Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**КУРСОВАЯ РАБОТА**

**по курсу "Вычислительные системы"**

**II семестр**

**Задание 9**

**«Сортировка и поиск»**

Студент: Фролов М.А.

Группа: М8О-106Б-20

Оценка:

Дата:

Подпись:

Оглавление

[1. Общая информация 3](#_Toc73231542)

[2. Постановка задачи 4](#_Toc73231543)

[3. Общий метод решения 5](#_Toc73231544)

[4. Общие сведения о программе 5](#_Toc73231545)

[5. Таблица переменных и констант 6](#_Toc73231546)

[6. Функциональное назначение 7](#_Toc73231547)

[7. Сложностные оценки 7](#_Toc73231548)

[8. Программа на Си 7](#_Toc73231549)

[9. Заключение 20](#_Toc73231550)

[10. Список используемых источников 21](#_Toc73231551)

# Общая информация

Алгоритм линейного выбора с подсчётом:

При упорядочении таблицы этим методом необходима память для хранения исходной таблицы, а также дополнительно должна быть выделена память под счетчик для каждого элемента таблицы.

В этом алгоритме элементы не перемещаются. Он подобен сортировке таблицы адресов, поскольку таблица счетчиков определяет конечное положение элементов.

Просмотр таблицы начинается с первой записи. Ее ключ сравнивается с ключами последующих записей. При этом счетчик большего из сравниваемых ключей увеличивается на 1. При втором просмотре таблицы первый ключ уже не рассматривается, второй ключ сравнивается со всеми последующими. Результаты сравнений фиксируются в счетчиках. Для таблицы, содержащей N элементов, этот процесс повторяется N-1 раз.

После выполнения всех просмотров счетчик каждого элемента указывает, какое количество ключей таблицы меньше ключа этого элемента. Эти счетчики используются затем в качестве индексов элементов результирующей таблицы. Поместив записи в результирующую таблицу в соответствии со значениями их счетчиков, получим упорядоченную таблицу. Другими словами, если известно, что некоторый ключ превышает ровно 27 других, то после сортировки соответствующий элемент должен занять 28-е место.

# Постановка задачи

Составить программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

Программа должна вводить значения элементов неупорядоченной таблицы и проверять работу процедуры сортировки в трех случаях:

1. элементы таблицы с самого начала упорядочены;
2. элементы таблицы расставлены в обратном порядке;
3. элементы таблицы не упорядочены.

В последнем случае можно использовать встроенные процедуры генерации псевдослучайных чисел.

Для каждого вызова процедуры сортировки необходимо печатать исходное состояние таблицы и результаты сортировки. После выполнения сортировки программа должна вводить ключи и для каждого из них выполнять поиск в упорядоченной таблице с помощью процедуры двоичного поиска и печатать найденные элементы, если они присутствуют в таблице.

В процессе отладки и тестирования рекомендуется использовать команды обработки текстовых файлов ОС UNIX и переадресацию ввода-вывода. Тестовые данные необходимо заранее поместить в текстовые файлы.

В качестве текста для записей таблицы взять изображение ASCII-графики. Каждый элемент таблицы, содержащий ключ и текст записи, распечатывать в отдельной строке.

# Общий метод решения

Работа идёт с четырьмя программами языка Си – основной и тремя вспомогательными для работы с таблицей и строками, тремя заголовочными файлами (содержат описания структур, необходимых для работы соответственных программ).

Основная программа считывает таблицу со стандартного ввода и печатает её из внутреннего представления. Потом таблица сортируется и выводится в отсортированном виде.

Далее программе на стандартный ввод подаются ключи, и с помощью бинарного поиска программа находит самый левый элемент в таблице с заданным значением или сообщение о том, что элементов с таким ключом в таблице нет.

# Общие сведения о программе

Программное и аппаратное обеспечения для запуска данной программы на ПК не ограничено в выборе. Операционная система семейства Linux - Ubuntu. Язык программирования Си.

Имена файлов:

1. kursach9.c;
2. vector.h;
3. vector\_functions.c;
4. vector\_norm.h
5. vector\_norm\_functions.c;
6. tabl\_el.h;
7. tabl\_el\_functions.c.

# Таблица переменных и констант

Описание переменных и структур в заголовочном файле “ tabl\_el.h”:

Структура key:

|  |  |
| --- | --- |
| char k1[28]; | Первая часть ключа |
| int k2; | Вторая часть ключа |

Структура tab\_el:

|  |  |
| --- | --- |
| key key; | Ключ |
| char val[100]; | Данные |
| int size\_of\_val; | Длина данных |
| int size\_of\_k1; | Длина строки для первой части ключа |

Описание переменных и структур в заголовочном файле “vector.h”:

Структура vector\_tabl\_el:

|  |  |
| --- | --- |
| tabl\_el \*buf; | Указатель на буфер таблицы |
| int size; | Размер таблицы |
| int cap; | Вместительность таблицы |
| int first; | Индекс первой строки таблицы |

Описание переменных и структур в заголовочном файле “vector\_norm.h”:

Структура vector\_i:

|  |  |
| --- | --- |
| int \*buf; | Указатель на буфер массива |
| int size; | Размер массива |
| int cap; | Вместительность массива |
| int first; | Индекс первого элемента массива |

# Функциональное назначение

Программа предназначена для демонстрации сортировки линейным выбором с подсчётом любой таблицы по заданному ключу.

# Сложностные оценки

Ввод и вывод таблицы –

Сортировка линейным выбором с подсчётом – (за происходит подсчёт и проставление индексов в выделенный массив и за происходит перенос элементов в новый массив соответственно новым индексам)

Бинарный поиск –

# Программа на Си

kursach9.c:

#include <stdio.h>

#include <stdbool.h>

#include "tabl\_el.h"

#include "vector.h"

#include "vector\_norm.h"

bool read\_tabl(vector\_tabl\_el \*v){

char c;

c=getchar();

if(c==EOF){

return false;

}

tabl\_el a;

while(c!='\n' && c!=EOF){

ungetc(c,stdin);

if(!read\_tabl\_el(&a)){

return false;

}

vt\_push\_back(v,a);

c=getchar();

}

if(c==EOF){

return false;

}

return true;

}

void lin\_sort\_with\_count(vector\_tabl\_el \*v){

vector\_i V;

vi\_create(&V);

for(int i=0;i<vt\_size(v);i++){

vi\_push\_back(&V,1);

}

for(int i=0;i<vt\_size(v);i++){

for(int j=i+1;j<vt\_size(v);j++){

if(keycmp(vt\_get(v,i).key,vt\_get(v,j).key)<0){

vi\_plus\_one(&V,i);

} else if(keycmp(vt\_get(v,i).key,vt\_get(v,j).key)>=0){

vi\_plus\_one(&V,j);

}

}

}

vector\_tabl\_el v1;

vt\_create(&v1);

tabl\_el a;

for(int i=0;i<vi\_size(&V);i++){

vt\_push\_back(&v1,a);

}

for(int i=0;i<vi\_size(&V);i++){

vt\_set(&v1,vi\_get(&V,i)-1,vt\_get(v,i));

}

for(int i=0;i<vi\_size(&V);i++){

vt\_set(v,i,vt\_get(&v1,i));

}

vt\_destroy(&v1);

vi\_destroy(&V);

}

tabl\_el binary\_search(vector\_tabl\_el v, key a, int \*number){

int l=0;

int r=v.size-1;

int mid;

while(r-l>1){

mid=(r+l)/2;

if(keycmp(a,vt\_get(&v,mid).key)<0){

l=mid;

} else {

r=mid;

}

}

if(keycmp(a,vt\_get(&v,l).key)==0){

\*number=l+1;

return vt\_get(&v,l);

} else if(keycmp(a,vt\_get(&v,r).key)==0){

\*number=r+1;

return vt\_get(&v,r);

} else {

\*number=0;

return vt\_get(&v,r);

}

}

int main(){

printf("\n");

vector\_tabl\_el v;

vt\_create(&v);

if(!read\_tabl(&v)){

return 0;

}

printf("Input table:\n\n");

print\_vector(&v);

lin\_sort\_with\_count(&v);

printf("Sorted table:\n\n");

print\_vector(&v);

key a;

tabl\_el b;

char c;

int val;

c=getchar();

while(c!=EOF){

val=0;

ungetc(c,stdin);

if(!read\_key(&a)){

return 0;

}

b=binary\_search(v,a,&val);

if(val==0){

printf("No specified key in this list\n\n");

} else {

printf("Line:");

print\_tabl\_el(b);

printf("Number of line:%d\n\n",val);

}

c=getchar();

}

}

vector.h:

#ifndef \_VECTOR\_H

#define \_VECTOR\_H

#include <stdbool.h>

#include "tabl\_el.h"

typedef struct {

tabl\_el \*buffer;

int size;

int cap;

int first;

} vector\_tabl\_el;

void vt\_create(vector\_tabl\_el \*v);

void vt\_destroy(vector\_tabl\_el \*v);

int vt\_size(vector\_tabl\_el \*v);

bool vt\_push\_back(vector\_tabl\_el \*v, tabl\_el f);

tabl\_el vt\_get(vector\_tabl\_el \*v, int i);

void print\_tabl\_el(tabl\_el a);

void print\_vector(vector\_tabl\_el \*v);

void vt\_set(vector\_tabl\_el \*v,int i, tabl\_el val);

#endif

vector\_functions.c:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include "tabl\_el.h"

#include "vector.h"

void vt\_create(vector\_tabl\_el \*v){

v->size=0;

v->first=0;

v->cap=0;

v->buffer=NULL;

}

void vt\_destroy(vector\_tabl\_el \*v){

v->size=0;

v->first=0;

v->cap=0;

free(v->buffer);

v->buffer=NULL;

}

tabl\_el vt\_get(vector\_tabl\_el \*v,int i){

return v->buffer[(v->first+i)%v->cap];

}

void vt\_set(vector\_tabl\_el \*v,int i, tabl\_el val){

v->buffer[(v->first+i)%v->cap]=val;

}

int vt\_size(vector\_tabl\_el \*v){

return v->size;

}

bool vt\_grow\_buffer(vector\_tabl\_el \*v){

int tmp=v->cap\*3/2;

if(v->cap==0){

tmp=9;

}

tabl\_el \*n=realloc(v->buffer,sizeof(tabl\_el)\*tmp);

if(n==NULL){

return false;

} else {

v->buffer=n;

for(int i=v->cap-v->first-1;i>=0;i--){

v->buffer[(tmp+i-(v->cap-v->first))%tmp]=v->buffer[(v->first+i)%tmp];

}

v->first=(tmp-(v->cap-v->first))%tmp;

v->cap=tmp;

return true;

}

}

bool vt\_push\_back(vector\_tabl\_el \*v, tabl\_el val){

if(v->size>=v->cap){

if(!(vt\_grow\_buffer(v))){

return false;

}

}

v->buffer[(v->first+v->size)%v->cap]=val;

v->size++;

return true;

}

void print\_tabl\_el(tabl\_el a){

for(int j=0;j<a.size\_of\_k1;j++){

printf("%c",a.key.k1[j]);

}

printf("|%d|",a.key.k2);

for(int j=0;j<a.size\_of\_val;j++){

printf("%c",a.val[j]);

}

printf("\n");

}

void print\_vector(vector\_tabl\_el \*v){

tabl\_el a;

for(int i=0;i<vt\_size(v);i++){

a=vt\_get(v,i);

print\_tabl\_el(a);

}

printf("\n");

}

vector\_norm.h:

#ifndef \_VECTOR\_NORM\_H

#define \_VECTOR\_NORM\_H

#include <stdbool.h>

typedef struct {

int \*buffer;

int size;

int cap;

int first;

} vector\_i;

void vi\_create(vector\_i \*v);

void vi\_destroy(vector\_i \*v);

int vi\_size(vector\_i \*v);

bool vi\_grow\_buffer(vector\_i \*v);

bool vi\_push\_back(vector\_i \*v, int f);

int vi\_get(vector\_i \*v, int i);

void vi\_plus\_one(vector\_i \*v, int i);

#endif

vector\_norm\_functions.c:

#include <stdio.h>

#include <stdbool.h>

#include "vector\_norm.h"

void vi\_create(vector\_i \*v){

v->size=0;

v->first=0;

v->cap=0;

v->buffer=NULL;

}

void vi\_destroy(vector\_i \*v){

v->size=0;

v->first=0;

v->cap=0;

free(v->buffer);

v->buffer=NULL;

}

void vi\_plus\_one(vector\_i \*v, int i){

v->buffer[(v->first+i)%v->cap]++;

}

int vi\_get(vector\_i \*v,int i){

return v->buffer[(v->first+i)%v->cap];

}

int vi\_size(vector\_i \*v){

return v->size;

}

bool vi\_grow\_buffer(vector\_i \*v){

int tmp=v->cap\*3/2;

if(v->cap==0){

tmp=9;

}

int \*n=realloc(v->buffer,sizeof(int)\*tmp);

if(n==NULL){

return false;

} else {

v->buffer=n;

for(int i=v->cap-v->first-1;i>=0;i--){

v->buffer[(tmp+i-(v->cap-v->first))%tmp]=v->buffer[(v->first+i)%tmp];

v->buffer[(v->first+i)%tmp]=0;

}

v->first=(tmp-(v->cap-v->first))%tmp;

v->cap=tmp;

return true;

}

}

bool vi\_push\_back(vector\_i \*v,int val){

if(v->size>=v->cap){

if(!(vi\_grow\_buffer(v))){

return false;

}

}

v->buffer[(v->first+v->size)%v->cap]=val;

v->size++;

return true;

}

tabl\_el.h:

#ifndef \_TABL\_EL\_H

#define \_TABL\_EL\_H

#include <stdbool.h>

typedef struct{

char k1[28];

int k2;

} key;

typedef struct{

int size\_of\_k1;

int size\_of\_val;

char val[100];

key key;

} tabl\_el;

bool read\_tabl\_el(tabl\_el \*a);

bool read\_key(key \*a);

int keycmp(key a, key b);

#endif

tabl\_el\_functions.c:

#include <stdio.h>

#include <stdbool.h>

#include "tabl\_el.h"

bool read\_tabl\_el(tabl\_el \*a){

for(int i=0;i<27;i++){

a->key.k1[i]=' ';

}

char w;

a->size\_of\_k1=0;

for(int i=0;i<27;i++){

w=getchar();

if(w=='|'){

break;

}

if(w==EOF){

return false;

}

a->key.k1[i]=w;

a->size\_of\_k1++;

}

scanf("%d",&a->key.k2);

w=getchar();

if(w==EOF){

return false;

}

a->size\_of\_val=0;

for(int i=0;i<99;i++){

w=getchar();

if(w=='\n'){

break;

}

if(w==EOF){

return false;

}

a->val[i]=w;

a->size\_of\_val++;

}

return true;

}

bool read\_key(key \*a){

for(int i=0;i<27;i++){

a->k1[i]=' ';

}

char w;

for(int i=0;i<27;i++){

w=getchar();

if(w=='|'){

break;

}

if(w==EOF){

return false;

}

a->k1[i]=w;

}

scanf("%d",&a->k2);

getchar();

return true;

}

int keycmp(key a, key b){

for(int i=0;i<27;i++){

if(a.k1[i]>b.k1[i]){

return -1;

} else if(a.k1[i]<b.k1[i]){

return 1;

}

}

if(a.k2>b.k2){

return -1;

} else if(a.k2<b.k2){

return 1;

}

return 0;

}

# Заключение

Линейный выбор с подсчётом относится к тем видам сортировки, которые “не хватают с неба звёзд” по сложностным оценкам памяти и времени, однако являются очень простыми для понимания. Если стоит задача отсортировать массив или таблицу, и при этом мы не ограничены в ресурсах, то линейный выбор с подсчётом отлично подойдёт в таком случае, т.к. реализовать его очень просто и быстро.

# Список используемых источников

1. Методические указания к выполнению курсовых работ. Зайцев В. Е.
2. [Линейный выбор с подсчетом (сравнение и подсчет) (lektsii.org)](https://lektsii.org/6-54558.html)