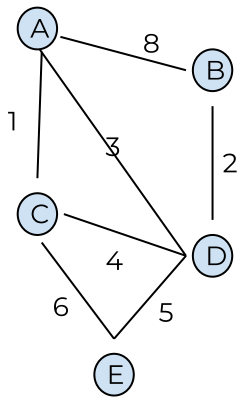
Chapter 4: Graph Algorithms (Needham & Hodler, 2019):
Pathfinding and Graph Search

Pathfinding algorithms: built on top of graph search algorithms; explore routes between nodes; used to identify optimal routes through a graph for logistics planning, least cost call or IP routing and gaming simulation

Examples

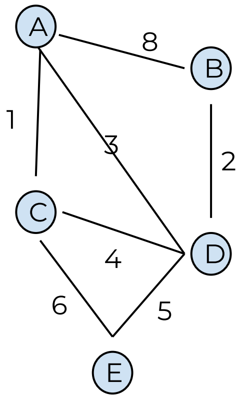
1. Shortest Path (A* and Yen's) - find the shortest path between two nodes
   a. Calculated by relationship/edge weights



Starting at A,
the shortest path to B would be:
A->D (3)
D->B (2)
Total weight = 5

This is the lowest weight to get from A to B.
Going directly A->B costs 8
Going from A->C->D->B costs a total of 7

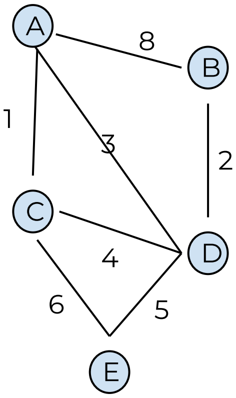
2. All Pairs Shortest Path and Single Source Shortest Path - for finding the shortest path between all pairs or from a chosen node to all others
   a. All Pairs: optimized calculation of shortest paths from all nodes to all other nodes



All Pairs (matrix representation, using shortest path combo):

	A	**B**	**C**	**D**	**E**
A	0	8	1	3	7
B	8	0	6	2	7
C	1	6	0	4	6
D	3	2	4	0	5
E	7	7	6	5	0

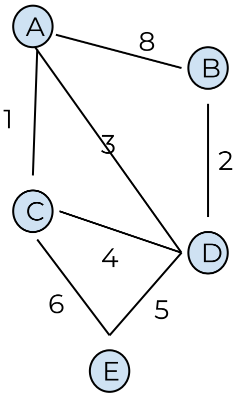
   b. Single Source: shortest path from a route node to all other nodes (accumulating the least weight)



Shortest path from a root node (A), to all other nodes via lowest cumulative weight:

ADB (5) + ACE (7) = 12
A score of 12 to visit all nodes (ABCDE)

3. Minimum Spanning Tree - find a connected tree structure with the smallest cost for visiting all nodes from a chosen node by starting at a given node and traversing ALL nodes via lowest-weight paths



Shortest path with min. weight connecting all the nodes

Starting from B:
B -> D (2) + D -> A (3) + A -> C (1) + C -> E (6)
Total = 12 to hit every node once
Going from D to E is “cheaper”, but then leaving E to go back to E or to C adds substantially, so we only want to go to E once, if possible. We would never go from A to B or B to A directly.

Resulting “Tree”:

```
B
|
D
|
A
|
C
|
E
```

4. Random Walk - useful pre-processing / sampling step for ML workflows and other graph algorithms, select which direction to go / how to traverse randomly; aka drunkard’s walk

See Jupyter Notebook on my GitHub exploring the phenomenon of Random Walks and the CLT in python:

https://github.com/krashr-ds/framingham-ms/blob/main/Random%20Walks.ipynb

Explore a graph for:

- General discovery
- Explicit search

NOT necessarily computationally optimal

Breadth Search

Traverse across nodes at a level before descending to the next sub-level

Depth Search

Traverse down to the bottom of a subtree before moving to the next child or sibling node