# Karat - Coding Questions

## Academic Schedule

### Part 1

You are a developer for a university. Your current project is to develop a system for students to find courses they share with friends. The university has a system for querying courses students are enrolled in, returned as a list of (ID, course) pairs.

Write a function that takes in a collection of (student ID number, course name) pairs and returns, for every pair of students, a collection of all courses they share.

Examples:

```
Sample Input:
enrollments1 = [
["58", "Linear Algebra"],
["94", "Art History"],
["94", "Operating Systems"],
["17", "Software Design"],
["58", "Mechanics"],
["58", "Economics"],
["17", "Linear Algebra"],
["17", "Political Science"],
["94", "Economics"],
["25", "Economics"],
["58", "Software Design"],
]
Sample Output (pseudocode, in any order):
find_pairs(enrollments1) =>
{
"58,17": ["Software Design", "Linear Algebra"]
"58,94": ["Economics"]
"58,25": ["Economics"]
"94,25": ["Economics"]
"17,94": []
"17,25": []
}
Additional test cases:
Sample Input:
enrollments2 = [
["0", "Advanced Mechanics"],
["0", "Art History"],
["1", "Course 1"],
["1", "Course 2"],
["2", "Computer Architecture"],
["3", "Course 1"],
["3", "Course 2"],
["4", "Algorithms"]
]
```

```
Sample output:
find_pairs(enrollments2) =>
{
"1,0":[]
"2,0":[]
"2,1":[]
"3,0":[]
"3,1":["Course 1", "Course 2"]
"3,2":[]
"4,0":[]
"4,1":[]
"4,2":[]
"4,3":[]
}
Sample Input:
enrollments3 = [
["23", "Software Design"],
["3", "Advanced Mechanics"],
["2", "Art History"],
["33", "Another"],
]
Sample output:
find_pairs(enrollments3) =>
{
"23,3": []
"23,2": []
"23,33":[]
"3,2": []
"3,33": []
"2,33": []
}
All Test Cases:
find_pairs(enrollments1)
find_pairs(enrollments2)
find_pairs(enrollments3)
Complexity analysis variables:
n: number of student,course pairs in the input
s: number of students
c: total number of courses being offered (note: The number of courses any student can take i
```

## Part 2

Students may decide to take different "tracks" or sequences of courses in the Computer Science curriculum. There may be more than one track that includes the same course, but each student follows a single linear track from a "root" node to a "leaf" node. In the graph below, their path always moves left to right.

Write a function that takes a list of (source, destination) pairs, and returns the name of all of the courses that the students could be taking when they are halfway through their track of courses.

Examples:

```
Sample input 1:
all_courses_1 = [
["Logic", "COBOL"],
["Data Structures", "Algorithms"],
["Creative Writing", "Data Structures"],
["Algorithms", "COBOL"],
["Intro to Computer Science", "Data Structures"],
["Logic", "Compilers"],
["Data Structures", "Logic"],
["Graphics", "Networking"],
["Networking", "Algorithms"],
["Creative Writing", "System Administration"],
["Databases", "System Administration"],
["Creative Writing", "Databases"],
["Intro to Computer Science", "Graphics"],
]

Sample output 1 (in any order):
["Data Structures", "Networking", "Creative Writing", "Databases"]

All paths through the curriculum (midpoint highlighted):

Intro to C.S. -> Graphics -> Networking -> Algorithms -> Cobol
Intro to C.S. -> Data Structures -> Algorithms -> COBOL
Intro to C.S. -> Data Structures -> Logic -> COBOL
Intro to C.S. -> Data Structures -> Logic -> Compiler
Creative Writing -> Databases -> System Administration
Creative Writing -> System Administration
Creative Writing -> Data Structures -> Algorithms -> COBOL
Creative Writing -> Data Structures -> Logic -> COBOL
Creative Writing -> Data Structures -> Logic -> Compilers

Visual representation:

             __________    ____________
                |          |    |            |
                | Graphics |    | Networking |
           ---->|__________|--->|____________|
           |                       |
           |                       |
           |                       |  ______________
____________   |                       |  |            |
|          |   |    ______________     >->| Algorithms |--\     _____________
| Intro to |   |    |            |    /   |____________|   \    |           |
| C.S.     |---+    | Data       |   /                      >-->| COBOL     |
|__________|    \   | Structures |--+     ______________   /    |___________|
                 >->|____________|   \    |            |  /
____________    /                     \-->| Logic      |-+      _____________
|          |   /    ______________        |____________|  \     |           |
| Creative |  /     |            |                         \--->| Compilers |
| Writing  |-+----->| Databases  |                              |___________|
|__________|  \     |____________|-\     _________________________
           \                    \    |                       |
            \--------------------+-->| System Administration |
                                     |_______________________|
Sample input 2:
all_courses_2 = [
["Course_3", "Course_7"],
["Course_0", "Course_1"],
["Course_1", "Course_2"],
["Course_2", "Course_3"],
["Course_3", "Course_4"],
["Course_4", "Course_5"],
["Course_5", "Course_6"],
]

Sample output 2 (in any order):
["Course_2", "Course_3"]

Complexity analysis variables:

n: number of pairs in the input
c: number of courses in the input
```

## Book Endings

### Part 1

You are reading a Build Your Own Story book. It is like a normal book except that choices on some pages affect the story, sending you to one of two options for your next page.

These choices are really stressing you out, so you decide that you'll either always pick the first option, or always pick the second option.

You start reading at page 1 and read forward one page at a time unless you reach a choice or an ending.

The choices are provided in a list, sorted by the page containing the choice, and each choice has two options of pages to go to next. In this example, you are on page 3, where there is a choice. Option 1 goes to page 14 and Option 2 goes to page 2.

choices1 = [[3, 14, 2]] => [current_page, option_1, option_2] The ending pages are also given in a sorted list:

endings = [6, 15, 21, 30]

Given a list of endings, a list of choices with their options, and the choice you will always take (Option 1 or Option 2), return the ending you will reach. If you get stuck in a loop, return -1.

Example:

```
find_ending(endings, choices1, 1) (always Option 1)
  Start: 1 -> 2 -> 3(choice) -> 14 -> 15(end) => Return 15
```

```
find_ending(endings, choices1, 2) (always Option 2)
  Start: 1 -> 2 -> 3(choice) -> 2 -> 3(choice) -> 2... => Return -1
```

Additional inputs:

```
choices2 = [[5, 11, 28], [9, 19, 29], [14, 16, 20], [18, 7, 22], [25, 6, 30]]
choices3 = []
choices4 = [[2, 10, 15], [3, 4, 10], [4, 3, 15], [10, 3, 15]]
```

```
Complexity Variable:
n = number of pages
(endings and choices are bound by the number of pages)

All Test Cases - camelCase:
findEnding(endings, choices1, 1) => 15
findEnding(endings, choices1, 2) => -1
findEnding(endings, choices2, 1) => 21
findEnding(endings, choices2, 2) => 30
findEnding(endings, choices3, 1) => 6
findEnding(endings, choices3, 2) => 6
findEnding(endings, choices4, 1) => -1
findEnding(endings, choices4, 2) => 15

//[4, 2]
1->2->3->4
```

## Part 2

As you are reading the book multiple times, you notice that you always get bad endings. You start to suspect there is no way to get to the good endings, so you decide to find out.

Good and bad endings will be separate lists of page numbers, like this:

good_endings = [10, 15, 25, 34] bad_endings = [21, 30, 40]

Given lists of good endings, bad endings, and a list of the choices along with their options, return a collection of all the reachable good endings, if any.

Examples:

```
choices1 = [[3, 16, 24]]
find_good_endings(good_endings, bad_endings, choices1)
  Start: 1 -> 2 -> 3(choice) -> 16 -> 17... -> 21(bad ending)
                   |
                   -> 24 -> 25(good ending)
Thus it is possible to reach the good ending at 25 but no others, so we return [25].

choices2 = [[3, 16, 20]]
find_good_endings(good_endings, bad_endings, choices2)
  Start: 1 -> 2 -> 3(choice) -> 16 -> 17... -> 21(bad ending)
```

```
                    |
                    > 20 -> 21(bad ending)
No good ending is reachable, so return [].

Additional Inputs:
choices3 = [[3, 2, 19], [20, 21, 34]]
choices4 = []
choices5 = [[9, 16, 26], [14, 16, 13], [27, 29, 28], [28, 15, 34], [29, 30, 38]]
choices6 = [[9, 16, 26], [13, 31, 14], [14, 16, 13], [27, 12, 24], [32, 34, 15]]
choices7 = [[3, 9, 10]]

Complexity Variable:
n = number of pages
(endings and choices are bound by the number of pages)

All Test Cases - snake_case:
find_good_endings(good_endings, bad_endings, choices1) => [25]
find_good_endings(good_endings, bad_endings, choices2) => []
find_good_endings(good_endings, bad_endings, choices3) => [34]
find_good_endings(good_endings, bad_endings, choices4) => [10]
find_good_endings(good_endings, bad_endings, choices5) => [15, 34]
find_good_endings(good_endings, bad_endings, choices6) => [15, 25, 34]
find_good_endings(good_endings, bad_endings, choices7) => [10]

All Test Cases - camelCase:
findGoodEndings(goodEndings, badEndings, choices1) => [25]
findGoodEndings(goodEndings, badEndings, choices2) => []
findGoodEndings(goodEndings, badEndings, choices3) => [34]
findGoodEndings(goodEndings, badEndings, choices4) => [10]
findGoodEndings(goodEndings, badEndings, choices5) => [15, 34]
findGoodEndings(goodEndings, badEndings, choices6) => [15, 25, 34]
findGoodEndings(goodEndings, badEndings, choices7) => [10]
```

## Camping

### Part 1

You are going on a camping trip, but before you leave you need to buy groceries. To optimize your time spent in the store, instead of buying the items from your shopping list in order, you plan to buy everything you need from one department before moving to the next.

Given an unsorted list of products with their departments and a shopping list, return the time saved in terms of the number of department visits eliminated.

Example:

```
products = [
    ["Cheese",          "Dairy"],
    ["Carrots",         "Produce"],
    ["Potatoes",        "Produce"],
    ["Canned Tuna",     "Pantry"],
    ["Romaine Lettuce", "Produce"],
    ["Chocolate Milk",  "Dairy"],
    ["Flour",           "Pantry"],
    ["Iceberg Lettuce", "Produce"],
    ["Coffee",          "Pantry"],
    ["Pasta",           "Pantry"],
    ["Milk",            "Dairy"],
    ["Blueberries",     "Produce"],
    ["Pasta Sauce",     "Pantry"]
]

list1 = ["Blueberries", "Milk", "Coffee", "Flour", "Cheese", "Carrots"]


"Produce","Dairy", "Pantry"


For example, buying the items from list1 in order would take 5 department visits,
whereas your method would lead to only visiting 3 departments, a difference of 2 departments

Produce(Blueberries)->Dairy(Milk)->Pantry(Coffee/Flour)->Dairy(Cheese)->Produce(Carrots) = 5
New: Produce(Blueberries/Carrots)->Pantry(Coffee/Flour)->Dairy(Milk/Cheese) = 3 department v

list2 = ["Blueberries", "Carrots", "Coffee", "Milk", "Flour", "Cheese"] => 2
list3 = ["Blueberries", "Carrots", "Romaine Lettuce", "Iceberg Lettuce"] => 0
list4 = ["Milk", "Flour", "Chocolate Milk", "Pasta Sauce"] => 2
list5 = ["Cheese", "Potatoes", "Blueberries", "Canned Tuna"] => 0

All Test Cases:
shopping(products, list1) => 2
shopping(products, list2) => 2
shopping(products, list3) => 0
shopping(products, list4) => 2
shopping(products, list5) => 0

Complexity Variable:
n: number of products
```

## Part 2

You and your friends are driving to a Campground to go camping. Only 2 of you have cars, so you will be carpooling.

Routes to the campground are linear, so each location will only lead to 1 location and there will be no loops or detours. Both cars will leave from their starting locations at the same time. The first car to pass someone's location will pick them up. If both cars arrive at the same time, the person can go in either car.

Roads are provided as a directed list of connected locations with the duration (in minutes) it takes to drive between the locations. [Origin, Destination, Duration it takes to drive]

Given a list of roads, a list of starting locations and a list of people/where they live, return a collection of who will be in each car upon arrival to the Campground.

```
Bridgewater--(30)-->Caledonia--(15)-->New Grafton--(5)-->Campground
                                       ^
Liverpool---(10)---Milton-----(30)-----^

roads1 = [
    ["Bridgewater", "Caledonia", "30"], <= The road from Bridgewater to Caledonia takes 30 m
    ["Caledonia", "New Grafton", "15"],
    ["New Grafton", "Campground", "5"],
    ["Liverpool", "Milton", "10"],
    ["Milton", "New Grafton", "30"]
]

starts1 = ["Bridgewater", "Liverpool"]
people1 = [
    ["Jessie", "Bridgewater"], ["Travis", "Caledonia"],
    ["Jeremy", "New Grafton"], ["Katie", "Liverpool"]
]

Car1 path: (from Bridgewater): [Bridgewater(0, Jessie)->Caledonia(30, Travis)->New Grafton(4
Car2 path: (from Liverpool): [Liverpool(0, Katie)->Milton(10)->New Grafton(40, Jeremy)->Camp

Output (In any order/format):
    [Jessie, Travis], [Katie, Jeremy]


--------------------------------------
Riverport->Chester->Campground
             ^
Halifax------^

roads2 = [["Riverport", "Chester", "40"], ["Chester", "Campground", "60"], ["Halifax", "Ches
starts2 = ["Riverport", "Halifax"]
people2 = [["Colin", "Riverport"], ["Sam", "Chester"], ["Alyssa", "Halifax"]]

Riverport->Bridgewater->Liverpool->Campground

Output (In any order/format):
    [Colin, Sam], [Alyssa] OR [Colin], [Alyssa, Sam]
----------------------------------------
Riverport->Bridgewater->Liverpool->Campground

roads3 = [["Riverport", "Bridgewater", "1"], ["Bridgewater", "Liverpool", "1"], ["Liverpool"
starts3_1 = ["Riverport", "Bridgewater"]
starts3_2 = ["Bridgewater", "Riverport"]
starts3_3 = ["Riverport", "Liverpool"]
people3 = [["Colin", "Riverport"], ["Jessie", "Bridgewater"], ["Sam", "Liverpool"]]

Output (starts3_1/starts3_2):  [Colin], [Jessie, Sam] - (Cars can be in any order)
Output (starts3_3): [Jessie, Colin], [Sam]
----------------------------------------

All Test Cases: (Cars can be in either order)
carpool(roads1, starts1, people1) => [Jessie, Travis], [Katie, Jeremy]
carpool(roads2, starts2, people2) => [Colin, Sam], [Alyssa] OR [Colin], [Alyssa, Sam]
carpool(roads3, starts3_1, people3) => [Colin], [Jessie, Sam]
carpool(roads3, starts3_2, people3) => [Jessie, Sam], [Colin]
carpool(roads3, starts3_3, people3) => [Jessie, Colin], [Sam]
----------------------------------------
Complexity Variable:
n = number of roads
```

## Catch Cheaters

### Part 1

You are running a classroom and suspect that some of your students are passing around the answers to multiple choice questions disguised as random strings. Your task is to write a function that, given a list of words and a string, finds and returns the word in the list that is scrambled up inside the string, if any exists. There will be at most one matching word. The letters don't need to be in order or next to each other. The letters cannot be reused.

Test Case:

```
containsWord(Arrays.asList("cat", "baby", "dog", "bird", "car", "ax"), "tcabnihjs");
containsWord(Arrays.asList("cat", "baby", "dog", "bird", "car", "ax"), "tbcanihjs");
containsWord(Arrays.asList("cat", "baby", "dog", "bird", "car", "ax"), "baykkjl");
containsWord(Arrays.asList("cat", "baby", "dog", "bird", "car", "ax"), "bbabylkkj");
containsWord(Arrays.asList("cat", "baby", "dog", "bird", "car", "ax"), "ccc");
containsWord(Arrays.asList("cat", "baby", "dog", "bird", "car", "ax"), "breadmaking");
```

#### Part 2

After catching your classroom students cheating before, you realize your students are getting craftier and hiding words in 2D grids of letters. The word may start anywhere in the grid, and consecutive letters can be either immediately below or immediately to the right of the previous letter.

Given a grid and a word, write a function that returns the location of the word in the grid as a list of coordinates. If there are multiple matches, return any one.

```
grid1 = [
        ['c', 'c', 'x', 't', 'i', 'b'],
        ['c', 'c', 'a', 't', 'n', 'i'],
        ['a', 'c', 'n', 'n', 't', 't'],
        ['t', 'c', 's', 'i', 'p', 't'],
        ['a', 'o', 'o', 'o', 'a', 'a'],
        ['o', 'a', 'a', 'a', 'o', 'o'],
        ['k', 'a', 'i', 'c', 'k', 'i'],
    ]
word1 = "catnip"
word2 = "cccc"
word3 = "s"
word4 = "bit"
word5 = "aoi"
word6 = "ki"
```

```
word7 = "aaa"
word8 = "ooo"
grid2 = [['a']]
word9 = "a"
find_word_location(grid1, word1) => [ (1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (3, 4) ]
find_word_location(grid1, word2) =>
       [(0, 1), (1, 1), (2, 1), (3, 1)]
      OR [(0, 0), (1, 0), (1, 1), (2, 1)]
      OR [(0, 0), (0, 1), (1, 1), (2, 1)]
      OR [(1, 0), (1, 1), (2, 1), (3, 1)]
find_word_location(grid1, word3) => [(3, 2)]
find_word_location(grid1, word4) => [(0, 5), (1, 5), (2, 5)]
find_word_location(grid1, word5) => [(4, 5), (5, 5), (6, 5)]
find_word_location(grid1, word6) => [(6, 4), (6, 5)]
find_word_location(grid1, word7) => [(5, 1), (5, 2), (5, 3)]
find_word_location(grid1, word8) => [(4, 1), (4, 2), (4, 3)]
find_word_location(grid2, word9) => [(0, 0)]
```

## Part 3

The conflict with your students escalates, and now they are hiding multiple words in a single word grid. Return the location of each word as a list of coordinates. Letters cannot be reused across words.

```
grid1 = [
    ['b', 'a', 'b'],
    ['y', 't', 'a'],
    ['x', 'x', 't'],
]

words1_1 = ["by","bat"]

find_word_locations(grid1, words1_1) =>
([(0, 0), (1, 0)],
 [(0, 2), (1, 2), (2, 2)])

grid2 =[
    ['A', 'B', 'A', 'B'],
    ['B', 'A', 'B', 'A'],
    ['A', 'B', 'Y', 'B'],
    ['B', 'Y', 'A', 'A'],
    ['A', 'B', 'B', 'A'],
]
words2_1 = ['ABABY', 'ABY', 'AAA', 'ABAB', 'BABB']

([(0, 0), (1, 0), (2, 0), (2, 1), (3, 1)],
 [(1, 1), (1, 2), (2, 2)],
 [(3, 2), (3, 3), (4, 3)],
 [(0, 2), (0, 3), (1, 3), (2, 3)],
 [(3, 0), (4, 0), (4, 1), (4, 2)])

or

([(0, 0), (1, 0), (1, 1), (1, 2), (2, 2)],
 [(2, 0), (2, 1), (3, 1)],
 [(3, 2), (3, 3), (4, 3)],
 [(0, 2), (0, 3), (1, 3), (2, 3)],
 [(3, 0), (4, 0), (4, 1), (4, 2)])

or

([(0, 0), (0, 1), (1, 1), (1, 2), (2, 2)],
 [(2, 0), (2, 1), (3, 1)],
 [(3, 2), (3, 3), (4, 3)],
 [(0, 2), (0, 3), (1, 3), (2, 3)],
 [(3, 0), (4, 0), (4, 1), (4, 2)])

words2_2 = ['ABABA', 'ABA', 'BAB', 'BABA', 'ABYB']

([(0, 0), (1, 0), (2, 0), (3, 0), (4, 0)],
 [(3, 2), (4, 2), (4, 3)],
 [(0, 1), (0, 2), (1, 2)],
 [(0, 3), (1, 3), (2, 3), (3, 3)],
 [(1, 1), (2, 1), (3, 1), (4, 1)])

or

([(0, 0), (1, 0), (2, 0), (3, 0), (4, 0)],
 [(3, 2), (4, 2), (4, 3)],
 [(0, 1), (0, 2), (0, 3)],
 [(1, 2), (1, 3), (2, 3), (3, 3)],
 [(1, 1), (2, 1), (3, 1), (4, 1)])


Complexity analysis variables:

r = number of rows
c = number of columns
w = length of the word
```

## Cipher

### Part 1

Given a string message, place it into a matrix with row priority and column secondary, then read it out with column priority and row secondary to complete the encryption.

Example:

```
# message1 = "One does not simply walk into Mordor"
# rows1, cols1 = 6, 6

# message2 = "1.21 gigawatts!"
# rows2, cols2 = 5, 3
# rows3, cols3 = 3, 5

# print(transpose(message1, rows1, cols1))
# print("Oe y Mnss ioe iwnr nmatddoplooootlk r")
# print(transpose(message2, rows2, cols2))
# print("11iwt. gas2gat!")
# print(transpose(message2, rows3, cols3))
# print("1ga.it2gt1as w!")
```

## Part 2

You decide to create a substitution cipher. The cipher alphabet is based on a key shared amongst those of your friends who don't mind spoilers.

Suppose the key is: "The quick onyx goblin, Grabbing his sword ==}-------- jumps over the 1st lazy dwarf!".

We use only the unique letters in this key to set the order of the characters in the substitution table.

T H E Q U I C K O N Y X G B L R A S W D J M P V Z F

(spaces added for readability)

We then align it with the regular alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z T H E Q U I C K O N Y X G B L R A S W D J M P V Z F

0 1 2 3 4 5 6 7 8 0 2 1 3 4 a. c b
Which gives us the substitution mapping: A becomes T, B becomes H, C becomes E, etc.

Write a function that takes a key and a string and encrypts the string with the key.

Example: key = "The quick onyx goblin, Grabbing his sword ==}-------- jumps over the 1st lazy dwarf!" encrypt("It was all a dream.", key) -> "Od ptw txx t qsutg." encrypt("Would you kindly?", key) -> "Pljxq zlj yobqxz?"

Complexity analysis:

m: The length of the message k: The length of the key '''

## Part 3

Instead of catching up on the show your friends mounted a brute-force attack and cracked your code. They are complaining about spoilers again.

This time you choose an ambiguous arbitrary substitution cipher. Assume messages only contain letters A-Z. Each letter A-Z is mapped to some integer 1-26 (in any order), and you don't give the order to your friends.

Write a function that given an encrypted string and a list of known words returns all possible decryptions.

```
Examples:

dictionary1 = [ 'AXE', 'CAT', 'AT', 'OR', 'A', 'COO', 'CARD' ]
ciphertext1 = '123'
decrypt(dictionary1, ciphertext1) -> AXE, CAT, AT, OR
```

```
123 can be parsed as:
1 2 3 -> 3 distinct letters -> AXE, CAT
12 3 -> 2 distinct letters -> AT, OR
1 23 -> same

ciphertext2 = "122"
decrypt(dictionary1, ciphertext2) -> COO, AT, OR

122 can be parsed as:
1 2 2 -> COO
12 2 -> AT, OR
1 22 -> AT, OR

ciphertext3 = "102"
decrypt(dictionary1, ciphertext3) -> AT, OR

ciphertext4 = "1021"
decrypt(dictionary1, ciphertext4) -> AXE, CAT, AT, OR

ciphertext5 = "1105"
decrypt(dictionary1, ciphertext5) -> AXE, CAT

dictionary2 = [ 'BOXY', 'BORN', 'FORTH', 'FROTH', 'ARTERY', 'ACES', 'PORTO', 'THOR' ]
ciphertext6 = '10251826'
decrypt(dictionary2, ciphertext6) -> ARTERY, ACES, THOR, BORN, BOXY, FORTH, FROTH

All Test Cases:
decrypt(dictionary1, ciphertext1) -> AXE, CAT, AT, OR
decrypt(dictionary1, ciphertext2) -> COO, AT, OR
decrypt(dictionary1, ciphertext3) -> AT, OR
decrypt(dictionary1, ciphertext4) -> AXE, CAT, AT, OR
decrypt(dictionary1, ciphertext5) -> AXE, CAT
decrypt(dictionary2, ciphertext6) -> ARTERY, ACES, THOR, BORN, BOXY, FORTH, FROTH

Complexity analysis:

n: The length of the ciphertext
m: The size of the dictionary
k: The maximal length of each word in the dictionary
```

## Delivery Bot

### Part 1

You work in an automated robot factory. Once robots are assembled, they are sent to the shipping center via a series of autonomous delivery carts, each of which moves packages on a one-way route.

Given input that provides the (directed) steps that each cart takes as pairs, write a function that identifies all the start locations, and a collection of all of the possible ending locations for each start location.

In this diagram, starting locations are at the top and destinations are at the bottom - i.e. the graph is directed exclusively downward.

```
    A           E      J       Key:  [Origins]
   / \         / \      \                 \
B    C       F    L      M             [Destinations]
   \ /  \    /
    K     G
         / \
        H    I

paths = [
  ["B", "K"],
  ["C", "K"],
  ["E", "L"],
  ["F", "G"],
  ["J", "M"],
  ["E", "F"],
  ["C", "G"],
  ["A", "B"],
  ["A", "C"],
  ["G", "H"],

  ["G", "I"]
]

Expected output (unordered):
[
  "A": ["K", "H", "I"],
  "E": ["H", "L", "I"],
  "J": ["M"]
]
```

### Part 2

You are given a list of all available parts and a list of robots, each with its required parts. Your task is to determine which robots can be fully assembled given the available parts.

Input:

- `all_parts`: A list of available parts (strings).
- `required_parts`: A list where each element consists of a robot and its required parts (list of strings).

Output:

- A list of robots that can be fully assembled using only the available parts.

## Domain Analysis

### Question 1

```
/*
You are in charge of a display advertising program. Your ads are displayed on websites all o

counts = [ "900,google.com",
     "60,mail.yahoo.com",
     "10,mobile.sports.yahoo.com",
     "40,sports.yahoo.com",
     "300,yahoo.com",
     "10,stackoverflow.com",
     "20,overflow.com",
     "5,com.com",
     "2,en.wikipedia.org",
     "1,m.wikipedia.org",
     "1,mobile.sports",
     "1,google.co.uk"]

Write a function that takes this input as a parameter and returns a data structure containin

Sample output (in any order/format):

calculateClicksByDomain(counts) =>
com:                      1345
google.com:               900
stackoverflow.com:        10
overflow.com:             20
yahoo.com:                410
mail.yahoo.com:           60
mobile.sports.yahoo.com:  10
sports.yahoo.com:         50
com.com:                  5
org:                      3
wikipedia.org:            3
en.wikipedia.org:         2
m.wikipedia.org:          1
mobile.sports:            1
sports:                   1
uk:                       1
co.uk:                    1
google.co.uk:             1

n: number of domains in the input
(individual domains and subdomains have a constant upper length)
Time:O(n)
Space:O(n)
*/

import java.io.*;
import java.util.*;

import javax.sound.sampled.SourceDataLine;

public class Solution {
  public static void main(String[] argv) {
    String[] counts = {
      "900,google.com",
      "60,mail.yahoo.com",
      "10,mobile.sports.yahoo.com",
      "40,sports.yahoo.com",
      "300,yahoo.com",
      "10,stackoverflow.com",
      "20,overflow.com",
      "5,com.com",
      "2,en.wikipedia.org",
      "1,m.wikipedia.org",
      "1,mobile.sports",
      "1,google.co.uk"
    };
    System.out.println(calculateClicksByDomain(counts));

  }
```

## Question 2

```
/*
We have some clickstream data that we gathered on our client's website. Using cookies, we co

Write a function that takes two users' browsing histories as input and returns the longest c

Sample input:

/start


user0 = ["/start", "/green", "/blue", "/pink", "/register", "/orange", "/one/two"]
user1 = ["/start", "/pink", "/register", "/orange", "/red", "a"]
user2 = ["a", "/one", "/two"]
user3 = ["/pink", "/orange", "/yellow", "/plum", "/blue", "/tan", "/red", "/amber", "/HotRod
user4 = ["/pink", "/orange", "/amber", "/BritishRacingGreen", "/plum", "/blue", "/tan", "/re
user5 = ["a"]
user6 = ["/pink","/orange","/six","/plum","/seven","/tan","/red", "/amber"]

Sample output:

findContiguousHistory(user0, user1) => ["/pink", "/register", "/orange"]
findContiguousHistory(user0, user2) => [] (empty)
findContiguousHistory(user0, user0) => ["/start", "/green", "/blue", "/pink", "/register", "
findContiguousHistory(user2, user1) => ["a"]
findContiguousHistory(user5, user2) => ["a"]
findContiguousHistory(user3, user4) => ["/plum", "/blue", "/tan", "/red"]
findContiguousHistory(user4, user3) => ["/plum", "/blue", "/tan", "/red"]
findContiguousHistory(user3, user6) => ["/tan", "/red", "/amber"]

n: length of the first user's browsing history
m: length of the second user's browsing history

*/

import java.io.*;
import java.util.*;
import javax.sound.sampled.SourceDataLine;

public class Solution {
  public static void main(String[] argv) {
    String[] user0 = {"/start", "/green", "/blue", "/pink", "/register", "/orange", "/one/tw
    String[] user1 = {"/start", "/pink", "/register", "/orange", "/red", "a"};
    String[] user2 = {"a", "/one", "/two"};
    String[] user3 = {"/pink", "/orange", "/yellow", "/plum", "/blue", "/tan", "/red", "/amb
    String[] user4 = {"/pink", "/orange", "/amber", "/BritishRacingGreen", "/plum", "/blue",
    String[] user5 = {"a"};
    String[] user6 = {"/pink", "/orange", "/six", "/plum", "/seven", "/tan", "/red", "/amber
    //System.out.println(calculateClicksByDomain(counts));

   System.out.println(findContiguousHistory(user0, user1));
  }
```

## Question 3

```
// The people who buy ads on our network don't have enough data about how ads are working fo

// Our client provided us with a list of user IDs of customers who bought something on a lan

// # Each user completed 1 purchase.
// completed_purchase_user_ids = [
//   "3123122444","234111110", "8321125440", "99911063"]

// And our ops team provided us with some raw log data from our ad server showing every time

// ad_clicks = [
//   #"IP_Address,Time,Ad_Text",
//   "122.121.0.1,2016-11-03 11:41:19,Buy wool coats for your pets",
//   "96.3.199.11,2016-10-15 20:18:31,2017 Pet Mittens",
//   "122.121.0.250,2016-11-01 06:13:13,The Best Hollywood Coats",
//   "82.1.106.8,2016-11-12 23:05:14,Buy wool coats for your pets",
//   "92.130.6.144,2017-01-01 03:18:55,Buy wool coats for your pets",
//   "92.130.6.145,2017-01-01 03:18:55,2017 Pet Mittens",
// ]

// The client also sent over the IP addresses of all their users.

// all_user_ips = [
//   #"User_ID,IP_Address",
//    "2339985511,122.121.0.155",
//   "234111110,122.121.0.1",
//   "3123122444,92.130.6.145",
//   "39471289472,2001:0db8:ac10:fe01:0000:0000:0000:0000",
//   "8321125440,82.1.106.8",
//   "99911063,92.130.6.144"
// ]

// Write a function to parse this data, determine how many times each ad was clicked, then r
// Expected output:
// Bought Clicked Ad Text
// 1 of 2  2017 Pet Mittens
// 0 of 1  The Best Hollywood Coats
// 3 of 3  Buy wool coats for your pets
```

## Generation Graph

### Part 1

```
Given a list of connections as parent->child (like a directed graph). Identify the parent no



E.g:
[1,3]
[2,3]
[4,2]
[4,7]



parent - [1, 4]
child with 1 parent - [2, 7]
```

### Part 2

```
Suppose we have some input data describing a graph of relationships between parents and chil



[1 3 6]



[14, 4, 5, 6]
[14, 4, 8]



For example, in this diagram, 6 and 8 have common ancestors of 4 and 14.



            15
             |
      14  13
        |   |
1 2 4 12
\ / / | \ /
3 5 8 9
\ / \
6 7 11



parentChildPairs1 = [
(1, 3), (2, 3), (3, 6), (5, 6), (5, 7), (4, 5),
(4, 8), (4, 9), (9, 11), (14, 4), (13, 12),
(12, 9),(15, 13)
]


Write a function that takes the graph, as well as two of the individuals in our dataset, as


Sample input and output:


hasCommonAncestor(parentChildPairs1, 3, 8) => false
hasCommonAncestor(parentChildPairs1, 5, 8) => true
hasCommonAncestor(parentChildPairs1, 6, 8) => true
hasCommonAncestor(parentChildPairs1, 6, 9) => true
hasCommonAncestor(parentChildPairs1, 1, 3) => false
hasCommonAncestor(parentChildPairs1, 3, 1) => false
hasCommonAncestor(parentChildPairs1, 7, 11) => true
hasCommonAncestor(parentChildPairs1, 6, 5) => true
hasCommonAncestor(parentChildPairs1, 5, 6) => true


Additional example: In this diagram, 4 and 12 have a common ancestor of 11.


    11
   /  \
  10   12
 /  \
1 2 5
\ / /
3 6 7
\
4 8


parentChildPairs2 = [
(1, 3), (11, 10), (11, 12), (2, 3), (10, 2),
(10, 5), (3, 4), (5, 6), (5, 7), (7, 8),
]


hasCommonAncestor(parentChildPairs2, 4, 12) => true
hasCommonAncestor(parentChildPairs2, 1, 6) => false
hasCommonAncestor(parentChildPairs2, 1, 12) => false


n: number of pairs in the input
```

## Part 3

```
Suppose we have some input data describing a graph of relationships between parents and chil
For example, in this diagram, the earliest ancestor of 6 is 14, and the earliest ancestor of
14
```

```
|
2      4
|    / | \
3   5  8  9
/ \ / \     \
15  6   7    11

Write a function that, for a given individual in our dataset, returns their earliest known a
Sample input and output:
pairs1 = [
(2, 3), (3, 15), (3, 6), (5, 6), (5, 7),
(4, 5), (4, 8), (4, 9), (9, 11), (14, 4),
]
findEarliestAncestor(pairs1, 8) => 14
findEarliestAncestor(pairs1, 7) => 14
findEarliestAncestor(pairs1, 6) => 14
findEarliestAncestor(pairs1, 15) => 2
findEarliestAncestor(pairs1, 14) => null or -1
findEarliestAncestor(pairs1, 11) => 14
```

## Mini Game

You're creating a game with some amusing mini-games, and you've decided to make a simple variant of the game **Mahjong**.

In this variant, players have a number of tiles, each marked with digits from 0 to 9. The tiles can be grouped into **pairs** or **triples** of the same tile. For example:

- If a player has the hand `"33344466"`, it contains a triple of 3s, a triple of 4s, and a pair of 6s.
- Similarly, the hand `"55555777"` contains a triple of 5s, a pair of 5s, and a triple of 7s.

A **complete hand** is defined as a collection of tiles where all the tiles can be grouped into any number of triples (zero or more) and **exactly one pair**. Each tile must be used in exactly one triple or pair.

Write a function that takes a string representation of a collection of tiles (in no particular order) and returns `True` or `False`, depending on whether or not the collection represents a complete hand.

### Examples:

```
tiles_1 = "88844"
# True. Base case - a pair and a triple.

tiles_2 = "99"
# True. Just a pair is enough.

tiles_3 = "55555"
# True. The triple and a pair can be of the same tile value.

tiles_4 = "22333333"
# True. A pair and two triples.
```

```
tiles_5 = "737974399494994773399777779973949479474779993"
# True. Multiple valid triples and one pair.

tiles_6 = "111333555"
# False. There are three triples (111, 333, 555) but no pair.

tiles_7 = "42"
# False. Two singles that do not form a pair.

tiles_8 = "888"
# False. A triple, but no pair.

tiles_9 = "100100000"
# False. A pair of 1s and two triples of 0s, but there's a leftover tile.

tiles_10 = "346664366"
# False. Three pairs and one triple.

tiles_11 = "8999998999898"
# False. A triple of 8, three triples of 9, but there's a leftover 8.

tiles_12 = "17610177"
# False. Triples of 1 and 7, but leftovers of 6 and other tiles.

tiles_13 = "600061166"
# False. A pair of 1, triple of 0, triple of 6, and a leftover of 6.

tiles_14 = "6996999"
# False. A pair of 6, a triple of 9, and another pair of 9.

tiles_15 = "03799449"
# False. A pair of 4s, a triple of 9s, and leftover tiles (0, 3, 7).

tiles_16 = "64444333355556"
# False. A pair of 6, but two pairs each of 3, 4, 5.

tiles_17 = "7"
# False. A single tile with no pairs.

tiles_18 = "776655"
# False. Three pairs.
```

```
complete(tiles_1) => True
complete(tiles_2) => True
complete(tiles_3) => True
complete(tiles_4) => True
complete(tiles_5) => True
complete(tiles_6) => False
complete(tiles_7) => False
complete(tiles_8) => False
complete(tiles_9) => False
complete(tiles_10) => False
complete(tiles_11) => False
complete(tiles_12) => False
complete(tiles_13) => False
complete(tiles_14) => False
complete(tiles_15) => False
complete(tiles_16) => False
complete(tiles_17) => False
complete(tiles_18) => False
```

**Complexity Variable:**

- **N** - Number of tiles in the input string.

## Part 2

You've decided to make a more advanced version of the Mahjong minigame, this time incorporating runs.

A run is a series of three consecutive tiles, like 123, 678, or 456. 0 counts as the lowest tile, so 012 is a valid run, but 890 is not. A complete hand now consists of a pair, and any number (including zero) of triples and/or three-tile runs.

Write a function that returns whether or not a hand is complete under these new rules.

```
hand1 = "11123"        # True. 11 123
hand2 = "12131"        # True. Also 11 123. Tiles are not necessarily sorted.
hand3 = "11123455"     # True. 111 234 55
hand4 = "11122334"     # True. 111 223 234
hand5 = "11234"        # True. 11 234
hand6 = "123456"       # False. Needs a pair
hand7 = "111333555777" # True. 111 333 555 77
hand8 = "11223344556677"  # True. 11 234 234 567 567 among others
hand9 = "12233444556677"  # True. 123 234 44 567 567
hand10 = "1123456789"  # False
hand11 = "00123457"    # False
hand12 = "0012345"     # False. A run is only three tiles
hand13 = "11890"       # False. 890 is not a valid run
hand14 = "99"          # True
hand15 = "11122344"    # False

All Test Cases
advanced(hand1) => True
advanced(hand2) => True
advanced(hand3) => True
advanced(hand4) => True
advanced(hand5) => True
advanced(hand6) => False
advanced(hand7) => True
advanced(hand8) => True
advanced(hand9) => True
advanced(hand10) => False
advanced(hand11) => False
advanced(hand12) => False
advanced(hand13) => False
advanced(hand14) => True
advanced(hand15) => False
```

## Most Powerful Card

```
While on vacation we have been playing a new deck building card game. In this game cards can

Beating is transitive, that is if A beats B and B beats C then A beats C. If A beats B then

We are trying to figure out which card is the most powerful, meaning the card that beats the

matchups_1 = [
  # giant beats wizard
  ['giant', 'wizard'],
  ['giant', 'nymph'],
  ['wizard', 'elf'],
  ['nymph', 'muse'],
  ['orc', 'elf'],
  ['orc', 'goblin'],
  ['orc', 'snake']
]

Even though the orc directly beats three cards (elf, goblin, snake), the giant is the most p

Write a function that takes in a list of card matchups and returns the card that beats the m

All test cases:
best_card(matchups_1) => giant
best_card(matchups_2) => roc
best_card(matchups_3) => vampire
best_card(matchups_4) => orc
best_card(matchups_5) => elf
best_card(matchups_6) => human
best_card(matchups_7) => dinosaur

Complexity variables:
N = the length of the matchups list
Time : O(N), worst case will be different
Space : O(N)
```

## Movie Recommendation

### Part 1

You are given a 2D list representing **follow/unfollow actions** between users. Each action has the format: `[follower, followed, action]`, where:

- `"CONNECT"` means the follower follows the followed user.
- `"DISCONNECT"` means the follower unfollows the followed user.

Example input:

```
events = [
    ["Nicole", "Alice", "CONNECT"],
    ["Nicole", "Alice", "DISCONNECT"],
    ["Charlie", "Alice", "CONNECT"],
    ["Edward", "Alice", "CONNECT"]
]
```

Given a threshold `n`, return a **2D list** where:

- The first sublist contains users **following less than `n` people**.
- The second sublist contains users **following `n` or more people**.

The order of users in the output **does not need to be sorted**.

Example outputs:

```
grouping(events, 3) => [["Nicole", "Alice", "Charlie", "Pam"], ["Bob", "Dennis", "Edward"]]
grouping(events, 1) => [[], ["Nicole", "Bob", "Alice", "Charlie", "Dennis", "Edward", "Pam"]
grouping(events, 10) => [["Nicole", "Bob", "Alice", "Charlie", "Dennis", "Edward", "Pam"], [
```

## Part 2

You are given a list of **movie ratings** in the format `[user, movie, rating]`.

Example input:

```
ratings = [
    ["Alice", "Frozen", "5"],
    ["Bob", "Mad Max", "5"],
    ["Dennis", "Mad Max", "4"],
    ["Bob", "Lost In Translation", "5"]
]
```

Two users are considered **similar** if:

- They have both watched the **same movie** and
- They both rated it **above 3**.

If two users are **similar**, they can **recommend** movies to each other that one has seen but the other has not.

Example:

- Bob and Dennis both watched **Mad Max** and rated it above 3.
- Bob has watched **Lost In Translation**, but Dennis has not.
- Bob can recommend **Lost In Translation** to Dennis.

Function output:

```
recommendations("Dennis", ratings) => ["Lost In Translation"]
```

## Passage Tracker

```
/*
We are writing software to analyze logs for toll booths on a highway. This highway is a divi
There are three types of toll booths:
* ENTRY (E in the diagram) toll booths, where a car goes through a booth as it enters the hi
* EXIT (X in the diagram) toll booths, where a car goes through a booth as it exits the high
* MAINROAD (M in the diagram), which have sensors that record a license plate as a car drive
        Entry Booth                      Exit Booth
            |                                |
            E                                X
           /                                  \
---<------------<---------M---------<-----------<---------<----
                                         (West-bound side)
===============================================================
                                         (East-bound side)
------>---------->---------M---------->---------->---------->------
           \                                   /
            X                                 E
            |                                 |
        Exit Booth                      Entry Booth
For our first task:
1-1) Read through and understand the code and comments below. Feel free to run the code and
1-2) The tests are not passing due to a bug in the code. Make the necessary changes to LogEn
We are interested in how many people are using the highway, and so we would like to count ho
A complete journey consists of:
* A driver entering the highway through an ENTRY toll booth.
* The driver passing through some number of MAINROAD toll booths (possibly 0).
* The driver exiting the highway through an EXIT toll booth.
For example, the following log lines represent a single complete journey:
90750.191 JOX304 250E ENTRY
91081.684 JOX304 260E MAINROAD
91483.251 JOX304 270E MAINROAD
91874.493 JOX304 280E EXIT
You may assume that the log only contains complete journeys, and there are no missing entrie
2-1) Write a function in LogFile named countJourneys() that returns how many
     complete journeys there are in the given LogFile.
We would like to catch people who are driving at unsafe speeds on the highway. To help us do
* Drive an average of 130 km/h or greater in any individual 10km segment of tollway.
* Drive an average of 120 km/h or greater in any two 10km segments of tollway.
For example, consider the following journey:
1000.000 TST002 270W ENTRY
1275.000 TST002 260W EXIT
In this case, the driver of TST002 drove 10 km in 275 seconds. We can calculate
that this driver drove an average speed of ~130.91km/hr over this segment:
10 km * 3600 sec/hr
------------------- = 130.91 km/hr
      275 sec
Note that:
```

```
* A license plate may have multiple journeys in one file, and if they drive at unsafe speeds
* We do not mark speeding if they are not on the highway (i.e. for any driving between an EX
* Speeding is only marked once per journey. For example, if there are 4 segments 120km/h or 
3-1) Write a function catchSpeeders in LogFile that returns a collection of license plates t
     If the same license plate drives at unsafe speeds during two different journeys, the li
*/
import static org.junit.Assert.assertEquals;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import javax.swing.text.Segment;
import com.sun.tools.javac.util.Log;
class LogEntry {
  /**
   * Represents an entry from a single log line. Log lines look like this in the file:
   *
   * 34400.409 SXY288 210E ENTRY
   *
   * Where:
   * * 34400.409 is the timestamp in seconds since the software was started.
   * * SXY288 is the license plate of the vehicle passing through the toll booth.
   * * 210E is the location and traffic direction of the toll booth. Here, the toll
   *     booth is at 210 kilometers from the start of the tollway, and the E indicates
   *     that the toll booth was on the east-bound traffic side. Tollbooths are placed
   *     every ten kilometers.
   * * ENTRY indicates which type of toll booth the vehicle went through. This is one of
   *     "ENTRY", "EXIT", or "MAINROAD".

   **/
  private final String timestamp;
  private final String licensePlate;
  private final String boothType;
  private final int location;
  private final String direction;
  public LogEntry(String logLine) {
    String[] tokens = logLine.split(" ");
    this.timestamp = tokens[0];
    this.licensePlate = tokens[1];
    this.boothType = tokens[3];
    this.location =
      Integer.parseInt(tokens[2].substring(0, tokens[2].length() - 1));
    String directionLetter = tokens[2].substring(tokens[2].length() - 1);
    if (directionLetter.equals("E")) {
      this.direction = "EAST";
    } else if (directionLetter.equals("W")) {
      this.direction = "WEST";
    } else {
      throw new IllegalArgumentException();
    }
  }
  public String getTimestamp() {
    return timestamp;
  }
  public String getLicensePlate() {
    return licensePlate;
  }
  public String getBoothType() {
    return boothType;
  }
  public int getLocation() {
    return location;
  }
  public String getDirection() {
    return direction;
  }
  @Override
  public String toString() {
    return String.format(
      "<LogEntry timestamp: %f  license: %s  location: %d  direction: %s  booth type: %s>",
      timestamp,
      licensePlate,
      location,
      direction,
      boothType
    );
  }
}
class LogFile {
  /*
   * Represents a file containing a number of log lines, converted to LogEntry
   * objects.
   */
  List<LogEntry> logEntries;
  public LogFile(BufferedReader reader) throws IOException {
    this.logEntries = new ArrayList<>();
    String line = reader.readLine();
    while (line != null) {
      LogEntry logEntry = new LogEntry(line.strip());
      this.logEntries.add(logEntry);
      line = reader.readLine();
    }
  }
  public LogEntry get(int index) {
    return this.logEntries.get(index);
  }
  public int size() {
    return this.logEntries.size();
  }
  public int countJourneys() {
  }
  public List<String> catchSpeeders() {
  }
}
public class Solution {
  public static void main(String[] argv) throws IOException {
    testLogFile();
    testLogEntry();
    testCountJourneys();
    testCatchSpeeders();
  }
// 2-1) Write a function in LogFile named countJourneys() that returns how many
//      complete journeys there are in the given LogFile.
  public static void testLogFile() throws IOException {
    System.out.println("Running testLogFile");
    try (
      BufferedReader reader = new BufferedReader(
        new FileReader("/content/test/tollbooth_small.log")
      );
    ) {
      LogFile log_file = new LogFile(reader);
      assertEquals(13, log_file.size());
      for (LogEntry entry : log_file.logEntries) {
        assert (entry instanceof LogEntry);
      }
    }
  }
  public static void testLogEntry() {
    System.out.println("Running testLogEntry");
    String log_line = "44776.619 KTB918 310E MAINROAD";
    LogEntry log_entry = new LogEntry(log_line);
    assertEquals(44776.619f, log_entry.getTimestamp(), 0.0001);
    assertEquals("KTB918", log_entry.getLicensePlate());
    assertEquals(310, log_entry.getLocation());
    assertEquals("EAST", log_entry.getDirection());
    assertEquals("MAINROAD", log_entry.getBoothType());
    log_line = "52160.132 ABC123 400W ENTRY";
    log_entry = new LogEntry(log_line);
    assertEquals(52160.132f, log_entry.getTimestamp(), 0.0001);
    assertEquals("ABC123", log_entry.getLicensePlate());
    assertEquals(400, log_entry.getLocation());
    assertEquals("WEST", log_entry.getDirection());
    assertEquals("ENTRY", log_entry.getBoothType());
  }
  public static void testCountJourneys() throws IOException {
    System.out.println("Running testCountJourneys");
    try (BufferedReader reader = new BufferedReader(new FileReader("/content/test/tollbooth_
        LogFile logFile = new LogFile(reader);
        assertEquals(3, logFile.countJourneys());
    }
    try (BufferedReader reader = new BufferedReader(new FileReader("/content/test/tollbooth_
        LogFile logFile = new LogFile(reader);
        assertEquals(63, logFile.countJourneys());
    }
  }
  public static void testCatchSpeeders() throws IOException {
    System.out.println("Running testCatchSpeeders");
    try (BufferedReader reader = new BufferedReader(new FileReader("/content/test/tollbooth_
        LogFile logFile = new LogFile(reader);
        List<String> ticketList = logFile.catchSpeeders();
        // ticketList should be a list similar to
        // ["TST002", "TST003", "TST003"]
        // In this case, TST002 had one journey with unsafe driving, and
        // TST003 had two journeys with unsafe driving. The license plates
        // may be in any order.
        Map<String, Integer> ticketCounts = new HashMap<>();
        for (String ticket : ticketList) {
            ticketCounts.put(ticket, ticketCounts.getOrDefault(ticket, 0) + 1);
        }
        assertEquals(1, (int) ticketCounts.get("TST002"));
        assertEquals(2, (int) ticketCounts.get("TST003"));
        assertEquals(2, ticketCounts.size());
    }
    try (BufferedReader reader = new BufferedReader(new FileReader("/content/test/tollbooth_
        LogFile logFile = new LogFile(reader);
        List<String> ticketList = logFile.catchSpeeders();
        assertEquals(10, ticketList.size());
    }
    try (BufferedReader reader = new BufferedReader(new FileReader("/content/test/tollbooth_
        LogFile logFile = new LogFile(reader);
        List<String> ticketList = logFile.catchSpeeders();
        assertEquals(129, ticketList.size());
    }
  }
}
```

## Picking Restaurant

```
/* Two friends are getting together to have dinner at a restaurant. They can't figure out wh
The two friends want to choose a restaurant such that:
* Neither of them already likes it, and
* It is liked by the most number of their mutual friends.
Suppose you have data about who is friends with whom and what restaurants they've liked.
friends_1 = [
[ "Ted", "Lily" ],
[ "Ted", "Robin" ],
[ "Lily", "Robin" ],
[ "Ted", "Marshall" ],
[ "Marshall", "Lily" ]
]
likes_1 = [
[ "Lily", "Restaurant_1", "Restaurant_17", "Restaurant_3" ],
[ "Ted", "Restaurant_17", "Restaurant_1" ],
[ "Robin", "Restaurant_5" ],
[ "Marshall", "Restaurant_17", "Restaurant_5", "Restaurant_4" ],
]
In this example, Ted and Lily should go to Restaurant_5. Their mutual friends are Robin and
Write a function that given the friendship data, the likes data, and two names, will return
All test cases:
restaurant_recs(friends_1, likes_1, "Ted", "Lily")      -> Restaurant_5
restaurant_recs(friends_2, likes_2, "Ted", "Lily")      -> Restaurant_5
restaurant_recs(friends_2, likes_2, "Ted", "Ranjit")    -> None
restaurant_recs(friends_2, likes_2, "Lily", "Marshall") -> None
restaurant_recs(friends_1, likes_4, "Ted", "Lily")      -> Restaurant_4
restaurant_recs(friends_2, likes_3, "Ted", "Lily")      -> Restaurant_4 or Restaurant_5
Complexity variables:
N = the number of people
M = the number of restaurants */
```

# Puzzle Checker

## Part 1

Determine if a N*N matrix is a "valid" matrix. The definition of a "valid" matrix is:

1. Each row must contain exactly the numbers 1 to N (no duplicates, no missing numbers)
2. Each column must contain exactly the numbers 1 to N (no duplicates, no missing numbers)

## Part 2

```
"""
A nonogram is a logic puzzle, similar to a crossword, in which the player is given a blank g

+-------------+
| 1  1  1  1 |
| 0  1  1  1 |
| 0  1  0  0 |
| 1  1  0  1 |
| 0  0  1  1 |
+-------------+

For each row and column, the instructions give the lengths of contiguous runs of black (0) c

These are valid solutions: [ 1, 0, 0, 1, 0 ] and [ 0, 0, 1, 1, 0 ] and also [ 0, 0, 1, 0, 1
This is not valid: [ 1, 0, 1, 0, 0 ] since the runs are not in the correct order.
This is not valid: [ 1, 0, 0, 0, 1 ] since the two runs of 0s are not separated by 1s.

Your job is to write a function to validate a possible solution against a set of instruction

Example instructions #1

matrix1 = [[1,1,1,1],
           [0,1,1,1],
           [0,1,0,0],
           [1,1,0,1],
           [0,0,1,1]]
rows1_1    =  [], [1], [1,2], [1], [2]
columns1_1 =  [2,1], [1], [2], [1]
validateNonogram(matrix1, rows1_1, columns1_1) => True

Example solution matrix:
matrix1 ->
                                   row
                +-------------+     instructions
                | 1  1  1  1 | <-- []
                | 0  1  1  1 | <-- [1]
                | 0  1  0  0 | <-- [1,2]
                | 1  1  0  1 | <-- [1]
                | 0  0  1  1 | <-- [2]
                +-------------+
                  ^  ^  ^  ^
                  |  |  |  |
  column       [2,1] | [2] |
  instructions      [1]   [1]


Example instructions #2

(same matrix as above)
rows1_2    =  [], [], [1], [1], [1,1]
columns1_2 =  [2], [1], [2], [1]
validateNonogram(matrix1, rows1_2, columns1_2) => False

The second and third rows and the first column do not match their respective instructions.

Example instructions #3

matrix2 = [
[ 1, 1 ],
[ 0, 0 ],
[ 0, 0 ],
[ 1, 0 ]
]
rows2_1    = [], [2], [2], [1]
columns2_1 = [1, 1], [3]
validateNonogram(matrix2, rows2_1, columns2_1) => False

The black cells in the first column are not separated by white cells.

n: number of rows in the matrix
m: number of columns in the matrix
"""
```

## Resource Access Log

### Question 1

Suppose we have an unsorted log file of accesses to web resources. Each log entry consists of an access time, the ID of the user making the access, and the resource ID. The access time is represented as seconds since 00:00:00, and all times are assumed to be in the same day. For example:

```
logs1 = [
["58523", "user_1", "resource_1"],
["62314", "user_2", "resource_2"],
["54001", "user_1", "resource_3"],
["200", "user_6", "resource_5"],
["215", "user_6", "resource_4"],
["54060", "user_2", "resource_3"],
["53760", "user_3", "resource_3"],
["58522", "user_22", "resource_1"],
["53651", "user_5", "resource_3"],
["2", "user_6", "resource_1"],
["100", "user_6", "resource_6"],
["400", "user_7", "resource_2"],
["100", "user_8", "resource_6"],
["54359", "user_1", "resource_3"],
]
```

Example 2:

```
logs2 = [
["300", "user_1", "resource_3"],
["599", "user_1", "resource_3"],
["900", "user_1", "resource_3"],
["1199", "user_1", "resource_3"],
["1200", "user_1", "resource_3"],
["1201", "user_1", "resource_3"],
["1202", "user_1", "resource_3"]
]
```

Write a function that takes the logs and returns each users min and max access timestamp Example Output:

```
user_3:[53760,53760]
user_2:[54060,62314]
user_1:[54001,58523]
user_7:[400,400]
user_6:[2,215]
user_5:[53651,53651]
user_4:[58522,58522]
user_8:[100,100]
```

## Question 2

Write a function that takes the logs and returns the resource with the highest number of accesses in any 5 minute window, together with how many accesses it saw.

Expected Output:

```
most_requested_resource(logs1) # => ('resource_3', 3)
```

Reason: resource_3 is accessed at 53760, 54001, and 54060

## Question 3

Write a function that takes the logs as input, builds the transition graph and returns it as an adjacency list with probabilities. Add START and END states.

Specifically, for each resource, we want to compute a list of every possible next step taken by any user, together with the corresponding probabilities. The list of resources should include START but not END, since by definition END is a terminal state.

Expected output for logs1:

```
transition_graph(logs1) # =>
{{
'START': {'resource_1': 0.25, 'resource_2': 0.125, 'resource_3': 0.5, 'resource_6': 0.125},
'resource_1': {'resource_6': 0.333, 'END': 0.667},
'resource_2': {'END': 1.0},
'resource_3': {'END': 0.4, 'resource_1': 0.2, 'resource_2': 0.2, 'resource_3': 0.2},
'resource_4': {'END': 1.0},
'resource_5': {'resource_4': 1.0},
'resource_6': {'END': 0.5, 'resource_5': 0.5}
}}
```

For example, of 8 total users, 4 users have resource_3 as a first visit (user_1, user_2, user_3, user_5), 2 users have resource_1 as a first visit (user_6, user_22), 1 user has resource_2 as a first visit (user_7), and 1 user has resource_6 (user_8) so the possible next steps for START are resource_3 with probability 4/8, resource_1 with probability 2/8, and resource_2 and resource_6 with probability 1/8.For example, of 8 total users, 4 users have resource_3 as a first visit (user_1, user_2, user_3, user_5), 2 users have resource_1 as a first visit (user_6, user_22), 1 user has resource_2 as a first visit (user_7), and 1 user has resource_6 (user_8) so the possible next steps for START are resource_3 with probability 4/8, resource_1 with probability 2/8, and resource_2 and resource_6 with probability 1/8. These are the resource paths per user for the first logs example, ordered by access time:

```
{{
'user_1': ['resource_3', 'resource_3', 'resource_1'],
'user_2': ['resource_3', 'resource_2'],
'user_3': ['resource_3'],
'user_5': ['resource_3'],
'user_6': ['resource_1', 'resource_6', 'resource_5', 'resource_4'],
'user_7': ['resource_2'],
'user_8': ['resource_6'],
'user_22': ['resource_1'],

Expected output for logs2:
transition_graph(logs2) # =>
'START': {'resource_3': 1.0},
'resource_3': {'resource_3: 0.857, 'END': 0.143}
}
```

## Snake Exits

### Part 1

```
/*
    We have a two-dimensional board game involving snakes.  The board has two types of squar
    impassable squares where snakes cannot go, and 0's represent squares through which snake
    Snakes can only enter on the edges of the board, and each snake can move in only one dir
    We'd like to find the places where a snake can pass through the entire board, moving in
    Here is an example board:
```

```
    col-->       0  1  2  3  4  5  6
    +----------------------
    row     0 |  +  +  +  0  +  0  0
    |       1 |  0  0  +  0  0  0  0
    |       2 |  0  0  0  0  +  0  0
    v       3 |  +  +  +  0  0  +  0
            4 |  0  0  0  0  0  0  0
    Write a function that takes a rectangular board with only +'s and 0's, and returns two c
    * one containing all of the row numbers whose row is completely passable by snakes, and
    * the other containing all of the column numbers where the column is completely passable
    Sample Inputs:
    board1 = [['+', '+', '+', '0', '+', '0', '0'],
    ['0', '0', '+', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '+', '0', '0'],
    ['+', '+', '+', '0', '0', '+', '0'],
    ['0', '0', '0', '0', '0', '0', '0']]
    board2 = [['+', '+', '+', '0', '+', '0', '0'],
    ['0', '0', '0', '0', '0', '+', '0'],
    ['0', '0', '+', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '+', '0', '0'],
    ['+', '+', '+', '0', '0', '0', '+']]
    board3 = [['+', '+', '+', '0', '+', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '+', '+', '0', '+', '0'],
    ['0', '0', '0', '0', '+', '0', '0'],
    ['+', '+', '+', '0', '0', '0', '+']]
    board4 = [['+']]
    board5 = [['0']]

    All test cases:
    findPassableLanes(board1) => Rows: [4], Columns: [3, 6]
    findPassableLanes(board2) => Rows: [], Columns: [3]
    findPassableLanes(board3) => Rows: [1], Columns: []
    findPassableLanes(board4) => Rows: [], Columns: []
    findPassableLanes(board5) => Rows: [0], Columns: [0]

    Complexity Analysis:
    r: number of rows in the board
    c: number of columns in the board
*/
```

## Part 2

```
Snakes may move in any of four directions - up, down, left, or right -
one square at a time, but they will never return to a square that
they've already visited. If a snake enters the board on an edge
square, we want to catch it at a different exit square on the board's
edge.
The snake is familiar with the board and will take the route to the nearest reachable exit,
move through to get there. Note that there may not be a reachable
exit.
Write a function that takes a rectangular board with only +'s and O's, along with a starting

Sample inputs:
```

```
board1 = [['+', '+', '+', '+', '+', '+', '+', '0', '0'],
['+', '+', '0', '0', '0', '0', '0', '+', '+'],
['0', '0', '0', '0', '0', '+', '+', '0', '+'],
['+', '+', '0', '+', '+', '+', '+', '0', '0'],
['+', '+', '0', '0', '0', '0', '0', '0', '+'],
['+', '+', '0', '+', '+', '0', '+', '0', '+']]
start1_1 = (2, 0) # Expected output = (5, 2)
start1_2 = (0, 7) # Expected output = (0, 8)
start1_3 = (5, 2) # Expected output = (2, 0) or (5, 5)
start1_4 = (5, 5) # Expected output = (5, 7)
board2 = [['+', '+', '+', '+', '+', '+', '+'],
['0', '0', '0', '0', '+', '0', '+'],
['+', '0', '+', '0', '+', '0', '0'],
['+', '0', '0', '0', '+', '+', '+'],
['+', '+', '+', '+', '+', '+', '+']]
start2_1 = (1, 0) # Expected output = null (or a special value
representing no possible exit)
start2_2 = (2, 6) # Expected output = null

board3 = [['+', '0', '+', '0', '+',],
['0', '0', '+', '0', '0',],
['+', '0', '+', '0', '+',],
['0', '0', '+', '0', '0',],
['+', '0', '+', '0', '+']]
start3_1 = (0, 1) # Expected output = (1, 0)
start3_2 = (4, 1) # Expected output = (3, 0)
start3_3 = (0, 3) # Expected output = (1, 4)
start3_4 = (4, 3) # Expected output = (3, 4)
board4 = [['+', '0', '+', '0', '+',],
['0', '0', '0', '0', '0',],
['+', '+', '+', '+', '+',],
['0', '0', '0', '0', '0',],
['+', '0', '+', '0', '+']]
start4_1 = (1, 0) # Expected output = (0, 1)
start4_2 = (1, 4) # Expected output = (0, 3)
start4_3 = (3, 0) # Expected output = (4, 1)
start4_4 = (3, 4) # Expected output = (4, 3)
board5 = [['+', '0', '0', '0', '+',],
['+', '0', '+', '0', '+',],
['+', '0', '0', '0', '+',],
['+', '0', '+', '0', '+']]
start5_1 = (0, 1) # Expected output = (0, 2)
start5_2 = (3, 1) # Expected output = (0, 1)

All test cases:
findExit(board1, start1_1) => (5, 2)
findExit(board1, start1_2) => (0, 8)
findExit(board1, start1_3) => (2, 0) or (5, 5)
findExit(board1, start1_4) => (5, 7)
findExit(board2, start2_1) => null (or a special value representing no
possible exit)
findExit(board2, start2_2) => null
findExit(board3, start3_1) => (1, 0)
findExit(board3, start3_2) => (3, 0)
findExit(board3, start3_3) => (1, 4)
findExit(board3, start3_4) => (3, 4)
findExit(board4, start4_1) => (0, 1)
findExit(board4, start4_2) => (0, 3)
findExit(board4, start4_3) => (4, 1)
findExit(board4, start4_4) => (4, 3)
findExit(board5, start5_1) => (0, 2)
findExit(board5, start5_2) => (0, 1)
*/
```

## Part 3

```
Snakes make nests of open spaces, but they avoid connecting their nest with other snakes' ne
Here is an example board:
col-->        0  1  2  3  4  5  6  7  8
+---------------------------
row      0 |  +  +  +  +  +  +  +  0  0
|       1 |  +  +  0  0  0  0  0  +  +
|       2 |  0  0  0  0  0  +  +  0  +
v       3 |  +  +  0  +  +  +  +  0  0
4 |  +  +  0  0  0  0  0  0  +
5 |  +  +  0  +  +  0  +  0  +

Given a board, return a collection that, for each nest, lists the nest's number of entrances
Sample inputs:
board1 = [['+', '+', '+', '+', '+', '+', '+', '0', '0'],
['+', '+', '0', '0', '0', '0', '0', '+', '+'],
['0', '0', '0', '0', '0', '+', '+', '0', '+'],
['+', '+', '0', '+', '+', '+', '+', '0', '0'],
['+', '+', '0', '0', '0', '0', '0', '0', '+'],
['+', '+', '0', '+', '+', '0', '+', '0', '+']]
board2 = [['+', '+', '+', '+', '+', '+'],
['0', '0', '0', '+', '0', '+'],
['+', '0', '+', '0', '0', '0'],
['+', '+', '+', '+', '+', '+']]
board3 = [['+', '0', '+', '+', '+', '0', '+', '0', '0'],
['+', '0', '+', '0', '0', '0', '0', '+', '+'],
['0', '0', '0', '0', '0', '+', '+', '0', '+'],
['+', '+', '+', '+', '+', '+', '+', '0', '0'],
['+', '+', '0', '0', '0', '0', '0', '0', '+'],
['+', '+', '0', '+', '+', '0', '+', '0', '+']]
board4 = [['+', '+', '+'],
['+', '0', '+'],
['+', '+', '+']]
board5 = [['+']]
board6 = [['0', '0'],
['0', '0']]

All test cases:
getNestEntranceCount(board1) => [2, 5]
getNestEntranceCount(board2) => [1, 1]
getNestEntranceCount(board3) => [2, 4, 3]
getNestEntranceCount(board4) => [0]
getNestEntranceCount(board5) => []
getNestEntranceCount(board6) => [4]
```

```
Complexity Analysis:
r: number of rows in the board
c: number of columns in the board
```

## Snowy Mountain

```
/*
You are planning out a trek across a snowy mountain. On the mountain it snows in the morning

* Snow piles up at each location, making that location higher.
* If it has not snowed at a particular location for 2 days, the snow there starts melting on
* You can climb up and down one level while moving to the next position.
* The player needs to cross the mountain with the least amount of climbing possible.
* The crossing attempts are limited to the days in the forecast because the weather is unpre

Write a function that, given the base altitude of locations on the mountain and a list of sn

For example, given the initial altitudes: [0,1,2,1]

          3
  altitude 2     -
          1   -   -      Side view of the mountain
          0 -

            0 1 2 3
            position

And the snow forecast for each morning:

[[1,0,1,0],   # On day zero, one unit of snow will fall on positions 0 and 2.

  [0,0,0,0],   # On day one, it will not snow.

  [1,1,0,2]]   # On day two, two units of snow will fall on position 3, and one unit on posi

This is the resulting mountain profile each evening, the player is represented by the letter

            Day 0            Day 1            Day 2

                                         starts melting
                                               ↓
         3     *                         3 P     *
altitude  2 P   -          no new snow     2 * * - *
         1 * -   -                       1 * -   -
         0 -               no melting      0 -
           0 1 2 3                         0 1 2 3

           position                        position

[0,1,2,1] -> [1,0,1,0] -> [1,1,3,1] ->X
[1,1,3,1] -> [1,1,3,1] -> X
[1,1,2,1] -> [2,2,2,3]  -> [2,1] -> [-1,-1]
              [1,1,0,2]
```

```
In the example above:
At the end of day 0, the mountain cannot be crossed. The steps are too high to climb.
At the end of day 1, there are no changes, still no crossing.
At the end of day 2, the mountain can be crossed by climbing once. Notice that in position 2

In case it's not possible to cross on any of the days, the function should return Null or [-

Expected results:

best_day_to_cross(altitudes_1, snow_1) -> [2, 1] at the end of day two, only one climb is re
best_day_to_cross(altitudes_2, snow_2) -> [0, 0] day zero is the best day to cross
best_day_to_cross(altitudes_3, snow_3) -> [2, 0] zero climbs are required at the end of day
best_day_to_cross(altitudes_4, snow_4) -> [-1,-1] no viable days, the steps are always too h
best_day_to_cross(altitudes_5, snow_5) -> [5, 1] melting can continue over a few days
best_day_to_cross(altitudes_6, snow_6) -> [0, 4] it requires 4 climbs

Complexity variables:

A - number of altitude positions
D - number of days in the forecast

*/
```

## Thrilling Teleporters

### Part 1

```
/*
We are developing a new board game, Thrilling Teleporters.
The board consists of consecutive squares from 0 to last_square, some of the spaces also con
The game is played as follows:
  1. Each turn, the player rolls a die numbered from 1 to die_sides.
  2. The player moves forward the rolled number of squares.
  3. The player stops at last_square if they reach it.
  4. If the player finishes on a square with a teleporter, they are moved to where the telep
Note: Only one teleporter is followed per turn.

A sample board with last_square 20 the following teleporters might look like this conceptual
teleporters1 = ["3,1", "4,2", "5,10"]
die_sides = 6
   +-----+
   v     |
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
      ^     |  |              ^
      +-----+  +--------------+
order -> 1, 2, 1, 2,10, 6      -> 1,2,10,6

Write a function that returns a collection of the possible squares a player can move to in a
- A collection of teleporter strings
- The number of sides on the die
- The square the player starts on
- The last square on the board
```

```
Example:
With a 6-sided die, starting at square 0 with a board ending at square 20 (as pictured above
Rolling a 1, 2 or 6 have no teleporters so they end at that square.
Rolling a 3 goes to 1, rolling a 4 goes to 2, rolling a 5 goes to 10.
The possible outcomes are, in order of die roll, are [1, 2, 1, 2, 10, 6].
If we remove the duplicates, the answer is [1, 2, 10, 6]
destinations(teleporters1, 6, 0, 20) => [1, 2, 10, 6]
Additional Inputs:
teleporters2 = ["5,10", "6,22", "39,40", "40,49", "47,29"]
teleporters3 = ["6,18", "36,26", "41,21", "49,55", "54,52",
                "71,58", "74,77", "78,76", "80,73", "92,85"]
teleporters4 = ["97,93", "99,81", "36,33", "92,59", "17,3",
                "82,75", "4,1", "84,79", "54,4", "88,53",
                "91,37", "60,57", "61,7", "62,51", "31,19"]
teleporters5 = ["3,8", "8,9", "9,3"]

Complexity Variable:
B = size of the board
Note: The number of teleporters, T, and the size of the die, D, are bounded by B.

All Test Cases:
(output can be in any order)
                           die  start  last
                          sides, pos, square
destinations(teleporters1,  6,    0,    20)  => [1, 2, 10, 6]
destinations(teleporters2,  6,   46,   100)  => [48, 49, 50, 51, 52, 29]
destinations(teleporters2, 10,    0,    50)  => [1, 2, 3, 4, 7, 8, 9, 10, 22]
destinations(teleporters3, 10,   95,   100)  => [96, 97, 98, 99, 100]
destinations(teleporters3, 10,   70,   100)  => [72, 73, 75, 76, 77, 79, 58]
destinations(teleporters4,  6,    0,   100)  => [1, 2, 3, 5, 6]
destinations(teleporters5,  6,    0,    20)  => [1, 2, 4, 5, 6, 8]
*/
```

## Part 2

```
Thrilling Teleporters allows players to randomize the teleporters each game. However, during

You'll be given the following inputs:
- A collection of teleporter strings
- The number of sides on the die
- The square the player starts on
- The last square on the board

Write a function that returns whether or not it is possible to get to the last square from t

Examples:
teleporters1 = ["10,8", "11,5", "12,7", "13,9"]
               +-------------------+
               |        +------+   |
               v        v      |   |
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
                     ^     ^           |   |
                     +-----|-----------+   |
                             +---------------+

With a 4 sided die, starting at square 0 with a board ending at square 20 (as pictured above
No matter what you roll, it's not possible to get past the teleporters from 10-13.
finishable(teleporters1, 4, 0, 20) => False

If an additional teleporter was added from square 2 to square 15, this would be possible to
teleporters2 = ["10,8", "11,5", "12,7", "13,9", "2,15"]
finishable(teleporters2, 4, 0, 20) => True

But if we started on square 9, then it is still impossible as square 2 cannot be reached.
finishable(teleporters2, 4, 9, 20) => False

Additional Input:
teleporters3 = ["10,8", "11,5", "12,1", "13,9", "2,15"]
teleporters4 = ["2,4", "9,8", "11,7", "12,6", "18,14",
                "19,16", "20,9", "21,14", "22,6", "23,26",
                "25,10", "28,19", "29,27", "31,29", "38,33",
                "39,17", "41,30", "42,28", "45,44", "46,36"]
teleporters5 = ["4,21", "11,18", "13,17", "16,17", "18,21",
                "22,11", "26,25", "27,9", "31,38", "32,43",
                "34,19", "35,19", "36,39", "38,25", "41,31"]

Complexity variable:
B = size of the board
Note: The number of teleporters and size of the die are bounded by B.

All Test Cases:
                        die, start, end
finishable(teleporters1, 4,   0,    20)  => False (Above)
finishable(teleporters2, 4,   0,    20)  => True  (Above)
finishable(teleporters2, 4,   9,    20)  => False (Above)
finishable(teleporters3, 4,   9,    20)  => True
finishable(teleporters4, 4,   0,    50)  => False
finishable(teleporters4, 6,   0,    50)  => True
finishable(teleporters5, 4,   0,    50)  => True
finishable(teleporters5, 2,   0,    50)  => False
```

## Tomb Raider

```
While exploring an ancient tomb for treasure, you've come across a room with magical spheres

For example, let's look at the following room layout:
room1 = ..b.r..r.R.B...b
Capital letters represent spheres.
Lower-case letters represent holes.
Periods represent empty spaces without holes or spheres.
In this case, the red sphere at position 9 can be taken to either 4 or 7, and the blue spher
Move R sphere 2 spaces left to r +---+
v| ..b.r..r.R.B...b |^
+-------+
Move B sphere 4 spaces right to b
To minimize the time holding the spheres:
```

```
Carry the red sphere from 9 to 7, a distance of 2
Carry the blue sphere from 11 to 15, a distance of 4
Total Distance: 6

Write a function that, given a room layout, determines the minimum
total distance carrying spheres required to put all the spheres in
appropriate holes.
Note: There will only be one sphere of each color, but there may be
multiple places to put each sphere. You may only hold one sphere at
once, and only distance carrying a sphere is counted.
Additional Input:
room2 = "RBGYygbr"
room3 = ".........."
room4 = "abcbabbcbaAabcabcabbBabbabcabcbbC"
room5 = "rRBGYygbr"
Complexity Variable:
N = size of the room
All Test Cases:
distance(room2) => 16
distance(room3) => 0 There are no spheres, so no distance is required. distance(room4) => 5
distance(room5) => 10
```

## Treasure Room

### Question 1

```
You are with your friends in a castle, where there are multiple rooms named after flowers. S

Each room contains a single instruction that tells the player which room to go to next.

*** instructions_1 ***

lily -------- daisy sunflower

| | |

v v v

jasmin -> tulip violet -> rose --->

^ | ^ ^ |

| | | | |

----- iris -----

*** instructions_2 ***

lily --------

|

v
```

```
jasmin -> tulip -- > violet

Write a function that takes two parameters as input:

* a list containing the treasure rooms, and

* a list of instructions represented as pairs of (source_room, destination_room)

and returns a collection of all the rooms that satisfy the following two conditions:

* at least two *other* rooms have instructions pointing to this room

* this room's instruction immediately leads to a treasure room

Examples

instructions_1 = [

["jasmin", "tulip"],

["lily", "tulip"],

["tulip", "tulip"],

["rose", "rose"],

["violet", "rose"],

["sunflower", "violet"],

["daisy", "violet"],

["iris", "violet"]

]

treasure_rooms_1 = ["lily", "tulip", "violet", "rose"]

treasure_rooms_2 = ["lily", "jasmin", "violet"]

instructions_2 = [

["jasmin", "tulip"],

["lily", "tulip"],

["tulip", "violet"],

["violet", "violet"]

]

treasure_rooms_3 = ["violet"]

filter_rooms(treasure_rooms_1, instructions_1) => ["tulip", "violet"]
```

```
* tulip can be accessed from rooms lily and jasmin. Tulip's instruction leads to a treasure

* violet can be accessed from daisy, sunflower and iris. Violet's instruction leads to a tre

filter_rooms(treasure_rooms_2, instructions_1) => []

* none of the rooms reachable from tulip or violet are treasure rooms

filter_rooms(treasure_rooms_3, instructions_2) => [tulip]

* tulip can be accessed from rooms lily and jasmin. Tulip's instruction leads to a treasure

All the test cases:

filter_rooms(treasure_rooms_1, instructions_1) => ["tulip", "violet"]

filter_rooms(treasure_rooms_2, instructions_1) => []

filter_rooms(treasure_rooms_3, instructions_2) => [tulip]

Complexity Analysis variables:

N: treasure rooms

P: instructions
```

## Question 2

We are playing a game where the player needs to follow instructions to find a treasure.

There are multiple rooms, aligned in a straight line, labeled sequentially from 0. Each room contains one instruction, given as a positive integer.

An instruction directs the player to move forward a specific number of rooms. The last instruction is "9" by convention, and can be ignored (there's no room to move after the last room).

The player starts the game in room number 0 and has to reach the treasure which is in the last room. The player is given an amount of money to start the game with. She must use this money wisely to get to the treasure as fast as possible.

The player can follow the instruction or pay 1, the instruction "2" may be changed to "1" or "3". A player cannot pay more than $1 to change the value of an instruction by more than one unit.

Write a function that takes a list of instructions and a total amount of money as input and returns the minimum number of instructions needed to reach the treasure room, or None/null/-1 if the treasure cannot be reached.

Examples
Note: The updated instructions are marked with *.

```
Example 1

instructions_2_1 =  [1, 1, 1, 9]

With $0, the player would follow 3 instructions:
Instructions:   [  1,  1,  1,  9]
Itinerary:      [  1,  1,  1,  9]
```

## Writing Application

### Part 1

We are building a word processor and we would like to implement a "word-wrap" functionality.

Given a list of words followed by a maximum number of characters in a line, return a collection of strings where each string element represents a line that contains as many words as possible, with the words in each line being concatenated with a single '-' (representing a space, but easier to see for testing). The length of each string must not exceed the maximum character length per line.

Your function should take in the maximum characters per line and return a data structure representing all lines in the indicated max length.

Examples:

```
words1 = [ "The", "day", "began", "as", "still", "as", "the",
          "night", "abruptly", "lighted", "with", "brilliant",
          "flame" ]

wrapLines(words1, 13) "wrap words1 to line length 13" =>

  [ "The-day-began",
    "as-still-as",
    "the-night",
    "abruptly",
    "lighted-with",
    "brilliant",
    "flame" ]

wrapLines(words1, 20) "wrap words1 to line length 20" =>

  [ "The-day-began-as",
    "still-as-the-night",
    "abruptly-lighted",
    "with-brilliant-flame" ]

words2 = [ "Hello" ]

wrapLines(words2, 5) "wrap words2 to line length 5" =>

  [ "Hello" ]
```

```
words3 = [ "Hello", "world" ]

wrapLines(words3, 5) "wrap words3 to line length 5" =>

  [ "Hello",
  "world" ]

words4 = ["Well", "Hello", "world" ]

wrapLines(words4, 5) "wrap words4 to line length 5" =>

  [ "Well",
  "Hello",
  "world" ]

words5 = ["Hello", "HelloWorld", "Hello", "Hello"]

wrapLines(words5, 20) "wrap words 5 to line length 20 =>

  [ "Hello-HelloWorld",
    "Hello-Hello" ]

All Test Cases:
          words,  max line length
wrapLines(words1, 13)
wrapLines(words1, 20)
wrapLines(words2, 5)
wrapLines(words3, 5)
wrapLines(words4, 5)
wrapLines(words5, 20)

n = number of words OR total characters
```

## Part 2

We are building a word processor and we would like to implement a "reflow" functionality that also applies full justification to the text.

Given an array containing lines of text and a new maximum width, re-flow the text to fit the new width. Each line should have the exact specified width. If any line is too short, insert '-' (as stand-ins for spaces) between words as equally as possible until it fits.

Note: we are using '-' instead of spaces between words to make testing and visual verification of the results easier.

Examples:

```
lines = [ "The day began as still as the",
          "night abruptly lighted with",
          "brilliant flame" ]

reflowAndJustify(lines, 24) "reflow lines and justify to length 24" =>
```

```
        [ "The--day--began-as-still",
          "as--the--night--abruptly",
          "lighted--with--brilliant",
          "flame" ] // <--- a single word on a line is not padded with spaces

reflowAndJustify(lines, 25) "reflow lines and justify to length 25" =>

        [ "The-day-began-as-still-as"
          "the-----night----abruptly"
          "lighted---with--brilliant"
          "flame" ]

reflowAndJustify(lines, 26) "reflow lines and justify to length 26" =>

        [ "The--day-began-as-still-as",
          "the-night-abruptly-lighted",
          "with----brilliant----flame" ]

reflowAndJustify(lines, 40) "reflow lines and justify to length 40" =>

        [ "The--day--began--as--still--as-the-night",
          "abruptly--lighted--with--brilliant-flame" ]

reflowAndJustify(lines, 14) "reflow lines and justify to length 14" =>

        ['The--day-began',
         'as---still--as',
         'the------night',
         'abruptly',
         'lighted---with',
         'brilliant',
         'flame']

All Test Cases:
                 lines, reflow width
reflowAndJustify(lines, 24)
reflowAndJustify(lines, 25)
reflowAndJustify(lines, 26)
reflowAndJustify(lines, 40)
reflowAndJustify(lines, 14)

n = number of words OR total characters
```