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: М. А. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б
Дата: 27 ноября 2020 г.
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.

Различия вариантов заключаются только в используемых структурах данных: **PATRICIA.**

1 Описание

Патриция - специальная структура данных, необходимая для хранения ключей (чаще всего - слов). Узлы дерева содержат в себе пару ссылок на узлы (или на себя, или на «младшие», а то и «старшие» узлы), ключ, значение и номер бита, который мы будем сравнивать при проходе через этот узел. Как не трудно догадаться, все операции основаны на сравнении определённых битов, так что временная сложность операций поиска, добавления и удаления элемента из дерева оценивается как O(k), где k — длина обрабатываемого элемента. Очевидно, что время работы не зависит от количества элементов в дереве. Данная реализация имеет одну закономерность - узлы дерева отсортированы (по мере прохождения вглубь) по возрастанию по номеру бита, который они хранят. Это позволит нам определять, является ли ссылка указателем «наверх» без использования специальных меток.

Что отличает Патрицию от ее предшественников (Trie, Prefix Trie) — в ней не происходит одностороннего ветвления, что обусловлено наличием того самого индекса — номера бита, необходимого для сравнения ключей.

Реализована программа в одном файле *main.c*. Сделано это из-за часто возникающей ошибки $Underfinedreferenceto...$ и для удобста заливания кода на чекер.

Стоит отдельно рассказать о сохранении и загрузке: прежде, чем записать информацию об узлах, я нумерую их. В моей программе для этого есть отдельная переменная *id*. Сделано это за тем, чтобы упроситить программирование: вместо рекурсивного вызова функции, я буду писать в поток номера его детей. Выгрузка данных происходит обратным образом: получаю айдишники и восстанавливаю таким образом указатели на детей.

Программа была сделана модульной, то есть при помощи шаблонов, чтобы можно было с легкостью поменять тип ключа. Также были реализованные необязательные геттеры и сеттеры, чтобы сохранить инкапсуляцию данных.

2 Исходный код

Ниже приведены методы и функции, которые используются в коде, и воспроизводят алгоритм, написанный ниже в литературе.

Стоит сказать, что в самой первой версии моего кода был использован алгоритм, описанный на лекции, который успешно работал, но у него был один большой минус – ключи хранились, в числе, высчитанный по хэш функции. Из-за этого в ключ можно было поместить лишь 13 символов. Далее код был переделан на свой собственный *string*. Код был модульным, то есть был сделан через *template*, поэтому замена произошла быстро, но потом возникли проблемы с удалением, которые я не мог решить. Но знакомый мне посоветовал нижеописанные статьи. После чего я убрал *string*, так как далее выяснилось он очень криво работал, и оставил *char**.

Ввод и вывод работают так, как написано в задании.

main.c	
static unsigned int StrLen(const char str)	Функция, которая подсчитывает количество символов в строке.
static inline bool Equal(char a , char b)	Функция, которая определяет равенство строк.
static inline int GetBit(const char key , int bit)	Функция, которая получает bit-ый бит от *key*
static inline int FirstDifBit(char *a, char *b)	Функция, которая получает первый отличный бит.
static inline void LowerCase(char *str)	Функция, которая переводит str в нижний регистр.
void Initialize(int b, T *k, unsigned long long v, TNode *l, TNode *r)	Метод структуры *TNode*, который инициализирует переменны в этой структуре.
bool Insert(T* key, unsigned long long value)	Метод структуры *TPatricia*, который осуществляет вставку.
bool TPatricia T ::Delete(T* k)	Метод структуры *TPatricia*, который осуществляет удаление.
void Save(std::ofstream & file)	Метод структуры *TPatricia*, который осуществляет сохранение дерева.
void Enumerate(TNode T *node, TNode T **nodes, int & index)	Метод структуры *TPatricia*, который осуществляет индексацию элементов в дереве. Используется в *Save*.
void Load(std::ifstream & file)	Метод структуры *TPatricia*, который осуществляет загрузку дерева.

struct $TNode$

```
template<class T>
  class TNode {
  private:
    int id = -1;
    int bit{};

    T *key;
    unsigned long long value{};
  public:
    TNode *left;
    TNode *right;

    int &GetRvalID() {
      return id;
    }
    int GetID() {
      return id;
    }
    void SetID(int iDiter) {
      id = iDiter;
    }
    unsigned long long &GetRvalVal() {
      return value;
    }
    unsigned long long GetVal() {
      return value;
    }
    void SetVal(unsigned long long valuer) {
      value = valuer;
    }
    T *GetKey() {
      return key;
    }
    void SetKey(T *keyer) {
      key = keyer;
    }
    int &GetRvalBit() {
      return bit;
    }
    int GetBit() {
      return bit;
    }
    void SetBit(int bitter) {
      bit = bitter;
    }

    void Initialize(int b, T *k, unsigned long long v, TNode *l, TNode *r);
    TNode();
```

```
49      TNode(int b, T *k, unsigned long long v);
50      TNode(int b, T *k, unsigned long long v, TNode *l, TNode *r);
51      friend std::ostream &operator<<(std::ostream &out, const TNode<T> &obj);
52    };
```

struct $TPatricia$

```
1   template<class T>
2     struct TPatricia {
3       TNode<T> *root;
4       int size;
5
6       TPatricia() {
7         root = new TNode<T>();
8         size = 0;
9       }
10
11      ~TPatricia() {
12        CleanUp(root);
13      }
14
15      void CleanUp(TNode<T> *node) {
16        if (node->left->GetBit() > node->GetBit()) {
17          CleanUp(node->left);
18        }
19        if (node->right->GetBit() > node->GetBit()) {
20          CleanUp(node->right);
21        }
22        delete node;
23      }
24
25      TNode<T> *Find(char *key) {
26        TNode<T> *p = root;
27        TNode<T> *q = root->left;
28
29        while (p->GetBit() < q->GetBit()) {
30          p = q;
31          q = (GetBit(key, q->GetBit()) ? q->right : q->left);
32        }
33        if (!Equal(key, q->GetKey())) {
34          return 0;
35        }
36        return q;
37      }
38
39      void KVCopy(TNode<T> *src, TNode<T> *dest) {
40        if (StrLen(dest->GetKey()) < StrLen(src->GetKey())) {
41          delete[] dest->GetKey();
42          dest->SetKey(new char[StrLen(src->GetKey()) + 1]);
43        }
```

```
44      strcpy(dest->GetKey(), src->GetKey());
45
46      dest->SetVal(src->GetVal());
47    }
48
49    TNode<T> *Insert(char *key, unsigned long long value);
50    bool Delete(char *k);
51    void Save(std::ofstream &file);
52    void Enumerate(TNode<T> *node, TNode<T> **nodes, int &index);
53    void Load(std::ifstream &file);
54  };
```

3 Консоль

```
/mnt/d/Documents/Projects/c++/DA-labs/cmake-build-linux/DA_lab2
+ a 1
+ A 2
+ aa 18446744073709551615
aa
A
-A
a
^D
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord

Process finished with exit code 0
```

4 Тест производительности

Сравнение производительности будет производиться с красно-черным деревом, представленный в стандартной библиотеке c++.

```
test 1000 lines
patricia: 0.000 ms
std::map: 0.001 ms

test 10000 lines
patricia: 0.003 ms
std::map: 0.005 ms

test 100000 lines
patricia: 0.023 ms
std::map: 0.047 ms
```

Для хранения слов PATRICIA оказалась много эффективней красно-черного дерева. Не удивительно, ведь сложность операций поиска/добавления/удаления в КЧ дереве — $O(log(n))$, где n — число элементов в дереве; в случае же Патриции, сложность этих же операций — $O(k)$, где k — длина слова. Притом Патриция сравнивает лишь определенные биты слов во время их поиска, а КЧ дереве же – все слово целиком. Нет, так же, затрат на балансировку Патриции в отличии от КЧ дерева. Интересно, что Патриция в результате вставки случайных слов сама по себе является почти сбалансированным деревом, и тем более она таковой является, чем длиннее слова в нее вставляются.

5 Выводы

Благодаря проделанной работе, я узнал что такое патриция. На практике выяснил как она работает, а также применил свой опыт по ООП, полученный на лекциях одноименного предмета.

Список литературы

[1] Dinesh P. Mehta, Sartaj Sahni. *Handbook of DATA STRUCTURES and APPLICATIONS.* — CHAPMAN & HALL/CRC

[2] *Дерево Патриция и с чем её едят – Медиум.*
URL: `https://medium.com/@grislava/дерево-патриция-и-с-чем-её-едят-a85c9f454bc4`
(дата обращения: 27.11.2020).

[3] *Позаимствованный код – CodeProject.*
URL: `https://www.codeproject.com/Articles/9497/Patricia-Trie-Template-Class`
(дата обращения: 27.11.2020).

[4] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008