Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Низкоуровневое программирование

Отчет по лабораторной работе №5
Двоичная куча

Работу выполнил:
Кечин В.В.
Группа:
3530901/90004
Преподаватель:
Алексюк А.О.

Санкт-Петербург
2021

Содержание

1. Цель работы

- Разработать статическую библиотеку, реализующую бинарную кучу.
- Разработать демонстрационную программу – консольное приложение, обеспечивающее ввод данных из файла, их обработку и вывод в файл; имена файлов передаются в качестве параметров командной строки.

2. Программа работы

- Прочитать, что такое двоичная куча, понять ее устройство и основные алгоритмы работы.
- Создать статическую библиотеку, реализующую двоичную кучу, на языке C.
- Создать консольное приложение, демонстрирующее работу библиотеки, и тесты к ней.

3. Двоичная куча

Двоичная куча - это такое двоичное дерево, для которого выполнены условия:

1. Значение в любой вершине не меньше, чем значения её потомков.
2. Глубина всех листьев (расстояние до корня) отличается не более чем на 1 слой.
3. Последний слой заполняется слева направо без «дырок».

Удобная структура данных для сортирующего дерева — массив A, у которого корневой элемент — A[0], а потомки элемента A[i] — A[2i+1] и A[2i+2]. При таком способе хранения условия 2 и 3 выполнены автоматически.

Над кучей можно выполнять следующие операции:

1. Добавить элемент в кучу. Сложность O(log n)
2. Исключить максимальный элемент из кучи. Время работы O(log n)
3. Изменить значение любого элемента. Время работы O(log n)

На основе этих операций можно выполнять следующие действия:

1. Превратить неупорядоченный массив элементов в кучу. Сложность O(n)
2. Отсортировать массив путём превращения его в кучу, а кучу в отсортированный массив. Время работы O(n log n)

4. Выполнение работы

Для начала создадим заголовочный файл для описания методов двоичной кучи и реализуем их.

```c
//heap.h
#ifndef UNTITLED_HEAP_H
#define UNTITLED_HEAP_H


void heapify(int *, int, int);

void increase(int *, int, int);

int *insert(int *, int, int);

int *build(int *, int);

void sort(int *, int);

int extract_max(int *, int);

#endif
```

```c
//heap.c
#include "heap.h"
#include <malloc.h>
#include <string.h>
#include <limits.h>


void heapify(int *arr, int size, int i) {
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    int largest = i;
    if (left < size && arr[left] > arr[largest]) largest = left;
    if (right < size && arr[right] > arr[largest]) largest = right;
    if (largest != i) {
        int v = arr[i];
        arr[i] = arr[largest];
        arr[largest] = v;
        heapify(arr, size, largest);
    }
}

void increase(int *arr, int i, int key) {
    if (key < arr[i]) return;
    arr[i] = key;
    while (i > 0 && arr[i / 2] < arr[i]) {
        int v = arr[i / 2];
        arr[i / 2] = arr[i];
        arr[i] = v;
        i = i / 2;
    }
}

int *insert(int *arr, int size, int element) {
    size++;
    arr = (int *) realloc(arr, (size) * sizeof(int));
    arr[size - 1] = INT_MIN;
    increase(arr, size - 1, element);
    return arr;
}


```

```
42 int *build(int *data, int size) {
43     int *arr = (int *) malloc(sizeof(int) * size);
44     memcpy(&arr, &data, size);
45     for (int i = size / 2; i >= 0; i--) {
46         heapify(arr, size, i);
47     }
48     return arr;
49 }
50
51 void sort(int *arr, int size) {
52     int s = size;
53     for (int i = s - 1; i >= 0; i--) {
54         int v = arr[i];
55         arr[i] = arr[0];
56         arr[0] = v;
57         s--;
58         heapify(arr, s, 0);
59     }
60 }
61
62 int extract_max(int *arr, int size) {
63     int max = arr[0];
64     arr[0] = arr[size - 1];
65     size--;
66     heapify(arr, size, 0);
67     return max;
68 }
```

Краткое описание реализованных методов:

- heapify - принимает на вход указатель массива, его размер и индекс элемента. Данный метод предназначен восстанавления свойства упорядоченности во всём поддереве, корнем которого является элемент A[i].
- increase - принимает на вход указатель массива, индекс элемента и новое значение элемента. Используется для добавления произвольного элемента в кучу.
- insert - принимает на вход указатель массива, его размер и значение нового элемента. Добавляет произвольный элемент в конец кучи и восстанавливает свойство упорядоченности в куче.
- build - принимает на вход указатель массива и его размер. Эта процедура предназначена для создания кучи из неупорядоченного массива входных данных.
- sort - принимает на вход указатель массива и его размер. Сортирует массив.
- extract max - принимает на вход указатель массива и его размер. Извлекает из массива максимальный элемент и затем восстанавливает свойство упорядоченности в куче.

Теперь реализуем тесты для нашей могучей кучки.

```
1 //tests.c
2 #include <assert.h>
3 #include <malloc.h>
4 #include <stdlib.h>
5 #include <limits.h>
6
7
```

```
 8 int *createArr(int size) {
 9     int *arr = (int *) malloc(sizeof(int) * size);
10     for (int i = 0; i < size; ++i) {
11         arr[i] = rand() % 20;
12     }
13     return arr;
14 }
15
16 void test_build() {
17     int flag = 0;
18     int size = 10;
19     int *data = createArr(size);
20     int *arr = build(data, size);
21     for (int i = 0; i < size; ++i) {
22         if (2 * i + 2 < size) if (arr[i] < arr[2 * i + 2]) flag = -1;
23         if (2 * i + 1 < size) if (arr[i] < arr[2 * i + 1]) flag = -1;
24     }
25     assert(flag == 0);
26     free(arr);
27     free(data);
28 }
29
30 void test_sort() {
31     int flag = 0;
32     int size = 10;
33     int *data = createArr(size);
34     int *arr = build(data, size);
35     sort(arr, size);
36     for (int i = 1; i < size; ++i) {
37         if (arr[i - 1] > arr[i]) flag = -1;
38     }
39
40     assert(flag == 0);
41     free(arr);
42     free(data);
43 }
44
45 void test_add() {
46     int flag = -1;
47     int size = 10;
48     int *data = createArr(size);
49     int *arr = build(data, size);
50
51     arr = insert(arr, size, 10101);
52     size++;
53     for (int i = 0; i < size; ++i) {
54         if (arr[i] == 10101) {
55             flag = 0;
56             break;
57         }
58     }
59     assert(flag == 0);
60
61     for (int i = 0; i < size; ++i) {
62         if (2 * i + 1 < size) if (arr[i] < arr[2 * i + 1]) flag = -1;
63         if (2 * i < size) if (arr[i] < arr[2 * i]) flag = -1;
64     }
65     assert(flag == 0);
66
67     free(arr);
```

```
68      free(data);
69  }
70
71  void test_max() {
72      int flag = 0;
73      int size = 10;
74      int *data = createArr(size);
75      int *arr = build(data, size);
76      int orig_max = INT_MIN;
77      for (int i = 0; i < size; ++i) {
78          if (data[i] > orig_max) orig_max = data[i];
79      }
80      int new_max = extract_max(arr, size);
81      assert(new_max == orig_max);
82      size--;
83      for (int i = 0; i < size; ++i) {
84          if (arr[i] == new_max) {
85              flag = -1;
86              break;
87          }
88      }
89      assert(flag == 0);
90      free(arr);
91      free(data);
92  }
```

Запустив тесты из метода main, легко убедиться, что двоичная куча реализована верно. Теперь добавим чтение и запись в файлы, путь к которым будет передаваться в аргументах программы.

```
1   //main.c
2   #include "heap.h"
3   #include <stdio.h>
4   #include <malloc.h>
5   #include "tests.c"
6
7
8   int numberCount(FILE *file) {
9       fseek(file, 0, SEEK_SET);
10      int count = 0;
11      while (1) {
12          int value;
13          if (fscanf(file, "%d", &value) == 1)
14              count++;
15          if (feof(file))
16              return count;
17      }
18  }
19
20  void read(FILE *file, int size, int *data) {
21      fseek(file, 0, SEEK_SET);
22      for (int i = 0; i < size; ++i) {
23          fscanf(file, "%d", &data[i]);
24      }
25  }
26
27  void tests(){
28      test_sort();
29      test_max();
30      test_add();
31      test_build();
```

```
32 }
33
34 int main(int argc, char **argv) {
35     if (argc != 3) return -1;
36     FILE *fin = fopen(argv[1], "r");
37     int size = numberCount(fin);
38     int *data = (int *) malloc(sizeof(int) * size);
39     read(fin, size, data);
40     fclose(fin);
41     int *arr = build(data, size);
42     for (int i = 0; i < size; ++i) {
43         printf("%d\n", arr[i]);
44     }
45     FILE *fout = fopen(argv[2], "w");
46     for (int i = 0; i < size; ++i) {
47         fprintf(fout, "%d\n", arr[i]);
48     }
49     fclose(fout);
50     free(arr);
51     //tests();
52     return 0;
53 }
```

И наконец, вынесем нашу структуру данных в статическую библиотеку.

```
1 #CMakeLists.txt
2 cmake_minimum_required(VERSION 3.6)
3 project(untitled C)
4
5 set(CMAKE_C_STANDARD 99)
6 add_library(heap STATIC heap.c heap.h)
7 add_executable(untitled main.c)
8 TARGET_LINK_LIBRARIES(untitled heap)
```

Выплнив код, убеждаемся в его работоспособности.

5. Выводы

Реализована двоичная куча на языке C. Структура данных вынесена в статическую библиотеку. Создано консольное приложение, позволяющие считывать массив из входного файла и записывать в выходной файл уже бинарную кучу. Также реализованы и выполнены юнит тесты для проверки работы двоичной кучи. Получен опыт работы на языке C.