Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 12

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные кучи»

Выполнил:

Кулешов Артём

Студент 1 курса 8 группы

Преподаватель: асс. Андронова М.В.

Минск, 2024

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

**Heap.h**

#pragma once

struct AAA

{

int x;

void print();

};

namespace heap

{

enum CMP

{

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

};

struct Heap

{

int size;

int maxSize;

void\*\* storage; // данные

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

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

{

size = 0;

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

compare = f;

};

int left(int ix);

int right(int ix);

int parent(int ix);

bool isFull() const

{

return (size >= maxSize);

};

bool isEmpty() const

{

return (size <= 0);

};

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();

void unionHeap(void\* x);

void extractI();

void scan(int i) const;

};

Heap create(intmaxsize, CMP(\*f)(void\*, void\*));

};

**Main.cpp**

#include"Heap.h"

#include<iostream>

using namespace std;

heap::CMP cmpAAA(void\* a1, void\* a2) //Функция сравнения

{

#define A1 ((AAA\*)a1)

#define A2 ((AAA\*)a2)

heap::CMP

rc = heap::EQUAL;

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

rc = heap::GREAT;

else

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

rc = heap::LESS;

return rc;

#undef A2

#undef A1

}

//-------------------------------

int main()

{

setlocale(LC\_ALL, "rus");

int k, choice;

heap::Heap h1 = heap::create(30, cmpAAA);

heap::Heap h2 = heap::create(30, cmpAAA);

for (;;)

{

cout << "1 - вывод первой кучи на экран" << endl;

cout << "2 - добавить элемент в первой кучу" << endl;

cout << "3 - вывод второй кучи на экран" << endl;

cout << "4 - добавить элемент в второй кучу" << endl;

cout << "5 - объединение двух куч" << endl;

cout << "6 - удалить минимальный элемент первой кучи" << endl;

cout << "7 - удалить i-й элемент первой кучи" << endl;

cout << "0 - выход" << endl;

cout << "Ваш 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: h2.scan(0);

break;

case 4:

{AAA\* a = new AAA;

cout << "Введите ключ" << endl; cin >> k;

a->x = k;

h2.insert(a);

h1.insert(a);

}

break;

case 5:

h1.scan(0);

break;

case 6: h1.extractMin();

break;

case 7: h1.extractI();

break;

default: cout << endl << "Введена неверная команда!" << endl;

}

} return 0;

}

**Heap.cpp**

#include"Heap.h"

#include<iostream>

#include<iomanip>

using namespace std;

void AAA::print()

{

cout << x;

}

namespace heap

{

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

{

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

}

int Heap::left(int ix) // получение левого элемента

{

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

}

int Heap::right(int ix) // получение правого элемента

{

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

}

int Heap::parent(int ix) // получение родительского узла

{

return (ix + 1) / 2 - 1;

}

void Heap::swap(int i, int j) // меняем местами

{

void\* buf = storage[i];

storage[i] = storage[j];

storage[j] = buf;

}

void Heap::heapify(int ix) //восстанавливаем кучу

{

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);

}

}

}

void Heap::insert(void\* x) // вставляем элемент в кучу

{

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);

}

}

}

void\* Heap::extractMax() // удаляем максимальное значение из кучи

{

void\* rc = nullptr;

if (!isEmpty())

{

rc = storage[0];

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

size--;

heapify(0);

} return rc;

}

void Heap::scan(int i) const//Вывод значений элементов на экран

{

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';

}

void Heap::extractI() // удаление i-го элемента

{

int i;

cout << "Введите i:";

cin >> i;

cout << endl;

if (!isEmpty()) // Проверка, что куча не пуста

{

for (int j = i; j < size - 1; j++) // Сдвиг элементов после i на одну позицию влево

{

storage[j] = storage[j + 1];

}

size--; // Уменьшение размера кучи на 1

heapify(0); // Восстановление свойств кучи

}

}

void Heap::extractMin() // удаление минимального

{

int min = 100, a; // Инициализация переменных min и a

int\*\* A = (int\*\*)storage; // Приведение указателя storage к типу int\*\*

void\* rc = nullptr; // Инициализация указателя rc

if (!isEmpty()) // Проверка, что куча не пуста

{

for (int i = 0; i < size; i++) // Поиск минимального значения в куче

{

if (\*A[i] < min) // Если текущий элемент меньше min, обновляем min и запоминаем индекс a

{

min = \*A[i];

a = i;

}

}

int i = parent(a); // Вычисление индекса родителя элемента a

for (int i = a; i < size - 1; i++) // Сдвиг элементов после a на одну позицию влево

{

storage[i] = storage[i + 1];

}

size--; // Уменьшение размера кучи на 1

heapify(0); // Восстановление свойств кучи

}

}

void Heap::unionHeap(void\* x) // объединение двух куч в одну

{

int i;

for (i = 0; i < size - 1; i++) // Сдвиг элементов кучи на одну позицию влево

{

storage[i] = storage[i + 1];

}

heapify(0); // Восстановление свойств кучи

storage[i = ++size - 1] = x; // Добавление нового элемента x в конец кучи

while (i > 0 && isLess(storage[parent(i)], storage[i])) // Пока не достигнут корень и нарушено свойство кучи, выполняем перестановку элементов

{

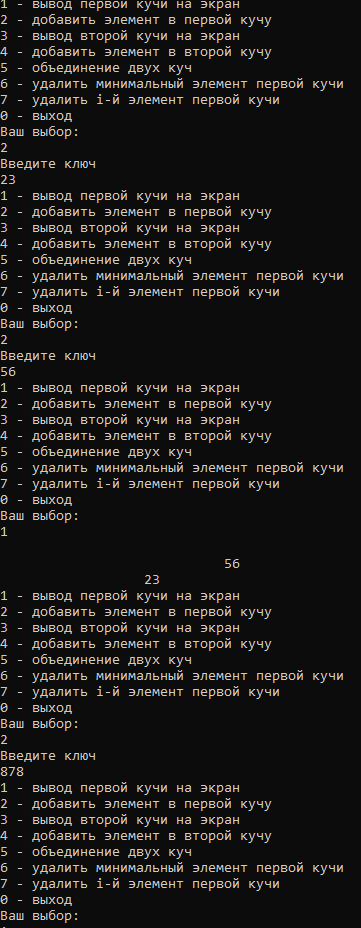
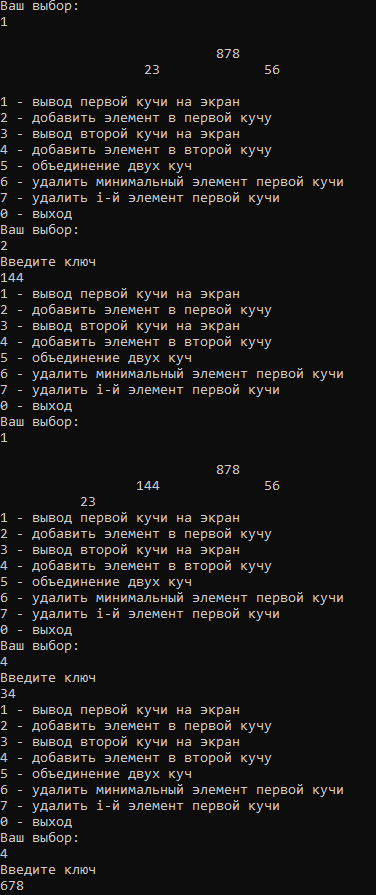
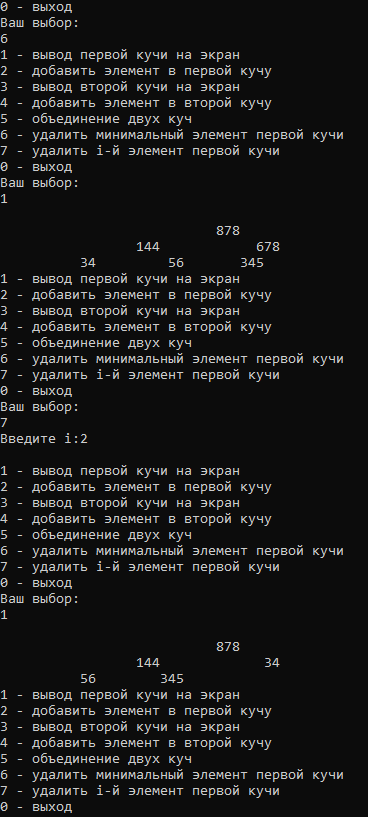
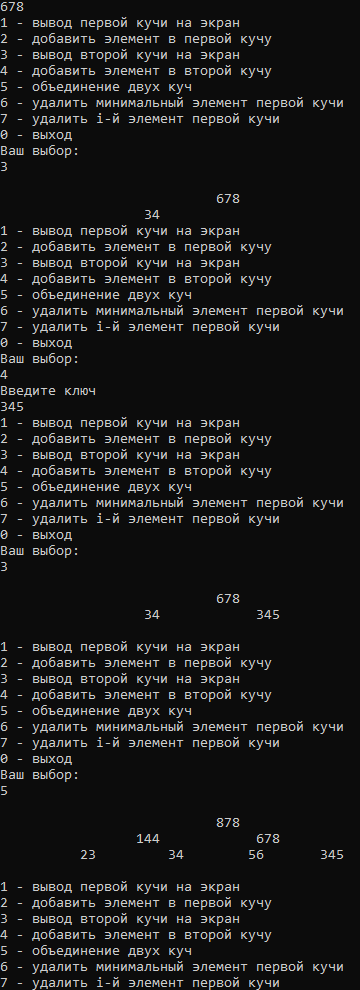
swap(parent(i), i);

i = parent(i);

}

}

}

Вариант №7

int Heap::countLeaves()

{

intcount = 0;

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

{

if (left(i) == -1 &&right(i) == -1)

{

count++;

}

}

returncount;

}

Вариант №10

intHeap::sumVertices()

{

intsum = 0;

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

{

sum += \*((int\*)storage[i]);

}

returnsum;

}

Вариант №16

intHeap::sumLeaves()

{

intsum = 0;

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

{

if (left(i) == -1 &&right(i) == -1)

{

sum+= \*((int\*)storage[i]);

}

}

returnsum;

}

}

