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

ЛАБОРАТОРНАЯ РАБОТА

ПОИСКИ

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**Постановка задачи**

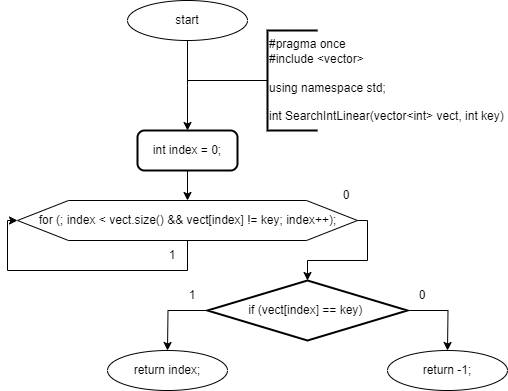
Реализовать алгоритмы поиска элементов.

**Анализ задачи**

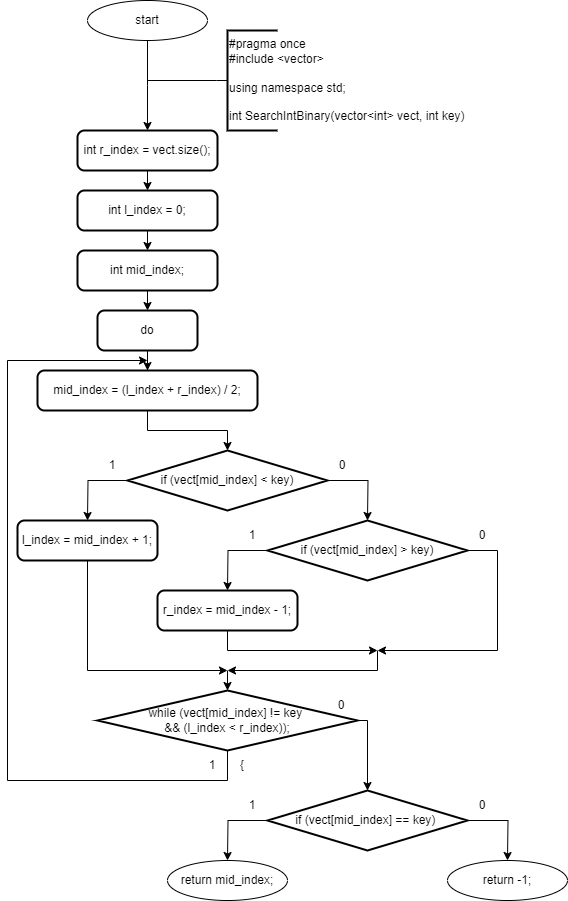
Алгоритмы для поиска целочисленных значений: линейный, бинарный, интерполяционный.

Алгоритмы для поиска подстроки в строке: линейный, Бойера-Мура, Кнута-Мориса-Пратта.

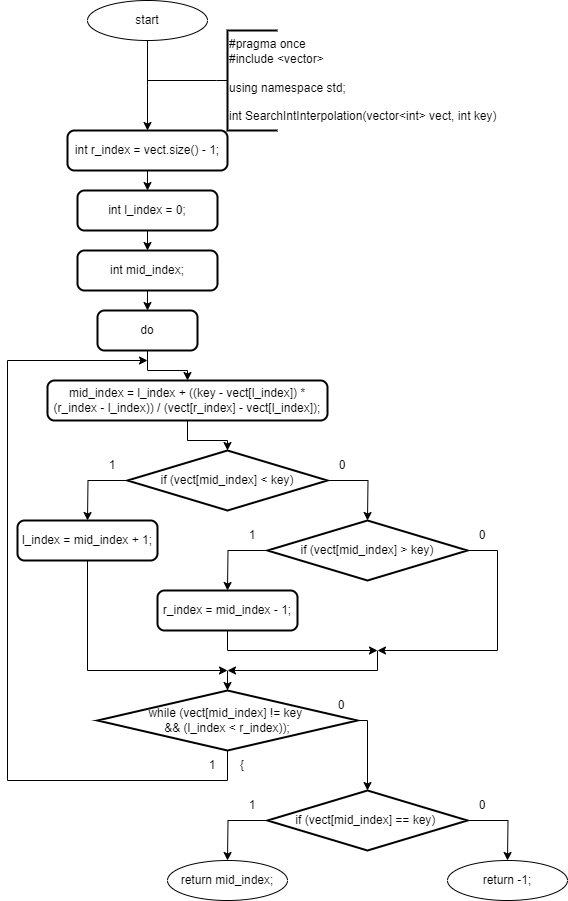
**Блок-схема**



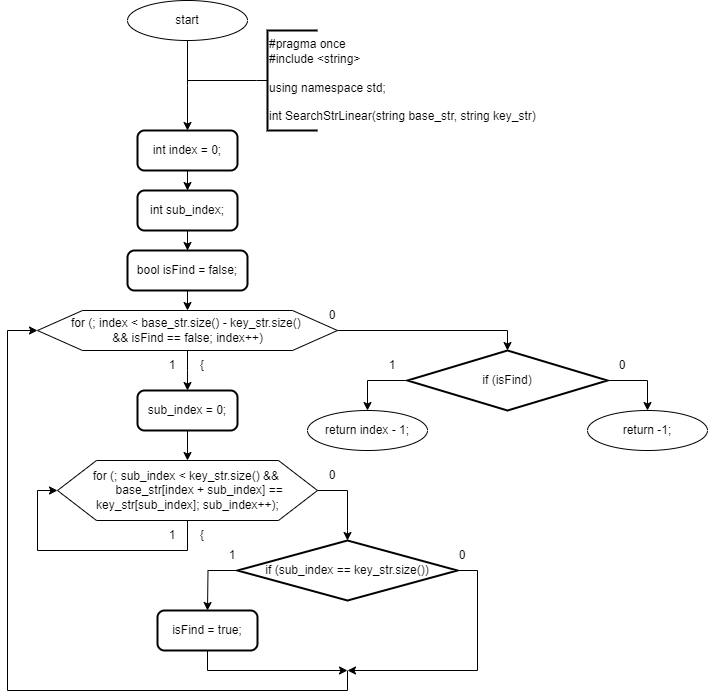
Функция прямого поиска



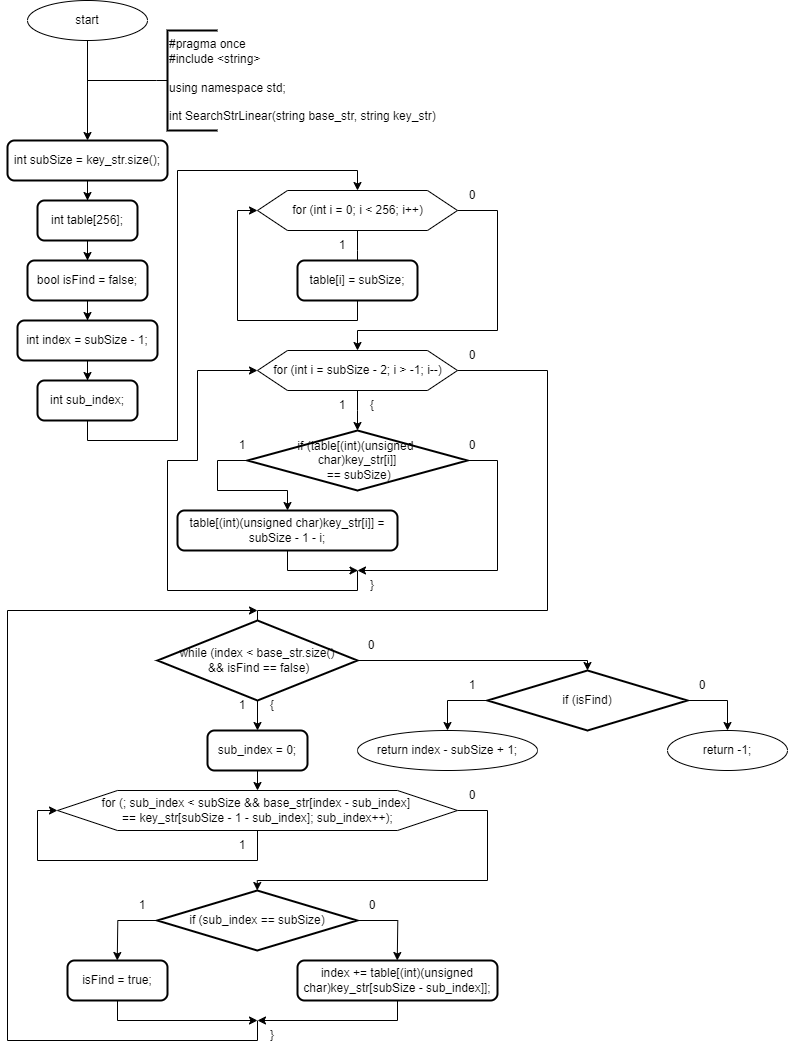
Функция бинарного поиска



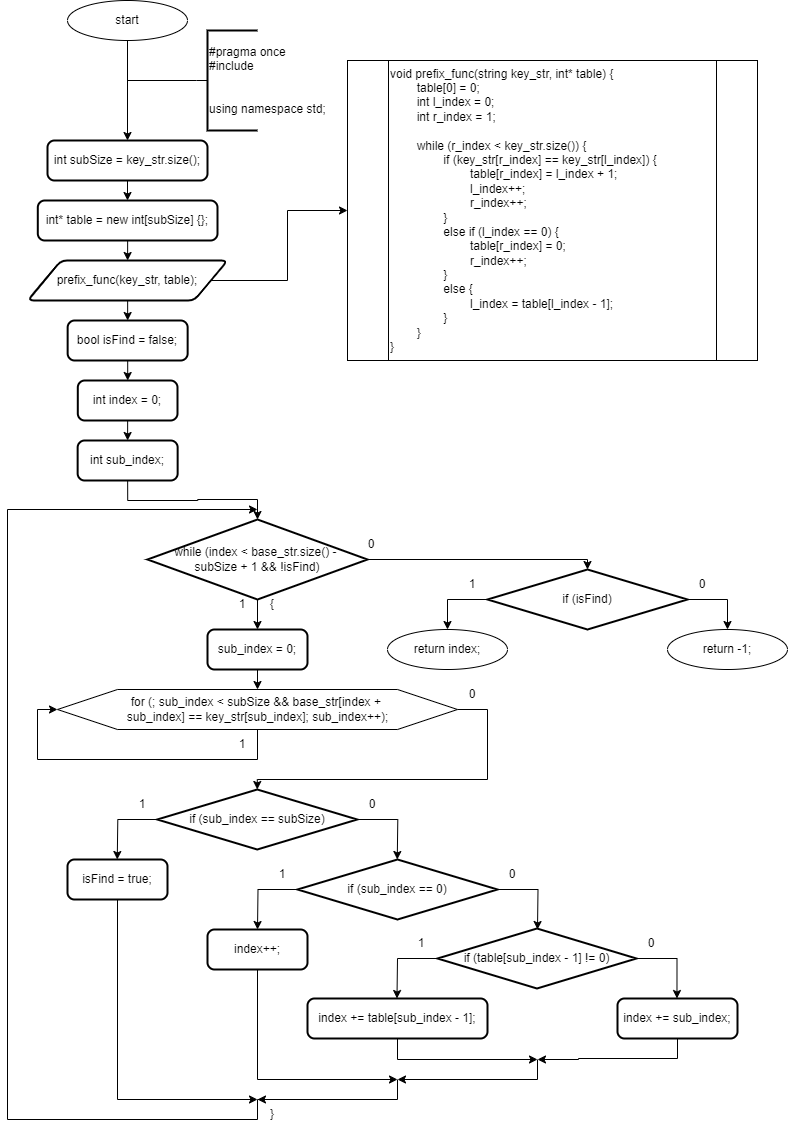
Функция интерполяционного поиска



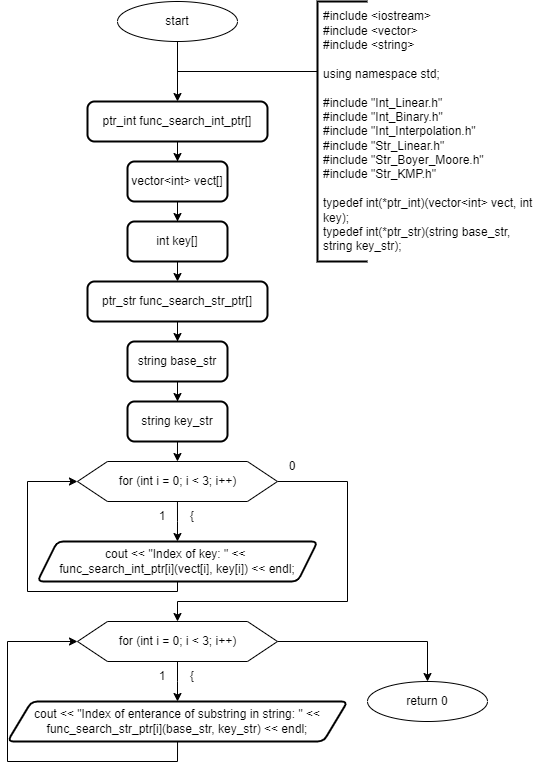
Функция прямого поиска подстроки



Функция алгоритма Буэра Мура



Функция алгоритма КМП



Основная функция

**Код**

Int\_Linear.h

#pragma once

#include <vector>

using namespace std;

int SearchIntLinear(vector<int> vect, int key) {

int index = 0;

for (; index < vect.size() && vect[index] != key; index++);

if (vect[index] == key) {

return index;

}

else {

return -1;

}

}

Int\_Interpolation.h

#pragma once

#include <vector>

using namespace std;

int SearchIntInterpolation(vector<int> vect, int key) {

int l\_index = 0;

int r\_index = vect.size() - 1;

int mid\_index = 0;

do {

mid\_index = l\_index +

((key - vect[l\_index]) \* (r\_index - l\_index)) /

(vect[r\_index] - vect[l\_index]);

if (vect[mid\_index] < key) {

l\_index = mid\_index + 1;

}

else if (vect[mid\_index] > key) {

r\_index = mid\_index + 1;

}

} while (vect[mid\_index] != key && l\_index < r\_index);

if (vect[mid\_index] == key) {

return mid\_index;

}

else {

return -1;

}

}

Int\_Binary.h

#pragma once

#include <vector>

using namespace std;

int SearchIntBinary(vector<int> vect, int key) {

int r\_index = vect.size();

int l\_index = 0;

int mid\_index;

do {

mid\_index = (l\_index + r\_index) / 2;

if (vect[mid\_index] < key) {

l\_index = mid\_index + 1;

}

else if (vect[mid\_index] > key) {

r\_index = mid\_index - 1;

}

} while (vect[mid\_index] != key && (l\_index < r\_index));

if (vect[mid\_index] == key) {

return mid\_index;

}

else {

return -1;

}

}

Str\_Linear.h

#pragma once

#include <string>

using namespace std;

int SearchStrLinear(string base\_str, string key\_str) {

int index = 0;

int sub\_index;

bool isFind = false;

for (; index < base\_str.size() - key\_str.size() && isFind == false; index++) {

sub\_index = 0;

for (; sub\_index < key\_str.size() &&

base\_str[index + sub\_index] == key\_str[sub\_index]; sub\_index++);

if (sub\_index == key\_str.size()) {

isFind = true;

}

}

if (isFind) {

return index - 1;

}

else {

return -1;

}

}

Str\_Boyer\_Moore.h

#pragma once

#include <string>

using namespace std;

int SearchStrBoyerMoore(string base\_str, string key\_str) {

int subSize = key\_str.size();

int table[256];

bool isFind = false;

int index = subSize - 1;

int sub\_index;

for (int i = 0; i < 256; i++) {

table[i] = subSize;

}

for (int i = subSize - 2; i > -1; i--) {

if (table[(int)(unsigned char)key\_str[i]] == subSize) {

table[(int)(unsigned char)key\_str[i]] = subSize - 1 - i;

}

}

while (index < base\_str.size() && isFind == false) {

sub\_index = 0;

for (; sub\_index < subSize &&

base\_str[index - sub\_index] == key\_str[subSize - 1 - sub\_index]; sub\_index++);

if (sub\_index == subSize) {

isFind = true;

}

else {

index += table[(int)(unsigned char)key\_str[subSize - sub\_index]];

}

}

if (isFind) {

return index - subSize + 1;

}

else {

return -1;

}

}

Str\_KMP.h

#pragma once

#include <string>

using namespace std;

void prefix\_func(string key\_str, int\* table) {

table[0] = 0;

int l\_index = 0;

int r\_index = 1;

while (r\_index < key\_str.size()) {

if (key\_str[r\_index] == key\_str[l\_index]) {

table[r\_index] = l\_index + 1;

l\_index++;

r\_index++;

}

else if (l\_index == 0) {

table[r\_index] = 0;

r\_index++;

}

else {

l\_index = table[l\_index - 1];

}

}

}

int SearchStrKMP(string base\_str, string key\_str) {

int subSize = key\_str.size();

int\* table = new int[subSize] {};

prefix\_func(key\_str, table);

bool isFind = false;

int index = 0;

int sub\_index;

while (index < base\_str.size() - subSize + 1 && !isFind){

sub\_index = 0;

for (; sub\_index < subSize &&

base\_str[index + sub\_index] == key\_str[sub\_index]; sub\_index++);

if (sub\_index == subSize) {

isFind = true;

}

else if (sub\_index == 0) {

index++;

}

else {

if (table[sub\_index - 1] != 0) {

index += table[sub\_index - 1];

}

else {

index += sub\_index;

}

}

}

if (isFind) {

return index;

}

else {

return -1;

}

}

Searching.cpp

#pragma once

#include <string>

using namespace std;

void prefix\_func(string key\_str, int\* table) {

table[0] = 0;

int l\_index = 0;

int r\_index = 1;

while (r\_index < key\_str.size()) {

if (key\_str[r\_index] == key\_str[l\_index]) {

table[r\_index] = l\_index + 1;

l\_index++;

r\_index++;

}

else if (l\_index == 0) {

table[r\_index] = 0;

r\_index++;

}

else {

l\_index = table[l\_index - 1];

}

}

}

int SearchStrKMP(string base\_str, string key\_str) {

int subSize = key\_str.size();

int\* table = new int[subSize] {};

prefix\_func(key\_str, table);

bool isFind = false;

int index = 0;

int sub\_index;

while (index < base\_str.size() - subSize + 1 && !isFind){

sub\_index = 0;

for (; sub\_index < subSize &&

base\_str[index + sub\_index] == key\_str[sub\_index]; sub\_index++);

if (sub\_index == subSize) {

isFind = true;

}

else if (sub\_index == 0) {

index++;

}

else {

if (table[sub\_index - 1] != 0) {

index += table[sub\_index - 1];

}

else {

index += sub\_index;

}

}

}

if (isFind) {

return index;

}

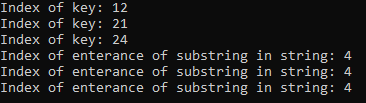
else {

return -1;

}

}

**Решение**



**Выводы**

В ходе выполнения были изучены методы поиска значений в массивах и поиска подстроки в строке, и некоторые особенности их работы.

**Github**

<https://github.com/Hitikov/Lab_Searching>