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

Доп. задачи

Взяты из лабораторной работы № 11

**Вариант 14**

Вершина бинарного дерева содержит ключ и два указателя на потомков. Написать функцию вычисления среднего арифметического всех элементов дерева.

#include <iostream>

#include <vector>

using namespace std;

template<typename T>

class BinaryHeap

{

private:

vector<T> heap;

// Просеивание элемента вверх

void siftUp(int index)

{

if (index == 0)

return;

int parentIndex = (index - 1) / 2;

if (heap[index] > heap[parentIndex])

{

swap(heap[index], heap[parentIndex]);

siftUp(parentIndex);

}

}

// Просеивание элемента вниз

void siftDown(int index)

{

int leftChild = 2 \* index + 1;

int rightChild = 2 \* index + 2;

int largest = index;

if (leftChild < heap.size() && heap[leftChild] > heap[largest])

largest = leftChild;

if (rightChild < heap.size() && heap[rightChild] > heap[largest])

largest = rightChild;

if (largest != index)

{

swap(heap[index], heap[largest]);

siftDown(largest);

}

}

public:

// Вставка элемента в кучу

void insert(T data)

{

heap.push\_back(data);

siftUp(heap.size() - 1);

}

// Извлечение максимального элемента из кучи

T extractMax()

{

if (heap.empty())

{

cerr << "Пустая куча." << endl;

exit(1);

}

T maxElement = heap[0];

heap[0] = heap.back();

heap.pop\_back();

siftDown(0);

return maxElement;

}

// Проверка, является ли куча пустой

bool isEmpty() const

{

return heap.empty();

}

};

void average()

{

BinaryHeap<int> heap;

// Вставка элементов в кучу

heap.insert(5);

heap.insert(10);

heap.insert(15);

heap.insert(20);

heap.insert(25);

heap.insert(30);

heap.insert(35);

heap.insert(40);

heap.insert(45);

int sum = 0;

int count = 0;

// Извлечение элементов из кучи и вычисление суммы и количества

while (!heap.isEmpty())

{

int num = heap.extractMax();

sum += num;

count++;

}

// Вычисление среднего арифметического

if (count > 0)

{

double average = static\_cast<double>(sum) / count;

cout << "Среднее арифметическое: " << average << endl;

}

else

{

cout << "Пустая куча." << endl;

}

}

int main()

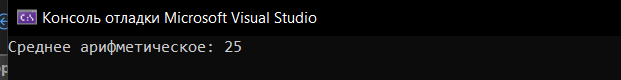
{

setlocale(0, "ru");

average();

return 0;

}

****

**Вариант 8**

Вершина бинарного дерева содержит ключ, 2 целых числа и два указателя на потомков. Написать функцию удаления вершины с максимальной суммой 2 целых значений узла.

#include <iostream>

#include <vector>

using namespace std;

// Создаем структуру Node для хранения данных

struct Node

{

int dataF;

int dataS;

int key;

};

// Создаем класс BinaryHeap

class BinaryHeap

{

private:

vector<Node> heap; // Вектор для хранения кучи

// Методы для работы с кучей

void heapifyUp(int index);

void heapifyDown(int index);

public:

// Методы класса

bool isEmpty() const;

void insert(int dataF, int dataS, int key);

void removeMax();

void print() const;

};

// Метод восстановления свойств кучи после вставки элемента

void BinaryHeap::heapifyUp(int index)

{

if (index == 0)

return;

int parentIndex = (index - 1) / 2;

if (heap[index].dataF + heap[index].dataS > heap[parentIndex].dataF + heap[parentIndex].dataS)

{

swap(heap[index], heap[parentIndex]);

heapifyUp(parentIndex);

}

}

// Метод восстановления свойств кучи после удаления элемента

void BinaryHeap::heapifyDown(int index)

{

int leftChildIndex = 2 \* index + 1;

int rightChildIndex = 2 \* index + 2;

int largestIndex = index;

if (leftChildIndex < heap.size() && heap[leftChildIndex].dataF + heap[leftChildIndex].dataS > heap[largestIndex].dataF + heap[largestIndex].dataS)

largestIndex = leftChildIndex;

if (rightChildIndex < heap.size() && heap[rightChildIndex].dataF + heap[rightChildIndex].dataS > heap[largestIndex].dataF + heap[largestIndex].dataS)

largestIndex = rightChildIndex;

if (largestIndex != index)

{

swap(heap[index], heap[largestIndex]);

heapifyDown(largestIndex);

}

}

// Метод проверки, является ли куча пустой

bool BinaryHeap::isEmpty() const

{

return heap.empty();

}

// Метод добавления нового элемента в кучу

void BinaryHeap::insert(int dataF, int dataS, int key)

{

heap.push\_back(Node{ dataF, dataS, key });

heapifyUp(heap.size() - 1);

}

// Метод удаления элемента с максимальной суммой

void BinaryHeap::removeMax()

{

if (isEmpty())

{

cout << "Куча пуста." << endl;

return;

}

swap(heap[0], heap[heap.size() - 1]);

heap.pop\_back();

heapifyDown(0);

}

// Метод вывода элементов кучи

void BinaryHeap::print() const

{

if (isEmpty())

{

cout << "Куча пуста." << endl;

return;

}

for (const auto& node : heap)

{

cout << node.key << " (" << node.dataF << " " << node.dataS << ")" << endl;

}

}

// Удаление вершины с максимальной суммой 2 целых значений узла

void deleteMax()

{

BinaryHeap heap;

heap.insert(10, 11, 19);

heap.insert(16, 15, 21);

heap.insert(13, 5, 17);

heap.insert(14, 16, 23);

heap.insert(11, 6, 15);

heap.insert(12, 18, 25);

heap.insert(17, 7, 13);

heap.insert(17, 7, 27);

heap.insert(17, 7, 11);

heap.print();

cout << endl;

heap.removeMax();

cout << endl;

heap.print();

}

int main()

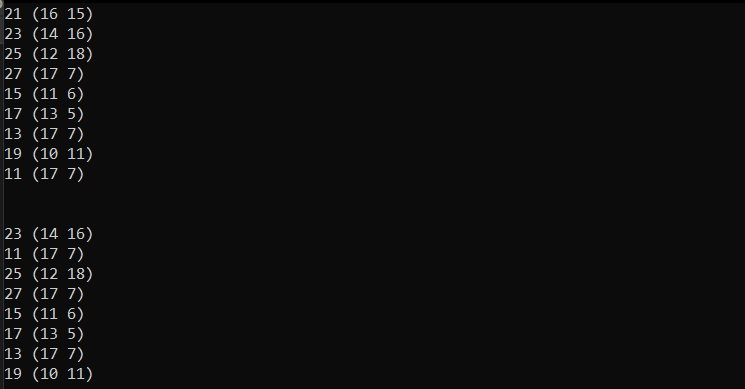
{

setlocale(0, "ru");

deleteMax();

return 0;

}

****

**Вариант 3**

Вершина бинарного дерева содержит ключ, строку и два указателя на потомков. Написать функцию вывода всех элементов дерева по уровням: корень дерева, вершины 1-го уровня, вершины 2-го уровня, ... и т. д.

#include <iostream>

#include <vector>

using namespace std;

// Создаем класс BinaryHeap

template<typename T>

class BinaryHeap

{

private:

template<typename T>

class Node

{

public:

T data;

int key;

int level; // Новое поле для хранения уровня узла

Node(T data, int key, int level)

{

this->data = data;

this->key = key;

this->level = level;

}

};

Node<T>\*\* heapArray; // Массив для хранения элементов кучи

int capacity; // Максимальная вместимость кучи

int size; // Текущий размер кучи

public:

BinaryHeap(int capacity);

~BinaryHeap();

void insert(T data, int key);

void print();

};

template<typename T>

BinaryHeap<T>::BinaryHeap(int capacity)

{

this->capacity = capacity;

this->size = 0;

this->heapArray = new Node<T>\*[capacity]; // Выделение памяти для массива кучи

}

template<typename T>

BinaryHeap<T>::~BinaryHeap()

{

delete[] heapArray; // Освобождение памяти, занятой массивом кучи

}

template<typename T>

void BinaryHeap<T>::insert(T data, int key)

{

if (size >= capacity)

{

cout << "Куча заполнена!" << endl;

return;

}

int level = 0;

if (size > 0)

{

level = heapArray[(size - 1) / 2]->level + 1; // Вычисление уровня нового узла

}

Node<T>\* newNode = new Node<T>(data, key, level); // Создание нового узла

heapArray[size] = newNode; // Добавление узла в массив кучи

int current = size;

int parent = (current - 1) / 2;

// Восстановление свойств кучи: перемещение нового узла вверх по дереву

while (current > 0 && heapArray[parent]->key < newNode->key)

{

heapArray[current] = heapArray[parent];

current = parent;

parent = (parent - 1) / 2;

}

heapArray[current] = newNode;

size++;

}

template<typename T>

void BinaryHeap<T>::print()

{

vector<vector<Node<T>\*>> levels; // Вектор уровней

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

{

int level = heapArray[i]->level;

if (level >= levels.size())

{

levels.resize(level + 1); // Если уровень еще не существует, добавляем его

}

levels[level].push\_back(heapArray[i]); // Добавляем узел в соответствующий уровень

}

// Вывод уровней по порядку

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

{

for (int j = 0; j < levels[i].size(); j++)

{

if (i == 0)

{

cout << "Корень: " << "Ключ: " << levels[i][j]->key << ", Значение: " << levels[i][j]->data << endl;

}

else

{

cout << "Уровень: " << i << ", Ключ: " << levels[i][j]->key << ", Значение: " << levels[i][j]->data << endl;

}

}

}

}

void getHeap()

{

BinaryHeap<int> heap(10);

heap.insert(19, 5);

heap.insert(21, 10);

heap.insert(17, 15);

heap.insert(23, 20);

heap.insert(15, 25);

heap.insert(25, 30);

heap.insert(13, 35);

heap.insert(27, 40);

heap.insert(11, 45);

heap.print();

}

int main()

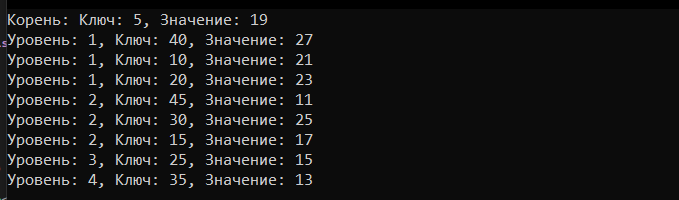
{

setlocale(0, "ru");

getHeap();

return 0;

}

****