COMP2521 23T3

Graph Traversal

Kevin Luxa

cs2521@cse.unsw.edu.au

graph traversal
bfs and dfs
path checking
path finding

Problems on Graphs

Common problems on graphs:

- Is there a path between two vertices?
- What is the shortest path between two vertices?
- Is the graph connected?
- If we remove an edge, is the graph still connected?
- Which vertices are reachable from a particular vertex?
- Is there a cycle that passes through all vertices?

Graph Traversal

All of the above problems can be solved by
a systematic exploration of a graph via its edges.

This systematic exploration is called traversal or search.

Graph Traversal

PROBLEM

Does a path exist between vertices src and $dest$?

Possible approach:

1. examine vertices adjacent to src
2. if any of them is $dest$, we're done!
3. otherwise, check vertices two edges away from src
4. repeat looking further and further away from src

The above summarises one form of graph traversal.

Graph Traversal

Two primary methods for graph traversal/search:

Breadth-first search (BFS)

- Prioritises visiting all neighbours over path-following
  - “Go wide”
- Implemented iteratively (using a queue)

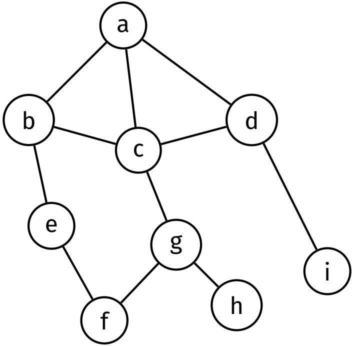
Depth-first search (DFS)

- Prioritises path-following over visiting all neighbours
  - “Go deep”
- Implemented recursively or iteratively (using a stack)

Graph Traversal

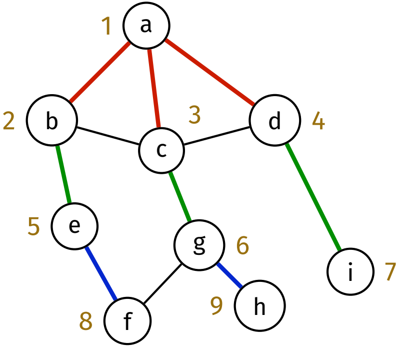
BFS vs. DFS

In what order would BFS and DFS visit the vertices of this graph?

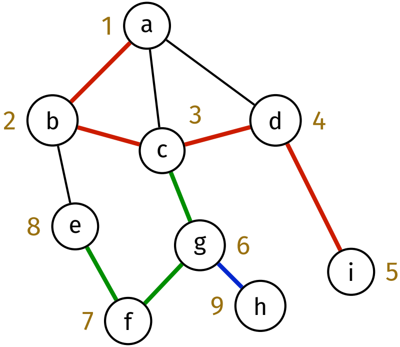


Graph Traversal

BFS vs. DFS



Breadth-first search



Depth-first search

Breadth-First Search

Breadth-first search visits vertices
in order of distance from the starting vertex.

It visits the starting vertex,
then the neighbours of the starting vertex,
then the neighbours of those neighbours,
etc.

BFS is implemented iteratively using a queue.

Breadth-First Search

Data structures

Data structures used in BFS:

- Visited array
    - To keep track of which vertices have been visited
- Predecessor array
    - To keep track of the predecessor of each vertex
    - The predecessor of v is the vertex from which we reached v
        - i.e., the vertex before v on the path to v
- Queue
    - First-in-first-out data structure
    - Stores unvisited vertices in the order that they should be visited

Breadth-First Search

Algorithm

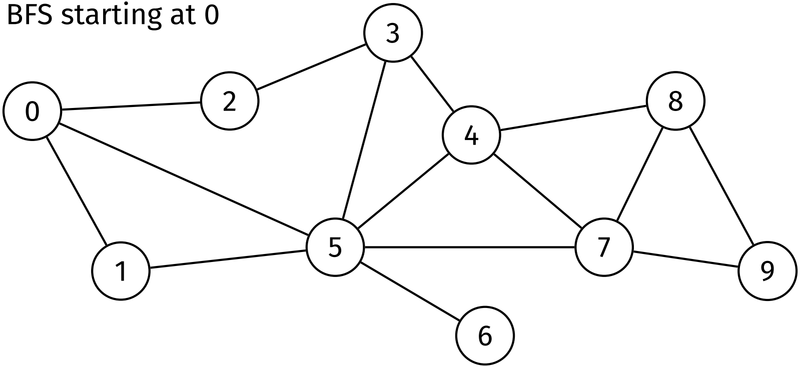
Algorithm:

1. Create/initialise data structures:
    - Initialise visited array to false
    - Initialise predecessor array to -1
    - Create empty queue
2. Mark starting vertex as visited and enqueue it
3. While the queue is not empty:
    1. Dequeue a vertex
        - Let this vertex be v
    2. **Explore** v - that is, for each of v's unvisited neighbours:
        1. Mark it as visited
        2. Set its predecessor to v
        3. Enqueue it

Breadth-First Search

Example

BFS starting at 0



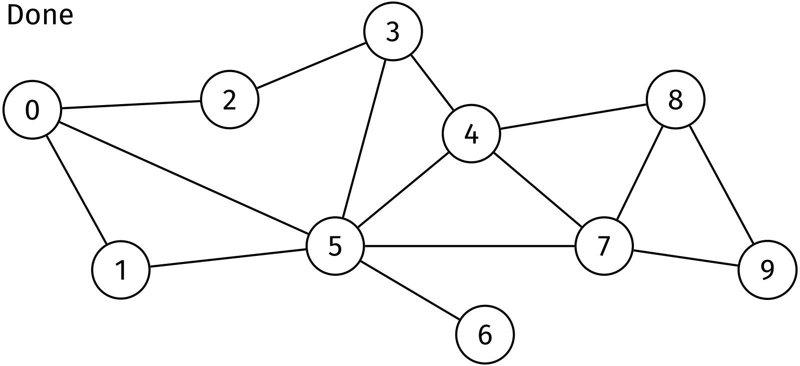
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited										
pred										

queue

Breadth-First Search

Example

Done



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

Breadth-First Search

Pseudocode

```
bfs($G$, $src$):
    **Inputs:** graph $G$, starting vertex $src$

    create visited array, predecessor array and queue $Q$

    **for each** vertex $v$ in $G$:
        visited[$v$] = false
        predecessor[$v$] = -1

    visited[$src$] = true
    enqueue $src$ into $Q$

    **while** $Q$ is not empty:
        $v$ = dequeue from $Q$
        **for each** neighbour $w$ of $v$ where visited[$w$] $=$ false:
            visited[$w$] = true
            predecessor[$w$] = $v$
            enqueue $w$ into $Q$
```

Breadth-First Search

Simplification

When using a predecessor array in BFS,
the predecessor array can double as a visited array

`predecessor`$[v]$ $= -1$ means v is not visited

Breadth-First Search

Simplification

```
bfs($G$, $src$):
    Inputs: graph $G$, starting vertex $src$

    create predecessor array and queue $Q$

    for each vertex $v$ in $G$:
        predecessor[$v$] = -1

    predecessor[$src$] = $src$ // <- mark src as visited
    enqueue $src$ into $Q$

    while $Q$ is not empty:
        $v$ = dequeue from $Q$
        for each neighbour $w$ of $v$ where predecessor[$w$] $=$ -1:
            predecessor[$w$] = $v$
            enqueue $w$ into $Q$
```

Breadth-First Search

Analysis

BFS is $O(V + E)$ when using the adjacency list representation:

- Typical queue implementation has $O(1)$ enqueue and dequeue
- Each vertex is visited at most once $\Rightarrow O(V)$
- For each vertex, all of its edges are considered once $\Rightarrow O(E)$

Path-Finding with BFS

A BFS finds the shortest path between the starting vertex and all other vertices.

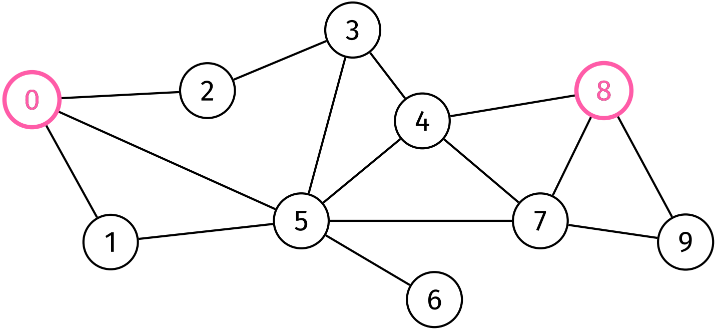
- Shortest path in terms of the number of edges

The shortest path between src and $dest$ can be found by tracing backwards through the predecessor array (from $dest$ to src).

Path-Finding with BFS

Example: Shortest path from 0 to 8

0 8

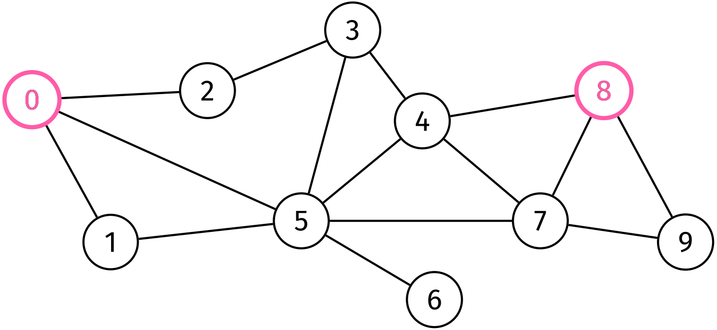


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	-1	0	0	2	5	0	5	5	4	7

Path-Finding with BFS

Example: Shortest path from 0 to 8

0 8

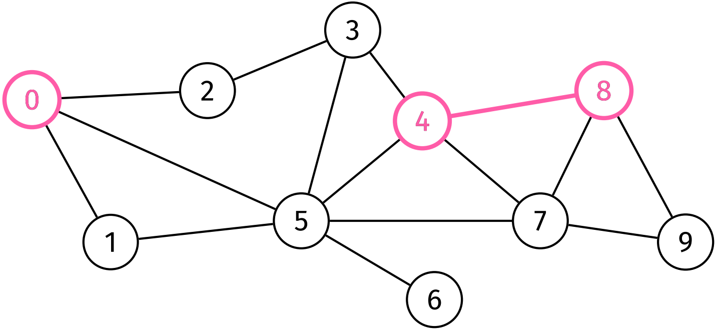


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	-1	0	0	2	5	0	5	5	4	7

Path-Finding with BFS

Example: Shortest path from 0 to 8

0 4 → 8

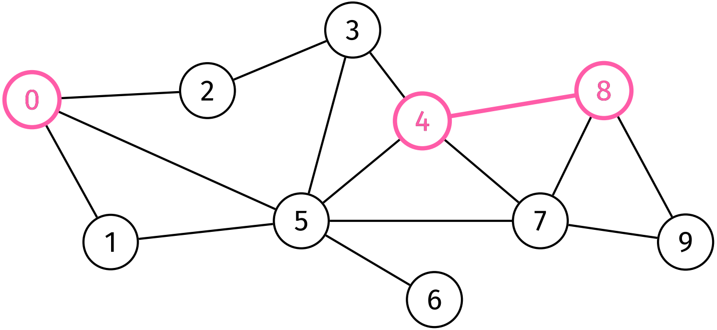


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	-1	0	0	2	5	0	5	5	4	7

Path-Finding with BFS

Example: Shortest path from 0 to 8

0 4 → 8

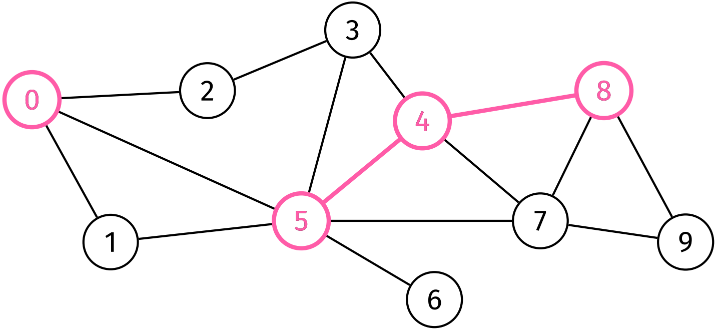


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	-1	0	0	2	5	0	5	5	4	7

Path-Finding with BFS

Example: Shortest path from 0 to 8

0 5 → 4 → 8

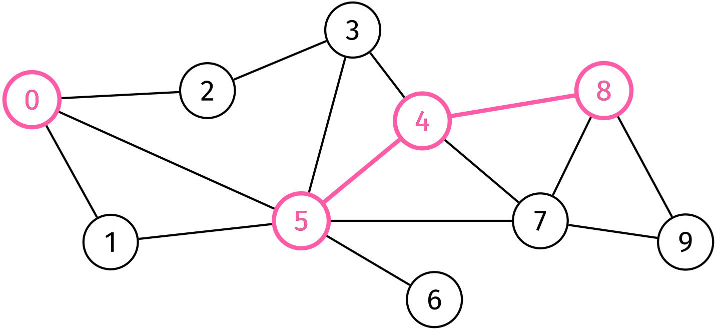


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	-1	0	0	2	5	0	5	5	4	7

Path-Finding with BFS

Example: Shortest path from 0 to 8

0   5 → 4 → 8

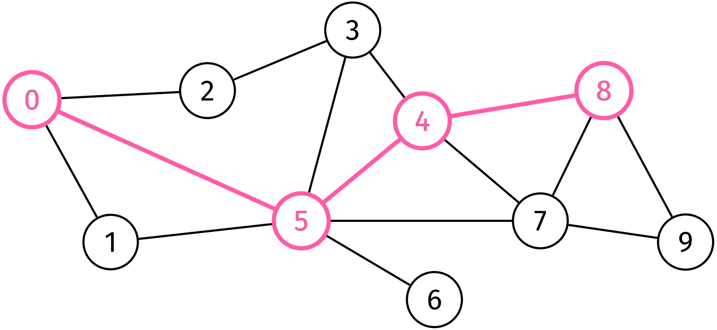


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	-1	0	0	2	5	0	5	5	4	7

Path-Finding with BFS

Example: Shortest path from 0 to 8

$$0 \longrightarrow 5 \rightarrow 4 \rightarrow 8$$



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	-1	0	0	2	5	0	5	5	4	7

Path-Finding with BFS

```
findPathBfs($G$, $src$, $dest$):
    Inputs: graph $G$, vertices $src$ and $dest$

    ... BFS starting from src ...

    if predecessor[$dest$] $\neq$ -1:
        $v$ = $dest$
        while $v \neq src$:
            print $v$, "<-"
            $v$ = predecessor[$v$]

        print $src$
```

Depth-First Search

Depth-first search goes as far down one path
as possible until it reaches a dead end,
then backtracks until it finds a new path to take,
then repeats

DFS can be implemented recursively or iteratively.

Recursive Depth-First Search

Depth-first search is described recursively as:

1. Mark current vertex as visited
    - The first time, this is the starting vertex
2. For each neighbour of the current vertex:
    1. If it has not been visited:
        1. Recursively traverse starting from that vertex

The recursion naturally induces backtracking.

Recursive Depth-First Search

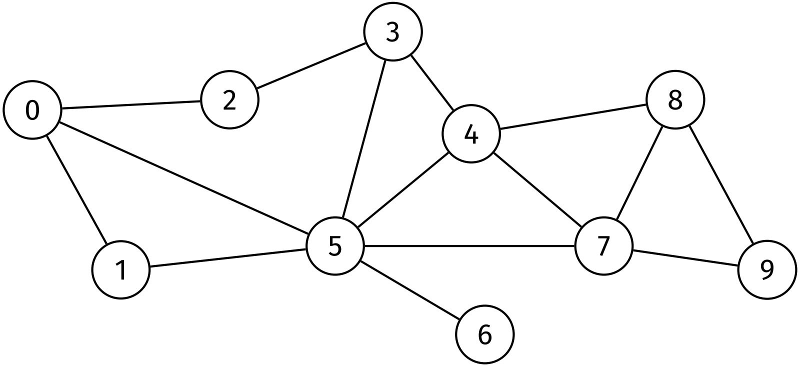
Pseudocode

```
dfs($G$, $src$):
    **Inputs:** graph $G$, starting vertex $src$

    create visited array, initialised to false
    dfsRec($G$, $src$, visited)

dfsRec($G$, $v$, visited):
    **Inputs:** graph $G$, vertex $v$, visited array

    visited[$v$] = true // "visit" v
    for each neighbour $w$ of $v$:
        if visited[$w$] $=$ false:
            dfsRec($G$, $w$, visited)
```

Recursive Depth-First Search

Example

DFS starting at 0



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	0	0	0	0	0	0	0	0	0	0

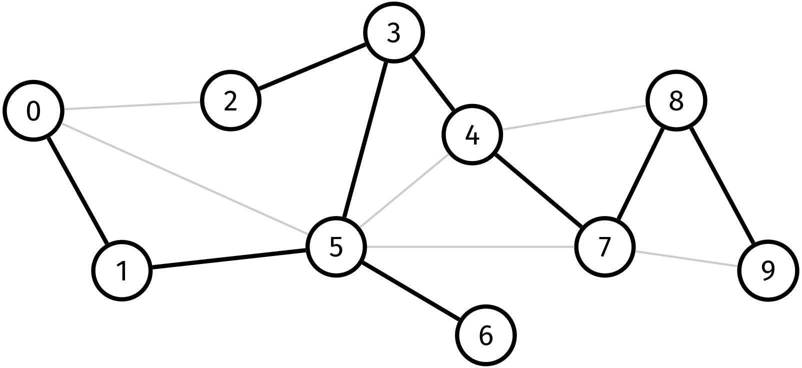
visit order

call stack

Recursive Depth-First Search

Example

Done



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1

visit order 0 1 5 3 2 4 7 8 9 6

call stack

Recursive Depth-First Search

Analysis

Recursive DFS is $O(V + E)$ when using the adjacency list representation:

- Each vertex is visited at most once $\Rightarrow O(V)$
    - Function is called on each vertex at most once
- For each vertex, all of its edges are considered once $\Rightarrow O(E)$

Path-Checking with Recursive DFS

Recursive DFS can be adapted to check if a path exists between two vertices.

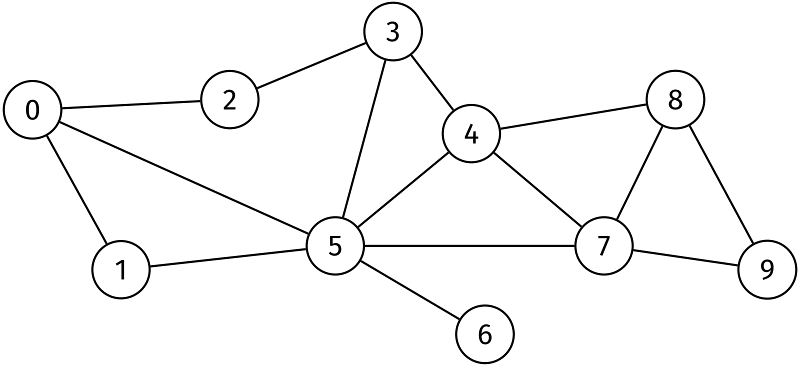
Idea:

- To check if a path exists between src and $dest$:
  - If $src = dest$, then there is a path (the empty path)
  - Otherwise, for each neighbour of src, recursively check if there is a path from that neighbour to $dest$

Path-Checking with Recursive DFS

Example

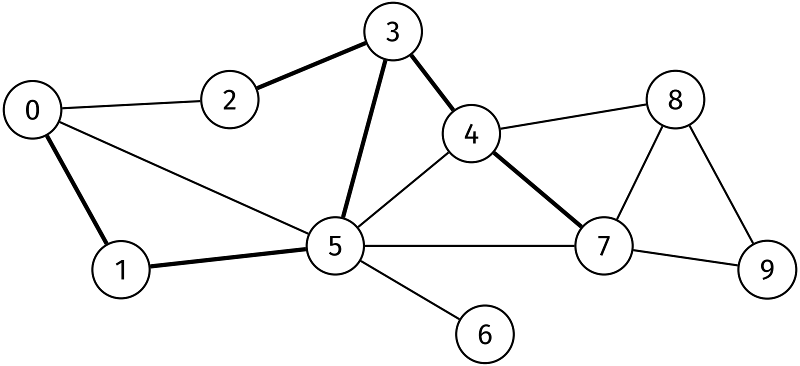
Does there exist a path between 0 and 7 in this graph?



Path-Checking with Recursive DFS

Example

Answer: Yes



Path-Checking with Recursive DFS

Pseudocode

```
hasPath($G$, $src$, $dest$):
    **Inputs:** graph $G$, vertices $src$ and $dest$
    **Output:** true if there is a path from $src$ to $dest$
            false otherwise

    create visited array, initialised to false
    **return** dfsHasPath($G$, $src$, $dest$, visited)

dfsHasPath($G$, $v$, $dest$, visited):
    **Inputs:** graph $G$, vertices $v$ and $dest$, visited array

    visited[$v$] = true
    **if** $v = dest$:
        **return** true

    **for each** neighbour $w$ of $v$:
        **if** visited[$w$] $=$ false:
            **if** dfsHasPath($G$, $w$, $dest$, visited):
                **return** true

    **return** false
```

Path-Checking with Recursive DFS

Analysis

$O(V+E)$ when using the adjacency list representation:

- Algorithm is just a modified recursive DFS with return statements

Path-Finding with Recursive DFS

Knowing whether a path exists can be useful.

Knowing what the path is can be even more useful.

Idea:

- Record the predecessor of each vertex during the DFS
- Trace backwards through the path after the DFS

Path-Finding with Recursive DFS

Pseudocode

```
findPath($G$, $src$, $dest$):
    Inputs: graph $G$, vertices $src$ and $dest$

    create predecessor array, initialised to -1
    predecessor[$src$] = $src$

    if dfsFindPath($G$, $src$, $dest$, predecessor):
        $v$ = $dest$
        while $v \neq src$:
            print $v$, "<-"
            $v$ = predecessor[$v$]
        print $src$
```

Path-Finding with Recursive DFS

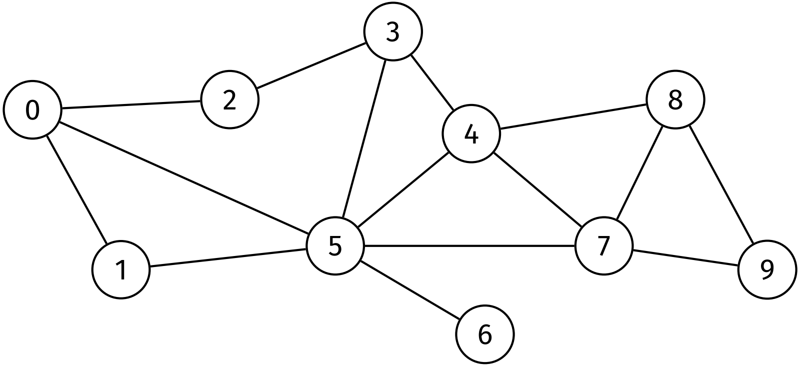
Pseudocode

```
dfsFindPath($G$, $v$, $dest$, predecessor):
    if $v = dest$:
        return true

    for each neighbour $w$ of $v$:
        if predecessor[$w$] $= -1$:
            predecessor[$w$] = $v$
            if dfsFindPath($G$, $w$, $dest$, predecessor):
                return true

    return false
```

Path-Finding with Recursive DFS

Example

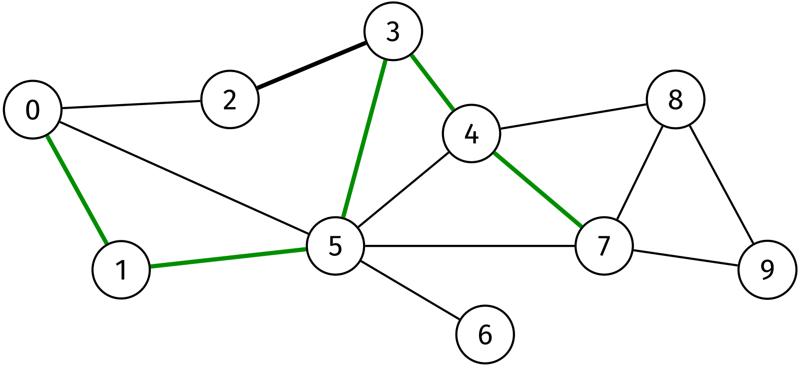
Find a path from 0 to 7



Path-Finding with Recursive DFS

Example

Path found:



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
pred	0	0	3	5	3	1	-1	4	-1	-1

Clearly, DFS is not guaranteed to find the shortest path.

Iterative Depth-First Search

DFS can be implemented iteratively.

- Similar to BFS, with a few crucial differences:
    - DFS uses a stack instead of a queue
    - BFS marks a vertex as visited when enqueuing it
    - DFS marks a vertex as visited after popping it from the stack, instead of when pushing it onto the stack

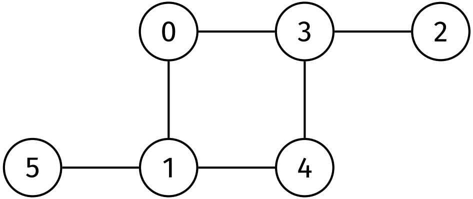
Iterative Depth-First Search

Pseudocode

```
dfs($G$, $src$):
    Inputs: graph $G$, vertex $src$

    created visited array, predecessor array and stack $S$
    for each vertex $v$ in $G$:
        visited[$v$] = false
        predecessor[$v$] = -1

    push $src$ onto $S$

    while $S$ is not empty:
        $v$ = pop from $S$
        if visited[$v$] = true:
            continue // i.e., return to start of loop

        visited[$v$] = true

        for each neighbour $w$ of $v$ where visited[$w$] = false:
            predecessor[$w$] = $v$
            push $w$ onto $S$
```

Iterative Depth-First Search

Why mark a vertex as visited after popping it, instead of when pushing it?



Iterative Depth-First Search

Analysis

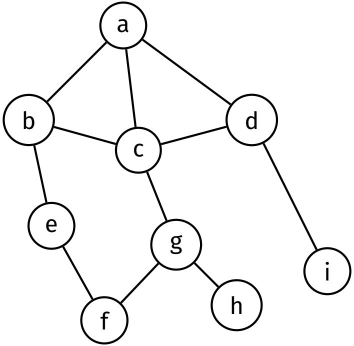
Iterative DFS is $O(V + E)$ when using the adjacency list representation.

- Typical stack implementation has $O(1)$ push and pop
- Each vertex visited at most once $\Rightarrow O(V)$
- For each vertex, all of its edges are considered $\Rightarrow O(E)$

Spanning Trees

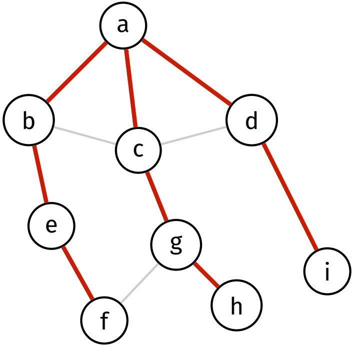
The edges traversed in a graph traversal form a spanning tree.

Consider the following graph:

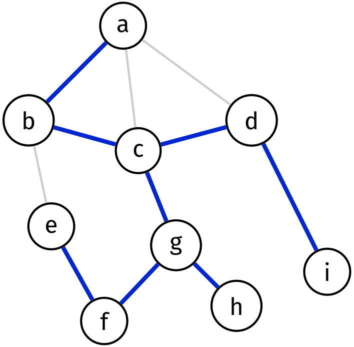


Spanning Trees

A traversal starting at vertex ‘a’ forms the following spanning trees:



Breadth-first search



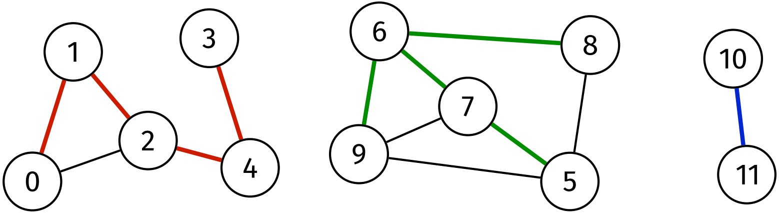
Depth-first search

Unconnected Graphs

If a graph is not connected,
a graph traversal starting from a given vertex
will not traverse the entire graph

Solution
After initial traversal is complete,
perform traversal again on an unvisited vertex,
repeat until all vertices are visited

This produces a spanning forest



Unconnected Graphs

```
dfs($G$):
    **Inputs:** graph $G$

    create predecessor array, initialised to -1

    for each vertex $v$ in $G$:
        if predecessor[$v$] $= -1$:
            dfsRec($G$, $v$, predecessor)

    ...
```

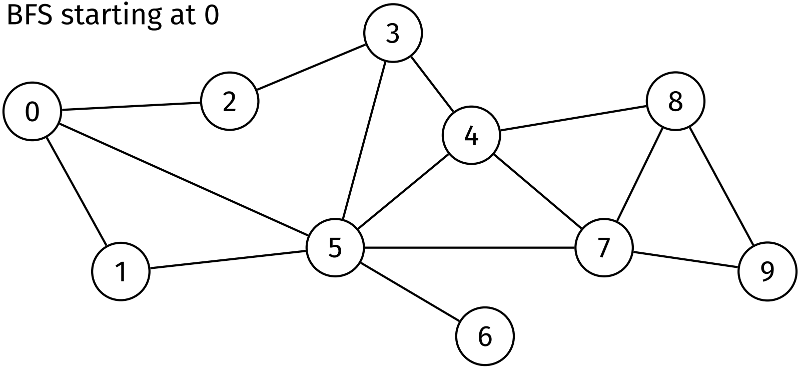
Feedback

https://forms.office.com/r/aPF09YHZ3X

Appendix

BFS Example

BFS starting at 0



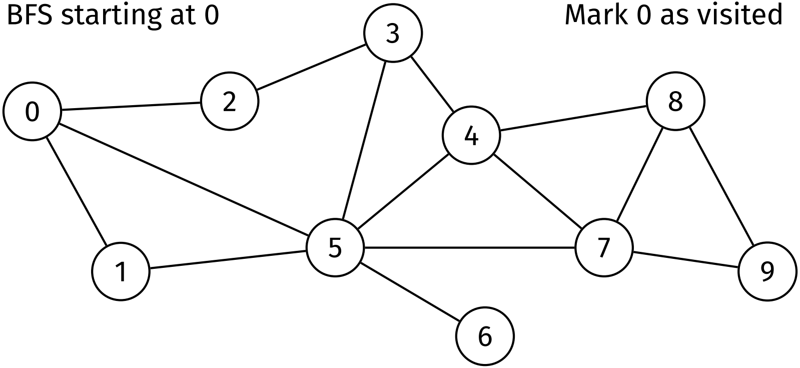
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	0	0	0	0	0	0	0	0	0	0
pred	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

queue

BFS Example

BFS starting at 0

Mark 0 as visited

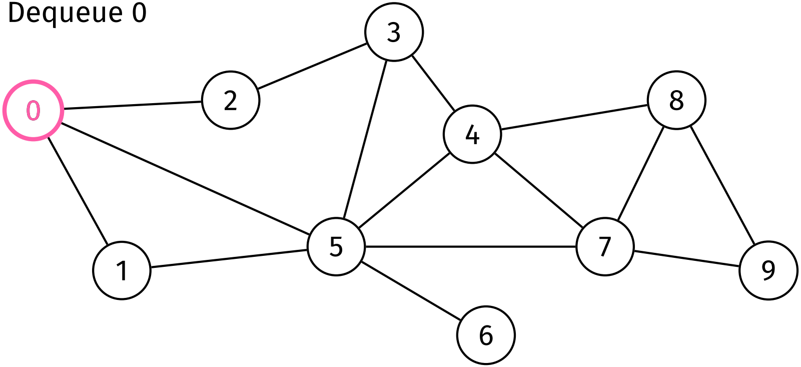


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0
pred	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

queue 0

BFS Example

Dequeue 0

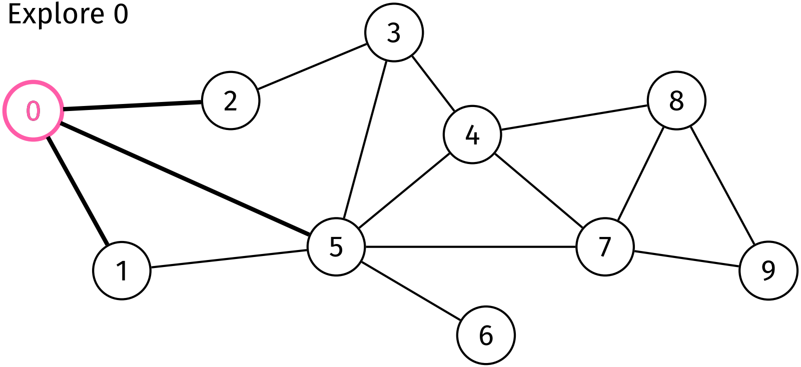


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0
pred	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

queue 0

BFS Example

Explore 0

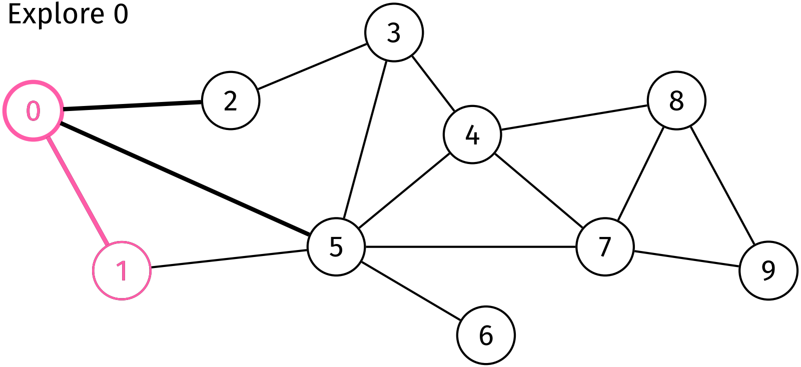


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0
pred	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

queue 0

BFS Example

Explore 0



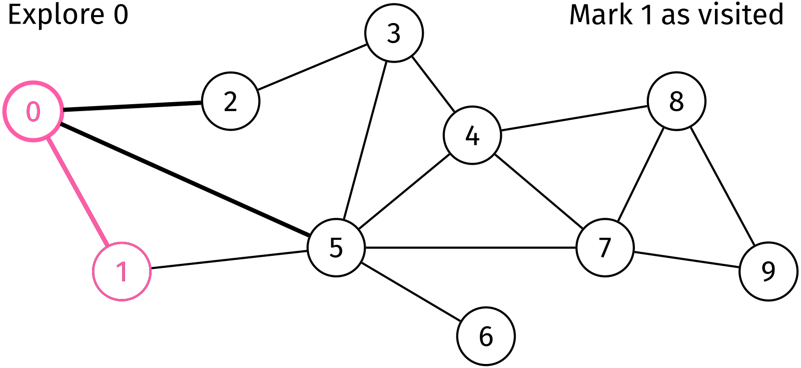
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0
pred	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

queue 0

BFS Example

Explore 0

Mark 1 as visited

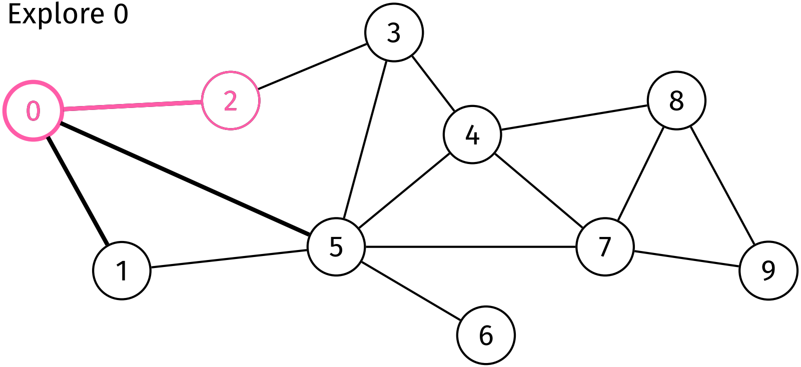


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0
pred	-1	0	-1	-1	-1	-1	-1	-1	-1	-1

queue 0 1

BFS Example

Explore 0



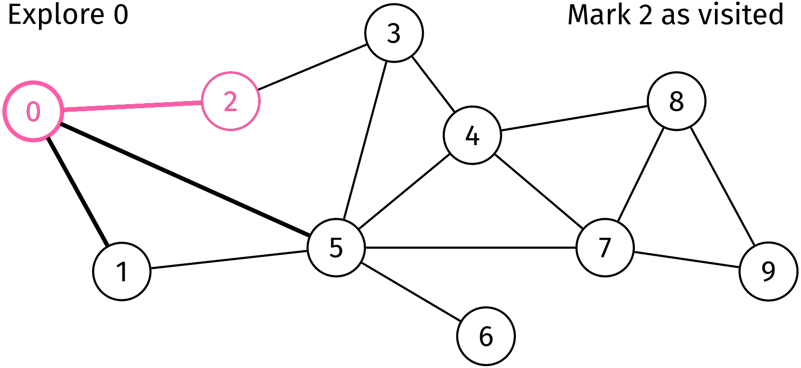
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0
pred	-1	0	-1	-1	-1	-1	-1	-1	-1	-1

queue 0 1

BFS Example

Explore 0

Mark 2 as visited

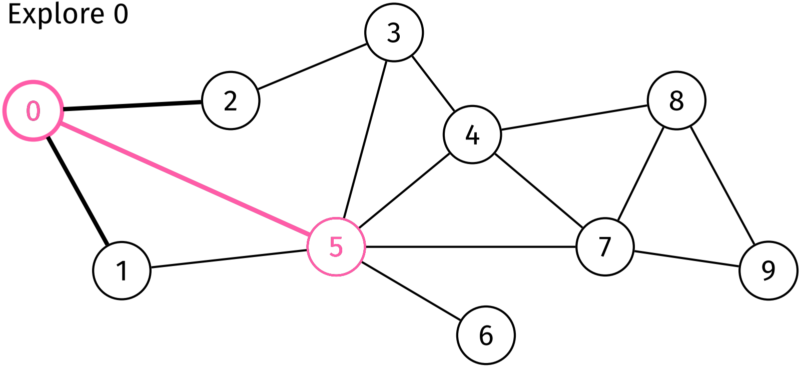


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	0	0	0	0	0
pred	-1	0	0	-1	-1	-1	-1	-1	-1	-1

queue 0 1 2

BFS Example

Explore 0



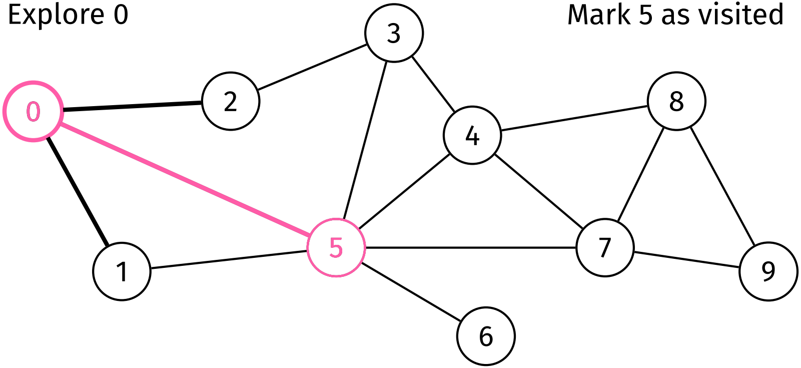
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	0	0	0	0	0
pred	-1	0	0	-1	-1	-1	-1	-1	-1	-1

queue 0 1 2

BFS Example

Explore 0

Mark 5 as visited



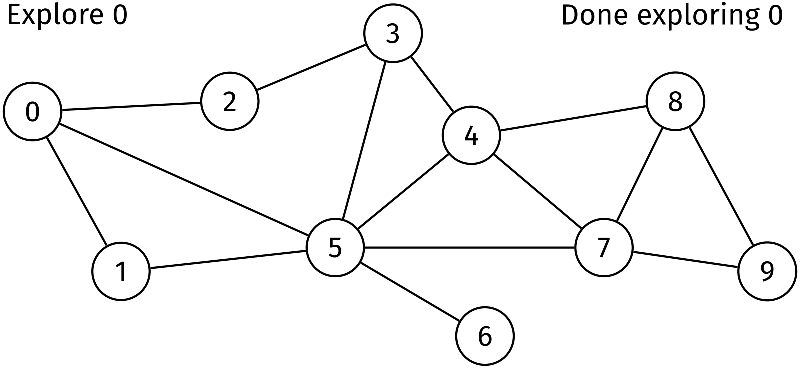
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 0

Done exploring 0

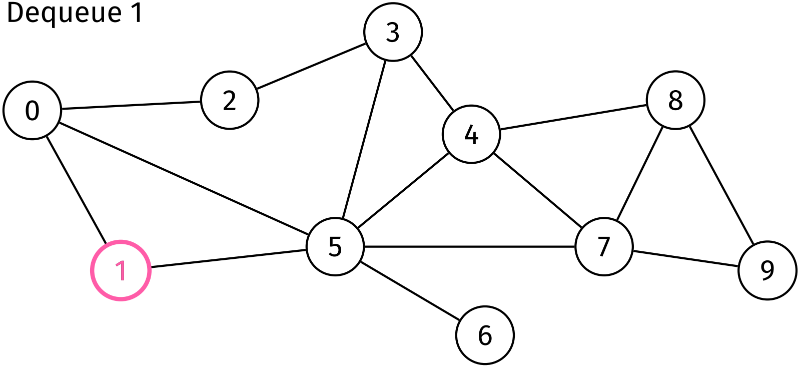


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Dequeue 1

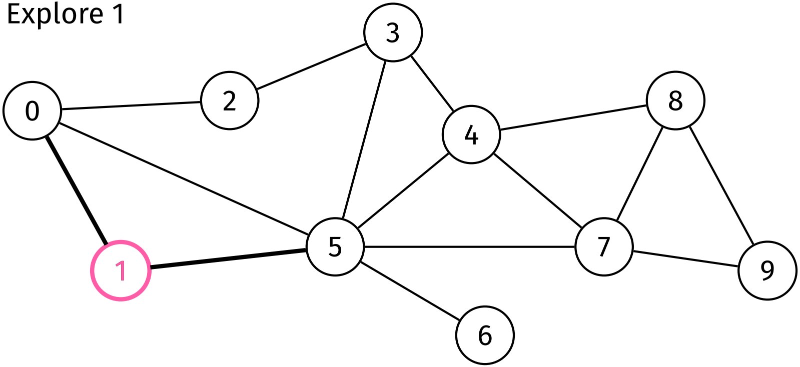


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 1



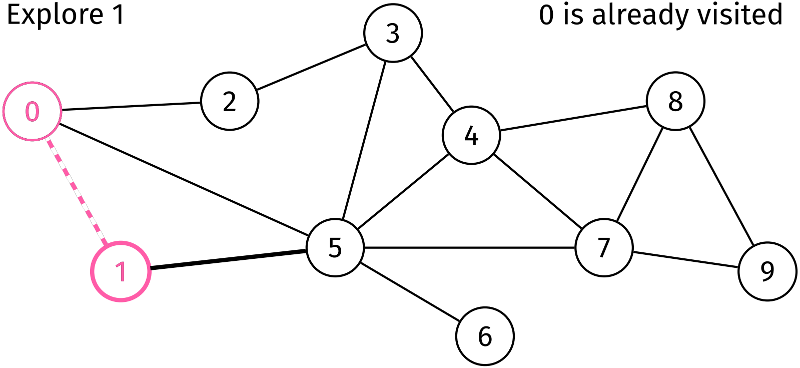
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 1

0 is already visited



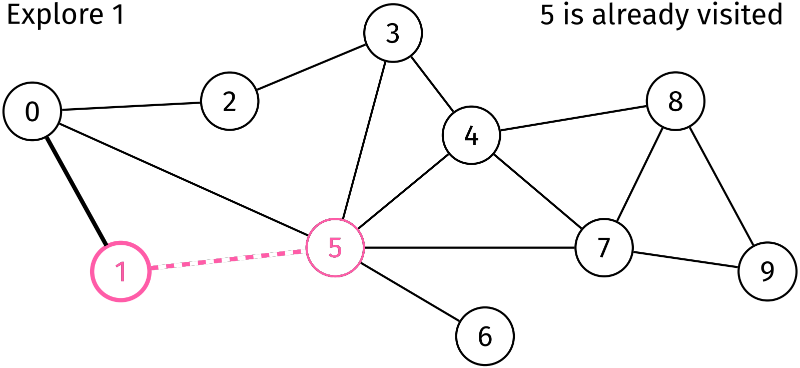
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 1

5 is already visited



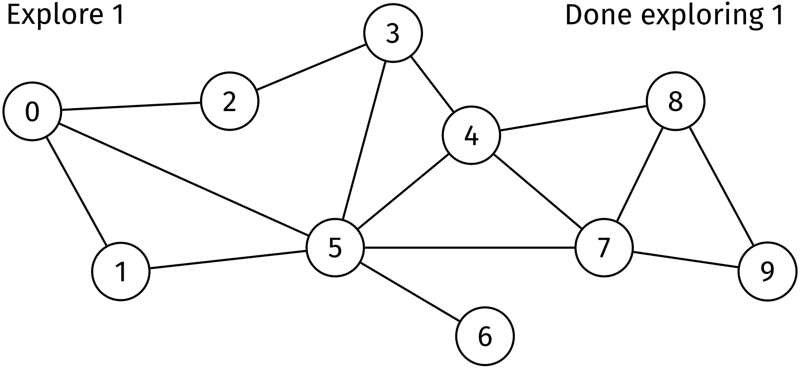
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 1

Done exploring 1

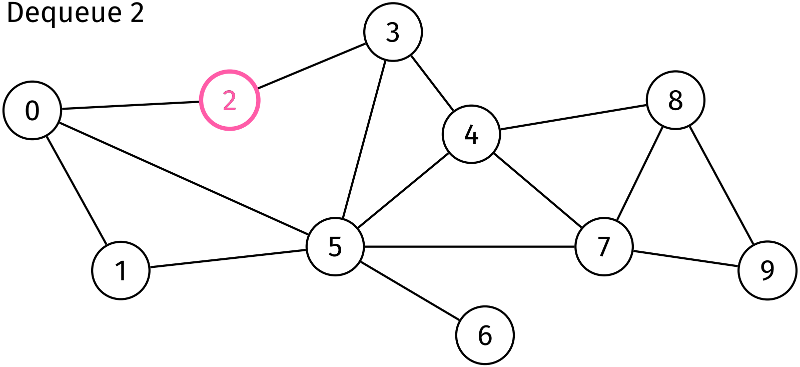


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Deque 2

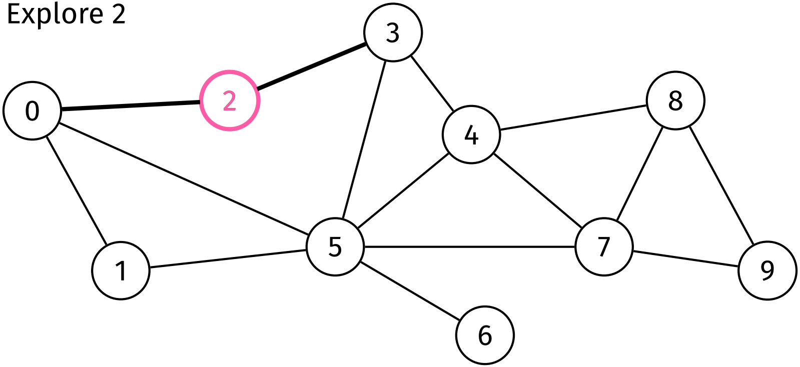


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 2



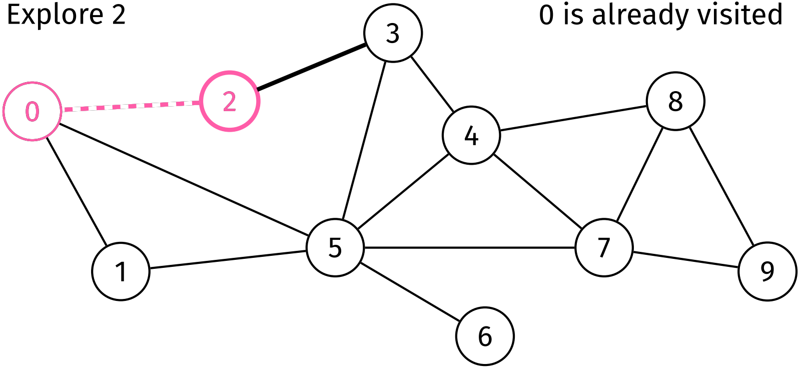
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 2

0 is already visited

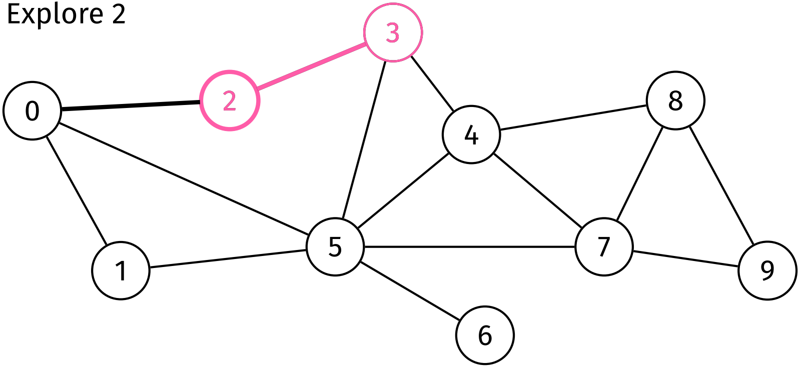


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 2



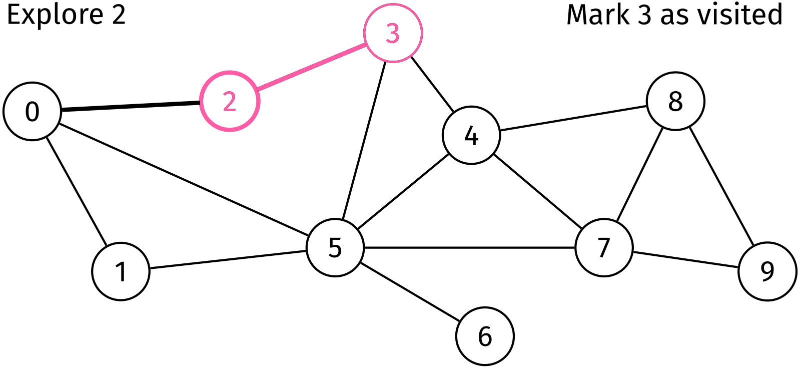
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	0	0	1	0	0	0	0
pred	-1	0	0	-1	-1	0	-1	-1	-1	-1

queue 0 1 2 5

BFS Example

Explore 2

Mark 3 as visited



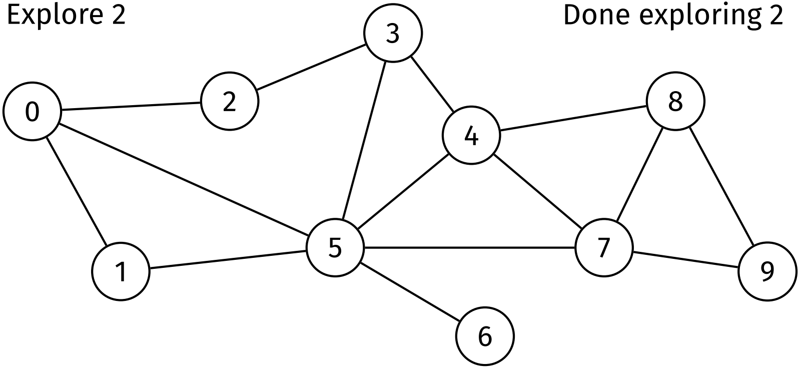
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Explore 2

Done exploring 2

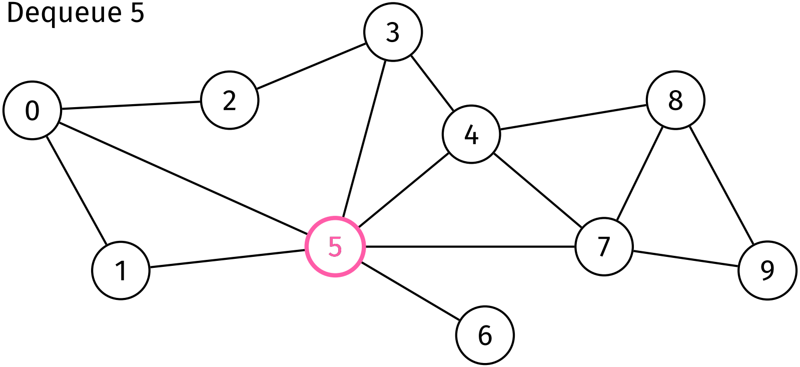


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Deqeue 5

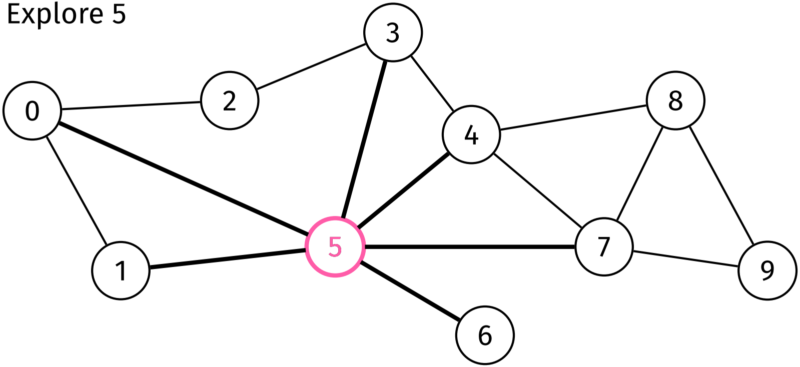


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Explore 5



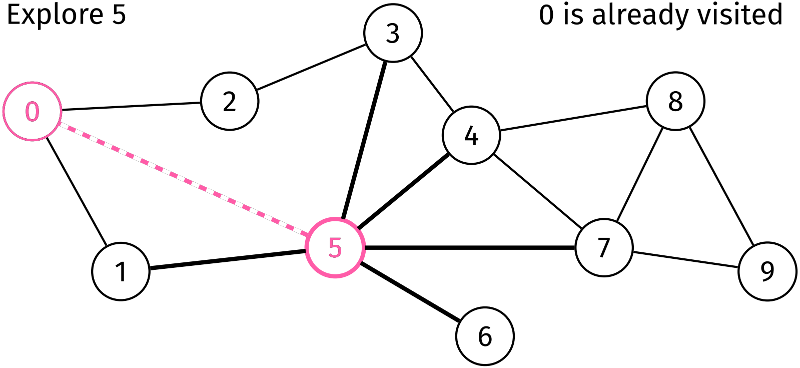
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Explore 5

0 is already visited



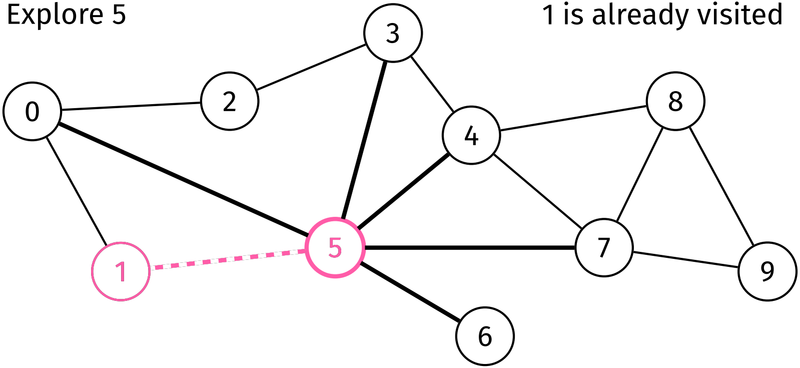
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Explore 5

1 is already visited



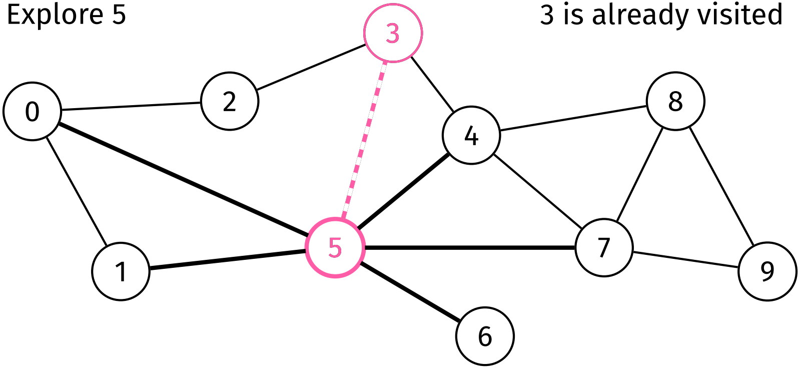
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Explore 5

3 is already visited

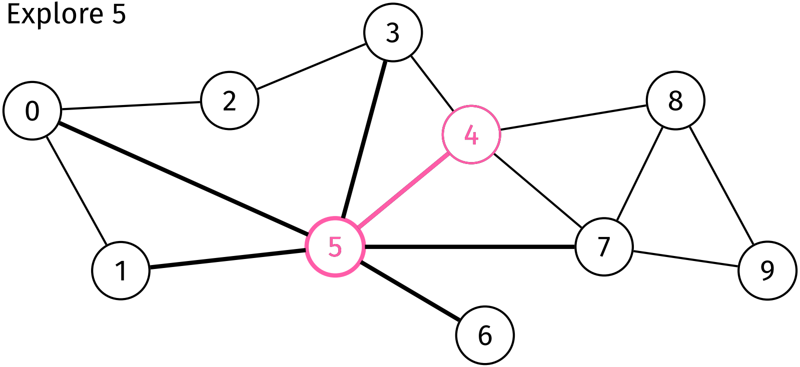


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Explore 5



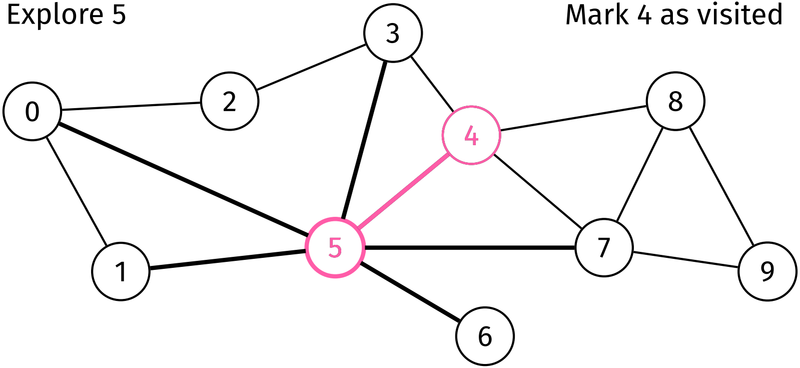
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0
pred	-1	0	0	2	-1	0	-1	-1	-1	-1

queue 0 1 2 5 3

BFS Example

Explore 5

Mark 4 as visited

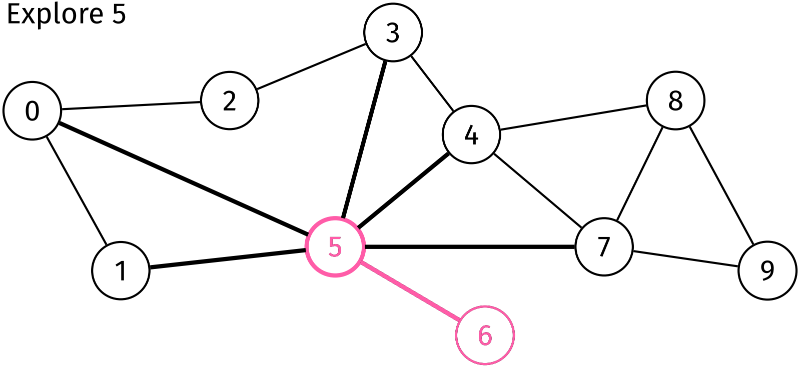


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0
pred	-1	0	0	2	5	0	-1	-1	-1	-1

queue 0 1 2 5 3 4

BFS Example

Explore 5



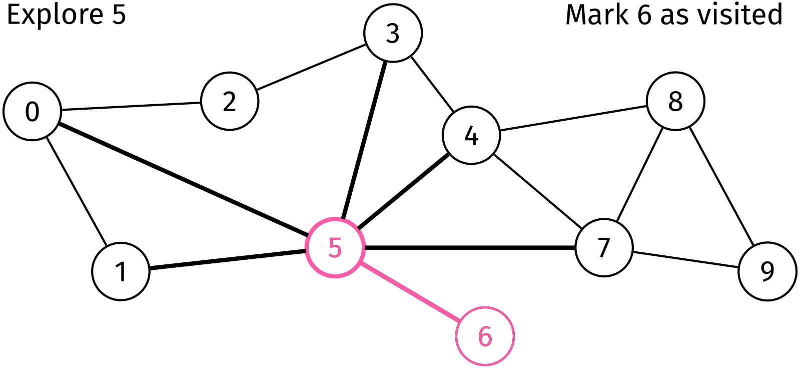
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0
pred	-1	0	0	2	5	0	-1	-1	-1	-1

queue 0 1 2 5 3 4

BFS Example

Explore 5

Mark 6 as visited

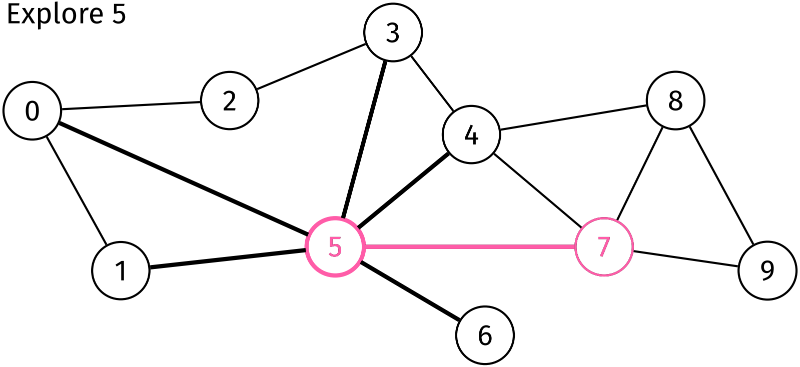


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	0	0	0
pred	-1	0	0	2	5	0	5	-1	-1	-1

queue 0 1 2 5 3 4 6

BFS Example

Explore 5



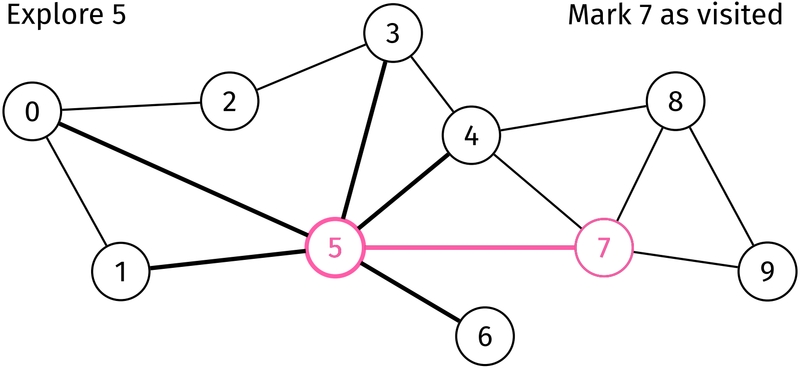
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	0	0	0
pred	-1	0	0	2	5	0	5	-1	-1	-1

queue 0 1 2 5 3 4 6

BFS Example

Explore 5

Mark 7 as visited



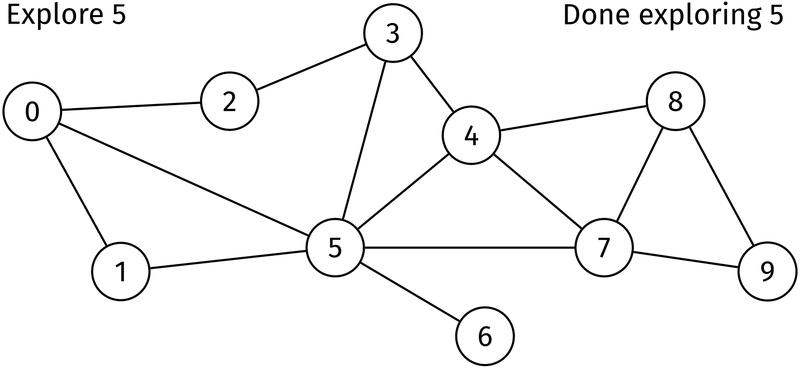
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 5

Done exploring 5

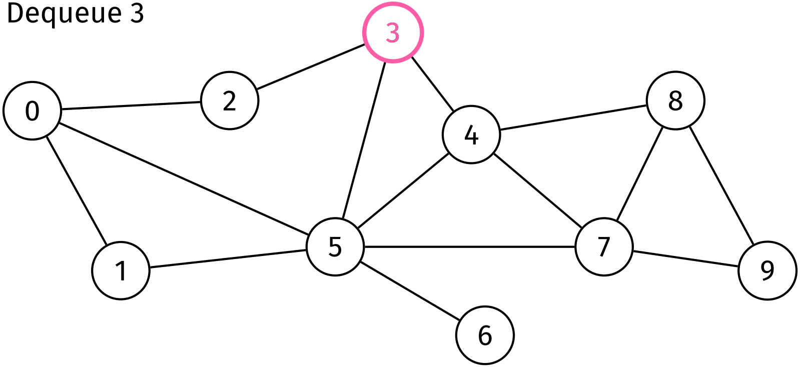


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Dequeue 3

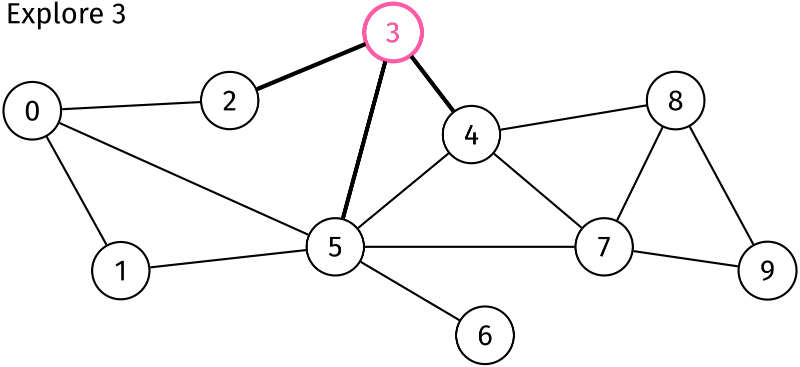


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 3



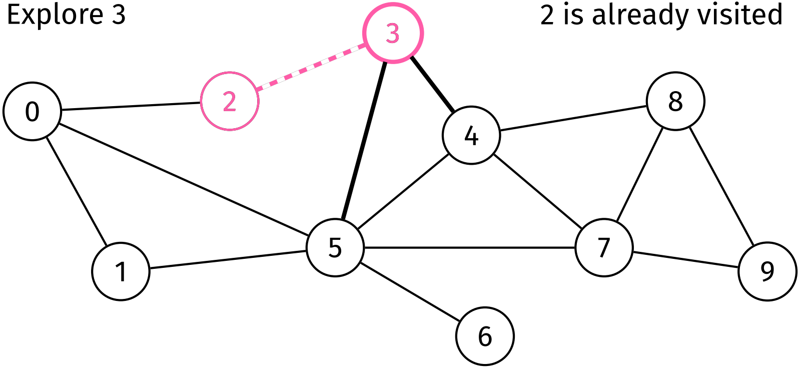
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 3

2 is already visited



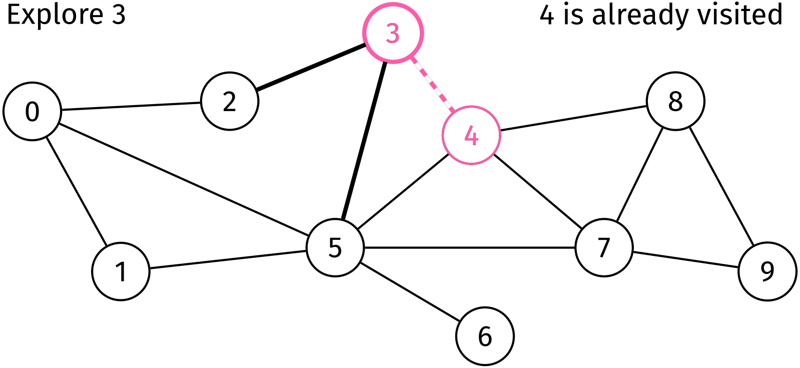
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 3

4 is already visited



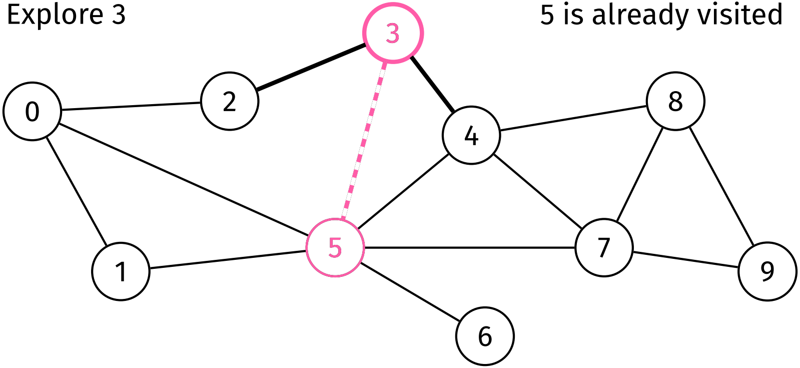
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 3

5 is already visited



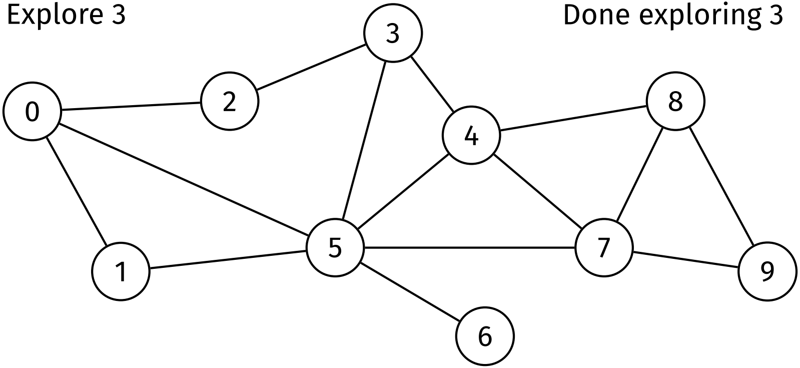
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 3

Done exploring 3

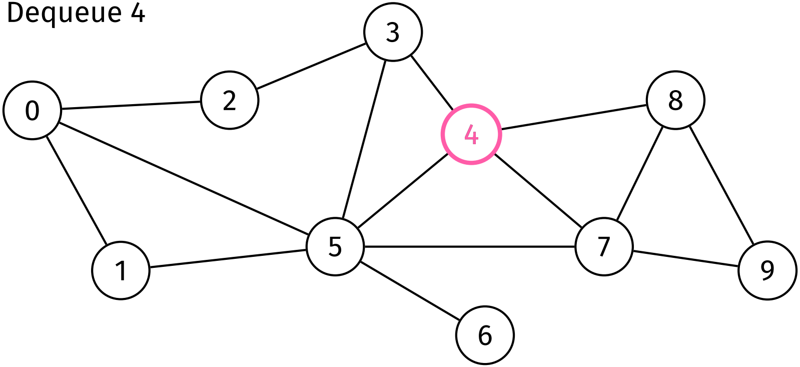


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Dequeue 4

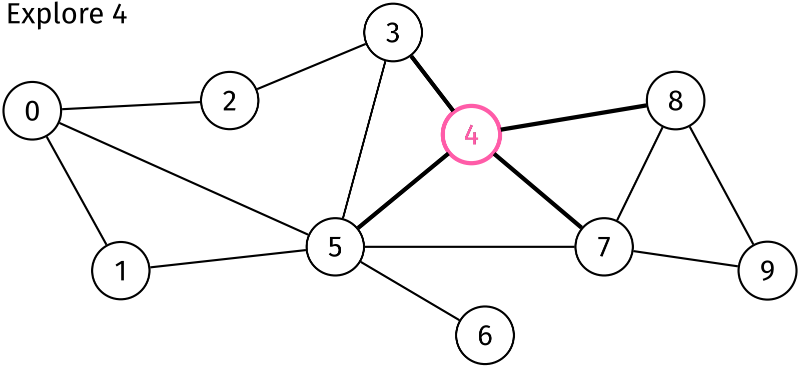


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 4



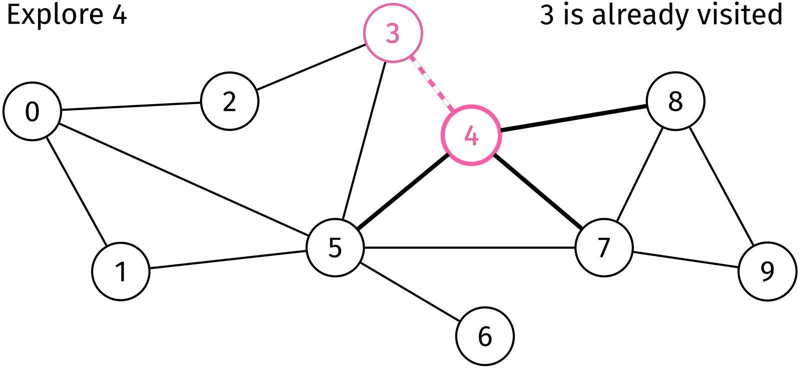
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 4

3 is already visited

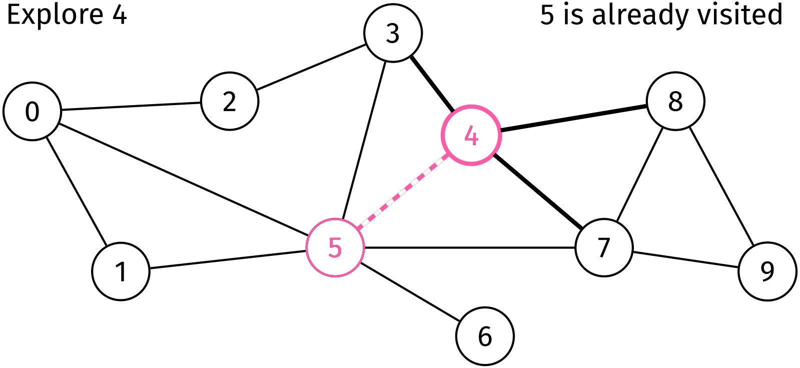


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 4



5 is already visited

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

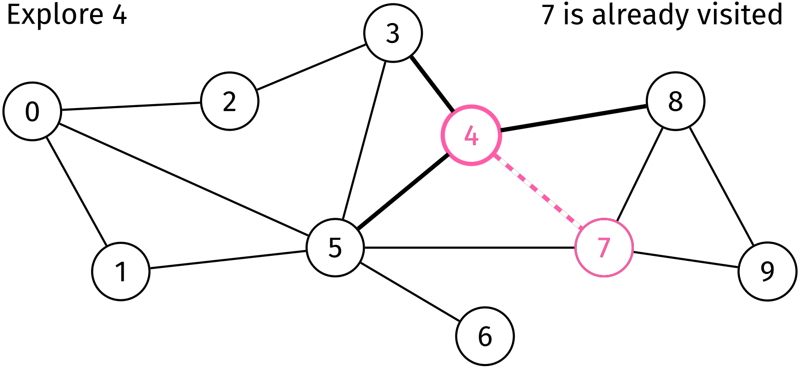
queue 0 1 2 5 3 4 6 7

Graph Traversal
BFS
DFS
Ideas/Issues
Appendix
BFS Example
DFS Example
Path-Checking Example

BFS Example

Explore 4

7 is already visited

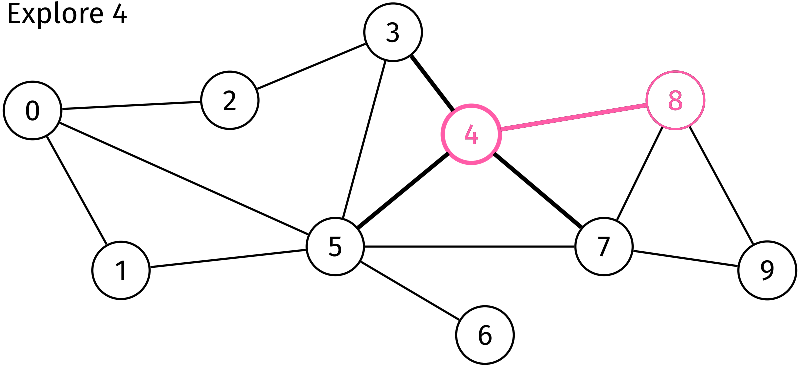


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 4



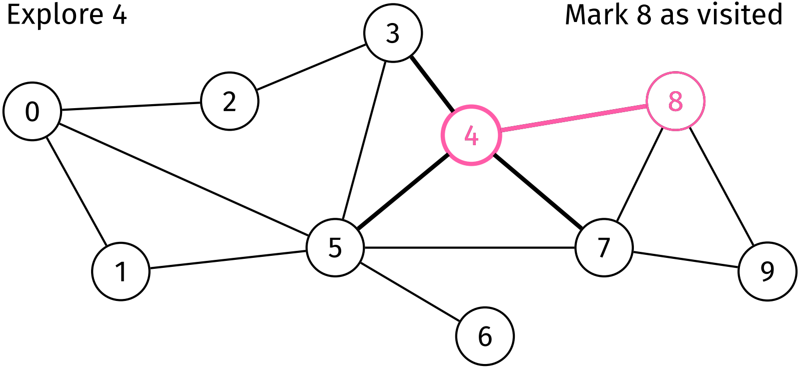
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	0	0
pred	-1	0	0	2	5	0	5	5	-1	-1

queue 0 1 2 5 3 4 6 7

BFS Example

Explore 4

Mark 8 as visited



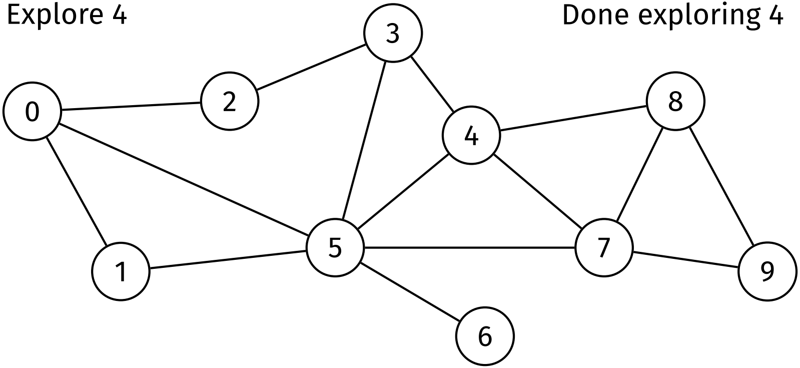
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 4

Done exploring 4

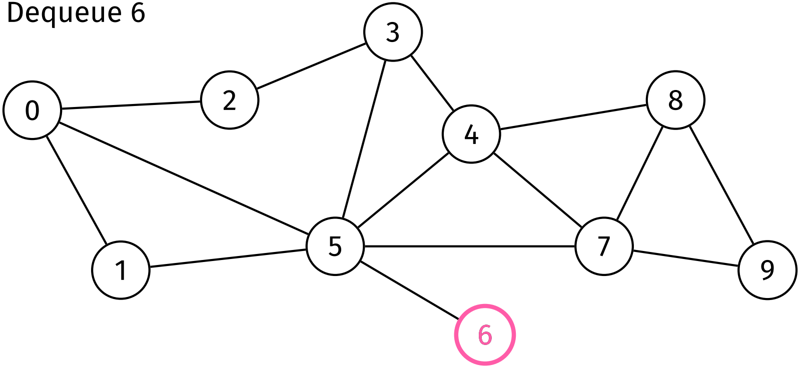


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Deque 6

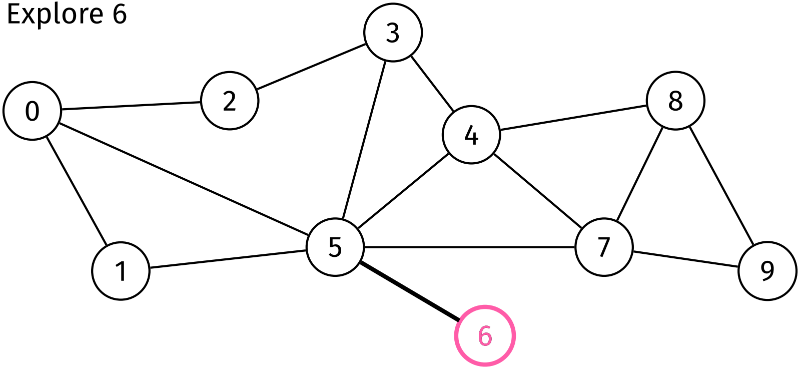


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 6



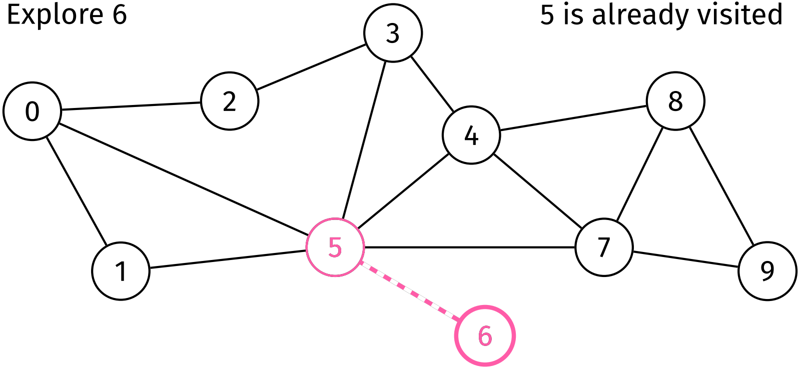
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 6

5 is already visited



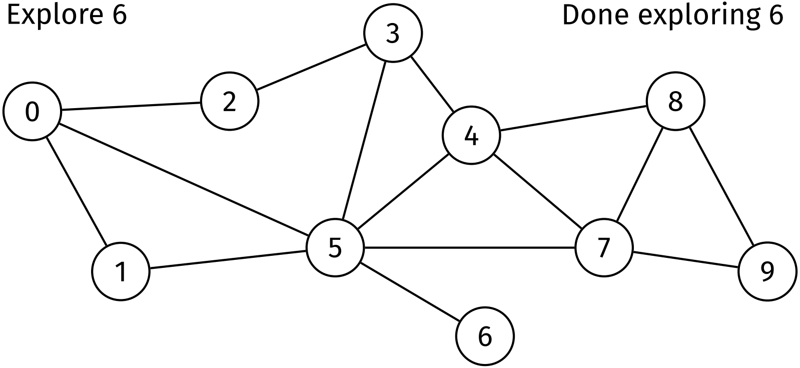
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 6

Done exploring 6

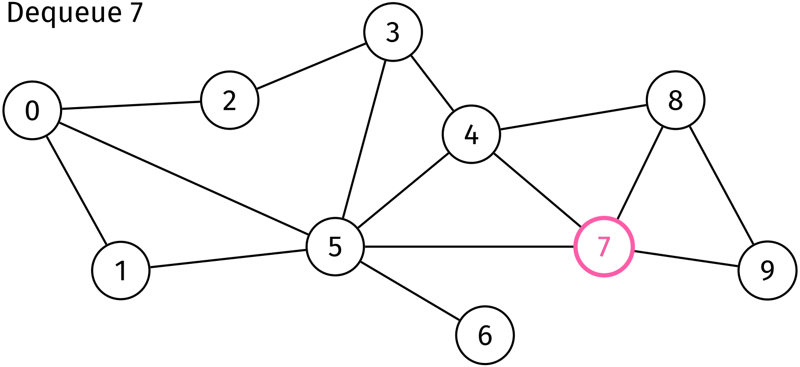


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Dequeue 7

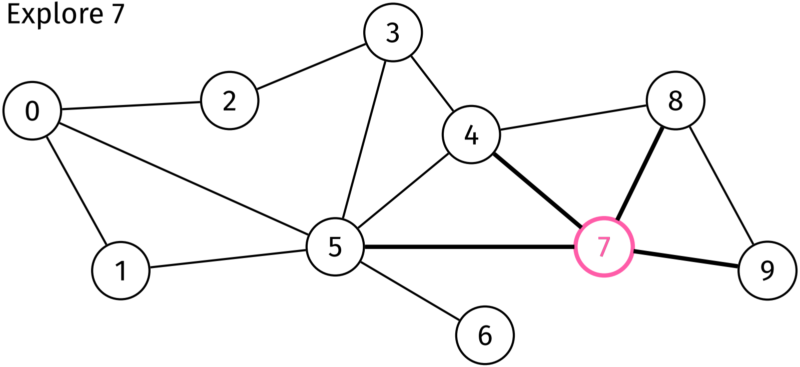


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 7



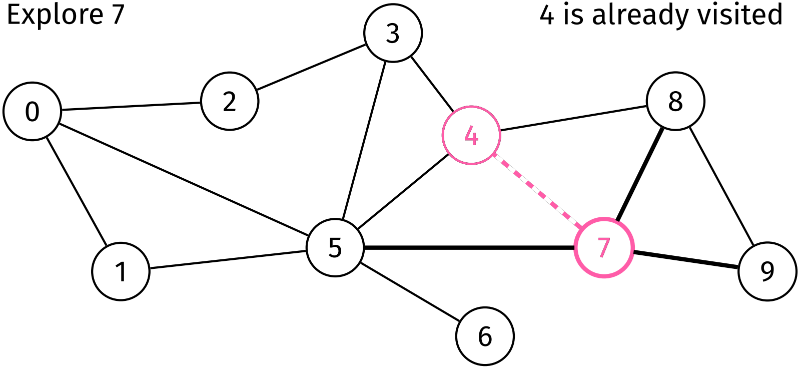
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 7

4 is already visited



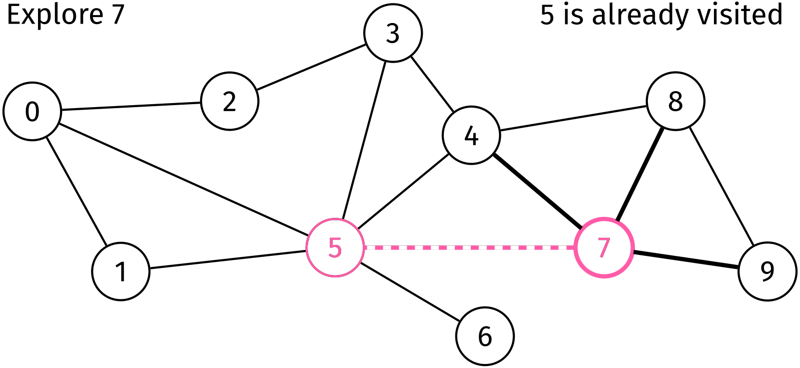
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 7

5 is already visited



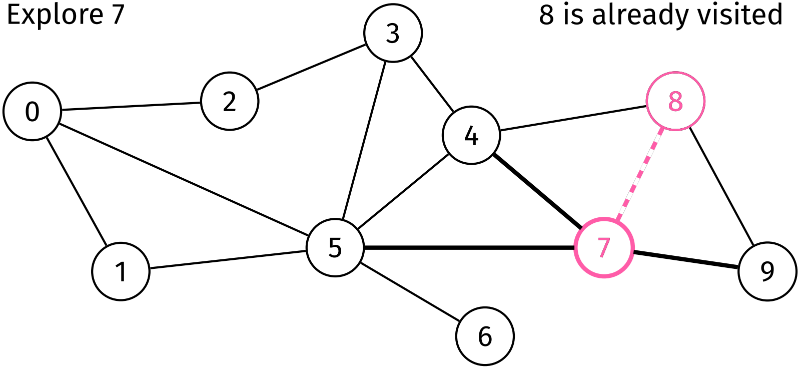
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 7

8 is already visited

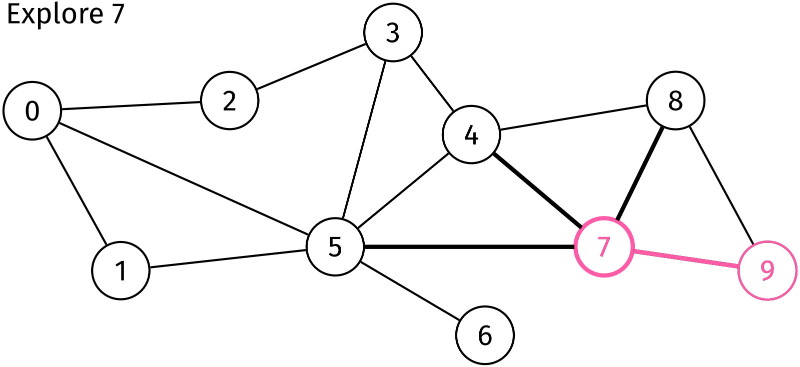


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 7



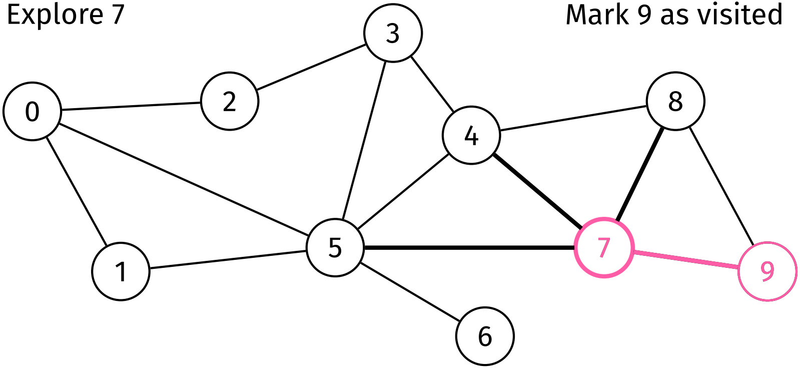
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	0
pred	-1	0	0	2	5	0	5	5	4	-1

queue 0 1 2 5 3 4 6 7 8

BFS Example

Explore 7

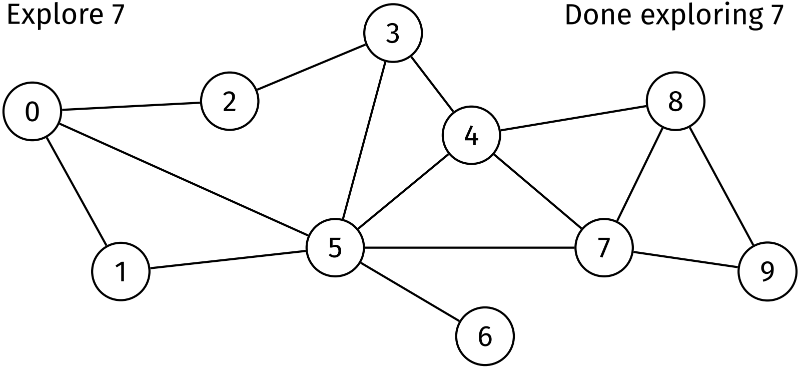
Mark 9 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example



Explore 7

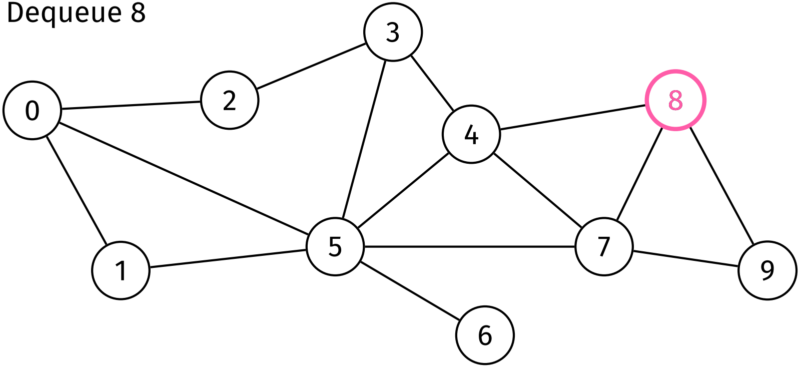
Done exploring 7

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Dequeue 8

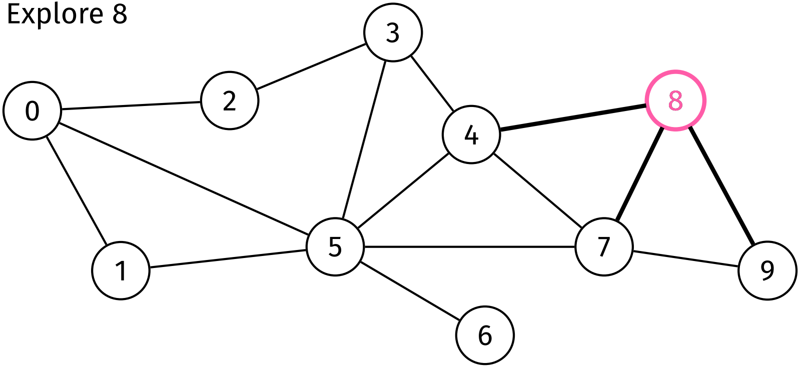


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 8



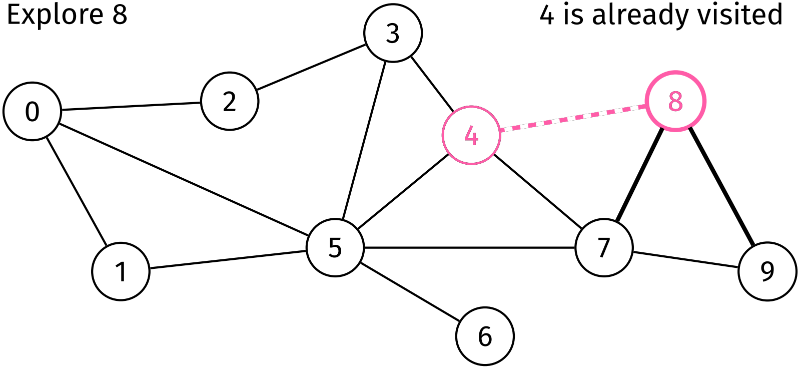
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 8

4 is already visited



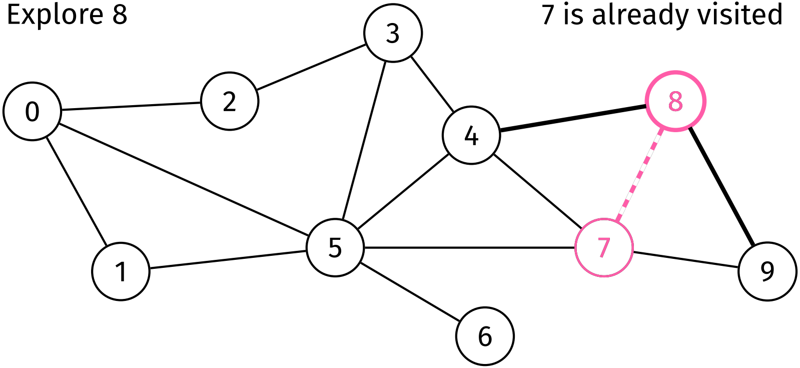
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 8

7 is already visited



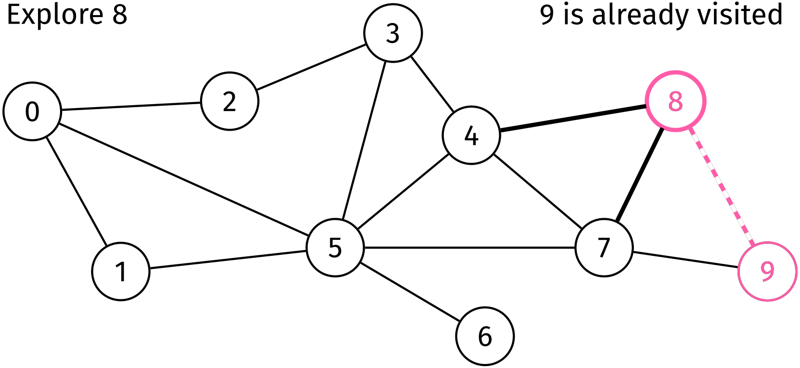
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 8

9 is already visited



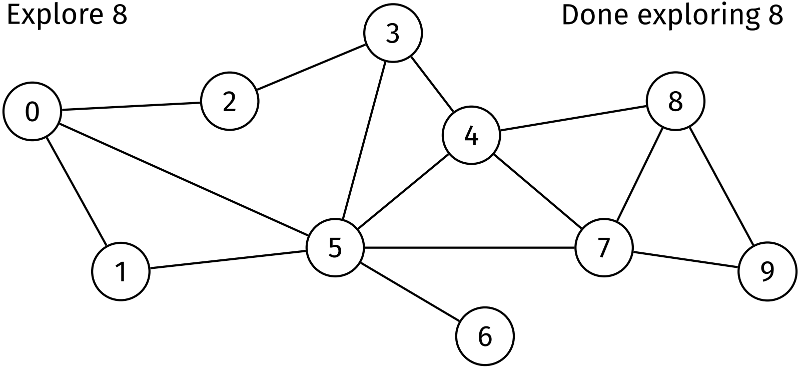
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 8

Done exploring 8

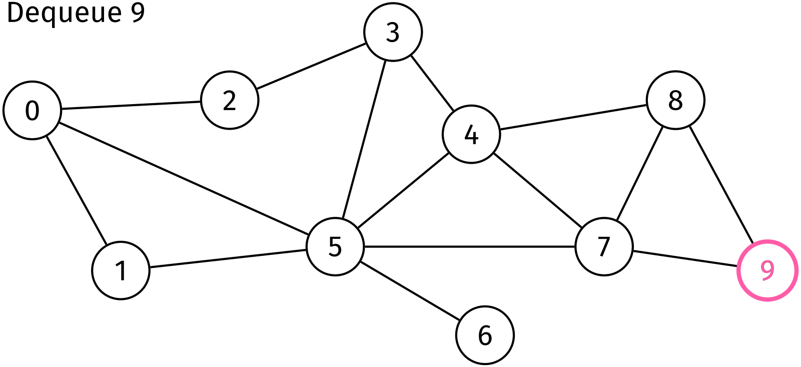


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Dequeue 9

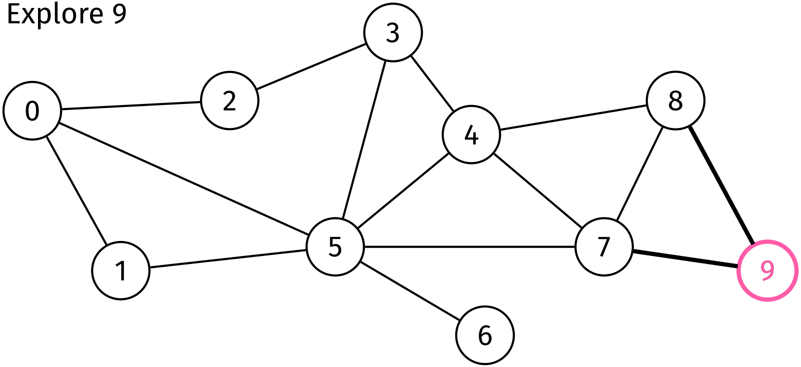


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 9



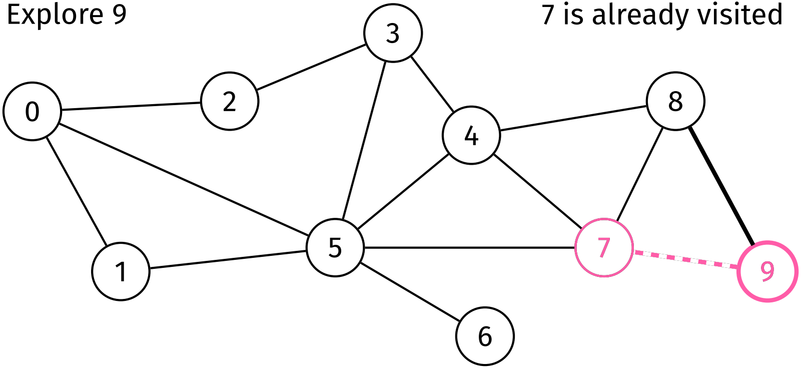
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 9

7 is already visited



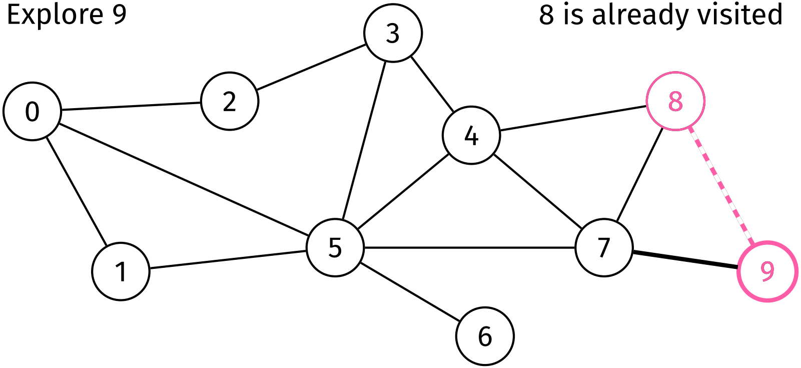
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 9

8 is already visited



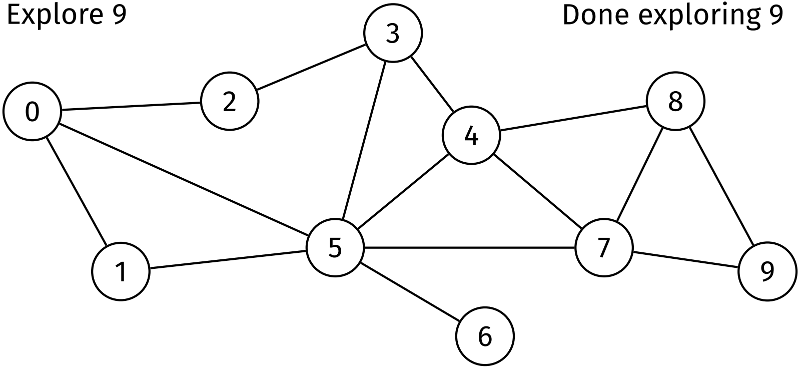
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Explore 9

Done exploring 9

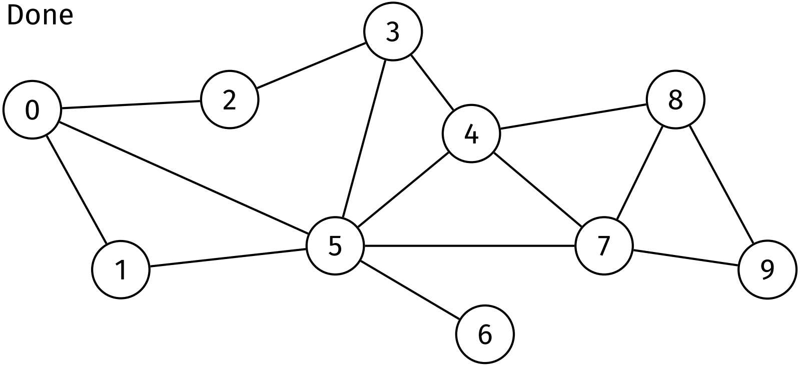


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

BFS Example

Done

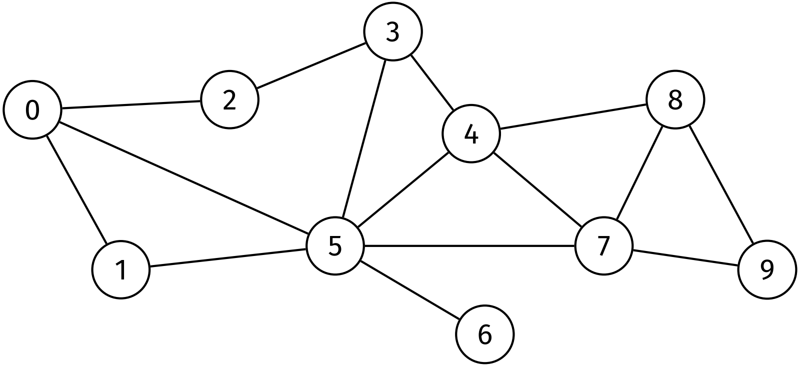


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1
pred	-1	0	0	2	5	0	5	5	4	7

queue 0 1 2 5 3 4 6 7 8 9

DFS Example

DFS starting at 0

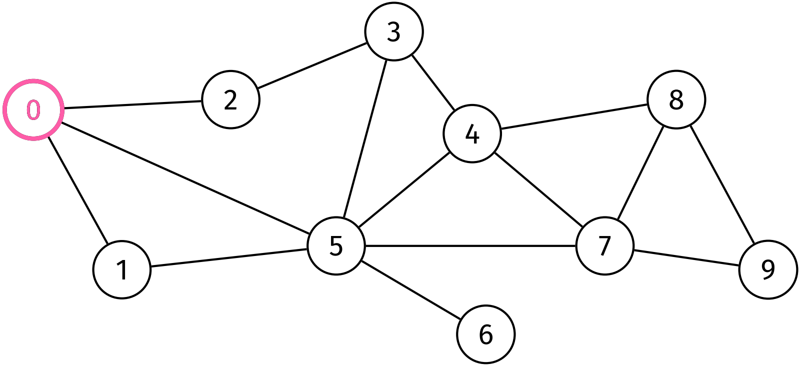


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	0	0	0	0	0	0	0	0	0	0

visit order

call stack

DFS Example



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	0	0	0	0	0	0	0	0	0	0

visit order

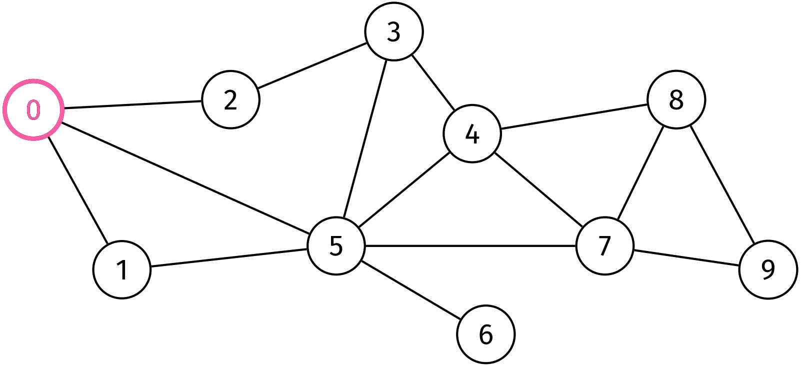
dfs(0)

call stack

Graph Traversal
BFS
DFS
Ideas/Issues
Appendix
BFS Example
DFS Example
Path-Checking Example

DFS Example

Mark 0 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0

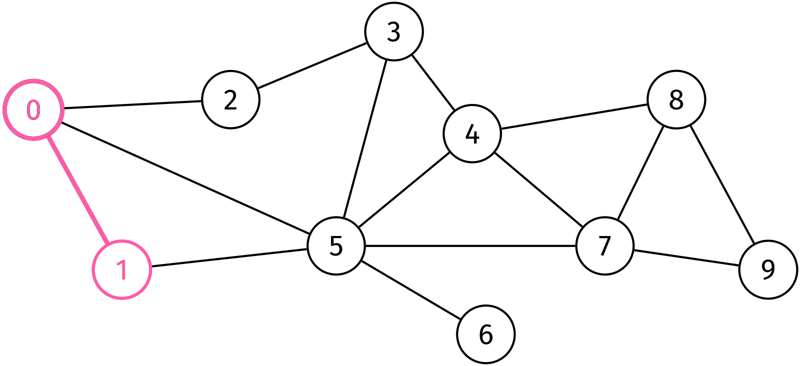
visit order 0

dfs(0)

call stack

DFS Example

1 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0

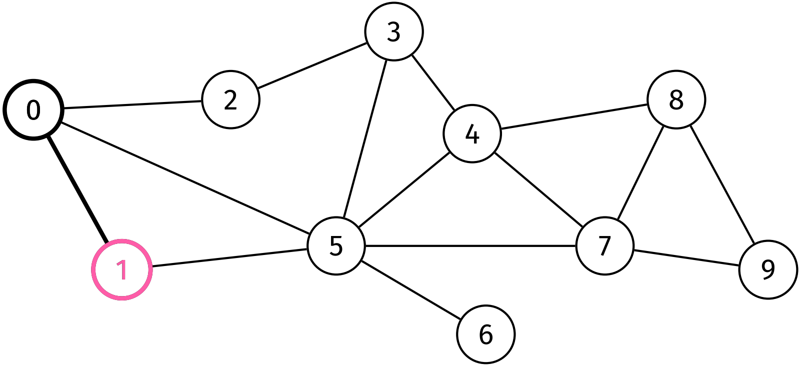
visit order 0

dfs(0)

call stack

DFS Example

Recurse into 1



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0

visit order 0

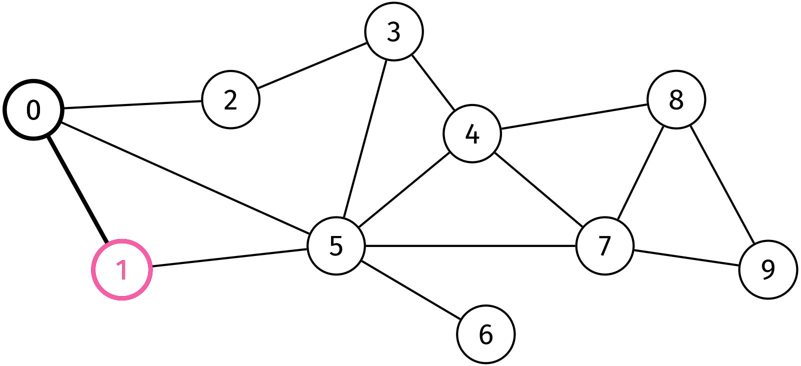
dfs(1)

dfs(0)

call stack

DFS Example

Mark 1 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0

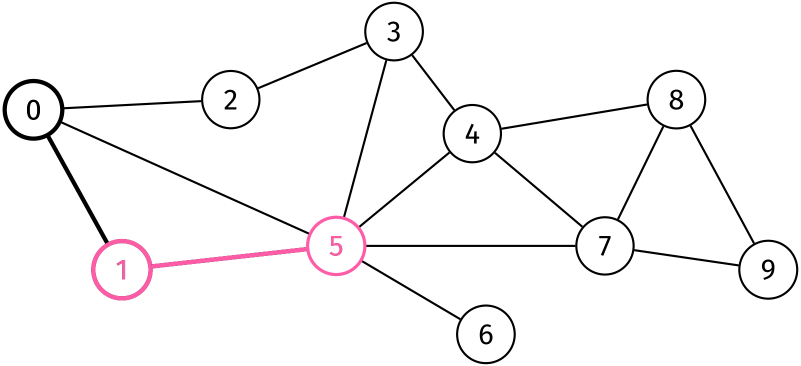
visit order 0 1

dfs(1)
dfs(0)
call stack

Graph Traversal
BFS
DFS
Ideas/Issues
Appendix
BFS Example
DFS Example
Path-Checking Example

DFS Example

5 has not been visited



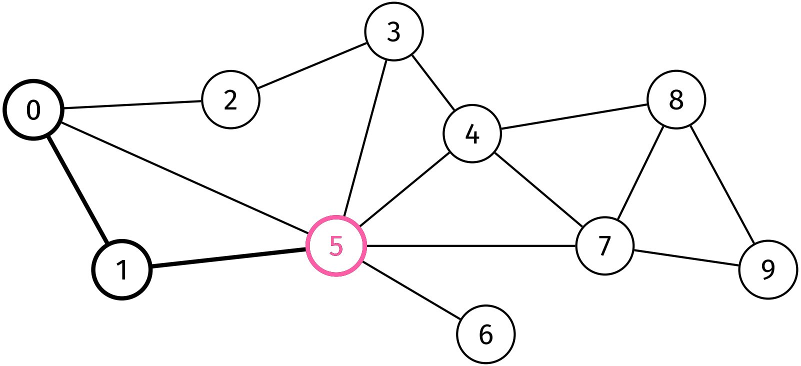
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0

visit order 0 1

dfs(1)
dfs(0)
call stack

DFS Example

Recurse into 5



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0

visit order 0 1

dfs(5)

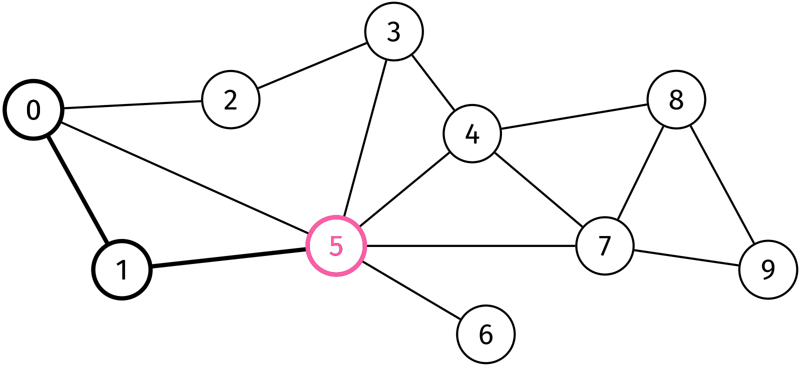
dfs(1)

dfs(0)

call stack

DFS Example

Mark 5 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	1	0	0	0	0

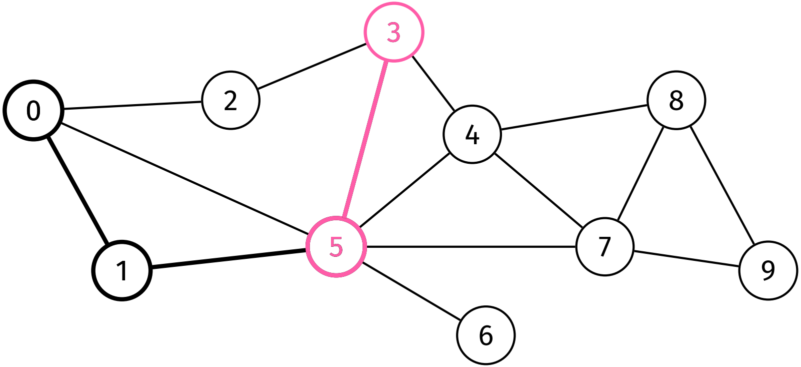
visit order 0 1 5

dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

3 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	1	0	0	0	0

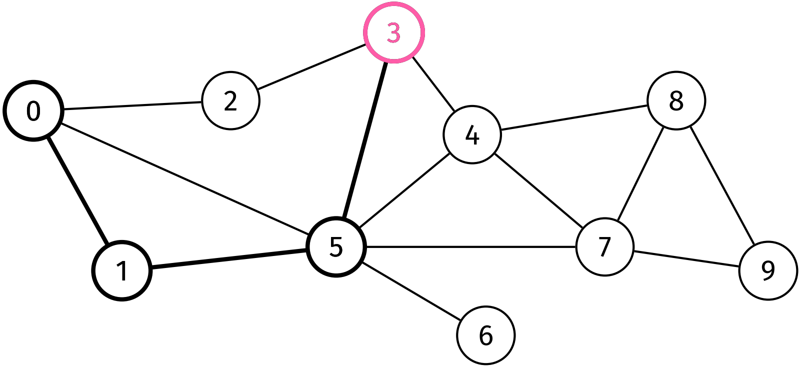
visit order 0 1 5

dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Recurse into 3



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	1	0	0	0	0

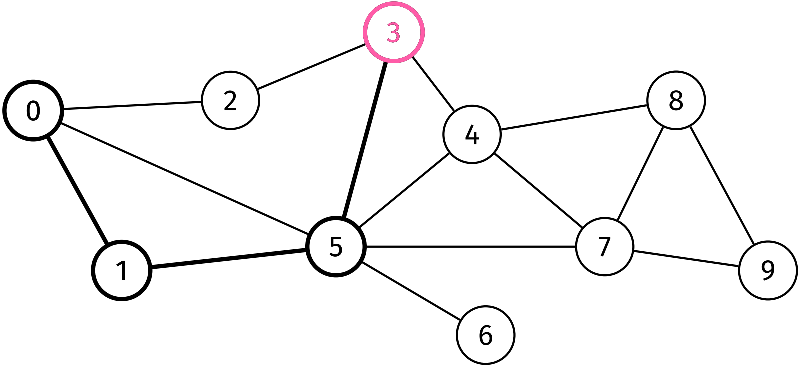
visit order 0 1 5

dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Mark 3 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	1	0	1	0	0	0	0

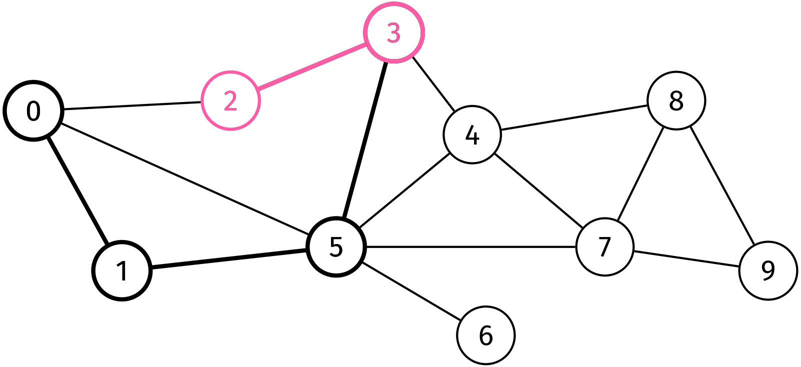
visit order 0 1 5 3

dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

2 has not been visited



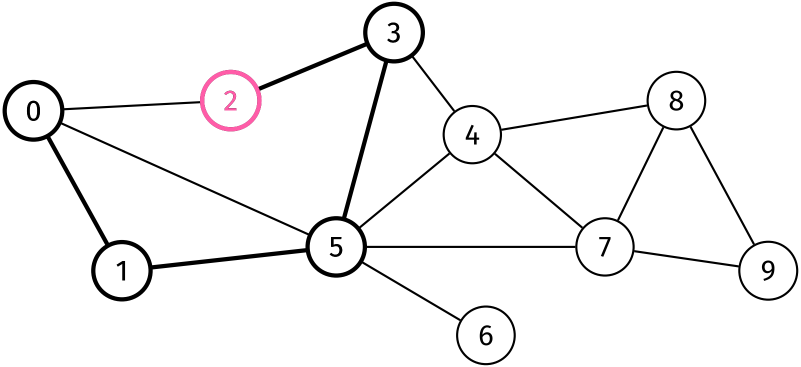
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	1	0	1	0	0	0	0

visit order 0 1 5 3

dfs(3)
dfs(5)
dfs(1)
dfs(0)
call stack

DFS Example

Recurse into 2



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	1	0	1	0	0	0	0

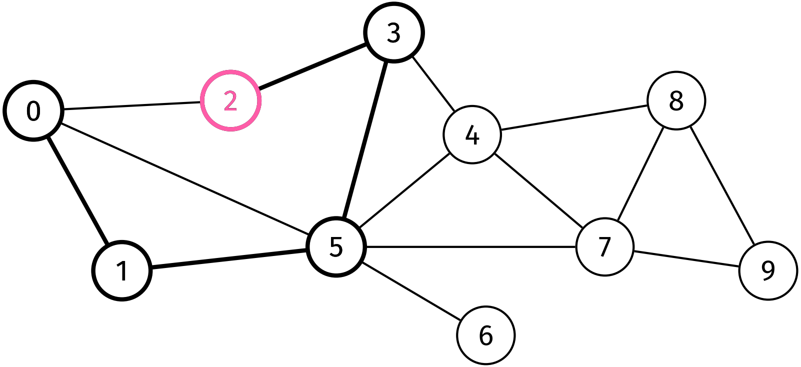
visit order 0 1 5 3

dfs(2)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Mark 2 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

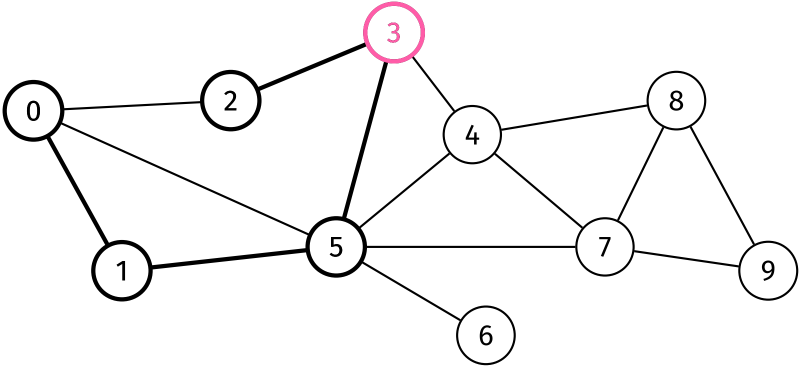
visit order 0 1 5 3 2

dfs(2)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

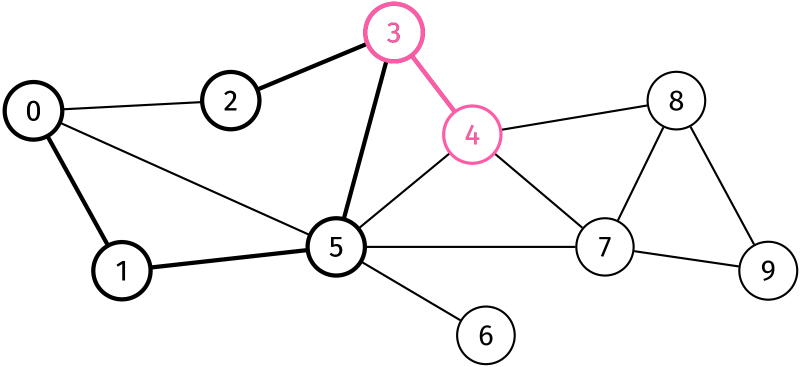
visit order 0 1 5 3 2

dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

4 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

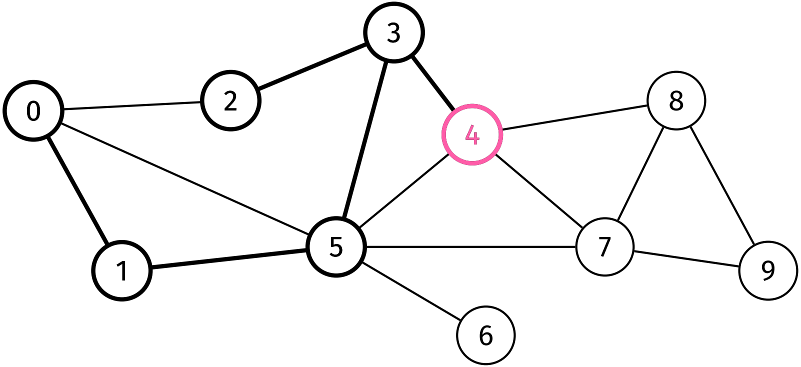
visit order 0 1 5 3 2

dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Recurse into 4



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

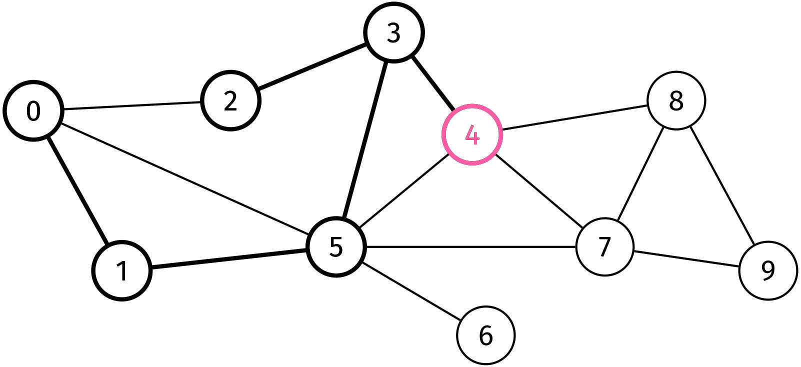
visit order 0 1 5 3 2

dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Mark 4 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0

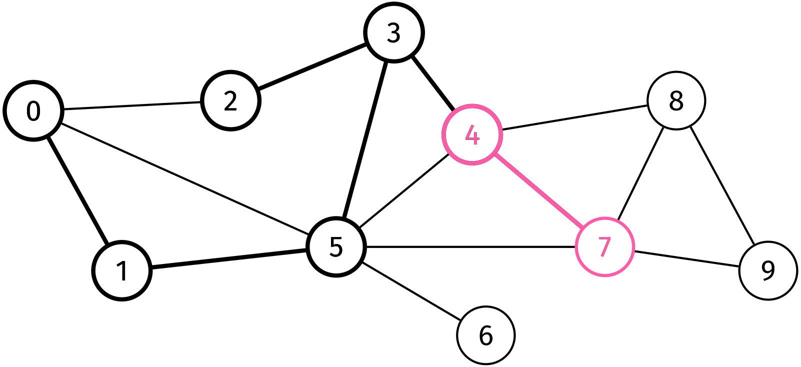
visit order 0 1 5 3 2 4

dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

7 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0

visit order 0 1 5 3 2 4

dfs(4)

dfs(3)

dfs(5)

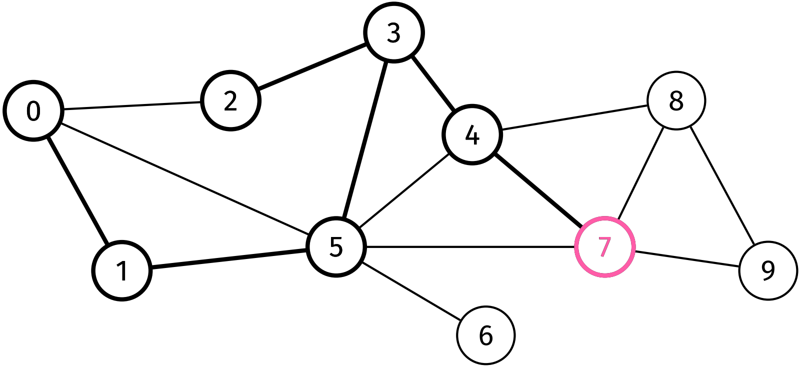
dfs(1)

dfs(0)

call stack

DFS Example

Recurse into 7



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0

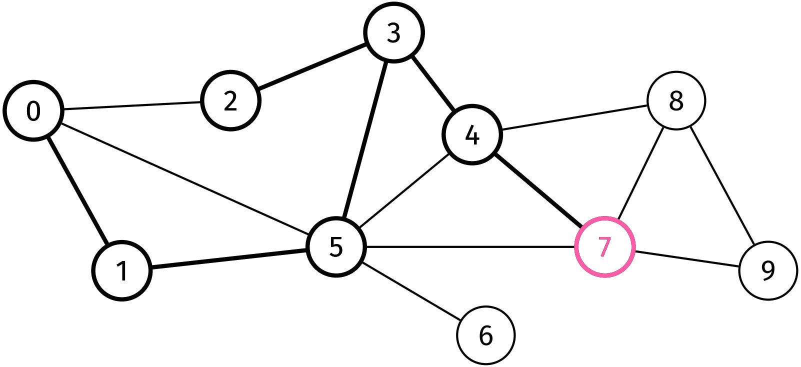
visit order 0 1 5 3 2 4

dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Mark 7 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

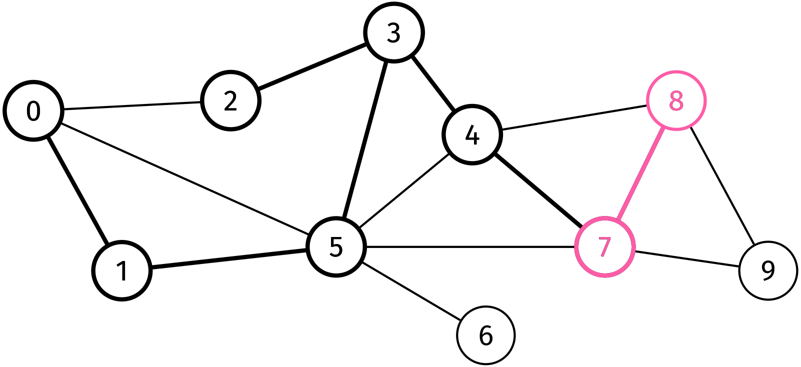
visit order 0 1 5 3 2 4 7

dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

8 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

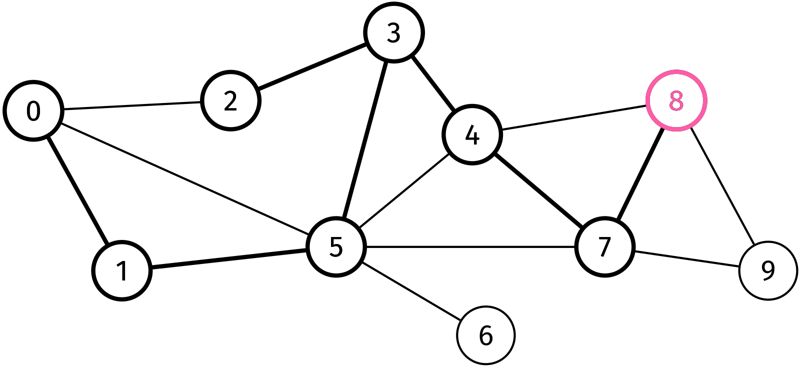
visit order 0 1 5 3 2 4 7

dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Recurse into 8



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

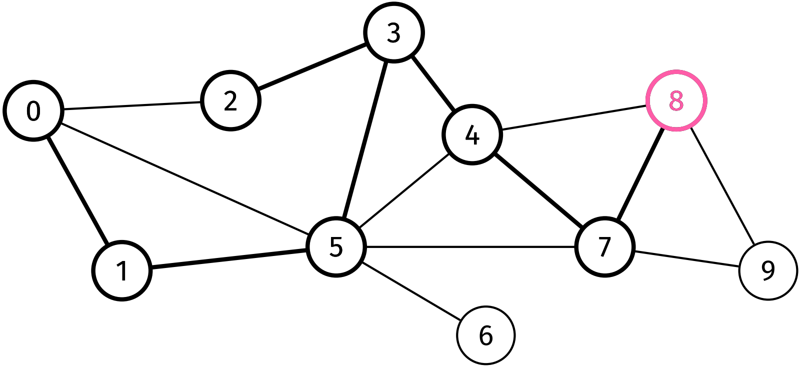
visit order 0 1 5 3 2 4 7

dfs(8)
dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Mark 8 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	0

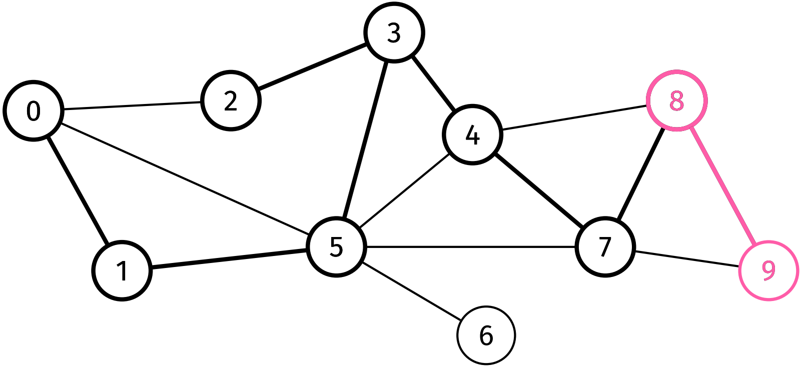
visit order 0 1 5 3 2 4 7 8

dfs(8)
dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

9 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	0

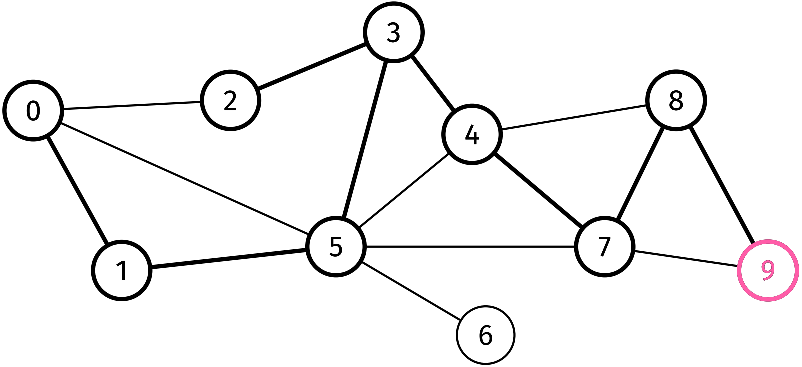
visit order 0 1 5 3 2 4 7 8

dfs(8)
dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Recurse into 9



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	0

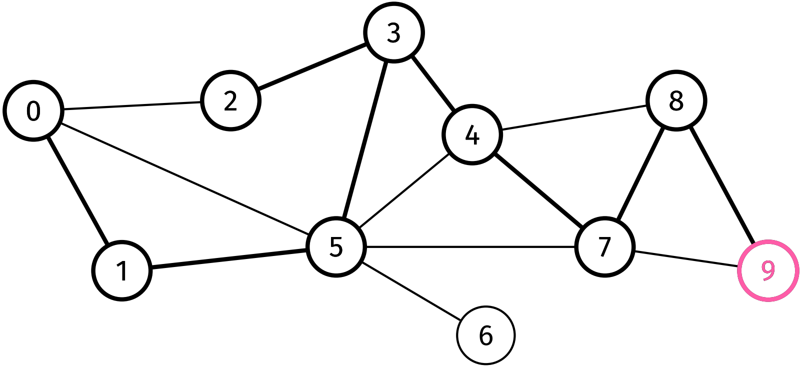
visit order 0 1 5 3 2 4 7 8

dfs(9)
dfs(8)
dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Mark 9 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

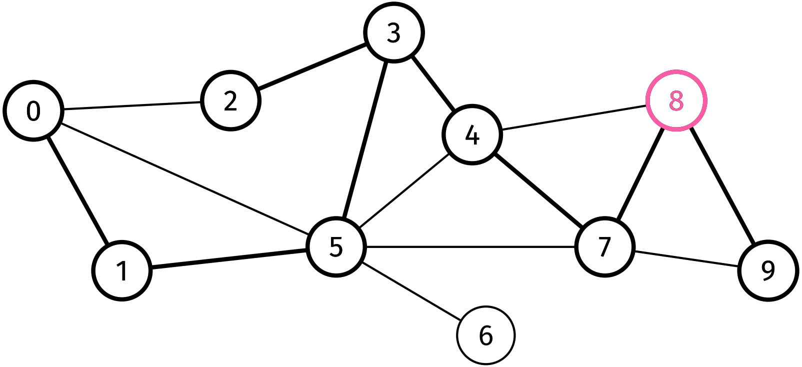
visit order 0 1 5 3 2 4 7 8 9

dfs(9)
dfs(8)
dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

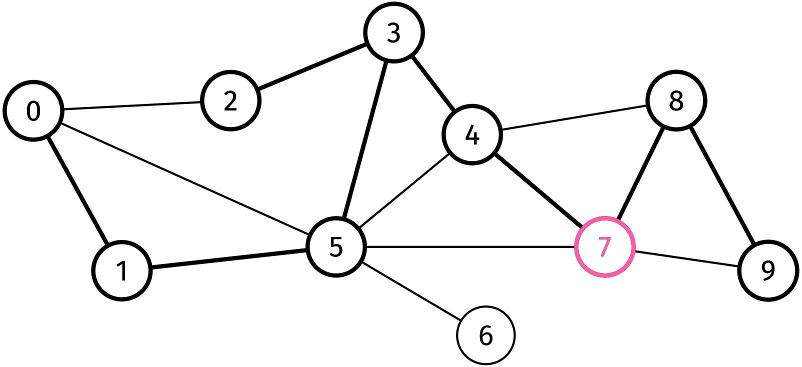
visit order 0 1 5 3 2 4 7 8 9

dfs(8)
dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

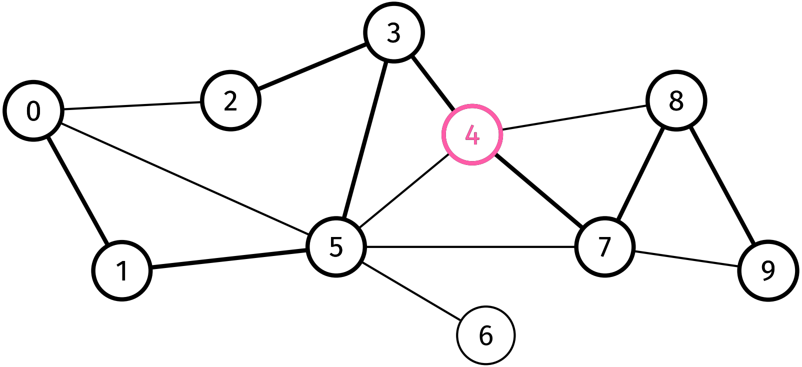
visit order 0 1 5 3 2 4 7 8 9

dfs(7)
dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

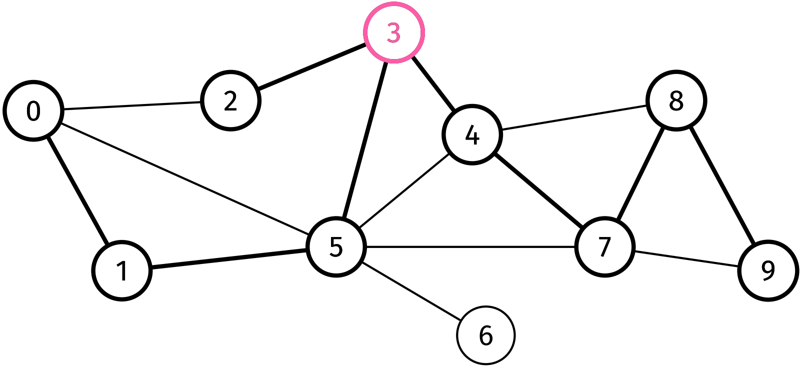
visit order 0 1 5 3 2 4 7 8 9

dfs(4)
dfs(3)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Return



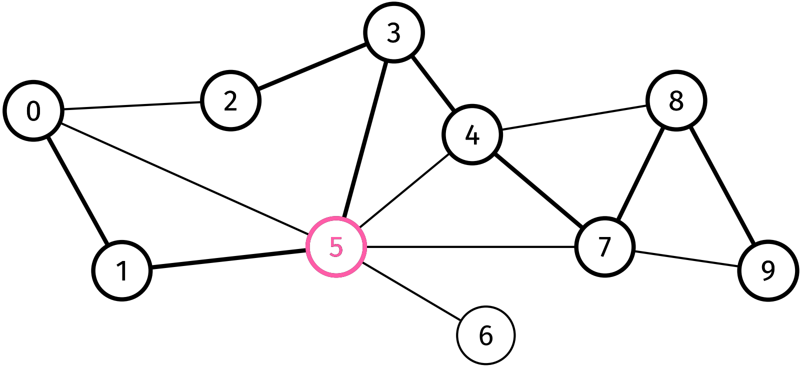
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

visit order 0 1 5 3 2 4 7 8 9

dfs(3)
dfs(5)
dfs(1)
dfs(0)
call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

visit order 0 1 5 3 2 4 7 8 9

dfs(5)

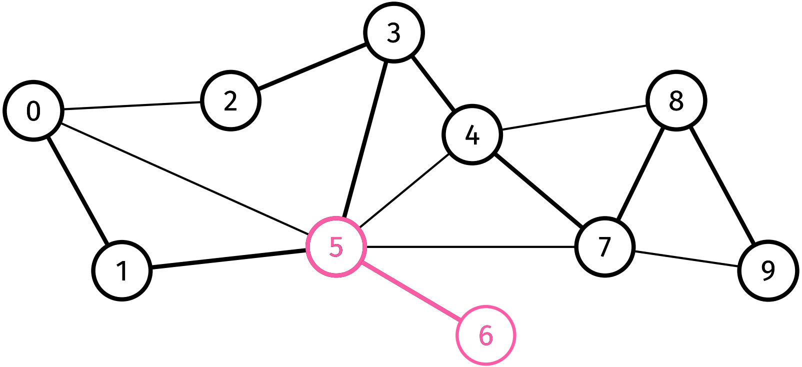
dfs(1)

dfs(0)

call stack

DFS Example

6 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

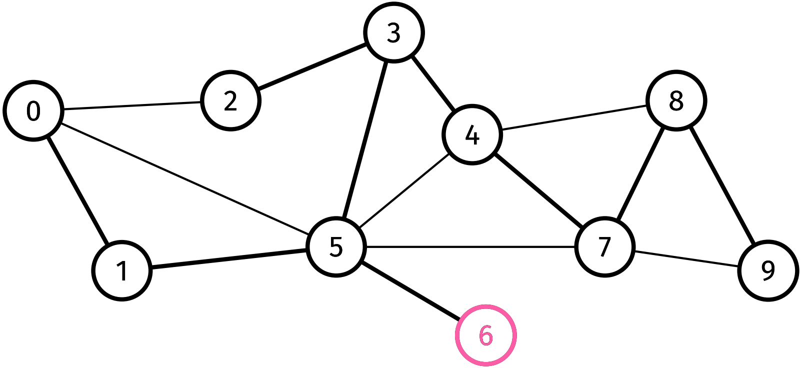
visit order 0 1 5 3 2 4 7 8 9

dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Recurse into 6



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	1	1

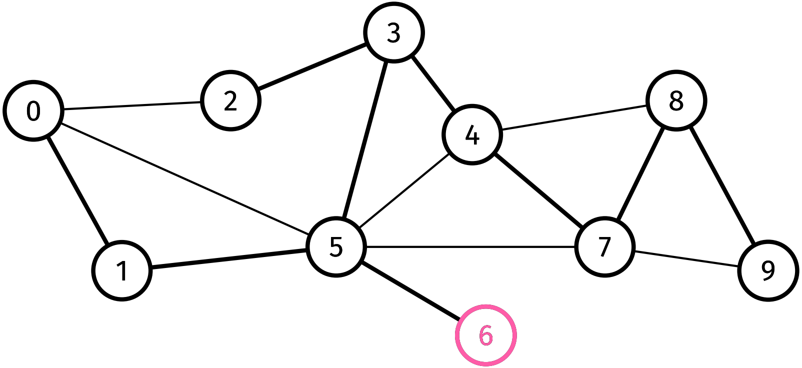
visit order 0 1 5 3 2 4 7 8 9

dfs(6)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Mark 6 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1

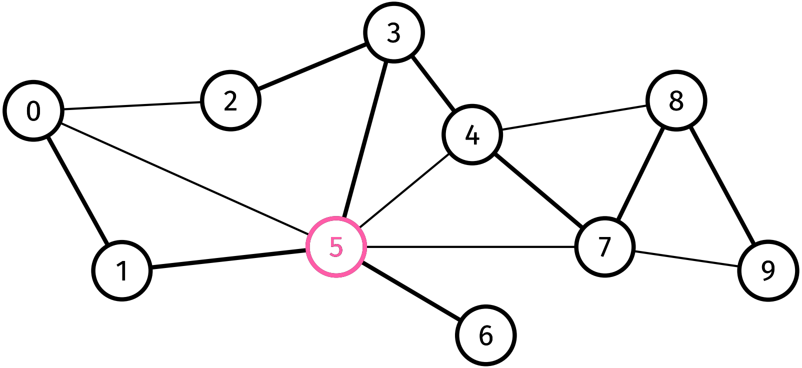
visit order 0 1 5 3 2 4 7 8 9 6

dfs(6)
dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1

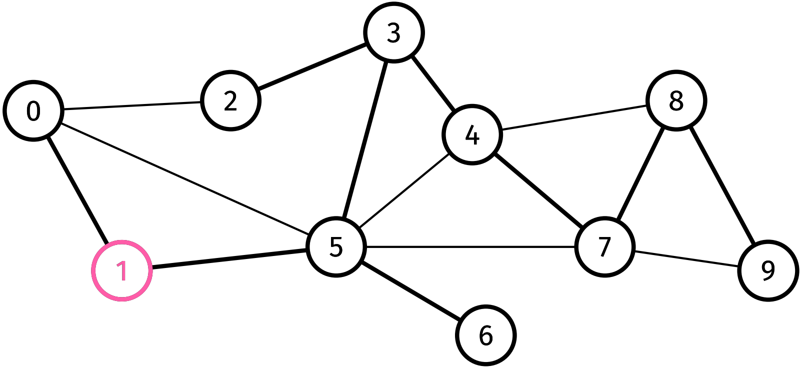
visit order 0 1 5 3 2 4 7 8 9 6

dfs(5)
dfs(1)
dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1

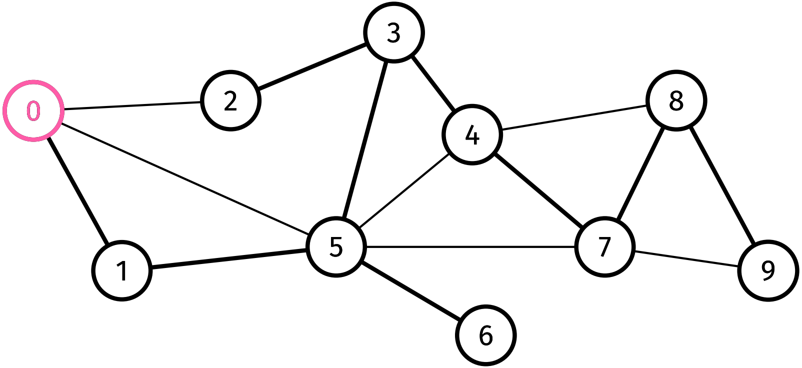
visit order 0 1 5 3 2 4 7 8 9 6

dfs(1)
dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1

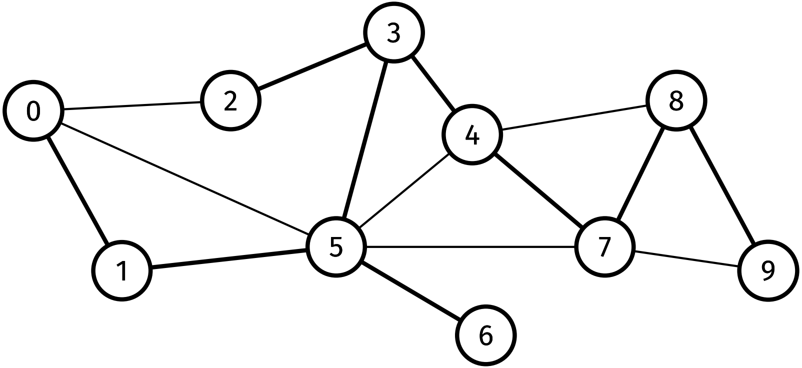
visit order 0 1 5 3 2 4 7 8 9 6

dfs(0)

call stack

DFS Example

Return



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	1	1	1	1

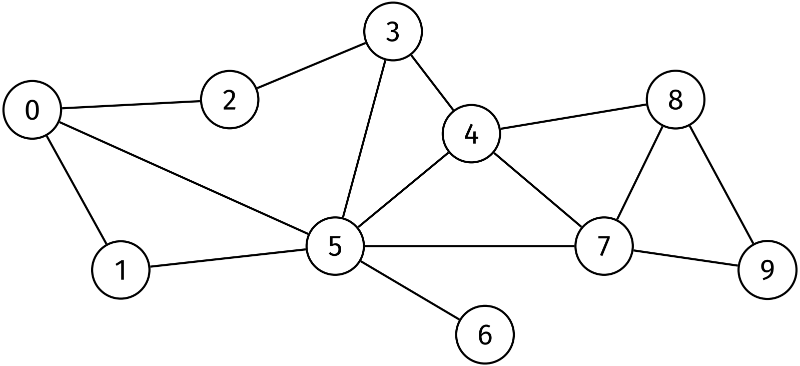
visit order 0 1 5 3 2 4 7 8 9 6

call stack

Path-Checking with Recursive DFS

Example

Is there a path between 0 and 7?

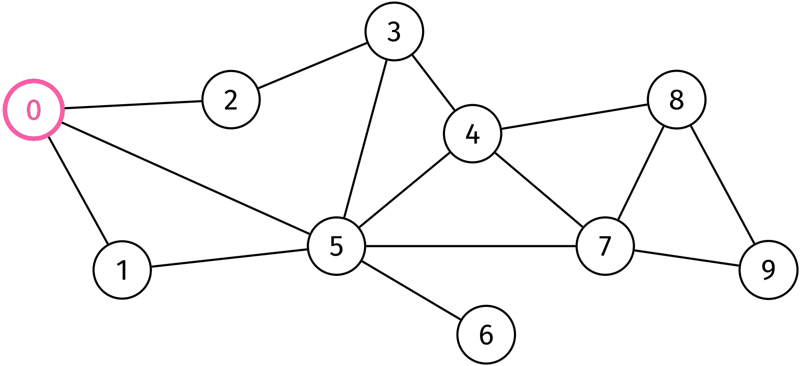


	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	0	0	0	0	0	0	0	0	0	0

call stack

Path-Checking with Recursive DFS

Example



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	0	0	0	0	0	0	0	0	0	0

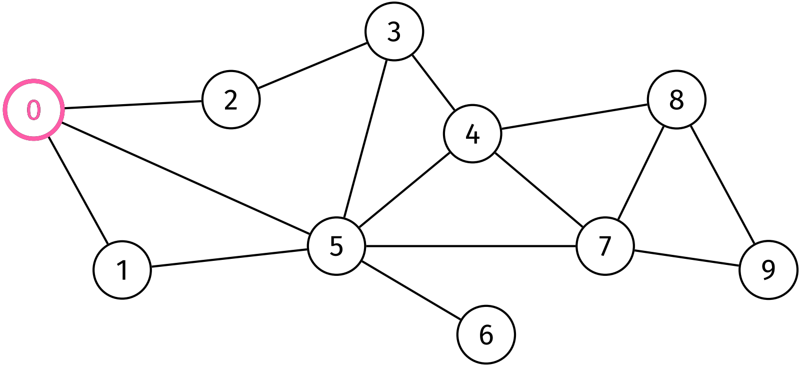
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Mark 0 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0

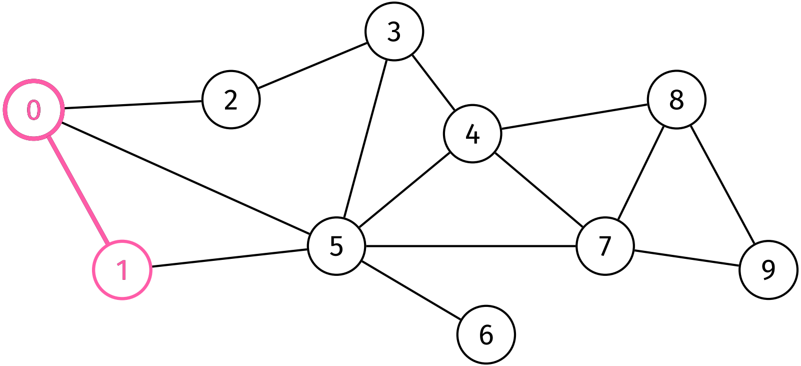
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

1 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0

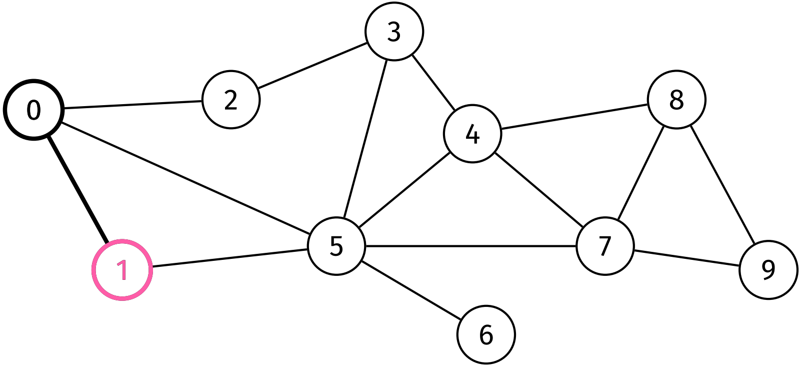
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Recurse into 1



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	0	0	0	0	0	0	0	0	0

path(1, 7)?

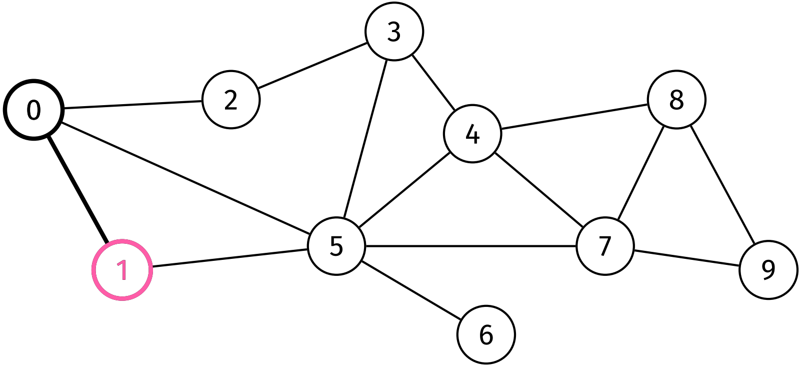
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Mark 1 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0

path(1, 7)?

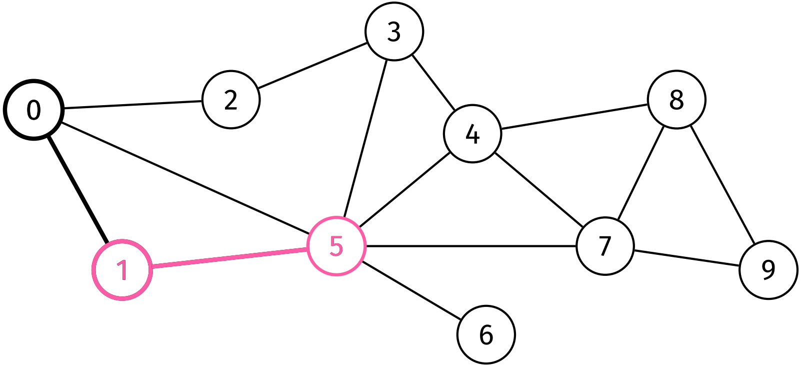
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

5 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0

path(1, 7)?

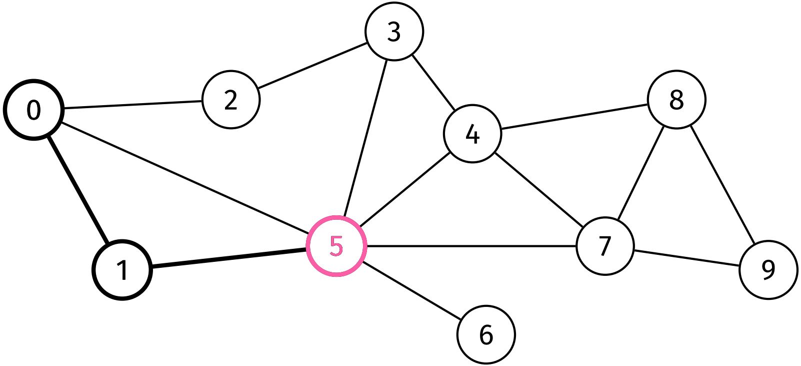
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Recurse into 5



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	0	0	0	0	0

path(5, 7)?

path(1, 7)?

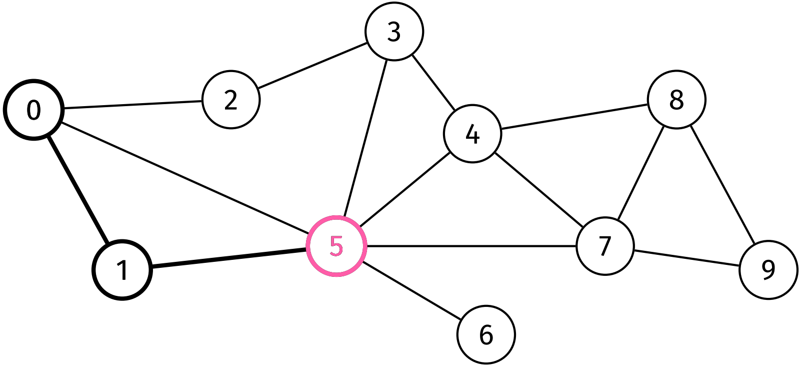
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Mark 5 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	1	0	0	0	0

path(5, 7)?

path(1, 7)?

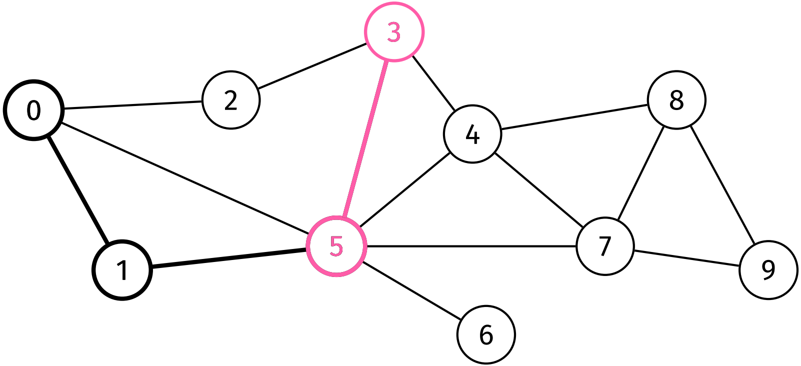
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

3 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	1	0	0	0	0

path(5, 7)?

path(1, 7)?

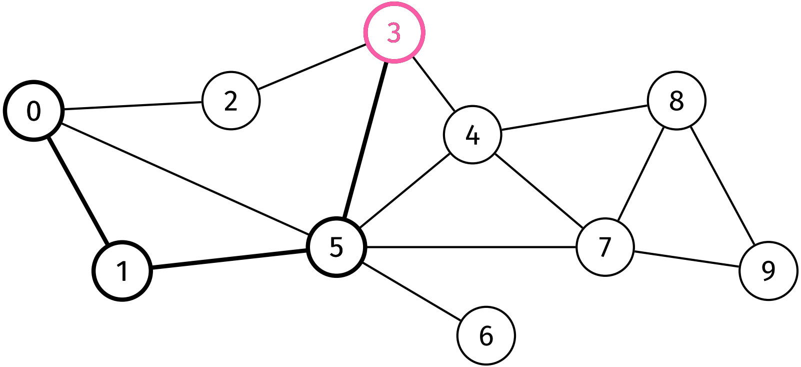
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Recurse into 3



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	0	0	1	0	0	0	0

path(3, 7)?

path(5, 7)?

path(1, 7)?

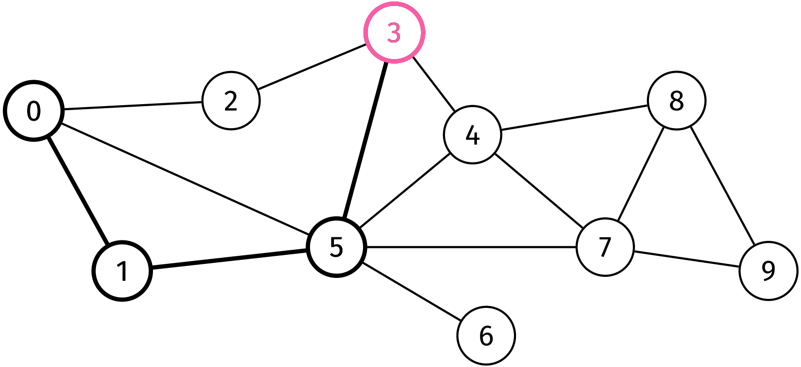
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Mark 3 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	1	0	1	0	0	0	0

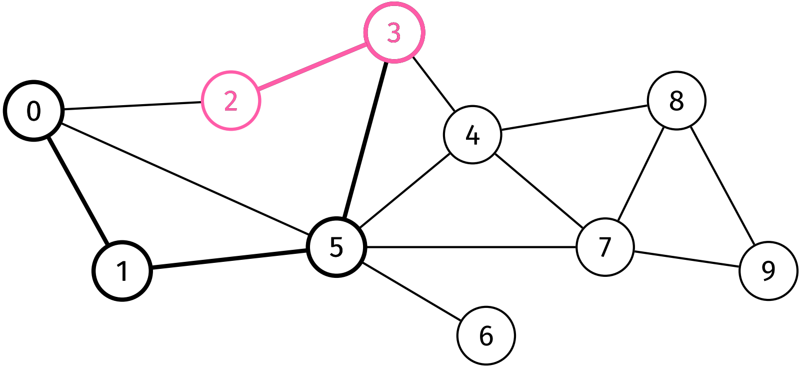
path(3, 7)?
path(5, 7)?
path(1, 7)?
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

2 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	1	0	1	0	0	0	0

path(3, 7)?

path(5, 7)?

path(1, 7)?

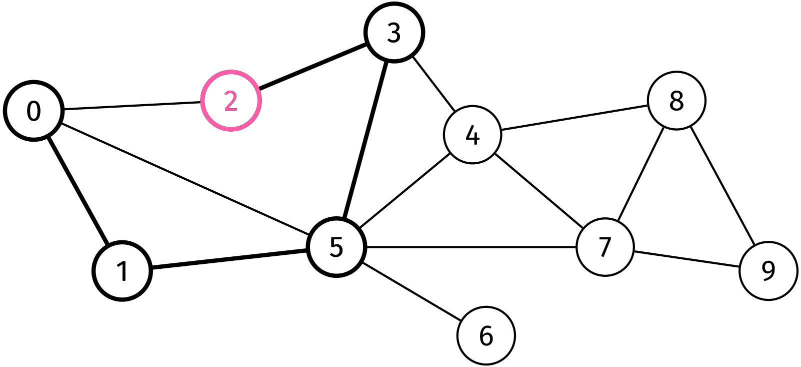
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Recurse into 2



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	0	1	0	1	0	0	0	0

path(2, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

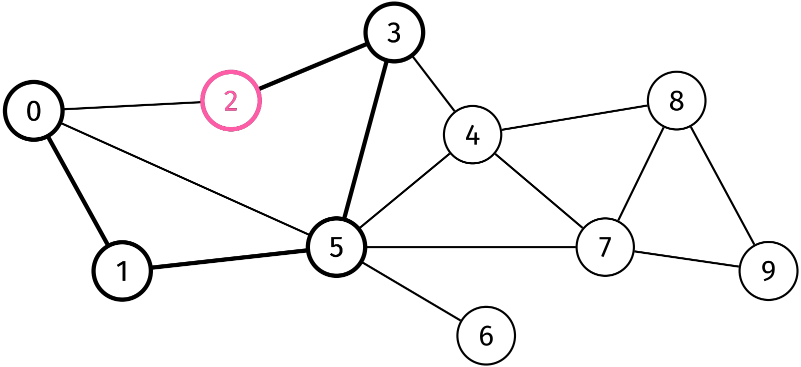
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Mark 2 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

path(2, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

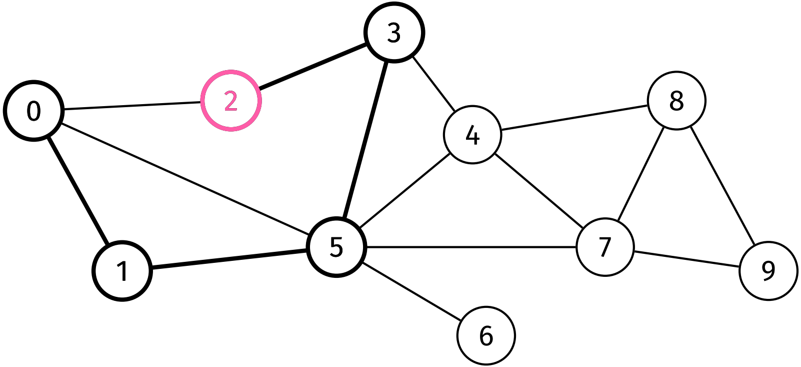
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Return false



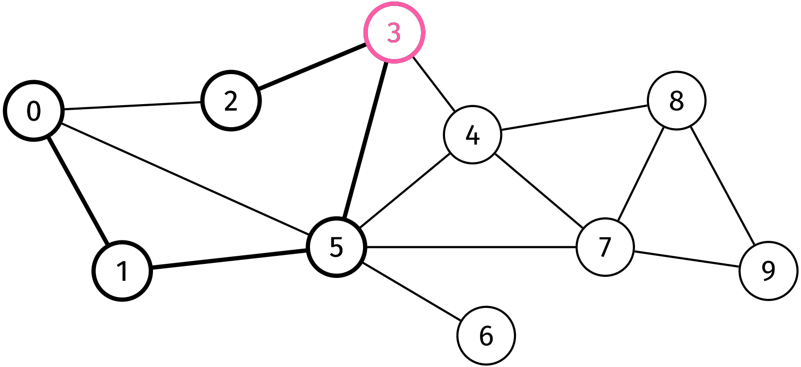
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

path(2, 7)?
path(3, 7)?
path(5, 7)?
path(1, 7)?
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

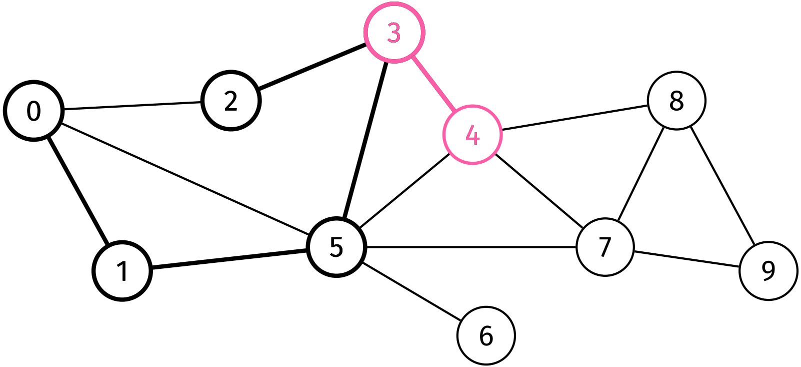
path(3, 7)?
path(5, 7)?
path(1, 7)?
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

4 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

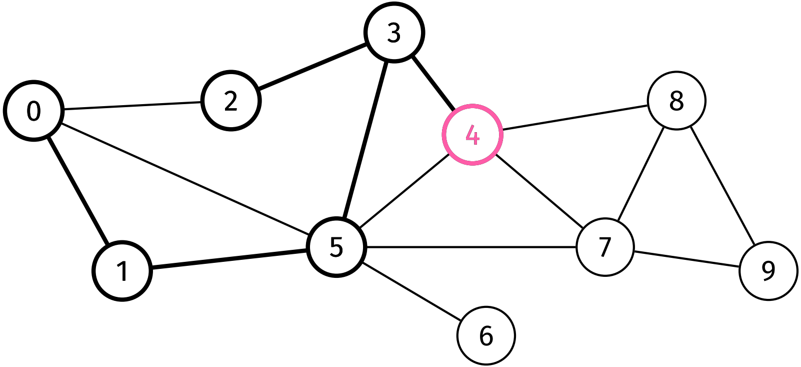
path(3, 7)?
path(5, 7)?
path(1, 7)?
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Recurse into 4



visited	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	0	1	0	0	0	0

path(4, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

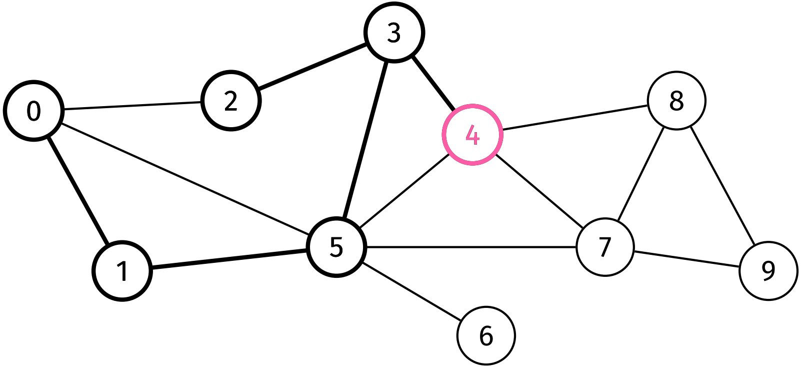
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Mark 4 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0

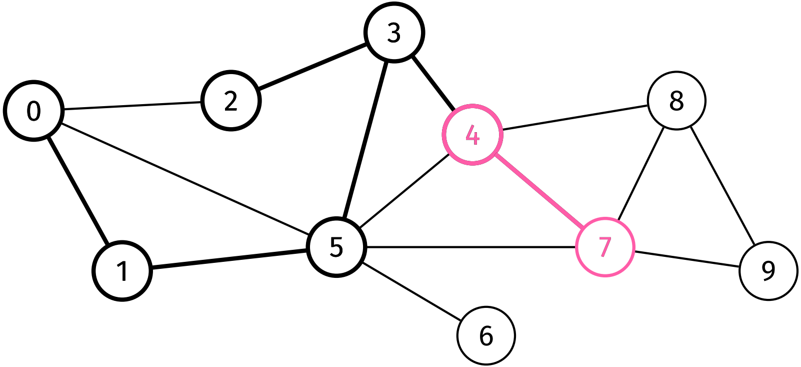
path(4, 7)?
path(3, 7)?
path(5, 7)?
path(1, 7)?
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

7 has not been visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0

path(4, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

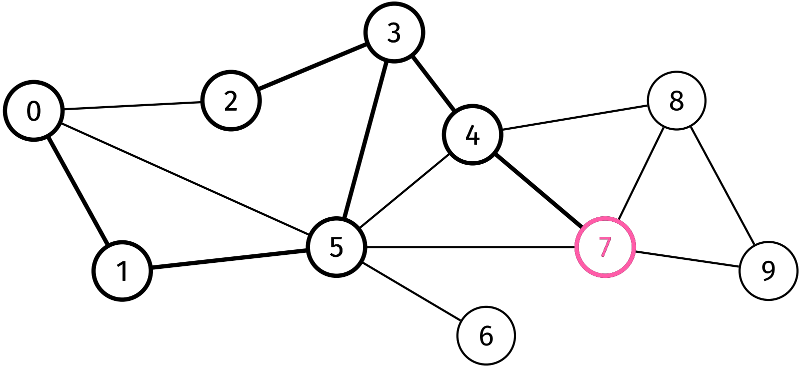
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Recurse into 7



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	0	0	0

path(7, 7)?

path(4, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

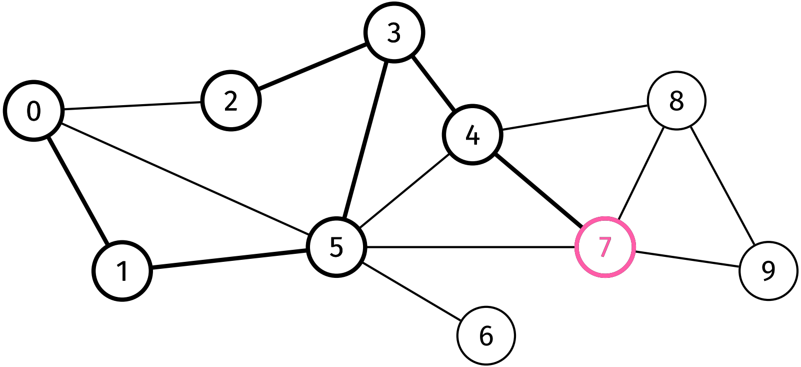
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Mark 7 as visited



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

path(7, 7)?

path(4, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

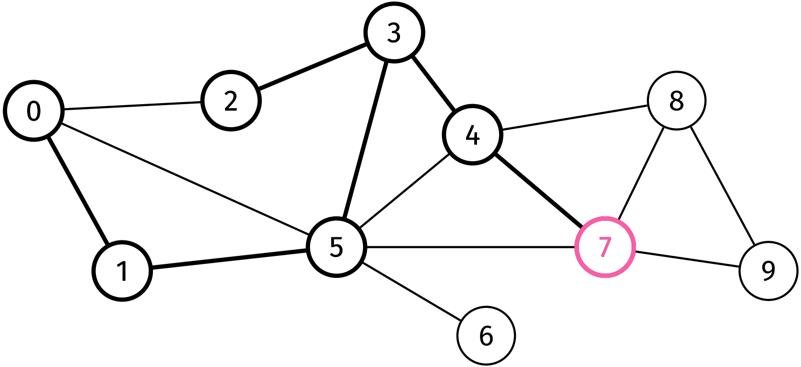
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Return true



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

path(7, 7)?

path(4, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

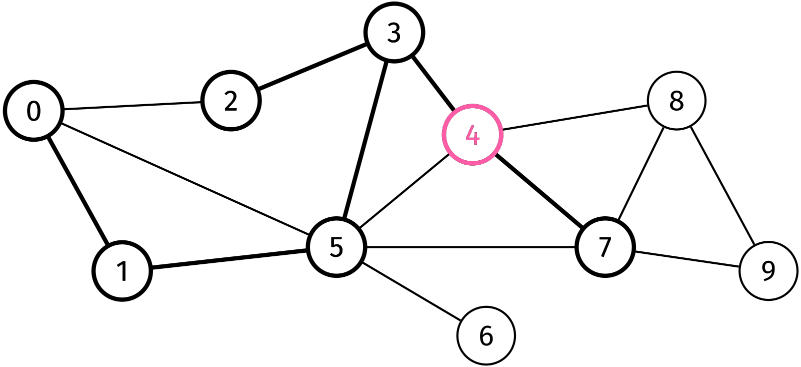
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Return true



path(4, 7)?

path(3, 7)?

path(5, 7)?

path(1, 7)?

path(0, 7)?

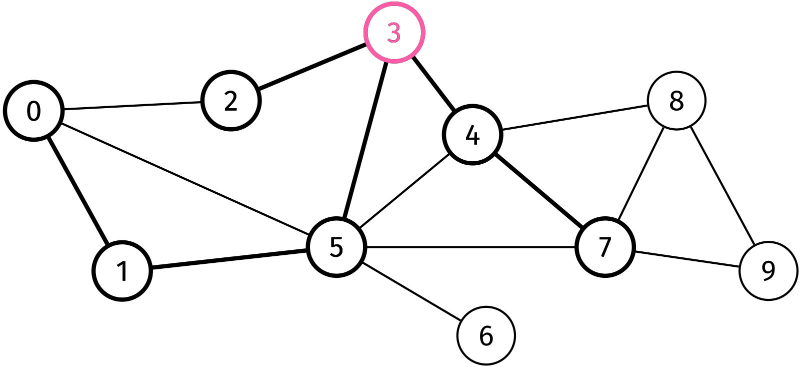
call stack

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

Path-Checking with Recursive DFS

Example

Return true



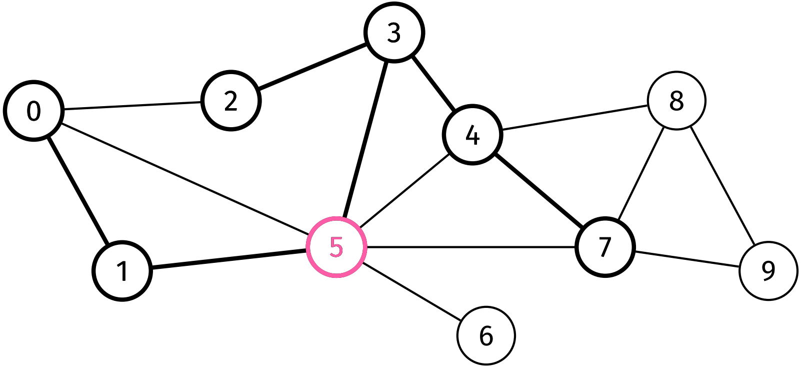
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

call stack
path(3, 7)?
path(5, 7)?
path(1, 7)?
path(0, 7)?

Path-Checking with Recursive DFS

Example

Return true



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

path(5, 7)?

path(1, 7)?

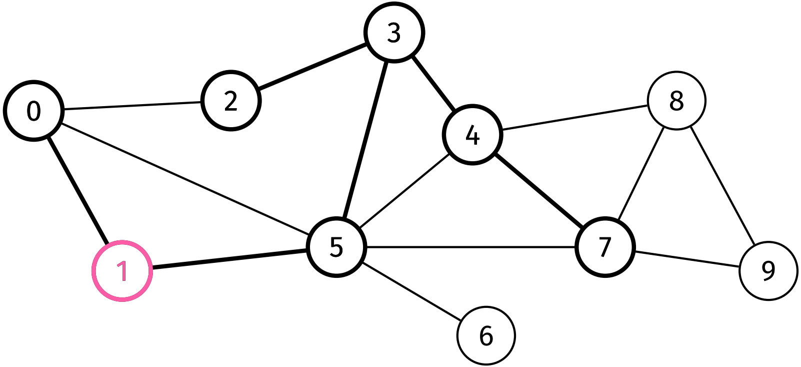
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Return true



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

path(1, 7)?

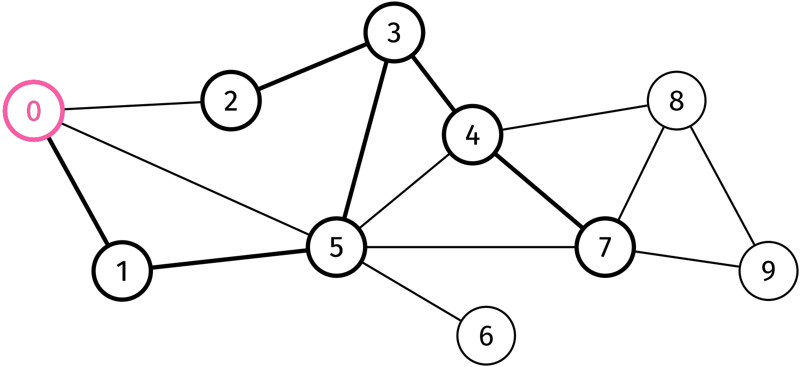
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Return true



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

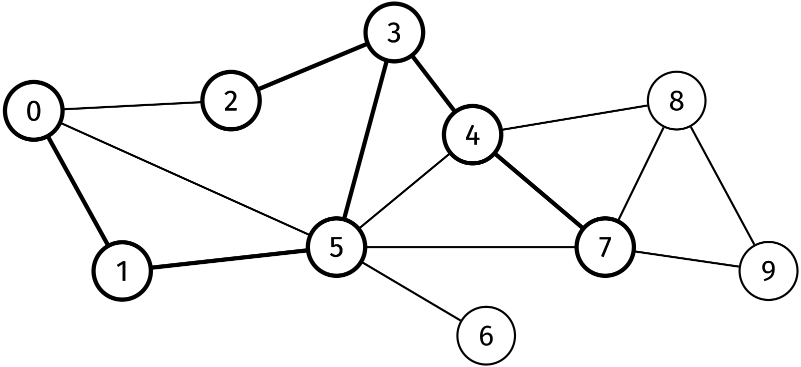
path(0, 7)?

call stack

Path-Checking with Recursive DFS

Example

Answer: Yes



	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
visited	1	1	1	1	1	1	0	1	0	0

call stack