Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

Национальный Исследовательский Университет

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №5

по курсу «Дискретный анализ»

Студент:	Хренникова А. С.
Группа:	М8О-208Б-19
Преподаватель:	Капралов Н. С.
Подпись:	
Оценка:	
Дата:	

Москва, 2021

Лабораторная работа №5

Задача: 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).

Вариант: Найти в заранее известном тексте поступающие на вход образцы.

Формат входных данных: Текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.

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

Москва, 2021

1 Описание

Для текста строится суффиксное дерево алгоритмом Укконена, после по этому дереву производится поиск слов, подданых на вход.

Алгоритм, который изобрел Эско Укконен для построения суффиксного дерева за линейное время, вероятно, самый простой из таких алгоритмов. По сравнению с некоторыми другими алгоритмами он обладает свойством "интерактивности", которое иногда может оказаться полезным.

В алгоритме Укконена мы берем префиксы нашего текста и строим для них неявное дерево, с помощью последовательного добавления суффиксов от большего к меньшему. Сложность такого алгоритма достаточно большая, поэтому он оптимизируется. Одним из улучшений является добавление суффиксных ссылок. Теперь все суффиксы одного префикса добавляются за линейное время. Еще одной оптимизацией является сокращение памяти, то есть вместо хранения строк, хранятся указатели на эти них. Таким образом, листья и суффиксные ссылки добавляются за константное время, нет лишних прохождений по дереву. В итоге сложность является линейной.

2 Исходный код:

```
#include <iostream>
#include <algorithm>
#include <map>
#include <vector>
#include <string>

class TSuffTree {
public:
	TSuffTree(std::string& s);
	~TSuffTree();
	void Create();
	std::vector<int> TreeSearch(std::string& s);

private:
	struct Node {
		std::map<char, Node*> children;
		int left;
		int* right;
		Node* sufLink;
		int sufIndex;

		Node() :
			left(-2), sufIndex(-1)
		{
		}

		Node(int l, int* r) :
			left(l), right(r), sufIndex(-1)
		{
		}
	};

	void Clear(Node* node);
	void GetIndex(Node* n, int height);
	bool List(Node* n);
	int Length(Node* n);
	void Extensions(int pos);
	int Skip(Node* currNode);
	void RecSearch(Node* node, std::vector<int>& res);
	std::vector<int> Search(Node* node, std::string& s);

	Node* root;
	std::string str;
	int end;
	Node* active;
	Node* lastNew;
	int activeEdge;
	int activeLength;
	int remainingSuff;
};

TSuffTree::TSuffTree(std::string& s) {
	root = new Node();
	str = s;
```

```
	end = 0;
	activeEdge = -1;
	activeLength = 0;
	remainingSuff = 0;
	active = root;
	for (int i = 0; i < str.size(); i++) {
		Extensions(i);
	}
}

TSuffTree::~TSuffTree() {
	Clear(root);
}

void TSuffTree::Create() {
	GetIndex(root, 0);
}

std::vector<int> TSuffTree::TreeSearch(std::string& s) {
	return Search(root, s);
}

bool TSuffTree::List(Node* n) {
	if (n->sufIndex == -1) {
		return false;
	}
	else {
		return true;
	}
}

int TSuffTree::Length(Node* n) {
	return *(n->right) - (n->left) + 1;
}

int TSuffTree::Skip(Node* currNode) {
	if (activeLength >= Length(currNode)) {
		activeEdge += Length(currNode);
		activeLength -= Length(currNode);
		active = currNode;
		return 1;
	}
	return 0;
}

void TSuffTree::Extensions(int pos) {
	end = pos;
	remainingSuff++;
	lastNew = nullptr;
	while (remainingSuff > 0) {
		if (activeLength == 0) {
			activeEdge = pos;
		}
		if (active->children[str[activeEdge]] == 0) {
			Node* newNode = new Node(pos, &end);
			active->children[str[pos]] = newNode;
```

```
                        if (lastNew != nullptr) {
                                lastNew->sufLink = active;
                                lastNew = nullptr;
                        }
                } else {
                        Node* next = active->children[str[activeEdge]];
                        if (Skip(next)) {
                                continue;
                        }
                        if (str[next->left + activeLength] == str[pos]) {
                                if (lastNew != nullptr && active != root) {
                                        lastNew->sufLink = active;
                                        lastNew = nullptr;
                                }
                                activeLength++;
                                break;
                        }
                        int* splitEnd = new int;
                        *splitEnd = next->left + activeLength - 1;
                        Node* split = new Node(next->left, splitEnd);
                        split->sufLink = root;
                        active->children[str[next->left]] = split;
                        Node* list = new Node(pos, &end);
                        split->children[str[pos]] = list;
                        next->left += activeLength;
                        split->children[str[next->left]] = next;
                        if (lastNew != nullptr) {
                                lastNew->sufLink = split;
                        }
                        lastNew = split;
                }
                remainingSuff--;
                if (active == root && activeLength > 0) {
                        activeLength--;
                        activeEdge = pos - remainingSuff + 1;
                }
                else if (active != root) {
                        active = active->sufLink;
                }
        }
}

void TSuffTree::GetIndex(Node* n, int height) {
        int leaf = 1;
        for (auto it = n->children.begin(); it != n->children.end(); it++) {
                leaf = 0;
                GetIndex(it->second, height + Length(it->second));
        }
        if (leaf == 1) {
                n->sufIndex = str.size() - height;
        }
}

void TSuffTree::RecSearch(Node* node, std::vector<int>& res) {
        if (List(node)) {
```

```
                res.push_back(node->sufIndex + 1);
        }
        else {
                for (auto it = node->children.begin(); it != node->children.end(); it++) {
                        if (it->second) {
                                RecSearch(it->second, res);
                        }
                }
        }
}

std::vector<int> TSuffTree::Search(Node* node, std::string& s) {
        int i = 0;
        bool out = true;
        std::vector<int> res;
        while (out && node->children[s[i]]) {
                node = node->children[s[i]];
                for (int j = node->left; j <= *(node->right); j++, i++) {
                        if (i == s.size() || s[i] != str[j]) {
                                out = false;
                                break;
                        }
                }
        }
        if (i == s.size()) {
                RecSearch(node, res);
        }
        return res;
}

void TSuffTree::Clear(Node* node) {
        for (auto it = node->children.begin(); it != node->children.end(); it++) {
                if (it->second) {
                        Clear(it->second);
                }
        }
        if (node != root && !List(node)) {
                delete node->right;
        }
        delete node;
        return;
}

int main() {
        std::string s0, t;
        std::cin >> s0;
        s0 += "$";
        TSuffTree tree(s0);
        tree.Create();
        int ind = 1;
        while (std::cin >> t) {
                std::vector<int> answer;
                if (t.size() > 0 && t.size() < s0.size()) {
                        answer = tree.TreeSearch(t);
                }
                if (answer.size() > 0) {
```

```
                        std::cout << ind << ": ";
                        sort(answer.begin(), answer.end());
                        for (int i = 0; i < answer.size() - 1; i++) {
                                std::cout << answer[i] << ", ";
                        }
                        std::cout << answer[answer.size() - 1] << '\n';
                }
                ind++;
        }
        return 0;
}
```

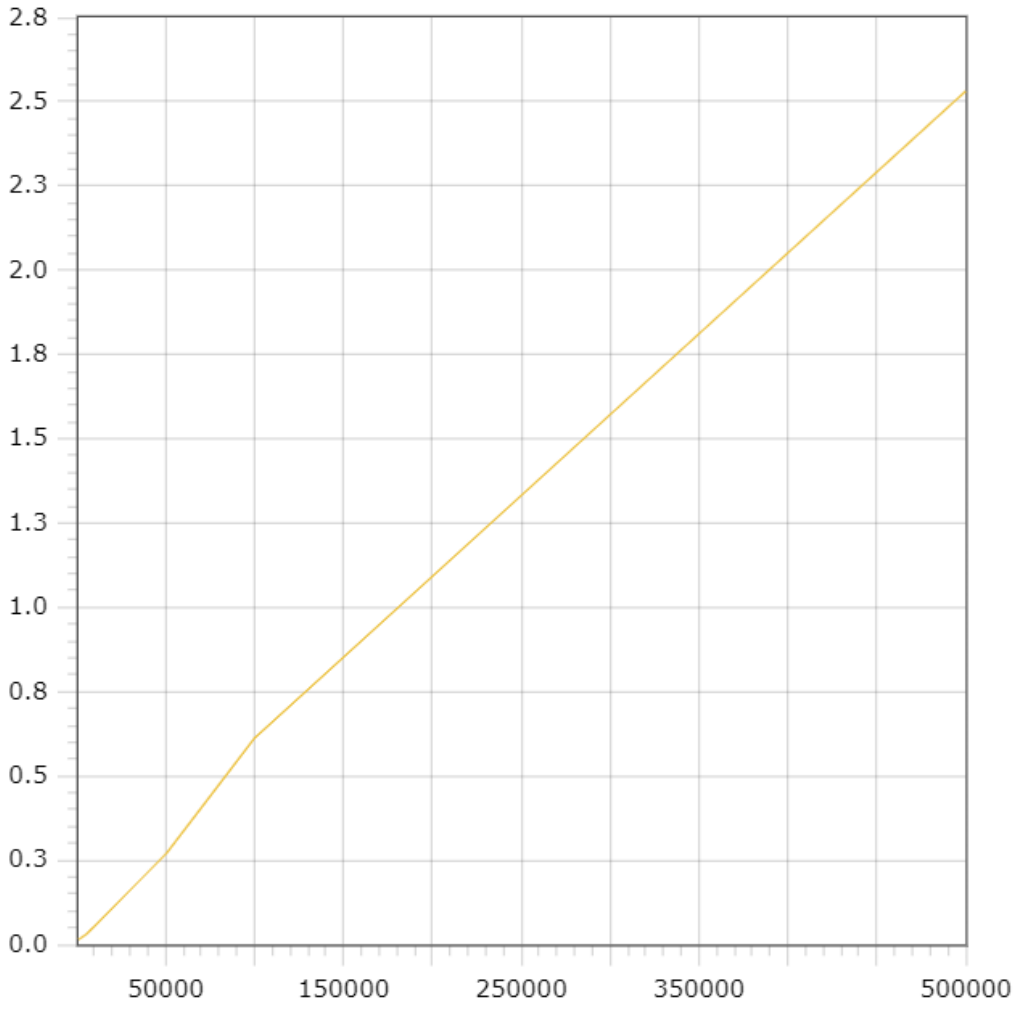
3 Консоль:

```
lina_tucha@LAPTOP-44CRFC1U:~/dir/da$ ./1122
abcdabc
abcd
1: 1
bcd
2: 2
bc
3: 2, 6
lina_tucha@LAPTOP-44CRFC1U:~/dir/da$ cat t100.txt
aabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaa
baabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaaba
abaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaab
aabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaaba
bcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabc
aabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaa
baabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaaba
abaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaab
aabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaaba
bcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabc
aabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabcaabaabaabc
aab
1: 1, 4, 7, 11, 14, 17, 21, 24, 27, 31, 34, 37, 41, 44, 47, 51, 54, 57, 61, 64, 67, 71, 74, 77, 81, 84, 87, 91,
94, 97, 101, 104, 107, 111, 114, 117, 121, 124, 127, 131, 134, 137, 141, 144, 147, 151, 154, 157, 161,
164, 167, 171, 174, 177, 181, 184, 187, 191, 194, 197, 201, 204, 207, 211, 214, 217, 221, 224, 227, 231,
234, 237, 241, 244, 247, 251, 254, 257, 261, 264, 267, 271, 274, 277, 281, 284, 287, 291, 294, 297, 301,
304, 307, 311, 314, 317, 321, 324, 327, 331, 334, 337, 341, 344, 347, 351, 354, 357, 361, 364, 367, 371,
374, 377, 381, 384, 387, 391, 394, 397, 401, 404, 407, 411, 414, 417, 421, 424, 427, 431, 434, 437, 441,
444, 447, 451, 454, 457, 461, 464, 467, 471, 474, 477, 481, 484, 487, 491, 494, 497, 501, 504, 507, 511,
514, 517, 521, 524, 527, 531, 534, 537, 541, 544, 547, 551, 554, 557, 561, 564, 567, 571, 574, 577, 581,
584, 587, 591, 594, 597, 601, 604, 607, 611, 614, 617, 621, 624, 627, 631, 634, 637, 641, 644, 647, 651,
654, 657, 661, 664, 667, 671, 674, 677, 681, 684, 687, 691, 694, 697, 701, 704, 707, 711, 714, 717, 721,
724, 727, 731, 734, 737, 741, 744, 747, 751, 754, 757, 761, 764, 767, 771, 774, 777, 781, 784, 787, 791,
794, 797, 801, 804, 807, 811, 814, 817, 821, 824, 827, 831, 834, 837, 841, 844, 847, 851, 854, 857, 861,
864, 867, 871, 874, 877, 881, 884, 887, 891, 894, 897, 901, 904, 907, 911, 914, 917, 921, 924, 927, 931,
934, 937, 941, 944, 947, 951, 954, 957, 961, 964, 967, 971, 974, 977, 981, 984, 987, 991, 994, 997
```

4 Тест производительности:

Тест представляет собой сгенерированный текст из символов алфавита и строку для поиска.

	Размер текста	Время
1	1000	0,014 с
2	5000	0,029 с
3	10000	0,054 с
4	50000	0,268 с
5	100000	0,612 с
6	500000	2,531 с



Москва, 2021

5 Выводы:

В ходе выполнения данной лабораторной работы я познакомилась с алгоритмом Укконена для построения суффиксного дерева. Этот алгоритм используется преимущественно тогда, когда известен лишь один текст и будет осуществляться поиск нескольких паттернов, когда длина текста много больше длин паттернов.

Москва, 2021