CS5110: THE FINAL PROJECT.


```
<p> Daniel Murrow \& Hermès Henry </p>
```

TABLE OF CONTENTS.

01

Greedy Algorithm:

Prim's for finding an MST.

02

Divide-and-Conquer:

Girvan-Newman for community detection.

03

Dynamic Programming:

Bellman-Ford to compute shortest path.

04

Approx. Algorithms:

Maximum Clique problem in undirected graphs.

05

A demo of our GUI.

06

Thanks \& Q+A.

01 GREEDY ALGORITHM: PRIM'S ALGORITHM

 An algorithm for finding the Minimum Spanning Tree (MST) of a graph, assuming a few traits about the graph.

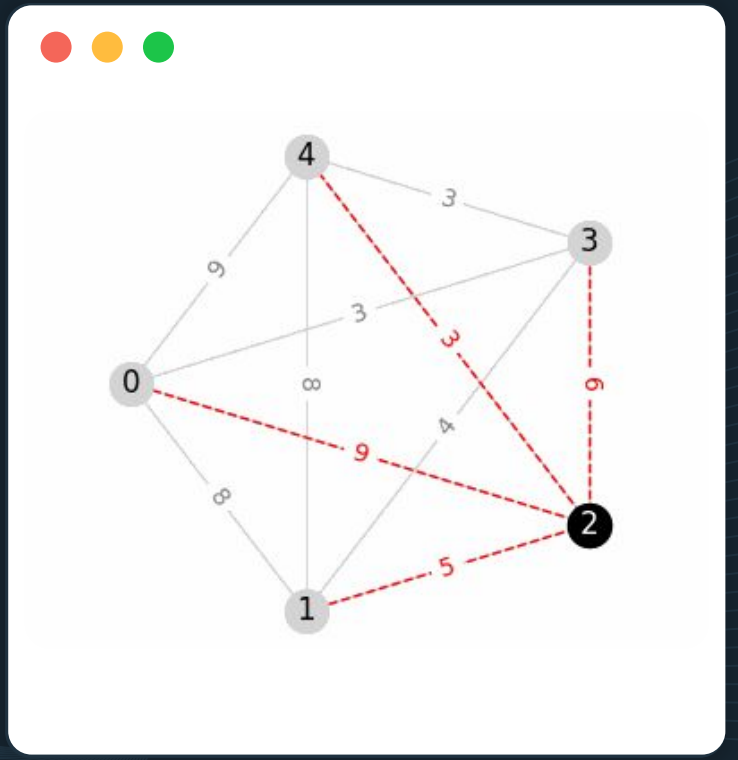
QUICK FACTS ABOUT PRIM's.

INPUT

A weighted, undirected, and connected graph.

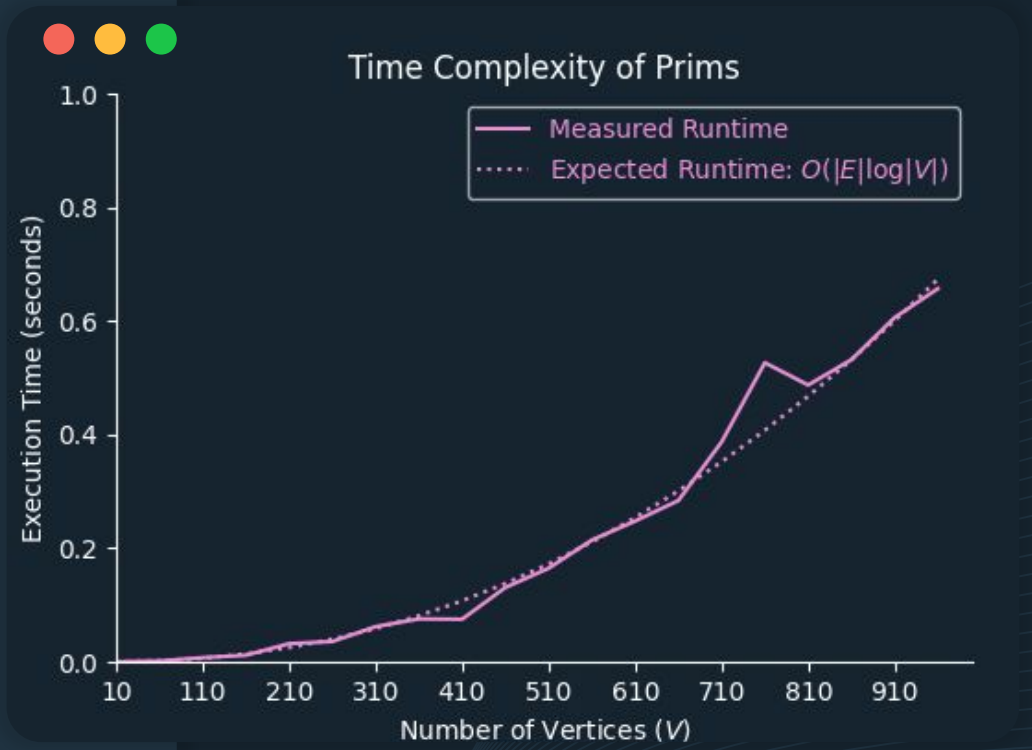
OUTPUT

An MST with *V* vertices and *V - 1* edges.



TIME COMPLEXITY: $O(|E| \log |V|)$.

Our implementations definitely has room for improvement. As discussed in our write-up, the theoretically fastest runtime would be $O(|E| + |V| \log |V|)$ where $|E|$ represents number of edges and $|V|$ number of vertices.



THE PRIORITY QUEUE.

Our implementation of Prim's uses a Binary minheap Priority Queue. (Python built-in)

Using a Fibonacci Priority Queue would increase our time complexity by speeding up each push operation.

	pop	push
Binary	$\Theta(\log n)$	$\Theta(\log n)$
Fibonacci	$\Theta(\log n)$	$\Theta(1)$

Using this alternate data structure alongside with keeping track of Vertices instead of Edges could further improve our runtime to the theoretical limit of $O(|E| + |V| \log |V|)$

02 DIVIDE-AND-CONQUER: GIRVAN-NEWMAN

 An algorithm for detecting at least two distinct communities or clusters within a given graph.

THREE STEPS FOR GIRVAN-NEWMAN.

Calculate the betweenness of every edge in the graph.

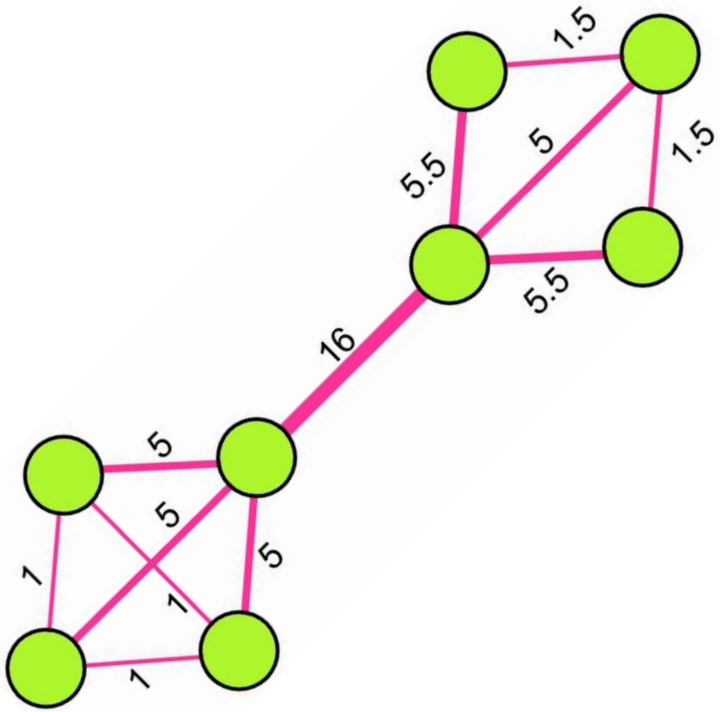
Identify the edge with the highest betweenness and remove it from the graph.

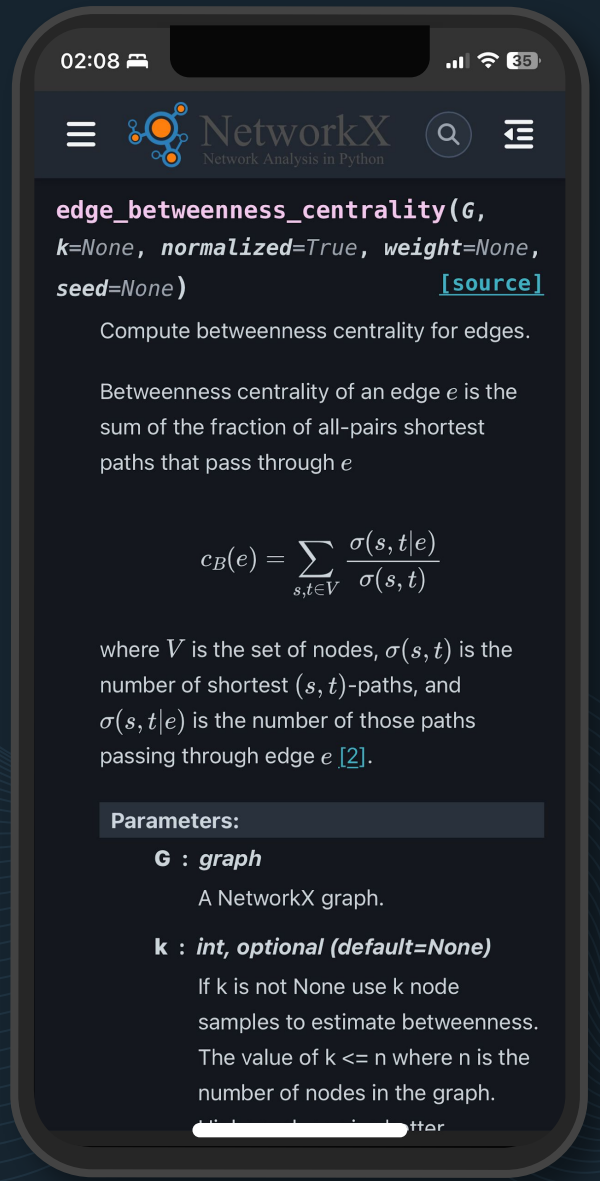
Repeat this process until two communities (or clusters) are disconnected.

SO WHAT IS BETWEENNESS?

Well, it's exactly what you think. A measure of how *between* an edge is relative to every single node in the graph.

Calculate the shortest path from every node to every other node using BFS or Dijkstra's. The edge that appears in the most shortest paths is the most between.





BETWEENNESS?

THEY HAVE A FUNCTION FOR THAT!

Throughout this project, we found ourselves thinking, *"Man, it would be so much easier if that was already a thing!"* and NetworkX almost always delivered.

LET'S TALK ABOUT TIME COMPLEXITY.

$O(|V||E|)$

Brande's algorithm for calculating edge betweenness.

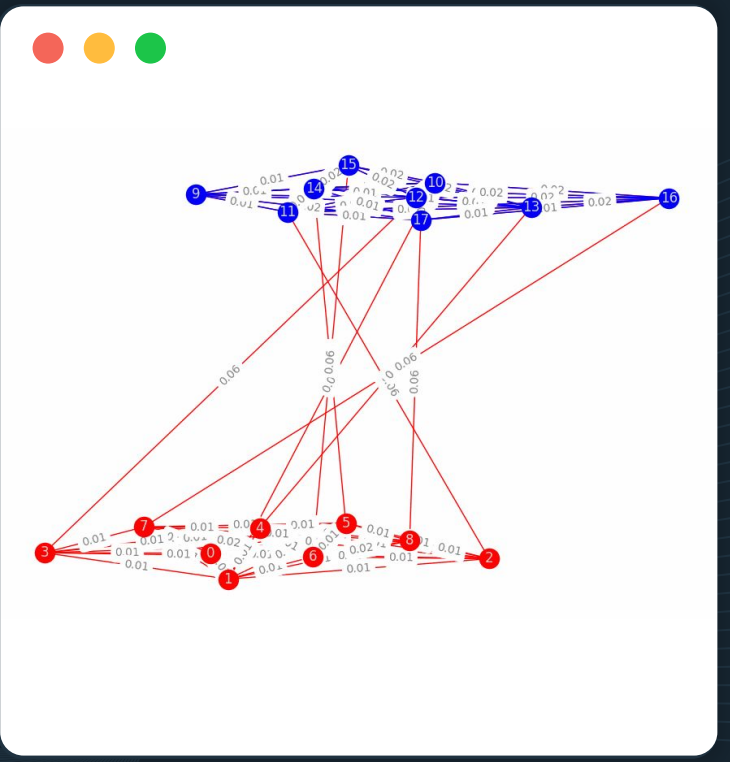
Repeat E times.

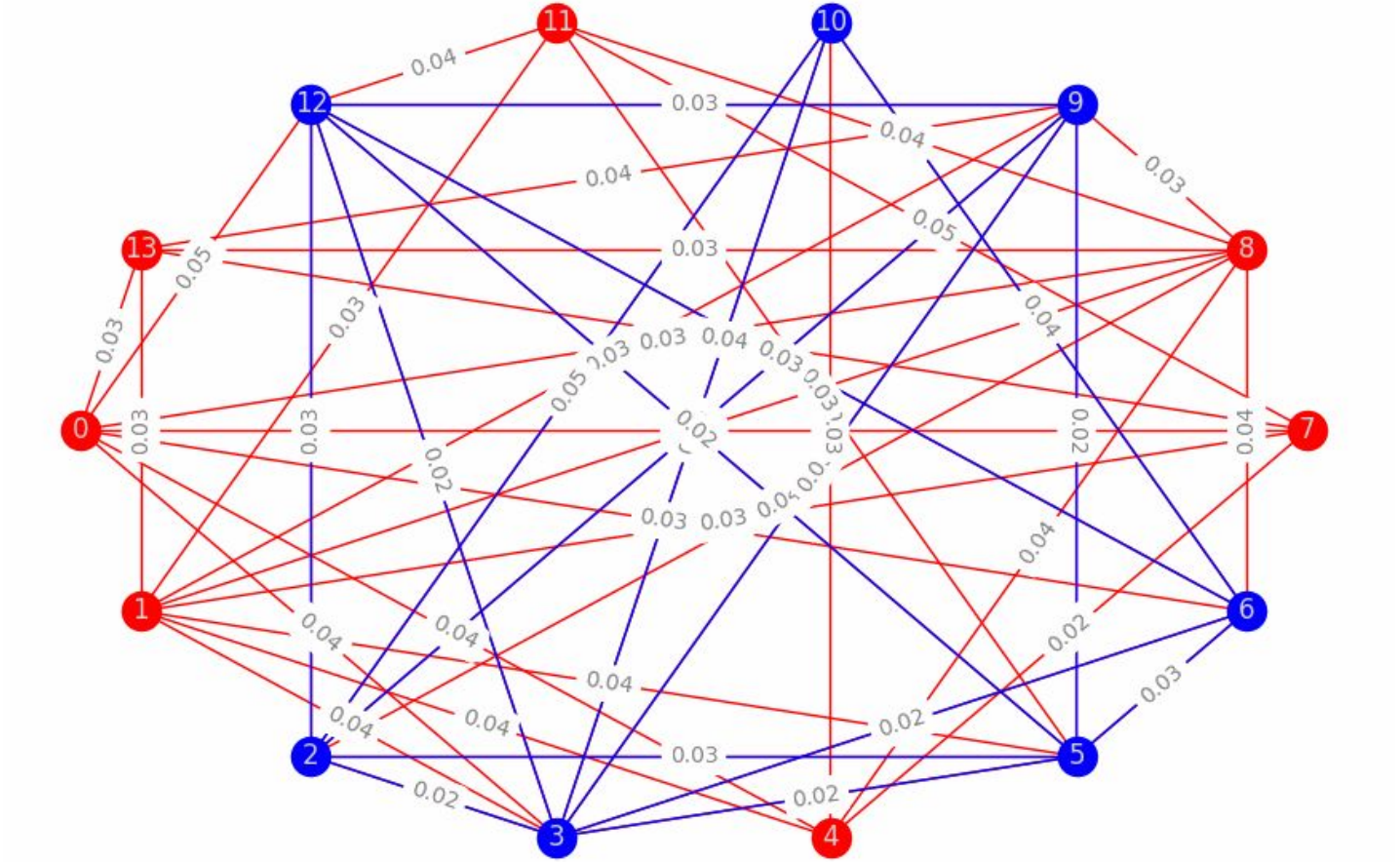
Put it all together!

$O(|E|(|V||E|))$

aka

O(|V||E|^2)





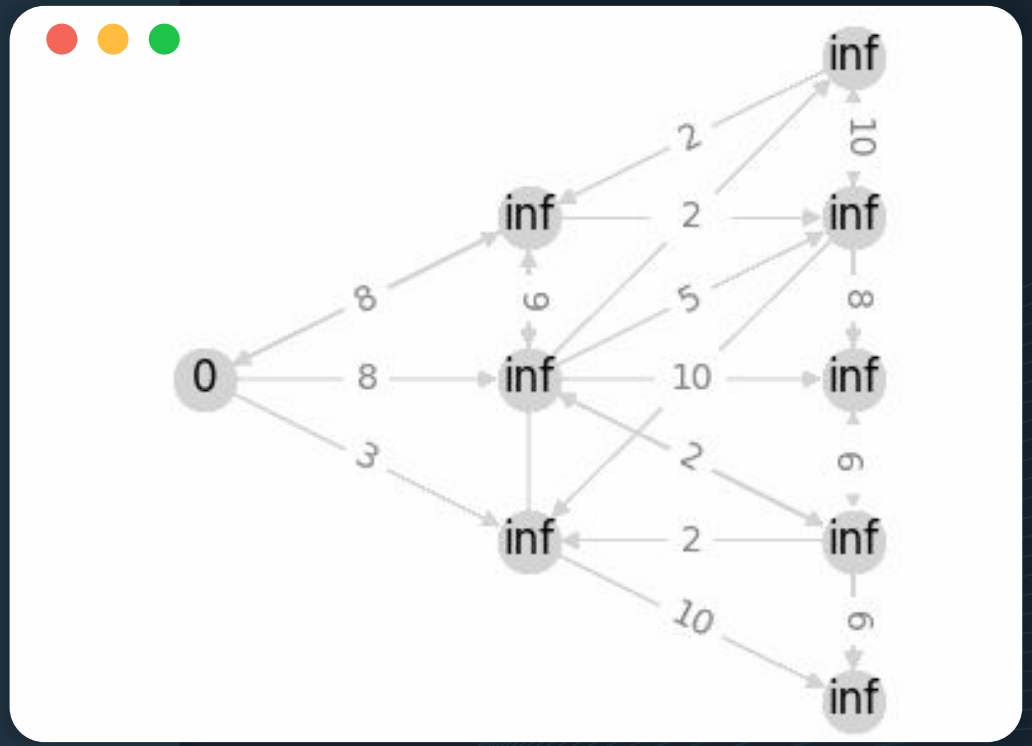
03 DYNAMIC PROGRAMMING: BELLMAN-FORD

 An algorithm used to compute shortest paths in a directed graph with negative edge weights (but no negative cycles)

THIS ONE COMES WITH AN *ASTERISK**

- Directed Graph
- Negative Edge Weights
  - (But no negative cycles)

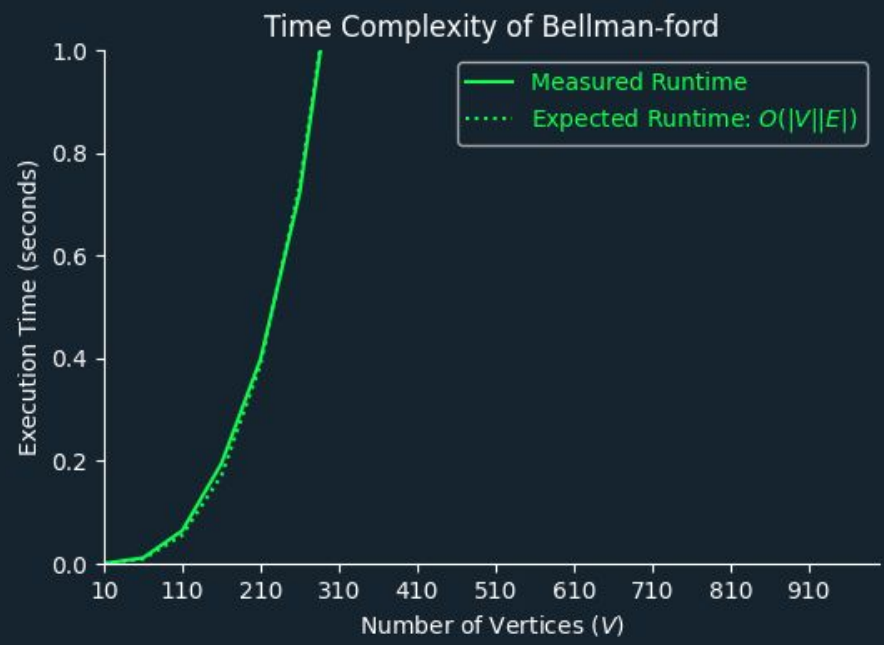
That being said, we can use it to detect negative cycles and, with a simple modification, the exact nodes in the cycle!



NOW FOR TIME COMPLEXITY AGAIN.

$O(|V||E|)$

Initialize at $O(|V|)$
and iterate through
every edge
$|V| - 1$ times.



AND HOW TO TEST IT?

Two modes:

- Normal mode where it simply finds the single-source shortest path of a directed graph
- Negative cycle finding mode

Normal is easy!

Just test it on some known negative edge weights graphs.

Negative cycles, not so much...

To oversimplify it, here's three questions:

1. Can the algorithm detect negative cycles when the source is part of it?
2. Can the algorithm detect negative cycles that don't include the source?
3. Will the algorithm complete successfully when all negative cycles are unreachable from the source?

04 APPROX. ALGORITHM: MAXIMUM CLIQUE

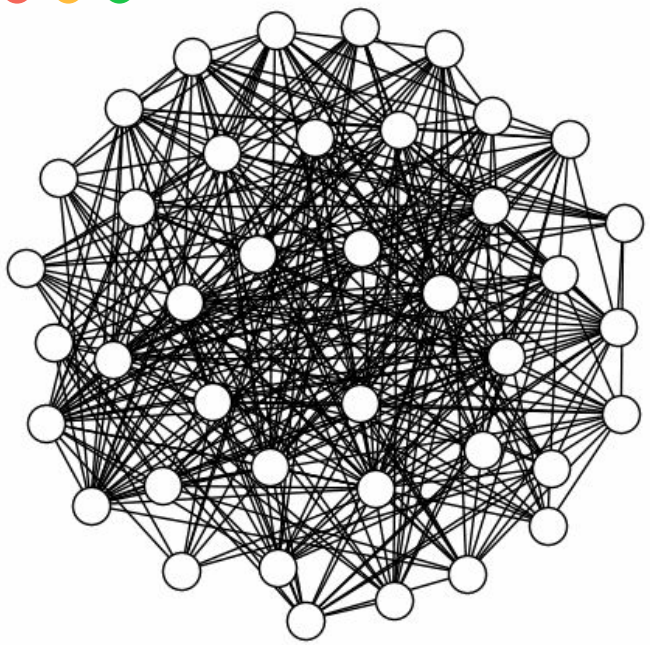
 An NP-Complete problem relevant to the fields of community detection, bioinformatics, and computational chemistry.

DOESN'T SOUND EASY.

(SPOILER: IT WASN'T)

The problem posed is to find the largest complete subgraph in a given undirected graph.

No polynomial-time algorithm has been found. But there exist many approximation algorithms. One of these is the Ramsey algorithm.



WHO'S THIS RAMSEY FELLA?

The Ramsey algorithm works by choosing a node *v* then recursing on the subgraphs induced by the neighbors and non-neighbors of *v*, respectively. Then the clique and independent sets are built by choosing the largest of the sets found so far in that recursion.

HOW DOES HE COMPARE?

	10 Nodes 2 Clique	100 Nodes 20 Clique	1000 Nodes 200 Clique
Ramsey	5.8s	23.8s	276.4s
NetworkX	5.7s	16.2s	127.5s

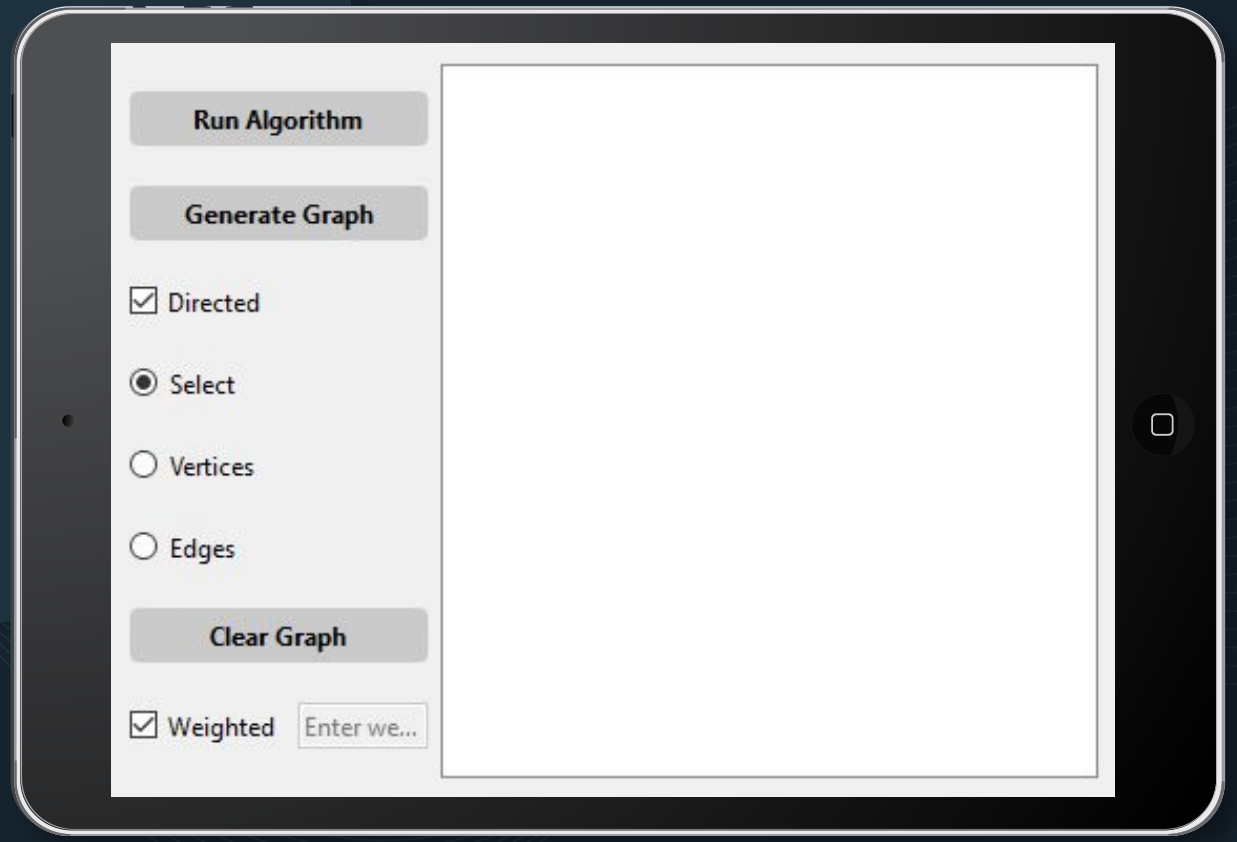
The choice of *v* can alter the result. For example, if a node is chosen that is adjacent to half the nodes in the largest clique in the graph, then the other half will be non-neighbors and the true maximum clique will never be found.

05 A DEMO OF OUR GUI

 Add a brief introduction of your section here: Let’s dive in and get to know some interesting facts about animals!

FEATURES OF OUR GUI:

- Fully interactive
  - Place and move nodes and edges with custom weights
  - Toggle directedness
  - Run algorithms on your graph or clear the scene
  - Generate random graph with our custom random graph generation class (too many features to list)



THANK YOU!

Do you have any questions?

Daniel:
murrowds@appstate.edu

Hermès:
henryhe@appstate.edu

See our GitHub repository:
github.com/MidnightHermes/CS5110-Final-Project