МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

Отчет

Практическая работа №10

Дисциплина Структуры и алгоритмы обработки данных

Тема. Поиск в тексте образца. Алгоритмы. Эффективность алгоритмов.

Выполнил студент Черномуров С. А.

Фамилия И.О.

Группа ИКБО-13-21

Номер группы

Москва 2022

Вариант №23

Задание 1

Разработать и реализовать алгоритм Кнута-Морриса-Пратта

1. Условие задачи:
   Дан пакет из n документов. Каждый документ = это текст протокола собрания коллектива. В протоколе есть фраза: Слушали сообщение: после которой через пробел следует фамилия и инициалы (записаны по формату: Иванов И.И.) выступившего. Сформировать массив данных по выступившим для каждого протокола.
2. Постановка задачи:
   Дано. Текст и искомая подстрока.
   Результат. Индекс начала искомой подстроки, если подстрока не найдена – -1.
3. Модель решения:
   Для нахождения образца в строке образец и строка склеиваются символом, заведомо не содержащемся ни в строке, ни в образце. Далее вызывается префикс-функция от строки, которая находит массив максимальных длин префиксов строки, которые одновременно являются суффиксами этой строки. Если значение массива совпадает с длиной искомой подстроки, то текущий номер(с поправкой на длину подстроки) указывает на номер последнего элемента искомой подстроки.
   Алгоритм Кнута-Морриса-Пратта позволяет находить префикс-функцию от строки за линейное время. Можно заметить, что префикс-функция от следующего элемента превосходит не более чем на единицу префикс-функцию от текущего элемента. Расчет префикс-функции текущего элемента производится следующим образом:
   1) Временной переменной присваивается значение префикс-функции предыдущего элемента(изначально массив длин префиксов обнулен).
   2) Пока временная переменная не обнулится, или текущий элемент строки не станет равен «предыдущему» (индекс которого – временная переменная, причем для этого элемента уже рассчитано значение префикс-функции(то есть, уже найдены совпадения)).
   3) В случае, если текущий элемент строки равен, «предыдущему», то значению префикс-функции от текущего элемента присваивается инкрементированное значение временной переменной (совпадение было найдено, и инкрементирование показывает, что этот элемент также является частью подстроки). Если же это условие не выполнилось, то значение префикс-функции от текущего элемента обнуляется (текущий символ не содержится в искомой подстроке)

Рассмотрим поэтапно заполнение массива префиксов для строки ab@acab, где ab – искомая подстрока, acab – строка, в которой проходит поиск подстроки. Зелёным обозначены уже рассчитанные значения префикс-функций, красным – те, которые рассчитываются в данный момент. Также заметим, что первое значение рассчитано изначально, и всегда равно нулю (следующий элемент считается на основе предыдущего):

a	b	@	a	c	a	b
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	2

Длина искомой подстроки=2, откуда следует, что лишь на последнем элементе был найден конец искомой подстроки, останется лишь получить индекс начала искомой подстроки.

Префиксом строки S называется подстрока строки S, первый символ которой совпадает с первым символом строки S.

Суффиксом строки S называется подстрока строки S, последний символ которой совпадает с последним символом строки S.

4. Декомпозиция:
   1) Вычисление массива длин префиксов строки:
   Предусловие. Строка s.
   Постусловие. Вектор длин префиксов строки s.

```
vector <int> prefix(string s){

	vector <int> pref_length(s.size(), 0);

	for (int i = 1; i < s.size(); i++) {
		int cur_len = pref_length[i - 1];

		while (cur_len > 0 && s[i] != s[cur_len])
			cur_len = pref_length[cur_len -   1];

		if (s[i] == s[cur_len]) pref_length[i] = cur_len + 1;
		else pref_length[i] = cur_len;
	}
	return pref_length;
}
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	s=”ab@acab”	Prefix={0,1,0,1,0,1,2}
2	s=”ab@fff”	Prefix={0,1,0,0,0,0}

2) **Поиск первого вхождения искомой подстроки по массиву префиксов:**

Предусловие. text – строка, содержащая текст, в которой будет производиться поиск образца, templ – строка, содержащая искомый образец.

Постусловие. Позиция первого вхождения исходного образца, если образец не найден - -1.

```
int search(string text, string templ) {

	vector <int> prefix_vec = prefix(templ + "@" + text);

	for (int i = templ.size() + 1; i < prefix_vec.size(); i++)
		if (prefix_vec[i] == templ.size()) return i - 2 * templ.size();

	return -1;
}
```

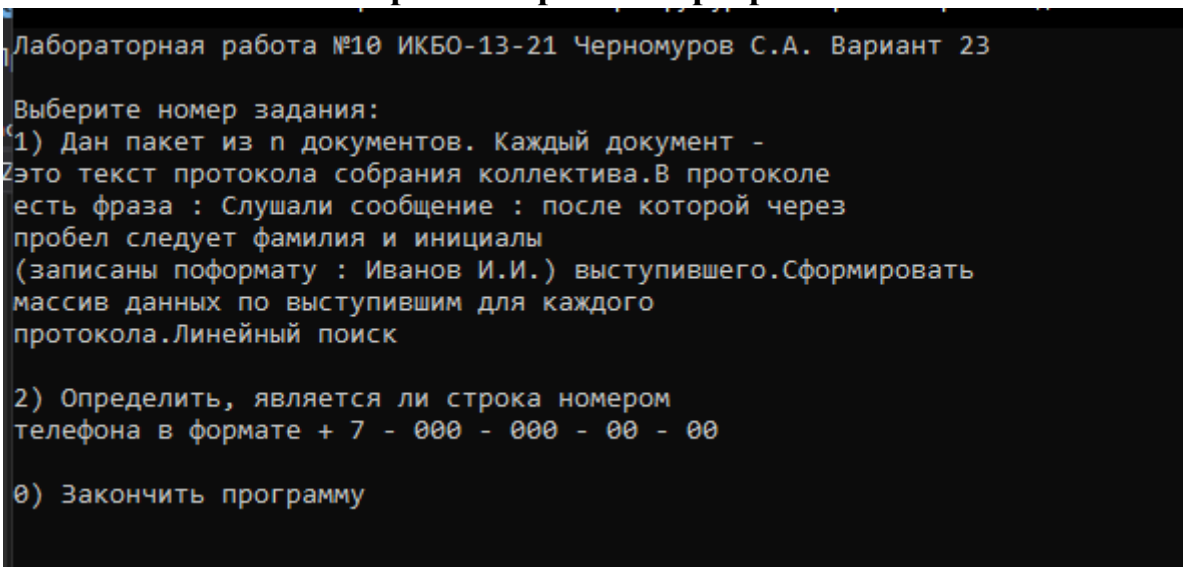
Тест функции:

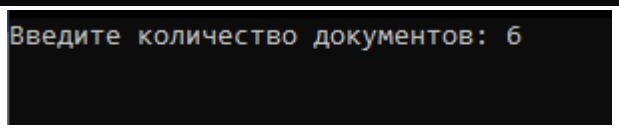
Номер теста	Исходные данные	Ожидаемый результат
1	text=”acab”, templ=”ab”	2
2	text=”fff”, templ=”ab”	-1

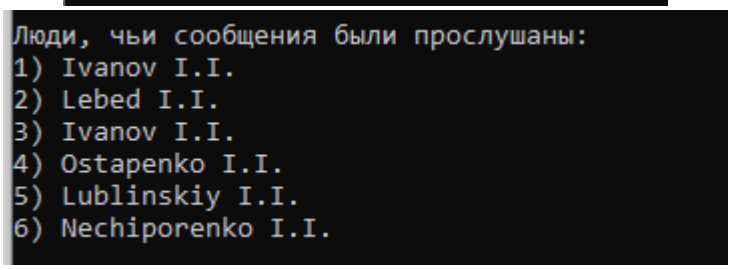
5. Эффективность работы алгоритма:

n	**T(n), наносек**	**$T_т$=f(C+M)**	**$T_п$=Cф+Mф**
100	38500	15000	430
1000	167400	150000	3134
10000	1457700	1500000	30172
100000	14257000	15000000	300554
1000000	143090400	150000000	3000256

Тестирование работы программы







Задание 2

Разработать алгоритм и функцию поиска образца в тексте с применением регулярных выражений для второй задачи варианта

1. Условие задачи:
   Определить, является ли строка номером телефона в формате +7-000-000-00-00
2. Регулярное выражение для проверки номера телефона:

   ```
   [\+][7][\-][0-9]{3}[\-][0-9]{3}[\-][0-9]{2}[\-][0-9]{2}
   ```

   \+ и - это специальные символы, перед ними пишется.
   [0-9] – диапазон цифр..
   {n} – количество повторений одного и того же диапазона.

3. Функция, реализующая проверку входной строки на соответствие регулярному выражению:
   Предусловие. text – проверяемая строка.
   Постусловие. True – если строка соответствует регулярному выражению, в противному случае false.

```
bool IsValidPhone(string text) {
	const regex exp("[\+][7][\-][0-9]{3}[\-][0-9]{3}[\-][0-9]{2}[\-][0-
9]{2}");

	return regex_match(text, exp);
}
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	+7-923-511-88-16	true
2	111-111-1-1-111	false

Тестирование работы программы

```
Лабораторная работа №10 ИКБО-13-21 Черномуров С.А. Вариант 23

Выберите номер задания:
1) Дан пакет из n документов. Каждый документ -
это текст протокола собрания коллектива.В протоколе
есть фраза : Слушали сообщение : после которой через
пробел следует фамилия и инициалы
(записаны поформату : Иванов И.И.) выступившего.Сформировать
массив данных по выступившим для каждого
протокола.Линейный поиск

2) Определить, является ли строка номером
телефона в формате + 7 - 000 - 000 - 00 - 00

0) Закончить программу

2
```

```
Введите строку, которую нужно проверить: +7-923-511-88-16

Строка является номером телефона
```

```
Введите строку, которую нужно проверить: 111-111-1-1-111

Строка не является номером телефона
```

Полный код программы на языке C++

Файл main.cpp (основной алгоритм программы):

```
#include "functions.h"
#include <iostream>
#include <regex>
#include <string>
#include <vector>
#include <fstream>
#include <chrono>
using namespace std;
using namespace chrono;

int main() {
	setlocale(LC_ALL, "");
	srand(time(NULL));
```

```
	cout << "Лабораторная работа №10 ИКБО-13-21 Черномуров С.А. Вариант 23"
<< endl << endl;
	cout << "Выберите номер задания:\n1) Дан пакет из n документов. Каждый
документ - \nэто текст протокола собрания коллектива.В протоколе\nесть фраза :
Слушали сообщение : после которой через\nпробел следует фамилия и
инициалы\n(записаны поформату : Иванов И.И.) выступившего.Сформировать\nмассив
данных по выступившим для каждого\nпротокола.Линейный поиск\n\n2) Определить,
является ли строка номером \nтелефона в формате + 7 - 000 - 000 - 00 - 00\n\n0)
Закончить программу\n\n";

	int choice1;

	do {
		cin >> choice1;

		if (choice1 != 1 && choice1 != 2  && choice1 != 0) cout << "Введено
неверное значение, попробуйте снова.\n";
	} while (choice1 != 1 && choice1 != 2 && choice1 != 0);

	system("cls");
	switch (choice1) {
	case 1: {
		cout << "Введите количество документов: ";
		int n;
		cin >> n;
		vector <string> people;
		int k = 0;
		int a=0;
		for (int i = 0; i < n; i++) {

			ifstream file("C:/Users/KLINY-ПК/Desktop/.data/Учёба/2
семестр/Структуры и алгоритмы обработки
данных/Zadanie10/Zadanie10/files/Protokol"+to_string(i+1)+".txt");
			if (file.is_open()) {
				string line;
				while (getline(file, line)) {
					string copy = line;
					//auto begin = steady_clock::now();
					int start_pos = search(line, "Slushali
soobschenie:");
					//auto end = steady_clock::now();
					//auto time = duration_cast<nanoseconds>(end -
begin);
					//k += time.count();
					copy.erase(0, start_pos);
					int end_pos = search(copy, ".") +
2+start_pos;// line.find(".", start_pos) + 2;
					if (start_pos!=-1 && end_pos!=-1 &&
end_pos>start_pos) people.push_back(line.substr(start_pos, end_pos -
start_pos+1));
				}
			}
			file.close();
		}
		//cout << k << " deistvii";
		cout << "\nЛюди, чьи сообщения были прослушаны:\n";
		for (int i = 0; i < people.size(); i++)
			cout << i + 1 << ") " << people[i].erase(0, 22) << endl;

		cout << "\n\n";
		break;
	}
	case 2:
	{
		cout << "Введите строку, которую нужно проверить: ";
```

```
                string s;
                cin >> s;
                if (IsValidPhone(s)) cout << "\nСтрока является номером
телефона\n\n";
                else cout << "\nСтрока не является номером телефона\n\n";
                break;
        }

        case 0:
                return 0;
        }

        main();
}
```

Файл functions.h (содержит функции):

```
#pragma once
#include <vector>
#include <iostream>
#include <string>
#include <regex>

using namespace std;

vector <int> prefix(string s){

        vector <int> pref_length(s.size(), 0);

        for (int i = 1; i < s.size(); i++) {
                int cur_len = pref_length[i - 1];
                //k++;
                while (cur_len > 0 && s[i] != s[cur_len])
                {
                        /*k++;*/ cur_len = pref_length[cur_len - 1];
                }

                if (s[i] == s[cur_len]) {pref_length[i] = cur_len + 1;
        }
                else { pref_length[i] = cur_len; }
        }
        return pref_length;
}

int search(string text, string templ) {
        vector <int> prefix_vec = prefix(templ + "@" + text);

        for (int i = templ.size() + 1; i < prefix_vec.size(); i++) {
                //k++;
                if (prefix_vec[i] == templ.size()) return i - 2 * templ.size();
        }
        return -1;
}

bool IsValidPhone(string text) {
        const regex exp("[\+][7][\-][0-9]{3}[\-][0-9]{3}[\-][0-9]{2}[\-][0-
9]{2}");

        return regex_match(text, exp);
}
```

Ответы на вопросы:

1. Что называют строкой?
   Строка – это набор символов. В языке программирования C++ обычно используются два типа строк:
   - Строки, являющиеся объектами строкового класса (строковый класс стандартной библиотеки C++)
   - C-strings (строки C-стиля)

2. Что называют префиксом строки?
   Префиксом строки S называется подстрока строки S, первый символ которой совпадает с первым символом строки S.

3. Что называют суффиксом строки?
   Суффиксом строки S называется подстрока строки S, последний символ которой совпадает с последним символом строки S.

4. Асимптотическая сложность последовательного поиска подстроки в строке?
   Асимптотическая сложность последовательного поиска подстроки в строке O(p*(n-p)), где p – длина образца, n – длина текста.
5. В чем особенность поиска образца алгоритмом Бойера –Мура?
   Важной особенностью алгоритма Бойера-Мура является то, что он выполняет сравнения в шаблоне справа налево в отличие от других алгоритмов.
6. Приведите асимптотическую сложность алгоритма Бойера –Мура поиска подстроки в строке по времени и памяти.
   Сложность по времени алгоритма Бойера-Мура O(n).
   Сложность по памяти алгоритма Бойера-Мура O(p+N), где p – размер образца, N – мощность алфавита.
7. Приведите пример входных данных для реализации эффективного метода прямого поиска подстроки в строке.
   Для алгоритма Кнута-Морриса-Пратта – “ab@absdbafgbsfasbgsc”.
8. Приведите пример строки, для которой поиск подстроки "aaabaaa" будет более эффективным, если делать его методом Кнута, Морриса и Пратта, чем, если делать его методом Бойера и Мура. И наоборот.
   КМП эффективнее работает при строке «aaabaaa».
   БМ эффективнее работает при строке «aaaaaa».
9. Объясните, как влияет размер таблицы кодов в алгоритме Бойера и Мура на скорость поиска.
   Размер таблицы кодов влияет на скорость поиска по двум причинам:

- Время, необходимое для выделения памяти под эту таблицу различно для различных кодировок/мощностей алфавита
- Вторую причину проще отобразить на картинке (56-57 строки). Чем длиннее массив BMT тем дольше его заполнять.

```
54    int  i, Pos;
55    int  BMT[12560];
56    for ( i = 0; i < 256; i ++ )
57      BMT[i] = ssl;
58    for ( i = ssl-1; i >= 0; i-- )
59      if ( BMT[(short)(substring[i])] == ssl )
60        BMT[(short)(substring[i])] = ssl - i - 1;
61    Pos = ssl - 1;
62    while ( Pos < sl )
63      if ( substring[ssl - 1] != string[Pos] )
64        Pos = Pos + BMT[(short)(string[Pos])];
65      else
66        for ( i = ssl - 2; i >= 0; i-- ){
67          if ( substring[i] != string[Pos - ssl + i + 1] ) {
68            Pos += BMT[(short)(string[Pos - ssl + i + 1])] - 1;
69            break;
70          }
71          else
72            if ( i == 0 )
73              return Pos - ssl + 1;
74          cout << "\t" << i << endl;
75        }
76    }
77  return res;
78  }
```

10.За счет чего в алгоритме Бойера и Мура поиск оптимален в большинстве случаев?

Преимущество алгоритма Бойера-Мура в том, что ценой некоторого количества предварительных вычислений над шаблоном, шаблон сравнивается с исходным текстом не во всех позициях – часть проверок пропускается как заведомо не дающие результата.

11.Поясните влияние префикс-функции в алгоритме Кнута, Морриса и Пратта (КМП) на организацию поиска подстроки в строке.

Префикс-функция напрямую влияет на организацию поиска подстроки в строке, так как поиск нужного индекса производится по массиву длин префиксов, сгенерированному префикс-функцией.

12.Приведите пример префикс-функции для поиска образца в тексте для алгоритма КМП.

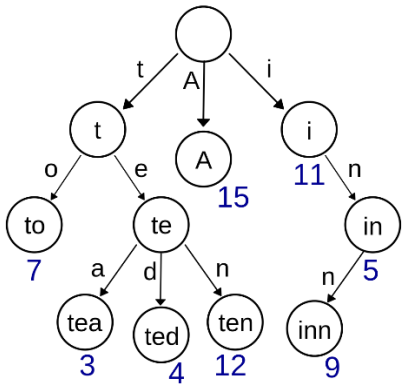
```
vector <int> prefix(string s){

      vector <int> pref_length(s.size(), 0);
```

```
	for (int i = 1; i < s.size(); i++) {
		int cur_len = pref_length[i - 1];
		//k++;
		while (cur_len > 0 && s[i] != s[cur_len])
		{
			/*k++;*/ cur_len = pref_length[cur_len - 1];
		}

		if (s[i] == s[cur_len]) {pref_length[i] = cur_len + 1;
	}
		else { pref_length[i] = cur_len; }
	}
	return pref_length;
}
```

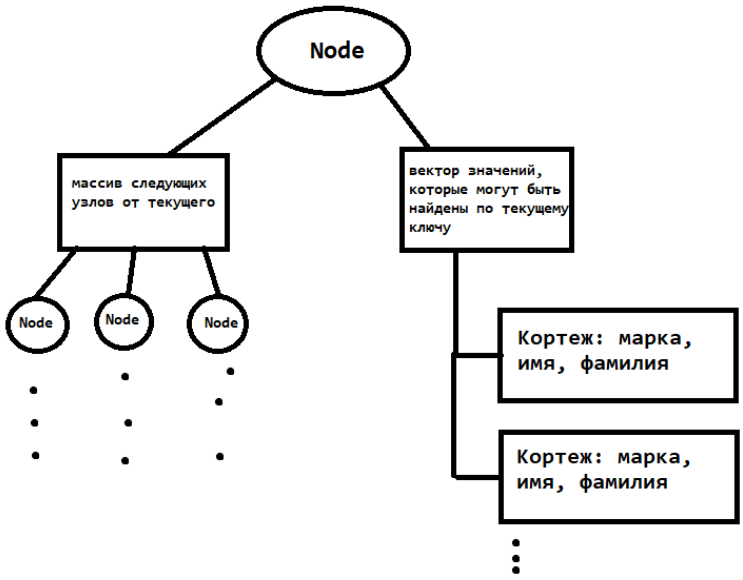
13.В чем особенность поиска образца алгоритмом Рабина и Карпа?
Алгоритм имеет уникальную особенность находить любую из заданных *k* строк одинаковой длины в среднем (при правильном выборе хеш-функции) за время O(*n*) независимо от размера *k*.
14.Приведите асимптотическую сложность алгоритма Рабина и Карпа поиска подстроки в строке.
Асимптотическая сложность алгоритма Рабина-Карпа O(p*n), где p – длина образца, n – длина текста.
15.Что такое бор?
Бор – структура данных для хранения набора строк, представляющая из себя подвешенное дерево с символами на ребрах. Строки получаются последовательной записью всех символов, хранящихся на ребрах между корнем бора и терминальной вершиной.
16.Какие структуры хранения данных используются для реализации простого бора?
Для реализации простого бора можно использовать многосвязный список или ассоциативный массив.
17.Приведите пример бора и реализуйте его одним из способов. Объясните алгоритм поиска образца с использованием бора.



Поиск по бору осуществляется проходом по бору от корня по символам слова. Если в процессе прохода алгоритм пришел в

несуществующий узел бора, то слова нет, если не пришел – то есть.

1. Структура узла бора:



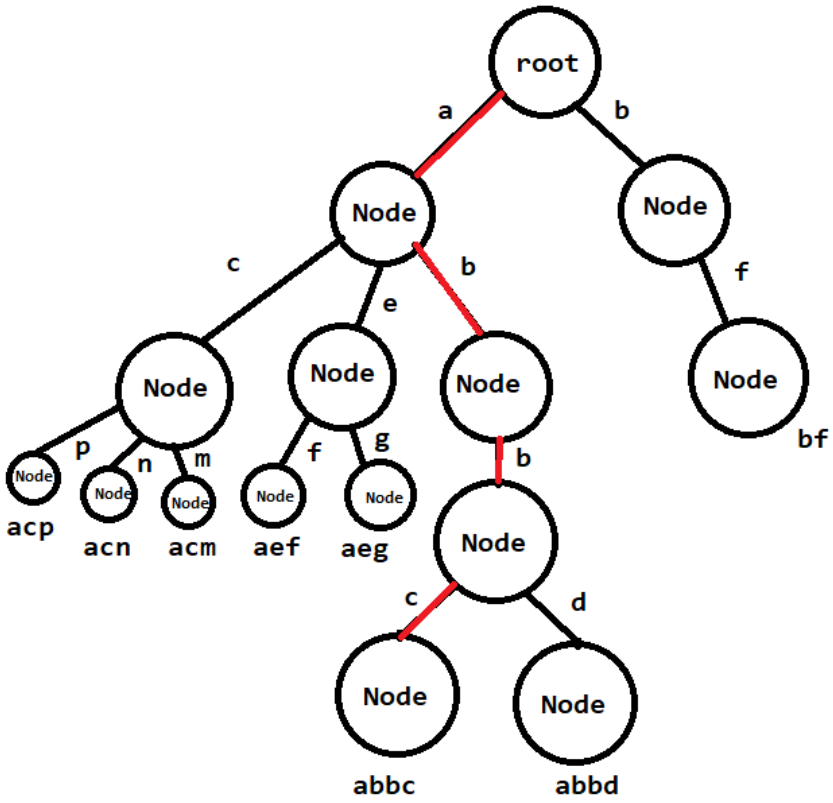

```
struct Node {
	Node* next[1000];

	vector <tuple<string, string, string>> info;

	Node() {
		for (int i = 0; i < 100; i++)
			next[i] = nullptr;

	}
};
```

2. Тестовый пример:
   Нужные данные хранятся в боре, строка-ключ к ним – “abbc”. Чтобы получить данные по этому ключу, нужно попасть в узел, соответствующий этому ключу. Пройдем по бору по соответствующим ребрам(каждый проход по ребру соответствует одному символу в строке-ключе):



Проход по соответствующим ребрам бора показан красными линиями. Таким образом, мы попали в узел, доступный по ключу “abbc”, и теперь можем смотреть или изменять доступные по нему данные.

3. **Поиск по бору:**

   Предусловие. s – строка-ключ, root-указатель на корневой узел бора.
   Постусловие. Ссылка на найденный по ключу узел current_vertex, в противном случае nullptr.

```
Node* is_found(string s, Node* root);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	s=”abbc”, root=указатель на корень бора, содержащего ключ “abbc”	current_vertex от ключа abbc
2	s=”abbc”, root=указатель на корень бора, НЕ содержащего ключ “abbc”	nullptr

```
Node* is_found(string s, Node* root) {
	Node* current_vertex = root;
	for (int i = 0; i < s.size(); i++) {
		current_vertex = current_vertex->next[s[i] - '0'];
		if (current_vertex == nullptr) return nullptr;
	}
	return current_vertex;
}
```

Вывод

В ходе работы выполнено задание в соответствии с поставленным вариантом. Также были получены знания и практические навыки по работе алгоритмов поиска подстрок в тексте.