МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Отчет

по лабораторной работе №6

по курсу «Конструирования программного обеспечения»

по теме «Организация таблиц в трансляторах и работа с ними»

Выполнил:

ст. гр. ПИ-19а

Саевский О.В.

Проверили:

Чернышова А.В.

Московченко А. В.

Донецк – 2022

**Вариант 11**:Таблица меток.

Хеш-таблица — это структура данных, в которой все элементы хранятся в виде пары ключ-значение, где:

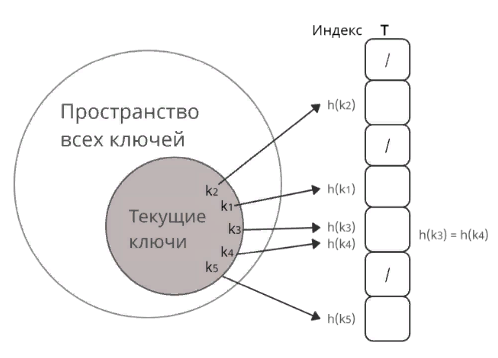
ключ — уникальное число, которое используется для индексации значений;

значение — данные, которые с этим ключом связаны.

В хеш-таблице обработка новых индексов производится при помощи ключей. А элементы, связанные с этим ключом, сохраняются в индексе. Этот процесс называется хешированием.

Пусть k — ключ, а h(x) — хеш-функция.

Тогда h(k) в результате даст индекс, в котором мы будем хранить элемент, связанный с k.

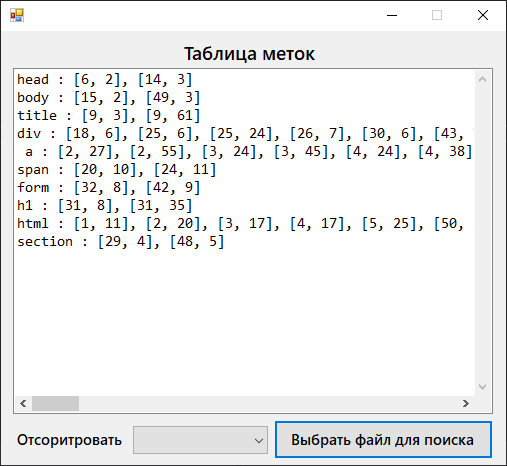


Когда хеш-функция генерирует один индекс для нескольких ключей, возникает конфликт: неизвестно, какое значение нужно сохранить в этом индексе. Это называется коллизией хеш-таблицы.

Есть несколько методов борьбы с коллизиями:

метод цепочек;

метод открытой адресации: линейное и квадратичное зондирование.



using System;

using System.Collections.Generic;

using System.IO;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace Operations\_Table

{

public partial class MainForm : Form

{

Table table = null;

string[] operations = { "head", "body", "title", "div", "a", "span", "form", "h1", "html", "section" };

public MainForm()

{

InitializeComponent();

OperationsList.SelectedText = "";

}

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

{

if (OpenSource.ShowDialog() == DialogResult.OK)

{

FilePath.Text = OpenSource.FileName;

table = new Table();

OperationsList.Items.Clear();

LinesList.Items.Clear();

StreamReader reader = new StreamReader(OpenSource.FileName);

string line;

int lineNumber = 0;

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

{

++lineNumber;

for (int i = 0; i < operations.Length; ++i)

{

MatchCollection collection = Regex.Matches(line, Regex.Escape(operations[i]));

if (collection.Count > 0)

{

table.AddKey(operations[i]);

for (int j = 0; j < collection.Count; ++j)

{

bool isSingleOperator = operations[i].Length == 1;

bool left = collection[j].Index > 0 && (line[collection[j].Index - 1] == operations[i][0]);

bool right = collection[j].Index < line.Length - 1 && ((line[collection[j].Index + 1] == operations[i][0]) || (line[collection[j].Index + 1] == '='));

bool isChar = (collection[j].Index < line.Length - 1 && line[collection[j].Index + 1] == '\'') && (collection[j].Index > 0 && line[collection[j].Index - 1] == '\'');

if (!(isSingleOperator && (left || right || isChar)))

{

left = false;

int count = 0;

int p = collection[j].Index;

while (p > 0 && !left)

{

if (line[--p] == '\"')

{

++count;

}

if (p == 0 && count % 2 != 0)

{

left = true;

}

}

right = false;

count = 0;

p = collection[j].Index;

while (p < line.Length - 1 && !right)

{

if (line[++p] == '\"')

{

++count;

}

if (p == line.Length - 1 && count % 2 != 0)

{

right = true;

}

}

if (!left && !right)

{

table.AddAttribute(operations[i], new Position(lineNumber, collection[j].Index + 1));

}

}

}

}

}

}

reader.Close();

table.Reorganize();

string[] selectedOperations = table.GetKeys().ToArray();

OperationsList.Items.Add("");

OperationsList.Items.AddRange(selectedOperations);

OperationsList.SelectedIndex = 0;

}

}

private void OperationsList\_SelectedIndexChanged(object sender, EventArgs e)

{

LinesList.Items.Clear();

if (OperationsList.SelectedIndex > -1)

{

if (OperationsList.SelectedItem.ToString().Equals(""))

{

LinesList.Items.Clear();

for (int i = 0; i < operations.Length; ++i)

{

string key = operations[i];

string item = (key.Length < 2 ? " " + key : key) + " : ";

List<Position> attributes = table.GetAttributes(key);

if (attributes != null)

{

item += string.Join(", ", attributes);

}

LinesList.Items.Add(item);

}

}

else

{

List<Position> positions = table.GetAttributes(OperationsList.SelectedItem.ToString());

foreach (Position position in positions)

{

LinesList.Items.Add("[" + position.Line + ", " + position.Column + "]");

}

}

}

}

}

}

using System.Collections.Generic;

namespace Operations\_Table

{

class Table

{

private Dictionary<string, List<Position>> table;

public Table()

{

table = new Dictionary<string, List<Position>>();

}

public bool AddKey(string key)

{

if (!table.ContainsKey(key))

{

table.Add(key, new List<Position>());

return true;

}

else

{

return false;

}

}

public bool AddAttribute(string key, Position attribute)

{

if (table.ContainsKey(key))

{

if (table[key].Find(value => value.Line == attribute.Line && value.Column == attribute.Column) == null)

{

table[key].Add(attribute);

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

public List<string> GetKeys()

{

return new List<string>(table.Keys);

}

public List<Position> GetAttributes(string key)

{

if (table.ContainsKey(key))

{

return table[key];

}

else

{

return null;

}

}

public void Reorganize()

{

List<string> keys = new List<string>(table.Keys);

keys = new List<string>(table.Keys);

foreach (string key in keys)

{

if (table[key].Count <= 0)

{

table.Remove(key);

}

}

}

}

class Position

{

public int Line;

public int Column;

public Position(int line, int column)

{

Line = line;

Column = column;

}

public override string ToString()

{

return "[" + Line + ", " + Column + "]";

}

}

}