ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"

Факультет КНТ

Кафедра ПИ

Индивидуальная работа

по курсу: «Алгоритмы и структуры данных»

по теме: «Хеш-таблицы»

Выполнил:

ст. гр. ПИ-19а

Саевский О.В.

Проверили:

Артёменко О.Г.

Серёженко О.А.

ДОНЕЦК – 2020

1. **Вариант индивидуальной работы**

Второй и предпоследний символы преобразуются в числа, суммируются, результат умножается на количество символов в ключе и делится по mod m (ключи могут быть разной длины). В качестве m берется n (длина вектора) или некоторое простое число p, ближайшее к n (p<n).

Метод разрешения конфликтов (коллизий) – метод прямого связывания.

**2. Теория**

Хеш-таблица (таблица с вычисляемым входом) – это такая структура, обращение к записям которой осуществляется не по ключу записи непосредственно, а по номеру элемента отображающего вектора, в который преобразуется значение ключа. Алгоритм преобразования ключа в номер (индекс) называется хеш-функцией (функцией расстановки, функцией хеширования).

В идеальном случае, когда заранее известны все пары ключ-значение, достаточно легко реализовать идеальную хеш-таблицу, в которой время поиска будет постоянным (используется идеальная хеш-функция, которая определяет положения в таблице по целым значениям и без столкновений).

Но в большинстве случаев приходится бороться с коллизиями. Обычно применяются методы цепочек и открытой индексации.

2.1. Метод цепочек (метод прямого связывания)

Этот метод часто называют открытым хешированием. Его суть проста — элементы с одинаковым хешем попадают в одну ячейку в виде связного списка.

То есть, если ячейка с хешем уже занята, но новый ключ отличается от уже имеющегося, то новый элемент вставляется в список в виде пары ключ-значение.

Если выбран метод цепочек, то вставка нового элемента происходит за *O*(1), а время поиска зависит от длины списка и в худшем случае равно *O*(n). Если количество ключей *n*, а распределяем по *m*-ячейкам, то соотношение n/m будет коэффициентом заполнения.

### 2.2. Открытая индексация (или закрытое хеширование)

Второй распространенный метод — открытая индексация. Это значит, что пары ключ-значение хранятся непосредственно в хеш-таблице. А алгоритм вставки проверяет ячейки в некотором порядке, пока не будет найдена пустая ячейка. Порядок вычисляется на лету.

Самая простая в реализации последовательность проб — линейное пробирование (или линейное исследование). Здесь все просто — в случае коллизии, следующие ячейки проверяются линейно, пока не будет найдена пустая ячейка.

А алгоритм поиска ищет ячейки в том же порядке, что и при вставке, пока не найдет нужный элемент или пустую ячейку, которая говорит о том, что ключ отсутствует. В случае если таблица будет заполнена, ее придется динамически расширять.

**3. Реализация**

Реализация хеш-функции таблицы представлена на рисунке 3.1.

public int Generate(){

int m = vector;

int first = (int)this.key[1];

int last = (int)this.key[this.key.Length - 2];

int summ = first + last;

int multiplication = summ \* this.key.Length;

int result = multiplication % m;

return result;

}

Рисунок 3.1 – Реализация хеш-функции

Реализация метода разрешения конфликтов представлена на рисунке 3.2.

if (SIZE == hsh.GetVector())

{

MessageBox.Show("Максимальная длина.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

button2.Enabled = false;

return;

}

if(textBox2.Text.Length < 2 || textBox3.Text == "")

{

MessageBox.Show("Длина ключа может быть не меньше двух.\nА текстовое поле не может быть пустым.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

int y;

if ((y = hsh.SearchHash(textBox2.Text, false)) != -1)

{

MessageBox.Show("Значение с таким ключом уже существует.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

Рисунок 3.2 – Метод решения конфликтов

**4. Исходный код**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace IndividASD

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

DataSet ds = new DataSet();

DataTable dt = new DataTable();

Hash hsh;

int SIZE = 0;

public class Hash

{

public class strh

{

public int value,

count;

}

public static int vector;

private static List<Node> list = new List<Node>();

private static List<strh> listhash = new List<strh>();

public Hash(int length)

{

vector = length;

}

public void Add(string key, string value)

{

Node node = new Node(key, value);

list.Add(node);

}

public int GetVector()

{

return vector;

}

public class Node

{

public string value;

public string key;

public int hash;

public Node(string key, string value)

{

this.key = key;

this.value = value;

this.hash = Generate();

}

public int Generate()

{

int m = vector;

int first = (int)this.key[1];

int last = (int)this.key[this.key.Length - 2];

int summ = first + last;

int multiplication = summ \* this.key.Length;

int result = multiplication % m;

bool item = false;

strh nod = new strh();

for (int i = 0; i < listhash.Count; i++)

{

if (result == listhash[i].value)

{

item = true;

listhash[i].count = listhash[i].count + 1;

continue;

}

}

if(item == false || listhash.Count == 0)

{

nod.value = result;

nod.count = 1;

listhash.Add(nod);

}

return result;

}

}

private class SearchNode

{

public string key;

public int hash;

public SearchNode(string key)

{

this.key = key;

this.hash = Generate();

}

public int Generate()

{

int m = vector;

int first = (int)this.key[1];

int last = (int)this.key[this.key.Length - 2];

int summ = first + last;

int multiplication = summ \* this.key.Length;

int result = multiplication % m;

return result;

}

}

public bool FindHash(string key)

{

SearchNode sd = new SearchNode(key);

bool find = false;

for(int i = 0; i < listhash.Count; i++)

{

if (sd.hash == listhash[i].value)

{

find = true;

break;

}

}

return find;

}

public void ClearLists()

{

list.Clear();

listhash.Clear();

}

public int GetLength()

{

return list.Count;

}

public string GetKey(int i)

{

return list[i].key;

}

public int GetHash(int i)

{

return list[i].hash;

}

public string GetValue(int i)

{

return list[i].value;

}

public void Sort()

{

Node node;

Node min;

int x;

for (int i = 0; i < list.Count; i++)

{

min = list[i];

x = i;

for (int j = i + 1; j < list.Count; j++)

{

if (min.hash > list[j].hash)

{

min = list[j];

x = j;

}

}

node = list[i];

list[i] = min;

list[x] = node;

}

strh nod;

strh mi;

for (int i = 0; i < listhash.Count; i++)

{

mi = listhash[i];

x = i;

for (int j = i + 1; j < listhash.Count; j++)

{

if (mi.value > listhash[j].value)

{

mi = listhash[j];

x = j;

}

}

nod = listhash[i];

listhash[i] = mi;

listhash[x] = nod;

}

}

public int SearchHash(string key, bool view)

{

int index = 0;

int size = 0;

int result = -1;

SearchNode snode = new SearchNode(key);

for(int i = 0; i < listhash.Count; i++)

{

if (snode.hash == listhash[i].value)

{

if(i == 0)

{

index = 0;

size = listhash[0].count;

}

else for(int a = 0; a < i; a++)

{

index = index + listhash[a].count;

size = listhash[a+1].count;

}

return SearchKey(index, key, size, view);

}

}

return -1;

}

public int SearchKey(int index, string key, int size, bool view)

{

string collision = null, findvalue = null;

int position = -1;

for(int i = index; i < (index + size); i++)

{

if (list[i].key == key)

{

position = i;

findvalue = list[i].value;

}

else

collision = collision + list[i].value + ",";

}

if (collision != null)

collision = collision.Remove(collision.Length - 1, 1);

if (position != -1 && view == true)

{

MessageBox.Show("Значение:" + findvalue + "\nПозиция: " + position + "\nЦепочка переполнения: " + collision , "Успех!",

MessageBoxButtons.OK);

}

return position;

}

public bool Delete(string key)

{

int res = SearchHash(key, false);

if (res != -1)

{

SearchNode sn = new SearchNode(key);

for(int i = 0; i < listhash.Count; i++)

{

if(sn.hash == listhash[i].value)

{

listhash[i].count--;

break;

}

}

list.RemoveAt(res);

return true;

}

else return false;

}

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void Form1\_Load(object sender, EventArgs e)

{

ds.Tables.Add("Hash-Table");

ds.Tables[0].Columns.Add("№");

ds.Tables[0].Columns.Add("Ключ");

ds.Tables[0].Columns.Add("Значение");

dataGridView1.DataSource = ds.Tables[0];

dataGridView1.Columns[0].Width = 100;

button2.Enabled = false;

button4.Enabled = false;

button5.Enabled = false;

}

private void button1\_Click(object sender, EventArgs e)

{

ds.Tables[0].Rows.Clear();

if (hsh!= null)

hsh.ClearLists();

StreamReader sr = new StreamReader("list.txt");

string line;

int hash;

int key;

string value;

string[] massive = new string[2];

hsh = new Hash(System.IO.File.ReadAllLines("list.txt").Length);

while ((line = sr.ReadLine()) != null)

{

massive = line.Split(' ');

hsh.Add(massive[0], massive[1]);

}

hsh.Sort();

for (int i = 0; i < hsh.GetLength(); i++)

{

ds.Tables[0].Rows.Add(hsh.GetHash(i), hsh.GetKey(i), hsh.GetValue(i));

}

button1.Enabled = false;

button2.Enabled = true;

button4.Enabled = true;

button5.Enabled = true;

}

private void button3\_Click(object sender, EventArgs e)

{

try

{

if (Convert.ToInt32(textBox1.Text) < 1)

{

Exception ex = new Exception();

throw ex;

}

hsh = new Hash(Convert.ToInt32(textBox1.Text));

ds.Tables[0].Rows.Clear();

hsh.ClearLists();

SIZE = 0;

}

catch(Exception ex)

{

MessageBox.Show("Значение заполнено неверно. Заполните еще раз!", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

button2.Enabled = true;

button4.Enabled = true;

button5.Enabled = true;

}

private void button2\_Click(object sender, EventArgs e)

{

if (SIZE == hsh.GetVector())

{

MessageBox.Show("Максимальная длина.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

button2.Enabled = false;

return;

}

if(textBox2.Text.Length < 2 || textBox3.Text == "")

{

MessageBox.Show("Длина ключа может быть не меньше двух.\nА текстовое поле не может быть пустым.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

int y;

if ((y = hsh.SearchHash(textBox2.Text, false)) != -1)

{

MessageBox.Show("Значение с таким ключом уже существует.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

ds.Tables[0].Rows.Clear();

hsh.Add(textBox2.Text, textBox3.Text);

hsh.Sort();

if (hsh.FindHash(textBox2.Text) == false)

SIZE++;

for (int i = 0; i < hsh.GetLength(); i++)

{

ds.Tables[0].Rows.Add(hsh.GetHash(i), hsh.GetKey(i), hsh.GetValue(i));

}

}

private void button5\_Click(object sender, EventArgs e)

{

if (textBox4.Text.Length < 2)

{

MessageBox.Show("Длина ключа может быть не меньше трёх.\nА текстовое поле не может быть пустым.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

if(hsh.Delete(textBox4.Text) == true)

{

ds.Tables[0].Rows.Clear();

for(int i = 0; i < hsh.GetLength(); i++)

{

ds.Tables[0].Rows.Add(hsh.GetHash(i), hsh.GetKey(i), hsh.GetValue(i));

}

}

else

{

MessageBox.Show("Нет значения в таблице.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

}

private void button4\_Click(object sender, EventArgs e)

{

if (textBox4.Text.Length < 2)

{

MessageBox.Show("Длина ключа может быть не меньше двух.\nА текстовое поле не может быть пустым.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

int res = hsh.SearchHash(textBox4.Text, true);

if (res == -1)

{

MessageBox.Show("Нет значения в таблице.", "Ошибка!",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

}

private void textBox4\_TextChanged(object sender, EventArgs e)

{

}

}

}

**5 Экранные формы**





