CoGrammar

GRAPHS

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Foundational Sessions Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(FBV: Mutual Respect.)**
- 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 submit these for upcoming Open Classes. You can submit these questions here: **SE Open Class Questions or DS Open Class Questions**

Foundational Sessions 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**

Reminder

Guided Learning Hours

By now, ideally you should have 7 GLHs per week accrued. Remember to attend any and all sessions for support, and to ensure you reach 112 GLHs by the close of your Skills Bootcamp.

Progression Criteria

✅ **Criterion 1: Initial Requirements**

- Complete 15 Guided Learning Hours and the first four tasks within two weeks.

✅ **Criterion 2: Mid-Course Progress**

- Software Engineering: Finish 14 tasks by week 8.
- Data Science: Finish 13 tasks by week 8.

✅ **Criterion 3: Post-Course Progress**

- Complete all mandatory tasks by 24th March 2024.
- Record an Invitation to Interview within 4 weeks of course completion, or by 30th March 2024.
- Achieve 112 GLH by 24th March 2024.

✅ **Criterion 4: Employability**

- Record a Final Job Outcome within 12 weeks of graduation, or by 23rd September 2024.

What is a graph in Computer Science?

A. A visual representation of data

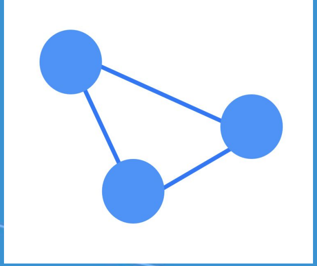
B. A data structure that consists of nodes and edges

C. An algorithm that helps in the sorting of data

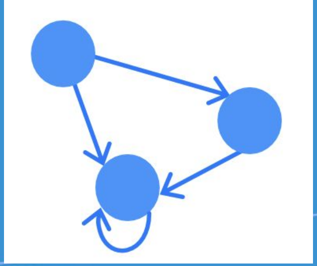
D. A concept which involves organising data in rows and columns

Which of the following graphs is a directed graph?

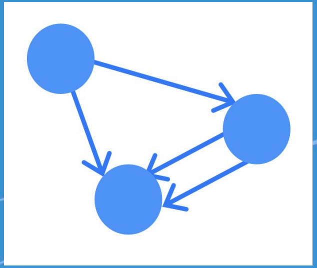
A. 1 – 2,
2 – 3,
3 - 1



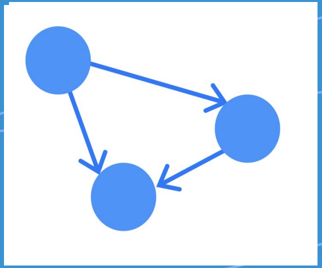
B. 1 → 2,
2 → 3,
3 → 3
3 ← 1



C. 1 → 2,
2 → 3,
3 ← 2,
1 → 3



D. 1 → 2,
2 → 3,
1 → 3



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

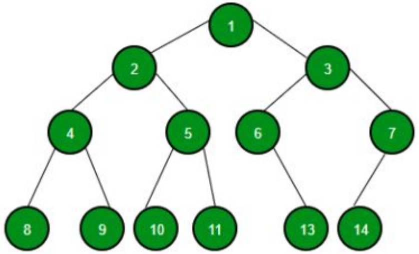
Recap of Trees

CoGrammar

Binary Trees

A tree structure where each node has at most two children.

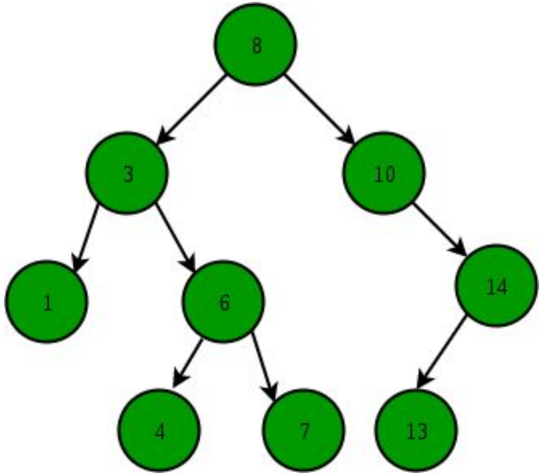
- **Properties**
  - Hierarchical structure with a single root.
  - Each node has zero, one, or two children.



Source: GeeksForGeeks

Binary Search Trees (BSTs)

A type of binary tree where each node has a key, and every node's key is greater than all keys in its left subtree and less than all keys in its right subtree.

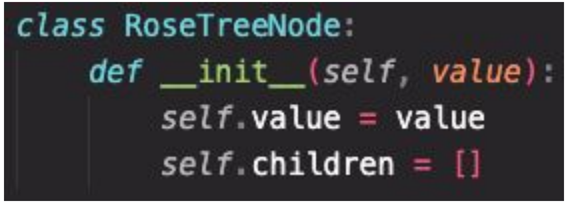


Source: GeeksForGeeks

Rose Trees

A tree in which each node can have an arbitrary number of children.

- Rose Trees are ideal for representing hierarchical structures with varying depths.
- The Rose Tree node is similar to the BST node, although it has an array for all it’s children, not just the left and right children.



```python
class RoseTreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []
```

B-Trees

A type of balanced tree data structure, commonly used in databases and file systems. They are a specialized form of a rose tree, where each node can have multiple children.

- **Properties:**
  - Each node in a B-Tree can **have several children**, determined by the tree's order.
  - Nodes keep data in sorted order, facilitating efficient access and insertion.
  - **Automatically balances itself** to maintain a low height.

Graphs Topics

1. Introduction to Graphs
2. Types of Graphs
3. Prim’s Shortest Path Algorithm
4. Implementing Graphs in Python

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?

Using a Graph to model a network

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

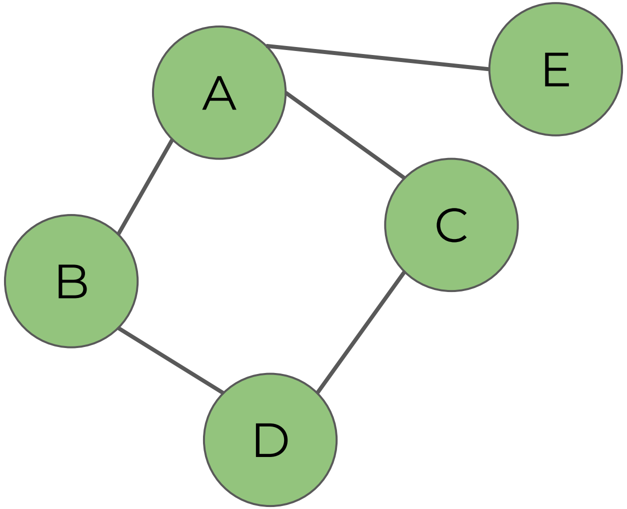
- **Let’s try and draw this together**
  - Can we determine the number of members?
  - Can we determine the number of connections a member has?
  - What is the shortest connection between two members?

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.

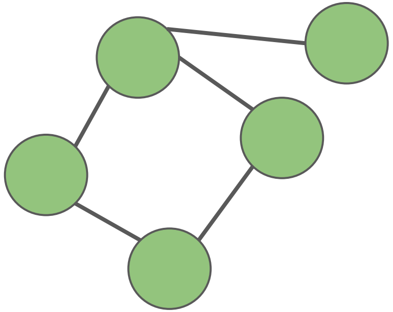
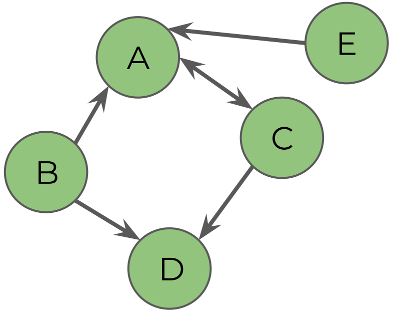
- Graphs are made up of two sets, the **Vertices (*V*)** and **Edges (*E*)**.
- Each element of *E* is a **pair** consisting of two elements from *V*.
- Vertices can be **labelled** and may be a **reference** to an external entity with additional information known as **attributes**.
- Edges can be **labelled**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

- 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.

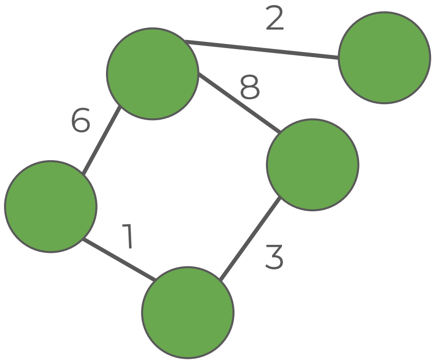
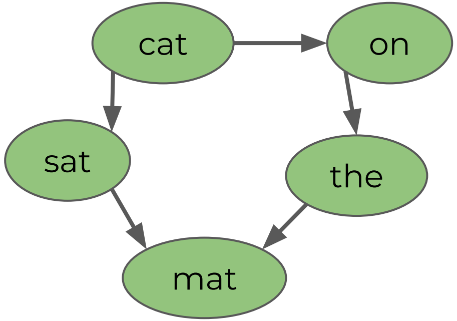


- **Neighbours:** Two nodes that are connected by an edge. This property is also known as **adjacency**.
    - *E.g. A is adjacent to B, A and B are neighbours*
- **Degree:** The number of other nodes that a node is connected to (i.e. the number of neighbours a node has).
    - *E.g. The degree of A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

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	

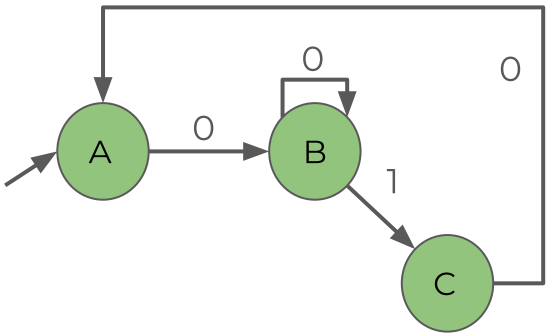
CoGrammar

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	

Cyclic Graphs

A directed graph which contains at least one cycle.

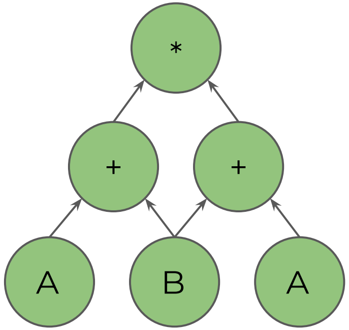
Applications: Task Scheduling, Manufacturing Processes, Finite State Machines (a mathematical model of computation)



Directed Acyclic Graphs

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

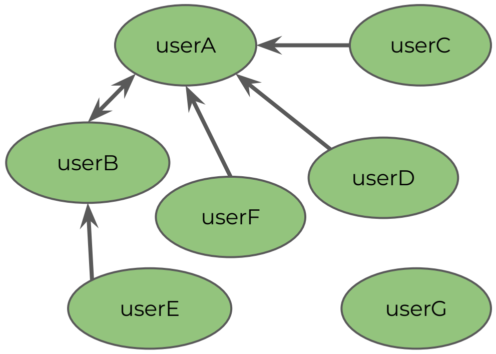
Applications: Dependency Resolution Systems e.g. package management, Project Management, Compiler Design



Disconnected Graphs

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

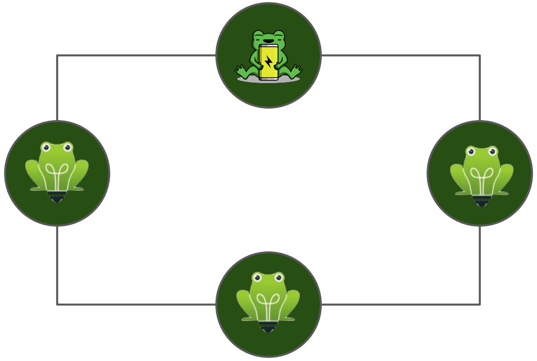
Applications: Social Networks, Transportation Networks, Component Analysis



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



Prim’s Minimum Spanning Tree

A greedy algorithm used to find the subset of edges that form a tree that includes every vertex and that has the minimum total number of edges (and weight) for an undirected graph.

- This algorithm is typically used for **weighted graphs**, but it can be used for unweighted graphs.
- There are other algorithms which could be used to accomplish this task but this one is one of the **simplest** to implement and **performs well** for connected, weighted graphs with medium density.

- The general method for Prim's algorithm for weighted graphs is:

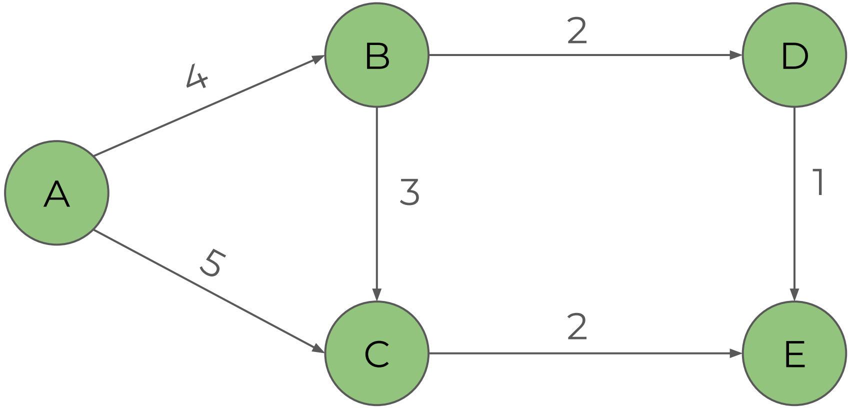
   1. Create an empty spanning tree (referred to as **MSP**) and a set of all the vertices in the graph (referred to as **G**).
   2. Select an arbitrary starting vertex and remove it from **G**.
   3. Identify all the edges connecting the vertices in **G** to vertices in **MSP**. This set of edges is known as the **cut**.
   4. Add the edge with the smallest weight to **MSP** and remove the vertex it connects to from **G**.
   5. Repeat steps 3 and 4 until **G** is empty.

- For unweighted graphs, in step 4 instead of choosing the edge with the smallest weight, any edge can be chosen.

CoGrammar

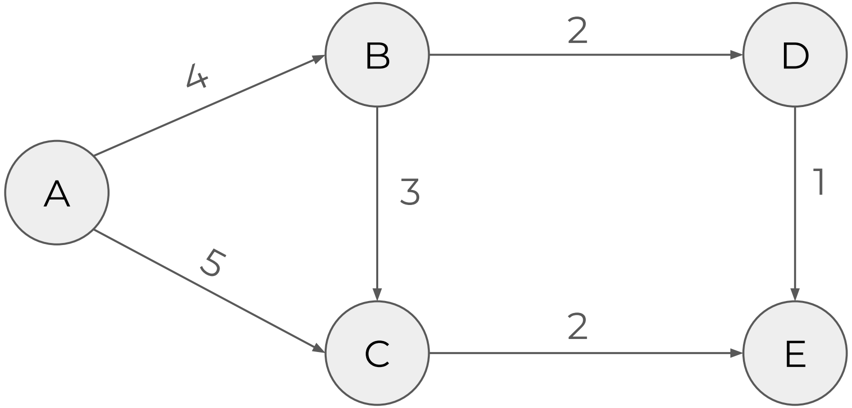
Worked Example: Prim’s MST

- Find the minimum spanning tree of the following graph:



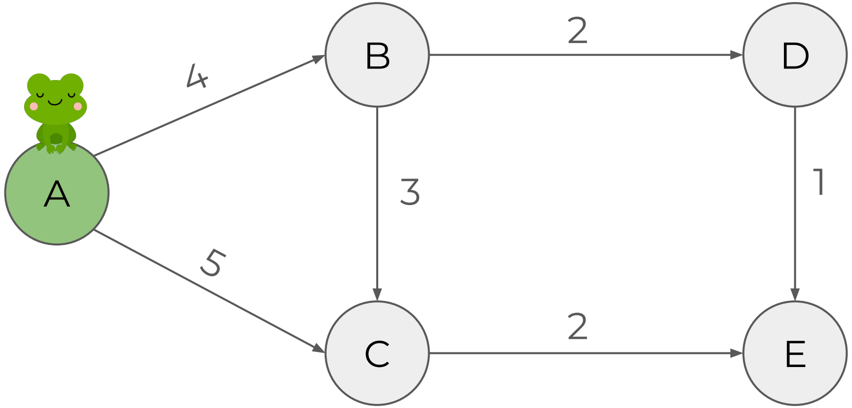
Worked Example: Prim's MST

1. Create an empty spanning tree (referred to as **MST**) and a set of all the vertices in the graph (referred to as **G**).



Worked Example: Prim’s MST

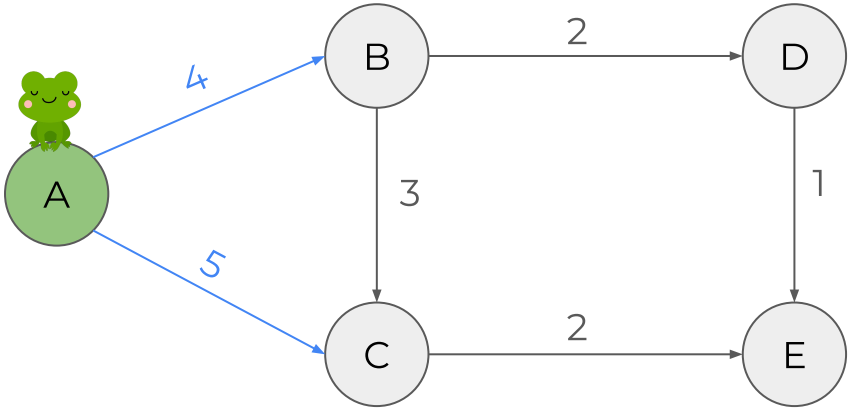
2. Select an arbitrary starting vertex, remove it from **G** and add it to the **MST**.



CoGrammar

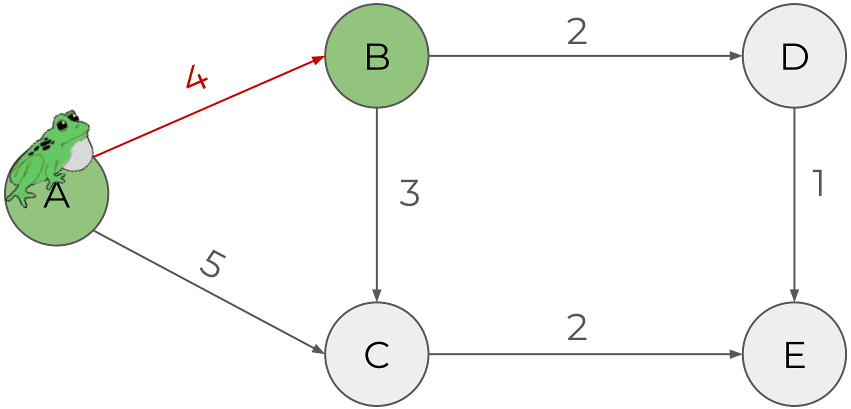
Worked Example: Prim’s MST

3. Identify all the edges connecting the vertices in **G** to vertices in **MST**. This set of edges is known as the **cut**.



Worked Example: Prim’s MST

4. Add the edge with the smallest weight to **MST** and remove the vertex it connects to from **G** and add it to the **MST**.

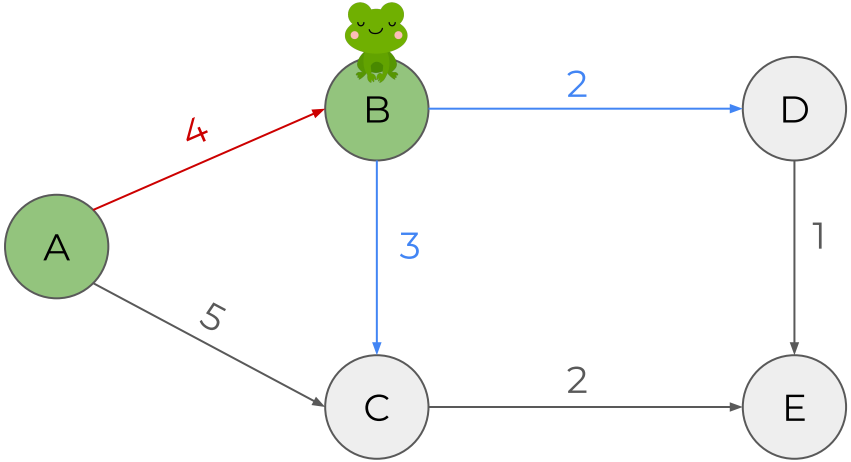


5. Repeat steps 3 and 4 until **G** is empty.

CoGrammar

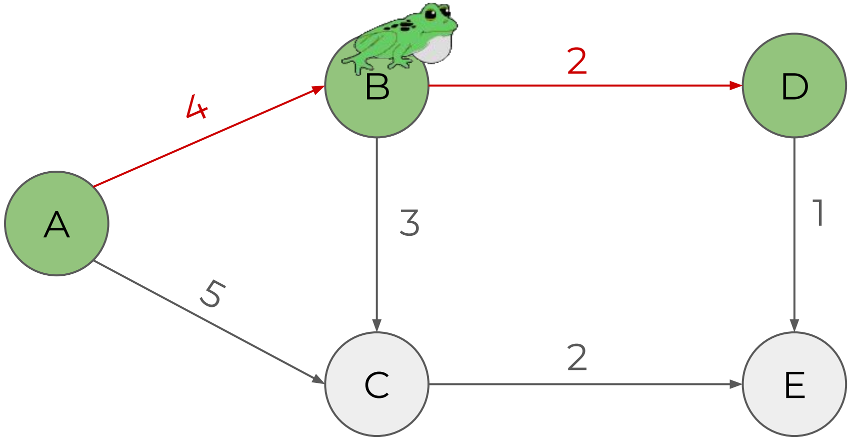
Worked Example: Prim's MST

3. Identify all the edges connecting the vertices in **G** to vertices in **MST**. This set of edges is known as the **cut**.



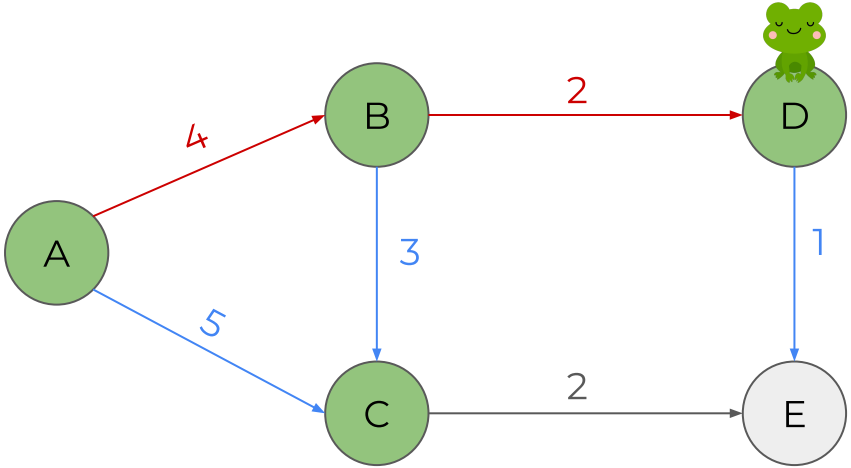
Worked Example: Prim’s MST

4. Add the edge with the smallest weight to **MST** and remove the vertex it connects to from **G** and add it to the **MST**.



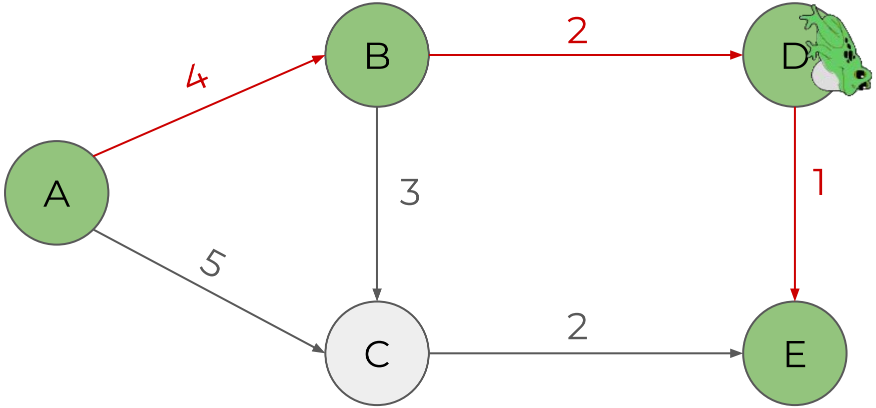
Worked Example: Prim’s MST

3. Identify all the edges connecting the vertices in **G** to vertices in **MST**.



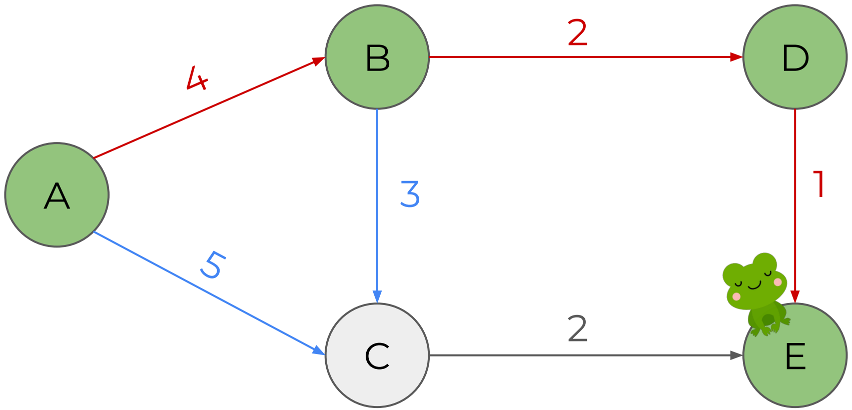
Worked Example: Prim’s MST

4. Add the edge with the smallest weight to **MST** and remove the vertex it connects to from **G** and add it to the **MST**.



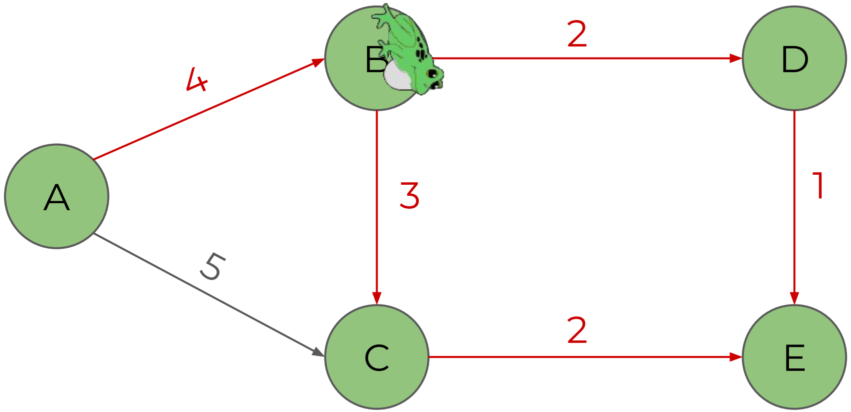
Worked Example: Prim’s MST

3. Identify all the edges connecting the vertices in **G** to vertices in **MST**.

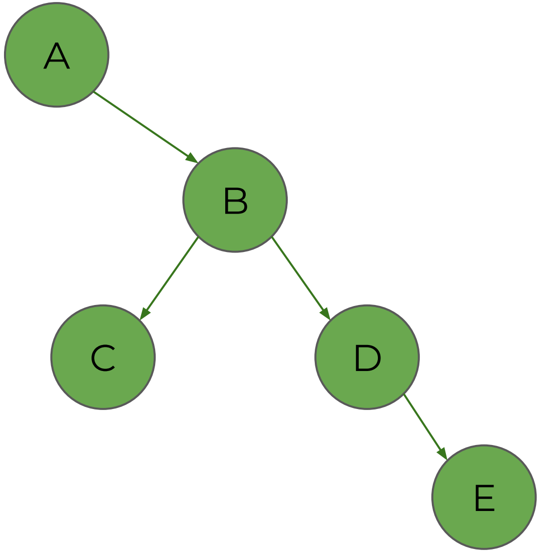


Worked Example: Prim's MST

4. Add the edge with the smallest weight to **MST** and remove the vertex it connects to from **G** and add it to the **MST**.

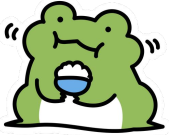


Worked Example: Prim’s MST



Total weight: 10

- **Greedy Algorithms:** Algorithms that build solutions to a problem piece by piece, choosing each piece by whichever will have the most obvious and immediate benefit.



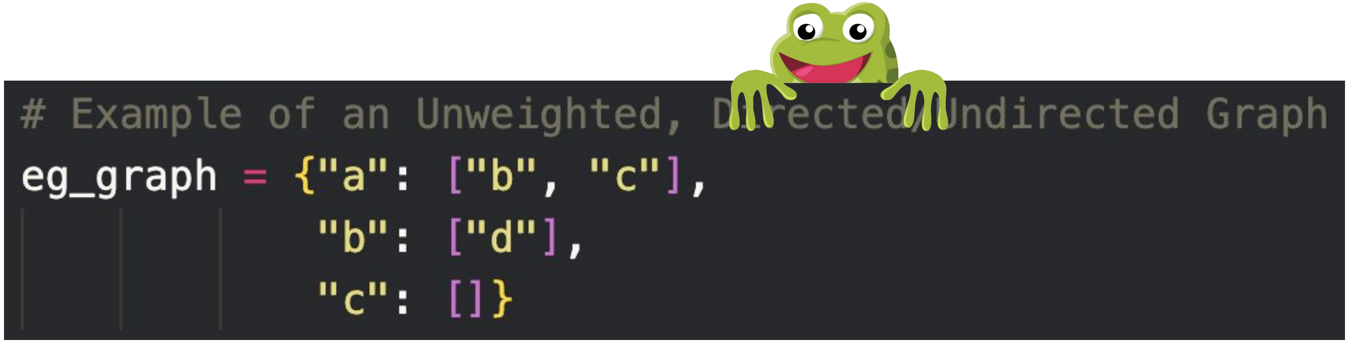
- Usually, greedy algorithms are not the most optimal solution but they may produce an approximately optimal solution in a reasonable amount of time.
- Prim's algorithm makes a series of **locally optimal decisions**, in order to find the **globally optimal solution**.
- Hence it is considered a **greedy algorithm**.
- It has also been proven to always produce an optimal solution for connected, weighted graphs.

Implementation of Graphs

- The simplest way to implement a graph is using **dictionaries**.

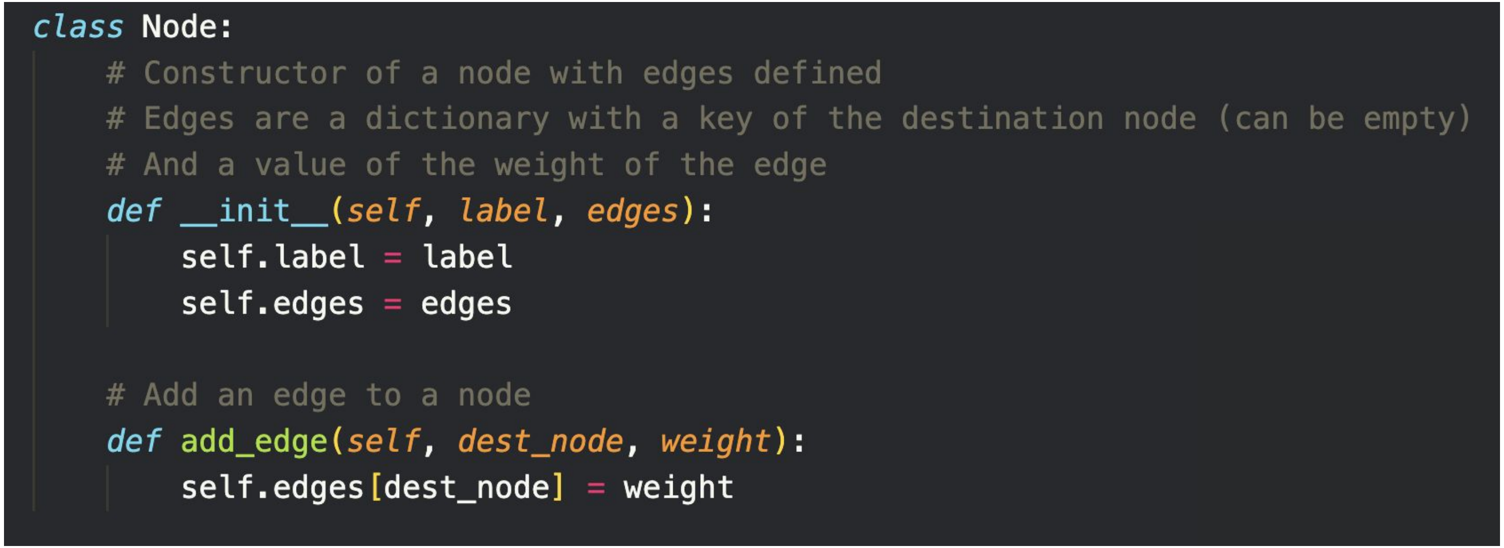
- This method can be used for undirected graphs and edge-weighted graphs. Can anyone guess how we can do that given the implementation below?

- We can use **recursive functions** to search for paths in the graph.



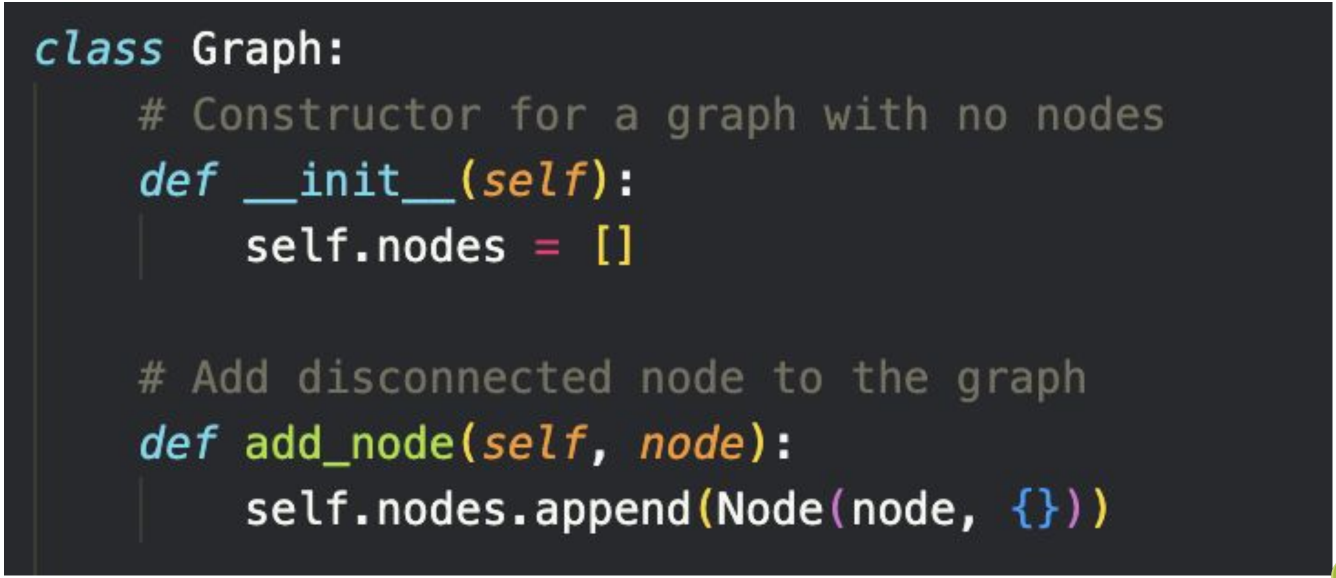
```
# Example of an Unweighted, Directed/Undirected Graph
eg_graph = {"a": ["b", "c"],
             "b": ["d"],
             "c": []}
```

- 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.

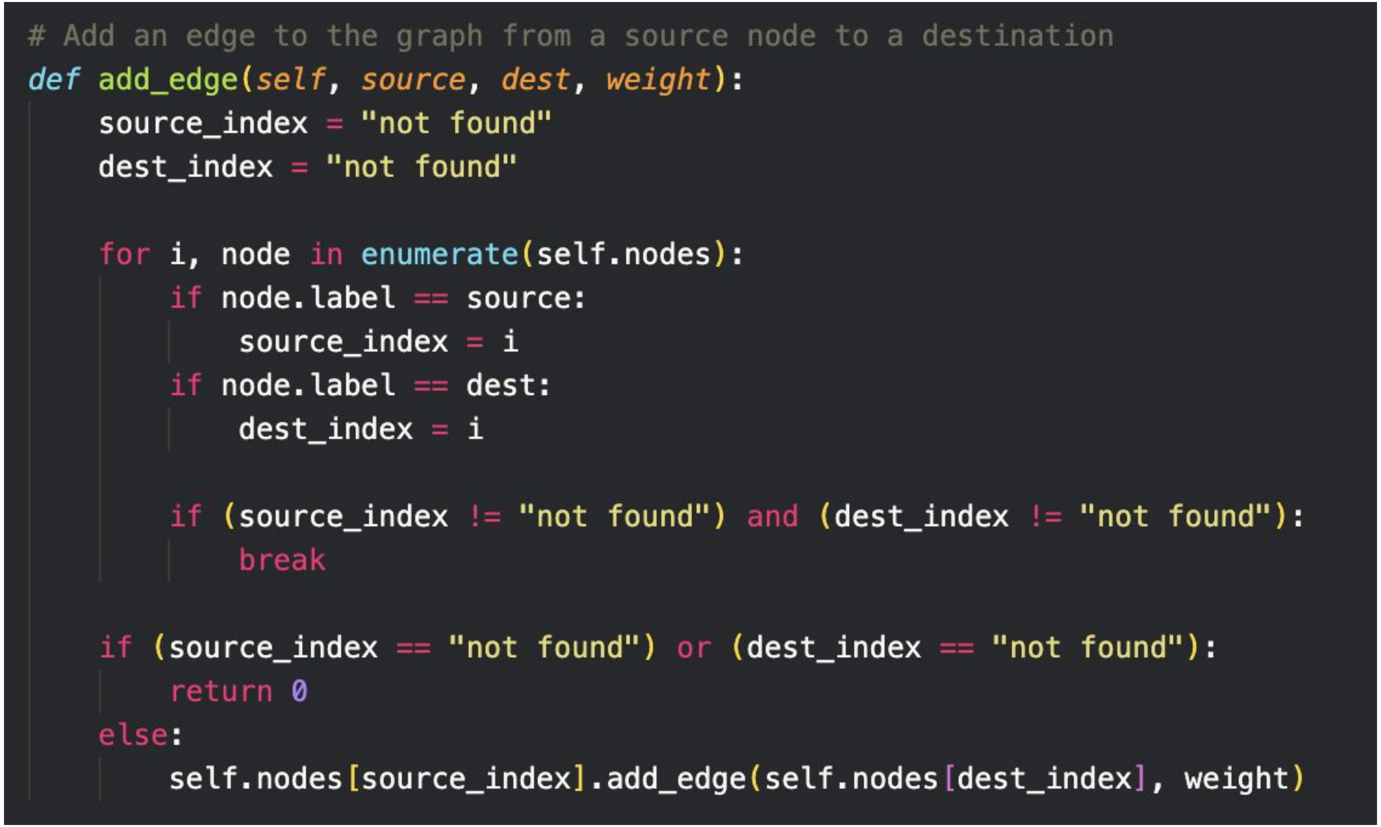


```
class Node:
    # Constructor of a node with edges defined
    # Edges are a dictionary with a key of the destination node (can be empty)
    # And a value of the weight of the edge
    def __init__(self, label, edges):
        self.label = label
        self.edges = edges

    # Add an edge to a node
    def add_edge(self, dest_node, weight):
        self.edges[dest_node] = weight
```

- 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).**



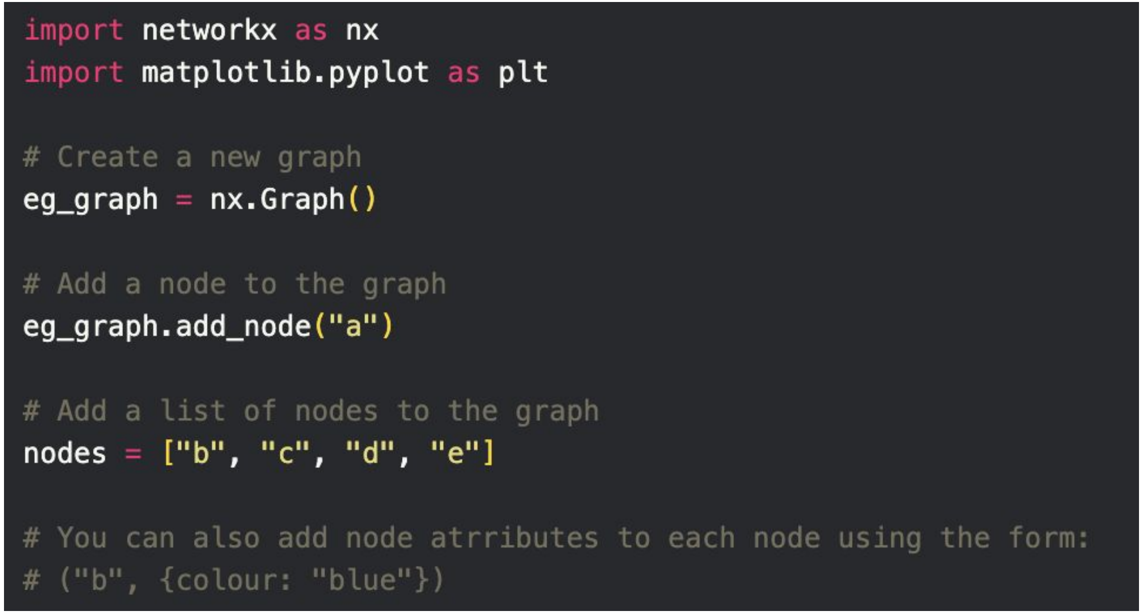
```python
class Graph:
    # Constructor for a graph with no nodes
    def __init__(self):
        self.nodes = []

    # 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)
```

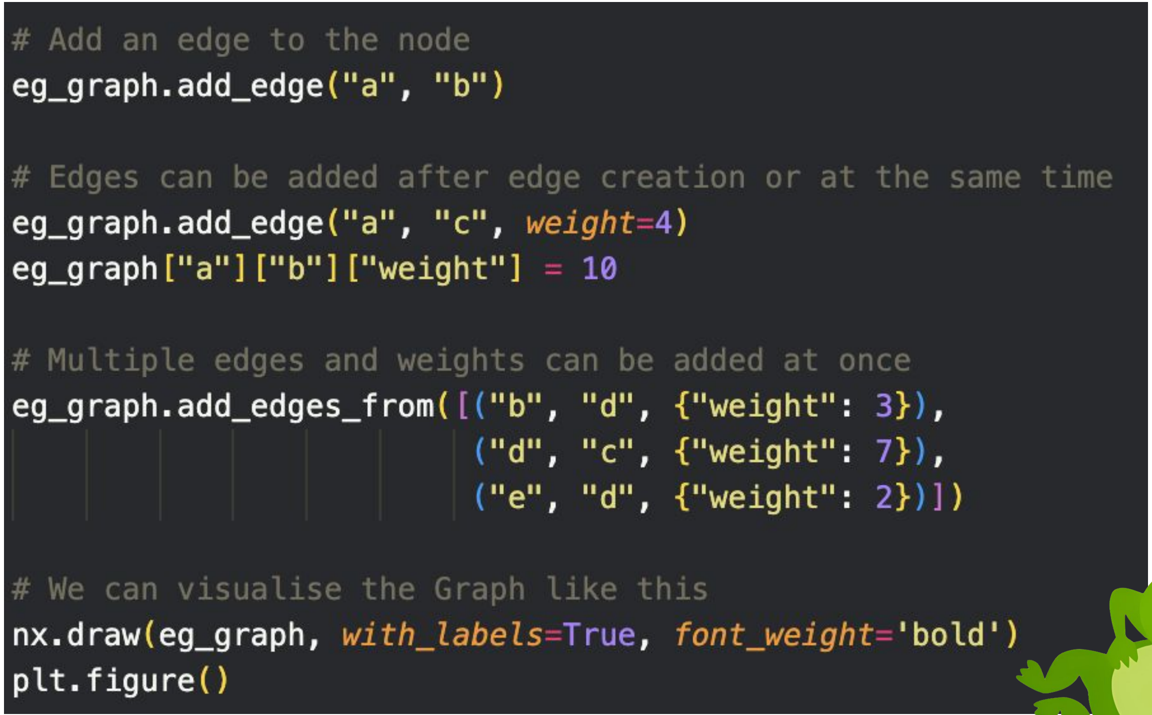
CoGrammar

- The best implementation of Graphs is by using a package called **Networkx** (which you can install by running “pip install networkx”).



```
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
D	C	4
C	B	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? (Determine the minimum spanning tree.)

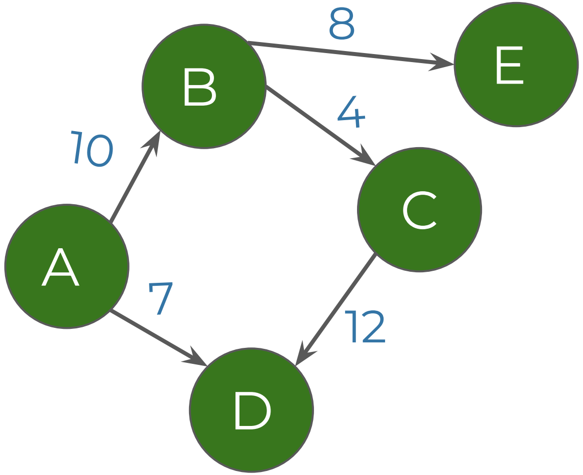
Worked Example

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

CoGrammar

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

Worked Example

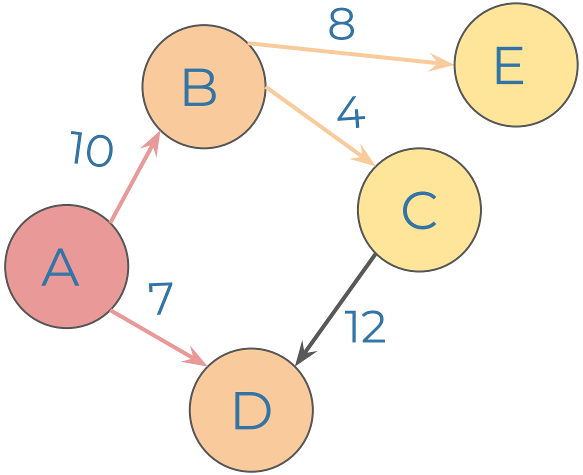
Consider a delivery company wanting to visualise their network of package collection points to help improve efficiency and organisation of delivery routes.

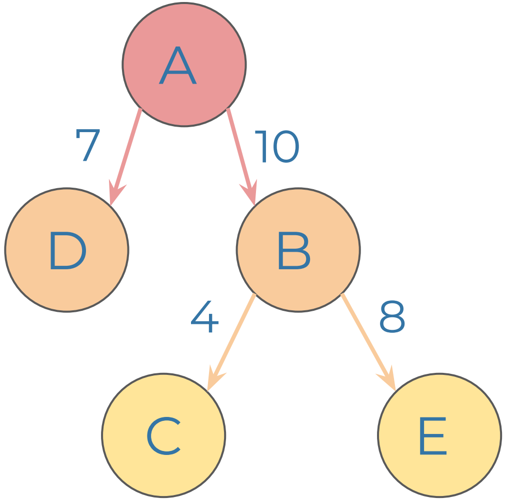
Start	End	Distance
A	B	10
C	D	12
A	D	7
B	C	4
B	E	8

CoGrammar

2. What is the shortest route that visits all the collection points? (Determine the minimum spanning tree).

Start with A



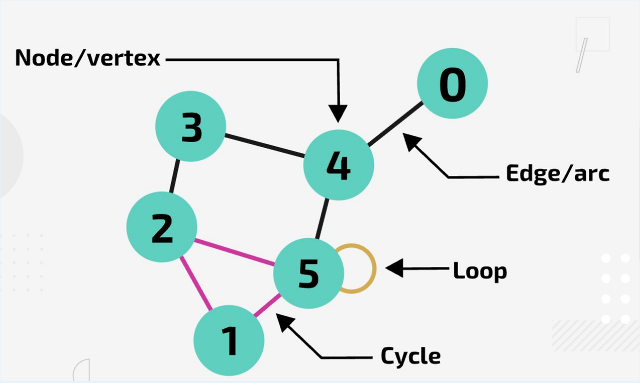


Summary

Graphs

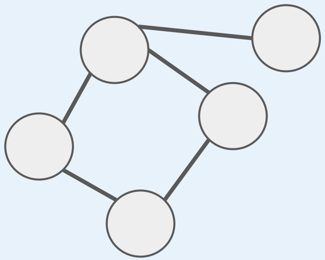
- ★ Non-linear data structure
- ★ Made up of nodes/vertices
- ★ Connected by edges/links

Terminology

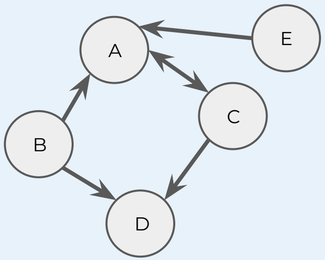


Summary

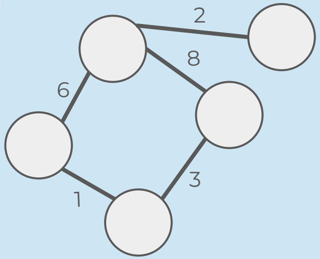
Types of Graphs



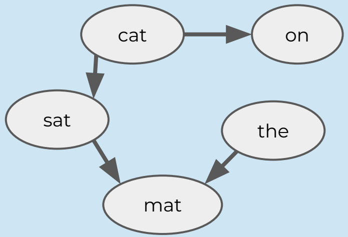
Undirected Graphs



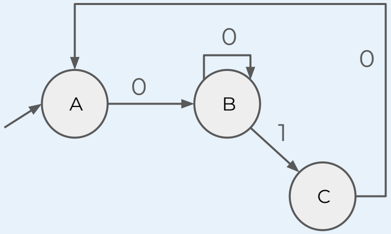
Directed Graphs



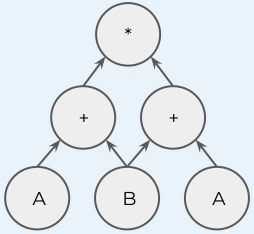
Weighted Graphs



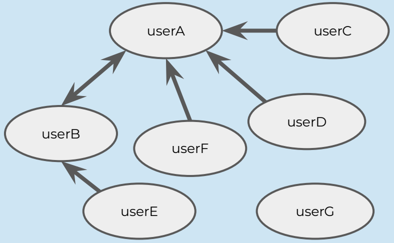
Vertex Labelled Graphs



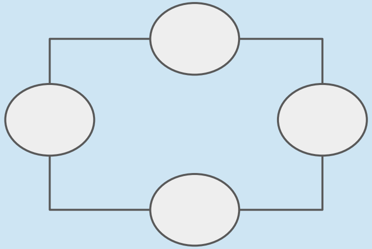
Cyclic Graphs



Directed Acyclic Graphs



Disconnected Graphs



Connected Graphs

Further Learning

- 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

What is a minimum spanning tree?

A. A tree that is made from the graph

B. The shortest path between two nodes in a graph

C. A tree that connects all nodes in a graph without forming cycles and has the minimum possible total edge weight

D. A tree that spans the entire graph with the maximum possible total edge weight

What is a cycle in a graph?

A. A path that visits every node at least once

B. A path that starts and ends at the same node

C. A set of edges that has the same weight in both directions

D. A node that has no edges

Questions and Answers

Questions around Graphs