****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа**

**по курсу «Языки и методы программирования»**

**II Семестр**

**Задание 9**

**Сортировка и поиск**

Группа: М80 – 107Б-18

Студент: Син Денис Дмитриевич

Преподаватель: Ридли Александра Николаевна

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2019.

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Демонстрация работы программы
6. Вывод

**Постановка задачи**.

Составить и отладить программу на языке СИ для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

**Метод сортировки**: Пирамидальная сортировка с просеиванием.

**Тип ключа**: Целый 4 байта.

**Хранение данных и ключей**: вместе.

**Общие сведения о программе**

Дерево проекта:

.

├── Makefile

├── check\_memory.sh

├── main.c

├── table.c

├── table.h

├── tests

│   ├── test1

│   ├── test2

│   └── test3

├── vector.c

├── vector.h

└── course\_project4.docx

Makefile – файл для сборки проекта

Check\_memory.sh – скрипт для запуска valgrind c помощью docker.

main.c – главный файл.

table.c – реализация таблицы и операций с ней.

table.h – заголовочный файл реализации таблицы

vector.c – реализация векторов

vector.h – заголовочный файл реализации векторов

tests – директория с тестовыми файлами

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо создать структуру таблицы, которая основана на векторе ключей и векторе значений, так как ключи и данные хранятся раздельно и реализовать пирамидальную сортировку и бинарный поиск.

Для пирамидальной сортировки необходимо реализовать процедуру просеивания элемента, которая необходима как в преобразовании вектора в кучу, так и для сортировки кучи. Просеивание элемента берет из тройки элементов наибольший и выставляет его в локальный корень, затем просеивает элемент индекс которого был переставлен с корнем. Чтобы построить кучу необходимо для n / 2 -1 элемента применить процедуру просеивания. В итоге после построения кучи у нас получается, что a[i] >= a[2 \* i + 1] и a[i] >= a[2 \* i] для каждого индекса. Сама сортировка заключается в обмене последнего элемента и первого и просеивания нового первого элемента, причем индекс последнего элемента постоянно уменьшается на 1. В итоге получается отсортированный вектор.

Так как у нас ключи и данные хранятся отдельно, то параллельно с операцией обмена ключей в сортировке необходимо делать соответствующие обмены данных. Алгоритм такой сортировки работает за O(n \* log n)

Бинарный поиск осуществляется просто, берем середину между границами и если она меньше чем искомый ключ сдвигаем левую границу на середину иначе сдвигаем правую границу. Алгоритм работает за O(log n)

**Основные файлы программы**.

**Makefile**

CC = gcc

CCKEYS = -Wall -Wextra -Werror -pedantic

LD = gcc

LDKEYS =

main: main.o table.o vector.o

    @$(LD) $(LDKEYS) main.o table.o vector.o -o main

main.o: main.c table.h

    @$(CC) $(CCKEYS) -c main.c

table.o: table.c table.h vector.h

    @$(CC) $(CCKEYS) -c table.c

vector.o: vector.c vector.h

    @$(CC) $(CCKEYS) -c vector.c

clean:

    @rm -rf main.o table.o vector.o main

**table.h**

#ifndef TABLE\_H\_

#define TABLE\_H\_

#include "vector.h"

typedef struct {

vector\_int keys;

vector\_str data;

} Table;

void table\_create(Table \*t);

void    table\_destroy(Table \*t);

void    table\_add\_entry(Table \*t, int key, char \*val);

void table\_build\_heap(Table \*t);

void    table\_heap\_sort(Table \*t);

char    \*table\_binary\_search(Table \*t, int key);

void    table\_print(Table \*t);

void table\_read(Table \*t, FILE \*f);

#endif

**table.c**

#include "table.h"

void table\_create(Table \*t)

{

vector\_int\_create(&t->keys);

vector\_str\_create(&t->data);

}

void    table\_destroy(Table \*t)

{

vector\_int\_destroy(&t->keys);

vector\_str\_destroy(&t->data);

}

void    table\_add\_entry(Table \*t, int key, char \*val)

{

vector\_int\_pushback(&t->keys, key);

vector\_str\_pushback(&t->data, val);

}

void swap\_int(int \*l, int \*r)

{

int tmp;

tmp = \*l;

\*l = \*r;

\*r = tmp;

}

void swap\_str(char \*\*l, char \*\*r)

{

char \*tmp;

tmp = strdup(\*l);

free(\*l);

\*l = strdup(\*r);

free(\*r);

\*r = strdup(tmp);

free(tmp);

}

void table\_sift\_entry(Table \*t, int idx, int max)

{

int idx\_large,

childl,

childr;

while (idx < max) {

idx\_large = idx;

childl = 2 \* idx + 1;

childr = childl + 1;

if (childl < max &&

vector\_int\_at(&t->keys, childl) > vector\_int\_at(&t->keys, idx\_large))

idx\_large = childl;

if (childr < max &&

vector\_int\_at(&t->keys, childr) > vector\_int\_at(&t->keys, idx\_large))

idx\_large = childr;

if (idx\_large != idx) {

swap\_int(&t->keys.data[idx], &t->keys.data[idx\_large]);

swap\_str(&t->data.data[idx], &t->data.data[idx\_large]);

idx = idx\_large;

} else

break;

}

}

void table\_build\_heap(Table \*t)

{

int n = vector\_int\_length(&t->keys) / 2 - 1;

while (n >= 0) {

table\_sift\_entry(t, n, vector\_int\_length(&t->keys));

--n;

}

}

void    table\_heap\_sort(Table \*t)

{

int end = vector\_int\_length(&t->keys) - 1;

table\_build\_heap(t);

while (end >= 0) {

swap\_int(&t->keys.data[0], &t->keys.data[end]);

swap\_str(&t->data.data[0], &t->data.data[end]);

table\_sift\_entry(t, 0, end);

--end;

}

}

char    \*table\_binary\_search(Table \*t, int key)

{

int left = -1,

right = vector\_int\_length(&t->keys),

middle;

while (left + 1 < right) {

middle = (right + left) / 2;

if (vector\_int\_at(&t->keys, middle) <= key)

left = middle;

else

right = middle;

}

if (vector\_int\_at(&t->keys, left) != key)

return NULL;

return vector\_str\_at(&t->data, left);

}

int table\_entry\_read(Table \*t, FILE \*f)

{

int key;

char val[100000];

char buf[10000];

bzero(buf, 10000);

bzero(val, 100000);

if (fgets(buf, 10000, f) == NULL)

return -1;

key = atoi(buf);

bzero(buf, 10000);

while (fgets(buf, 10000, f)) {

if (strcmp(buf, "\n") == 0)

break;

strcat(val, buf);

bzero(buf, 10000);

}

vector\_int\_pushback(&t->keys, key);

vector\_str\_pushback(&t->data, val);

return 1;

}

void table\_read(Table \*t, FILE \*f)

{

int n;

do {

n = table\_entry\_read(t, f);

} while (n != -1);

}

void    table\_print(Table \*t)

{

for (int i = 0; i < vector\_int\_length(&t->keys); ++i) {

fprintf(stdout, "%d:\n%s\n", vector\_int\_at(&t->keys, i), vector\_str\_at(&t->data, i));

}

fprintf(stdout, "\n");

}

**vector.h**

#ifndef VECTOR\_H\_

#define VECTOR\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define START\_SIZE 5

typedef struct {

char \*\*data;

int size;

int capacity;

} vector\_str;

typedef struct {

int \*data;

int size;

int capacity;

} vector\_int;

void vector\_str\_create(vector\_str \*v);

void vector\_str\_destroy(vector\_str \*v);

void vector\_str\_pushback(vector\_str \*v, char \*str);

int vector\_str\_length(vector\_str \*v);

char        \*vector\_str\_at(vector\_str \*v, int i);

void        vector\_str\_at\_store(vector\_str \*v, int i, char \*data);

void        vector\_str\_print(vector\_str \*v);

void vector\_int\_create(vector\_int \*v);

void vector\_int\_destroy(vector\_int \*v);

void vector\_int\_pushback(vector\_int \*v, int val);

int vector\_int\_length(vector\_int \*v);

int  vector\_int\_at(vector\_int \*v, int i);

void        vector\_int\_at\_store(vector\_int \*v, int i, int data);

void        vector\_int\_print(vector\_int \*v);

#endif

**vector.c**

#include "vector.h"

void        vector\_str\_create(vector\_str \*v)

{

    v->data = (char \*\*)malloc(sizeof(char \*) \* START\_SIZE);

    if (v->data == NULL)

        exit(1);

    v->size = 0;

    v->capacity = START\_SIZE;

}

void vector\_str\_destroy(vector\_str \*v)

{

    for (int i = 0; i < vector\_str\_length(v); ++i) {

        free(vector\_str\_at(v, i));

        vector\_str\_at\_store(v, i, NULL);

    }

    free(v->data);

    v->data = NULL;

    v->size = 0;

    v->capacity = 0;

}

static void vector\_str\_resize(vector\_str \*v, int new\_size)

{

    char \*\*new\_data;

    new\_data = (char \*\*)malloc(sizeof(char \*) \* new\_size);

    for (int i = 0; i < vector\_str\_length(v); ++i) {

        new\_data[i] = strdup(vector\_str\_at(v, i));

    }

    for (int i = 0; i < vector\_str\_length(v); ++i) {

        free(vector\_str\_at(v, i));

        vector\_str\_at\_store(v, i, NULL);

    }

    free(v->data);

    v->data = new\_data;

    v->capacity = new\_size;

}

void vector\_str\_pushback(vector\_str \*v, char \*str)

{

    if (v->size == v->capacity)

        vector\_str\_resize(v, v->size \* 2);

    vector\_str\_at\_store(v, v->size, str);

    v->size++;

}

void        vector\_str\_at\_store(vector\_str \*v, int i, char \*data)

{

    if (data == NULL)

        v->data[i] = NULL;

    else

        v->data[i] = strdup(data);

}

char        \*vector\_str\_at(vector\_str \*v, int i)

{

    return v->data[i];

}

int         vector\_str\_length(vector\_str \*v)

{

return v->size;

}

void        vector\_str\_print(vector\_str \*v)

{

    for (int i = 0; i < vector\_str\_length(v); ++i) {

        printf("%s ", vector\_str\_at(v, i));

    }

}

void        vector\_set\_to\_zero(vector\_int \*v)

{

    for (int i = 0; i < vector\_int\_length(v); ++i) {

        vector\_int\_at\_store(v, i, 0);

    }

}

void        vector\_int\_create(vector\_int \*v)

{

    v->data = (int \*)malloc(sizeof(int) \* START\_SIZE);

    if (v->data == NULL)

        exit(1);

    v->size = 0;

    v->capacity = START\_SIZE;

    vector\_set\_to\_zero(v);

}

void vector\_int\_destroy(vector\_int \*v)

{

    free(v->data);

    v->data = NULL;

    v->size = 0;

    v->capacity = 0;

}

static void vector\_int\_resize(vector\_int \*v, int new\_size)

{

    v->data = realloc(v->data, new\_size \* sizeof(int));

    v->capacity = new\_size;

}

void vector\_int\_pushback(vector\_int \*v, int val)

{

    if (v->size == v->capacity)

        vector\_int\_resize(v, v->size \* 2);

    vector\_int\_at\_store(v, v->size, val);

    v->size++;

}

int         vector\_int\_length(vector\_int \*v)

{

return v->size;

}

void        vector\_int\_at\_store(vector\_int \*v, int i, int data)

{

    v->data[i] = data;

}

int         vector\_int\_at(vector\_int \*v, int i)

{

    return v->data[i];

}

void        vector\_int\_print(vector\_int \*v)

{

    for (int i = 0; i < vector\_int\_length(v); ++i) {

        printf("%d ", vector\_int\_at(v, i));

    }

}

**main.c**

#include <stdio.h>

#include <string.h>

#include "table.h"

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

{

if (argc < 2) {

printf("Using: ./main file\n");

exit(1);

}

FILE \*f = fopen(argv[1], "r");

if (f == NULL) {

printf("File not exist");

exit(1);

}

int i;

char \*res;

Table t;

table\_create(&t);

table\_read(&t, f);

table\_print(&t);

table\_heap\_sort(&t);

table\_print(&t);

i = 2;

while (i < argc) {

res = table\_binary\_search(&t, atoi(argv[i]));

if (res == NULL)

printf("%d: Key not found\n", atoi(argv[i]));

else

printf("%d:\n%s", atoi(argv[i]), res);

++i;

}

table\_destroy(&t);

fclose(f);

return (0);

}

**Демонстрация работы программы**.

➜ KP4 git:(master) ✗ ls

course\_project3.docx check\_memory.sh main.c table.h vector.c ~$ourse\_project3.docx

Makefile file table.c tests vector.h

➜ KP4 git:(master) ✗ make

➜ KP4 git:(master) ✗ ./main tests/test1 2 3 4 5 6 7

-4:

python

-1:

javascript

0:

golang

1:

C++

4:

C

10:

perl

14:

ruby

16:

java

24:

Swift

-4:

python

-1:

javascript

0:

golang

1:

C++

4:

C

10:

perl

14:

ruby

16:

java

24:

Swift

2: Key not found

3: Key not found

4:

C

5: Key not found

6: Key not found

7: Key not found

➜ KP4 git:(master) ✗ ./main tests/test2 2 3 4 5 6 7 -4 24

24:

python

16:

Swift

14:

javascript

10:

java

4:

C++

1:

C

0:

Ruby

-1:

Perl

-4:

Golang

-4:

Golang

-1:

Perl

0:

Ruby

1:

C

4:

C++

10:

java

14:

javascript

16:

Swift

24:

python

2: Key not found

3: Key not found

4:

C++

5: Key not found

6: Key not found

7: Key not found

-4:

Golang

24:

python

➜ KP4 git:(master) ✗ ./main tests/test3 2 3 173 5 45 7 -4 -456

-123:

Ruby

234:

Perl

23:

java

12:

C

173:

Golang

-456:

C++

-11:

javascript

45:

Swift

64:

python

-456:

C++

-123:

Ruby

-11:

javascript

12:

C

23:

java

45:

Swift

64:

python

173:

Golang

234:

Perl

2: Key not found

3: Key not found

173:

Golang

5: Key not found

45:

Swift

7: Key not found

-4: Key not found

-456:

C++

**Вывод**

В данной курсовой работе я реализовал структуру таблицы с пирамидальной сортировкой и бинарным поиском по ключу в таблице.