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

Выполнил студент группы М80-208Б-20 Фаттяхетдинов Сильвестр Динарович.

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

Для реализации словаря необходимо использовать структуру данных PATRICIA. Отличительной чертой PATRICIA перед остальными деревьями является то, что это trie – «сжатое» дерево – а не tree. 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, в ней присутствуют обратные связи, которые при необходимости могут указать требуемый узел. Выходит, что PATRICIA довольно похожа на бинарное дерево и, по сути, выполняет весь его функционал. Однако если в нём 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 и заключается преимущество PATRICIA.

Описание программы

Любую большую задачи стоит разбить на подзадачи, что я и сделал. Было необходимо сделать следующее:

Реализовать оптимальную по памяти кодировку строк

Реализовать саму структуру данных PATRICIA и её основные методы

Реализовать сохранение дерева в бинарный файл с возможностью последующей выгрузки

Для реализации дерева были созданы два класса - Node и Patricia – для узла дерева и самого дерева соответственно.

В Node входят следующие поля:

std::string key - ключ

unsigned long long value - значение

int index – индекс, который проверяется при обходе

Node *left – указатель на левого потомка

Node *right – указатель на правого потомка

Patricia содержит одно единственное поле – объект root, экземпляр класса Node – указатель на корень дерева. Зато методов там предостаточно, ниже указан их полный список:

```
bool IsEmpty()- проверка дерева на пустоту
void Search(std::string &key) - поиск узла в дереве
bool Add(std::string &key, const unsigned long long &value) - добавление узла
в дерево
void Delete(std::string &key) - удаление узла из дерева
void ClearTree() + void ClearSubTree(Node *node) - полная очистка дерева
void Save(Node *node, std::ofstream &file) - сохранить дерево в бинарный файл
void Load(std::ifstream &file) - загрузить дерево из бинарного файла
void PrintTree(Node *node) - печать дерева (использовалось для дебага)
```

Также реализованы самостоятельные функции, которые используются в вышеуказанных методах:

```
bool GetIndex(std::string &word, int index) - получить бит по позиции index
(1 или 0)
int FirstDifferentBit(std::string &word1, std::string &word2) - получить
первый различающийся между двумя словами бит
```

Дневник отладки

Работу пришлось переделывать и не раз. Первыми, как обычно у меня это бывает, были ошибки в роде «Вывело Exists вместо Exist», поэтому программа падала на первом тесте.

Следующей остановкой стал 5ый тест, проблема была в функции удаления, т. к. я вначале планировал хранить коды в массивах bool’ов, а не строках, но потом передумал, забыл заменить условия вроде if(a) на if(a==’1’).

Непростым был 6ой тест – пришлось сменить кодировку, так как появлялся memory limit, оно и понятно – дополнять строку из одной буквы до длины 256 и кодировать её – получалось 1275 лишних байт на однобуквенную строку, что в корне не эффективно.

На 13 тесте пришлось повозиться с файлами – я использовал обычный filestream, файлы «не были бинарными», из-за чего получался runtime error.

После 15 теста пришлось снова переделать метод кодировки. В нодах стал хранить сами строки, а не их коды (экономия памяти уже в 5 раз), а для сравнения использовались битовые сдвиги.

Тест производительности

Я реализовал тесты вставок, поиска и удаления элементов для 1.000, 10.000 и 100.000 элементов, сравнивать эффективность PATRICIA буду с std::map.

ТЕСТ1 – 1.000 элементов

map

```
Pushed 1000 elements in 22ms
Found 1000 elements in 12ms
Deleted 1000 elements in 9ms
```

PATRICIA

```
Pushed 1000 elements in 49ms
Found 1000 elements in 32ms
Deleted 1000 elements in 15ms
```

ТЕСТ2 – 10.000 элементов

map

```
Pushed 10000 elements in 186ms
Found 10000 elements in 152ms
Deleted 10000 elements in 101ms
```

PATRICIA

```
Pushed 10000 elements in 343ms
Found 10000 elements in 148ms
Deleted 10000 elements in 102ms
```

ТЕСТ3 – 100.000 элементов

map

```
Pushed 100000 elements in 3024ms
Found 100000 elements in 2648ms
Deleted 100000 elements in 1817ms
```

PATRICIA

```
Pushed 100000 elements in 3193ms
Found 100000 elements in 2626ms
Deleted 100000 elements in 1537ms
```

Как видно, PATRICIA имеет линейную сложность и её время работы сопоставимо со временем работы std::map.

Недочёты

При условии **корректного** ввода программа должна работать корректно. Это и является недочётом – проверка на корректный ввод отсутствует.

Выводы

Данная лабораторная работа оказалась одной из самых сложных за всё время моего обучения, но, преодолевая эту сложность, я узнал о такой структуре данных как PATRICIA – крайне необычном дереве, о котором до ЛР2 по дискретному анализу я даже и не слышал.