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: Ю. Ю. Обыденкова
Преподаватель: А. Н. Ридли
Группа: М8О-308Б-18

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №2

Задача: 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.
Программа должна обрабатывать строки входного файла до его окончания. Необходимо реализовать следующие функции:

- Добавление в словарь пары из ключа и значения
- Удаление из словаря элемента по ключу
- Поиск элемента в словаре по ключу
- Сохранение словаря в бинарный файл
- Загрузка словаря из бинарного файла

Используемая структура данных: PATRICIA

1 Описание

Прежде чем приступить к описанию реализованной структуры, уместно рассказать о том классе деревьев, к которым она относится. Trie(название произошло от слова retireval)-деревья это структура данных, позволяющяя эффективно выполнять поиск, вставку и удаление в тех случаях, когда необходимо реализовать словарь, ключами в котором являются строки. Основная особенность таких деревьев заключается в хранении значений в нижней части дерева, то есть в листьях. В узлах, не являющихся листьями содержится набор символов алфавита, из которого состоят строки, и поставленных им в соответствие указателей на дочерние элементы. В алфавит необходимо добавить фиктивный символ, символизирующий окончание строки(и наличие её в данном дереве как ключа). В итоге, поиск элемента в дереве сводится к последовательной проверке символов переданного ключа и к дальнейшему спуску по дереву на основании значения этого символа.

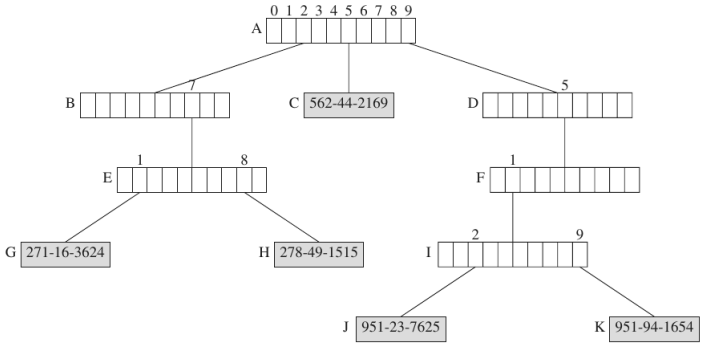

Рис. 1: Пример trie

Для trie, использующего битовое представление строк, Роберт Седжвик приводил такое определение.

> Под trie-деревом понимается бинарное дерево, имеющее ключи, связанные с каждым из его листьев, и рекурсивно определённое следующим образом. Trie-дерево, состоящее из пустого набора ключей представляет собой нулевую связь. Trie-дерево, состоящее из единственного ключа — это лист, содержащий данный ключ. И, наконец, trie-дерево, содержащее больше одного ключа — это внутренний узел, левая связь которого ссылается на trie-дерево с ключами, начинающимися с 0 разряда, а правая — на trie-дерево с ключами, начинающимися с 1 разряда, причем для конструирования поддеревьев ведущий разряд такого дерева должен быть удален.[2]

Trie может быть преобразован в сжатый trie. Сжатие производится следующим образом: если начиная с узла X до некоторого его дочернего узла Y, каждый узел имеет лишь одного потомка, то все такие узлы можно убрать из дерева, добавив их ключи в узел X(теперь в нём хранится не один символ, а строка).

Вернёмся к структуре данных, которую нужно реализовать по условию лабораторной работы. PATRICIA(Practical Algorithm To Retrieve Information Coded In Alphanumeric) — особый сжатый бинарный trie, в каждом узле которого хранится ключ(строка, возможно в битовом представлении), значение, число, представляющее собой номер бита, значение которого будет проверяться при вставке или поиске, и две ссылки на некоторые элементы дерева. Для любого узла верно, что значение номер бита для проверки у его потомков строго больше, чем у самого узла(если это не так, значит ссылка на потомка на самом деле является ссылкой на элемент, находящийся выше по дереву, а в таком случае он не является потомком нашего узла, такая ссылка называется обратной). Стоит отметить, что любой элемент дерева, кроме корня(или header'a, как его ещё называют), имеет ровно две ссылки, в то время как корень имеет только левую ссылку. Таким образом удаётся избегать однонаправленного ветвления. При вставке, удалении или поиске в одном узле необходимо сравнить всего один бит хранимого ключа и ключа, для которого совершается одно из трёх вышеприведённых действий. PATRICIA имеет сложность $O(h)$ для вставки, удаления и поиска, где h — высота дерева. Исходя из строения данной структуры, можно утверждать, что высота дерева не будет превышать битовой длины наибольшей из хранящихся в нем строк. Реализации алгоритмов вставки, удаления и.т.д. будут рассмотрены в следующем разделе.

2 Исходный код

1 Описание программы

Программа предоставляет интерфейс для работы с вышеописанной структурой данных, принимая во входной поток строки с командами. Существует 5 возможных команд.

- *+ KEY VALUE* — Добавляет в дерево пару из ключа KEY и значения VALUE, если такого элемента ещё нет в дереве
- *- KEY* — Удаляет из дерева элемент с ключом KEY, если он существует
- *KEY* — Осуществляет поиск элемента с ключом KEY в дереве
- *! Save PATH_TO_FILE* — Сохраняет словаря в файл в компактном бинарном представлении
- *! Load PATH_TO_FILE* — Загружает словарь из бинарного файла, заменяя им текущий

Само дерево реализовано как шаблонный класс, шаблонным параметром является тип хранимого значения, строки при этом представлены в виде TVector<unsigned char>(TVector — собственная реализация контейнера std::vector, которая применялась и в предыдущей лабораторной работе). Такое представление позволяет никоим образом не ограничивать длину строк, хранимых в дереве, и так же добавляет достаточно удобный интерфейс для работы со строками. Класс TPatricia имеет несколько публичных методов, позволяющих осуществлять вставку, удаление, поиск, сохранение в файл, загрузку из файла.
Рассмотрим реализации вышеописанных функций.

1.1 Поиск

Пусть в дереве мы ищем элемент с ключом *Key*. Начиная с корневого элемента дерева, будем спускаться по дереву. Проверим n-ный бит нашего ключа, где n — номер бита для проверки, сохранённый в каждом узле дерева. если этот бит равен 0, нужно спуститься в левое поддерево и продолжить поиск, в ином случае, нужно продолжить спуск в правое поддерево. Когда ссылка по которой мы перешли окажется обратной, это значит, что мы нашли место, где может хранится элемент, который мы ищем. Осталось только сравнить ключ узла и *Key*, если они совпадают, значит элемент найден. Нужно отметить, что в корне номер бита для проверки равен -1(при нумерации с нуля), так что из него нужно просто спуститься в левое поддерево.

1.2 Вставка

Стратегия для вставки элемента в дерево такова:

- Попытаться найти *Key* в дереве. Пусть *reachedKey* — ключ в узле *endNode*, где поиск элемента прекратился по причине перехода по обратной ссылке.
- Определить самую левую позицию *lBitPos*, в которой *Key* и *reachedKey* различаются.
- Создать новый узел, при этом значения бита для проверки в этом узле должно быть *lBitPos*, а ключ и значение соответственно равны ключу и значению элемента, который необходимо вставить. Вставить этот в некотором месте пути, который мы прошли при поиске ключа *Key* таким образом, чтобы не нарушалось свойство возрастания битов для проверки при спуске по дереву. Такая вставка ломает ссылку из некоторого узла p в q. Теперь эта ссылка должна вести в созданный узел.
- Если в *Key* на позиции *lBitPos* находится 1, то правая ссылка нового узла должна указывать на сам узел, а левая должна ссылаться на q. Если же на позиции *lBitPos* находится 0, то наоборот.

1.3 Удаление

Для удаления существует всего 3 случая.

- Если в дереве остался только корневой элемент и его нужно удалить, то дерево становится пустым.
- Если удаляемый элемент p содержит ссылку на самого себя, нужно найти его родителя *pParent*, а так же единственного потомка q. Ссылку из *pParent* в p нужно заменить на ссылку в q, а p просто удалить.
- Если удаляемый элемент p не содержит ссылок на себя, необходимо найти несколько узлов дерева. Нам нужен элемент q, содержащий обратную ссылку в p, также нужно найти узел r, содержащий обратную ссылку в q. Так же необходимо найти *qParent* — родителя q и запомнить, в какое поддерево q мы перешли при поиске узла r. Далее необходимо поменять местами ключи и значения элементов p и q, ссылку из r в q заменить на ссылку в узел p. Ссылку из *qParent* в q заменить на ссылку из *qParent* в то поддерево q, в которое мы переходили в процессе поиска узла r. Теперь можно удалить узел q.[4]

1.4 Сохранение в файл

Заведем в каждом узле дополнительное поле, которое будет уникальным идентификатором узла при сохранении в файл. Тогда, с помощью рекурсивного обхода в глубину можно пронумеровать вершины, заодно сохранив все вершины в TVector<Node*>. Далее, в файл записывается размер вектора узлов. После этого в файле сохраняется информация обо всех вершинах в следующем порядке: уникальный идентификатор, значение узла, размер ключа, ключ(строка), номер бита для проверки. Заметим, что если в дереве n элементов, то ссылок в нем $2n - 1$. Далее в файл записываются все ссылки дерева в формате: идентификатор узла, из которого выходит ссылка, идентификатор узла, в который идет ссылка, символ, сигнализирующий, левая или правая это ссылка.

1.5 Считывание из файла

Благодаря тем данным, которые мы сохранили в файл, из них можно быстро сконструировать дерево. Достаточно просто считать число элементов в дереве(b), создать TVector<Node*> размера n, заполнить данными и проставить ссылки.

2 Таблица функций и методов

TPatricia.h, вспомогательная функция	
bool getNthBit(const TVector<unsigned char>& lhs, size_t index)	Возвращает значения бита строки под номером index(если такого нет, возвращается false)
TPatricia.h, template<typename T> class TPatricia	
TPatricia() = default	Конструктор по умолчанию для дерева
~TPatricia()	Рекурсивный деструктор
TOptional<T> operator [] (const TVector<unsigned char>& string) const	Ищет элемент в дереве по ключу
bool Insert(TVector<unsigned char> string, T data)	Добавляет элемент в дерево, возвращает false, если такой элемент в дереве есть
bool Insert(TVector<unsigned char> string, T data)	Добавляет элемент в дерево, возвращает false, если такой элемент в дереве есть
bool Erase(const TVector<unsigned char>& string)	Удаляет элемент из дерева, если успешно удалено, возвращает true
void ScanFromFile(const char* filename)	Считывает дерево из файла и заменяет им текущее дерево
void PrintToFile(const char* filename) const	Записывает дерево в бинарном представлении в файл
void CountIds(Node* node, int& id, TVector<Node*>& nodes) const	Рекурсивно нумерует узлы уникальными идентификаторами, при этом записывая их в переданный вектор, помечена как константная потому, что не изменяет содержимое дерева, за исключением некоторых технических данных
void DeleteTree(Node* node)	Рекурсивно удаляет дерево
Node* SearchParentNode(Node* node) const	Находит родительский узел переданного узла

Node* SearchKey(const TVector<unsigned char>& string, Node** back = nullptr) const	Ищет узел по ключу. Возвращает узел, полученный после первого перехода по обратной ссылке при поиске, не гарантируя равенство переданного в функцию ключа и ключа в узле. Если задан параметр back, то после окончания работы функции в нем будет находится узел, имеющий обратную ссылку на найденный
Node* SearchParentNode(Node* node) const	Находит родительский узел переданного узла
template<typename T> class TVector	
TVector() = default;	Конструктор без параметров
TVector(size_t newSize)	Конструктор, принимающий размер нового вектора
TVector(size_t newSize, T defaultVal)	Конструктор, принимающий размер и значение по умолчанию
TVector(const TVector& other)	Конструктор копирования, копирует значения из переданного вектора
TVector(TVector&& other)	Конструктор перемещения, перемещает в создаваемый объект указатель из other, в other указатель становится nullptr
TVector()	Деструктор, освобождает выделенную память.
T& operator[] (size_t index)	Оператор доступа по индексу, возвращает data[index].
const T& operator[] (size_t index) const	Константная версия оператора доступа по индексу.
void PushBack(const T& elem)	Добавляет elem в конец вектора, при необходимости совершая реаллокацию памяти
T* begin()	Возвращает указатель на память, хранящуюся внутри вектора, название данной и трёх последующих функций не соответствует codestyle из-за того, что функции begin() и end() используются в range-based for.

T* end()	Возвращает указатель на хранящуюся память, увеличенный на размер вектора.
const T* begin() const	Константная версия метода begin().
const T* end() const	Константная верстия метода end().
TVector& operator=(const TVector& other)	Копирующий оператор присваивания, возвращает ссылку на текущий объект.
TVector& operator=(TVector&& other)	Перемещающий оператор присваивания, перемещает указатель на память из other, указатель в other становится nullptr. Возвращает ссылку на текущий объект.
void ShrinkToFit()	Сжимает кусок памяти, которым владеет вектор до его реального размера.
size_t Size() const	Возвращает размер вектора.
std::ostream& operator « (std::ostream& os, const TVector<T>& vec)	Перегрузка оператора вывода для TVector
bool operator == (const TVector<T>& lhs, const TVector<T>& rhs)	Перегрузка оператора равенства для TVector
bool operator != (const TVector<T>& lhs, const TVector<T>& rhs)	Перегрузка оператора неравенства для TVector
TOptional.h, template<typename T> class TOptional	
TOptional() = default	Конструктор по умолчанию
explicit TOptional(const T& newValue)	Конструктор от значения типа T
const T& operator* () const	Константный оператор разыменования
T& operator* ()	Оператор разыменования
operator bool() const	Оператор приведения к типу bool
main.cpp	
TVector<unsigned char> strToVec(const char* str)	Трансформирует строку в TVector<unsigned char>
int main()	Реализует интерфейс для доступа к дереву

3 Консоль

```
julia@julia21:~$ ls
main.cpp  Makefile  solution  TOptional.h  TPatricia.cpp  TPatricia.h  TVector.h
julia@julia21:~$ cat Makefile
CXX = g++
CXXFLAGS = -std=c++11 -O2 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic
FILES = main.cpp TPatricia.cpp
NAME = solution

all: example

example:
$(CXX) $(CXXFLAGS) -o $(NAME) $(FILES)
clean:
rm -f *.o $(NAME)
julia@julia21:~$ make
g++ -std=c++11 -O2 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic -o solution main.cpp TPatricia.cpp
julia@julia21:~$ nano input
julia@julia21:~$ cat input
+ a 10
a
+ abc 654
ABC
+ ttt 123
+ flgfk 677667
TtT
! Save file
-ttt
-flgfk
ttt
-abc
abc
! Load file
abc
ttt
julia@julia21:~$ ./solution <input
OK
OK: 10
```

```
OK
OK: 654
OK
OK
OK: 123
OK
OK
OK
NoSuchWord
OK
NoSuchWord
OK
OK: 654
OK: 123
julia@julia21:~$ hexdump file
0000000 0004 0000 0000 0000 0000 0000 0000 0000
0000010 000a 0000 0000 0000 0001 0000 0000 0000
0000020 ff61 ffff ffff ffff 01ff 0000 0000 0000
0000030 7b00 0000 0000 0000 0300 0000 0000 0000
0000040 7400 7474 0003 0000 0000 0000 0002 0000
0000050 0000 0000 5723 000a 0000 0000 0005 0000
0000060 0000 0000 6c66 6667 056b 0000 0000 0000
0000070 0300 0000 0000 0000 8e00 0002 0000 0000
0000080 0300 0000 0000 0000 6100 6362 0009 0000
0000090 0000 0000 0000 0000 0000 0000 0001 0000
00000a0 0000 0000 0130 0000 0000 0000 0100 0000
00000b0 0000 0000 3100 0001 0000 0000 0000 0002
00000c0 0000 0000 0000 0230 0000 0000 0000 0200
00000d0 0000 0000 0000 3100 0002 0000 0000 0000
00000e0 0003 0000 0000 0000 0330 0000 0000 0000
00000f0 0300 0000 0000 0000 3100 0003 0000 0000
0000100 0000 0000 0000 0000 0000 0030
000010b
```

4 Тест производительности

Напишем простую программу для генерации тестов и замера времени выполнения данных тестов. Она генерирует заданное количество пар из строк случайной длины от 1 до 256, содержащих случайные символы латинского алфавита, и чисел типа unsigned long long.

```
1  #include <ctime>
2  #include <random>
3  #include <map>
4  #include <limits>
5  #include <tuple>
6  #include <string>
7  #include <cstdlib>
8  #include <iostream>
9  #include <chrono>
10 #include <iomanip>
11
12 #include "TPatricia.h"
13 #include "profile.h"
14
15 using namespace std;
16
17 default_random_engine rng;
18
19 uint64_t get_number(uint64_t min = 0,uint64_t max = numeric_limits<unsigned long long
       >::max()) {
20     uniform_int_distribution<unsigned long long> dist_ab(min, max);
21     return dist_ab(rng);
22 }
23
24 string get_string() {
25     size_t string_size = get_number(1,256);
26     string string;
27     string.resize(string_size);
28     for (size_t i = 0; i < string_size; ++i) {
29         string[i] = 'a' + get_number(0,25);
30     }
31     return string;
32 }
33
34 TVector<unsigned char> convert_string(const string& s) {
35     TVector<unsigned char> vector(s.size());
36     for (size_t i = 0; i < s.size(); ++i) {
37         vector[i] = s[i];
38     }
39     return vector;
40 }
41
```

```
42 string convert_string(TVector<unsigned char>& vec) {
43     string str;
44     str.resize(vec.Size());
45     for (size_t i = 0; i < str.size(); ++i) {
46         str[i] = vec[i];
47     }
48     return str;
49 }
50
51 int main() {
52     rng.seed(std::chrono::system_clock::now().time_since_epoch().count());
53     size_t count;
54     cin >> count;
55     vector<pair<TVector<unsigned char>, unsigned long long>> test_data(count);
56     fstream test_data_log("file.test", ios::out);
57     {
58     LOG_DURATION("Generate")
59         for (size_t i = 0; i < count; ++i) {
60             test_data[i].first = convert_string(get_string());
61             test_data[i].second = get_number(0, numeric_limits<unsigned long long>::max
                    ());
62             test_data_log << test_data[i].first << " " << test_data[i].second << "\n";
63         }
64     }
65
66     cout << "Patricia test\n";
67     TPatricia<unsigned long long> patricia;
68     {
69         LOG_DURATION("Insert time")
70         for (size_t i = 0; i < count; ++i) {
71             patricia.Insert(test_data[i].first, test_data[i].second);
72         }
73     }
74
75     {
76          LOG_DURATION("Save to file")
77          patricia.PrintToFile("file");
78     }
79
80     std::shuffle(test_data.begin(), test_data.end(), rng);
81     {
82         LOG_DURATION("Write and erase one element")
83         for (size_t i = 0; i < count; ++i) {
84             patricia.Erase(test_data[i].first);
85             patricia.Insert(test_data[i].first, test_data[i].second);
86         }
87     }
88     {
89         LOG_DURATION("Erase time")
```

```
90          for (size_t i = 0; i < count; ++i) {
91              patricia.Erase(test_data[i].first);
92          }
93      }
94      {
95          LOG_DURATION("Erase empty time")
96          for (size_t i = 0; i < count; ++i) {
97              patricia.Erase(test_data[i].first);
98          }
99      }
100
101     {
102         LOG_DURATION("Scan from file")
103         patricia.ScanFromFile("file");
104     }
105
106
107     vector<pair<string, unsigned long long>> map_test(test_data.size());
108     for (size_t i = 0; i < map_test.size(); ++i) {
109         map_test[i] = {convert_string(test_data[i].first), test_data[i].second};
110     }
111     cout << "std::map test\n";
112     map<string, unsigned long long> map;
113     {
114         LOG_DURATION("Insert time")
115         for (size_t i = 0; i < count; ++i) {
116             map[map_test[i].first] = map_test[i].second;
117         }
118     }
119
120     std::shuffle(test_data.begin(), test_data.end(), rng);
121
122     {
123         LOG_DURATION("Write and erase one element")
124         for (size_t i = 0; i < count; ++i) {
125             map.erase(map_test[i].first);
126             map[map_test[i].first] = map_test[i].second;
127         }
128     }
129     {
130         LOG_DURATION("Erase time")
131         for (size_t i = 0; i < count; ++i) {
132             map.erase(map_test[i].first);
133         }
134     }
135     {
136         LOG_DURATION("Erase empty time")
137         for (size_t i = 0; i < count; ++i) {
138             patricia.Erase(test_data[i].first);
```

```
139         }
140     }
141
142     return 0;
143 }
```

1 Протокол тестирования производительности

```
julia@julia21:~$ ./test
10000
Generate: 183 ms
Patricia test
Insert time: 16 ms
Save to file: 9 ms
Write and erase one element: 46 ms
Erase time: 25 ms
Erase empty time: 0 ms
Scan from file: 37 ms
std::map test
Insert time: 24 ms
Write and erase one element: 50 ms
Erase time: 24 ms
Erase empty time: 24 ms
julia@julia21:~$ ./test
100000
Generate: 1533 ms
Patricia test
Insert time: 235 ms
Save to file: 189 ms
Write and erase one element: 523 ms
Erase time: 308 ms
Erase empty time: 0 ms
Scan from file: 377 ms
std::map test
Insert time: 307 ms
Write and erase one element: 602 ms
Erase time: 310 ms
Erase empty time: 278 ms
julia@julia21:~$ ./test
1000000
```

```
Generate: 15157 ms
Patricia test
Insert time: 3345 ms
Save to file: 16384 ms
Write and erase one element: 6522 ms
Erase time: 3895 ms
Erase empty time: 9 ms
Scan from file: 3870 ms
std::map test
Insert time: 3941 ms
Write and erase one element: 7646 ms
Erase time: 4156 ms
Erase empty time: 3514 ms
```

Продемонстрировано преимущество в скорости PATRICIA по сравнению с std::map. Это неудивительно, ведь PATRICIA не сравнивает ключи целиком при поиске.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я научилась эффективнее использовать функции заголовочного файла chrono. Работа со временем как с безразмерной величиной может приводить к недоразумениям и ошибкам конвертации временных единиц измерения. Для избежания таких ошибок и предусмотрена библиотека chrono. Библиотека реализует следующие концепции: интервалы времени – duration; моменты времени – time_point; таймеры – clock. Также я узнала о строении и тонкостях реализации структуры данных PATRICIA — это префиксное дерево, в котором префиксы бинарны — то есть, каждый узел-ключ хранит информацию об одном бите. Само дерево реализовано как шаблонный класс, шаблонным параметром является тип хранимого значения. Такое представление позволяет никоим образом не ограничивать длину строк, хранимых в дереве, и так же добавляет достаточно удобный интерфейс для работы со строками. PATRICIA работает гораздо эффективнее других подобных структур на символьных ключах большой длины, в чем я лично убедилась, написав бенчмарк.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] Роберт Седжвик. *Фундаментальные алгоритмы, 3-я редакция.* — Издательский дом «ДиаСофт», 2001. Перевод с английского: С. Н. Козлов, Ю. Н. Артеменко, О. А. Шадрин — 688 с. (ISBN 966-793-89-5(рус.))

[3] *Лекции по курсу «Дискретный анализ» МАИ.*

[4] Dinesh P. Mehta, Sartaj Sahni. *Handbook of Data Structures and Applications, 2nd Edition.* — Chapman and Hall/CRC, 2018. (ISBN 9781498701853).