GROUP 11 PROJECT PROPOSAL

TOMMENIX YU, REX XU, SANSEN WEI

ABSTRACT. We plan to do some basic algorithms with graph theory that may include the coloring problem, finding the max/min flows, and some other topics in combinatorial optimization.

CONTENTS

1. BACKGROUND

There are a lot of reasons for one to do graph theory. One might be purely mathematical interest: the famous four color problem. In terms of graph theory, it states that for loopless planar graph G, its chromatic number is $\chi(G) \leq 4$, which means that given any separation of a map into contiguous countries, we can color the countries with 4 colors in a way that no 2 adjacent countries are colored the same. Or we might be keen on practical optimization problems such as which is the best route (minimize distance on road) to choose if we are traveling across Europe. The basic of all these is a well grounded implementation of the graph object.

2. PLAN FOR IMPLEMENTATION

We will first implement the graph object that contains basic features such as the vertices and the edges. We will then add more feature to the graph such as

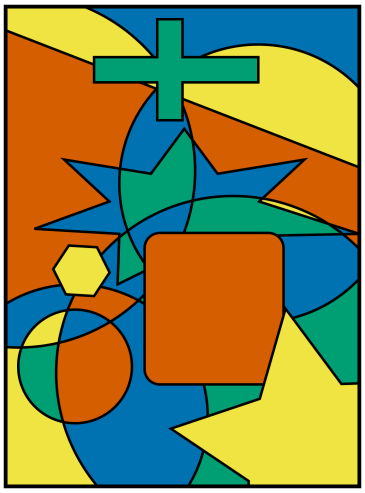
- multiple edges between two vertices;

- maximum flow for each edge;
- surfaces formed by the graph;
- connected vertices;
- ⋮

With these children objects, we can then implement some algorithms such as the minimal coloring with n colors, find the maximum flow of a graph with one start point and one end-point, or even find connectedness of two student in the MCAM program judging by the class they are taking.

2.1. **coloring problem.**

The well-known four color problem is hard to prove, but it is easy to implement and test on particular given graphs, as long as they are not really massive and pathological. We may start from larger n (maybe 10?) for more complicated graphs. For instance look at the following graph:



Maybe the first thing to do is to convert the actual graph into a graph object with only vertices and edges. Since there's a duality of the isolated areas and the vertices in a graph, we can also implement the flip method to find the converted graph we need.

2.2. **Possible implementations.**

After some discussion, we've came up with some directions to go. I list them from the most doable to the least, in our perspective.

(1) The first and probably the most important thing to do is to implement a method that returns a possible coloring with n colors, for at least suitable n.
Then, we'd like to try for a graph the least color it took to fill it.

(2) As we are discussing the problem, we realized that even though all 2d map have a corresponding graph, there's no guarantee that every graph has a 2d correspondence. Otherwise for only 5 countries we'd counter-prove the four color theorem.

So our next goal is to learn how to distinguish 2d maps with other graphs.

One of our first thought is to use the well known Euler's formula $V - E + F = 2$. However, we soon realized that it is neither sufficient (we can create a lot of graphs with the same set of (V, E, F)) nor necessary (islands). But we'll probably make some progress on this.

(3) This leads to another problem: are there anyway to create some equivalent class in the set of all graphs of n nodes. What I mean is that, for instance, the same structure with nodes rotated can be colored in the same method, so we'd not count them multiple times.

(4) If, by any chance, there's a good way of reduction of cases that reduces the possible graphs from $O(e^{n^2})$ for n nodes to a much smaller order, we might be able to plot the least colors needed for all graphs of n nodes and observe its distribution. This, though, seems extremely hard.

(5) Eventually, after we finalize an algorithm for the coloring problem, we might be interested in its runtime scaling with respect to the number of verticies, and induced graph topologies (from given 2D maps). A scalable algorithm is more favorable.

3. Tests to do

We will start testing with some very basic graphs that we can easily compute manually. For instance, we can check with bare eyes whether the coloring of a graph satisfies the condition we want. Then we can try with larger data sets and write up a test method to check whether the program works.

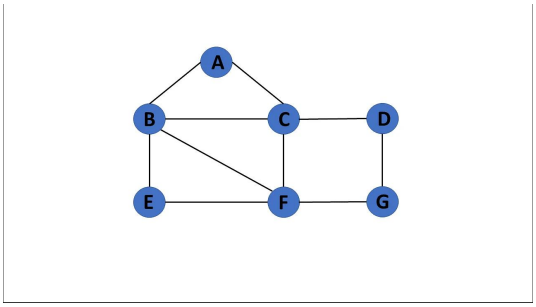
For practice, we can use the world map and see if the least color is at most 4. This sounds like a pretty good test to do in terms of practice. Further this line, we can use our characterization of 2d-graph to generate graphs and check whether the least color is at most 4.

4. Some basic codes for simple graphs, vertices and edges

We start to define the a group as an object.

We added the object graph, and implemented some methods such as find vertices, find edges, add edges, show edges, etc. The codes are in the graph.ipynb file.

For instance, we used this starting graph:



And we want to put it into an object, by defining each vertices by all the adjacent vertices it connects with edges, for which we used the dictionary:

```
graph2 = {   "a" : {"b", "c"},
             "b" : {"a", "c", "e", "f"},
             "c" : {"a", "b", "d", "f"},
             "d" : {"c", "g"},
             "e" : {"b", "f"},
             "f" : {"b", "c", "e", "g"},
             "g" : {"d", "f"}
         }

A = Graph(graph2)
```

Then, we use a test "show gd" function that tells us what is the dictionary behind the object:

```
A.show_gd()
```

and the result is:

```
{'a': {'b', 'c'},
 'b': {'a', 'c', 'e', 'f'},
 'c': {'a', 'b', 'd', 'f'},
 'd': {'c', 'g'},
 'e': {'b', 'f'},
 'f': {'b', 'c', 'e', 'g'},
 'g': {'d', 'f'}}
```

So it is fine. We tested all the methods and they function just right. So, after a few trials (all in the jupyter notebook file) we added a vertex "h" with no edges, and added edges $(1, 2), (2, 3), (3, 4)$ and the resulting dictionary is:

```
{'a': {'b', 'c'},
 'b': {'a', 'c', 'e', 'f'},
 'c': {'a', 'b', 'd', 'f'},
 'd': {'c', 'g'},
```

```
'e': {'b', 'f'},
'f': {'b', 'c', 'e', 'g'},
'g': {'d', 'f'},
'h': set(),
'2': {'1', '3'},
'1': {'2'},
'3': {'2', '4'},
'4': {'3'}}
```

Where the value of "h", which is set(), is correct since it is an empty set. If we .add something into it it will represent as others do.

APPENDIX A. REFERENCE

For basic graph objects:

https://python-course.eu/applications-python/graphs-python.php

For using networkx to visualize a graph:

https://www.geeksforgeeks.org/visualize-graphs-in-python/