Лабораторная работа №2 по курсу дискретного анализа: сбалансированные деревья.

Выполнил студент группы М80-208Б-20 Морозов Артем Борисович.

Условие

Кратко описывается задача:

1. Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которых разработать программу-словарь. Ключ – регистронезависимые слова <= 256 символов, значения – числа в диапазоне от 0 до 2^64 – 1 (unsigned long long).
2. Необходимо реализовать структуру данных PATRICIA.

Метод решения

Метод решения довольно понятен: чтобы реализовать словарь, который требуется по заданию, мы должны реализовать структуру данных PATRICIA. Особенность PATRICIA в том, что это по сути не tree, к которым мы все привыкли (BST, RB, AVL, B), а trie – сжатое дерево. Однако PATRICIA – это усовершенствованный trie, так как, в отличие от остальных видов trie, в PATRICIA содержится бит, который должен проверяться с целью выбора пути из этого узла. Также в PATRICIA присутствуют обратные связи, которые при необходимости укажут нам на нужный узел. Таким образом, PATRICIA очень похожа на бинарное дерево и, по сути, выполняет весь его функционал. Однако если в BST время поиска было O(h), где h – высота дерева, в PATRICIA для выполнения одного поиска требуется всего лишь lgN сравнений разрядов и одного сравнения полного ключа, где N – количество узлов в дереве. В этом у PATRICIA есть определенное преимущество.

Описание программы

Программа выполняет 3 подзадания: реализацию основных операций с PATRICIA, битовое сравнение ключей во время работы основных операций, а также загрузку дерева в файл и чтение из него. Дерево реализуется на основе двух структур: PNode и Patricia. В PNode входят следующие поля: std:: string key для хранения ключа по заданию, unsigned long long value для хранения значения по заданию, PNode* left и *right – указатели налево и направо, а также поле short int index для хранения проверяемого бита в конкретной вершине. В структуре Patricia содержится только одно поле – это указатель на голову нашего дерева: PNode* header. Далее в структуре реализуются основные методы – проверка дерева на пустоту, поиск в дереве, вставка в дерево, его удаление и полная очистка. Структуры PNode и Patricia

находятся в специальном пространстве имен PatriciaTrie, чтобы отделить функции деревца от функций иного предназначения. Во время реализации поиска, вставки и удаления были разработаны специальные функции ToLower, GetIndexBit и FirstDifferentBit, которые переводят строку в нижний регистр, получают конкретный бит (0 или 1) в переданной строке и находят в двух переданных строках первый различающийся бит соответственно. Они также описаны в специальном пространстве имен StringBitFunctions. Ниже описываю полный функционал своей лабораторной работы.

void ToLower(std:: string& key) - переводит строку в нижний регистр
bool GetIndexBit(std:: string& string, short int index) - возвращает 1 если бит по индексу index в строке string 1, 0 если 0
short int FirstDifferentBit(std:: string& first_string, std:: string& second_string) - возвращает первый различающийся бит в переданных строках
bool IsTreeEmtpy() - проверяет, пустое дерево или нет
void Search(std:: string& key) - выполняет поиск в дереве
bool Insert(std:: string& key, const unsigned long long& value) - выполняет вставку в дерево
void Erase(std:: string& key) - удаляет ноду с ключом key
void ClearNode(PNode* node) - рекурсивная чистка дерева
void Clear() - запуск рекурсивной чистки дерева от головы
void SaveToFile(PNode* current_node, std:: ofstream& f) - запись дерева в файл
void LoadFromFile(std:: ifstream& f) - чтение дерева из файл

Дневник отладки

Дневник отладки во 2 лабораторной работе действительно очень и очень послужной. Первым делом не проходил шестой тест, так как изначально была принята идея реализовывать перевод строки сразу при ее вводе, причем при этом добивать строку ведущими нулями до максимальной длины (1280 == 256*5). Идея успехом совсем не увенчалась, так как чекер показывал ML, да и сразу было понятно, что по памяти это было не выгодно. После усовершенствования алгоритма и перевода только нужных символов в биты без ведущих нулей чекер не принимал 13 тест, выдавав RE. Я сразу понял, что это файлы, так как в них и был изначально не уверен. Мне пришлось поменять взгляд на работу с файловой системой и использовать более низкоуровневый поход – бинарную запись. При работе с файлом как с текстовым возникала проблема некой пустой строки, которая впоследствии почему-то не конвертировалась в ull при добавлении в дерево. С новым видом работы с файловой системой проблемы были быстро и незамедлительно устранены. Последняя проблема – 15 тест, который также показывал ML. Вот тут мне пришлось окончательно поменять свою идею кодировки

и не менять строки сразу же на последовательности бит. Было принято решение хранить в нодах дерева изначальные строки, а при сравнении и выделение нужного нам бита просто в специальной функции при помощи битовых операций и битовых сдвигов смотреть конкретный бит, что намного-намного-НАМНОГО быстрее. В итоге чекер выдал вердикт ОК за время 3,571 сек и с использованной памятью 223.57mb.

Тест производительности

В своем бенчмарк-тесте я сравниваю работу своей PATRICIA с обычным контейнером std:: map из STL. Я создаю 4 файла, в которых будет 1000, 10000, 100000 и 500000 строк соответственно с разными командами, связанными непосредственно с операциями в дереве (добавление, удаление и поиск), и засекаю время работы std:: map и PATRICIA.

1 тест. 1000 строк:

Результат выполнения на контейнере std:: map:

DA > lab2 > benchmark > MAP_RESULTS.txt

```
1  Insertion time in map: 25 ms
2  Searching time in map: 21 ms
3  Erasing time in map: 17 ms
4
```

Результат выполнения на PATRICIA:

DA > lab2 > benchmark > PATRICIA_RESULTS.txt

```
1  Insertion time in PATRICIA: 2 ms
2  Searching time in PATRICIA: 3 ms
3  Erasing time in PATRICIA: 1 ms
4
```

2 тест. 10000 строк.

Результат выполнения на std:: map:

DA > lab2 > benchmark > MAP_RESULTS.txt

```
1  Insertion time in map: 338 ms
2  Searching time in map: 927 ms
3  Erasing time in map: 328 ms
4
```

Результат выполнения на PATRICIA:

DA > lab2 > benchmark > PATRICIA_RESULTS.txt

```
1  Insertion time in PATRICIA: 42 ms
2  Searching time in PATRICIA: 68 ms
3  Erasing time in PATRICIA: 42 ms
4
```

3 тест. 100000 строк.

Результат выполнения на std:: map:

DA > lab2 > benchmark > MAP_RESULTS.txt

```
1  Insertion time in map: 3170 ms
2  Searching time in map: 3406 ms
3  Erasing time in map: 2954 ms
4
```

Результат выполнения на PATRICIA:

DA > lab2 > benchmark > PATRICIA_RESULTS.txt

```
1  Insertion time in PATRICIA: 732 ms
2  Searching time in PATRICIA: 1165 ms
3  Erasing time in PATRICIA: 400 ms
4
```

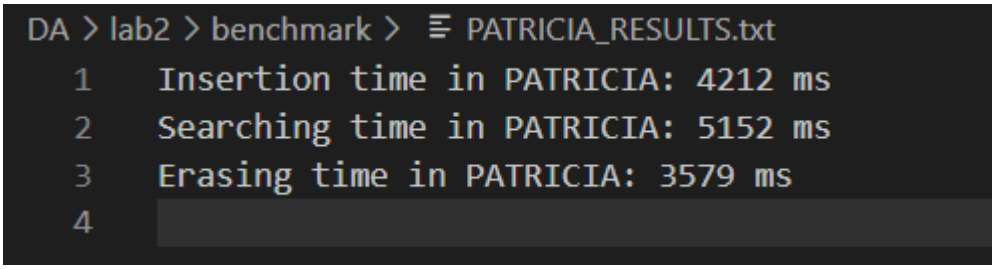
4 тест. 500000 строк.

Результат выполнения на std:: map:

DA > lab2 > benchmark > MAP_RESULTS.txt

```
1  Insertion time in map: 17496 ms
2  Searching time in map: 14291 ms
3  Erasing time in map: 16357 ms
4
```

Результат выполнения на PATRICIA:



Как мы видим, PATRICIA работает намного быстрее. Как я уже и говорил, связано это с тем, что мы пропускаем множество ненужных сравнений благодаря особенностью, заключающейся в хранении нужного бита, по которому мы проверяем наши ключи в нодах.

Недочёты

Недочетов в программе обнаружено не было, однако стоит упомянуть, что программа работает только при условии корректного ввода, так как была разработана исключительно в учебных целях. Любой неправильный ввод может убить работоспособность моей программы.

Выводы

Данная лабораторная работа была очень непростой, однако и дала мне очень многое. Я узнал о такой прекрасной структуре данных, как trie, в частности о PATRICIA-trie. Многие говорят, что это очень сложная структура данных, однако я же наоборот считаю, что это очень удобная и простая в понимании вещь! В ней нет балансировок, она максимально проста, как тот же BST, но при этом работает эффективнее него! Несмотря на то, что, насколько я знаю, PATRICIA-trie мало где используется, и о ней мало информации, я думаю, что она отлично справляется с функционалом, подобным второй лабораторной – с хранением какого-то большого похожих между собой данных.