# Лабораторна робота № 10. Алгоритми пошуку

# з використанням таблиць

***Мета:*** закріпити знання про алгоритми пошуку, що вимагають додаткової пам’яті; одержати навички виконання операцій пошуку із використанням таблиць прямого доступу, справочників та хешованих таблиць.

**1 Вимоги**

**1.1 Розробник**

* Макаренко Владислав Олександрович
* Студент 1-го курсу
* Групи КІТ-120а

**1.2 Загальне завдання**

Для вмісту файла створити таблицю прямого доступу або хеш-таблицю у відповідності до завдання з табл. 10.1. Перевірити працездатність створених таблиць на прикладі операцій пошуку

**1.2 Загальне завдання**

Хеш-таблиця з повторним хешуванням

**2 Описи програм**

**Код програми**

#include <iostream>  
#include <string>  
using namespace std;  
int Insert(int *key*, string &*surname*, int *number*, int *adr*);  
int Search(int *key*);  
int Hash(int *key*);  
  
#define N 20 //кількість записів у таблиці  
#define EMPTY -1  
  
struct Student  
{  
 int key; //средний бал  
 int adr = EMPTY;  
 string surname;  
 int number;  
};  
  
Student **hashTable**[N]; //хеш - таблиця  
  
  
  
int main() {  
 int keys[N];  
 for (int i = 0; i < N; ++i) {  
 keys[i] = rand() % 40 + 60;  
 }  
 string surname[N]={"Ivanov", "Petrov", "Sidorov", "Smirnov", "Popov", "Sokolov"};  
 int numbers[N];  
 for (int i = 0; i < N; ++i) {  
 numbers[i] = rand() % 444 + 1;  
 }  
 int i;  
 int res;  
 int key;  
 cout <<"\nKeys -> ";  
 for (i = 0; i < N; i++)  
 cout << keys[i] <<" ";  
 for (i = 0; i < N; i++)  
 Insert(keys[i], surname[rand() % 6], numbers[i], i);  
 cout <<"\nHashed table \n\n ";  
 cout << "Average(key)" << " - " << "adr" << " " << "Surname" << " " << "number" << endl;  
 for (i = 0; i < N; i++) {  
 cout << " ";  
 cout << hashTable[i].key;  
 cout << " - " << hashTable[i].adr;  
 cout.width(12);  
 cout << hashTable[i].surname;  
 cout.width(9);  
 cout << hashTable[i].number << "\n";  
 }  
 cout <<" Input searched code (key < 0 - exit) -> ";  
 cin >> key;  
 clock\_t start, end;  
 while (key > 0) {  
 start = clock();  
 res = Search(key);  
 end = clock();  
 if (res==EMPTY) {  
 cout << " NOT FOUND \n";  
 } else {  
 for (int j = 0; j < N; ++j) {  
 if (**hashTable**[j].adr == res) {  
 cout << " ";  
 cout << hashTable[j].key;  
 cout << " - " << hashTable[j].adr;  
 cout.width(12);  
 cout << hashTable[j].surname;  
 cout.width(9);  
 cout << hashTable[j].number << "\n";  
 }  
 }  
 }  
 double time = (double )(end - start) / CLOCKS\_PER\_SEC;  
 printf("Время поиска: %.10f\n", time);  
 cout <<" Input searched code (key < 0 - exit) -> ";  
 cin >> key;  
 }  
}  
  
int Hash(int *key*) //функція хешування  
{  
 return (*key* % N);  
}  
  
int Insert(int *key*, string &*surname*, int *number*, int *adr*) //додавання ключа у хеш-таблицю  
{  
 int real\_hash;  
 real\_hash=Hash(*key*);  
 if (**hashTable**[real\_hash].adr != EMPTY) {  
 int my\_hash = real\_hash;  
 do {  
 real\_hash = Hash(real\_hash + 1);  
 } while(!((my\_hash==real\_hash)||(**hashTable**[real\_hash].adr==EMPTY)));  
 if (**hashTable**[real\_hash].adr != EMPTY)  
 return 0;  
 }  
 **hashTable**[real\_hash].key = *key*;  
 hashTable[real\_hash].surname = surname;  
 **hashTable**[real\_hash].number = *number*;  
 **hashTable**[real\_hash].adr = *adr*;  
 return 1;  
}  
  
int Search(int *key*) //функція пошука ключа  
{ int addr, a1;  
 addr=Hash(*key*);  
 if (**hashTable**[addr].adr==EMPTY)  
 return EMPTY; //місце вільне – ключа немає в таблиці  
 else  
 if (**hashTable**[addr].key==*key*)  
 return **hashTable**[addr].adr; //ключ знайдений на своєму місці  
 else //місце зайняте іншим ключем  
 { a1=Hash(addr+1);  
 //Пошук, поки не знайдений ключ чи не зроблене повне обертання  
 while((**hashTable**[a1].key != *key*)&&(a1!=addr))  
 a1=Hash(a1+1);  
 if (**hashTable**[a1].key != *key*)  
 return EMPTY;  
 else  
 return **hashTable**[a1].adr;  
 }  
}

**Результати виконання програми**

За алгоритмом коду демонструємо роботу програми для 20 елементів (див. рис. 1).

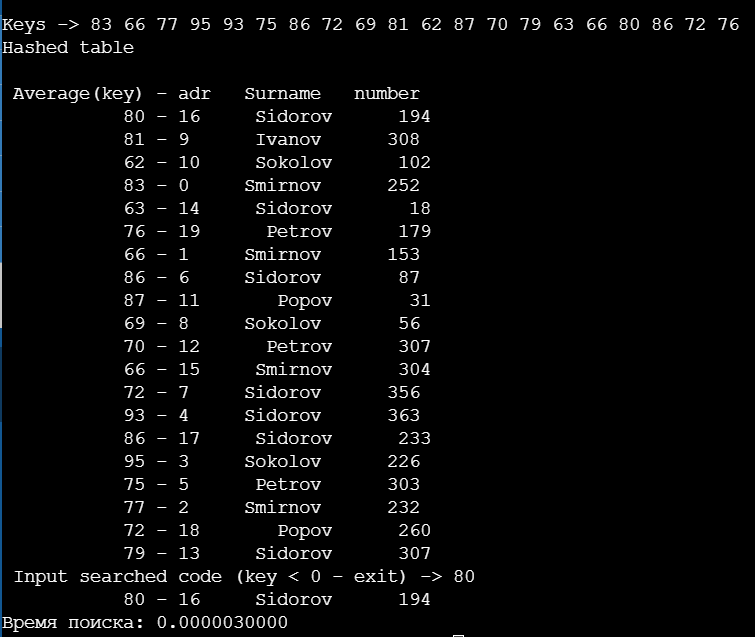


Рисунок 1 – Результати роботи першого кроку

Таблиця 1 – Результати тестування алгоритму пошуку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кількість елементів | 20 | 100 | 1000 | 10000 |
| Кількість порівнянь | 2 | 2 | 2 | 2 |
| Час пошуку | 0.000003 | 0.000021 | 0.000024 | 0.000027 |

**Висновок:** на цій лабораторній роботі ми закріпили знання про алгоритми пошуку, що вимагають додаткової пам’яті; одержали навички виконання операцій пошуку із використанням таблиць прямого доступу, справочників та хешованих таблиць.