**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра CАПР**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 2**

Тема: Двоичные деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 0309 |  | Головатюк К.А. |
| Преподаватель |  | Тутуева А.В |

Санкт-Петербург

2021

### Постановка задачи

Реализовать класс двоичного дерева в соответствии со своим вариантом. Класс двоичной кучи.

### Цель работы

Научиться работать с двоичными деревьями, стеком и очередью.

### Описание реализуемого класса и методов.

Среда разработки – Visual Studio

Язык программирования – C++

В ходе выполнения лабораторной работы было создано три класса: класс двоичной кучи, класс стека, класс итератора.

Таблица 1 – описание класса Heap

|  |  |  |
| --- | --- | --- |
| Название метода/поля | Описание | Оценка временной сложности  (Для методов) |
| static const int SIZE = 100 | Максимальный размер |  |
| int\* h | Массив элементов |  |
| int HeapSize | Размер |  |
| void remove(int) | Удаление элемента по ключу | O(log(2)n) |
| void insert(int) | Добавление элемента по ключу | O(log(2)n) |
| bool contains(int) | Поиск элемента по ключу | O(log(2)n) |
| Iterator create\_dft\_iterator() | Создание итератора, метод в глубину | O(1) |
| Iterator create\_bft\_iterator() | Создание итератора, метод в ширину | O(1) |
| void outHeap() | Вывод в виде кучи | O(n) |
| void out() | Вывод в виде списка | O(n) |
| void heapify(int) | Упорядочивание кучи | O(n-m) |

Таблица 2 – описание класса Stack

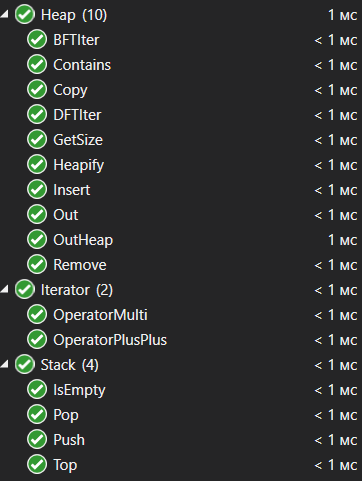
|  |  |  |
| --- | --- | --- |
| Название метода/поля | Описание | Оценка временной сложности  (Для методов) |
| linklst\* top | Поле содержащее структуру хранящую в себе указатель на следующий элемент стека |  |
| Size\_t size | Размер списка |  |
| void push\_back(int) | Добавление в конец списка | О(n) |
| void push(int) | Добавление в начало списка | O(1) |
| int Top(); | Возвращение позиции верхнего элемента | O(1) |
| int pop(); | Извлечение первого элемента | O(1) |
| bool isEmpty(); | проверка на пустоту стека | O(1) |

Таблица 3 – описание класса Iterator

|  |  |  |
| --- | --- | --- |
| Название метода/поля | Описание | Оценка временной сложности  (Для методов) |
| Stack stack | Поле стека |  |
| int\* heap | Поле кучи |  |
| int size | Размер кучи |  |
| int pos | Текущая позиция |  |
| int Top(); | Возвращение позиции верхнего элемента |  |
| int operator\*() const; | Оператор получения значения итератора | O(1) |
| Iterator& operator++(); | Проход по итератору | O(1) |

### Описание реализованных unit-тестов

Для проверки реализованных методов были написаны unit-тесты. Их названия представлены ниже.



Unit-тесты разделены на группы по классам.

### Пример работы :

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

### Листинг

**heap.h**

#pragma once

#include "iterator.h";

class Heap

{

static const int SIZE = 100;

int \*h;

int HeapSize;

public:

Heap();

~Heap();

void remove(int);

void insert(int);

bool contains(int);

Iterator create\_dft\_iterator();

Iterator create\_bft\_iterator();

int \*copy();

int getSize();

void outHeap();

void out();

void heapify(int);

};

**heap.cpp**

#include "heap.h"

#include <iostream>

using namespace std;

Heap::Heap()

{

h = new int[SIZE];

HeapSize = 0;

}

Heap::~Heap()

{

delete (h);

}

void Heap::insert(int n)

{

int i, parent;

i = HeapSize;

h[i] = n;

parent = (i - 1) / 2;

while (parent >= 0 && i > 0)

{

if (h[i] > h[parent])

{

int temp = h[i];

h[i] = h[parent];

h[parent] = temp;

}

i = parent;

parent = (i - 1) / 2;

}

HeapSize++;

}

bool Heap::contains(int value)

{

for (int i = 0; i < HeapSize; i++)

{

if (h[i] == value)

return true;

}

return false;

}

int \*Heap::copy()

{

int heap[100];

for (int i = 0; i < HeapSize; i++)

{

heap[i] = h[i];

}

return heap;

}

int Heap::getSize()

{

return HeapSize;

}

Iterator Heap::create\_dft\_iterator()

{

return Iterator(h, HeapSize, 'd');

}

Iterator Heap::create\_bft\_iterator()

{

return Iterator(h, HeapSize, 'b');

}

void Heap::outHeap(void)

{

int i = 0;

int k = 1;

while (i < HeapSize)

{

while ((i < k) && (i < HeapSize))

{

cout << h[i] << " ";

i++;

}

cout << endl;

k = k \* 2 + 1;

}

}

void Heap::out(void)

{

for (int i = 0; i < HeapSize; i++)

{

cout << h[i] << " ";

}

cout << endl;

}

void Heap::remove(int value)

{

for (int i = 0; i < HeapSize; i++)

{

if (h[i] == value)

{

if (i == 0)

{

h[0] = h[HeapSize - 1];

HeapSize--;

heapify(0);

}

else

{

HeapSize--;

for (int j = i; j < HeapSize; j++)

{

h[j] = h[j + 1];

heapify(j);

}

}

}

}

}

void Heap::heapify(int i)

{

int left, right;

int temp;

left = 2 \* i + 1;

right = 2 \* i + 2;

if (left < HeapSize)

{

if (h[i] < h[left])

{

temp = h[i];

h[i] = h[left];

h[left] = temp;

heapify(left);

}

}

if (right < HeapSize)

{

if (h[i] < h[right])

{

temp = h[i];

h[i] = h[right];

h[right] = temp;

heapify(right);

}

}

}

**iterator.h**

#ifndef ITERATOR\_H

#define ITERATOR\_H

class Stack

{

private:

struct linklst

{

int pos = -1;

linklst \*next = nullptr;

};

linklst \*top;

public:

Stack();

~Stack();

void push(int);

void push\_back(int);

int Top();

int pop();

bool isEmpty();

};

class Iterator

{

private:

Stack stack;

int \*heap;

int size;

int pos;

char type;

public:

Iterator(int \*, int, char \_type);

Iterator();

~Iterator();

int operator\*() const;

Iterator &operator++();

};

#endif

**iterator.cpp**

#include "iterator.h"

#include <iostream>

Iterator::Iterator()

{

heap = nullptr;

type = 'd';

pos = 0;

size = 0;

}

Iterator::~Iterator() {

delete(heap);

}

Iterator::Iterator(int \*\_heap, int \_size, char \_type)

{

heap = new int[\_size];

heap = \_heap;

size = \_size;

type = \_type;

pos = 0;

}

int Iterator::operator\*() const

{

if (pos > -1)

return heap[pos];

return -1;

}

//int\* Iterator::operator->() const{

// if (pos > -1)

// return &heap[pos];

// return nullptr;

//}

Iterator &Iterator::operator++()

{

if (pos != -1)

{

if (type == 'b')

{

if (stack.isEmpty())

{

if (pos \* 2 + 1 < size)

stack.push\_back(pos \* 2 + 1);

if (pos \* 2 + 2 < size)

stack.push\_back(pos \* 2 + 2);

}

pos = stack.pop();

if (pos != -1)

{

if (pos \* 2 + 1)

stack.push\_back(pos \* 2 + 1);

if (pos \* 2 + 2)

stack.push\_back(pos \* 2 + 2);

}

}

else if (type == 'd')

{

if (pos \* 2 + 2 < size)

{

stack.push(pos \* 2 + 2);

}

if (pos \* 2 + 1 < size)

{

pos = pos \* 2 + 1;

}

else

{

pos = stack.Top();

stack.pop();

}

}

}

return \*this;

}

Stack::Stack()

{

top = nullptr;

}

Stack::~Stack()

{

while (top)

{

linklst \*r = top;

top = top->next;

delete r;

}

delete top;

}

void Stack::push(int pos)

{

linklst \*r = new linklst();

r->pos = pos;

r->next = top;

top = r;

}

void Stack::push\_back(int pos)

{

linklst \*r = new linklst();

r->pos = pos;

if (top == nullptr)

{

r->next = top;

top = r;

return;

}

linklst \*n = top;

while (n->next)

{

n = n->next;

}

n->next = r;

}

int Stack::Top()

{

if (top)

return top->pos;

return -1;

}

int Stack::pop()

{

if (!isEmpty())

{

linklst \*r = top;

top = top->next;

int x = r->pos;

delete r;

return x;

}

return -1;

}

bool Stack::isEmpty()

{

if (top == nullptr)

return true;

return false;

}

**main.cpp**

#include <iostream>

#include "heap.h"

using namespace std;

int main() {

Heap heap;

heap.insert(1);

heap.insert(2);

heap.insert(3);

heap.insert(4);

heap.insert(5);

heap.insert(6);

heap.insert(7);

heap.insert(8);

heap.insert(9);

heap.insert(10);

heap.insert(11);

heap.insert(12);

heap.insert(13);

heap.insert(14);

heap.insert(15);

heap.insert(16);

heap.outHeap();

cout << endl;

heap.out();

auto iter = heap.create\_dft\_iterator();

cout << endl;

for (int i = 0; i < 16; i++) {

cout << \*iter << endl;

++iter;

}

}