Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа
«Алгоритмы Бойлера Мура и Кнута Морриса Прата»

Выполнил:
студент первого курса
ЭТФ группы РИС-23-3б
Акбашева Софья Руслановна

Проверила:
Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Алгоритмы Бойлера Мура и Кнута Морриса Прата

Цель: знакомство с простыми методами поиска, Бойлера Мура и Кнута Морриса Прата.

Постановка задачи: реализовать поиск заданного элемента, с методов Бойлера Мура и Кнута Морриса Прата.

Анализ задачи

1) В цикле do-while пользователю предлагается выбрать способ поиска: алгоритм Бойлера-Мура или алгоритм Кнута-Мориса-Прата. Ввод пользователя проверяется на корректность: выбор должен быть в диапазоне от 1 до 2.

2) Пользователь вводит две строки: строку, в которой будет производиться поиск, и подстроку, которую нужно найти. Для ввода строк используется функция getline, которая позволяет вводить строки с пробелами.

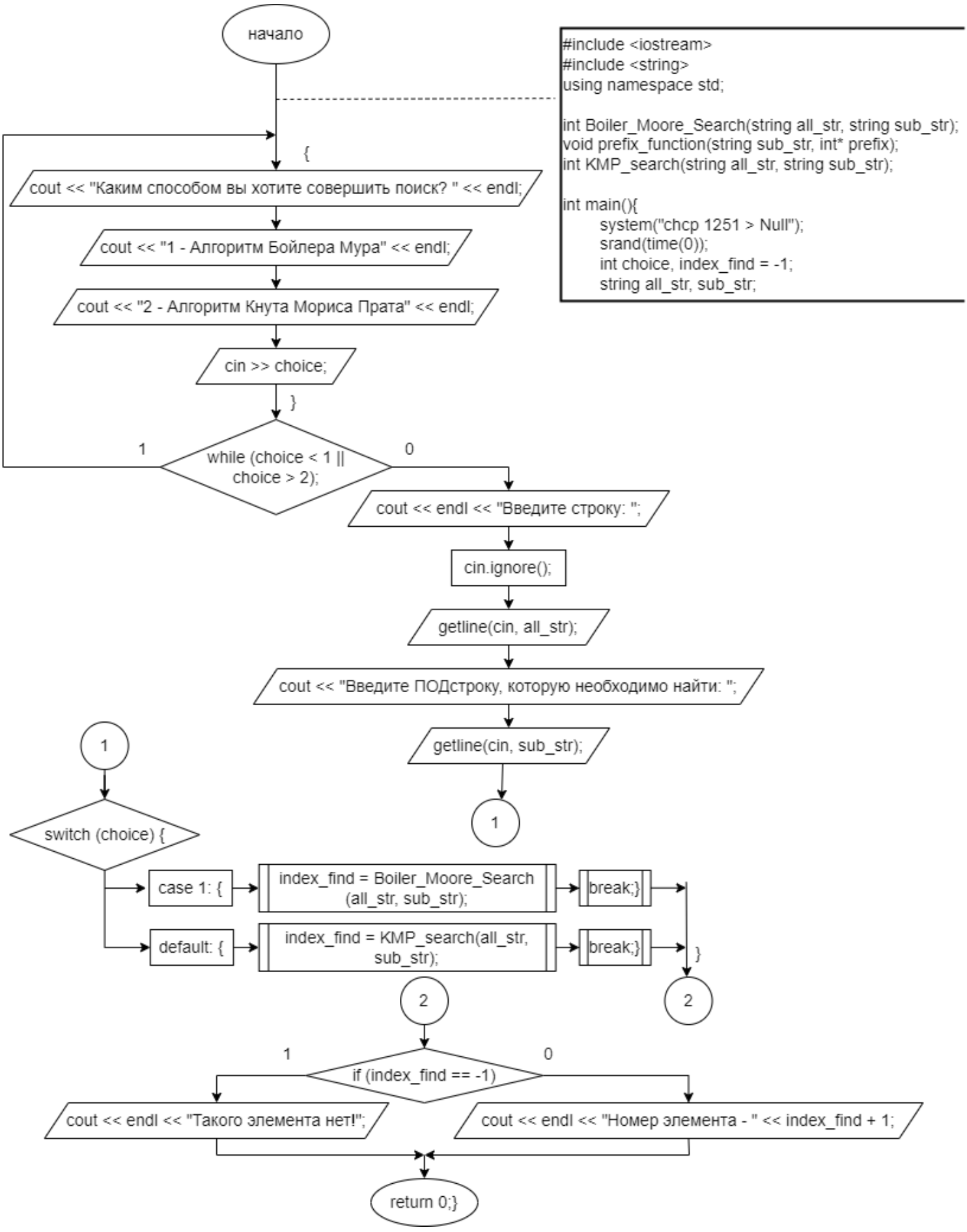
3) Если подстрока не найдена, выводится сообщение "Такого элемента нет!". Если подстрока найдена, выводится сообщение "Номер элемента - " и номер позиции, на которой найдена подстрока.

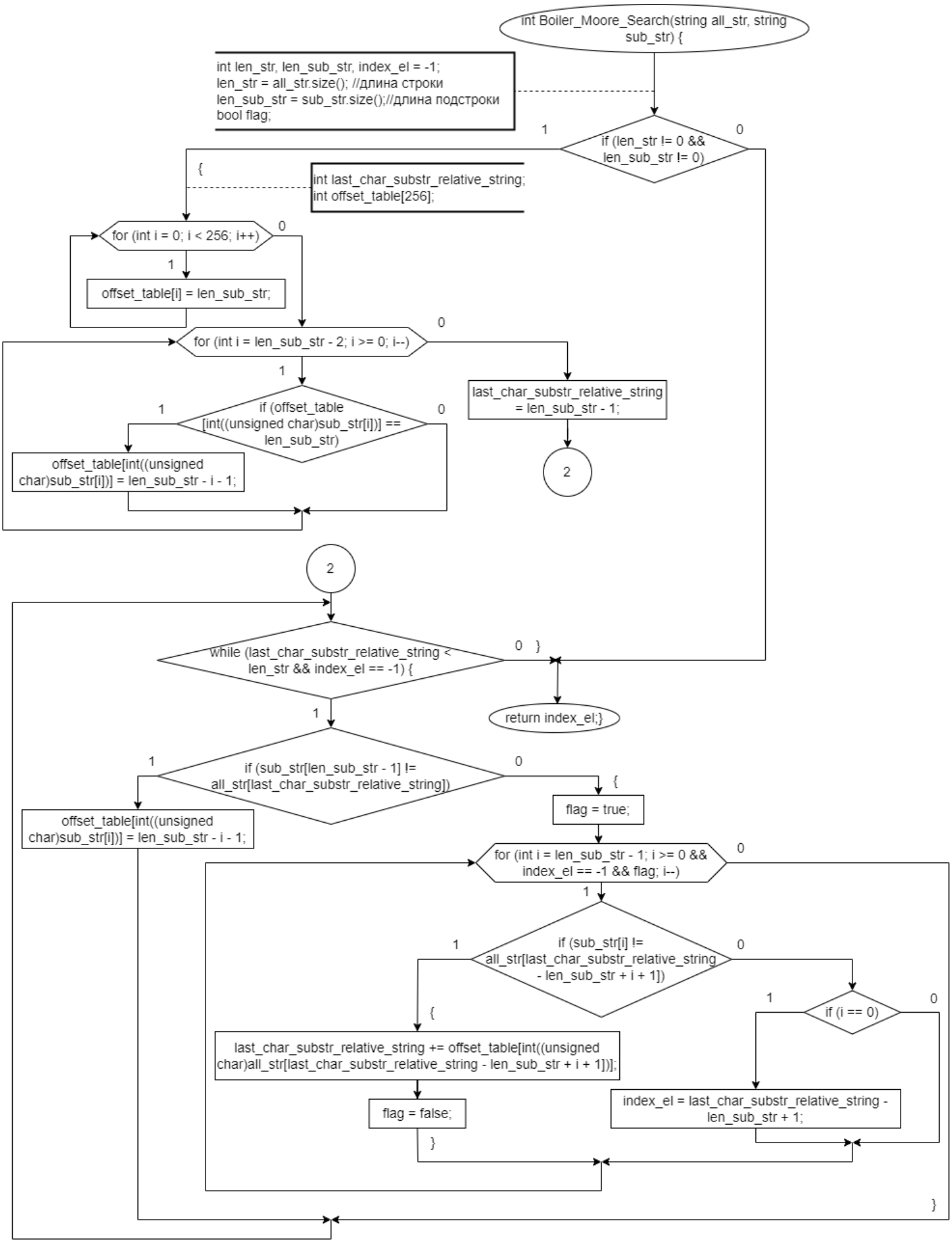
4) Алгоритм Бойлера-Мура. В начале функции Boiler_Moore_Search определяются переменные для хранения длины строки и подстроки, а также для хранения результата поиска. Затем проверяется, что строка и подстрока не пусты. Если это так, то начинается основная часть алгоритма. Сначала создается массив смещений offset_table, который будет использоваться для ускорения поиска. Каждый элемент массива инициализируется значением len_sub_str, что означает, что все символы подстроки считаются не найденными. Затем для каждого символа подстроки, который присутствует в подстроке, в массиве offset_table обновляется соответствующий элемент, чтобы указать смещение от конца подстроки до следующего вхождения этого символа. Далее определяется переменная last_char_substr_relative_string, которая указывает позицию последнего символа подстроки относительно строки. Затем начинается цикл, который продолжается до тех пор, пока

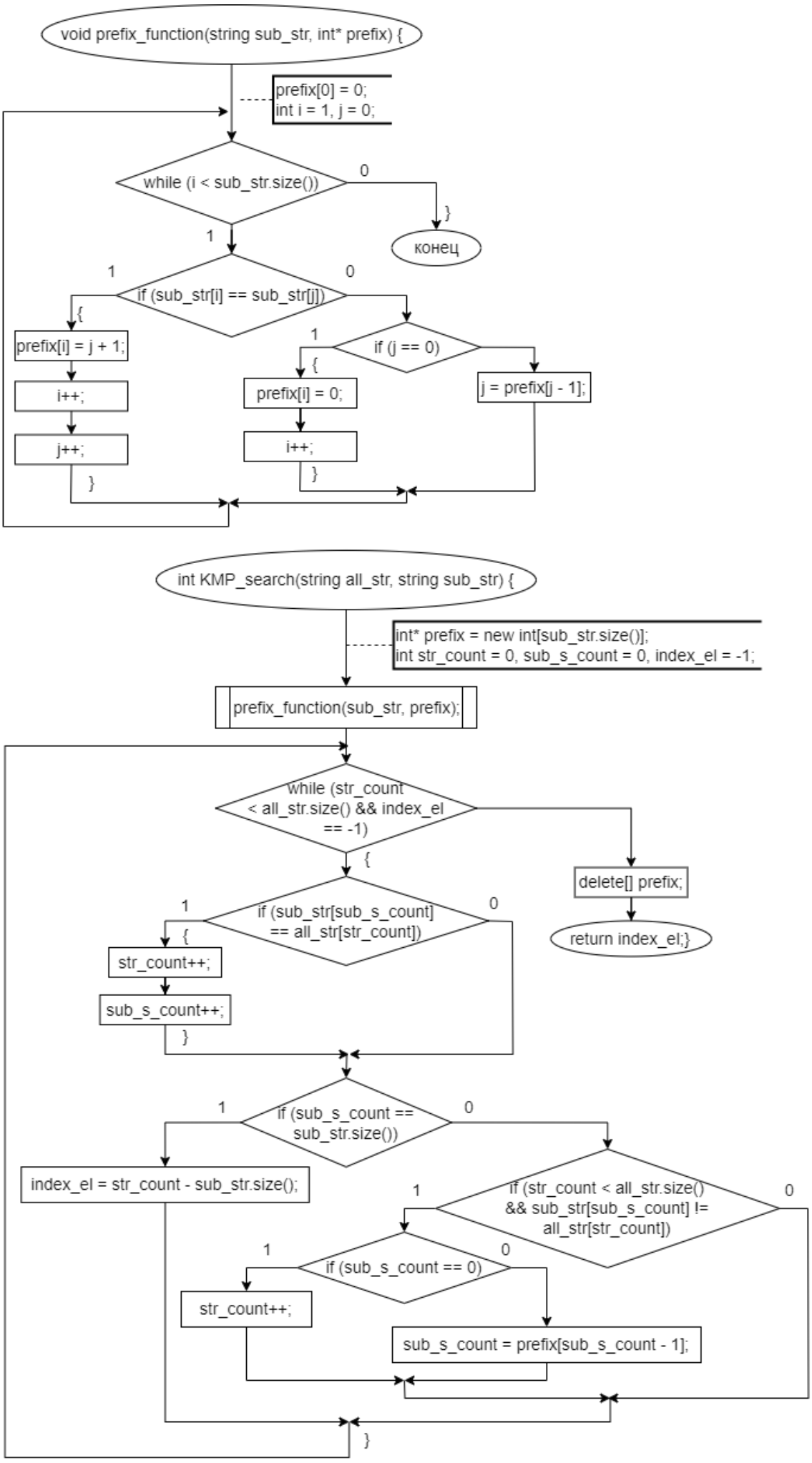
last_char_substr_relative_string не достигнет конца строки. Внутри цикла проверяется, совпадает ли последний символ подстроки с символом строки на позиции last_char_substr_relative_string. Если это так, то начинается проход по подстроке с правого края. Если символ подстроки не совпадает с символом строки, то подстрока смещается на соответствующее смещение в массиве offset_table. Если все символы подстроки совпадают с символами строки, то возвращается индекс начала подстроки в строке. В конце функции возвращается значение переменной index_el, которое является результатом поиска. Если подстрока не найдена, возвращается -1.

5) В функции prefix_function создается массив prefix, который будет хранить значения префикс-функции для подстроки sub_str. Затем выполняется итерация по каждому символу подстроки, начиная с первого. Если текущий символ подстроки совпадает с предыдущим символом подстроки, то значение префикс-функции увеличивается на 1. Если текущий символ подстроки не совпадает с предыдущим символом подстроки, то значение префикс-функции устанавливается в 0. Функция KMP_search использует префикс-функцию для поиска подстроки sub_str в строке all_str. Она итерируется по каждому символу строки all_str и сравнивает его с текущим символом подстроки sub_str. Если символы совпадают, то счетчик символов строки увеличивается, а счетчик символов подстроки увеличивается на 1. Если все символы подстроки найдены в строке, то возвращается индекс первого символа подстроки в строке. Если подстрока не найдена, то используется значение префикс-функции для определения, с какого символа подстроки следует продолжить поиск. В конце функции KMP_search освобождается память, выделенная для массива prefix. В конце функции возвращается значение переменной index_el, которое является результатом поиска. Если подстрока не найдена, возвращается -1.

Блок схема


начало
#include <iostream>
#include <string>
using namespace std;
int Boiler_Moore_Search(string all_str, string sub_str);
void prefix_function(string sub_str, int* prefix);
int KMP_search(string all_str, string sub_str);
int main(){
system("chcp 1251 > Null");
srand(time(0));
int choice, index_find = -1;
string all_str, sub_str;
{
cout << "Каким способом вы хотите совершить поиск? " << endl;
cout << "1 - Алгоритм Бойлера Мура" << endl;
cout << "2 - Алгоритм Кнута Мориса Прата" << endl;
cin >> choice;
}
while (choice < 1 || choice > 2);
1
0
cout << endl << "Введите строку: ";
cin.ignore();
getline(cin, all_str);
cout << "Введите ПОДстроку, которую необходимо найти: ";
getline(cin, sub_str);
1
1
switch (choice) {
case 1: {
index_find = Boiler_Moore_Search (all_str, sub_str);
break;}
default: {
index_find = KMP_search(all_str, sub_str);
break;}
}
2
2
if (index_find == -1)
1
0
cout << endl << "Такого элемента нет!";
cout << endl << "Номер элемента - " << index_find + 1;
return 0;}


int Boiler_Moore_Search(string all_str, string sub_str) {
int len_str, len_sub_str, index_el = -1;
len_str = all_str.size(); //длина строки
len_sub_str = sub_str.size();//длина подстроки
bool flag;
if (len_str != 0 && len_sub_str != 0)
1
0
{
int last_char_substr_relative_string;
int offset_table[256];
for (int i = 0; i < 256; i++)
0
1
offset_table[i] = len_sub_str;
for (int i = len_sub_str - 2; i >= 0; i--)
0
1
if (offset_table [int((unsigned char)sub_str[i])] == len_sub_str)
1
0
offset_table[int((unsigned char)sub_str[i])] = len_sub_str - i - 1;
last_char_substr_relative_string = len_sub_str - 1;
2
2
while (last_char_substr_relative_string < len_str && index_el == -1) {
0 }
1
return index_el;}
if (sub_str[len_sub_str - 1] != all_str[last_char_substr_relative_string])
1
0
offset_table[int((unsigned char)sub_str[i])] = len_sub_str - i - 1;
{
flag = true;
for (int i = len_sub_str - 1; i >= 0 && index_el == -1 && flag; i--)
0
1
if (sub_str[i] != all_str[last_char_substr_relative_string - len_sub_str + i + 1])
1
0
{
last_char_substr_relative_string += offset_table[int((unsigned char)all_str[last_char_substr_relative_string - len_sub_str + i + 1])];
flag = false;
}
if (i == 0)
1
0
index_el = last_char_substr_relative_string - len_sub_str + 1;
}


void prefix_function(string sub_str, int* prefix) {
prefix[0] = 0;
int i = 1, j = 0;
while (i < sub_str.size())
0
}
конец
1
1
if (sub_str[i] == sub_str[j])
0
{
prefix[i] = j + 1;
i++;
j++;
}
1
if (j == 0)
{
prefix[i] = 0;
i++;
}
j = prefix[j - 1];
int KMP_search(string all_str, string sub_str) {
int* prefix = new int[sub_str.size()];
int str_count = 0, sub_s_count = 0, index_el = -1;
prefix_function(sub_str, prefix);
while (str_count
< all_str.size() && index_el
== -1)
{
delete[] prefix;
return index_el;}
1
if (sub_str[sub_s_count]
== all_str[str_count])
0
{
str_count++;
sub_s_count++;
}
1
if (sub_s_count ==
sub_str.size())
0
index_el = str_count - sub_str.size();
1
if (str_count < all_str.size()
&& sub_str[sub_s_count] !=
all_str[str_count])
0
1
if (sub_s_count == 0)
0
str_count++;
sub_s_count = prefix[sub_s_count - 1];
}

Код

```
#include <iostream>
#include <string>
using namespace std;

int Boiler_Moore_Search(string all_str, string sub_str);//поиск Бойлера Мура
void prefix_function(string sub_str, int* prefix); //префикс функция
int KMP_search(string all_str, string sub_str); //поиск Кнута Морриса Прата

int main(){
	system("chcp 1251 > Null");
	srand(time(0));
	int choice, index_find = -1;
	string all_str, sub_str;

	do { //выбор поиска
		cout << "Каким способом вы хотите совершить поиск? " << endl;
		cout << "1 - Алгоритм Бойлера Мура" << endl;
		cout << "2 - Алгоритм Кнута Мориса Прата" << endl;
		cin >> choice;
	} while (choice < 1 || choice > 2);

	cout << endl << "Введите строку: ";
	cin.ignore();
	getline(cin, all_str);
	cout << "Введите ПОДстроку, которую необходимо найти: ";
	getline(cin, sub_str);

	switch (choice) { //выбор поиска
	case 1: {
		index_find = Boiler_Moore_Search(all_str, sub_str);
		break;
	}
	default: {
		index_find = KMP_search(all_str, sub_str);
		break;
	}
	}

	if (index_find == -1) {
		cout << endl << "Такого элемента нет!";
	}
	else {
		cout << endl << "Номер элемента - " << index_find + 1;
	}

	return 0;
}

int Boiler_Moore_Search(string all_str, string sub_str) {//поиск Бойлера Мура
	int len_str, len_sub_str, index_el = -1;
	len_str = all_str.size(); //длина строки
	len_sub_str = sub_str.size();//длина подстроки
	bool flag;

	if (len_str != 0 && len_sub_str != 0) { //если строка и подстрока
существуют
		int last_char_substr_relative_string;
		int offset_table[256];//массив смещения по таблице ASCII из всех
символов таблицы
		for (int i = 0; i < 256; i++) { //заполнение полей символов
случайными элементами
			offset_table[i] = len_sub_str;
		}
		for (int i = len_sub_str - 2; i >= 0; i--) { //генерация таблицы
смещения для элементов, которые есть в подстроке
```

```
                    //substr = char (-128 до 127), приводим к unsigned char (от 0
до 255) и приводим к типу int согласно таблице ASCII, чтобы получить нужный индекс
                    if (offset_table[int((unsigned char)sub_str[i])] ==
len_sub_str) { //если, смещение элемента подстроки равно длине подстроки
                        offset_table[int((unsigned char)sub_str[i])] =
len_sub_str - i - 1;//шаг смещения - "длина подстроки" - "расстояние от конца" -
"перевод длины в индекс"
                    }
                }
                last_char_substr_relative_string = len_sub_str - 1;//позиция
последнего символа подстроки относительно строки
                while (last_char_substr_relative_string < len_str && index_el == -1)
{//пока не дойдем до конца строки
                    if (sub_str[len_sub_str - 1] !=
all_str[last_char_substr_relative_string]) {//если последний элемент подстроки НЕ
равен элементу строки на позиции last_char_substr_relative_string
                        last_char_substr_relative_string +=
offset_table[int((unsigned
char)all_str[last_char_substr_relative_string])];//смещаем подстроку, пользуясь
таблицей смещения
                    }
                    else {//если последний элемент подстроки РАВЕН элементу строки
на позиции last_char_substr_relative_string
                        flag = true;
                        for (int i = len_sub_str - 1; i >= 0 && index_el == -1 &&
flag; i--) {//проход по подстроке с правого края
                            if (sub_str[i] !=
all_str[last_char_substr_relative_string - len_sub_str + i + 1]) {//если к-л
элемент не равен - смещаем по таблице последнего символа строки
                                last_char_substr_relative_string +=
offset_table[int((unsigned char)all_str[last_char_substr_relative_string -
len_sub_str + i + 1])];//смещение по строке
                                flag = false; //надо сместить подстроку по
строке
                            }
                            else if (i == 0) {//если все равны то возвращаем
индекс начала подстроки в строке
                                index_el = last_char_substr_relative_string
- len_sub_str + 1;//вычисление индекса
                            }
                        }
                    }
                }
            }
            return index_el; //возврат индекса
}

void prefix_function(string sub_str, int* prefix) { //префикс функция
            prefix[0] = 0;
            int i = 1, j = 0;
            while (i < sub_str.size()) {//иду по ПОДстроке
                if (sub_str[i] == sub_str[j]) {//если в подстроке найдены совпадающие
элементы
                    prefix[i] = j + 1;
                    i++;
                    j++;
                }
                else {
                    if (j == 0) {//если на первом эл подстроки
                        prefix[i] = 0;
                        i++;
                    }
                    else {
                        j = prefix[j - 1];
                    }
                }
            }
```

```
}

int KMP_search(string all_str, string sub_str) { //поиск Кнута Морриса Прата
	int* prefix = new int[sub_str.size()]; //префикс
	int str_count = 0, sub_s_count = 0, index_el = -1;
	prefix_function(sub_str, prefix); //префикс функция

	while (str_count < all_str.size() && index_el == -1) {//пока не пройду всю строку или не найду индекс
		if (sub_str[sub_s_count] == all_str[str_count]) {//если совпал эл из строки и подстроки
			str_count++;//увеличиваю счетчик по строке
			sub_s_count++;//увеличиваю счетчик по подстроке
		}
		if (sub_s_count == sub_str.size()) {//если найдена вся подстрока
			index_el = str_count - sub_str.size();
		}
		else if (str_count < all_str.size() && sub_str[sub_s_count] != all_str[str_count]) {//если НЕ найдена вся подстрока
			if (sub_s_count == 0) {//если проверяю с первого символа подстроки
				str_count++;//увеличиваю счетчик по строке
			}
			else {//если проверяю НЕ с первого символа подстроки
				sub_s_count = prefix[sub_s_count - 1];
			}
		}
	}
	delete[] prefix;//освобождаю память
	return index_el;//вощзвращаю индекс
}
```

Результат работы

1) Алгоритм Бойлера Мура

```
Каким способом вы хотите совершить поиск?
1 - Алгоритм Бойлера Мура
2 - Алгоритм Кнута Мориса Прата
1

Введите строку: shreg frgtsrgtjuergt f
Введите ПОДстроку, которую необходимо найти: rgts

Номер элемента - 8
```

2) Алгоритм Кнута Морриса Прата

```
Каким способом вы хотите совершить поиск?
1 - Алгоритм Бойлера Мура
2 - Алгоритм Кнута Мориса Прата
2

Введите строку: ehge ff wegf egfyfj
Введите ПОДстроку, которую необходимо найти: ff

Номер элемента - 6
```

Вывод

В ходе работы я применила знания о работе различных методов поиска, в числе которых алгоритмы Бойлера Мура и Кнута Морриса Прата. Мне

удалось реализовать поставленную задачу: были созданы все необходимые методы поиска.

GitHub

Ссылка: https://github.com/SonyAkb/sorting-and-searching.git