

## Отображения

Требуется построить параметризованный класс, который реализует отображение где ключем является строка, а значением — некоторый другой класс.

Интерфейс отображения должен поддерживать следующие операции:

- добавить пару (ключ, значение);
- искать значение по указанному ключу;
- удалить ключ и соответствующее значение;
- получить количество хранящихся ключей;
- итератор по множеству ключей и значений.

В качестве примеров и тестов можно рассмотреть отображение строк на целые числа и отображения строк на строки (как в словаре). Тесты должны предусматривать возможность загрузки множества из файла или автоматической генерации.

Варианты заданий различаются способом реализации отображения.

**Задание 2.1.** Обычное дерево поиска.

**Задание 2.2.** Сбалансированное дерево поиска.

**Задание 2.3.** Красно-черное дерево поиска.

**Задание 2.4.** В-дерево.

**Задание 2.5.** Хеш-множество по методу списков.

**Задание 2.6.** Хеш-множество по методу линейных проб.

## Упорядоченное множество строк

Такое множество позволяет выбирать упорядоченные подмножества и определять следующий или предыдущий элемент для какого-либо элемента множества (строки).

Реализация должна поддерживать следующий интерфейс

- добавить строку в множество;
- удалить строку;
- искать данную строку;
- получить количество элементов в множестве;
- итератор по части множества. Это означает, что после указания некоторой строки, мы можем перебирать последовательно упорядоченные строки множества от указанной в одну или другую сторону.

Для тестов следует предусмотреть загрузку из файла и автоматическую генерацию множества.

Варианты задания различаются основой реализации множества.

**Задание 2.7.** Обычное дерево для хранения указателей на строки.

**Задание 2.8.** Сбалансированное дерево для хранения указателей на строки.

**Задание 2.9.** Красно-черное дерево поиска для хранения указателей на строки.

**Задание 2.10.** В-дерево для хранения указателей на строки.

## Множество точек $R^2$

Такое множество хранит координаты  $(x, y)$  точек плоскости и позволяет выбирать точки, лежащие в некоторой окрестности заданной точки. Предполагается, что все точки находятся внутри изначально заданного прямоугольника.

Реализация должна поддерживать следующий интерфейс

- добавить точку в множество;
- удалить точку;
- есть ли точка в множестве;
- получить количество точек в множестве;
- получить список точек, лежащих в данной прямоугольной окрестности заданной точки.

Для тестов следует предусмотреть загрузку из файла и автоматическую генерацию множества.

Варианты задания различаются основой реализации множества.

**Задание 2.11.** Диапазон изменения по  $y$  делится на равные части, каждой такой части соответствует сбалансированное по значению  $x$  дерево точек, у которых  $y$  лежит в данной части.

**Задание 2.12.** Двумерный аналог дерева. Корень — это исходный прямоугольник. Поделив прямоугольник на 4 равные части, получим вершины первого уровня, и т.д. Деление вершин продолжается до тех пор, пока в прямоугольниках содержатся точки. Таким образом, концевые прямоугольники содержат ровно 1 точку.

## Файловый контейнер

Требуется реализовать контейнер данных наподобие файловой системы с возможностью создавать и уничтожать файлы и читать/записывать в них байтовые массивы некоторой длины. Для работы захватывается большой кусок памяти (виртуальный диск), в котором выделяются служебные области модельной файловой системы в соответствии с тем или иным методом ее реализации.

Реализация должна поддерживать следующий интерфейс.

- создать файл;
- удалить файл;
- копировать, переименовать файл;
- получить список существующих файлов;
- получить длину файла;
- прочитать/записать заданное количество байт по указанному смещению от начала файла;

Более сложный вариант предполагает введение меток времени создания и обновления файла и других атрибутов, а также возможность иерархической файловой системы.

Варианты задания отличаются принципами реализации файловой системы.

**Задание 2.13.** По аналогии с файловой системой FAT (WIN), т.е. на основе односвязных списков файловых блоков с хранением ссылок между блоками в отдельной таблице. (Более подробно принципы реализации обсуждаются в рабочем порядке.)

**Задание 2.14.** По аналогии с файловой системой EXT (UNIX), т.е. на основе множеств блоков с древовидным хранением номеров файловых блоков. (Более подробно принципы реализации обсуждаются в рабочем порядке.)

**Задание 2.15.** На основе двусвязных списков файловых блоков с хранением ссылок между блоками непосредственно в блоках. (Более подробно принципы реализации обсуждаются в рабочем порядке.)

**Задание 2.16.** По аналогии с файловой системой NTFS, т.е. на основе множеств блоков с хранением номеров связанных фрагментов файловых блоков в отдельном служебном файле. (Более подробно принципы реализации обсуждаются в рабочем порядке.)

## Архиватор файла

Требуется реализовать модельный архиватор/деархиватор файлов и исследовать степень сжатия для различных входных файлов.

Реализация должна поддерживать следующий интерфейс

- сжать указанный файл;
- восстановить указанный файл.

Тесты должны предусматривать исследование степени сжатия для файлов различного содержания (тексты, программы, уже сжатые файлы и пр.)

Варианты задания различаются используемыми алгоритмами.

**Задание 2.17.** Метод Хаффмена (адаптивный и неадаптивный). (Более подробно принципы реализации обсуждаются в рабочем порядке.)

**Задание 2.18.** Метод арифметического кодирования (адаптивный и неадаптивный). (Более подробно принципы реализации обсуждаются в рабочем порядке.)

**Задание 2.19.** Алгоритм LZW. (Более подробно принципы реализации обсуждаются в рабочем порядке.)

**Задание 2.20.** Алгоритм RLE по 8 битовым плоскостям (каждый порядковый бит в байте кодируется отдельно) (Более подробно принципы реализации обсуждаются в рабочем порядке.)

## Анализ файлов

**Задание 2.21.** Требуется найти все файлы с одинаковыми именами и все файлы с одинаковым содержанием в рамках указанного каталога и всех его подкаталогов.

**Задание 2.22.** Требуется найти все различия (с точностью до строк) между двумя текстовыми файлами, используя алгоритм наибольшей общей подпоследовательности.