**REPORT**

**[Prac 6]**



**과 목 : 알고리즘06**

**담당교수 : 주종화 교수님**

**학 과 : 컴퓨터공학과**

**학 번 : 2021111971**

**이 름 : 이재혁**

텍스트, 클립아트이(가) 표시된 사진

자동 생성된 설명

**프로그램 구현 환경**

프로그래밍언어 : C++

텍스트 에디터 : Visual Studio Code

운영체제 : MAC OS

**문제 1 : 스트링 매칭 알고리즘 구현**

**프로그램 설계**

Matching 클래스를 구현해 공통된 text와 pattern을 가지고 text에서 pattern 스트링 탐색

**소스코드**

// 2021111971 이재혁 문제1

#include <chrono> // 함수 시간측정을 위한 라이브러리

#include <cmath> // 해시 연산을 위한 (제곱) 라이브러리

#include <iostream>

#include <string>

using namespace std;

using namespace chrono;

class Matching {

private:

char \*text;

char \*pattern;

int tlen;

int plen;

public:

Matching() {

string input;

cout << "TextString 입력" << endl;

getline(cin, input);

tlen = input.length();

text = new char[tlen];

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

text[i] = input[i];

}

cout << "PatternString 입력" << endl;

getline(cin, input);

plen = input.length();

pattern = new char[plen];

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

pattern[i] = input[i];

}

}

~Matching() {

delete[] text;

delete[] pattern;

}

void BruteForce() {

cout << "----------- BruteForce -----------" << endl;

system\_clock::time\_point start\_time = system\_clock::now();

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

if (text[i] == pattern[0]) {

// text의 문자가 pattern의 첫글자와 같다면

// pattern 길이 만큼 문자비교

int isMatching = 1; // 매칭여부를 저장하는 변수

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

if (text[i + j] != pattern[j]) {

// 만약 중간에 다른 값이 들어있다면

isMatching = 0; // 매칭 X

break;

}

}

if (isMatching) {

// 문자열이 매칭되었다면 범위 출력

cout << "Matching" << endl;

cout << "TextString[" << i << "] ~ TextString["

<< i - 1 + plen << "]" << endl;

}

}

}

system\_clock::time\_point end\_time = system\_clock::now();

nanoseconds nano = end\_time - start\_time;

cout << "걸린시간(nano sec) : " << nano.count() << endl;

}

void KMP() {

cout << "-------------- KMP --------------" << endl;

system\_clock::time\_point start\_time = system\_clock::now();

int \*table; // Pattern의 최대 접두부를 저장하는 배열

table = new int[plen];

int preIndex = 0; // 접두사 시작 인덱스

table[0] = 0; // 첫 글자의 최대 접두부는 0

for (int sufIndex = 1; sufIndex < plen; sufIndex++) {

// pattern의 1번 index부터 접미사 시작인덱스

while (preIndex > 0 && pattern[preIndex] != pattern[sufIndex]) {

// 접두사와 접미사의 시작을 비교해 겹치는 부분이 있거나

// 맨 처음 접두사 까지 시작인덱스 변경

preIndex = table[preIndex - 1];

}

if (pattern[preIndex] == pattern[sufIndex]) {

// 접두사의 시작과 접미사의 시작이 같다면

table[sufIndex] = ++preIndex;

// 이전까지 접두부 길이 + 1

} else {

// 그렇지 않았다면 preIndex가 pattern의 맨 앞 -> 0

// 겹치는 부분이 없고, 현재 비교하는 자리도 다름

table[sufIndex] = preIndex;

}

}

// 접두부 테이블을 활용해 문자열 비교

int pindex = 0; // pattern 자리를 확인할 index

// text를 모두 확인하면서 pattern이 존재하는지 확인

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

while (pindex > 0 && text[i] != pattern[pindex]) {

// pattern 시작 위치 조정

pindex = table[pindex - 1];

// pindex가 현재 위치에서 접두부와 겹치는 부분이 있다면

// 그 접두부 다음 값 부터 다시 비교

}

if (text[i] == pattern[pindex]) {

pindex++; // 매칭된 index + 1 = 매칭되는 길이

if (pindex == plen) {

// pattern의 길이동안 문자열이 일치했다면

// pattern이 포함되어있음

cout << "Matching" << endl;

// 현재 위치에서 pattern 문자열의 길이 만큼 앞쪽에서

// 부터 pattern 존재

cout << "TextString[" << (i + 1 - pindex)

<< "] ~ TextString[" << i << "]" << endl;

pindex = table[pindex - 1];

}

}

}

system\_clock::time\_point end\_time = system\_clock::now();

nanoseconds nano = end\_time - start\_time;

cout << "걸린시간(nano sec) : " << nano.count() << endl;

delete[] table;

}

void RabinKarp() {

cout << "----------- RabinKarp ----------" << endl;

system\_clock::time\_point start\_time = system\_clock::now();

int \*hash; // text 문자의 부분 문자열을 해싱해 배열에 저장

hash = new int[tlen - plen + 1]; // pattern의 길이 만큼 총 길이가 압축

hash[0] = 0; // text의 초기 해싱값 저장

int q = 1000003; // 해싱한 값이 1000003(소수)이 넘지 않도록 mod연산

int patternHash = 0; // pattern의 해싱값 저장

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

// 해싱 방법

// 현재 문자에 'A'를 빼 적당히 작은 수로 맞춘뒤 (알파벳은 0~25로

// 설정) 앞자리를 가장 높은 자리로 설정 해 10의 plen-1 제곱부터

// 한차수씩 내려옴

hash[0] += (text[i] - 'A' + 1) \* (int)pow(10, plen - i - 1) % q;

hash[0] = hash[0] % q;

patternHash +=

(pattern[i] - 'A' + 1) \* (int)pow(10, plen - i - 1) % q;

patternHash = patternHash % q;

}

for (int i = 1; i < tlen - plen + 1; i++) {

// 현재 해싱값은, 이전 해싱값에서 가장 앞자리를 계산해 뺀뒤,

// 10을 곱해 (이전 해싱값의 두번째 값부터 모두 한자리씩 올라감)

// 새로운 값(마지막값)을 더한다

hash[i] =

(hash[i - 1] -

(text[i - 1] - 'A' + 1) \* (int)pow(10, plen - 1) % q + q) %

q;

// + q 를 하는 이유는 최상위 값을 뺐을 때 음수가 되면

// 해싱값이 달라질 수 있음

hash[i] = (hash[i] \* 10 + (text[i + plen - 1] - 'A' + 1)) % q;

}

for (int i = 0; i < tlen - plen + 1; i++) {

// hash배열의 값과, pattern의 값과 비교

if (hash[i] == patternHash) {

cout << "hashMatcing" << endl;

// 해시 매칭 발생 (text에서 pattern의 개수와 유사해야 좋은

// 해싱)

int isMatching =

1; // 해시 매칭된 값이 실제 pattern값인지 확인필요

for (int j = 0; j < plen; j++) {

// 매칭된 위치 부터 text와 pattern을 비교

if (text[i + j] != pattern[j]) {

isMatching = 0;

// 해시매칭된 값의 실제값과, pattern값이 다르면

// 해싱한 값이 같은 다른 문자열

break;

}

}

if (isMatching) { // 해시매칭 값이 실제 pattern값 인지 확인

cout << "Matching" << endl;

cout << "TextString[" << (i) << "] ~ TextString["

<< i + plen - 1 << "]" << endl;

}

}

}

system\_clock::time\_point end\_time = system\_clock::now();

nanoseconds nano = end\_time - start\_time;

cout << "걸린시간(nano sec) : " << nano.count() << endl;

delete[] hash;

}

};

int main() {

Matching m;

m.BruteForce();

m.KMP();

m.RabinKarp();

}

// A STRING SEARCHING EXAMPLE CONSISTING OF A GIVEN PATTERN STRING

// STRING

// 접두부 테이블 test case

// acbdacba

// acacabacacabacacac

// ababd

// adcbadeadcbadcbadcf

// adcbadcf

// ababacabacaabacaaba

// abacaaba

**결과분석**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

매 실행마다 실행시간이 다를 때도 있지만 꾸준하게 KMP가 좋은 성능을 내고 있다.

시간단위가 나노로 매우 짧아 코드를 실행시킬 때 cpu의 상황에 따라 다른 실행 결과가 나오는 것으로 생각된다.

**문제 2 : 스트링 매칭 알고리즘 응용**

**input.txt 생성**

파일생성.cpp

#include <fstream>

#include <iostream>

#include <random> // OS의 균일한 난수 생성보장

using namespace std;

int main() {

random\_device rd; // 시드값을 얻기 위한 random\_device

mt19937 gen(rd()); // random\_device 를 통해 난수 생성 엔진을 초기화

uniform\_int\_distribution<int> dis(0, 3); // 0 부터 3의 균열한 난수 생성

ofstream fwriter; // 텍스트 파일에 입력할 객체

fwriter.open("input.txt"); // input.txt

for (int i = 0; i < 100000; i++) {

switch (dis(gen)) {

case 0:

fwriter.write("A", 1);

break;

case 1:

fwriter.write("C", 1);

break;

case 2:

fwriter.write("G", 1);

break;

case 3:

fwriter.write("T", 1);

break;

}

}

fwriter.close();

}

생성된 input.txt

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**프로그램 설계**

문제 1의 BruteForce 알고리즘과, KMP알고리즘을 가져와 비교한다.

랜덤으로 생성한 염기서열 pattern을 text에서 찾아 시작index를 파일에 적어 넣는 시간을 비교한다.

**소스코드**

// 2021111971 이재혁 문제2

#include <chrono> // 함수 시간측정을 위한 라이브러리

#include <fstream>

#include <iostream>

#include <random>

#include <string>

#include <vector>

using namespace std;

using namespace chrono;

class PatternWriter {

private:

int n, m;

string text;

char \*pattern;

public:

PatternWriter(int textLen, int patternLen) {

random\_device rd; // 시드값을 얻기 위한 random\_device

mt19937 gen(rd()); // random\_device 를 통해 난수 생성 엔진을 초기화

uniform\_int\_distribution<int> dis(0, 3); // 0 부터 3의 균열한 난수 생성

n = textLen, m = patternLen;

// input으로 부터 n개의 문자를 읽어와 text로 사용

ifstream r("input.txt");

getline(r, text);

text = text.substr(0, n);

pattern = new char[m];

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

switch (dis(gen)) { // 생성한 난수에 따라 4개의 문자 중 하나로 설정

case 0:

pattern[i] = 'A';

break;

case 1:

pattern[i] = 'C';

break;

case 2:

pattern[i] = 'G';

break;

case 3:

pattern[i] = 'T';

break;

}

}

// 길이가 m인 랜덤 패턴

cout << "Find " << pattern << endl;

// 현재 어떤 패턴이 설정되어 있는지 확인

}

void BruteForce(int file = 0) {

cout << "BruteForce" << endl;

system\_clock::time\_point start\_time = system\_clock::now();

ofstream writer;

if (file) { // 매개변수가 있다면

writer.open("output\_b2.txt"); // output\_b2에 작성

} else { // 없다면

writer.open("output\_b1.txt"); // output\_b2에 작성

}

writer << pattern << " : "; // 현재 패턴 기록

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

if (text[i] == pattern[0]) {

// text의 문자가 pattern의 첫글자와 같다면

// pattern 길이 만큼 문자비교

int isMatching = 1; // 매칭여부를 저장하는 변수

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

if (text[i + j] != pattern[j]) {

// 만약 중간에 다른 값이 들어있다면

isMatching = 0; // 매칭 X

break;

}

}

if (isMatching) {

// 문자열이 매칭되었다면 파일에 시작 인덱스 작성

writer << i << " ";

}

}

}

writer << endl;

system\_clock::time\_point end\_time = system\_clock::now();

nanoseconds nano = end\_time - start\_time;

cout << "걸린시간(nano sec) : " << nano.count() << endl;

}

void KMP(int file = 0) {

cout << "KMP" << endl;

system\_clock::time\_point start\_time = system\_clock::now();

ofstream writer;

if (file) { // 매개변수가 있다면

writer.open("output\_k2.txt"); // output\_k2에 작성

} else { // 없다면

writer.open("output\_k1.txt"); // output\_k1에 작성

}

writer << pattern << " : "; // 현재 패턴 기록

int \*table; // Pattern의 최대 접두부를 저장하는 배열

table = new int[m];

int preIndex = 0; // 접두사 시작 인덱스

table[0] = 0; // 첫 글자의 최대 접두부는 0

for (int sufIndex = 1; sufIndex < m; sufIndex++) {

// pattern의 1번 index부터 접미사 시작인덱스

while (preIndex > 0 && pattern[preIndex] != pattern[sufIndex]) {

// 접두사와 접미사의 시작을 비교해 겹치는 부분이 있거나

// 맨 처음 접두사 까지 시작인덱스 변경

preIndex = table[preIndex - 1];

}

if (pattern[preIndex] == pattern[sufIndex]) {

// 접두사의 시작과 접미사의 시작이 같다면

table[sufIndex] = ++preIndex;

// 이전까지 접두부 길이 + 1

} else {

// 그렇지 않았다면 preIndex가 pattern의 맨 앞 -> 0

// 겹치는 부분이 없고, 현재 비교하는 자리도 다름

table[sufIndex] = preIndex;

}

}

// 접두부 테이블을 활용해 문자열 비교

int pindex = 0; // pattern 자리를 확인할 index

// text를 모두 확인하면서 pattern이 존재하는지 확인

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

while (pindex > 0 && text[i] != pattern[pindex]) {

// pattern 시작 위치 조정

pindex = table[pindex - 1];

// pindex가 현재 위치에서 접두부와 겹치는 부분이 있다면

// 그 접두부 다음 값 부터 다시 비교

}

if (text[i] == pattern[pindex]) {

pindex++; // 매칭된 index + 1 = 매칭되는 길이

if (pindex == m) {

// 문자열이 매칭되었다면 파일에 시작 인덱스 작성

writer << i << " ";

}

}

}

system\_clock::time\_point end\_time = system\_clock::now();

nanoseconds nano = end\_time - start\_time;

cout << "걸린시간(nano sec) : " << nano.count() << endl;

delete[] table;

}

void setPattern(string p) {

// 새로운 문자열로 패턴 세팅

delete[] pattern;

m = p.length();

pattern = new char[m];

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

pattern[i] = p[i];

}

}

};

int main() {

int n, m;

cout << "n : text 길이, m : pattern 길이, 입력 (n >= m) >> ";

cin >> n >> m;

PatternWriter p(n, m);

// text길이가 n, pattern 길이가 m인 무작위 염기서열 탐색

p.BruteForce();

p.KMP();

cout << "======== Find ACCGTAT ========" << endl;

p.setPattern("ACCGTAT");

p.BruteForce();

p.KMP();

// text길이가 1000~100000, pattern 길이가 5~40인 무작위 염기서열 탐색

for (int i = 1000; i <= 100000; i \*= 10) {

for (int j = 5; j <= 40; j += 5) {

cout << "=== text length : " << i << " pattern length : " << j

<< " ===" << endl;

PatternWriter p2(i, j);

// ouput\_b2, k2 에 기록

p2.BruteForce(2);

p2.KMP(2);

}

}

}

**결과 분석**

text의 길이가 100,000

pattern의 길이가 7인 무작위 pattern

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

TGCTCAC탐색에서

KMP 알고리즘이 BruteForce 알고리즘보다 우세했다.

text의 길이가 100,000

ACCGTAT pattern탐색

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명ACCGTAT의 탐색은 BruteForce가 우세했다.

KMP알고리즘에서 접두부 테이블을 생성하는 시간이 오래 걸리는 것을 의심할 수 있었다.

output\_b1.txt (BruteForce 알고리즘이 기록한 파일)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

output\_k1.txt (KMP 알고리즘이 기록한 파일)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

input.txt에서 올바르게 탐색한 것을 확인할 수 있다.

text길이가 1000~100000, pattern 길이가 5~40인 무작위 염기서열 탐색

text length, 1000에서 일부 결과

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

text length 10,000에서 일부 결과

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

text length 100,000에서 일부 결과

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

그래프로 분석

1. Text Length : 1,000

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

1. Text Length : 10,000

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

1. Text Length : 100,000

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

그래프를 통해 text문자열의 길이가 길어질수록 pattern의 길이와 상관없이 KMP알고리즘의 성능이 떨어지는 것을 확인할 수 있다.

이유를 생각해보면

1. 텍스트에 pattern의 길이만큼 잘랐을 때 접두사, 접미사가 같은 확률이 낮아진다.

즉, KMP알고리즘이 탐색하기 좋은 문자열이

모든 부분 문자열보다 상대적으로 작아진다.

1. pattern으로 비교해봐야 하는 횟수가 크다.

KMP 알고리즘은 N <= 10,000 && M <= 15인 조건에서 BruteForce알고리즘보다 효율이 좋다 평가할 수 있다.