

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Работа допущена к защите

 Руководитель

« 13 » 05 20 19 г.

КУРСОВОЙ ПРОЕКТ

по дисциплине «Алгоритмы и структуры данных»

на тему: «Приоритетные очереди: возможные реализации и сравнение подходов»

Студент  Парахин П.Г.

Шифр 160084

Учебно-научно-исследовательский институт информационных технологий

Специальность 09.03.04 «Программная инженерия»

Группа 61-ПГ

Руководитель  Фролов А.И.

Оценка: « отлично »

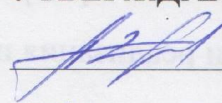
Дата 29.05.19

Орел 2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

УТВЕРЖДАЮ:

 Зав. кафедрой
«11» 03 2019г.

ЗАДАНИЕ
на курсовой проект

по дисциплине «Алгоритмы и структуры данных»

Студент Парахин П.Г.

Шифр 160084

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 61-ПГ

1 Тема курсового проекта

«Приоритетные очереди: возможные реализации и сравнение подходов»

2 Срок сдачи студентом законченной работы «11» мая 2019

3 Исходные данные

Поток заявок на обслуживание, характеризующихся длительностью обслуживания. Наивысший приоритет имеют те заявки, которые требуют наименьшего времени на обслуживание.

4 Содержание курсового проекта

Анализ и выбор методов представления приоритетной очереди

Реализация алгоритмов: добавления, удаления, получения элемента с высшим приоритетом в приоритетной очереди

Проектирование и реализация программы

Оценка сложности алгоритма

5 Отчетный материал курсового проекта

Пояснительная записка курсового проекта

Руководитель _____  Фролов А.И.

Задание принял к исполнению: «11» марта 2019

Подпись студента _____ 

Содержание

Введение	5
1 Анализ и выбор методов представления приоритетной очереди	6
2 Алгоритмы для приоритетной очереди	8
2.1 Вставка	8
2.2 Удаление	9
2.3 Получение элемента с максимальным приоритетом	11
3 Проектирование и реализация программы	12
3.1 Общие сведения	12
3.2 Описание пользовательского интерфейса	13
4 Оценка сложности и сравнительный анализ	16
Заключение	19
Список использованных материалов	20
Приложение А (обязательное) Листинг программы	21

Введение

Каждый элемент в приоритетной очереди имеет связанный с ним приоритет. Если программе нужно удалить элемент из очереди, она выбирает элемент с наивысшим приоритетом. Как хранятся элементы в приоритетной очереди, не имеет значения, если программа всегда может найти элемент с наивысшим приоритетом.

Некоторые операционные системы используют приоритетные очереди для планирования заданий. В операционной системе UNIX все процессы имеют разные приоритеты. Когда процессор освобождается, выбирается готовый к исполнению процесс с наивысшим приоритетом. Процессы с более низким приоритетом должны ждать завершения или блокировки процессов с более высокими приоритетами.

Концепция приоритетных очередей также используется при управлении авиаперевозками. Наивысший приоритет имеют самолеты, у которых кончается топливо во время посадки. Второй приоритет имеют самолеты, заходящие на посадку. Третий приоритет имеют самолеты, находящиеся на земле, так как они находятся в более безопасном положении, чем самолеты в воздухе. Приоритеты изменяются со временем, так как у самолетов, которые пытаются приземлиться, в конце концов, закончится топливо.

Целью данной курсовой работы является анализ вышеупомянутой структуры данных, а также разработка приложения, демонстрирующего работу алгоритмов добавления, удаления и получения элемента с максимальной важностью для очереди с приоритетом.

Для достижения этой цели будут решены следующие задачи:

1. Анализ структуры данных двоичная куча;
2. Разработка программного средства, демонстрирующего работу алгоритмов;

3. Оценка эффективности использованных алгоритмов;

1 Анализ и выбор методов представления приоритетной очереди

Очередь с приоритетами (англ. priority queue) — абстрактный контейнер, похожий на обычную очередь, но имеющий ряд особенностей:

- Каждому элементу очереди с приоритетами сопоставлено некоторое значение, именуемое приоритетом этого элемента. Приоритеты допускают сравнение друг с другом;
- Функция извлечения из очереди с приоритетами возвращает тот элемент, приоритет которого является максимальным.

Очередь с приоритетами, построенная на обычном массиве или списке, позволяет выполнять вставку за $O(1)$, но поиск и удаление элемента с максимальным приоритетом будут иметь сложность $O(N)$. Существует более быстрая реализация очереди с приоритетами, подразумевающая применение особого способа организации данных, получившего название двоичная куча (англ. binary heap).

Бинарное (двоичное) дерево — это дерево, в котором максимальное число прямых потомков (степень дерева) равно двум. Эти потомки называются левым и правым поддеревьями.

Двоичная куча(пирамида) — это двоичное дерево, для которого выполняются следующие условия:

1. Значение в любой вершине t_i и значение в каждом из потомков t_j удовлетворяют отношению $t_i \geq t_j$ (куча для максимума) или $t_i \leq t_j$ (куча для минимума);
2. На i -ом слое 2^i вершин, кроме последнего. Слои нумеруются с нуля;
3. Последний слой заполнен слева направо [3];

Пирамиду можно построить на массиве, используя соотношения между индексами ячеек массива для определения связей элементов. На рисунке 1 показан пример пирамиды, содержащей 8 элементов, и её отображение на массив.

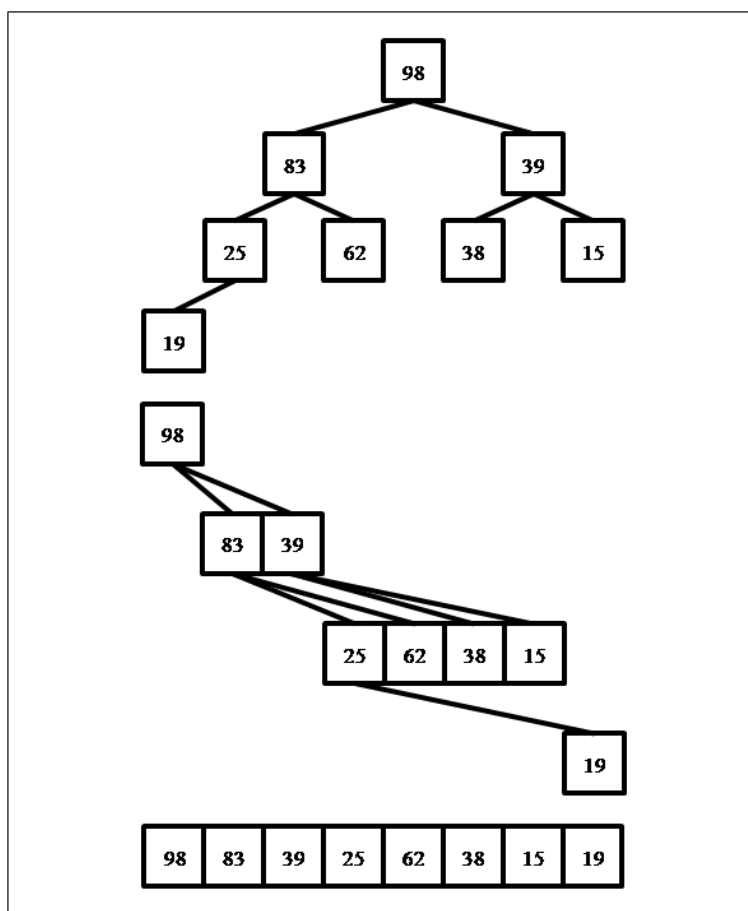


Рисунок 1 — Пример двоичной кучи и её отображение на массив

Если индексация элементов массива начинается с нуля, то непосредственные потомки элемента с индексом i будут иметь индексы $(2 * i + 1)$ и $(2 * i + 2)$, а родителем будет являться элемент с индексом $((i - 1) / 2)$. Так, потомками элемента с индексом 0 будут элементы с индексами 1 и 2, потомками элемента с индексом 1 — элементы с индексами 3 и 4, родителем элемента с индексом 7 будет элемент с индексом 3.

В данном курсовом проекте очередь с приоритетами будет представлена как двоичная куча, хранящаяся в одномерном массиве.

2 Реализация алгоритмов приоритетной очереди

2.1 Вставка

/// Добавление значения [value] в кучу

```
void add(T value) {
    this._data.add(value);
    this._siftUp(this._data.length - 1);
}
```

```
функция добавление(значение) {
    добавить_в_конец_массива(значение)
    протолкнуть_вверх(индекс_последнего_элемента_массива)
}
```

Выполняет добавление элемента в кучу за время $O(\log n)$. Добавление произвольного элемента в конец кучи, и восстановление свойства упорядоченности с помощью функции `_siftUp`.

Работа функции: если элемент больше своего отца, условие 1 соблюдено для всего дерева, и больше ничего делать не нужно. Иначе, мы меняем местами его с отцом. После чего выполняем `siftUp` для этого отца. Иными словами, слишком маленький элемент всплывает наверх. Процедура выполняется за время $O(\log n)$.

/// Восстановление свойств кучи

/// проталкиванием вверх элемента с индексом [index]

```
void _siftUp(int index) {
    while (index > 0) {
        int parent = (index - 1) ~/ 2;
        if (this._data[index] <= this._data[parent]) return;
        this._swapData(index, parent);
    }
}
```



```

        index = parent;
    }
}

функция протолкнуть_вверх(индекс) {
    пока (индекс > 0) {
        индекс_родителя = отбросить_дробную_часть((индекс - 1) / 2)
        если (массив[индекс] <= массив[индекс_родителя]) {
            возврат
        }
        обменять_местами_элементы_массива(индекс, индекс_родителя)
        индекс = индекс_родителя
    }
}

```

2.2 Удаление

/// Удаление значения из кучи

/// с индексом [index]

```

void delete(int index) {
    this._data[index] = this._data.last;
    this._data.removeLast();
    this._siftDown(index);
}

функция удалить(индекс) {
    массив[индекс] = массив.последний_элемент
    массив.удалить_последний_элемент
    протолкнуть_вниз(индекс)
}

```

Выполняет удаление элемента из кучи за время $O(\log n)$. На место элемента с заданным индексом становится последний элемент в массиве, представляющем кучу, после чего он проталкивается вниз для восстановления свойств двоичной кучи.

```

/// Восстановление свойств кучи
/// проталкиванием вниз элемента с индексом [index]
void _siftDown(int index) {
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    int largest = index;
    if (left < this._data.length &&
        this._data[left] > this._data[index]) {
        largest = left;
    }
    if (right < this._data.length &&
        this._data[right] > this._data[largest]) {
        largest = right;
    }
    if (largest != index) {
        this._swapData(index, largest);
        this._siftDown(largest);
    }
}

функция протолкнуть_вниз(индекс) {
    индекс_левый = 2 * индекс + 1
    индекс_правый = 2 * индекс + 2
    индекс_большой = индекс

```

```

если (индекс_левый < массив.длина и
    массив[индекс_левый] > массив[индекс]) {
    индекс_большой = индекс_левый
}
если (индекс_правый < массив.длина и
    массив[индекс_правый] > массив[индекс]) {
    индекс_большой = индекс_правый
}
если (индекс_большой != индекс) {
    обменять_местами_элементы_массива(индекс, индекс_большой)
    протолкнуть_вниз(индекс_большой)
}
}

```

Работа функции: если i -й элемент меньше, чем его сыновья, всё поддереву уже является кучей, и делать ничего не надо. В противном случае меняем местами i -й элемент с наименьшим из его сыновей, после чего выполняем `_siftDown` для этого сына. Процедура выполняется за время $O(\log n)$.

2.3 Получение элемента с максимальным приоритетом

/// Вернуть элемент с максимальным приоритетом

```

T get max {
    try {
        return this._data.first;
    } catch (StateError) {
        return null;
    }
}

функция получить_максимум() {

```



```
        вернуть массив.первый_элемент
    }

```

Функция работает за время $O(1)$ и возвращает первый элемент массива, хранящего элементы двоичной кучи, без удаления.

/// Извлечение максимального значения из кучи

```
T popMax() {
    if (this._data.isEmpty) return null;
    T result = this.max;
    this.delete(0);
    return result;
}

функция извлечь_максимум() {
    если (массив.пустой) {
        вернуть ничего
    }
    результат = получить_максимум()
    удалить(0)
    вернуть результат
}

```

Выполняет извлечение максимального элемента из кучи за время $O(\log n)$. Извлечение выполняется следующим образом:

- 1.Значение корневого элемента (он и является максимальным) сохраняется для последующего возврата;
- 2.Корневой элемент удаляется вызовом функции delete;

3 Проектирование и реализация программы

3.1 Общие сведения

Для создания программы был использован перспективный язык программирования Dart и платформа Flutter для создания кроссплатформенных мобильных приложений для систем Android и iOS.

Само приложение состоит из отдельных модулей. Модуль `bin_hear.dart` содержит в себе обобщённый класс `BinHear`, реализующий двоичную кучу, которая может хранить в себе любой заданный тип данных. Модуль `priority_queue.dart` содержит реализацию приоритетной очереди. Модуль `main.dart` отвечает за старт приложения. Модули `my_tab.dart` и `todo_app.dart` содержат описание графического интерфейса и реализуемый функционал приложения. В соответствии с условием задания, приложение организует работу с заметками, учитывая их приоритеты. В приложении очередь хранит заметки, которые являются строками, сама очередь является свойством класса в модуле `my_tab.dart`.

/// Очередь для хранения заметок

```
PriorityQueue<String> _queue = new PriorityQueue<String>();
```

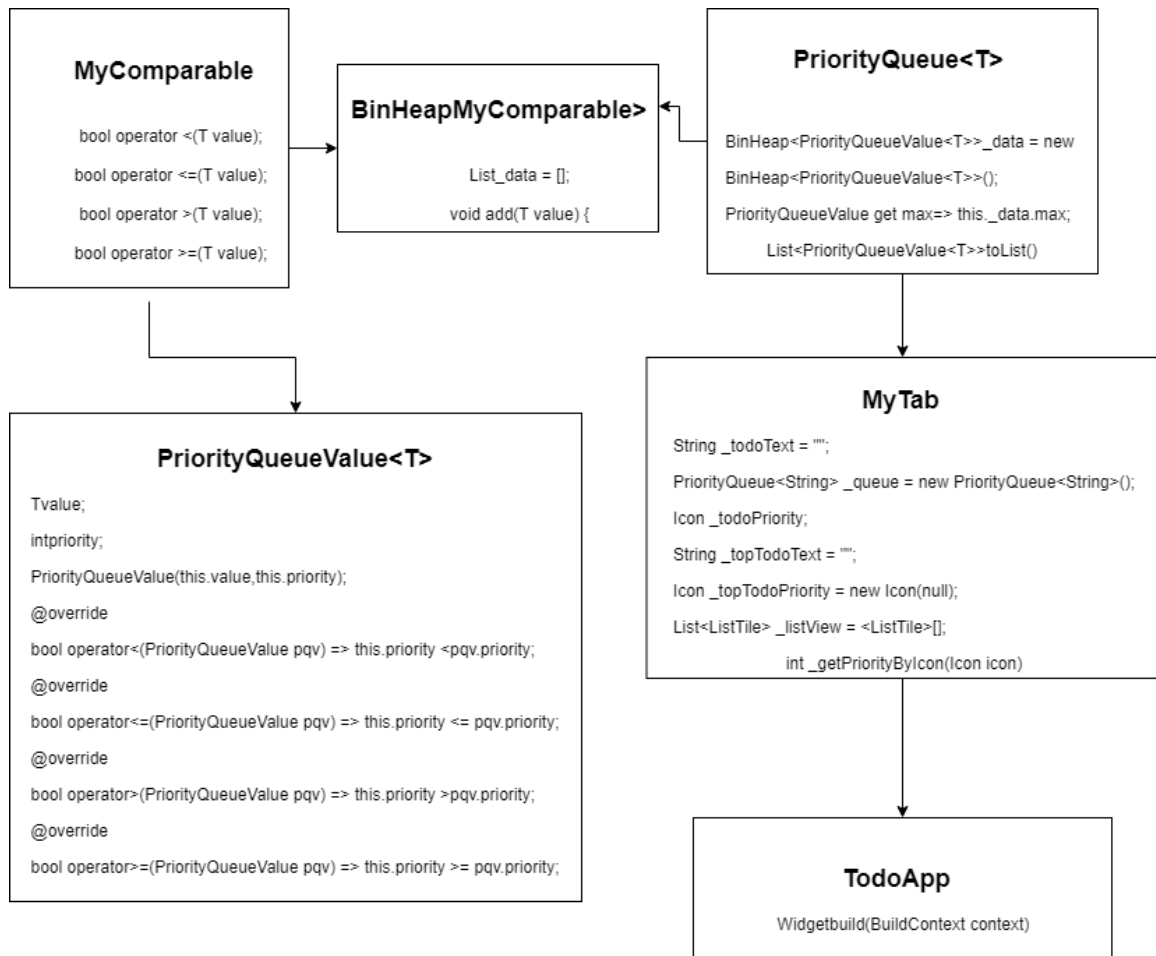


Рисунок 2 — Диаграмма классов

3.2 Описание пользовательского интерфейса

Графический пользовательский интерфейс данного приложения представлен несколькими вкладками внутри одного окна. С содержимым первой вкладки можно ознакомиться на рисунке 3.

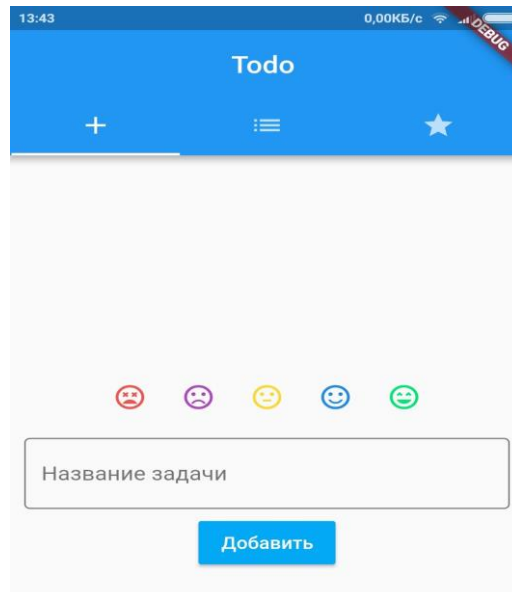


Рисунок 3 — Вкладка, содержащая форму добавления задачи

Для добавления задачи необходимо ввести её название и задач приоритет в виде смайлика, где самый высокий приоритет имеет самый левый красный и соответственно самый низкий — зелёный справа.

Процесс ввода задачи показан на рисунке 3.

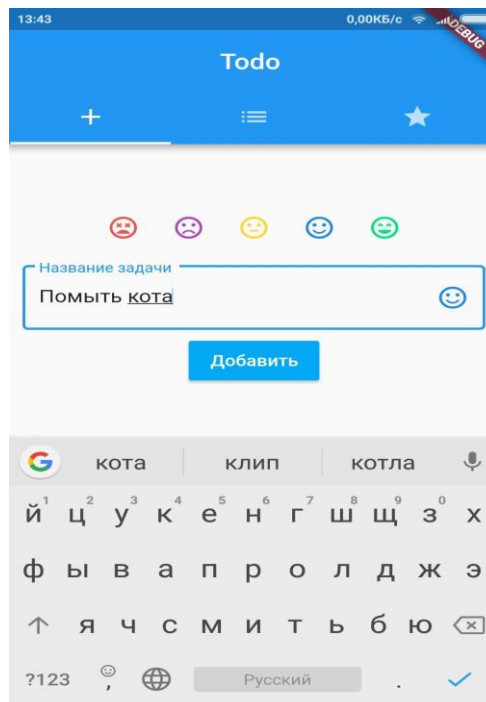


Рисунок 4 — Добавление задачи в очередь

Вторая вкладка отображает список текущих задач в произвольном порядке, пример на рисунке 5.

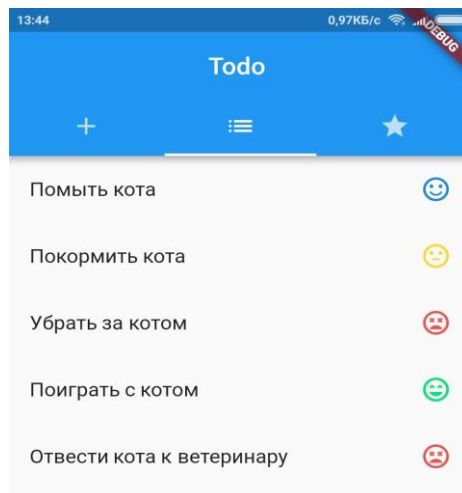


Рисунок 5 — Список введенных задач

Третья вкладка показывает задачу с самым высоким приоритетом в данный момент. Пример на рисунке 6.

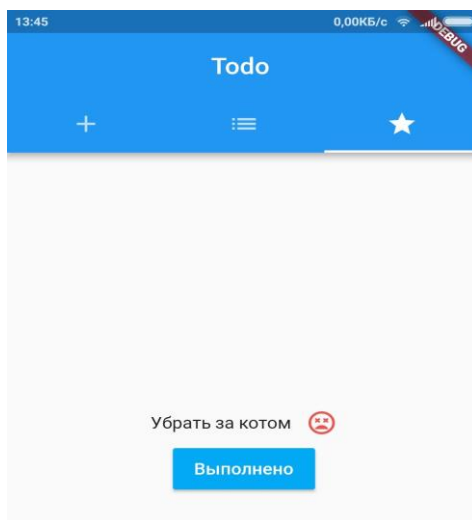


Рисунок 6 — Задача, имеющая самый высокий приоритет

4 Оценка сложности и сравнительный анализ

Экспериментальным путём были проведены замеры времени работы программы на разных объёмах данных для 3 алгоритмов: вставки, удаления, извлечения элемента с высшим приоритетом.

```
// Объём входных данных
const int MAX_N = 1000000;
const int DELTA_N = 1000;
const int MIN_N = DELTA_N;
// Функция, тестирующая алгоритм
void test() {
    final random = new Random(new DateTime.now().millisecondsSinceEpoch);
    if (new File('result.txt').existsSync()) new File('result.txt').deleteSync();
    for (int n = MIN_N; n <= MAX_N; n += DELTA_N) {
        final q = new PriorityQueue<int>();
        int t0 = Timeline.now;
        // ...
        int t1 = Timeline.now;
        new File('result.txt').writeAsStringSync(
            'n: ${n}\ntime: ${(t1 — t0) / 1000}\n', mode: APPEND
        );
    }
}
```

Функция test для каждого алгоритма отличается в нескольких строчках кода, она засекает время до начала теста и сразу же после теста, полученная разница записывается в файл, на основе которого были построены графики, показанные на рисунках: 7, 8, 9.

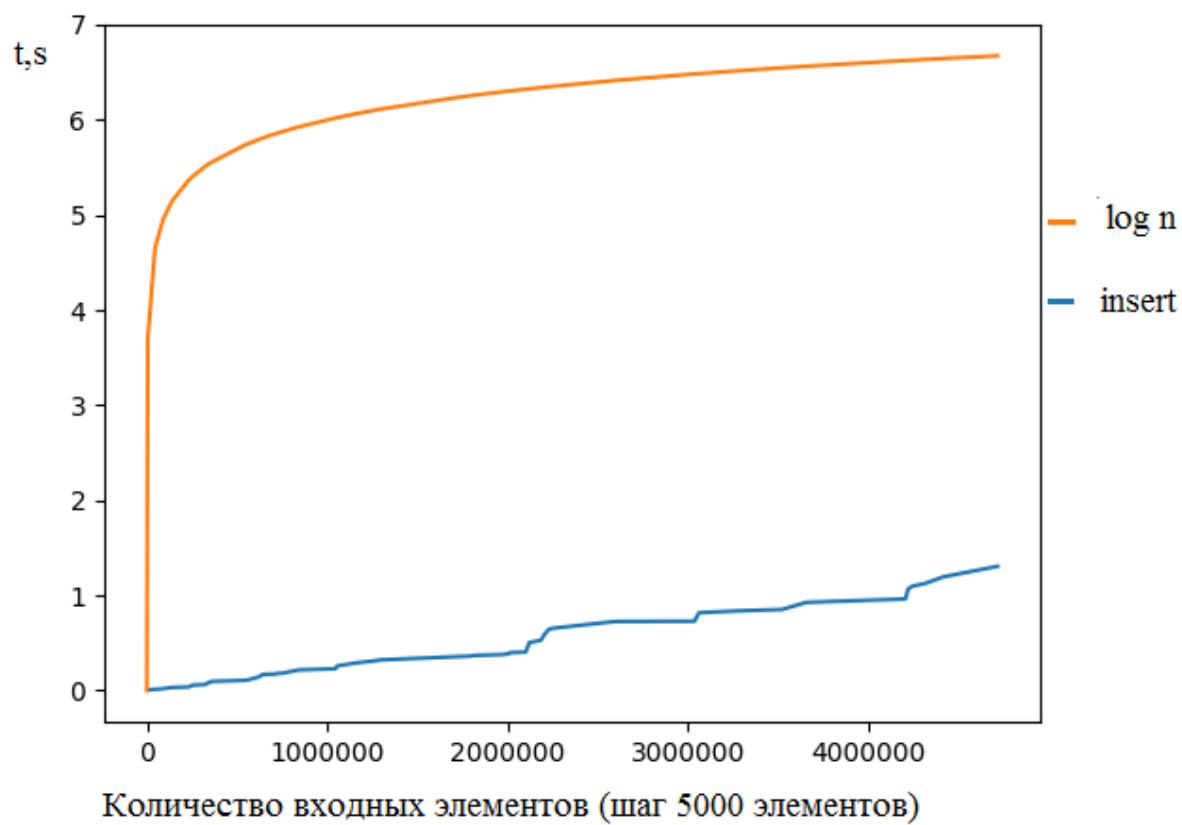


Рисунок 7 — Зависимость времени работы от входных данных для алгоритма

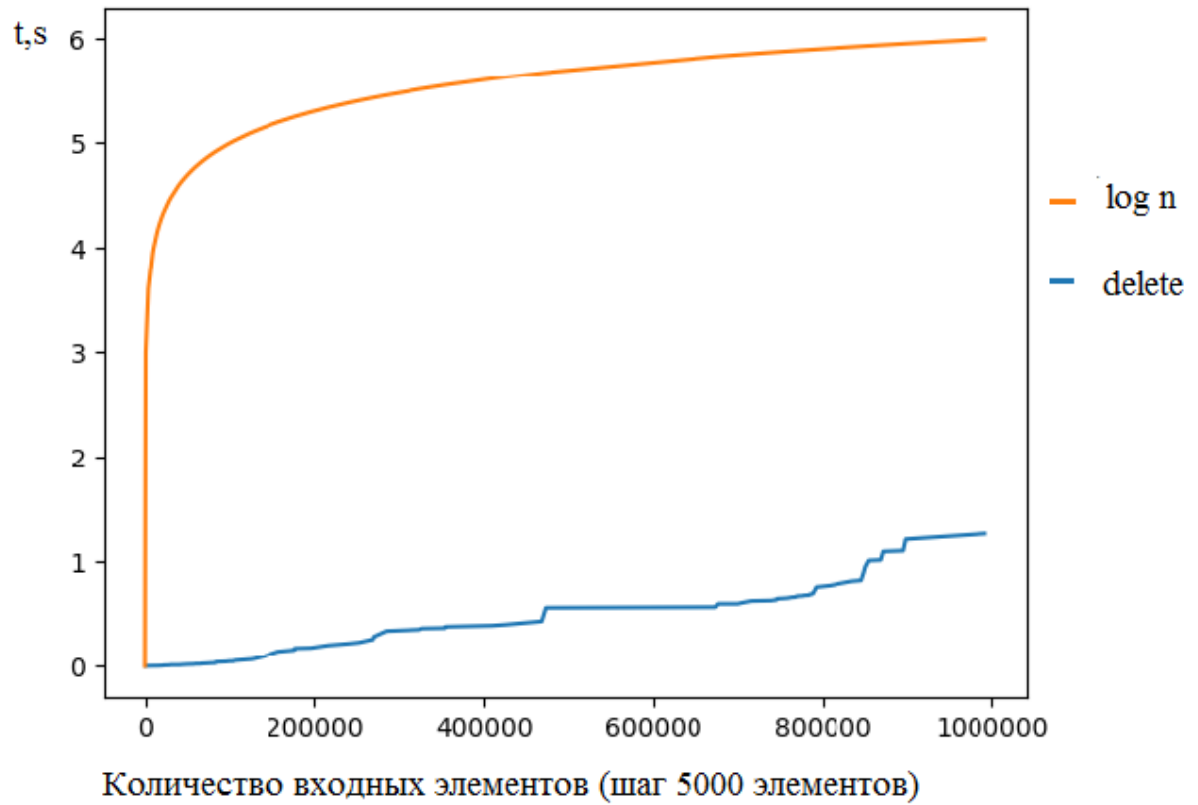


Рисунок 8 — Зависимость времени работы от входных данных для алгоритма удаления

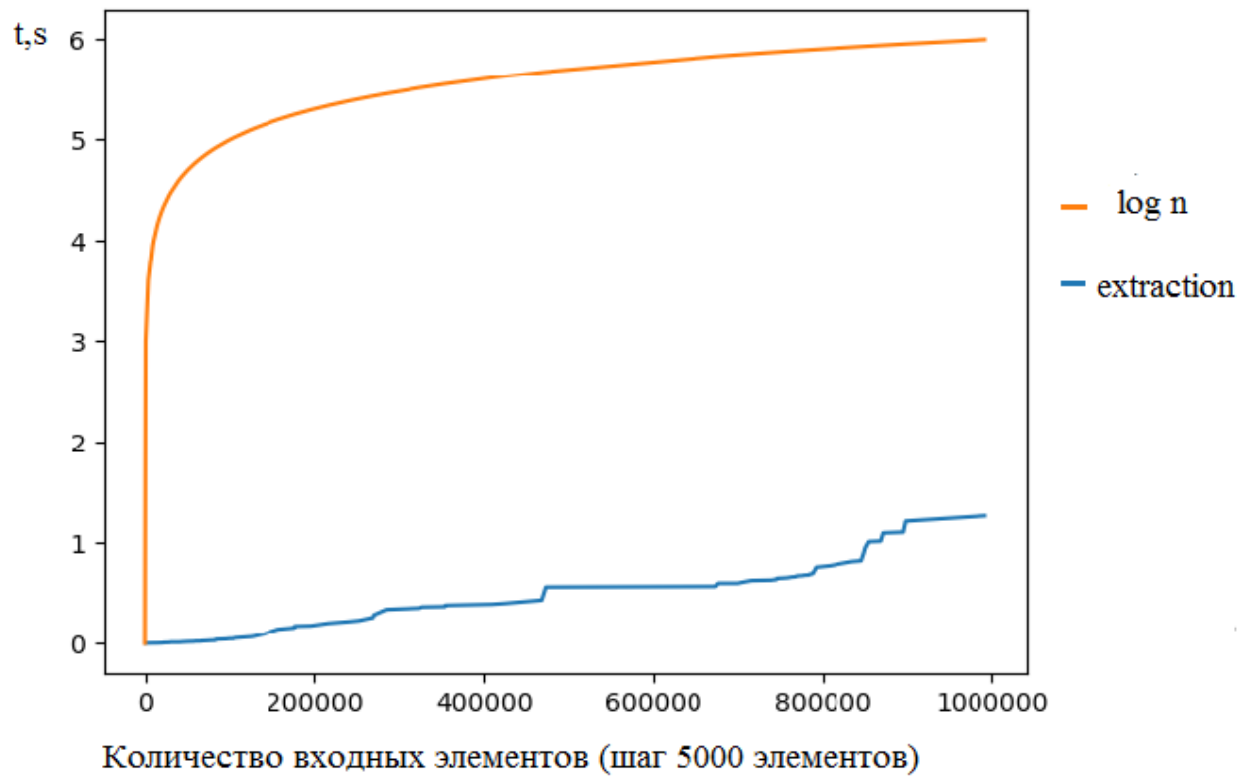


Рисунок 9 — Зависимость времени работы от входных данных для алгоритма извлечения элемента с максимальным приоритетом

На графиках видно, что для всех алгоритмов зависимость времени работы от размера входных данных является логарифмической, соответственно на основании проведённых тестов было доказано, что сложность всех трёх алгоритмов — $O(\log n)$.

Заключение

В результате проведённой работы была создана программа, реализующая алгоритмы добавления, удаления и получение элемента с максимальным приоритетом в приоритетной очереди. При этом был дан обзор структуре данных двоичная куча, а также проведён анализ эффективности работы алгоритмов. Разработанное приложение было протестировано и подробно описано в данной работе.

Поставленные задачи были решены в полной мере. Соответственно, цель курсовой работы можно считать достигнутой.

Список использованных материалов

1. Вирт, Н. Алгоритмы и структуры данных: Пер. с англ. – М.: Мир, 1989 – 360 с., ил.
2. Двоичная куча — Викиконспекты [Электронный ресурс] // Викиконспекты. М., 2016. URL: http://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D0%BA%D1%83%D1%87%D0%B0. (Дата обращения: 15.05.2018).
3. Treaps и T-Treaps [Электронный ресурс] / П. Райков. // Computer technologies department, ITMO University, Saint-Petersburg. М., 2016. URL: <http://rain.ifmo.ru/cat/data/theory/trees/treaps-2006/article.pdf>. (Дата обращения: 15.05.2018).

Приложение А (обязательное)

Листинг программы

```
// main.dart
import 'package:flutter/material.dart';
import "components/todo_app.dart";
void main() => runApp(new TodoApp());

// my_comparable.dart
abstract class MyComparable<T> {
    bool operator <(T value);
    bool operator <=(T value);
    bool operator >(T value);
    bool operator >=(T value);
}

// priority_queue.dart
import "bin_heap.dart";
import "priority_queue_value.dart";
class PriorityQueue<T> {
    BinHeap<PriorityQueueValue<T>> _data = new
    BinHeap<PriorityQueueValue<T>>();
    PriorityQueueValue get max => this._data.max;
    List<PriorityQueueValue<T>> toList() {
        return this._data.toList();
    }
    void insert(int priorityIndex, T value) =>
        this._data.add(new PriorityQueueValue(value, priorityIndex));
```

```

void delete(int valueIndex) =>
    this._data.delete(valueIndex);
PriorityQueueValue<T> popMax() => this._data.popMax();
}

// priority_queue_value.dart
import "my_comparable.dart";
class PriorityQueueValue<T> extends MyComparable<PriorityQueueValue> {
    T value;
    int priority;
    PriorityQueueValue(this.value, this.priority);
    @override
    bool operator <(PriorityQueueValue pqv) => this.priority < pqv.priority;
    @override
    bool operator <=(PriorityQueueValue pqv) => this.priority <= pqv.priority;
    @override
    bool operator >(PriorityQueueValue pqv) => this.priority > pqv.priority;
    @override
    bool operator >=(PriorityQueueValue pqv) => this.priority >= pqv.priority;
}

// bin_heap.dart
import "my_comparable.dart";
class BinHeap<T> extends MyComparable<T> {
    List<T> _data = <T>[];
    void add(T value) {
        this._data.add(value);
        this._siftUp(this._data.length - 1);
    }
}

```

```

void delete(int index) {
    this._data[index] = this._data.last;
    this._data.removeLast();
    this._siftDown(index);
}

T popMax() {
    if (this._data.isEmpty) return null;
    T result = this.max;
    this.delete(0);
    return result;
}

T get max {
    try {
        return this._data.first;
    } catch (StateError) {
        return null;
    }
}

List<T> toList() {
    return new List.from(this._data);
}

void _siftDown(int index) {
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    int largest = index;
    if (left < this._data.length && this._data[left] > this._data[index]) {
        largest = left;
    }
}

```



```

        if (right < this._data.length && this._data[right] > this._data[largest])
        {
            largest = right;
        }
        if (largest != index) {
            this._swapData(index, largest);
            this._siftDown(largest);
        }
    }

    void _siftUp(int index) {
        while (index > 0) {
            int parent = (index - 1) ~/ 2;
            if (this._data[index] <= this._data[parent]) return;
            this._swapData(index, parent);
            index = parent;
        }
    }

    void _swapData(int index1, int index2) {
        T tmp = this._data[index1];
        this._data[index1] = this._data[index2];
        this._data[index2] = tmp;
    }
}

```

```

// todo_app.dart

import 'package:flutter/material.dart';
import "my_tab.dart";

class TodoApp extends StatelessWidget {
    @override

```

```

Widget build(BuildContext context) {
    return new MaterialApp(
        title: 'Todo',
        theme: new ThemeData.light(),
        home: new MyTab(),
    );
}
}

// my_tab.dart
import 'package:flutter/material.dart';
import "../algo_struc/priority_queue.dart";
class MyTab extends StatefulWidget {
    @override
    _MyTab createState() => new _MyTab();
}
class _MyTab extends State<MyTab> {
    String _todoText = "";
    PriorityQueue<String> _queue = new PriorityQueue<String>();
    Icon _todoPriority;
    String _topTodoText = "";
    Icon _topTodoPriority = new Icon(null);
    List<ListTile> _listView = <ListTile>[];
    int _getPriorityByIcon(Icon icon) {
        if (icon.icon == Icons.sentiment_very_dissatisfied) {
            return 5;
        } else if (icon.icon == Icons.sentiment_dissatisfied) {
            return 4;
        } else if (icon.icon == Icons.sentiment_neutral) {

```

```

    return 3;
} else if (icon.icon == Icons.sentiment_satisfied) {
    return 2;
}
return 1;
}

Icon _getPriorityByInt(int priority) {
    if (priority == 5) {
        return new Icon(
            Icons.sentiment_very_dissatisfied,
            color: Colors.red[400],
        );
    } else if (priority == 4) {
        return new Icon(
            Icons.sentiment_dissatisfied,
            color: Colors.purple[400],
        );
    } else if (priority == 3) {
        return new Icon(
            Icons.sentiment_neutral,
            color: Colors.yellow[600],
        );
    } else if (priority == 2) {
        return new Icon(
            Icons.sentiment_satisfied,
            color: Colors.blue[600],
        );
    }
    return new Icon(

```

```

        Icons.sentiment_very_satisfied,
        color: Colors.greenAccent[400],
    );
}

void _addTodo() {
    if (this._todoPriority == null) return;
    int priority = this._getPriorityByIcon(this._todoPriority);
    if (this._todoText.isNotEmpty) {
        print("Insert(${priority}, '${this._todoText}')");
        setState(() {
            this._queue.insert(priority, this._todoText);
            this._setTopTodo();
            this._listView.add(new ListTile(
                trailing: new Icon(
                    this._todoPriority.icon,
                    color: this._todoPriority.color,
                ),
                title: new Text(this._todoText),
            ));
            this._todoPriority = null;
        });
    }
}

void _deleteTodo() {
    setState(() {
        var max = this._queue.popMax();
        this._setTopTodo();
        if (this._listView.isNotEmpty) {
            var k = 0;

```

```

    this._listView.removeWhere((ListTile w) {
        var value = (w.title as Text).data;
        var priority = this._getPriorityByIcon(w.trailing as Icon);
        var b = (value == max.value) && (priority == max.priority);
        return b && (k++ == 0);
    });
}
});
}

void _setTopTodo() {
    setState(() {
        var max = this._queue.max;
        if (max == null) {
            this._topTodoText = "";
            this._topTodoPriority = null;
        } else {
            print("max: ${max.priority},${max.value}");
            this._topTodoText = max.value;
            this._topTodoPriority = this._getPriorityByInt(max.priority);
        }
    });
}

void _changeTodoText(String text) {
    setState(() {
        this._todoText = text;
    });
}

@override
Widget build(BuildContext context) {

```

```

return new DefaultTabController(
  length: 3,
  child: new Scaffold(
    appBar: new AppBar(
      title: new Center(
        child: new Text(
          "Todo",
          textAlign: TextAlign.center,
        ),
      ),
    ),
    bottom: new TabBar(
      tabs: [
        new Tab(icon: new Icon(Icons.add)),
        new Tab(icon: new Icon(Icons.list)),
        new Tab(icon: new Icon(Icons.star)),
      ],
    ),
  ),
  body: new TabBarView(
    children: <Widget>[
      new Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          new Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              new IconButton(
                icon: new Icon(
                  Icons.sentiment_very_dissatisfied,

```

```

        color: Colors.red[400],
      ),
      onPressed: () {
        setState(() {
          this._todoPriority = new Icon(
            Icons.sentiment_very_dissatisfied,
            color: Colors.red[400],
          );
        });
      },
    ),
    new IconButton(
      icon: new Icon(
        Icons.sentiment_dissatisfied,
        color: Colors.purple[400],
      ),
      onPressed: () {
        setState(() {
          this._todoPriority = new Icon(
            Icons.sentiment_dissatisfied,
            color: Colors.purple[400],
          );
        });
      },
    ),
    new IconButton(
      icon: new Icon(
        Icons.sentiment_neutral,
        color: Colors.yellow[600],

```



```

    ),
    onPressed: () {
      setState(() {
        this._todoPriority = new Icon(
          Icons.sentiment_neutral,
          color: Colors.yellow[600],
        );
      });
    },
  ),
  new IconButton(
    icon: new Icon(
      Icons.sentiment_satisfied,
      color: Colors.blue[600],
    ),
    onPressed: () {
      setState(() {
        this._todoPriority = new Icon(
          Icons.sentiment_satisfied,
          color: Colors.blue[600],
        );
      });
    },
  ),
  new IconButton(
    icon: new Icon(
      Icons.sentiment_very_satisfied,
      color: Colors.greenAccent[400],
    ),

```

```

onPressed: () {
  setState(() {
    this._todoPriority = new Icon(
      Icons.sentiment_very_satisfied,
      color: Colors.greenAccent[400],
    );
  });
},
),
],
),
new Container(
  padding: new EdgeInsets.symmetric(
    vertical: 10.0,
    horizontal: 10.0,
  ),
  child: new TextField(
    onChanged: this._changeTodoText,
    decoration: new InputDecoration(
      suffixIcon: this._todoPriority,
      border: new OutlineInputBorder(
        borderSide: new BorderSide(
          width: 1.0,
        ),
      ),
    ),
    labelText: "Название задачи",
  ),
),
),
),

```

```

    new RaisedButton(
      child: new Text("Добавить"),
      color: Colors.lightBlue,
      textColor: Colors.white,
      onPressed: this._addTodo,
    ),
  ],
),
new ListView(
  children: this._listView,
),
new Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    new Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        new Container(
          padding: new EdgeInsets.only(right: 10.0),
          child: Text(this._topTodoText),
        ),
        new Container(
          child: this._topTodoPriority,
        ),
      ],
    ),
    new Container(
      padding: new EdgeInsets.only(top: 10.0),
      child: new RaisedButton(

```

```
        child: new Text("Выполнено"),  
        color: Colors.lightBlue,  
        textColor: Colors.white,  
        onPressed: this._deleteTodo,  
      )  
    ),  
  ],  
),  
],  
,  
,  
,  
);  
}  
}
```