Homework 4

SNU L0444.200, 2017 Fall

Due: 12/6 (Wed), 23:59

TA: Jinyung Kim, Prof: Kwangkeun Yi

이번 숙제에서는,

- 최종적으로 만들 '누구나 모험가'의 부품을 준비한다.
- 프로그램의 실행비용을 생각하며 효율적인 프로그램을 작성하는 연습을 해 본다.
- 모든 경우를 빠짐없이 고려하고, 경계 조건에 유의하여 항상 올바르게 동작하는 프로그램을 작성하는 연습을 해 본다.

위치정보의 표현 이번 숙제에서는 숙제 1와 숙제 2에서 다뤄본 위치정보를 혼합하여, 시간과 이름과 키워드들이 모두 기록된 위치정보를 표현한다. 다음과 같은 다섯 개의 키를 가지는 파이썬의 딕셔너리 (Dictionary) 자료형을 이용한다.

- `latitude`: 위도를 나타내는 -90 이상 90 이하의 실수값 (`float`)
- `longitude`: 경도를 나타내는 -180 이상 180 이하의 실수값 (`float`)
- `timestamp`: 시각을 나타내는 0 이상의 정수값 (Unix time: 1970년 1월 1일 00:00:00 UTC로부터 몇 초가 지났는지) (`int`)
- `name`: 이 장소의 이름을 나타내는 문자열 (`string`)
- `keywords`: 이 장소를 표현하는 키워드들을 나타내는 문자열들의 리스트 (`list[string]`)

Exercise 1 “올바른 꼴의 위치정보인가”

위처럼 표현된 위치정보 하나가 인자로 주어진다. 올바른 꼴의 위치정보인지 판단하여, 올바르면 `True`를, 그렇지 않으면 `False`를 돌려주는 함수 `is_valid1(loc)`를 작성해 본다. `loc`는 위에서 표현된 위치정보 딕셔너리이다. 함수가 리턴하는 값은 `bool` 타입이다.

위치정보가 올바른 꼴이라는 것은 다음과 같이 정의한다 (11/29 수정): 첫째, 다섯 개의 키가 모두 있어야 한다.둘째, 주어진 다섯 개의 키를 제외하고 다른 키가 있어서는 안 된다. 셋째, 다섯 개의 키에 저장된 값들은 주어진 타입의 값이어야 한다.

Exercise 2 “값까지 올바른 위치정보인가”

위처럼 표현된 위치정보 하나가 인자로 주어진다. 이번에는 값까지 올바른 위치정보인지 판단하여, 올바르면 `True`를, 그렇지 않으면 `False`를 돌려주는 함수 `is_valid2(loc)`를 작성해 본다. `loc`는 위에서 표현된 위치정보 딕셔너리이다. 함수가 리턴하는 값은 `bool` 타입이다.

위치정보가 값까지 올바르다는 것은 다음과 같이 정의한다: 첫째, 올바른 꼴의 위치정보여야 한다. (Exercise 1 참고) 둘째, 위도와 경도의 범위가 주어진 범위 안에 있어야 한다. 셋째, 시각은 0 이상의 정수여야 한다.

Exercise 3 “값까지 올바른 위치정보들만 남겨 정렬하기”

위처럼 표현된 위치정보들의 리스트가 인자로 주어진다. 이 중 값까지 올바른 위치정보들만 남긴 뒤, 이를 시간순으로 정렬한 리스트를 돌려주는 함수 `sanitize_and_sort(loc_lst)`를 작성해 본다. `loc_lst`는 위에서 표현된 위치정보 딕셔너리들의 리스트이다. 함수가 리턴하는 리스트 역시 위치정보 딕셔너리들의 리스트이다.

리스트에 있는 모든 위치는 시각이 다르다고 가정한다. 즉, 같은 시각에 여러 곳에 동시에 있었던 경우는 없다.

여러 가지 정렬 알고리즘을 강의와 실습 시간에 배웠다. 원소의 갯수의 제곱에 비례하는 시간에 걸리는 정렬 알고리즘보다는 빠른 것을 사용하여 정렬해 보도록 한다. 또는, 기본으로 제공하는 정렬 라이브러리를 사용해도 된다.

Exercise 4 “얼마나 비슷한 곳인가?”

위처럼 표현된 위치정보 두 개가 인자로 주어진다. 두 위치 간 키워드를 기반으로 한 유사도를 계산하고자 한다.

먼저, 문자열들의 집합 A와 문자열들의 집합 B사이의 유사도는 두 집합의 교집합 크기를 두 집합의 합집합 크기로 나눈 값으로 정의할 수 있다. 예를 들어 집합 A = {피자, 파스타, 와인}, 집합 B = {

파스타, 커피, 피자}라고 할 때, 교집합 A ∩ B = {피자,파스타}, 합집합 A ∪ B = {피자, 파스타, 와인, 커피}가 되므로, 집합 A, B 사이의 유사도는 2/4 = 0.5가 된다. 집합 A와 집합 B가 모두 공집합일 경우에는 나눗셈이 정의되지 않으니 따로 1.0으로 정의한다.

다만, 우리가 다루는 자료형의 경우는 키워드에 중복이 있을 수 있다. 여러 사람이 이 위치에 대해 남긴 기록들을 바탕으로 키워드를 산정한 것이기 때문이다. 따라서 위의 유사도 계산을 단순히 집합 뿐 아니라 중복을 허용하는 경우에 대해서도 다음과 같이 확장한다. A는 키워드 "피자"를 3개 가지고 있고, B는 키워드 "피자"을 5개 가지고 있다고 하자. 이 경우 A ∩ B는 키워드 "피자"를 3과 5 중 작은 값인 3개, A ∪ B는 키워드 "피자"를 3과 5 중 큰 값인 5개 가지게 되는 것으로 정의하면 해결된다. 예를 들어 A = {피자, 피자, 파스타, 파스타, 와인}, B = {피자, 파스타, 파스타, 스테이크, 맥주}라고 하면, A ∩ B = {피자, 파스타, 파스타}, A ∪ B = {피자, 피자, 파스타, 파스타, 와인, 스테이크, 맥주}가 되므로, 유사도는 3/7, 약 0.4285...가 된다.

위와 같이 두 위치 간 유사도를 계산하는 함수 `similarity(loc1, loc2)`를 작성해 본다. `loc1`과 `loc2`는 위에서 표현된 위치정보 딕셔너리이다. 함수가 리턴하는 값은 `float` 타입이다.

만일 둘 중 하나라도 값까지 올바른 위치정보가 아니라면, -1.0을 돌려주도록 한다.

Exercise 5 "마지막으로 어디에 있었을까?"

위처럼 표현된 위치정보들의 리스트가 인자로 주어진다. 단, 이 리스트는 시간순으로 정렬된 리스트이다. 누군가의 시간별 위치에 대한 정보이다.

주어진 리스트를 바탕으로 이 사람이 특정한 시간에 어느 곳에 있었는지를 알아내고 싶다. 다만, 정확히 해당되는 특정 시간에 해당되는 위치가 포함되지 않을 수도 있으므로, 해당 시각 값보다 같거나 작은, 그러면서도 가장 큰 시각 값을 갖는 위치를 구해야 한다. 해당 위치의 이름을 돌려주면 된다.

`where_you_at(loc_list, t)`를 작성해 본다. `loc_list`는 위에서 표현된 위치정보 딕셔너리들의 리스트이며, 정렬되어 있으며, 리스트에 있는 모든 위치는 값까지 올바르고, 시각은 모두 서로 다르다. `t`는 우리가 찾고자 하는 시각을 표현하는 타임스탬프 값으로 0 이상의 `int` 타입의 값이다. 함수가 리턴하는 값은 `string` 타입으로, 조건을 만족하는 위치의 이름이다.

예를 들어 100초에 서울대입구역에, 200초에 낙성대역에, 300초에 사당역에 있었다고 하자. 찾고자 하는 시각이 200으로 주어지면 답은 낙성대역이다. 시각이 199로 주어지면 답은 서울대입구역이다.

주어진 리스트의 모든 원소를 탐색하며 조건에 맞는 답을 구해내는 것보다 효율적인 방법을 우리는 강의와 실습 시간에 배웠다. 이를 활용하여 해결하도록 한다.