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: Э. Л. Носов
Преподаватель: А. А. Кухтичев
Группа: М8О-307Б
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 $2^{64} - 1$..

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: PATRICIA.

1 Описание

Требуется написать реализацию структуры данных PATRICIA.

Основная идея заключается в том, чтобы при поиске одинаковых строк не сравнивать все строки целиком, а сравнивать отдельные биты, как бы накладывая на каждое слово маску.

Структура дерева такова, что в каждом узле хранится пара «ключ-значение»; номер бита, который необходимо проверить и указатели на левый и правый узел, причем если указатель ведет к узлу, номер бита которого больше текущего, то такой указатель называют «прямой ссылкой», в противном случае его называют «обратной ссылкой».

Начальное состояние непустого дерева: имеется корневая вершина, один из указателей которой указывает на саму себя, причем хранящийся номер бита должен быть «левее самого левого».

При переходах из узла в узел необходимо проверить указанный в текущем узле номер бита для искомого значения, после чего, если бит нулевой, осуществить переход по левому указателю, иначе - по правому.

При вставке мы осуществляем переходы до тех пор, пока не встретим обратную ссылку. Встретив обратную ссылку необходимо выяснить номер первого бита, отличающегося у искомого ключа и у ключа, на который указывает обратная ссылка. После этого необходимо найти место для вставки в дереве такое чтобы образующаяся последовательность номеров битов была возрастающей, для чего необходимо либо разорвать текущую обратную ссылку, либо при откате назад вставить нужный элемент после первого элемента, чей номер бита будет меньше номера бита вставляемого элемента. Один из указателей при этом должен указывать на самого себя, а другой продолжать разорванную связь, какая именно - определяется на основании правила перехода.

При удалении необходимо найти удаляемый элемент, если одной из ссылок он ссылается на себя, то его можно просто удалить. Иначе необходимо найти также элемент, ссылающийся на него обратной ссылкой, его родителя и потомка, а также элемент ссылающийся на ссылающийся. Тогда ссылающийся на ссылающийся должен ссылаться на удаляемый элемент, значение удаляемого элемента заменяется значением ссылающегося, а родитель ссылающегося начинает ссылаться на его потомка.

Поиск выполняется посредством переходов до первой обратной ссылки.

При сохранении в файл сначала записывается заголовок, после чего дерево обходится в глубину в порядке Корень-Левое поддерево-Правое поддерево, производится запись необходимых данных узла, после чего записывается информация о том, необходимо ли обходить его поддеревья.

При загрузке дерева из файла выполняется проверка заголовка, если заголовок совпадает, то имеющееся дерево отчищается и на основании файла выстраивается новое.

2 Исходный код

PATRICIA.hpp	
int AddNode(KeyType & key, ValueType value)	Public функция для добавления новой пары «ключ-значение» с ключом типа KeyType и значением типа ValueType в дерево, обертка над InsertNode.
int DeleteNode(KeyType & key)	Public функция для удаления пары «ключ-значение» из дерева по ключу типа KeyType.
ValueType* FindNode(KeyType & key)	Public функция для поиска элемента в дереве по ключу типа KeyType, обертка над NodeSearch.
int SaveInFile(KeyType name)	Public функция для сохранения дерева в файл с именем name.
int LoadFromFile(KeyType name)	Public функция для загрузки дерева из файла с именем name.
void Clear()	Public функция для очистки дерева, является оберткой над RemoveTree.
int SaveTree(TNode * node, FILE * f)	Рекурсивная private функция для записи узла *node в файл *f.
TNode* LoadTree(TNode * node, bool isRight, FILE * f)	Рекурсивная private функция для чтения узла *node из файла *f.
void RemoveTree(TNode* node)	Рекурсивная private функция для очистки поддеревьев узла *node.
bool GetBit(int BitNumber, int ArraySize, KeyType & Array)	Private функция для просмотра BitNumber бита в массиве Array размера ArraySize.
int FindBitDifference(TNode &Recieving, TNode &Incoming)	Private функция для нахождения номера бита,отличающегося у ключей Incoming и Recieving узлов.
int InsertNode(TNode &Recieving, TNode &Incoming)	Рекурсивная private функция для вставки элемента из Incoming.
ValueType* NodeSearch(TNode & node, KeyType & key, int previousBitNumber)	Рекурсивная private функция для поиска элемента с ключом key.
string.hpp	
ValueType* cstr()	Функция, возвращающая указатель на массив символа, хранящийся в строке.
void clear()	Функция, отчищающая строку.
int Size()	Функция, возвращающая размер строки.
void ToLower()	Функция, переводящая все символы в нижний регистр.
void Reallocate(int NewCapacity)	Функция реаллокации на размер NewCapacity
void UpdateLength()	Функция, для обновления длины строки.
bool StringsEqual(const char* a, const char* b)	Функция сравнения константных массивов символов.

Реализация методов, объявленных в файле PATRICIA.hpp находится в файле PATRICIA.cpp.

```
class TPatricia {

    public:

        static const char FILE_HEADER [];
        static const int SIZE_OF_HEADER;

        using KeyType = TString;
        using ValueType = unsigned long long;

        TPatricia();
        void Clear();
        int AddNode(KeyType &, ValueType);

        int DeleteNode(KeyType&);

        ValueType* FindNode(KeyType&);

        int SaveInFile(KeyType);

        int LoadFromFile(KeyType);

        ~TPatricia();
    private:

        struct TNode{
            KeyType Key;
            ValueType Value;
            int BitNumber;
            TNode * Left;
            TNode * Right;

            TNode();

            ~TNode();

            TNode& operator = (const TNode& second);
        };

        TNode *Root;
        unsigned long long TreeSize;

        void SaveTree(TNode *, FILE *);

        TNode* LoadTree(TNode *, bool, FILE *);

        void RemoveTree(TNode*);

        bool GetBit(int, int, KeyType&);

        int FindBitDifference(TNode &, TNode &);

        int InsertNode(TNode &, TNode &);

        ValueType* NodeSearch(TNode &, KeyType&, int);

};

class TString{
    public:
    using ValueType = char;

```

```
63
64      TString();
65      TString(int arg_capacity);
66      TString(const TString &other);
67      TString(ValueType* &&other, int Size);
68
69      TString(const ValueType* other, int Size) ;
70      void PushBack(ValueType new_elem);
71
72      ValueType* cstr();
73
74      void clear();
75
76      ~TString();
77
78      int Size();
79
80      ValueType& operator [](int index);
81
82      bool operator == (const TString& str2) const;
83
84      bool operator == (const ValueType * str2) const;
85
86      TString& operator = (const TString& other);
87      TString& operator = (ValueType* &&other);
88
89      TString& operator = (const ValueType* other);
90      void ToLower();
91      friend std::ostream& operator<< (std::ostream &out, const TString &str);
92      friend std::istream& operator>> (std::istream &in, TString &str);
93
94      private:
95      ValueType* str;
96      int length;
97      int capacity;
98
99      void Reallocate(int NewCapacity);
100
101     void Updatelength();
102
103 };
104
105
106 std::ostream& operator<< (std::ostream &out, const TString &str);
107
108 std::istream& operator >> (std::istream& in, TString& str);
```

3 Консоль

```
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ ./a.out
+ word 32
OK
+ world 45
OK
word
OK: 32
world
OK: 45
- word
OK
- world
OK
word
NoSuchWord
world
NoSuchWord
```

4 Тест производительности

Для теста производительности была использована исходная программа, а так же написанная для этого программа с использованием средств STL.

```
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ g++ -g lab2.cpp -o lab2
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ g++ -g test2.cpp -o test2
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ python3 tests.py test 1000
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ time ./lab2 < test
real    0m0.012s
user    0m0.005s
sys     0m0.000s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ time ./test2 < test
real    0m0.015s
user    0m0.006s
sys     0m0.000s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ python3 tests.py test 100000
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ time ./lab2 < test
real    0m0.863s
user    0m0.303s
sys     0m0.094s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ time ./test2 < test
real    0m0.894s
user    0m0.373s
sys     0m0.088s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ python3 tests.py test 1000000
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ time ./lab2 < test
real    0m8.794s
user    0m3.995s
sys     0m0.512s
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/DA/lab2$ time ./test2 < test
real    0m9.550s
user    0m4.872s
sys     0m0.556s
```

Как видно на тестах размером 1000, 100000 и 1000000 запросов (где 0,4 запросов связаны с добавлением, 0,4 связаны с удалением и 0,2 связаны с поиском) реализация с использованием средств стандартной библиотеки (std::map, std::string) оказалась незначительно медленнее моей реализации структуры данных PATRICIA, что, вероятно, связано, с одной стороный с тем, что std::map используют в качестве своей основы красно-черное дерево поиска, являющееся сбалансированным, но при этом строки постоянно сравниваются между собой, а PATRICIA сбалансированной в общем случае не является, но при этом имеет много меньшее количество сравнений.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я научился обрабатывать исключения, в частности исключения связанные с нехваткой памяти, на практике применил основы объектно-ориентированного программирования при создании собственных классов, собственными руками реализовал структуру данных PATRICIA.