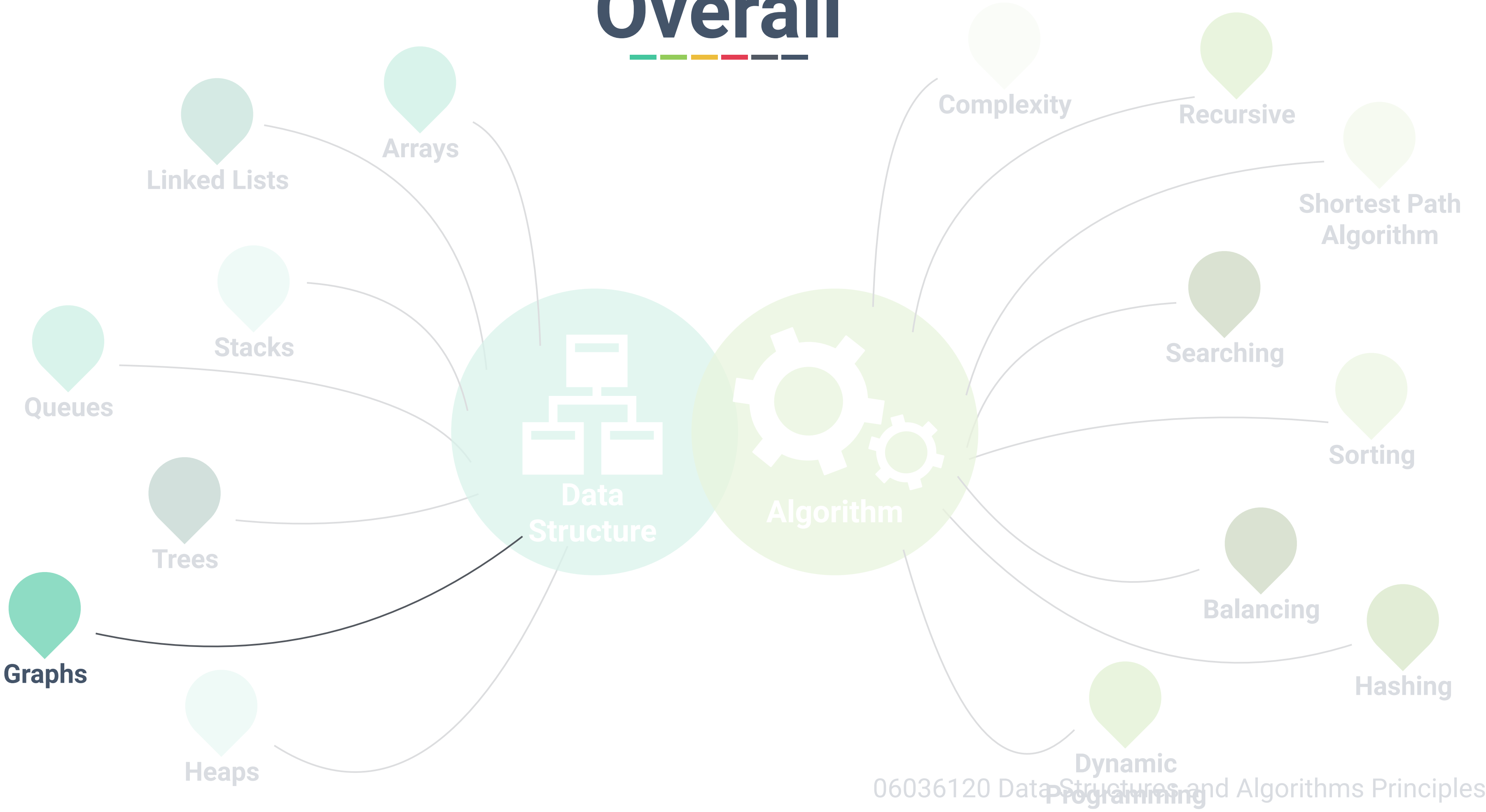
Faculty of
Information Technology
King Mongkut's Institute of Technology Ladkrabang

Chapter 11: Graph Algorithms

Dr. Sirasit Lochanachit

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Overall



Outline

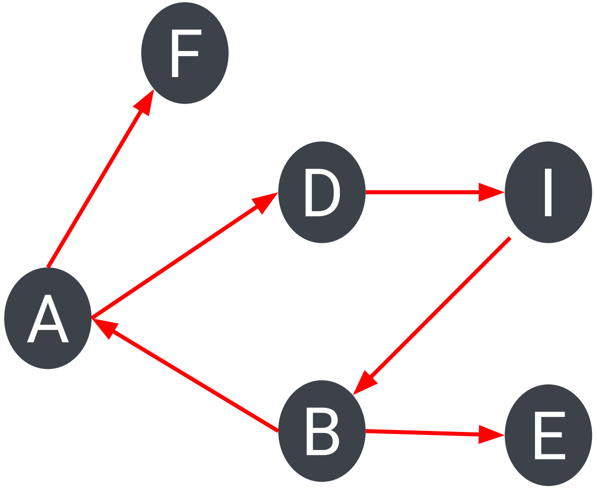
Graphs

- Definition, elements and types
- Graph Representation

Graph Algorithms

- Traversal
  - Depth-first
  - Breadth-first
- Shortest Path

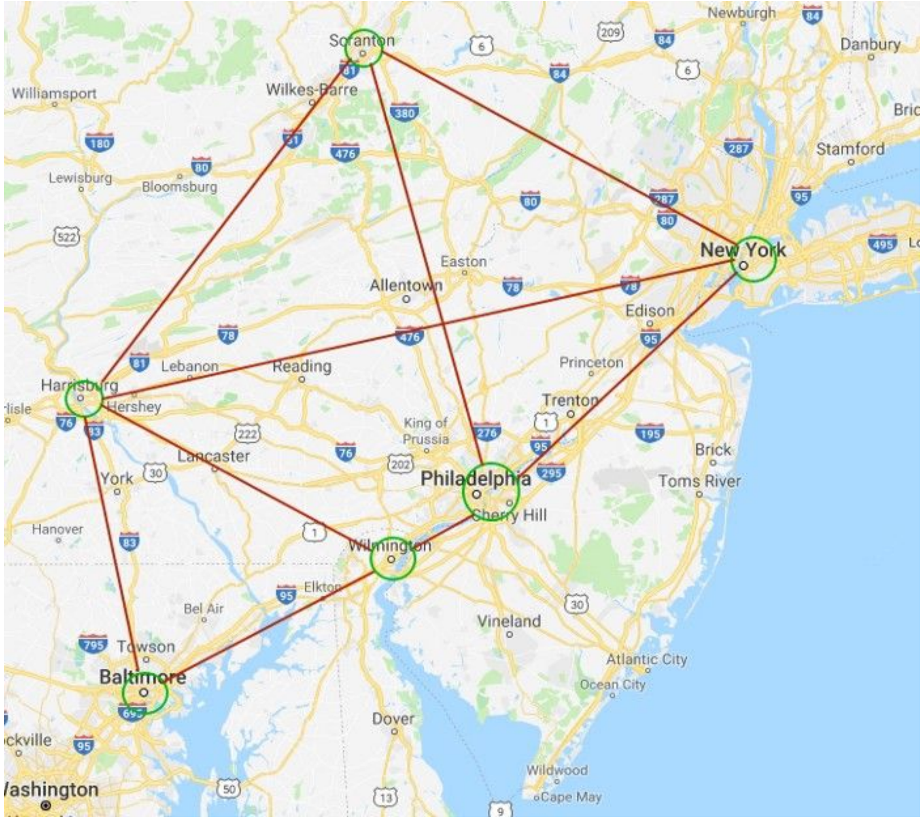
What is a Graph?

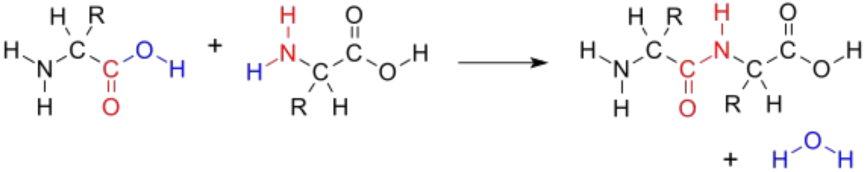


- A **graph** is a set of objects, called vertices or **nodes**, where the actual data is stored and a collection of connections between them, called **edges** or arcs[1].
- A graph can be used to represent relationships between pairs of objects.

[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

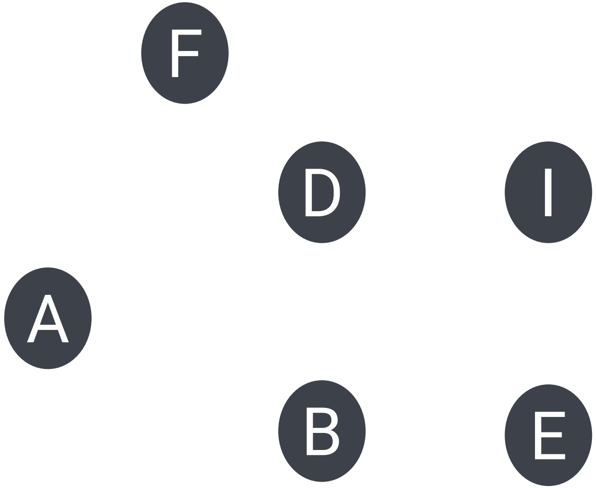
Graphs



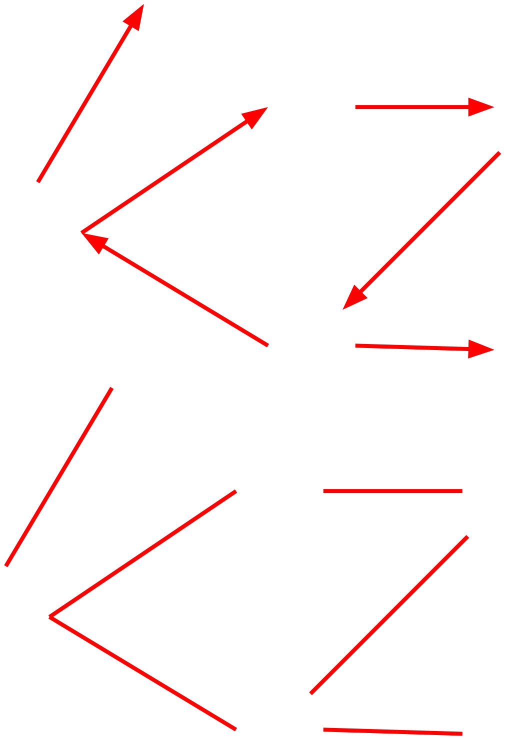


Ref: https://miro.medium.com/max/700/1*2jL0bKbT8ttskAlEVDv1yg.jpeg

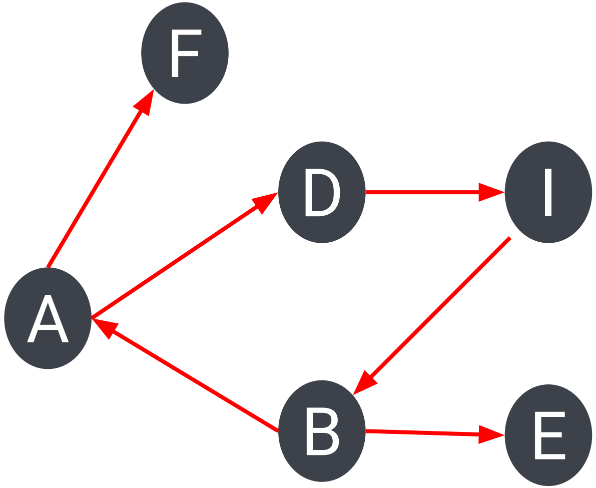
The Basic Elements of Graph



The Basic Elements of Graph



The Basic Elements of Graph



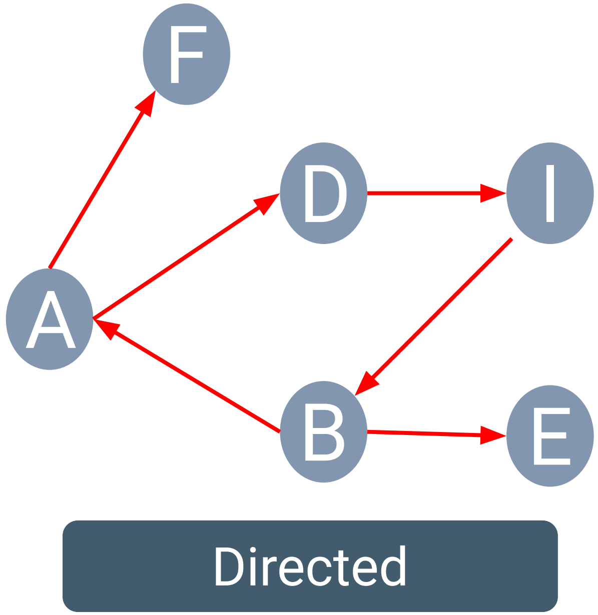
- Formally, a graph *G* is a set *V* of vertices and a collection *E* of pairs of vertices, called edges[1].

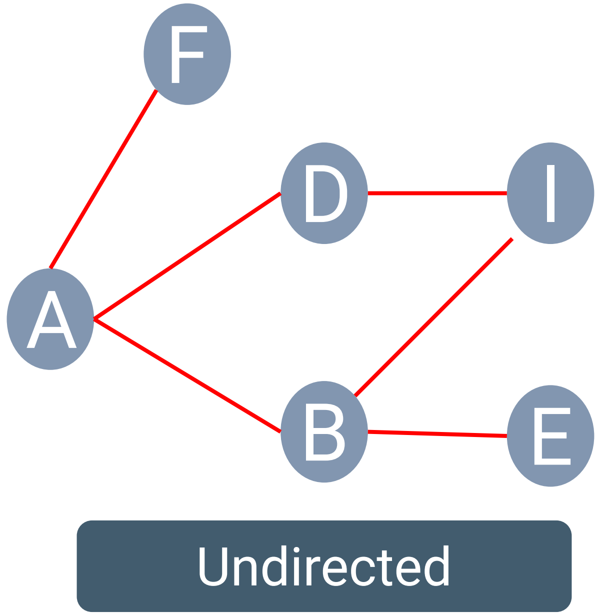
V = {A, B, C, D, E, F, I}

E = {(A, F), (A, D), (B, A), (D, I), (I, B), (B, E)}

[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

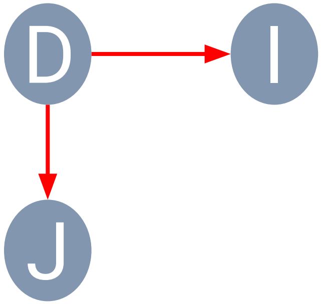
Graph Types





- An edge (u, v) is directed from u to v if the pair (u, v) is ordered.
- An edge (u, v) is undirected if the pair (u, v) is not ordered.

Graph Terminology

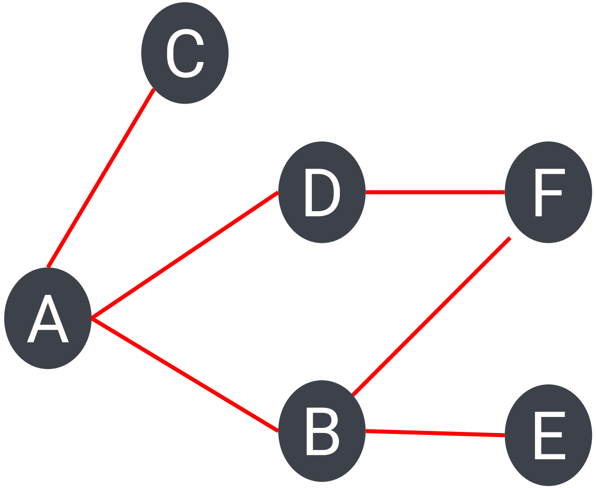


V = {D, I, J}

E = {(D, I), (D, J)}

- **Endpoints**: Two nodes (*u*, *v*) that are joined by an edge.
  - These two nodes are **adjacent**.
- **Origin**: First endpoint (*u*) on a directed edge.
- **Destination**: Second endpoint (*v*) on a directed edge.

Graph Representation

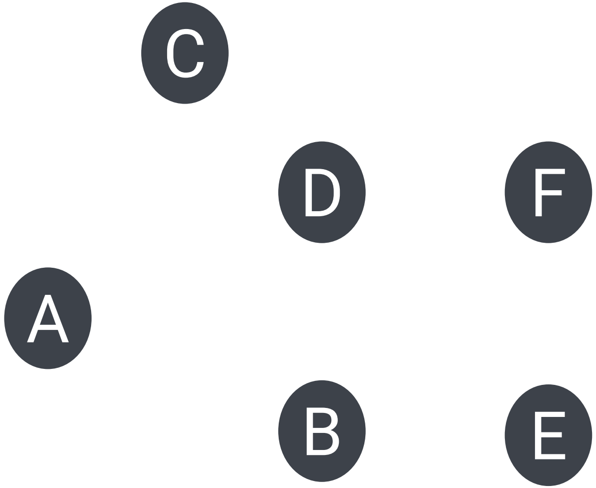


	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

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

Adjacency Matrix

Graph Representation

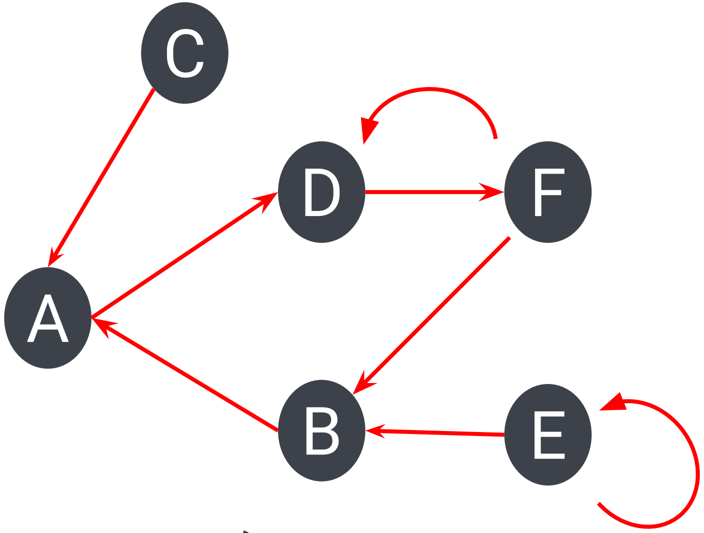


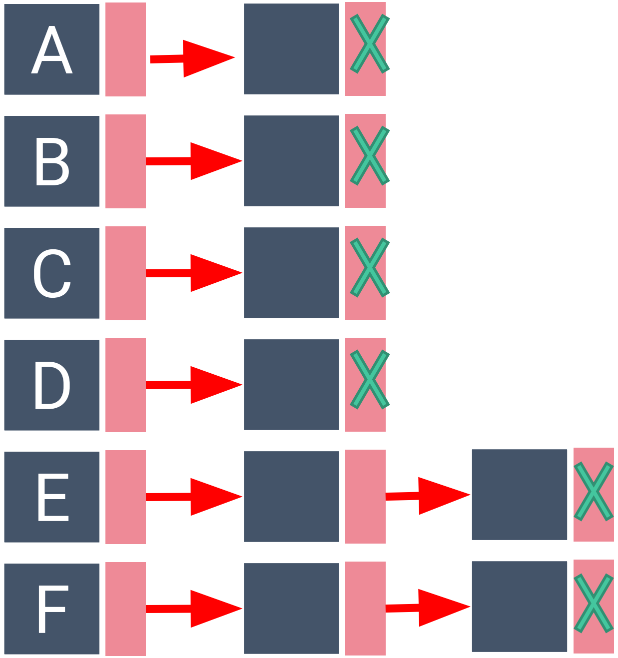
- V = {A, B, C, D, E, F}
- E =

	A	B	C	D	E	F
A	0	0	0	1	0	0
B	1	0	0	0	0	0
C	1	0	0	0	0	0
D	0	0	0	0	0	1
E	0	1	0	0	1	0
F	0	1	0	1	0	0

Adjacency Matrix

Graph Representation





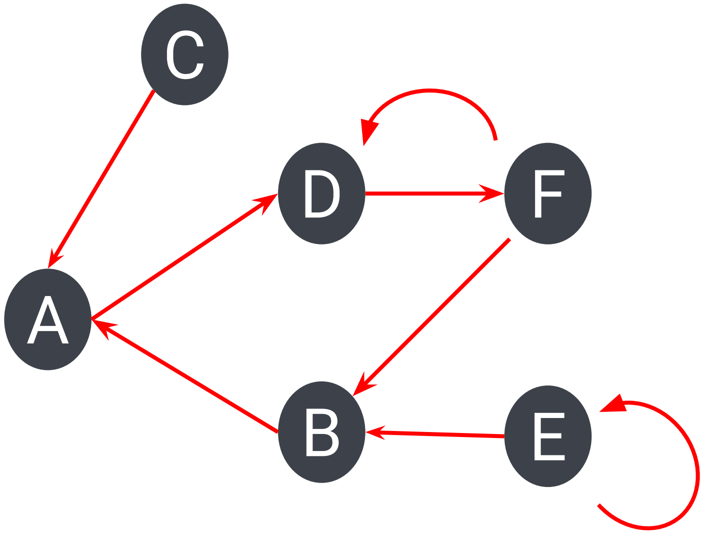
- V = {A, B, C, D, E, F}
- E = {(C, A), (A, D), (B, A), (D, F), (F, D), (E, B), (F, B), (E, E)}

Adjacency List

Graph Representation

Node Representation

Node	Name	Phone
A	Able	
B	Baker	
C	Charlie	
D	Denver	
E	Ethan	
F	Fred	

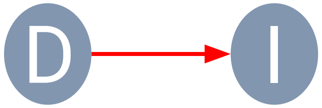


Edge Representation

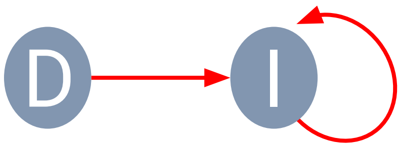
	A	B	C	D	E	F
A	0	0	0	1	0	0
B	1	0	0	0	0	0
C	1	0	0	0	0	0
D	0	0	0	0	0	1
E	0	1	0	0	1	0
F	0	1	0	1	0	0

Adjacency Matrix

Graph Terminology



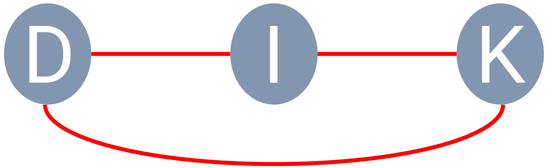
- A **path** is a sequence of nodes and edges that starts at a node and ends at a node such that each node is adjacent to the next one[2].
- Formally, a path is a sequence of nodes $V_1, V_2, V_3, \ldots, V_n$ where $(V_1, V_2), (V_2, V_3), \ldots, (V_{n-1}, V_n) \in E$.
- A **loop** is a special case of path where two endpoints are the same.
    - An edge that starts and ends with the same node.



[2] Bradley N. Miller, David L. Ranum, Problem Solving with Algorithms and Data Structures using Python, 2011

Graph Properties

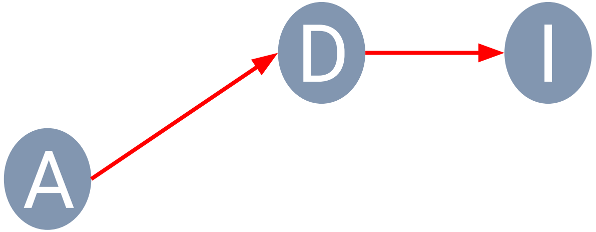
- A **cycle** is a path that starts and ends at the same node, having at least one edge.
- A **simple path** is a path that does not contain the same edge more than once.
- A **simple cycle** is a simple path that starts and ends at the same node.



Graph Properties Example

Simple?

Cyclic?

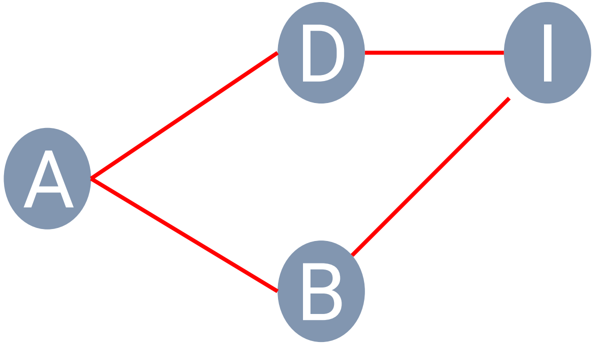


A, D, I

Graph Properties Example

Simple?

Cyclic?

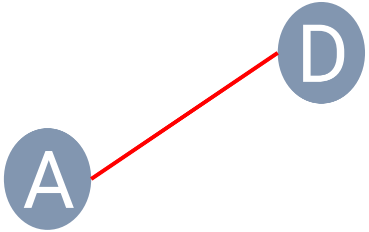


A, D, I, B, A

Graph Properties Example

Simple?

Cyclic?

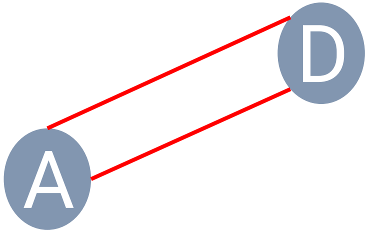


A, B, A

Graph Properties Example

Simple?

Cyclic?

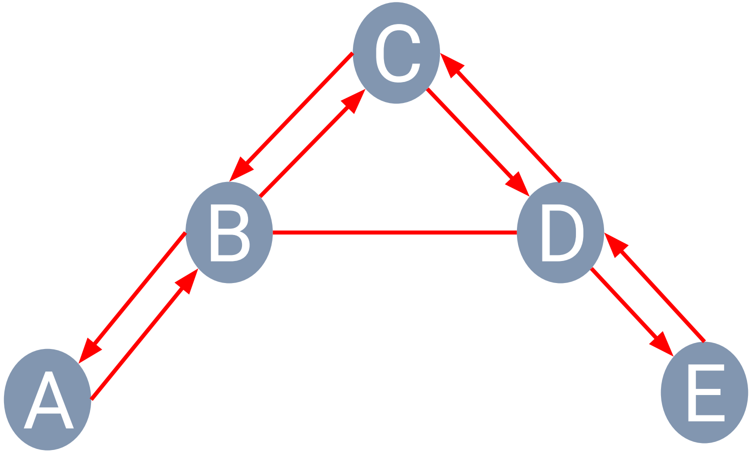


A, D, A

Graph Properties Example

Simple?

Cyclic?

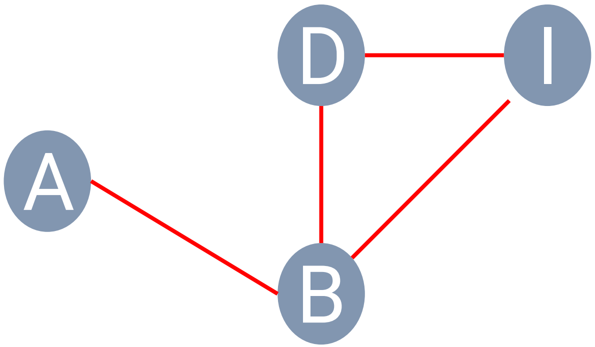


A, B, C, D, E, D, B, A

Graph Properties Example

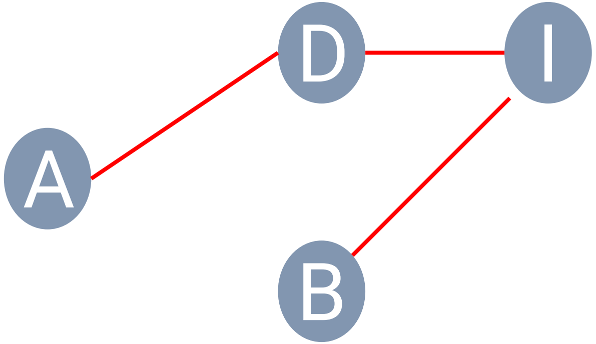
Simple?

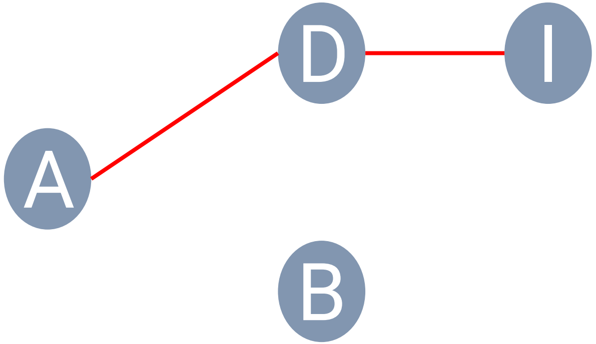
Cyclic?



A, B, I, D, B, A

Graph Notations





- A graph is **connected** if, for any two nodes, there is a path between them.
- The **in-degree** of a node *v* is the number of the incoming edges of *v*.
- The **out-degree** of a node *v* is the number of the outgoing edges of *v*.

Graph Algorithms

- Traversals
  - Depth-first traversal
  - Breadth-first traversal
- Minimum Spanning Tree
  - Prim-Jarnik Algorithm
  - Kruskal's Algorithm
- Shortest Path
  - Dijkstra Algorithm
- Etc.

Graph Traversals

- A **traversal** is a systematic procedure for exploring a graph by examining all of its nodes and edges.
- Graph traversal algorithms are key to answering many fundamental questions about graphs involving the notion of **reachability**, that is, in determining how to travel from one node to another while following paths of a graph[1].
- Two efficient graph traversal algorithms: **depth-first traversal** and **breadth-first traversal**.

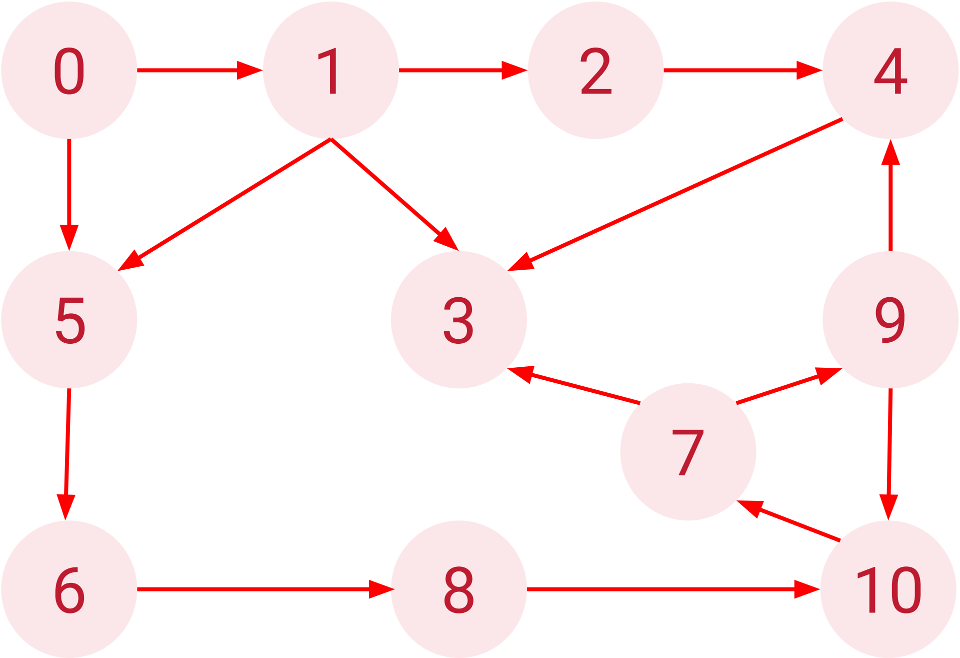
[1] Michael T. Goodrich et al., Data Structures and Algorithms in Python, 2013

Graph Traversals

Depth-first traversal

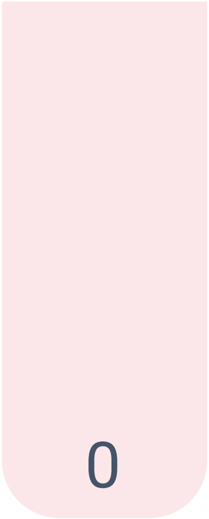
Breadth-first traversal

Depth-First Traversal with Stack



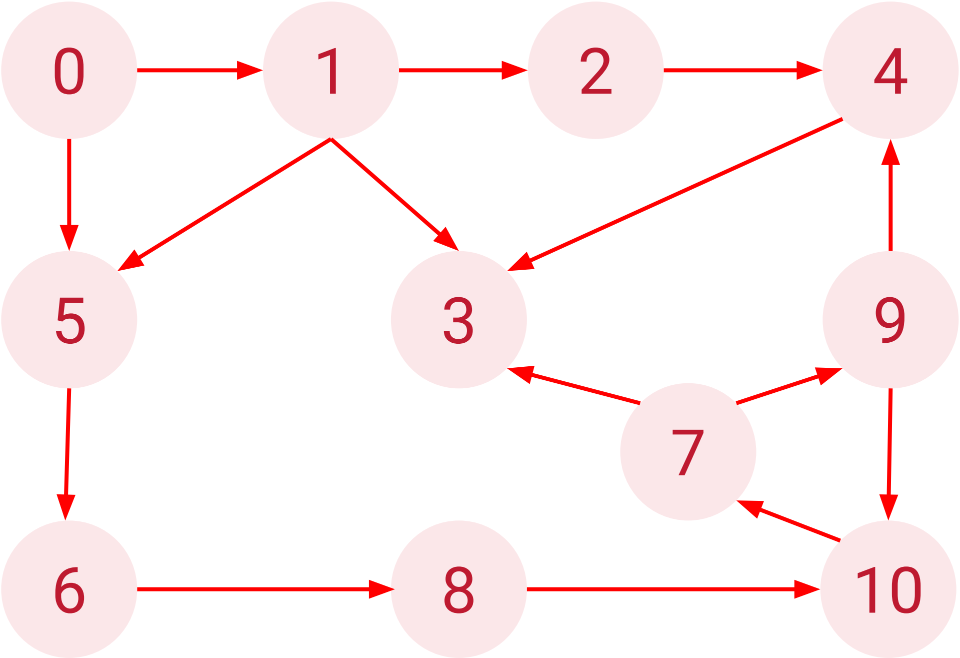
Output :

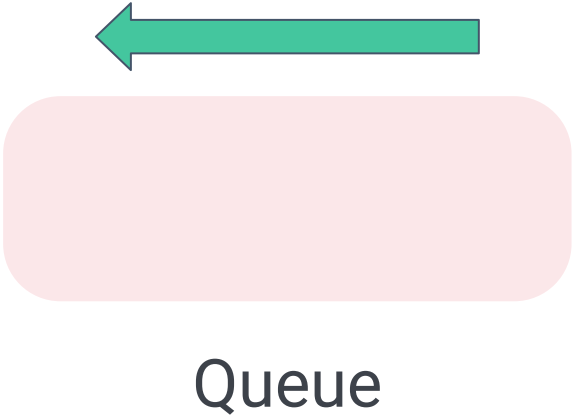
Depth-First Traversal with Stack



Output : 0

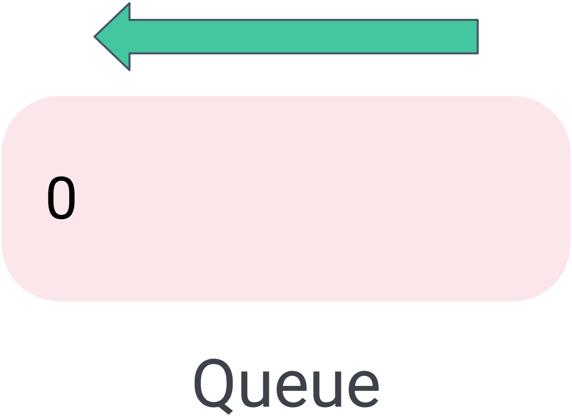
Breadth-First Traversal with Queue





Output :

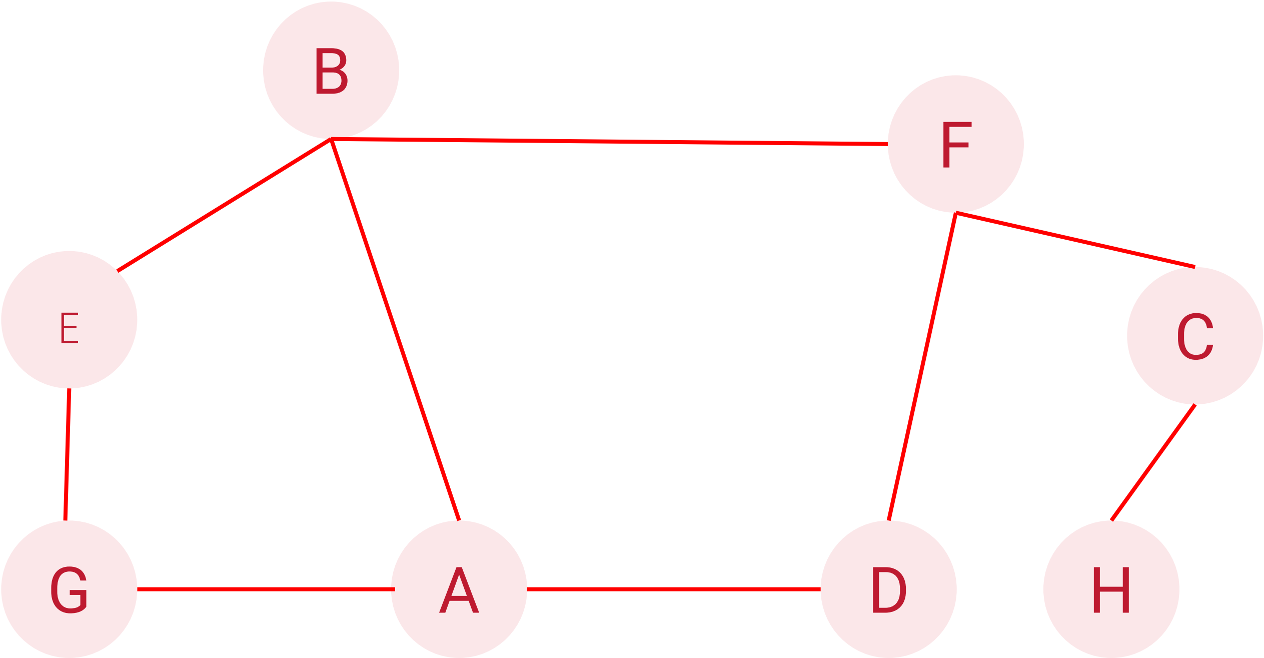
Breadth-First Traversal with Queue



Output :

Depth-First and Breadth-First Exercise

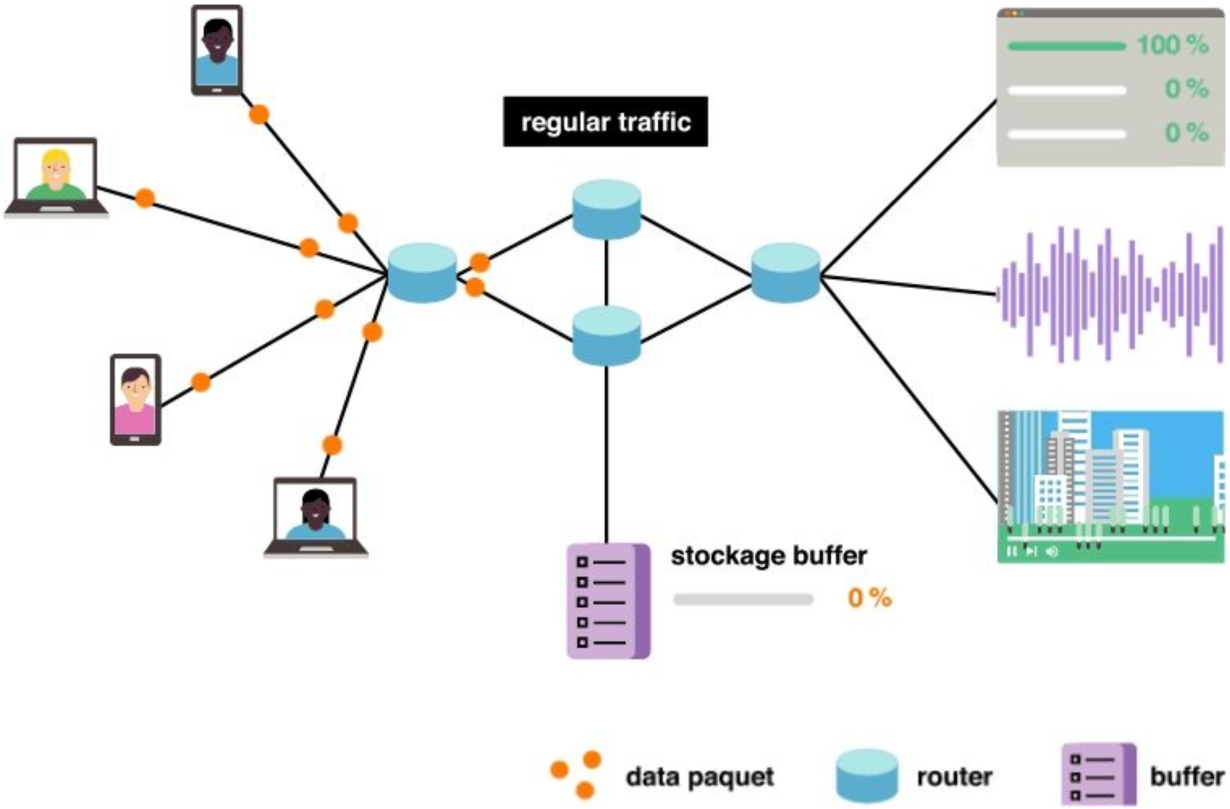
Rule: Access node in ascending order (A-Z)



Graph Traversals

- On undirected and directed graph with n nodes and m edges.
  - A DFS traversal can be performed in $O(n + m)$ time.
  - A BFS traversal can be conducted in $O(n + m)$ time

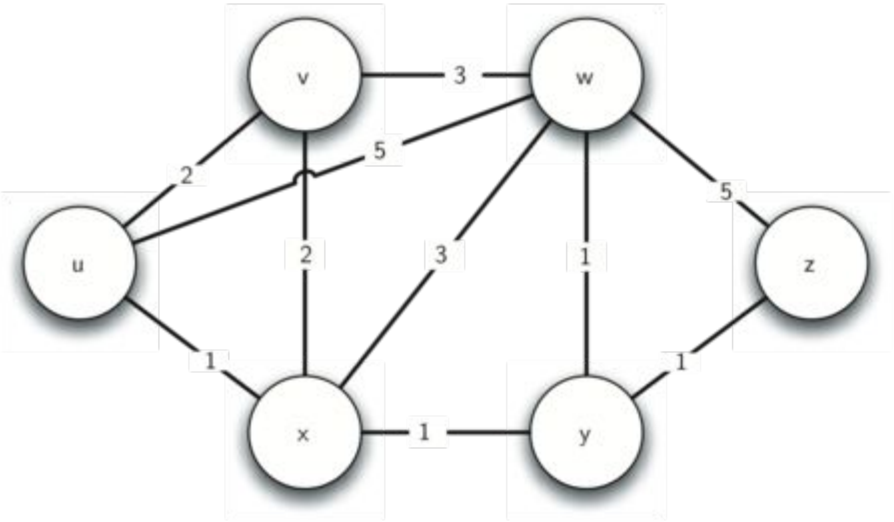
Shortest Path Problem



Ref: https://hellofuture.orange.com/en/telecoms-networks-well-equipped-to-deal-with-traffic-peaks/

Shortest Path Problem

- The network of routers can be represented as a graph with weighted edges.

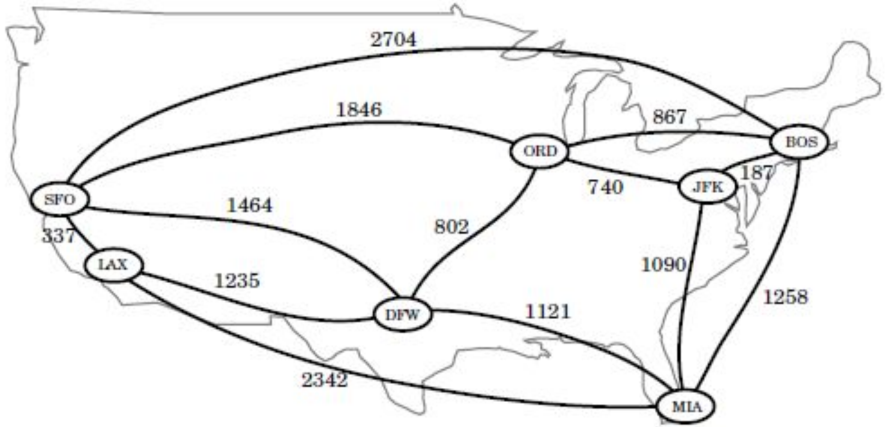


Shortest Path Problem

- The breadth first strategy can be used to find a **shortest path** from some starting node to every other node in a connected graph.
  - This approach is suitable in cases where each edge is equal to others.
  - However, for other situations, this approach is not efficient.
- It is natural, therefore, to consider graphs whose edges are <u>not weighted equally</u>.

Weighted Graphs

- A **weight graph** is a graph that has a numeric label *w*(*e*) associated with each edge *e*, called the **weight** of edge *e*.
- For $e = (u, v)$, $w(u, v) = w(e)$.
- Such weights might represent:
  - Costs
  - Lengths
  - Capacities
  - etc.



Defining Shortest Paths in a Weighted Graph

- Let *G* be a weighted graph.
- The **length** (or **weight**) of a **path** is the sum of the weights of the edges of *P*.
  - P = ((v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k))
  - Length of P, denoted w(P) is defined as $$w(P) = \sum_{i=0}^{k-1} w(v_i, v_{i+1}).$$
- The distance from a node *u* to a node *v* in *G*, denoted *d*(*u*, *v*) is the length of a minimum-length path (also called **shortest path**) from *u* to *v*.

Shortest Paths Algorithms

- Shortest path in a graph with all equal weights can be solved with breadth-first traversal algorithm.
- Distance cannot be arbitrarily low negative numbers.
  - For instance, the weight of edges represent the cost to travel between cities. If someone pay you to go between the cities, the cost would be negative.
  - Edge weights in *G* should be nonnegative (that is, $w(e) >= 0$) for each edge.

Dijkstra's Algorithm

- An iterative algorithm that provides the shortest path from one starting node to all other nodes in the graph[2].
- Apply **greedy method** to solve the problem by repeatedly selecting the best choice from among those available in each iteration.
  - Useful for optimising cost function over a collection of objects.
- "Weight" breadth-first search starting at the source node *s*.
- Used in link-state routing protocols in computer network.

[2] Bradley N. Miller, David L. Ranum, Problem Solving with Algorithms and Data Structures using Python, 2011

Dijkstra's Algorithm Variables

- dist[*v*] keeps the <u>shortest/minimum length</u> from the source node *s* for each node *v* in the graph.
  - Initially,
    - dist[*s*] = 0
    - dist[*v*] = Inf for each *v* != s
  - In practice, dist[v] can be set to a very large number than any real distance in the problem.

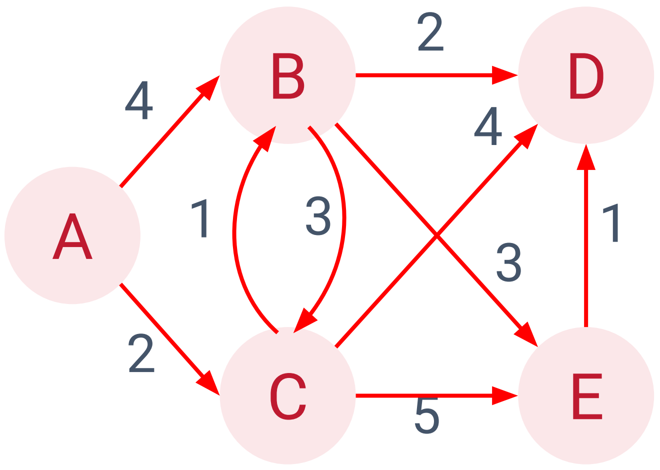
Dijkstra's Algorithm Variables

- *Q* is a set of all the unvisited nodes, called the <u>unvisited</u> set.
- *prev*[v] is used to keep track of the previous node that provides the shortest path from *s*.

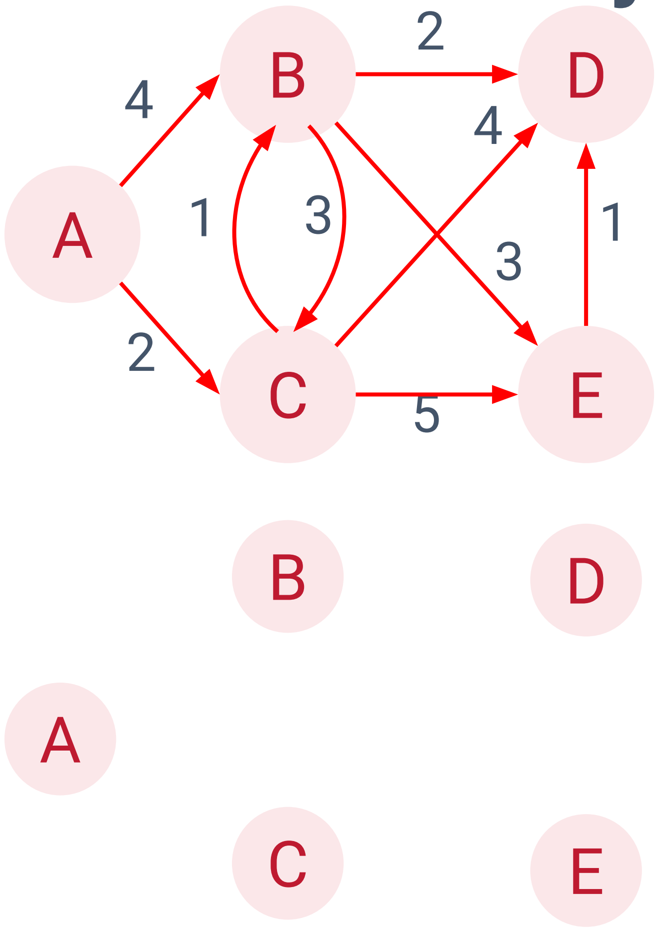
Dijkstra's Algorithm

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]

        remove u from Q

        for each neighbor v of u:           // only v
that are still in Q
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Dijkstra's Algorithm



Dijkstra's Algorithm



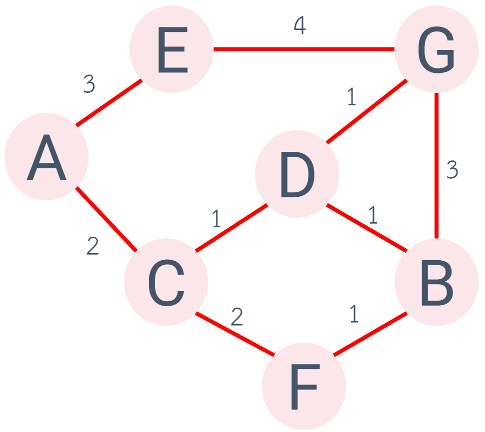
Q = {'A', 'B', 'C', 'D', 'E'}

dist = {'A': 0, 'B': ∞, 'C': ∞, 'D': ∞, 'E': ∞}

prev = {'A': None, 'B': None, 'C': None, 'D': None, 'E': None}

Node	Cumulative weight #1	Cumulative weight #2	Cumulative weight #3	Cumulative weight #4	Route
A					
B					
C					
D					
E					

Dijkstra’s Algorithm Exercise



Find the shortest path for each node from node A

Find the distance from A to F

Dijkstra’s Algorithm

- Dijkstra’s algorithm works only when the weights are all positive.
  - If there is a negative weight on one of the edges in the graph, the algorithm would never exit.
- Another problem is a complete representation of the graph must be presented for the algorithm to run.
  - Every router has a complete map of all the routers: Not practical.
- Other algorithms allow each router to discover the graph as they go.
  - For instance, distance vector routing algorithm (Computer Networks).
  - Each node computes best path without full view of graph, exchanging link information as they go.