Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных средств

Лабораторная работа № 5

«Принципы работы с хеш-таблицами и двусвязными списками»

Проверил: Выполнил:

асс. каф. ЭВМ ст. гр. 250504

И.Г. Скиба П.А. Щербо

МИНСК 2023

Цель работы – освоить принципы работы с хеш-таблицами и двусвязными списками.

Условие:

Написать программу поиска с кэшем.

Программа должна выполнять поиск IP адреса по доменному имени в файле, содержащем DNS таблицу. Поиск необходимо реализовать через кэш с алгоритмом LRU.

1. Пользователь задает доменное имя.
2. Программа проверяет наличие данных в кэше:
   1. если HIT (присутствует в кэше) – возвращает IP адрес пользователю.
   2. если MISS (отсутствует в кэше) – программа производит поиск по файлу, помещает запись в кэш и возвращает IP адрес.

Кэш должен содержать значения <key, value>, где k – это доменное имя, а v – IP адрес. Данные в кэш помещаются по алгоритму LRU.

DNS-файл содержит данные вида:

lms.bsuir.by IN A 192.168.251.23 (тип 1)

lms2.bsuir.by IN CNAME lms.bsuir.by (тип 2)

Код программы:

typedef struct cache Node {

char key[MAX\_LINE\_LENGTH];

char value[MAX\_LINE\_LENGTH];

struct cacheNode\* next;

struct cacheNode\* prev;

} cache\_node\_t;

typedef struct cache {

int size;

int capacity;

cache\_node\_t\* head;

cache\_node\_t\* tail;

} cache\_t;

typedef struct hashNode {

char key[MAX\_LINE\_LENGTH];

char value[MAX\_LINE\_LENGTH];

struct hashNode\* next;

} hash\_node\_t;

typedef struct hashTable {

int size;

hash\_node\_t\*\* table;

} hash\_table\_t;

void addElement(hash\_table\_t\* hash\_table, cache\_t\* cache) {

char s[MAX\_LINE\_LENGTH];

printf("Введите элемент для добавления: ");

rewind(stdin);

fgets(s, MAX\_LINE\_LENGTH, stdin);

s[strlen(s) - 1] = '\0';

char\* value = getFromCache(cache, hash\_table, s);

if (value != NULL) {

printf("Элемент уже существует в кэше.\n");

} else {

value = getIP(hash\_table, s);

if (value != NULL) {

addToCache(cache, hash\_table, s, value);

printf("Элемент успешно добавлен в кэш.\n");

} else {

printf("Элемент не найден в файле.\n");

printf("Если вы хотите добавить его, введите 1.\n");

printf("Если вы хотите ввести другой элемент, введите 2.\n");

printf("Если вы хотите перезапустить, введите 3.\n");

int option = 0;

scanf("%d", &option);

switch (option) {

case 1:

addDomainToFile(s);

break;

case 2:

break;

case 3:

printf("Введите ваш выбор: ");

return;

default:

printf("Неверный выбор. Попробуйте еще раз.\n");

break;

}

}

}

}

char\* validIP(void) {

char s[MAX\_LINE\_LENGTH];

while (1) {

rewind(stdin);

fgets(s, MAX\_LINE\_LENGTH, stdin);

s[strlen(s) - 1] = '\0';

int i = 0;

int sum = 0;

int fl = 1;

int point = 0;

while (s[i] != '\0') {

if ((s[i] < '0' || s[i] > '9') && s[i] != '.') {

fl = 0;

printf("Введите корректный IP адресс\n");

break;

}

else {

sum \*= 10;

sum += s[i] - 48;

if (s[i + 1] == '.') {

if (sum > 255) {

printf("Число может быть до 255!\n");

fl = 0;

break;

}

point++;

if (point > 3) {

printf("IP адресс должен быть написан в формате 255.255.255.255\n");

fl = 0;

break;

}

sum = 0;

}

}

i++;

}

if (point < 3 && fl != 0) {

printf("IP адресс должен быть написан в формате 255.255.255.255\n");

fl = 0;

break;

}

if (fl == 0) {

printf("Введите IP адресс: ");

rewind(stdin);

}

else {

break;

}

}

return strdup(s);

}

void addDomainToFile(char\* domain) {

FILE\* file = fopen("dns.txt", "r+");

char s[MAX\_LINE\_LENGTH];

rewind(stdin);

printf("Это домен или второе название?(1 или 2): ");

fgets(s, MAX\_LINE\_LENGTH, stdin);

s[strlen(s) - 1] = '\0';

if (strcmp(s, "1") == 0) {

strncat(domain, " IN A ", 7);

rewind(stdin);

printf("Введите IP адресс: ");

strcpy(s, validIP());

s[strlen(s) - 1] = '\0';

strncat(domain, s, MAX\_LINE\_LENGTH);

}

else {

strncat(domain, " IN CNAME ", 11);

rewind(stdin);

printf("Введите домен: ");

fgets(s, MAX\_LINE\_LENGTH, stdin);

s[strlen(s) - 1] = '\0';

strncat(domain, s, MAX\_LINE\_LENGTH);

}

fseek(file, 0, SEEK\_END);

fputs("\n", file);

fputs(domain, file);

fclose(file);

}

void findDomain(char\* IP) {

FILE\* file = fopen("dns.txt", "r");

char s[MAX\_LINE\_LENGTH];

fseek(file, 0, SEEK\_SET);

int fl = 0;

char domain[MAX\_LINE\_LENGTH];

rewind(stdin);

while (fgets(s, MAX\_LINE\_LENGTH, file)) {

char\* buf;

buf = strtok(s, " \n");

strcpy(domain, buf);

buf = strtok(NULL, " \n");

buf = strtok(NULL, " \n");

if (strcmp(buf, "A") == 0) {

buf = strtok(NULL, " \n");

if (strcmp(buf, IP) == 0) {

fl = 1;

break;

}

}

}

if (fl == 0) {

printf("Этого IP нет в файле\n");

return;

}

printf("Домен: %s\n", domain);

fseek(file, 0, SEEK\_SET);

rewind(stdin);

while (fgets(s, MAX\_LINE\_LENGTH, file)) {

char\* buf;

char canonical[MAX\_LINE\_LENGTH];

buf = strtok(s, " \n");

strcpy(canonical, buf);

buf = strtok(NULL, " \n");

buf = strtok(NULL, " \n");

if (strcmp(buf, "CNAME") == 0) {

buf = strtok(NULL, " \n");

if (strcmp(buf, domain) == 0) {

printf("Канонический вид: %s\n", canonical);

}

}

}

}

void freeMemory(cache\_t\* cache, hash\_table\_t\* hash\_table) {

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

hash\_node\_t\* hash\_node = hash\_table->table[i];

while (hash\_node != NULL) {

hash\_node\_t\* temp = hash\_node;

hash\_node = hash\_node->next;

free(temp);

}

}

free(hash\_table->table);

cache\_node\_t\* cache\_node = cache->head;

while (cache\_node != NULL) {

cache\_node\_t\* temp = cache\_node;

cache\_node = cache\_node->next;

free(temp);

}

}

char\* getIP(hash\_table\_t\* hash\_table, char\* key) {

int index = hashFunction(key);

hash\_node\_t\* hash\_node = hash\_table->table[index];

while (hash\_node != NULL) {

if (strcmp(hash\_node->key, key) == 0) {

return hash\_node->value;

}

hash\_node = hash\_node->next;

}

return NULL;

}

cache\_node\_t\* createNode(char\* key, char\* value) {

cache\_node\_t\* new\_node = (cache\_node\_t\*)malloc(sizeof(cache\_node\_t));

strcpy(new\_node->key, key);

strcpy(new\_node->value, value);

new\_node->next = NULL;

new\_node->prev = NULL;

return new\_node;

}

void addNodeToHead(cache\_t\* cache, cache\_node\_t\* node) {

node->next = cache->head;

node->prev = NULL;

if (cache->head != NULL) {

cache->head->prev = node;

}

else {

cache->tail = node;

}

cache->head = node;

}

void removeTail(cache\_t\* cache) {

cache\_node\_t\* temp = cache->tail;

cache->tail = cache->tail->prev;

if (cache->tail != NULL) {

cache->tail->next = NULL;

}

else {

cache->head = NULL;

}

free(temp);

}

void addToCache(cache\_t\* cache, hash\_table\_t\* hash\_table, char\* key, char\* value) {

cache\_node\_t\* new\_node = createNode(key, value);

addNodeToHead(cache, new\_node);

cache->size++;

if (cache->size > cache->capacity) {

hash\_node\_t\* hash\_node = hash\_table->table[hashFunction(cache->tail->key)];

if (strcmp(hash\_node->key, cache->tail->key) == 0) {

hash\_table->table[hashFunction(cache->tail->key)] = hash\_node->next;

}

else {

while (hash\_node->next != NULL && strcmp(hash\_node->next->key, cache->tail->key) != 0) {

hash\_node = hash\_node->next;

}

if (hash\_node->next != NULL) {

hash\_node\_t\* temp = hash\_node->next;

hash\_node->next = temp->next;

free(temp);

}

}

removeTail(cache);

cache->size--;

}

hash\_node\_t\* hash\_node = hash\_table->table[hashFunction(key)];

while (hash\_node->next != NULL && strcmp(hash\_node->next->key, key) != 0) {

hash\_node = hash\_node->next;

}

if (hash\_node->next != NULL) {

free(hash\_node->next);

}

hash\_node->next = createHashNode(key, value);

hash\_table->size++;

}

char\* getFromCache(cache\_t\* cache, hash\_table\_t\* hash\_table, char\* key) {

hash\_node\_t\* hash\_node = hash\_table->table[hashFunction(key)];

while (hash\_node != NULL && strcmp(hash\_node->key, key) != 0) {

hash\_node = hash\_node->next;

}

if (hash\_node == NULL) {

return NULL;

}

cache\_node\_t\* cache\_node = cache->head;

while (cache\_node != NULL && strcmp(cache\_node->key, key) != 0) {

cache\_node = cache\_node->next;

}

if (cache\_node == NULL) {

return NULL;

}

if (cache\_node != cache->head) {

if (cache\_node == cache->tail) {

cache->tail = cache\_node->prev;

cache->tail->next = NULL;

}

else {

cache\_node->prev->next = cache\_node->next;

cache\_node->next->prev = cache\_node->prev;

}

addNodeToHead(cache, cache\_node);

}

return cache\_node->value;

}

void printCache(cache\_t\* cache) {

cache\_node\_t\* current\_node = cache->head;

printf("Cache:\n");

while (current\_node != NULL) {

printf("%s %s\n", current\_node->key, current\_node->value);

current\_node = current\_node->next;

}

}

void initHashTable(hash\_table\_t\* hash\_table, cache\_t\* cache, char\* s) {

FILE\* file = fopen("dns.txt", "r");

char line[MAX\_LINE\_LENGTH];

char key[MAX\_LINE\_LENGTH];

char value[MAX\_LINE\_LENGTH];

char\* token;

while (fgets(line, MAX\_LINE\_LENGTH, file)) {

token = strtok(line, " \n");

strcpy(key, token);

if (strcmp(s,key)== 0){

token = strtok(NULL, " \n");

token = strtok(NULL, " \n");

if (strcmp(token, "A") == 0) {

token = strtok(NULL, " \n");

strcpy(value, token);

hash\_node\_t\* new\_node = createHashNode(key, value);

int hash = hashFunction(key);

new\_node->next = hash\_table->table[hash];

hash\_table->table[hash] = new\_node;

hash\_table->size++;

}

else {

if (strcmp(token, "CNAME") == 0) {

token = strtok(NULL, " \n");

initHashTable(hash\_table, cache, token);

char\* IP = getIP(hash\_table, token);

hash\_node\_t\* new\_node = createHashNode(key, IP);

int hash = hashFunction(key);

new\_node->next = hash\_table->table[hash];

hash\_table->table[hash] = new\_node;

}

}

}

}

fclose(file);

}

hash\_node\_t\* createHashNode(char\* key, char\* value) {

hash\_node\_t\* new\_node = (hash\_node\_t\*)malloc(sizeof(hash\_node\_t));

strcpy(new\_node->key, key);

strcpy(new\_node->value, value);

new\_node->next = NULL;

return new\_node;

}

int hashFunction(char\* key) {

int hash = 0;

for (int i = 0; i < strlen(key); i++) {

hash += key[i];

}

return hash % TABLE\_SIZE;

}

void menu(hash\_table\_t\* hash\_table, cache\_t\* cache) {

int fl = 1;

int choice = 0;

while (1) {

char s[MAX\_LINE\_LENGTH];

printf("\n----- Меню -----\n");

printf("1. Добавить элемент\n");

printf("2. Ввести доменное имя для получения IP-адреса\n");

printf("3. Показать таблицу\n");

printf("4. Ввести IP-адрес для получения доменного имени\n");

printf("5. Выход\n");

printf("-----------------\n");

printf("Введите ваш выбор: ");

if (fl == 1) {

choice = 0;

scanf("%d", &choice);

rewind(stdin);

}

if (choice == 1) {

addElement(hash\_table, cache);

}

else if (choice == 2) {

printf("Введите доменное имя: ");

rewind(stdin);

fgets(s, MAX\_LINE\_LENGTH, stdin);

s[strlen(s) - 1] = '\0';

char\* value = getFromCache(cache, hash\_table, s);

if (value != NULL) {

printf("IP-адрес: %s\n", value);

} else {

initHashTable(hash\_table, cache, s);

char\* IP = getIP(hash\_table, s);

if (IP != NULL) {

addToCache(cache, hash\_table, s, IP);

value = getFromCache(cache, hash\_table, s);

printf("IP-адрес: %s\n", value);

} else {

printf("Доменное имя не найдено.\n");

}

}

}

else if (choice == 3) {

printCache(cache);

}

else if (choice == 4) {

printf("Введите IP-адрес: ");

rewind(stdin);

fgets(s, MAX\_LINE\_LENGTH, stdin);

s[strlen(s) - 1] = '\0';

findDomain(s);

}

else if (choice == 5) {

freeMemory(cache, hash\_table);

break;

}

else{

printf("Неверный выбор");

}

fl = 1;

}

}

int main(void) {

hash\_table\_t hash\_table = { 0, (hash\_node\_t\*\*)calloc(TABLE\_SIZE, sizeof(hash\_node\_t\*)) };

cache\_cache = { 0, CACHE\_SIZE, NULL, NULL };

menu(&hash\_table, &cache);

return 0;

}

Блок схема программы (рис.1):

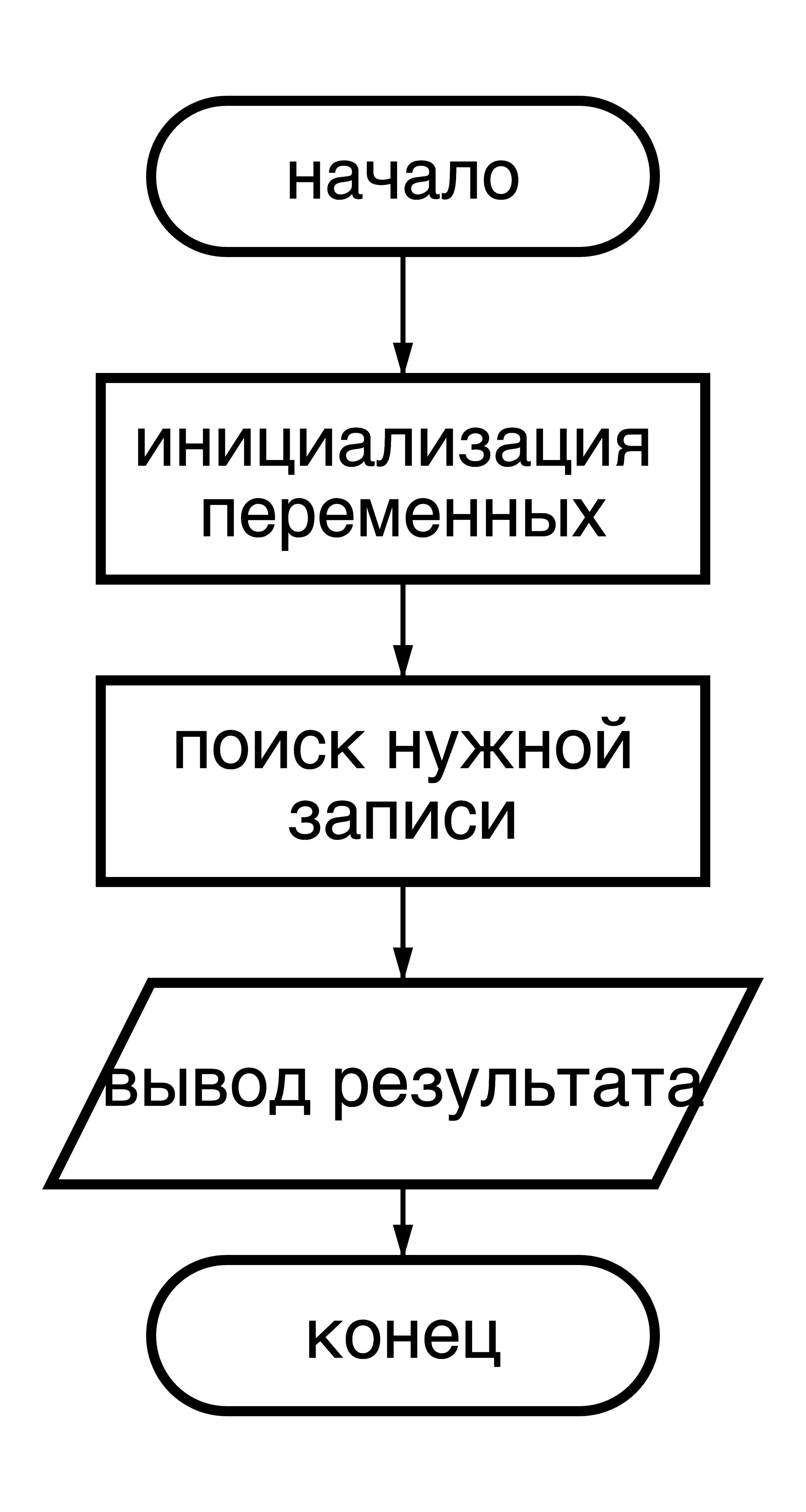


Рисунок 1- Блок схема программы

Результат поиска записи (рис.2, рис.3):

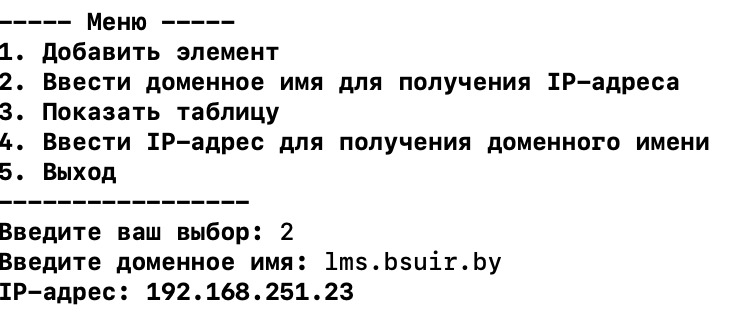


Рисунок 2 - Результат поиска записи (тип 1)

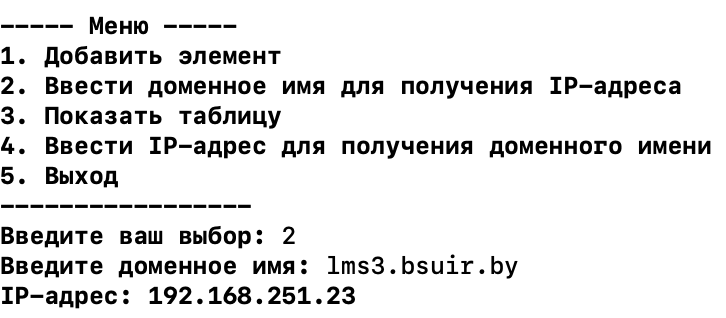


Рисунок 3 - Результат поиска записи (тип 2)