Министерство образования и науки Российской Федерации

Федеральное государственное образовательное учреждение высшего профессионального образования

«Московский авиационный институт»

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

2 семестр

Курсовой проект

По курсу «Алгоритмы и структуры данных»

Задание IX

Работ сдал: студент группы М8О-204Б-22

Филиппов Фёдор Иванович

Работу принял: преподаватель информатики

Потенко Максим Алексеевич

Москва, 2023

Оглавление

Цель работы

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.

Задание

Метод сортировки: Быстрая сортировка Хоара (нерекурсивный вариант).

Тип ключа таблицы: Целый.

Теоретическая справка

Быстрая сортировка Хоара

Быстрая сортировка Хоара (или просто быстрая сортировка) — это один из самых эффективных алгоритмов сортировки, который базируется на принципе "разделяй и властвуй". Алгоритм быстрой сортировки нерекурсивным методом, также известный как итеративная быстрая сортировка, представляет собой модификацию классической рекурсивной реализации алгоритма, использующую стек для хранения границ подмассивов.

Основная идея быстрой сортировки заключается в следующем:

1. Выбирается опорный элемент из массива. Обычно в качестве опорного элемента выбираются первый, последний или средний элемент массива.

2. Массив разбивается на две подгруппы: элементы, которые меньше или равны опорному, и элементы, которые больше опорного.

3. Рекурсивно применяется алгоритм быстрой сортировки к каждой подгруппе.

4. Результаты сливаются в один отсортированный массив.

Итеративная реализация быстрой сортировки использует стек для хранения границ подмассивов, которые нужно отсортировать. Вместо рекурсивных вызовов функции быстрой сортировки, стек позволяет сохранять информацию о подмассивах, которые нужно обработать в последующих итерациях цикла. Это позволяет избежать переполнения стека вызовов и снизить накладные расходы, связанные с рекурсией.

Основные шаги итеративной быстрой сортировки:

1. Создается стек и помещается в него начальные границы всего массива.

2. Пока стек не пуст, выполняются следующие действия:

    - Извлекается верхний элемент стека, содержащий границы подмассива.

    - Выбирается опорный элемент в соответствии с выбранным правилом.

    - Массив разбивается на две подгруппы вокруг опорного элемента.

- Границы каждой подгруппы помещаются в стек для последующей обработки.

3. По окончании сортировки стек будет пустым, а исходный массив будет отсортирован.

Итеративная быстрая сортировка Хоара обычно работает быстрее рекурсивной реализации за счет снижения накладных расходов, связанных с рекурсией, и более эффективного использования кэша процессора.

Бинарный поиск

Бинарный поиск — это эффективный алгоритм поиска элемента в отсортированном массиве или списке. Он также основан на принципе "разделяй и властвуй" и позволяет быстро находить элемент, проверяя только небольшую часть данных на каждом шаге.

Принцип работы бинарного поиска:

1. На первом шаге определяются границы поиска. Исходный массив или список должен быть предварительно отсортирован в порядке возрастания или убывания.

2. Устанавливается начальная левая граница (left) и правая граница (right) в соответствии с размером массива или списка.

3. На каждой итерации бинарного поиска вычисляется средний индекс (mid) путем деления суммы left и right на 2 (mid = (left + right) / 2).

4. Сравнивается искомый элемент с элементом, находящимся в середине массива или списка.

    - Если искомый элемент равен элементу в середине, поиск завершается, и индекс найденного элемента возвращается.

- Если искомый элемент меньше элемента в середине, правая граница сдвигается на mid - 1.

- Если искомый элемент больше элемента в середине, левая граница сдвигается на mid + 1.

5. Процесс повторяется, сокращая интервал поиска в два раза на каждой итерации, пока искомый элемент не будет найден или пока границы не сойдутся (left > right).

6. Если искомый элемент не найден, возвращается значение, указывающее на отсутствие элемента (обычно -1 или другое специальное значение).

Бинарный поиск является эффективным алгоритмом благодаря своей способности быстро сокращать интервал поиска вдвое на каждом шаге. Это позволяет быстро находить элемент даже в больших массивах или списках. Он имеет временную сложность O(log n), где n - количество элементов в массиве или списке.

Используемое оборудование

ЭВМ iMac 2012 Late 2012 21.5’

Процессор Intel Core i5 4 ядра

ОП 16324 Мб

НМД 524288 Мб

Реализация программы

Описание функции spisok.c

table* read_table(FILE* fd): Эта функция считывает данные из файла, где каждая строка содержит ключ (целое число) и строку. Она создает и возвращает указатель на таблицу table, заполнив ее элементами, ключами и строками из файла. Таблица также содержит информацию о типе сортировки (возрастающей, убывающей или несортированной).

elem* create_elem(long long key, char* line): Эта функция создает элемент elem, который содержит целочисленный ключ и строку. Она возвращает указатель на созданный элемент.

elem* search(table* t, long long key): Эта функция выполняет бинарный поиск элемента в таблице t по заданному ключу key. Она возвращает указатель на элемент, если ключ найден, иначе возвращает NULL.

void swap(elem a, elem** b):** Эта функция обменивает два указателя на элементы a и b. Она используется для сортировки элементов в таблице.

void sort(table* t): Эта функция выполняет сортировку элементов в таблице t методом чётно-нечётной сортировки по ключу. Она изменяет порядок элементов в таблице таким образом, чтобы они были упорядочены в

возрастающем порядке по ключу.

void print_table(table* t): Эта функция выводит содержимое таблицы t на стандартный вывод (консоль). Каждая строка содержит ключ и соответствующую строку из элемента таблицы.

void free_elem(elem* e): Эта функция освобождает память, занимаемую элементом e, включая строку и сам элемент.

void free_table(table* t): Эта функция освобождает память, занимаемую таблицей t, включая элементы и строки, а также саму таблицу.

Распечатка программы

Table.h

```
#ifndef TABLE
#define TABLE

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_LEN 250
#define LIST_SIZE 20

typedef struct elem {
    long long key;
    char* line;
```

```
} elem;

typedef enum {
    NOT,
    UP,
    DOWN,
} sorting;

typedef struct table {
    int size;
    elem** list;
    sorting sorting;
} table;

table* read_table(FILE* fd);

elem* create_elem(long long key, char* line);

elem* search(table* t, long long key);

void swap(elem** a, elem** b);

void sort(table* t);

void print_table(table* t);

void free_elem(elem* e);

void free_table(table* t);
```

```
#endif
```

Table.c

```
#include "table.h"


table* read_table(FILE* fd) {
    table* t = (table*)malloc(sizeof(table));
    t->size = 0;
    t->list = (elem**)malloc(sizeof(elem*) * LIST_SIZE);
    t->sorting = NOT;

    bool unsorted = false;
    long long key;
    long long prev_key = INT64_MIN;
    char buffer[MAX_LEN];
    while (fscanf(fd, "%lld", &key) && fgets(buffer, MAX_LEN, fd) != NULL) {
        char* line = malloc(MAX_LEN * sizeof(char));
        strncpy(line, buffer, MAX_LEN);
        t->list[t->size] = create_elem(key, line);
        t->size++;

        // В первую итерацию не проверяем тип сортировки
        if (prev_key == INT64_MIN) {
            prev_key = key;
            continue; // Пропускаем дальше
        }
        // Проверяем тип отсортированности
        if (key > prev_key && t->sorting != DOWN && !unsorted) {
            t->sorting = UP;
```

```
        } else if (key < prev_key && t->sorting != UP && !unsorted) {
            t->sorting = DOWN;
        } else {
            t->sorting = NOT;
            unsorted = true;
        }

        prev_key = key;
    }
    return t;
}

elem* create_elem(long long key, char* line) {
    elem* e = (elem*)malloc(sizeof(elem));
    e->key = key;
    e->line = line;
    return e;
}

// Бинарный поиск в зависимости от типа отсортированности
elem* search(table* t, long long key) {
    int left = 0;
    int right = t->size - 1;
    sorting type = t->sorting;

    while (left <= right) {
        int middle = left + (right - left) / 2;

        if (t->list[middle]->key == key) {
```

```
            return t->list[middle];
        }
        if ((type == UP && t->list[middle]->key < key)
        || (type == DOWN && t->list[middle]->key > key))
        {
            left = middle + 1;
        }
        else {
            right = middle - 1;
        }
    }

    return NULL; // Key not found
}

// Функция для обмена элементов
void swap(elem** a, elem** b) {
    elem* temp = *a;
    *a = *b;
    *b = temp;
}

// Функция для чётно-нечётной сортировки по ключу
void sort(table* t) {
    int n = t->size;
    int sorted = 0; // Флаг, указывающий на то, был ли произведен обмен на последней итерации

    while (!sorted) {
```

```
        sorted = 1;

        // Проход по четным элементам и сравнение соседних ключей
        for (int i = 0; i < n - 1; i += 2) {
            if (t->list[i]->key > t->list[i + 1]->key) {
                swap(&(t->list[i]), &(t->list[i + 1]));
                sorted = 0; // Обмен произведен, таблица может быть не отсортированной
            }
        }

        // Проход по нечетным элементам и сравнение соседних ключей
        for (int i = 1; i < n - 1; i += 2) {
            if (t->list[i]->key > t->list[i + 1]->key) {
                swap(&(t->list[i]), &(t->list[i + 1]));
                sorted = 0; // Обмен произведен, таблица может быть не отсортированной
            }
        }
    }
    t->sorting = UP;
}

void print_table(table* t) {
    if (t == NULL) {
        return;
    }
    for (int i = 0; i < t->size; i++) {
        printf("%lld %s", t->list[i]->key, t->list[i]->line);
    }
}
```

```
void free_elem(elem* e) {
    if (e == NULL) {
        return;
    }
    free(e->line);
    free(e);
    return;
}

void free_table(table* t) {
    for (int i = 0; i < t->size; i++) {
        free_elem(t->list[i]);
    }
    free(t->list);
    free(t);
}
```

Main.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include "table.h"

int main(int argc, char** argv) {
    if (argc != 2 || argv[1] == NULL) {
        printf("Использование: lab <файл>\n");
```

```
    exit(1);
}

FILE* fd = fopen(argv[1], "r");
if (fd == NULL) {
    perror("Ошибка открытия файла");
    exit(1);
}

table* t = read_table(fd);
table* t_sorted = NULL;

// Если стих неотсортирован
if (t->sorting == NOT) {
    // Создаем отсортированную копию стиха
    t_sorted = (table*)malloc(sizeof(table));
    t_sorted->size = t->size;
    t_sorted->list = (elem**)malloc(sizeof(elem*) * t->size);
    for (int i = 0; i < t->size; i++) {
        t_sorted->list[i] = create_elem(t->list[i]->key, strdup(t->list[i]->line));
    }

    sort(t_sorted);
} else {
    t_sorted = t;
}

while (1) {
    long long key;
```

```
int choice = 0;
printf("\n    1. Вывести изначальный стих\n");
printf("    2. Вывести отсортированный стих\n");
printf("    3. Поиск по ключу\n");
printf("    4. Выйти\n");

scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Стих:\n");
        print_table(t);
        break;
    case 2:
        printf("Отсортированный стих:\n");
        print_table(t_sorted);
        break;
    case 3:
        printf("Введите ключ: ");
        scanf("%lld", &key);
        elem* e = search(t_sorted, key);
        if (e != NULL) {
            printf("Строка: %s", e->line);
        } else {
            printf("Ключа не существует");
        }
        break;
    case 4:
        if (t->sorting == NOT) {
```

```
                free(t_sorted);
            }
            free_table(t);
            return 0;
            break;
        default:
            printf("Некорректный ввод\n");
            break;
    }
  }

  return 0;
}
```

Результат работы программы

```
    1. Вывести изначальный стих
    2. Вывести отсортированный стих
    3. Поиск по ключу
    4. Выйти
1
Стих:
12  И редкий солнца луч, и первые морозы,
11  И мглой волнистою покрыты небеса,
1  Унылая пора! Очей очарованье!
3  Приятна мне твоя прощальная краса –
6  В багрец и в золото одетые леса,
13  И отдаленные седой зимы угрозы.
8  В их сенях ветра шум и свежее дыханье,
5  Люблю я пышное природы увяданье,

    1. Вывести изначальный стих
    2. Вывести отсортированный стих
    3. Поиск по ключу
    4. Выйти
2
Отсортированный стих:
1  Унылая пора! Очей очарованье!
3  Приятна мне твоя прощальная краса –
5  Люблю я пышное природы увяданье,
6  В багрец и в золото одетые леса,
8  В их сенях ветра шум и свежее дыханье,
11  И мглой волнистою покрыты небеса,
12  И редкий солнца луч, и первые морозы,
13  И отдаленные седой зимы угрозы.

    1. Вывести изначальный стих
    2. Вывести отсортированный стих
    3. Поиск по ключу
    4. Выйти
3
Введите ключ: 3
Строка:  Приятна мне твоя прощальная краса –

    1. Вывести изначальный стих
    2. Вывести отсортированный стих
    3. Поиск по ключу
    4. Выйти
```

Вывод

В ходе выполнения данной лабораторной работы мною была составлена и отлажена программа на языке С, реализующая сортировку таблицы с использованием метода быстрой сортировки Хоара в нерекурсивном варианте, а также двоичный поиск по ключу в таблице.

Хочется отметить, что данная реализация быстрой сортировки позволила мне лучше понять, как использовать стек во избежание рекурсии функций.

Знания, полученные в ходе выполнения данной работы, будут полезны мне в дальнейшем при разработке программ, требующих упорядочивания данных и быстрого поиска. Теперь я обладаю навыками реализации алгоритмов сортировки и поиска, которые можно применять в различных областях программирования и обработки данных.

Использованные источники

1. https://ru.wikipedia.org/wiki/Быстрая_сортировка
2. https://developerc.github.io/hoar.html
3. https://medium.com/nuances-of-programming/введение-в-бинарный-поиск-715e93b7efd2