МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Слабая куча

Студент гр. 9383 _____ Корсунов А.А.

Преподаватель _____ Попова Е.В.

Санкт-Петербург
2020

Цель работы.

Познакомиться и научиться работать со слабой кучей и написать программу на языке программирования C++.

Основные теоретические положения.

Сортировка — последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.

Куча — это специальная форма данных по типу дерева, удовлетворяющая условиям кучи - если B есть узел-потомок узла A, то ключ(A) ≥ ключ(B).

Слабая куча — куча по типу бинарного полного дерева, у которой только один из потомков удовлетворяет условию кучи (правый потомок), левый же потомок может быть как меньше, так и больше родителя. У корневого узла кучи нет левого потомка, потому как корневой узел должен быть максимальным, что исключает возможность левого потомка такого узла быть больше своего родителя.

Задание.

32. Дан массив чисел. Проверить, что он представляет слабую кучу.

Ход работы.

После анализа поставленного задания был разработан алгоритм, который определяет, является ли массив слабой кучей, и написана программа, реализующая этот алгоритм.

Алгоритм:

1) Заводим два массива, первый — под элементы слабой кучи, второй — под 'биты' (пояснение в описание);

2) Считываем вначале массив, который нужно проверить (является ли он слабой кучей);

3) Считываем массив битов только для элементов, у которых есть потомки (если требуется поменять потомки местами — в массив битов для текущего индекса родителя заносим единицу, если не надо — заносим ноль);

4) Проверяем корневой узел на наличие левого потомка с помощью битового массива (если есть левый потомок, то массив не является слабой кучей);

5) Проходим по всем 'родителям' массива элементов слабой кучи (для этого используется битовый массив), сравнивая родителя с правым потомком;

6) Если текущий родитель меньше текущего правого потомка, то массив не является слабой кучей;

7) Если в ходе сравнивая родителя и правого потомка обнаружилось, что все родители больше своих правых потомков, то такой массив — слабая куча;

Пояснения:

*битовый массив — массив, состоящий из нулей и единиц, ноль показывает, что для родителя по i-му индексу обмена потомками не производилось, а единица — что обмен потомками был совершен;

мы проходим по всем родителям, определяя родителя с помощью формул 2<индекс родителя>+<элемент в массиве битов по индексу родителя> — для левого потомка и 2*<индекс родителя>+1-<элемент в массиве битов по

индексу родителя> (так битовый массив позволит определить, совершался ли обмен потомками для текущего родителя в массиве слабой кучи);

Разработана программа с использованием рекурсии и библиотеки vector.

Был создан класс Weak_Heap:

- Свойства класса (все приватные):
    1) vector<int> wheap — свойство, хранящее массив чисел;
    2) int len — свойство, хранящее длину массива;
    3) vector<int> bit – свойство, хранящее массив битов;
    4) int len_bit – свойство, хранящее длину массива битов;
- Методы класса (все публчиные):
    1) void read() - считывает через cin массив с консоли и записывает элементы в wheap;
    2) void set_len(int) - устанавливает len согласно переданному аргументу;
    3) void check(int, bool*) - рекурсивный метод, который проверяет являются ли массив слабой кучей путем сравнение родителя и правого потомка (правый потомок в слабой куче определяется по формуле 2*<индекс родителя>+1-<элемент в массиве битов по индексу родителя>, левый — 2*<индекс родителя>+<элемент в массиве битов по индексу родителя>;
    4) void print_wheap() - метод, с помощью которого можно вывести элемента вектора в консоль;
    5) void print_bit() - метод, с помощью которого можно вывести массив битов
    6) void read_bit() - метод, считывающий массив битов с консоли (считывание происходит только для элементов из массива слабой кучи, у которых есть хотя бы один потомок);
    7) bool check_root() - метод, который проверяет, если ли у корневого узла левый потомок (если есть — возвращает false, если нет — true);

В main создается экземпляр класса Wake_Heap и вызываются публичные методы, после чего в заранее созданную переменную flag типа bool будет записано false, если массив — слабая куча. В конце проверяется flag и производится соответствующий вывод в консоль.

Пример работы программы.

№ п/п	Входные данные	Выходные данные	Комментарии
1	Введите длину массива 3 Введите массив для проверки 340 12 339 Введите значение родителя для битового массива '1' - если потомки поменяны местами или '0' - если нет Родитель 340 имеет потомков: 12 Для родителя 340 с индексом 0 введите значение для битового массива: 0 Родитель 12 имеет потомков: 339 Для родителя 12 с индексом 1 введите значение для битового массива: 0	Получился следующий массив битов: 0 0 Массив является слабой кучей	
2	Введите длину массива 3 Введите массив для проверки 340 12 339 Введите значение родителя для битового массива '1' - если потомки поменяны местами или '0' - если нет Родитель 340 имеет потомков: 12 Для родителя 340 с индексом 0 введите значение для битового массива: 0 Родитель 12 имеет потомков: 339 Для родителя 12 с индексом 1 введите значение для битового массива: 1	Получился следующий массив битов: 0 1 Родитель 12 меньше своего правого потомка 339 Массив не является слабой кучей	
3	Введите длину массива 16 Введите массив для проверки 80 10 52 51 89 12 70 25 41 96 22 35 76 87 40 16 Введите значение родителя для битового массива '1' - если потомки поменяны местами или '0' - если	Получился следующий массив битов: 0 0 0 0 1 0 0 0 Родитель 10 меньше своего правого потомка 51	

	нет Родитель 80 имеет потомков: 10 Для родителя 80 с индексом 0 введите значение для битового массива: 0 Родитель 10 имеет потомков: 52 и 51 Для родителя 10 с индексом 1 введите значение для битового массива: 0 Родитель 52 имеет потомков: 89 и 12 Для родителя 52 с индексом 2 введите значение для битового массива: 0 Родитель 51 имеет потомков: 70 и 25 Для родителя 51 с индексом 3 введите значение для битового массива: 0 Родитель 89 имеет потомков: 41 и 96 Для родителя 89 с индексом 4 введите значение для битового массива: 1 Родитель 12 имеет потомков: 22 и 35 Для родителя 12 с индексом 5 введите значение для битового массива: 0 Родитель 70 имеет потомков: 76 и 87 Для родителя 70 с индексом 6 введите значение для битового массива: 0 Родитель 25 имеет потомков: 40 и 16 Для родителя 25 с индексом 7 введите значение для битового массива: 0	Родитель 12 меньше своего правого потомка 35 Родитель 70 меньше своего правого потомка 87 Массив не является слабой кучей	
4	Введите длину массива 16 Введите массив для проверки 97 83 52 76 89 33 73 78 66 32 20 48 67 58 45 13 Введите значение родителя для битового массива '1' - если потомки поменяны местами или '0' - если нет Родитель 97 имеет потомков: 83 Для родителя 97 с индексом 0 введите значение для битового массива: 0 Родитель 83 имеет потомков: 52 и 76	Получился следующий массив битов: 0 0 0 1 0 1 0 0 Массив является слабой кучей	

	Для родителя 83 с индексом 1 введите значение для битового массива: 0 Родитель 52 имеет потомков: 89 и 33 Для родителя 52 с индексом 2 введите значение для битового массива: 0 Родитель 76 имеет потомков: 73 и 78 Для родителя 76 с индексом 3 введите значение для битового массива: 1 Родитель 89 имеет потомков: 66 и 32 Для родителя 89 с индексом 4 введите значение для битового массива: 0 Родитель 33 имеет потомков: 20 и 48 Для родителя 33 с индексом 5 введите значение для битового массива: 1 Родитель 73 имеет потомков: 67 и 58 Для родителя 73 с индексом 6 введите значение для битового массива: 0 Родитель 78 имеет потомков: 45 и 13 Для родителя 78 с индексом 7 введите значение для битового массива: 0		
5	Введите длину массива 8 Введите массив для проверки 1 1 1 1 1 1 1 1 Введите значение родителя для битового массива '1' - если потомки поменяны местами или '0' - если нет Родитель 1 имеет потомков: 1 Для родителя 1 с индексом 0 введите значение для битового массива: 0 Родитель 1 имеет потомков: 1 и 1 Для родителя 1 с индексом 1 введите значение для битового массива: 1 Родитель 1 имеет потомков: 1 и 1 Для родителя 1 с индексом 2 введите значение для	Получился следующий массив битов: 0 1 1 1 Массив является слабой кучей	

	битового массива: 1 Родитель 1 имеет потомков: 1 и 1 Для родителя 1 с индексом 3 введите значение для битового массива: 1		

Иллюстрация работы программы.

*IDE – Code::Blocks 20.03

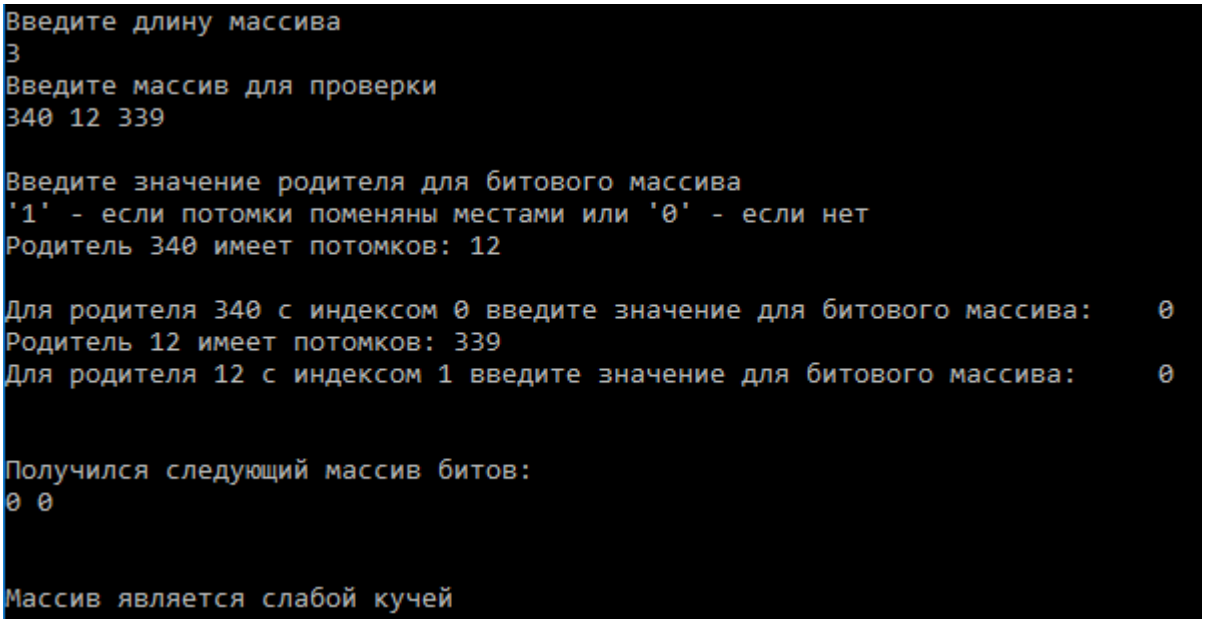

Рисунок 1 - Пример работы программы с входными данными №1

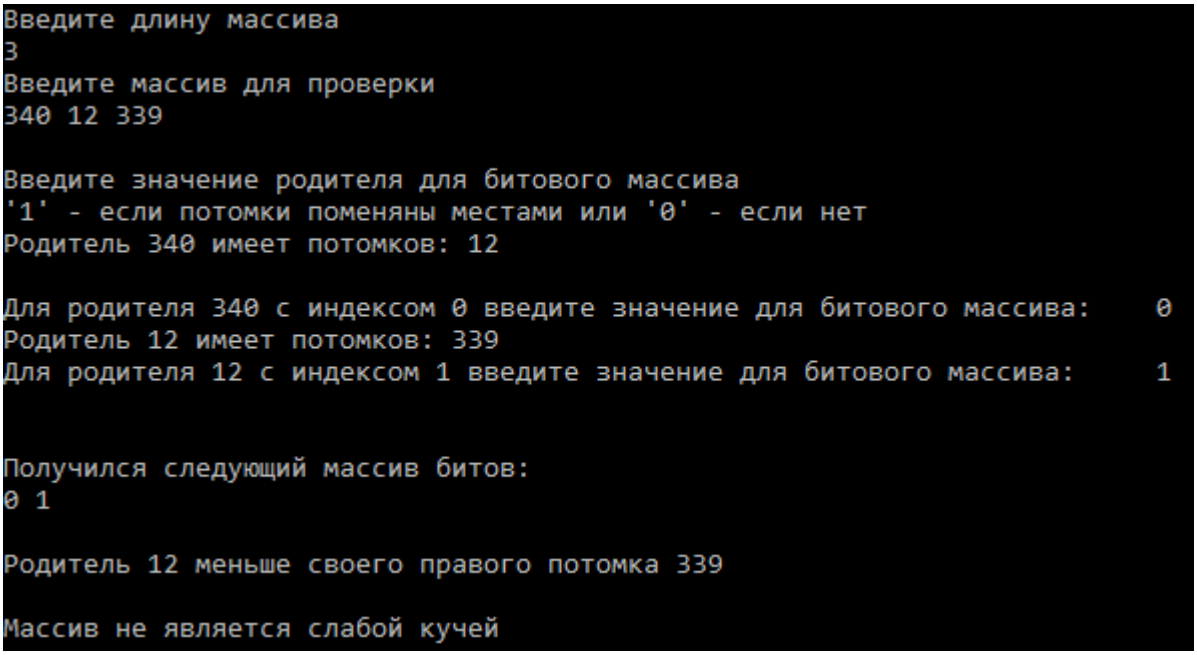

Рисунок 2 - Пример работы программы с входными данными №2

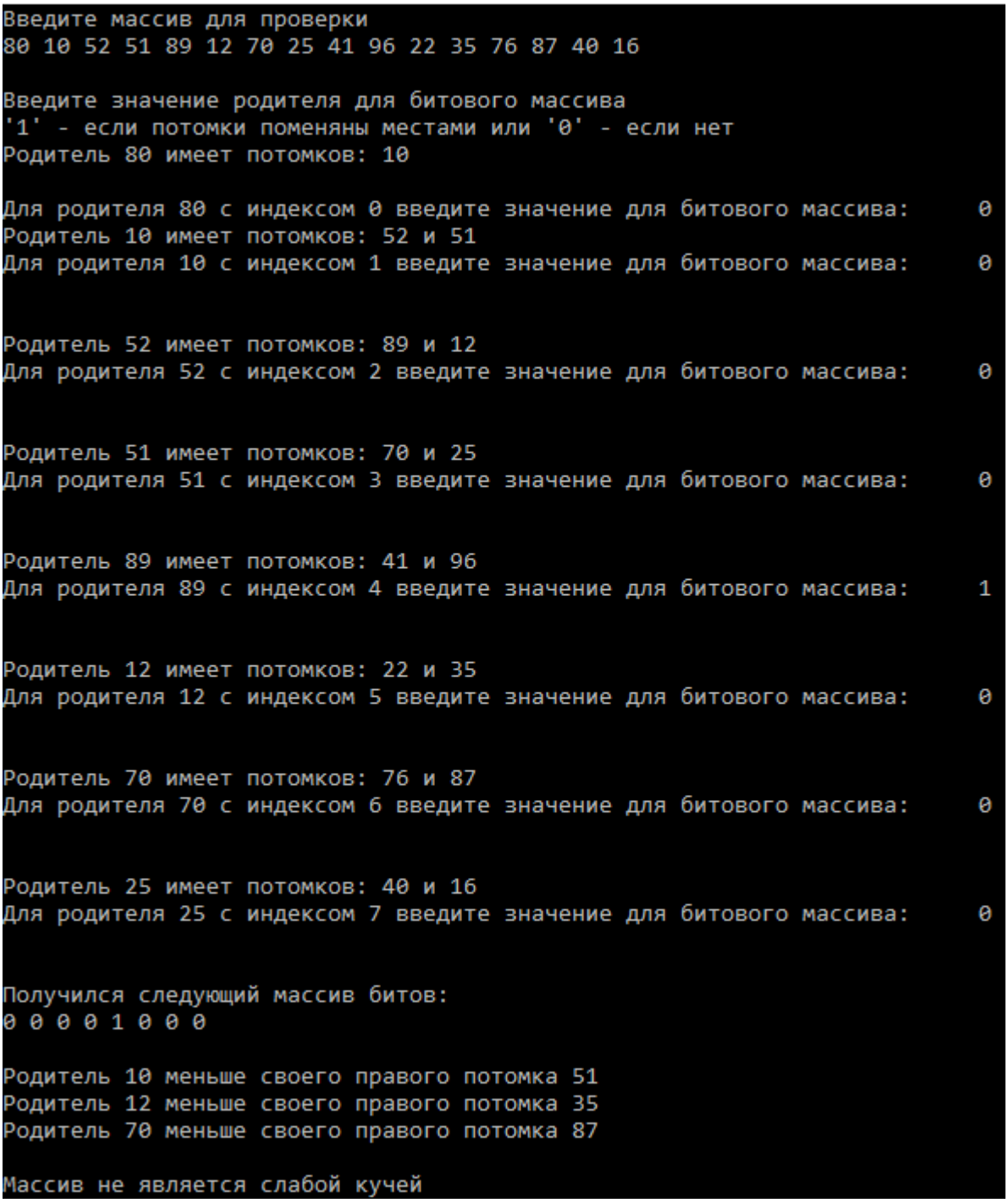

Рисунок 3 - Пример работы программы с входными данными №3

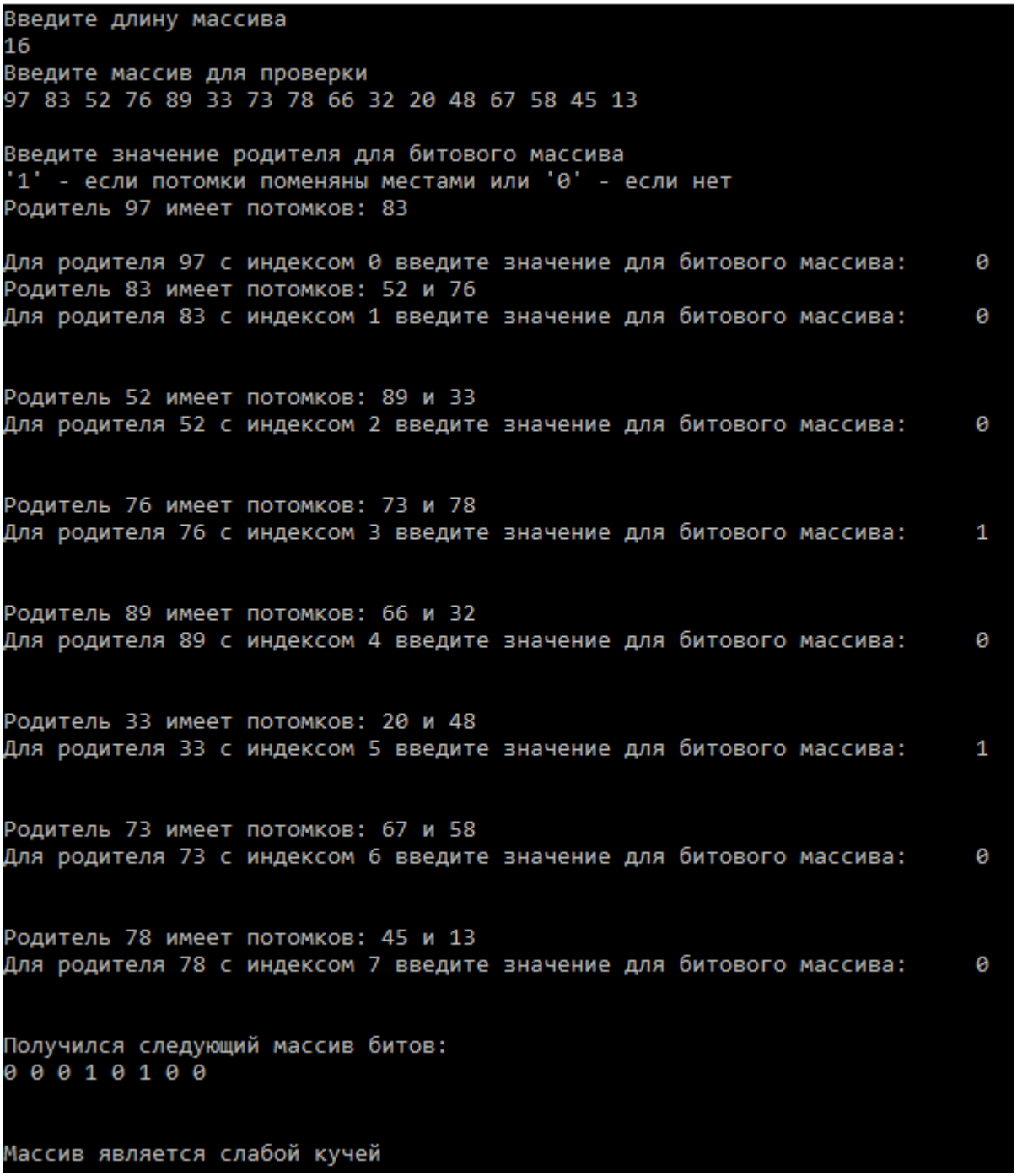

Рисунок 4 — Пример работы программы с входными данными №4

```
Введите длину массива
8
Введите массив для проверки
1 1 1 1 1 1 1 1

Введите значение родителя для битового массива
'1' - если потомки поменяны местами или '0' - если нет
Родитель 1 имеет потомков: 1

Для родителя 1 с индексом 0 введите значение для битового массива:      0
Родитель 1 имеет потомков: 1 и 1
Для родителя 1 с индексом 1 введите значение для битового массива:      1


Родитель 1 имеет потомков: 1 и 1
Для родителя 1 с индексом 2 введите значение для битового массива:      1


Родитель 1 имеет потомков: 1 и 1
Для родителя 1 с индексом 3 введите значение для битового массива:      1


Получился следующий массив битов:
0 1 1 1


Массив является слабой кучей
```

Рисунок 5 — Пример работы программы с входными данными №5

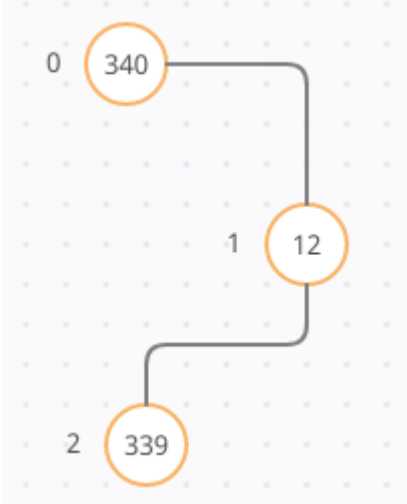

Рисунок 6 — Графическое представление слабой кучи в примере №1 (в ячейках кругов обозначен вес (значение массиву по i-ому индексу), слева от кругов обозначен индекса элемента в массиве

Выводы.

Произошло ознакомление со слабой кучей и была написана программа на языке программирования C++.

ПРИЛОЖЕНИЕ А

Файл main.cpp

```
#include <iostream>
#include <windows.h>
#include <vector>
#include "Weak_Heap.h"

using namespace std;

int main()
{
    SetConsoleCP(1251); // функции для работы с кириллицей (IDE CODE::BLOCKS плохо работает с кириллицей, поэтому приходится подключать соотвествущие функции
    SetConsoleOutputCP(1251);
    bool flag = true; //флаг для определения слабой кучи
    int len; //длина массива
    cout << "Введите длину массива\n";
    cin >> len;
    Weak_Heap arr;
    arr.set_len(len); //передать длину в свойство len класса
    arr.read(); //ввести массив в свойство wheap класса
    arr.read_bit(); //ввести массив в свойство bit класса
     if (arr.check_root()) //проверка значения в битовом массиве для корневого узла
    {
       arr.check(0, &flag); //рекурсивная функция определения слабой кучи
    }
```

```
    else
    {
        flag = false;
            cout << "\nКорневой узел в слабой кучи не может иметь левого потомка\n";
    }
    if (flag)
    {
        cout << "\nМассив является слабой кучей\n";
    }
    else
    {
        cout << "\nМассив не является слабой кучей\n";
    }
    return 0;
}
```

Файл Wake_Heap.h:

```
#pragma once
#include <vector>
#include <iostream>

using namespace std;

class Weak_Heap
{
private:
    vector<int> wheap; //свойство под массив
    vector<int> bit; //свойство под массив битов
```

```
    int len; //свойство под длину массива
    int len_bit;    //длина массива битов
public:
    void read();    //считывание массива слабой кучи с консолт
    void set_len(int);  //установка длины массива слабой кучи
    void check(int, bool*); //проверка массива (является ли он слабой кучей)
    void print_wheap(); //вывод содержимого вектора wheap в консоль
    void print_bit();   //вывод сожержимого вектора bit в консоль
      void read_bit();      //считывание значений для битоговго массива из консоли
       bool check_root();   //проверка значения в битовом массиве для корневого узла
};
```

Файл Wake_Heap.cpp:

```
#include "Weak_Heap.h"

bool Weak_Heap::check_root()
{
    if (bit[0] == 1)
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

```
void Weak_Heap::read_bit()
{
   cout << "\n";
   int key;
   cout << "Введите значение родителя для битового массива\n";
   cout << "'1' - если потомки поменяны местами или '0' - если нет\n";
    cout <<"Родитель " << wheap[0] << " имеет потомков: " << wheap[1] << "\n\n";
   cout << "Для родителя " << wheap[0] <<" с индексом " << 0 << " введите значение для битового массива:\t";
   cin >> key;
   if (key == 0 || key == 1)
   {
      bit[0] = key;
   }
   for (int i = 1; i < len; i++)
   {
      if (2*i < len)
      {
            cout <<"Родитель " << wheap[i] << " имеет потомков: " << wheap[2*i];
         if (2*i+1 < len)
         {
            cout << " и " << wheap[2*i+1] << "\n";
         }
         else
         {
            cout << "\n";
         }
```

```
                cout << "Для родителя " << wheap[i] <<" с индексом " << i << "
введите значение для битового массива:\t";
                cin >> key;
                if (key == 0 || key == 1)
                {
                   bit[i] = key;
                }
                cout << "\n\n";
                len_bit = i+1;
             }
          }
          cout << "Получился следующий массив битов:";
          print_bit();
          cout << "\n";
       }

       void Weak_Heap::read()
       {
          int key;
          cout << "Введите массив для проверки\n";
          for (int i = 0; i < len; i++)
          {
             cin >> key;
             wheap.push_back(key);
             bit.push_back(0);
          }
       }

       void Weak_Heap::print_wheap()   //метод, выводящий содержимое вектора
на экран
```

```
{
   cout << "\n";
   for (int i = 0; i < len; i++)
   {
      cout << wheap[i] << " ";
   }
   cout << "\n";
}

void Weak_Heap::print_bit()
{
   cout << "\n";
   for (int i = 0; i < len_bit; i++)
   {
      cout << bit[i] << " ";
   }
   cout << "\n";
}

void Weak_Heap::set_len(int len)
{
   this->len = len;
}

void Weak_Heap::check(int index, bool* flag)
{
    if (wheap[index] < wheap[2*index+1-bit[index]] && 2*index+1-bit[index]
< len) //если правый потомок больше родителя
   {
      *flag = false;
```

```
                cout << "Родитель " << wheap[index] << " меньше своего правого потомка " << wheap[2*index+1-bit[index]]<< "\n";
            }
            if (2*index+bit[index] < len && index != 0) //для левого потомка
            {
               check(2*index+bit[index], flag);
            }
            if (2*index+1-bit[index] < len)    //для правого потомка
            {
               check(2*index+1-bit[index], flag);
            }
        }
```