CSE2011 Problem Solving, 2016 Spring

2013312343 이상헌

**Homework 3-2**

**1. 문제 이해**

(1) 문제

**Problem 3-2: Rotating a String**

- We are trying to rotate a string P times (to the left). For example, if we rotate a string “ABCDEFG” 3 times, the resulting string is “DEFGABC”. For a given string and P, develop an algorithm to rotate a string only using multiple swaps of two characters so as to minimize the number of swaps. (You can only swap two characters in the string).

- For example, if a string is “ABC” and P=1, then the minimum number of swaps is 2. (ABC -> CBA -> BCA).

- The length of P is less than 1000.

(2) 중요 정보

- 문자열과 rotate 수를 입력 받는다.

- 입력된 문자열의 길이는 1000 이하이다.

- 문자열 내의 임의의 두 문자를 swap하고, count한다.

(3) 문제 정의

- input 정보를 통해 최소의 swap으로 문자열을 rotate 수만큼 rotate하는 경로 및 swap 횟수를 출력하는 프로그램을 구현한다.

**2. 문제 해결**

(1) 해결 방법 결정

*조건에 따라 경우를 나눈다.*

- 최소의 swap으로 문자열을 rotate하는 방법을 생각해보던 중 입력 받는 조건에 따라 경우를 나눈다면, 각 경우에 대해서는 동일한 방법으로 경로를 파악할 수 있다는 사실을 발견하였다. 문제에서의 input은 문자열과 왼쪽으로의 rotate 수이다. 문자열의 길이와 rotate 수와의 대소를 통해 경우를 나눈다.

- 아이디어

문자열 내에 rotate 수만큼의 위치를 기준으로 오른쪽과 왼쪽의 문자열로 나눈다. (문자열 안에서 rotate 수 번째의 문자를 기준점이라 하자. 기준점은 왼쪽 문자열에 포함된다.) 입력 받은 rotate 수의 크기에 따라 오른쪽과 왼쪽의 문자열 중에 어떤 문자열을 먼저 옮겨야 하는지를 생각한다. 즉, 입력 받은 문자열의 길이의 절반과 rotate 수와의 대소를 비교하여 경우를 나눈다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | **C** | D | E | F | G | H |

왼쪽 문자열

오른쪽 문자열

Table 1. 문자열 ABCDEFGH, rotate 수 3을 입력 받은 경우의 기준점

만일 문자열의 길이의 절반보다 rotate 수가 더 작다면, 기준점의 오른쪽에 위치하는 문자열을 먼저 제 자리로 가도록 swap한다. 기준점의 오른쪽에 있는 문자열의 가장 왼쪽부터 오른쪽 방향으로 각각의 문자를 rotate 결과에 위치하도록 swap한다. 이 단계를 수행하면 문자열은 왼쪽부터 (문자열 길이 – rotate 수)번째의 문자까지는 모두 rotate 결과와 동일한 위치에 위치한다. Rotate 결과에 정확히 위치한 문자 이외의 나머지 문자들(원래 기준점의 왼쪽에 위치한 문자열)은 똑같은 방법으로 왼쪽부터 오른쪽 방향으로 rotate 결과와 동일한 위치에 있도록 swap한다.

예를 들어, ABCDE라는 문자열과 2라는 rotate 수를 입력 받았다고 하자. Rotate 수인 2는 문자열의 절반인 5/2보다 작기 때문에, 기준점의 오른쪽 문자열의 문자들을 왼쪽에서 오른쪽 방향으로 먼저 swap한다. (rotate 수는 2이므로 기준점의 왼쪽 문자열은 AB, 오른쪽 문자열은 CDE가 된다.) Table 2는 각각의 swap하는 문자 및 단계를 나타낸다. 빨간색으로 표시된 문자들이 swap되어 다음 문자열을 나타내고, 밑줄 그어진 문자열은 기준점의 오른쪽 문자열들을 나타낸다.

|  |  |
| --- | --- |
| Swap 수 | 문자열 |
| 0 | **A** B **C** D E |
| 1 | C **B** A **D** E |
| 2 | C D **A** B **E** |
| 3 | C D E B A |

Table 2. 기준점의 오른쪽 문자열을 제자리에 위치하도록 swap.

이후, 기준점의 왼쪽에 위치해 있었던 문자열(AB)을 rotate 결과에 만족하는 위치에 있도록 swap해야한다. (문자열의 길이 – rotate 수)번째 문자 다음의 문자부터 문자열 끝까지 원래 있던 문자를 검사하고, 각각의 순서에 맞도록 swap한다. B의 위치부터 AB의 첫 번째 문자인 A가 나타날 때까지 왼쪽으로 검사한 후, A가 나타나면 첫 번째인 B의 위치와 swap한다. 예시의 경우, A와 B만 swap 하면 결과가 나타나지만, 보통의 경우 여러 번의 swap이 일어날 것이다.

|  |  |
| --- | --- |
| Swap 수 | 문자열 |
| 3 | C D E **B** **A** |
| 4 | C D E A B |

Table 3. 나머지 문자열을 제자리에 위치하도록 swap.

만일 문자열의 길이의 절반보다 rotate 수가 더 크다면, 반대로 기준점의 왼쪽에 위치하는 문자열부터 제자리에 위치하도록 swap한다. 여기서, 문자열이 정확한 위치에 있도록 하기 위해서 기준점의 왼쪽에 위치하는 문자열의 가장 오른쪽 문자부터 왼쪽 순으로 swap한다. 이후 제자리에 위치하지 않은, 기준점의 오른쪽에 위치해 있던 문자열을 왼쪽부터 검사하여 제자리에 위치하도록 swap한다. Table 4는 위와 같은 과정을 나타내었다.

|  |  |
| --- | --- |
| Swap 수 | 문자열 |
| 0 | A B **C** D **E** |
| 1 | A **B** E **D** C |
| 2 | **A** D **E** B C |
| 3 | **E** **D** A B C |
| 4 | D E A B C |

Table 4. 문자열의 길이/2 < rotate 수인 경우 swap 과정.

만일 문자열의 길이의 절반과 rotate 수가 같다면, 기준점의 왼쪽에 위치한 문자열과 오른쪽에 위치한 문자열의 길이는 같다. 이는 문자열의 길이의 절반만큼의 swap을 한다면 모든 문자가 rotate 결과의 위치에 위치하도록 된다.

|  |  |
| --- | --- |
| Swap 수 | 문자열 |
| 0 | **A** B C **D** E F |
| 1 | D **B** C A **E** F |
| 2 | D E **C** A B **F** |
| 3 | D E F A B C |

Table 5. 문자열의 길이/2 = rotate 수인 경우 swap 과정.

- 이점

문자열의 길이와 rotate 수는 항상 대소가 정해지기 때문에 전체 문자열을 두 부분으로 나눌 수 있다. 최소의 swap으로 문자열을 rotate하기 위해서는 나누어진 두 문자열 중에 길이가 긴 문자열을 먼저 swap 한 후에, 나머지 문자를 swap해야 한다. 이는 길이가 짧은 문자열을 먼저 swap한다면, 정확한 위치에 자리잡을 수 없을 가능성이 있기 때문이다. 또한 마지막 swap 과정에서 문자가 정확한 위치에 있는 경우에는 swap을 하지 않음으로써 swap수를 최소화할 수 있다.

**3. 코드 작성**

- 본 아이디어 및 설명을 바탕으로 코드를 작성하였다.

int cal\_S(int p, char s[], int swap1[], int swap2[]) {

int n = 0, len = 0, i = 0, j = 0;

char s\_chg[1000], temp;

n은 swap 횟수를 저장하는 변수이다. Len은 입력 받은 문자열의 길이를 저장하는 변수이다. I, j는 반복문, 조건문 등에서 쓰이는 count variable이다. Swap 과정 중의 문자열을 저장할 필요성을 느끼고, s\_chg라는 문자열을 새로 만들었다. Temp는 swap과정에서 필요한 변수이다.

while (s[len] != '\0') len++;

입력 받은 문자열의 길이를 계산한다. 문자열의 맨 끝에는 ‘\0’값이 저장되기 때문에, ‘\0’이 나타날 때까지 count한다.

if (len < p) p %= len;

if (p%len == 0) return 0;

만일 입력 받은 rotate 수가 문자열의 길이보다 크다면, rotate 수를 문자열의 길이로 나눈 나머지를 새로 저장하였다. 또한 rotate 수가 문자열 길이의 배수이거나 0이면 rotate할 필요가 없기 때문에 0을 리턴 하고 함수를 종료한다.

for (i = 0; i < len; i++) {

s\_chg[i] = s[i];

}

S\_chg라는 문자열에 입력 받은 s 문자열을 복사한다. S\_chg 문자열은 후에 swap 중간 과정의 문자열 상태를 저장하는 데에 쓰인다.

if (len / 2 > p) {

for (i = 0; i < len - p; i++) {

swap1[n] = i;

swap2[n] = p + i;

temp = s\_chg[swap1[n]];

s\_chg[swap1[n]] = s\_chg[swap2[n]];

s\_chg[swap2[n]] = temp;

n++;

}

for (i = 0; i < p - 1; i++) {

j = 0;

while (1) {

if (s[i] == s\_chg[len - p + i + j])

break;

j++;

}

if (j != 0) {

swap1[n] = len - p + i;

swap2[n] = len - p + i + j;

temp = s\_chg[swap1[n]];

s\_chg[swap1[n]] = s\_chg[swap2[n]];

s\_chg[swap2[n]] = temp;

n++;

}

}

}

문자열의 길이의 절반보다 rotate 수가 더 작은 경우이다. 첫 번째 for문을 통해 기준점을 기준으로 오른쪽에 위치한 문자열의 가장 왼쪽 문자부터 제자리에 가도록 swap 한다. Swap1과 swap2에 위치를 저장함과 동시에 s\_chg 문자열의 문자도 바꿈으로써, 현재 문자열의 상태를 저장한다. 두 번째 for문에서는 나머지 문자열의 문자들을 제자리에 가도록 swap한다. 나머지 문자열의 가장 왼쪽부터 시작하여, 첫 번째 문자가 나타날 때까지 j를 count한다. 위치를 찾은 경우 정확한 자리와 swap 한다. 만일 문자가 정확한 위치에 있다면 j=0이 되고, swap을 하지 않는다.

else if (len / 2 < p) {

for (i = 0; i < p; i++) {

swap1[n] = p - i - 1;

swap2[n] = len - i - 1;

temp = s\_chg[swap1[n]];

s\_chg[swap1[n]] = s\_chg[swap2[n]];

s\_chg[swap2[n]] = temp;

n++;

}

for (i = 0; i < len - p - 1; i++) {

j = 0;

while (1) {

if (s[p + i] == s\_chg[i + j])

break;

j++;

}

if (j != 0) {

swap1[n] = i;

swap2[n] = i + j;

temp = s\_chg[swap1[n]];

s\_chg[swap1[n]] = s\_chg[swap2[n]];

s\_chg[swap2[n]] = temp;

n++;

}

}

}

문자열의 길이의 절반보다 rotate 수가 더 큰 경우이다. 첫 번째 for문을 통해 기준점의 왼쪽에 위치하는 문자열의 가장 오른쪽 문자부터 제자리에 위치하도록 swap한다. 두 번째 for문에서는 나머지 문자들을 제자리로 가도록 swap한다. 이는 문자열의 길이의 절반보다 rotate수가 더 작은 경우와 동일하게 진행한다.

else {

for (i = 0; i < p; i++) {

swap1[n] = i;

swap2[n] = i + p;

n++;

}

}

만일 문자열의 길이의 절반과 rotate 수가 같다면, 문자열의 길이의 절반만큼의 swap을 통해 rotate 결과를 얻을 수 있다.

return n;

}

**4. 확인 및 분석**

- 본 아이디어를 이용하여 프로그램을 작성하였다. 이 프로그램이 정확한 결과값을 도출하는지를 알아보기 위해서 간단한 예시를 입력하고, 결과를 비교하였다.

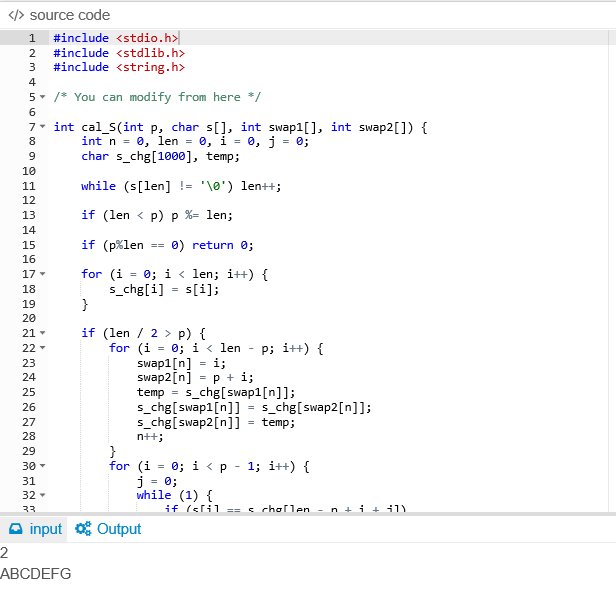


Figure 1. 예시 1 Input

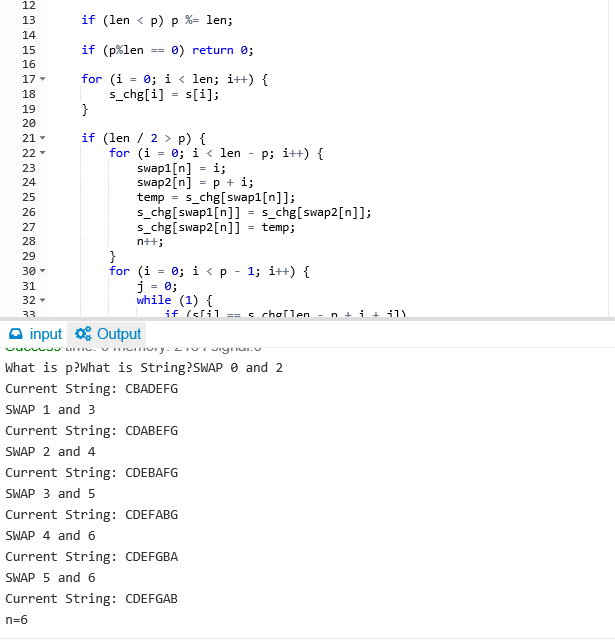


Figure 2. 예시 1 Output

- Figure 1은 rotate 수에 2를, 문자열에 ABCDEFG를 입력한 경우이다. 문자열의 길이의 절반보다 rotate 수가 더 작기 때문에 첫 번째 경우에 해당한다. 문자열을 AB/CDEFG로 나누고, 오른쪽에 위치한 CDEFG의 문자들을 제자리에 위치하도록 가장 왼쪽 문자부터 swap한다. (C -> D-> E -> F -> G) 이러한 결과로 Figure 2의 CDEFGBA라는 문자열이 나타난다. 이후에 BA를 swap하여 CDEFGAB라는 정확한 결과가 나타나는 것을 확인할 수 있다.

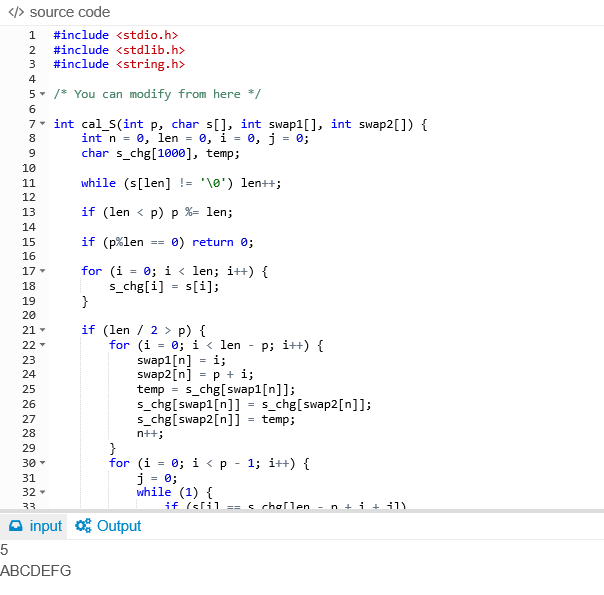


Figure 3. 예시 2 Input

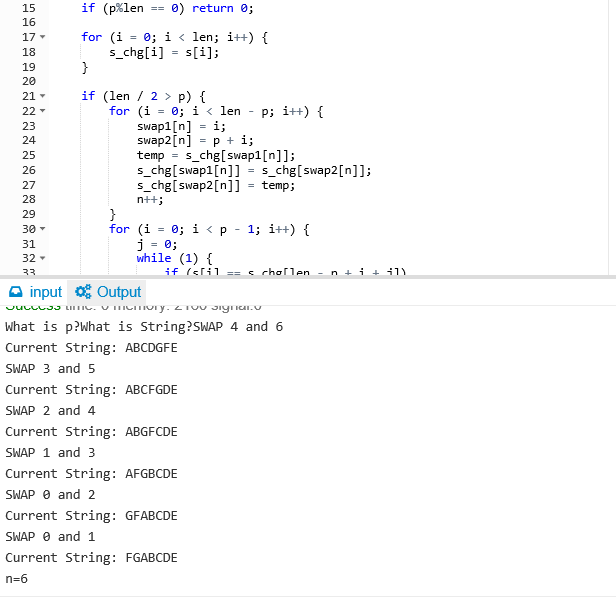


Figure 4. 예시 2 Output

- Figure 3은 rotate 수에 5를, 문자열에 ABCDEFG를 입력한 경우이다. 문자열의 길이의 절반보다 rotate 수가 더 크므로 두 번째 경우에 해당한다. 문자열을 ABCDE/FG로 나누고, 먼저 오른쪽 문자열의 문자들을 제자리에 위치하도록 가장 오른쪽 문자부터 swap한다. (E -> D -> C -> B -> A) 이러한 결과로 Figure 4에 GFABCDE라는 문자열이 나타난다. 이후 GF를 swap하여 FGABCDE라는 정확한 결과가 나타났다는 것을 확인할 수 있다.

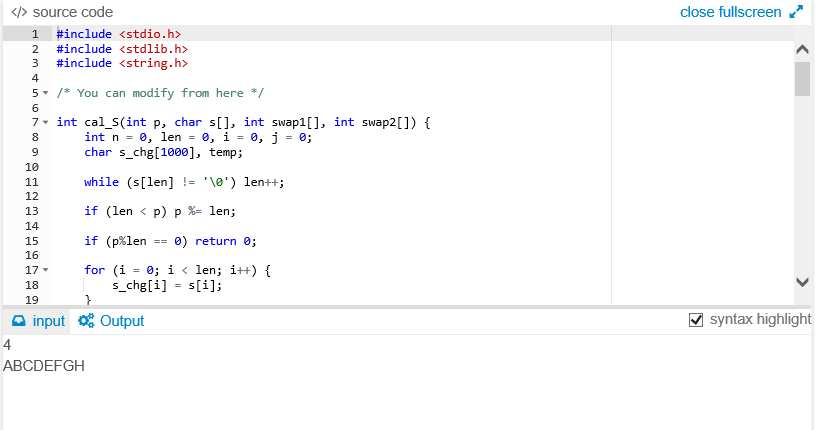


Figure 5. 예시 3 input

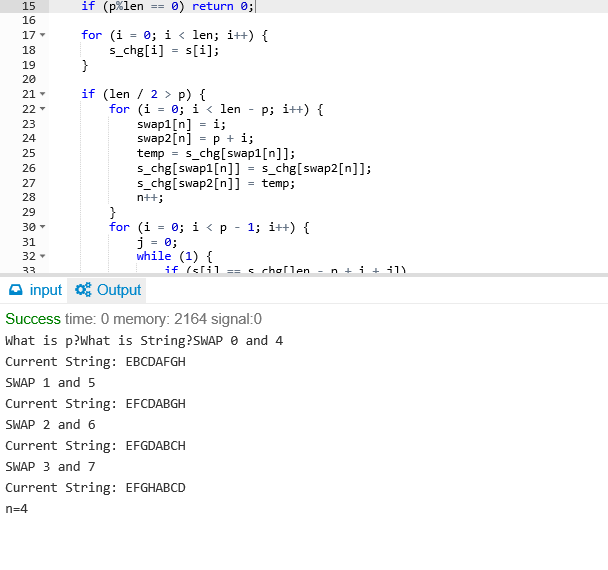


Figure 6. 예시 3 output

- Figure 5는 rotate 수에 4를, 문자열에 ABCDEFGH를 입력한 경우이다. 문자열의 길이의 절반과 rotate 수가 같으므로 세 번째 경우에 해당한다. 문자열을 ABCD/EFGH로 나누고, A와 E, B와 F, C와 G, D와 H를 swap 한다. 그 결과 Figure 6와 같이 문자열의 길이의 절반인 4번의 swap으로 정확한 rotate 결과를 나타낸다는 것을 확인할 수 있다.

**5. 고찰**

- 본 문제는 입력 받은 문자열과 rotate 수를 바탕으로, 최소한의 문자 swap을 통해 문자열을 rotate 수만큼 왼쪽으로 rotate하는 과정을 출력하는 프로그램을 구현하는 것이다. Rotate 과정은 두 문자의 swap으로만 이루어진다는 것이 본 문제의 특별히 고려해야 할 점이다. 문자열의 길이와 rotate 수의 크기를 비교하여, 세 가지 경우로 나누어 문제를 해결하였다. 특히 문자열을 두 부분으로 나누고 긴 문자열을 먼저 swap함으로써, swap 수를 최소화하였다. 본 아이디어를 통한 프로그램이 최소한의 swap을 충족하는지는 미지수이지만, 정확한 결과를 나타낸다는 것을 여러 번의 시행을 통해 확인하였다. 문자열을 두 부분으로 나누는 것이 아닌, 세 부분 혹은 네 부분으로 나누어 swap 해야 하는 우선순위에 따라 swap을 하는 아이디어도 생각해보았지만, 구현에 어려움을 느꼈다.

**6. 참고문헌 및 사용**

- [www.ide.com](http://www.ide.com) : C언어 구현 및 실행.

- 이진규 교수님, [문제해결기법] CSE2011\_2016spring\_Lecture\_Note05, 2016