Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №** 6

по дисципліні «Теорія алгоритмів»

на тему: Пошукові структури даних. Хеш-таблиці і хеш-функції

|  |  |
| --- | --- |
| Виконав:  студент групи ІА-з91  Москаленко В.В.  Дата здачі \_\_\_\_\_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_\_\_\_\_ | Перевірив:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Київ 2020

1. **Задача**

Реалізувати хеш-функцію для користувацького типу даних.

Реалізувати пошукову хеш-таблицю.

1. **Постановка проблеми.**

Реалізувати хеш-функцію для користувацького типу даних(ім’я та прізвище), реалізувати пошукову хеш-таблицю для розв’язку використати власну структуру даних. Необхідно вручну реалізувати хешування рядків та пам’ятати, що хеш формується для забезпечення рівномірного розподілу даних.

1. **Побудова моделі.**

Використовуємо власну структуру даних, яка складається з масивів та змінних, що використовується для хешування користувацького типу даних. Механізм хешування для типу string буде імплементований за допомогою базової або подвійної хеш-функції.

1. **Розроблення алгоритму.**

Первинна хеш-функція ключа виконується за допомогою використання контрольної суми або SHA. Як правило хеш-функції будують на основі однокрокових стискаючих функцій двух змінних, де x1, x2 та y – подвійні вектори довжини m, n, n при чому n – довжина згортки, а m – довжина блока повідомлення. Для отримання значення хешу повідомлення спочатку розбивається на блоки довжини m, а потім до отриманих розбитих блоків приміняємо наступну послідовну процедуру вираховування згортки.

Ho = v,

Hi = f(Mi,Hi-1), i = 1,.., N,

h(M) = HN

Тут v – константа або ініціалізуючий вектор, вона вибирається за певним набором правил.

1. **Перевірка правильності алгоритму.**

Перевірка правильності алгоритму виконується за допомогою рівномірного розподілу значень хеш-функцій при випадковому виборі значень аргумента. Алгоритм перевіряється за двома показниками : не можливість фабрикації та неможливість модифікації. Першу вимогу ми перевірили за характеристикою високої складності підбору повідомлення з правильним значенням згортки. Другу умову ми перевірили за характеристикою високої складності підбору для заданого повідомлення з відомим значенням згортки другого повідомлення з правильним значенням згортки.

1. **Реалізація алгоритму**

import java.util.ArrayList;  
import java.util.Scanner;  
  
public class Lab6{  
public static void main(String[] args){  
Scanner sc = new Scanner([System.in](http://system.in/));  
System.out.print("Введите размерность таблицы: ");  
int n = sc.nextInt();  
System.out.println();  
Hasher<Integer, String> hash = new Hasher<Integer, String>(n);  
long start = System.nanoTime();  
for (int i=0;i<n;i++)  
hash.put(i, "John", "Freek");  
long finish = System.nanoTime();  
long ans = finish -start;  
hash.printTable();  
System.out.println("Add time - "+ ans);  
  
long start1 = System.nanoTime();  
for (int i=0;i<n;i++){  
String s = hash.get(i);  
}  
long finish1 = System.nanoTime();  
long ans1 = finish1 -start1;  
System.out.println("Search time - "+ ans1);  
}  
}  
  
class Hasher<K, V> {  
  
private static class LinkedListNode<K, V> {  
public LinkedListNode<K, V> next;  
public LinkedListNode<K, V> prev;  
public K key;  
public V value, value1;  
public LinkedListNode(K k, V v, V v1) {  
key = k;  
value = v;  
value1 = v1;  
}  
  
public String printForward() {  
String data = "(" + value + " " + value1 + ")";  
if (next != null) {  
return data + "->" + next.printForward();  
} else {  
return data;  
}  
}  
}   
  
private ArrayList<LinkedListNode<K, V>> arr;  
public Hasher(int capacity) {  
  
arr = new ArrayList<LinkedListNode<K, V>>();  
arr.ensureCapacity(capacity);  
for (int i = 0; i < capacity; i++) {  
arr.add(null);  
}  
}  
  
  
public V put(K key, V value, V value1) {  
LinkedListNode<K, V> node = getNodeForKey(key);  
if (node != null) {  
V oldValue = node.value;  
node.value = value;  
return oldValue;  
}  
  
node = new LinkedListNode<K, V>(key, value, value1);  
int index = getIndexForKey(key);  
if (arr.get(index) != null) {  
[node.next](http://node.next/) = arr.get(index);  
node.next.prev = node;  
}  
arr.set(index, node);  
return null;  
}  
  
public V remove(K key) {  
LinkedListNode<K, V> node = getNodeForKey(key);  
if (node == null) {  
return null;  
}  
  
if (node.prev != null) {  
[node.prev.next](http://node.prev.next/) = [node.next](http://node.next/);  
} else {  
int hashKey = getIndexForKey(key);  
arr.set(hashKey, [node.next](http://node.next/));  
}  
  
if ([node.next](http://node.next/) != null) {  
node.next.prev = node.prev;  
}  
return node.value;  
}  
  
public V get(K key) {  
if (key == null) return null;  
LinkedListNode<K, V> node = getNodeForKey(key);  
return node == null ? null : node.value;  
}  
  
private LinkedListNode<K, V> getNodeForKey(K key) {  
int index = getIndexForKey(key);  
LinkedListNode<K, V> current = arr.get(index);  
while (current != null) {  
if (current.key == key) {  
return current;  
}  
current = [current.next](http://current.next/);  
}  
return null;   
}  
  
public int getIndexForKey(K key) {  
return Math.abs(key.hashCode() % arr.size());  
}  
  
public void printTable() {  
for (int i = 0; i < arr.size(); i++) {  
String s = arr.get(i) == null ? "" : arr.get(i).printForward();  
System.out.println(i + ": " + s);  
}  
}  
}

1. **Перевірка програми**

Перевірка програми була здійснена на різних вхідних даних. Хеш-функція перевірена на користувацькому типі даних, можливість зациклюваності в програмі відсутня.

