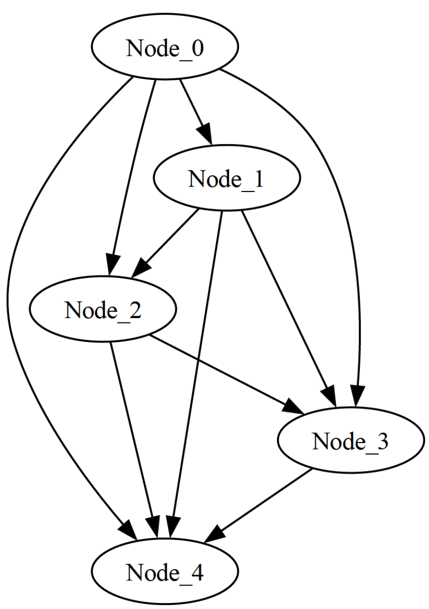
Interactive Visualization
Homework 3
Devanshu Haldar
Worked with Jaesok Kang For Part 2

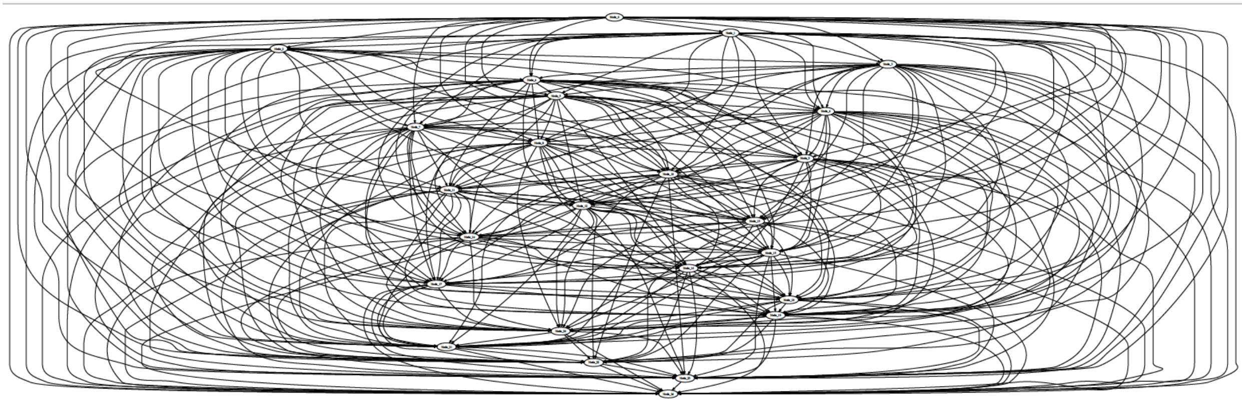
NOTE: Code for all parts is pasted at the end of the document.

Part 1:

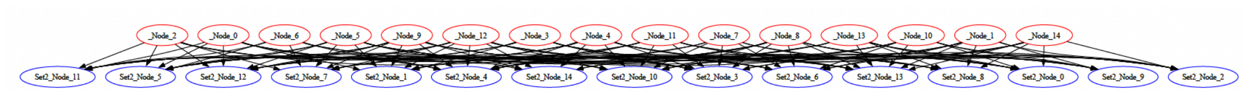
In part 1, I generated fake data of values and letters. I included randint to randomize the nodes and edge connections for some of the graphs.



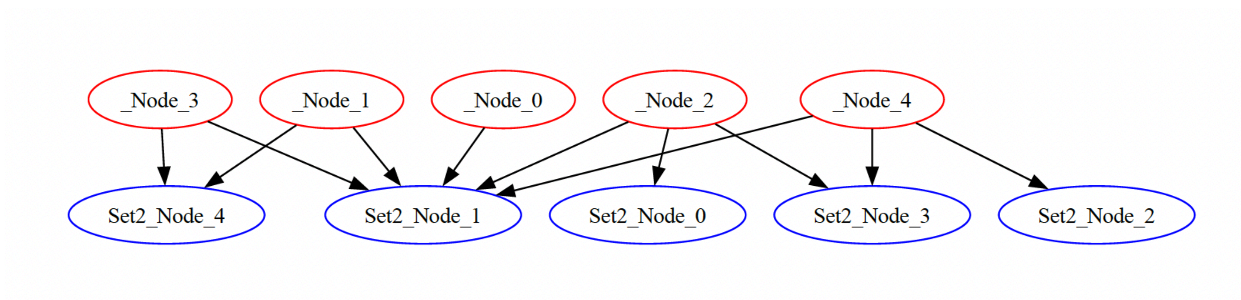
In this graph I wanted to see how GraphViz interacts with a clique of size 5. As we can see it is very simple to create and visualize a Clique of smaller size. However, below we can see that a clique generated by GraphViz struggles with higher node sizes. It is a mess and very difficult to interpret what node is connected to what. This is a clique graph of size 25. Restructuring of the data, in terms of simplification, will be required here.



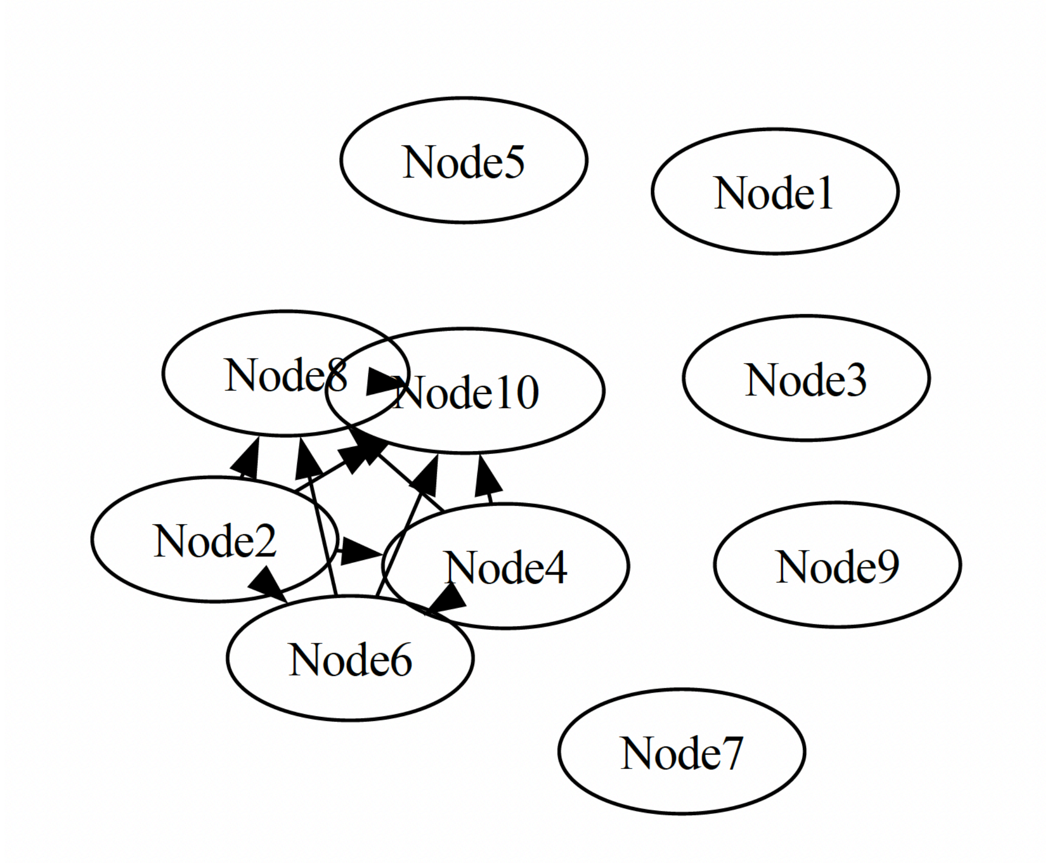
This is a similar experience I had when using GraphViz to create bipartite graphs with randomly generated nodes and edge connections. It was important to understand this comparison (how much a bipartite graph can handle in terms of number of nodes to still be interpretable). Below is a graph of high number of nodes (15 approximately on each side). This creates too many edges crossing one another. When I began to work on part 2.2 of this assignment I faced the same problem with course and instructor sets. I needed to dile down the size of my data for interpretability.



It is much stronger to represent your data (especially in a bipartite) as below:



On top of disconnected component graphs, I tried out a sparse graph using GraphViz. The result is shown below:



I personally believe this representation has high potential. Especially if we begin to think about relationships and networking. However, it is clear this can be improved. The clique on the left is too close to eachother and overlapping can be seen. If this was more spaced out with edges and nodes not overlapping, this would be much stronger of a visualization.

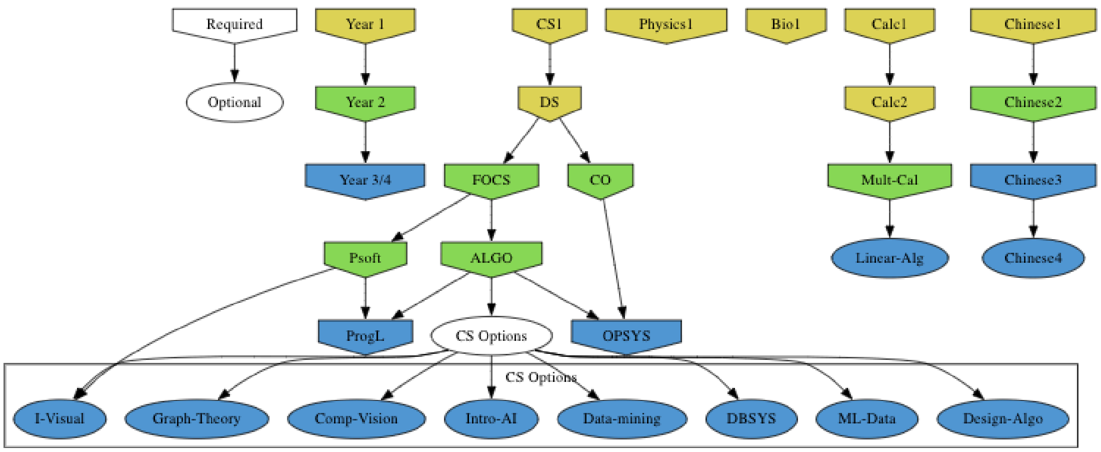
Part 2.1:

```
strict digraph {
   shape1 [label="Required"shape=invhouse]
   shape2 [label="Optional"]

   y1 [label="Year 1"shape=invhouse style="filled" fillcolor="#e3d952"]
   y2 [label="Year 2"shape=invhouse style="filled" fillcolor="#7de352"]
   y3 [label="Year 3/4"shape=invhouse style="filled" fillcolor="#42a3db"]
   ##e3d952
   shape1->shape2
```

```
  y1->y2
  y2->y3

Fall1 [label="CS1" shape=invhouse style="filled" fillcolor="#e3d952"]
Fall3 [label="Physics1"shape=invhouse style="filled" fillcolor="#e3d952"]
Fall2 [label="Bio1"shape=invhouse style="filled" fillcolor="#e3d952"]
Math1 [label="Calc1"shape=invhouse style="filled" fillcolor="#e3d952"]
Math2 [label="Calc2"shape=invhouse style="filled" fillcolor="#e3d952"]
Math3 [label="Mult-Cal"shape=invhouse style="filled" fillcolor="#7de352"]
Math4 [label="Linear-Alg" style="filled" fillcolor="#42a3db"]
HAS1 [label="Chinese1"shape=invhouse style="filled" fillcolor="#e3d952"]
HAS2 [label="Chinese2"shape=invhouse style="filled" fillcolor="#7de352"]
HAS3 [label="Chinese3"shape=invhouse style="filled" fillcolor="#42a3db"]
HAS4 [label="Chinese4" style="filled" fillcolor="#42a3db"]


Spring1 [label="DS"shape=invhouse style="filled" fillcolor="#e3d952"]


// Second Year
Fall5 [label="FOCS"shape=invhouse style="filled" fillcolor="#7de352"]
Fall6 [label="CO"shape=invhouse style="filled" fillcolor="#7de352"]


Spring5 [label="CS Options"]
Spring55 [label="ALGO"shape=invhouse style="filled" fillcolor="#7de352"]
Spring6 [label="Psoft"shape=invhouse style="filled" fillcolor="#7de352"]


// Third Year
Fall9 [label="OPSYS"shape=invhouse style="filled" fillcolor="#42a3db"]
Fall10 [label="ML-Data"  style="filled" fillcolor="#42a3db"]
Fall11 [label="DBSYS" style="filled" fillcolor="#42a3db"]
Fall12 [label="I-Visual" style="filled" fillcolor="#42a3db"]
Fall13 [label="Data-mining" style="filled" fillcolor="#42a3db"]
Fall14 [label="Intro-AI"  style="filled" fillcolor="#42a3db"]
Fall15 [label="Comp-Vision"  style="filled" fillcolor="#42a3db"]
Fall16 [label="Graph-Theory"  style="filled" fillcolor="#42a3db"]
Fall17 [label="Design-Algo"  style="filled" fillcolor="#42a3db"]
```

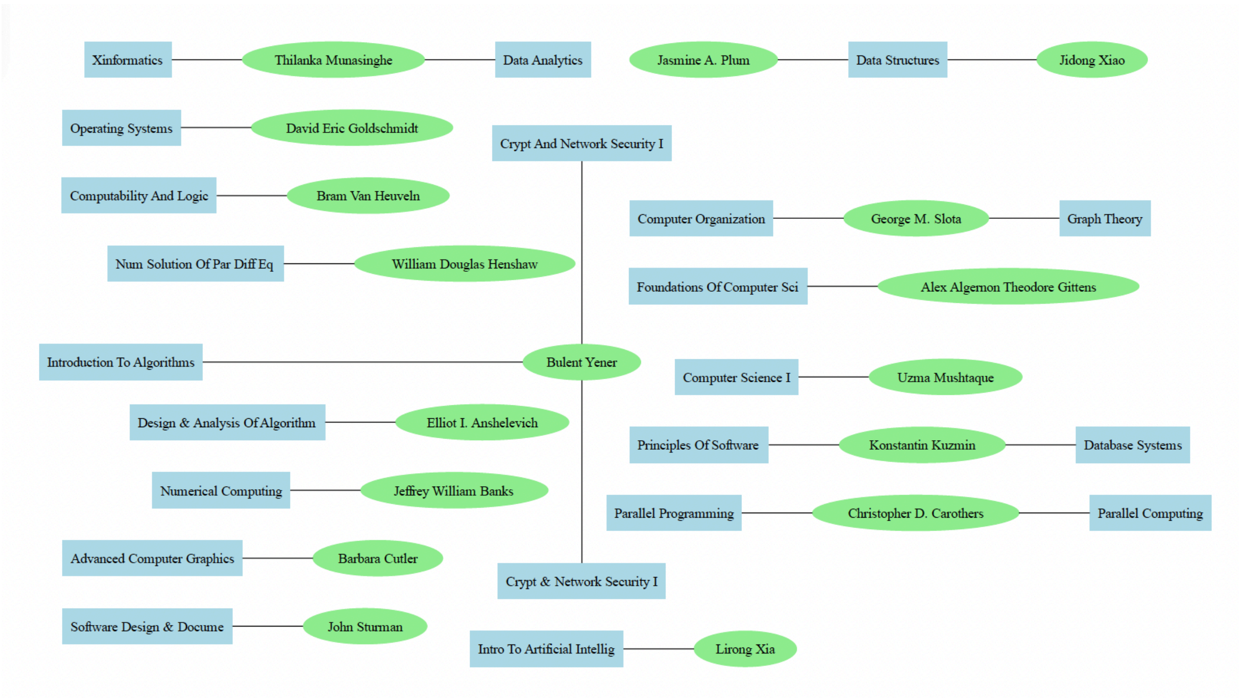
```
Spring9 [label="ProgL"shape=invhouse style="filled" fillcolor="#42a3db"]



Fall1->Spring1
Spring1->Fall5
Spring1->Fall6
Fall5->Spring55
Fall5->Spring6
Spring55->Spring9
Spring6->Spring9
Fall6->Fall9
Spring55->Fall9
Spring55->Spring5
Spring5->Fall10
Spring5->Fall11
Spring5->Fall12
Spring5->Fall13
Spring5->Fall14
Spring5->Fall15
Spring5->Fall16
Spring5->Fall17
Spring6->Fall12

Math1->Math2
Math2->Math3
Math3->Math4

HAS1->HAS2
HAS2->HAS3
HAS3->HAS4


subgraph cluster1 {
  Fall10;
  Fall11;
  Fall12;
  Fall13;
```

```
    Fall14;
    Fall15;
    Fall16;
    Fall17;
    label="CS Options";
}

}
```



Caption: In this visualization we implemented the use of coloring and shapes of nodes to represent a variety of things. First of which, depending on the shape, the node/class represents whether or not is required or optional. Furthermore, if the node is in color yellow it is a first year required course, in green it is a second year, and in blue it is a 3rd and 4th year. We decided to combine the 3rd and 4th years as most of the courses are optional and have similar if not the same requirements.

Part 2.2:



Caption: For this specific visualization, we first produced a bipartite graph that was strictly horizontal and had sets of nodes. This stretched the data very far wide and impacted the ease of perception. By expanding its verticality, you can see all the nodes spaced out in front of you (rather than having to scroll in the other version). The color coding and shapes of nodes also support the differentiation in both sets. Also it is important to note that the visualization works on a smaller set of nodes. Once we reached around 40-70 nodes, it became very difficult to interpret with edges overlapping and nodes on top of eachother. This was the issue as we did not remove courses with the second digit being 9. Once that was removed a lot of courses such as the Master's Thesis that connected all the professors helped space out the connections of the edges and nodes.

Part 3:

My experience with GraphViz, in general, has been mostly positive. My problem with this software is its capability to handle larger sets of data. If I want to incorporate more nodes into a graph: it struggles. It could introduce shrinking of nodes and edges to create space for more nodes and edges for example. This is not the case. However, for smaller sets of data it is pretty easy to use and presents visualizations that are easy to interpret. The tool has helped create moderately-complex graphs. For both parts in part 2, I believe we produced strong visualizations for what courses exist and some of the relations the courrses have with professors.As previously said, I believe that GraphViz struggles with larger sizes of nodes. It was also explained in part 1. However, if I was to do this by hand it would not have been better produced. Larger sets of data would take a massive amount of time for a person to draw by hand. Therefore a program is perfect. For small sets of data, GraphViz does what a person could do very well. To make this tool stronger, I would recommend incorporating more dynamic capabilities. For example, in a graph of many many nodes, we can introduce smaller nodes to make space for more nodes. And if the graph becomes large with lets say 500 nodes, we can have a zoom in option to zoom into cliques and parts of the visualization. I found the tool to be very easy to install. Since I mainly used the tool in cooperation with Python, it was easy to implement the library through Python and have the code up and running. I will always provide tools that are incorporated as libraries in Python a strong review.

CODE

Part 1:

```
import graphviz
import random

def create_fake_data():
   graph = graphviz.Digraph('FakeGraph')

   for i in range(2):
       node_name = f'Node_{i}'
       graph.node(node_name)

   for i in range(20):   #connnect random edges
       source_node = f'Node_{random.randint(0, 19)}'
       target_node = f'Node_{random.randint(0, 19)}'
       graph.edge(source_node, target_node)

   graph.view()



def create_bipartite_graph(num_nodes_, num_nodes_2, probability = .5):
```

```python
    graph = graphviz.Digraph('RandomBipartiteGraph')

    for i in range(num_nodes_):
        node_name = f'_Node_{i}'
        graph.node(node_name, color='red')

    for i in range(num_nodes_2):
        node_name = f'Set2_Node_{i}'
        graph.node(node_name, color='blue')

    for i in range(num_nodes_):
        for j in range(num_nodes_2):
            source_node = f'_Node_{i}'
            target_node = f'Set2_Node_{j}'
            if random.uniform(0, 1) < probability:
                graph.edge(source_node, target_node)

    graph.view()


def create_clique_graph(num_nodes):
    graph = graphviz.Digraph('CliqueGraph')

    for i in range(num_nodes):
        node_name = f'Node_{i}'
        graph.node(node_name)

    for i in range(num_nodes):
        for j in range(i + 1, num_nodes):
            source_node = f'Node_{i}'
            target_node = f'Node_{j}'
            graph.edge(source_node, target_node)

    graph.view()


def create_disconnected_components_graph():
    graph = graphviz.Digraph('DisconnectedComponentsGraph')

    for i in range(1, 6):
        graph.node(f'_Node{i}', color='red')
        if i < 5:
```

```python
            graph.edge(f'_Node{i}', f'_Node{i+1}')

    for i in range(1, 4):
        graph.node(f'_Node{i}', color='blue')
        if i < 3:
            graph.edge(f'_Node{i}', f'_Node{i+1}')

    graph.view()



def create_sparse_graph():
    graph = graphviz.Digraph('SparseGraph', engine='neato')

    graph.attr(nodesep='1', ranksep='1')

    for i in range(1, 11):
        graph.node(f'Node{i}')

    for i in range(1, 11):
        for j in range(i + 1, 11):
            if i != j and i % 2 == 0 and j % 2 == 0:
                graph.edge(f'Node{i}', f'Node{j}')

    graph.view()


if __name__ == "__main__":

    # create_fake_data()
    # create_clique_graph(25)
    # create_bipartite_graph(15,15)
    # create_disconnected_components_graph()
    create_sparse_graph()
```

Part 2. 1

```python
import json
import os

def generate_dot_code(course_data, prerequisites_data):
   allowed_subjects = ['CSCI', 'MATH']
   dot_code = "digraph CoursePrerequisites {\n"
   crn_to_course_name = {}

   for subject_data in course_data:
       subject_code = subject_data.get('code', '')
       courses = subject_data.get('courses', [])

       if subject_code not in allowed_subjects:
           continue

       for course_info in courses:
           if 'crse' not in course_info or 'id' not in course_info or 'title' not in
course_info or 'sections' not in course_info:
               continue

           course_id = str(course_info['crse'])
           course_name = course_info['id']
           title = course_info['title']

           crn = course_info['sections'][0].get('crn', '')
           crn_to_course_name[crn] = course_name

           if 'instructor' in course_info['sections'][0]:
               professor = course_info['sections'][0]['instructor']
           else:
               professor = "Unknown"

           subject = course_info.get('subj', "Unknown")
           crn = course_info['sections'][0].get('crn', '')
           print(crn)

           prerequisites_info = prerequisites_data.get(str(crn),
{}).get('prerequisites', {})
           # print(prerequisites_data.get(str(crn)))
           prerequisite_code = extract_prerequisite_code(prerequisites_info)
```

```python
            # print(prerequisites_info)
            # print(prerequisite_code)

            dot_code += f'  {course_name} [label="{title}\n({course_name})\\nProf:
{professor}\\nSubject: {subject}\\nCRSE: {course_id}"]\n'
            dot_code += f'  {prerequisite_code} -> {course_name}\n'

    dot_code += "}\n"
    return dot_code

def extract_prerequisite_code(prerequisites_info):
    if prerequisites_info:
        type = prerequisites_info.get('type', '')
        if type == 'and':
            nested_prerequisites = prerequisites_info.get('nested', [])
            for nested_item in nested_prerequisites:

                print(nested_item)
                # course_crn = nested_item.get('nested', []).get('course', '')
                # print(course_crn)
        elif type == 'or':
            nested_prerequisites = prerequisites_info.get('nested', [])
            for nested_item in nested_prerequisites:
                pass
        elif type == 'course':
            course_crn = prerequisites_info.get('course', '')
            print(course_crn)
    return ''


if __name__ == "__main__":
    print("Current Working Directory:", os.getcwd())
    with open('/Users/jaeseok/Developer/IV/courses.json', 'r') as course_file:
        course_json = json.load(course_file)

    with open('/Users/jaeseok/Developer/IV/prerequisites.json', 'r') as
prerequisite_file:
        prerequisites_json = json.load(prerequisite_file)

    dot_code = generate_dot_code(course_json, prerequisites_json)
    print(dot_code)
```

```
output_file_path = '/Users/jaeseok/Developer/IV/subjects.dot'
with open(output_file_path, 'w') as output_file:
    output_file.write(dot_code)

print(f"DOT code written to {output_file_path}")
```

Part 2. 2

```
import os
import json
import graphviz

def parse(file):
   fp = open(file)
   data = json.load(fp)
   return data, fp

# Past 4 Spring Semesters
courseset1 = "quacs-data/semester_data/202301/courses.json"
courseset2 = "quacs-data/semester_data/202309/courses.json"

data1 = parse(courseset1)[0]
data2 = parse(courseset2)[0]

data = data1 + data2

course_dict = dict()

for course in data[14]['courses']:
   instructor_set = set()
   title = course['title']
   id = course['id']
   if id[6] != '9':
       for section in course['sections']:
           for timeslot in section['timeslots']:
```

```
            instructors = timeslot['instructor']

            if ',' in instructors:
                instructors = instructors.split(',')
            else:
                instructors = [instructors]

            for instructor in instructors:
                instructor = instructor.strip()
                if (instructor != "Shianne M. Hulbert" and instructor != 'TBA'):
                    instructor_set.add(instructor)

    course_dict[title] = instructor_set

# graph = graphviz.Graph()

# for course, instructors in course_dict.items():
#     graph.node(course, color='lightblue', style='filled', shape='box')
#     for instructor in instructors:
#         graph.node(instructor, color='lightgreen', style='filled', shape='ellipse')

# # Add edges to connect courses to instructors
# for course, instructors in course_dict.items():
#     for instructor in instructors:
#         graph.edge(course, instructor)

# graph.view()

graph = graphviz.Graph(engine='circo', graph_attr={'scale': '0.5'})

# Add nodes with different colors for courses and instructors
for course, instructors in course_dict.items():
   graph.node(course, color='lightblue', style='filled', shape='box')
   for instructor in instructors:
       graph.node(instructor, color='lightgreen', style='filled', shape='ellipse')

# Add edges to connect courses to instructors
for course, instructors in course_dict.items():
   for instructor in instructors:
       graph.edge(course, instructor)

graph.view()
```

```
# Save the graph to a file and render it
# graph.render('bipartite_course_graph', format='png', cleanup=True)
```