Happy Returns Coding Test

1. Card Sort

Task

You are given a stack of boarding cards for various transportations that will take you from a point A to point B via several stops on the way. All of the boarding cards are out of order and you don't know where your journey starts, nor where it ends. Each boarding card contains information about seat assignment, origin/destination, and means of transportation (such as flight number, bus number etc). Provide a library that lets you sort this kind of list and present back a description of how to complete your journey. For instance the library should be able to take an unordered set of boarding cards, provided in a format defined by you, and produce this list:

- Take train 78A from Madrid to Barcelona. Sit in seat 45B.
- Take the airport bus from Barcelona to Gerona Airport. No seat assignment.
- From Gerona Airport, take flight SK455 to Stockholm. Gate 45B, seat 3A. Baggage drop at ticket counter 344.
- From Stockholm, take flight SK22 to New York JFK. Gate 22, seat 7B. Baggage will we automatically transferred from your last leg.
- You have arrived at your final destination.

The list should be defined in a format that's compatible with the input format. Document the input and output your library accepts / returns. You can assume that no city will be repeated.

Requirements

- Use any language of your choice.
- Do not use any 3rd party framework (meaning that don't search for a card sorting API and send us that, but feel free to use additional libraries not included with the core distribution of your chosen language). Start all code from scratch.
- The structure of the code should be extendable to make building in support for any means of transportation / extra information required about a specific type of transportation easy.
- The implementation of your sorting algorithm should work with any set of boarding passes, as long as there is always an unbroken chain between all the legs of the trip. i.e. it's one continuous trip with no interruptions.

- The algorithm doesn't need to consider that departure / arrival are in the correct order. In fact there is no information about any such times on the boarding passes. It is just assumed that your next connection is waiting for you when you arrive at a destination.
- The algorithm should have the lowest possible order of complexity (Big O notation) you could think of.
- Although we believe that working as a team is the best way to tackle problems, this task is meant to be an assessment of your personal skills, so please refrain from asking friends / cooperating with other people while solving this task.

What we look at

We are also interested in how you structure your code so that it's easily extendable, and easy to modify / understand by others. We are also interested in seeing how efficient the sorting algorithm you implement is.

Other Questions

2.a Remove Dupes

Create a function that takes an array as its argument and returns an array with all duplicate values removed.

- Order should be preserved.
- The resulting array should preserve the first instance of an element and remove all subsequent instances.
- You can assume that the instances in the array can be compared using the built in language conditionals.

```
>>> remove_dupes([1,2,1,3,2,1,7])
[1, 2, 3, 7]
```

2.b Remove Second Dupe

Modify the function so the ordering of the returned array is based on the second instance. If an element does not exist twice, the first instance of it is used for ordering.

```
>>> remove_dupes([1,2,1,3,2,1,7])
[1, 3, 2, 7]
```

2.c Remove Nth Dupe

Modify the function function so it takes a second argument N. The function should now order the returning

array based on the Nth instance of the element. If an element does not exist N times, the last instance of it is used for ordering.

```
>>> remove_dupes([1,2,1,3,2,1,7], 3)
[3, 2, 1, 7]
```

3. Flood Fill

Create a function that simulates a flood fill like it exists in any painting program.

Inputs

- A matrix where each location represents a pixel that has a certain color.
- A location on the matrix to start the fill.
- A color to fill.

The function should return a matrix where the given pixel is changed to the new color and all adjacent pixels of the original color are changed to the new color as well.

Example: If the color red was input and the pixle with the x in image 1 was passed to the function you would end up with image 2.

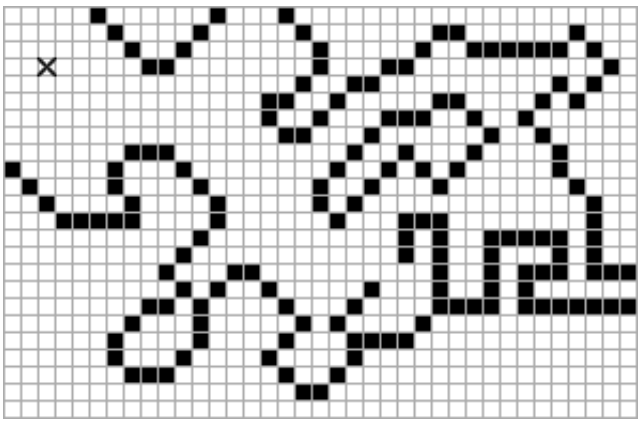
image 1: 

image 2: