Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Дискретный анализ»

Студент: Т. А. Габдуллин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №5

Задача: Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из входных строк, необходимо воспользоваться полученным суффиксным деревом для решения своего варианта задания. Алфавит строк: строчные буквы латинского алфавита (т.е., от a до z).

Вариант алгоритма: 2. Поиск с использованием суффиксного массива

Входные данные: Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

Выходные данные: Для каждого образца, найденного в тексте, нужно распечатать строчку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

1 Алгоритм

АВЛ-дерево - дерево, у которого выстота левого поддерева любого узла отличается от высоты правого не более чем на 1. Оно является частным случаем бинарного дерева. Для чего нужно соблюдать балансировку? При определенных "плохих"данных дерево может вырождаться в линейный список. Тогда в худшем случае, добавление, поиск и удаление будут производиться за линейное время. Но если использовать балансировку, то мы будем иметь классическое дерево – дерево с большой "кроной". Такая структуризация данных уменьшит время работы с деревом – все приведенные выше операции будут производиться за $O(logn)$. Балансировка работает за $O(1)$ – мы всегда будем иметь сбалансированное дерево, вне зависимости от количества входных данных.

В целом, процедуры добавления и удаления элементов ничем не отличаются от таких же операций в бинарном дереве. Только вот в AVL-дереве после каждых действий с деревом необходимо подниматься вверх от удаленного или добавленного элемента и вычислять балансы родительских узлов. Изначально баланс равен 0. Если в какой то вершине нарушена балансировка - т.е. баланс равен 2 или -2, то производятся повороты дерева вокруг этой вершины. Перейдем к более подробному описаню алгоритма действий с AVL-деревом.

Вставка в AVL-дерево

▶ Вставляем элемент как в обычное дерево поиска

▶ Поднимаемся вверх и пересчитываем баланс: пришли из левого поддерева – прибавим 1 к балансу, из правого – вычтем 1

▶ Баланс = 0 – высота не изменилась – балансировка окончена

▶ Баланс = ± 1 – высота изменилась, продолжаем подъем

▶ Баланс = ± 2 – инвариант нарушен, делаем соответствующие повороты, если получаем новый баланс, равный 0, то останавливаемся, иначе (± 1) – продолжаем подъем

Удаление из AVL-дерева

▶ Удаляем элемент как в обычном дереве поиска

▶ Поднимаемся вверх от удаленного элемента и пересчитываем баланс: пришли из левого поддерева – вычтем 1 из баланса, из правого – прибавим 1

▶ Баланс = ± 1 – высота не изменилась – балансировка окончена

▶ Баланс = 0 – высота уменьшилась – продолжаем

▶ Баланс = ± 2 – инвариант нарушен, делаем соответствующие повороты, если получаем новый баланс, равный 0, то продолжаем подъем, иначе – останавливаемся

Линейная оценка: $\emptyset(n)$

Описание взято с сайта *neerc.ifmo.ru*.

2 Листинг

1. main.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <string.h>
#include "TAvlTree.h"
#include "node.h"
using namespace std;
void ToLowerCase(char* Buffer)
{
    for(int i = 0; i < 256; ++i)
    {
        if(Buffer[i] == '\0' || Buffer[i] == '\n') break;
        if(Buffer[i] >= 'A' && Buffer[i] <= 'Z')
        {
            Buffer[i] += 32;
        }
    }
}
int main(int argc, char* argv[])
{
    TAvlTree* tree = new TAvlTree();
    unsigned long long int tmpVal;
    int command = getchar();
    char tmpStr[256];
    while(command != EOF)
    {
        if(command == '+') // add
        {
            scanf("%s", tmpStr);
            ToLowerCase(tmpStr);
            size_t len = strlen(tmpStr);
            scanf("%llu", &tmpVal);
            getchar();
            tree->Insert(tmpVal, tmpStr, &(tree->Root), NULL, len);
        }
        else if(command == '-') // delete
        {

            scanf("%s", tmpStr);
            ToLowerCase(tmpStr);
            size_t len = strlen(tmpStr);
            getchar();
            if(tree->Remove(tmpStr, &(tree->Root), len)) printf("OK\n");
            else printf("NoSuchWord\n");
        }
        else if(command == '!')
```

```
47          {
48              getchar();// skip space
49              command = getchar();
50
51              if(command == 'L') // Load
52              {
53                  while(command != ' ') command = getchar();
54                  scanf("%s", tmpStr); // path/to/file
55                  getchar();
56                  FILE* fPtr = fopen(tmpStr, "r");
57                  if(fPtr == NULL)
58                  {
59                      perror("ERROR: can't open file");
60                      while(command != '\n') command = getchar();
61                      continue;
62                  }
63                  tree->SimpleDelete(tree->Root);// we will delete all nodes!
64                  tree->Root = NULL;
65                  tree->LkpLoad(&(tree->Root), NULL, fPtr); // load new tree
66                  fclose(fPtr);
67                  printf("OK\n");
68              }
69              else // Save
70              {
71                  while(command != ' ') command = getchar();
72                  scanf("%s", tmpStr); // path/to/file
73                  getchar();
74                  FILE* fPtr = fopen(tmpStr, "w");
75                  if(fPtr == NULL)
76                  {
77                      perror("ERROR: can't open file");
78                      while(command != '\n') command = getchar();
79                      continue;
80                  }
81                  tree->LkpSave(tree->Root, fPtr);
82                  fclose(fPtr);
83                  printf("OK\n");
84              }
85          }
86          else // search
87          {
88              ungetc(command, stdin);
89              scanf("%s", tmpStr);
90              ToLowerCase(tmpStr);
91              size_t len = strlen(tmpStr);
92              getchar();
93              tree->Search(tmpStr, &(tree->Root), len);
94          }
95          command = getchar();
```

```
    }
    delete tree;
    return 0;
}
```

2. TAvlTree.h

```
#ifndef TAVL_TREE
#define TAVL_TREE
#include "node.h"
#include <stdbool.h>
class TAvlTree
{
public:

    TAvlTree();
    ~TAvlTree();
    int Height(TNode*);
    int CountBalance(TNode*);

    bool Insert(unsigned long long int, char*, TNode**, TNode**, size_t);
    bool Search(char* str, TNode**, size_t);
    bool Remove(char*, TNode**, size_t);

    void SimpleLeftRotate(TNode*);
    void SimpleRightRotate(TNode*);
    void SimpleDelete(TNode*);

    void LeftRotation(TNode*);
    void RightRotation(TNode*);
    void BigLeftRotation(TNode*);
    void BigRightRotation(TNode*);

    void RecountBalanceInsert(TNode*, bool);
    void RecountBalanceDelete(TNode*, bool);

    void LkpLoad(TNode**, TNode*, FILE*);
    void LkpSave(TNode*, FILE*);
    void UpdateRoot();
    TNode* Root;
private:
};
#endif // TAVL_TREE
```

3 Выводы

Благодаря лабораторной работе 2 я познакомился с новыми структурами данных – сбалансированными деревьями. Они хорошо подходят для хранения различной информации, потому что мы исключаем тот случай, когда представление данных может выродиться в линейный список.

Список литературы

[1] 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.))

[2] *Авл-Дерево*
URL:`http://neerc.ifmo.ru/wiki/index.php?title=Авл-Дерево`