19장

진급 프로젝트: 스크래블:아트 단어 게임

19장 진급 프로젝트: 스크래블:아트 단어 게임

19.1 문제 이해하기

19 프로젝트 문제

주어진 타일로 만들 수 있는 단어들을 알려주는 프로그램을 작성하라.

- 타일 수는 매번 달라질 수 있다. 즉, 정해진 타일 개수는 없다.
- 타일에는 점수가 표시되어 있지 않다. 즉, 모든 타일은 점수가 같다.
- 타일은 문자열로 주어진다(예: tiles = "hijklmnop").
- 프로그램은 타일로 만들어낼 수 있는 단어를 ('ink', 'oil', 'kiln')처럼 튜플 형태로 보여 줘야 한다.

19.1 문제 이해하기

- 입력받은 단어 목록을 여러분이 다룰 수 있는 형태로 만들어야 한다. 긴 문자열에 들어 있는 여러 단어를 각 단어들이 포함된 튜플로 만들어라.

- 단어 목록에 있는 단어들을 주어진 타일로 만들 수 있는지 판단하라.

19.1.1 올바른 단어 목록의 표현 방식 바꾸기

- 단어들을 나중에 사용하기 위해 튜플로 만들기 : 큰 문자열에 단어를 모두 가지고 있으면 다루기가 힘들기 때문이다.

- 올바른 단어들이 하나의 문자열로 주어진다. 이 문자열을 사람이 볼 때는 한 '줄'에 한 단어가 있지만, 컴퓨터가 볼 때는 그냥 문자들이 들어 있는 긴 시퀀스에 불과하다

19.1.1.1 문제를 그림으로 표현하기

- 사람이 볼 때는 문자열이 체계적으로 정리된 것처럼 보이며, 쉽게 단어를 하나하나 구분할 수 있다. 하지만 컴퓨터는 단어들이 문자열에 들어 있다는 개념을 모른다. 컴퓨터는 주어진 문자열을 """art\nhue\nink\noil\n...\ncrosshatching"""와 같이 인식한다.

- 사람이 눈으로 볼 때는 줄이 바뀐 것이지만, 컴퓨터가 볼 때는 그냥 새줄 문자라는 한 문자에 지나지 않는다. 새줄 문자는 \n으로 표현한다. 따라서 전체 문자열에서 모든 새줄 문자를 찾아야 단어들을 서로 구분할 수 있다.

19.1.1.1 문제를 그림으로 표현하기

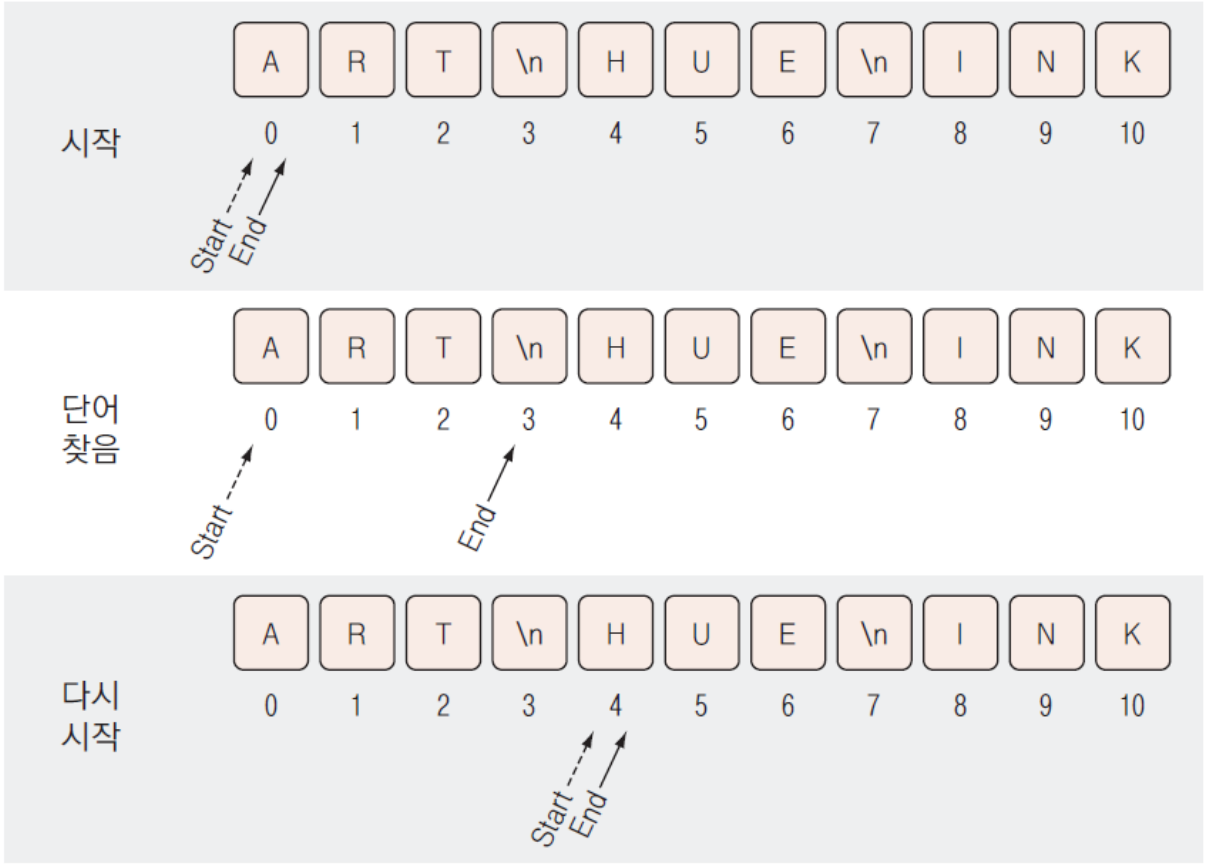

그림 19-1
문자열을 단어로 나누는 방법. 맨 위의 '시작'을 보면 start와 end 인덱스 변수를 사용한다. 가운데의 '단어 찾음'을 보면 새줄 문자를 만나면 end 인덱스 증가를 멈춘다. 마지막 '다시 시작'을 보면 start와 end를 새줄 문자 다음으로 변경해서 단어 검색을 계속한다

19.1.1.2 몇 가지 예 생각해보기

프로그램을 작성할 때는 프로그램이 잘 작동하는지 알아볼 수 있게 몇 가지 테스트 케이스를 만들어야 한다. 간단한 경우와 복잡한 경우를 몇 가지 생각해보라.

- 예를 들어 words = """art"""처럼 사용 가능한 단어 목록에 단어가 1개만 들어 있는 경우도 있고, 문제 설명에 주어진 예제처럼 단어가 여러 개 들어 있을 수도 있다.

19.1.1.3 해법을 의사 코드로 추상화하기

문자열의 글자를 모두 살펴봐야 하므로 루프를 사용하자. 루프 안에서는 새줄 문자를 찾았는지 판단해야 한다.

- 새줄 문자를 찾으면 현재까지 찾아낸 단어를 저장하고, start와 end 인덱스를 새줄 문자 뒤로 재설정해야 한다.
- 새줄 문자를 찾지 못했다면 end 인덱스만 1씩 증가시키면서 새줄 문자를 찾는다.

```
word_string = """art
hue
ink
"""
start와 end를 0으로 설정한다
올바른 단어를 담기 위한 빈 튜플을 만든다
for letter in word_string:
    if letter가 새줄 문자:
        start부터 end까지의 단어를 튜플에 넣는다
        start와 end를 새줄 문자 바로 뒤 위치로 다시 설정한다
    else:
        end를 1 증가시킨다
```

19.1.2 주어진 타일로 올바른 단어 만들기

» 이제 올바른 단어들이 들어 있는 목록(튜플)이 있을 때, 주어진 타일로 올바른 단어를 만들 수 있는지 판단하는 로직을 생각할 차례다.

19.1.2.1 문제를 그림을 표현하기

두 번째 문제를 해결하기 위한 로직은 두 가지 방식으로 접근할 수 있다.

- 가지고 있는 타일을 살펴보는 것부터 시작한다. 타일의 조합(combination)을 모두 찾은 후,조합이 올바른 단어 목록에 있는지 검사한다.
- 올바른 단어 목록을 뒤지면서 각 단어가 현재 가지고 있는 타일로 생성할 수 있는 단어인지 검사한다.

19.1.2.1 문제를 그림을 표현하기

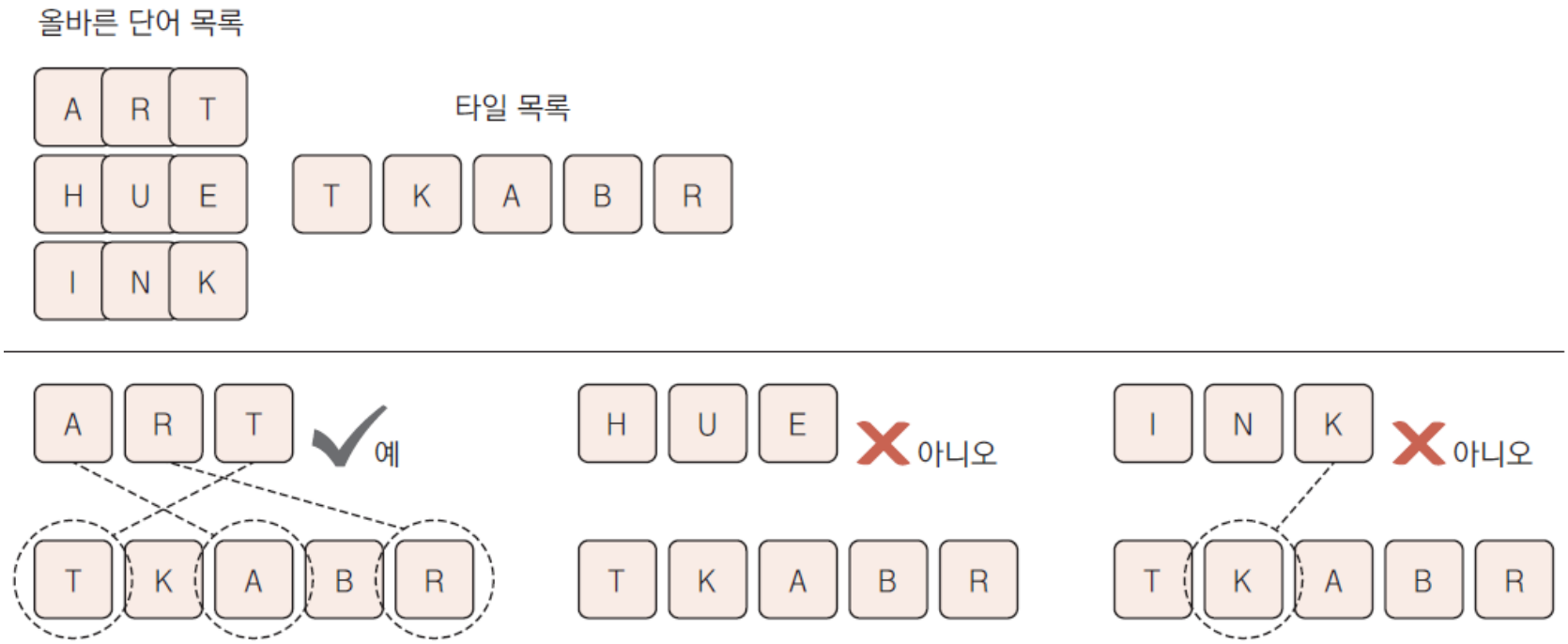

그림 19-2
올바른 단어 목록과 타일 목록이 주어졌을 때 올바른 단어를 가져와서 그 단어에 있는 문자가 타일 목록에 다 들어 있는지 검사할 수 있다. 문자가 다 들어 있다면 그 단어를 만들 수 있는 단어 목록에 추가한다. 단어에 있는 문자 중 하나라도 타일 목록에 없다면 주어진 타일로는 그 단어를 만들 수 없다

19.1.2.2 몇 가지 예 생각해보기

코드가 잘 처리해야 하는 특별한 경우

- 타일이 하나만 있는 경우: 올바른 단어를 하나도 만들 수 없다.
- 주어진 타일로 올바른 단어를 딱 한 개만 만들 수 있는 경우: tiles = "art"라면 art라는 단어만 만들 수 있다.
- 주어진 타일로 올바른 단어를 두 개만 만들 수 있는 경우: tiles = "euhtar"라면 art와 hue라는 두 단어만 만들 수 있다.
- 주어진 타일로 올바른 단어를 하나 만들고 타일이 몇 개 남는 경우: tiles = "tkabr"라면 art를 만들 수 있고 k와 b가 남는다.
- 올바른 단어에 있는 문자 타일을 다 가지고 있지만, 단어에서는 같은 문자를 2번 이상 사용하는데 타일은 한 개밖에 없는 경우: tiles = "colr"라면 o가 1개뿐이므로 color라는 단어를 만들 수 없다.

19.1.2.3 해법을 의사 코드로 추상화하기

- 올바른 단어를 하나씩 가져와서 타일을 가지고 그 단어를 만들 수 있는지 모두 살펴봐야 한다. 따라서 루프가 필요하다.

- 이 루프는 올바른 단어에 대한 첫 번째 루프에 내포된 루프여야 한다. 내포된 루프에서 살펴보다가 문자가 타일에 없으면 즉시 루프를 끝내고, 타일이 있으면 루프를 계속 진행한다.

19.1.2.3 해법을 의사 코드로 추상화하기

```
for word in valid_words:
    for letter in word:
        if letter가 tiles에 있지 않음:
            나머지 글자를 살펴보는 일을 중단하고 다음 단어를 처리한다
        else:
            letter를 tiles에서 제거한다(글자 중복 처리를 위해)
    if tiles에서 word에 들어 있는 모든 글자를 찾음:
        word를 found_words에 추가한다
```

19.2 코드를 여러 부분으로 나누기

19.2 코드를 여러 부분으로 나누기

보통 첫 번째 덩어리는 주어진 입력을 확인하고, 프로그램에서 쓸 수 있는 정보를 담기 위해 준비하는 부분이다.(코드 19-1)

- 예술과 관련된 올바른 단어 목록(문자열)을 저장하고, 시작할 타일들(문자열)을 저장한다.
- 올바른 단어를 찾기 위해 start와 end 인덱스를 초기화한다.
- 찾아낸 올바른 단어를 넣기 위해 빈 튜플을 만들어 저장한다.
- 타일로 만들 수 있는 단어를 찾아내 넣기 위해 빈 튜플을 만들어 저장한다.

19.2 코드를 여러 부분으로 나누기


```
words = """art ---- 올바른 단어들을 큰 문자열로 저장한다
hue
ink
...
crosshatching
"""


tiles = "hijklmnop"

all_valid_words = () ---- 올바른 단어들을 저장하기 위한 빈 튜플
start = 0 ---- 검색 중인 단어의 시작 위치 인덱스를 초기화한다
end = 0 ---- 검색중인 단어의 끝 위치 인덱스를 초기화한다
found_words = () ---- 타일로 만들 수 있는 단어들을 저장하기 위한 빈 튜플
```

코드 19-1
스크래블:아트. 초기화 부분

19.2 코드를 여러 부분으로 나누기

이 프로그램에서 두 번째 코드 덩어리는 올바른 단어들이 들어 있는 큰 문자열에서 각 단어를 포함하는 튜플을 만들어내는 부분이다.(코드 19-2)

- 여기서는 올바른 언어가 들어가는 튜플에 어떻게 새 단어를 추가할지 고민해봐야 한다.
- 단어를 한 번에 하나씩 찾아서 튜플에 추가해야 하므로 두 튜플을 연결하는 연결 연산(+)을 사용하되, 방금 찾은 단어 하나만 들어 있는 튜플을 지금까지 찾은 단어가 모두 들어 있는 튜플 뒤에 붙이면 된다.

19.2 코드를 여러 부분으로 나누기

```
for char in words: ---- 모든 문자를 가져오면서 루프를 돈다
    if char == "\n": ---- 문자가 새줄 문자인지 검사한다
        all_valid_words = all_valid_words + (words[start:end],) ---- 올바른 단어가 들어 있는 튜플 뒤에 단어가 하나 들어 있는 튜플을 덧붙인다
        start = end + 1 ----┐
        end = end +1 ----┘ 다음 단어의 시작 부분으로 start와 end를 옮긴다
    else:
        end = end + 1 ---- end가 다음 글자를 가리키게 만든다
```

코드 19-2
스크래블:아트. 큰 문자열에서 올바른 단어 목록을 만드는 부분

- 단어를 큰 문자열에서 읽어야 하므로 문자열의 각 문자를 가져오면서 루프를 돈다.
- 가져온 문자가 새줄 문자라면 한 단어의 끝에 도달한 것이다. 따라서 start와 end를 사용한 슬라이스를 통해 단어를 저장한다.
- 그 후 start와 end를 모두 새줄 문자 다음 위치로 다시 설정한다. 바로 다음 단어가 시작되는 위치다.
- 큰 문자열에서 루프 변수에 가져온 단어가 새줄 문자가 아니라면 아직 단어가 끝나지 않았으므로 end 인덱스를 증가시킨다.

19.2 코드를 여러 부분으로 나누기

» 세 번째이자 마지막 코드 덩어리는 타일로 올바른 단어들을 만들 수 있는지 검사하는 부분이다.(코드 19-3)

» 두 번째 덩어리와 마찬가지로 세 번째 덩어리도 이전에 작성했던 의사 코드를 코드로 옮기면서 부족한 부분을 채워 넣으면 된다.

» (1) 같은 문자가 여러 번 나오는 경우를 어떻게 처리할 것인가?

- 올바른 단어에 포함된 타일을 없애도록 코드를 작성한다. 올바른 단어를 튜플에서 하나 가져올 때마다 tiles_left라는 변수를 사용한다.
- 처음 tiles_left에는 가지고 있는 모든 타일이 들어 있다. 하지만 단어의 각 문자를 처리할 때마다 tiles_left를 사용해 남은 타일을 추적한다. 이렇게 하려면 단어에서 문자를 처리할 때 그 문자에 해당하는 타일이 있으면 tiles_left 가 그 문자를 제외한 나머지 문자만 포함하도록 갱신해야 한다.

19.2 코드를 여러 부분으로 나누기

(2) 타일에서 단어에 들어 있는 글자를 모두 찾은 경우를 어떻게 알 수 있을까?

- 단어의 각 문자를 가져와서 그 문자에 해당하는 타일을 찾으면 tiles_left에서 그 문자를 하나 제거한다.
- 따라서 올바른 단어의 길이와 tiles_left에서 제거한 타일의 개수가 일치하면 타일로 그 단어를 만들 수 있다는 걸 알 수 있다(문자에 해당하는 타일을 찾지 못하면 그 문자를 tiles_left에서 제거할 수 없으므로 단어 길이가 제거한 타일 개수보다 더 커진다).
- 제거한 타일 개수는 처음 가지고 있던 타일 개수에서 단어에 대한 루프를 끝낸 후 tiles_left에 남은 타일의 개수를 뺀 값이다.

19.2 코드를 여러 부분으로 나누기


```
for word in all_valid_words: ---- 올바른 단어들을 살펴본다
    tiles_left = tiles ---- tiles_left는 타일 중복을 처리한다
    for letter in word: ---- 올바른 단어의 모든 문자를 처리한다
        if letter not in tiles_left: ---- tiles_left 안에 문자가 없으면 루프를 중단한다
            break
        else:
            index = tiles_left.find(letter) ---- tiles_left에서 문자 위치를 찾는다
            tiles_left = tiles_left[:index]+tiles_left[index+1:] ---- 찾은 문자를 없앤 새 tiles_left를 만든다
    if len(word) == len(tiles)-len(tiles_left): ---- 타일로 단어를 만들 수 있는지 검사한다
        found_words = found_words + (word,) ---- found_words에 단어를 추가한다

print(found_words)
```

코드 19-3
스크래블:아트. 주어진 타일로 만들 수 있는 올바른 단어를 모두 찾는 부분

19.2 코드를 여러 부분으로 나누기

코드 안에는 내포된 루프가 들어 있다.

- 바깥쪽 루프는 올바른 단어를 처리하고, 안쪽 루프는 올바른 단어에 들어 있는 문자를 처리한다.
- 타일에 없는 문자를 발견하면 내부 루프를 끝내고 다음 단어로 넘어간다.
- 타일에 문자가 있으면 단어에 있는 다음 문자를 처리한다.
- tiles_left는 처리 중인 문자에 해당하는 타일을 제외한 남은 타일들을 저장한다.
- 어떤 문자를 타일에서 찾을 때마다 그 위치를 가지고 문자열 슬라이스를 통해 그 문자를 제외한 나머지문자들로 이뤄진 문자열을 만들어 저장한다.
- 마지막 단계는 올바른 단어의 모든 문자를 타일에서 찾아냈는지 검사하고, 모든 문자를 찾은 경우 찾은 단어 목록에 해당 단어를 추가한다.

19.3 요약

19.2 코드를 여러 부분으로 나누기

- 요청 받은 내용에서 중요한 부분을 그림으로 그려서 이해하라.
- 요청 받은 내용에 대한 단순한 테스트 케이스를 몇 가지 만들어서 이해하라.
- 문제의 각 부분을 일반화하고, 각 부분을 해결할 수 있는 공식이나 로직을 만들어라.
- 의사 코드는 유용하다. 특히 조건문이나 루프 등의 요소가 들어가는 알고리즘 로직을 작성할 경우 더 유용하다.
- 코드를 여러 덩어리로 나눠서 생각하라. 전체 프로그램을 자연스럽게 여러 부분으로 나눌 수 있는지 항상 질문하라. 예를 들어 변수 초기화, 알고리즘 구현, 정리 코드 등으로 코드를 구성할 수 있는지 생각해 보라.