**Лабораторная работа №9**

1).Реализовать хеширование методом прямого связывания. Вывести на экран построенные списки, организовать поиск элемента по ключу.

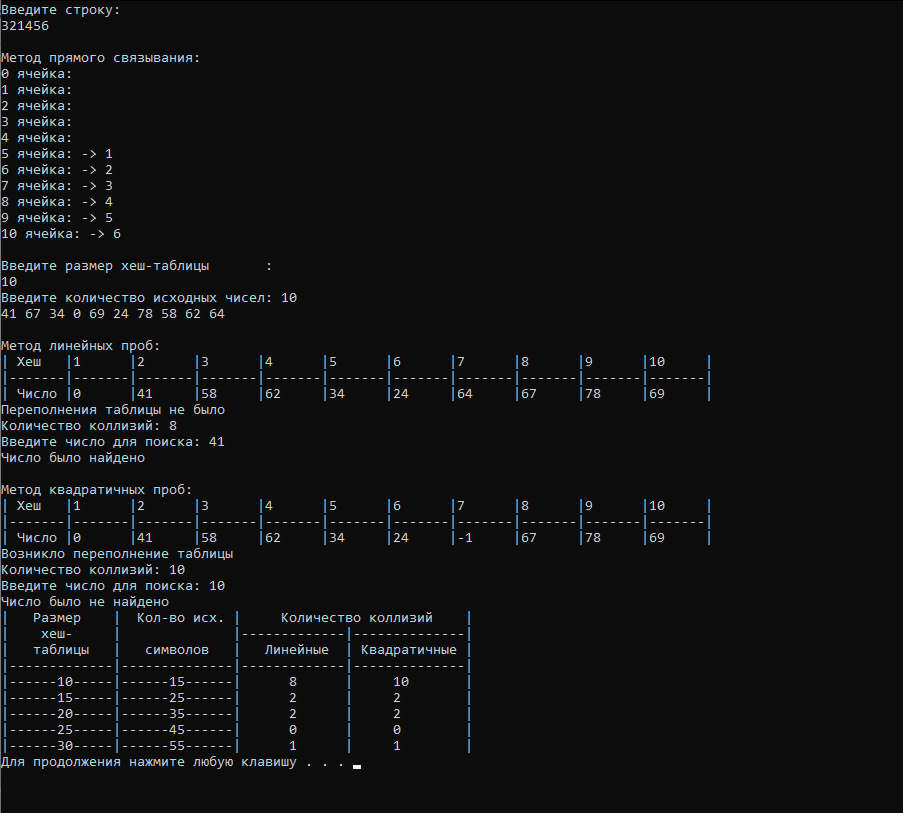
2).Реализовать хеширование методом открытой адресации,

использовать два способа разрешения коллизий: линейные и квадратичные пробы. Вывести на экран заполненные хеш-таблицы.

3).Подсчитать и сравнить количество коллизий при линейных и квадратичных пробах, построить таблицу.

4). Дополнительное задание (на 5+):

Организовать поиск элемента с заданным ключом для методов прямого связывания и открытой адресации.



В данной лабораторной работе я реализовал хеширование методом прямого связывания и открытой адресации. В прямом связывании организовал поиск по ключу. В методе открытой адресации реализовал два способа решения коллизий: линейный и квадратичный. Программа так же подсчитывает количество коллизий и выводит на экран. И так же в этом методе присутствует поиск по ключу.

#include <iostream>

#include <string.h>

#include <iomanip>

#include <sstream>

#define mass 11

using namespace std;

int m;// = 10;

int table[1000] = {0};

int n;

int a[1000];

bool overflow;

int collisions, collisions1, collisions2;

class open\_address

{

public:

int n=0;

int col=0;

char table[mass];

virtual int hash(char value)=0;

virtual void print\_name()=0;

void add(char \*value)

{

// col=0;

for(int i=0; i<(int)strlen(value); ++i)

{

table[hash(value[i])]=value[i];

}

}

};

int HashF(int num)

{

return num % m;

}

class first

{

public:

char data;

first \*next;

};

class hashlist

{

private:

first \*head=NULL;

public:

void add(char value)

{

first \*new\_node=new first;

new\_node->data=value;

new\_node->next=NULL;

if(head==NULL)

{

head=new\_node;

}

else {

first \*pos=head;

while(pos->next!=NULL)

{

if(pos->data==value)

{

break;

}

pos=pos->next;

}

if(pos->data!=value)

{

pos->next=new\_node;

}

}

}

void print()

{

first \*pos=head;

while(pos!=NULL)

{

cout<<" -> "<<pos->data;

pos=pos->next;

}

}

int search(char value)

{

first \*pos=head;

int index=0;

bool flag=false;

while(pos!=NULL)

{

index++;

if(pos->data==value)

{

flag=true;

break;

}

pos=pos->next;

}

if(!flag)

{

index = 0;

}

return index;

}

};

class priam\_link

{

public:

hashlist table[mass];

int hash(char value)

{

return value%mass;

}

void add(char \*value)

{

for(int i=0; i<(int)strlen(value); ++i)

{

table[hash(value[i])].add(value[i]);

}

}

void print() {

cout<<endl<<"Метод прямого связывания:"<<endl;

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

cout<<i<<" ячейка:";

table[i].print();

cout<<endl;

}

cout<<endl;

}

void search(char value) {

if(table[hash(value)].search(value)!=0) {

cout<<"Метод прямого связывания: Элемент '"<<value<<"' найден в "<<hash(value)<<" ячейке с "<<table[hash(value)].search(value)<<" индексом"<<endl;

}

else {

cout<<"Метод прямого связывания: Элемент '"<<value<<"' не найден."<<endl;

}

}

};

// Создает хеш-таблицу методом линейных проб

void CreateTable1()

{ int i;

for (i = 0; i < 1000; i++) // Очищаем хэш-таблицу

table[i] = -1;

collisions = 0;

collisions1 = 0;

int h, h0, g;

for(i = 0; i < n; i++)

{ h = HashF(a[i]);

h0 = h;

g = 1;

while (1)

{

if (table[h] == a[i]) // Если число повторяется

{

break;

}

if (table[h] == -1) // Если нашли пустое место в хэш-таблице

{

table[h] = a[i];

break;

}

else

{

collisions++;

collisions1++;

}

if (g >= m) // Если переполнение

{ overflow = true;

return;

}

h = h0 + g;

if (h >= m)

h -= m;

g++;

}

}

overflow = false;

}

// Создает хеш-таблицу методом квадратичных проб

void CreateTable2()

{ int i;

for (i = 0; i < 1000; i++) // Очищаем хэш-таблицу

table[i] = -1;

collisions = 0;

collisions2 = 0;

int h, d;

for (i = 0; i < n; i++)

{ h = HashF(a[i]);

d = 1;

while (1)

{

if (table[h] == a[i]) // Если число повторяется

break;

if (table[h] == -1) // Если нашли пустое место в хэш-таблице

{

table[h] = a[i];

break;

}

else

{

collisions++;

collisions2++;

}

if (d >= m) // Если переполнение

{

overflow = true;

return;

}

h += d;

if (h >= m)

h -= m;

d += 2;

}

}

overflow = false;

}

// Выводит хеш-таблицу на экран

void ShowTable()

{

cout << ("| Хеш |");

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

cout << i+1 << "\t|";

cout << "\n|-------|";

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

cout << "-------|";

cout << ("\n| Число |");

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

cout << table[i] << "\t|";

cout << endl;

if (!overflow) cout << "Переполнения таблицы не было\n" ;

else cout << "Возникло переполнение таблицы\n";

cout << "Количество коллизий: " << collisions << endl;

}

void ShowTable1()

{

cout << "|------" << m << "-----|" << "------" << n << "------|" << " " << collisions1 << "\t |" << "\t " << collisions2 << "\t |\n";

}

// Ищет число в хеш-таблице, построенной методом линейных проб

bool Search1(int num)

{ int h = HashF(num);

for(int g = h; g < m; ++h)

{ if (table[h] == num) // Если число найдено

return true;

}

return false; // Если переполнение

}

// Ищет число в хеш-таблице, построенной методом квадратичных проб

bool Search2(int num)

{ int h, d;

h = HashF(num);

d = 1;

while (1)

{ if (table[h] == num) // Если число найдено

return true;

if (d >= m) // Если переполнение

return false;

h += d;

if (h >= m)

h -= m;

d += 2;

}

}

int main(int argc, char\* argv[])

{

setlocale(LC\_CTYPE, "Russian");

priam\_link priam\_link\_hash;

stringstream ss;

char string[100];

cout<<"Введите строку:"<<endl;

int number=0;

cin >> number;

ss << number;

ss >> string;

system("cls");

string[strcspn(string,"\n")]=0;

priam\_link\_hash.add(string);

priam\_link\_hash.print();

cout << ("Введите размер хеш-таблицы : ") <<'\n'; cin >> m;

cout << ("Введите количество исходных чисел: "); cin >> n;

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

{ a[i] = rand() % 100;

std::cout << a[i] << ' ';

}

cout << ("\n\nМетод линейных проб:\n");/////////////////////////////////////

CreateTable1();

ShowTable();

// Число для поиска

int k; cout << ("Введите число для поиска: "); cin >> k;

if (Search1(k))

cout << ("Число было найдено\n");

else

cout << ("Число было не найдено\n");

cout << ("\nМетод квадратичных проб:\n");///////////////////////////////////

CreateTable2();

ShowTable ();

cout << ("Введите число для поиска: ");

cin >> k;

if (Search2(k))

cout << ("Число было найдено\n");

else

cout << ("Число было не найдено\n");

cout << ("| Размер | Кол-во исх. | Количество коллизий |\n");

cout << ("| хеш- | |-------------|--------------|\n");

cout << ("| таблицы | символов | Линейные | Квадратичные |\n");

cout << ("|-------------|--------------|-------------|--------------|\n");

m=5;

n=5;

for(int i=0;i<5;i++)

{

m=m+5;

n=n+10;

CreateTable1();

CreateTable2();

ShowTable1();

}

system("pause");

return 0;

}