

# 03 Автодополнение

[Jump to bottom](#)

Alexander Morozov edited this page 16 days ago · 9 revisions

---

Ссылка на задание: [https://classroom.github.com/a/X\\_MWiPbQ](https://classroom.github.com/a/X_MWiPbQ)

## Задание

Напишите реализацию автодополнения для  $N$  строк и положительных весов. На выход программе нужно выдать все строки, которые начинаются с заданного префикса в порядке убывания весов.

Автодополнение очень чувствительно к скорости работы. Поэтому автодополнение нужно реализовать на отсортированном множестве, а поиск в этом множестве нужно делать бинарным поиском.

Задание состоит из трёх частей:

## Часть 1

Нужно реализовать неизменяемый класс `term`, который содержит в себе:

- Конструктор принимающий строку и её вес
- Операторы сравнения термов
- Конструкторы/операторы копирования, присваивания, перемещения должны работать корректно
- Метод `to_string` который возвращает строковое представление строки в формате "weight term"
- Оператор вывода в поток (тот же формат, что и результат `to_string`)
- Метод `by_reverse_weight_order` который возвращает функтор принимающий на вход два термина и сравнивает их с учётом весов в обратном порядке

- Метод `by_prefix_order(int r)` который возвращает функтор принимающий на вход два термина и сравнивает их по первым `r` символам без учёта весов

## Часть 2

Нужно реализовать класс `binary_search_deluxe`, который реализует бинарный поиск и удовлетворяет требованиям:

- Экземпляр этого класса нельзя создавать
- Статический метод `int first_index_of(term[] a, term key, Func comparator)`, который принимает отсортированный массив `a` и должен выдать индекс начала `key` в массиве `a`
- Статический метод `int last_index_of(term[] a, term key, Func comparator)`, который принимает отсортированный массив `a` и должен выдать индекс конца `key` в массиве `a`

`Func comparator` - это некоторый объект (класс с переопределённым оператором круглые скобки, функция, лямбда), который принимает два термина и возвращает `true` если первый терм меньше второго. Пример такого компаратора:

```
auto cmp = [](const term& t1, const term& t2) { return t1 > t2; };
```

Вместо `Func` Вам нужно написать такой тип для которого случаи: класс с переопределённым оператором круглые скобки, функция и лямбда будут компилироваться и работать корректно.

## Часть 3

Нужно реализовать неизменяемый класс `autocomplete`, который использует внутри себя `term` и `binary_search_deluxe`. Этот класс должен содержать в себе следующие методы:

- Конструктор, принимающий массив термов
- Метод `all_matches(string prefix)` возвращает массив термов, которые подходят под заданный префикс
- Метод `number_of_matches(string prefix)` возвращает количество термов, которые подходят под заданный префикс

## PS

1. Для сигнатур методов нужно выбрать правильные модификаторы доступа
2. Работа должно удовлетворять общим требованиям [Требования к выполнению домашних заданий](#)

▼ Pages 18
<input type="text" value="Find a Page..."/>
<a href="#">Home</a>
<a href="#">00 Тестовое задание: sort</a>
<a href="#">00 Тестовое задание: uniq</a>
<a href="#">02 Рандомизированная очередь и дек</a>
<a href="#">03 Автодополнение</a>
<a href="#">04 8 Puzzle</a>
<a href="#">05 Двумерные деревья</a>
<a href="#">08 Алгоритм сжатия Барроуза Уилера</a>
<a href="#">Вопросы к экзамену</a>
<a href="#">Для тех, кто боится заскучать летом</a>
<a href="#">Домашние задания</a>
<a href="#">Дополнительные материалы</a>
<a href="#">Инструкция по работе с домашними заданиями в GitHub</a>
<a href="#">Материалы лекций</a>
<a href="#">План лекций</a>
Show 3 more pages...

Clone this wiki locally

<https://github.com/itiviti-cpp/wiki/wiki.git>

