Министерство общего и профессионального

образования Российской Федерации

Пермский государственный технический университет

Лабораторная работа №11

# "Поиск данных с помощью хэш-таблиц"

Вариант №12

Выполнила студентка группы РИС 23-3б:

Федорова О.И.

Проверила

Доцент кафедры ИТАС

Полякова О.А.

Пермь 2024

Постановка задачи:

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Предусмотреть сохранение массива в файл и загрузку массива из файла.
3. Предусмотреть возможность добавления и удаления элементов из массива (файла).
4. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу.
5. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | ФИО, №паспорта, №телефона | ФИО | H(k)= [M (kAmod1)], 0<A<1, mod1 – получение дробной части, [] – получение целой части | Метод цепочек |

2. Код программы:

#include <iostream>

#include <fstream>

#include <string>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <ctime>

using namespace std;

const int m = 1; // количество ячеек таблицы

const double a = M\_PI\_4;

int collisionscnt = 0;

struct node { // структура для узла

string key = ""; // ключ элемента

string value = ""; // значение элемента

node\* next = nullptr; // указатель на следующий элемент

node\* prev = nullptr; // указатель на предыдущий элемент

};

struct hashtable { // структура для hash-таблицы

node\* table[m];

hashtable() {

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

table[i] = nullptr; // заполнение таблицы нулями

}

}

};

double mod1(double k){ // функция, которая возвращает дробную часть

int intpart = static\_cast<int>(k);

return k - intpart;

}

double gethash(double k) { // hash-функция для типа int

return static\_cast<int>(m\* mod1(k \* a));

}

double gethash(string line) { // hash-функция ля типа string

int n = 0;

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

n += static\_cast<int>(pow(line[i], 2) \* M\_2\_SQRTPI + abs(line[i]) \* M\_SQRT1\_2);

}

return gethash(abs(n));

}

bool add(hashtable& table, string key, string elem) { // добавление нового элемента в список

node\* newnode = new node; // создание нового элемента

newnode->key = key; // присваивание ключа элементу

newnode->value = elem; // присваивание значения элементу

newnode->next = nullptr; // присваивание указателя элементу

newnode->prev = nullptr; // присваивание указателя элементу

int hash = gethash(key); // индекс ключа

if (table.table[hash] == nullptr) { // если ячейка свободна

table.table[hash] = newnode; // помещение элемента в ячейку

return true;

}

else {

node\* cur = table.table[hash];

while (cur->next != nullptr) { // поиск свободной ячейки

cur = cur->next;

}

cur->next = newnode; // присваивание элементу ячейку

newnode->prev = cur; // связь узлов

collisionscnt++; // увеличение счетчика

return true;

}

}

bool remove(hashtable& table, string key) { // удаление элемента по ключу

int hash = gethash(key); // индекс ключа

node\* cur = table.table[hash];

while (cur != nullptr) {

if (cur->key == key) { // если заданный ключ найден

if (cur->prev != nullptr) // если элемент не первый в списке

cur->prev->next = cur->prev; // связь узлов

else

table.table[hash] = cur->next; // связь узлов

if (cur->next != nullptr) // если элемент не последний в списке

cur->next->prev = cur->next; // связь узлов

delete cur; // удаление элемента

return true;

}

cur = cur->next; // переход к следующему элементу списка

}

return false;

}

node\* get(hashtable& table, string key) { // получение элемента по ключу

int hash = gethash(key); // индекс ключа

node\* cur = table.table[hash];

while (cur != nullptr) {

if (cur->key == key) // если заданный ключ найден

return cur;

cur = cur->next; // переход к следующему элементу списка

}

return nullptr;

}

void print(hashtable& table) { // вывод таблицы

for (int i = 0; i < m; i++) { // прохождение по таблице

node\* cur = table.table[i];

while (cur != nullptr) { // прохождение по списку

cout << "[" << cur->key << ": " << cur->value << "]\n";

cur = cur->next; // переход к следующему элементу списка

}

}

cout << endl;

}

string surnames[] = { "Иванов", "Смирнов", "Орлов", "Сидоров", "Лебедев",

"Морозов", "Новиков", "Зайцев", "Волков", "Соловьев" };

string names[] = { "Олег", "Леонид", "Сергей", "Андрей", "Алексей",

"Павел", "Михаил", "Максим", "Илья", "Владислав" };

string patronymics[] = { "Иванович", "Леонидович", "Сергеевич", "Андреевич", "Алексеевич",

"Павлович", "Михайлович", "Максимович", "Ильич", "Владиславич" };

string generatefullname() { // создание случайного ФИО

return surnames[rand() % 10] + " " + names[rand() % 10] + " " + patronymics[rand() % 10];

}

string correct(int n, int len) {

string str = to\_string(n); // преобразование значения в строку

while (str.size() < len) {

str = "0" + str; // добавление символов

}

while (str.size() > len) {

str.erase(0, 1); // удаление ненужных символов

}

return str;

}

string generatephonenumber() { // создание случайного дня рождения

return "+7 " + correct(rand() % 10, 3) + " " + correct(rand() % 10, 3) + "-" + correct(rand() % 10, 2) + "-" + correct(rand() % 10, 2);

}

string generatepasswordnumber() { // создание случайного номера паспорта

return correct(rand() % 10000, 4) + " " + correct(rand() % 1000000 \* 100 + rand(), 6);

}

void save(hashtable& table, ofstream& file) { // функция для сохранения таблицы в файл

for (int i = 0; i < m; i++) { // прохождение по таблице

node\* cur = table.table[i];

while (cur != nullptr) { // прохождение по списку

file << "[" << cur->key << "\n";

file << cur->value << "]\n";

cur = cur->next; // переход к следующему элементу списка

}

}

cout << endl;

}

void clean\_table(hashtable& table) { // функция для очищения таблицы

for (int i = 0; i < m; i++) { // прохождение по таблице

node\* cur = table.table[i];

while (cur != nullptr) { // прохождение по списку

node\* temp = cur;

cur = cur->next; // переход к следующему элементу списка

delete temp; // удаление элемента

}

table.table[i] = nullptr;

}

}

void recovery(hashtable& table, ifstream& file) { // функция для восстановления данных из файла в таблицу

string key, value;

getline(file, key); // считывание данных

if (key != "") {

getline(file, value); // считывание данных

add(table, key, value); // добавление данных в таблицу

while (getline(file, key)) {

getline(file, value); // считывание данных

add(table, key, value); // добавление данных в таблицу

}

}

}

int main() {

system("chcp 1251");

srand(time(NULL));

hashtable table; // создание таблицы

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

string FIO = generatefullname(); // создание ключа

string human = FIO + " | " + generatephonenumber() + " | " + generatepasswordnumber(); // создание элемента таблицы

add(table, FIO, human); // добавление элемента в таблицу

}

cout << "Hash-таблица: " << endl;

print(table);

int index = rand() % m;

while (table.table[index] == nullptr) {

index = rand() % m;

}

node\* randomhuman = table.table[index];

string key\_remove = randomhuman->key; // создание ключа для удаления

cout << "Удаление по ключу: " << key\_remove << endl;

if (remove(table, key\_remove))

cout << "Элемент успешно удален." << endl;

else

cout << "Элемент не найден." << endl;

print(table); // вывод таблицы

index = rand() % m;

while (table.table[index] == nullptr) {

index = rand() % m;

}

randomhuman = table.table[index];

string get\_key = randomhuman->key; // создание ключа для получения элемента

cout << "Получение по ключу: " << get\_key << endl;

node\* node\_get = get(table, get\_key);

if (node\_get != nullptr)

cout << "Элемент успешно найден." << node\_get->value << endl;

else

cout << "Элемент не найден." << endl;

print(table);

cout << "Число коллизий: " << collisionscnt << endl;

ofstream file;

file.open("file.txt"); // доступ к файлу для считывания

save(table, file); // сохранение данных в файл

clean\_table(table); // очищение таблицы

file.close();

ifstream ffile;

ffile.open("file.txt");

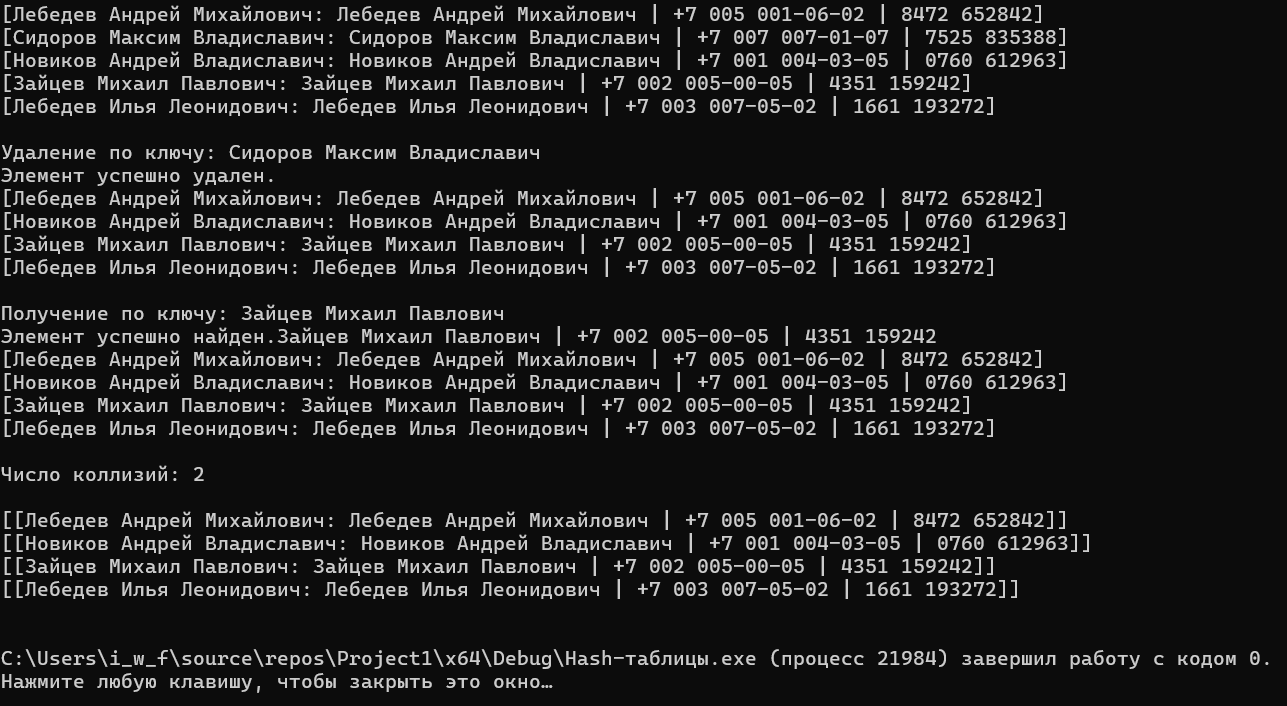
recovery(table, ffile); // восстановление данных в таблицу

print(table);

ffile.close();

return 0;

}

3. Результат работы программы:

**Метод цепочек**

1. Анализ задачи:
2. Создать структуру node, которая содержит структуру узла.
3. Создать структуру hashtable, которая содержит структуру hash-таблицы.
4. Создать функцию mod1, которая возвращает дробную часть числа.
5. Создать hash-функцию для типа int и string, которая задана.
6. Создать функцию добавления элемента в свободную ячейку списка, который находится в таблице.
7. Создать функцию удаления элемента по ключу. Функция получает индекс ключа в таблице и с помощью цикла выискивает нужный элемент в списке. При нахождении нужного элемента он удаляется, а оставшийся разрыв устраняется.
8. Создать функцию нахождения элемента по ключу. Функция получает индекс ключа в таблице и с помощью цикла выискивает нужный элемент в списке. При нахождении нужного элемента функция возвращает его.
9. Создать функцию для вывода таблицы.
10. Создать три функции для генерации случайных ФИО, номера телефона и номера паспорта.
11. В функции main первым делом через цикл генерируются элементы и добавляются в таблицу.
12. Далее рандомным образом выбирается ключ для удаления и вызывается функция удаления элемента. Программа выводит уведомление, прошло ли все успешно, а затем новую таблицу.
13. Снова рандомным образом выбирается ключ для получения элемента и вызывается функция получения элемента. Программа выводит уведомление, прошло ли все успешно, а затем полученный элемент.

**Метод линейного пробирования**

1. Анализ задачи:

1) Создать структуру, которая используется для представления узла в hash-таблице. Каждый узел содержит ключ и значение.  
  
2) Создать структуру, которая используется для представления самой hash-таблицы. Она содержит массив указателей на узлы.  
  
3) Создать функцию, которая используется для получения дробной части числа. Она возвращает число, вычисленное как разность между исходным числом и его целой частью.  
  
4) Создать функцию, которая используется для вычисления hash-значения для чисел типа int. Она возвращает результат умножения числа на m и вычитания его целой части.  
  
5) Создать функцию, которая используется для вычисления hash-значения для строк. Она проходит по всем символам строки, вычисляет их квадраты и складывает их. Затем она вычисляет абсолютное значение полученного числа и вызывает функцию gethash для этого числа.

1. Создать функцию, которая используется для добавления нового элемента в hash-таблицу. Она создает новый узел, присваивает ему ключ и значение, вычисляет hash-значение ключа и пытается поместить узел в соответствующую ячейку. Если ячейка свободна, узел помещается в нее. Если ячейка занята, функция ищет следующую свободную ячейку, увеличивая счетчик cnt. Если свободная ячейка найдена, узел помещается в нее. Если свободной ячейки не найдено, функция возвращает false, что означает неудачное добавление.  
     
   7) Создать функцию, которая используется для удаления элемента из hash-таблицы по ключу. Она вычисляет hash-значение ключа и пытается найти узел с этим ключом. Если узел найден, он удаляется из списка, и функция возвращает true. Если узел не найден, функция возвращает false, что означает неудачное удаление.

8) Создать функцию, которая  используется для получения элемента из hash-таблицы по ключу. Она вычисляет hash-значение ключа и пытается найти узел с этим ключом. Если узел найден, функция возвращает указатель на этот узел. Если узел не найден, функция возвращает nullptr.  
9) Создать функцию, которая  используется для вывода содержимого hash-таблицы. Она проходит по всем ячейкам таблицы и выводит информацию о каждом узле, если он не равен nullptr.  
  
10) Массивы surnames, names и patronymics содержат фамилии, имена и отчества соответственно.  
  
11) Создать функцию, которая используется для генерации случайного полного имени. Она выбирает случайный элемент из каждого массива и объединяет их в строку.

12) Создать функцию, которая  используется для форматирования чисел в строку. Она преобразует число в строку, добавляет нули в начале, если длина строки меньше заданной, и удаляет ненужные символы, если длина строки больше заданной.  
13) Создать функцию, которая  используется для генерации случайного номера телефона. Она создает строку, состоящую из префикса "+7", трех случайных цифр, трех случайных цифр, дефиса и двух случайных цифр.  
14) Создать функцию, которая используется для генерации случайного номера паспорта. Она создает строку, состоящую из четырех случайных цифр, пробела и шести случайных цифр.  
  
15) Создать функцию, которая используется для сохранения содержимого hash-таблицы в файл. Она проходит по всем ячейкам таблицы и записывает ключи и значения в файл.

1. 16) Создать функцию, которая  используется для очистки hash-таблицы. Она проходит по всем ячейкам таблицы и удаляет каждый узел, устанавливая его значение в nullptr.  
     
   17) Создать функцию, которая используется для восстановления данных из файла в hash-таблицу. Она считывает строки из файла, разделенные на ключи и значения, и добавляет их в таблицу с помощью функции add.  
     
   18) В функции main первым делом через цикл генерируются элементы и добавляются в таблицу.
2. Далее рандомным образом выбирается ключ для удаления и вызывается функция удаления элемента. Программа выводит уведомление, прошло ли все успешно, а затем новую таблицу.
3. Снова рандомным образом выбирается ключ для получения элемента и вызывается функция получения элемента. Программа выводит уведомление, прошло ли все успешно, а затем полученный элемент.
4. Код программы:

#include <iostream>

#include <fstream>

#include <string>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <ctime>

using namespace std;

const int m = 3; // количество ячеек таблицы

const double a = M\_PI\_4;

int collisionscnt = 0;

struct node { // структура для узла

string key = ""; // ключ элемента

string value = ""; // значение элемента

};

struct hashtable { // структура для hash-таблицы

node\* table[m];

hashtable() {

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

table[i] = nullptr; // заполнение таблицы нулями

}

}

};

double mod1(double k) { // функция, которая возвращает дробную часть

int intpart = static\_cast<int>(k);

return k - intpart;

}

double gethash(double k) { // hash-функция для типа int

return static\_cast<int>(m \* mod1(k \* a));

}

double gethash(string line) { // hash-функция для типа string

int n = 0;

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

n += static\_cast<int>(pow(line[i], 2) \* M\_2\_SQRTPI + abs(line[i]) \* M\_SQRT1\_2);

}

return gethash(abs(n));

}

bool add(hashtable& table, string key, string elem) { // добавление нового элемента в список

node\* newnode = new node; // создание нового элемента

newnode->key = key; // присваивание ключа элементу

newnode->value = elem; // присваивание значения элементу

int hash = gethash(key); // индекс ключа

int cnt = 0;

if (table.table[hash] == nullptr) { // если ячейка свободна

table.table[hash] = newnode; // помещение элемента в ячейку

return true;

}

cnt++;

while (table.table[(hash + cnt) % m] != nullptr && ((hash + cnt) % m) != hash) { // поиск свободной ячейки

cnt++;

collisionscnt++;

}

if (table.table[(hash + cnt) % m] == nullptr) { // если ячейка свободна

table.table[(hash + cnt) % m] = newnode; // помещение элемента в ячейку

return true;

}

else

return false;

}

bool remove(hashtable& table, string key) { // удаление элемента по ключу

int hash = gethash(key); // индекс ключа

int memory\_hash = hash;

if (table.table[hash] != nullptr) { // если заданный ключ найден

if (table.table[hash]->key == key) {// если элемент не первый в списке

table.table[hash] = nullptr; // связь узлов

return true;

}

hash++;

}

else

hash++;

while ((hash % m) != memory\_hash) {

if (table.table[hash % m] != nullptr) {

if (table.table[hash % m]->key == key) {

table.table[hash] = nullptr;

return true;

}

else

hash++;

}

else

hash++;

}

return false;

}

node\* get(hashtable& table, string key) { // получение элемента по ключу

int hash = gethash(key); // индекс ключа

int memory\_hash = hash;

if (table.table[hash] != nullptr) {

if (table.table[hash]->key == key)

return table.table[hash];

hash++;

}

else hash++;

while ((hash % m) != memory\_hash) {

if (table.table[hash % m] != nullptr) {

if (table.table[hash % m]->key == key)

return table.table[hash % m];

else hash++;

}

else hash++;

}

return nullptr;

}

void print(hashtable& table) { // вывод таблицы

bool flag = true;

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

if (table.table[i] != nullptr) {

flag = false;

cout << "[" << table.table[i]->key << ": " << table.table[i]->value << "] " << endl;

}

}

}

string surnames[] = { "Иванов", "Смирнов", "Орлов", "Сидоров", "Лебедев",

"Морозов", "Новиков", "Зайцев", "Волков", "Соловьев" };

string names[] = { "Олег", "Леонид", "Сергей", "Андрей", "Алексей",

"Павел", "Михаил", "Максим", "Илья", "Владислав" };

string patronymics[] = { "Иванович", "Леонидович", "Сергеевич", "Андреевич", "Алексеевич",

"Павлович", "Михайлович", "Максимович", "Ильич", "Владиславич" };

string generatefullname() { // создание случайного ФИО

return surnames[rand() % 10] + " " + names[rand() % 10] + " " + patronymics[rand() % 10];

}

string correct(int n, int len) {

string str = to\_string(n); // преобразование значения в строку

while (str.size() < len) {

str = "0" + str; // добавление символов

}

while (str.size() > len) {

str.erase(0, 1); // удаление ненужных символов

}

return str;

}

string generatephonenumber() { // создание случайного дня рождения

return "+7 " + correct(rand() % 10, 3) + " " + correct(rand() % 10, 3) + "-" + correct(rand() % 10, 2) + "-" + correct(rand() % 10, 2);

}

string generatepasswordnumber() { // создание случайного номера паспорта

return correct(rand() % 10000, 4) + " " + correct(rand() % 1000000 \* 100 + rand(), 6);

}

void save(hashtable& table, ofstream& file) { // функция для сохранения таблицы в файл

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

if (table.table[i] != nullptr) {

file << table.table[i]->key << endl;

file << table.table[i]->value << endl;

}

}

}

void clean\_table(hashtable& table) { // функция для очищения таблицы

for (int i = 0; i < m; i++) { // прохождение по таблице

node\* cur = table.table[i];

delete cur; // удаление элемента

table.table[i] = nullptr;

}

}

void recovery(hashtable& table, ifstream& file) { // функция для восстановления данных из файла в таблицу

string key, value;

getline(file, key); // считывание данных

if (key != "") {

getline(file, value); // считывание данных

add(table, key, value); // добавление данных в таблицу

while (getline(file, key)) {

getline(file, value); // считывание данных

add(table, key, value); // добавление данных в таблицу

}

}

}

int main() {

system("chcp 1251");

srand(time(NULL));

hashtable table; // создание таблицы

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

string FIO = generatefullname(); // создание ключа

string human = FIO + " | " + generatephonenumber() + " | " + generatepasswordnumber(); // создание элемента таблицы

add(table, FIO, human); // добавление элемента в таблицу

}

cout << "Hash-таблица: " << endl;

print(table);

int index = rand() % m;

while (table.table[index] == nullptr) {

index = rand() % m;

}

node\* randomhuman = table.table[index];

string key\_remove = randomhuman->key; // создание ключа для удаления

cout << "Удаление по ключу: " << key\_remove << endl;

if (remove(table, key\_remove))

cout << "Элемент успешно удален." << endl;

else

cout << "Элемент не найден." << endl;

print(table); // вывод таблицы

index = rand() % m;

while (table.table[index] == nullptr) {

index = rand() % m;

}

randomhuman = table.table[index];

string get\_key = randomhuman->key; // создание ключа для получения элемента

cout << "Получение по ключу: " << get\_key << endl;

node\* node\_get = get(table, get\_key);

if (node\_get != nullptr)

cout << "Элемент успешно найден." << node\_get->value << endl;

else

cout << "Элемент не найден." << endl;

print(table);

cout << "Число коллизий: " << collisionscnt << endl;

ofstream file;

file.open("ffile.txt"); // доступ к файлу для считывания

save(table, file); // сохранение данных в файл

clean\_table(table); // очищение таблицы

file.close();

ifstream ffile;

ffile.open("ffile.txt");

recovery(table, ffile); // восстановление данных в таблицу

cout << "Восстановленная из файла таблица: " << endl;

print(table);

ffile.close();

return 0;

}

1. Результат работы программы:

