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 _____ Ноздрин В.Я.

Преподаватель _____ Попова Е.В.

Санкт-Петербург
2020

Цель работы.

Ознакомиться со структурой данных **WeakHeap** или слабая куча, реализовать программу, работающую со слабой кучей.

Задание 31.

Дан массив пар типа «число – бит». Предполагая, что этот массив представляет слабую кучу, вывести ее на экран в наглядном виде.

Теория.

Куча – специализированная структура данных типа **дерево**, которая удовлетворяет **свойству кучи,** а также является полным деревом. Свойство кучи: любой родитель **больше** или равен чем любой из его потомков (**max куча**). Аналогично можно определить **min кучу**, указав, что любой родитель **меньше** или равен чем любой из своих потомков.

Слабая куча – специализированная структура данных типа **бинарное дерево**, которая удовлетворяет свойству **слабой кучи.**

Свойство слабой кучи – для любого A узла верно, что A будет больше любого узла из **правого** поддерева (**max куча**). Имеет место быть и свойство определенное со знаком меньше, тогда куча была бы **min кучей**. В случае, когда B является **левым** узлом-потомком узла A, значения A и B никак не связаны.

Поскольку в корне кучи, даже слабой, нужен именно максимальный по величине элемент, у корня левого поддерева нет.

Существует общепринятая реализация **слабой кучи на массиве**. В нулевой адрес помещается значение корня дерева. Далее для каждого индекса **i** его правый и левый потомки помещаются в ячейки с адресами $2i + bit[i]$ и $2i + 1 - bit[i]$**,** где **bit[i]** – специальный массив битов. В зависимости от его значения, правый и левый потомки будут находиться либо в адресах $2i$ и $2i + 1$, либо наоборот в адресах $2i + 1$ и $2i$. Другими словами этот массив позволяет

легко менять местами поддеревья в слабой куче. Также заметим, что для корня **bit[0]=0** и по формулам считается только правый потомок, а адрес левого совпадает с самим корнем. Потому и принято, что левого поддерева у корня нет.

Ниже приведен пример слабой кучи на массиве по следующим данным.

97	83	52	76	89	33	73	78	66	32	20	48	67	58	45	13
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

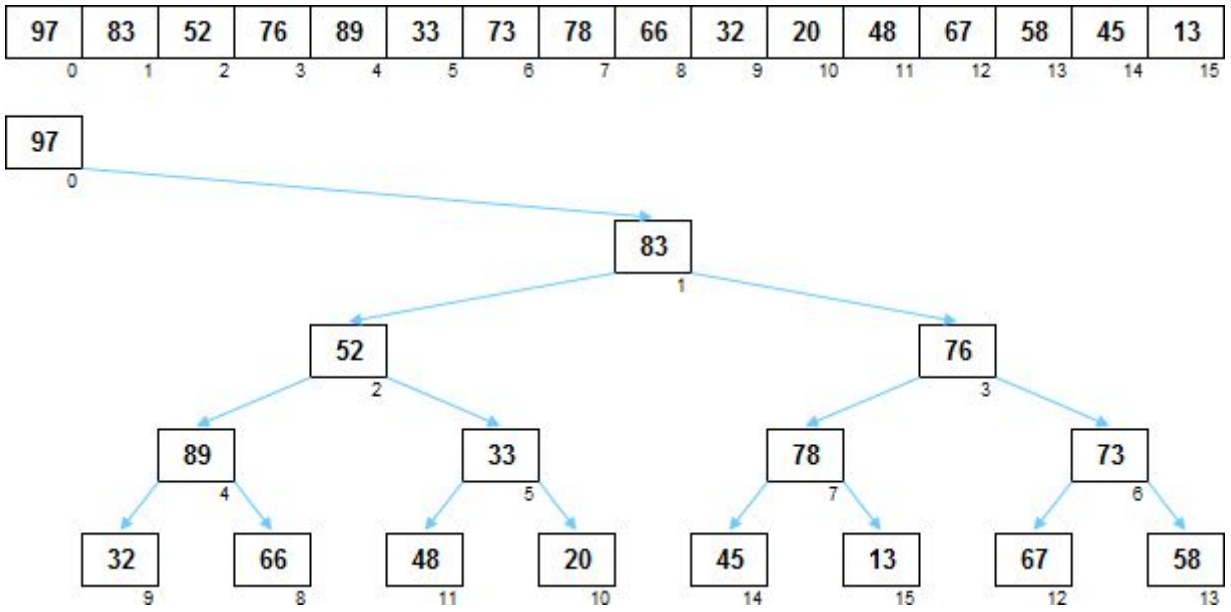
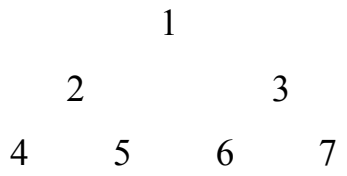
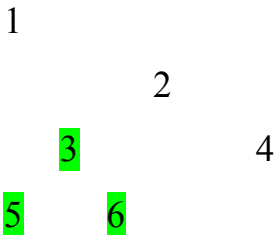
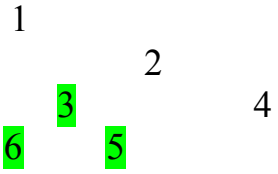


Рис.1 Пример реализации слабой кучи на массиве. Массив бит в данном случае состоит из нулей, кроме 3, 4, 5 индексов.

Выполнение работы. Алгоритм вывода на экран.

1. Так как слабая куча является **бинарным деревом**, в первую очередь было принято решение реализовать структуру бинарного дерева и решать задачу вывода бинарного дерева.
2. Реализована печать бинарного дерева. Выполняется проход в ширину по дереву и выполняется печать по уровням с соответствующими отступами.
3. После этого была реализована функция, строящая по массиву, реализующему слабую кучу, бинарное дерево. Далее куча печатается как бинарное дерево.

Пример работы программы.

Входные данные	**Выходные данные**
Печать обычного бинарного дерева	
[(1, false), (2, false), (3, **false**), (4, false), (5, false), (6, false)]	
[(1, false), (2, false), (3, **true**), (4, false), (5, false), (6, false)]	

Разработанный программный код см. в приложении А.

Вывод.

Была реализована функция, осуществляющая вывод слабой кучи на экран на языке программирования C++.

Приложение А
Исходный код программы

Название файла: **main.cpp**

```
#include <iostream>
#include <utility>
#include <cmath>

#include "WeakHeap.hpp"
#include "BinTree.hpp"

int main () {
  size_t size = 6;
  auto array = new std::pair<int, bool> [size];
  array[0] = std::pair<int, bool> (1, false);
  array[1] = std::pair<int, bool> (2, false);
  array[2] = std::pair<int, bool> (3, false);
  array[3] = std::pair<int, bool> (4, false);
  array[4] = std::pair<int, bool> (5, false);
  array[5] = std::pair<int, bool> (6, false);
  WeakHeap wh(array, size);
  BinTree bt(&wh);
  bt.print();
  return 0;
}
```

Название файла: **WeakHeap.hpp**

```
#ifndef LB5_AND_CW_WEAKHEAP_HPP
#define LB5_AND_CW_WEAKHEAP_HPP
#include <utility>
/* * * * * * * * * * * * * * * * * * * * */
/*! 31. Слабая куча. Weak heap.           *
 * Дан массив пар типа «число – бит».     *
 * Предполагая, что этот массив           *
 * представляет слабую кучу, вывести      *
 * её на экран в наглядном виде.          */
/* * * * * * * * * * * * * * * * * * * * */
/*! Сортировка слабой кучей. Демонстрация*/
/* * * * * * * * * * * * * * * * * * * * */
/*! i -> 2i + bit[i]        // left
 *  i -> 2i + 1 - bit[i]    // right
 *  i -> i div 2            // parent
 */
class WeakHeap{
 public:
  WeakHeap(): m_size(0), m_data(nullptr), m_bit(nullptr) {};
  WeakHeap(std::pair<int, bool> *array, size_t size) {
    m_size = size;
    m_data = new int [size];
    m_bit = new bool [size];
    for (size_t i = 0; i < size; i++) {
      m_data[i] = array[i].first;
      m_bit[i] = array[i].second;
    }
  };
  ~WeakHeap() { delete [] m_data; delete [] m_bit; };
```

```
  int getSize() { return m_size; };
  int* getData() { return m_data; };
  bool* getBit() { return m_bit; };
  int Join(int v, int w) {
    if (m_data[v] < m_data[w]) {
      std::swap(m_data[v], m_data[w]);
      m_bit[v] ^= 1;
      return 1;
    }
    return 0;
  };       /// ???
  int Up(int v) {
    if ((v % 2)^(m_bit[v/2]))
      return v/2;
    return Up(v/2);
  };                /// returns Right parent
  void SiftUp(int v) {
    int w;
    while (Join(v,w=Up(v)))
      v = w;
  };           /// sifts WeakHeap
 protected:
  int   m_size;
  int   *m_data;
  bool  *m_bit;
 private:
};
#endif //LB5_AND_CW_WEAKHEAP_HPP
```

Название файла: **Queue.hpp**

```
#ifndef LB5_AND_CW_QUEUE_HPP
#define LB5_AND_CW_QUEUE_HPP
template <class T>
class Queue {
 public:
  /*!--------Queue Element--------*/
  class Element {
   public:
    Element(T data) : m_data(data),  m_prev(nullptr), m_next(nullptr) {};
    T m_data;
    Element *m_prev, *m_next;
  };
  /*!----------------------------*/
 public:
  Queue() : m_first(nullptr), m_last(nullptr) {};
  Element*  top() { return m_last; };
  void      pop() {
    if (m_last) {
      m_last = m_last->m_prev;
      if (m_last) { delete m_last->m_next; m_last->m_next = nullptr; }
    }
  };
  void      push(T data) {
    auto *elem = new Element(data);
    if (m_first) { elem->m_next = m_first; m_first->m_prev = elem;
      m_first = elem; } else { m_first = elem; m_last = elem; }
  };
 private:
```

```
  Element *m_first, *m_last;
};
#endif //LB5_AND_CW_QUEUE_HPP
```

Название файла: **BinTree.hpp**

```
#ifndef LB5_AND_CW_BINTREE_HPP
#define LB5_AND_CW_BINTREE_HPP
#define MAX(a, b) ((a < b) ? b : a)
#include <iostream>
#include <cmath>
#include "Queue.hpp"
#include "WeakHeap.hpp"
class BinTree {
 public:
  BinTree(): m_data(0), m_empty(false), m_left(nullptr), m_right(nullptr) {};
   explicit BinTree(int data): m_data(data), m_empty(false), m_left(nullptr),
m_right(nullptr) {};
   explicit BinTree(WeakHeap *wh) : m_data(wh->getData()[0]), m_empty(false),
m_left(nullptr), m_right(nullptr) {
    int size = wh->getSize();
    auto tmp = new BinTree*[size];
    for (int i = 1; i < size; i++)
      tmp[i] = new BinTree(wh->getData()[i]);
    tmp[0] = this;
    for (int i = 0; i < size; i++) {
      int bit = wh->getBit()[i];
      if (2*i+bit < size && i != 0)
        tmp[i]->m_left = tmp[2*i+bit];
      if (2*i+1-bit < size)
        tmp[i]->m_right = tmp[2*i+1-bit];
    }
    delete [] tmp;
  };
  ~BinTree(){ delete m_left; delete m_right; };
  [[maybe_unused]] int height() {
    if (this->m_left == nullptr) {
      if (this->m_right == nullptr) { return 0;
      } else { return 1 + this->m_right->height(); }
    } else {
      if (this->m_right == nullptr) { return 1 + this->m_left->height();
       } else { return 1 + MAX(this->m_left->height(), this->m_right->height());
}
    }
  };
  [[maybe_unused]] void complete(){
    int h = this->height(); Queue<BinTree*> queue; queue.push(this);
    while (h > 0) { Queue<BinTree *> new_queue;
      while (queue.top()) {
        if (queue.top()->m_data->m_left == nullptr) {
          queue.top()->m_data->m_left = new BinTree;
          queue.top()->m_data->m_left->m_empty = true;
        } if (queue.top()->m_data->m_right == nullptr) {
          queue.top()->m_data->m_right = new BinTree;
          queue.top()->m_data->m_right->m_empty = true;
        } new_queue.push(queue.top()->m_data->m_left);
        new_queue.push(queue.top()->m_data->m_right);
        queue.pop();
      } queue = new_queue; h--; }
```

```
  };
  [[maybe_unused]] void print(){
    this->complete();
    Queue<BinTree*> queue;
    queue.push(this);
    int level = 0;
    int height = this->height();
    while (queue.top()) {
      Queue<BinTree*> new_queue;
      for (int i = 0; i < pow(2,height - level) - 1 ; i++) printf("     ");
      while (queue.top()) {
        if (queue.top()->m_data->m_empty)
          printf("     ");
        else
          printf("%5d", queue.top()->m_data->m_data);
           for (int i = 0; i < pow(2,height - level + 1) - 1 ; i++) printf("
");
        if (queue.top()->m_data->m_left)
          new_queue.push(queue.top()->m_data->m_left);
        if (queue.top()->m_data->m_right)
          new_queue.push(queue.top()->m_data->m_right);
        queue.pop();
      }
      printf("\n");
      queue = new_queue;
      level++;
    }

  };
  [[maybe_unused]] void setData(int data)        { m_data = data;  };
  [[maybe_unused]] void setLeft(BinTree* node)   { m_left = node;  };
  [[maybe_unused]] void setRight(BinTree* node)  { m_right = node; };
  [[maybe_unused]] [[nodiscard]] int getData() const  { return m_data;  };
  [[maybe_unused]] BinTree*  getLeft()  { return m_left;  };
  [[maybe_unused]] BinTree*  getRight() { return m_right; };
 protected:
  int m_data;
  bool m_empty;
  BinTree* m_left;
  BinTree* m_right;
};
/*
BinTree bt(1);
bt.setLeft(new BinTree(2));
bt.getLeft()->setLeft(new BinTree(4));
bt.getLeft()->setRight(new BinTree(5));
bt.setRight(new BinTree(3));
bt.getRight()->setLeft(new BinTree(6));
bt.getRight()->setRight(new BinTree(7));
bt.print();
*/

#endif //LB5_AND_CW_BINTREE_HPP
```