알고리즘_6~7일차 (2/5~2/6)

목차

문자열 (string)

---

1. 문자의 표현

1-1. 컴퓨터에서의 문자표현

> 메모리에 저장한다.
>
> 각 문자에 대응하는 숫자를 정하고 이것을 메모리에 저장하는 방법을 사용
>
> 영어가 대소문자 합쳐 52가지이므로 6비트(64가지)면 모두 표현할 수 있다.
> (이를 코드체계라고 한다.)

네트워크의 발전

> 네트워크 발전되기 전 각 지역 별로 코드체계를 정해 놓고 사용했음

> 네트워크(인터넷)가 발전하면서 서로 간에 정보를 주고 받을 때 정보를 달리 해석하는 문제가 생김

> 그래서 혼동을 피하기 위해 만든 표준안 (ASCII 코드)

1-2. ASCII 코드

> ASCII (America Standard Code for Information Interchange)

> 문자 인코딩 표준
> 7bit 인코딩으로 128문자 표현
> (33개의 출력 불가능한 제어문자 + 공백 포함 95개의 출력 가능한 문자)

확장 아스키

> 표준 문자 이외의 악센트 문자, 도형 문자, 특수 문자, 특수 기호 등 부가적인 문자를 128개 추가할 수 있게 하는 부호

> 1Byte 내의 8bit를 모두 사용함으로써 추가적인 문자 표현 가능

> 컴퓨터 생산자, SW 개발자가 여러가지 다양한 문자에 할당할 수 있음
> -> 이렇게 할당된 확장 부호는 표준 아스키와 같이 서로 다른 프로그램, 컴퓨터 사이에 교환되지 못한다.

> 따라서 표준 아스키는 하드웨어 및 소프트웨어 사이에서 세계적으로 통용,
> 확장 아스키는 프로그램, 컴퓨터, 프린터가 그것을 해독할 수 있도록 설계되어 있어야 한다.

1-3. 유니코드

> 다국어 처리를 위한 표준

> 오늘날 대부분 컴퓨터는 문자를 읽고 쓰는데 ASCII형식을 사용
>
> 각 국가들마다 자국의 문자 표현을 위한 코드체계를 만들어서 사용
>
> 국가 간 정보를 주고 받을 때 문제 발생
> (자국 코드체계를 타국이 가지고 있지 않으면 정보의 해석 오류 발생)
>
> 이를 방지하기 위해 마련한 표준이 유니코드

유니코드의 분류

> 유니코드도 Character Set으로 분류된다.
> (UCS-2, UCS-4)

> 유니코드를 저장하는 변수의 크기를 정의
>
> But, 바이트 순서에 대해서 표준화하지 못했음
>
> 파일 인식 시 해당 파일이 UCS-2, UCS-4 중 무엇인지 인식하고 구분해서 다르게 구현해야 하는 문제 발생
> -> 유니코드의 적당한 외부 인코딩이 필요하게 됨

---

2. 문자열

2-1. 문자열의 분류

대분류
문자열 (string)

중분류
2-1. variable length
2-2. fixed length

소분류
2-1-1. length controlled : java 언어에서의 문자열
2-1-2. delimited : c 언어에서의 문자열

2-2. 문자열 처리

C언어에서 문자열 처리

> 문자열은 문자들의 배열 형태로 구현된 응용 자료형
>
> 문자배열에 문자열을 저장할 때는 항상 마지막에 끝을 표시하는 널문자('\0')을 넣어줘야 한다.

Java(객체지향 언어)에서 문자열 처리

> 문자열 데이터를 저장, 처리해주는 클래스를 제공한다.
>
> string 클래스를 사용
>
> 문자열 처리에 필요한 연산을 연산자, 메소드 형태로 제공
> (보다 풍부한 연산을 제공)

Python에서 문자열 처리

> char 타입 없음
> 텍스트 데이터의 취급방법이 통일되어 있음

> 문자열 기호
> '(홀따옴표), "(쌍따옴표), '''(홀따옴표 3개), """(쌍따옴표 3개)

> +연결: 문자열 + 문자열 (이어 붙여주는 역할)
> *반복 : 문자열 * 수 (수만큼 문자열 반복)

> 문자열은 시퀀스 자료형으로 분류
> (인덱싱, 슬라이싱 연산 사용 가능)

> 문자열 클래스에 제공되는 메소드
> replace(), split(), isalpha(), find()

> 문자열은 튜플과 같이 요소를 변경할 수 없음

String 처리의 기본적인 차이점

> C언어 : 아스키 코드(ASCII)로 저장
> jAVA : 유니코드(UTF16, 2byte)로 저장
> Python : 유니코드(UTF8)로 저장

2-3. 문자열 뒤집기

> 자기 문자열에서 뒤집는 방법

> 새로운 빈 문자열을 만들어 소스의 뒤부터 읽어 타겟에 쓰는 방법
> -> swap을 위한 임시 변수가 필요
> -> 반복수행을 문자열 길이의 반만을 수행해야 한다.

2-4. 문자열 비교

> Java에서는 equals( ) 메소드를 제공

> Python에서는 ==연산자와 is연산자를 제공

2-5. 문자열 숫자를 정수로 변환하기

> Python에서는 숫자와 문자변환 함수를 제공한다.

> java에서는 숫자 클래스의 parse 메소드를 제공한다.

---

3. 패턴매칭

패턴매칭에 사용되는 알고리즘

> 고지식한 패턴 검색 알고리즘
> KMP 알고리즘
> 보이어-무어 알고리즘
> 카프-라빈 알고리즘

3-1. 고지식한 알고리즘

> 본문 문자열을 처음부터 끝까지 차례대로 순회하면서 패턴 내의 문자들을 일일히 비교하는 방식으로 동작

> 매우 비효율적인 방법
> (그러나, 반드시 구현할 수 있어야 한다.)

시간 복잡도 : O(M x N)

> 최악의 경우 텍스트의 모든 위치에서 패턴을 비교해야 되기 때문이다.

교재 코드

```
p = "is" # 찾을 패턴
t = "This is a book~!" # 전체 테스트
M = len(p) # 찾을 패턴의 길이
N = len(t) # 전체 텍스트의 길이

def BruteForce(p, t):
    i = 0   # t의 인덱스
    j = 0   # p의 인덱스
    while j < M and i < N:
        if t[i] != p[j]:
            i = i - j
            j = -1
        i = i + 1
        j = j + 1
    if j == M:   # 검색성공
        return i - M
```

```
    else:         # 검색실패
        return -1
```

강의 실습 코드

```
## 고지식한 알고리즘 문제 강의 실습
## 인풋, 아웃풋은 2/5 SWEA 문제 참고
def f(pat, txt, M, N):
    # txt에서 비교 시작 위치 i
    for i in range(N-M+1):
        for j in range(M):
            # 불일치하면 다음 시작위치로 가
            if txt[i+j] != pat[j]:
                break
        # 패턴 매칭에 성공해서
        # for문이 잘 끝난 경우에는 1을 반환해
        else:
            return 1
    # 모든 위치에서 비교가 끝난 경우
    return 0

T = int(input())
for tc in range(1, T+1):
    pat = input()
    txt = input()
    M = len(pat)
    N = len(txt)

    # txt의 시작위치를 i라고 하면
    # 마지막 비교위치는 N-3까지만 보면 된다.

    ans = f(pat, txt, M, N)
    print(f'#{tc} {ans}')
```

3-2. KMP 알고리즘

> 불일치가 발생한 텍스트 스트링의 앞 부분에 어떤 문자가 있는지를 미리 알고 있으므로,
> 불일치가 발생한 앞 부분에 대하여 다시 비교하지 않고 매칭을 수행

> 패턴을 전처리하여 배열 next[M]을 구해서 잘못된 시작을 최소화
> **next[M] : 불일치가 발생했을 경우 이동할 다음 위치**

> 시간 복잡도 : O(M+N)

> 반복패턴(유전자 데이터), 문장 비교 등 KMP 알고리즘이 가능한 패턴에서만 사용 가능

3-3. 보이어-무어 알고리즘

> 오른쪽에서 왼쪽으로 비교
> 대부분 상용 소프트웨어에서 채택하고 있음

> 패턴 오른쪽 끝에 있는 문자가 불일치하고 이 문자가 패턴 내에 존재하지 않는 경우:
> 이동 거리는 패턴의 길이 만큼이 된다.

> 오른쪽 끝 문자가 불일치하지만, 이 문자가 패턴 내에 있는 경우:
> 패턴에서 일치하는 문자를 찾아서 해당 길이만큼 이동한다.

> 텍스트 문자를 다 보지 않아도 된다.
> 시간 복잡도 : O(mn)

※ 교재 p.47 예시 보기

문자열 매칭 알고리즘 비교

> 찾고자 하는 문자열 패턴의 길이 m, 총 문자열 길이 n

> 고지식한 패턴 알고리즘
> O(mn)

> 카프-라빈 알고리즘
> θ(n)

> KMP 알고리즘
> θ(n)

> 보이어-무어 알고리즘
> 최선 : q?(n), 최악 : θ(mn)
> 입력에 따라 다르지만 일반적으로 θ(n)보다 시간이 덜 든다.

---

4. 문자열 암호화

4-1. 시저 암호화

> 1만큼 평행했다.
> (공백=A, A=B, B=C, B=C...) 1만큼 평행했을 때 1을 키값이라 한다.

4-2. 단일 치환 암호화

> 문자 변환표를 이용한 암호화
> 시저 암호화보다 훨씬 강력한 암호화 기법

> 복호화 하기 위해서는 모든 키의 조합(key space)가 필요하다.
> 단일 치환 암호의 키의 총수는 26!

4-3. bit열의 암호화

> 배타적 논리합(exclusive-or) 연산 사용

> key가 1이면 반전, 0이면 그대로

---

5. 문자열 압축

> Run-length encoding 알고리즘

> 같은 값이 몇 번 반복되는가를 나타냄으로써 압축
> 예시: ABBBBBBBBA를 압축하면 A1B8A1

> 이미지 파일포맷 중 BMP 파일포맷의 압축방법

> 좀 더 효율적이고 일반적인 압축방법 : 허프만 코딩 알고리즘

---

5. 과목평가 리뷰

델타 문제: IM평가 단골문제

행+열의 합을 통으로 구하는 문제도 나온다. (풍선팡 같은 문제인데 행+열을 전부인)
(조만간 연습문제 풀어보기로 함)

과목평가 1번 문제 : 가장자리 빼고 사격문제

1. 조건을 추가 : 주변의 개수가 4개면 등
2. 아예 범위를 가장자리 제외하고 잡기

!!!!!!!중요!!!!!!!!!!
1차원 배열, 2차원 배열 중 델타 문제
파리퇴치, 어디에 단어가 문제는 복습 필수