Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

ДИСКРЕТНІ СТРУКТУРИ

ЛАБОРАТОРНА РОБОТА №2

ТЕМА: Хеш-таблиця

Виконали:

Студенти гр. ІТ-82

Бірюченко О.

Жук О.

Михайлова Д.

Кавун Ф.

Перевірив:

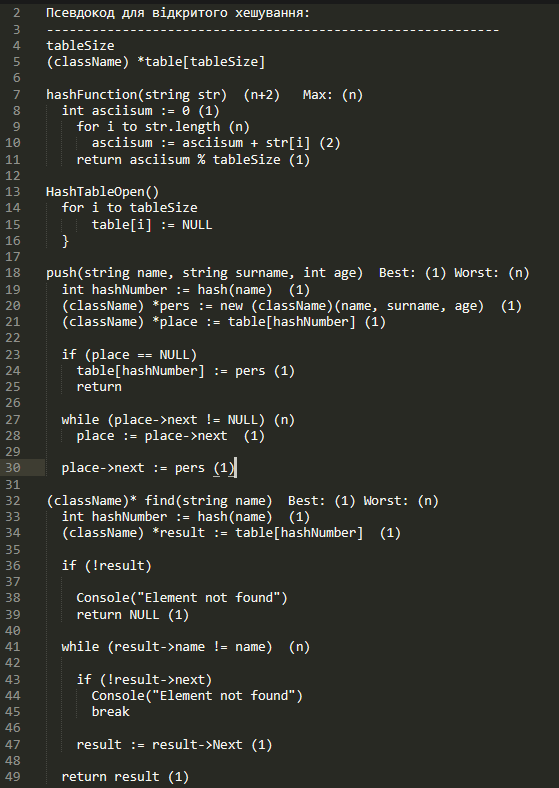
Сергеєв Д. С..

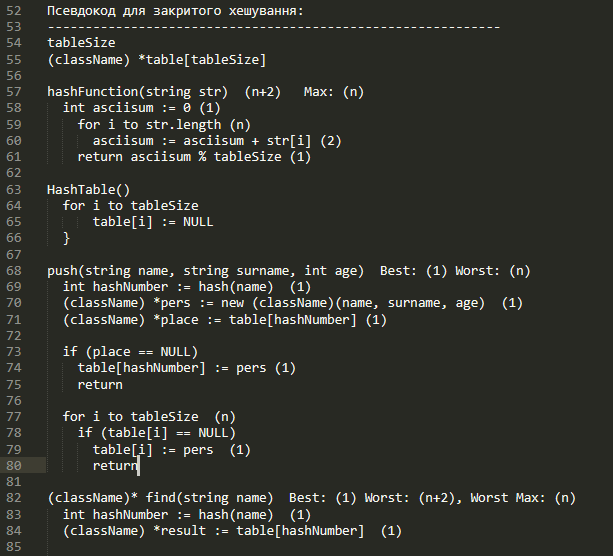
Київ – 2019

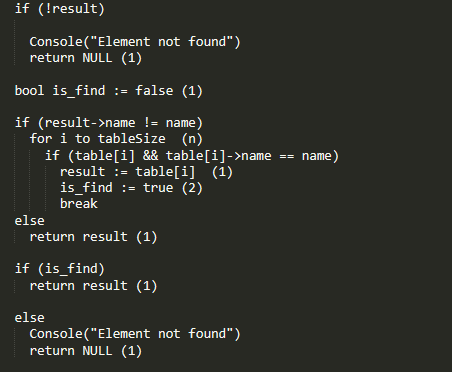
1. Обране хешування:

Закрите (відкрита індексація) та відкрите хешування (метод ланцюжків).

2. Складність алгоритму та псевдокод:







3. Програма:

#include "pch.h"

#include <iostream>

#include <string>

#include <list>

#include <algorithm>

#include <vector>

#include <fstream>

#include <ctime>

#define PRIME\_SIZE 1000

using namespace std;

class Person

{

public:

string name;

string surname;

int age;

Person(string name, string surname, int age = 0)

{

this->name = name;

this->surname = surname;

this->age = age;

}

};

class HashTable

{

Person \*table[PRIME\_SIZE];

static int hash(string str)

{

int asciisum = 0;

for (int i = 0; i < str.length(); i++)

{

asciisum += str[i];

}

return asciisum % PRIME\_SIZE;

}

public:

HashTable()

{

for (int i = 0; i < PRIME\_SIZE; i++)

{

table[i] = NULL;

}

}

~HashTable()

{

for (int i = 0; i < PRIME\_SIZE; i++)

{

delete table[i];

}

}

void push(string name, string surname, int age)

{

int hashNumber = hash(name);

Person \*pers = new Person(name, surname, age);

Person \*place = table[hashNumber];

if (place == NULL)

{

table[hashNumber] = pers;

return;

}

for (int i = 0; i < PRIME\_SIZE; i++) if (table[i] == NULL) { table[i] = pers; return; }

}

Person\* find(string name)

{

int hashNumber = hash(name);

Person \*result = table[hashNumber];

if (!result)

{

cout << "Element not found" << endl;

return NULL;

}

bool is\_find = false;

if (result->name != name)

{

for (int i = 0; i < PRIME\_SIZE; i++)

{

if (table[i] && table[i]->name == name)

{

result = table[i];

is\_find = true;

break;

}

}

}

else

{

return result;

}

if (is\_find) return result;

else

{

cout << "Element not found" << endl;

return NULL;

}

}

};

class PersonOpen

{

public:

PersonOpen \*next;

string name;

string surname;

int age;

PersonOpen()

{

this->next = NULL;

}

PersonOpen(string name, string surname, int age = 0)

{

this->name = name;

this->surname = surname;

this->age = age;

this->next = NULL;

}

~PersonOpen()

{

if (this->next != NULL)

{

delete this->next;

}

}

};

class HashTableOpen

{

PersonOpen \*table[PRIME\_SIZE];

static int hash(string str)

{

int asciisum = 0;

for (int i = 0; i < str.length(); i++)

{

asciisum += str[i];

}

return asciisum % PRIME\_SIZE;

}

public:

HashTableOpen()

{

for (int i = 0; i < PRIME\_SIZE; i++)

{

table[i] = NULL;

}

}

~HashTableOpen()

{

for (int i = 0; i < PRIME\_SIZE; i++)

{

delete table[i];

}

}

void push(string name, string surname, int age)

{

int hashNumber = hash(name);

PersonOpen \*pers = new PersonOpen(name, surname, age);

PersonOpen \*place = table[hashNumber];

if (place == NULL)

{

table[hashNumber] = pers;

return;

}

while (place->next != NULL)

{

place = place->next;

}

place->next = pers;

}

PersonOpen\* find(string name)

{

int hashNumber = hash(name);

PersonOpen \*result = table[hashNumber];

if (!result)

{

cout << "Element not found" << endl;

return NULL;

}

while (result->name != name)

{

if (!result->next)

{

cout << "Element not found" << endl;

break;

}

result = result->next;

}

return result;

}

};

int main()

{

ifstream fin;

vector <string> inp, inpC;

for (int i = 97; i < 107; i++) {

for (int j = 97; j < 107; j++) {

for (int k = 97; k < 107; k++) {

char s[] = { (char)i, (char)j, (char)k };

string str(s, 3);

inpC.push\_back(str);

}

}

}

for (int i = 97; i < 107; i++) {

for (int j = 98; j < 107; j++) {

for (int k = 99; k < 107; k++) {

char s[] = { (char)i, (char)j, (char)k };

string str(s, 3);

inp.push\_back(str);

}

}

}

cout << "-----------------Add names to hashtable----------------\n";

HashTable newTable, newTableC;

string x;

unsigned int start\_time, finish\_time, count = 0;

string letter;

fin.open("name1M.txt");

start\_time = clock();

while (!fin.eof() && count < 700) {

fin >> x;

newTable.push(inp[count], x, (rand() % 99));

count++;

}

finish\_time = clock();

fin.close();

cout << "Add elements without collision: " << (finish\_time - start\_time)<< "ms\n";

fin.open("name1M.txt");

count = 0;

start\_time = clock();

while (!fin.eof() && count < 700) {

fin >> x;

newTableC.push(inpC[(rand() % 500)], x, (rand() % 99));

count++;

}

finish\_time = clock();

fin.close();

cout << "Add elements with collision: " << (finish\_time - start\_time) << "ms\n";

cout << "-----------------------Find element----------------------\n";

double dstart\_time = (double)clock();

Person \* search = newTable.find("abc");

double dfinish\_time = (double)clock();

if (search)

{

cout << search->surname << endl;

}

cout << "Search elements: " << dfinish\_time - dstart\_time << "mcs\n";

dstart\_time = (double)clock();

Person \* searchD = newTableC.find("bca");

dfinish\_time = (double)clock();

if (search)

{

cout << searchD->surname << endl;

}

cout << "Search elements with collision: " << dfinish\_time - dstart\_time << "mcs\n";

cout << "-----------------Add names to open hashtable----------------\n";

HashTableOpen newTableOpen, newTableOpenC;

count = 0;

fin.open("name1M.txt");

start\_time = clock();

while (!fin.eof() && count < 700) {

fin >> x;

newTableOpen.push(inp[count], x, (rand() % 99));

count++;

}

finish\_time = clock();

fin.close();

cout << "Add elements without collision: " << (finish\_time - start\_time) << "ms\n";

fin.open("name1M.txt");

count = 0;

start\_time = clock();

while (!fin.eof() && count < 700) {

fin >> x;

newTableOpenC.push(inpC[(rand() % 500)], x, (rand() % 99));

count++;

}

finish\_time = clock();

fin.close();

cout << "Add elements with collision: " << (finish\_time - start\_time) << "ms\n";

cout << "-----------------------Find element----------------------\n";

dstart\_time = (double)clock();

PersonOpen \* searchOpen = newTableOpen.find("cba");

dfinish\_time = (double)clock();

if (search)

{

cout << searchOpen->surname << endl;

}

cout << "Search elements: " << dfinish\_time - dstart\_time << "mcs\n";

dstart\_time = (double)clock();

PersonOpen \* searchOpenD = newTableOpenC.find("cba");

dfinish\_time = (double)clock();

if (search)

{

cout << searchOpenD->surname << endl;

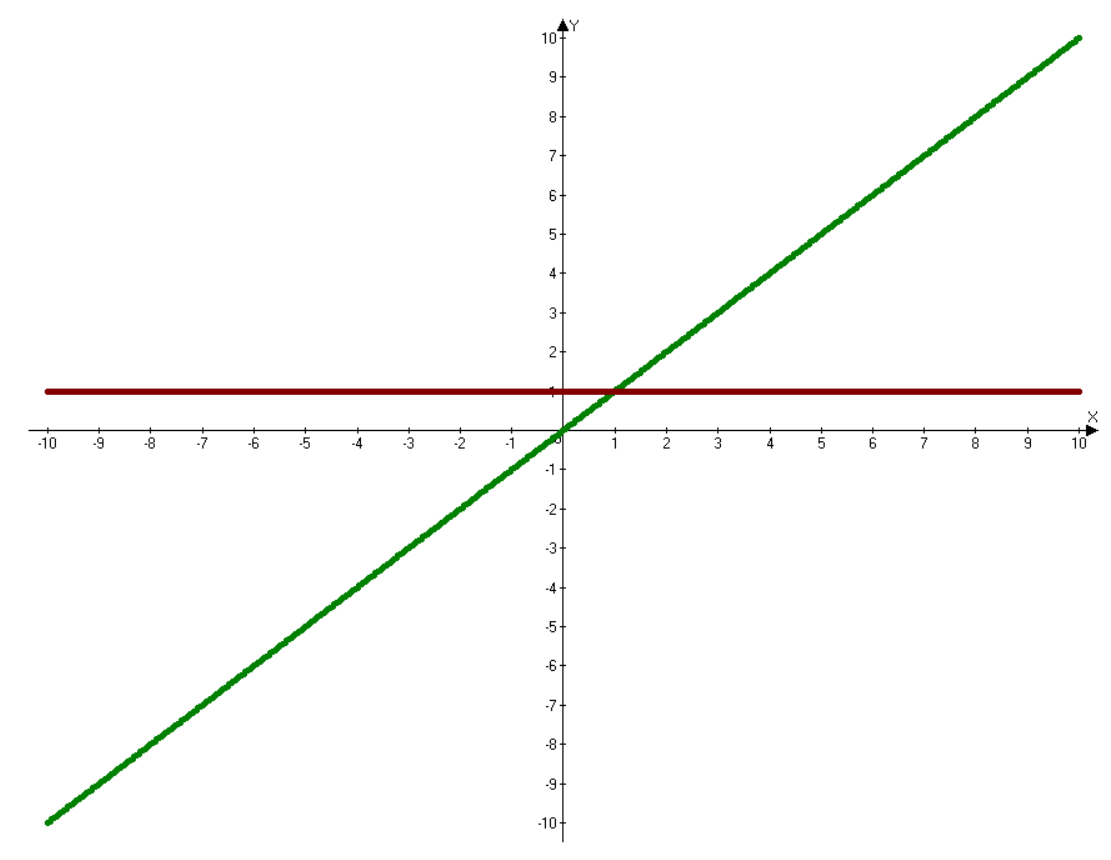
}

cout << "Search elements: " << dfinish\_time - dstart\_time << "mcs\n";

return 0;

}

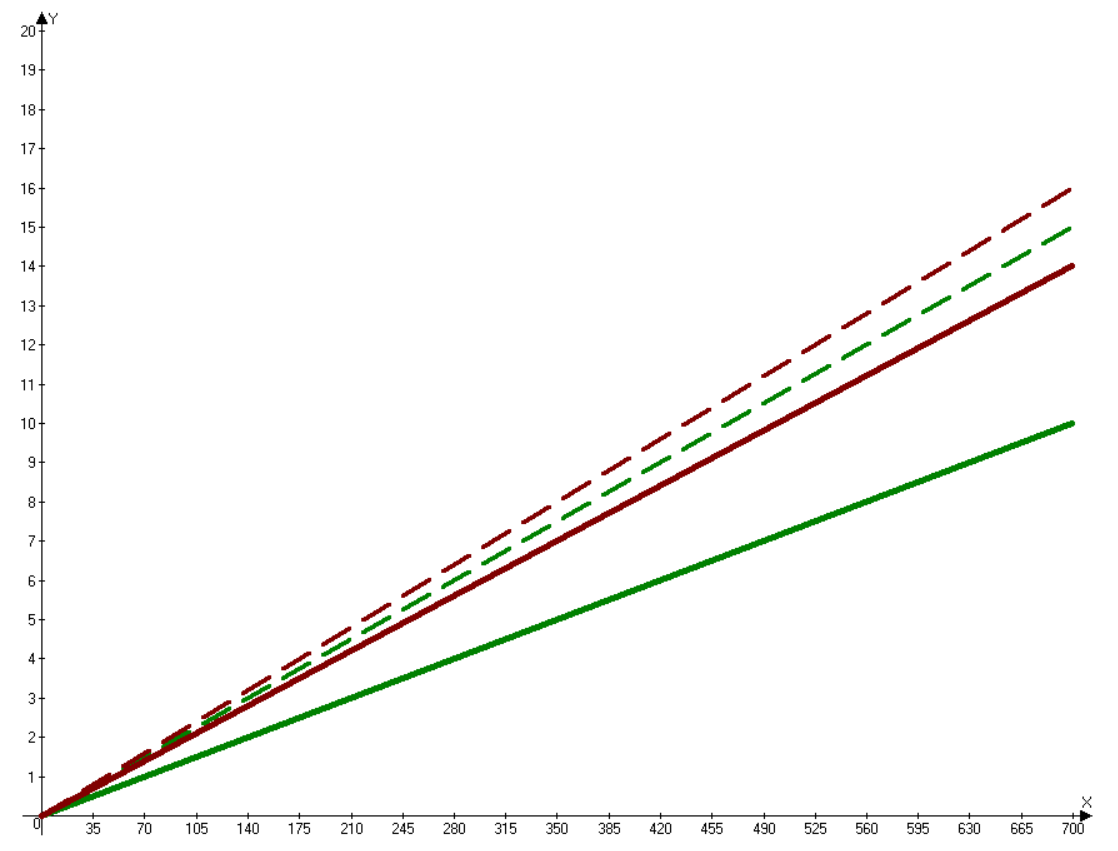
3. Графіки та час виконання



Графік 1,2

Червоний - складність  закритого та відкритого хешування O(1); зелений- складність  закритого та відкритого хешування O(n);

Час виконання:



Графік 3

Зелений (закрите хешування), червоний (відкрите хешування): суцільна - час додавання до хеш-таблиці (за умови, що колізія не відбудеться); штрих – час додавання до хеш-таблиці (за умови що колізія відбудеться)

Пошук відбувається менш ніж за мілісекунду у будь якому випадку.

