**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-36 Перминова П.А.

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Дата сдачи:

# Оглавление

[**Оглавление**](#_7v5hgj7z06u9) **2**

[**Описание задачи**](#_y7ke7xp5j57x) **3**

[**Описание метода/модели**](#_quneo8rmnu9) **4**

[**Выполнение задачи**](#_7p8elp5p0726) **9**

[**Заключение**](#_dcznlh2rz6f1) **13**

# Описание задачи

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

* Фибоначиеву кучу
* Бинарную 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 (англ. heap) — абстрактная структура данных, поддерживающая следующие операции:

* Нахождение минимума.
* Удаление минимума.
* Добавление нового элемента в кучу.

Другое название, лучше отражающее функциональность — очередь с приоритетами (англ. priority queue).

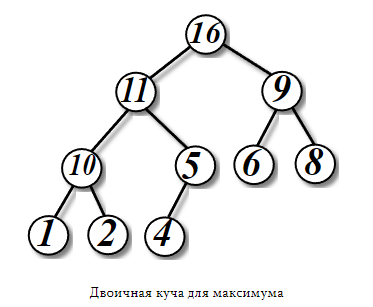
Кучи используются во многих алгоритмах. Например, кучи используются в алгоритмах поиска кратчайшего пути, а также с помощью кучи можно проводить сортировку (путём превращения массива в кучу, а кучу в отсортированный массив).

**Устройство двоичной кучи**

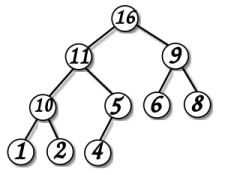
Двоичная куча (пирамида, сортирующее дерево, англ. binary heap) — реализация очереди с приоритетами, использующая корневое дерево, для которого выполнены три условия:

* Значение в любой вершине не больше, чем значения ее потомков.
* У любой вершины не более двух сыновей.
* Слои заполняются последовательно сверху вниз и слева направо, без «дырок».

Заметим, что двоичная куча строится неоднозначно: например, значения сыновей, которые являются листами, всегда можно менять местами. Фиксирована только сама структура и предикат «родитель не больше детей».



Удобная структура данных для сортирующего дерева — массив A, у которого первый элемент, A[1] — элемент в корне, а потомками элемента A[i] являются A[2i] и A[2i+1] (при нумерации элементов с первого). При нумерации элементов с нулевого, корневой элемент — A[0], а потомки элемента A[i] — A[2i+1] и A[2i+2]. При таком способе хранения условия 2 и 3 выполнены автоматически.



Высота кучи определяется как высота двоичного дерева. То есть она равна количеству рёбер в самом длинном простом пути, соединяющем корень кучи с одним из её листьев. Высота кучи есть О(logN), где N — количество узлов дерева. Над кучей можно выполнять следующие операции:

* Добавить элемент в кучу. Сложность О(logN).
* Исключить максимальный элемент из кучи. Время работы О(logN). • Изменить значение любого элемента. Время работы О(logN).

На основе этих операций можно выполнять следующие действия:

* Превратить неупорядоченный массив элементов в кучу. Сложность О(N).
* Отсортировать массив путем превращения его в кучу, а кучу в отсортированный массив. Время работы О(NlogN).

Здесь N — количество элементов кучи. Пространственная сложность — О(1) для всех вышеперечисленных операций и действий.

Подробное описание и алгоритмы этих действий и процедуры Heapify, необходимой для их выполнения, приведены ниже.

**Базовые процедуры**

**Восстановление свойств кучи**

Если в куче изменяется один из элементов, то она может перестать удовлетворять свойству упорядоченности. Для восстановления этого свойства служит процедура Heapify. Она восстанавливает свойство кучи в дереве, у которого левое и правое поддеревья удовлетворяют ему. Эта процедура принимает на вход массив элементов A и индекс i. Она

восстанавливает свойство упорядоченности во всём поддереве, корнем которого является элемент A[i].

Если i-й элемент больше, чем его сыновья, всё поддерево уже является кучей, и делать ничего не надо. В противном случае меняем местами i-й элемент с наибольшим из его сыновей, после чего выполняем Heapify для этого сына.

Процедура выполняется за время О(logN).

**Построение кучи**

Эта процедура предназначена для создания кучи из неупорядоченного массива входных данных.

Заметим, что, если выполнить Heapify для всех элементов массива A, начиная с последнего и кончая первым, он станет кучей. В самом деле, легко доказать по индукции, что к моменту выполнения Heapify(A, i) все поддеревья, чьи корни имеют индекс больше i, - кучи, и, следовательно, после выполнения Heapify(A, i) кучей будут все поддеревья, чьи корни имеют индекс, не меньший i.

Кроме того, Heapify(A,i) не делает ничего, если i>N/2 (при нумерации с первого элемента), где N — количество элементов массива. В самом деле, у таких элементов нет потомков, следовательно, соответствующие поддеревья уже являются кучами, так как содержат всего один элемент.

Таким образом, достаточно вызвать Heapify для всех элементов массива A, начиная (при нумерации с первого элемента) с [N/2]-го и кончая первым.

**Изменение значения элемента**

Процедура Heap\_Increase\_Key заменяет элемент кучи на новый ключ со значением, не меньшим значения исходного элемента. Обычно эта процедура используется для добавления произвольного элемента в кучу. Временная сложность О(NlogN).

Если элемент меньше своего отца, условие 1 соблюдено для всего дерева, и больше ничего делать не нужно. Если он больше, мы меняем местами его с отцом. Если после этого отец больше деда, мы меняем местами отца с дедом и так далее. Иными словами, слишком большой элемент всплывает наверх.

В случае, когда необходимо уменьшить значение элемента, можно вызвать Heapify.

**Добавление элемента**

Выполняет добавление элемента в кучу за время О(logN).

Добавление произвольного элемента в конец кучи, и восстановление свойства упорядоченности с помощью Heap\_Increase\_Key.

**Извлечение максимального элемента**

Выполняет извлечение максимального элемента из кучи за время О(logN).

Извлечение выполняется в четыре этапа:

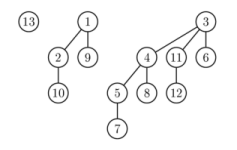
* значение корневого элемента (он и является максимальным) сохраняется для последующего возврата
* последний элемент копируется в корень, после чего удаляется из кучи • вызывается Heapify для корня
* сохранённый элемент возвращается

**Биномиальная куча**

Биномиальная куча (англ. binomial heap) — структура данных, реализующая абстрактный тип данных «очередь с приоритетом», которая представляет собой набор биномиальных деревьев с двумя свойствами:

* ключ каждой вершины не меньше ключа её родителя;
* все биномиальные деревья имеют разный размер.

Из этих свойств вытекают два следствия. Во-первых, корень каждого из деревьев имеет наименьший ключ среди его вершин. Во-вторых, суммарное количество вершин в биномиальной кучи однозначно определяет размеры входящих в неё деревьев. Например, биномиальная куча с 13 = 23 + 22 + 20 вершинами состоит из трех деревьев высотой 3, 2 и 0 и имеющих, соответственно, 8, 4 и 1 элементов (см. рис. 2)



Следующие операции выполняются за время О(logN), где N — число вершин:

Вставка нового элемента (амортизированное О(1))

* Нахождение элемента с минимальным ключом
* Удаление элемента с минимальным ключом
* Уменьшение значения ключа данного элемента
* Удаление данного элемента
* Объединение двух куч.

Таким образом, биномиальная куча является сливаемой кучей, то есть кроме стандартных операций очереди с приоритетом (добавления, удаления, извлечения минимума, изменения ключей) предоставляет дополнительную операцию слияния двух куч.

# Выполнение задачи

Программа написана на языке программирования Python.

Классы, описанные в программе:

* Node (нужен для создания узлов биномиального дерева, в нём есть конструктор, который содержит следующие поля: значение узла (ключ), список потомков и степень вершины)
* BinomialTree (класс биномиального дерева, содержит следующие методы: union(), is\_empty(), clear(), size().)
* BinomialHeap (класс, содержащий методы биномиальной кучи, содержит следующие методы: add\_element(), union(), get\_min\_tree(), remove\_min(), is\_empty(), size(), clear(), \_fix\_heap().)
* MinHeap (класс двоичной кучи, может принимать список вершин, который в конструкторе с помощью функции heapify() преобразуется в кучу.)

В основном блоке кода программы создаются объекты классов и выполняются пять тестов, засекается время для тысячи операций вставки, поиска и удаления элементов из куч, вычисляется среднее время на одну операцию, а также максимальные время одной операции вставки, поиска и удаления.

Ниже приведены графики зависимости времени от количества элементов массива для операций вставки, поиска и удаления. Использовалось среднее время выполнения тысячи операций.

График зависимости времени от количества элементов кучи (операция вставки):

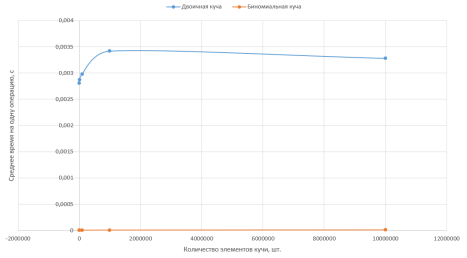


График зависимости максимального времени от количества элементов кучи (операция вставки):

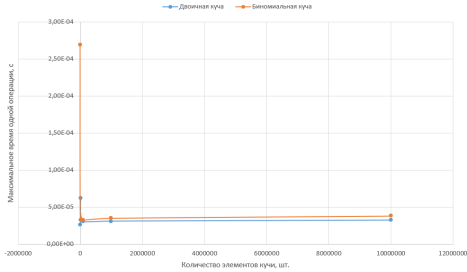


График зависимости времени от количества элементов (операция поиска):

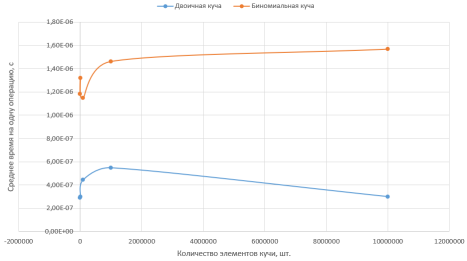


График зависимости максимального времени от количества элементов (операция поиска):

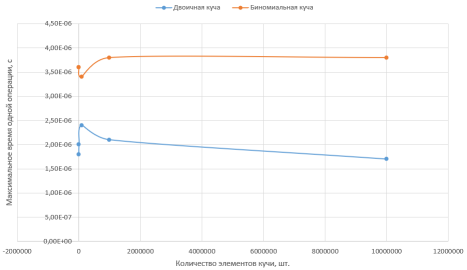
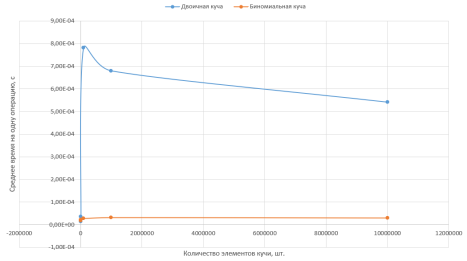
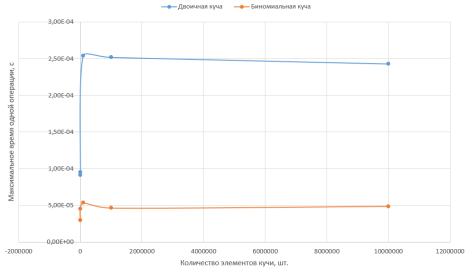


График зависимости времени от количества элементов (операция удаления):

График зависимости максимального времени от количества элементов (операция удаления):



# Заключение

В ходе лабораторной работы были реализованы двоичная и биномиальная кучи. Проанализировав результаты работы программы, можно сделать вывод, что у исследованных структур данных действительно одинаковая асимптотическая сложность выполнении операций поиска и удаления – O(logN), и при выполнении операции добавления элемента в кучу у биномиальной кучи асимптотическая сложность О(1), а у бинарного – O(logN). Проанализировав графики, можно сделать вывод, что биномиальная куча более быстродейственна при выполнении операций вставки и удаления элементов, а двоичная куча – при операции поиска.