Лабораторная работа № 2 по курсу дискретного анализа: Сбаласированные деревья

Выполнил студент группы 08-207 МАИ *Чекменев Вячеслав Алексеевич.*

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. Вариант задания: Патриция.

3. Операции: вставка, поиск, удаление элемента, сохранение словаря в файл, загрузка словаря из файла.

Метод решения

Из прошлого опыта я решил выделить для узла и дерева отдельные классы с независимой структурой для лучшей модульности.

Описание программы

Программа состоит из:

- двух заголовочных файлов, один принадлежит классу самого дерева, второй структуре узла дерева
- двух файлов с описанием функций для класса и структуры
- и клиента

Основые методы дерева:

- $ReturnNode()$ – идея состоит в том, чтобы спускаться по дереву вниз, пока бит не станет меньше либо равен биту с прошлого рекурсивного вызова, тогда останавливаем рекурсию и сравниваем наше значение со значением в нвнешнем узле. Если они равны, то узел найден, иначе - нет.
- $AddNode()$ – делаем то же, что и в $ReturnNode()$, если узел не найден, то ищем элемент еще раз, но с условием, что бит (различия) должен только увеличиваться, тогда между узлом, в который придем и его отцом вставим элемент.

- *DeleteNode*() – здесь будет сложно уместить, но идея в том, что находим элемент Bnode, который ссылается на наш элемент Anode снизу, тогда есть всего три случая:

    1. Bnode лист – все его ссылки ведут наверх
    2. Bnode == Anode
    3. Bnode – не лист, самый сложный случай, но надо по сути переназначить ссылки, что работает за константу

Дневник отладки

1. 17.04 – начал писать поиск и вставку исключительно по лекциям, не подглядывая в алгоритм
2. 19.04 – ничего почти не работает, решил посмотреть статью на медиуме по патриции
3. 23.04 – написал удаление и вставку, однако использовал указатель на отца, что выглядит как плохой код
4. 24.04 – исправил указатель на отца, написав функцию по поиску отца
5. 25.04 – написал удаление и заслал на чекер, получил WA на 5 тесте, думаю, что это связано со вставкой
6. 27.04 – нашел ошибку во вставке, неправильно распределял указатели прошел 5 тест, получил ML на 6 тесте
7. 29.04 – переписал предствление ключа в структуре так чтобы хранить не последовательность из 0 и 1, а слово, записанное латиницей, получил WA на 7 тесте, думаю, что это связано с удалением
8. 01.05 – сделал так, что бит отличия будет отсчитываться от правого конца слова, однако пройти 7 тест это не помогло
9. 03.05 – нашел ошибку в удалении, неправильно было прописано удаление при случае 2, исправил, прошел тест 7, получил WA на 9, это уже файлы
10. 05.05 – начал писать файлы, идея была в том, чтобы просто вставлять в новое дерево при загрузке
11. 06.05 – получил RE на 12 тесте, подумал, что это связано с идеей загрузки через вставку, переписал так, чтобы вставки не было, а узлы просто создавались в памяти, однако получил RE на 9 тесте

12. 06.05 – устал перекидывать код в один файл, чтобы отослать на чекер, решил воспользоваться make, однако постоянно получал CE из-за Time Limit

13. 08.05 – спросил у преподавателей и 10 одногруппников, в итоге мы нашли ошибку, она была в том, что я по привычке подключил библиотеку bits/stdc++.h, которая отнимает много времени для компиляции, подключил все библиотеки отдельно, заработало

14. 10.05 – вернулся к прошлой идее и нашел две ошибки –

    (a) Ссылался на NULL в одном месте

    (b) Не проверял файл на пустоту при загрузке

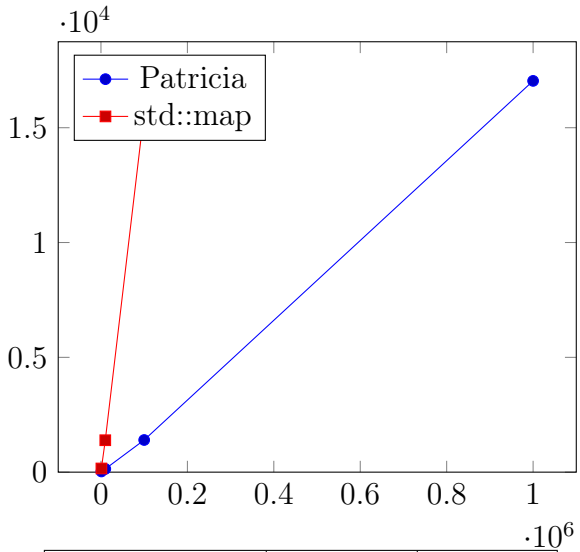
    исправил, получил OK.

Здесь я не изложил все ошибки, так как было много мелких чисто по невнимательности.

Тест производительности

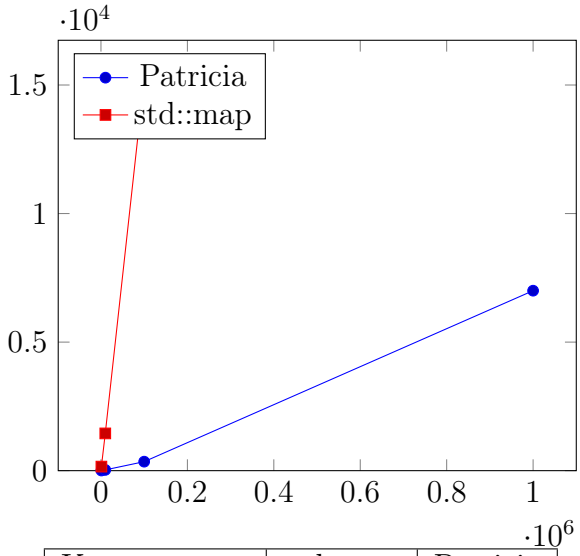
Во всех графиках по оси Y отложено время выполнения (в миллисекундах), по оси X — количество команд.

Вставка



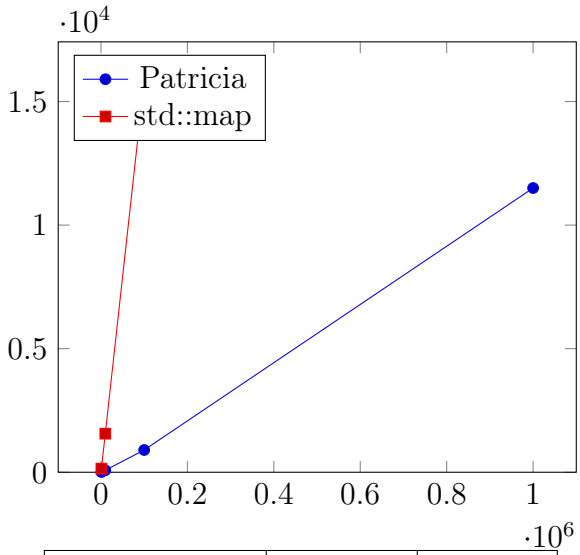
Кол-во строк	std::map	Patricia
1000	160	30
10000	1390	140
100000	15703	1400
1000000	—	17040

Поиск



Кол-во строк	std::map	Patricia
1000	158	6
10000	1449	30
100000	15218	350
1000000	—	7000

Удаление



Кол-во строк	std::map	Patricia
1000	146	10
10000	1561	80
100000	15832	900
1000000	—	11500

Таким образом, операции над Patricia имеют ту же сложность, что и над std::map, по времени работает быстрее чем std::map, однако я думаю, что это из-за специфики рандомных данных, так как патриция работает по маске. Поэтому больше узлов больше занимают память, однако это повышает скорость.

Недочёты

В коде, который получил OK на чекере некоторые функции делают немного больше, чем нужно, например функция поиска всегда считает бит различия, однако это нужно только один раз при вставке. А также написано много других ненужных функций чисто для отладки, таких как Traverse(), Print()...

Выводы

Эта лабораторная работа оказалась самой сложной за два курса, я никогда так долго не работал над одной задачей, могу выделить основые моменты, которые, я думаю, окажутся полезными в будущем:

1. не писать грязный код, на каждом шаге очищать его от ненужного мусора, который в будущем может помещать и вызвать баги
2. работа с отладчиком, я использовал gdb, узнал много новых фичей в нем, которые помогли найти много segfault-ов
3. распределение программы на несколько частей, не только физическое разделение на файлы, но и логическое
4. прорабатывать одну идею до конца, не переключаться на другие, пока не зайду в тупик
5. использовать контроль версий git: я использовал github, туда заливал код в новую ветку и, если считал, что этот чем-то сильно лучше того, что лежит в мастер ветке, делал merge, так мне была видее весь прогресс

В общем данная ЛР выдалась сложной, но интересной и полезной.