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

Мандрик Алексей Иванович

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.

**Код программы:**

**Heap.cpp**

#include "heap.h"

// Метод класса AAA для вывода ключа

void AAA::print()

{

std::cout << x;

}

// Метод класса AAA для получения приоритета

int AAA::getPriority() const

{

return x;

}

namespace heap

{

// Функция создания кучи

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

{

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

}

// Метод класса Heap для получения индекса левого потомка

int Heap::left(int ix)

{

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

}

// Метод класса Heap для получения индекса правого потомка

int Heap::right(int ix)

{

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

}

// Метод класса Heap для получения индекса родительского узла

int Heap::parent(int ix)

{

return (ix + 1) / 2 - 1;

}

// Метод класса Heap для обмена элементов

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

{

void\* buf = storage[i];

storage[i] = storage[j];

storage[j] = buf;

}

// Метод класса Heap для преобразования в кучу с корнем в ix

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 класса Heap для вставки 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 класса Heap для извлечения максимального элемента

void\* Heap::extractMax()

{

void\* rc = nullptr;

if (!isEmpty())

{

rc = storage[0];

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

size--;

heapify(0);

} return rc;

}

// Метод класса Heap для извлечения минимального элемента

void\* Heap::extractMin()

{

void\* rc = nullptr;

if (!isEmpty())

{

rc = storage[size - 1];

storage[size - 2] = storage[size - 1];

size--;

heapify(0);

}

return rc;

}

// Метод класса Heap для вывода значений элементов на экран

void Heap::scan(int i) const

{

int probel = 20;

std::cout << '\n';

if (size == 0)

std::cout << "Куча пуста";

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

{

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

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

if (u == y)

{

std::cout << '\n';

if (y == 0)

y = 2;

else

y += y \* 2;

}

probel /= 2;

}

std::cout << '\n';

}

// Метод класса Heap для преобразования в мин-кучу с корнем в ix

void Heap::minHeapify(int ix)

{

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

if (l > 0 && isLess(storage[l], storage[ix])) irm = l;

if (r > 0 && isLess(storage[r], storage[irm])) irm = r;

if (irm != ix)

{

swap(ix, irm);

minHeapify(irm);

}

}

// Метод класса Heap для удаления элемента по индексу ix

void Heap::deleteI(int ix)

{

if (ix >= size) return;

if (ix == size - 1)

{

size--;

return;

}

swap(ix, size - 1);

size--;

minHeapify(ix);

}

// Метод класса Heap для объединения кучи с другой кучей other

void Heap::merge(Heap& other)

{

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

{

insert(other.storage[i]);

}

other.size = 0;

}

// Метод класса Heap для объединения кучи с другой кучей other и освобождения памяти other

void Heap::unionHeap(Heap& other)

{

merge(other);

delete[] other.storage;

}

}

using namespace std;

// Функция сравнения для класса AAA

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

{

#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");

// Создание кучи

heap::Heap h = heap::create(100, cmpAAA);

while (true) {

int choice;

cout << "\nМеню:\n";

cout << "1. Вывести кучу\n";

cout << "2. Добавить элемент в кучу\n";

cout << "3. Удалить минимальный/максимальный/элемент по индексу\n";

cout << "4. Объединить кучи\n";

cout << "5. Выход\n";

cout << "Введите ваш выбор: ";

cin >> choice;

switch (choice) {

case 1:

h.scan(0);

break;

case 2: {

int priority;

cout << "Введите приоритет нового элемента: ";

cin >> priority;

AAA\* newElem = new AAA;

newElem->x = priority;

h.insert(newElem);

break;

}

case 3:

int removeChoice;

cout << "1. Удалить минимальный элемент\n";

cout << "2. Удалить максимальный элемент\n";

cout << "3. Удалить элемент по индексу\n";

cout << "Введите ваш выбор: ";

cin >> removeChoice;

switch (removeChoice) {

case 1:

h.extractMin();

break;

case 2:

h.extractMax();

break;

case 3:

int index;

cout << "Введите индекс элемента для удаления: ";

cin >> index;

h.deleteI(index);

break;

default:

cout << "Некорректный выбор\n";

break;

}

break;

case 4: {

int heapSize;

cout << "Введите размер новой кучи: ";

cin >> heapSize;

heap::Heap newHeap = heap::create(heapSize, cmpAAA);

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

int priority;

cout << "Введите приоритет элемента " << i + 1 << ": ";

cin >> priority;

AAA\* newElem = new AAA;

newElem->x = priority;

newHeap.insert(newElem);

}

h.unionHeap(newHeap);

break;

}

case 5:

return 0;

default:

cout << "Некорректный выбор\n";

break;

}

}

return 0;

}

**Heap.h**

#pragma once

#include <iostream>

#include <iomanip>

// Структура AAA с одним целочисленным полем x

struct AAA

{

int x;

void print(); // Метод для печати значения x

int getPriority() const; // Метод для получения приоритета

};

// Пространство имен heap для работы с кучей

namespace heap

{

// Перечисление для сравнения элементов кучи

enum CMP

{

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

};

// Структура Heap для представления кучи

struct Heap

{

int size; // Текущий размер кучи

int maxSize; // Максимальный размер кучи

void\*\* storage; // Хранилище элементов кучи

CMP(\*compare)(void\*, void\*); // Указатель на функцию сравнения

// Конструктор кучи

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

{

size = 0;

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

compare = f;

};

// Методы для вычисления индексов дочерних и родительского узла

int left(int ix);

int right(int ix);

int parent(int ix);

// Проверка, полна ли куча

bool isFull() const

{

return (size >= maxSize);

};

// Проверка, пуста ли куча

bool isEmpty() const

{

return (size <= 0);

};

// Методы сравнения элементов

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;

};

// Метод для обмена элементов местами

void swap(int i, int j);

// Метод для преобразования поддерева с корнем в узле ix в кучу

void heapify(int ix);

// Метод для вставки нового элемента в кучу

void insert(void\* x);

// Метод для извлечения максимального элемента из кучи

void\* extractMax();

// Метод для вывода значений элементов кучи на экран

void scan(int i) const;

// Метод для преобразования поддерева с корнем в узле ix в мин-кучу

void minHeapify(int ix);

// Метод для извлечения минимального элемента из кучи

void\* extractMin();

// Метод для удаления элемента по индексу ix из кучи

void deleteI(int ix);

// Метод для объединения текущей кучи с другой кучей other

void merge(Heap& other);

// Метод для объединения текущей кучи с другой кучей other и освобождения памяти other

void unionHeap(Heap& other);

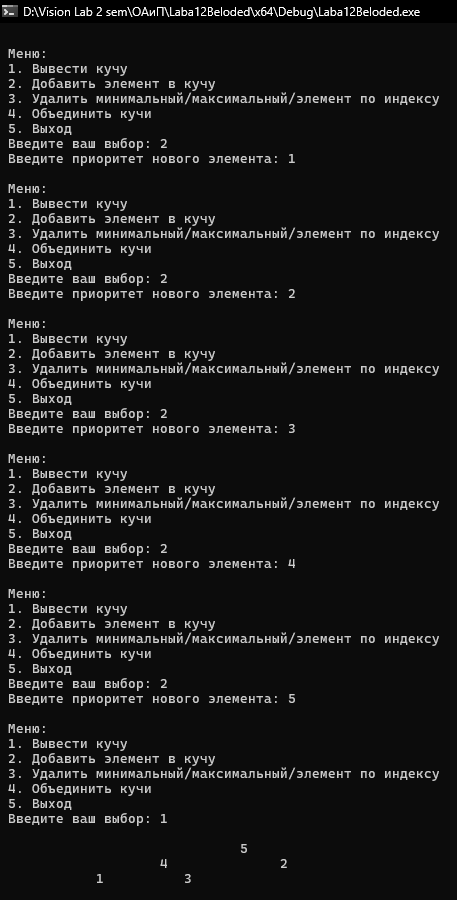
};

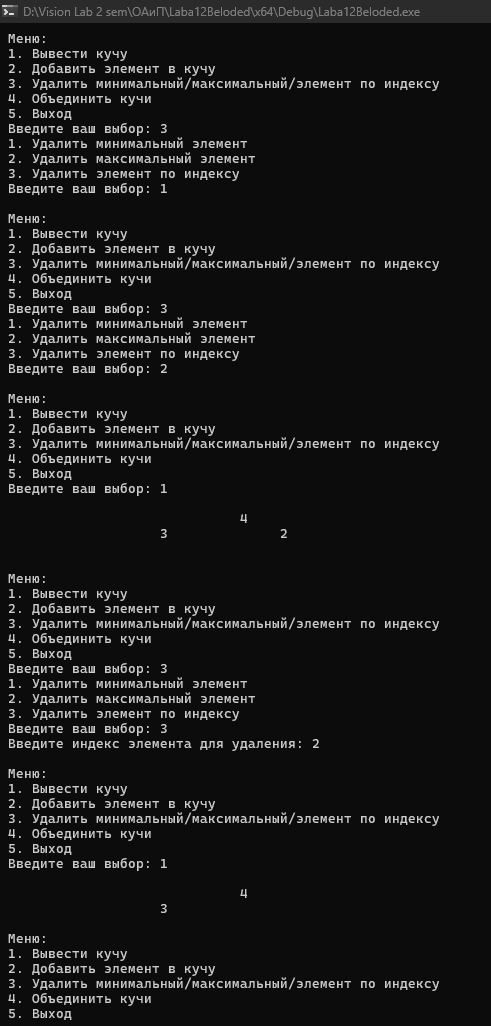
// Функция для создания кучи

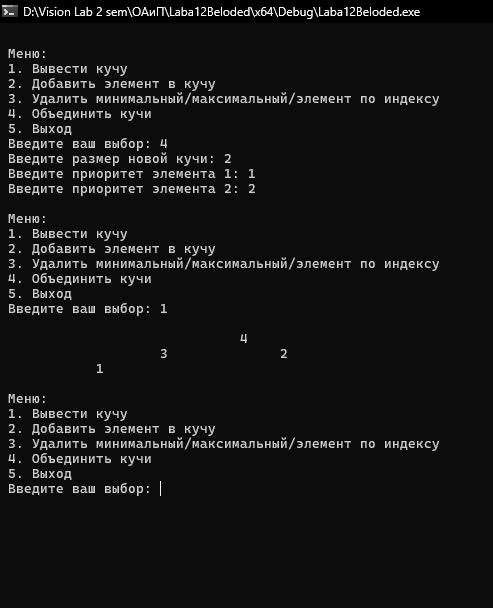
Heap create(int maxsize, CMP(\*f)(void\*, void\*));

};

**Результат выполнения:**





  
  
  
  
  
**Дополнительные задания:  
Добавил функции:**

Среднее арифметическое всех элементов:

Количество узлов с чётными ключами:

Сумма вершин кучи:

**Heap.h**

#pragma once

#include <iostream>

#include <iomanip>

// Структура AAA с одним целочисленным полем x

struct AAA

{

int x;

void print(); // Метод для печати значения x

int getPriority() const; // Метод для получения приоритета

};

// Пространство имен heap для работы с кучей

namespace heap

{

// Перечисление для сравнения элементов кучи

enum CMP

{

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

};

// Структура Heap для представления кучи

struct Heap

{

int size; // Текущий размер кучи

int maxSize; // Максимальный размер кучи

void\*\* storage; // Хранилище элементов кучи

CMP(\*compare)(void\*, void\*); // Указатель на функцию сравнения

// Конструктор кучи

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

{

size = 0;

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

compare = f;

};

// Методы для вычисления индексов дочерних и родительского узла

int left(int ix);

int right(int ix);

int parent(int ix);

// Проверка, полна ли куча

bool isFull() const

{

return (size >= maxSize);

};

// Проверка, пуста ли куча

bool isEmpty() const

{

return (size <= 0);

};

// Методы сравнения элементов

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;

};

// Метод для обмена элементов местами

void swap(int i, int j);

// Метод для преобразования поддерева с корнем в узле ix в кучу

void heapify(int ix);

// Метод для вставки нового элемента в кучу

void insert(void\* x);

// Метод для извлечения максимального элемента из кучи

void\* extractMax();

// Метод для вывода значений элементов кучи на экран

void scan(int i) const;

// Метод для преобразования поддерева с корнем в узле ix в мин-кучу

void minHeapify(int ix);

// Метод для извлечения минимального элемента из кучи

void\* extractMin();

// Метод для удаления элемента по индексу ix из кучи

void deleteI(int ix);

// Метод для объединения текущей кучи с другой кучей other

void merge(Heap& other);

// Метод для объединения текущей кучи с другой кучей other и освобождения памяти other

void unionHeap(Heap& other);

double average() const;

int countEvenKeys() const;

void printSum() const;

};

// Функция для создания кучи

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

};

**Heap.cpp**

#include "heap.h"

// Метод класса AAA для вывода ключа

void AAA::print()

{

std::cout << x;

}

// Метод класса AAA для получения приоритета

int AAA::getPriority() const

{

return x;

}

namespace heap

{

// Функция создания кучи

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

{

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

}

// Метод класса Heap для получения индекса левого потомка

int Heap::left(int ix)

{

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

}

// Метод класса Heap для получения индекса правого потомка

int Heap::right(int ix)

{

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

}

// Метод класса Heap для получения индекса родительского узла

int Heap::parent(int ix)

{

return (ix + 1) / 2 - 1;

}

// Метод класса Heap для обмена элементов

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

{

void\* buf = storage[i];

storage[i] = storage[j];

storage[j] = buf;

}

// Метод класса Heap для преобразования в кучу с корнем в ix

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 класса Heap для вставки 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 класса Heap для извлечения максимального элемента

void\* Heap::extractMax()

{

void\* rc = nullptr;

if (!isEmpty())

{

rc = storage[0];

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

size--;

heapify(0);

} return rc;

}

// Метод класса Heap для извлечения минимального элемента

void\* Heap::extractMin()

{

void\* rc = nullptr;

if (!isEmpty())

{

rc = storage[size - 1];

storage[size - 2] = storage[size - 1];

size--;

heapify(0);

}

return rc;

}

// Метод класса Heap для вывода значений элементов на экран

void Heap::scan(int i) const

{

int probel = 20;

std::cout << '\n';

if (size == 0)

std::cout << "Куча пуста";

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

{

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

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

if (u == y)

{

std::cout << '\n';

if (y == 0)

y = 2;

else

y += y \* 2;

}

probel /= 2;

}

std::cout << '\n';

}

// Метод класса Heap для преобразования в мин-кучу с корнем в ix

void Heap::minHeapify(int ix)

{

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

if (l > 0 && isLess(storage[l], storage[ix])) irm = l;

if (r > 0 && isLess(storage[r], storage[irm])) irm = r;

if (irm != ix)

{

swap(ix, irm);

minHeapify(irm);

}

}

// Метод класса Heap для удаления элемента по индексу ix

void Heap::deleteI(int ix)

{

if (ix >= size) return;

if (ix == size - 1)

{

size--;

return;

}

swap(ix, size - 1);

size--;

minHeapify(ix);

}

// Метод класса Heap для объединения кучи с другой кучей other

void Heap::merge(Heap& other)

{

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

{

insert(other.storage[i]);

}

other.size = 0;

}

// Метод класса Heap для объединения кучи с другой кучей other и освобождения памяти other

void Heap::unionHeap(Heap& other)

{

merge(other);

delete[] other.storage;

}

// Метод для подсчета среднего арифметического всех элементов

double Heap::average() const {

double sum = 0;

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

sum += ((AAA\*)storage[i])->x;

}

return sum / size;

}

// Метод для подсчета количества узлов с четными ключами

int Heap::countEvenKeys() const {

int count = 0;

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

if (((AAA\*)storage[i])->x % 2 == 0) {

++count;

}

}

return count;

}

// Метод для вывода суммы всех вершин кучи

void Heap::printSum() const {

int sum = 0;

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

sum += ((AAA\*)storage[i])->x;

}

std::cout << "Сумма всех вершин кучи: " << sum << std::endl;

}

}

using namespace std;

// Функция сравнения для класса AAA

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

{

#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");

// Создание кучи

heap::Heap h = heap::create(100, cmpAAA);

while (true) {

int choice;

cout << "\nМеню:\n";

cout << "1. Вывести кучу\n";

cout << "2. Добавить элемент в кучу\n";

cout << "3. Удалить минимальный/максимальный/элемент по индексу\n";

cout << "4. Объединить кучи\n";

cout << "5. Среднее арифметическое всех элементов:\n Количество узлов с чётными ключами:\n Сумма вершин кучи:\n";

cout << "6. Выход\n";

cout << "Введите ваш выбор: ";

cin >> choice;

switch (choice) {

case 1:

h.scan(0);

break;

case 2: {

int priority;

cout << "Введите приоритет нового элемента: ";

cin >> priority;

AAA\* newElem = new AAA;

newElem->x = priority;

h.insert(newElem);

break;

}

case 3:

int removeChoice;

cout << "1. Удалить минимальный элемент\n";

cout << "2. Удалить максимальный элемент\n";

cout << "3. Удалить элемент по индексу\n";

cout << "Введите ваш выбор: ";

cin >> removeChoice;

switch (removeChoice) {

case 1:

h.extractMin();

break;

case 2:

h.extractMax();

break;

case 3:

int index;

cout << "Введите индекс элемента для удаления: ";

cin >> index;

h.deleteI(index);

break;

default:

cout << "Некорректный выбор\n";

break;

}

break;

case 4: {

int heapSize;

cout << "Введите размер новой кучи: ";

cin >> heapSize;

heap::Heap newHeap = heap::create(heapSize, cmpAAA);

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

int priority;

cout << "Введите приоритет элемента " << i + 1 << ": ";

cin >> priority;

AAA\* newElem = new AAA;

newElem->x = priority;

newHeap.insert(newElem);

}

h.unionHeap(newHeap);

break;

}

case 5: {

h.scan(0);

cout << "Среднее арифметическое всех элементов: " << h.average() << endl;

cout << "Количество узлов с четными ключами: " << h.countEvenKeys() << endl;

h.printSum();

break;

}

case 6:

return 0;

default:

cout << "Некорректный выбор\n";

break;

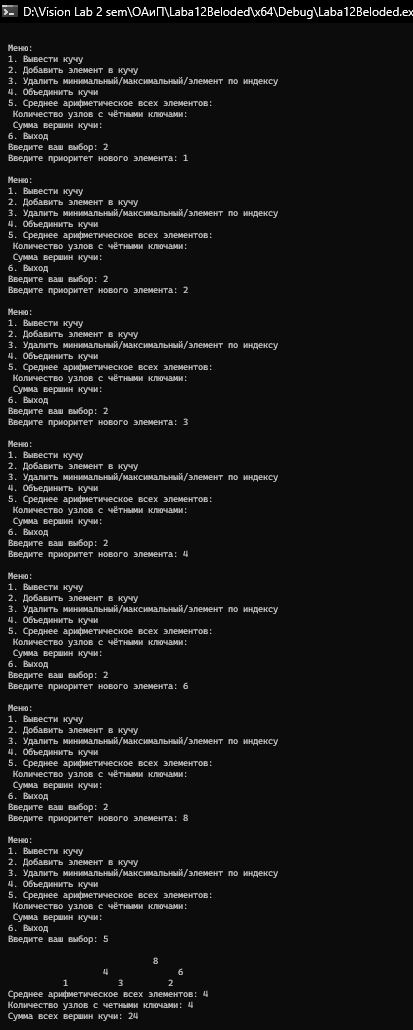
}

}

return 0;

}

**Результат выполнения:**

****