Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Звіт про виконання лабораторної роботи №4

Хешування методом ланцюгів. Хешування відкритою адресацією.

Виконав:
Студент групи ФЕП-22
Серафим Д.В.

Львів-2022

Частина 1. Хешування методом ланцюгів.

1. Створити нову бібліотеку Hash (файли Hash.h, Hash.cpp). У бібліотеці Hash, згідно описаних в теоретичній частині алгоритмів, створити функції ChainedHashInsert(…),ChainedHashSearch(…),ChainedHashDelete(…) та ChainedHashShow(…) для вставки, пошуку, видалення даних та відображення хеш-таблиці у вигляді стрічок ланцюгів даних.

2. У цій же бібліотеці створити хеш-функції для реалізації хешування методами ділення та множення (див. Теоретичні відомості).

3. Створити новий проект Lab_4_1 та підключити до нього бібліотеку Hash. У функції main() проекту створити масив лінійних списків розміру m. (використати двозв'язний список з бібліотеки List з курсу „Дискретна математика”, або стандартну бібліотеку List з STL). Розмір масиву повинен задаватися динамічно.

4. У функції main() проекту реалізувати меню для операцій вставки, пошуку, видалення даних у хеш-таблиці та відображення отриманих результатів.

5. Відкомпілювати проект та продемонструвати його роботу для наборуданих, отриманого від викладача та різних хеш-функцій.

```
from random import randint

decor = "=" + "-" * 20 + "="

name = ["Давид", "Далемил", "Далемир", "Далібор", "Дан", "Данило", "Данко",
"Дантур", "Дар", "Дарій", "Дарибог",
        "Даромир", "Денис", "Демид", "Дем'ян", "Держикрай", "Держислав",
"Дибач", "Дивозір", "Дій", "Дмитро",
        "Добрибій", "Добривод", "Добрик", "Добрило", "Добриня", "Добрисвіт",
"Добровіст", "Доброгост", "Добродум",
        "Добролик", "Добролюб", "Добромир", "Добромисл", "Доброслав", "Доморад",
"Домослав", "Дорогобуг", "Дорогомир",
        "Дорогомисл", "Дорогосил", "Дорогочин", "Дорофій", "Домінік", "Драган",
"Драгомир"]


def show_table(hashtable):
    for i in range(len(hashtable)):
        print(i, end=" ")
        for j in hashtable[i]:
            print('-->', end=" ")
            print(j, end='')
        print()
    print(decor)


HashTable = [[] for _ in range(10)]


def hashing(key_value):
    return key_value % len(HashTable)


def chained_hash_insert(hashtable, key_value, value):
    hash_key = hashing(key_value)
    hashtable[hash_key].append(value)


def chained_hash_delete(hashtable, key_value):
    hash_key = hashing(key_value)
    hashtable[hash_key].pop()
```

```
def chained_hash_search(hashtable, value):
    for i in range(len(hashtable)):
        for j in hashtable[i]:
            if value == j:
                for k in hashtable[i]:
                    print('-->', end=" ")
                    print(k, end='')


def start():
    print(decor)
    how_start = input(
        "Введіть яким чином хочете заповнити вершини:\n0 - Вручну\n1 -
Автоматично з відомими вхідними\n2 - Автоматично\n=")
    print(decor)
    if int(how_start) == 0:
        num = 0
        input_data_num = ""
        input_data_name = ""
        print("Якщо хочеш зупинити процес введи в консоль STOP")

        while str(input_data_num) != "STOP" and str(input_data_name) != "STOP":
            num += 1
            input_data_num = input(f"Введіть номер для особи №{num}: ")
            input_data_name = input(f"Введіть Імя особи №{num}: ")

            if str(input_data_num) != "STOP" and str(input_data_name) != "STOP":
                chained_hash_insert(HashTable, int(input_data_num),
input_data_name)

        show_table(HashTable)

    elif int(how_start) == 1:

        chained_hash_insert(HashTable, 7, 'Добролюб')
        chained_hash_insert(HashTable, 25, 'Держислав')
        chained_hash_insert(HashTable, 20, 'Домослав')
        chained_hash_insert(HashTable, 9, 'Дар')
        chained_hash_insert(HashTable, 20, 'Далемил')

        chained_hash_search(HashTable, 'Punjab')

        show_table(HashTable)

    elif int(how_start) == 2:
        num = int(input("Введіть кількість елементів скільки потрібно
згенерувати:"))
        for i in range(num):
            chained_hash_insert(HashTable, randint(0, 100), name[randint(0,
len(name) - 1)])

        show_table(HashTable)

    else:

        print("Введено дані котрі не подялгають умові!")


start()
print(HashTable)

# delete(HashTable, 9)
# show_table(HashTable)
```

```
=--------------------=
Введіть яким чином хочете заповнити вершини:
0 - Вручну
1 - Автоматично з відомими вхідними
2 - Автоматично
=1
=--------------------=
0 --> Домослав--> Далемил
1
2
3
4
5 --> Держислав
6
7 --> Добролюб
8
9 --> Дар
=--------------------=
[['Домослав', 'Далемил'], [], [], [], [], ['Держислав'], [], ['Добролюб'], [], ['Дар']]

Process finished with exit code 0
```

```
=--------------------=
Введіть яким чином хочете заповнити вершини:
0 - Вручну
1 - Автоматично з відомими вхідними
2 - Автоматично
=0
=--------------------=
Якщо хочеш зупинити процес введи в консоль STOP
Введіть номер для особи №1: 213
Введіть Імя особи №1: Dima
Введіть номер для особи №2: 32
Введіть Імя особи №2: Tanya
Введіть номер для особи №3: 345
Введіть Імя особи №3: Ivan
Введіть номер для особи №4: 436
Введіть Імя особи №4: Petro
Введіть номер для особи №5: 325
Введіть Імя особи №5: Solo
Введіть номер для особи №6: 2365
Введіть Імя особи №6: Petro
Введіть номер для особи №7: STOP
Введіть Імя особи №7:
0
1
2 --> Tanya
3 --> Dima
4
5 --> Ivan--> Solo--> Petro
6 --> Petro
7
8
9
=--------------------=
[[], [], ['Tanya'], ['Dima'], [], ['Ivan', 'Solo', 'Petro'], ['Petro'], [], [], []]

Process finished with exit code 0
```

```
=--------------------=
Введіть яким чином хочете заповнити вершини:
0 - Вручну
1 - Автоматично з відомими вхідними
2 - Автоматично
=2
=--------------------=
Введіть кількість елементів скільки потрібно згенерувати:50
0 --> Далемир--> Добрибій--> Добрисвіт--> Добролик--> Держикрай--> Дан--> Добриня
1 --> Дорогобуг--> Далібор--> Дорогобуг--> Добриня--> Данко--> Доброгост--> Драган
2 --> Держислав--> Добролик--> Добромисл--> Добрибій--> Дорогочин
3 --> Дорогомисл--> Дмитро--> Драгомир--> Драган
4 --> Дарій--> Даромир--> Дарій--> Добролик--> Добровіст--> Добрисвіт--> Драган--> Добролюб--> Доброслав--> Доброслав--> Доморад
5 --> Дмитро--> Дивозір--> Дантур--> Дан
6 --> Дем'ян--> Дорогомир--> Драгомир
7 --> Даромир--> Дорофій--> Данило--> Дарибог--> Далемир
8 --> Добрик--> Держикрай
9 --> Демид--> Данило
=--------------------=
[['Далемир', 'Добрибій', 'Добрисвіт', 'Добролик', 'Держикрай', 'Дан', 'Добриня'], ['Дорогобуг', 'Далібор', 'Дорогобуг', 'Добриня', 'Данко', 'Доброгост', 'Драган'], ['Держислав', 'Доброли

Process finished with exit code 0
```

Частина 2. Хешування відкритою адресацією.

1. У бібліотеці Hash, згідно описаних в теоретичній частині алгоритмів,створити функції HashInsert(…), HashSearch(…), HashDelete(…) таHashShow(…)для вставки, пошуку, видалення даних та відображення хеш-таблиці з відкритою адресацією. Функція HashDelete(…)повинна бути реалізована так, щоб порожня комірка таблиці відрізнялася від комірки з видаленими даними (див. Теоретичні відомості).

2. У цій же бібліотеці створити хеш-функції для реалізації лінійного квадратичного та подвійного дослідження. Допоміжну хеш-функцію реалізувати методом ділення (див. Теоретичні відомості).

3. Створити новий проект Lab_4_2 та підключити до нього бібліотеку Hash. У функції main() проекту створити хеш-таблицю розміру m. Розмір таблиці повинен задаватися динамічно.

4. У функції main() проекту реалізувати меню для вибору різних варіантів дослідження. Для кожного з варіантів досліджень створити підменю для реалізації операцій вставки, пошуку, видалення даних у хеш-таблиці та відображення отриманих результатів.

5. Відкомпілювати проект та продемонструвати його роботу для набору даних, отриманого від викладача.

5.1.У випадку квадратичного дослідження підбором констант c1 та c2 добитися повного заповнення хеш-таблиці.

5.2.У випадку подвійного дослідження повного заповнення таблиці можна досягнути, обравши такі допоміжні функції:h k k m 1 □□□ mod , h k k m 2 □□□□□ 1 mod 1 □□□□

```
decor = "=" + "-" * 20 + "="


class Node:  # Структура даних вузла
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

    def __str__(self):
        return "<Node: (%s, %s), next: %s>" % (self.key, self.value, self.next
!= None)

    def __repr__(self):
        return str(self)


class HashTable:  # Хеш-таблиця з окремим ланцюжком
    """Створення таблиці"""

    def __init__(self):
        self.capacity = 50  # Ємність для внутрішнього масиву
        self.size = 0
```

```
        self.buckets = [None] * self.capacity

    def hash(self, key):  # Створює хеш для заданого ключа
        hashsum = 0
        for idx, c in enumerate(key):  # Для кожного символу в ключі
            hashsum += (idx + len(key)) ** ord(c)
            hashsum = hashsum % self.capacity
        return hashsum

    def hash_insert(self, key, value):  # Вставте пару ключ-значення в хеш-
таблицю Output: void
        self.size += 1
        index = self.hash(key)
        node = self.buckets[index]  # Перейти до вузла, що відповідає хешу
        if node is None:  # Якщо вузол пустий Створити вузол, додати його,
повернути
            self.buckets[index] = Node(key, value)
            print(self.buckets)

            return
        prev = node
        while node is not None:
            prev = node
            node = node.next
        prev.next = Node(key, value)  # Додайте новий вузол у кінець списку з
указаним ключем/значенням

    """
    Знайдіть значення даних на основі ключа
    Вхід: ключ - рядок
    Вивід: значення зберігається під "ключем" або "Немає", якщо не знайдено
    """

    def hash_search(self, key):  # Пошук
        index = self.hash(key)
        node = self.buckets[index]

        while node is not None and node.key != key:
            node = node.next
        if node is None:
            return None
        else:
            return node.value


    """
    Видалити вузол, збережений у ключі
    Вхід: ключ - рядок
    Вихід: видалено значення даних або Немає, якщо не знайдено
    """

    def hash_delete(self, key):
        index = self.hash(key)
        node = self.buckets[index]
        prev = None
        while node is not None and node.key != key:
            prev = node
            node = node.next
        if node is None:
            return None
        else:
            self.size -= 1
            result = node.value
            if prev is None:
                self.buckets[index] = node.next
            else:
```

```python
                prev.next = prev.next.next
            return result  # Повернути видалений результат

    def show_table(self):
        array_inside = []

        for i in range(len(self.buckets)):
            print(i, end=" ")
            for j in self.buckets[i]:
                print('-->', end=" ")
                print(j, end='')
            print()
        print(decor)


ht = HashTable()  # Створіть нову хеш-таблицю

ht.hash_insert("phoneDirectory", ["555-555-5555", "444-444-4444"])
ht.hash_insert("phoneDirectory1", ["555-555-5555", "4434-444-4444"])
ht.hash_insert("phoneDirectory2", ["555-555-5555", "444-66676-4444"])
ht.hash_insert("phoneDirectory3", ["555-555-5555", "444-4343-4444"])
ht.hash_insert("phoneDirectory4", ["555-555-5555", "4434-444-4444"])
ht.hash_insert("phoneDirectory5", ["555-555-5555", "444-66676-4444"])
ht.hash_insert("phoneDirectory6", ["555-555-5555", "444-4343-4444"])

print(ht.hash_search("phoneDirectory"))
print(ht.hash_search("phoneDirectory5"))
```

```
"C:\Users\Dmytro\Desktop\ЛНУ-Рік-2\1 півріччя\АСД\file\Scripts\python.exe" "C:\Users\Dmytro\Desktop\ЛНУ-Рік-2\1 півріччя\АСД\Lab_04\file\main_2.py"
[None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None,
[None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None,
[None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, <Node: (phoneDirectory2, ['555-555-5555', '444-66676-4444']), next: False>, None, No
[None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, <Node: (phoneDirectory2, ['555-555-5555', '444-66676-4444']), next: False>, None, No
[None, None, None, None, None, None, None, None, <Node: (phoneDirectory4, ['555-555-5555', '4434-444-4444']), next: False>, None, None, None, None, None, None, None, None, None, <Node: (phoneDi
[None, None, None, None, None, None, <Node: (phoneDirectory5, ['555-555-5555', '444-66676-4444']), next: False>, None, <Node: (phoneDirectory4, ['555-555-5555', '4434-444-4444']), next: False>,
[None, None, None, None, None, None, <Node: (phoneDirectory5, ['555-555-5555', '444-66676-4444']), next: False>, None, <Node: (phoneDirectory4, ['555-555-5555', '4434-444-4444']), next: False>,
['555-555-5555', '444-444-4444']
['555-555-5555', '444-66676-4444']

Process finished with exit code 0
```

Висновок: На цій лабораторній роботі я ознайомився з такими темами Хешування методом ланцюгів. Хешування відкритою адресацією.