**Graphing Homework**

|  |  |  |
| --- | --- | --- |
| **Mastering Graphs** | | **Summary**  Using all that we’ve learned to solve graphing problems :D. |
| Intentionally left blank, for spacing.hehe | | |

# **Overview**

For this homework you will be provided with a repl.it template for you to copy and paste. The template contains the classes/methods/functions you will need to implement for each problem. Your main will import each class and you can run the test there :).

I will provide some sample data for testing, but it's on you to fully test your code!!!

Advice:

1. Create helper functions to help you test!!!!!
2. Be as performant as possible, think of how you can if you can optimize the brute force solution. The point of the class is to write performant solutions to problems.
3. Break your code into sub parts. Clearly define what each part is doing, this will help you write cleaner code, and debug better
4. Ask for help!!! Your Instructor’s and TA’s are here to help you, use them!!!

C++: <https://repl.it/@NicholasReid1/CppTemplate>

Java: <https://repl.it/@NicholasReid1/JavaTemplate>

Python: <https://repl.it/@NicholasReid1/PythonTemplate>

# **Problem 1. Undirected Graph’n**

Let's make a Graph Class for unweighted undirected graphs (UDGs). In this class vertices will be represented by strings, and edges will be stored as adjacency lists.

## **Problem:**

Implement the following for the class:

1. Constructor(List<List<string>> edgesList)
   1. Takes in a list of edges represented as pairs, and converts that into a Graph with adjacency lists
2. Method: addEdge(string v1, string v2) -> void
   1. Simply makes an edge between two vertices. Be careful!! What if a vertex does not exist! We shouldn't make an edge :)\
3. Method: numOfConnectedComponents() -> int
   1. Here's a picture for reference :). <https://screenshot.googleplex.com/kc8kusumVKY>
4. Method: hasCycle() -> bool
   1. Determine if there is a cycle in the UDG

# **Problem 2. Traversin’ Trees**

A ***Binary Tree*** is a rooted tree in which each node has no more than 2 children and every node has at most an in-degree of one. Let's extend this definition to an ***N-ary tree***. If a tree is a rooted tree in which each node has no more than N children, it is called ***N-ary tree***.

## **Problem:**

Given a N-ary tree, return the ***zigzag level order*** traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

### **Example:**

Input:

***3***

***/ \***

***9 20***

***/ \***

***15 7***

Output:

***[***

***[3],***

***[20,9],***

***[15,7]***

***]***

***Note: Do not reverse the array at the end to achieve the zig-zag result, do it in-place as you're traversing downward.***

# **Problem 3. Foreign Exchange**

The foreign exchange (also known as FX or forex) market is a global marketplace for exchanging national currencies against one another. Currencies trade against each other as exchange rate pairs. For example, EUR/USD.

## **Problem:**

Given a list of currency rates, return the conversion rate of one currency to another.

### **Example:**

exchangeRates = [ {“USD”,”EUR”,0.75}, {“EUR”, “GBP”, 0.1}, {“GBP”, “AUD”, 0.5}]

Input -> (exchangeRates, “USD”, “AUD”) Input -> (exchangeRates, “AUD”, “USD”)

Output -> 0.0375 Output -> 26.67

# **Problem 4. Word Ladder**

**Problem:**

Given two words (beginWord and endWord), and a dictionary's word list, return the shortest transformation sequence from beginWord to endWord, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list. Note that beginWord is not a transformed word.

**Assumptions:**

* Return an empty array if there is no such transformation sequence.
* All words have the same length.
* All words contain only lowercase alphabetic characters.
* You may assume no duplicates in the word list.
* You may assume beginWord and endWord are non-empty and are not the same.

### **Example 1:**

Input:

beginWord = "hit",

endWord = "cog",

wordSet = {"hot","dot","dog","lot","log","cog"}

Output: ["hit", "hot", "dot", "dog", "cog"] or ["hit", "hot",”lot”,”log”,”cog”]

### **Example 2:**

Input:

beginWord = "hit"

endWord = "cog"

wordSet = {"hot","dot","dog","lot","log"}

Output: [] (cog is not in the wordSet, no way to get to it)

# **Problem 5. Longest Increasing Path in a Matrix**

**Problem:**

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may ***NOT*** move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

### **Example 1:**

Input: nums =

[

[9,9,4],

[6,6,8],

[2,1,1]

]

Output: 4

### **Example 2:**

Input: nums =

[

[3,4,5],

[3,2,6],

[2,2,1]

]

Output: 4

# **Problem 6. TechX**

Congratulations!! You’re starting your first day at the top school in the world TechX. At TechX, each student is given a total list of classes they must take for the year. For an input of N, we define the total list of classes as 0 -> n-1.

Just like any university, classes at TechX may have prerequisites. For example, to take course 1 you have to first take course 0, which is expressed as a pair: [1,0]

**Problem:**

Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

### **Example 1:**

Input: 2, [[1,0]]

Output: [0,1]

Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1].

### **Example 2:**

Input: 4, [[1,0],[2,0],[3,1],[3,2]]

Output: [0,1,2,3] or [0,2,1,3]

Explanation: There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0.

So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3].