**Project 1 Report**

**Exact Appearance Algorithm Implementation**

**Class: Advanced Analysis of Algorithms**

**Semester: Spring 2019**

**Instructor: Xiuzhen Huang**

**Student Name: Ahmad Banisaeed**

**Student ID: 50527268**

**Report**

The Exact appearance Algorithm looks for an exact appearance of a substring s of length L in the k strings s1, s2, . . . , sk.

Input: k strings, (the length of each string is m), and a positive integer L, the length of the motif, (L<m).

Output: motif of length L.

Step 1: For each of the length-L substring x of the string s1 do

Step 2: For i=2 to k do

Step 3: Compare x with si to see if x is a substring of si;

Step 4: If yes, then mark the position;

Step 5: Else break;

Step 6: Endfor;

Step 7: If i == (k+1)

Step 8: then Return x;

Step 9: Endfor;

Step 10: Return (“No exact appearance of a common motif”);

**Source Code ( exactAppearance.cpp)**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Computer Science 2019

//

// Spring 2019

//

// Assignment # 01

//

// Due Date: Wednesday, February, 27

//

// Instructor: Dr. Xiuzhen Huang;

//

// Programmer: Ahmad Banisaeed

//

// Description: Algorithm looks for an exact appearance of a substring s of length L in the k strings s1, s2, . . . , sk.

//

// Input: k strings, (the length of each string is m), and a positive integer L, the length of the motif, (L<m).

//

// Output: motif of length L.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<iostream>

#include <fstream>

#include<string>

#include<cstring>

#include<vector>

using namespace std;

// Functions prototypes

int KMPMatch(string, string);

vector<int> failureFunction(string);

int Tlenght;

int Plenght;

int\* F;

vector<int> position;

int motiffound;

int main() {

ifstream inputFile("test.txt");

std::string line;

vector<string> strings;

while(getline(inputFile,line)) {

strings.push\_back(line);

// cout << line << "\n";

}

cout << "\n\n";

vector<string>::iterator it;

for (it = strings.begin(); it != strings.end(); ++it) {

cout << \*it << '\n';

}

cout << "\n\n";

inputFile.close();

motiffound = 0;

int k = strings.size();

//cout << " total strings we have is " << k << endl;

int m = strings.at(0).size();

cout << "\nfirst string length is " << m;

Tlenght = m;

int L;

cout << "\nPlease enter the length of the motif: ";

cin >> L;

Plenght = L;

if ( L >= m ) {

cout << "Error: the length of motif must be less than the lenght of string.\n";

exit(0);

}

string x;

x.resize(L);

//cout << "\nPattern Size is " << x.size() << "\n";

position.resize(k);

int found = 0;

for (int i = 0; i <= m - L; i++) {

x = strings.at(0).substr(i,L);

position.at(0) = i;

//cout << " x = " << x << endl;

for (int j = 1; j < k; j++) {

position.at(j) = KMPMatch(strings.at(j),x);

if (position.at(j) != -1)

found++;

}

//cout << "found = " << found << " k = " << k << endl;

if (found == k - 1) {

cout << "motif found: " << x << endl;

motiffound = 1;

exit(0);

}

found = 0;

}

//cout << "checking pattern (" << x << ") in string " << 0

// << " with rest of strings." << endl;

for (int j = 1; j < k; j++) {

position.at(j) = KMPMatch(strings.at(j),x);

//cout << "pattern (" << x << ") found in string " << j

// << " at index " << position.at(j) << endl;

}

if(motiffound == 0)

cout << "motif of length " << L << " was not found!\n";

return 0;

}

int KMPMatch(string T, string p) {

vector<int> F = failureFunction(p);

int i = 0; // T index

int j = 0; // p index

while (i < Tlenght) {

//cout << T[i] << " " << p[j] << "\n";

if (p[j] == T[i]) {

if(j == Plenght - 1) {

return i - j; // match

break;

}

else {

i = i + 1;

j = j + 1;

}

}

else {

if (j > 0) {

j = F[j-1];

}

else {

i += 1;

}

}

}

return -1;

}

vector<int> failureFunction(string p) {

vector<int> shift(Plenght);

shift[0]= 0;

int pos = 1;

string temp = p.substr(0,Plenght);

// cout << "temp " << temp << endl;

// cout << "temp[pos -1] temp[cnd]" << endl;

while (pos < Plenght) {

string prefix = temp.substr(0,pos);

string suffix = prefix.substr(1,pos);

// cout << "temp " << temp.substr(0,pos) << endl;

// cout << "prefix " << prefix << endl;

// cout << "suffix " << suffix << endl;

for(int r = 1; r <= pos - 1; r++) {

if(prefix.substr(0,r) == suffix.substr(pos- 1 - r,r)) {

shift[pos] = r;

}

}

// cout << pos << " " << temp.substr(0,pos) << " " << shift[pos]

// << endl;

pos++;

}

return shift;

}

Input: test.txt

atgaccgggatactgatAAAAAAAAGGGGGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataAAAAAAAAGGGGGGGa

tgagtatccctgggatgacttAAAAAAAAGGGGGGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgAAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAAGGGGGGGcttatag

gtcaatcatgttcttgtgaatggatttAAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtAAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttAAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAAGGGGGGGaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAAGGGGGGGa

Output:

![A screenshot of a cell phone

Description automatically generated]()

![A screenshot of a computer

Description automatically generated]()