**Отчет по курсовой работе № 9** по курсу

Практикум программирование

Студент группы: М8О - 110Б - 23, Гула Дмитрий Александрович, № по списку: 6,

Контакты: guladmitriy1331@gmail.com

Работа выполнена: ««18» мая 2024г.

Преподаватель: каф. 806 Почечура А. А.

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан «25» мая 2024 г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Тема:** Сортировка таблицы методом пирамидальной сортировки и двоичный поиск по ключу
2. **Цель работы: Составить программу на языке Си с использованием процедур и функций для сортировки таблицы методом пирамидальной сортировки с просеиванием и выполнения двоичного поиска по ключу.**
3. **Задание**

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.

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.

1. **Оборудование** (лабораторное):

ЭВМ , процессор . имя узла сети с ОП Мб, НМД Мб. Терминал

адрес . Принтер Другие устройства



*Оборудование ПЭВМ студента, если использовалось:*

Процессор \_Intel Core i3-12500H \_ с ОП 8 Гб НМД 256 Гб. Монитор 1920x1080~144Hz Другие устройства



1. **Программное обеспечение** (лабораторное):

Операционная система семейства , наименование версия



интерпретатор команд версия

Система программирования версия

Редактор текстов

версия

Утилиты операционной системы



Прикладные системы и программы



Местонахождение и имена файлов программ и

данных

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства \_Linux\_, наименование \_Ubuntu\_ версия 22.04.2 интерпретатор команд \_GNU bash\_ версия 5.1.16.

Система программирования С. Редактор текстов emacs версия 29.1

Утилиты операционной системы

Прикладные системы и программы Emacs

Местонахождение и имена файлов программ и данных на домашнем компьютере /home/

1. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блоксхема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

**Идея**

Целью данной курсовой работы является разработка программы на языке Си, которая будет сортировать таблицу методом пирамидальной сортировки (с просеиванием) и выполнять двоичный поиск по ключу. Программа должна проверить работу сортировки на трех наборах данных: упорядоченных, отсортированных в обратном порядке и неупорядоченных. После сортировки программа должна выполнить двоичный поиск для заданных ключей и вывести результаты поиска.

**Метод**

Программа состоит из трех основных файлов:

1. Заголовочный файл table.h, содержащий определения структур и прототипы функций.
2. Файл tableFunctions.c, в котором реализованы функции для сортировки и поиска по таблице.
3. Файл main.c, который предоставляет интерфейс пользователя и демонстрирует работу программы.
4. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].
5. **Инициализация таблицы**

При запуске программы будет создана таблица с начальной емкостью 16 элементов. Программа выведет меню для пользователя:

Программа начинает работу

* h - помощь
* 0 - завершить программу
* 1 [key, value] - добавить элемент в таблицу
* 2 - отсортировать таблицу
* 3 - вывести таблицу
* 4 [key] - найти элемент в таблице по ключу

1. **Добавление элементов в таблицу**

Пользователь может выбрать опцию 1 для добавления элемента в таблицу. Он вводит ключ и значение:

1 12.5 34.7

Элемент добавляется в таблицу.

1. **Сортировка таблицы**

Пользователь может выбрать опцию 2 для сортировки таблицы.

Применяется алгоритм Heap Sort.

1. **Вывод таблицы**

Пользователь может выбрать опцию 3 для вывода содержимого таблицы:

|Key |Value |

|##################################|################################|

12.500000 34.700000

1. **Поиск элемента по ключу**

Пользователь может выбрать опцию 4 для поиска элемента по ключу:

4 12.5

Индекс элемента с ключём 12.500000: 0

1. **Завершение программы**

Пользователь может выбрать опцию 0 для завершения программы. Программа освобождает выделенную память и завершает работу.

Таким образом, программа предоставляет пользователю простой интерфейс для работы с таблицей, позволяя добавлять, сортировать, выводить и искать элементы.

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

#ifndef C\_TABLESTRUCT\_H  
#define C\_TABLESTRUCT\_H  
  
#include <stdio.h>  
#include <stdbool.h>  
#include <stdlib.h>  
#include <assert.h>  
#include <string.h>  
  
  
typedef struct Key Key;  
typedef Key\* Key\_handle;  
typedef struct Value Value;  
typedef Value\* Value\_handle;  
typedef struct Table Table;  
typedef Table\* Table\_handle;  
  
struct Key {  
 double dbl;  
};  
  
struct Value {  
 double el\_value;  
};  
  
struct Table {  
 Key \*data\_keys;  
 Value \*data\_values;  
 int current\_size;  
 int capacity;  
 bool isSorted;  
};  
  
  
Table\_handle table\_init(int cap);  
bool is\_table\_empty(Table\_handle table);  
void destroy\_table(Table\_handle table);  
void grow\_buff(Table\_handle table);  
int string\_len(double value);  
char\* add\_spaces(int n);  
void \* push\_elem\_to\_table(Table\_handle table, char \*key\_and\_value);  
void print\_table(Table\_handle table);  
void print\_line\_by\_index(Table\_handle table, int index);  
double get\_value\_by\_index(Table\_handle table, int index);  
double get\_key\_by\_index(Table\_handle table, int index);  
int keycmp(Table\_handle table, int index1, int index2);  
void swap\_table\_elements(Table\_handle table, int index1, int index2);  
void heapify(Table\_handle table, int data\_size, int i);  
void heapSort(Table\_handle table, int data\_size);  
int search\_elem(Table\_handle table, double required\_key);  
  
#endif //C\_TABLESTRUCT\_H

#ifndef C\_INTERFACE\_H  
#define C\_INTERFACE\_H  
#include "TableStruct.h"  
  
  
void print\_menu();  
char\* create\_double\_string(double double1, double double2);  
void push\_elem(Table\_handle table, double key, double value);  
int find\_elem\_by\_key(Table\_handle table, double key);  
  
#endif //C\_INTERFACE\_H

#include "TableStruct.h"  
  
  
Table\_handle table\_init(int cap){  
 Table\_handle table = malloc(sizeof (Table));  
 table->capacity = cap;  
 table->current\_size = 0;  
 table->isSorted = false;  
 table->data\_keys = malloc(sizeof (Key) \* cap);  
 table->data\_values = malloc(sizeof (Value) \* cap);  
 return table;  
}  
  
bool is\_table\_empty(Table\_handle table){  
 return table->current\_size == 0;  
}  
  
void destroy\_table(Table\_handle table){  
 free(table->data\_keys);  
 free(table->data\_values);  
 free(table);  
}  
  
void grow\_buff(Table\_handle table){  
 table->capacity \*= 2;  
 table->data\_keys = realloc(table->data\_keys, sizeof (Table) \* table->capacity);  
 table->data\_values = realloc(table->data\_values, sizeof (Table) \* table->capacity);  
 if (table->data\_keys == NULL && table->data\_values == NULL) {  
 perror("Ошибка в перезаписывание памяти");  
 exit(EXIT\_FAILURE);  
 }  
}  
  
void \* push\_elem\_to\_table(Table\_handle table, char \*key\_and\_value){  
 if(table->capacity <= table->current\_size + 1){  
 grow\_buff(table);  
 }  
 int result = sscanf(key\_and\_value, "%lf/%lf", &table->data\_keys[table->current\_size].dbl, &table->data\_values[table->current\_size].el\_value);  
 if (result != 2) {  
 printf("Неправильно введены значения\n");  
 return NULL;  
 }  
 table->current\_size += 1;  
 table->isSorted = false;  
}  
  
int string\_len(double value){  
 char buffer[64];  
 int length = snprintf(buffer, sizeof(buffer), "%f", value);  
 if (length < 0) {  
 perror("snprintf");  
 return 1;  
 }  
 return length;  
}  
  
  
char\* add\_spaces(int n) {  
 char \*space\_string = malloc(n + 1);  
 if (space\_string == NULL) {  
 fprintf(stderr, "Ошибка выделения памяти\n");  
 exit(1);  
 }  
 for (int i = 0; i < n; i++) {  
 space\_string[i] = ' ';  
 }  
 space\_string[n] = '\0';  
 return space\_string;  
}  
  
  
  
void print\_table(Table\_handle table){  
 if(is\_table\_empty(table)){  
 printf("Таблица пустая\n");  
 return;  
 }  
 printf("|Key |Value |\n");  
 printf("|##################################|################################|\n");  
 int table\_size = table->current\_size;  
 for(int i = 0; i < table\_size; i++){  
 int key\_len = 34 - string\_len(table->data\_keys[i].dbl);  
 char \*spaces = add\_spaces(key\_len);  
 printf("%lf %s %lf\n", table->data\_keys[i].dbl, spaces, table->data\_values[i].el\_value);  
 }  
 printf("\n");  
}  
  
void print\_line\_by\_index(Table\_handle table, int index){  
 if(index < 0){  
 printf("Индекс должен быть больше нуля\n");  
 return;  
 }  
 Key\_handle key = &table->data\_keys[index];  
 Value\_handle value = &table->data\_values[index];  
 int key\_len = 34 - string\_len(key->dbl);  
 char \*spaces = add\_spaces(key\_len);  
 printf("|Key |Value |\n");  
 printf("|##################################|################################|\n");  
 printf("%lf %s %lf\n",key->dbl, spaces, value->el\_value);  
 printf("\n");  
}  
  
double get\_value\_by\_index(Table\_handle table, int index){  
 return table->data\_values[index].el\_value;  
}  
  
double get\_key\_by\_index(Table\_handle table, int index){  
 return table->data\_keys[index].dbl;  
}  
  
int keycmp(Table\_handle table, int index1, int index2){  
 if(table->data\_keys[index1].dbl > table->data\_keys[index2].dbl){  
 return -1;  
 }else if(table->data\_keys[index1].dbl < table->data\_keys[index2].dbl){  
 return 1;  
 }  
 return 0;  
}  
  
void swap\_table\_elements(Table\_handle table, int index1, int index2) {  
 Value temp\_value = table->data\_values[index1];  
 Key temp\_keys = table->data\_keys[index1];  
 table->data\_keys[index1] = table->data\_keys[index2];  
 table->data\_values[index1] = table->data\_values[index2];  
 table->data\_keys[index2] = temp\_keys;  
 table->data\_values[index2] = temp\_value;  
}  
  
void heapify(Table\_handle table, int data\_size, int i){  
 int largest = i;  
 int left = 2 \* i + 1;  
 int right = 2 \* i + 2;  
 if(left < data\_size && keycmp(table, left, largest) == -1){  
 largest = left;  
 }  
 if(right < data\_size && keycmp(table, right, largest) == -1){  
 largest = right;  
 }  
 if(largest != i){  
 swap\_table\_elements(table, i, largest);  
 heapify(table, data\_size, largest);  
 }  
}  
  
void heapSort(Table\_handle table, int data\_size){  
 for(int i = data\_size / 2 - 1; i >= 0; i--){  
 heapify(table, data\_size, i);  
 }  
 for (int i = data\_size - 1; i >= 0; i--){  
 swap\_table\_elements(table, 0, i);  
 heapify(table, i, 0);  
 }  
 table->isSorted = true;  
}  
  
int search\_elem(Table\_handle table, double required\_key) {  
 int left = 0;  
 int right = table->current\_size - 1;  
 int middle;  
  
 while (left <= right) {  
 middle = (left + right) / 2;  
  
 if (table->data\_keys[middle].dbl < required\_key) {  
 left = middle + 1;  
 } else if (table->data\_keys[middle].dbl > required\_key) {  
 right = middle - 1;  
 } else {  
 return table->data\_values[middle].el\_value;  
 }  
 }  
  
 return -1;  
}

#include "Interface.h"  
  
void print\_menu(){  
 printf("h - помощь\n");  
 printf("0 - завершить программу\n");  
 printf("1 [key, value] - добавить элемент в таблицу\n");  
 printf("2 - отсортировать таблицу\n");  
 printf("3 - вывести таблицу\n");  
 printf("4 [key] - найти элемемент в таблице по ключу\n");  
}  
  
  
char\* create\_double\_string(double double1, double double2){  
 int buffer\_size = 60;  
 char \*result = (char \*)malloc(buffer\_size \* sizeof(char));  
 if (result == NULL) {  
 fprintf(stderr, "Ошибка выделения памяти\n");  
 exit(1);  
 }  
 snprintf(result, buffer\_size, "%.2f/%.2f", double1, double2);  
 return result;  
}  
  
void push\_elem(Table\_handle table, double key, double value){  
 char \*key\_and\_value = create\_double\_string(key, value);  
 push\_elem\_to\_table(table, key\_and\_value);  
}  
  
int find\_elem\_by\_key(Table\_handle table, double key){  
 if (table->isSorted == false){  
 printf("Таблица не была отсортирована\n");  
 return -1;  
 }else{  
 return search\_elem(table, key);  
 }  
}

#include "TableStruct.h"  
#include "Interface.h"  
  
int main(){  
 printf("Программа начинает работу\n");  
 Table\_handle table = table\_init(16);  
 char command;  
 print\_menu();  
 while (scanf(" %c", &command) != EOF) {  
 switch (command) {  
 case 'h': {  
 print\_menu();  
 break;  
 }  
 case '0':{  
 printf("Завершаем программу...");  
 destroy\_table(table);  
 exit(0);  
 }  
 case '1': {  
 double key;  
 double value;  
 scanf("%lf %lf", &key, &value);  
 push\_elem(table, key, value);  
 break;  
 }  
 case '2': {  
 heapSort(table, table->current\_size);  
 break;  
 }  
 case '3': {  
 print\_table(table);  
 break;  
 }  
 case '4': {  
 double key;  
 scanf("%lf", &key);  
 if(find\_elem\_by\_key(table, key) == -1){  
 printf("Элементов с таким ключём нет\n");  
 break;  
 }else{  
 printf("Индекс элемента с ключём %lf: %d\n", key, find\_elem\_by\_key(table, key));  
 }  
 break;  
 }  
 default:  
 printf("Ошибка: Неверный выбор меню. Попробуйте еще раз.\n");  
 }  
 }  
}

C:\Users\Дима\CLionProjects\C\kp9\cmake-build-debug\kp9.exe

Программа начинает работу

h - помощь

0 - завершить программу

1 [key, value] - добавить элемент в таблицу

2 - отсортировать таблицу

3 - вывести таблицу

4 [key] - найти элемемент в таблице по ключу

р

Ошибка: Неверный выбор меню. Попробуйте еще раз.

Ошибка: Неверный выбор меню. Попробуйте еще раз.

h

h - помощь

0 - завершить программу

1 [key, value] - добавить элемент в таблицу

2 - отсортировать таблицу

3 - вывести таблицу

4 [key] - найти элемемент в таблице по ключу

1 1 1

1 2 2

1 3 3

3

|Key |Value |

|##################################|################################|

1.000000 1.000000

2.000000 2.000000

3.000000 3.000000

1 0.2 1323

1 0.01 32

1 13213 32131

1 111 111

3

|Key |Value |

|##################################|################################|

1.000000 1.000000

2.000000 2.000000

3.000000 3.000000

0.200000 1323.000000

0.010000 32.000000

13213.000000 32131.000000

111.000000 111.000000

h

h - помощь

0 - завершить программу

1 [key, value] - добавить элемент в таблицу

2 - отсортировать таблицу

3 - вывести таблицу

4 [key] - найти элемемент в таблице по ключу

4 111

Таблица не была отсортирована

Элементов с таким ключём нет

2

3

|Key |Value |

|##################################|################################|

0.010000 32.000000

0.200000 1323.000000

1.000000 1.000000

2.000000 2.000000

3.000000 3.000000

111.000000 111.000000

13213.000000 32131.000000

4 3

Индекс элемента с ключём 3.000000: 3

4 111

Индекс элемента с ключём 111.000000: 111

4 13213

Индекс элемента с ключём 13213.000000: 32131

h

h - помощь

0 - завершить программу

1 [key, value] - добавить элемент в таблицу

2 - отсортировать таблицу

3 - вывести таблицу

4 [key] - найти элемемент в таблице по ключу

3

|Key |Value |

|##################################|################################|

0.010000 32.000000

0.200000 1323.000000

1.000000 1.000000

2.000000 2.000000

3.000000 3.000000

111.000000 111.000000

13213.000000 32131.000000

0

Завершаем программу...

Process finished with exit code 0

1. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб.  или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы



1. **Выводы**

В данной работе была разработана и отлажена программа на языке Си для обработки таблицы. Программа успешно выполняет заданные действия: печать списка, вставка нового элемента, сортировка таблицы, вывод таблицы, а также поиск элемемента в таблице по ключу.

Подпись студента