Welcome to the CoGrammar Graphs Lecture

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Coding Interview Workshop Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions]

Coding Interview Workshop Housekeeping cont.

- For all **non-academic questions**, please submit a query: [www.hyperiondev.com/support]

- Report a **safeguarding** incident: [www.hyperiondev.com/safeguardreporting]

- We would love your **feedback** on lectures: [Feedback on Lectures]

CoGrammar

Skills Bootcamp 8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar

Graphs

May 2024

Learning Objectives

- ❖ Define and illustrate the fundamental concepts of graphs, including vertices (nodes), edges, and the different types of graphs and their representations.
- ❖ Implement graph traversal algorithms in Python and JavaScript, focusing on depth-first search (DFS) and breadth-first search (BFS).
- ❖ Apply graphs to solve problems such as finding the shortest path, detecting cycles, and understanding the concepts of connectivity and graph components.

Learning Objectives

- ❖ Construct and analyze graphs from real-world data to model networks, relationships, and paths, demonstrating the practical applications of graphs in areas like social network analysis, network routing, and resource allocation.
- ❖ Evaluate the efficiency and limitations of graph algorithms, understanding how the choice of data structure and implementation affects the performance of these algorithms in different scenarios.

What is a graph in Computer Science?

A. A visual representation of data
B. A data structure that consists of nodes and edges
C. An algorithm that helps in the sorting of data
D. A concept which involves organising data in rows and columns

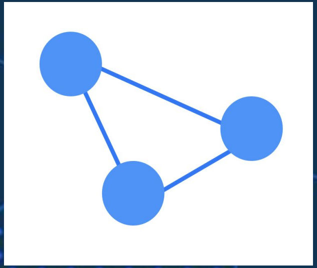
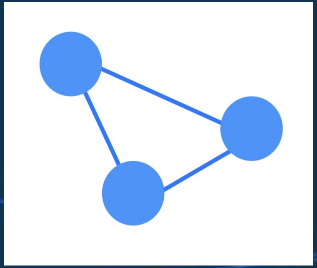
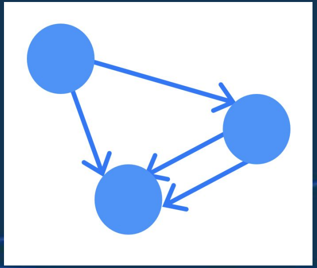
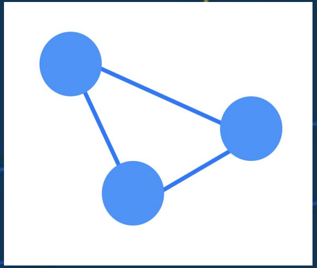
Which of the following graphs is a directed graph?

A. 1 – 2,
2 – 3,
3 - 1

B. 1 → 2,
2 → 3,
3 → 3
3 ← 1

C. 1 → 2,
2 → 3,
3 ← 2,
1 → 3

D. 1 → 2,
2 → 3,
1 → 3

CoGrammar

What is the degree of a node in a graph?

A. The number of nodes surrounding the node

B. The sum of the weights of the edges from the node

C. The number of edges connected to a node

D. The label assigned to a node

CoGrammar

Simulating Social Networks

Consider a social networking site, like Facebook or LinkedIn, that supports the addition of new, unique members. Members can choose to follow other members, which forms a link between them.

- ➢ What **data structure** could be used to represent a network like this made up of **members and connections**?
- ➢ How could we use this structure to be able to report on the **total number of members** in the network, the **number of connections** a member has and the **shortest number of connections** between two members?

Modelling networks

- **Can we graphically represent a network?**
  - Use shapes to represent the members
  - Use lines connecting the shapes to represent the connections

- **Let's try and draw this together**
  - Can we determine the number of members?
  - Can we determine the number of connections of a member?
  - What is the shortest connection between two members?

CoGrammar

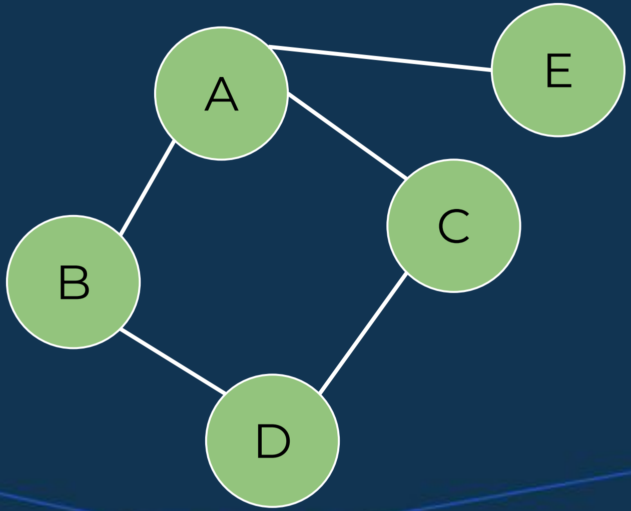
Graphs

A non-linear data structure made up of vertices or nodes and connected by edges or arcs, that is used to represent complex relationships between objects.

- Graphs are made up of two sets, the **Vertices (*V*)** and **Edges (*E*)**.
- Each element of *E* is a **pair** consisting of two elements from *V*.
- Vertices can be **labelled** and may be a **reference** to an external entity with additional information known as **attributes**.
- Edges can be **labelled** and the pairs can be **ordered** depending on the type of the graph.

CoGrammar

- ❖ Graphs are depicted visually using circles or boxes to represent vertices, and lines (or arrows) between the circles or boxes to represent edges. This can only be done for small datasets.

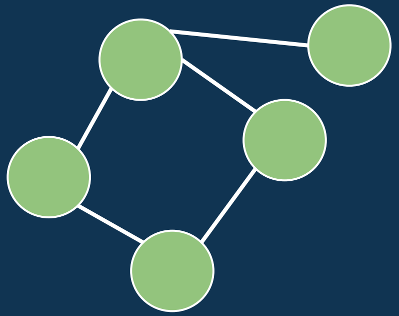
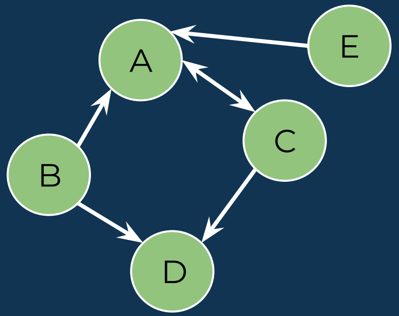


- **Neighbours:** Two nodes that are connected by an edge. This property is also known as **adjacency**.
  - *E.g. A is adjacent to B, A and B are neighbours*
- **Degree:** The number of other nodes that a node is connected to (i.e. the number of neighbours a node has).
  - *E.g. The degree of A is 3*
- **Loop:** An edge that connects a node to itself.

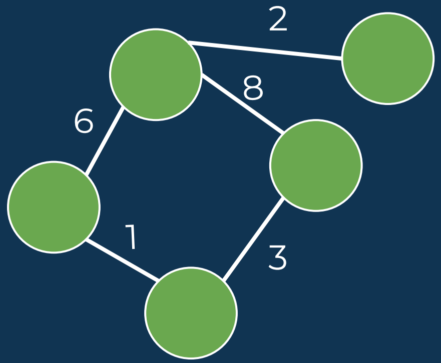
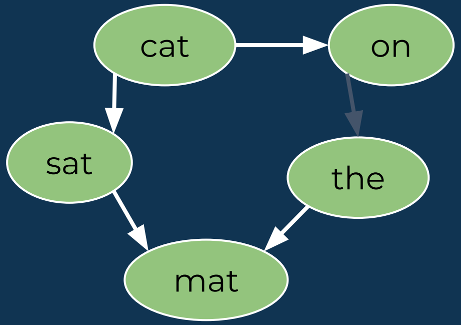
- ❖ **Path:** A sequence of nodes that are connected by edges.
  - ➢ *E.g. (A, B, D) denoted using **sequence notation ()***
- ❖ **Cycle:** A closed path which starts and ends at the same node and no node is visited more than once.
  - ➢ *E.g. {A, C, D, B, A}*

CoGrammar

Types of Graphs

Undirected Graphs Edges connecting nodes have no direction. For this graph, the order of the pairs of vertices in the edge set does not matter. **Applications:** Social networks, Recommendation Systems	
Directed Graphs Edges connecting nodes have specified directions. Edges may also be bidirectional. This graph cannot contain any loops. **Applications:** Maps, Network Routing, WWW	

CoGrammar

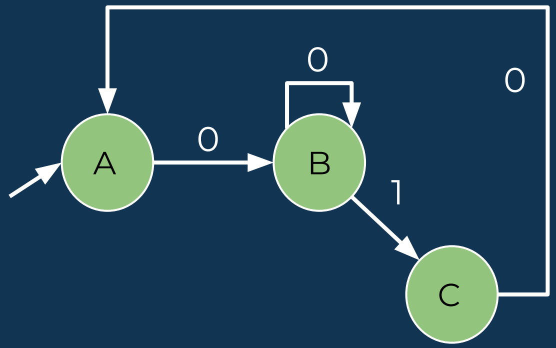
Weighted/Labelled Graphs Directed/Undirected graphs that have values associated with each of its edges. These values can record any information relating to the edge e.g. distance between nodes. **Applications:** Transportation Networks, Financial/Transactional Networks	
Vertex Labelled Graphs Directed/Undirected graphs where the vertices/nodes in the graph are labelled with information which identifies the vertex. **Applications:** Biological Networks (molecular structures), Semantic Networks	

CoGrammar

Cyclic Graphs

A directed graph which contains at least one cycle.

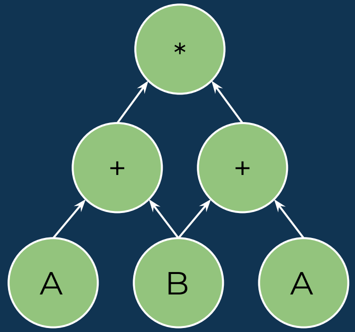
<u>Applications:</u> Task Scheduling, Manufacturing Processes, Finite State Machines (a mathematical model of computation)



Directed Acyclic Graphs

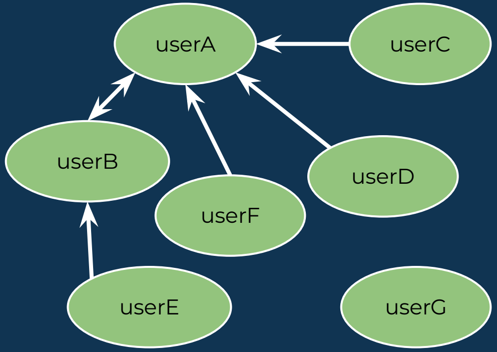
Also known as a DAG. A directed graph with no cycles. Various use-cases across fields.

<u>Applications:</u> Dependency Resolution Systems e.g. package management, Project Management, Compiler Design



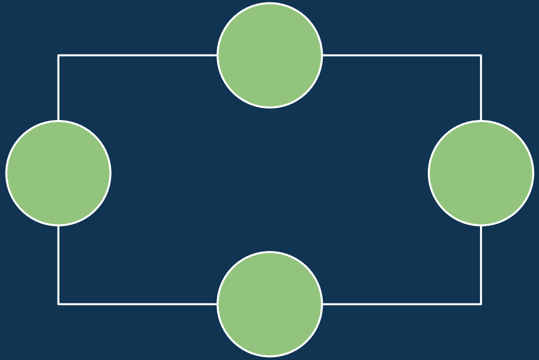
Disconnected Graphs
The graph contains nodes which are not connected via an edge to any other nodes in the graph.

Applications: Social Networks, Transportation Networks, Component Analysis



Connected Graphs
There is a path from any node in the graph to any other node in the graph.

Applications: Communication Networks, Routing Algorithms, Circuit Design, Data Analysis



CoGrammar

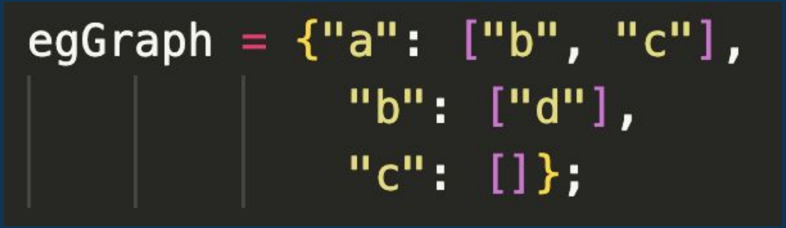
Let’s Breathe!

Let’s take a small break before moving on to the next topic.

CoGrammar

Implementation of Graphs

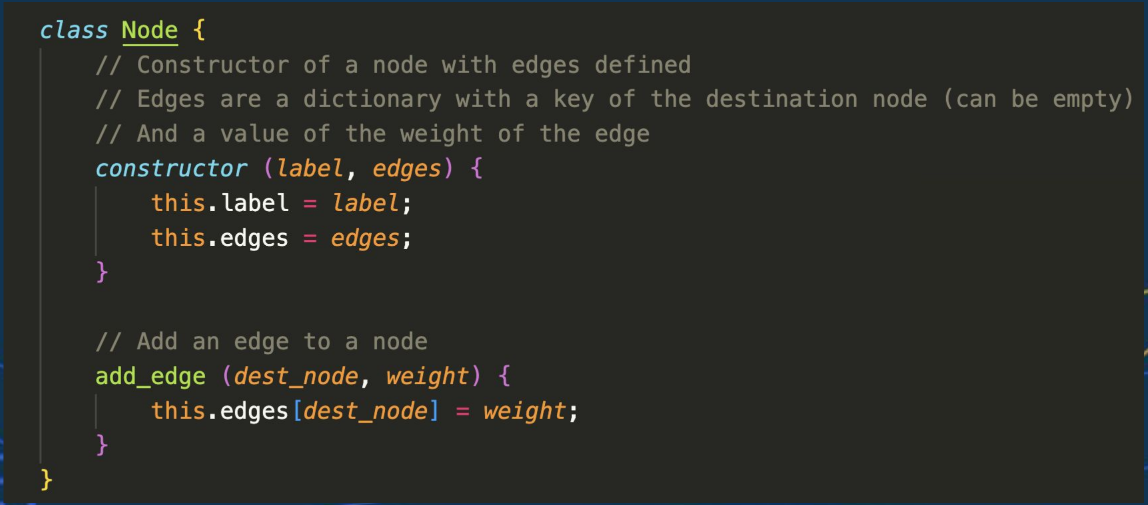
- ❖ The simplest way to implement a graph is using **dictionaries or maps**.
- ❖ This method can be used for undirected graphs and edge-weighted graphs. Can anyone guess how we can do that given the implementation below?
- ❖ We can use **recursive functions** to search for paths in the graph.



```
egGraph = {"a": ["b", "c"],
           "b": ["d"],
           "c": []};
```

CoGrammar

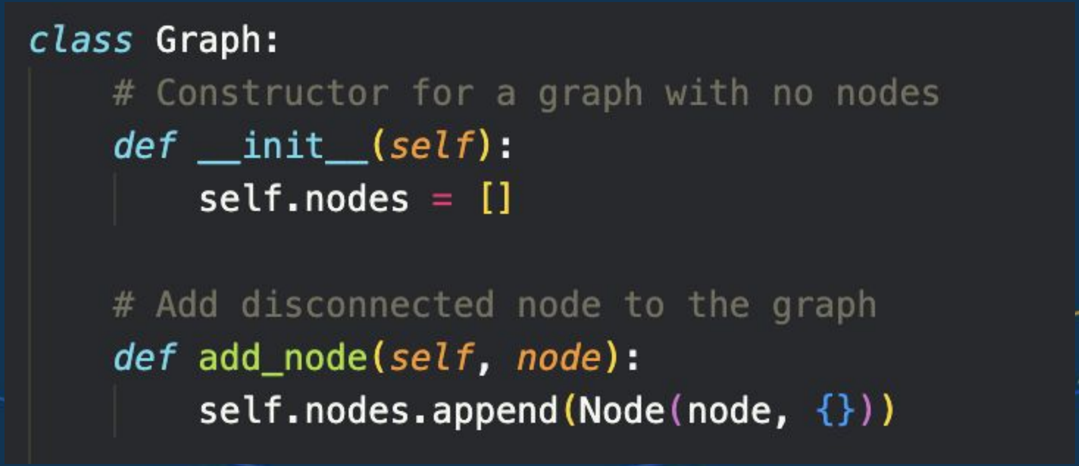
- ❖ For a more complicated implementation with a little more control over the structure and implementation, an **OOP approach** can be taken.
- ❖ We create a **Node class** which stores the data in the node and the weights of all the connected nodes.



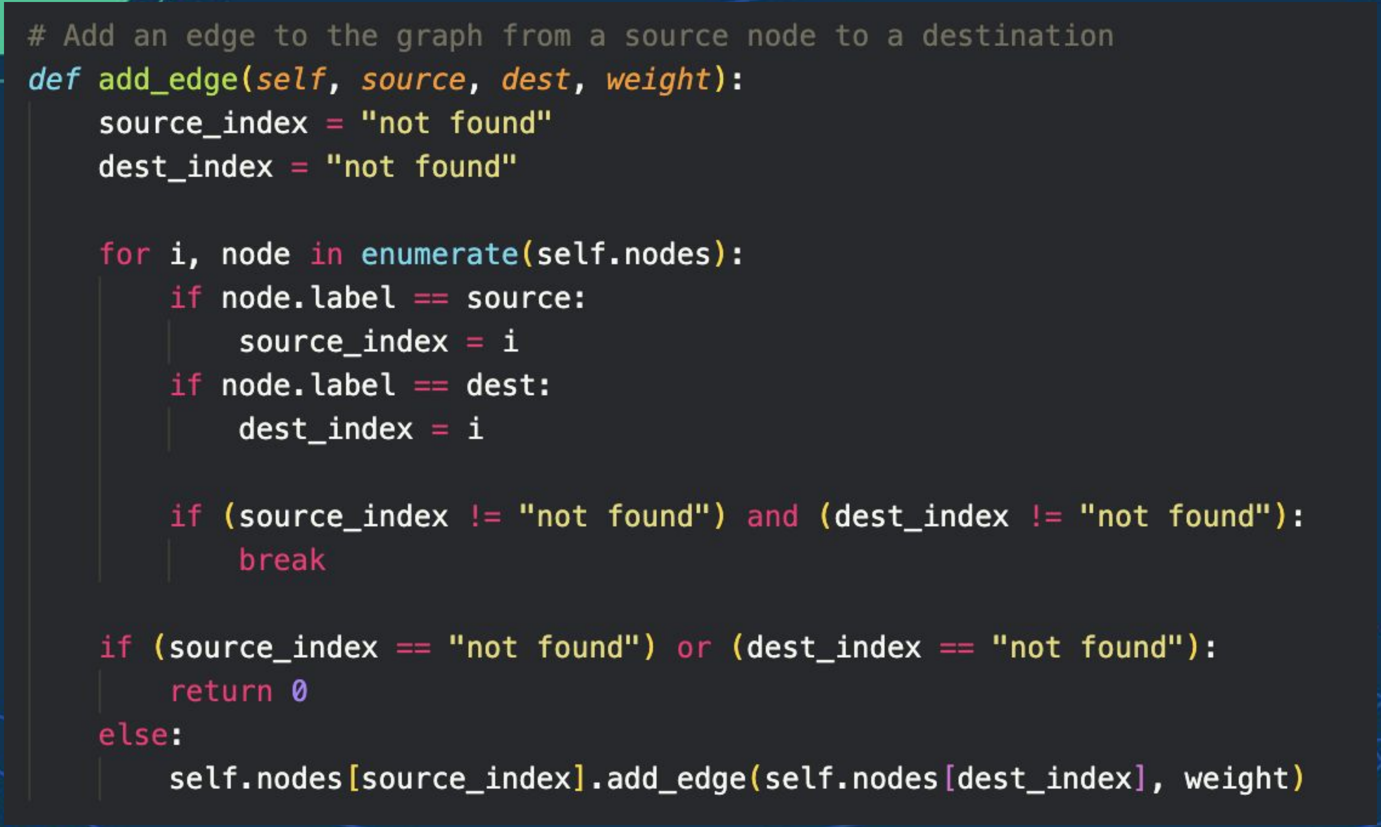
```
class Node {
    // Constructor of a node with edges defined
    // Edges are a dictionary with a key of the destination node (can be empty)
    // And a value of the weight of the edge
    constructor (label, edges) {
        this.label = label;
        this.edges = edges;
    }

    // Add an edge to a node
    add_edge (dest_node, weight) {
        this.edges[dest_node] = weight;
    }
}
```

CoGrammar

- We create a **Graph class** to store all the nodes in the Graph.
- In this class we add functions to **add nodes**, **add edges**, **print out the graph** and **any additional functions (like searching, sorting etc).**

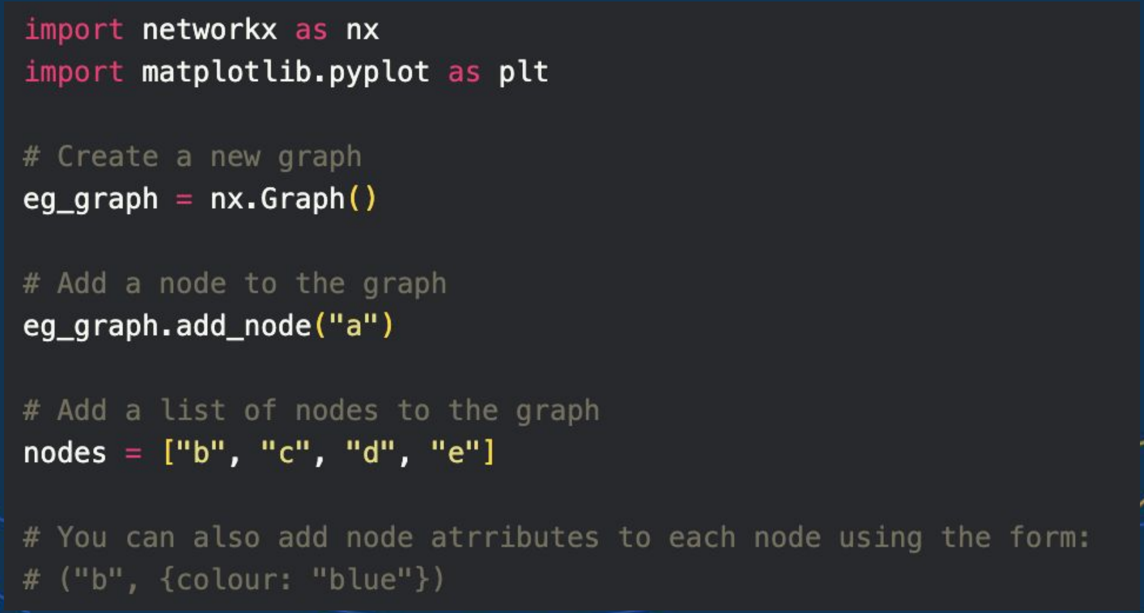


```python
class Graph:
    # Constructor for a graph with no nodes
    def __init__(self):
        self.nodes = []

    # Add disconnected node to the graph
    def add_node(self, node):
        self.nodes.append(Node(node, {}))
```



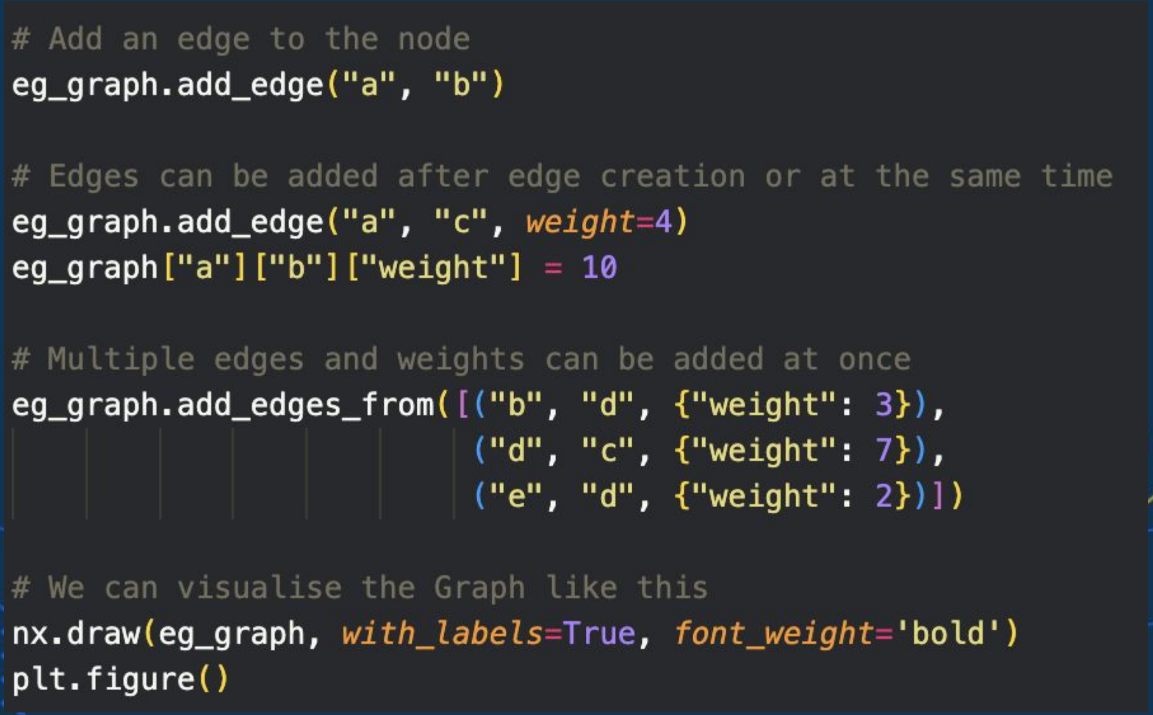
```python
# Add an edge to the graph from a source node to a destination
def add_edge(self, source, dest, weight):
    source_index = "not found"
    dest_index = "not found"

    for i, node in enumerate(self.nodes):
        if node.label == source:
            source_index = i
        if node.label == dest:
            dest_index = i

        if (source_index != "not found") and (dest_index != "not found"):
            break

    if (source_index == "not found") or (dest_index == "not found"):
        return 0
    else:
        self.nodes[source_index].add_edge(self.nodes[dest_index], weight)
```

❖ The best implementation of Graphs is by using a package called **NetworkX** or **JSNetworkX** (install using pip or npm).



```python
import networkx as nx
import matplotlib.pyplot as plt

# Create a new graph
eg_graph = nx.Graph()

# Add a node to the graph
eg_graph.add_node("a")

# Add a list of nodes to the graph
nodes = ["b", "c", "d", "e"]

# You can also add node atrributes to each node using the form:
# ("b", {colour: "blue"})
```

❖ NetworkX allows use to **add nodes, edges, weights, attributes and directions** as well as **visualise graphs**, easily and efficiently.



```
# Add an edge to the node
eg_graph.add_edge("a", "b")

# Edges can be added after edge creation or at the same time
eg_graph.add_edge("a", "c", weight=4)
eg_graph["a"]["b"]["weight"] = 10

# Multiple edges and weights can be added at once
eg_graph.add_edges_from([("b", "d", {"weight": 3}),
                         ("d", "c", {"weight": 7}),
                         ("e", "d", {"weight": 2})])

# We can visualise the Graph like this
nx.draw(eg_graph, with_labels=True, font_weight='bold')
plt.figure()
```

Worked Example

Consider a delivery company wanting to visualise their network of package collection points to help improve efficiency and organisation of delivery routes.

Start	End	Distance
A	B	10
C	D	12
A	D	7
B	C	4
B	E	8

1. Using the package collection points provided, create a graph for this scenario.
   a. Determine the weights of each link.
   b. Add directions to the links.
   c. What type of graph is this?

2. What is the shortest route that visits all the collection points, assuming all links are bidirectional? (Determine the minimum spanning tree.)

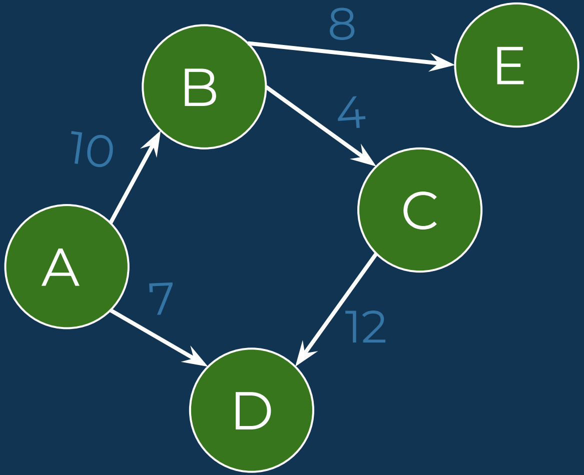
CoGrammar

Worked Example

Consider a delivery company wanting to visualise their network of package collection points to help improve efficiency and organisation of delivery routes.

Start	End	Distance
A	B	10
C	D	12
A	D	7
B	C	4
B	E	8

1. Using the package collection points provided, create a graph for this scenario.
   a. Determine the weights of each link.
   b. Add directions to the links.
   c. What type of graph is this?



Directed Acyclic Graph

CoGrammar

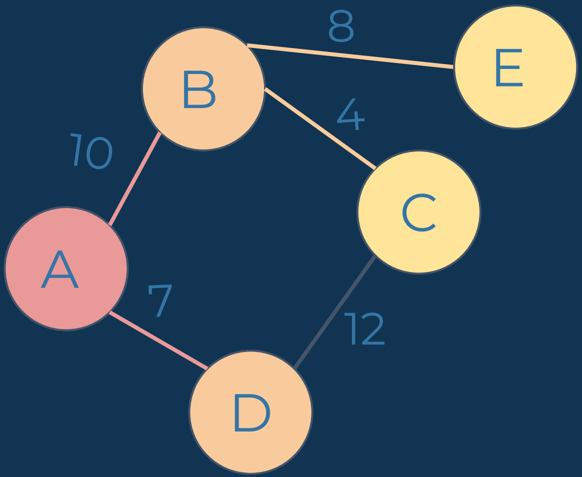
Worked Example

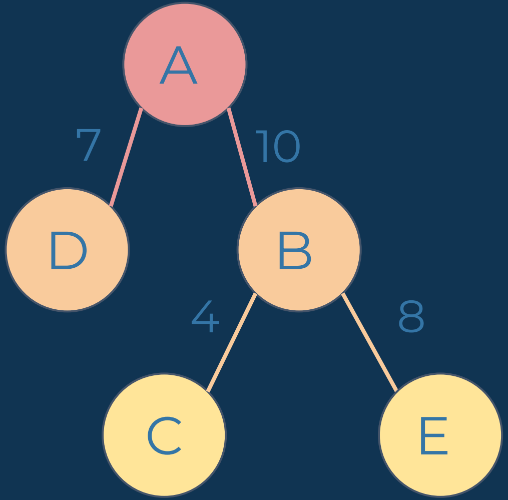
Consider a delivery company wanting to visualise their network of package collection points to help improve efficiency and organisation of delivery routes.

Start	End	Distance
A	B	10
C	D	12
A	D	7
B	C	4
B	E	8

2. What is the shortest route that visits all the collection points, assuming all links are bidirectional? (Determine the minimum spanning tree).

Start with A





Portfolio Assignment: SE

Network Routing Simulator

Objective: Develop a command-line tool in Python that simulates network routing using graph algorithms.

CoGrammar

Portfolio Assignment: SE

Requirements:

- ➢ Implement a graph class in Python to represent the network.
- ➢ Allow users to input network data in a standard format.
- ➢ Simulate packet routing between nodes, demonstrating the shortest path and alternative routes.
- ➢ Include error handling for disconnected nodes and unreachable destinations.
- ➢ Provide clear output showing the steps of the routing process and results.
- ➢ Include a README file explaining how to use the tool.
- ➢ Provide example inputs and outputs for different network scenarios.

CoGrammar

Portfolio Assignment: DS

Social Network Analysis

Objective: Analyze a real-world social network dataset to extract meaningful insights using graph theory.

Portfolio Assignment: DS

Requirements:

- ➢ Obtain a social network dataset.
- ➢ Clean and preprocess the data to construct a graph where nodes represent users and edges represent connections.
- ➢ Analyze the structure of the graph (e.g., degree distribution, clustering coefficient).
- ➢ Perform centrality analysis to find key influencers.
- ➢ Identify communities and clusters within the network.
- ➢ Write a report detailing the methodology, analysis, and insights.
- ➢ Include visualizations and code snippets in the report.
- ➢ Provide a README file showing how to visualise the analysis.

CoGrammar

Portfolio Assignment: WD

Interactive Graph Visualizer

Objective: Develop an interactive web application that visualizes graphs and demonstrates graph traversal algorithms.

CoGrammar

Portfolio Assignment: WD

Requirements:

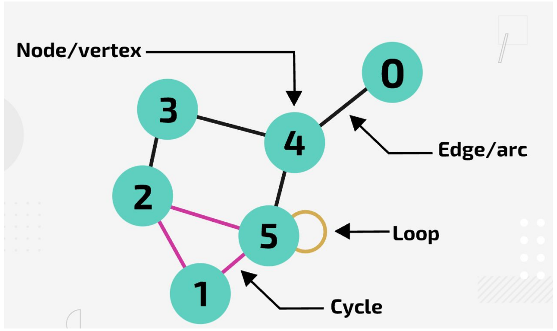
- ➢ Use HTML, CSS, and JavaScript to create the user interface.
- ➢ Allow users to draw and edit graphs by adding/removing vertices and edges.
- ➢ Implement visual representations for different types of graphs (directed, undirected, weighted).
- ➢ Implement a JavaScript backend that processes graph data.
- ➢ Use an appropriate graph library for graph operations.
- ➢ Include error handling for invalid graph inputs.
- ➢ Provide interactive examples within the application.
- ➢ Include a README file explaining how to use the application.
- ➢ Provide example graphs and use cases.

Summary

Graphs

- ★ Non-linear data structure
- ★ Made up of nodes/vertices
- ★ Connected by edges/links

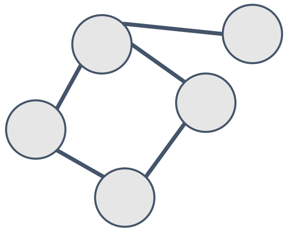
Terminology

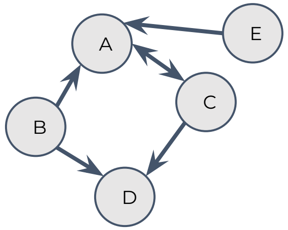


CoGrammar

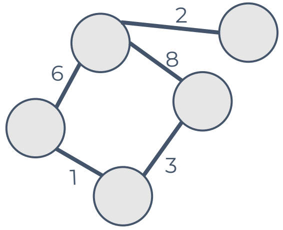
Summary

Types of Graphs

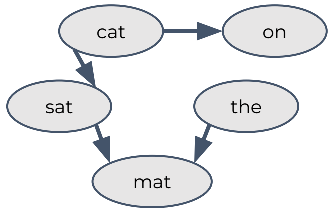

Undirected Graphs



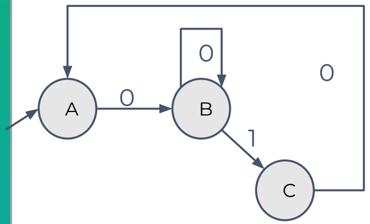
Directed Graphs



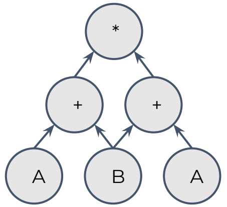
Weighted Graphs



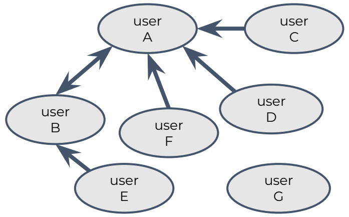
Vertex Labelled Graphs



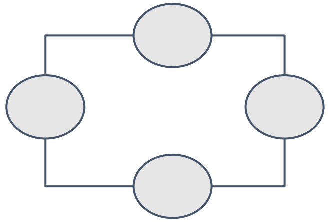
Cyclic Graphs



Directed Acyclic Graphs



Disconnected Graphs


Connected Graphs

CoGrammar

Additional Resources

- ❖ Ada Computer Science - Introduction to Graphs with lots of visuals
- ❖ Portland State University - Comprehensive guide to graphs, with a more Mathematics centered approach
- ❖ Simplilearn - In-depth introduction to graphs, covers the different types well, lots of visual guides
- ❖ GeeksForGeeks - Prim’s Algorithm explanation and implementation

CoGrammar

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

Thank you for attending

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar