자료구조 과제 4

* KMP 알고리즘 –

20151523

김동현

- 실행결과

1) 제공된 data

1-1) 성공한 case

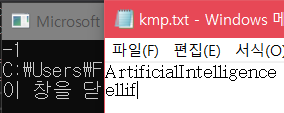
시계이(가) 표시된 사진

자동 생성된 설명

스크린샷이(가) 표시된 사진

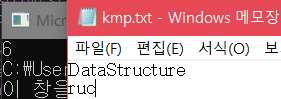
자동 생성된 설명

1-2) 실패한 case



2) 임의의 data

2-1) 성공한 case



2-2) 실패한 case

스크린샷이(가) 표시된 사진

자동 생성된 설명

- 코드 및 알고리즘 설명

#include <stdio.h>

#include <string.h>

#define max\_string\_size 100

#define max\_pattern\_size 100

int pmatch(char \*string, char \*pat);

void fail(char \*pat);

int failure[max\_pattern\_size];

char string[max\_string\_size];

char pat[max\_pattern\_size];

: 사용되는 함수는 pmatch와 fail 2가지 이다. 그리고 문자열을 받을 string과 패턴 문자열을 받을 pat, failure data를 담을 failure를 각각 타입에 맞추어 전역 array 변수로 선언한다.

int main() {

FILE\* fp = fopen("kmp.txt", "r");

fgets(string, sizeof(string), fp);

fgets(pat, sizeof(pat), fp);

fail(pat);

printf("%d", pmatch(string, pat));

fclose(fp);

return 0;

}

: main함수에서는 간단하게 파일포인터를 설정하여 kmp.txt의 문자열을 fgets로 읽어와서 string과 pat에 각각 저장한다. 그 다음 fail 함수를 통해 pat을 argument로 전달하여 failure 배열에 pat을 바탕으로 failure data를 fail()함수를 통해 저장한다. 그 다음 pmatch의 결과를 바로 출력해줌으로써 실행 결과를 사용자가 얻게 되고, 파일포인터를 비워줌으로써 프로그램은 종료된다.

int pmatch(char \*string, char \*pat)

{

int i = 0, j = 0;

int lens = strlen(string);

int lenp = strlen(pat);

while (i < lens && j < lenp) {

if (string[i] == pat[j]) {

i++; j++;

}

else if (j == 0) i++;

else j = failure[j - 1] + 1;

}

return ((j == lenp) ? (i - lenp) : -1);

}

: string 문자열과 pat 문자열을 비교하는 함수이다. I를 string배열 인덱스, j를 pat배열 인덱스로 사용하여서 i가 string의 길이를 넘거나 j가 pat의 길이를 넘으면 종료하게 되어있다. While 반복문 안에서 만일 해당 I,j의 인덱스에서 string과 pat의 문자가 일치할 경우, 두 문자열 모두 그 다음 문자를 비교하기 위해 i++, j++을 해준다. 일치하지 않을 경우는 2가지로 나뉘는데 그때의 pat의 글자가 첫번째 글자인 경우 단순히 string을 한 칸 당겨서 다시 비교하면 되므로 i++만 해주는데, pat이 첫번째 글자가 아닌 경우, 다시 처음 비교했던 string의 i의 다음 칸으로 가는 것이 아닌, failure data를 통하여 정보가 확인된 위치에서의 비교를 굳이 하지 않고, 최적화된 위치로 j를 초기화 하여준다.

잠깐 여기서 예를 들어서 KMP 알고리즘을 설명하자면 string이 “abcabde..”이고, pat이 “abcabc”인 경우, i가 0~4까지 일치하므로 i++, j++이 되다가, I,j = 5일 때 d != c로 틀리게 된다. 기존의 문자열 검색 알고리즘이라면 i=1로 돌아가서 다시 pat과 비교할테지만, KMP알고리즘에서는 failure 데이터를 바탕으로 pat의 접두사, 접미사가 같은 문자열 중 가장 큰 길이를 이용하는데, abcabc에서는 ‘abc’를 공통된 접두사, 접미사로 가지므로 failure[5] = 2가 된다. 따라서 string에서 i=5인 c를 pat의 j=2인 c에 맞추어 계속하여 비교해주면 필요없이 처음부터 다시 비교해야 할 필요가 없어지게 된다. 그래서 j= failure[j-1] + 1을 통해 j를 최적화된 위치로 초기화 해주는 것이다.

마지막 반환 값은 만일 j가 pat의 길이와 같다면, 즉 다시 말하여 pat의 모든 글자를 다 비교했는데 반복문이 종료되었다면 무사히 해당 pat을 검색하는 데에 성공하였으므로 그때의 string 시작점을 반환하고, 아닐 시에는 실패하였다는 의미이므로 -1을 반환한다.

void fail(char \*pat)

{

int i, n = strlen(pat);

int j;

failure[0] = -1;

for (j = 1; j < n; j++) {

i = failure[j - 1];

while ((pat[j] != pat[i + 1]) && (i >= 0))

i = failure[i];

if (pat[j] == pat[i + 1])

failure[j] = i + 1;

else failure[j] = -1;

}

}

: fail 함수는 위의 알고리즘 설명에서 하였 듯이, pat의 접두사, 접미사를 분석하여 failure data를 미리 설정하는 역할을 한다. 문자가 하나일때(j=0)는 고려하지 않으므로 -1로 초기화한 뒤, pat[j]이 pat[i + 1]과 같다면 접두사와 접미사가 겹친다는 의미이므로 +1씩 해주다가 그 다음 j에서 접두사와 접미사를 비교할 때에 만약 다른 문자가 나오면 다시 아예 접두사와 접미사가 겹치지 않게 되므로 -1로 돌아간다.