**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Российский химико-технологический университет имени Д.И. Менделеева»**

**Кафедра информационных компьютерных технологий**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8**

Выполнил студент группы КС-30 Лобачев Дмитрий Сергеевич

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/DSLobachev\_30/blob/main/Algorithms/Laba8/Laba8.cpp

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 17.04.2023

**Оглавление**

[Описание задачи 3](#_Toc131331759)

[Описание метода/модели 4](#_Toc131331760)

[Выполнение задачи. 5](#_Toc131331761)

[Заключение 13](#_Toc131331762)

# Описание задачи

В рамках лабораторной работы необходимо реализовать бинарную кучу (мин или макс), а также 1 из ниже приведенных структур куч:

* Фибоначчиеву кучу

Для реализованных куч выполнить следующие действия:

* Наполнить кучу N кол-ва элементов (где N = 10 ^ i, i от 3 до 7).
* После заполнения кучи необходимо провести следующие тесты:
* 1000 раз найти минимум/максимум
* 1000 раз удалить минимум/максимум
* 1000 раз добавить новый элемент в кучу

Для всех операция требуется замерить время на выполнения всей 1000 операций и рассчитать время на одну операцию, а также запомнить максимальное время которое требуется на выполнение одной операции если язык позволяет его зафиксировать, если не позволяет воспользоваться хитростью и рассчитывать усредненное время на каждые 10, 25, 50, 100 операций, и выбирать максимальное из полученных результатов, чтобы поймать момент деградации структуры и ее перестройку.

По полученным в задании 2 данным построить графики времени выполнения операций для усреднения по 1000 операций, и для максимального времени на 1 операцию.

# Описание метода/модели

**Куча** — это специализированная структура данных типа дерево, которая удовлетворяет свойству кучи: если B является узлом-потомком узла A, то ключ(A) ≥ ключ(B). Из этого следует, что элемент с наибольшим ключом всегда является корневым узлом кучи, поэтому иногда такие кучи называют max-кучами (в качестве альтернативы, если сравнение перевернуть, то наименьший элемент будет всегда корневым узлом, такие кучи называют min-кучами). Не существует никаких ограничений относительно того, сколько узлов-потомков имеет каждый узел кучи, хотя на практике их число обычно не более двух.

Над кучами обычно проводятся следующие операции:

* найти максимум или найти минимум: найти максимальный элемент в max-куче или минимальный элемент в min-куче, соответственно
* удалить максимум или удалить минимум: удалить корневой узел в max- или min-куче, соответственно
* увеличить ключ или уменьшить ключ: обновить ключ в max- или min-куче, соответственно
* добавить: добавление нового ключа в кучу.
* слияние: соединение двух куч с целью создания новой кучи, содержащей все элементы обеих исходных.

**Бинарная куча** - такое двоичное дерево, для которого выполнены три условия:

1. Значение в любой вершине не меньше, чем значения её потомков.
2. Глубина всех листьев (расстояние до корня) различается не более чем на 1 слой.
3. Последний слой заполняется слева направо без «дырок».

Основной особенностью двоичной кучи является то, что каждый из узлов кучи не может иметь более чем двух потомков. Двоичная куча отлично представляется в виде одномерного массива, при этом: нулевой элемент массива всегда является вершиной кучи, а первый и второй потомок вершины с индексом i получают свои положения на основании формул: 2 \* i + 1 левый, 2 \* i + 2 правый.

**Фибоначчиевая куча** — это набор деревьев, удовлетворяющих свойству минимальной кучи, то есть ключ дочернего элемента всегда больше или равен ключу родительского элемента. Это подразумевает, что минимальный ключ всегда находится в корне одного из деревьев. Деревья не имеют заданной формы, и в крайнем случае куча может содержать каждый элемент в отдельном дереве. Эта гибкость позволяет выполнять некоторые операции ленивым образом, откладывая работу на более поздние операции.

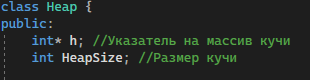
# Выполнение задачи.

Для реализации данного метода сортировки использовался язык программирования C++.

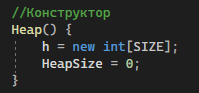
## Бинарная куча:

**Класс кучи**

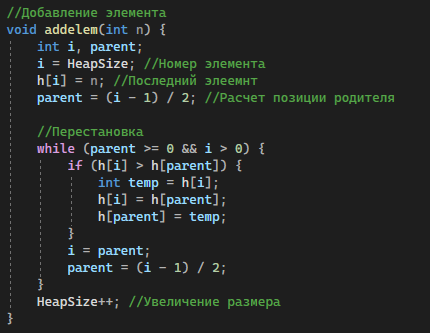
1. Класс Heap имеет поля h– указатель на массив кучи и HeapSize – размер кучи.



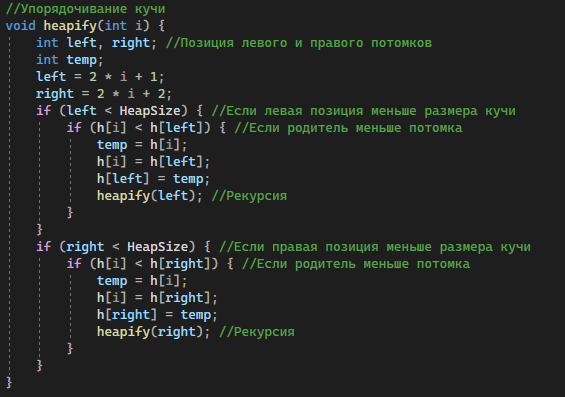
Также в классе существует базовый конструктор:



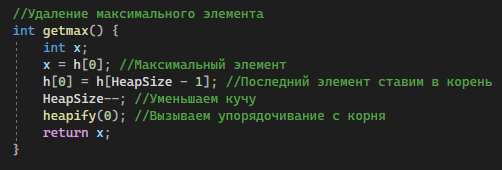
1. Функция добавления addelem(). На вход функция принимает значение нового ключа узла. Данный элемент заносится в конец массива. После этого определяется родитель этого элемента. После ряда перестановок мы получаем кучу с новым элементом.



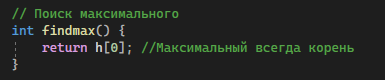
1. Функция упорядочивания heapify(). Функция принимает на вход номер узла. Далее по формулам находятся положения его левого и правого потомков. Если полученное правое или левое значение меньше размера кучи, то происходит упорядочивание. Идет проверка того, что родитель меньше своего потомка. В зависимости от того правый потомок больше или левый, происходит перестановка этого потомка и родителя. Затем для нового потомка вызывается рекурсия этой функции.



1. Функция удаления максимального элемента getmax(). После удаления элемента, на его место ставится последний элемент кучи и происходит упорядочивание.

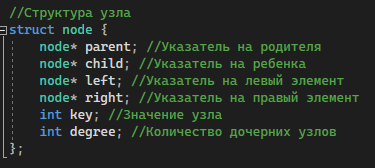


1. Функция поиска максимального элемента findmax(). В бинарной куче, первый элемент всегда является максимальным.



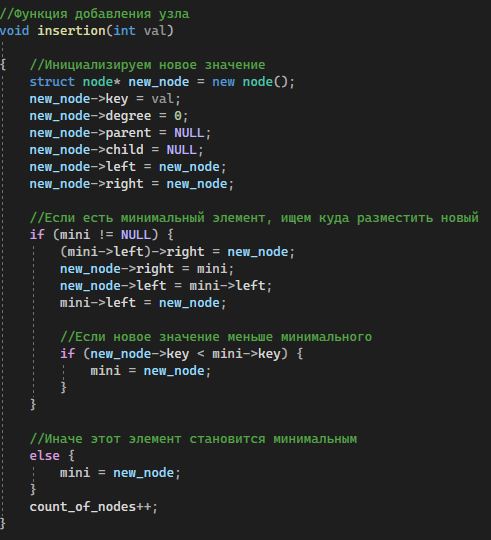
## Фибоначчиевая куча:

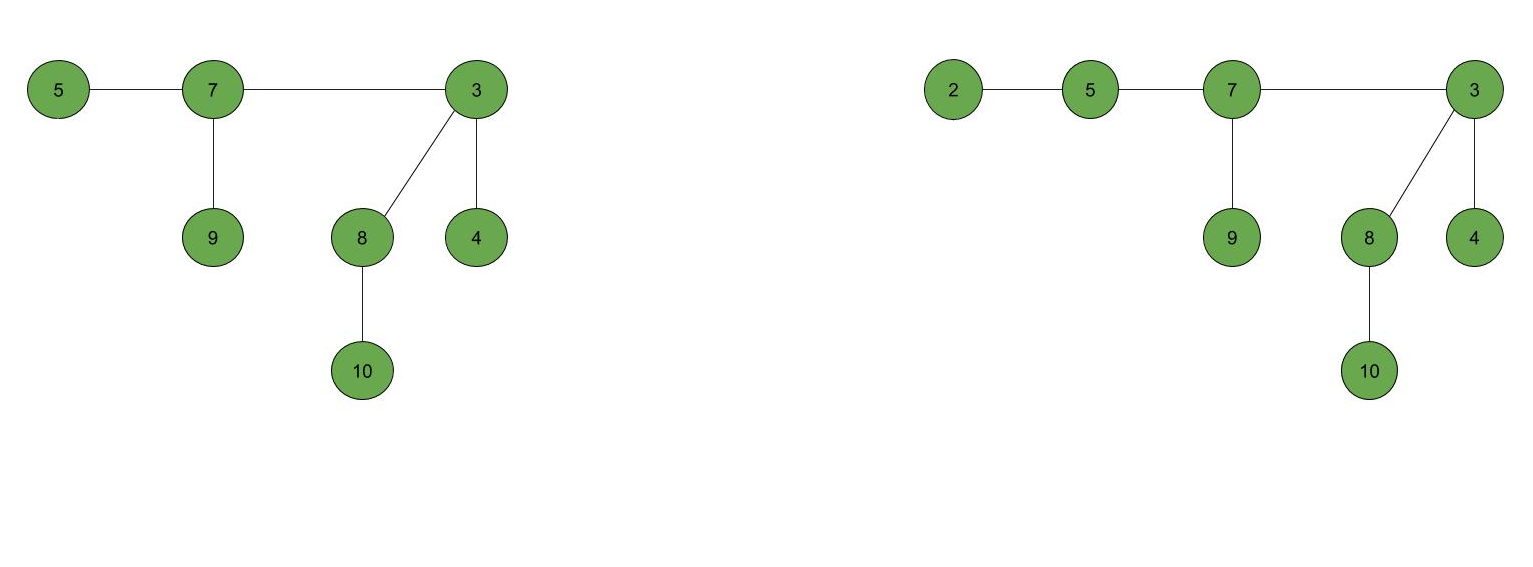
* + - 1. Структура для элементов кучи node содержит поля parent, child, left, right – указатели на соответствующие элементы, key – значение узла и degree – количество дочерних узлов.



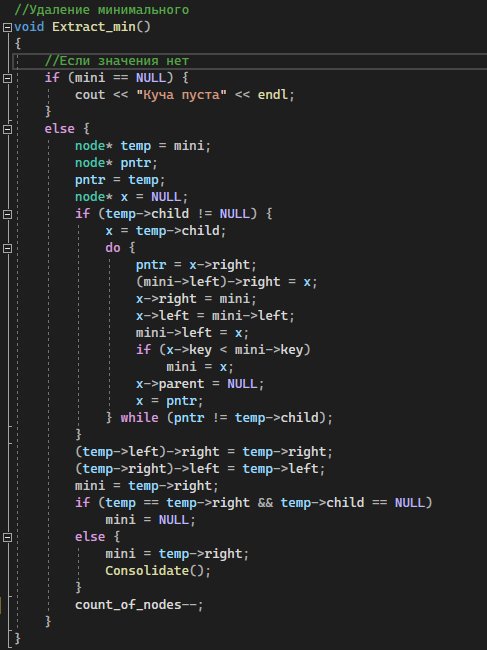
**Функции и методы**

1. Функция добавления элемента insertion(). Данная функция на вход получает значение узла. Инициализируем новый узел со значением на входе. После этого смотрим куда размещать элемент. Если это единственный элемент, то он становится минимальным. Иначе мы размещаем данный элемент левее минимального.

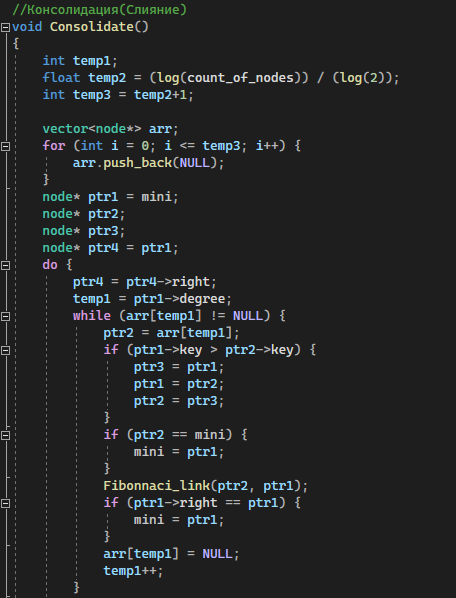


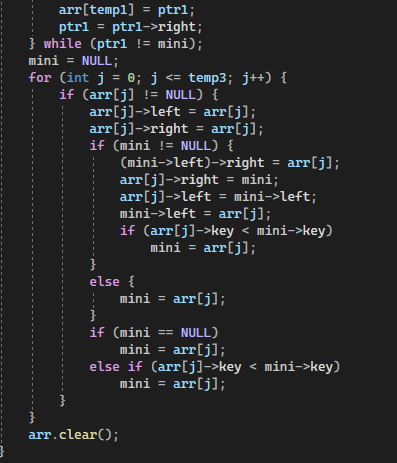


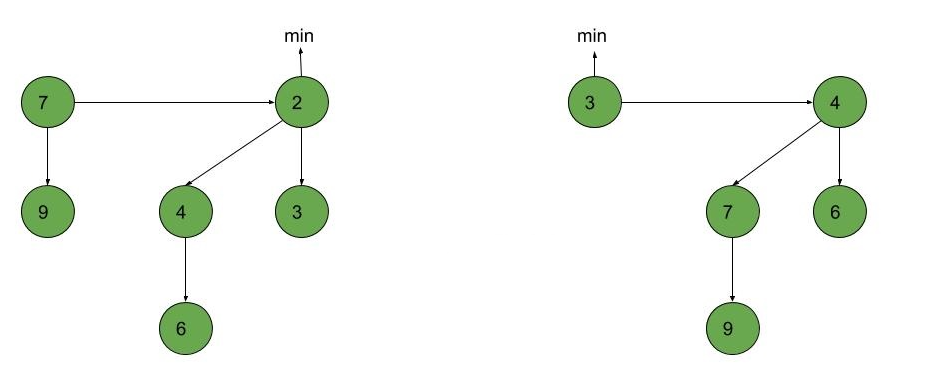
1. Функция удаления минимального значения Extract\_min(). Проверяем, имеется ли такой узел. Ищем и удаляем минимальный узел. После этого начинаем связывать все оставшиеся узлы после удаления с другими узлами.



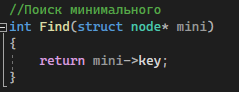








1. Функция поиска минимального элемента Find(). В фибоначчиевой куче на минимальный элемент всегда хранится отдельный указатель.



## Тестирование

В рамках лабораторной работы было проведено тестирование алгоритма на 1000 операций удаления, вставки и поиска. Для каждой из операций было замерено время. Полученные данные были представлены в виде таблиц:

***Вставка***



*Вставка в кучах*

По результатам было решено построить общие графики и график среднего времени.

***Поиск***

******

*Поиск в кучах*

По результатам было решено построить общие графики и график среднего времени.

***Удаление***



*Поиск в рандомизированном дереве*

По результатам было решено построить общий график времени.

# Заключение

Проанализировав графики, можно заметить, что скорость операций поиска и добавления в обоих кучах примерно одинакова, когда как скорость удаления элемента в фибоначчиевой куче гораздо меньше, чем в бинарной. Этой можно объяснить тем, что в фибоначчиевой куче используется больше связей в элементах, чем в бинарной, и после удаления элемента необходимо пройтись по всей куче, чтобы найти место для оставшихся элементов.