Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»



Отчет по лабораторной работе №3

по дисциплине «Структура и алгоритмы обработки данных»

по теме «Методы поиска подстроки в строке»

Выполнила: студент группы

БВТ1902

Бубенцов Н.В.

Проверил:

Москва

2021 г.

**Оглавление**

[Цель работы 3](#__RefHeading___Toc8473_3187879512)

[Задание 1 3](#__RefHeading___Toc8475_3187879512)

[Код программы 3](#__RefHeading___Toc8477_3187879512)

[Задание 2 «Пятнашки»](#__RefHeading___Toc8479_3187879512) 6

[Код программы](#__RefHeading___Toc8481_3187879512) 6

[Снимки экрана работы программ 1](#__RefHeading___Toc8483_3187879512)4

[Вывод 14](#__RefHeading___Toc8485_3187879512)

## Цель работы

Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска.

## Задание 1

Алгоритмы:

1. Кнута-Морриса-Пратта
2. Упрощенный Бойера-Мура

## Код программы

#include <iostream>

#include <cstring>

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include <string>

#include <algorithm>

#include <cctype>

using namespace std;

int cmp\_ptr(int lenpatt, int lenst, char st[80], char patt[80], int mas[256])

{

int i, j, metka;

metka = i = j = lenpatt - 1;

while (i > 0 && j < lenst)

{

if (st[j] != patt[i])

{

metka = metka + mas[st[metka]];

i = lenpatt - 1;

j = metka;

}

else

{

i = i--;

j = j--;

}

}

if (j >= lenst)

return -1;

else

return j;

}

using namespace std;

int KMPSearch(char\* string, char\* substring)

{

int sl, ssl;

int res = -1;

sl = strlen(string);

ssl = strlen(substring);

if (sl == 0)

cout << "Неверно задана строка\n";

else if (ssl == 0)

cout << "Неверно задана подстрока\n";

else

{

int i, j = 0, k = -1;

int\* d;

d = new int[1000];

d[0] = -1;

while (j < ssl - 1)

{

while (k >= 0 && substring[j] != substring[k])

{

k = d[k];

}

j++;

k++;

if (substring[j] == substring[k])

d[j] = d[k];

else

d[j] = k;

}

i = 0;

j = 0;

while (j < ssl && i < sl)

{

while (j >= 0 && string[i] != substring[j])

{

j = d[j];

}

i++;

j++;

}

delete[] d;

res = j == ssl ? i - ssl : -1;

}

return res;

};

int main()

{

setlocale(LC\_ALL, "Russian");

int A;

int B;

cout << "выбор задачи ";

cin >> A;

switch (A)

{

case 1:

{

char\* string = \_strdup("eeeeeejjjjjjffffggg");

char\* substring = \_strdup("ej");

cout << endl;

cout << endl << "результат " << KMPSearch(string, substring) << endl;

return 0;

}

case 2:

{

char st[80] = { "iiiooorrrttt" }, patt[80] = { "io" };

int i, j, lenst, lenpatt, mas[256], metka;

lenst = strlen(st);

lenpatt = strlen(patt);

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

mas[i] = lenpatt;

for (i = lenpatt - 2; i >= 0; i--)

if (mas[patt[i]] == lenpatt)

mas[patt[i]] = (lenpatt - 1 - i);

i = cmp\_ptr(lenpatt, lenst, st, patt, mas);

if (i > 0)

printf("Результат %d", i);

else

printf("Слово не найдено");

return 0;

}

}

}

## Задание 2 «Пятнашки»

Написать программу, определяющую, является ли данное расположение «решаемым», то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

## Код программы

#include <iostream>

#include <set>

#include <map>

#include <string>

#include <vector>

#include <time.h>

using namespace std;

typedef vector<int> Vertex;

class Graph

{

public:

Graph(vector<int>& StartCondition);

bool HasASolution();

void GetNextVert(Vertex& v, vector<Vertex>& next);

void AStar();

void PrintAns();

char WhichMove(Vertex from, Vertex to);

int CountTheHeuristic(Vertex v);

int LinearConf(Vertex v);

private:

Vertex Start;

Vertex Final;

map<Vertex, int> f;

map<Vertex, int> g;

map<Vertex, Vertex> parents;

};

char Graph::WhichMove(Vertex from, Vertex to)

{

int FirstSpacePosition = 0;

int SecSpacePosition = 0;

for (int i = 0; i < 16; i++)

{

if (from[i] == 0)

{

FirstSpacePosition = i;

break;

}

}

for (int i = 0; i < 16; i++)

{

if (to[i] == 0)

{

SecSpacePosition = i;

break;

}

}

switch (SecSpacePosition - FirstSpacePosition)

{

case 1:

return 'R';

break;

case -1:

return 'L';

break;

case 4:

return 'D';

break;

case -4:

return 'U';

default:

break;

}

return 0;

}

void Graph::PrintAns()

{

int NumOfMoves = 0;

string ReverseWay;

Vertex tmp = Final;

while (tmp != Start)

{

NumOfMoves++;

Vertex prev = parents[tmp];

ReverseWay += WhichMove(tmp, prev);

tmp = prev;

}

cout << NumOfMoves << endl;

for (int i = (int)ReverseWay.length() - 1; i >= 0; i--)

{

cout << ReverseWay[i];

}

}

void Graph::AStar()

{

if (!HasASolution())

{

cout << -1;

exit(0);

}

set<pair<int, Vertex>> q;

set<Vertex> used;

g.insert(pair<Vertex, int>(Start, 0));

f.insert(pair<Vertex, int>(Start, g[Start] + CountTheHeuristic(Start)));

q.insert(pair<int, Vertex>(f[Start], Start));

while (!q.empty())

{

Vertex tmp = q.begin()->second;

q.erase(q.begin());

if (tmp == Final)

{

PrintAns();

return;

}

used.insert(tmp);

vector<Vertex> NextVert;

GetNextVert(tmp, NextVert);

for (auto v : NextVert)

{

int TentScore = g[tmp] + 1;

if (used.find(v) != used.end() && TentScore >= g[v])

{

continue;

}

if (used.find(v) == used.end() || TentScore < g[v])

{

parents.insert(pair<Vertex, Vertex>(v, tmp));

g.insert(pair<Vertex, int>(v, TentScore));

f.insert(pair<Vertex, int>(v, g[v] + CountTheHeuristic(v)));

if (q.find({ f[v] - g[v],v }) == q.end())

q.insert(pair<int, Vertex>(f[v] - g[v], v));

}

}

}

}

int Graph::CountTheHeuristic(Vertex v)

{

int Heuristic = 0;

for (int i = 0; i < v.size(); i++)

{

int elem = v[i];

if (elem == 0)

{

Heuristic += abs(3 - i / 4) + abs(3 - i % 4);

}

else

{

Heuristic += abs((elem - 1) / 4 - i / 4) + abs((elem - 1) % 4 - i % 4);

}

}

return Heuristic;

}

void Graph::GetNextVert(Vertex& v, vector<Vertex>& next)

{

int SpacePosition = 0;

for (int i = 0; i < 16; i++)

{

if (v[i] == 0)

{

SpacePosition = i;

break;

}

}

int i = 0;

if (SpacePosition < 15 && SpacePosition % 4 != 3)

{

next.push\_back(v);

swap(next[i][SpacePosition], next[i][SpacePosition + 1]);

i++;

}

if (SpacePosition > 0 && SpacePosition % 4 != 0)

{

next.push\_back(v);

swap(next[i][SpacePosition], next[i][SpacePosition - 1]);

i++;

}

if (SpacePosition >= 4)

{

next.push\_back(v);

swap(next[i][SpacePosition], next[i][SpacePosition - 4]);

i++;

}

if (SpacePosition <= 11)

{

next.push\_back(v);

swap(next[i][SpacePosition], next[i][SpacePosition + 4]);

}

}

Graph::Graph(vector<int>& g)

{

int j = 1;

for (auto i : g)

{

Start.push\_back(i);

Final.push\_back(j);

j++;

}

Final[15] = 0;

}

bool Graph::HasASolution()

{

int inv = 0;

for (int i = 0; i < 16; ++i)

if (Start[i])

for (int j = 0; j < i; ++j)

if (Start[j] > Start[i])

++inv;

for (int i = 0; i < 16; ++i)

if (Start[i] == 0)

inv += 1 + i / 4;

if ((inv & 1) == 1)

{

return false;

}

else

{

return true;

}

}

int main()

{

vector<int> g;

for (int i = 0; i < 16; i++)

{

g.push\_back(i);

}

for (int i = 0; i < 16; i++)

{

int idx = rand() % 16;

int tmp = g[idx];

g[idx] = g[0];

g[0] = tmp;

}

for (int i = 0; i < 16; i++)

{

cout << g[i] << " ";

if ((i+1) % 4 == 0)

{

cout << endl;

}

}

time\_t st = clock();

Graph graph(g);

graph.AStar();

time\_t ft = clock();

cout << endl << ft - st;

return 0;

}

## Снимки экрана работы программ

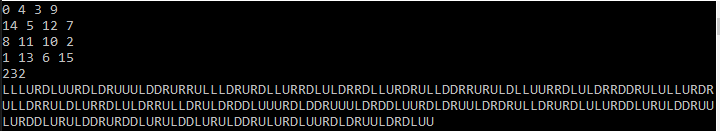
Задача 1

Кнута-Морриса-Пратта

Бойера-Мура



Задача 2



## Вывод

Я реализовал методы поиска подстроки в строке, предусмотрел возможность существования пробела, и оценил время работы каждого алгоритма поиска. А так же написал алгоритм решения «пятнашек» с использование эвристики Линейный конфликт + Манхэттенское расстояние, способный выдать решение с малым числом ходов.