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: А. А. Кабанов
Преподаватель: С. А. Михайлова
Группа: М8О-301Б-22

Дата:
Оценка:
Подпись:

Москва, 2024

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 264 - 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, отсутсвие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Вариант структуры данных: PATRICIA

1 Описание

PATRICIA (Practical Algorithm To Retrieve Information In Alphanumeric) trie позволяет эффективно решать следующую задачу - проверять нахождения объектов в дереве только с минимальным количеством проверок (в отличие от compact trie, при поиске объекта в котором необходимо сравнить все биты информации).

PATRICIA имеет следующие свойства:

1. Корень - header, у него только левый потомок;
2. Каждый узел в дереве - слово/объект;
3. У каждого узла есть номер проверяемого бита (бит header равен 0 или неопределен);
4. Есть левая и правая ссылки (прямые или обратные). Прямая сслыка гарантирует, что значение проверяемого бита увеличится. Обратные ссылки ведут либо на ту же вершину, либо на элемент сверху в дереве (на пути от header до текущего);
5. В каждую вершину ведется ровно одна обратная ссылка.

Поиск осуществляется так: идем из хэдера в левый потомок. Если бит ключа равен 0, идем по левой ссылке. Иначе - по правой ссылке. Делаем до момента перехода по обратной ссылке. Проверяем ключ, если он совпадает, мы нашли ключ.

Вставка осуществляется через поиск, вычисление первого несовпавшего бита, создание вершины (и корректное распределение левой и правой ссылки создаваемой вершины) и разбитие ссылки (не наршуая свойство возрастания значения проверяемого бита). Если дерево изначально пустое, вставим хэдер: вершину с одной левой обратной ссылкой на себя.

Удаление имеет три случая:

1. Если есть только хэдер или это ребенок, удаляем его.
2. Если удаляемый элемент имеет только одну на другой элемент (одну прямую ссылку), берем ссылку родителя на него и направляем на ребенка.
3. Если удаляемый элемент X имеет две ссылки на другие элементы, заменим его ключ на ключ Q - элемента, ссылающегося на X и удаляем Q. P - элемент, ссылающийся на Q, теперь P ссылается на заменимый элемент.

2 Исходный код

Создадим структуры PATRICIA trie, вершины и методы для них, а также осуществим сериализацию и десериализацию дерева.

```
#include <bits/stdc++.h>

using namespace std;

struct Node;
struct Trie;

using bit_t = int32_t;
using stype = string;
using vtype = uint64_t;
using key = pair<stype, vtype>;
using nptr = Node*;
using tptr = Trie*;

const bit_t MAX_LEN_KEY = 257;
const bit_t BIT_COUNT = 5;

struct Node {
  key k;
  bit_t bit;
  Node* l;
  Node* r;
  vtype num;

  Node() {
    k.first = nullptr;
    k.second = -1;
    bit = -1;
    l = this;
    r = this;
  }

  Node(stype word, vtype value, bit_t b) : l(this), r(this) {
    k.first = word;
    k.second = value;
    bit = b;
  }

  Node(stype word, vtype value, bit_t b, nptr left, nptr right) {
    k.first = word;
    k.second = value;
    bit = b;
    l = left;
    r = right;
```

```
45    }
46
47    ~Node() {
48      l = nullptr;
49      r = nullptr;
50      k.first = "\0";
51    };
52  };
53
54  bool get_bit(stype& key, bit_t bit_id) {
55    bit_t byte_n = bit_id / 8;
56    bit_t bit_n = bit_id % 8;
57    bit_t symbol;
58    if (bit_id < 0 || byte_n >= key.size()) {
59      return false;
60    } else {
61      symbol = key[byte_n];
62    }
63    return (symbol & (128 >> bit_n)) != 0;
64  }
65
66  bit_t leftmost_bit(stype& key1, stype& key2) {
67    bit_t i = 0;
68    bit_t max_sz = max(key1.size(), key2.size());
69    while (key1[i] == key2[i]) {
70      i++;
71      if (i == max_sz) {
72        return i * 8;
73      }
74    }
75    i = i * 8;
76    while (get_bit(key1, i) == get_bit(key2, i)) {
77      i++;
78    }
79    return i;
80  }
81
82  struct Trie {
83    nptr header;
84
85    Trie() { header = nullptr; }
86
87    ~Trie() { destructR(header); }
88
89    void destructR(nptr n) {
90      if (n == nullptr) {
91        return;
92      } else if (n->l->bit > n->bit) {
93        destructR(n->l);
```

```
94      } else if (n->r != nullptr && n->r->bit > n->bit) {
95        destructR(n->r);
96      }
97    }
98
99    nptr search(stype& k) {
100     if (header == nullptr) {
101       cout << "NoSuchWord\n";
102       return nullptr;
103     }
104     nptr p = this->header;
105     nptr c = this->header->l;
106     while (p->bit < c->bit) {
107       p = c;
108       c = get_bit(k, c->bit) ? c->r : c->l;
109     }
110     if (c->k.first.compare(k) == 0) {
111       cout << "OK: " << c->k.second << '\n';
112     } else {
113       cout << "NoSuchWord\n";
114       return nullptr;
115     }
116     return c;
117   }
118
119   nptr insert(stype k, vtype d) {
120     if (this->header == nullptr) {
121       auto newNode = new Node(k, d, 0, nullptr, nullptr);
122       newNode->l = newNode;
123       this->header = newNode;
124       cout << "OK\n";
125       return header;
126     } else {
127       nptr p;
128       nptr t;
129       nptr x;
130       p = this->header;
131       t = this->header->l;
132       while ((p->bit < t->bit)) {
133         p = t;
134         t = (get_bit(k, t->bit) ? t->r : t->l);
135       }
136       if (t->k.first.compare(k) == 0) {
137         cout << "Exist\n";
138         return nullptr;
139       }
140       bit_t i = leftmost_bit(k, t->k.first);
141       // cout << "!: " << i << "\n";
142       p = this->header;
```

```
143       x = this->header->l;
144       auto newNode = new Node(k, d, i, nullptr, nullptr);
145       while ((p->bit < x->bit) && (x->bit < newNode->bit)) {
146         p = x;
147         x = (get_bit(k, x->bit) ? x->r : x->l);
148       }
149       newNode->l = (get_bit(k, i) ? x : newNode);
150       newNode->r = (get_bit(k, i) ? newNode : x);
151       if (x == p->r) {
152         p->r = newNode;
153       } else {
154         p->l = newNode;
155       }
156       cout << "OK\n";
157       return t;
158     }
159   }
160
161   bool remove(stype& k) {
162     if (header == nullptr) {
163       cout << "NoSuchWord\n";
164       return false;
165     }
166     if (header->l == header) { // header is an only node
167       if (header->k.first.compare(k) == 0) {
168         delete header;
169         header = nullptr;
170         cout << "OK\n";
171         return true;
172       } else {
173         cout << "NoSuchWord\n";
174         return false;
175       }
176     }
177     nptr prePrev = nullptr, parOfPrev = nullptr, previous = header,
178     current = header->l;
179     while (previous->bit < current->bit) {
180       prePrev = previous;
181       previous = current;
182       current = get_bit(k, current->bit) ? current->r : current->l;
183     }
184     if ((current->k.first).compare(k) != 0) {
185       std::cout << "NoSuchWord\n";
186       return false;
187     }
188     if (current == previous) {
189       if (prePrev->l == current) {
190         prePrev->l = (current->l == current ? current->r : current->l);
191       } else {
```

```
192         prePrev->r = (current->l == current ? current->r : current->l);
193       }
194       delete current;
195       std::cout << "OK\n";
196       return true;
197     }
198     nptr p = current, q = previous, qPar = prePrev, r = nullptr;
199     current = header->l;
200     r = header;
201     while (r->bit < current->bit) {
202       r = current;
203       current = get_bit(q->k.first, current->bit) ? current->r : current->l;
204     }
205     bool flag = get_bit(r->k.first, q->bit);
206     if (r->r == q) {
207       r->r = p;
208     } else {
209       r->l = p;
210     }
211     if (qPar->r == q) {
212       qPar->r = flag ? q->r : q->l;
213     } else {
214       qPar->l = flag ? q->r : q->l;
215     }
216     p->k.first = q->k.first;
217     p->k.second = q->k.second;
218     delete q;
219     std::cout << "OK\n";
220     return true;
221   }
222
223   int save(stype& filename) {
224     // look solution
225   }
226   int saveR(nptr node, std::ofstream& file) {
227     // look solution
228   }
229   int load(string filename) {
230     // look solution
231   }
232 };
233
234 void to_lowercase(stype& s) {
235   for (size_t i = 0; s[i] != '\0'; ++i) {
236     if (s[i] >= 'A' && s[i] <= 'Z') {
237       s[i] = s[i] - 'A' + 'a';
238     }
239   }
240 }
```

3 Консоль

```
[anton@home lab2-patricia]$ make
g++ -O2 -lm -fno-stack-limit -std=c++20 -x c++ solution.cpp -o executable
[anton@home lab2-patricia]$ cat test/test2.in
a
+ a 1
A
a
- a
A
+ aa 2
aa
+ b 3
b
- aa
b
[anton@home lab2-patricia]$ make run <test/test2.in
./executable
NoSuchWord
OK
OK: 1
OK: 1
OK
NoSuchWord
OK
OK: 2
OK
OK: 3
OK
OK: 3
```

4 Тест производительности

Тесты производительности представляют из себя следующее: сравнение времени работы PATRICIA и map, основанный на красно-черном дереве. Входной файл - вставка от 1000 до 10000000 элементов

```
PATRICIA:
elements 1000 0.0543351173401
elements 10000 0.246738894903
elements 100000 0.57096939
elements 1000000 15.2940719128
elements 10000000 30.297783852
std::map:
elements 1000 0.00444197654724
elements 10000 0.0231420993805
elements 100000 0.57096939
elements 1000000 15.38923597336
elements 10000000 48.7939031124
```

Как видно, PATRICIA работает так же эффективно, как и std::map.

5 Выводы

Я реализовал дерево PATRICIA, которая является самой сложной в курсе, по моему мнению. Я потратил много времени на отладку операции вставки и удаления из-за сложностей переноса теоретических знаний в код.

Концепция PATRICIA trie очень хороша идейно. Проверка только некоторых необходимых битов для нахождения ключа экономит много времени, в отличие от compact trie.

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: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 978-5-8459-1794-2 (рус.))