Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: Н. А. Абдыкалыков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-23
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №2

Задача: Требуется разработать программу-словарь для хранения пар (слово, число). Программа должна в интерактивном режиме обрабатывать команды добавления, удаления, поиска элементов, а также выполнять сохранение и загрузку всего словаря в компактный бинарный файл.

Вариант структуры данных: Реализация словаря на основе структуры данных PATRICIA.

Ограничения на ключи: `Ключи - регистронезависимые слова до 256 знаков латинского алфавита.`

1 Описание

Для решения задачи была реализована структура данных **PATRICIA-дерево** — пространственно-эффективная реализация префиксного дерева (Trie). В отличие от обычного Trie, которое создает узел для каждого символа, PATRICIA **пропускает общие префиксы и создает узлы ветвления только в точках первого битового различия между словами.** Этот принцип построения "сжатого"дерева обеспечивает значительную экономию памяти. Реализация поддерживает регистронезависимые операции путем приведения ключей к нижнему регистру Дерево состоит из **листовых узлов**, хранящих пару ключ-значение, и **внутренних узлов**, которые содержат только индекс бита для навигации.

Основные операции основаны на побитовом анализе ключей:

- **Поиск (Find):** Спуск по дереву на основе битов ключа до листового узла с последующей полной проверкой ключа.
- **Вставка (Insert):** Поиск ближайшего ключа, определение первого различающегося бита и вставка нового внутреннего узла ветвления на этом уровне.
- **Удаление (Remove):** Поиск узла, его удаление вместе с родительским узлом ветвления и "поднятие"соседней ветви для сохранения структуры.

Для команд `! Save` и `! Load` реализован механизм сериализации. Он использует рекурсивный обход для записи дерева в бинарный файл с маркерами типов узлов и "магическим числом"для проверки целостности файла.

2 Исходный код

Программа написана на языке C++ с использованием стандартной библиотеки. Код структурирован модульно и разделен на три логических блока: вспомогательные утилиты, класс `Patricia`, реализующий основную структуру данных, и набор функций-обработчиков команд.

Основная логика инкапсулирована в классе `Patricia`, а взаимодействие с пользователем происходит в функции `main` через вызовы функций-обработчиков.

Структура программы	
`class Patricia`	Основной класс, инкапсулирующий PATRICIA-дерево. Содержит приватную структуру `Node` для узлов дерева и реализует всю логику работы со словарем.
`int main()`	Главная функция. Организует цикл чтения команд из стандартного ввода, вызывает соответствующие функции-обработчики для каждой команды и отлавливает возможные исключения.
`handle_add, handle_remove, handle_search, handle_bang_command`	Функции-обработчики команд. Разбирают строку с командой, извлекают аргументы (слово, число, путь к файлу) и вызывают соответствующие публичные методы класса `Patricia`.
`Patricia::insert, remove, find`	Публичные методы класса `Patricia`, реализующие основные операции словаря: добавление, удаление и поиск ключа в дереве.
`Patricia::save, load`	Публичные методы, отвечающие за сериализацию (сохранение) и десериализацию (загрузку) всего дерева в компактный бинарный формат.
`to_lower, get_bit`	Вспомогательные функции. `to_lower` приводит строку к нижнему регистру для регистронезависимых операций, а `get_bit` позволяет получить значение конкретного бита в строке, что является основой для навигации по дереву.

Структура для хранения данных о тексте определена следующим образом:

```
#include <string>
#include <cstdint>
#include <memory>
struct Node {
std::string key;
uint64_t value;
int bit_index;

std::shared_ptr<Node> left;
std::shared_ptr<Node> right;

Node(const std::string& k, uint64_t v);
Node(int b_idx);

bool is_leaf() const;
};
```

3 Консоль

Демонстрация работы программы: компиляция исходного кода, просмотр содержимого тестового файла `input.txt`, содержащего последовательность команд для словаря, и запуск исполняемого файла с перенаправлением ввода для их обработки. Для наглядности сверхдлинные строки в выводе команды `cat` были сокращены.

```
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -std=c++17 -
O2 -o patricia_dict patricia_dict.cpp
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ cat input.txt
+ a 1
+ A 2
+ aaaaa...aaaa 18446744073709551615
aaaaa...aaaa
A
- A
a
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ ./patricia_dict < input.txt
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
```

4 Тест производительности

Для оценки эффективности реализованного PATRICIA-дерева было проведено сравнение его производительности со стандартной реализацией словаря `std::map` из STL, который обычно основан на сбалансированном красно-черном дереве. Асимптотическая сложность поиска в PATRICIA-дереве составляет $O(K)$, где K — длина ключа. Время операции не зависит от количества элементов в словаре. Сложность поиска в `std::map` составляет $O(K \cdot \log N)$, где N — количество элементов, так как на каждом из $\log N$ уровней дерева происходит сравнение строк длиной K.

Тестирование проводилось на большом наборе данных: в обе структуры (PATRICIA и `std::map`) было добавлено 1,000,000 (10^6) уникальных слов, после чего выполнялся 1,000,000 операций поиска. Для замера времени использовалась библиотека `<chrono>`. Компиляция обоих вариантов производилась с флагом оптимизации `-O2`.

```
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -O2 -std=c++17 patricia_pe
o patricia_tester
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ ./patricia_tester
PATRICIA tree: 1,000,000 insertions and 1,000,000 lookups took: 1.2158 seconds

laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -O2 -std=c++17 map_perf.c
o map_tester
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ ./map_tester
std::map: 1,000,000 insertions and 1,000,000 lookups took: 2.5831 seconds
```

Как показывают результаты, реализация на PATRICIA-дереве выполняет поиск примерно в 2 раза быстрее, чем высокооптимизированная структура `std::map`. Это объясняется фундаментальным различием в их алгоритмах. Время поиска в PATRICIA не зависит от общего числа элементов в словаре (N), а определяется только длиной ключа (K), так как навигация по дереву происходит напрямую на основе битов ключа. В то же время, производительность `std::map` деградирует с ростом числа элементов из-за логарифмического фактора $\log N$, связанного с глубиной дерева. Это демонстрирует преимущество PATRICIA-дерева для задач, требующих максимально быстрых строковых поисков в очень больших словарях.

5 Valgrind

Для проверки программы на наличие утечек памяти и других ошибок при работе с ней была использована утилита `Valgrind` с инструментом `Memcheck`. Сначала была выполнена компиляция программы с флагами `-g` (для добавления отладочной информации) и `-O0` (для отключения оптимизации компилятора). Тестирование проводилось на наборе команд, который задействует все основные операции с динамической памятью: создание, сложное ветвление и удаление узлов дерева. Запуск `Valgrind` производился с флагом `-leak-check=full` для максимально детального анализа.

```
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -std=c++17 -
g -O0 -o patricia_dict patricia_dict.cpp
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ valgrind --leak-check=full ./patrici
==2693== Memcheck, a memory error detector
==2693== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==2693== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==2693== Command: ./patricia_dict
==2693==
OK
OK
OK
OK: 10
OK
NoSuchWord
OK
OK: 10
OK
==2693==
==2693== HEAP SUMMARY:
==2693==     in use at exit: 122,880 bytes in 6 blocks
==2693==   total heap usage: 12 allocs, 6 frees, 197,088 bytes allocated
==2693==
==2693== LEAK SUMMARY:
==2693==    definitely lost: 0 bytes in 0 blocks
==2693==    indirectly lost: 0 bytes in 0 blocks
==2693==      possibly lost: 0 bytes in 0 blocks
==2693==    still reachable: 122,880 bytes in 6 blocks
==2693==         suppressed: 0 bytes in 0 blocks
==2693==
```

==2693== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Анализ результатов: Как видно из вывода `Valgrind`, в программе-словаре отсутствуют утечки памяти, что подтверждается строкой `definitely lost: 0 bytes in 0 blocks`. Некоторое количество памяти помечено как `still reachable`, что является нормальным поведением для программ, использующих стандартную библиотеку C++, и не считается ошибкой в коде. Отчет `ERROR SUMMARY: 0 errors` также подтверждает отсутствие других ошибок при работе с памятью. Это свидетельствует о корректной реализации управления ресурсами с помощью умных указателей `std::shared_ptr`.

6 gprof

Для проверки эффективности программы и определения функций, на которые приходится основная вычислительная нагрузка, используется профилировщик `gprof`. Сначала программа компилируется с флагами `-pg` для записи данных профилирования и `-O2` для включения оптимизации, чтобы анализ отражал производительность реального приложения. Проверка выполняется на тестовом наборе команд, после чего создается файл `gmon.out`. Затем выполняется анализ данных профилирования с помощью `gprof` и бинарного файла программы-словаря, и на их основе создается отчет.

laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ g++ -std=c++17 -
O2 -pg -o patricia_gprof patricia_dict.cpp
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ ./patricia_gprof < input.txt > /de
laysou@DESKTOP-QPGEK53:/mnt/c/Users/abdyk/Desktop/Lr2$ gprof ./patricia_gprof gmon.out | h
n 15

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	6	0.00	0.00	frame_dummy
0.00	0.00	0.00	5	0.00	0.00	std::_Sp_counted_ptr_inplace<Patricia::Node, ...>::_M_destr
0.00	0.00	0.00	5	0.00	0.00	std::_Sp_counted_ptr_inplace<Patricia::Node, ...>::_M_dispo
0.00	0.00	0.00	3	0.00	0.00	handle_add(Patricia&, ...)
0.00	0.00	0.00	3	0.00	0.00	handle_remove(Patricia&, ...)
0.00	0.00	0.00	3	0.00	0.00	handle_search(Patricia&, ...)
0.00	0.00	0.00	3	0.00	0.00	Patricia::insert(...)
0.00	0.00	0.00	3	0.00	0.00	Patricia::remove(...)

Анализ результатов: Из раздела "плоский профиль"отчета можно сделать вывод о структуре вызовов в программе. Поскольку выполнение на малом тестовом наборе данных происходит практически мгновенно, профилировщик сообщает `no time accumulated`. Однако отчет все еще информативен благодаря столбцу `calls`, который показывает количество вызовов каждой функции.

Видно, что высокоуровневые функции-обработчики (`handle_add`, `handle_remove` и т.д.) и соответствующие им ключевые методы класса (`Patricia::insert`, `Patricia::remove`) были вызваны по 3 раза, что точно соответствует командам в тестовом файле. Основная работа программы, таким образом, сосредоточена внутри этих ключевых методов, а не во вспомогательных операциях, что говорит о правильной архитектуре приложения. Также в отчете присутствуют вызовы внутренних функций стандартной библиотеки (например, `_M_destroy` и `_M_dispose` для умных указателей `shared_ptr`), что подтверждает корректное управление памятью.

7 Выводы

PATRICIA-дерево является высокоэффективной структурой данных для реализации словарей со строковыми ключами. Его ключевое преимущество — временная сложность операций $O(K)$ (где K — длина ключа), которая не зависит от общего количества элементов N в структуре. Это обеспечивает стабильно высокую производительность даже на очень больших наборах данных, в отличие от сбалансированных деревьев (например, `std::map`), чья сложность $O(K \cdot \log N)$ деградирует с ростом N.

В ходе выполнения лабораторной работы была реализована программа-словарь на основе PATRICIA-дерева. Основной задачей было создание производительной структуры, превосходящей стандартные аналоги на больших объемах данных. Это было достигнуто благодаря реализации ключевого принципа PATRICIA: навигации по дереву на основе отдельных битов ключа, что позволяет избежать дорогостоящих полных сравнений строк на каждом шаге спуска. Этот опыт наглядно демонстрирует, как выбор специализированной структуры данных, учитывающей природу хранимой информации, может кардинально повысить производительность, и знакомит с устройством одного из классических и эффективных алгоритмов для работы со строковыми ключами.

Также был составлен отчет при помощи системы TEX, которая позволяет автоматизировать процесс создания качественной технической документации.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 3-е издание.* — Издательский дом «Вильямс», 2013. Перевод с английского под редакцией И. В. Красикова. — 1328 с. (ISBN 978-5-8459-1794-2)

[2] *Радиксальное дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Радиксальное_дерево` (дата обращения: 20.09.2025).

[3] *Префиксное дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Префиксное_дерево` (дата обращения: 20.09.2025).