



알고리즘 설계 HW #2

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2020년 5월 20일

학부	정보통신공학과
학년	3학년
성명	김소원
학번	12181748

1. 개요

- * KMP를 아래와 같은 제약조건을 만족하도록 구현하고 동작여부 보이기
- (1) KMP알고리즘을 확장하여 숫자와 특수기호도 검색할 수 있도록 만들기.
- * 주어진 "RFC2616_modified.txt"화일을 텍스트로 사용
- (2) 이런 탐색이 가능하도록 알고리즘을 수정한 뒤 어떻게 수정했는지 자세히 기술
- (3) "aba", "aa" 이 검색되는 것을 보이기
- (4) "similar" "satisfy" "refer" "representation" "connections"이 검색되는 것을 보이기

2. 상세 설계내용

벡터를 이용하여 기존 kmp알고리즘을 확장하여 코드를 짚으며 숫자와 특수기호를 검색 가능하게 만들었다. 기존의 kmp알고리즘을 구현시켰을 때에는 문자열이 다음줄로 넘어가면 입력을 받지 못하는 점이 있었다. 따라 주어진 RFC2616_modified.txt 파일이 텍스트로 사용하여 입력을 받으면 첫번째 줄에는 텍스트가 입력되고 RFC2616_modified.txt파일의 두번째 문장이 패턴으로 받아져 이 두 문장이 서로 kmp알고리즘 코드가 적용돼 오류가 나는 현상이 발생하였다. 따라 매우 문자열이 긴 텍스트를 고려하여 텍스트를 입력받았을 때 다음줄로 문자열이 넘어갈 수 있도록 코드를 짚으며 텍스트 안에 "aba" "aa" "similar" "satisfy" "refer" "representation" "connections" 이 각각 몇 개의 단어가 포함되는지, 몇번째 위치에서 나오는지를 알아보았다.

먼저 벡터를 이용하여 pi 배열을 얻는 match 함수 코드를 짚는데 처음에 먼저 패턴의 길이를 받아주는 plength 를 만들었다. 그다음 pi 의 길이를 알기 위해 배열 pi 를 먼저 초기화 해주고 for 문을 돌렸다. For 문에서 i가 패턴크기만큼 돌 때 j가 0보다 크고 p[i]가 p[j]랑 같지 않다면 불일치가 일어난다. 반면 p[i]가 p[j]랑 같다면 j 증가하는 코드를 짚는다. 여기서 pi[i]는 주어진 문자열의 0~i 까지의 부분 문자열 중에서 접두사(prefix) == 접미사(suffix)가 될 수 있는 문자열 중에서 가장 긴 것의 길이를 말하며 이때 접두사가 0부터 i까지의 부분 문자열과 같지 않도록 한다. 만약 같다면 pi[i]=0 이 되게 한다. i 는 텍스트의 현재 비교위치, j 는 패턴의 현재 비교위치를 나타내는 변수인데 kmp 알고리즘에서는 텍스트와 패턴을 비교할 때 이미 앞에서 일치했던 부분은 다시 비교할 필요가 없다. 따라 미리 전처리에 사용해둔 pi 배열을 이용하여 앞부분 비교를 번거롭게 다시 할 필요가 없는 것이다. 즉 i, j 에서 다시 시작할 필요가 없다는 말이다.

이제 벡터를 이용한 KMP 코드를 본다면 먼저 match 과정에서 만든 pi 배열을 전달받는다. pi 배열을 전달받으면 중간시도를 건너뛰고 간단하게 할 수 있기 때문이다. 먼저 텍스트의 길이인 tlength 와 plength 를 int 로 선언한 후 텍스트의 현재 비교위치인 변수 i 가 텍스트의 총 길이까지 for 문을 돌 때 문자열이 다를 경우엔 텍스트와 패턴을 그 다음부터 비교한다. j = pi[j - 1];이게 바로 그 코드이다. 이때 t 는 전체 텍스트를 나타내며 p 는 찾으려는 패턴을 나타낸다. 만약 텍스트 배열과 패턴 배열이 같을 때 마지막 문자열까지 같다면 위치를 저장한다. 만약 맞지 않을 경우 문자열을 다음 열부터 비교한다. 그리고 다음 패턴의 비교위치를 하나씩 증가하며 또 비교한다.

메인 함수를 살펴본다면 먼저 string input;을 선언한다. 이는 긴 텍스트를 입력받기 위해 필요한 부분이다. t 는 텍스트, p 는 패턴으로 설정하여 string 으로 선언하고 텍스트를 입력받기 위한 while 문을 작성한다. 처음 텍스트를 입력받기 위해 getline(cin,t)로만 코드를 짜보았다. 그러나 이렇게 되면 여러 줄로 되어있는 엄청 긴 텍스트는 받지 못하는 현상이 발생되었다. 따라 while 문을 이용하여 수정해보았는데 while(getline(cin,t)) 즉 텍스트를 받도록 설정하며 ^end 가 나오면 텍스트 문장이 끝나도록 설정하였다. 그리고 모든 긴 텍스트를 받기위해 input += " "+t ; 로 모든 줄에 있는 문자열을 더하였다. 즉 텍스트를 다 입력하였으면 마지막에 ^end 를 입력해 끝내면 된다. 그리고 패턴 값은 다음줄로 넘어가지 않으므로 getline(cin,p)로 간단하게 패턴 값을 받았다. 첨에 패턴 값도 텍스트 값을 받는 것과 마찬가지로 while(getline(cin,p))을 이용하고 마지막에 ^end 를 사용하여 패턴 값 입력을 종료시키도록 만들었는데 이렇게 되면 aa 나 ababab 같은 경우의 패턴의 개수를 딱 1 번만 받게 되는 상황이 발생한다. 또한 텍스트안에 패턴 refer 의 개수를 알아볼때 예를들어 preferred 가 있다면 그 안에 refer 는 패턴 개수에 포함이 안되었다. 이러한 경우를 막기위해 패턴 입력을 getline(cin,p)로 수정하였다. matched.size()를 이용해 텍스트에 포함된 패턴 개수를 출력하고 index 를 이용해 텍스트에 포함된 패턴 위치도 출력해보았다.

***주어진 텍스트가 "ababab"이고 패턴인 "abab"인 경우 찾아진 위치가 1뿐만이 아니 라 3에서도 찾을 수 있는가? 또 다른 예로, "aaaaa"에서 "aa"를 찾으면 4번 찾아 지는가?**

이에 관한 대답을 한다면 둘다 찾아질 수 있다. 텍스트에 ababab를 입력하고 패턴에 abab 를입력받으면 패턴 개수는 2개, 패턴의 위치는 1,3이 나온다. 또 다른 예로 aaaaa를 텍스트에 입력하고 aa를 패턴으로 입력받으면 4번 찾아진다. kmp알고리즘을 확장하여 코드를 짰기

때문에 가능한것이다. 아래는 직접 결과값을 나오게 해보았다.

C:\WINDOWS\system32\cmd.exe

텍스트: ababab

^end

패턴: abab

텍스트에 포함된 패턴 갯수: 2

텍스트에 포함된 패턴 위치: 1

텍스트에 포함된 패턴 위치: 3

계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe

텍스트: aaaaaa

^end

패턴: aa

텍스트에 포함된 패턴 갯수: 4

텍스트에 포함된 패턴 위치: 1

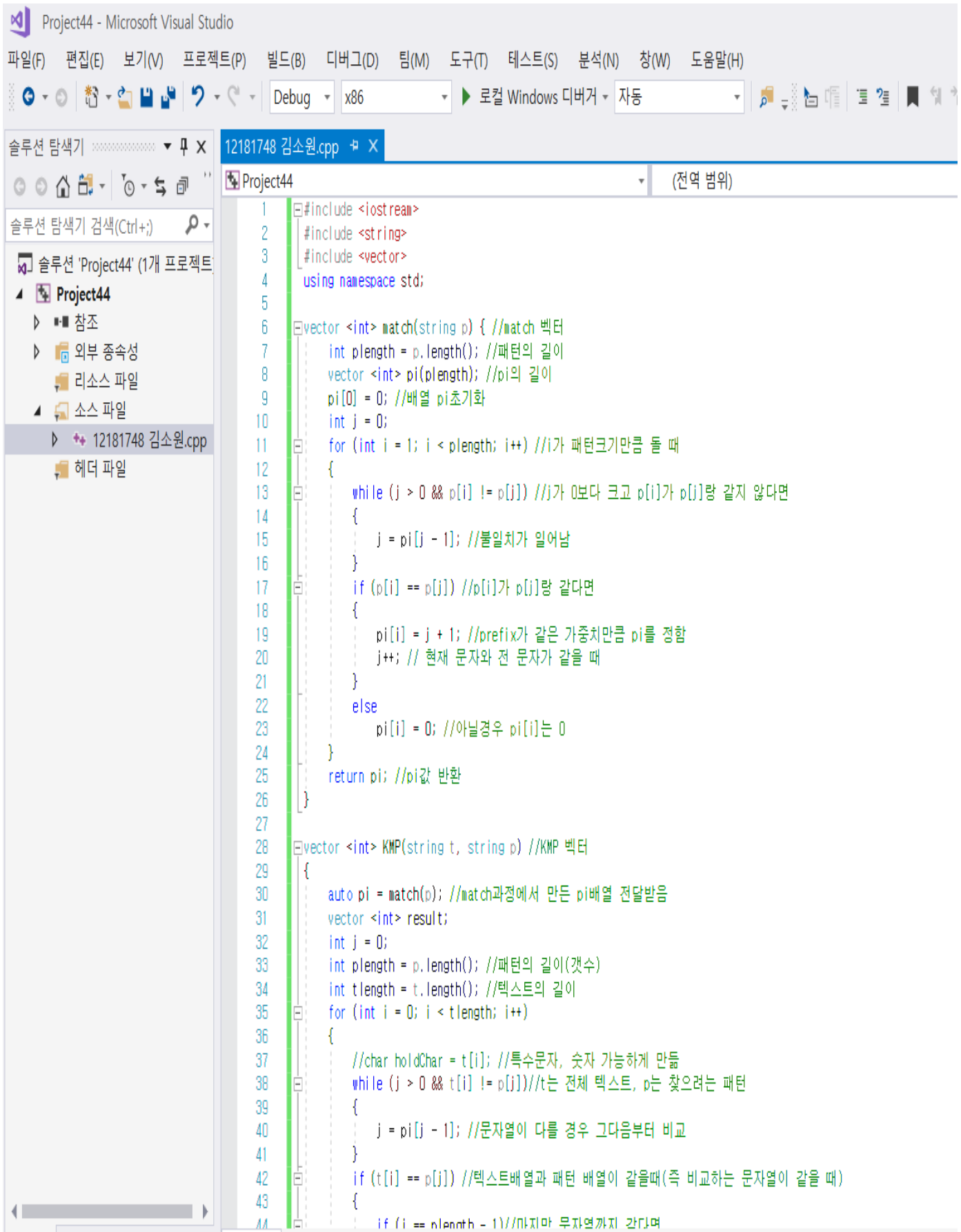
텍스트에 포함된 패턴 위치: 2

텍스트에 포함된 패턴 위치: 3

텍스트에 포함된 패턴 위치: 4

계속하려면 아무 키나 누르십시오 . . .

3. 실행 화면



Project44 - Microsoft Visual Studio

파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H)

Debug x86 로컬 Windows 디버거 자동

솔루션 탐색기

솔루션 탐색기 검색(Ctrl+;)

솔루션 'Project44' (1개 프로젝트)

- Project44
 - 참조
 - 외부 종속성
 - 리소스 파일
 - 소스 파일
 - 12181748 김소원.cpp
 - 헤더 파일

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 using namespace std;
5
6 vector<int> match(string p) { //match 벡터
7     int plength = p.length(); //패턴의 길이
8     vector<int> pi(plength); //pi의 길이
9     pi[0] = 0; //배열 pi 초기화
10    int j = 0;
11    for (int i = 1; i < plength; i++) //i가 패턴크기만큼 둘 때
12    {
13        while (j > 0 && p[i] != p[j]) //j가 0보다 크고 p[i]가 p[j]랑 같지 않다면
14        {
15            j = pi[j - 1]; //불일치가 일어남
16        }
17        if (p[i] == p[j]) //p[i]가 p[j]랑 같다면
18        {
19            pi[i] = j + 1; //prefix가 같은 가장치만큼 pi를 정함
20            j++; // 현재 문자와 전 문자가 같을 때
21        }
22        else
23            pi[i] = 0; //아닐경우 pi[i]는 0
24    }
25    return pi; //pi값 반환
26 }
27
28 vector<int> KMP(string t, string p) //KMP 벡터
29 {
30     auto pi = match(p); //match과정에서 만든 pi배열 전달받음
31     vector<int> result;
32     int j = 0;
33     int plength = p.length(); //패턴의 길이(갯수)
34     int tlength = t.length(); //텍스트의 길이
35     for (int i = 0; i < tlength; i++)
36     {
37         //char holdChar = t[i]; //특수문자, 숫자 가능하게 만들
38         while (j > 0 && t[i] != p[j]) //t는 전체 텍스트, p는 찾으려는 패턴
39         {
40             j = pi[j - 1]; //문자열이 다를 경우 그다음부터 비교
41         }
42         if (t[i] == p[j]) //텍스트배열과 패턴 배열이 같을때(즉 비교하는 문자열이 같을 때)
43         {
44             if (i == plength - 1) //마지막 문자열까지 같다면
```

8 김소원.cpp

```
43     {
44         if (j == plength - 1) // 마지막 문자열까지 같다면
45         {
46             result.push_back(i - plength + 1); // 위치를 저장
47             j = pi[j]; // 맞지 않은 문자열 다음 열부터 비교
48         }
49         else
50             j++; // 하나씩 증가
51     }
52 }
53 }
54 return result; // 결과값 반환
55 }
56 }
```

리소스 파일
소스 파일
12181748 김소원.cpp
헤더 파일

```
56
57 int main() {
58     string input; // 모든 텍스트의 값
59     string t, p; // t는 텍스트 p는 패턴
60     cout << "텍스트: ";
61
62     while (getline(cin, t)) { // 문자열(텍스트)이 다음줄로 넘어갈 수 있게 만든 while문
63         if (t == "^end") // 만약 텍스트가 ^end 나오면
64             break; // 종료
65
66         input += " " + t; // input은 모든 줄에 있는 문자열(텍스트)
67     }
68
69     cout << "패턴: ";
70
71     getline(cin, p); // 패턴 값
72
73     auto matched = KMP(input, p); // 텍스트는 input, 패턴은 p
74     cout << "텍스트에 포함된 패턴 갯수: " << matched.size() << endl;
75     cout << endl;
76     for (auto index : matched)
77         cout << "텍스트에 포함된 패턴 위치: " << index << endl;
78     cout << endl;
79
80 }
81
82
83 }
```

4. 분석 및 결론

(1) aba를 패턴으로 검색한 결과

```
C:\WINDOWS\system32\cmd.exe

3.12 Range Units
HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HTTP/1.1 uses range units in the Range (section 14.35) and Content-Range (section 14.16) header fields. An entity can be broken down into subranges according to various structural units.

    range-unit      = bytes-unit | other-range-unit
    bytes-unit      = "bytes"
    other-range-unit = token
The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations MAY ignore ranges specified using other units.

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.
^end
패턴: aba
텍스트에 포함된 패턴 갯수: 11

텍스트에 포함된 패턴 위치: 89
텍스트에 포함된 패턴 위치: 91
텍스트에 포함된 패턴 위치: 93
텍스트에 포함된 패턴 위치: 95
텍스트에 포함된 패턴 위치: 97
텍스트에 포함된 패턴 위치: 99
텍스트에 포함된 패턴 위치: 101
텍스트에 포함된 패턴 위치: 103
텍스트에 포함된 패턴 위치: 105
텍스트에 포함된 패턴 위치: 107
텍스트에 포함된 패턴 위치: 109

계속하려면 아무 키나 누르십시오 . . .
```

(2) aa를 패턴으로 검색한 결과

C:\WINDOWS\system32\cmd.exe

8.12 Range Units

HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HTTP/1.1 uses range (section 14.16) header fields. An entity can be broken down into subranges according to various structural units.

```
range-unit      = bytes-unit | other-range-unit
bytes-unit      = "bytes"
other-range-unit = token
```

The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations MAY ignore ranges specified using other units.

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.

end

패턴: aa

텍스트에 포함된 패턴 갯수: 45

텍스트에 포함된 패턴 위치: 125
텍스트에 포함된 패턴 위치: 126
텍스트에 포함된 패턴 위치: 127
텍스트에 포함된 패턴 위치: 128
텍스트에 포함된 패턴 위치: 129
텍스트에 포함된 패턴 위치: 130
텍스트에 포함된 패턴 위치: 131
텍스트에 포함된 패턴 위치: 132
텍스트에 포함된 패턴 위치: 133
텍스트에 포함된 패턴 위치: 134
텍스트에 포함된 패턴 위치: 135
텍스트에 포함된 패턴 위치: 136
텍스트에 포함된 패턴 위치: 137
텍스트에 포함된 패턴 위치: 138
텍스트에 포함된 패턴 위치: 139
텍스트에 포함된 패턴 위치: 140
텍스트에 포함된 패턴 위치: 141
텍스트에 포함된 패턴 위치: 142
텍스트에 포함된 패턴 위치: 143
텍스트에 포함된 패턴 위치: 144
텍스트에 포함된 패턴 위치: 145
텍스트에 포함된 패턴 위치: 146
텍스트에 포함된 패턴 위치: 147
텍스트에 포함된 패턴 위치: 148
텍스트에 포함된 패턴 위치: 149
텍스트에 포함된 패턴 위치: 150
텍스트에 포함된 패턴 위치: 151
텍스트에 포함된 패턴 위치: 152
텍스트에 포함된 패턴 위치: 153

텍스트에 포함된 패턴 위치: 153
텍스트에 포함된 패턴 위치: 154
텍스트에 포함된 패턴 위치: 155
텍스트에 포함된 패턴 위치: 156
텍스트에 포함된 패턴 위치: 157
텍스트에 포함된 패턴 위치: 158
텍스트에 포함된 패턴 위치: 159
텍스트에 포함된 패턴 위치: 160
텍스트에 포함된 패턴 위치: 161
텍스트에 포함된 패턴 위치: 162
텍스트에 포함된 패턴 위치: 163
텍스트에 포함된 패턴 위치: 164
텍스트에 포함된 패턴 위치: 165
텍스트에 포함된 패턴 위치: 166
텍스트에 포함된 패턴 위치: 167
텍스트에 포함된 패턴 위치: 168
텍스트에 포함된 패턴 위치: 169

계속하려면 아무 키나 누르십시오 . . .

(3) similar를 패턴으로 검색한 결과

```
C:\WINDOWS\system32\cmd.exe

equivalent and could be substituted for each other with no significant change in semantics. A weak entity tag can only be
used for weak comparison.

An entity tag MUST be unique across all versions of all entities associated with a particular resource. A given entity t
ag value MAY be used for entities obtained by requests on different URIs. The use of the same entity tag value in conjun
ction with entities obtained by requests on different URIs does not imply the equivalence of those entities.

3.12 Range Units
HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HT
TP/1.1 uses range units in the Range (section 14.35) and Content-Range (section 14.16) header fields. An entity can be br
oken down into subranges according to various structural units.

    range-unit      = bytes-unit | other-range-unit
    bytes-unit      = "bytes"
    other-range-unit = token

The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations MAY ignore ranges specified using other uni
ts.

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.
^end
패턴: similar
텍스트에 포함된 패턴 갯수: 5

텍스트에 포함된 패턴 위치: 1828
텍스트에 포함된 패턴 위치: 14733
텍스트에 포함된 패턴 위치: 16502
텍스트에 포함된 패턴 위치: 45082
텍스트에 포함된 패턴 위치: 45385

계속하려면 아무 키나 누르십시오 . . .
```

(4) satisfy를 패턴으로 검색한 결과

```
C:\WINDOWS\system32\cmd.exe

    opaque-tag = quoted-string
A "strong entity tag" MAY be shared by two entities of a resource only if they are equivalent by octet equality.

A "weak entity tag," indicated by the "W/" prefix, MAY be shared by two entities of a resource only if the entities are
equivalent and could be substituted for each other with no significant change in semantics. A weak entity tag can only be
used for weak comparison.

An entity tag MUST be unique across all versions of all entities associated with a particular resource. A given entity t
ag value MAY be used for entities obtained by requests on different URIs. The use of the same entity tag value in conjun
ction with entities obtained by requests on different URIs does not imply the equivalence of those entities.

3.12 Range Units
HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HT
TP/1.1 uses range units in the Range (section 14.35) and Content-Range (section 14.16) header fields. An entity can be br
oken down into subranges according to various structural units.

    range-unit      = bytes-unit | other-range-unit
    bytes-unit      = "bytes"
    other-range-unit = token

The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations MAY ignore ranges specified using other uni
ts.

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.
^end
패턴: satisfy
텍스트에 포함된 패턴 갯수: 1

텍스트에 포함된 패턴 위치: 2537

계속하려면 아무 키나 누르십시오 . . .
```

(5) refer를 패턴으로 검색한 결과

```
C:\WINDOWS\system32\cmd.exe

3.12 Range Units
HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HTTP/1.1 uses range units in the Range (section 14.35) and Content-Range (section 14.16) header fields. An entity can be broken down into subranges according to various structural units.

    range-unit      = bytes-unit | other-range-unit
    bytes-unit      = "bytes"
    other-range-unit = token

The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations MAY ignore ranges specified using other units.

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.
^end
패턴: refer
텍스트에 포함된 패턴 갯수: 12

텍스트에 포함된 패턴 위치: 448
텍스트에 포함된 패턴 위치: 1206
텍스트에 포함된 패턴 위치: 1621
텍스트에 포함된 패턴 위치: 3011
텍스트에 포함된 패턴 위치: 5115
텍스트에 포함된 패턴 위치: 9279
텍스트에 포함된 패턴 위치: 25401
텍스트에 포함된 패턴 위치: 27317
텍스트에 포함된 패턴 위치: 28207
텍스트에 포함된 패턴 위치: 31093
텍스트에 포함된 패턴 위치: 31971
텍스트에 포함된 패턴 위치: 33370

계속하려면 아무 키나 누르십시오 . . .
```

(6) representation를 패턴으로 검색한 결과

```
C:\WINDOWS\system32\cmd.exe

An entity tag MUST be unique across all versions of all entities associated with a particular resource. A given entity tag value MAY be used for entities obtained by requests on different URIs. The use of the same entity tag value in conjunction with entities obtained by requests on different URIs does not imply the equivalence of those entities.

3.12 Range Units
HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HTTP/1.1 uses range units in the Range (section 14.35) and Content-Range (section 14.16) header fields. An entity can be broken down into subranges according to various structural units.

    range-unit      = bytes-unit | other-range-unit
    bytes-unit      = "bytes"
    other-range-unit = token

The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations MAY ignore ranges specified using other units.

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.
^end
패턴: representation
텍스트에 포함된 패턴 갯수: 8

텍스트에 포함된 패턴 위치: 3648
텍스트에 포함된 패턴 위치: 3973
텍스트에 포함된 패턴 위치: 4117
텍스트에 포함된 패턴 위치: 4245
텍스트에 포함된 패턴 위치: 4318
텍스트에 포함된 패턴 위치: 4461
텍스트에 포함된 패턴 위치: 4534
텍스트에 포함된 패턴 위치: 27945

계속하려면 아무 키나 누르십시오 . . .
```

(7) connections를 패턴으로 검색한 결과

```
C:\선택 C:\WINDOWS\system32\cmd.exe

3.12 Range Units
HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HTTP/1.1 uses range units in the Range (section 14.35) and Content-Range (section 14.16) header fields. An entity can be broken down into subranges according to various structural units.

    range-unit      = bytes-unit | other-range-unit
    bytes-unit      = "bytes"
    other-range-unit = token

The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations MAY ignore ranges specified using other units.

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.
^end
패턴: connections
텍스트에 포함된 패턴 갯수: 11

텍스트에 포함된 패턴 위치: 899
텍스트에 포함된 패턴 위치: 4716
텍스트에 포함된 패턴 위치: 4954
텍스트에 포함된 패턴 위치: 6653
텍스트에 포함된 패턴 위치: 6860
텍스트에 포함된 패턴 위치: 11058
텍스트에 포함된 패턴 위치: 11662
텍스트에 포함된 패턴 위치: 11866
텍스트에 포함된 패턴 위치: 13843
텍스트에 포함된 패턴 위치: 14481
텍스트에 포함된 패턴 위치: 26493

계속하려면 아무 키나 누르십시오 . . .
```

주어진 "RFC2616_modified.txt"화일을 텍스트로 사용하며 "aba", "aa" "similar" "satisfy" "refer" "representation" "connections" 총 7개의 패턴이 검색되는 것을 보았다. 각 패턴이 텍스트에 포함된 개수와 위치를 알아보았는데 이때 뛰어쓰기도 위치로 한칸이라 생각하였다. "RFC2616_modified.txt"에 포함된 모든 문장이 끝나면 텍스트가 끝났다는 걸 알려주기 위해 마지막에 다 ^end를 입력하였다.

먼저 aba를 패턴을 했을 때를 보면 텍스트에 포함된 패턴 갯수는 총 11개가 나온다. abababababab 와 같은 문자열을 발견하였을 때 단 하나의 패턴 aba 갯수만을 세는 것이 아닌 첫번째 칸부터의 aba, 3번째 칸의 aba와 같이 abababababab 속에 있는 aba를 다 갯수에 포함 시키기 때문이다. 텍스트에 포함된 패턴의 위치는 위의 결과값과 같이 총 11개의 갯수에 맞는 위치값이 나왔다.

그다음 aa를 패턴으로 했을 때를 보면 텍스트에 포함된 패턴의 갯수는 총 45개가 나왔다. 단어에 aa가 포함되어있으면 그 단어의 aa까지 갯수에 포함된 것이며 패턴 위치 역시 45개에 맞는 위치 값들이 나왔다.

Similar를 패턴으로 했을 때를 살펴보면 텍스트에 포함된 패턴의 갯수는 5개가 나왔다. 패턴 위치 역시 similar 5개가 있는 각각의 위치 값들이 나왔다. Satisfy는 패턴의 갯수가 1개 나왔으며 satisfy 패턴의 위치는 2537이라는 것을 알 수 있었다.

refer를 패턴으로 했을 때를 보면 텍스트에 포함된 패턴의 갯수는 총 12개가 나왔다. 단어에 refer가 포함되어있으면 그 단어의 refer까지 갯수에 포함된 것이다. 예를 들어 prefer이면 prefer의 refer도 포함되었다는 뜻이다. 패턴 위치는 refer 패턴 12개가 있는 각각의 위치 값들이 나왔다

representation과 connections 또한 각각을 패턴으로 했을 때를 보면 텍스트에 포함된 패턴의 갯수는 representation은 8개, connections는 11개가 나왔다. 패턴 위치 역시 representation은 8개의 representation이 있는 위치를 나타내었으며 connections은 11개의 connections 각각의 위치 값들이 나왔다

KMP알고리즘이 여러 제약조건을 만족하도록 확장하여 구현하고 동작함을 보임으로써 kmp 알고리즘이 어떻게 구성되었는지 이해가 더 잘 되었으며 KMP 알고리즘이 매우 편리하다는 것을 알게되었다. 또한 kmp알고리즘을 이용하여 패턴 탐색이 더 쉬워졌으며 KMP알고리즘을 필요로 하는 코드를 짤 때 요번 과제 했던 것을 잘 기억하여 효율적인 코드를 구현해 낼 것이다.

5. 참고문헌 및 참고자료

- 유튜브 kmp algorithm for string matching- part 1,2,3

- Algorithms in C++, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching

(공)저: Robert Sedgewic