Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 12

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные кучи»

Выполнил:

Студент 1 курса 10 группы

Жамойдо Артём Игоревич

Преподаватель: асс. Андронова М.В.

Минск, 2024

В проект добавить следующие функции: удаление минимального **extractMin**; удаление i-ого элемента **extractI**; объединение **unionHeap** двух куч в одну.

**Lab\_12.cpp**

#include "heap.h"

#include <iostream>

using namespace std;

using namespace heap;

// Функция сравнения элементов AAA для кучи

CMP cmpAAA(void\* a1, void\* a2)

{

#define A1 ((AAA\*)a1)

#define A2 ((AAA\*)a2)

CMP rc = EQUAL; // Устанавливаем равенство по умолчанию

if (A1->x > A2->x)

rc = GREAT;

else

if (A2->x > A1->x)

rc = LESS;

return rc;

#undef A2

#undef A1

}

int main()

{

setlocale(LC\_ALL, "rus");

int k, choice;

// Создание двух куч h1 и h2

Heap h1 = create(30, cmpAAA);

Heap h2 = create(30, cmpAAA);

// Указатель на вторую кучу

Heap\* ph2 = &h2;

for (;;)

{

cout << "1 - Вывод кучи на экран" << endl;

cout << "2 - Добавить элемент" << endl;

cout << "3 - Удалить максимальный элемент" << endl;

cout << "4 - Удалить минимальный элемент" << endl;

cout << "5 - Удалить i-ый элемент" << endl;

cout << "6 - Объединение двух бинарных куч(конец - ввод отрицательного числа)" << endl;

cout << "0 - Выход" << endl;

cout << "Cделайте выбор:" << endl;

cin >> choice;

switch (choice)

{

case 0: exit(0);

case 1: h1.scan(0);

break;

case 2:

{

AAA\* a = new AAA;

cout << "Введите ключ:" << endl;

cin >> k;

a->x = k;

h1.insert(a);

}

break;

case 3: h1.extractMax();

break;

case 4: h1.extractMin();

break;

case 5:

int i;

cout << "Введите i:\n";

cin >> i;

h1.extractI(i);

break;

case 6:

int j = 1;

cout << "Введите вторую кучу:\n";

while (j > 0) // Цикл ввода данных для второй кучи

{

AAA\* b = new AAA;

cout << "Введите ключ:" << endl;

cin >> j;

if (j > 0)

{

b->x = j;

h2.insert(b);

}

}

// Вызов метода объединения двух куч

h1.unionHeap(ph2);

break;

}

}

return 0;

}

**Heap.cpp**

#include "heap.h"

#include <iostream>

#include <iomanip>

using namespace std;

// Метод печати данных из объекта AAA

void AAA::print() const

{

cout << x;

}

// Метод получения приоритета

int AAA::getPriority() const

{

return x;

}

namespace heap

{

// Функция создания объекта класса Heap

Heap create(int maxsize, CMP(\*f)(void\*, void\*))

{

return \*(new Heap(maxsize, f));

}

// Метод определения левого потомка элемента по индексу ix

int Heap::left(int ix) const

{

return (2 \* ix + 1 >= size) ? -1 : (2 \* ix + 1);

}

// Метод определения правого потомка элемента по индексу ix

int Heap::right(int ix) const

{

return (2 \* ix + 2 >= size) ? -1 : (2 \* ix + 2);

}

// Метод определения родителя элемента по индексу ix

int Heap::parent(int ix)

{

return (ix + 1) / 2 - 1;

}

// Метод обмена данных между двумя элементами

void Heap::swap(int i, int j)

{

void\* buf = storage[i];

storage[i] = storage[j];

storage[j] = buf;

}

// Метод просеивания элемента вниз по куче

void Heap::heapify(int ix)

{

int l = left(ix), r = right(ix), irl = ix;

if (l > 0)

{

if (isGreat(storage[l], storage[ix]))

irl = l;

if (r > 0 && isGreat(storage[r], storage[irl]))

irl = r;

if (irl != ix)

{

swap(ix, irl);

heapify(irl);

}

}

}

// Метод добавления нового элемента в кучу

void Heap::insert(void\* x)

{

int i;

if (!isFull())

{

storage[i = ++size - 1] = x;

while (i > 0 && isLess(storage[parent(i)], storage[i]))

{

swap(parent(i), i);

i = parent(i);

}

}

}

// Метод извлечения максимального элемента из кучи

void\* Heap::extractMax()

{

void\* rc = nullptr;

if (!isEmpty())

{

rc = storage[0];

storage[0] = storage[size - 1];

size--;

heapify(0);

}

return rc;

}

// Метод вывода данных кучи на экран

void Heap::scan(int i) const

{

int probel = 20;

cout << '\n';

if (size == 0)

cout << "Куча пуста";

for (int u = 0, y = 0; u < size; u++)

{

cout << setw(probel + 10) << setfill(' ');

((AAA\*)storage[u])->print();

if (u == y)

{

cout << '\n';

if (y == 0)

y = 2;

else

y += y \* 2;

}

probel /= 2;

}

cout << '\n';

}

// Метод удаления минимального элемента

void\* Heap::extractMin()

{

void\* rc = nullptr;

// Проверка на пустоту

if (!isEmpty())

{

// Смещаем данные в массиве и уменьшаем size

rc = storage[size - 1];

storage[size - 1] = storage[0];

size--;

heapify(size - 1);

}

return rc;

}

// Метод удаления по индексу

void\* Heap::extractI(int i)

{

void\* rc = nullptr;

// Проверка на пустоту

if (!isEmpty())

{

// Смещаем данные в массиве

rc = storage[i];

storage[i] = storage[size - 1];

size--;

heapify(i);

}

return rc;

}

// Метод объединения двух куч

void Heap::unionHeap(Heap\* h2)

{

void\* rc = nullptr;

// Записываем данные и добавляем элементы в главную кучу

for (int i = 0; i < h2->size; i++)

{

rc = h2->storage[i];

insert(rc);

}

heapify(0);

}

}

**Heap.h**

#pragma once

// Структура AAA для хранения целочисленных данных и соответствующих методов

struct AAA

{

int x;

void print() const;

int getPriority() const;

};

namespace heap

{

// Перечисление CMP для определения типов сравнения

enum CMP

{

LESS = -1, EQUAL = 0, GREAT = 1

};

// Структура Heap для реализации кучи

struct Heap

{

int size;

int maxSize;

void\*\* storage;

CMP(\*compare)(void\*, void\*);

// Конструктор класса Heap

Heap(int maxsize, CMP(\*f)(void\*, void\*))

{

size = 0;

storage = new void\* [maxSize = maxsize];

compare = f;

};

// Методы для работы с индексами в куче

int left(int ix) const;

int right(int ix) const;

int parent(int ix);

// Проверка на полноту и пустоту кучи

bool isFull() const

{

return (size >= maxSize);

};

bool isEmpty() const

{

return (size <= 0);

};

// Методы сравнения элементов в куче

bool isLess(void\* x1, void\* x2) const

{

return compare(x1, x2) == LESS;

};

bool isGreat(void\* x1, void\* x2) const

{

return compare(x1, x2) == GREAT;

};

bool isEqual(void\* x1, void\* x2) const

{

return compare(x1, x2) == EQUAL;

};

void swap(int i, int j);

void heapify(int ix);

void insert(void\* x);

void\* extractMax();

void\* extractMin(); // Метод извлечения минимального элемента

void\* extractI(int i); // Метод извлечения элемента по индексу

void scan(int i) const;

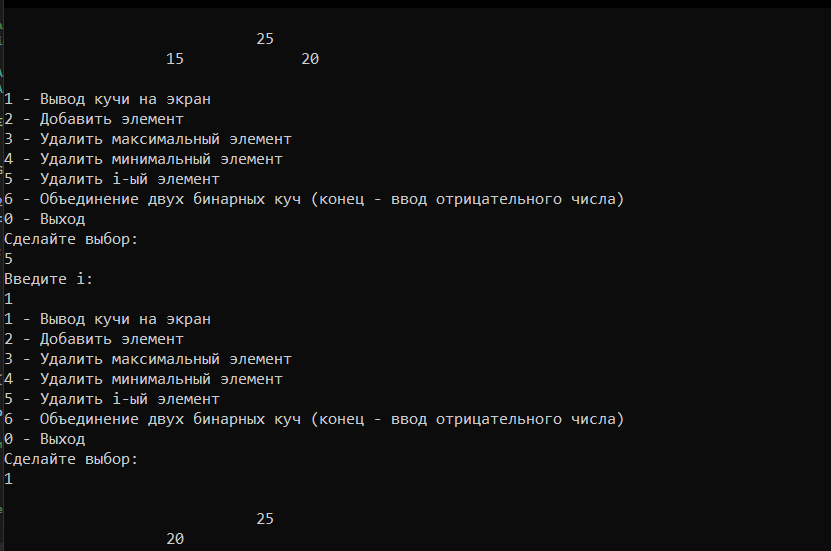
void unionHeap(Heap\* h2); // Метод объединения двух куч

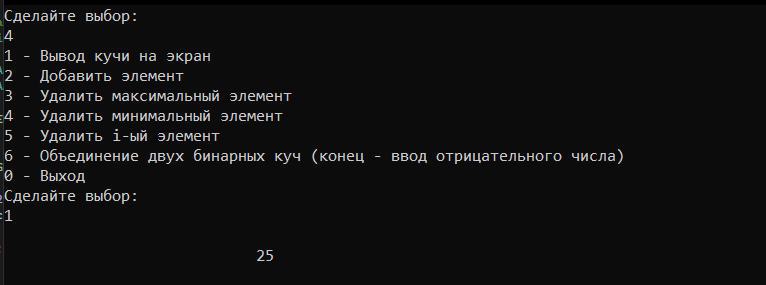
};

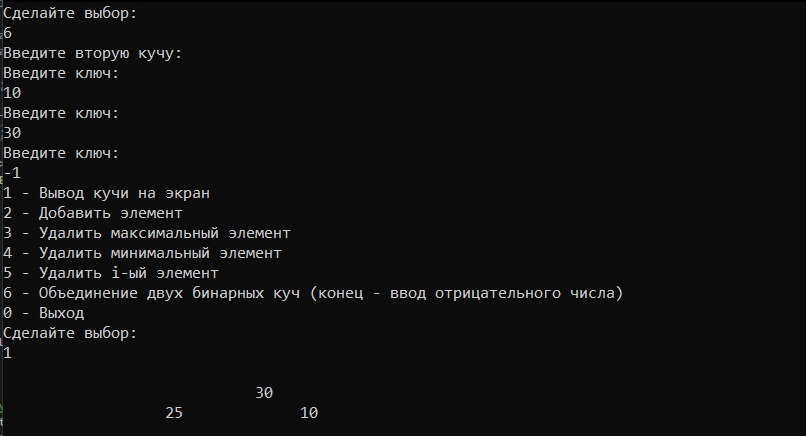
// Функция создания кучи с заданным размером и функцией сравнения

Heap create(int maxsize, CMP(\*f)(void\*, void\*));

};

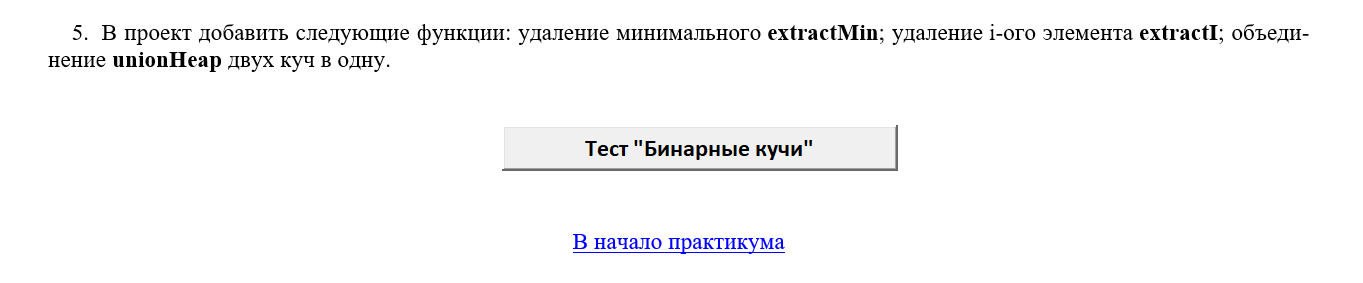
****

****

****

Доп. задачи

**Варианты и доп. задачи отсутствуют в практикуме:**

****