Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студентка: А. Довженко
Преподаватель: Д. Е. Ильвохин
Группа: 08-207

Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №2

Необходимо создать программную библиотеку, реалзиующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длинной 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.

Структура данных: PATRICIA

1 Описание

В данной лабораторной работе мне нужно разработать словарь на предложенной структуре данных – PATRICIA. PATRICIA – одно из видов trie-деревьев, придуманное с целью ликвидации недостатков обычного trie-дерева: создание дополнительных узлов из-за однонаправленного ветвления и хранение двух различных типов узлов, которое приводит к усложнениям при работе с деревом. В PATRICIA мы избегаем однонаправленного ветвления благодаря тому, что в каждом узле содержится бит, который должен проверяться с целью выбора пути из этого узла. Внешние узлы исключаются при помощи того, что данные хранятся во внутренних узлах, а связи с внешними узлами заменяются связями, которые указывают в обратном направлении вверх на требуемый узел. Эти два изменения позволяют представлять trie-деревья как бинарные деревья, состоящие из узлов с ключом и двумя связями, а также дополнительным полем под бит. PATRICIA позволяет выполнять поиск N ключей в дереве, содержащем всего N узлов. Для выполнения одного поиска требуется всего лишь около lgN сравнений разрядов и одного сравнения полного ключа.

Данное дерево решено было реализовывать на основе двух структур TPatriciaNode и TPatricia. В первой хранится только лишь корень дерева и количество элементов в нем, во второй – ключ, значение, бит различия, два указателя на следующие узлы и индекс, который используется для сохранения и загрузки словаря. Также была реализована "обертка"словаря. Она повторяет функциональность созданного дерева и предоставляет стандартный интерфейс для работы со словарем. (Де)сериализацию решено было вынести в отдельный модуль для упрощения логики. Для нее я использую массив, в которое помещаются узлы дерева. При записи в файл указатели заменяются индексами, а при чтении индексы преобразуются в указатели. Это позволяет уменьшить сложность де(сериализации). В целях экономии памяти под ключ я выделяю не массив рамером 257, а работаю с динамической памятью. В словаре реализован интерфейс обработки ошибок для удобной отладки. Всегда приятно, когда понятно, в каком именно месте программа не работает.

2 Исходный код

main.c	
void WordToLower(char *word)	Перевод входной строки в нижний регистр.
cmdType ReadLine(FILE *in, TKey key, TValue *val)	Считывание входной строки.
TPatricia.c	
TPatricia TreeCreate(void);	Создание дерева.
TPatriciaNode NodeCreate(void);	Создание узла дерева.
void TreeDestroy(TPatricia tree);	Удаление дерева.
void NodeDestroy(TPatriciaNode node, TPatricia tree);	Удаление узла дерева.
void NodeFill(TPatriciaNode node, int32t skip, TKey key, TValue val, TPatriciaNode left, TPatriciaNode right);	Заполнение узла дерева.
void TreeDebugPrint(TPatricia tree);	Печать дерева.
int32t GetBit(TKey key, int32t idx);	Получение бита из ключа.
TPatriciaNode NodeSearch(TPatricia tree, TKey key);	Поиск ключа в дереве.
int TreeInsert(TPatricia tree, TKey key, TValue val);	Вставка нового узла в дерево.
int TreeDelete(TPatricia tree, TKey key);	Удаление узла из дерева.
TDict.c	
Dict DictCreate(void);	Создание словаря.
int DictSearch(Dict dict, TKey key, TValue *val);	Поиск в словаре.
int DictAdd(Dict dict, TKey key, TValue val);	Добавление в словарь.
int DictRemove(Dict dict, TKey key);	Удаление из словаря.
int DictSave(Dict dict, const char *filename);	Сохранение словаря.
int DictLoad(Dict *dict, const char *filename);	Загрузка словаря.
serialization.c	
int DictWrite(FILE *f, TPatricia tree);	Запись словаря.
int DictRead(FILE *f, TPatricia tree);	Чтение словаря.
int DictWriteLinear(FILE *f, TPatricia tree)	Запись записей словаря с линейной сложностью.

int GetArray(TPatriciaNode node, TPatriciaNode *arr, int32t idx)	Помещение узлов дерева в массив.
int DictReadLinear(FILE *f, TPatricia tree);	Чтение записей в словарь с линейной сложностью.
int SetArray(FILE *f, TPatriciaNode *arr);	Получение массива с узлами дерева из файла.

```
1  typedef char *TKey;
2  typedef uint64_t TValue;
3
4  typedef struct patriciaNode *TPatriciaNode;
5
6  struct patriciaNode {
7      int32_t skip;
8      TKey key;
9      TValue value;
10     TPatriciaNode left;
11     TPatriciaNode right;
12     int32_t idx;
13 };
14
15 typedef struct {
16     TPatriciaNode root;
17     int32_t size;
18 } *TPatricia;
```

3 Консоль

```
karma@karma:~/mai_study/DA/da2$ cat tests/01.t
+ SFc 584494
+ MHpNcLDB 63683
-mhpncldb
FzEDXXN
sfc
+ K 391396
i
-k
hLyHHT
+ QfhF 418977
+ Xyki 19986
nbJCwqCg
-yWrjPZmB
-xyki
-sfc
Mht
+ MQfLZm 744858
mqflzm
+ Q 798978
karma@karma:~/mai_study/DA/da2$ ./da2 <tests/01.t
OK
OK
OK
NoSuchWord
OK: 584494
OK
NoSuchWord
OK
NoSuchWord
OK
OK
NoSuchWord
NoSuchWord
OK
OK
NoSuchWord
OK
OK: 744858
```

OK

4 Тест производительности

Я случайным образом генерирую пять файлов, которые представляют из себя следующее:
1 тест – 1 000 000 элементов добавляются.
2 тест – 1 000 элементов добавляются.
3 тест – 25 000 добавляются, а затем удаляются.
4 тест – 1 000 элементов добавляются, а затем удаляются.
5 тест – 500 000 добавляются, каждый пятый удаляется сразу после добавления.

Каждый элемент состоит из ключа и значения. Для сравнения со своей структурой беру std::map из STL, сложность которой составляет $O(\lg(n))$.

```
karma@karma:~/mai_study/DA/da2$ ./da2 <tests/01.t
Type dict: PATRICIA
Time 2.335290 sec
Type dict: map
Time 3.259372 sec
karma@karma:~/mai_study/DA/da2$ ./da2 <tests/02.t
Type dict: PATRICIA
Time 0.001359 sec
Type dict: map
Time 0.001961 sec
karma@karma:~/mai_study/DA/da2$ ./da2 <tests/03.t
Type dict: PATRICIA
Time 0.059402 sec
Type dict: map
Time 0.087260 sec
karma@karma:~/mai_study/DA/da2$ ./da2 <tests/04.t
Type dict: PATRICIA
Time 0.002078 sec
Type dict: map
Time 0.002224 sec
karma@karma:~/mai_study/DA/da2$ ./da2 <tests/05.t
Type dict: PATRICIA
Time 0.962480 sec
Type dict: map
Time 1.727051 sec
```

Как видно из тестов, моя структура выигрывает у map, как на больших, так и на малых данных. Map из stl реализован на основе красно-черного дерева, в котором

операции вставки и удаления выполняются за O($\lg(n)$), в то время как в PATRICIA эти операции составляют O(h). При тестировании я использовала ключи длиной от 10 до 20 знаков, вполне возможно, что мой выигрыш можно объяснить тем, что в сгенерированных файлах было много ключей большой длины, как со схожими префиксами, так и с разными. Можно предположить, что благодаря этому удалось получить почти идеально сбалансированное дерево (по сравнению с тем, которое могло бы быть, если бы ключи были совсем небольшие) и также, что характерно для патриции, сэкономить время на сравнении ключей. Как известно, в PATRICIA сравнивается только 1 бит ключа, в то время как в красно-черном дереве весь ключ. Эту гипотезу подтверждает и тот факт, что мой выигрыш во времени пропорционален размеру теста.

5 Отладка

1 посылка на чекер: Compilation error.
Функция TPatriciaNode NodeCreate(void), которая должна возвращать узел дерева, не возвращала ничего. Исправлено добавлением return'а.
2 посылка на чекер: Compilation error.
В длинном условии if (skip = -1 && !strcmp(key, "empty") && val == 0) все проверяемые случаи нужно было "завернуть"в скобки.
3 и 4 посылки на чекер: Time limit exceeded at test 03.t.
В ходе отладки было обнаружено, что удаление узлов дерева обрабатывает не все случаи. После исправления удаления Time limit изчез.
5 посылка на чекер: Wrong answer at test 04.t
В ходе отладки было обнаружено, что вставка новых элементов в некоторых случаях работает неправильно. Конкретно тогда, когда новый узел нужно вставить между существующими узлами (терялся указатель на дочерний узел). Было исправлено получение бита из строки. Полученный бит не соответствовал тому, который есть на самом деле, потому что битовый сдвиг совершался неправильно.
6 посылка на чекер: Compilation error
После удаления всех многострочных комментариев ошибка компиляции была исправлена.
7 послыка на чекер: Wrong answer at test 02.t
Забыла закомментировать функцию печати дерева для отладки.
8 посылка на чекер: Wrong answer at test 04.t
При получении ошибки печаталось Error, поменяла на ERROR. Не помогло.
9 посылка на чекер: Wrong answer at test 04.t.
Из-за двойного освобождения памяти программа некорректно работала. Добавла условия на проверку существования корня (ошибка была при удалении корня дерева).

10 посылка на чекер: OK
Было решено уменьшить сложность (де)сериализации.
11 посылка на чекер: Compilation error
В коде существовало условие, которое всегда было истинно. Этот участок кода удалён.
12 послыка на чекер: OK

6 Выводы

PATRICIA удобна тем, что позволяет идентифицировать разряды, которые отличают ключи, после чего встраивать их в структуру данных без избыточных узлов. Это обеспечивает быстрое попадание от любого искомого ключа к единственному ключу в структуре, который мог бы быть равен искомому. Если сравнивать PATRICIA-деревья со стандартными trie-деревьями, то первые содержат в два раза меньше узлов и являются почти идеально сбалансированными.

Вставка и поиск случайного ключа в дереве требует в среднем приблизительно lgN битовых сравнений. Количество сравнений никогда не превысит длины ключа. Для PATRICIA не существует никаких ограничений на длину искомых ключей, к тому же оно также эффективно и для ключей переменной длины. Сложность поиска, вставки и удаления в PATRICIA составляет O(h), где h – высота дерева. Сложности данных операций в моей программе близки к этому. Также в данной лабораторной работе необходимо было выполнить сериализацию и десериализацию. В первой версии моей программы запись осуществлялась линейно, а чтение словаря из файла составляло за O(nh). Во второй версии де(сериализация) была усовершенствована благодаря индексировании узлов на этапе записи в файл. Это позволило добиться сложности обеих операций O(n).

В открытых источниках информации о PATRICIA представлено не так много, как, например, о структурах других вариантов данной лабораторной работы или о классическом trie-дереве. Мне показалось это странным, возможно это не самый удобный или не самый эффективный вариант реализации префиксных деревьев. В любом случае, идея, заложенная в основе PATRICIA-деревьев, показалась мне необычной и было приятно ознакомиться с ней глубже. Во время поиска материала я узнала о такой интересной вариации PATRICIA-дерева как Merkle Patricia Trie, которую использовали при создании Эфириума.