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: К. М. Воронов
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19

Дата:
Оценка:
Подпись:

Москва, 2020

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. 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 используемой структуры данных: PATRICIA.

1 Описание

Требуется написать реализацию словаря (ассоциативного массива) с помощью структуры данных PATRICIA. Помимо простого добавления, поиска и удаления требуется реализовать запись на диск и чтение с диска всего словаря.

Патриция - это структура данных в виде дерева с особым строением.

Как говорилось в [1], её основная идея заключается в том, что мы будем идти не по всем битам подряд, а по определённым, то есть некоторыми "скачками".

Номера сравниваемых битов мы будем хранить непосредственно в узлах Патриции.

Патриция начинается с так называемого хедера, который не имеет правой ссылки, значение сравниваемого бита в нём равно -1. К каждому элементу Патриции существует обратная ссылка для того, чтобы этот элемент найти. Так как при проходе дерева в глубину номера сравниваемых битов увеличиваются, мы можем легко определить, когда перешли по обратной ссылке - номер бита соответствующего узла будет меньше или равен номеру бита предыдущего.

Добавление элемента в Патрицию осуществляется следующим образом:
1. Если дерево пустое - добавляем хедер и указываем его левую ссылку на него самого.
2. Если не пустое, то запускаем поиск, пока не перейдём по обратной ссылке. Если бит нашего слова, номер которого записан, в ячейке равен единице, то идём налево, иначе - направо. Хедер пропускаем, то есть идём по его левой ссылке. Как только мы перейдём по обратной ссылке, можно приступать непосредственно к добавлению. Если бит добавляемого элемента, номер которого записан в этом узле, равен 0, то мы "разрываем"левую связь и вставлем его туда, иначе - правую. Номер бита в новом узле будет равен первому отличающемуся биту между добавляемым словом и тем, что находится в узле, от которого мы "разрываем"ссылку. В зависимости от его значения, мы расставляем ссылки от нового элемента.

Удаление в Патриции сложнее. Для начала мы также запускаем поиск удаляемого элемента. В случае успеха надо рассмотреть 4 случая:
1. Если в дереве присутствует только хедер, мы просто удаляем его.
2. Если правая или левая ссылка удаляемого элемента указывает на него самого, то вторая указывает на другой элемент, назовём его A. Также существует предок нашего элемента, пусть будет X, который одной из ссылок указывает на него. Мы просто перенаправляем эту ссылку от X к A и удаляем искомый элемент.
3. Если у удаляемого элемента нет обратных ссылок (назовём его X), тогда всегда есть следующие элементы: Q - который ссылается обратной на X, R - который ссылается обратной на Q, N - на который ссылается Q другой ссылкой и M - предок Q. Мы копируем в элемент X значение Q и удаляем вместо него старый Q. Для этого

нам надо убрать все указывающие на него ссылки. Ссылка от R теперь вместо Q, должна указывать на X, от M - на N. Теперь спокойно удаляем старый Q.
4. Если надо удалить хедер, и он не единственый элемент в дереве, тогда копируем в него значение из элемента X, который ссылается на хедер обратной ссылкой и теперь удаляем его. Если у X есть ссылки, которые указывают на него самого, то действуем, как в случае 2. Если же нет, то находим элемент A - предка X, элемент B - который ссылается на X обратной ссылкой, и элемент C, на который указывает второй ссылкой X. Соединяем A с C и B с хедером, после чего удаляем X.

Чтобы сохранить дерево на диск, надо создать массив указателей на узлы Патриции (пронумеровать узлы и заполнить его), записать в файл количество узлов и их данные. При считывании надо также создать массив указателей на узлы Патриции (предварительно их создав), заполнить его и соединить.

2 Исходный код

Для хранения слов я создал структуру TString с двумя полями: массивом символов S (размера 256) и Capacity, которое будет хранить количество ненулевых элементов в массиве. Также перегрузил операторы ==, !=, [ ] и =. Реализовал методы Clear (очистка массива), Sizev (количество ненулевых элементов в массиве), Push (добавление элемента в массив).

```
struct TString {
    char S[TL] = {0};
    int Capacity = 0;

    char& operator[](int i) {
        return S[i];
    }

     void Clear() {
        memset(S,0,TL);
        Capacity = 0;
    }

    void Push(char z) {
        S[Capacity] = z;
        Capacity += 1;
    }

    TString& operator = (const TString &vec) {
        memcpy(S, vec.S, vec.Capacity);
        Capacity = vec.Capacity;
        return *this;
    }

    bool operator == (const TString &a) {
        if (Capacity != a.Capacity){
            return false;
        }

        for (int i = 0; i < a.Capacity; ++i) {
            if(a.S[i] != S[i]) {
                return false;
            }
        }
        return true;
    }

    bool operator != (const TString &a) {
        if (Capacity != a.Capacity){
            return true;
        }
```

```
42
43          for (int i = 0; i < a.Capacity; ++i) {
44              if (a.S[i] != S[i]) {
45                  return true;
46              }
47          }
48          return false;
49      }
50
51       int Sizev() {
52          return Capacity;
53      }
54  };
```

Создадим структуру TPatricia (узел Патриции) с полями Id (понадобится для сохранения (загрузки) дерева на диск (с диска)), Nbit (номер бита), Value (хранимое число), TString (хранимое слово), TPatricia *Left (ссылка на левого), TPatricia *Right (ссылка на правого). Добавим туда деструктор.

```
1   struct TPatricia {
2
3       int Id = -1;
4       int Nbit;
5       unsigned long long Value;
6       TString Pv;
7       TPatricia *Left = nullptr;
8       TPatricia *Right = nullptr;
9
10      ~TPatricia() {
11          if ((Left != nullptr) && (Left->Nbit > Nbit)) {
12              delete Left;
13          }
14          if ((Right != nullptr) && (Right->Nbit > Nbit)) {
15              delete Right;
16          }
17      }
18
19  };
```

Для реализации записи и чтения дерева на диск/с диска я использую структуру TPointers с полем TPatricia* Patr.

```
1   struct TPointers {
2       TPatricia *Patr;
3   };
```

Считывание данных происходит через TString.

Функции для работы с Патрицией

lab2.cpp	
int Searchnumber (TString &pv, TString &t)	Номер первого отличающегося бита двух слов
bool Number (TString &t, int bit)	Значение бита на указанной позиции.
void Search(TPatricia *&p, TString &t)	Поиск элемента в дереве.
void Add(TPatricia *&p, unsigned long long value, TString &t)	Добавление элемента в дерево.
void Patricia_delete(TPatricia *&p, TString &t)	Удаление элемента из дерева.
void Numbers$_P atricia(TPatricia * p, TPointers * ptr, int\&index)$	Заполнение массива указателей на узлы Патриции, пронумеровка узлов.

3 Консоль

```
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab2$ cat test
+ a 1
+ b 2
a
c
! Save a
-a
a
! Load a
a
-a
-b
b
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab2$ cat test | ./solution
OK
OK
OK: 1
NoSuchWord
OK
OK
NoSuchWord
OK
OK: 1
OK
OK
NoSuchWord
```

4 Тест производительности

Реализованная структура данных сравнивается с std::map, в котором ключи представлены TString, а значения unsigned long long.

Для вставки в std::map используется метод insert(TString), для удаления erase(TString), а для поиска find(TString). Запись и чтение из файла не реализовано в std::map, поэтому будем сравнивать только операции вставки, удаления и поиска. Тесты состоят из 10^3, 10^5 и 10^6 строк.

```
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab2$ cat test1.txt | ./solution
Patricia: 1ms
map: 1ms
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab2$ cat test2.txt | ./solution
Patricia: 209ms
map: 353ms
kirill@kirill-VirtualBox:~/Рабочий стол/DA/lab2$ cat test3.txt | ./solution
Patricia: 3138ms
map: 4887ms
```

Видно, что Patricia работает быстрее, так как в отличие от std::map там сравниваются не строки, а биты.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я вспомнил, как писать деревья, познакомился с очень интересными структурами данных, в частности с Патрицией, и изучил их работу. Также научился пользоваться Valgrind и записывать/считывать дерево на диск/с диска. Вспомнил про хеш-функции и их недостатки.

Список литературы

[1] Лекции Н.К. Макарова, Московский авиационный институт.

[2] Mehta D.P., Sahni S. *Handbook of Data Structures and Applications* (дата обращения 10.30.2020).