Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»
Кафедра «школы бакалавриата (школа)»

Оценка работы_____
Руководитель от УрФУ

Суффиксное дерево. Алгоритм Укконена. Поиск наибольшей общей подстроки.

ОТЧЕТ
Вид практики Учебная практика
Тип практики Практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

Руководитель практики от предприятия (организации) Вахрушев В. А.
ФИО руководителя **Подпись**

Студент Ященко Р.А.
ФИО студента

Специальность (направление подготовки) 02.03.03 Математическое обеспечение и администрирование информационных систем

Группа МЕН-282203

Екатеринбург

2020

Оглавление

Введение

Будем называть текстом T строку из n символов t1 . . . tn, а каждое окончание текста t_i. . . t_n — его суффиксом.

Суффиксное дерево (ST) — это способ представления текста. Неформально говоря, чтобы построить ST для текста T = t_1 . . . t_n, нужно приписать специальный символ $ в конец текста, взять все n + 1 суффиксов, подвесить их за начала и склеить все ветки, идущие по одинаковым буквам. В каждом листе записывается номер суффикса, заканчивающегося в этом листе. Номером суффикса является индекс его начала в тексте T.

Заметим, что ни один суффикс в ST не может полностью лежать в другом суффиксе, поскольку они заканчиваются специальным символом $. Таким образом, листьев в ST всегда будет n + 1 для строки t_1 . . . t_n, то есть столько же, сколько суффиксов. Но общее число вершин в суффиксном дереве квадратично.

Разберемся теперь, как хранить суффиксное дерево, используя линейную память. Для этого оставим в ST только вершины разветвления, то есть имеющие не менее двух детей. Вместо строки для ребра будем хранить ссылку на сегмент текста T[i..j]. В таком виде суффиксное дерево называется сжатым.

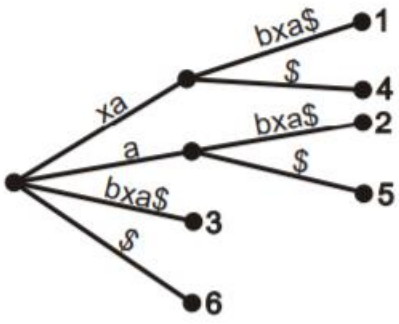

Рис. 1. Пример сжатого суффиксного дерева для строки xabxa$

Заметим, что, так как теперь каждая из внутренних вершин является вершиной разветвления, то она добавляет к своему поддереву как минимум один лист. Листьев же в ST всего n + 1 для строки $t_1 \ldots t_n$, поэтому внутренних вершин может быть в диапазоне 1 . . . n.

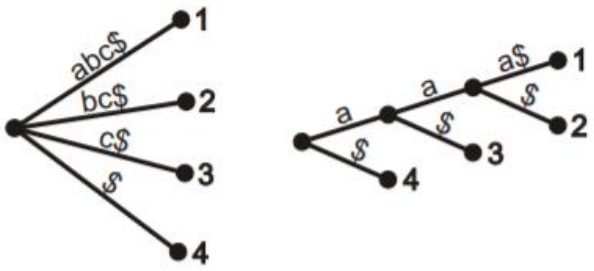

Рис. 2. Крайние случаи для числа внутренних вершин в сжатом суффиксном дереве: одна внутренняя вершина для строки abc$, три — для строки aaa$

Таким образом, всего вершин и ребер в сжатом суффиксном дереве будет линейное число, значит оно будет занимать линейную память.

Применение. Поиск наибольшей общей подстроки.

Даны тексты T_1 и T_2. Требуется найти длину их наибольшей общей подстроки. Для начала рассмотрим самый простой алгоритм, решающий эту задачу. В этом случае перебираем длину наибольшей общей подстроки, ее начала в текстах T_1 и T_2 и просто сравниваем подстроки. Тогда время работы алгоритма будет $O(n^4)$, где n — максимум из длин текстов.

Опишем решение с помощью суффиксного дерева. Построим суффиксное дерево для конкатенации исходных текстов $T = T_1T_2$. Также для удобства можно между этими текстами вставить еще один специальный символ, но можно обойтись и без него. Назовем «длинными» и «короткими» суффиксами текста T такие суффиксы, которые начинаются в текстах T_1 и T_2 соответственно. Для каждой внутренней вершины выясним: есть ли у нее одновременно потомки, соответствующие «короткому» и «длинному» суффиксам. Это можно сделать обходом в глубину. Для листа можно выяснить тип заканчивающегося в нем суффикса, зная индекс начала суффикса в тексте T. Поэтому можно узнать типы суффиксов для всех

вершин обходом в глубину, при возвращении сохраняя для каждой родительской вершины типы суффиксов, которые хранились для ее детей.

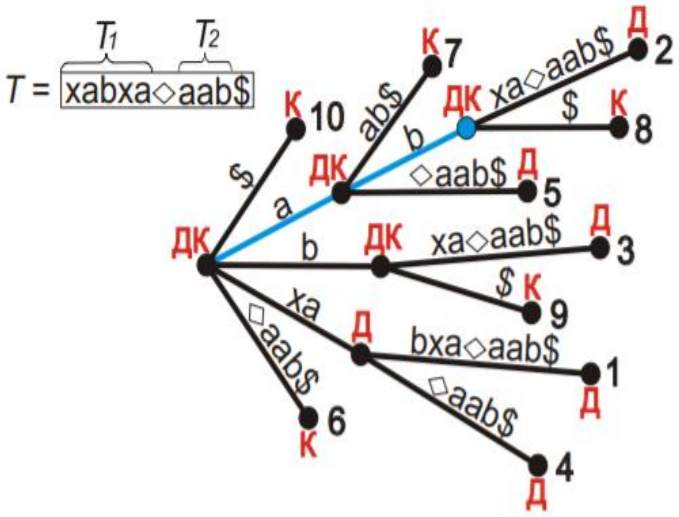

Рис. 3. Наибольшая общая подстрока текстов xabxa и aab равна ab

На рисунке 3 изображен пример, на котором буквы Д и К у вершины обозначают, что в ее поддереве заканчиваются «длинный» и «короткий» суффиксы соответственно. Если для вершины хранятся оба типа суффиксов (ДК), значит строка, соответствующая этой вершине в ST, встречается в T как минимум в двух местах, начинаясь в T_1 и T_2. Найдем самую удаленную от корня такую вершину. Ее глубина — ответ для задачи. Сложность этого алгоритма $O(|T_1| + |T_2|)$.

Алгоритм Укконена

Алгоритм Укконена – алгоритм построения суффиксного дерева для заданной строки за линейное время O(n).

Построение:

Правила:

1) Для фазы i+1 если суффикс S[j..i] заканчивается на последнем символе ребра листа, то тогда добавляем символ S[i+1] в конец.

2) Для фазы i+1 если суффикс S[j..i] заканчивается где-то в середине ребра и следующий символ не S[i+1], тогда нужно создать новое ребро с символом S[i+1].

3) Для фазы i+1 если суффикс S[j..i] заканчивается где-то в середине ребра и следующий символ это S[i+1], тогда ничего не делаем.

Суффиксная ссылки. Для каждой вершины с суффиксом x@, где x это один символ и @, возможно, пустая подстрока, где есть другая вершина с символом x. Эта вершина – суффиксная ссылка первой вершины. Если @ пустая, тогда суффиксная ссылка ведет к корню.

Некоторые хитрости при построении:

1) Идя вниз, если число символов на ребре меньше, чем число символов, которые еще следует пройти, тогда переходим прямо в конец ребра. Если число символов на ребре больше, чем число символов, которые нужно пройти, тогда переходим к тому символу, который нам нужен.

2) Останавливает итерацию, если использовано правило 3.

3) Храним глобальный конец на листе для правила 1.

Активная точка – это точка, откуда начинается обход до следующей фазы или расширения дерева. Всегда начинается с корня, а потом поменяется.

Активная вершина -вершина, с которой начнется активная точка.

Активное ребро – используется для выбора ребра из активной вершины. Оно имеет индекс символа.

Активная длина – как далеко идти по активному ребру.

Правила активной точки:

1) Если правило 3 было применено, то активная длина увеличивается на 1, если активная длина не больше, чем количество символов на ребре.

2) Если правило 3 было применено и если активная длина становится больше, чем количество символов на ребре, тогда меняется активная вершина, активное ребро и активная длина.

3) Если активная длина равна нулю, тогда всегда начинаем искать символ с корня.

4) Если правило 2 было применено и если активная вершина – корень, тогда активное ребро увеличивается на один и активная длина уменьшается на один.

5) Если правило 2 было применено и активная вершина не корень, тогда следуем по суффиксной ссылке и делаем активную вершину ссылкой и ничего не меняем.

Построение дерева и поиск наибольшей общей подстроки

Класс SuffixNode.java:

```
import java.util.HashMap;

public class SuffixNode {

    private Integer start;
    private Integer end;
    private SuffixNode suffixLink;
    private Boolean leaf;
    private Boolean[] type;
    private HashMap<Character, SuffixNode> children;

    public SuffixNode(Integer start, Integer end, Boolean leaf) {
        this.start = start;
        this.end = end;
        this.suffixLink = null;
        this.leaf = leaf;
        this.children = new HashMap<>();
        this.type = new Boolean[2];
        this.type[0] = Boolean.FALSE;
        this.type[1] = Boolean.FALSE;
    }

    public Boolean isTypeA() {
        return type[0];
    }

    public Boolean isTypeB() {
        return type[1];
    }

    public void setType(Boolean a, Boolean b) {
        this.type[0] = a;
        this.type[1] = b;
    }

    public Boolean[] getType(){
        return this.type;
    }

    public SuffixNode getChild(Character c) {
        return children.get(c);
    }

    public void addChild(Character c, SuffixNode n) {
        children.put(c, n);
    }

    public Boolean containsChild(Character c) {
        return children.containsKey(c);
    }

    public HashMap<Character, SuffixNode> getChildren() {
        return children;
    }

    public void setSuffixLink(SuffixNode n) {
        this.suffixLink = n;
```

```
    }

    public SuffixNode getSuffixLink() {
        return suffixLink;
    }

    public void setStart(Integer start) {
        this.start = start;
    }

    public Integer getStart() {
        return start;
    }

    public Integer getEnd() {
        return end;
    }

    public Boolean isLeaf() {
        return leaf;
    }
}
```

Класс SuffixTree.java:

```
import java.util.*;
import java.util.Stack;

public class SuffixTree {
    private SuffixNode activeNode;
    private SuffixNode lastNewNode;
    private Integer activeEdge;
    private Integer activeLength;
    private Integer remaining;
    private Integer splitIndex;
    private Integer globalEnd;
    private SuffixNode root;
    private String string;
    private Integer maxHeight;
    private Integer substringStartIndex;

    public SuffixTree(String string1, String string2) {
        this.splitIndex = string1.length();
        this.string = string1 + "#" + string2 + "$";
        this.activeEdge = -1;
        this.activeLength = 0;
        this.remaining = 0;
        this.root = new SuffixNode(-1, -1, Boolean.FALSE);
        this.activeNode = this.root;
        this.globalEnd = -1;
        buildTree();
    }

    private SuffixNode createNode(Integer start, Integer end, Boolean leaf) {
        SuffixNode newNode = new SuffixNode(start, end, leaf);
        newNode.setSuffixLink(this.root);
        return newNode;
    }

    public void buildTree() {
```

```
        for(int i = 0; i < string.length(); i++) {
            this.globalEnd++;
            this.remaining = this.remaining + 1;
            this.lastNewNode = null;
            Boolean showstopper = Boolean.FALSE;

            while(this.remaining > 0) {
                showstopper = Boolean.FALSE;
                if(this.activeLength == 0) {
                    this.activeEdge = i;
                }


if(!this.activeNode.containsChild(this.string.charAt(this.activeEdge))) {
                    SuffixNode nn = createNode(i, i, Boolean.TRUE);

this.activeNode.addChild(this.string.charAt(this.activeEdge), nn);

                    if(this.lastNewNode != null) {
                        this.lastNewNode.setSuffixLink(activeNode);
                        this.lastNewNode = null;
                    }

                } else {
                    SuffixNode nn =
activeNode.getChild(this.string.charAt(this.activeEdge));

                    if(this.activeLength >= getLength(nn)) {
                        this.activeEdge = this.activeEdge + getLength(nn);
                        this.activeLength = this.activeLength -
getLength(nn);
                        this.activeNode = nn;
                        showstopper = Boolean.TRUE;

                    } else {
                        if(this.string.charAt(i) ==
this.string.charAt(nn.getStart() + this.activeLength)) {

                            if(this.lastNewNode != null && activeNode !=
this.root) {

this.lastNewNode.setSuffixLink(this.activeNode);
                                this.lastNewNode = null;
                            }
                            this.activeLength = this.activeLength + 1;
                            break;
                        }

                        SuffixNode newNode = createNode(i, i, Boolean.TRUE);
                        SuffixNode preSplit = createNode(nn.getStart(),
nn.getStart() + this.activeLength - 1, Boolean.FALSE);

this.activeNode.addChild(this.string.charAt(this.activeEdge), preSplit);
                        nn.setStart(nn.getStart() + this.activeLength);
                        preSplit.addChild(this.string.charAt(nn.getStart()),
nn);
                        preSplit.addChild(this.string.charAt(i), newNode);

                        if(this.lastNewNode != null) {
                            this.lastNewNode.setSuffixLink(preSplit);
                        }
```

```
                        this.lastNewNode = preSplit;
                    }
                }

                if(!showstopper) {

                    this.remaining = this.remaining - 1;

                    if(this.activeNode == this.root && this.activeLength > 0)
{
                        this.activeLength = this.activeLength - 1;
                        this.activeEdge = this.activeEdge + 1;
                    } else if(this.activeNode != this.root) {
                        this.activeNode = this.activeNode.getSuffixLink();
                    }
                }
            }
        }
    }

    public Integer getEnd(SuffixNode n){
        if(n.isLeaf()) return this.globalEnd;
        else return n.getEnd();
    }

    public Integer getLength(SuffixNode n) {
        return getEnd(n) - n.getStart() + 1;
    }

    private void markNodes(SuffixNode n) {
        ArrayList<SuffixNode> vertexes = new ArrayList<>();;

        Stack<SuffixNode> task = new Stack<>();
        task.push(n);
        while (!task.isEmpty()) {

            SuffixNode i = task.peek();
            task.pop();

            if (i.isLeaf()) {
                if (i.getStart() <= this.splitIndex) {
                    i.setType(Boolean.TRUE, Boolean.FALSE);
                } else {
                    i.setType(Boolean.FALSE, Boolean.TRUE);
                }
                continue;

            } else {

                vertexes.add(i);
                for (Character c : i.getChildren().keySet()) {
                    task.push(i.getChild(c));
                }

                markHelper(vertexes);

                continue;
            }
        }
        markHelper(vertexes);
    }

    public void markHelper(ArrayList<SuffixNode> n ){
```

```
        for (int k = 0; k < n.size(); k++) {
            Boolean[] types = new Boolean[2];
            types[0] = Boolean.FALSE;
            types[1] = Boolean.FALSE;
            for (Character c : n.get(k).getChildren().keySet()) {
                if(n.get(k).getChild(c).getType()[0]) types[0] =
Boolean.TRUE;
                if(n.get(k).getChild(c).getType()[1]) types[1] =
Boolean.TRUE;
            }
            n.get(k).setType(types[0], types[1]);
        }


    }

    public String getLongestCommonSubstring() {
        this.maxHeight = 0;
        this.substringStartIndex = 0;
        markNodes(this.root);
        findLCS(this.root);

        return this.string.substring(this.substringStartIndex,
this.substringStartIndex + this.maxHeight);
    }

    private void findLCS(SuffixNode n) {

        Stack<Struct> task = new Stack<>();
        task.push(new Struct(0, n));
        while (!task.isEmpty()) {

            Struct i = task.peek();
            task.pop();
            if (i.getN() == null) {
                continue;
            }

            if (!i.getN().isTypeA() || !i.getN().isTypeB()) {
                continue;
            } else {
                for (Character c : i.getN().getChildren().keySet()) {
                    SuffixNode nn = i.getN().getChild(c);
                    if (nn.isTypeA() && nn.isTypeB()) {
                        if (this.maxHeight < i.getH() + getLength(nn)) {
                            this.maxHeight = i.getH() + getLength(nn);
                            this.substringStartIndex = getEnd(nn) - i.getH()
- getLength(nn) + 1;
                        }
                        task.push(new Struct(i.getH() + getLength(nn), nn));
                    }
                }
            }
            continue;
        }
    }
}
```

Список литературы

1) Юрий Лифшиц. Построение суффиксного дерева за линейное время.
2) http://web.stanford.edu/~mjkay/gusfield.pdf
3) http://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-6/
4) https://www.cs.helsinki.fi/u/ukkonen/SuffixT1withFigs.pdf

ОТЗЫВ

ФИО Ященко Роман Александрович

Вид практики: Учебная практика

Тип практики: Практика по получению первичных профессиональных умений и навыков научно-исследовательской деятельности

Организация: Департамент математики, механики и компьютерных наук (ДММиКН) ИЕНиМ УрФУ

с 17.02.2020 по 31.05.2020

Студент Ященко Роман Александрович группы МЕН-282203 института ИЕНиМ за время прохождения практики осуществил следующие мероприятия

Для реализации алгоритма, сформулированного руководителем изучил необходимую литературу. Написал код программы.

В период практики студент Ященко Р.А.

(краткая характеристика уровня подготовки и отношения практиканта к работе)

проявил большую самостоятельность и получил необходимые для прохождения практики навыки в написании программ. В течение практики рекомендую уделять больше времени с руководителем

Оценка за практику Зачет, 90 баллов

«_____» _____ 20____ г.

РОП/Заведующий кафедры/Директор департамента (*любой из трех*) (подпись)___

Руководитель практики Вахрушев В.А.

2020-6-29 10:24