Practice Problems 4

Problem 1 - Marathon runners

Create a class `MarathonScore`, with three fields: `name`, `city`, and `time`. `name` and `city` are Strings, `time` is a float. Write an appropriate `__init__` method to initialize a new object of this class.

Write a function `read_marathon_data(filename)` that takes as an argument a filename, and reads the records from the `marathon_data.csv` file attached on Piazza; it should return the list of objects of type `MarathonScore` - one for each row of `marathon_data.csv`.

File format
The `marathon_data.csv` contains a table with three columns, like below:

Time	Name	Event/Place
2:55:18.4	Johnny Hayes	London Olympics England
2:52:45.4	Robert Fowler	Yonkers United States
2:46:52.8	James Clark	New York City United States
2:46:04.6	Albert Raines	New York City United States
2:42:31.0	Henry Barrett	Polytechnic Marathon London England

The `.csv` file is just a text file, containing each row in a seperate line, columns separated by commas:

```
Time,Name,Event/Place
2:55:18.4,Johnny Hayes,London Olympics England
2:52:45.4,Robert Fowler,Yonkers United States
2:46:52.8,James Clark,New York City United States
2:46:04.6,Albert Raines,New York City United States
2:42:31.0,Henry Barrett,Polytechnic Marathon London England
```

Reading files
As a reminder, to read and parse the `.csv` file you can use the `csv` module.

To import the `csv` module, you can use

```
import csv
```

To open file use syntax:

```
with open(filename, "r") as file:
    ...
```

You can read the content of the file, using `csv.reader` as follows:

```
    data = [ record for record in csv.reader(file) ]
```

Problem 2 - incremental graph

Write a function `when_connected(n, edges)` that takes as its argument a size of a graph `n` and a description of a graph in which edges are appearing one by one. The list `edges` contains pairs (a_i, b_i) indicating that at time i appears an edge between vertices (a_i, b_i).

The function should return when the graph became connected.

That is, the returned value T should be the smallest integer, such that the graph on `n` vertices given by the first T edges from the `edges` is connected.

Note
Full points for an algorithm running in time $O(m + n \log n)$; partial points will be given for slower algorithm.

Hint
You should follow the same blueprint as the implementation of the Kruskal algorithm from lecture 13. Here is a reminder:

1. Create a list `color`, storing a color for each vertex (denoting what connected component the vertex is in). Initially each vertex has its own color.
2. Loop over the edges in the list `edges`.
    1. If the endpoints of two vertices already have the same color, skip this edge.
    2. If two vertices have different color, recolor all vertices of one of those colors, into the other.
    3. If there is only one color class remaining, return the current index.
    To get full points (an algorithm running in time $O(m + n \log n)$, whenever you see an edge which endpoints have two different color, of the two color classes recolor the one that has fewer vertices. To do this, you will need one more list, which stores, for each color class, a list of vertices of that color.

Problem 3 - Dictionaries

Write a function `best_score(submissions)` that takes as an argument a list of pairs $(name, score)$, where `name` is a string, and `score` is a positive integer, for example:

```
submissions = [
    ("Alice", 82),
    ("Bob", 91),
    ("Alice", 77),
    ("Diana", 91),
    ("Charlie", 68),
    ("Bob", 85),
    ("Charlie", 74)
]
```

It should return a dictionary, containing, for each of the names, the maximum score received.
Example

```
submissions = [
    ("Alice", 82),
    ("Bob", 91),
    ("Alice", 77),
    ("Diana", 91),
    ("Charlie", 68),
    ("Bob", 85),
    ("Charlie", 74)
]
best_score(submissions) == {'Alice': 82, 'Bob': 91, 'Diana': 91, 'Charlie':
74}
```