна уметь загружать данные из текстового файла, преобразовывать их в хеш-таблицу и находить нужные идентификаторы. Важно также анализировать, насколько эффективен поиск, подсчитывая количество сравнений и среднее число коллизий.

Кроме этого, необходимо создать удобный графический интерфейс, чтобы пользователь мог загружать файлы, просматривать данные, искать идентификаторы и видеть результаты поиска. Интерфейс должен содержать кнопки и другие элементы управления для удобной работы с программой.

ОПИСАНИЕ АЛГОРИТМА РАБОТЫ ПРОГРАММЫ

Работа программы начинается с загрузки текстового файла, содержащего список идентификаторов. После выбора файла его содержимое отображается в графическом интерфейсе, а идентификаторы передаются в хеш-таблицу, где каждому из них присваивается хеш-ключ, вычисленный по заданной формуле — ***сумма кодов первых двух символов***.

Если несколько идентификаторов получают одинаковый «хеш», они группируются в списке. Затем программа предоставляет возможность поиска по таблице: пользователь вводит идентификатор, и программа проверяет, есть ли он в соответствующей группе. При этом подсчитывается количество сравнений, необходимых для нахождения результата.

Графический интерфейс содержит основные элементы управления: загрузку файла, поле поиска, кнопку выполнения поиска, а также отображение результатов — найденного идентификатора, его «хеша» и статистики. Дополнительно реализована кнопка «Найти все», которая выводит список всех идентификаторов и их хеш-значений.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В лабораторной работе приложение реализовано на языке Python, использующее библиотеку *«tkinter»* для создания графического интерфейса. Программа выполняет обработку идентификаторов методом хеширования, а также предоставляет пользователю удобные инструменты для работы с загруженными данными.



Рис. 1 – Импорт библиотеки.

Приложение основано на двух классах:

1. *«HashTable»*– предназначен для хранения идентификаторов, их хеширования и обработки коллизий.
2. *«IdentifierApp»*– обеспечивает взаимодействие пользователя с программой, позволяя загружать файлы, выполнять поиск и анализировать результаты.

**Реализация хеш-таблицы**

Функционирование хеш-таблицы основано на использовании ***простой хеш-функции***, вычисляющей сумму кодов *первых двух символов идентификатора*. В случае возникновения коллизий (совпадения хеш-значений у разных идентификаторов) применяется метод ***списка с простым перебором***, который позволяет корректно обрабатывать дубликаты значений.

Основные методы класса *«HashTable»*:

* **insert(identifier)** – добавляет идентификатор в хеш-таблицу, при необходимости разрешая коллизии.

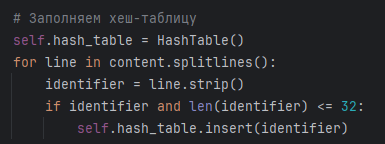


Рис.2 – Пример заполнения таблицы*.*

* **search(identifier)** – выполняет поиск идентификатора, подсчитывая количество сравнений.

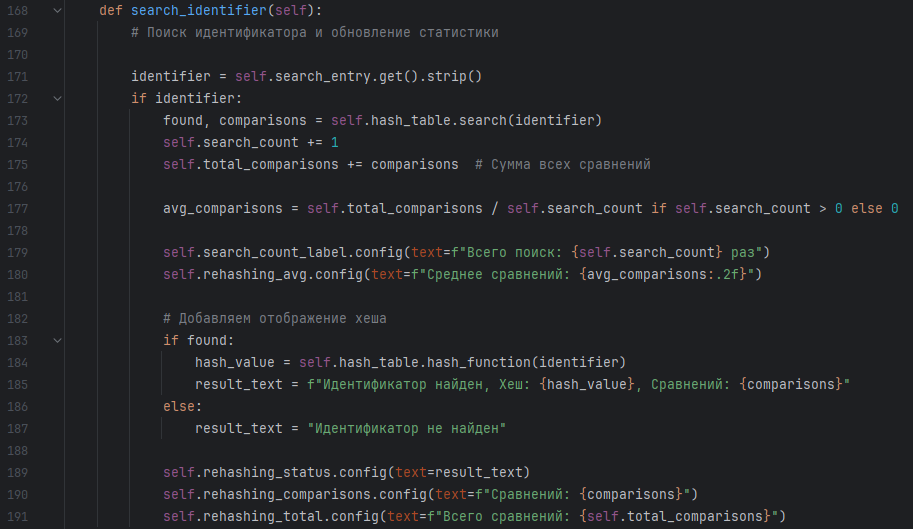


Рис.3 – Пример реализации функции поиска идентификатора.

* **calculate\_collisions()** – вычисляет среднее количество коллизий, возникающих при заполнении таблицы.

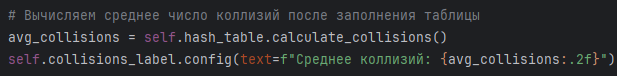


Рис.4 – Пример реализации вычисления среднего числа коллизий.

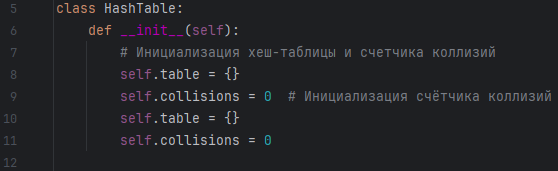


Рис.5 – Пример класса *«HashTable».*

**Разработка графического интерфейса**

Графическая оболочка программы реализована с использованием библиотеки *«tkinter»* и включает два основных блока:

1. **Левая панель** предназначена для отображения загруженного файла. Здесь выводится список всех идентификаторов, загруженных в программу.
2. **Правая панель** содержит элементы управления, включая кнопки загрузки файла, поиска идентификатора, отображения всех записей и сброса данных.

Ключевые функции класса *«IdentifierApp»:*

* **load\_file()** – открывает и загружает текстовый файл с идентификаторами, передавая их в хеш-таблицу.
* **search\_identifier()** – выполняет поиск введенного пользователем идентификатора, отображая количество сравнений.
* **find\_all()** – выводит полный список идентификаторов вместе с их хеш-значениями.
* **reset\_results()** – очищает интерфейс программы, сбрасывает таблицу и статистику поиска.

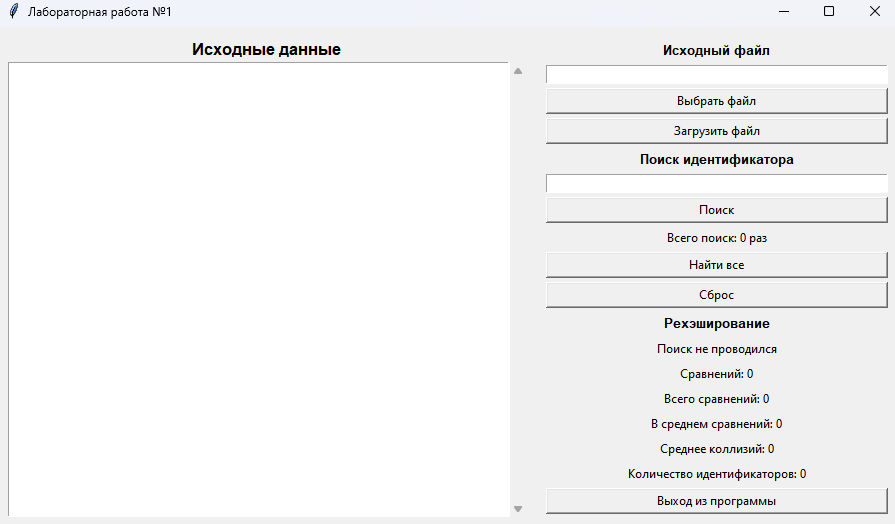


Рис. 6 – Пример графического интерфейса.

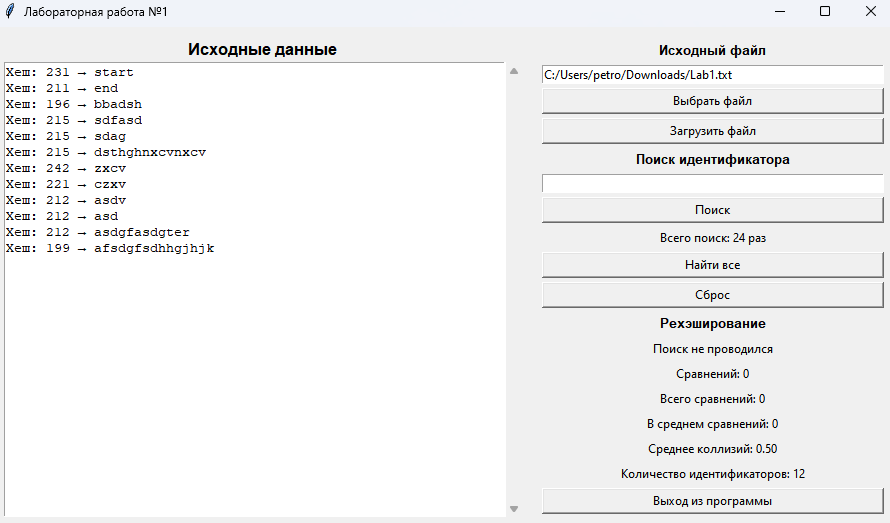


Рис. 7 – Пример работы программы.

**Алгоритм работы программы**

Программа работает по следующему сценарию:

1. Пользователь **выбирает текстовый файл** и загружает его в программу.
2. Все идентификаторы **добавляются в хеш-таблицу**, где для каждого вычисляется «хеш». В случае совпадений значения объединяются в список.
3. После загрузки пользователь может:

* **Выполнить поиск конкретного идентификатора**, введя его в поле ввода.
* **Просмотреть полный список идентификаторов с их хешами**, нажав кнопку «Найти все».
* **Сбросить данные**, очистив хеш-таблицу и интерфейс.

1. В процессе работы программа **собирает статистику**, включая среднее количество коллизий и сравнений.

ПРИЛОЖЕНИЕ

import tkinter as tk

from tkinter import filedialog, scrolledtext

class HashTable:

def \_\_init\_\_(self):

# Инициализация хеш-таблицы и счетчика коллизий

self.table = {}

self.collisions = 0 # Инициализация счётчика коллизий

self.table = {}

self.collisions = 0

def hash\_function(self, identifier):

# Хеш-функция: сумма кодов первых двух букв идентификатора

first = ord(identifier[0]) if len(identifier) > 0 else 0

second = ord(identifier[1]) if len(identifier) > 1 else 0

return first + second

def insert(self, identifier):

# Добавление идентификатора в хеш-таблицу

key = self.hash\_function(identifier)

if key not in self.table:

self.table[key] = []

else:

self.collisions += 1 # Подсчёт коллизий

self.table[key].append(identifier)

def search(self, identifier):

# Поиск идентификатора в таблице и подсчет сравнений

key = self.hash\_function(identifier)

comparisons = 0

if key in self.table:

for item in self.table[key]:

comparisons += 1

if item == identifier:

return True, comparisons

return False, comparisons

def calculate\_collisions(self):

# Вычисление среднего количества коллизий

if not self.table:

return 0

total\_collisions = sum(len(v) - 1 for v in self.table.values() if len(v) > 1)

return total\_collisions / len(self.table)

class IdentifierApp:

def \_\_init\_\_(self, root):

# Создание графического интерфейса приложения

self.root = root

self.root.title("Лабораторная работа №1")

self.root.geometry("900x500")

self.root.resizable(True, True) # Позволяет изменять размер окна

self.search\_count = 0 # Инициализация счетчика поисков

self.total\_comparisons = 0 # Инициализация суммы сравнений

self.hash\_table = HashTable()

# Левая панель - отображение файла

self.left\_frame = tk.Frame(root, padx=10, pady=10)

self.left\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

self.label\_file\_content = tk.Label(self.left\_frame, text="Исходные данные", font=("Arial", 12, "bold"))

self.label\_file\_content.pack()

self.text\_area = scrolledtext.ScrolledText(self.left\_frame, wrap=tk.WORD, width=50, height=20)

self.text\_area.pack(fill=tk.BOTH, expand=True)

# Правая панель - управление

self.right\_frame = tk.Frame(root, padx=10, pady=10)

self.right\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

# Блок "Исходный файл"

self.label\_file = tk.Label(self.right\_frame, text="Исходный файл", font=("Arial", 10, "bold"))

self.label\_file.pack()

self.file\_entry = tk.Entry(self.right\_frame, width=40)

self.file\_entry.pack()

self.button\_choose\_file = tk.Button(self.right\_frame, text="Выбрать файл", command=self.choose\_file)

self.button\_choose\_file.pack()

self.button\_load\_file = tk.Button(self.right\_frame, text="Загрузить файл", command=self.load\_file)

self.button\_load\_file.pack()

# Блок "Поиск идентификатора"

self.label\_search = tk.Label(self.right\_frame, text="Поиск идентификатора", font=("Arial", 10, "bold"))

self.label\_search.pack()

self.search\_entry = tk.Entry(self.right\_frame, width=40)

self.search\_entry.pack()

self.button\_search = tk.Button(self.right\_frame, text="Поиск", command=self.search\_identifier)

self.button\_search.pack()

self.search\_count\_label = tk.Label(self.right\_frame, text="Всего поиск: 0 раз")

self.search\_count\_label.pack()

self.button\_find\_all = tk.Button(self.right\_frame, text="Найти все", command=self.find\_all)

self.button\_find\_all.pack()

self.button\_reset = tk.Button(self.right\_frame, text="Сброс", command=self.reset\_results)

self.button\_reset.pack()

# Метрики поиска

self.label\_rehashing = tk.Label(self.right\_frame, text="Рехэширование", font=("Arial", 10, "bold"))

self.label\_rehashing.pack()

self.rehashing\_status = tk.Label(self.right\_frame, text="Поиск не проводился")

self.rehashing\_status.pack()

self.rehashing\_comparisons = tk.Label(self.right\_frame, text="Сравнений: 0")

self.rehashing\_comparisons.pack()

self.rehashing\_total = tk.Label(self.right\_frame, text="Всего сравнений: 0")

self.rehashing\_total.pack()

self.rehashing\_avg = tk.Label(self.right\_frame, text="В среднем сравнений: 0")

self.rehashing\_avg.pack()

self.collisions\_label = tk.Label(self.right\_frame, text="Среднее коллизий: 0")

self.collisions\_label.pack()

self.identifier\_count\_label = tk.Label(self.right\_frame, text="Количество идентификаторов: 0")

self.identifier\_count\_label.pack()

self.button\_exit = tk.Button(self.right\_frame, text="Выход из программы", command=self.root.quit)

self.button\_exit.pack()

for widget in self.right\_frame.winfo\_children():

widget.pack\_configure(fill=tk.X, expand=True)

def choose\_file(self):

# Выбор файла с идентификаторами

file\_path = filedialog.askopenfilename(filetypes=[("Text files", "\*.txt")])

if file\_path:

self.file\_entry.delete(0, tk.END)

self.file\_entry.insert(0, file\_path)

def load\_file(self):

# Загрузка файла и обработка идентификаторов

file\_path = self.file\_entry.get()

if file\_path:

try:

with open(file\_path, "r", encoding="utf-8") as file:

content = file.read()

self.text\_area.delete(1.0, tk.END)

self.text\_area.insert(tk.END, content)

# Заполняем хеш-таблицу

self.hash\_table = HashTable()

for line in content.splitlines():

identifier = line.strip()

if identifier and len(identifier) <= 32:

self.hash\_table.insert(identifier)

identifier\_count = sum(len(v) for v in self.hash\_table.table.values()) # Считаем все идентификаторы

self.identifier\_count\_label.config(text=f"Количество идентификаторов: {identifier\_count}")

# Вычисляем среднее число коллизий после заполнения таблицы

avg\_collisions = self.hash\_table.calculate\_collisions()

self.collisions\_label.config(text=f"Среднее коллизий: {avg\_collisions:.2f}")

except Exception as e:

print(f"Ошибка загрузки файла: {e}")

def search\_identifier(self):

# Поиск идентификатора и обновление статистики

identifier = self.search\_entry.get().strip()

if identifier:

found, comparisons = self.hash\_table.search(identifier)

self.search\_count += 1

self.total\_comparisons += comparisons # Сумма всех сравнений

avg\_comparisons = self.total\_comparisons / self.search\_count if self.search\_count > 0 else 0

self.search\_count\_label.config(text=f"Всего поиск: {self.search\_count} раз")

self.rehashing\_avg.config(text=f"Среднее сравнений: {avg\_comparisons:.2f}")

# Добавляем отображение хеша

if found:

hash\_value = self.hash\_table.hash\_function(identifier)

result\_text = f"Идентификатор найден, Хеш: {hash\_value}, Сравнений: {comparisons}"

else:

result\_text = "Идентификатор не найден"

self.rehashing\_status.config(text=result\_text)

self.rehashing\_comparisons.config(text=f"Сравнений: {comparisons}")

self.rehashing\_total.config(text=f"Всего сравнений: {self.total\_comparisons}")

def show\_hash\_table(self):

self.text\_area.delete(1.0, tk.END)

for key, values in self.hash\_table.table.items():

self.text\_area.insert(tk.END, f"Хеш: {key} -> {', '.join(values)}\n")

def find\_all(self):

self.text\_area.delete(1.0, tk.END) # Очищаем поле вывода

identifier\_count = 0 # Подсчёт всех идентификаторов

for key, values in self.hash\_table.table.items():

for value in values:

self.text\_area.insert(tk.END, f"Хеш: {key} → {value}\n") # Выводим идентификаторы и их хеши

identifier\_count += 1 # Учитываем количество идентификаторов

self.identifier\_count\_label.config(

text=f"Количество идентификаторов: {identifier\_count}") # Обновляем счётчик

self.search\_count += 1 # Засчитываем "глобальный поиск" как один раз

self.search\_count\_label.config(text=f"Всего поиск: {self.search\_count} раз")

self.search\_count += 1 # Засчитываем "глобальный поиск" как один раз

self.search\_count\_label.config(text=f"Всего поиск: {self.search\_count} раз")

def reset\_results(self):

self.hash\_table = HashTable()

self.text\_area.delete(1.0, tk.END)

self.search\_entry.delete(0, tk.END)

self.file\_entry.delete(0, tk.END)

self.search\_count = 0

self.total\_comparisons = 0

self.search\_count\_label.config(text="Всего поиск: 0 раз")

self.rehashing\_avg.config(text="Среднее сравнений: 0")

self.rehashing\_total.config(text="Всего сравнений: 0")

self.collisions\_label.config(text="Среднее коллизий: 0")

self.identifier\_count\_label.config(text="Количество идентификаторов: 0")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = IdentifierApp(root)

root.mainloop()