Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Аппаратное обеспечение интеллектуальных систем»

Тема: «Применение хэширования для ассоциативного поиска по ключам»

**Выполнил:**

Студент 2 курса

Группы ИИ-21

Литвинюк Т. В.

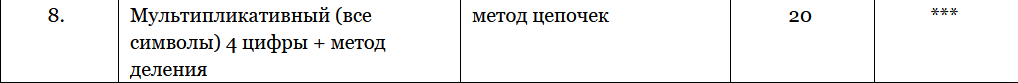
**Проверил:**

Скарубо А. О.

Брест 2023

**Цель:** изучить основные принципы хэширования, построить программную модель, обеспечивающую формирование хэш-таблиц и ассоциативный поиск по ключам.

**Ход работы**



Код программы:

from os import system

num\_of\_colls, num\_of\_records, num\_of\_appeals = 0, 0, 0

class HashItem:

def \_\_init\_\_(self, key="", val="", name="") -> None:

self.key, self.val, self.name = key, val, name

self.link = None

def TO\_SET(self, key, val, name):

self.key, self.val, self.name = key, val, name

def TO\_CLEAR(self, key):

if self.key == key:

self.key, self.val, self.name = "", "", ""

print("Удалено:", key)

num\_of\_records -= 1

elif self.link == None:

print("Такого ключа не существует")

else:

self.link.TO\_CLEAR(key)

def TO\_SHOW(self, key):

if self.key == key:

print(f"{self.val=}\t{self.name=}")

elif self.link == None:

print("Такого ключа не существует")

else:

self.link.TO\_SHOW(key)

def make\_link(self, link):

if self.link is None:

self.link = link

global num\_of\_colls

num\_of\_colls += 1

else:

self.link.make\_link(link)

def is\_empty(self):

return self.key == self.val == self.name == ""

class Hash\_tabl:

def \_\_init\_\_(self, length: int) -> None:

self.tabl = [HashItem() for \_ in range(length)]

def add(self, key, val, name):

index = get\_index(key)

if self.tabl[index].is\_empty():

self.tabl[index].TO\_SET(key, val, name)

else:

self.tabl[index].make\_link(HashItem(key, val, name))

global num\_of\_appeals

num\_of\_appeals += 1

def edit(self, key, val, name):

index = get\_index(key)

if self.tabl[index].key == key:

self.tabl[index].TO\_SET(key, val, name)

else:

self.tabl[index].link.TO\_SET(key, val, name)

def delete(self, key):

index = get\_index(key)

self.tabl[index].TO\_CLEAR(key)

global num\_of\_appeals

num\_of\_appeals += 1

def show(self, key):

index = get\_index(key)

self.tabl[index].TO\_SHOW(key)

global num\_of\_appeals

num\_of\_appeals += 1

def get\_index(key):

arr = [ord(x) for x in key]

mult = 1

for el in arr:

mult \*= el

mult %= 10000 # отделяет последние 4 цифры

mult %= 23

print(f"Индекс ключа \"{key}\" : {mult}")

return mult

hash\_tabl = Hash\_tabl(20)

while True:

menu = input("1. Добавление записи\n2. Редактирование записи\n3. Удаление записи\n4. Вывод таблицы\n5. Выход из программы, провести расчеты\n>> ")

system('cls')

match menu:

case '1':

key = input("Введите ключ: ")

val = input("Введите значение: ")

name = input("Введите имя: ")

hash\_tabl.add(key, val, name)

num\_of\_records += 1

case '2':

key = input("Введите ключ: ")

val = input("Введите значение: ")

name = input("Введите имя: ")

hash\_tabl.edit(key, val, name)

case '3':

key = input("Введите ключ: ")

hash\_tabl.delete(key)

case '4':

key = input("Введите ключ: ")

hash\_tabl.show(key)

case '5':

break

system('cls')

print(

f"""Всего коллизий: {num\_of\_colls}.

Всего записей в таблице: {num\_of\_records}.

Всего обращений к таблице: {num\_of\_appeals}.

Вероятность возникновения коллизий: {num\_of\_colls/num\_of\_records}.

Среднее количество обращений к таблице во время поиска записи по ключу: {num\_of\_appeals/num\_of\_records}."""

)