Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

­­­­­

Основы алгоритмизации и программирования

Отчет по лабораторной работе №15

«Хеширование»

Выполнил: Усов А.М.

Студент группы 310901

Преподаватель: Кабариха В. А.

Минск 2023

Цель: сформировать знания и умения по работе с подпрограммами, приобрести навыки написания программ с использованием хеш-функций.

Задание . Составить хеш-функцию в соответствии с заданным вариантом и

проанализировать ее. При необходимости доработать хеш-функцию.

Используя полученную хеш-функцию разработать на языке

программирования С++ программу, которая должна выполнять следующие

функции:

* создавать хеш-таблицу;
* добавлять элементы в хеш-таблицу;
* просматривать хеш-таблицу;
* искать элементы в хеш-таблице;
* удалять элементы из хеш-таблицы.

Описание хеш-функции

Хеш-функция основана на возведении суммы кодов символов ключа в

квадрат и извлечение из полученного квадрата нескольких средних цифр. При

этом коды символов умножаем на частное кода и произведения тройки на

порядковый номер символа в ключе (1ч6). Звучит убого, вот так выглядит

формула суммы:

где - код символа с индексом "i";

Возведенная в квадрат сумма колеблется от 7997584 до 22781529, а это

семизначное или восьмизначное число. Для адресации сегментов хештаблицы необходимо четырехзначное число, не превышающее 2000. Откинем

у квадрата суммы 2 первых и два последних разряда, так у нас получится

трехзначное или четырехзначное число. Для того, чтобы адрес не превысил

максимально допустимый адрес 1999, будем брать остаток от деления на 2000

до тех пор, пока он не попадет в нужный диапазон.

*Экспериментальный анализ хеш-функции*

Экспериментальное исследование проводится следующим образом:

формируются случайным образом ключи заданного формата в

количестве, превышающем количество сегментов хеш-таблицы в 2…3 раза;

для каждого сформированного ключа вычисляется хеш-функция, и

подсчитывается, сколько раз вычислялся адрес того или иного сегмента хештаблицы.

#include <iostream>

#include <string>

using namespace std;

template <typename T>

class Stack {

class Node {

T data;

Node\* prev;

public:

Node(): prev(nullptr) {}

Node(T data, Node\* node): data (data), prev(node) {}

T getData() { return data; }

Node\* getPrev() { return prev; }

void setPrev(Node\* prev) { this->prev = prev; }

void setData(T data) { this->data = data; }

};

Node\* tail;

int size;

public:

Stack(): tail(nullptr), size(0) {}

void push(T data){

if (tail == nullptr) {

tail = new Node(data,nullptr);

}

else {

Node\* newNode = new Node(data,tail);

tail = newNode;

}

size++;

}

T pop() {

if (size == 0) {

throw "Stack is empty";

}

T data = tail->getData();

tail = tail->getPrev();

size--;

return data;

}

T peek() {

if (size == 0) {

throw "Stack is empty";

}

return tail->getData();

}

bool isEmpty() {

return size == 0;

}

};

float solveOPZ(string expression) {

Stack<float> stack;

for (int i = 0; i < expression.size(); i++) {

if (expression[i] == ' ') {

continue;

}

int x = expression[i] -'0';

if (expression[i] >= '0' && expression[i] <= '9') {

float number = 0;

while (expression[i] != ' ' && i < expression.size()) {

number = number \* 10 + (expression[i] - '0');

i++;

}

stack.push(number);

}

else {

float a = stack.pop();

float b = stack.pop();

switch (expression[i]) {

case '+': {

stack.push(a + b);

break;

}

case '-': {

stack.push(b - a);

break;

}

case '\*': {

stack.push(a \* b);

break;

}

case '/': {

if (a == 0) {

throw "Division by zero";

}

stack.push(b/ a);

break;

}

}

}

}

if (stack.isEmpty()) {

throw "Invalid expression";

return -1;

}

else {

return stack.pop();

}

}

int main() {

string expression1 = "3 1 +";

string expression2 = "12 5 \* 10 -";

string expression3 = "1 2 30 + \*";

string expression4 = "2 10 + 2 4 + 6 – 2 /";

cout << solveOPZ(expression1) << endl;

cout << solveOPZ(expression2) << endl;

cout << solveOPZ(expression3) << endl;

cout << solveOPZ(expression4) << endl;

if (solveOPZ(expression1) == 4) {

cout << "Test 1 passed" << endl;

}

else {

cout << "Test 1 failed" << endl;

}

if (solveOPZ(expression2) == 50) {

cout << "Test 2 passed" << endl;

}

else {

cout << "Test 2 failed" << endl;

}

if (solveOPZ(expression3) == 32) {

cout << "Test 3 passed" << endl;

}

else {

cout << "Test 3 failed" << endl;

}

if (solveOPZ(expression4) == 6) {

cout << "Test 4 passed" << endl;

}

else {

cout << "Test 4 failed" << endl;

}

string expression;

cout << "Enter expression in reverse polish notation: " << endl;

cout << " use this format: 3 1 + " << endl;

getline(cin, expression);

cout << "Result : " << solveOPZ(expression) << endl;

return 0;

}

# Результат работы программы представлен на рисунке 1.

# 

Рисунок 1 – Результат выполнения программы