Лабораторная работа № 2 по курсу дискретного анализа: словарь

Выполнил студент группы М8о-207Б-18 МАИ *Токарев Никита*.

Условие

1. Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до 2^{64} - 1. Разным словам может быть поставлен в соответствие один и тот же номер.

2. Патриция. Тип ключа: регистронезависимая последовательность букв английского алфавита длиной не более 256 символов. Тип значения: числа от 0 до 2^{64} - 1.

Метод решения

В данной работе нужно было реализовать 5 операций: поиск в дереве, удаление элемента из дерева, добавление элемента в дерево, загрузка структуры из бинарного файла, сохранение в бинарный файл.

1. Поиск в дереве: Беру номер уникального бита от текущего элемента и получаю по этому номеру бит в данном ключе, который я хочу найти. Если бит 1, то иду вправо, иначе влево. Поиск заканчивается, когда мы переходим по обратному указателю в некий элемент, который сравниваем с данным(искомым).

2. Добавление в дерево: В первую очередь ищем место для вставки с помощью алгоритма поиска, описанного выше. Затем я сравниваю найденный ключ и данный. В итоге, если данных с таким ключом нету, я сравниваю их битовые маски и получаю номер первого отличного бита(буду называть n). Затем же с помощью алгоритма поиска я прохожу по дереву в надежде найти место для вставки. После нахождения место я вставляю согласно правилам: если бит(по n) в добавляемом ключе == 1 то правый указатель на себя, левый же есть указатель: прежний указатель узла к которому добавляется данный узел. Если же если бит(по n) в добавляемом ключе == 0 то наоборот.

3. Удаление из дерева: Ищу данный элемент с помощью алгоритма поиска. Если элемент был найден то алгоритм может принимать 2-дейтсвия. Если же удаляемый узел – лист с указателем на себя, то я получаю указатель удаляемого элемента, который указывает не на себя. Затем же в родительском узле заменяю прежний указатель на удаляемый элемент, на полученный. Если же удалемый элемент(p)

без указателя на себя то я свапаю ключи и значения удаляемого элемента и элемента(q), который имеет обратный указатель на удаляемый. Затем же для q я нахожу элемент(r), который имеет обратный указатель на q. Далее меняю обратный указатель r с q на p. Дальше же, грубо говоря я затираю q на r. Если r = q, то данный элемент удаляется.

4. Загрузка и сохранение дерева: Сохраняю и загружаю соответсвенно в бинарный файл. Первый аргумент – размер дерева. Если же дерево пустое, то генерируется пустой файл. Иначе я создаю массив, который в неком определенном порядке содержит необходимые узлы с соответсвующими порядковыми номерами. Начиная с нулевого элемента я записываю данные в бинарный файл, указывая необходимую информацию, а также id(порядковые номера при обходе) левого и правого узла. Соответсвенно при загрузке файла в нужной последовательности я получаю массив с расставленными указателями на узлы(элементы массива). В результате полученный массив из узлов и является Патрицией.

Описание программы

Программа реализована одним файлом. В первую очередь хотелось бы описать параметры класса TPatricia. Параметры класса: корневой узел, а также количество узлов в дереве. Что касается узла то он представляет из себя строку типа char*(ключ), числовое значение, номер уникального бита, id, который нужен при сохранении и загрузке дерева, а также указатели на левого и правого сына. Для узла определен конструктор, деструктор(в деструкторе происходит удаление строки), функция получения числового значения. Также хотелось бы отметить, что основной класс TPatricia имеет доступ ко все приватным полям узла. Что касается самого класса дерева, то в нем реализованы 5 основных функций, а также вспомогательные в private секции. Вспомогательными функциями являются такие функции: нахождение родителя элемента, нахождение элемента, имеющего обратный указатель на заданный, функция побитового сравнения двух ключей, функция получения первого отличного бита двух указанных ключей, функция получения n-го бита, а также рекурсивная функция удаления дерева. Что касается основных функций, то примерные алгоритмы описаны выше, однако хотелось бы отметить несколько важных моментов.

Дневник отладки

При тестировании были замечены такие ошибки: ошибка выполнения в четвертом тесте, ошибки компиляции и неправильный ответ в 3-м тесте

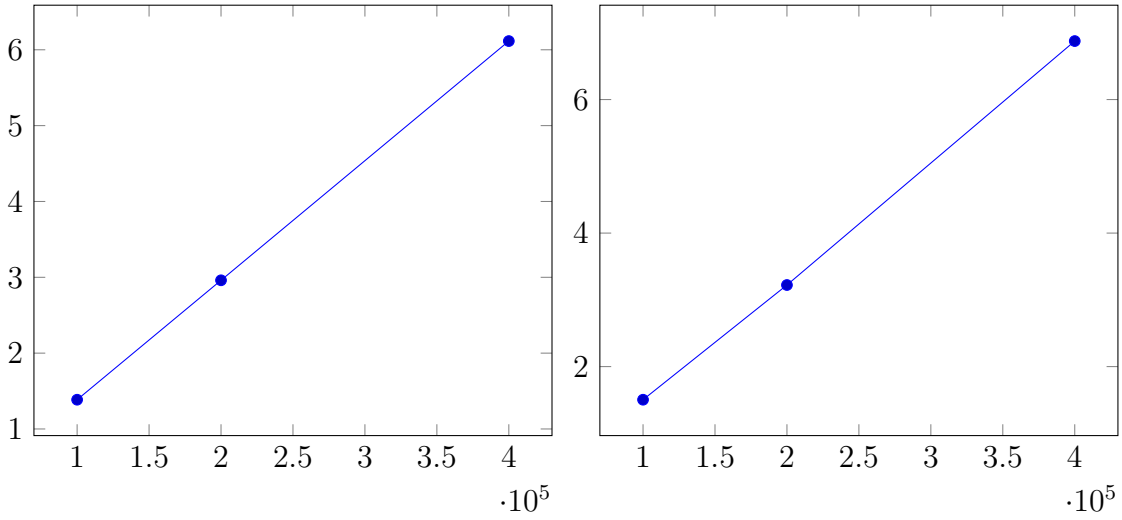
1. Ошибки компиляции -> исправлял согласно стандартам C++;

2. Ошибка выполнения в 4 тесте -> удаление корня происходило неккоректно: удалялся ключ в данном объекте, однако сам корневой элемент был цел. При обращении к ключу головнога элемента происходила ошибка.

3. Неправильный ответ в 3-м тесте -> Неправильная логика подсчета первого отличающегося бита в двух ключах, когда размеры ключей не одинаковы;

Тест производительности

В данных тестах используется строки только из 50 элементов, сгенерированные рандомно. Поэтому возможен случай, что некоторые ключи будут совпадать и не будут добавлены в дерево.
Добавление элементов(слева) и удаление элементов(справа):



Пояснения к графику:

1. За 1.38563с добавляется 100000 узлов
2. За 2.96047с добавляется 200000 узлов
3. За 611501с добавляется 400000 узлов

Далее, добавленные узлы в таком же порядке, как и добавлялись,удаляются.

1. За 1.5039с удаляются 100000 узлов
2. За 3.22216с удаляются 200000 узлов
3. За 6.87627с удаляются 400000 узлов

Выводы

В ходе данной работы я познакомился с такой структурой именнуемой Patricia trie. Patricia представляет собой сжатый trie. Patricia позволяет довольно эффекитвно работать с символьными ключами, интерпритируя ключ в битовую последовательность. В

данном дереве операции вставки и удаления происходят за линейное время O(h), где h высота дерева. Также в данной работе был реализован простой бенчмаркт с помощью которого, я смог протестировать функции удаления и вставки элемента в дерево.