SCC.121: Fundamentals of Computer Science Sorting, Trees and Graphs

Shortest Paths in Graphs

Today’s Lecture

Aim:

- Distances and **Shortest Paths** in graphs
- BFS in directed graphs
- Computing shortest paths in weighted graphs using **Dijkstra’s algorithm**

Distances and Shortest Paths in Graphs

Lancaster University

Distances and Shortest Paths in Graphs

<u>Edge weight/Distance:</u>
Time it takes to go from London to Birmingham

Distances and Shortest Paths in Graphs

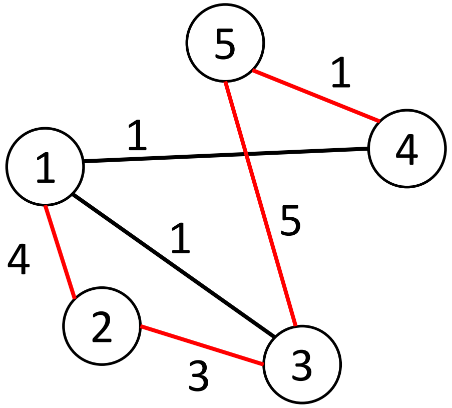
Red Path:
Shortest path from Manchester to Leeds

Distances and Shortest Paths in Graphs

Red Path:
Shortest path from Manchester to Newcastle

Paths

- Path between nodes u and v:
  - Sequence of (non-repeating) nodes: $u, w_1, w_2, \ldots, v$
  - In the example, **(1,2,3,5,4) is a path between 1 to 4**



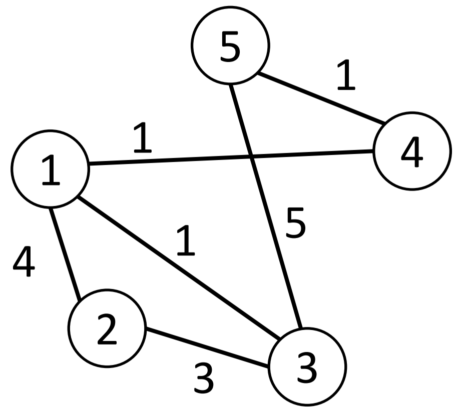
Shortest Paths

- **Shortest path** between nodes u and v = **among all paths between u and v, the path that has the smallest total edge weight**

- Example:
  - (1,2,3,5,4) is a path between 1 and 4 but not a shortest path
  - (1,4) is a shortest path between 1 and 4

  - (5,3) is a path between 5 and 3 but not a shortest path
  - (5,4,1,3) is a shortest path between 5 and 3

  - (1,2) is a shortest path between 1 and 2
  - (1,3,2) is **also** a shortest path between 1 and 2



Shortest Path Problems

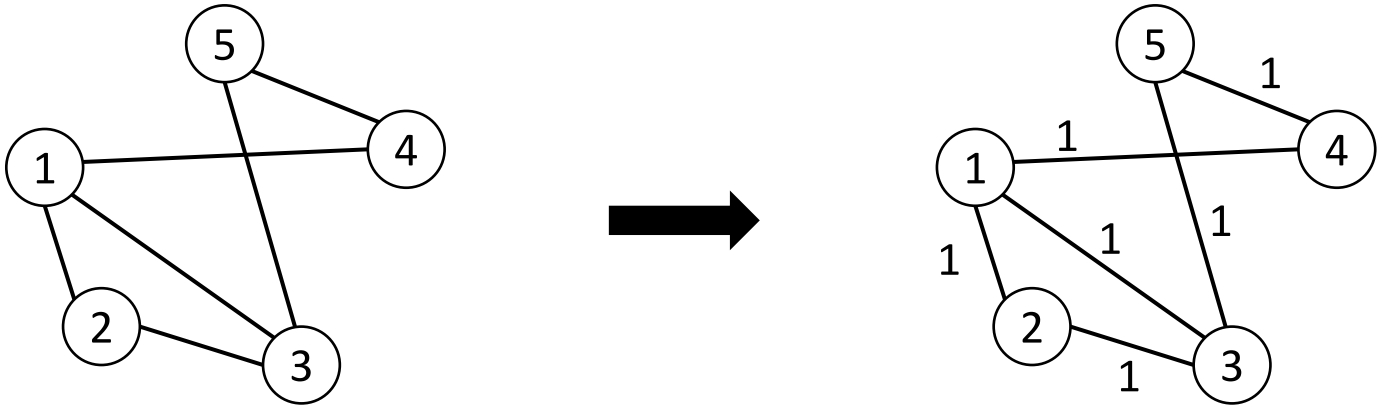
1. **Shortest distances (to source):**
   Given a graph $G = (V, E)$ and a source node s, compute the distance from all nodes $v \in V$ to s

2. **Shortest paths (to source):**
   Given a graph $G = (V, E)$ and a source node s, compute for each node $v \in V$ the shortest path from v to s.

3. **Pathfinding (from source to target):**
   Given a graph $G = (V, E)$, a source node s and a target node t, compute the shortest path from s to t.

Shortest Paths in Unweighted Graphs

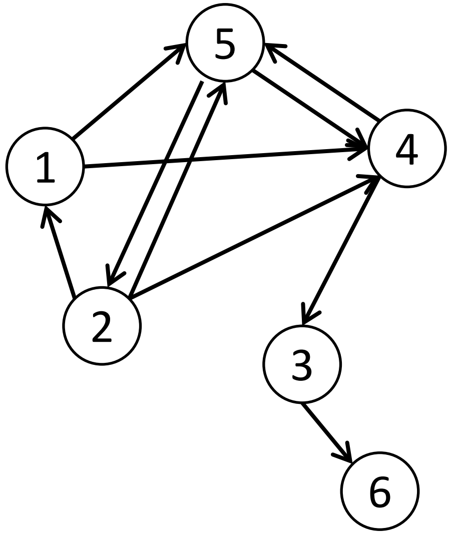
- Consider an unweighted graph G.



- **BFS search** can compute shortest paths when G is an **unweighted** graph
  - <u>Shortest path</u>: contains less edges than any other paths
  - Even on directed graphs.

BFS Traversal on Directed Graphs

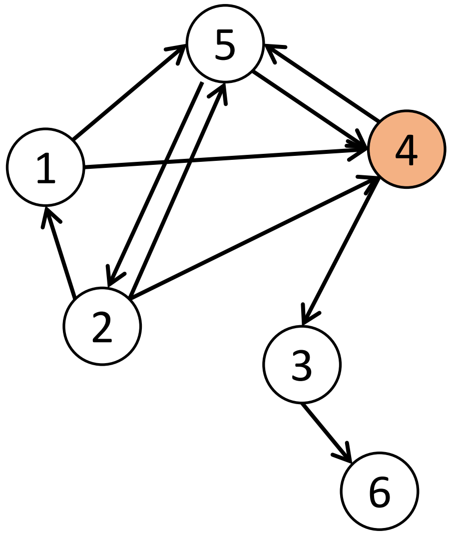
- Consider a directed, unweighted graph G.



	(1)	(2)	(3)	(4)	(5)	(6)
(1)	0	0	0	1	1	0
(2)	1	0	0	1	1	0
(3)	0	0	0	0	0	1
(4)	0	0	1	0	1	0
(5)	0	1	0	1	0	0
(6)	0	0	0	0	0	0

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

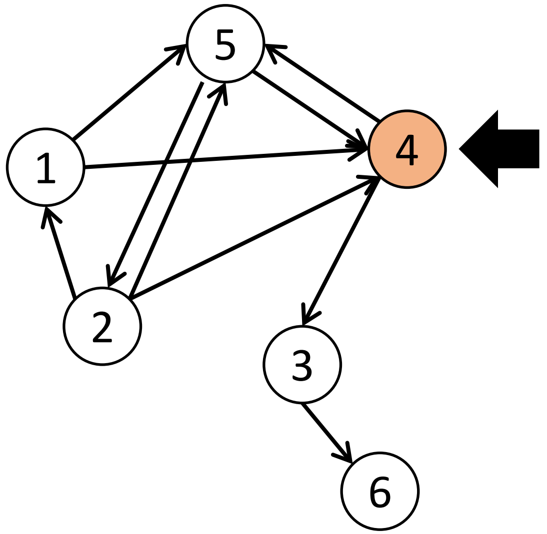
4

Traversal

Traversed edges

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

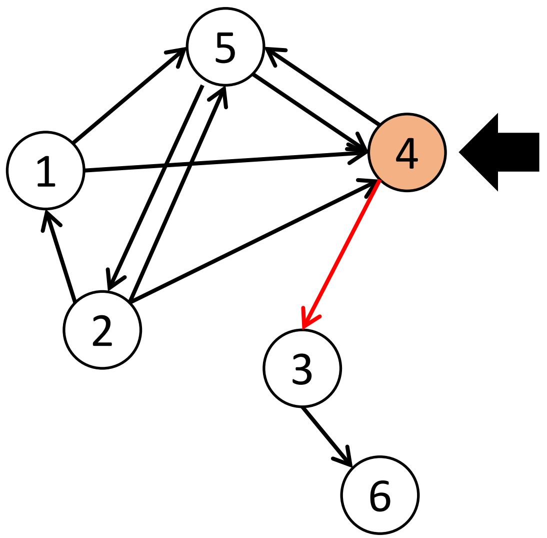
Traversal

4

Traversed edges

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

3

Traversal

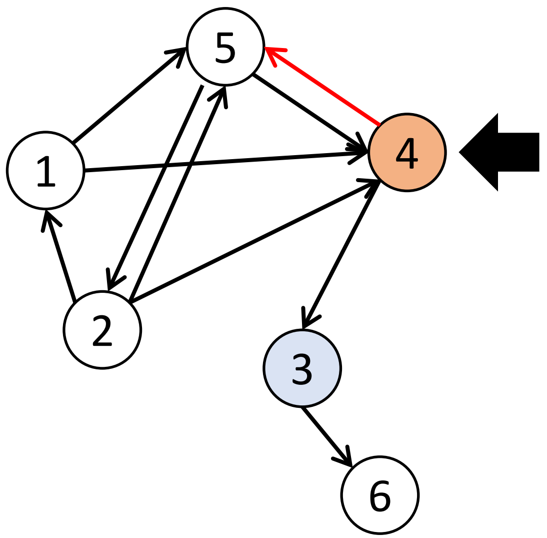
4, 3

Traversed edges

(4,3)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

5
3

Traversal

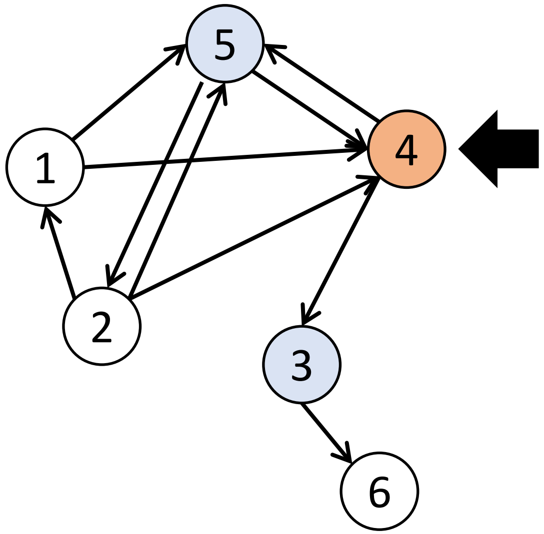
4, 3, 5

Traversed edges

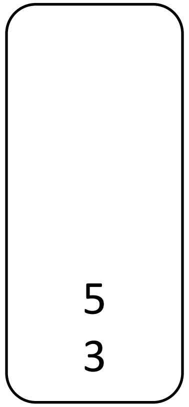
(4,3), (4,5)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue



Traversal

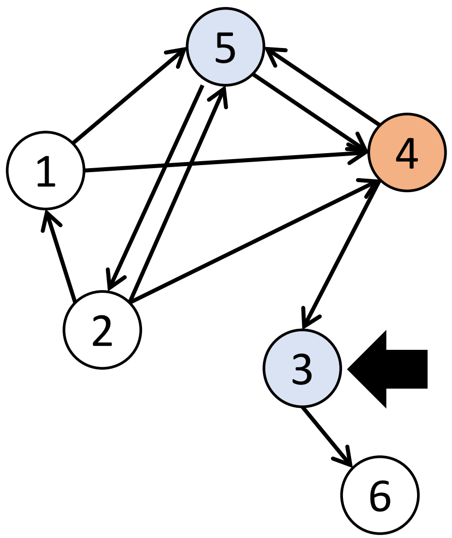
4, 3, 5

Traversed edges

(4,3), (4,5)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

5

Traversal

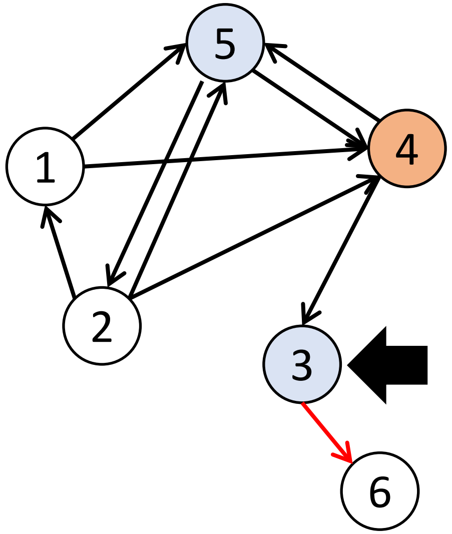
4, 3, 5

Traversed edges

(4,3), (4,5)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

6
5

Traversal

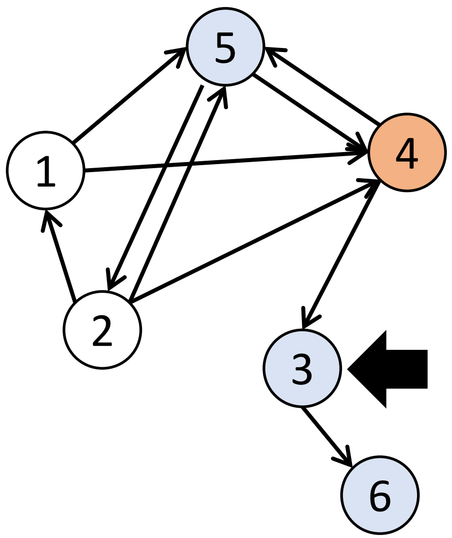
4, 3, 5, 6

Traversed edges

(4,3), (4,5), (3, 6)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

6
5

Traversal

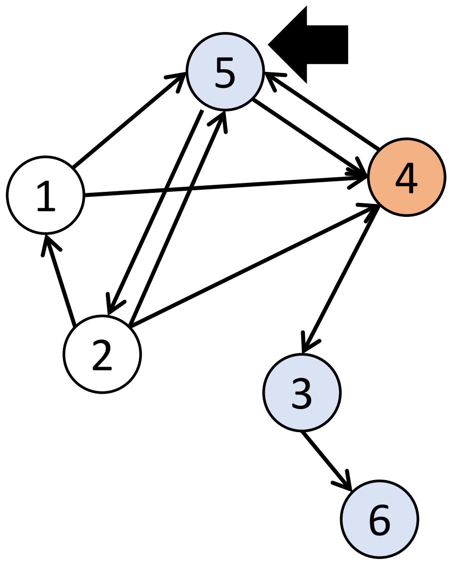
4, 3, 5, 6

Traversed edges

(4,3), (4,5), (3, 6)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

6

Traversal

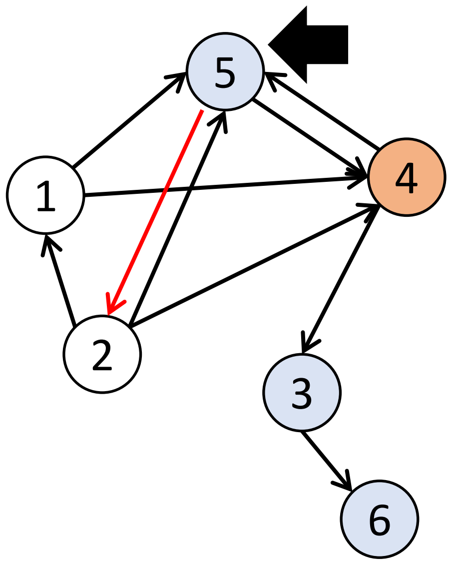
4, 3, 5, 6

Traversed edges

(4,3), (4,5), (3, 6)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

2
6

Traversal

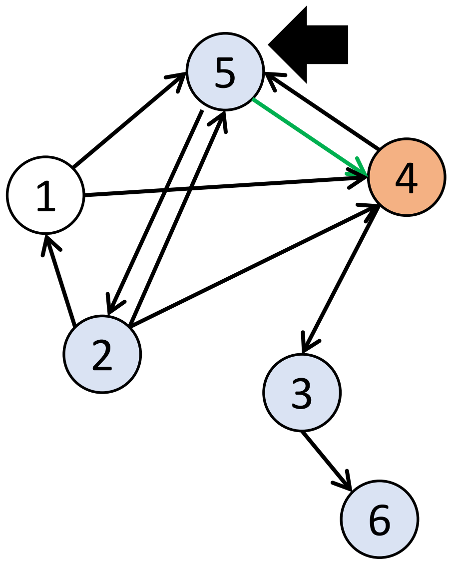
4, 3, 5, 6, 2

Traversed edges

(4,3), (4,5), (3, 6), (5,2)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

2
6

Traversal

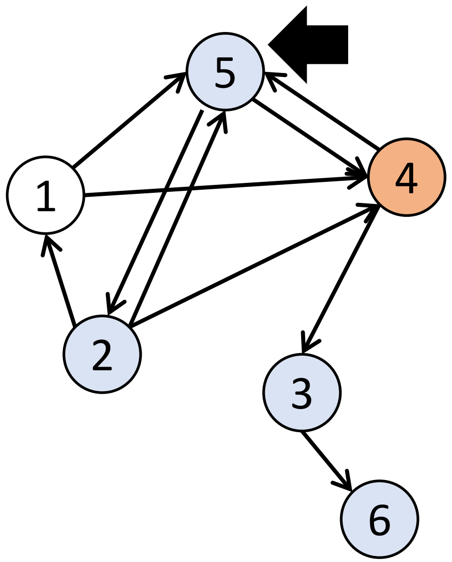
4, 3, 5, 6, 2

Traversed edges

(4,3), (4,5), (3, 6), (5,2)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

2
6

Traversal

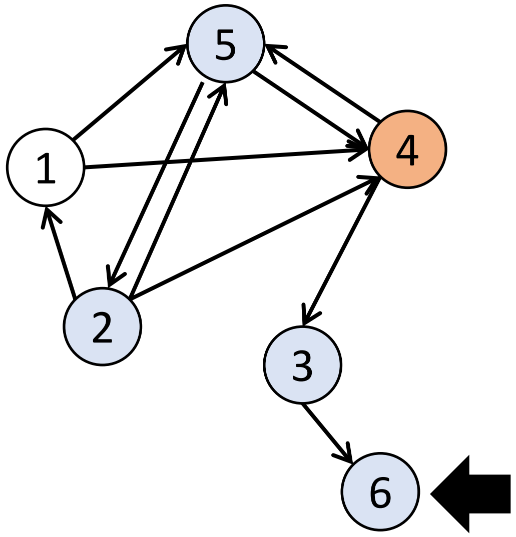
4, 3, 5, 6, 2

Traversed edges

(4,3), (4,5), (3, 6), (5,2)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

2

Traversal

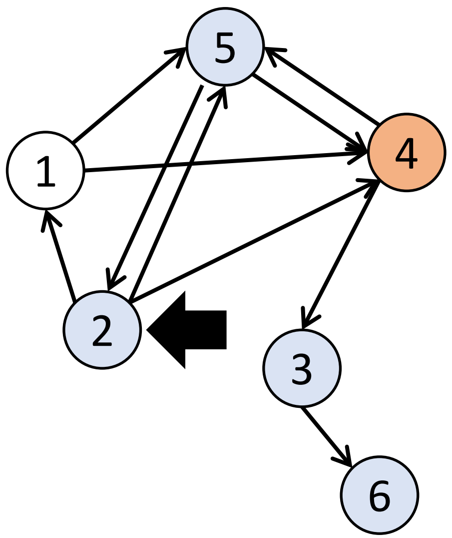
4, 3, 5, 6, 2

Traversed edges

(4,3), (4,5), (3, 6), (5,2)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

Traversal

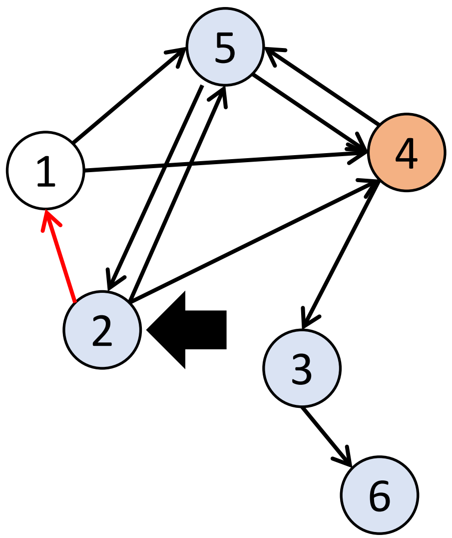
4, 3, 5, 6, 2

Traversed edges

(4,3), (4,5), (3, 6), (5,2)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

1

Traversal

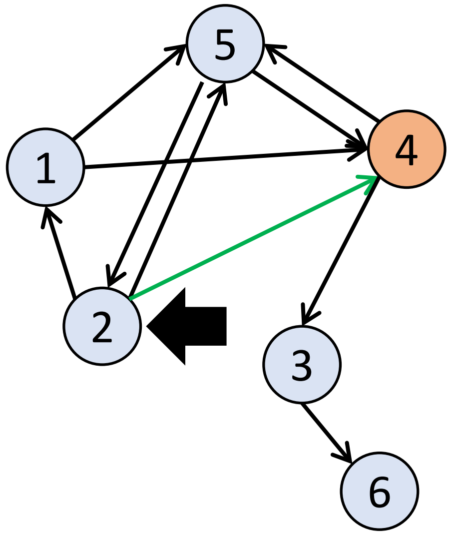
4, 3, 5, 6, 2, 1

Traversed edges

(4,3), (4,5), (3, 6), (5,2), (2,1)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

1

Traversal

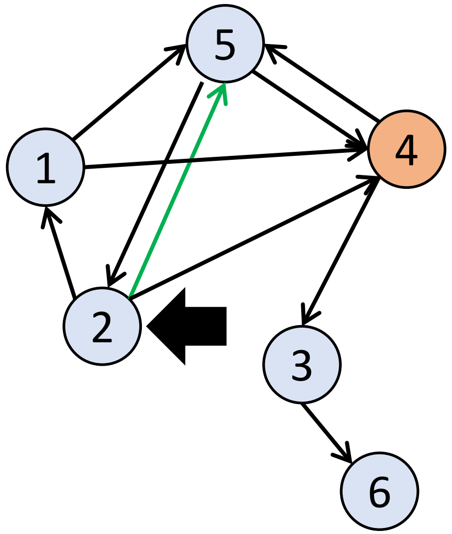
4, 3, 5, 6, 2, 1

Traversed edges

(4,3), (4,5), (3, 6), (5,2), (2,1)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

1

Traversal

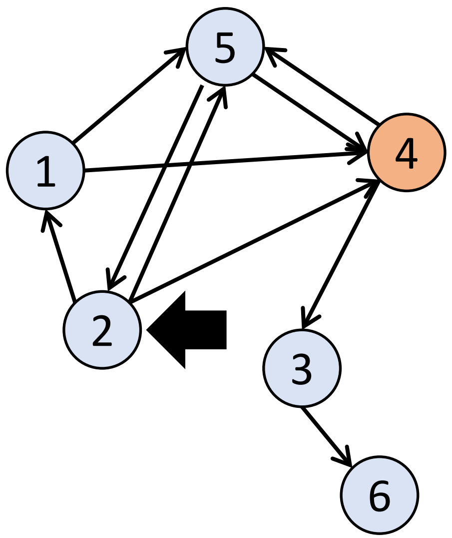
4, 3, 5, 6, 2, 1

Traversed edges

(4,3), (4,5), (3, 6), (5,2), (2,1)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

1

Traversal

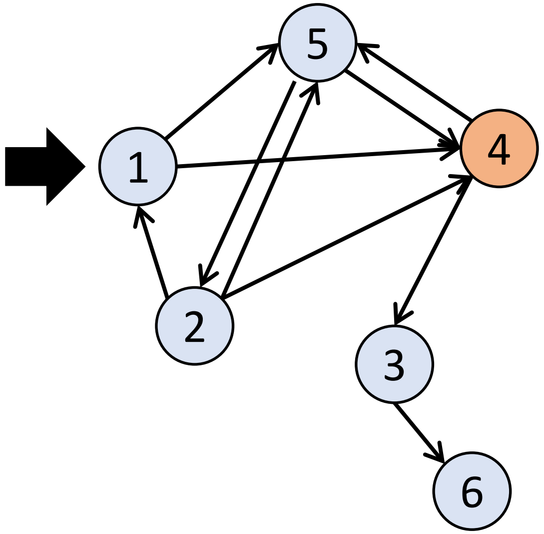
4, 3, 5, 6, 2, 1

Traversed edges

(4,3), (4,5), (3, 6), (5,2), (2,1)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

Traversal

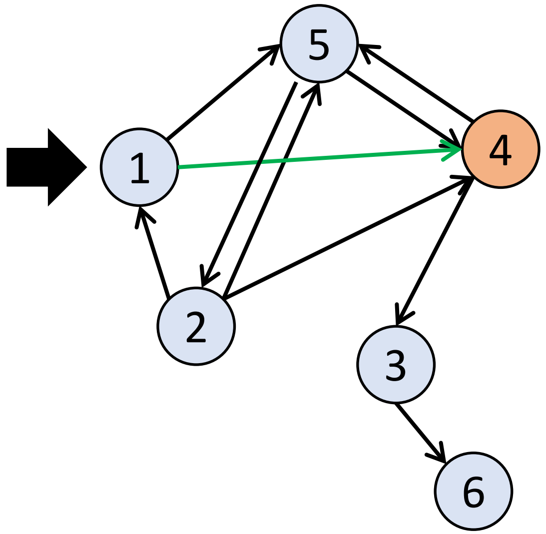
4, 3, 5, 6, 2, 1

Traversed edges

(4,3), (4,5), (3, 6), (5,2), (2,1)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

Traversal

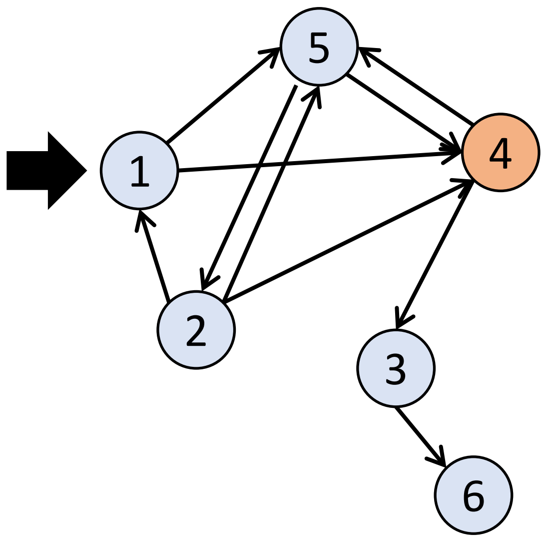
4, 3, 5, 6, 2, 1

Traversed edges

(4,3), (4,5), (3, 6), (5,2), (2,1)

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

Traversal

4, 3, 5, 6, 2, 1

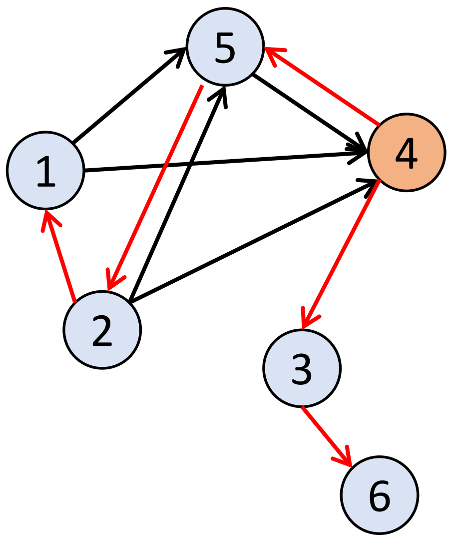
Traversed edges

(4,3), (4,5), (3, 6), (5,2), (2,1)

- BFS is done now.

BFS Traversal on Directed Graphs

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

Traversal

4, 3, 5, 6, 2, 1

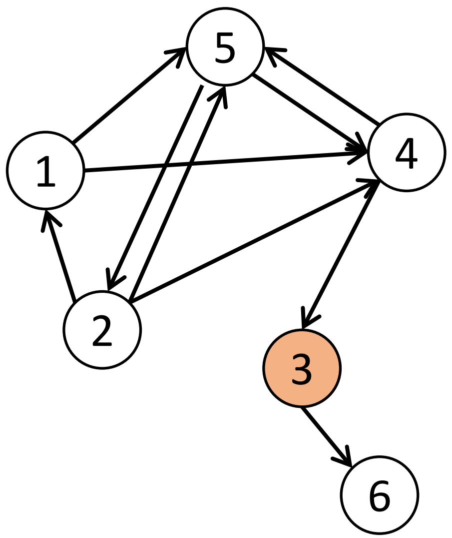
Traversed edges

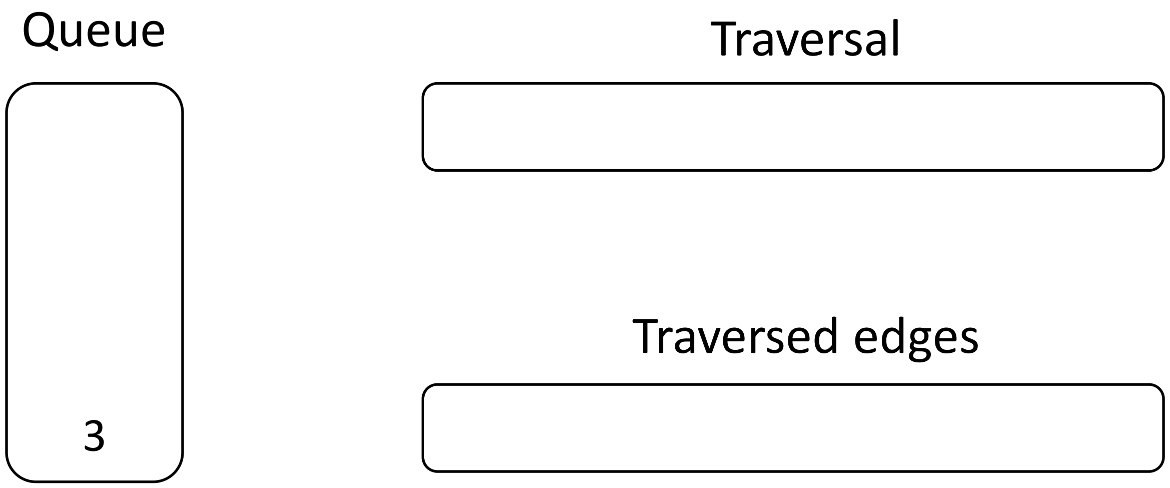
(4,3), (4,5), (3, 6), (5,2), (2,1)

- BFS tree spans all nodes because all nodes can be reached from 4.

When Graphs are not Strongly Connected

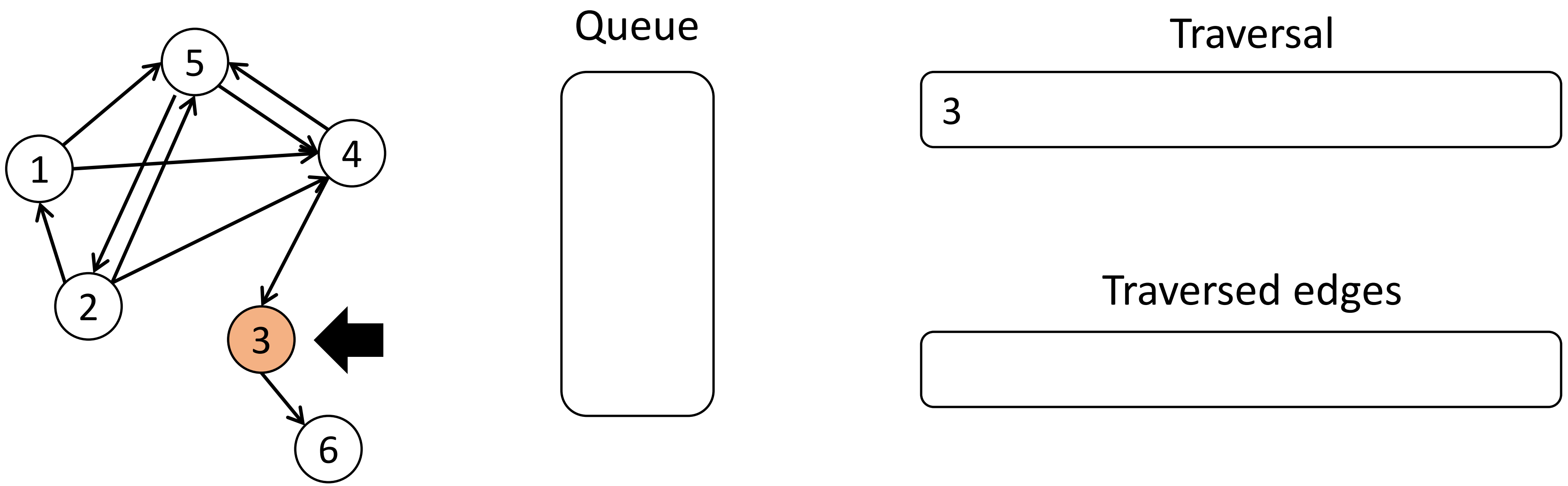
- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.





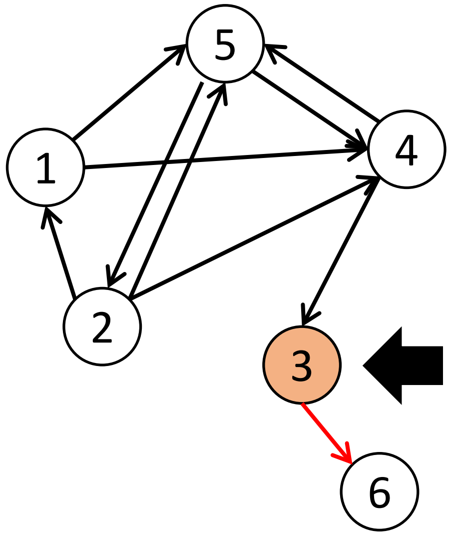
When Graphs are not Strongly Connected

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



When Graphs are not Strongly Connected

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

6

Traversal

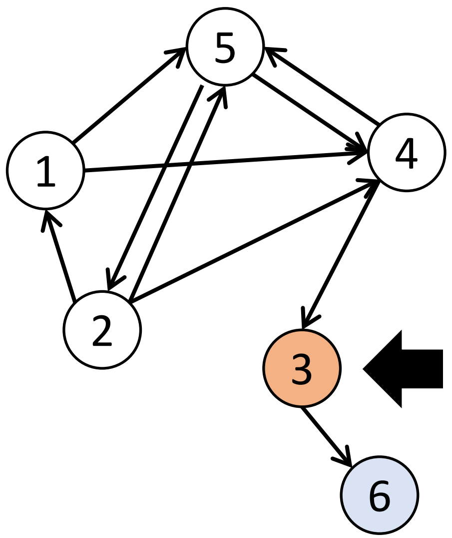
3

Traversed edges

(3,6)

When Graphs are not Strongly Connected

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

6

Traversal

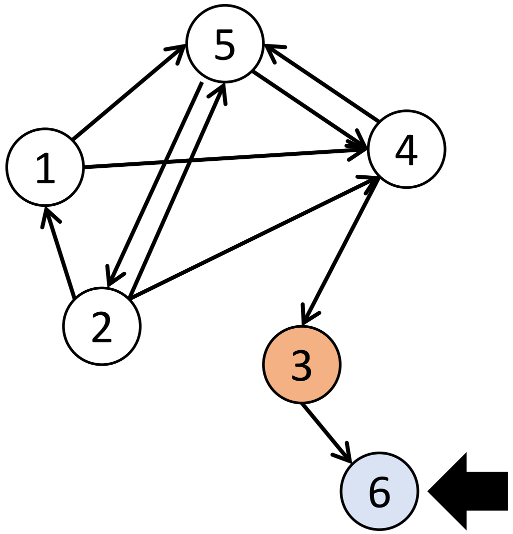
3

Traversed edges

(3,6)

When Graphs are not Strongly Connected

- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



Queue

Traversal

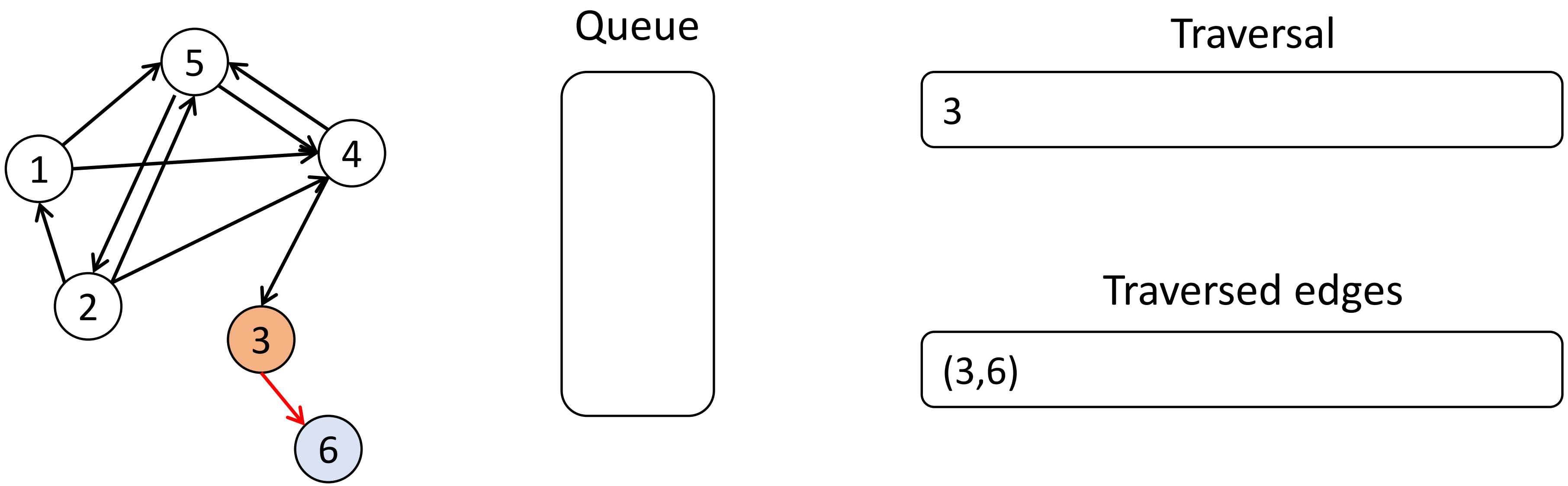
3

Traversed edges

(3,6)

When Graphs are not Strongly Connected

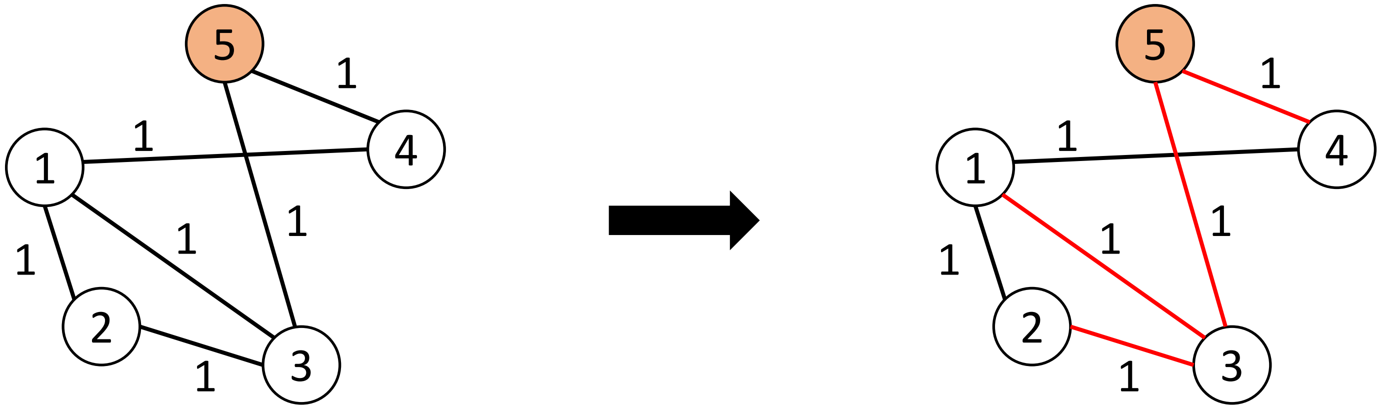
- Consider a directed, unweighted graph G. Compute BFS tree rooted at 4.



- BFS is done…
- When the graph is **not strongly-connected**, the **BFS may not visit all nodes**

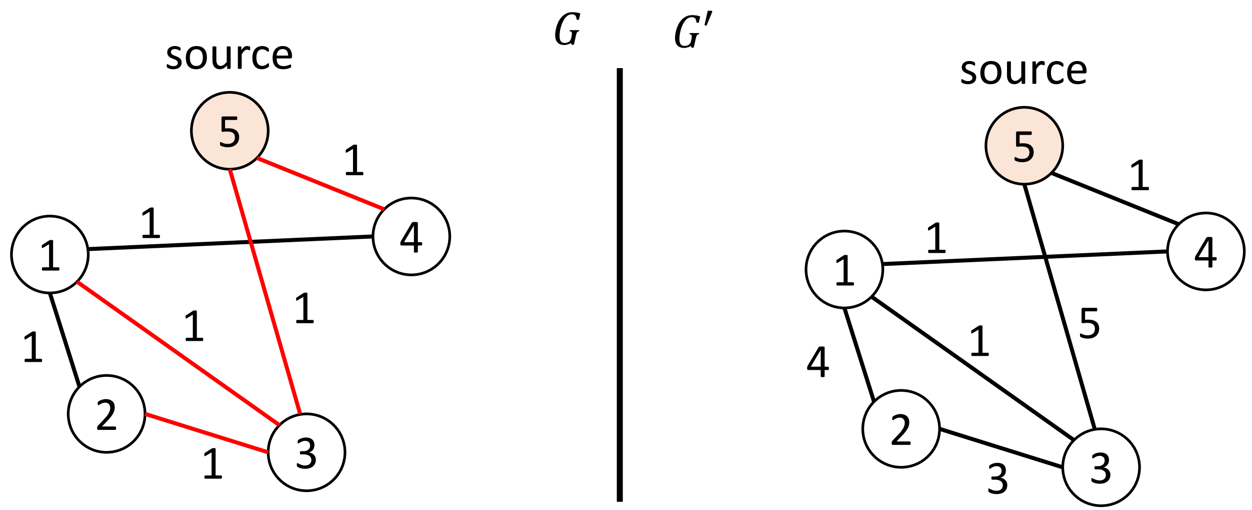
Shortest Paths in Unweighted Graphs

- BFS tree rooted at node 5



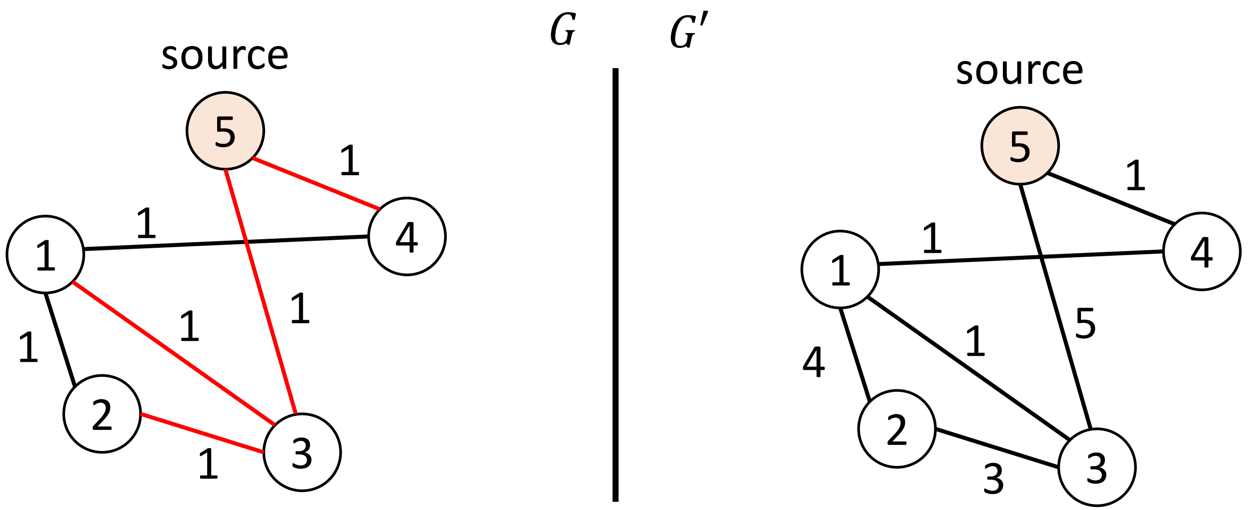
- BFS tree **contains** shortest paths from s to any node in G, when G is an (edge) unweighted graph.

Shortest Paths in Weighted Graphs?



- **Shortest path** in weighted graph:
  - **The sum of weights on that path should be smaller than the sum of weights on any other path.**

Shortest Paths in Weighted Graphs?

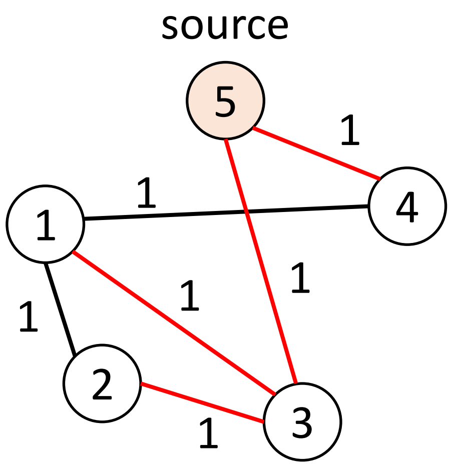


- Shortest path from source (5) to node 3 in G' ?

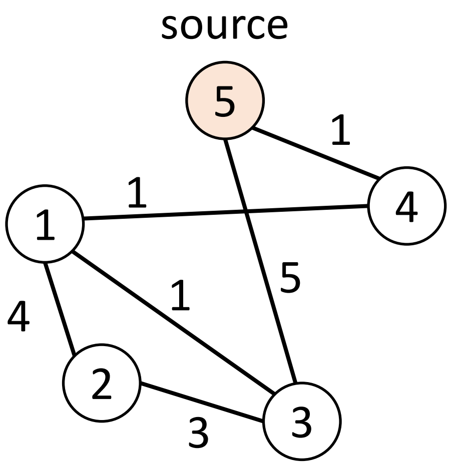
  (5,4,1,3) or 5 -> 4 -> 1 -> 3

- Shortest path when you consider **weights (or distances), not hops**.

Shortest Paths in Weighted Graphs?



G G'



- How do you compute shortest paths on weighted graph ?
  - BFS does not work. Why?
  - **Because BFS tree does not take edge weights into account.**

Shortest Paths in Weighted Graphs

- Use **Dijkstra's algorithm** to solve the three shortest path problems
  1. Shortest distances (from source)
  2. Shortest paths (from source)
  3. Pathfinding (from source)

- Linked to network routing protocols or robot exploration (or search) algorithms.
- Several extensions:
  - A* search algorithm
  - Bellman-Ford algorithm
  - Floyd-Warshall algorithm

Shortest Distances via Dijkstra's Algorithm

- Keep two arrays: **visited**[] and **distanceToSource**[]

1. Start at source, and initialize:
   - **Initialize visited status** of all nodes (besides source) to unvisited.
   - **Initialize distances to source** to ∞ for all nodes (except source), and to 0 for the source.

2. While not all nodes have been visited:
   - Visit (unvisited) node v with smallest distanceToSource.
   - For all neighbors of v, if the path going through v is shorter, then update distance.

For visited node v and neighbor u:
If distToS[u] > distToS[v] + weight(u,v),
then distToS[u] = distToS[v] + weight(u,v)

3. Return distanceToSource.

Shortest Distances via Dijkstra's Algorithm

Lancaster University

Some intuition:

1. At the start, and through most of the algorithm, these are "approximate distances"
   - ∞ implies the node is "very far", or more precisely unreachable,
   - 0 implies the node is the source.

2. The **key idea** of Dijkstra's algorithm is to **improve these approximations, as you visit more and more nodes, until all distances to the source are accurate**.

3. (**Invariant**) When a node is visited, their distance to the source is accurate.

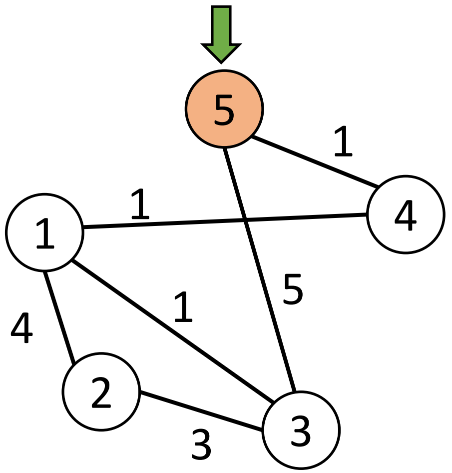
Shortest Distances via Dijkstra’s algorithm

```
function Dijkstra(int source){
    for(int i = 1; i <= graph.size; i++){
        distanceToSource[i-1] = infinity
        visited[i-1] = false
    }
    distToS[source-1] = 0;

    while(there remains unvisited vertices){
        visitedNode = unvisited node with min distanceToSource    // Nodes are ints from 1 to size
        visited[visitedNode-1] = true;
        for(each neighborNode of visitedNode){
            estimatedDistance = distanceToSource[visitedNode] + weight(visitedNode, neighborNode);
            if(estimatedDistance < distanceToSource[visitedNode])
                 distanceToSource[visitedNode] = estimatedDistance;
        }
    }
    return distanceToSource; }
```

Dijkstra's Algorithm: Example Execution

- Source is node 5.



visited

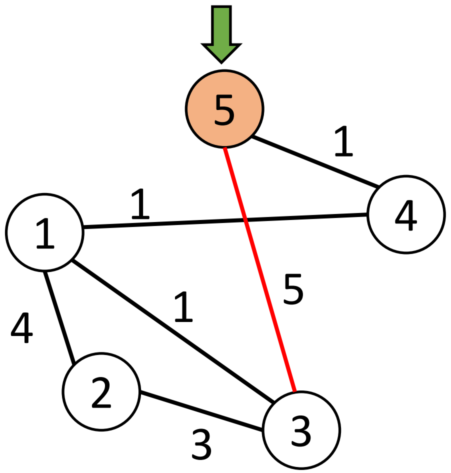
F	F	F	F	T

DistanceToSource

∞	∞	∞	∞	0

Dijkstra’s Algorithm: Example Execution

- Look at neighbors of 5.



visited

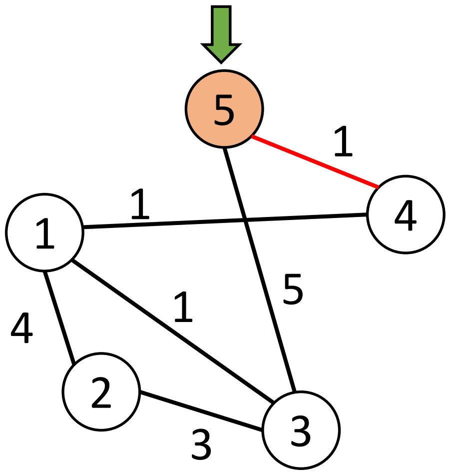
F	F	F	F	T

DistanceToSource

∞	∞	5	∞	0

Dijkstra’s Algorithm: Example Execution

- Look at neighbors of 5.



visited

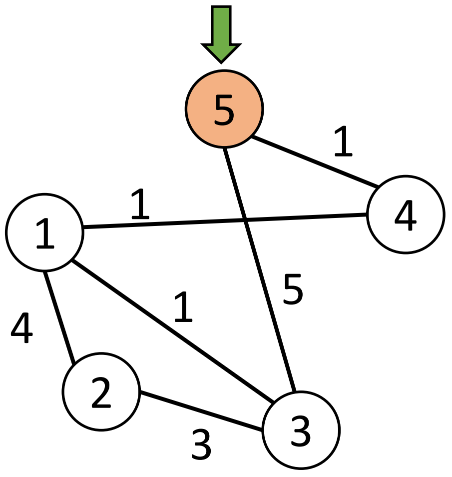
F	F	F	F	T

DistanceToSource

∞	∞	5	1	0

Dijkstra’s Algorithm: Example Execution

- Next, visit node with smallest distance to source among unvisited nodes.



visited

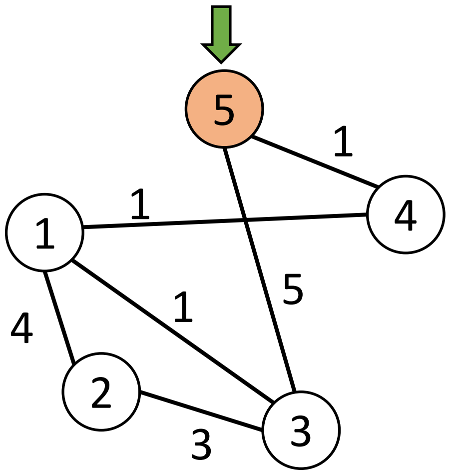
F	F	F	F	T

DistanceToSource

∞	∞	5	1	0

Dijkstra's Algorithm: Example Execution

- Next, visit node with smallest distance to source among unvisited nodes.



visited

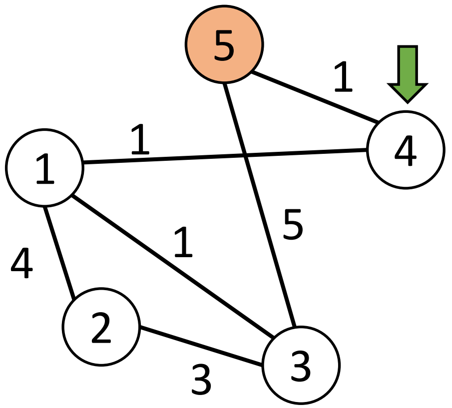
F	F	F	F	T

DistanceToSource

∞	∞	5	1	0

Dijkstra’s Algorithm: Example Execution

- Now, we visit node 4 and update the distance to source of its (unvisited) neighbors.



visited

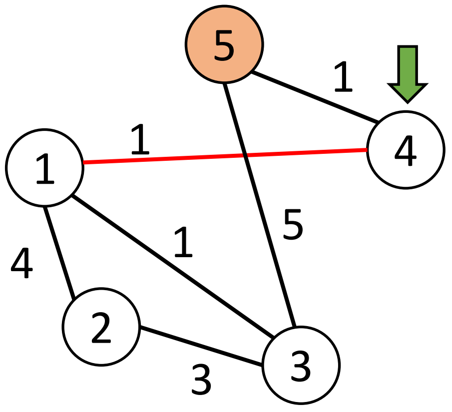
F	F	F	T	T

DistanceToSource

∞	∞	5	1	0

Dijkstra's Algorithm: Example Execution

- Now, we visit node 4 and update the distance to source of its (unvisited) neighbors.



visited

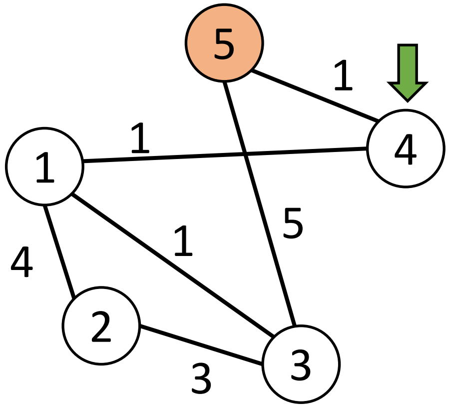
F	F	F	T	T

DistanceToSource

2	∞	5	1	0

Dijkstra’s Algorithm: Example Execution

- Next, visit node with smallest distance to source among unvisited nodes.



visited

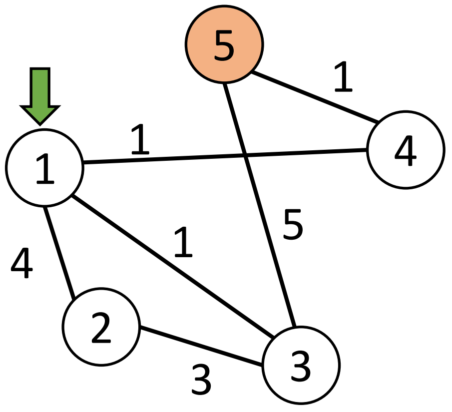
F	F	F	T	T

DistanceToSource

2	∞	5	1	0

Dijkstra's Algorithm: Example Execution

- Now, we visit node 1 and update the distance to source of its (unvisited) neighbors.



visited

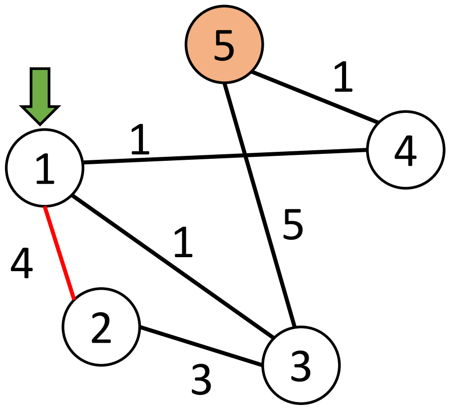
T	F	F	T	T

DistanceToSource

2	∞	5	1	0

Dijkstra’s Algorithm: Example Execution

- Now, we visit node 1 and update the distance to source of its (unvisited) neighbors.



visited

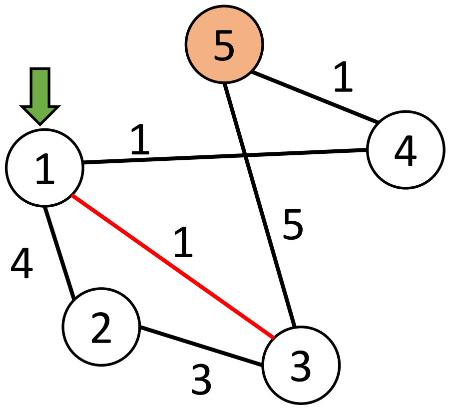
T	F	F	T	T

DistanceToSource

2	6	5	1	0

Dijkstra’s Algorithm: Example Execution

- Now, we visit node 1 and update the distance to source of its (unvisited) neighbors.



visited

T	F	F	T	T

DistanceToSource

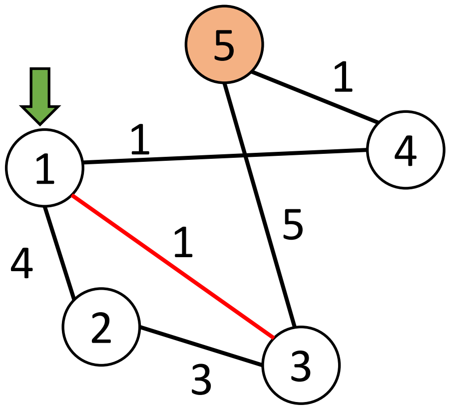
2	6	5	1	0

$3 < 5$

distanceToSource[node 1] + weight(node 1, node 3) < distanceToSource[node 3]

Dijkstra's Algorithm: Example Execution

- Now, we visit node 1 and update the distance to source of its (unvisited) neighbors.



visited

T	F	F	T	T

DistanceToSource

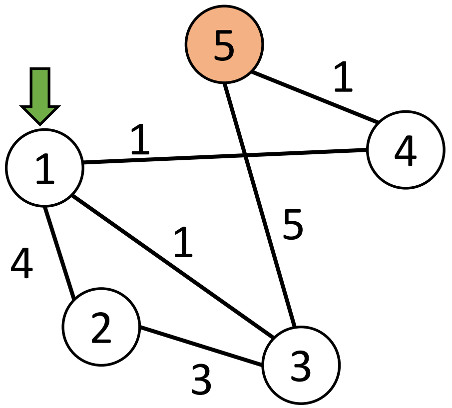
2	6	3	1	0

$3 < 5$

distanceToSource[node 1] + weight(node 1, node 3) < distanceToSource[node 3]

Dijkstra's Algorithm: Example Execution

- Next, visit node with smallest distance to source among unvisited nodes.



visited

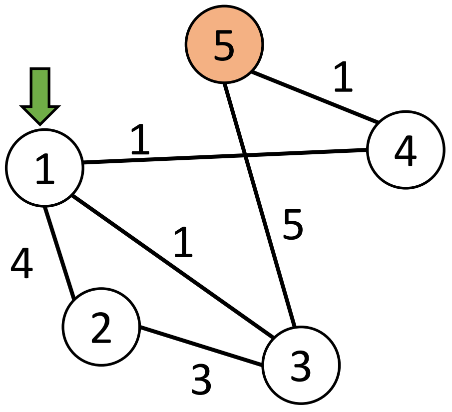
T	F	F	T	T

DistanceToSource

2	6	3	1	0

Dijkstra’s Algorithm: Example Execution

- Next, visit node with smallest distance to source among unvisited nodes.



visited

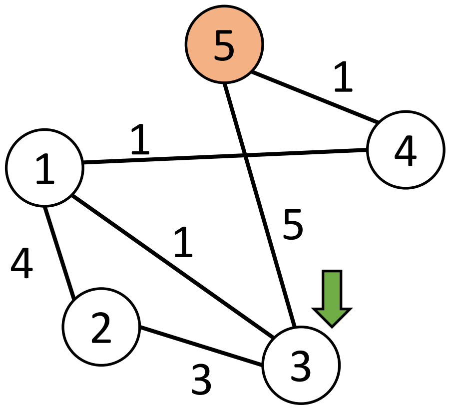
T	F	F	T	T

DistanceToSource

2	6	3	1	0

Dijkstra's Algorithm: Example Execution

- Now, we visit node 3 and update the distance to source of its (unvisited) neighbors.



visited

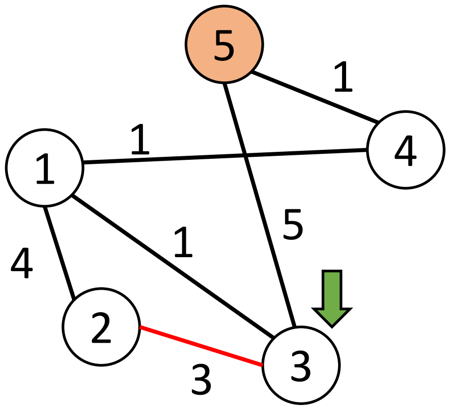
T	F	T	T	T

DistanceToSource

2	6	3	1	0

Dijkstra's Algorithm: Example Execution

- Now, we visit node 3 and update the distance to source of its (unvisited) neighbors.



visited

T	F	T	T	T

DistanceToSource

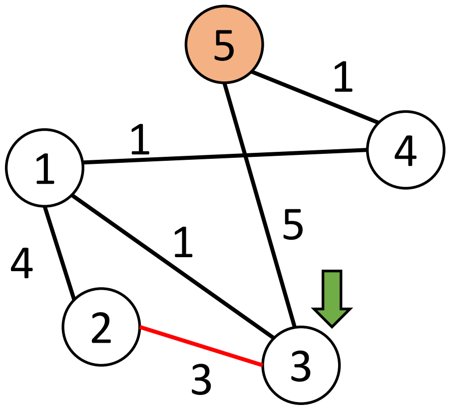
2	6	3	1	0

$6\ vs\ 6$

distanceToSource[node 3] + weight(node 3, node 2) < distanceToSource[node 2]

Dijkstra’s Algorithm: Example Execution

- Now, we visit node 3 and update the distance to source of its (unvisited) neighbors.



visited

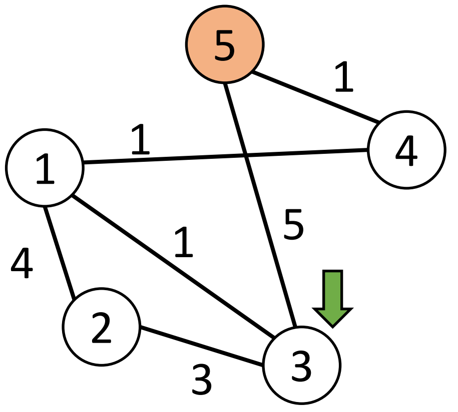
T	F	T	T	T

DistanceToSource

2	6	3	1	0

Dijkstra's Algorithm: Example Execution

- Next, visit node with smallest distance to source among unvisited nodes.



visited

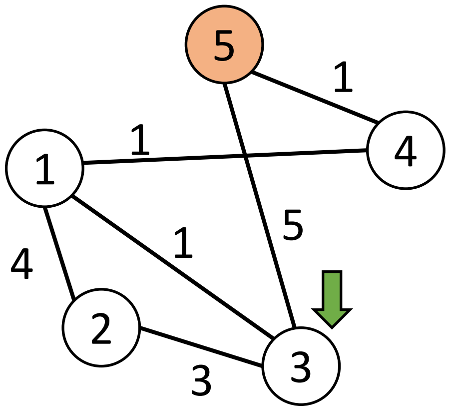
T	F	T	T	T

DistanceToSource

2	6	3	1	0

Dijkstra’s Algorithm: Example Execution

- Next, visit node with smallest distance to source among unvisited nodes.



visited

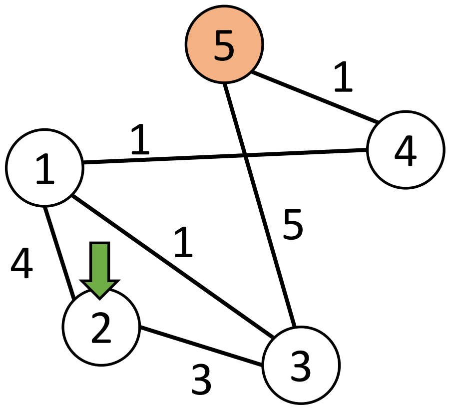
T	F	T	T	T

DistanceToSource

2	6	3	1	0

Dijkstra’s Algorithm: Example Execution

- Now, we visit node 3 and update the distance to source of its (unvisited) neighbors.



visited

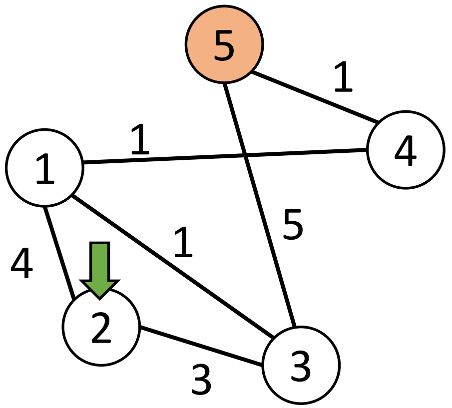
T	T	T	T	T

DistanceToSource

2	6	3	1	0

Dijkstra's Algorithm: Example Execution

- There are no unvisited neighbors, but also no unvisited nodes remaining.



visited

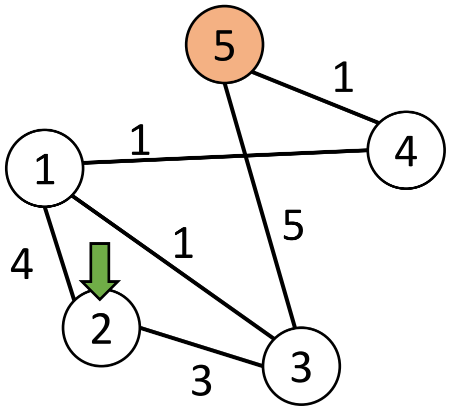
T	T	T	T	T

DistanceToSource

2	6	3	1	0

Dijkstra’s Algorithm: Example Execution

- There are no unvisited neighbors, but also no nodes remains unvisited.



visited

T	T	T	T	T

DistanceToSource

2	6	3	1	0

- Dijkstra’s algorithm terminates by returning distanceToSource

Shortest Paths via Dijkstra's Algorithm

- Keep three arrays: visited[], **previousNode**[] and distanceToSource[]

1. Start at source, and initialize:
   - Initialize visited status of all nodes (besides source) to unvisited.
   - Initialize distances to source to ∞ for all nodes (except source), and to 0 for the source.
   - **Initialize previousNodes entries** to null.

2. While not all nodes have been visited:
   - Visit (unvisited) node v with smallest distanceToSource.
   - For all neighbors of v, if the path going through v is shorter, then update distance **and set previousNode[neighbor] to v**

3. Return distanceToSource and **previousNode**.

Pathfinding via Dijkstra's Algorithm

- Find shortest paths between two nodes: source and target.

- <u>In Dijkstra's algorithm:</u>
  2. While not all nodes have been visited:
     - Visit (unvisited) node v with smallest distanceToSource.
     - **If current visited node is the target node, return distanceToSource[v] and pathToSource[v], where pathToSource[v] = (previousNode[v], previousNode[previousNode[v]], …).**
     - For all neighbors of v, if the distance of the path going through v is a better approximation, …

  3. Throw an exception: "target node not found"

Complexity of Dijkstra's Algorithm

- **Worst-case time complexity of $\boldsymbol{O(n^2)}$:**
  - We visit each node once, and during each visit, we do $O(1)$ operations per neighbors.
  - This amounts to $O(m)$ operations where m is the number of edges.

  - And to get each new visited node, we find the unvisited node with minimum distance to source, which takes $O(n)$ operations.
  - In total, this amounts to $O(n^2)$ operations.

- We can use **better data structures** (priority queues based on Fibonacci heaps, or self-balanced binary search trees, or other heaps) to improve Dijkstra's algorithm to:

$$\boldsymbol{O(m + n \log n)} \textbf{ worst-case time complexity}$$

Correctness of Dijkstra's Algorithm

- Dijkstra's algorithm only works **if all edge weights are non-negative**

- **Correctness sketch:**
  - **Invariant =** Whenever a node is visited, its distance to the source is correct.

  - Base case: True for the source at the beginning of the algorithm.
  - Induction step:
    1. For each newly visited node v_i, it has minimum value in distanceToSource array among all unvisited nodes.
    2. If there exists a shorter path from the source to v_i, then there must be an edge in that path between a visited node and an unvisited node u.
    3. In distanceToSource, the value of u should be smaller than distanceToSource[v_i], which is a contradiction.

Better Shortest Path Algorithms?

1. When we have extra information (heuristic) on distances from some nodes to source, we can use the **A* algorithm**.

2. If we have edges with negative weights (but not negative cycles) then you can use **Bellman-Ford.**

3. If you are fine with approximating the distances (or getting some “almost” shortest paths), then there are faster algorithms…

A* Search Algorithm

- For pathfinding only (from specified source to specified target), not from the source to all other nodes

- **Core idea**: use heuristics (on the distance from current node to specified target) to guide (i.e., be more efficient in) the search
  - For example, if **graph weights** represent **Manhattan distances** (e.g., driving along roads)
  - Then the **heuristics** could be **straight-line distance** (e.g., distance as the bird flies)

- If $d(v)$ is the estimated distance from source to v in Dijkstra's algorithm, and $h(v)$ the heuristic distance from node v to the target, then **in A* we compare the values**:

$$d(v) + h(v)$$

Bellman-Ford

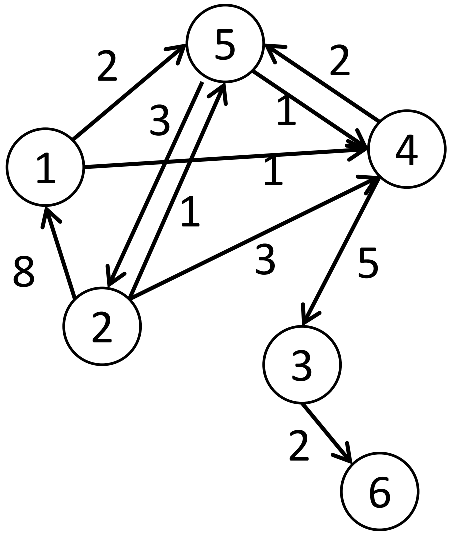
- Computes shortest path from specified source to all other nodes (like Dijkstra's algorithm)
  - Works for wider class of graphs, as it can work for (some) graphs with negative weights
  - Still, a major problem if there is a **negative cycle** (with negative total weight sum)
  - When there is a negative cycle, Bellman-Ford finds that cycle (but not all shortest paths)

- **Core idea** (like Dijkstra's algorithm): **compute (over)estimations of the distances** from the source to all other nodes, **which you iteratively improve upon** (or **relax**).
  - In Dijkstra's algorithm, you use a priority queue to decide which edge to relax,
  - In Bellman-Ford, you relax all edges every iteration, for a total of $n - 1$ iterations

- **Worst-Case Time Complexity**: $O(n \cdot m)$

Dijkstra's algorithm on Directed Graphs

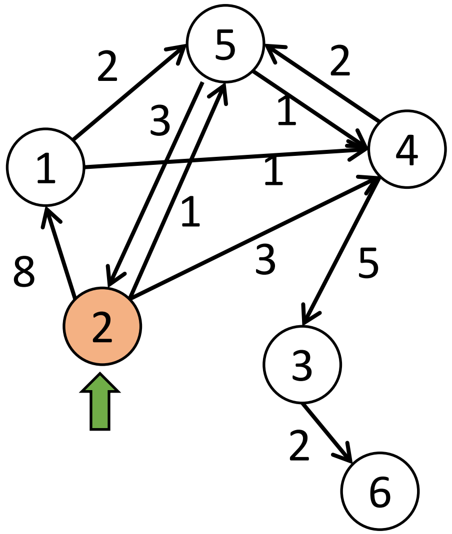
- Consider a directed, weighted graph G.



	(1)	(2)	(3)	(4)	(5)	(6)
(1)	0	0	0	1	2	0
(2)	8	0	0	3	1	0
(3)	0	0	0	0	0	2
(4)	0	0	5	0	2	0
(5)	0	3	0	1	0	0
(6)	0	0	0	0	0	0

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	F	F

DistanceToSource

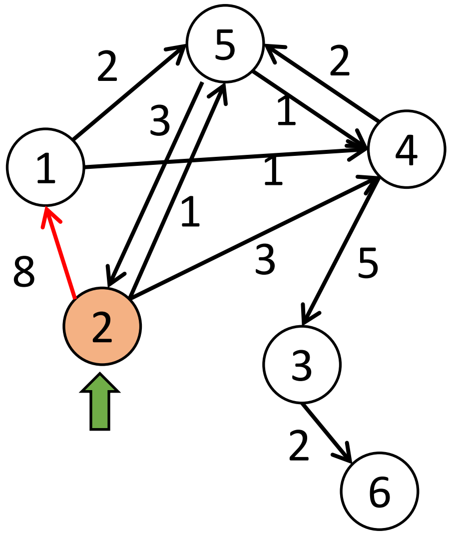
∞	0	∞	∞	∞	∞

previousNode

null	null	null	null	null	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	F	F

DistanceToSource

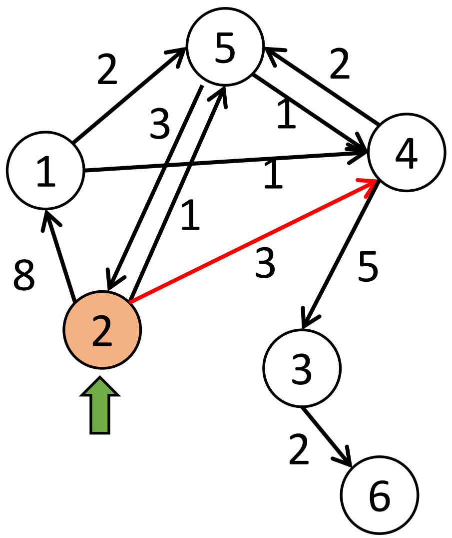
8	0	∞	∞	∞	∞

previousNode

2	null	null	null	null	null

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	F	F

DistanceToSource

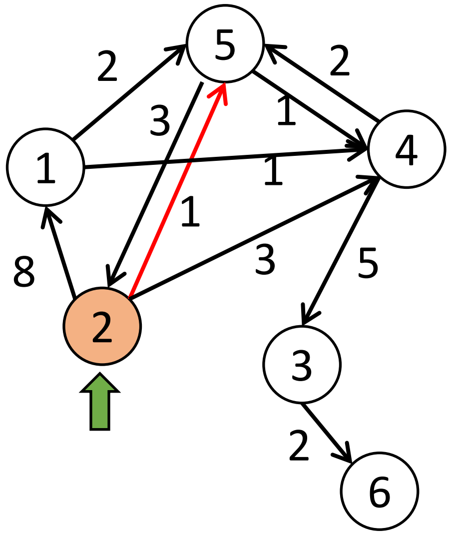
8	0	∞	3	∞	∞

previousNode

2	null	null	2	null	null

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	F	F

DistanceToSource

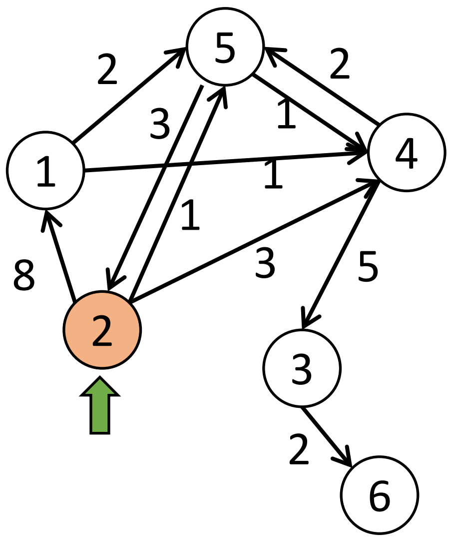
8	0	∞	3	1	∞

previousNode

2	null	null	2	2	null

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	F	F

DistanceToSource

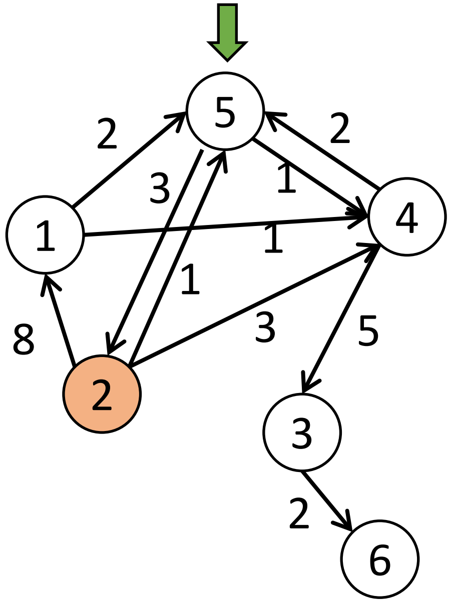
8	0	∞	3	1	∞

previousNode

2	null	null	2	2	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	T	F

DistanceToSource

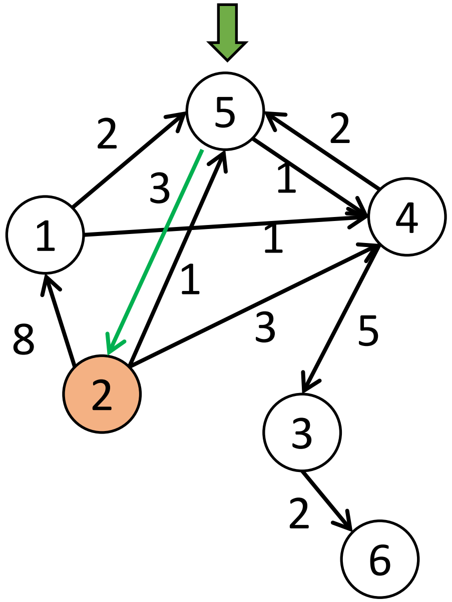
8	0	∞	3	1	∞

previousNode

2	null	null	2	2	null

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	T	F

DistanceToSource

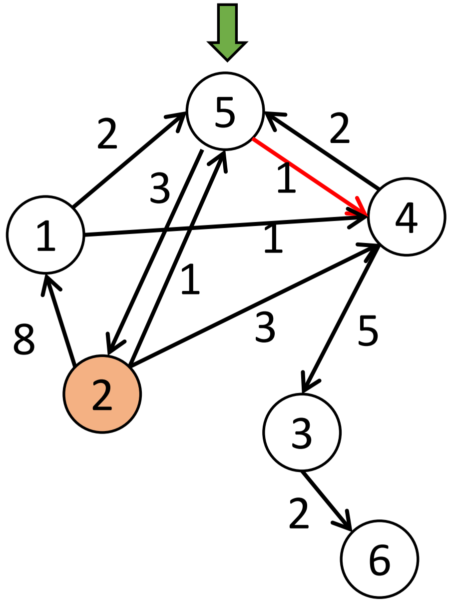
8	0	∞	3	1	∞

previousNode

2	null	null	2	2	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	T	F

DistanceToSource

8	0	∞	3	1	∞

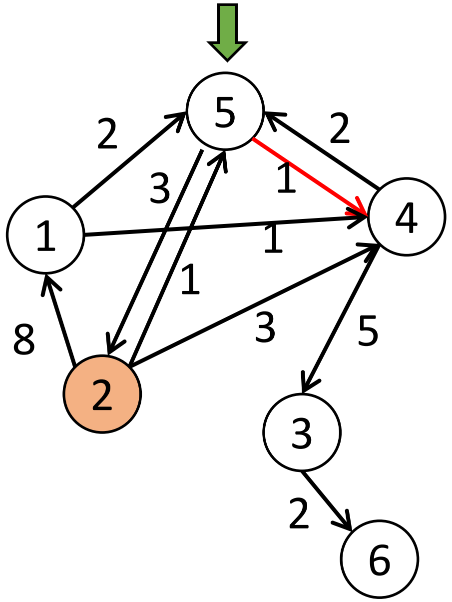
previousNode

2	null	null	2	2	null

distToSource[node 5] + weight(node 5, node 4) < distToSource[node 4]

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	T	F

DistanceToSource

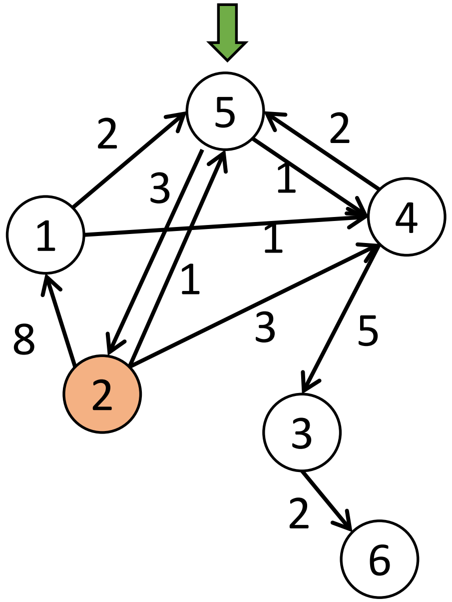
8	0	∞	2	1	∞

previousNode

2	null	null	5	2	null

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	F	T	F

DistanceToSource

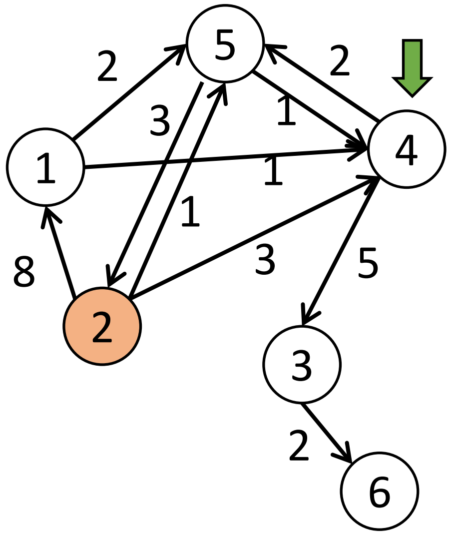
8	0	∞	2	1	∞

previousNode

2	null	null	5	2	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	T	T	F

DistanceToSource

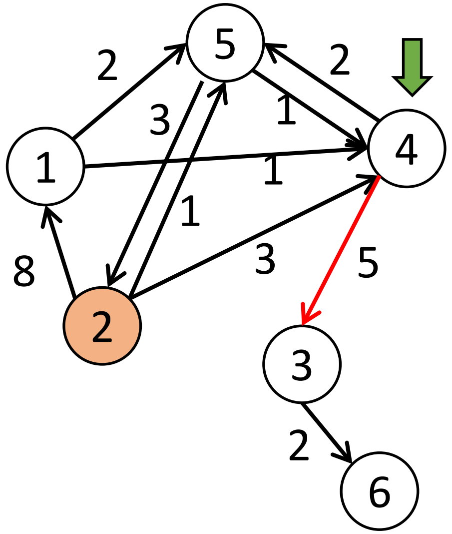
8	0	∞	2	1	∞

previousNode

2	null	null	5	2	null

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	T	T	F

DistanceToSource

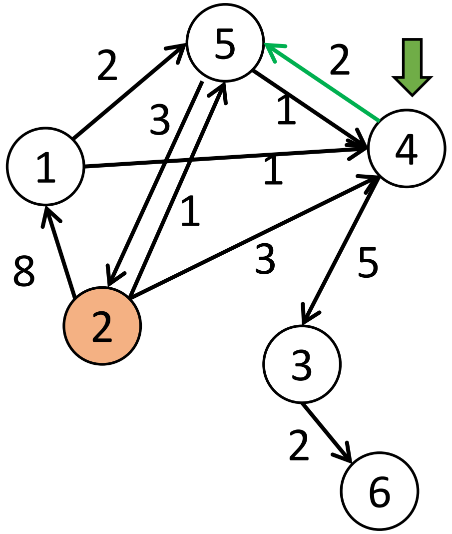
8	0	7	2	1	∞

previousNode

2	null	4	5	2	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	T	T	F

DistanceToSource

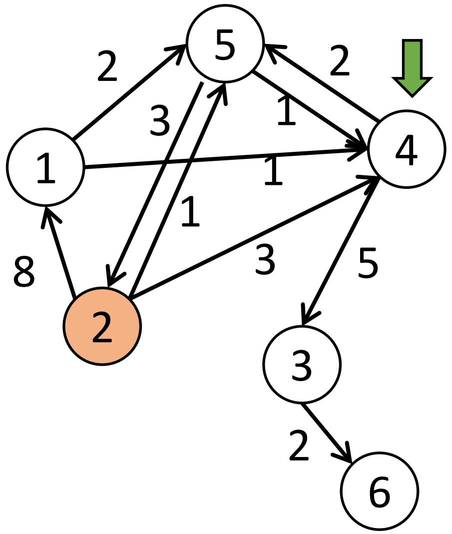
8	0	7	2	1	∞

previousNode

2	null	4	5	2	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	T	T	F

DistanceToSource

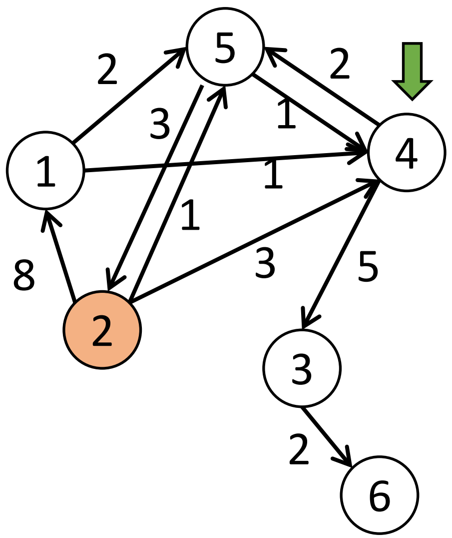
8	0	7	2	1	∞

previousNode

2	null	4	5	2	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	F	T	T	F

DistanceToSource

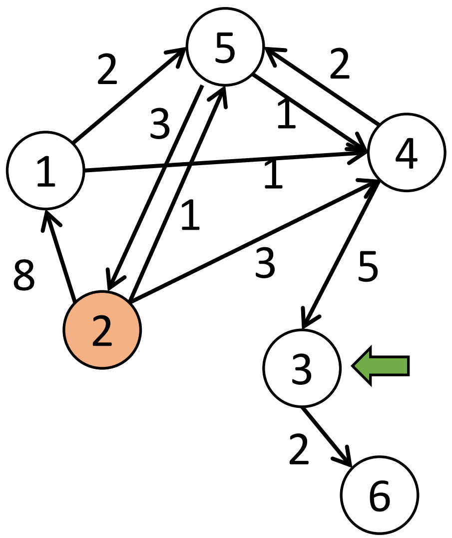
8	0	7	2	1	∞

previousNode

2	null	4	5	2	null

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	T	T	T	F

DistanceToSource

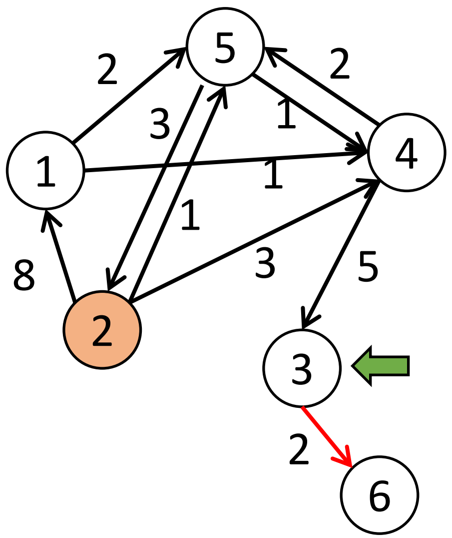
8	0	7	2	1	∞

previousNode

2	null	4	5	2	null

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	T	T	T	F

DistanceToSource

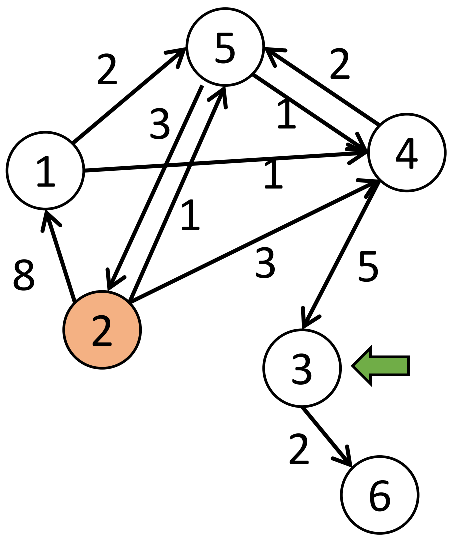
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	T	T	T	F

DistanceToSource

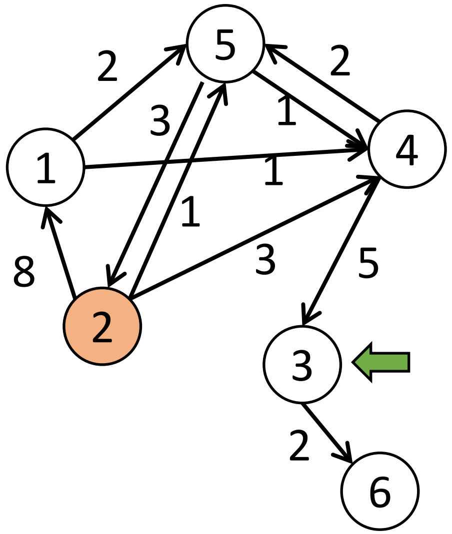
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

F	T	T	T	T	F

DistanceToSource

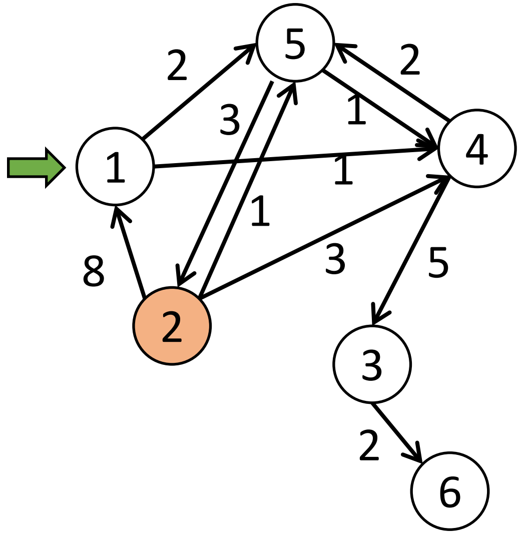
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

T	T	T	T	T	F

DistanceToSource

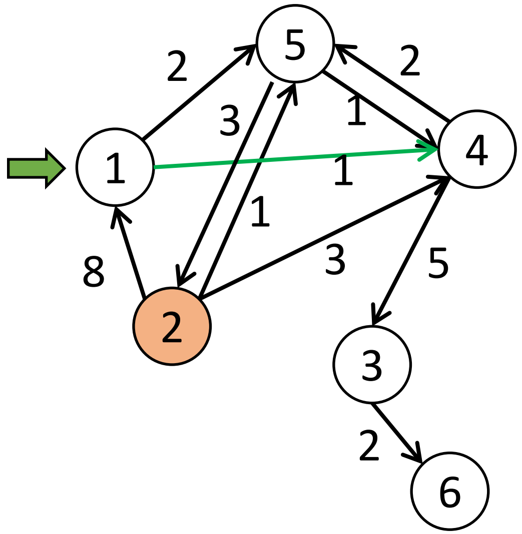
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

T	T	T	T	T	F

DistanceToSource

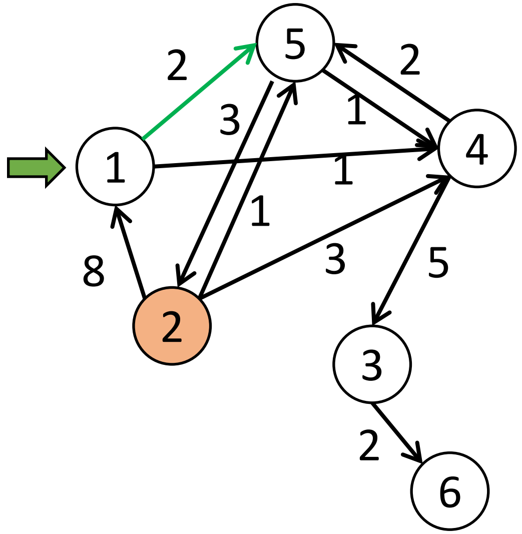
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

T	T	T	T	T	F

DistanceToSource

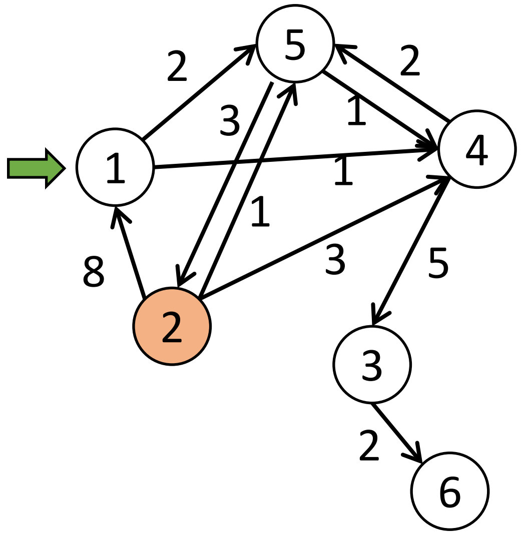
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

T	T	T	T	T	F

DistanceToSource

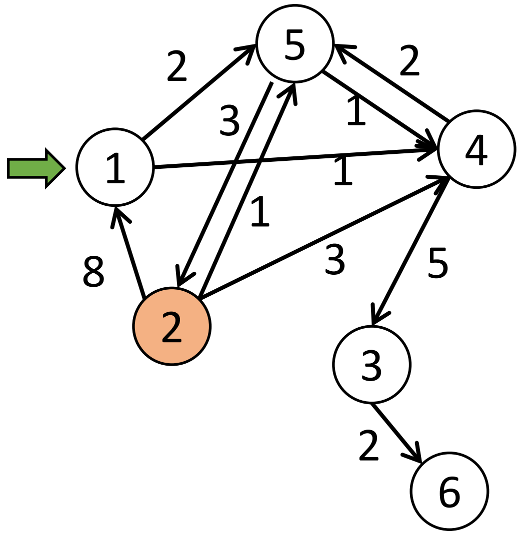
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra’s algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

T	T	T	T	T	F

DistanceToSource

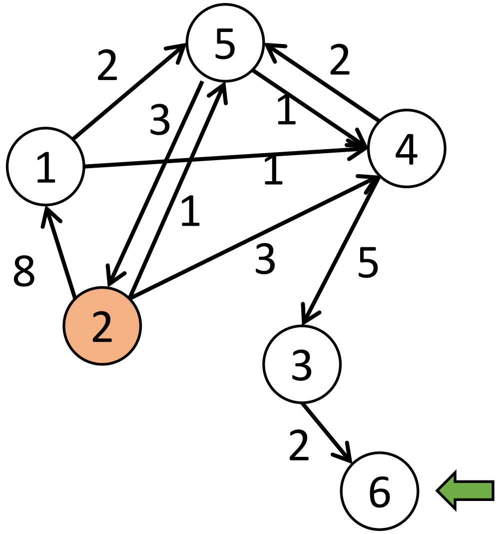
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



visited

T	T	T	T	T	T

DistanceToSource

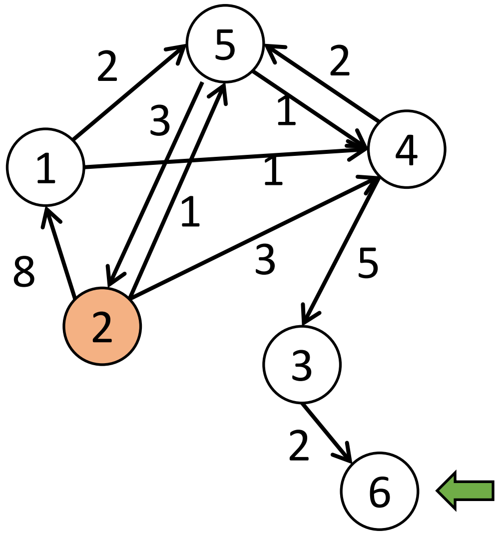
8	0	7	2	1	9

previousNode

2	null	4	5	2	3

Dijkstra's algorithm on Directed Graphs

- Consider a directed, weighted graph G. Computes shortest paths from node 2.



No unvisited or unreachable nodes

visited

T	T	T	T	T	T

DistanceToSource

8	0	7	2	1	9

previousNode

2	null	4	5	2	3

- Dijkstra's algorithm terminates now.

Summary

Today's lecture:

- More examples of BFS graph traversals (undirected and directed),
- BFS graph traversals do not suffice to copmute shortest paths in weighted graphs,
- Instead, Dijkstra's algorithm can compute shortest paths (and more) on weighted graphs (with non-negative edge weights).

- **Next Lecture:** Greedy algorithms.
- **Any questions?**