Лабораторная работа № 2 по курсу дискретного анализа: Сбалансированные деревья

Выполнил студент группы М8О-208Б-21 МАИ *Королев Илья*.

Условие

- 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.

  Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

- + word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

  - word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

  word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

  ! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды.

  ! Load /path/to/file — загрузить словарь из файла.

  Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутсвие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

- Различия вариантов заключаются только в используемых структурах данных:

  Красно-чёрное дерево.

Метод решения

Для решения данной задачи было реализованно 7 основных функций.

1. Поиск: findNode осуществляет поиск узла в дереве по заданному ключу key.

- Установить указатель current на корень дерева.
- Пока указатель current не равен указателю на NILL:
  - Если ключ текущего узла равен искомому ключу, возвращаем указатель на текущий узел current.
  - Если ключ текущего узла больше искомого ключа, тогда надо перейти к левому потомку узла current.
  - Если ключ текущего узла меньше искомого ключа, тогда надо перейти к правому потомку узла current.
- Если узел с заданным ключом не найден, вернуть NULL.

2. Вставка: insertNode вставка нового узла в красно-черное дерево.

   - Создание нового узла с помощью функции createNode.
   - Поиск места для вставки узла: начинается поиск с корня дерева. Если ключ меньше ключа текущего узла, то продолжается поиск в левом поддереве, иначе - в правом. При этом сохраняется указатель на текущий узел и его родителя.
   - Если в дереве уже есть узел с таким же ключом, то вставка не производится и функция возвращает 1.
   - Вставка нового узла на свободное место. Если найденный родительский узел был NULL, то новый узел становится корнем дерева. Если дерево было пустым, то новый узел становится корнем дерева и окрашивается в черный цвет. Иначе, если ключ нового узла меньше ключа родителя, то новый узел становится левым потомком родителя, иначе - правым потомком родителя.
   - Перебалансировка дерева: вызов функции insertFixup для восстановления свойств красно-черного дерева после вставки нового узла.
   - Если вставка прошла успешно, функция возвращает 0. Если при создании нового узла произошла ошибка, то функция возвращает 2.

3. Балансировка: insertFixup используется для восстановления свойств красно-черного дерева после операции вставки узла.

   - В начале функции проверяется, является ли новый узел корнем дерева. Если да, то устанавливается его цвет в черный, и он становится корнем дерева. Если нет, то выполняется цикл, пока родитель нового узла не станет черным, или не будет выполнено определенное условие.
   - В теле цикла выполняются следующие проверки:
     - Если родитель нового узла является левым потомком своего родителя, то дядя нового узла находится справа от родителя.

- Если родитель нового узла является правым потомком своего родителя, то дядя нового узла находится слева от родителя.

- Далее выполняются следующие случаи:
  - Если дядя нового узла красный, то цвета родителя и дяди меняются на черный, а цвет родителя родителя на красный. Новый узел становится указывать на своего дедушку.
  - Если дядя нового узла черный, а новый узел является правым потомком своего родителя, то выполняется левый поворот относительно родителя, после чего переходим к третьему случаю.
  - Если дядя нового узла черный, а новый узел является левым потомком своего родителя, то цвета родителя и родителя родителя меняются местами, после чего выполняется правый поворот относительно родителя родителя.

- Далее цикл продолжается, пока родитель нового узла не станет черным или не будет выполнено условие. В конце функции цвет корня дерева устанавливается в черный.

4. Удаление: DeleteNode: эта функция удаляет узел с заданным ключом из красно-черного дерева.

   - Используется функция findNode для поиска узла с заданным ключом. Если узел не найден, функция возвращает значение 3, что означает неудачу.
   - Определяется узел delnode, который нужно удалить, и сохраняется его цвет.
   - Определяется узел x, который заменит delnode после удаления. Если delnode не имеет левого потомка, x устанавливается равным правому потомку delnode. В противном случае, если delnode не имеет правого потомка, x устанавливается равным левому потомку delnode. Если delnode имеет обоих потомков, находится наименьший узел в правом поддереве delnode, и его цвет сохраняется.
   - Если родитель удаляемого узла delnode равен NULL, то delnode является корнем дерева, и его можно удалить, заменив на x. Если нет, функция transplant используется для замены delnode на x.
   - Если полученный цвет чёрный, тогда вызывается функция deleteFixup для восстановления свойств красно-черного дерева.
   - Удаляется delnode, и функция возвращает значение 0, что означает успех.

5. Балансировка: deleteFixup - эта Функция используется в красно-черном дереве для восстановления свойств дерева после удаления узла.

   - Если брат узла x красный, то цвета родителя узла x и брата меняются местами, а затем выполняется поворот относительно родителя так, чтобы брат

был на месте бывшего родителя. После этого случай превращается во второй или третий.

- Если брат узла x черный, а оба потомка брата также черные, то цвет брата меняется на красный, x перемещается вверх по дереву, и процесс повторяется.
- Если брат узла x черный, а правый потомок брата красный, а левый потомок брата черный, и x является левым потомком своего родителя, то цвета брата и его правого потомка меняются местами, и выполняется левый поворот относительно брата. После этого случай превращается в четвертый.
- Если брат узла x черный, а левый потомок брата красный, а правый потомок брата черный, и x является правым потомком своего родителя, то цвета брата и его левого потомка меняются местами, и выполняется правый поворот относительно брата. После этого случай превращается во второй.
- Если брат узла x черный, а правый потомок брата красный, и x является правым потомком своего родителя, то цвета родителя, брата и его правого потомка меняются местами, затем выполняется левый поворот относительно родителя, и процесс завершается.
- Если брат узла x черный, а левый потомок брата красный, и x является левым потомком своего родителя, то цвета родителя, брата и его левого потомка меняются местами, затем выполняется правый поворот относительно родителя, и процесс завершается.

6. Сохранение дерева. Функция save сохраняет данные дерева в файл.

   Если текущий узел является sentinel, то функция прекращает свою работу. Сохраняется длина ключа текущего узла в файл. Сохраняется ключ текущего узла в файл. Сохраняется значение текущего узла в файл. Рекурсивно вызывается функция для левого потомка текущего узла. Рекурсивно вызывается функция для правого потомка текущего узла.

7. Загрузка дерева. Функция load() используется для загрузки данных из файла в дерево в формате, записанном функцией save().

   Создается новое дерево newTree с помощью функции createTree(). Читается целое число buferSize с помощью функции fread(). Если достигнут конец файла или произошла ошибка, то дерево newTree удаляется, и функция завершается. Если чтение прошло успешно, то считывается buferSize символов с помощью функции fread(). Если не удалось считать нужное количество символов, то дерево newTree удаляется, и функция завершается. Считывается значение Key типа unsigned long long int с помощью функции fread(). Если чтение прошло успешно, переводится значение buffer в строку с помощью добавления нуль-терминатора. В дерево newTree вставляется новый узел с ключом buffer и значением Key. Эти шаги повторяются, пока файл не будет полностью прочитан. Удаляется старое дерево tree с помощью функции

destroy(), освобождается память, занимаемая tree->sentinel, и в дерево tree записываются данные из newTree.

Описание программы

Вся программа реализована одним файлом. Кратко про вспомогательные функции:

Функция strlen вычисляет длину char* str путем итерации по указателю s до тех пор, пока не будет достигнут конец строки (который обозначен символом ”) и возвращает число символов в строке.

Функция strcpy копирует содержимое char* src в char* dest, начиная с первого символа строки dest и продвигая указатель на символ каждый раз, когда копируется символ из src в dest.

Функция strcmp сравнивает два массива char* str1 и char* str2 путем сравнения символов в каждой строке попарно, начиная с первого символа каждой строки.

Структура узла дерева определена в структуре Node и включает следующие поля:

- key - ключ узла, представленный строкой до 256 символов включительно
- value - значение узла, которое представлено 64-битным целым числом
- color - цвет узла, который может быть либо красным, либо черным, бинарное значение, BLACK соответствует true
- left и right - указатели на левого и правого потомков узла соответственно
- parent - указатель на родительский узел

Структура дерева определена в структуре Tree и включает два поля:

- root - указатель на корень дерева
- sentinel - граничный элемент дерева, который является фиктивным узлом и используется для определения конца дерева

Также вспомогательные функции: Функция transplant заменяет один узел дерева на другой узел. Левый(leftRotate) и правый(rightRotate) поворот. Программа считывает команды из стандартного ввода и вызывает соответствующие функции для выполнения каждой команды.

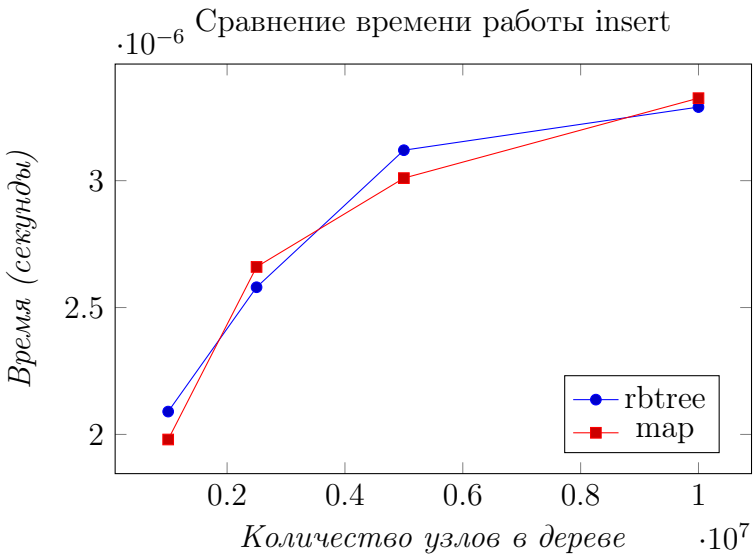
Дневник отладки

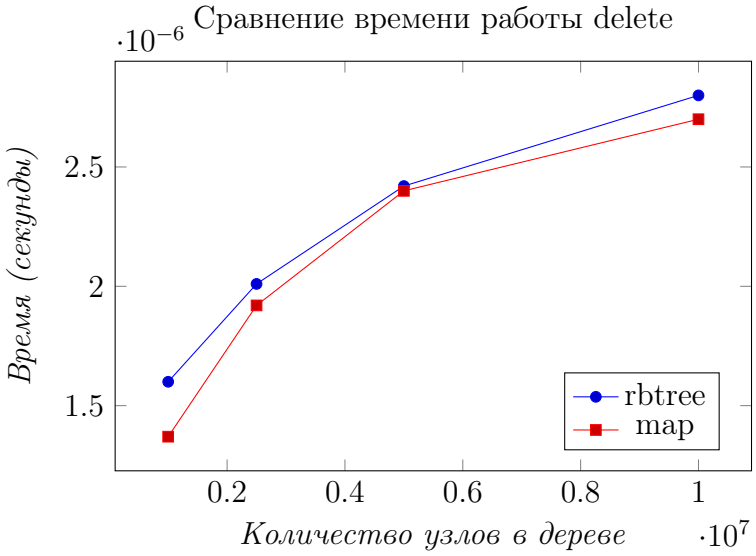
Проблем с деревом почти не было. На 5м тесте падало, но это из-за того, что было не правильно реализовано удаление. Переписав полностью функцию удаления, дерево стало работать правильно, но падала на 7 тесте с WA. Реализовав функции Save Load, все равно падало на 7 тесте, но с ошибкой RE. Оставив все также, но убрав pritf("ERROR"), все зашло.

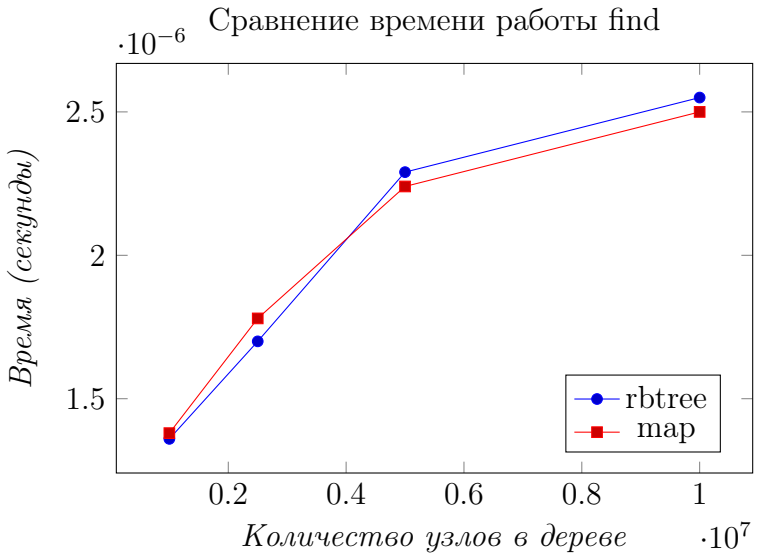
Тест производительности

При тестировании использовались файлы из строк, состоящих из фиксированной длины ключа (=10) и случайного значения (от 0 до $2^{64} - 1$). Для тестирования надо в идеале добавить один элемент в дерево и засечь время этой операции, но при 50 000

000 строк в коде 1.6ГБ, в которых только + операция, все равно элемент добавляется очень быстро. Поэтому было принято решение добавлять в деревья много новых узлов, и усреднять время их вставки. Я добавлял по 100000 строк. аналогично для удаления и вставки. По тестам видно, что сложность O(log n).







Выводы

В ходе выполнения лабораторной работы было создано красно-чёрное дерево, реализующее словарь для хранения букв английского алфавита длиной до 256 символов и их числовых значений от 0 до 264 - 1.

Были реализованы следующие операции:

1. добавление слова в словарь с заданным числовым значением;
2. удаление слова из словаря;
3. поиск числового значения слова в словаре;
4. сохранение словаря в бинарном компактном представлении на диск;
5. загрузка словаря из файла.
6. Также была обеспечена обработка возможных ошибок, включая системные ошибки и ошибки формата данных при сохранении и загрузке словаря.

Использование красно-чёрного дерева позволило обеспечить эффективность операций добавления, удаления и поиска слов в словаре, сохраняя при этом баланс дерева и обеспечивая оптимальное распределение элементов по его узлам.