알고리즘 설계과제

POA #262 – 패턴 매칭 알고리즘

2017156019 염민규

문제

#262. 다음 패턴 매칭 알고리즘의 정확성을 증명해보아라. 입력 요소는 문자열 S[1..n]과 패턴 P[0..m - 1]로 구성되며, 1 <= m <= n 이다. 이 알고리즘은 문자열 S 에서 패턴 P 의 첫 번째 연속 발생을 찾는다. 즉, S[p..p + m – 1] = P 인 경우, ℓ = p 이고, 패턴 P 가 문자열 S 에 없는 경우 ℓ = n – m + 1 이다.

```
Function match(P,S,n,m)
  Comment Find the pattern P[0..m - 1] in string S[1..n] # S 안에 P 가 있는지 찾는 알고리즘
1.  ℓ := 0; matched := false # 초기 ℓ 값 : 0, 초기 'matched' bool 값 : false
2.  While (ℓ <= n – m) ∧ ¬ matched do
    # ℓ 이 n – m 이하이고 (AND) matched 가 false 이면 수행
3.  ℓ := ℓ + 1 # ℓ 값 1 증가
4.  r := 0; matched := true # r 값에 0 대입, matched 값 true 로 변경
5.  while (r < m) ∧ matched do # r 이 m 보다 작거나, matched 가 true 인 경우 수행
6.  matched := matched ∧ (P[r] = S[ ℓ + r])
    # matched 나 (P[r] = S[ ℓ + r ]) 둘 모두가(AND) true 일 경우 matched 에 true 값 대입
7.  r := r + 1 # r 값 1 증가 ( 반복문 돌면서 r 은 1 씩 증가 )
8.  Return(ℓ) # ℓ 값 리턴
```

알고리즘 이해 및 분석

알고리즘을 내가 직접 달아 놓은 주석과 함께 자세히 이해해보도록 하겠다. 알고리즘의 취지는 문자열 S 안에 특정 패턴 P 가 있는지 찾는 알고리즘으로, 기다란 문자열 S 안에, 비교적 짧은 문자열인 P (패턴이라고 명명)가 있는지 알아내는 것이다.

매개변수로 P, S, n, m 이 필요한데, 이는 각각 찾을 문자열이자 패턴인 P, 탐색 당할 문자열인 S, 문자열 S 의 길이인 n, 패턴 P 의 길이인 m 이다.

초기값으로 ℓ 은 0, matched 즉, 패턴이 일치하는지를 판별해주는 bool 값은 false 로 주어진다. 변수 ℓ 이 정확히 무엇을 의미하는지 잘 모르겠으니 조금 더 살펴보겠다.

ℓ 값은 알고리즘이 수행될 때마다 1 씩 선형으로 증가한다. 그리고 r 값은 ℓ 이 반복문에 의해 1 씩 증가할 때마다 0 으로 초기화된다. 여기서 r 값도 무엇인지 잘 모르겠으니 뒷부분을 더 분석해보겠다. 그리고 matched 값은 계속 true 로 초기화된다.

그 후, 반복문이 추가로 실행된다. 조건은 r 이 m 즉, P 의 크기보다 작거나, matched 가 true 인 경우. 조건에 충족했을 경우 matched 값은 matched 가 true 이거나 P 의 r 번째 인덱스가 S 의 ℓ+r 번째 인덱스가 같은 경우 true 가 적용된다. 이는 둘 다 조건을 충족하지 못하면 false 가 대입된다는 뜻.

이후 r 값은 반복문이 끝날 때까지 1 씩 선형으로 증가한다.

이 두 반복문이 모두 조건을 충족하지 못하게 되어 끝이 나면 최종 ℓ 값을 리턴 하는 알고리즘이다.

5 번째 줄과 6 번째 줄을 통해 r 과 ℓ 이 무엇을 뜻하는지 유추할 수 있다. R 이 m 보다 작다는 의미는 즉, P 의 크기보다 작다는 뜻. 이는 변수 r 이 패턴 P 를 탐색하는데 쓰이는 '현재 탐색중인 인덱스 변수"라는 것이다. 그리고 ℓ 은 탐색 당하는 문자열인 S 의 "현재 탐색중인 인덱스 변수"로 쓰이는데, 변수 r 을 더하여 사용된다. 왜 r 값과 ℓ 값을 더한 값을 문자열 S 를 탐색하는 인덱스 변수로 쓸까? 만약 S 문자열의 특정 인덱스 값과 패턴 P 의 0 번째 인덱스 값이 일치한다고 가정해보겠다. 이는 실제로 6 번째 줄의 조건 중 하나인 (P[r] = S[ℓ + r])를

의미하는데, 이 조건이 충족하여 반복문이 호출될 때마다 r 값은 1 씩 증가 즉, 패턴 P 의 1 번째.. 2 번째.. 3 번째.. 조건이 계속 충족한다면 m -1 번째까지 탐색할 것이고, 문자열 S 또한 S 의 ℓ 번째.. ℓ+1 번째.. ℓ+2 번째.. 조건이 계속 충족한다면 ℓ-m+1 번째까지 탐색할 것이다. 무엇을 뜻하는가? 바로 패턴 P 가 문자열 S 에 존재하는지 알아내는 부분이다. (이렇게 직접 풀어서 제 자신에게 설명하니 드디어 이해가 됐다.) 패턴 P 의 m-1 번째 요소와 S 의 ℓ-m+1 까지 일치한다는 결론이 나오면, "문자열 S 에 패턴 P 가 존재한다"는 말과 동일하다.

여기서 만약, P 의 첫 글자와 S 의 첫 문자가 같지만, 비교하는 중간에 다른 경우는 어떻게 흐름이 이어질까? 예를 들면 P = "알고리즘" , S = "나는알곤이찜을좋아한다" 라고 가정해보겠다. S 의 2 번째 인덱스 요소가 P 의 0 번째 인덱스 요소와 "알"로 같아 matched 가 true 로 바뀌어 반복문이 시작된다. 허나 비교를 더 해보니 P 의 1 번째 인덱스 요소는 "고"이고, S 의 3 번째 인덱스 요소는 "곤"이다. 결국 첫 문자는 같아도 비교를 해보니 같은 패턴이 존재한게 아님을 알았기에, 6 번째 줄에 의해 matched 는 false 가 되고 반복문이 종료되게 된다.

이어서 다른 부분을 분석해보자. 2 번째 줄, 반복문의 조건에서 보면 ℓ 은 n - m 까지만 증가한다. 이는 S 의 n-m 번째 인덱스 까지만 탐색하겠다는 뜻인데, 왜인가 생각해보면 어차피 그 이후는 탐색 해봤자 패턴 P 의 문자열보다 짧은 부분만 남기 때문에 일치할 수 없기 때문이다.(의미가 없다) 전체적인 알고리즘은 ℓ 이 S 를 탐색할 수 있는 유효한 마지막 인덱스번호를 넘어가거나, matched 가 true 즉, 패턴이 끝내 같은 부분이 존재하면 끝나며 return 값으로는 ℓ 값 즉, 패턴 P 가 매칭되기 시작한 문자열 S 의 첫 인덱스 값이 되게 된다.

만약 패턴 P 가 문자열 S 에 존재하지 않는다면 어떻게 될까? 이는 #262 에 대한 사전 설명에도 나와 있듯이, n-m+1 이 나오게 될 것이고, 이는 2 번째 줄 조건인 (ℓ <= n – m) 에 대해 false 이기 때문, 탐색이 끝났고, "문자열 S 안에 패턴 P 가 존재하지 않는다"라는 다른 의미를 내포하고 있다.

결론적으로, 이 알고리즘은 "패턴이 존재하는지 여부"만 알려주는 게 목적이 아닌, "패턴이 존재한다면, 문자열의 어느 위치부터 해당 패턴이 존재하는지" 를 알려주는 알고리즘이다.

함수 선언

함수명 : match

매개 변수 : String P, String S, int n, int m

1. String P : 패턴이며, 문자열 S 에 존재하는지 비교하기 위한 문자열
2. String S : 패턴이 존재하는지 탐색할 문자열
3. Int n : 문자열 S 의 길이
4. Int m : 패턴 P 의 길이 리턴 값

리턴 값

1. 매칭 성공 시, 매칭된 패턴이 시작되는 문자열 시작점 인덱스 값
2. 매칭 실패 시, 마지막으로 찾으려고 시도했던 인덱스 값

목적 : 문자열 속에 특정 패턴이 존재하는지 알아내고, 만약 존재한다면 패턴이 시작되는 문자열의 위치를 알아내는

지역 변수

1. I
   - ✓ 현재 탐색중인 문자열 S 의 인덱스 (위치) 번호
   - ✓ POA #262 수도코드의 변수 ℓ 과 동일
2. matched
   - ✓ 패턴 P 와 문자열 S 의 요소를 비교하여 동일한지 아닌지 여부를 알려주는 bool 값
   - ✓ True : 현재까지 패턴 P 와 문자열 S 가 매칭됨 (같음)
   - ✓ False : 패턴 P 와 문자열 S 가 매칭되지 않는 상태 (같지 않음)

- ✓ POA #262 수도코드의 변수 matched 와 동일

3. r

   - ✓ 문자열 S 안에서 P 의 0 번째 인덱스 요소와 같은 문자가 발견되면 인덱스 참조 목적으로 사용되는 변수
   - ✓ r 값은 반복문이 돌 때마다 0 으로 초기화
   - ✓ POA #262 수도코드의 변수 r 과 동일

전역 변수

1. pattern

   - ✓ 패턴으로, match 함수의 매개변수 P 에 대응되는 input 변수

2. Str

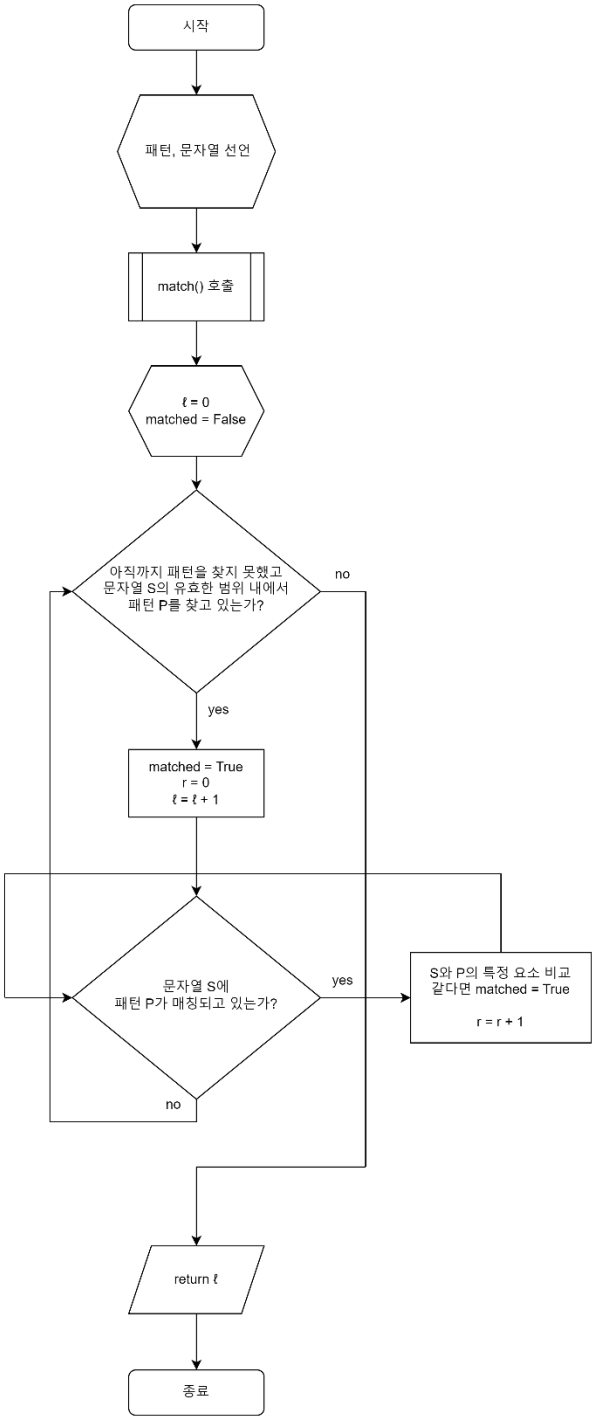
   - ✓ 패턴이 존재하는지 탐색할 문자열로, 매개변수 S 에 대응되는 input 변수

3. result

   - ✓ match 함수의 return 값을 받아오는 변수

순서도

‘알고리즘 이해 및 분석’을 가시성 좋게 순서도로 표현해보자.



구현

언어 : Python 개발

환경 : VS Code

코드

패턴매칭 알고리즘.py > ...

```python
# POA #262

def match(P,S,n,m):
    l = 0
    matched = False

    while(l <= n - m and matched == False):
        l += 1
        r = 0
        matched = True

        while(r < m and matched == True):
            matched = (matched == True) and (P[r] == S[l + r])
            r += 1

    return l
```

시연

출력 결과를 깔끔히 하기 위해 아래와 같은 코드를 추가하였음

```python
if(result < len(str) - len(pattern) + 1):
    print("[%s] 라는 패턴은 문자열에 존재하네요!"%(pattern))
    print("해당 패턴이 문자열 [%s]의 %d번째 인덱스부터 시작합니다."%(str,result))
else:
    print("패턴 [%s]이 문자열 [%s]에 존재하지 않습니다!"%(pattern, str))
```

패턴이 문자열에 존재한다는 의미를 가진 값이 return 되었을 경우를 조건으로 걸었다. 만약 패턴이 문자열에 존재하지 않는다면 n - m 보다 l 이 커지는 시점에서 반복이 종료되어 l 값이 return 되었기 때문에 return 값을 가지는 result 값은 문자열 str 의 길이에서 pattern 의 길이를 뺀 값에 1 을 더한 값, 즉 len(str) – len(pattern) + 1 이다. 그러므로, 이 값보다 작은 값이 result 에 있다면 패턴이 문자열에 존재한다고 구분할 수 있을 것이다.

1. 패턴 P 가 문자열 S 에 있는 경우

```
17    pattern = "algo"
18    str = "ilovealgorithm"
19    result = match(pattern,str,len(str),len(pattern))
```

패턴 : “algo”

문자열 : “ilovealgorithm”

결과

```
[algo] 라는 패턴은 문자열에 존재하네요!
해당 패턴이 문자열 [ilovealgorithm]의 5번째 인덱스부터 시작합니다.
PS C:\Users\염민규\Desktop\코딩 틀\git\Python>
```

정상적으로 작동함을 알 수 있다.

2. 패턴 P 가 문자열 S 에 없는 경우

```
17    pattern = "알고리즘"
18    str = "나는알곤이찜을좋아한다"
19    result = match(pattern,str,len(str),len(pattern))
```

패턴 : “알고리즘”

문자열 : “나는알곤이찜을좋아한다”

결과

```
패턴 [알고리즘]이 문자열 [나는알곤이찜을좋아한다]에 존재하지 않습니다!
PS C:\Users\염민규\Desktop\코딩 틀\git\Python>
```

정상적으로 작동함을 알 수 있다.

버그

여러 번 테스트를 진행해보다가 이 코드의 결점을 찾아냈다. 패턴이 문자열의 0 번째 인덱스부터 일치하는 경우는 찾아내지 못하는 것이었다.

예를 들어보자.

```
17  pattern = "alg"
18  str = "algorithm"
19  result = match(pattern,str,len(str),len(pattern))
```

패턴 : “alg”

자열 : “algorithm”

결과

```
패턴 [alg]이 문자열 [algorithm]에 존재하지 않습니다!
PS C:\Users\염민규\Desktop\코딩 툴\git\Python>
```

해당 패턴이 문자열의 0 번째 인덱스부터 시작한다는 결과가 나와야 하는데 이를 찾아내지 못하고 있다.

수도코드대로 코드를 짰는데 무슨 일인가? 그냥 넘길 수 없다. 해결해보자

원인

```
4       l = 0
5       matched = False
6
7   ∨   while(l <= n - m and matched == False):
8           l += 1
9           r = 0
10          matched = True
```

해당 코드의 4 번째 줄을 보면, match 함수가 실행되고 l 값을 0 으로 초기화 시켜준다. 그런데 첫 반복문에 진입하자 l 의 값을 바로 1 증가시키고 시작한다. 이는 문자열 s 의 1 번째 인덱스부터 비교해버린다. 그러니 s 의 0 번째 인덱스부터 패턴이 일치하면 찾아내지 못했던 것이다.

손 놓고 지켜보겠는가? 내 사전에 예외란 없다. 해결해보자.

해결

```python
def match(P,S,n,m):
    matched = False

    if(S[0] == P[0]):
        l = 0
        r = 0
        matched = True
        while(r < m and matched == True):
            matched = (matched == True) and (P[r] == S[l+r])
            r += 1
        if(matched == True):
            return 0

    l = 0
    while(l <= n - m and matched == False):
        l += 1
        r = 0
        matched = True

        while(r < m and matched == True):
            matched = (matched == True) and (P[r] == S[l + r])
            r += 1

    return l
```

패턴 P 가 문자열 S 의 0 번째 인덱스부터 일치하는 경우는 특이케이스로 분류하였다. match 함수에 처음 진입하면 S 의 0 번째 인덱스가 P 의 0 번째 인덱스와 같은 지 비교한다. 만약 같다면 기존 코드와 동일하게 l 과 r 값을 0 으로 초기화하여 선언하고, while 문을 통해 계속 같은 지 비교하고, P 가 S 에 존재함이 증명되면 0 을 return 한다. 만약 중간에 매칭이 안되어 반복문을 탈출한다면 다시 l 을 0 으로 초기화하고 기존의 코드 수행과 동일하게 진행하도록 한다.

이제 여러 테스트케이스를 통해 예외나 버그가 없는지 확인해보자

1. 패턴 P 가 문자열 S 의 중간에서 매칭되는 경우

```
17    pattern = "go"
18    str = "algorithm"
19    result = match(pattern,str,len(str),len(pattern))
```

패턴 : “go”

문자열 : “algorithm”

결과

```
[go] 라는 패턴은 문자열에 존재하네요!
해당 패턴이 문자열 [algorithm]의 2번째 인덱스부터 시작합니다.
PS C:\Users\염민규\Desktop\코딩 툴\git\Python>
```

정상적으로 작동함을 알 수 있다.

2. 패턴 P 가 문자열 S 의 0 번째 인덱스부터 매칭되는 경우

```
17    pattern = "alg"
18    str = "algorithm"
19    result = match(pattern,str,len(str),len(pattern))
```

패턴 : “alg”

문자열 : “algorithm”

결과

```
[alg] 라는 패턴은 문자열에 존재하네요!
해당 패턴이 문자열 [algorithm]의 0번째 인덱스부터 시작합니다.
PS C:\Users\염민규\Desktop\코딩 툴\git\Python>
```

이전과는 달리 버그가 해결되어 정상적으로 작동함을 알 수 있다.

3. 패턴 P 가 문자열 S 의 맨 끝 위치에서 매칭되는 경우

```
28    pattern = "thm"
29    str = "algorithm"
30    result = match(pattern,str,len(str),len(pattern))
```

패턴 : “thm”

문자열 : “algorithm”

결과

```
[thm] 라는 패턴은 문자열에 존재하네요!
해당 패턴이 문자열 [algorithm]의 6번째 인덱스부터 시작합니다.
PS C:\Users\염민규\Desktop\코딩 툴\git\Python>
```

정상적으로 작동함을 알 수 있다.

4. 패턴 P 가 문자열 S 에 존재하지 않는 경우

```
28  pattern = "lol"
29  str = "algorithm"
30  result = match(pattern,str,len(str),len(pattern))
```

패턴 : “lol”

문자열 : “algorithm”

결과

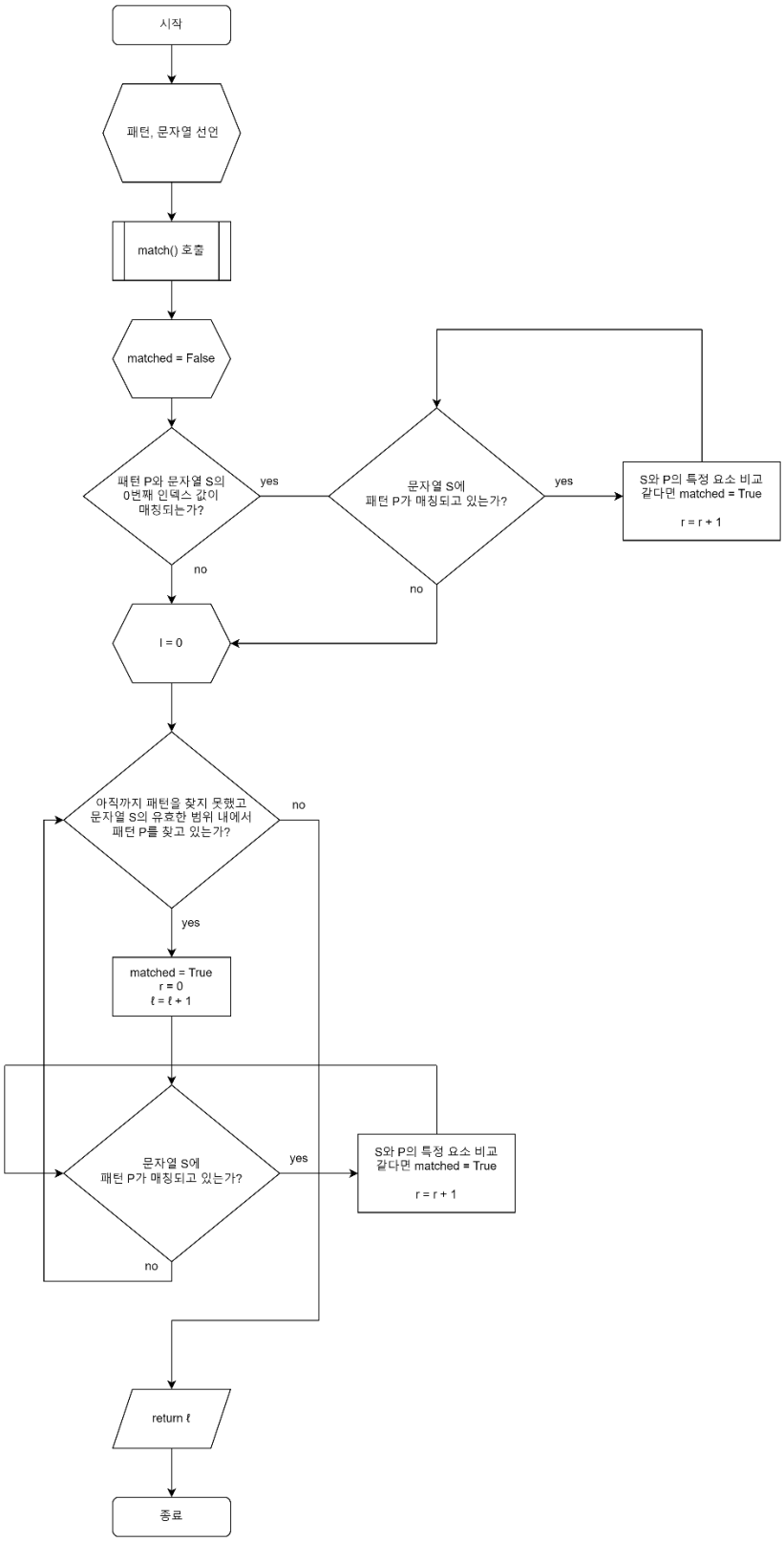
```
패턴 [lol]이 문자열 [algorithm]에 존재하지 않습니다!
PS C:\Users\염민규\Desktop\코딩 툴\git\Python>
```

정상적으로 작동함을 알 수 있다.

처음 짰던 코드는 POA262 ver.1 이라고 이름 짓고, 어떤 버그도 허용하지 않도록 보완한 이 코드는 POA262 ver.2 라고 짓겠다.

그럼 POA262 ver.2 의 순서도를 갱신해줄 필요가 있다. 갱신 시켜주자.

POA262 ver.2 순서도



시간복잡도

POA262 ver.1 은 시간복잡도가 최악의 경우는 어떻게 될까? 예를 들어보자. 문자열 S 는 "aaaaaaaaaa"이고, 패턴 P 는 "aaaab"라고 가정하겠다. 이럴 경우 문자열 S 의 모든 부분에서 P 의 0 번째 인덱스 요소와 같기 때문에 모든 반복문을 거치게 되고, 반복문 안에서도 P 의 마지막 인덱스까지 반복문이 도달해야 비로소 반복문을 탈출하게 된다. 그리고 이는 S 문자열의 끝부분까지 지속되다 결국 "패턴이 존재하지 않는다"는 결론에 도달하게 된다.

가장 바깥쪽 반복문을 도는 횟수는 S 의 길이 – P 의 길이 즉, 5 번이고, 한 번 반복문이 돌 때마다 안쪽 반복문을 도는 횟수는 P 의 길이, 즉 5 번이다. 이는 S 의 길이를 n 이라고 가정했을 때 (n/2) ^ 2 만큼의 시간을 소요하며, 이는 O(n^2/4) 즉, O(n^2)만큼의 시간복잡도를 가진다는 것을 알 수 있다.

그렇다면, 내 입맛대로 보완한 POA262 ver.2 의 시간복잡도는 어떻게 될까?

최악의 경우를 생각해보자. 문자열 S 는 "aaaaaaaaaa"이고, 패턴 P 는 "aaaab"이라고 동일하게 가정하자. 첫 if 문을 만족하여 패턴이 끝까지 매칭되는지 알아보기 위해 반복문을 돌렸으나 끝내 마지막 인덱스에서 일치하지 않는다는 것을 알아버렸다. 이럴 경우, n/2 만큼의 시간을 잡아먹었고, 이후 실행되는 코드는 기존과 같다. 그렇다면 T = n/2 + n^2 로, n 이 충분히 커진다면 n/2 는 n^2 에 비해 무시할 수 있을 정도로 작아질 것이다. 그러므로 POA262 ver.2 는 최악의 경우 O(n^2)의 시간복잡도를 가진다.

소감

솔직히 재밌었다. 내가 배우는 알고리즘 중, 하나를 그 어느 알고리즘보다 정밀하게 분석해보고, 코드로 구현도 해보았으며, 막상 또 구현을 해보니 결점이 존재한다는 것을 발견하였고, 이를 보완하기 위해 머리를 굴려 나만의 알고리즘으로 완성시키는 일련의 과정이 정말 재밌었다.
그리고 내가 만든 알고리즘의 시간복잡도도 내가 스스로 계산해보려 노력하니 내가 알고리즘 수업을 듣고 있다는 체감이 확 와 닿았다. 비록 내가 짠 알고리즘이 효율적이지 못할 수도 있다. 더 간결하고 효율적인 코드가 존재할 지도 모른다. 더 나아가서 순서도가 틀렸을 수도 있다.
시간복잡도 계산도 엉터리일 수 있다.(이는 과제를 올리면서 시간복잡도 계산이 맞는지 여쭤볼 생각이다.) 설계 과정마저 전체적으로 허술할 수도 있다. 허나 이 일련의 과정을 통해 변수

하나하나가 어떤 의미를 갖는지 까지 누구에게 설명해주기 위한 설계를 해보았다는 점. 내 코드를 내가 리뷰까지 하는 이 경험은 앞으로 도움이 안 될 수가 없다고 생각한다. 며칠전에 마이크로소프트에서 일하는 한 직원의 하루 브이로그 영상을 유튜브에서 보았다. 오후 일과 중, 화상회의를 통해 각자 자신의 코드를 리뷰하는 장면을 보았는데 서로 자신의 코드를 리뷰하며 이런 순서도를 가지고, 어느 시간복잡도를 가지며, 이런 점이 아쉬운데 보완할 수 있을까 라는 주제로 각자 얘기를 하고 있는 모습을 보았다. 너무 멋있었다. 아마 그런 모습이 이 과제와 비슷하다 라고 보기엔 힘들겠지만, 아주아주 크게 보면 내가 오늘 진행한 이 과제를 가지고 누구에게 리뷰를 진행한다면, 좋게 말해 비슷하게 보일지도 모른다고 생각한다.