Stack 을 활용한 문자열 검열

201811567 주용한

전체 풀이 개요 및 아이디어

문제의 알고리즘을 나열해 보면

1. 앞에서부터 순차 탐색, 중복 있으면 삭제(없으면 종료)
2. 뒤에서부터 순차 탐색, 중복 있으면 삭제(없으면 종료)
3. 다시 1 번으로 돌아가기 반복

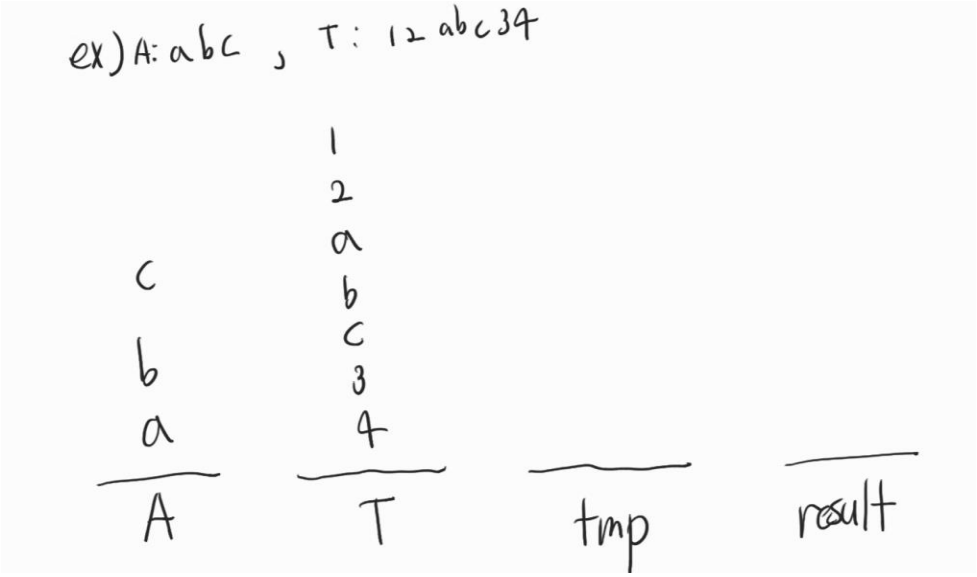
문제 해결을 위해 기본적으로 1 번 기능을 수행하는 재귀함수를 만들어 2 번과 3 번을 같이 해결하는 알고리즘을 만들었다.

사용자로부터 입력 값을 받아 2 개의 A(검열자), T(전체 문자열)스택에 집어넣고 중간다리 역할의 tmp, result 스택 2 개 총 4 개의 스택을 통해 재귀함수를 구현하였다. 결과값은 다시 T 스택에 담긴다.

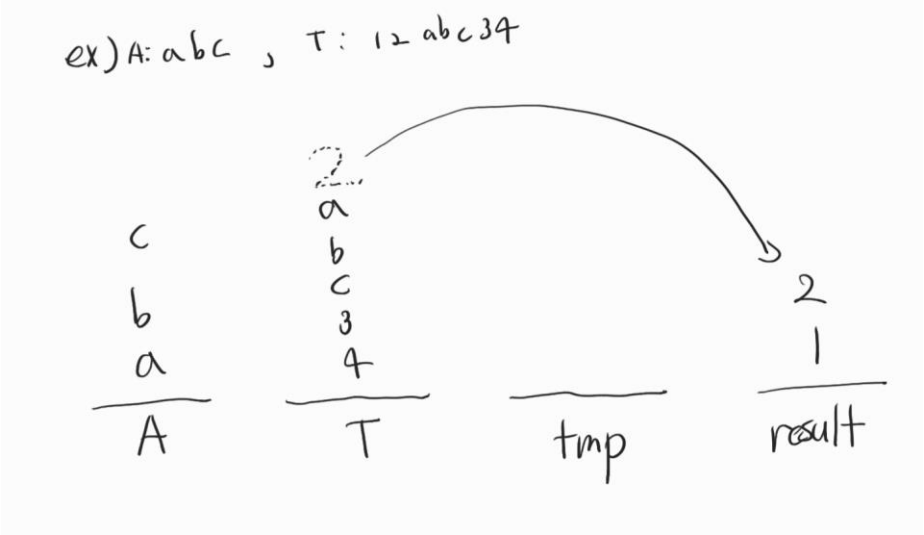
또한 과제 요구사항에 맞춰 입출력을 제외하고는 예를 들어 배열 등 스택 외의 어떠한 자료구조도 사용하지 않고 알고리즘을 구현하였다.

단계적 설명

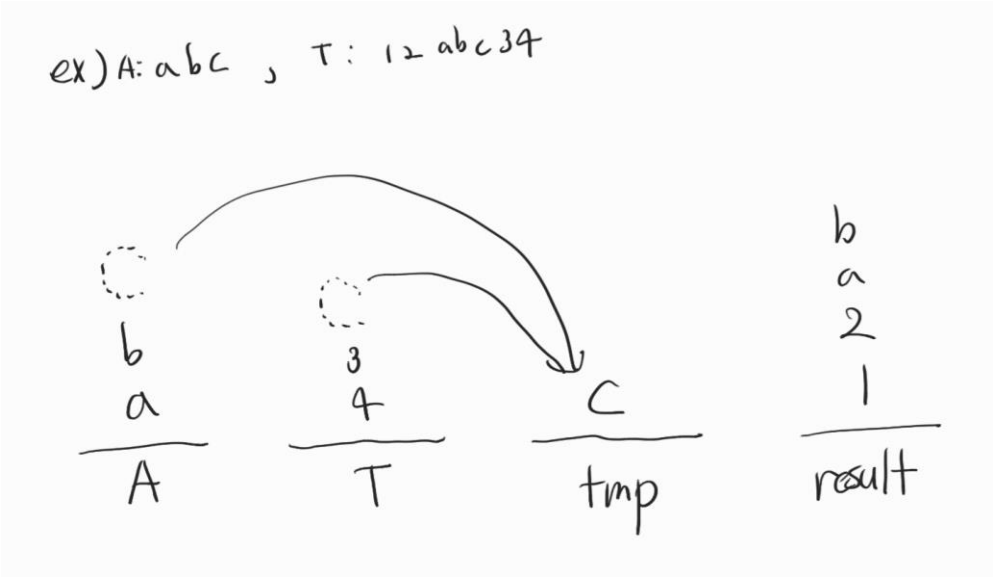
Ex) A : abc, T : 12abc34 라고 하자



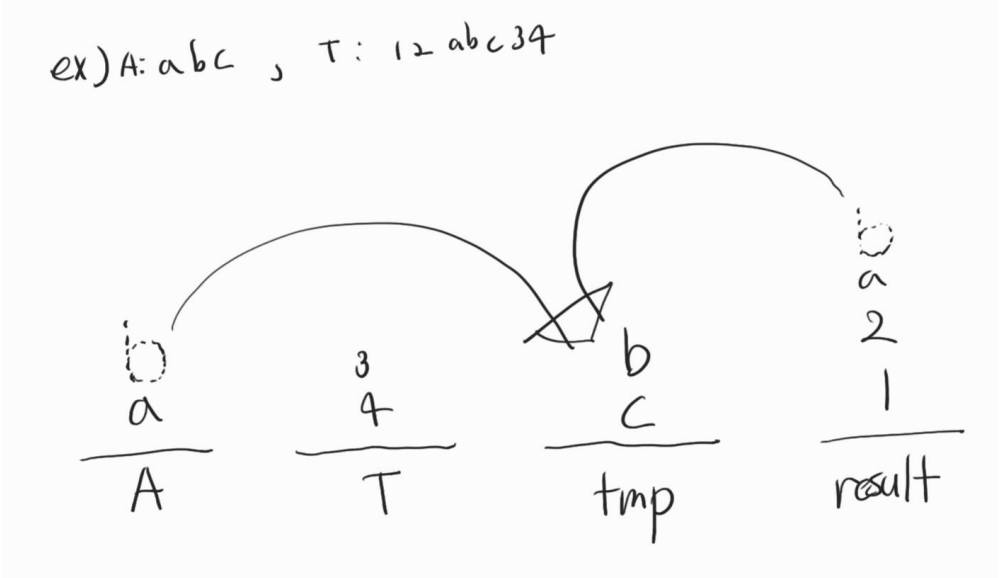
1 단계 구현을 위해 A 는 앞에서부터 스택에 입력하고 T 는 뒤에서부터 스택에 입력한다.



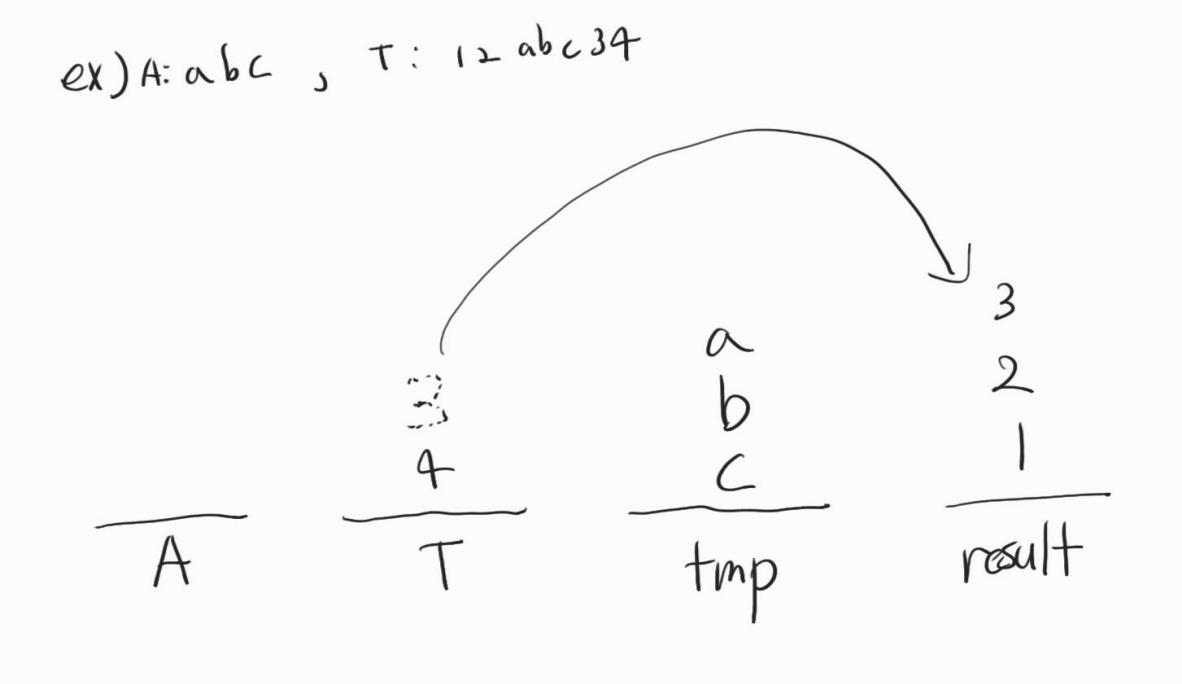
A 의 top 과 T 의 top 이 같아질 때까지 T 를 result 로 옮긴다.



A 와 T 의 top 이 같아지면 A 를 pop 하고 T 를 result 가 아닌 tmp 로 옮긴다.



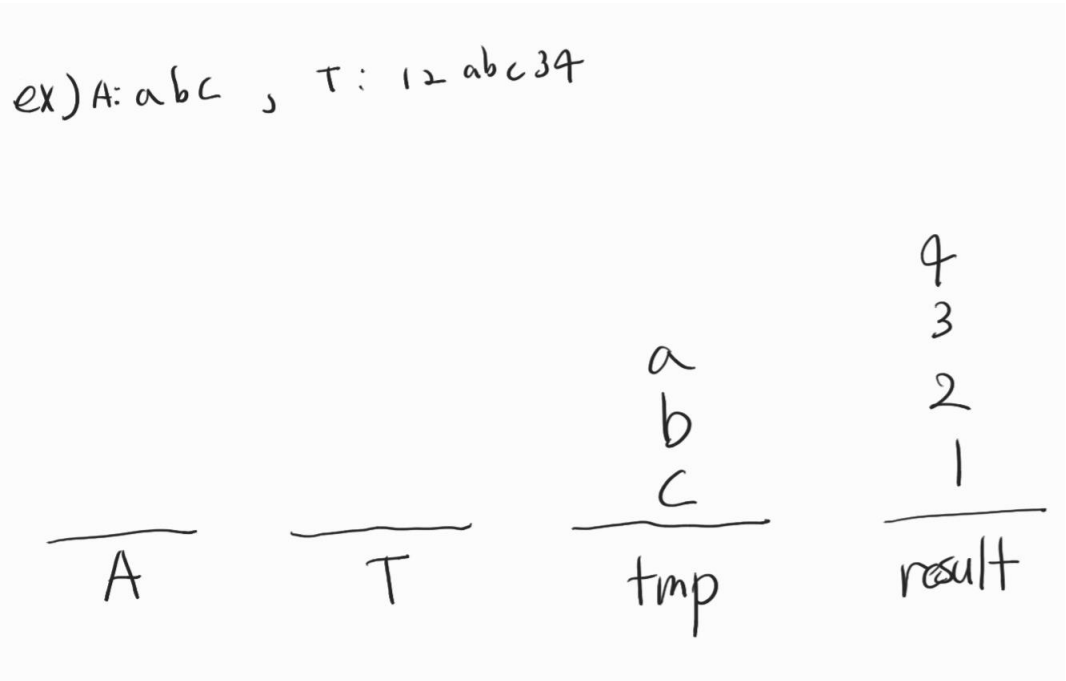
tmp 에 1 개 이상 node 가 생기면 A 와 result 를 비교해서 같으면 tmp 로 옮기기로 한다.



i) 위 그림의 경우 A 를 모두 옮겨 비운 상황 = 같은 문자열이 있을 경우이다. 그럴 경우 A 는 계속 비어 있게 되고 T 는 나머지 node 들을 원래대로 result 에 옮기게 된다. (맨 앞에서 발견된 문자열이 삭제되는 과정)

ii) 만약 A, result 를 tmp 로 옮기는 도중 A 가 다 비워지지 않고 result 와 다른 값을 갖는 상황이 발생한다면 그건 서로 다른 문자열을 갖는 경우 이므로 tmp 의 node 들을 다시 pop 하고 A, result 에 push 한다.

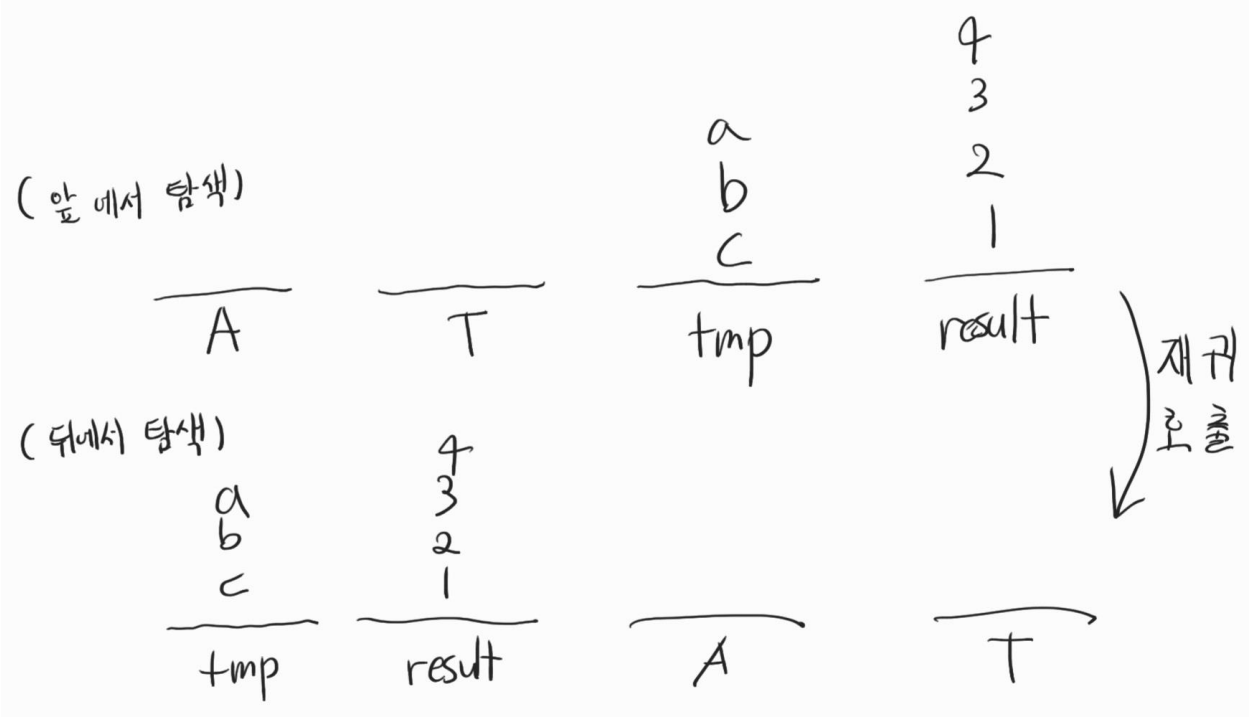
tmp 는 다시 비워져 사용할 수 있게 되고 A 와 result 는 원상복구 된다.



맨 앞의 문자열을 찾아내는 작업을 마치고 나면 A 의 node 는 역순으로 tmp 에 저장되고 T 의 node 들은 역순으로 result 에 저장된다.

만약 위에서 A 가 다 비워지지 않는 경우 = 같은 문자열을 찾지 못한 경우

이므로 함수가 종료된다.



A 의 역순으로 정리된 tmp 와 T 의 역순으로 정리된 result 를 얻고 기존의 A 와 T 는 비워지게 된다.

tmp 자리에 비워진 A 를, result 자리에 비워진 T 를 넣어 재귀호출을 한다.

위 그림대로 반복적으로 재귀호출을 한다면 stack 의 특징을 이용해 앞, 뒤, 앞, 뒤를 반복하며 문자열을 찾아내고 삭제할 수 있다.

문제점과 보완 방법

재귀 호출된 함수는 최종적으로 옮겼던 스택이 result 스택인지 아니면 T 스택인지 알 수 없다. 실행 횟수에 따라 옮겨진 스택이 계속 바뀌며 역순으로 정리될 수 있다.

따라서 함수의 반환형을 int 타입으로 하고 실행 횟수의 짝, 홀을 알려준다.

실제 함수의 head 부분이다.

```
// Parameter : 입력 받는 스택 2 개, 재귀 이용 스택 2 개 총 4 개의 스택과 함수 호출의 횟수 count
int censorship(Stack *stackA, Stack *stackT, Stack *tmp, Stack *result, int count);
```

입력 받은 2 개의 스택과 재귀를 위한 2 개의 스택 총 4 개의 스택을 매개변수로 갖고 맨 마지막의 입력 값으로는 실행 횟수인 count 값을 받는. 재귀호출을 하면서 실행횟수를 다음 함수에게 전달해 주어 최종적으로 몇 번 실행되었는지 홀수 번 실행이면 1 을, 짝수 번 실행이면 0 을 반환한다.

사용자는 int 형 반환 값을 이용해 스택 반환결과를 따로 처리하면 된다.

Ex)

```
int count = censorship();
if(count){
        while(!SIsEmpty(&result)) SPush(&stackT, SPop(&result));       //실행 횟수가 홀수인
}                                                                      //역순 정리
while(!SIsEmpty(&stackT))   printf("%c", SPop(&stackT));               //값 출력
```