Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: О. А. Мезенин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-21
Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №2

Задача: Необходимо создать программную библиотеку, реализующую структуру данных PATRICIA, 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.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).

! Load /path/to/file — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

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 на английском языке возникшую ошибку.

Также необходимо провести профилирование программы с помощью утилиты gprof и провести поиск утечек памяти с помощью утилиты valgrind.

1 Описание

PATRICIA (Practical Algorithm To Retrieve Information Coded In Alphanumeric) представляет из себя структуру данных, в которой смешаны структуры BST и Trie. Основная идея заключается в том, чтобы проходить дерево как в бинарном дереве поиска, но сравнивать ключи не полностью, а только соответствующие биты.

Каждый узел имеет, помимо исходных данных, индекс бита и указатели на левый и правый узлы. Указатели могут быть прямые (указывающие на узел ниже) и обратные (указывающие на собственный узел или узел выше).

Поиск осуществляется следующим образом. Когда идёт сравнение с очередным узлом, из исходного ключа берётся бит, чей номер соответствует индексу бита в данном узле. Если этот бит равен 0, то осуществляется переход по левому указателю, иначе – по правому. Поиск заканчивается тогда, когда происходит переход по обратному указателю. Затем ключ последнего узла сравнивается с исходным ключом, после чего делается вывод о результате поиска.

Перед вставкой ключа осуществляется поиск. Если ключ не найден, то происходит непосредственно вставка нового узла. Берётся найденный при поиске узел, затем побитого сравниваются ключ в этом узле и исходный ключ. Индекс бита нового узла будет номером первого несовпадающего бита. Затем при помощи поиска по дереву ищется такое место для вставки нового узла, что после вставки сохранится следующий принцип: индексы узлов от корня до листов должны возрастать.

Рассмотрим общий случай удаления. При удалении узла X необходимо найти такую вершину P, которая бы обратным указателем ссылалась на узел X. После этого необходимо переместить исходные данные из узла P в узел X, а узел P удалить, при этом перевязав узлы, ссылающиеся на узел P.

2 Исходный код

Интерфейс для патриции выглядит следующим образом:

```
class TPatricia {
    struct TNode {
        std::string key;
        uint64_t value;
        size_t index;
        TNode *left;
        TNode *right;

        TNode(const std::string &key, const uint64_t &value, const size_t &index)
        : key(key), value(value), index(index), left(nullptr), right(nullptr) {}
    };
// Methods
private:
    void Destroy();
    void RecursiveDestroy(TNode *node);
    void Insert(const std::string &key, const uint64_t &value, const size_t &index);
    size_t CalculateNewIndex(const std::string &inputKey, const std::string &foundedKey
        );
    TNode* Find(const std::string &key);
    std::tuple<TPatricia::TNode*, TPatricia::TNode*, TPatricia::TNode*> FindABC(const
        std::string &key);
    void SaveData(const TNode *node, std::ofstream &stream);
    void RecursiveSave(const TNode *node, std::ofstream &stream);
    TNode* LoadData(std::ifstream &stream);
public:
    ~TPatricia();
    void Add(const std::string &key, const uint64_t &value);
    void Erase(const std::string &key);
    uint64_t At(const std::string &key);
    void Save(const std::string &path);
    void Load(const std::string &path);
// Variables
private:
    TNode *root = nullptr;
};
```

patricia.cpp	
void Add(const std::string &key, const uint64_t &value)	Функция для вставки пары key-value. Выбрасывает исключение TKeyAlreadyExists, если ключ уже существует
void Insert(const std::string &key, const uint64_t &value, const size_t &index)	Функция для непосредственной вставки узла

size_t CalculateNewIndex(const std::string &inputKey, const std::string &foundedKey)	Функция ищет первый несовпадающий бит двух ключей
uint64_t At(const std::string &key)	Функция возвращает значение ключа. Выбрасывает исключение TNoSuchKey, если ключ не найден
TNode* Find(const std::string &key)	Функция поиска узла
std::tuple<TPatricia::TNode*, TPatricia::TNode*, TPatricia::TNode*> FindABC(const std::string &key)	Функция поиска узла. Возвращает найденный элемент, предыдущий за ним и предпредыдущий
void Save(const std::string &path)	Функция, запускающая рекурсивное сохранение дерева в файл
void RecursiveSave(const TNode *node, std::ofstream &stream)	Функция, рекурсивно сохраняющая узлы в файл
void SaveData(const TNode *node, std::ofstream &stream)	Вспомогательная функция для сохранения. Непосредственно записывает данные узла в файл
void Load(const std::string &path)	Функция, загружающая дерево из файла
TNode* LoadData(std::ifstream &stream)	Вспомогательная функция для загрузки данных из файла. Возвращает указатель на загруженный из файла узел
void Destroy()	Функция, запускающая рекурсивное удаление дерева
void RecursiveDestroy(TNode *node)	Функция рекурсивного удаления дерева. Удаление происходит только вниз по прямым ссылкам

В файле **custom_exception.hpp** представлены кастомные исключения.

```
1  class TCustomException : public std::exception {};
2  class TKeyAlreadyExists : public TCustomException {
3  public:
4      const char *what() const noexcept {
5          return "Exist";
6      }
7  };
8  class TNoSuchKey : public TCustomException {
9  public:
10     const char *what() const noexcept {
11         return "NoSuchWord";
12     }
13 };
```

3 Консоль

```
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ make lab2
g++ -std=c++2a -pedantic -Wall -Wextra -Werror patricia.cpp main.cpp -o lab2
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ cat my_tests/test1
+ qwerty 123
+ lol 11
-wrwr
+ warning 90823
-warning
+ lol 123
! Save ./patricia.bin
+ none 90000
none
! Load ./patricia.bin
none
warning
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ ./lab2 <my_tests/test1
OK
OK
NoSuchWord
OK
OK
Exist
OK
OK
OK: 90000
OK
NoSuchWord
NoSuchWord
```

4 Тест производительности

Тесты производительности представляют из себя следующее. В тестовом файле находятся 1000000 операций вставок, 1000000 операций поиска и 1000000 операций удаления. Реализация патриции будет сравниваться с красно-чёрным деревом из STL (std::map [1]). Время чтения элементов не учитывается. В качестве компаратора для std::map служит функция, проходящиеся по строкам и находя первое различие в регистронезависимых символах.

```
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ make benchmark
g++ -std=c++2a -pedantic -Wall -Wextra -Werror patricia.cpp benchmark.cpp -o
benchmark
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ ls -lh tests
итого 421M
-rw-rw-r--1 aprold aprold  28M апр 14 21:40 01.a
-rw-rw-r--1 aprold aprold 394M апр 14 21:40 01.t
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ ./benchmark <tests/01.t
Count of actions: 3000000
-----
Count of insert: 1000000
Patricia insert time: 1733819us
STL map insert time: 3423922us
-----
Count of find: 1000000
Patricia find time: 1604819us
STL map find time: 4419207us
-----
Count of erase: 1000000
Patricia erase time: 1868929us
STL map erase time: 3418129us
-----
Sum Patricia time: 5207567us
Sum STL map time: 11261258us
```

Как видно из теста, патриция везде выигрывает. Это по большей части связано с тем, что красно-чёрное дерево в процессе поиска сравнивает ключи, которые могут длиной 256 символов. Патриция же в процессе поиска сравнивает только один бит ключа, а полностью ключ проверяет только один раз в конце поиска.

5 Профилирование и поиск утечек памяти

Утилита **grpof** может показать, сколько времени занимает время выполнения и количество вызовов функций.

```
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ ./lab2_profiler <tests/01.t
>tmp
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ gprof ./lab2_profiler >gprof.txt
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ head -n 12 gprof.txt
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
18.64      0.33     0.33 418791102     0.00     0.00  std::__cxx11::basic_string<char
std::char_traits<char>,std::allocator<char>>::operator[](unsigned long) const
16.38      0.62     0.29  2997264     0.10     0.29  CaseInsensitiveEqual(
std::__cxx11::basic_string<char,std::char_traits<char>,
std::allocator<char>>const&,std::__cxx11::basic_string<char,
std::char_traits<char>,std::allocator<char>>const&)
12.43      0.84     0.22  1999118     0.11     0.13  TPatricia::Find(
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>const&)
9.60      1.01     0.17 239162911     0.00     0.00  std::__cxx11::basic_string<char,
std::char_traits<char>,std::allocator<char>>::size() const
8.47      1.16     0.15 438525079     0.00     0.00  std::__cxx11::basic_string<char,
std::char_traits<char>,std::allocator<char>>::_M_data() const
6.21      1.27     0.11  1224540     0.09     0.13  TPatricia::FindABC(
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>const&)
3.39      1.33     0.06   498979     0.12     0.22  TPatricia::Insert(
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>const&,
unsigned long const&,unsigned long const&)
```

Видно, что из функций патриции больше всего времени занимает поиск. А самое большое время выделяется на сравнение строк и взятие символа строки.

Утилита **valgrind** может показать, сколько байтов памяти выделено и освобождено, а также укажет на функции, в которых произошла утечка памяти, если таковая имеется.

```
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ valgrind --leak-check=full
-s ./lab2 <./tests/01.t >tmp
==40291== Memcheck,a memory error detector
```

```
==40291== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==40291== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==40291== Command: ./lab2
==40291==
==40291==
==40291== HEAP SUMMARY:
==40291==     in use at exit: 500,508 bytes in 4,988 blocks
==40291==   total heap usage: 70,164 allocs,65,176 frees,7,771,002 bytes allocated
==40291==
==40291== 500,508 (8 direct,500,500 indirect) bytes in 1 blocks are definitely
lost in loss record 5 of 5
==40291==    at 0x4849013: operator new(unsigned long) (in /usr/libexec/valgrind/vgpre
==40291==    by 0x10EA05: main (in /home/aprold/Documents/GitHub/MAI-DA/lab2/lab2)
==40291==
==40291== LEAK SUMMARY:
==40291==    definitely lost: 8 bytes in 1 blocks
==40291==    indirectly lost: 500,500 bytes in 4,987 blocks
==40291==      possibly lost: 0 bytes in 0 blocks
==40291==    still reachable: 0 bytes in 0 blocks
==40291==         suppressed: 0 bytes in 0 blocks
==40291==
==40291== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Видно, что утилита зафиксировала утечку памяти после выхода из функций main. В начале функции main выделялась память под патрицию, но в конце функции память не освобождалась. Вывод утилиты после исправления утечки памяти:

```
(venv) aprold@SAI:~/Documents/GitHub/MAI-DA/lab2$ valgrind --leak-check=full
-s ./lab2 <./tests/01.t >tmp
==40363== Memcheck,a memory error detector
==40363== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==40363== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==40363== Command: ./lab2
==40363==
==40363==
==40363== HEAP SUMMARY:
==40363==     in use at exit: 0 bytes in 0 blocks
==40363==   total heap usage: 70,164 allocs,70,164 frees,7,771,002 bytes allocated
==40363==
==40363== All heap blocks were freed --no leaks are possible
==40363==
==40363== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

6 Выводы

Выполнив данную лабораторную работу, узнал о структуре данных «Патриция» и о красно-чёрном дереве. Патрицию следует использовать, когда в дереве нужно хранить достаточно большие ключи, ведь патриция при поиске позволяет не проходится по всему ключу, а брать только один бит из этого ключа.

Незадолго до лабораторной работы узнал о профилировании, а в ходе лабораторной работы попрактиковался с профилировщиком grpof. Профилирование помогает собрать информацию о скорости работы функций, что, в свою очередь, можно использовать для оптимизации медленных участков кода.

Узнал об утилите valgrind, одной из функций которой является поиск утечек памяти. Эту утилиту считаю очень полезной, ведь с её помощью можно быстрее обнаружить и «заделать дыры» и во многих случаях существенно сэкономить память.

Список литературы

[1] *std::map – cppreference*
URL: `https://en.cppreference.com/w/cpp/container/map` (дата обращения: 14.04.2023).

[2] Dinesh P. Mehta, Sartaj Sahni. *Handbook of data structures and applications* — CHAPMAN & HALL/CRC, 2005. — ISBN 1-58488-435-5