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. В Спиридонов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2017

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.

Вариант дерева: PATRICIA.

Вариант ключа: регистронезависимая последовательность букв английского алфавита длиной не более 256 символов.

Вариант значения: числа от 0 до $2^{64} - 1$.

1 Описание

Нагруженное дерево (или Trie; Patricia - разновидность Trie-дерева) – структура данных реализующая интерфейс ассоциативного массива, то есть позволяющая хранить пары «ключ–значение». В большинстве случаев ключами выступают строки, однако в качестве ключей можно использовать любые типы данных, представимые как последовательность байт.

В узлах Trie хранятся односимвольные метки, а ключём, который соответствует некоему узлу является путь от корня дерева до этого узла.
Корень дерева, очевидно, соответствует пустому ключу.

Сжатое префиксное дерево Patricia является довольно быстрым и эффективным по памяти методом реализации словаря. Применение этой модели дерева может существенно увеличить продуктивность поиска и внесения элементов в словарь. Также есть не мало алгоритмов, которые построены на принципе дерева Patricia, например алгоритм поиска подстроки «Ахо — Корасик».

Существует 2 основных типа оптимизации нагруженного дерева:

1. Сжатое нагруженное дерево получается из обычного удалением промежуточных узлов, которые ведут к единственному не промежуточному узлу. Например, цепочка промежуточных узлов с метками a, b, c заменяется на один узел с меткой abc.

2. Patricia нагруженное дерево получается из сжатого (или обычного) удалением промежуточных узлов, которые имеют одного ребенка.

Рассмотрим основные операции, связанные с префиксным деревом типа Patricia.

1.1 Операция поиска строки в префиксном дереве.

Движемся от корня дерева. Если корень пустой, то поиск неудачный. Иначе, сравниваем ключ в узле с текущей строкой. Для этого воспользуемся функцией, которая вычисляет длину наибольшего общего префикса двух строк заданной длины.
В случае поиска нас интересуют три случая:

1. общий префикс может быть пустым, тогда надо рекурсивно продолжить поиск в младшей сестре данного узла, т.е. перейти по ссылке next;

2. общий префикс равен искомой строке x — поиск успешный, узел найден (тут мы существенно используем тот факт, что конец строки за счет наличия в нем терминального символа может быть найден только в листе дерева);

3. общий префикс совпадает с ключом, но не совпадает с x — переходим рекурсивно по ссылке link к старшему дочернему узлу, передавая ему для поиска строку x без найденного префикса.

Если общий префикс есть, но не совпадает с ключом, то поиск также является неудачным.

1.2 Операция вставки новой строки в префиксное дерево.

Вставка нового ключа (как и в двоичных деревьях поиска) очень похожа на поиск ключа. Естественно с несколькими отличиями. Во-первых, в случае пустого дерева нужно создать узел с заданным ключом и вернуть указатель на этот узел. Во-вторых, если длина общего префикса текущего ключа и текущей строки x больше нуля, но меньше длины ключа (второй случай недачного поиска), то надо разбить текущий узел на два, оставив в родительском узле найденный префикс, и поместив в дочерний узел p оставшуюся часть ключа. После разделения нужно продолжить процесс вставки в узле p строки x без найденного префикса.

1.3 Удаление ключа из префиксного дерева.

Как обычно, удаление ключа — самая сложная операция. Хотя в случае префиксного дерева все выглядит не столь страшно. Дело в том, что при удалении ключа удаляется всего один листовой узел, соответствующий суффиксу некоторому удаляемого ключа. Сначала мы находим этот узел, если поиск успешный, то мы его удаляем и возвращаем указатель на младшего брата.
В принципе, на этом процесс удаления можно было бы и закончить, однако возникает небольшая проблема — после удаления узла в дереве может образоваться цепочка из двух узлов t и p, в которой у первого узла t имеется единственный дочерний узел p. Следовательно, если мы хотим держать дерево в сжатой форме, то нужно соединить эти два узла в один, произведя операцию слияния.

2 Описание программы

По условию задачи было ясно, что красиво такую программу написать в одном файле не получится и придется её разбить на несколько основных файлов:

1. main.cpp (содержит основной метод main, LengthString - метод подсчёт длины строки, ToLower - метод перевода строки в нижний регистр)
2. PatriciaTree.h и PatriciaTree.cpp (описание и реализация класса TPatriciaTree)
3. StackContainer.h и StackContainer.cpp (описание и реализация класса TStack и TStackData)

Для удобства сборки проекта были написаны:

1. makefile (для сборки и автоматического тестирования проекта из консоли)
2. CMakeLists.txt (для автоматической сборки проекта в CLion, Visual Studio Code, Visual Studio Enterprise 2017)

Также были взяты «test_generator.py» и «wrapper.sh» с GitHub'a[1] и переделаны.

main.cpp	
void ToLower (char *t, int len)	Перевод char* в нижний регистр.
unsigned int LengthString (const char *string)	Подсчет строки.
int main (int argc, char **argv)	Точка входа в программу.
TPatriciaTree.h и TPatriciaTree.cpp	
TPatriciaTree ()	Пустой конструктор.
const unsigned long long *Search (const char *, unsigned int)	Поиск в дереве.
bool Insert (const char *, unsigned int, unsigned long long)	Вставка в дерево.
bool Remove (const char *, unsigned int)	Удаление из дерева.
void Load (const char *)	Загрузка дерева из файла.
void Save (const char *)	Сохранение дерева в файл.
virtual ~TPatriciaTree ()	Деструктор.
TPatriciaTree *next	Ссылка на брата.
TPatriciaTree *link	Ссылка на потомка.
char *key	Поле ключа.
unsigned int length	Длина ключа.
unsigned long long data	Поле значения.
StackContainer.h и StackContainer.hpp	
TStackData ()	Пустой конструктор.
T &GetData ()	Получение внутреннего значения.
TStackData <T> *GetNext	Получить следующий элемент стека.
TStackData <T> *GetPrev ()	Получить предыдущий элемент стека.
virtual ~TStackData ()	Деструктор элемента стека.
T data	Значение внутри элемента стека.
TStackData <T> *next	Ссылка на следующий элемент стека.
TStackData <T> *prev	Ссылка на предыдущий элемент стека.
TStack ()	Конструктор стека.
T *Top ()	Получения верхнего элемента стека.
void Pop ()	Метод удаление верхнего элемента .
void Push (T &)	Вставка элемента в стек.
bool Empty ()	Проверка пуст ли стек.
virtual ~TStack ()	Деструктор.
TStackData <T> *head	Поле головы стека.

3 Код дополнительных файлов

- makefile

```
FLAGS=-g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -O2
CC=g++
LINK=-lm

all: tree main

main: main.cpp
  $(CC) $(FLAGS) -c main.cpp
  $(CC) $(FLAGS) -o Patricia PatriciaTree.o main.o $(LINK)

maind: main.cpp
  $(CC) $(FLAGSD) -c main.cpp
  $(CC) $(FLAGSD) -o Patricia PatriciaTree.o main.o $(LINK)

mainp: main.cpp
  $(CC) $(FLAGS) -c main.cpp
  $(CC) $(FLAGS) -ftest-coverage -fprofile-arcs -o Patricia PatriciaTree.o main.o
       $(LINK)

tree: PatriciaTree.cpp
  $(CC) $(FLAGS) -c PatriciaTree.cpp

clear:
  rm -f *.o
  rm -fr *.dSYM

runtests:
  rm -rf tests
  mkdir tests
  sh wrapper.sh
```

- test_generator.py

```python
# -*- coding: utf-8 -*-
import sys
import random
import string
import copy
from random import choice
from string import ascii_uppercase
def get_random_key():
    return ''.join(choice(ascii_uppercase) for i in range(random.randint(1, 256))
        )

if __name__ == "__main__":
    count_of_tests = 1
    actions = [ "+", "-", "?", "!"]
    acts_file = ["Load test", "Save test"]
    for enum in range( count_of_tests ):
        keys = {}
        save_file = {}
        test_file_name = "tests/{:02d}".format( enum + 1 )
        with open( "{0}.t".format( test_file_name ), 'w' ) as output_file, \
             open( "{0}.a".format( test_file_name ), "w" ) as answer_file:

            for _ in range( random.randint(1000, 1000) ):
                action = random.choice( actions )
                if action == "+":
                    key = get_random_key()
                    value = random.randint( 1, 50)
                    output_file.write("+ {0} {1}\n".format( key, value ))
                    key = key.lower()
                    answer = "Exist"
                    if key not in keys:
                        answer = "OK"
                        keys[key] = value
                    answer_file.write( "{0}\n".format( answer ) )

                elif action == "?":
                    search_exist_element = random.choice( [ True, False ] )
                    key = random.choice( [ key for key in keys.keys() ] ) if
                        search_exist_element and len( keys.keys() ) > 0 else
                        get_random_key()
                    output_file.write( "{0}\n".format( key ) )
                    key = key.lower()
                    if key in keys:
                        answer = "OK: {0}".format( keys[key] )
                    else:
                        answer = "NoSuchWord"
                    answer_file.write( "{0}\n".format( answer ) )

```

```
46                  elif action == "-":
47                      key = get_random_key()
48                      output_file.write("- {0}\n".format(key))
49                      key = key.lower()
50                      answer = "NoSuchWord"
51                      if key in keys:
52                          del keys[key]
53                          answer = "OK"
54                      answer_file.write("{0}\n".format( answer ) )
55
56                  elif action == "!":
57                      act_file = random.choice(acts_file)
58                      if act_file == "Save test":
59                          output_file.write("{0} {1}\n".format( action, act_file ))
60                          save_file = keys.copy()
61                          answer = "OK"
62                      elif act_file == "Load test":
63                          output_file.write("{0} {1}\n".format( action, act_file ))
64                          keys = {}
65                          keys = save_file.copy()
66                          answer = "OK"
67                      answer_file.write( "{0}\n".format( answer ) )
```

- benchmark.cpp

```cpp
#include <cstdio>
#include <string.h>
#include <map>
#include <fstream>
#include <iostream>

void LowerString(std::string &);
void Parsing(char &, std::string &, unsigned long long &);


int main() {
    std::ofstream fout("benchTime");
    std::map<std::string, unsigned long long> map;
    std::string buffer;
    char action;
    unsigned long long value;
    clock_t start = clock();
    while(true) {
        Parsing(action, buffer, value);
        if(action == 'E') {
            break;
        }
        switch(action) {
            case '+':
                if(map.count(buffer) == 0) {
                    map[buffer] = value;
                    printf("OK\n");
                } else {
                    printf("Exist\n");
                }
                break;
            case '-':
                if (map[buffer] == 0) {
                    printf("NoSuchWord\n");
                }
                else {
                    map.erase(buffer);
                    printf("OK\n");
                }
                break;
            case 'F':
                if (map[buffer] == 0) {
                    printf("NoSuchWord\n");
                }
                else {
                    printf("OK: %lu\n", map[buffer]);
                }
                break;
```

```
49          }
50      }
51      clock_t finish = clock();
52      double time = (double) (finish - start)/ CLOCKS_PER_SEC;
53      fout << "Std map time:" << time << std::endl;
54      fout.close();
55      return 0;
56  }
57
58  void LowerString(std::string& buffer) {
59      int index = 0;
60      while (index < buffer.size()) {
61          if (buffer[index] >= 'A' && buffer[index] <= 'Z') {
62              buffer[index] += -'A' + 'a';
63          }
64          index++;
65      }
66  }
67
68  void Parsing(char &action, std::string& buffer, unsigned long long &value) {
69      char ch;
70      size_t i = 0;
71      ch = getchar();
72      if (ch == EOF) {
73          action = 'E';
74          return;
75      }
76      if (ch == '+') {
77          action = ch;
78          getchar();
79          std::cin >> buffer >>value;
80          getchar();
81          LowerString(buffer);
82      } else if (ch == '-') {
83          action = ch;
84          getchar();
85          std::cin >> buffer;
86          getchar();
87          LowerString(buffer);
88      } else if (ch == '!') {
89          getchar();
90          action = getchar();
91          while((ch = getchar()) != ' ') {
92              i++;
93          }
94          scanf("%s", buffer);
95          getchar();
96      } else {
97          action = 'F';
```

```
98          buffer[i++] = ch;
99          while((ch = getchar()) != '\n') {
100             buffer[i++] = ch;
101         }
102         buffer[i] = '\0';
103         LowerString(buffer);
104     }
105 }
```

- wrapper.sh

```
#!/bin/bash
make
make clear
rm -rf tests
mkdir -p tests
if ! python3 test_generator.py ; then
   echo "ERROR: Failed to python generate tests."
   exit 1
fi

for test_file in `ls tests/*.t`; do
    echo "Execute ${test_file}"
    if ! ./Patricia < $test_file > tmp ; then
        echo "ERROR"
        continue
    fi
    answer_file="${test_file%.*}"

    if ! diff -u "${answer_file}.a" tmp ; then
        echo "Failed"
    else
        echo "OK"
    fi
done
```

4 Консоль

```
$ make
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -O2
-c PatriciaTree.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -O2
-c main.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -O2
-o Patricia PatriciaTree.o main.o -lm
$ ./Patricia
g
NoSuchWord
+ g 4
OK
+ kij 6
OK
+ de 2
OK
de
OK: 2
- de
OK
+ lp 6
OK
! Save test
OK
- g
OK
- kij
OK
kij
NoSuchWord
! Load test
OK
kij
OK: 6
- e
NoSuchWord
+ ws 2
OK
```

5 Проверка на утечки памяти

Тестирование проводилось на файле в 10 000 строк, на системе Arch Linux, с помощью утилиты Valgrind.

```
$ valgrind ./Patricia < tests/01.t > tmp
==16246== Memcheck, a memory error detector
==16246== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16246== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==16246== Command: ./Patricia
==16246==
==16246==
==16246== HEAP SUMMARY:
==16246==     in use at exit: 0 bytes in 0 blocks
==16246==   total heap usage: 3,362,528 allocs, 3,362,528 frees, 235,824,982 bytes al
==16246==
==16246== All heap blocks were freed -- no leaks are possible
==16246==
==16246== For counts of detected and suppressed errors, rerun with: -v
==16246== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
$ wc tests/01.t
  10000   22476 1016277 tests/01.t
```

6 Тест производительности

Чтобы замерить скорость выполнения программы, в main были добавлены следующие строчки, для замера времени начала и конца выполнения.

```
...
clock_t start = clock();
...
clock_t end = clock();
    double time = (double)(end-start)/CLOCKS_PER_SEC;
    std::ofstream file("time");
    file << "Time : " << time << std::endl;
    file.close();
```

```
$ ./Patricia < tests/01.t > tmp
$ cat time
Time : 0.004019
$ ./Patricia < tests/02.t > tmp
$ cat time
Time : 0.07181
$ ./Patricia < tests/03.t > tmp
$ cat time
Time : 0.149677
$ wc tests/01.t
  492   954 65297 tests/01.t
$ wc tests/02.t
  25154   50598 3303285 tests/02.t
$ wc tests/03.t
  49993   99693 6573363 tests/03.t
$ g++ -g -std=c++11 benchmark.cpp
$ ./a.out < tests/01.t > tmp
$ cat benchTime
Std map time: 0.008452
$ ./a.out < tests/02.t > tmp
$ cat benchTime
Std map time: 0.103763
$ ./a.out < tests/03.t > tmp
$ cat benchTime
Std map time:0.355689
```

7 Выводы

В целом написание дерева не вызвало никаких трудностей и я был приятно удивлен, что Patricia быстрее, чем std::map.
Для себя я взялся за написание реализации Patricia и на нескольких других языках, потому что часто имею дело со словарями.

Список литературы

[1] *GitHub одного из семинаристов.*
URL: `https://github.com/toshunster/da`.

[2] *Шаблоны функций в C++.*
URL: `http://cppstudio.com/post/5165/`.

[3] *Шаблоны функций в C++.*
URL: `http://cppstudio.com/post/673/`.

[4] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))