SCV/2021

동전 옮기기

- Dynamic Programming -

안태진(taejin7824@gmail.com)
GitHub(github.com/taejin1221)
상명대학교 소프트웨어학과
201821002

Contents

Problem

문제

100 원짜리 동전과 10 원짜리 동전이 임의의 순서로 한 선 위에 나열되어 있다고 하자. 이제 여기서 '**두 손가락 이동**' 을 아래와 같이 정의하자.

- 단계 1: 임의의 두 동전을 선택한다.
- 단계 2: 단계 1 에서 선택한 두 동전을 둘의 순서를 유지한 채 임의의 위치로 이동한다. (두 동전 모두 제자리에 있거나 두 동전의 순서를 유지한다면 하나만 이동해도 된다.)

'두 손가락 이동' 후에도 다른 동전들 간의 순서는 그대로 유지된다. 예를 들어 100 원을 o , 10 원을 x 라 했을 때, 초기에 동전이 oxox**ox**xxoo 와 같이 나열되어 있다 하자. 이제 이들 중 굵게 표시된 두 동전을 선택하여 두 손가락 이동을 한번 한 경우, 나올 수 있는 여러 결과들 중에서 네 가지 결과만 아래에 표시했다 (아래 예시에 없는 다른 결과들 또한 나올 수 있음에 유의하자).

- **ox**oxoxxxoo
- ox**o**oxxx**x**oo
- oxox**o**xxo**x**o
- oxox**ox**xxoo

n 개의 동전이 나열되어 있는 두 상태 *S*, *T*와 함께 두 손가락 이동을 위해 선택할 두 동전의 위치가 주어졌을 때, 한번의 두 손가락 이동을 통해 *S*에서 *T*로의 변환이 가능한지 결정하는 프로그램을 작성하시오.

- 문제 설명
  - 간단히
    - O와 X로 이루어진 문자열이 2개 주어짐
    - 이후 인덱스가 2개 주어짐
    - S에서 그 두개를 움직여서 T를 만들 수 있는지 판단하는 문제
      - 단 그 두개의 순서는 바뀌면 안됨

예제 입력 1 복사

```
9
oxoxoxxxo
oxoxoxxox
4 5
```

- 가능

예제 입력 2 복사

```
10
oxoxxoxxxo
ooxoxxxxox
3 4
```

- 불가능

Problem

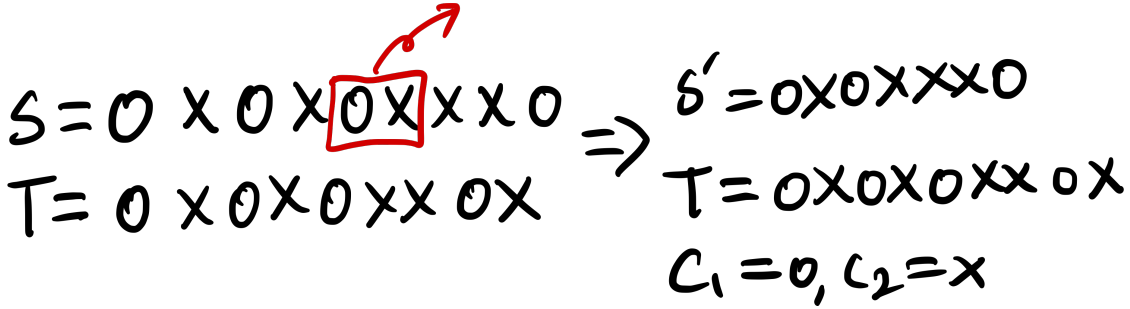
- 문제 설명
  - 작년 ICPC 예선 문제
    - Platinum 미만 중 두번째로 어려운 문제
    - 작년 기준 Platinum 미만을 다 풀면 5문제 => 학교 2등이어도 본선 진출
  - 재작년은 플레 미만이 3개...

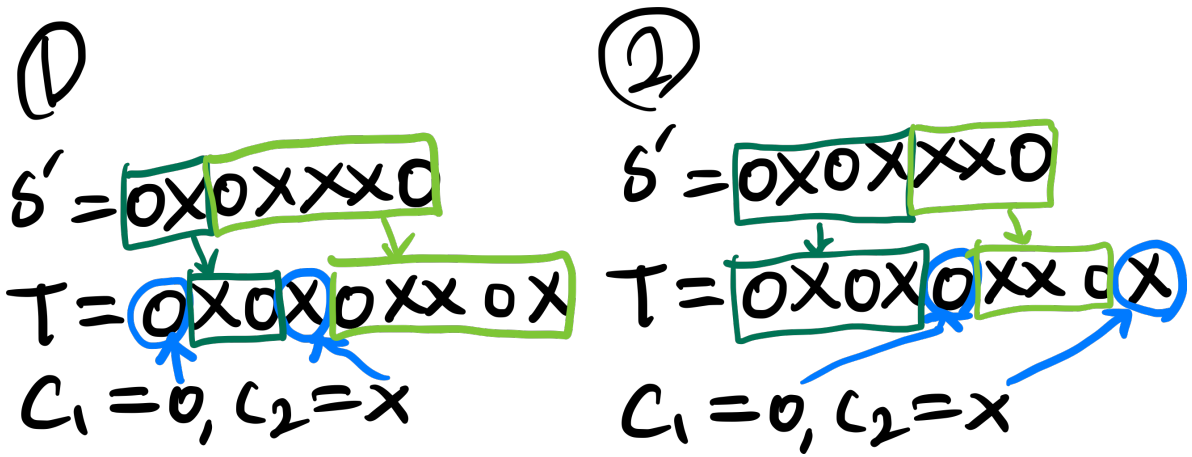
Contents

- Problem
- Approach
- Code

Approach

- Bruteforcing (1/2)
  - 일단 동전 두개를 직접 움직이면서 찾기보단 T에 끼워 맞추기로 함
    - 즉, 앞 동전이 O라면 T의 O 위치에만 넣고 "나머지는 동전 두개 빠진 S와 같은가?" 를 찾기로 함

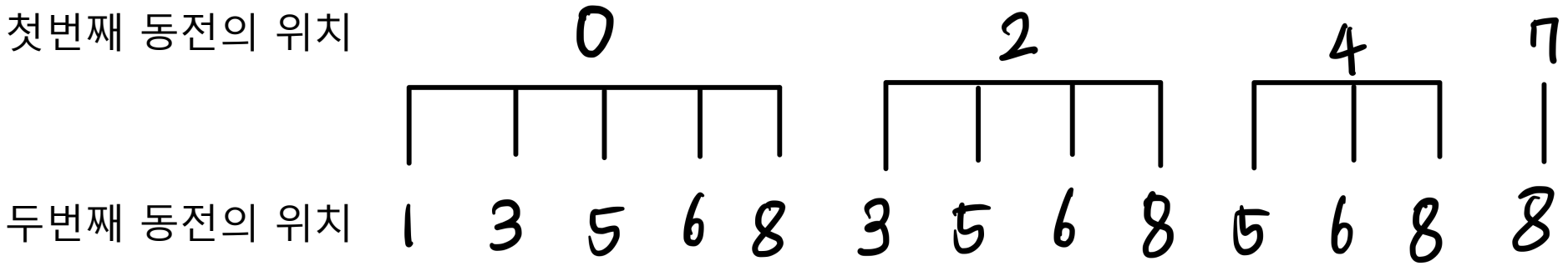




  - S에서 주어진 동전 2개를 빼 S ' 를 만듦
  - 동전 두개의 위치가 결정되면 나머지 동전은 그 순서를 유지한 채 같으면 됨
    - 동전 두개의 위치 정하기 => $O(n^2)$, 나머지 동전 비교하기 =>$O(n)$
    - 시간 복잡도 => $O(n^3)$ 이지만 $n \leq 10{,}000, 0.5s$라 절대 불가능

Approach

- Bruteforcing (2/2)
  - 하지만 재귀적 구조를 찾을 수 있음
    - 다음의 예제
      - S = oxoxoxxxo, T = oxoxoxxox
      - coin1 = 4, coin2 = 5
    - 두번째 동전의 위치는 계속 중복



Approach

- 중복 제거
  - 그렇다면 두번째 동전의 위치에 따라 그 뒤의 결과를 미리 구해놓자!

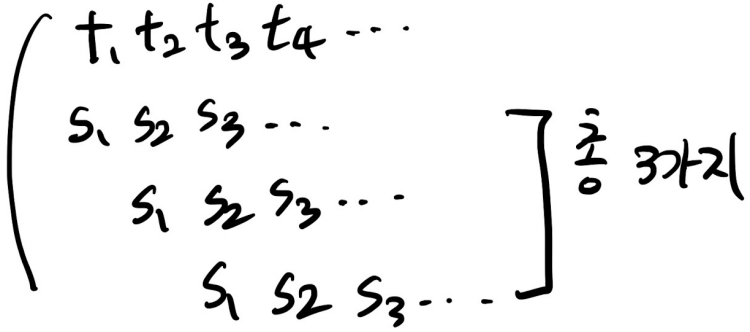
  - 하지만 첫번째 동전의 위치를 결정하고,
  두번째 동전의 위치를 결정하면 결국 최악의 경우 $O(n^2)$

  - 답은 결국 $O(n)$이기 때문에 동전 두개 중 하나의 위치를 결정하고,
  그 뒤의 결과는 $O(1)$으로 알아낼 수 있어야함
    - 그럼과 동시에 문자열 비교 또한 $O(1)$

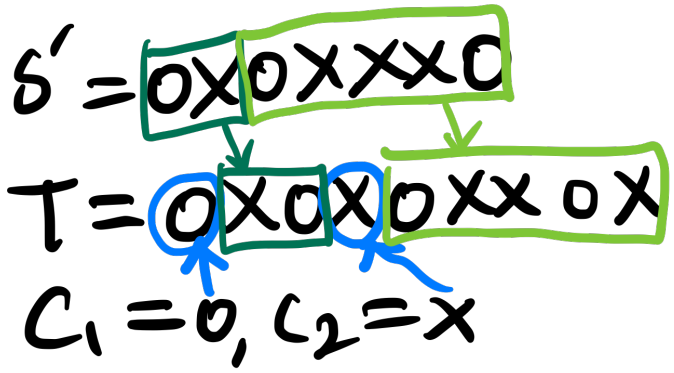
Approach

- DP의 등장
  - 그렇다면 dp[i]를 다음과 같이 설정
    - boolean dp[i] = (두번째 동전을 포함하고 i부터 끝까지 모두 같은가?)
      - 두번째 동전의 위치가 결정되었다면 그 동전의 뒷 부분은 무조건 S'의 뒷 부분과 똑같아야 함
        - 오른쪽 그림에서 X의 위치를 정했기 때문에 연두색 부분은 무슨 일이 있더라도 같아야 함

    - 또한 결국 동전은 두개이므로 모든 문자열에 대해 아래처럼 총 3가지의 경우 밖에 비교 못 함



O	X	O	X	O	X	X	O	X
O	X	O	X	X	X	O		
T	T	T	T	F	T	F		
	O	X	O	X	X	X	O	
	F	F	F	F	T	T	T	
		O	X	O	X	X	X	O
		T	T	T	T	T	F	F



Approach

- DP
  - 점화식
    - $dp[i] = \big((T[i] = C_2) \wedge postfix[i+1]\big) \vee \big((T[i] = S_2[i]) \wedge dp[i+1]\big)$
      - $postfix[i]$: $\forall S', T[i] = S_3[i]$
      - S_2: 한칸 뒤로 미룬 S'
      - S_3: 두칸 뒤로 미룬 S'

    - T[i]와 두번째 동전이 같고 뒤의 문자열이 모두 똑같거나
    - T[i]와 $S_2[i]$가 같고 뒤에서 두번째 동전을 포함한 뒤 문자열도 같다면
    - Postfix 또한 dp 테이블과 같이 채우게 되기 때문에 (뒤가 같은지 묻기 때문에 뒤에서 시작), dp[0][i]를 postfix로, dp[1][i]를 원래의 dp 테이블로 설정

Approach

- DP
  - dp 를 채웠다면 뒷 동전의 위치를 다 구한셈

  - 따라서 앞 동전의 위치만 결정 이는 비슷하게 prefix를 두어 조건 채택
    - $\exists i,\ (T[i] = C_1) \wedge prefix[i\ -1] \wedge dp[1][i+1]$
      - $prefix[i] = prefix[i\ -1] \wedge (S'[i] = T[i])$

      - 첫 동전 앞의 문자열이 같고 그 첫 동전 뒤에 동전을 포함하고,
        모두 같다면 S에서 T로 변환 가능한 것

Contents

- Problem
- Approach
- Code

Code

- 시작 부분
  1. 시작
  2. 빠른 입출력
  3. 입력
  4. S', S_2, S_3 만들기

```cpp
#include <iostream>

#define SIZE 10'001

using namespace std;

int main(void) {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;

    char strS[SIZE], strT[SIZE];
    cin >> strS >> strT;

    int c1Idx, c2Idx;
    cin >> c1Idx >> c2Idx;

    char coin1 = strS[c1Idx], coin2 = strS[c2Idx];

    char s1[SIZE] = { 0, }, s2[SIZE] = { 0, }, s3[SIZE] = { 0, };

    int s1Idx = 0, s2Idx = 1, s3Idx = 2;
    for (int i = 0; i < n; i++) {
        if (i != c1Idx && i != c2Idx)
            s1[s1Idx++] = strS[i], s2[s2Idx++] = strS[i], s3[s3Idx++] = strS[i];
    }
```

Code

- dp 테이블 채우기
  - $dp[i] = \big((T[i] = C_2) \wedge postfix[i+1]\big) \vee \big((T[i] = S_2[i]) \wedge dp[i+1]\big)$
  - dp[0][i]는 postfix

```
bool dp[2][SIZE];
dp[0][n - 1] = (s3[n - 1] == strT[n - 1]);
dp[1][n - 1] = (coin2 == strT[n - 1]);
for (int i = n - 2; i > 0; i--) {
    dp[0][i] = dp[0][i + 1] & (strT[i] == s3[i]);
    dp[1][i] = ((strT[i] == coin2) & dp[0][i + 1]) | ((strT[i] == s2[i]) & dp[1][i + 1]);
}
```

Code

- 답 구하기
  - $\exists i,\ (T[i] = C_1) \wedge prefix[i\ -1] \wedge dp[1][i+1]$
    - $prefix[i] = prefix[i\ -1] \wedge (S'[i] = T[i])$
  - prefix는 배열로하지 않고 변수로 설정
  - 가장 첫 문자가 coin인 경우
    - 마지막 ans와 같이 판단

```
bool ans = false;
bool prefix = (strT[0] == s1[0]);
for (int i = 1; i <= n - 1; i++) {
    if (coin1 == strT[i]) {
        if (prefix & dp[1][i + 1]) {
            ans = true;
            break;
        }
    }

    prefix = prefix & (s1[i] == strT[i]);
}

if (ans || ((strT[0] == coin1) & dp[1][1]))
    cout << "YES";
else
    cout << "NO";
cout << '\n';
```

P.S.

- Koosaga의 답

  L

  길이 $n-2$ 의 수열과 2 의 수열을 merge해서 (각각의 상대 순서를 유지해서) 원하는 수열을 만들 수 있는가를 찾는 문제이다. $dp(i,j)$ = (첫번째 수열의 앞 i개, 두번째 수열의 앞 j 개를 merge해서 $i+j$ 길이의 prefix와 match 시킬 수 있는가) 로 정의하면 $O(n)$ 에 문제가 해결된다. DP를 안 써도 풀릴 수도 있겠다.

  - 저렇게 문제를 다른 문제로 변환해서 푸는 연습이 필요할 것 같슴다
  - 근데 나 풀이 이해 못했음

감사합니다!