Задание 3 для первого семестра 2 курса

Общие замечания

Задания этого раздела предполагают реализацию множеств с быстрым доступом (быстрые деревья). Соответственно тесты должны показывать, что использование таких реализаций действительно эффективно, т.е. должны предусматривать работу с большими наборами данных. Например, в дерево добавляется несколько миллионов элементов, потом производится поиск каждого элемента, в процессе элементы добавляются или удаляются по ходу работы. Проверяется, что добавленные элементы действительно добавились, а удаленные действительно удалились.

Кроме этого, для деревьев нужно проверить, что они действительно имеют логарифмическую глубину. То есть, написать отдельную функцию, которая вычисляет глубину дерева, и протестировать как меняется глубина при добавлении большого количества элементов.

И еще надо сравнить эффективность своей реализации с библиотечной реализацийе STL map (или set). Так как библиотечная реализация предполагает универсальность использования, то в некоторых задачах она может оказаться менее эффективной, чем специально написанная реализация. Поэтому интересно сравнить как и насколько различается время работы вашей реализации и библиотечной реализции в разных случаях, особенно если удастся обнаружить пример, где ваша реализация будет работать эффективнее.

Отображения

Требуется построить параметризованный (template) класс, который реализует отображение где ключем является один класс, а значением — некоторый другой класс.

```
template <class key, class value>
```

Интерфейс отображения должен поддерживать следующие операции:

— добавить пару (ключ,значение);

— искать значение по указанному ключу;

— доступ к значению элемента дерева по указанному ключу;

— удалить ключ и соответствующее значение;

— получить количество хранящихся ключей;

— итератор по множеству пар (ключ, значение).

Доступ к значению элемента можно реализовать через итератор, т.е. функция поиска возвращает итератор, и он уже обеспечивает доступ к значению через операцию * или ->.

В качестве примеров и тестов можно рассмотреть отображение строк на целые числа, отображения строк на строки (как в словаре), отображение пар чисел (координат точки) на некоторый указатель и т.п. Тесты должны предусматривать возможность загрузки множества ключей-значений из файла или их автоматической генерации.

Узел дерева может быть реализован внешним или внутренним классом, но обязательное требование — сокрытие его членов (значения, указатели) для не относяшихся к дереву компонент программы.

Варианты заданий различаются способом реализации отображения.

1. Сбалансированное AVL дерево поиска (рекурсивные реализации, left-right).
2. Сбалансированное AVL дерево поиска (нерекурсивные реализации, left-right-parent).
3. Красно-черное дерево поиска.
4. 2-3-дерево (рекурсивные реализации, нет указателя parent).
5. 2-3-дерево (нерекурсивные реализации, есть указатель parent).
6. B-дерево (любая удобная реализация).
7. B+ дерево (любая удобная реализация)

Итератор должен реализовывать двунаправленный проход по множеству элементов в смысле упорядоченности их ключей.

Выполнение задания состоит из двух этапов.

Первый — параметризованная реализация быстрого дерева с параметризацией template<class Key, class Value>. Тестирование на простых типах данных на добавление, поиск, удаление большого количества элементов. Тестирование итератора. Проверка глубины дерева в зависимости от количества элементов. Сравниние с такими же тестами для std::map.

Второй — решение тестовой “практической” задачи.

Такими задачами могут быть, в частности,

A — частотный словарь: для большого текста определить сколько раз в нем встречается каждое слово.

B — конфигурационный файл: в файле записаны строки типа “имя = значение” (целое, вещественное, строковое), надо в программе быстро получать соответствующее значение по данному имени.

C — географическая карта: на плоскости есть множество точек с целочисленными координатами и каждой точке приписано некоторое “слово”. Надо быстро получать координаты по слову либо слово по координатам.

D — двумерная функция на нерегулярной сетке: на плоскости есть множество точек с целочисленными координатами и в каждой точке задано некоторое значение. Надо быстро искать значения по точкам и модифицировать функцию (изменять значения, добавлять, удалять точки).

E — база данных: есть массив записей базы, нужно добавлять, удалять, искать записи по заданному ключу, которым может быть значение некоторого поля записи базы данных.

Суть и конкретная постановка второго этапа обсуждается в рабочем порядке после выполнения первого этапа.