CSC461
INTRODUCTION TO DATA SCIENCE

Dr. Muhammad Sharjeel

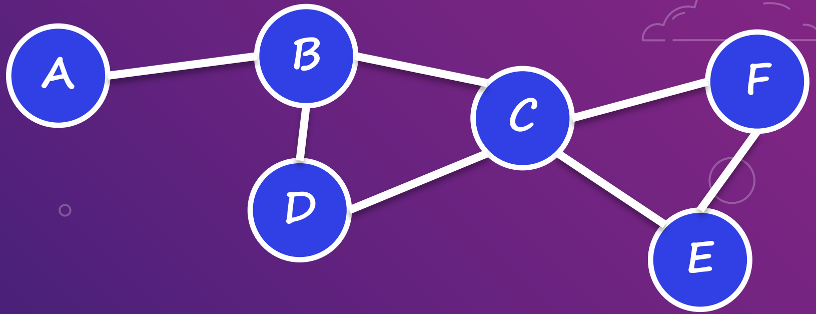
https://muhmmadsharjeel.github.io/

06

GRAPH PROCESSING

GRAPHS

- Networks are the systems of interrelated objects
- Graphs are the mathematical models for representing networks
- A graph is a collection of vertices (nodes) and edges, G = (V,E)



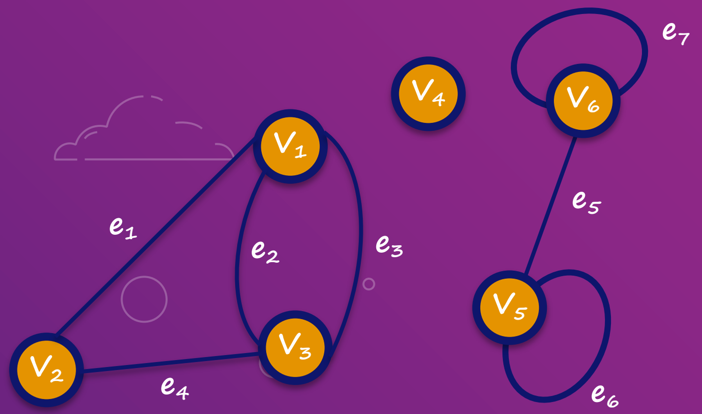
V = {A, B, C, D, E, F}
E = {(A, B), (B, D), (B, C), (D, C), (C, F), (C, E), (E, F)}

GRAPHS

- A graph (G) consists of two things:
  - Set of elements called vertices, points, or nodes of G [V = V(G)]
  - Set of unordered pairs of distinct vertices called edges of G [E = E(G)]
- Vertices “x” and “y” are said to be adjacent if there is an edge e = {x, y} joining them
- The edge “e” is said to be incident on each of its vertex’s x and y

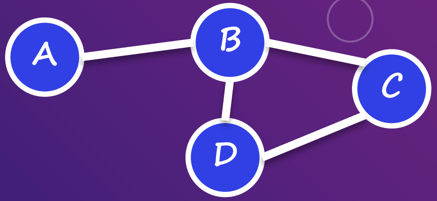
GRAPHS

- Vertex set = $\{v_1, v_2, v_3, v_4, v_5, v_6\}$
- Edge set = $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- e_1, e_2, and e_3 are incident on v_1
- v_2 and v_3 are adjacent to v_1
- e_6 and e_7 are loops
- e_2 and e_3 are parallel or multiple
- v_5 and v_6 are adjacent to themselves
- v_4 is an isolated vertex
- Endpoint $(e_5) = (v_5, v_6)$

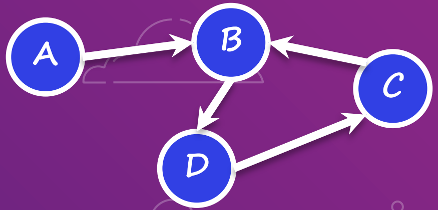


DIRECTED VS UNDIRECTED GRAPHS

- Undirected graphs have edges that do not have a direction
- Directed graphs have edges with direction



Directed

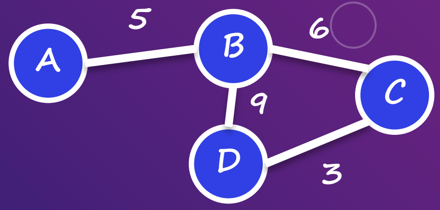


Undirected

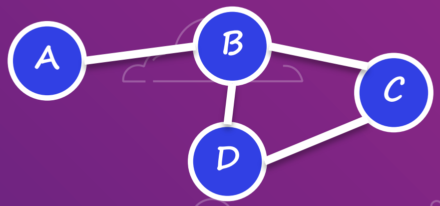
- Graphs of Facebook friends is undirected while the Twitter followers is an example of directed graph

WEIGHTED VS UNWEIGHTED GRAPHS

- Edges of a weighted graph will have weights
- Unweighted graph edges do not have weights



Weighted



Unweighted

- An airline which flies to difference routes would be a weighted graph
  - Airports are nodes, edges would represent flights (between airports), and edge weight would be the cost of flying between those airports

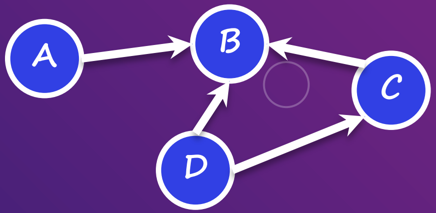
GRAPHS IN COMPUTERS

- There are a different ways graphs can be represented in a computer program
- Choice majorly depends on the use case
    - Graphs need to be modified dynamically (adding/removing nodes/edges)
    - Static graph just needs to be analyzed
- A good choice depends upon the operations that needs to be performed on the graph

- A graph can be represented using three main things:
    - Adjacency list
    - Adjacency dictionary
    - Adjacency matrix

ADJACENCY LIST

- For each node, store an array of the nodes that it connects to

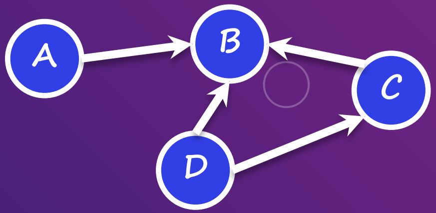


Node	Edges
A	[B]
B	[]
C	[B]
D	[B, C]

- Pros: easy to get all outgoing links from a given node, fast to add new edges (without checking for duplicates)
- Cons: deleting edges or checking existence of an edge requires scan through given node's full adjacency list

ADJACENCY DICTIONARY

- For each node, store a dictionary of the nodes that it connects to

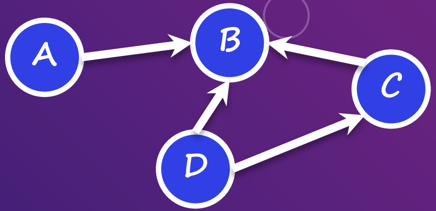


Node	Edges
A	{B: 1.0}
B	[]
C	{B: 1.0}
D	{B: 1.0, C: 1.0}

- Pros: easy to add/remove/query edges (requires dictionary lookups)
- Cons: overhead of using a dictionary

ADJACENCY MATRIX

- Store the connectivity of the graph as a matrix
- In virtually all cases, we'll get a sparse matrix



	A	B	C	D
A	0	0	0	0
B	1	0	1	1
C	0	0	0	1
D	0	0	0	0

- Pros/Cons: depends on sparse matrix format (implementation)

GRAPH ALGORITHMS

- The three algorithms that address different problem classes in graphs
  - Finding shortest paths in a graph – **Dijkstra's algorithm**
  - Finding important nodes in a graph – **PageRank algorithm**
  - Finding communities in a graph – **Girvan-Newman algorithm**

DIJKSTRA'S ALGORITHM

- Proposed by Edsger Dijkstra, Dutch computer scientist, in 1959
- Used for finding the shortest path, from starting node to target node
- Can only be applied to a weighted graph
- Only applicable when all weights are positive
- Used to find the shortest path between any two points, e.g., maps, telephone networks, IP routing etc.
- Advantages
  - Low complexity (almost linear)
  - Works for directed and weighted graphs when all edges have non-negative values
- Disadvantages
  - Does an obscured exploration that consumes a lot of time while processing
  - Could not handle negative edges

DIJKSTRA'S ALGORITHM

- Algorithm

Given: Graph $G = (V, E)$, Source s

Initialize:

$$D[s] \leftarrow 0,\ \ D[i \neq s] \leftarrow \infty$$
$$Q \leftarrow V$$

Repeat until Q empty:

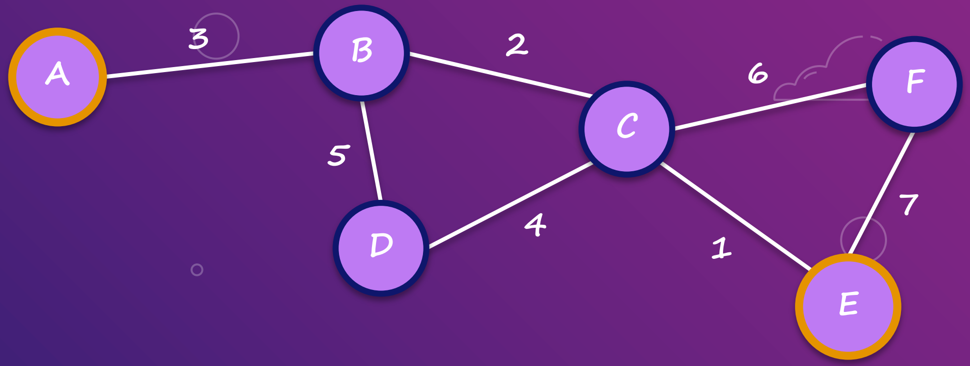
$i \leftarrow$ Remove element from Q with smallest D

For all j such that $(i, j) \in E$:

$$D[j] = \min(D[j], D[i] + 1)$$

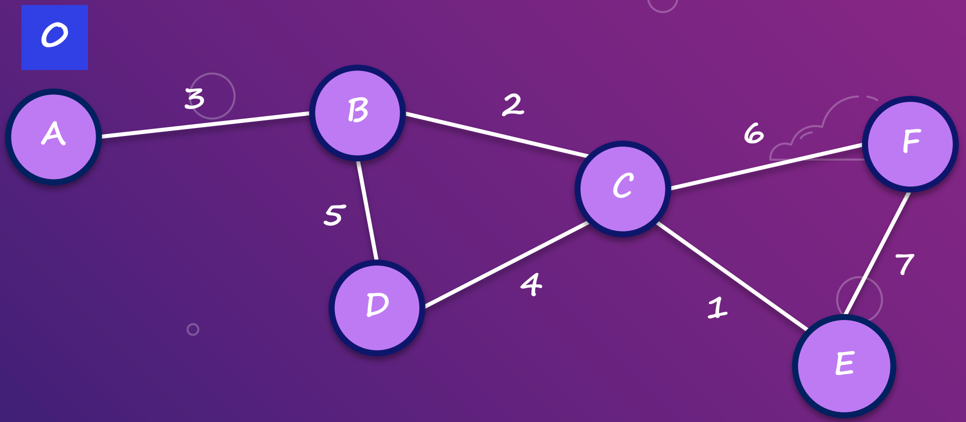
DIJKSTRA'S ALGORITHM

- Find the shortest path between two nodes (A---E)



DIJKSTRA'S ALGORITHM

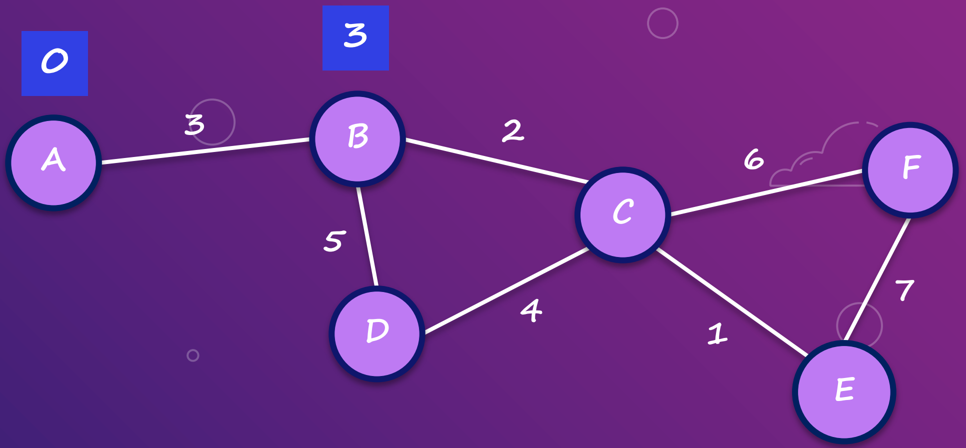
- Find the shortest path between two nodes (A---E)



D = [0, ∞, ∞, ∞, ∞, ∞]
Q = {A, B, C, D, E, F}

DIJKSTRA'S ALGORITHM

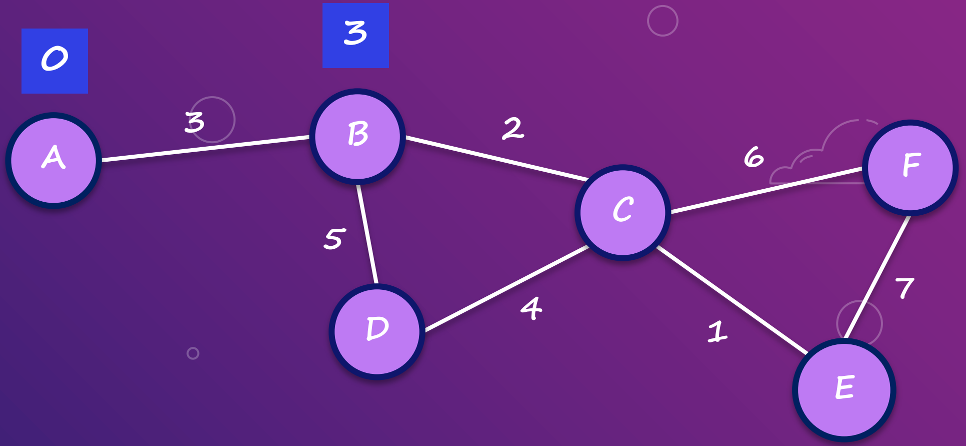
- Find the shortest path between two nodes (A---E)



D = [0, 3, ∞, ∞, ∞, ∞]
Q = {B, C, D, E, F}

DIJKSTRA'S ALGORITHM

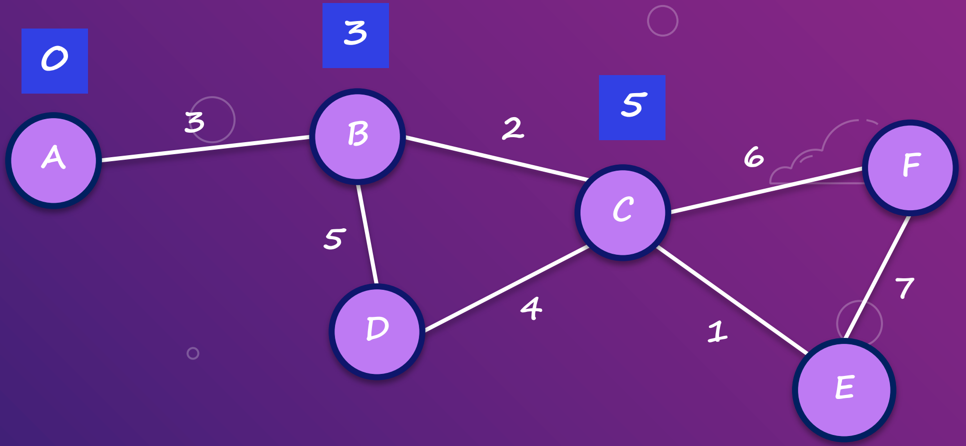
- Find the shortest path between two nodes (A---E)



D = [0, 3, (3+2) or (3+5+4), ∞, ∞, ∞]
Q = {B, C, D, E, F}

DIJKSTRA'S ALGORITHM

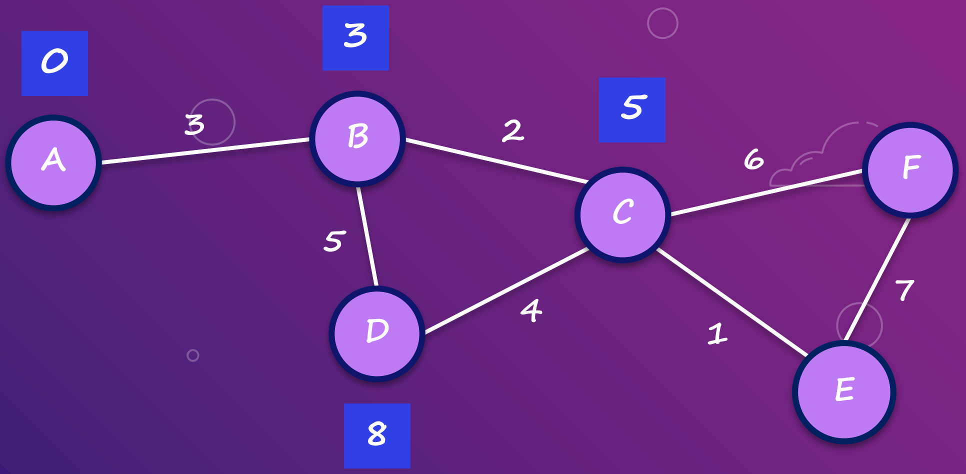
- Find the shortest path between two nodes (A---E)



D = [0, 3, 5, ∞, ∞, ∞]
Q = {B, C, D, E, F}

DIJKSTRA'S ALGORITHM

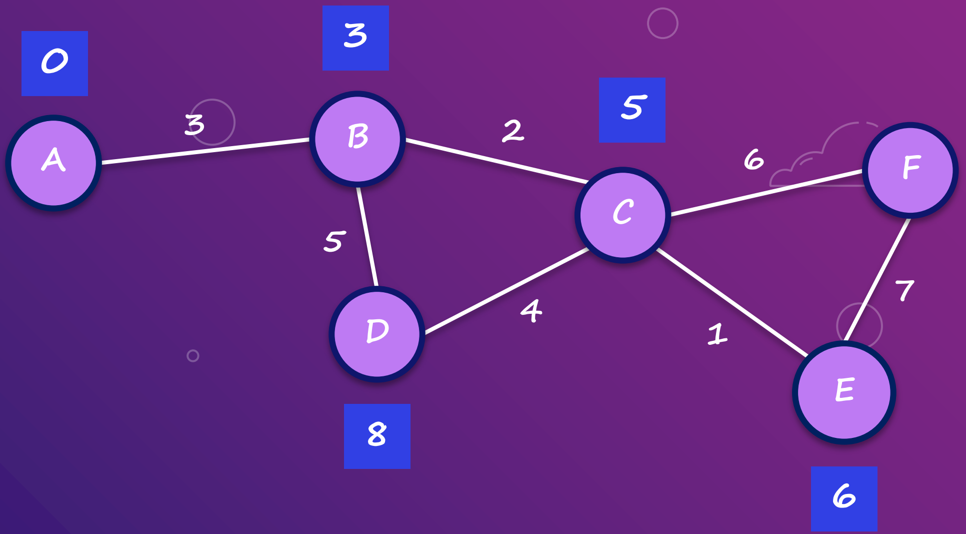
- Find the shortest path between two nodes (A---E)



D = [0, 3, 5, 8, ∞, ∞]
Q = {C, D, E, F}

DIJKSTRA'S ALGORITHM

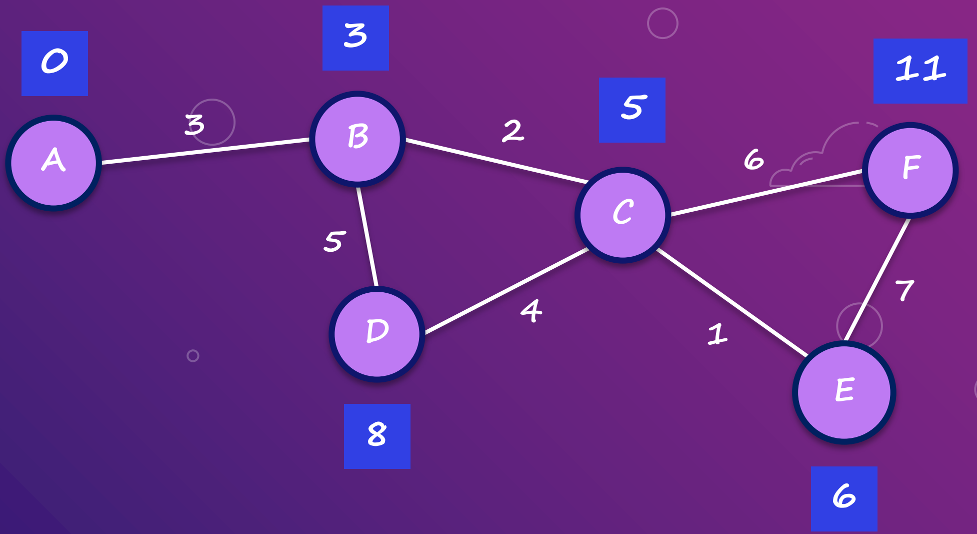
- Find the shortest path between two nodes (A---E)



D = [0, 3, 5, 8, 6, ∞]
Q = {C, D, E, F}

DIJKSTRA'S ALGORITHM

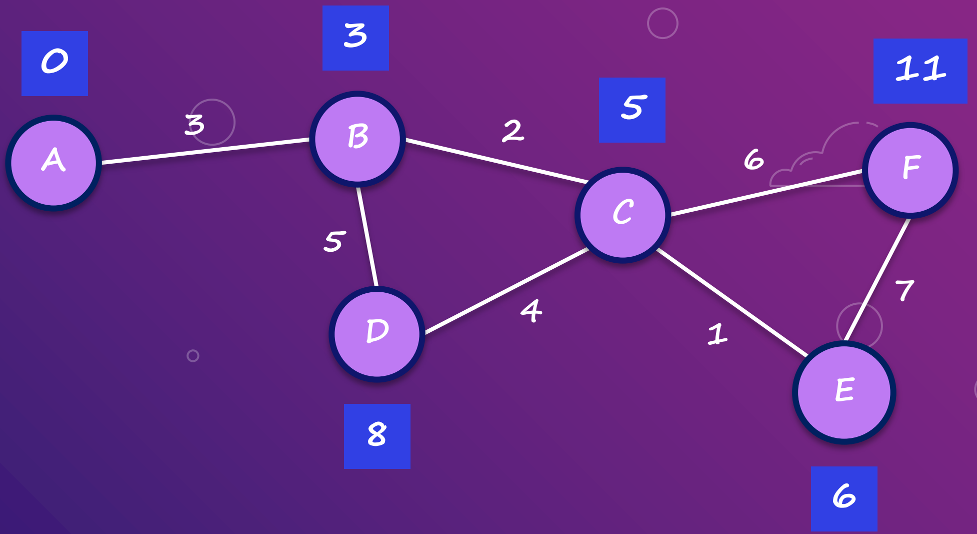
- Find the shortest path between two nodes (A---E)



D = [0, 3, 5, 8, 6, 11]
Q = {D, E, F}

DIJKSTRA'S ALGORITHM

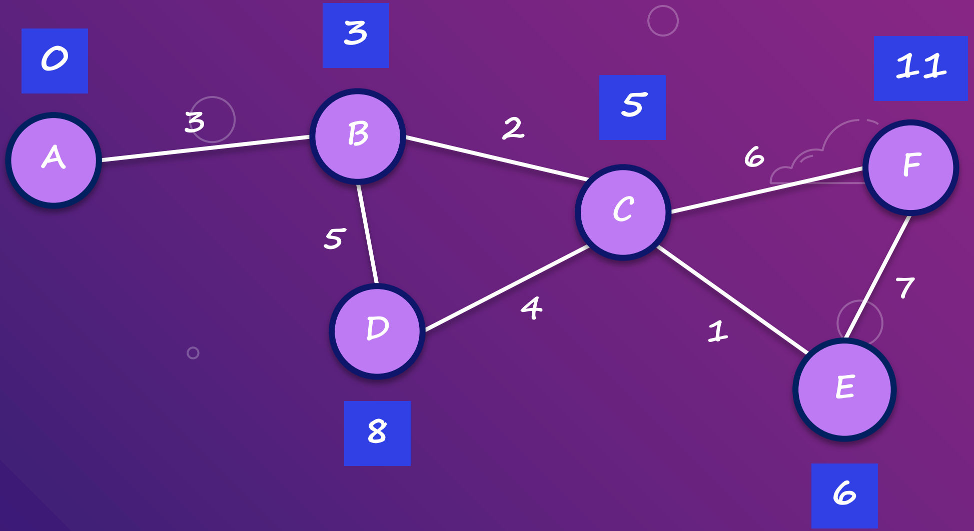
- Find the shortest path between two nodes (A---E)



D = [0, 3, 5, 8, 6, 11]
Q = {E, F}

DIJKSTRA'S ALGORITHM

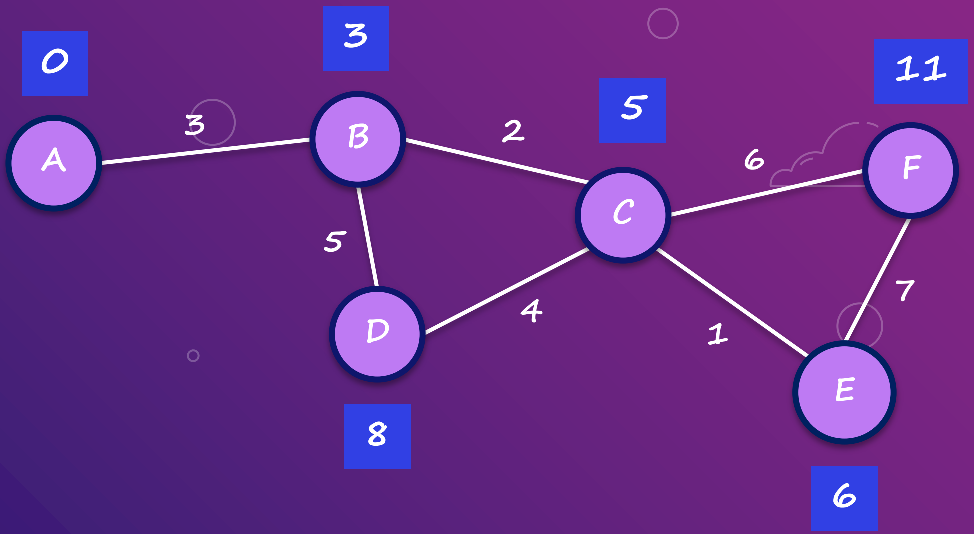
- Find the shortest path between two nodes (A---E)



D = [0, 3, 5, 8, 6, 11]
Q = {F}

DIJKSTRA'S ALGORITHM

- Find the shortest path between two nodes (A---E)



D = [0, 3, 5, 8, 6, 11]
Q = {}

PAGERANK ALGORITHM

- Used by Google to rank web pages
- It is a way of measuring the importance of web pages
- A page is only as important as the pages that link to it
- According to Google:

"PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites."

PAGERANK ALGORITHM

- The algorithm measures the importance of each node within the graph, using two factors:
  - Number of incoming relationships
  - Importance of the corresponding source nodes

$$PR(A) = (1 - d) + d\left(\frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)}\right)$$

where,
T_1 to T_n are pages that points to page A
d is a damping factor, which can be set between 0 (inclusive) and 1 (exclusive), usually set to 0.85
C(A) is defined as the number of links going out of page A

GIRVAN-NEWMAN ALGORITHM

- One of the first methods of “modern” community detection
- The idea is to recursively partition the network by removing edges, groups that are last to be partitioned are “communities”
- A community, with respect to graphs, is a subset of nodes that are densely connected to each other and loosely connected to the nodes in the other communities in the same graph
- Detecting communities in a network is one of the most important tasks in network analysis
  - Compute “betweenness” of edges in the network = number of shortest paths that pass through each edge
  - Remove edge(s) with highest betweenness, if this breaks the graph into subgraphs, recursively partition each one
- Result is a hierarchical partitioning of the graph

GRAPHS IN PYTHON

- NetworkX is the Python library for dealing with graphs
  - https://networkx.github.io/
- Simple Python interface for constructing graph, querying information, and running a large suite of algorithms
- Not suitable for large graphs

```
import networkx as nx

sG = nx.Graph()
dG = nx.DiGraph()

sG.add_node("A")
dG.add_node(”B")

sG.add_edge("A", "B")
dG.add_edges_from([("A", "B"), ("B", "C"), ("C", "A")])

sG.remove_edge("A", "B")
dG.remove_edges_from([("A", "B"), ("B", "C")])

sG.remove_node("A")
dG.remove_nodes_from([("A", "B")])
```

GRAPHS IN PYTHON

```
nx.draw(sG, with_labels=True)

print sG["C"]

for i in dG.edges():
    print i

sG.nodes()


nx.shortest_path_length(sG, source="A")

nx.pagerank(sG, alpha=0.9)

nx.girvan_newman(sG)
```

THANKS