Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Стратегічного управління"

|  |  |
| --- | --- |
|  |  |

**Звіт до лабораторної роботи №6**

з дисципліни "Алгоритми на структури даних"

Виконав: студент групи КН-321А

Бородай Д. А.

Перевірив: старший викладач

Мошко Є.О.м

Харків 2022

**Тема лабораторної роботи:** Хеш – таблиці та робота з ними

**Мета:** Вивчити роботу алгоритмів: прямої адресації, хеш-таблиці і відкритої адресації. Реалізувати перелічені вище алгоритми і провести їх порівняльний аналіз.

**Порядок виконання роботи**:

1. Написати алгоритми основних операцій (додавання, пошук і видалення) в Псевдокод для прямої адресації (метод ланцюжка), хеш-таблиці і відкритої адресації.
2. Реалізувати основні операції.
3. Провести аналіз цих алгоритмів і зробити висновок по їх застосовності.

**Завдання**

1. Нехай розмір хеш-таблиці дорівнює m = 20, а хеш-функція має вигляд: h(k) = [m(kA mod 1)], где А = ( 5 - l)/2. В які позиції потраплять ключі (див. таблицю, стовпець 3)?

2. Як буде виглядати хеш-таблиця з ланцюжками після того, як в неї послідовно помістили елементи з ключами (див. таблицю, стовпець 2) (в зазначеному порядку)? Число позицій в таблиці дорівнює 9, хеш-функція має вигляд h(k) = k mod 9.

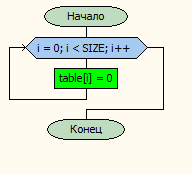
3. Виконайте додавання ключів (в зазначеному порядку, див. таблицю стовпець 2) в хеш-таблицю з відкритою адресацією розміру m = 11. Для обчислення послідовності проб використовується лінійний метод з h'(k) = k mod m. Виконайте те ж завдання, якщо використовується квадратичний метод з тієї ж h', c1 = 1, c2 = 3, а також для подвійного хешування з h1 = h' и h2(k) = 1 + (k mod (m - 1)).

Використовувати готові реалізації структур даних (наприклад, STL) заборонено, але можна використати реалізацію рядків (наприклад, std::string у C++).

## ОПИС АЛГОРИТМІВ ХЕШУВАННЯ

* 1. Пряма адресація

1.1.1 Створення хеш – таблиці



Блок – схема 1.1.1 – Створення хеш таблиці

void HashTable::Add(int key){

mas[HashFoo(key)] = key;

}

int HashTable::HashFoo(int key) {

double rez = (key \* ((sqrt(5) - 1) / 2));

double fraction, integer;

fraction = modf(rez, &integer);

int h = (int)m \* fraction;

return h;

}

Повний код програми наведено у додатку А.

Приклад роботи алгоритму:

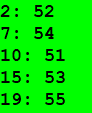
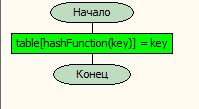


Рисунок 1.1.1 – Приклад хеш – таблиці ключів

1.1.2 Додавання нового елементу до таблиці



Блок – схема 1.1.2 – Алгоритм додавання нового елементу

Приклад коду:

void HashTable::Add(int key)

{

mas[HashFoo(key)] = key;

}

Повний код програми наведено у додатку А.

Приклад роботу алгоритму:

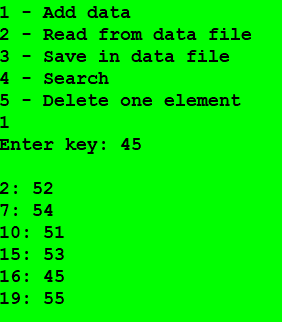
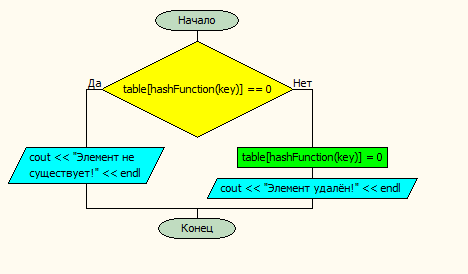


Рисунок 1.1.2 – Робота алгоритму додавання

1.1.3 Видалення одного із існуючих елементів



Блок–схема 1.1.3 – Видалення елементу таблиці

Приклад реалізації:

void HashTable::Delete(int key)

{

if (mas[HashFoo(key)] == 0)

cout << endl << "This element does not exist in hash table \n";

else { cout << "\nElement has been deleted \n"; mas[HashFoo(key)]=0;}

}

Повний код програми наведено у додатку А.

Приклад роботи алгоритму:

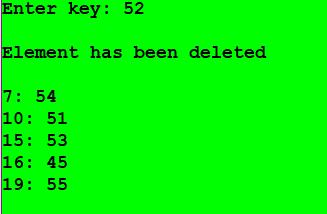
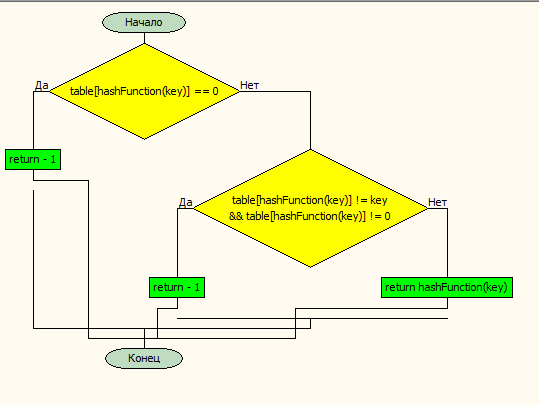


Рисунок 1.1.3 – Робота видалення елементу з таблиці

1.1.4 Пошук елементу



Блок–схема 1.1.4 – Алгоритм пошуку

Приклад коду алгоритма:

int HashTable::Search(int key) {

if (mas[HashFoo(key)] == 0){

cout << "\nThis element does not exist in hash table\n";

return -1;

} else return HashFoo(key);

}

Повник код програми наведено у додатку А.

Приклад роботи алгоритму:

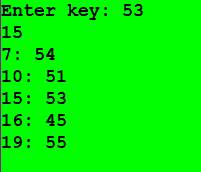
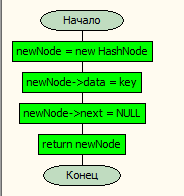


Рисунок 1.1.5 – Робота алгоритму пошуку

1.2 Створення хеш – таблиці ланцюжком

1.2.1 Створення хеш – таблиці ланцюжком



Блок – схема 1.2.1 – Створення хеш таблиці

HashTableWithChains::Node\* HashTableWithChains::Create(int key)

{

Node\* temp = new Node;

temp->data = key;

temp->next = nullptr;

return temp;

}

Повний код програми наведено у додатку А.

Приклад роботи алгоритму:

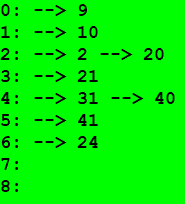
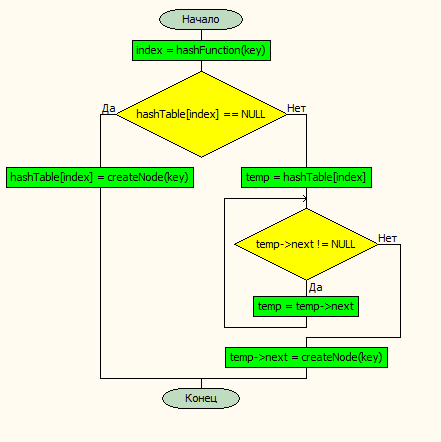


Рисунок 6.2.1 – Приклад хеш – таблиці ланцюжком

1.1.2 Додавання нового елементу до таблиці



Блок – схема 1.2.2 – Алгоритм додавання нового елементу

Приклад коду:

void HashTableWithChains::Add(int key)

{

int hash = HashFoo(key);

if (node[hash] == nullptr)

node[hash] = Create(key);

else {

Node\* temp = node[hash];

while (temp->next != nullptr)

temp = temp->next;

temp->next = Create(key);

}

}

int HashTableWithChains::HashFoo(int key){ return key % 9; }

Повний код програми наведено у додатку А.

Приклад роботу алгоритму:

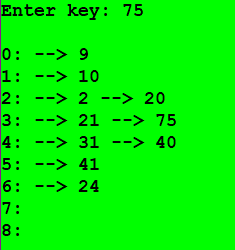
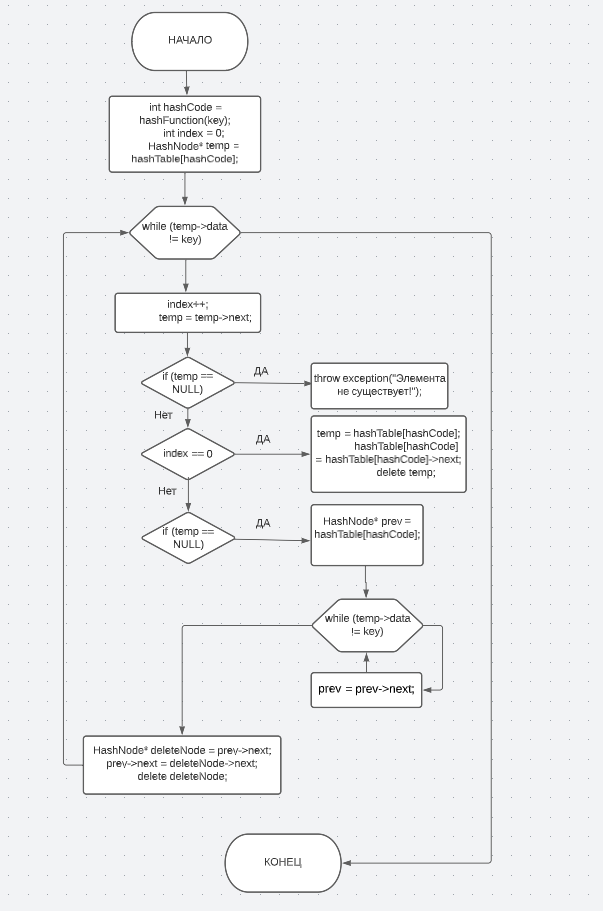


Рисунок 1.2.7 – Робота алгоритму додавання

1.1.3 Видалення одного із існуючих елементів



Блок–схема 1.2.3 – Видалення елементу таблиці

Приклад реалізації:

int HashTableWithChains::Delete(int key)

{

int hash = HashFoo(key);

int index = 0;

Node\* temp = node[hash];

while (temp->data != key)

{

index++;

temp = temp->next;

if (temp == nullptr)

return -1;

}

if (index == 0)

{

temp = node[hash];

node[hash] = node[hash]->next;

delete temp;

}

else {

Node\* previous = node[hash];

for (int i = 0; i < index - 1; i++)

{

previous = previous->next;

}

Node\* del\_node = previous->next;

previous->next = del\_node->next;

delete del\_node;

}

return 1;

}

Повний код програми наведено у додатку А.

Приклад роботи алгоритму:

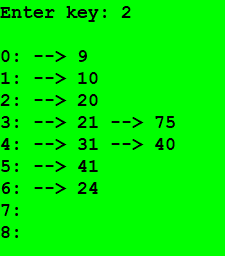
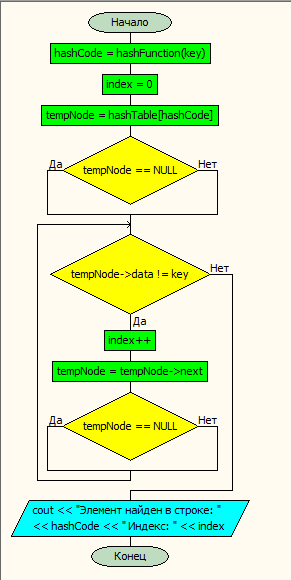


Рисунок 1.2.8 – Робота видалення елементу з таблиці

1.1.4 Пошук елементу



Блок–схема 1.1.9 – Алгоритм пошуку

Приклад коду алгоритма:

void HashTableWithChains::Search(int key)

{

int hash = HashFoo(key);

int index = 0;

Node\* temp = node[hash];

while (temp->data != key)

{

index++;

temp = temp->next;

if (temp == nullptr) {

index = -1;

break;

}

}

if (index == -1)

cout << endl << "Element hasn`t been found" << endl;

else

cout << endl << "Element has been found in " << hash

<< " point " << index << endl;

}

Повник код програми наведено у додатку А.

Приклад роботи алгоритму:

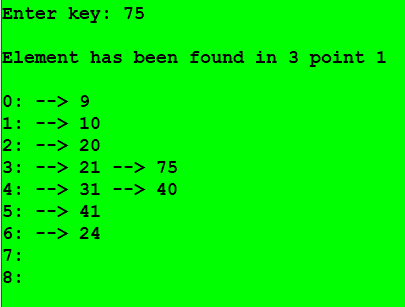
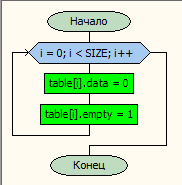


Рисунок 1.1.10 – Робота алгоритму пошуку

1.3 Відкрита адресація (лінійний метод, квадратичний та подвійного хешування)

1.3.1 Створння таблиці відкритої адресації



Блок – схема 1.3.1 – Створення таблиці

Приклад коду:

HashTableWithOpenAdressing::HashTableWithOpenAdressing()

{ for (size\_t i = 0; i < m2; i++) mas[i].data = 0; }

1.3.2 Лінійний метод

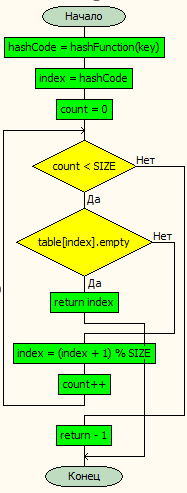


Рисунок 1.3.1 – Ліній метод

Приклад коду реалізаціїї таблиці методу:

void HashTableWithOpenAdressing::AddLinear(int key)

{

int poz = LinearProb(key);

mas[poz].data = key;

mas[poz].empty = false;

}

void HashTableWithOpenAdressing::DeleteLinear(int key)

{

int ind = SearchLinear(key);

if (ind == -1)

cout << "\nThis element does not exitst\n";

else

{

cout << "\nElement has been deleted\n";

mas[ind].data = 0;

mas[ind].empty = true;

}

}

int HashTableWithOpenAdressing::SearchLinear(int key)

{

int hash = HashFoo(key);

while (mas[hash].empty != true)

{

if (mas[hash].data == key)

{

return hash;

}

hash = (hash + 1) % m2;

}

return -1;

}

1.3.3 Квадратичний метод

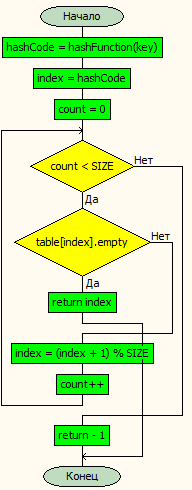


Рисунок 1.3.2 – Квадратичний метод

Приклад коду реалізаціїї таблиці методу:

void HashTableWithOpenAdressing::AddQuadratic(int key)

{

int poz = Quadraticprob(key);

mas[poz].data = key;

mas[poz].empty = false;

}

void HashTableWithOpenAdressing::DeleteQuadratic(int key)

{

int ind = SearchQuadratic(key);

if (ind == -1)

cout << "\nThis element does not exitst\n";

else

{

cout << "\nElement has been deleted\n";

mas[ind].data = 0;

mas[ind].empty = true;

}

}

int HashTableWithOpenAdressing::SearchQuadratic(int key)

{

int c1 = 1;

int c2 = 3;

int hash = HashFoo(key);

int h\_s = 0;

while (true)

{

for (size\_t i = 0; i < m2; i++)

{

h\_s = (hash + c1 \* i + c2 \* i \* i) % m2;

if (mas[h\_s].data == key)

{

return h\_s;

}

}

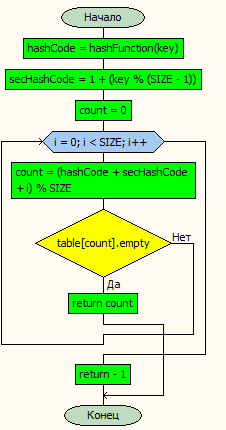
hash++;

}

return -1;

}

1.3.4 Метод подвійного хешування



Приклад коду реалізаціїї таблиці методу:

void HashTableWithOpenAdressing::AddDouble(int key)

{

int poz = DoubleProb(key);

mas[poz].data = key;

mas[poz].empty = false;

}

void HashTableWithOpenAdressing::DeleteDouble(int key)

{

int ind = SearchDouble(key);

if (ind == -1)

{

cout << "\nThis element does not exitst\n";

}

else

{

cout << "\nElement has been deleted\n";

mas[ind].data = 0;

mas[ind].empty = true;

}

}

int HashTableWithOpenAdressing::SearchDouble(int key)

{

int hash = HashFoo(key);

int hash2 = 1 + (key % (m2 - 1));

int kol = 0;

for (size\_t i = 0; i < m2; i++)

{

kol = (hash + hash2 \* i) % m2;

if (mas[kol].data == key)

return kol;

}

return -1;

}

## АНАЛІЗ СКЛАДНОСТІ АЛГОРИТМІВ

Аналіз хеш-таблиці з прямою адресацією

Так як даний вид хеш-таблиці передбачає те що в дану таблицю будуть записані лише такі дані котрі не будуть викликати колізію то тимчасова складність для функції додавання, пошуку, видалення у всіх випадках буде – .

Аналіз хеш-таблиці з ланцюжками

Для функцій додавання, пошуку та видалення складність виглядає наступним чином:

* Найгірший випадок – ;
* Найкращий випадок – ;
* Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

Аналіз хеш-таблиці з відкритою адресацією

Лінійне пробування

Для функцій додавання, пошуку та видалення складність виглядає наступним чином:

* Найгірший випадок – ;
* Найкращий випадок – ;

– Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

Квадратичне пробування

Для функцій додавання, пошуку та видалення складність виглядає наступним чином:

* Найгірший випадок – ;
* Найкращий випадок – ;

– Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

Подвійне хешування

Для функцій додавання, пошуку та видалення складність виглядає наступним чином:

* Найгірший випадок – ;
* Найкращий випадок – ;

– Середній випадок – , де n – кількість ключів котрі будуть вставлені в таблицю, m – розмір масиву таблиці;

# Висновок

Хеш-таблиця – це структура даних, що реалізує інтерфейс асоціативного масиву, тобто вона дозволяє зберігати пари виду "ключ-значення" і виконувати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення пари по ключу.

Хеш-таблиця з прямою адресацією – це хеш-таблиці, котрі використовують ін’єктивні хеш-функції і не потребуються у механізмі вирішення колізії. Дані хеш-таблиці є найшвидшими так як у будь-якій операції мають наймінімальнішу тимчасову складність, але використовуються лише у тому випадку коли точно відомо, що в даній таблиці будуть зберігатися дані котрі не будуть викликати колізій.

Хеш-таблиця з використанням методу ланцюжків – це така хеш-таблиця в котрій використовується технологія вирішення колізій, котра полягає в тому, що елементи сукупності з рівними хеш-значеннями зв’язуються в ланцюжок-список. Головною перевагою такої хеш-таблиці є те що вона ніколи не переповниться так як список це динамічна структура даних. Головним недоліком такої таблиці є те що при малому розмірі масиву таблиці, постане проблема в довжині ланцюжків із-за чого буде доволі довгий пошук необхідного елементу. Але при великому розмірі масиву таблиці та при мінімальних довжинах ланцюгів можливо добитися мінімальної тимчасової складності як і в хеш-таблиці з прямою адресацією.

Хеш-таблиця з відкритою адресацією – це хеш-таблиця в якій використовується технологія вирішення колізій, котра передбачає зберігання записів в самій хеш-таблиці. Для вирішення колізій в даній хеш-таблиці використовується методи повторного хешування.

Лінійне пробування – елементи хеш-таблиці послідовно проглядаються з деяким фіксованим інтервалом k між елементами. Недоліком даного виду пробування є кластеризація.

Квадратичне пробування – це таке пробування коли інтервал між елементами з кожним кроком збільшується на константу. Завдяки нелінійності такої адресації зменшується число проб при більшій кількості ключів-синонімів. Але мінусом такого пробування є те що навіть відносно невелике число проб може швидко привести до виходу за адресний простір невеликої таблиці внаслідок квадратичної залежності адреси від номера спроби.

Подвійне хешування – засноване на нелінійній адресації, що досягається за рахунок підсумовування значень основної і додаткової хеш-функції. Завдяки такому підходу знижується ефект кластеризації.

Хеш-таблицю з відкритою адресацією доцільно використовувати коли немає можливості витрачати пам’ять на зберігання вказівників списку, або якщо немає можливості зберігати елементи з зовні таблиці. Мінусами такої таблиці є те що вона може закінчитися, а також при більшому заповненні збільшується і тимчасова складність.

# Додаток А

Код Source.cpp

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

#define m 20

#define m1 9

#define m2 11

class HashTable

{

private:

int mas[m];

int HashFoo(int key);

public:

HashTable();

void Add(int key);

void Delete(int key);

int Search(int key);

void Print();

void Read\_file(string nameFile);

void Write\_file(string nameFile);

int Empty();

};

HashTable::HashTable()

{

for (size\_t i = 0; i < m; i++)

{

mas[i] = 0;

}

}

int HashTable::Empty()

{

for (size\_t i = 0; i < m; i++)

{

if (mas[i] != 0)

{

return 1;

}

}

return 0;

}

void HashTable::Add(int key)

{

mas[HashFoo(key)] = key;

}

int HashTable::HashFoo(int key) {

double rez = (key \* ((sqrt(5) - 1) / 2));

double fraction, integer;

fraction = modf(rez, &integer);

int h = (int)m \* fraction;

return h;

}

void HashTable::Read\_file(string name)

{

ifstream ifst;

int value;

ifst.open(name);

if (!ifst.is\_open()) {

throw exception("File hasn`t been found");

}

while (ifst >> value)

Add(value);

ifst.close();

}

void HashTable::Print() {

for (int i = 0; i < m; i++)

{

if (mas[i] != 0)

{

cout << i << ": " << mas[i] << endl;

}

}

}

void HashTable::Write\_file(string name)

{

ofstream ofst;

ofst.open(name);

if (!ofst.is\_open()) {

throw exception("File hasn`t been found");

}

for (int i = 0; i < m; i++)

{

if (mas[i] != 0) {

ofst << i << ": " << mas[i] << endl;

}

}

ofst.close();

}

int HashTable::Search(int key)

{

if (mas[HashFoo(key)] == 0)

{

cout << endl << "This element does not exist in hash table" << endl;

return -1;

}

else

{

return HashFoo(key);

}

}

void HashTable::Delete(int key)

{

if (mas[HashFoo(key)] == 0)

{

cout << endl << "This element does not exist in hash table" << endl;

}

else

{

cout << endl << "Element has been deleted" << endl;

mas[HashFoo(key)] = 0;

}

}

class HashTableWithChains

{

private:

class Node

{

public:

int data;

Node\* next;

};

Node\* node[9];

int HashFoo(int key);

Node\* Create(int key);

public:

HashTableWithChains();

void Add(int key);

int Delete(int key);

void Search(int key);

void Print();

void Read\_file(string name);

void Write\_file(string name);

};

HashTableWithChains::HashTableWithChains()

{

for (size\_t i = 0; i < 9; i++)

{

node[i] = nullptr;

}

}

void HashTableWithChains::Add(int key)

{

int hash = HashFoo(key);

if (node[hash] == nullptr)

{

node[hash] = Create(key);

}

else {

Node\* temp = node[hash];

while (temp->next != nullptr)

{

temp = temp->next;

}

temp->next = Create(key);

}

}

int HashTableWithChains::HashFoo(int key)

{

return key % 9;

}

HashTableWithChains::Node\* HashTableWithChains::Create(int key)

{

Node\* temp = new Node;

temp->data = key;

temp->next = nullptr;

return temp;

}

void HashTableWithChains::Print()

{

Node\* current = node[0];

for (int i = 0; i < 9; i++)

{

cout << i << ":";

while (current != nullptr)

{

cout << " --> " << current->data;

current = current->next;

}

cout << endl;

if (i == 8) {

break;

}

else {

current = node[i + 1];

}

}

}

void HashTableWithChains::Search(int key)

{

int hash = HashFoo(key);

int index = 0;

Node\* temp = node[hash];

while (temp->data != key)

{

index++;

temp = temp->next;

if (temp == nullptr) {

index = -1;

break;

}

}

if (index == -1)

{

cout << endl << "Element hasn`t been found" << endl;

}

else {

cout << endl << "Element has been found in " << hash

<< " point " << index << endl;

}

}

int HashTableWithChains::Delete(int key)

{

int hash = HashFoo(key);

int index = 0;

Node\* temp = node[hash];

while (temp->data != key)

{

index++;

temp = temp->next;

if (temp == nullptr)

return -1;

}

if (index == 0)

{

temp = node[hash];

node[hash] = node[hash]->next;

delete temp;

}

else {

Node\* previous = node[hash];

for (int i = 0; i < index - 1; i++)

{

previous = previous->next;

}

Node\* del\_node = previous->next;

previous->next = del\_node->next;

delete del\_node;

}

return 1;

}

void HashTableWithChains::Read\_file(string name)

{

ifstream ifst;

int value;

ifst.open(name);

if (!ifst.is\_open()) {

throw exception("File hasn`t been found");

}

while (ifst >> value)

Add(value);

ifst.close();

}

void HashTableWithChains::Write\_file(string name)

{

ofstream ofst;

ofst.open(name);

if (!ofst.is\_open()) {

throw exception("File hasn`t been found");

}

for (int i = 0; i < 9; i++)

{

if (node[i] != nullptr)

{

ofst << i << ": " << node[i] << endl;

}

}

ofst.close();

}

class HashTableWithOpenAdressing

{

private:

class Element

{

public:

int data;

bool empty;

};

Element mas[m2];

int HashFoo(int key);

int LinearProb(int key);

int Quadraticprob(int key);

int DoubleProb(int key);

public:

HashTableWithOpenAdressing();

void Clear();

int Empty();

void AddLinear(int key);

void AddQuadratic(int key);

void AddDouble(int key);

void DeleteLinear(int key);

int SearchLinear(int key);

void DeleteQuadratic(int key);

int SearchQuadratic(int key);

void DeleteDouble(int key);

int SearchDouble(int key);

void Print();

void Read\_file\_Linear(string name);

void Read\_file\_Quadratic(string name);

void Read\_file\_Double(string name);

void Write\_file(string name);

};

HashTableWithOpenAdressing::HashTableWithOpenAdressing()

{

for (size\_t i = 0; i < m2; i++)

{

mas[i].data = 0;

}

}

void HashTableWithOpenAdressing::Clear()

{

}

int HashTableWithOpenAdressing::Empty()

{

return 1;

}

void HashTableWithOpenAdressing::AddLinear(int key)

{

int poz = LinearProb(key);

mas[poz].data = key;

mas[poz].empty = false;

}

void HashTableWithOpenAdressing::AddQuadratic(int key)

{

int poz = Quadraticprob(key);

mas[poz].data = key;

mas[poz].empty = false;

}

void HashTableWithOpenAdressing::AddDouble(int key)

{

int poz = DoubleProb(key);

mas[poz].data = key;

mas[poz].empty = false;

}

void HashTableWithOpenAdressing::DeleteLinear(int key)

{

int ind = SearchLinear(key);

if (ind == -1)

{

cout << "\nThis element does not exitst\n";

}

else

{

cout << "\nElement has been deleted\n";

mas[ind].data = 0;

mas[ind].empty = true;

}

}

int HashTableWithOpenAdressing::SearchLinear(int key)

{

int hash = HashFoo(key);

while (mas[hash].empty != true)

{

if (mas[hash].data == key)

{

return hash;

}

hash = (hash + 1) % m2;

}

return -1;

}

void HashTableWithOpenAdressing::DeleteQuadratic(int key)

{

int ind = SearchQuadratic(key);

if (ind == -1)

{

cout << "\nThis element does not exitst\n";

}

else

{

cout << "\nElement has been deleted\n";

mas[ind].data = 0;

mas[ind].empty = true;

}

}

int HashTableWithOpenAdressing::SearchQuadratic(int key)

{

int c1 = 1;

int c2 = 3;

int hash = HashFoo(key);

int h\_s = 0;

while (true)

{

for (size\_t i = 0; i < m2; i++)

{

h\_s = (hash + c1 \* i + c2 \* i \* i) % m2;

if (mas[h\_s].data == key)

{

return h\_s;

}

}

hash++;

}

return -1;

}

void HashTableWithOpenAdressing::DeleteDouble(int key)

{

int ind = SearchDouble(key);

if (ind == -1)

{

cout << "\nThis element does not exitst\n";

}

else

{

cout << "\nElement has been deleted\n";

mas[ind].data = 0;

mas[ind].empty = true;

}

}

int HashTableWithOpenAdressing::SearchDouble(int key)

{

int hash = HashFoo(key);

int hash2 = 1 + (key % (m2 - 1));

int kol = 0;

for (size\_t i = 0; i < m2; i++)

{

kol = (hash + hash2 \* i) % m2;

if (mas[kol].data == key)

return kol;

}

return -1;

}

void HashTableWithOpenAdressing::Print()

{

for (int i = 0; i < m2; i++)

{

if (mas[i].data != 0)

{

cout << i << ": " << mas[i].data << endl;

}

}

}

void HashTableWithOpenAdressing::Read\_file\_Linear(string name)

{

ifstream ifst;

int value;

ifst.open(name);

if (!ifst.is\_open()) {

throw exception("File hasn`t been found");

}

while (ifst >> value)

AddLinear(value);

ifst.close();

}

void HashTableWithOpenAdressing::Read\_file\_Quadratic(string name)

{

ifstream ifst;

int value;

ifst.open(name);

if (!ifst.is\_open()) {

throw exception("File hasn`t been found");

}

while (ifst >> value)

AddQuadratic(value);

ifst.close();

}

void HashTableWithOpenAdressing::Read\_file\_Double(string name)

{

ifstream ifst;

int value;

ifst.open(name);

if (!ifst.is\_open()) {

throw exception("File hasn`t been found");

}

while (ifst >> value)

AddDouble(value);

ifst.close();

}

void HashTableWithOpenAdressing::Write\_file(string name)

{

ofstream ofst;

ofst.open(name);

if (!ofst.is\_open()) {

throw exception("File hasn`t been found");

}

for (int i = 0; i < m; i++)

{

ofstream ofst;

ofst.open(name);

if (!ofst.is\_open()) {

throw exception("File hasn`t been found");

}

for (int i = 0; i < 9; i++)

{

if (mas[i].data != 0)

{

ofst << i << ": " << mas[i].data << endl;

}

}

ofst.close();

}

ofst.close();

}

int HashTableWithOpenAdressing::HashFoo(int key)

{

return key % 11;

}

int HashTableWithOpenAdressing::LinearProb(int key)

{

int hash = HashFoo(key);

int i = hash;

int kol = 0;

while (kol < m2)

{

if (mas[i].empty)

return i;

i = (i + 1) % m2;

kol++;

}

return -1;

}

int HashTableWithOpenAdressing::Quadraticprob(int key)

{

int c1 = 1;

int c2 = 3;

int hash = HashFoo(key);

int h\_s = 0;

bool flag = false;

while (flag == false)

{

for (size\_t i = 0; i < m2; i++)

{

h\_s = (hash + c1 \* i + c2 \* i \* i) % m2;

if (mas[h\_s].empty)

{

return h\_s;

}

}

hash++;

}

return -1;

}

int HashTableWithOpenAdressing::DoubleProb(int key)

{

int hash = HashFoo(key);

int hash2 = 1 + (key % (m2 - 1));

int kol = 0;

for (size\_t i = 0; i < m2; i++)

{

kol = (hash + hash2 \* i) % m2;

if (mas[kol].empty)

return kol;

}

return -1;

}

int main()

{

srand(time(NULL));

int choose = -1;

while (choose != 0) {

cout << "\n1 - Hash table \n2 - Hash table with chains \n3 - Open addressing \n--> ";

cin >> choose;

if (choose == 1)

{

int choosing = -1;

HashTable a;

while (choosing != 0)

{

if (a.Empty() == 0)

{

cout << "\nEmpty";

}

else

{

cout << endl;

a.Print();

}

cout << endl << endl << endl;

string name1 = "al6\_1.txt";

int key;

cout << "1 - Add data" << endl;

cout << "2 - Read from data file" << endl;

cout << "3 - Save in data file" << endl;

cout << "4 - Search" << endl;

cout << "5 - Delete one element" << endl;

cout << "--> ";

cin >> choosing;

switch (choosing)

{

case 1:

cout << "Enter key: ";

cin >> key;

a.Add(key);

break;

case 2:

a.Read\_file(name1);

break;

case 3:

a.Write\_file(name1);

break;

case 4:

cout << "Enter key: ";

cin >> key;

cout << a.Search(key);

break;

case 5:

cout << "Enter key: ";

cin >> key;

a.Delete(key);

break;

}

}

}

else if (choose == 2)

{

int choosing = -1;

int p = 0;

HashTableWithChains a1;

while (choosing != 0)

{

if (p == 0)

{

cout << "\nEmpty";

}

else

{

cout << endl;

a1.Print();

}

cout << endl << endl << endl;

string name2 = "al6\_2.txt";

int key;

cout << "1 - Add data" << endl;

cout << "2 - Read from data file" << endl;

cout << "3 - Save in data file" << endl;

cout << "4 - Search" << endl;

cout << "5 - Delete one element" << endl;

cout << "--> ";

cin >> choosing;

switch (choosing)

{

case 1:

cout << "Enter key: ";

cin >> key;

a1.Add(key);

p = 1;

break;

case 2:

a1.Read\_file(name2);

p = 1;

break;

case 3:

a1.Write\_file(name2);

break;

case 4:

cout << "Enter key: ";

cin >> key;

a1.Search(key);

break;

case 5:

cout << "Enter key: ";

cin >> key;

a1.Delete(key);

break;

}

}

}

else

{

M:

HashTableWithOpenAdressing a3;

string name2 = "al6\_3.txt";

int choose3 = -1;

while (choose3 != 0) {

cout << "With what type of probing will the hash table be built: \n";

cout << "1 - Liner\n";

cout << "2 - Quadratic\n";

cout << "3 - Double hasing\n";

cout << "--> ";

cin >> choose3;

int p = 0;

if (choose3 == 1)

{

while (true)

{

if (p == 0)

{

cout << "\nEmpty";

}

else

{

cout << endl;

a3.Print();

}

cout << endl << endl << endl;

int choosing;

int key;

cout << "1 - Add data" << endl;

cout << "2 - Read from data file" << endl;

cout << "3 - Save in data file" << endl;

cout << "4 - Search" << endl;

cout << "5 - Delete one element" << endl;

cout << "6 - Change probing" << endl;

cout << "--> ";

cin >> choosing;

switch (choosing)

{

case 1:

cout << "Enter key: ";

cin >> key;

a3.AddLinear(key);

p = 1;

break;

case 2:

a3.Read\_file\_Linear(name2);

p = 1;

break;

case 3:

a3.Write\_file(name2);

break;

case 4:

cout << "Enter key: ";

cin >> key;

a3.SearchLinear(key);

break;

case 5:

cout << "Enter key: ";

cin >> key;

a3.DeleteLinear(key);

break;

case 6:

goto M;

break;

}

}

}

else if (choose3 == 2)

{

int choosing = -1;

while (choosing != 0)

{

if (p == 0)

{

cout << "\nEmpty";

}

else

{

cout << endl;

a3.Print();

}

cout << endl << endl << endl;

int key;

cout << "1 - Add data" << endl;

cout << "2 - Read from data file" << endl;

cout << "3 - Save in data file" << endl;

cout << "4 - Search" << endl;

cout << "5 - Delete one element" << endl;

cout << "6 - Change probing" << endl;

cout << "--> ";

cin >> choosing;

switch (choosing)

{

case 1:

cout << "Enter key: ";

cin >> key;

a3.AddQuadratic(key);

p = 1;

break;

case 2:

a3.Read\_file\_Quadratic(name2);

p = 1;

break;

case 3:

a3.Write\_file(name2);

break;

case 4:

cout << "Enter key: ";

cin >> key;

a3.SearchQuadratic(key);

break;

case 5:

cout << "Enter key: ";

cin >> key;

a3.DeleteQuadratic(key);

break;

case 6:

goto M;

break;

}

}

}

else if (choose3 == 3)

{

int choosing = -1;

while (choosing != 0)

{

if (p == 0)

{

cout << "\nEmpty";

}

else

{

cout << endl;

a3.Print();

}

cout << endl << endl << endl;

int key;

cout << "1 - Add data" << endl;

cout << "2 - Read from data file" << endl;

cout << "3 - Save in data file" << endl;

cout << "4 - Search" << endl;

cout << "5 - Delete one element" << endl;

cout << "6 - Change probing" << endl;

cout << "--> ";

cin >> choosing;

switch (choosing)

{

case 1:

cout << "Enter key: ";

cin >> key;

a3.AddDouble(key);

p = 1;

break;

case 2:

a3.Read\_file\_Double(name2);

p = 1;

break;

case 3:

a3.Write\_file(name2);

break;

case 4:

cout << "Enter key: ";

cin >> key;

a3.SearchDouble(key);

break;

case 5:

cout << "Enter key: ";

cin >> key;

a3.DeleteDouble(key);

break;

case 6:

goto M;

break;

}

}

}

}

}

}

}