Лабораторные работы №2-3 по курсу дискретного анализа: Сбалансированные деревья, Профилирование

Выполнил студент группы 08-203 **МАИ** ***Арусланов Кирилл***.

Условие

Общая постановка задачи:

Необходимо создать программную библиотеку, реализующую структуру данных PATRICIA (Practical Algorithm to Retrieve Information Coded in Alphanumeric), и на её основе разработать программу-словарь. В словаре каждому ключу, представляющему собой регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер от 0 до 2^64 - 1. Разным словам может соответствовать один и тот же номер.

Вариант задания:

Использование структуры данных PATRICIA.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь один из следующих форматов:

- `+ word 34` — добавить слово "word" с номером 34 в словарь. Программа должна вывести "OK", если операция прошла успешно, или "Exist", если слово уже находится в словаре.
- `- word` — удалить слово "word" из словаря. Программа должна вывести "OK", если слово существовало и было удалено, или "NoSuchWord", если слово не было найдено.
- `word` — найти слово "word" в словаре. Программа должна вывести "OK: 34", если слово найдено (где 34 — присвоенный номер), или "NoSuchWord", если слово не было обнаружено.
- `! Save /path/to/file` — сохранить словарь в бинарном компактном представлении на диск. В случае успеха вывести "OK", в случае ошибки — описание ошибки.
- `! Load /path/to/file` — загрузить словарь из файла. В случае успеха вывести "OK", в случае ошибки — описание ошибки. Программа должна корректно обрабатывать несовпадение формата файла.

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

Метод решения

Для решения задачи была реализована структура данных PATRICIA, которая является разновидностью префиксного дерева (trie), оптимизированного для хранения строк. PATRICIA

отличается от обычного trie тем, что в ней отсутствуют узлы с одним потомком, что позволяет экономить память и ускорять операции поиска, вставки и удаления.

Общий алгоритм:

1. **Вставка** (`Insert`): Для вставки ключа и соответствующего ему значения программа спускается по дереву, сравнивая биты ключа с битами в узлах. Когда достигается точка, где нужно добавить новый узел, создаётся новый узел с ключом и значением.
2. **Удаление** (`Erase`): Удаление ключа из дерева требует реорганизации структуры так, чтобы после удаления узла дерево оставалось корректным. Это может включать перестройку связей между узлами.
3. **Поиск** (`Find`): Поиск ключа выполняется аналогично вставке: спуск по дереву с сравнением битов до нахождения узла с искомым ключом или до достижения листа.
4. **Сохранение и загрузка** (`Save` **и** `Load`): Словарь сохраняется в бинарном формате на диск и загружается из файла. Формат файла включает количество узлов и данные каждого узла (ключ, значение, бит и ссылки на детей).

Для реализации использовались лекции и внешние источники, такие как статьи по PATRICIA и документация по C++.

Описание программы

Программа реализована в файле `TPatriciaTrie.cpp` и состоит из следующих основных компонентов:

- **Класс** `TPatriciaTrie`: Основной класс, реализующий структуру PATRICIA.
    - **Внутренняя структура** `Node`: Представляет узел дерева, содержащий ключ (`std::string`), значение (`std::uint64_t`), бит (`int`), и указатели на детей (`Node* children[2]`).
    - **Методы**:
        - `Insert`: Вставка ключа и значения.
        - `Erase`: Удаление ключа.
        - `Find`: Поиск ключа.
        - `Save`: Сохранение дерева в файл.
        - `Load`: Загрузка дерева из файла.
        - `DestructRecursive`: Рекурсивное освобождение памяти в деструкторе.
- **Вспомогательные функции**:
    - `BitGet`: Получение бита из строки.
    - `FirstDifferentBit`: Нахождение первого различающегося бита между двумя строками.

- **Функция** `main`: Обрабатывает команды из ввода, вызывает соответствующие методы `TPatriciaTrie` и выводит результаты.

Дневник отладки

Проблема 1: Некорректное удаление узлов в методе `Erase`.

- **Симптом**: После удаления ключа некоторые узлы оставались в дереве, что приводило к утечкам памяти.
- **Решение**: Исправлена логика удаления, чтобы корректно обновлять связи между узлами и освобождать память удалённых узлов.

Проблема 2: Ошибки при загрузке дерева из файла.

- **Симптом**: При загрузке дерева из файла возникали ошибки, связанные с несовпадением формата.
- **Решение**: Добавлена проверка формата файла и корректная обработка ошибок при чтении.

Проблема 3: Низкая производительность при большом количестве операций.

- **Симптом**: Программа работала медленно на больших тестах.
- **Решение**: Оптимизированы функции `FirstDifferentBit` и `BitGet` с использованием побитовых операций.

Тест производительности

Методика:

- Запущены тесты с разным количеством операций: 100,000, 300,000 и 1,000,000 для вставок, удалений и поисков.
- Измерено время выполнения с помощью `gprof` и теста производительности на разных объемах входных данных.
- Сравнены результаты, чтобы проверить, что рост времени работы при увеличении объема входных данных согласуется с заявленной сложностью.
- Проведён анализ памяти с помощью `valgrind --tool=memcheck` для проверки на утечки памяти и корректность управления ресурсами.

Результаты:

- Для 100,000 операций: общее время ~0.15 секунды.
- Для 300,000 операций: общее время ~0.35 секунды.
- Для 1,000,000 операций: общее время ~1.47 секунды.

- Теоретически сложность операций PATRICIA (вставка, удаление, поиск) составляет O(m), где `m` — длина ключа, так как операции зависят от количества битов, которые нужно сравнить. Однако в тестах ключи имеют примерно одинаковую длину (например, `key1`, `key300000` — длиной 4–8 символов), поэтому m можно считать константой. Таким образом, общее время выполнения для `n` операций пропорционально `n * m`, что в данном случае даёт линейный рост времени относительно `n`.

Анализ производительности с помощью `gprof`

Для теста с 1,000,000 вставок, 1,000,000 удалений и 1,000,000 поисков (всего 3,000,000 операций) был получен следующий профиль `gprof`:

Вывод `gprof` (Flat Profile)

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 51.70      0.76     0.76  1000000   760.00   772.90
TPatriciaTrie::Insert(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, unsigned long)
 28.57      1.18     0.42  1000000   420.00   427.10
TPatriciaTrie::Erase(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&)
 13.61      1.38     0.20                             main
  4.76      1.45     0.07                             _init
  1.36      1.47     0.02  1549978    12.90    12.90  frame_dummy
  0.00      1.47     0.00        1     0.00    12.90  TPatriciaTrie::TPatriciaTrie()


                     Call graph


granularity: each sample hit covers 4 byte(s) for 0.68% of 1.47 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     95.2    0.20    1.20                 main [1]
                0.76    0.01 1000000/1000000
TPatriciaTrie::Insert(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, unsigned long) [2]
                0.42    0.01 1000000/1000000
TPatriciaTrie::Erase(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) [3]
                0.00    0.00       1/1           TPatriciaTrie::TPatriciaTrie() [6]
                0.00    0.00       1/1549978     frame_dummy [5]
-----------------------------------------------
                0.76    0.01 1000000/1000000     main [1]
[2]     52.6    0.76    0.01 1000000
TPatriciaTrie::Insert(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, unsigned long) [2]
                0.01    0.00 1000000/1549978     frame_dummy [5]
-----------------------------------------------
```

```
                0.42    0.01 1000000/1000000     main [1]
[3]     29.1    0.42    0.01 1000000
TPatriciaTrie::Erase(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) [3]
                0.01    0.00  549976/1549978     frame_dummy [5]
-----------------------------------------------
                                                 <spontaneous>
[4]      4.8    0.07    0.00                 _init [4]
-----------------------------------------------
                0.00    0.00       1/1549978     main [1]
                0.00    0.00       1/1549978     TPatriciaTrie::TPatriciaTrie() [6]
                0.01    0.00  549976/1549978
TPatriciaTrie::Erase(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) [3]
                0.01    0.00 1000000/1549978
TPatriciaTrie::Insert(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, unsigned long) [2]
[5]      1.4    0.02    0.00 1549978         frame_dummy [5]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[6]      0.0    0.00    0.00       1         TPatriciaTrie::TPatriciaTrie() [6]
                0.00    0.00       1/1549978     frame_dummy [5]
-----------------------------------------------
```

Интерпретация вывода `gprof`

- **Общее время выполнения**: Программа завершила 3,000,000 операций за 1.47 секунды, что указывает на высокую эффективность структуры данных PATRICIA.
- `TPatriciaTrie::Insert`:
  - Занимает 51.70% времени (0.76 секунды), что является наибольшей долей. Это ожидаемо, так как вставка требует сравнения битов ключей и реорганизации дерева.
  - Среднее время на вызов: 760 наносекунд, что приемлемо для операции вставки.
- `TPatriciaTrie::Erase`:
  - Занимает 28.57% времени (0.42 секунды). Удаление требует меньше времени, чем вставка, так как не создаёт новых узлов, но всё равно включает реорганизацию.
  - Среднее время на вызов: 420 наносекунд.
- `main`:
  - Занимает 13.61% времени (0.20 секунды). Это связано с обработкой ввода/вывода и парсингом данных.

Выводы из `gprof`

1. **Основные затраты времени**: Операции вставки и удаления составляют основную часть времени выполнения, что соответствует теоретической сложности PATRICIA-дерева.

2. **Сложность операций**: Среднее время на вызов (`Insert`: 760 нс, `Erase`: 420 нс) соответствует сложности O(m), где `m` — длина ключа. В тестах m почти константно, что приводит к линейному росту времени относительно количества операций `n`.
3. **Потенциал оптимизации**: Ускорение вспомогательных функций (`FirstDifferentBit`, `BitGet`), часто вызываемых внутри `Insert` и `Erase`, может дополнительно улучшить производительность.

Анализ памяти с помощью `valgrind`

Для теста с 900,000 операций был выполнен анализ памяти с использованием `valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all --leak-resolution=med`.

- `--tool=memcheck`: Использует инструмент memcheck для анализа памяти. Это стандартный инструмент valgrind для поиска утечек памяти, недействительных обращений к памяти (например, чтение/запись за пределами выделенной памяти) и использования неинициализированных значений.
- `--leak-check=full`: Выполняет полный поиск утечек памяти, включая информацию о каждом блоке памяти, который не был освобождён.
- `--show-leak-kinds=all`: Показывает все типы утечек:
    - `definitely lost`: Память, которая точно потеряна (нет указателей на неё).
    - `indirectly lost`: Память, потерянная косвенно (например, указатель на неё есть, но он сам потерян).
    - `possibly lost`: Память, которая, возможно, потеряна (есть указатель, но он, вероятно, недоступен).
    - `still reachable`: Память, которая всё ещё доступна в конце программы, но не освобождена.
- `--leak-resolution=med`: Устанавливает средний уровень разрешения для объединения похожих утечек (чтобы уменьшить количество ложных срабатываний). Это разумный выбор для большинства программ.

Вот вывод:

Вывод `valgrind`

```
==62133== HEAP SUMMARY:
==62133==     in use at exit: 122,880 bytes in 6 blocks
==62133==   total heap usage: 700,011 allocs, 700,005 frees, 28,996,747 bytes
allocated
==62133==
==62133== 8,192 bytes in 1 blocks are still reachable in loss record 1 of 6
==62133==    at 0x484A2F3: operator new[](unsigned long) (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==62133==    by 0x4985733: std::basic_filebuf<char, std::char_traits<char>
>::_M_allocate_internal_buffer() (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x4983541: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x492E52C: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-
linux-gnu/libstdc++.so.6.0.32)
```

```
==62133==    by 0x10A437: main (TPatriciaTrie.cpp:280)
==62133==
==62133== 8,192 bytes in 1 blocks are still reachable in loss record 2 of 6
==62133==    at 0x484A2F3: operator new[](unsigned long) (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==62133==    by 0x4985733: std::basic_filebuf<char, std::char_traits<char>
>::_M_allocate_internal_buffer() (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x4983541: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x492E54F: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-
linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x10A437: main (TPatriciaTrie.cpp:280)
==62133==
==62133== 8,192 bytes in 1 blocks are still reachable in loss record 3 of 6
==62133==    at 0x484A2F3: operator new[](unsigned long) (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==62133==    by 0x4985733: std::basic_filebuf<char, std::char_traits<char>
>::_M_allocate_internal_buffer() (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x4983541: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x492E572: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-
linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x10A437: main (TPatriciaTrie.cpp:280)
==62133==
==62133== 32,768 bytes in 1 blocks are still reachable in loss record 4 of 6
==62133==    at 0x484A2F3: operator new[](unsigned long) (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==62133==    by 0x4987446: std::basic_filebuf<wchar_t, std::char_traits<wchar_t>
>::_M_allocate_internal_buffer() (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x4983721: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x492E5DB: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-
linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x10A437: main (TPatriciaTrie.cpp:280)
==62133==
==62133== 32,768 bytes in 1 blocks are still reachable in loss record 5 of 6
==62133==    at 0x484A2F3: operator new[](unsigned long) (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==62133==    by 0x4987446: std::basic_filebuf<wchar_t, std::char_traits<wchar_t>
>::_M_allocate_internal_buffer() (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x4983721: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x492E5FE: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-
linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x10A437: main (TPatriciaTrie.cpp:280)
==62133==
==62133== 32,768 bytes in 1 blocks are still reachable in loss record 6 of 6
==62133==    at 0x484A2F3: operator new[](unsigned long) (in
/usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==62133==    by 0x4987446: std::basic_filebuf<wchar_t, std::char_traits<wchar_t>
>::_M_allocate_internal_buffer() (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x4983721: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x492E621: std::ios_base::sync_with_stdio(bool) (in /usr/lib/x86_64-
linux-gnu/libstdc++.so.6.0.32)
==62133==    by 0x10A437: main (TPatriciaTrie.cpp:280)
==62133==
==62133== LEAK SUMMARY:
==62133==    definitely lost: 0 bytes in 0 blocks
==62133==    indirectly lost: 0 bytes in 0 blocks
```

```
==62133==      possibly lost: 0 bytes in 0 blocks
==62133==    still reachable: 122,880 bytes in 6 blocks
==62133==         suppressed: 0 bytes in 0 blocks
==62133==
==62133== For lists of detected and suppressed errors, rerun with: -s
==62133== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Интерпретация вывода `valgrind`

- **Нет утечек памяти**: Поле `definitely lost: 0 bytes` подтверждает, что программа корректно управляет памятью.
- `still reachable: 122,880 bytes in 6 blocks`:
    - Эти 6 блоков памяти (размером 8,192 или 32,768 байт каждый) связаны с буферами ввода/вывода, выделенными стандартной библиотекой C++ (`libstdc++`) для `std::basic_filebuf<char>` и `std::basic_filebuf<wchar_t>`.
    - Причина их появления — вызов `std::ios_base::sync_with_stdio(false)` в функции `main`. Этот вызов отключает синхронизацию между потоками C++ (`std::cin`, `std::cout`) и C (`stdin`, `stdout`), что приводит к выделению внутренних буферов для ускорения ввода/вывода. Эти буферы остаются доступными до завершения программы и классифицируются как "still reachable".
- **Почему** 6 **блоков не потеряны**: Это не утечка памяти, так как буферы освобождаются операционной системой при завершении программы. Их присутствие — побочный эффект оптимизации ввода/вывода.

Связь с синхронизацией C и C++

- Вызов `std::ios_base::sync_with_stdio(false)` ускоряет работу с вводом/выводом, но приводит к выделению дополнительных буферов, которые остаются в памяти до конца выполнения программы.
- Без этого вызова буферы не выделялись бы, но производительность ввода/вывода снизилась бы из-за синхронизации с потоками C.

Итоговые выводы по тестам производительности

- **Производительность**: PATRICIA-дерево демонстрирует сложность операций O(m), где `m` — длина ключа. В тестах с почти константной длиной ключей общее время линейно зависит от количества операций n, что соответствует ожидаемому поведению.
- **Управление памятью**: Программа не имеет утечек памяти, а 6 блоков "still reachable" в `valgrind` — это результат оптимизации ввода/вывода через `std::ios_base::sync_with_stdio(false)`, что не является проблемой.
- **Рекомендации**: Для дальнейшего улучшения можно сосредоточиться на оптимизации побитовых операций в функциях `FirstDifferentBit` и `BitGet`, а также сохранить текущую настройку ввода/вывода для поддержания высокой скорости выполнения.

Недочёты

- **Отсутствие видимости мелких функций в `gprof`**: Функции `FirstDifferentBit`, `BitGet` и `Find` не отображаются из-за низкой точности выборки. Использование `callgrind` может решить эту проблему.
- **Накладные расходы на ввод/вывод**: `main` тратит время на обработку строк, что можно оптимизировать с помощью буферизации.

Выводы

Реализованная структура данных PATRICIA эффективно решает задачу хранения и поиска строк с учётом регистронезависимости. Она находит применение в системах, где требуется быстрый поиск по строковым ключам, например, в базах данных или поисковых системах.

Область применения:

- Хранение и поиск DNS-записей.
- Реализация автодополнения в текстовых редакторах.
- Сжатие данных на основе префиксов.

Сложность программирования:

- Реализация PATRICIA требует глубокого понимания работы с битами и управления памятью.
- Основные трудности связаны с корректным удалением узлов.

Возникшие проблемы:

- Обеспечение корректного удаления узлов и отсутствия утечек памяти.
- Оптимизация производительности для больших объёмов данных.

Общий итог:

- Программа успешно реализует все требуемые операции с PATRICIA-деревом.
- Отсутствуют утечки памяти, что подтверждено анализом `valgrind`.
- Производительность соответствует теоретической сложности, но может быть улучшена с помощью дополнительных оптимизаций.