Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу дискретного анализа

Студент: А. В. Синявский
Преподаватель: Н. А. Зацепин
Группа: М8О-308Б-18
Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа № 5 по курсу дискретного анализа

Выполнил студент группы М80-308Б-18 МАИ *Синявский Андрей.*

Условие

1. Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из выходных строк, необходимо воспользоваться полученным суффисным деревом для решения своего варианта задания.

   Алфавит строк: строчные буквы латинского алфавита (т.е. от a до z).

2. Вариант 03.

   Найти образец в тексте используя статистику совпадений.

Метод решения

Реализуем алгоритм Укконена построения сжатого суффиксного дерева, затем создаём при помощи этого алгоритма суффиксное дерево для образца, и наконец рассчитываем статистику совпадений для текста при помощи созданного дерева,выводя индексы, в которых результат равен длине образца.

1. Алгоритм Укконена.

   Алгоритм состоит из n фаз, где n - длина текста. На i-ой фазе все суффиксы продляются на i-ый символ строки. Будем использовать ряд оптимизаций алгоритма, чтобы прийти к амортизационной оценке O(1) для одной фазы, и O(n) для построения дерева соответственно. Одна из оптимизаций - использование суффиксных ссылок, чтобы не спускаться для каждого суффикса от корня. После перехода по суффиксной ссылке можно также сэкономить время, по возможности спускаясь к концу суффикса сразу на длину ребра. При создании листа, можно сразу записать в метку ребра весь суффикс целиком.И ещё одна оптимизация: если во время продления оказалось, что ничего делать не нужно, так как добавляемый символ уже исходит из текущей позиции, то также произойдёт для всех остальных суффиксов в дереве, и текущую фазу можно заканчивать.

2. Статистика совпадений.

   Алгоритм подразумевает существование массива, каждый элемент которого ms[i] содержит длину наибольшей подстроки текста, начинающейся в i-ой позиции, и совпадающей с КАКОЙ-ТО подстрокой образца.

   Для вычисления строится суффиксное дерево для образца. Первый элемент массива явно вычисляется как наибольший путь в дереве, совпадающий с началом

текста. Далее, аналогично с алгоритмом Укконена, переходим по суффиксной ссылке, используем скачки на длину ребра, чтобы перейти в позицию в дереве, в которой кончается совпадающая строка из первого элемента без первого символа, и продолжаем дальше искать совпадения.

Стоит заметить, что нам не нужен весь массив ms, так как нам интересны только позиции тех его элементов, значение в которых равно длине образца (мы ведь ищем полные вхождения), поэтому хранить массив нам не нужно, достаточно внутри него иногда выписывать в стандартный вывод нужную позицию текста.

Описание программы

Узлы представлены объектами класса TNode, суффиксное дерево объектом класса T_STree. Вычисление статистики совпадений реализовано публичным методом дерева.

В узле хранятся индексы начала и конца подстроки текста, которую представляет ребро, входящее в этот узел, набор детей-узлов, и суффиксная ссылка.

В классе суффиксного дерева содержится текст, по которому строится дерево и ссылка на корень дерева. Остальные поля и методы (разумеется, кроме конструктора/-деструктора, а также меодов Destroy и MatchStatistics), используется для реализации алгоритма Укконена и вычисления статистики совпадений.

Класс	Значение
class TNode	узел суффиксного дерева
class T_STree	класс суффиксного дерева
Метод класса T_STree	Значение
T_STree(std::string)	конструктор, запускает алгоритм Укконена
void Extend(...)	Реализация фазы алгоритма Укконена
void AddSLink(...)	запоминает переданную в него вершину, как потенциальное начало суффиксной ссылки, и подвязывает ссылку от предыдущей такой вершины к текущей
bool JumpDown(...)	пытается осуществить скачок по ребру, возвращает true в случае успеха, false иначе
void Destroy(...)	Рекурсивная функция удаления дерева

Дневник отладки

При создании этой таблицы была использована история посылок. К сожалению, мне доступны только последние 15, но их было значительно больше

Время	ведрикт	Описание
2020/09/06 13:45:11	Неправильный ответ	Ошибка в алгоритме статистики совпадений
2020/09/06 16:15:39	Неправильный ответ	Предыдущая ошибка исправлена, теперь WA не на 3, а на 5 тесте
2020/09/13 12:26:16	Ошибка выполнения	Внезапно падает на 10 тесте. Очередная ошибка в поиске статистики совпадений
2020/09/14 06:18:32	Ожидает подтверждения	отказался от хранения массива совпадений, помогло

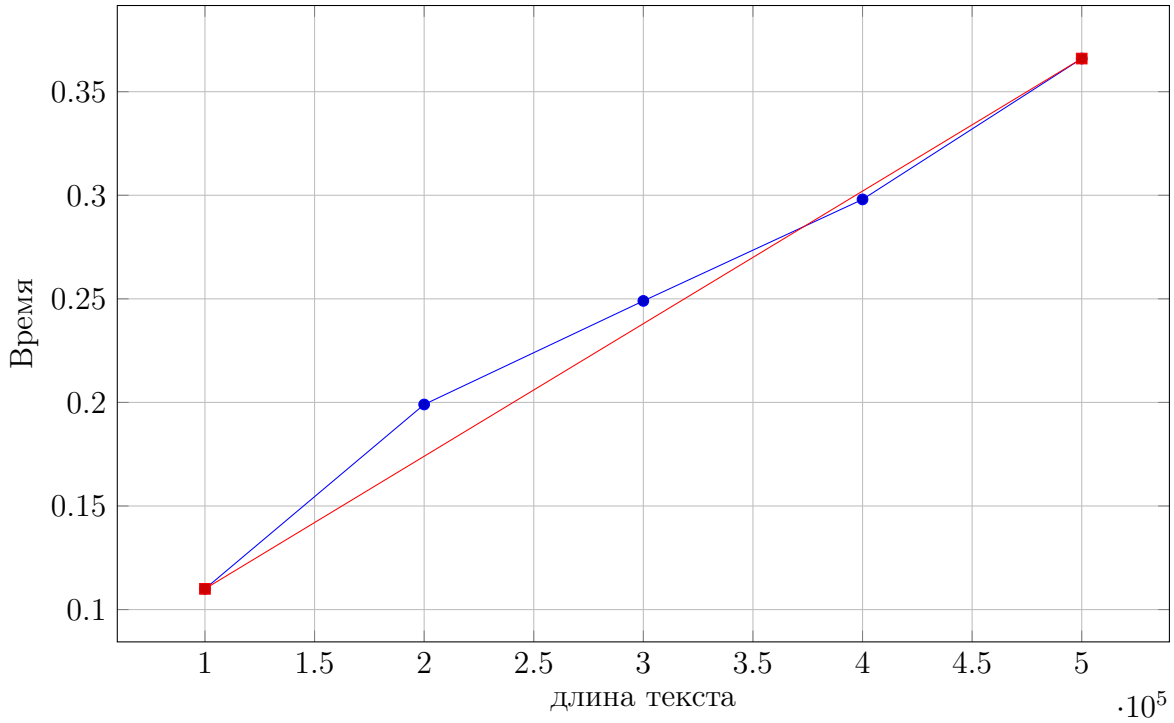
Тест производительности

Для генерации тестов использовалась следующая программа:

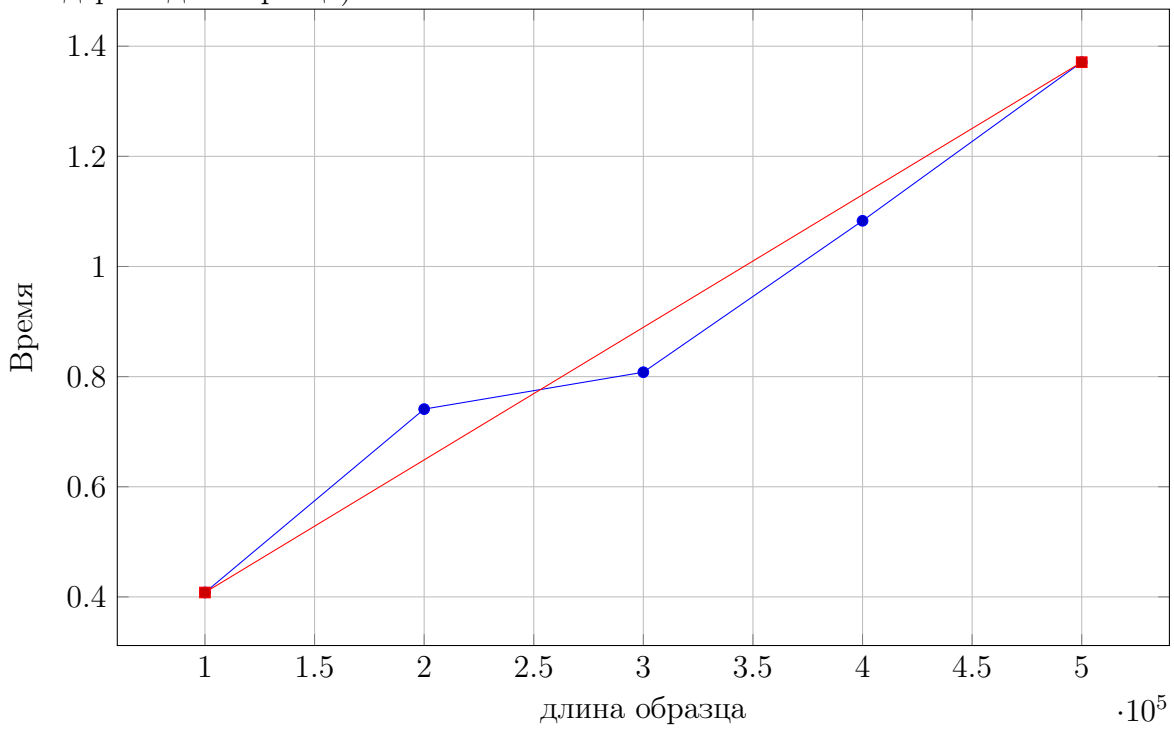
```
#include <iostream>
#include <ctime>
#include <fstream>
#include <cstdlib>

int main (int arc, char **argv) {
    //if (arc < ) return 0;
    size_t pattern_size = std::atoi(argv[1]);
    size_t text_size = std::atoi(argv[2]);
    std::string filename = argv[3];
    std::ofstream os(filename);
    srand(time(0));
    for (size_t i = 0; i < pattern_size; i++) {
        os << static_cast<char>(rand() % 6 + 97);
    }
    os << '\n';
    for (size_t i = 0; i < text_size; i++) {
        os << static_cast<char>(rand() % 6 + 97);
    }
    os << '\n';
    return 0;
}
```

Тест статистики совпадений: образец длины 100, текст переменой длины



Тест создания суффиксного дерева: текст длины 1, образец переменной длины (т. к. строим дерево для образца)



ЛИСТИНГ

solution.cpp

```
#include <iostream>
#include "SuffixTree.h"

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::string pattern, text;
    std::cin >> pattern >> text;
    pattern += '$';
    T_STree tree(pattern);
    pattern.pop_back();

    tree.MatchStatistics(text);
    return 0;
}
```

SuffixTree.h

```
#ifndef DA_LAB5_SUFFIXTREE_H
#define DA_LAB5_SUFFIXTREE_H

#include <iostream>
#include <string>
#include <map>

typedef std::string::iterator t_index;

class TNode {
public:
    t_index begin, end;
    std::map<char, TNode*> children;
    TNode* sLink;
    TNode(t_index begin, t_index end): begin(begin), end(end), sLink(nullptr)
    {};
    ~TNode() {};
};

class T_STree {
```

```
public:
    T_STree(std::string str);
    void MatchStatistics(std::string& _text);
    ~T_STree();

private:
    std::string text;
    TNode *root, *forSLink, *current;
    size_t length, remainder;
    t_index edge;
    void Extend(t_index symbIter);
    void AddSLink(TNode* node);
    bool JumpDown(t_index pos, TNode* node);
    void Destroy(TNode* node);
};


#endif //DA_LAB5_SUFFIXTREE_H
```

SuffixTree.cpp

```
#include "SuffixTree.h"

T_STree::T_STree(std::string str) {
    text = std::move(str);
    remainder = 0;
    length = 0;
    root = new TNode(text.end(), text.end());
    current = root->sLink = forSLink = root;
    for (t_index i = text.begin(); i != text.end(); ++i) {
        Extend(i);
    }
}

void T_STree::Extend(t_index symbIter) {
    remainder++;
    forSLink = root;
    while (remainder) {
        if (!length) edge = symbIter;
        auto it = current->children.find(*edge);
        TNode* next = (it == current->children.end()) ? nullptr : it->second;
        if (!next) {
```

```
            auto* leaf = new TNode(symbIter, text.end());
            current->children[*edge] = leaf;
            AddSLink(current);
        } else {
            if (JumpDown(symbIter, next)) {
                continue;
            }
            if (*(next->begin + length) == *symbIter) {
                length++;
                AddSLink(current);
                break;
            }
            auto *leaf = new TNode(symbIter, text.end());
            auto *split = new TNode(next->begin, next->begin + length);
            current->children[*edge] = split;
            split->children[*symbIter] = leaf;
            next->begin += length;
            split->children[*next->begin] = next;
            AddSLink(split);
        }
        remainder--;
        if (current == root && length) {
            length--;
            edge = symbIter - remainder + 1;
        } else current = current->sLink ? current->sLink : root;
    }
}

void T_STree::AddSLink(TNode *node) {
    if (forSLink != root) forSLink->sLink = node;
    forSLink = node;
}

bool T_STree::JumpDown(t_index pos, TNode *node) {
    int newEdgeLen = min(node->end, pos + 1) - node->begin;
    if (length >= newEdgeLen) {
        edge += newEdgeLen;
        length -= newEdgeLen;
        current = node;
        return true;
    }
    return false;
```

```
}

void T_STree::Destroy(TNode *node) {
    for (auto & i : node->children) Destroy(i.second);
    delete node;
}

T_STree::~T_STree() {
    Destroy(root);
}

void T_STree::MatchStatistics(std::string& _text) {
    size_t res = 0;
    length = 0;
    current = root;
    size_t i = 0;
    t_index iter = _text.begin();
    while (i <= _text.size()) {
        if (!length) edge = iter;
        auto it = current->children.find(*edge);
        TNode* next = (it == current->children.end()) ? nullptr : it->second;
        if (next) {
            size_t edge_len = next->end - next->begin;
            if (length >= edge_len) {
                length -= edge_len;
                edge += edge_len;
                current = next;
                continue;
            }
            bool flag = true;
            for (t_index edge_it = next->begin + length; edge_it != next->end
            && flag; edge_it++) {
                if (*edge_it != *iter) {
                    flag = false;
                    if (res == text.size()-1) std::cout << i+1 << '\n';
                    if (res) res--;
                    i++;
                    if (current == root) {
                        if (length) length--;
                        else iter++;
                        edge++;
                    } else current = current->sLink ? current->sLink : root;
```

```
                    continue;
                } else {
                    iter++;
                    res++;
                    length++;
                }
            }
        } else {
            if (res == text.size()-1) std::cout << i+1 << '\n';
            if (res) res--;
            i++;
            if (current == root) {
                if (length) length--;
                else iter++;
                edge++;
            } else current = current->sLink ? current->sLink : root;
        }
    }
}
```

Недочёты

Основной недочёт - это нечитаемый код метода MatchStatistics.

Выводы

Проделав данную работу, я изучил Алгоритм Укконена, узнал как устроены суффиксные деревья и как их можно применять для поиска подстрок в тексте.