Міністерство освіти і науки України

Національний технічний університет

«Харківський політехнічний інститут»

Навчально-науковий інститут комп’ютерних наук та інформаційних технологій

ЗВІТ

про виконання лабораторної роботи №6

Варіант 7

за дисципліною

«Алгоритми і структури даних»

Виконав:

студент групи КН-320Б

Миргород В.І.

Перевірила:

старший викладач

Мошко Є.О.

Харків 2022

**Лабораторна робота №6**

**Тема лабораторної роботи:** Хеш-таблиці та робота з ними

**Мета:** Вивчити роботу алгоритмів: прямої адресації, хеш-таблиці і відкритої  
адресації. Реалізувати перелічені вище алгоритми і провести їх порівняльний аналіз.

**Завдання за варіантом:**

|  |  |  |
| --- | --- | --- |
| **7** | 56, 18, 55, 9, 50, 1, 10, 88, 27 | 36, 37, 38, 39, 40 |

**Завдання:**

1. Нехай розмір хеш-таблиці дорівнює m = 20, а хеш-функція має вигляд:  
h(k) = [m(kA mod 1)], где А = (5 - l)/2.  
В які позиції потраплять ключі (див. таблицю, стовпець 3)?  
2. Як буде виглядати хеш-таблиця з ланцюжками після того, як в неї  
послідовно помістили елементи з ключами (див. таблицю, стовпець 2) (в  
зазначеному порядку)? Число позицій в таблиці дорівнює 9, хеш-функція має вигляд h(k) = k mod 9.  
3. Виконайте додавання ключів (в зазначеному порядку, див. таблицю  
стовпець 2) в хеш-таблицю з відкритою адресацією розміру m = 11. Для обчислення послідовності проб використовується лінійний метод з h'(k) = k mod m. Виконайте те ж завдання, якщо використовується квадратичний метод з тієї ж h', c1 = 1, c2 = 3, а також для подвійного хешування з h1 = h' и h2(k) = 1 + (k mod (m - 1)) .

Аналіз завдання:

Хеш-таблиця – це структура даних, що реалізовує інтерфейс асоціативного масиву. Така таблиця складається з пар ключів та значень. Для формування хеш-таблиці застосовується хеш-функція – це така функція, яка за значенням ключа підраховує індекс цього елемента у таблиці. Хеш-таблиця дозволяє швидко знаходити дані за значенням ключа.

Структура вхідних та вихідних даних:

Початкові дані – значення ключів за варіантом. Додаткові дані користувач може ввести з клавіатури.

**Алгоритм розв’язання задачі:**

Завдання 1:

Створюємо хеш-таблицю, додаємо задані ключі у таблицю. Користувач має можливість додати або видалити рядок таблиці. Використовуємо звичайне хешування, що не передбачає обробку колізії.

Метод вставки значення:

void HashTable::insertItem(int key, int data)

{

if (Count() == m && type == 1) {

cout << "Таблица заполнена.\n";

return;

}

int index = hashFunction(key);

if (line[index].key != 0) {

if (type == 1) {

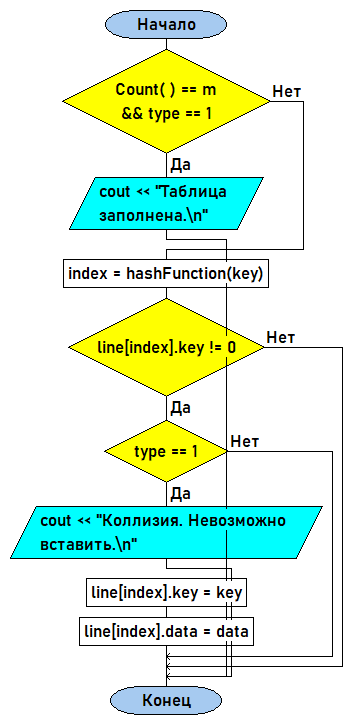
cout << "Коллизия. Невозможно вставить.\n";

return;

line[index].key = key;

line[index].data = data;

}



Індекс елемента в таблиці визначається хеш-функцією за формулою.

int hashFunction(int key)

{

double x = (key \* A);

int k = (x \* 1000);

k = k % 1000;

x = ((double)k / 1000);

return x \* m;

}

Щоб видалити рядок (пару значень), викликається метод deleteItem(). За ключем підраховується значення хеш-функції, далі за знайденим індексом видаляється рядок з таблиці.

void HashTable::deleteItem(int key)

{

int index = hashFunction(key);

if (line[index].key == key) {

line[index].key = 0;

line[index].data = 0;

}

else {

while (line[index].key != key) {

index++;

if (index >= m) {

cout << "Такого элемента нет.\n";

return;

}

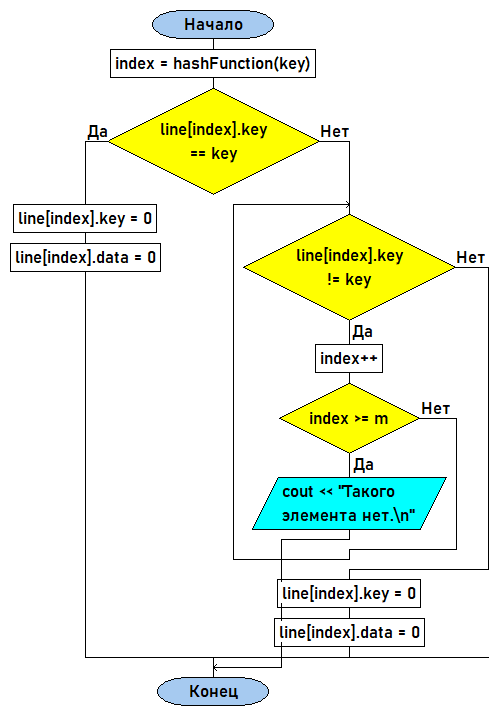
}

line[index].key = 0;

line[index].data = 0;

}

}



Завдання 2:

Створити хеш-таблицю. Під час хешування застосувати метод ланцюжків для розв’язання колізії.

Особливість такого методу – в комірці даних зберігається не тільки інформація, а й покажчик на наступну пару значень, що має такий самий результат хеш-функції.

void insertItem(int key, int data) {

if (Count() == m) {

cout << "Таблица заполнена.\n";

return;

}

int index = hashFunction(key);

if (list[index].key != 0) {

HashTableList\* temp = &list[index];

while (temp->next != NULL)

temp = temp->next;

temp->next = new HashTableList;

temp->next->key = key;

temp->next->data = data;

return;

}

list[index].key = key;

list[index].data = data;

}

Під час додавання нового значення, перевіряємо рядок на пустоту. Якщо вже є значення, то додаємо цей елемент після попереднього, та встановлюємо покажчик.

void deleteItem(int key) {

int index = hashFunction(key);

if (list[index].key == 0)

return;

if (list[index].next == NULL) {

list[index].key = 0;

list[index].data = 0;

return;

}

if (list[index].key != key) {

HashTableList\* temp = list[index].next;

while (temp->next != NULL) {

if (temp->key == key) {

temp->key = temp->next->key;

temp->data = temp->next->data;

}

temp = temp->next; //последний

}

HashTableList\* temp1 = &list[index];

while (temp1->next->next != NULL) {

temp1 = temp1->next;

}

temp1->next = NULL;

delete temp;

return;

}

HashTableList\* temp = &list[index];

list[index].key = temp->next->key;

list[index].data = temp->next->data;

while (temp->next->next != NULL) {

temp->key = temp->next->key;

temp->data = temp->next->data;

temp = temp->next;

}

delete temp->next;

temp->next = NULL;

}

Видалення елементів в такій хеш-таблиці відрізняється додатковими перевірками, та зміною покажчиків за потреби.

Завдання 3:

Створити хеш-таблицю. Для розв’язання колізії використовувати метод лінійного хешування, квадратичного хешування та подвійного хешування.

Ці методи допомагають запобігти колізії в хеш-таблиці. Головний принцип їх роботи – під час виникнення колізії перераховувати хеш-функцію за її правилом.

Також потрібно перевіряти, якщо таблиця заповнена значеннями, то динамічно збільшувати її розмір.

Метод лінійного хешування:

int linealHash(int key) {

int hash = hashFunction(key);

int i = 0;

while (line[hash + i].key != 0) {

i++;

}

return hash + i;

}

Метод квадратичного хешування:

int squareHash(int key) {

int hash = hashFunction(key);

int i = 0;

hash = (hash + i + 3 \* i \* i) % m;

while (line[hash].key != 0) {

hash = (hash + i + 3 \* i \* i) % m;

i++;

}

return hash;

}

Метод подвійного хешування:

int doubleHash(int key) {

int hash = hashFunction(key);

int i = 0;

hash = (hash + i \* (1 + (key % (m - 1)))) % m;

while (line[hash].key != 0) {

hash = (hash + i \* (1 + (key % (m - 1)))) % m;

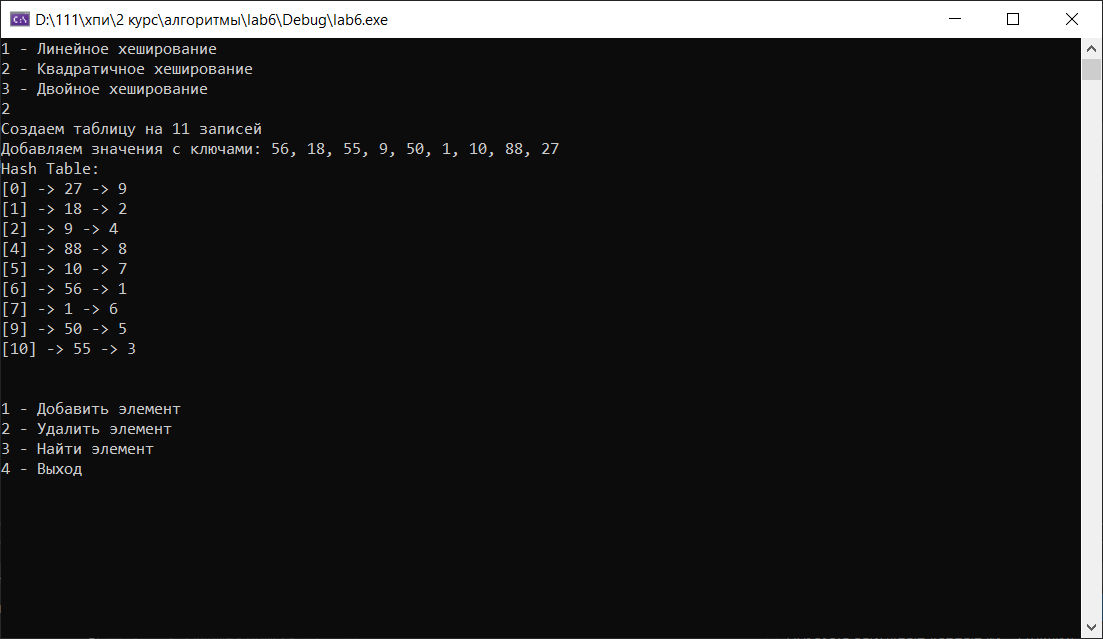
i++;

}

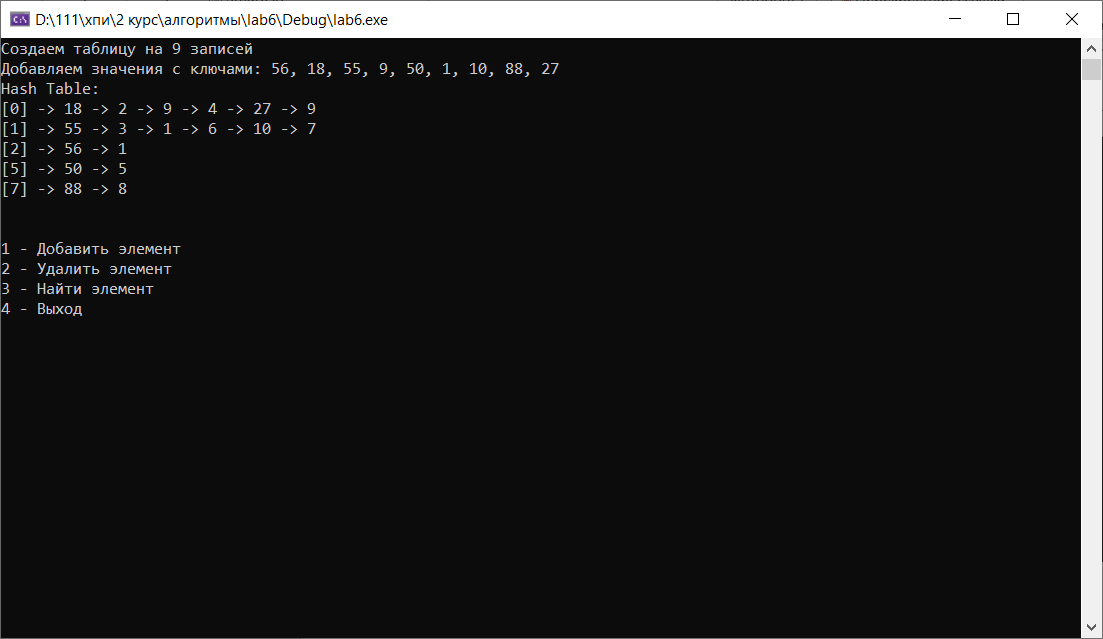
return hash;

}

Приклад роботи програми: квадратичне хешування



Приклад роботи програми: метод ланцюгів



Оцінки складності алгоритмів для хеш-таблиці

Алгоритми прямої адресації:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Найкращий | Середній | Найгірший |
| Додавання елементу | O(1) | O(n/k) | O(n) |
| Пошук | O(1) | O(n/k) | O(n) |
| Видалення | O(1) | O(n/k) | O(n) |

Алгоритми відкритої адресації:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Найкращий | Середній | Найгірший |
| Додавання елементу | O(1) | O(1) | O(1) |
| Пошук | O(1) | O(1) | O(1) |
| Видалення | O(1) | O(1) | O(1) |

**Висновки:**

Хеш-таблиці є досить зручним структурою для зберігання асоціативних даних. Вони дозволяють швидко додавати, видаляти та знаходити потрібні дані в таблиці.

Але під час використання хеш-таблиць може з’являтися колізія – ситуація, коли для кількох різних ключів хеш-функція має однакове значення. Існують деякі методи розв’язання колізії – методи ланцюгів, методи відкритої адресації. Найбільш ефективними є методи відкритої адресації. Найкращим з них вважається метод подвійного хешування. Алгоритми відкритої адресації дозволяють мати складність O(1) для будь-якої дії навіть у найгіршому випадку.