Graphs II, Tries

Exam Prep 10

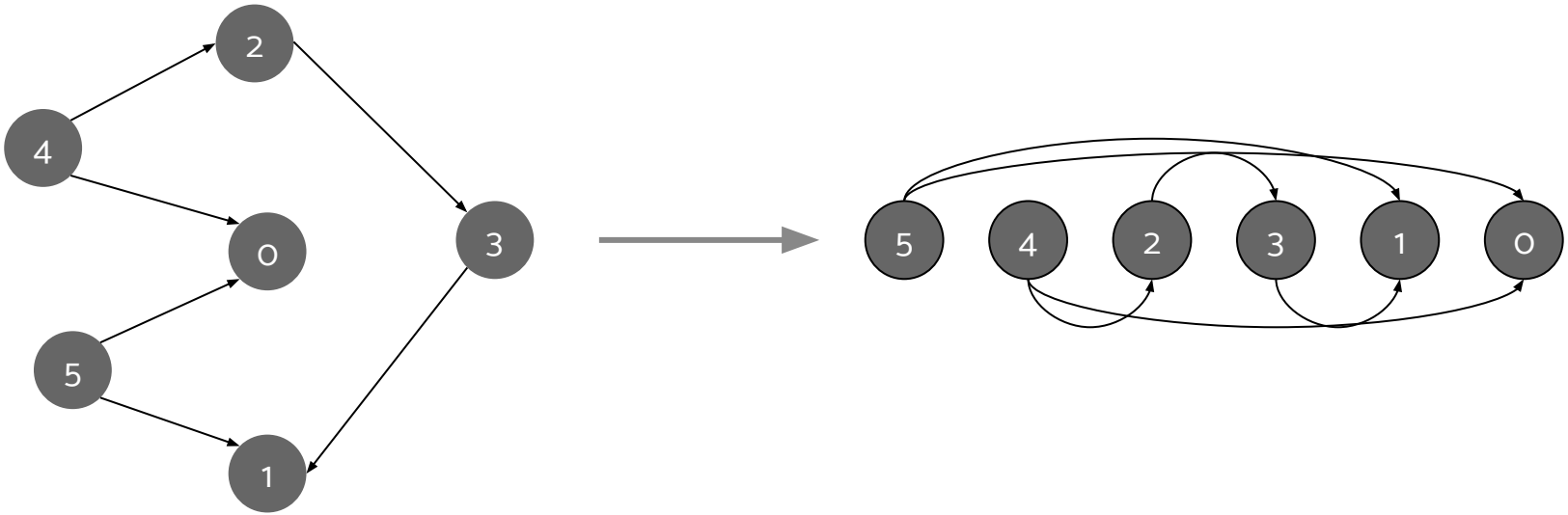
Announcements

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	4/1 Project 2B/2C due				4/5 Lab 09 due	
					4/12 Lab 10 due	

Content Review

Topological Sort

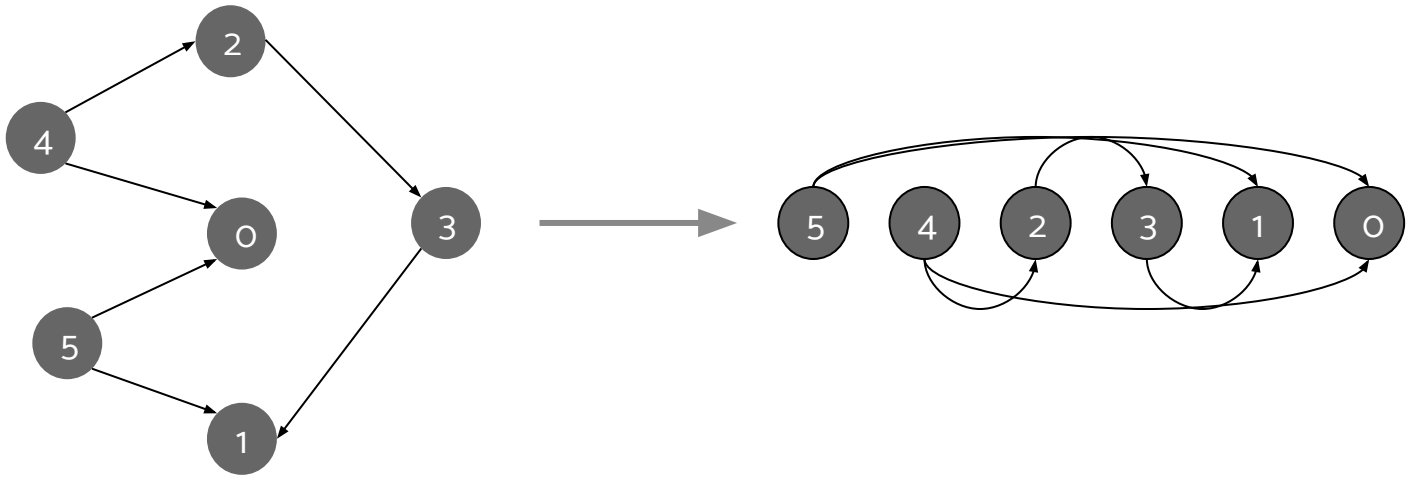
Topological Sort is a way of transforming a directed acyclic graph into a linear ordering of vertices, where for every directed edge u v, vertex u comes before v in the ordering.



Topological Sort

Key Ideas:

- Not having a topological sort indicates a that the graph has directed cycle (only works on DAGs)
- Most DAGs have multiple topological sorts
- Source node: a node that has no incoming edges
- Sink node: a node that has no outgoing edges



Graph Algorithm Runtimes

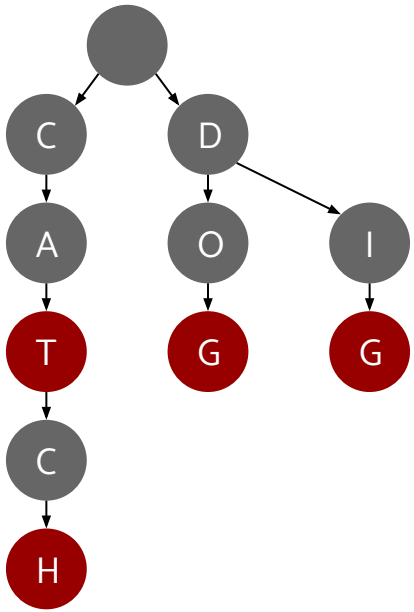
For a graph with V vertices and E edges:

Graph Algorithm	Runtime
DFS	O (V + E)
BFS	O (V + E)
Dijkstra's	O((V + E) log V)
A*	O((V + E) log V)
Prim's	O((V + E) log V)
Kruskal's	O(E log E)

Tries

Tries are special trees mostly used for language tasks.

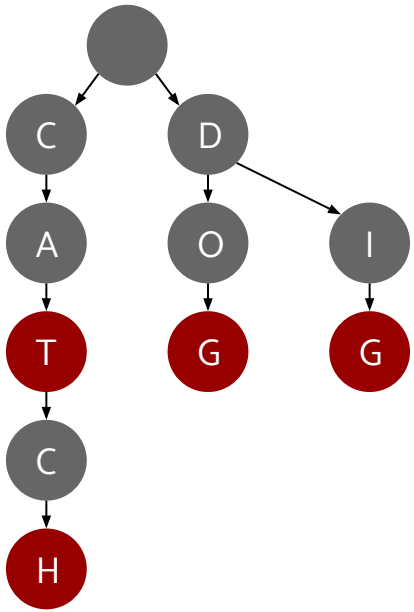
Each node in a trie is marked as being a word-end (a “key”) or not, so you can quickly check whether a word exists within your structure.



Trie Operations

Longest prefix of: follow the trie until the letters no longer match, keeping track of the most recent "end"

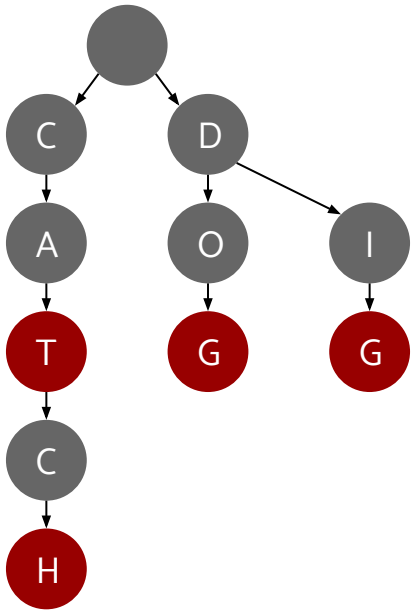
longestPrefixOf("catchall") → "catch"



Trie Operations

Keys with prefix: follow until the end of the prefix, then traverse all words below that node.

keysWithPrefix("ca") → "catch", "cat"



Worksheet

1 Multiple MSTs

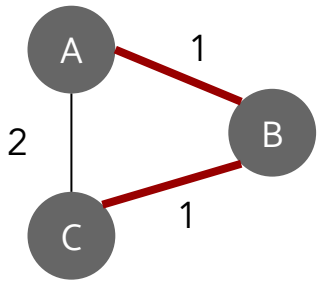
Recall a graph can have multiple MSTs if there are multiple spanning trees of minimum weight.

1. If some of the edge weights are identical, there will never/sometimes/always be multiple MSTs.
2. If all of the edge weights are identical, there will never/sometimes/always be multiple MSTs.

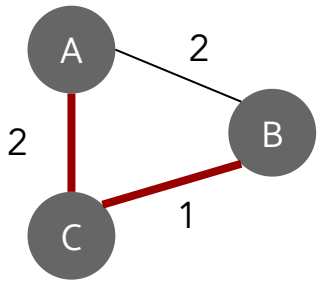
1 Multiple MSTs

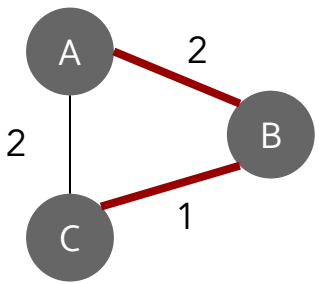
Recall a graph can have multiple MSTs if there are multiple spanning trees of minimum weight.

1. If some of the edge weights are identical, there will never/**sometimes**/always be multiple MSTs.
2. If all of the edge weights are identical, there will never/sometimes/always be multiple MSTs.



Only 1 MST



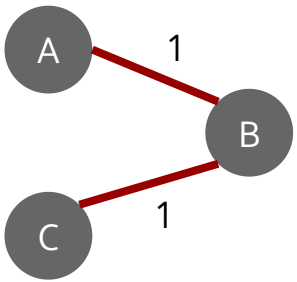


Multiple MSTs

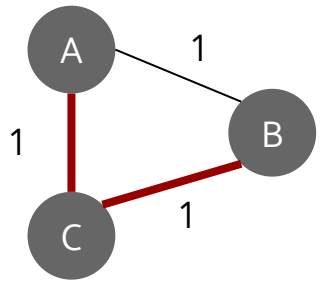
1 Multiple MSTs

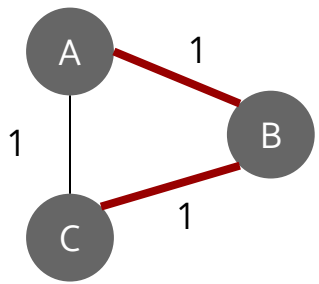
Recall a graph can have multiple MSTs if there are multiple spanning trees of minimum weight.

1. If some of the edge weights are identical, there will never/**sometimes**/always be multiple MSTs.
2. If all of the edge weights are identical, there will never/**sometimes**/always be multiple MSTs.



Only 1 MST





Multiple MSTs

1 Multiple MSTs

Suppose we have a connected, undirected graph G with N vertices and N edges, where all the edge weights are identical. Find the maximum and minimum number of MSTs in G and explain your reasoning.

1 Multiple MSTs

Suppose we have a connected, undirected graph G with N vertices and N edges, where all the edge weights are identical. Find the maximum and minimum number of MSTs in G and explain your reasoning.

Min: 3, **Max**: N

1 Multiple MSTs

Suppose we have a connected, undirected graph G with N vertices and N edges, where all the edge weights are identical. Find the maximum and minimum number of MSTs in G and explain your reasoning.

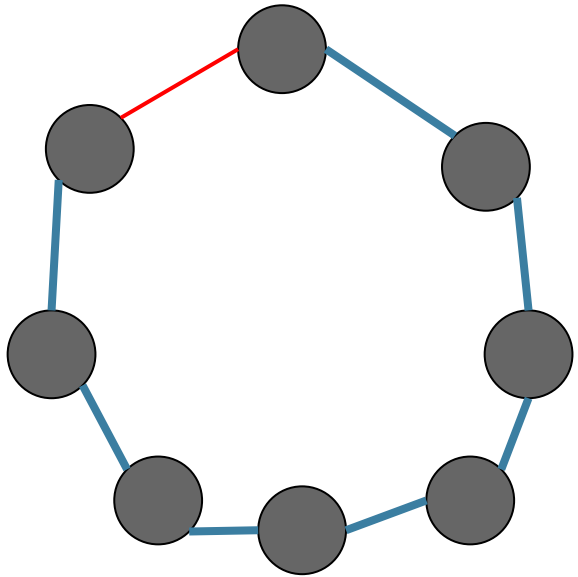
Min: 3, **Max**: N

Find a **cycle** in the graph. **Excluding one edge** in the cycle creates an MST.
The minimum cycle is 3 edges and the maximum is N edges.

1 Multiple MSTs

Suppose we have a connected, undirected graph G with N vertices and N edges, where all the edge weights are identical. Find the maximum and minimum number of MSTs in G and explain your reasoning.

Example:



1 Multiple MSTs

It is possible that Prim's and Kruskal's find different MSTs on the same graph G. Given any graph G with integer edge weights, modify G to ensure that Prim's and Kruskal's will always find the same MST. You may not modify Prim's or Kruskal's.

1 Multiple MSTs

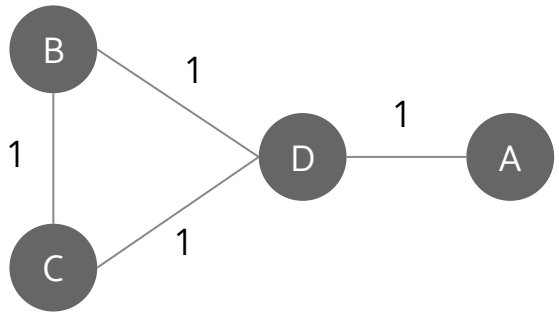
It is possible that Prim's and Kruskal's find different MSTs on the same graph G. Given any graph G with integer edge weights, modify G to ensure that Prim's and Kruskal's will always find the same MST. You may not modify Prim's or Kruskal's.

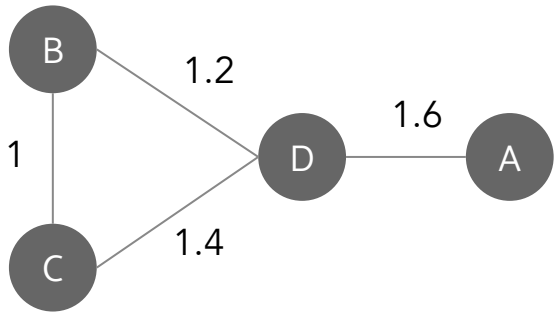
Answer: If the edge weights are **unique**, there is only one MST. To make the edge weights unique, **add a small offset** to each edge. The offset has to be **unique** and **less than 1/E** to avoid changing the actual MST.

1 Multiple MSTs

It is possible that Prim's and Kruskal's find different MSTs on the same graph G. Given any graph G with integer edge weights, modify G to ensure that Prim's and Kruskal's will always find the same MST. You may not modify Prim's or Kruskal's.

Example:





2 Topological Sorting for Cats

Describe at a high level how to perform a topological sort using an algorithm we already know (hint: it involves DFS), and provide the time complexity.

2 Topological Sorting for Cats

Describe at a high level how to perform a topological sort using an algorithm we already know (hint: it involves DFS), and provide the time complexity.

Answer: **Reverse** the edges, then perform a **postorder traversal** until all vertices are visited.
Alternative: Perform a **DFS traversal** from every vertex with indegree 0. Record DFS postorder in a list. Topological ordering is given by the **reverse** of that list (reverse postorder).

Runtime: O(V + E) for DFS

2 Topological Sorting for Cats

First, provide a logical reasoning for the following claim (or a proof!): Every DAG has at least one source node (no incoming edges), and at least one sink node (no outgoing edges).

2 Topological Sorting for Cats

First, provide a logical reasoning for the following claim (or a proof!): Every DAG has at least one source node (no incoming edges), and at least one sink node (no outgoing edges).

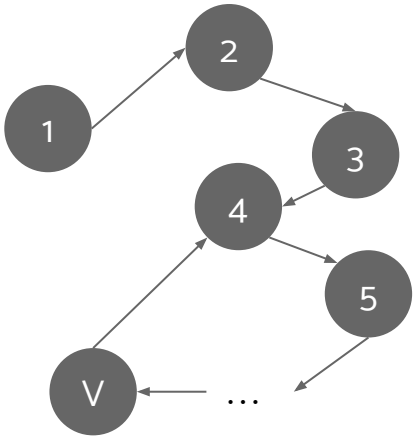
Answer: Suppose a DAG exists with no sinks. Then every node has at least one outgoing edge. If we traverse all V vertices of this graph, we can always take an outgoing edge to some other node. This means at the last Vth (last) vertex we visit, we must take an outgoing edge to some already-visited node. But this means that there is a cycle, which contradicts that our graph is a DAG.

Suppose a DAG exists with no sources. Reversing the edges makes every source a sink, and apply the proof from above.

2 Topological Sorting for Cats

First, provide a logical reasoning for the following claim (or a proof!): Every DAG has at least one source node (no incoming edges), and at least one sink node (no outgoing edges).

Example:



2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Computes the number of incoming edges to a vertex
    public int[] computeInDegrees() {
        int[] indegree = ______________________;
        for (_____________________________) {
            for (__________________________) {
                ____________________________
            }
        }
        return indegree;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Computes the number of incoming edges to a vertex
    public int[] computeInDegrees() {
        int[] indegree = new int[V()];
        for (_____________________________) {
            for (_________________________) {
                ____________________________
            }
        }
        return indegree;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Computes the number of incoming edges to a vertex
    public int[] computeInDegrees() {
        int[] indegree = new int[V()];
        for (int i = 0; i < V(); i++) {
            for (__________________________) {
                ____________________________
            }
        }
        return indegree;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Computes the number of incoming edges to a vertex
    public int[] computeInDegrees() {
        int[] indegree = new int[V()];
        for (int i = 0; i < V(); i++) {
            for (int w : adj(i)) {
                ----------------------------
            }
        }
        return indegree;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Computes the number of incoming edges to a vertex
    public int[] computeInDegrees() {
        int[] indegree = new int[V()];
        for (int i = 0; i < V(); i++) {
            for (int w : adj(i)) {
                indegree[w]++;
            }
        }
        return indegree;
    }
}
```

The runtime is O(V + E), because we iterate through all the vertices and edges in the graph in the first method.

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Finds all source nodes in the graph
    public List<Integer> findAllSourceNodes(int[] indegree) {
        List<Integer> sources = new ArrayList<>();
        for (_____________________________) {
            if (_________________________) {
                ________________________
            }
        }
        return ___________________;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Finds all source nodes in the graph
    public List<Integer> findAllSourceNodes(int[] indegree) {
        List<Integer> sources = new ArrayList<>();
        for (int i = 0; i < V(); i++) {
            if (__________________________) {
                __________________________
            }
        }
        return ____________________;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Finds all source nodes in the graph
    public List<Integer> findAllSourceNodes(int[] indegree) {
        List<Integer> sources = new ArrayList<>();
        for (int i = 0; i < V(); i++) {
            if (indegree[i] == 0) {
                ________________________
            }
        }
        return ___________________;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

```
public class DAG extends Graph {
    // Finds all source nodes in the graph
    public List<Integer> findAllSourceNodes(int[] indegree) {
        List<Integer> sources = new ArrayList<>();
        for (int i = 0; i < V(); i++) {
            if (indegree[i] == 0) {
                sources.add(i);
            }
        }
        return ____________________;
    }
}
```


2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

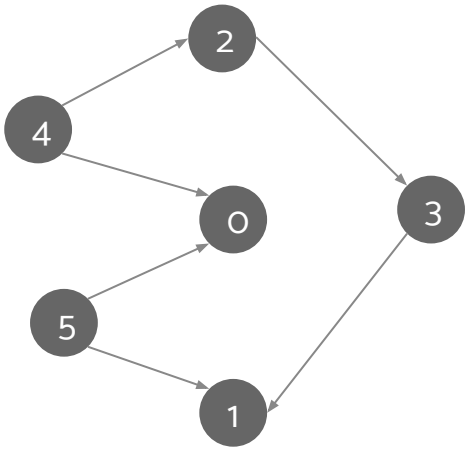
```
public class DAG extends Graph {
    // Finds all source nodes in the graph
    public List<Integer> findAllSourceNodes(int[] indegree) {
        List<Integer> sources = new ArrayList<>();
        for (int i = 0; i < V(); i++) {
            if (indegree[i] == 0) {
                sources.add(i);
            }
        }
        return sources;
    }
}
```

The runtime is O(V). This is because we only iterate through the vertices.

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



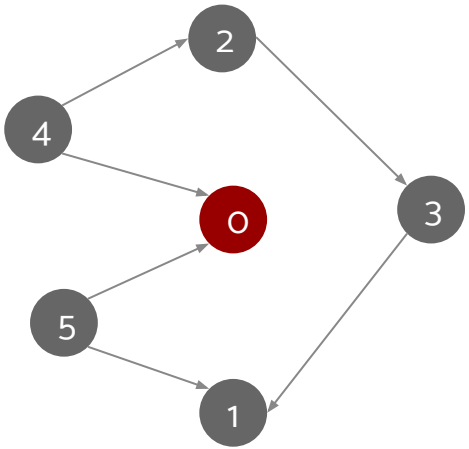
Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Incoming Edges
[0, 0, 0, 0, 0, 0]

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



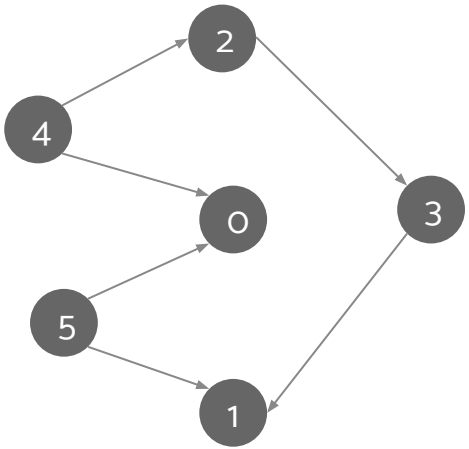
Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Incoming Edges
[0, 0, 0, 0, 0, 0]

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



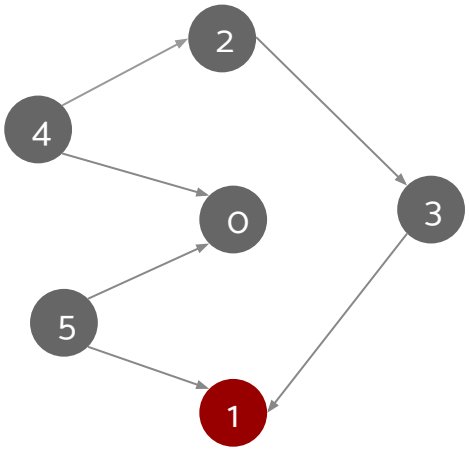
Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Incoming Edges
[0, 0, 0, 0, 0, 0]

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



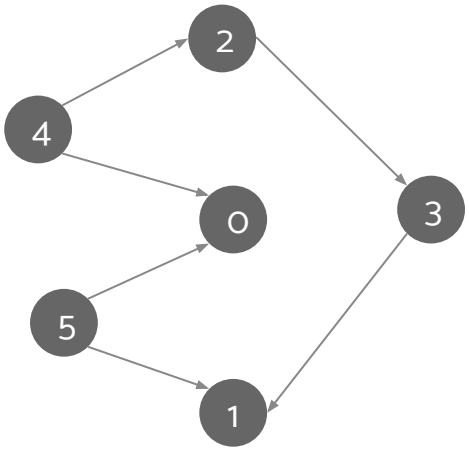
Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Incoming Edges
[0, 0, 0, 0, 0, 0]

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



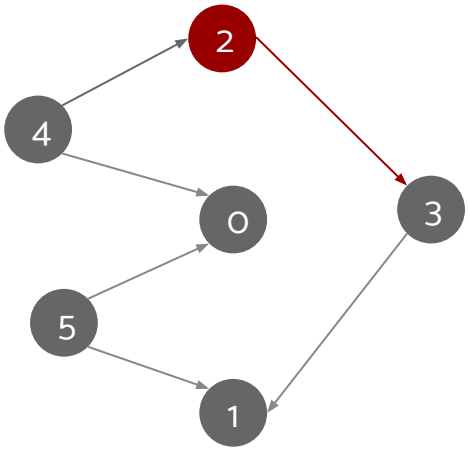
Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Incoming Edges
[0, 0, 0, 0, 0, 0]

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

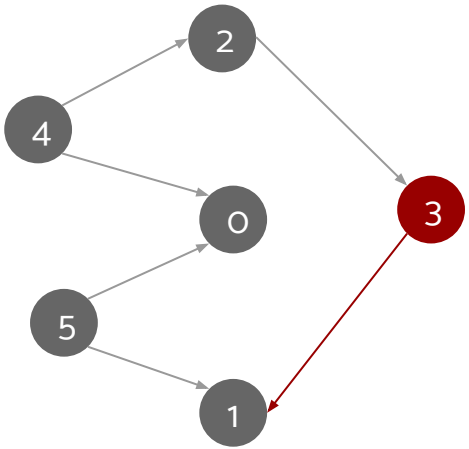
Incoming Edges
[0, 0, 0, 1, 0, 0]

counts[3] += 1

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

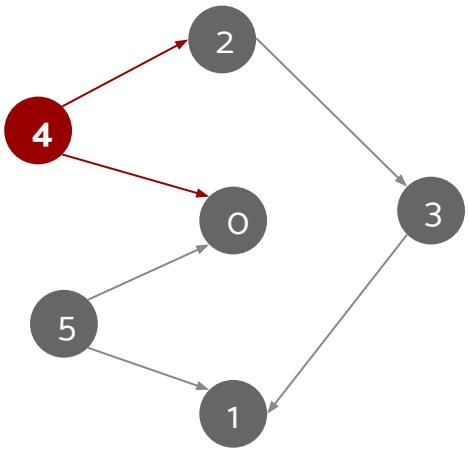
Incoming Edges
[0, 1, 0, 1, 0, 0]

counts[1] += 1

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

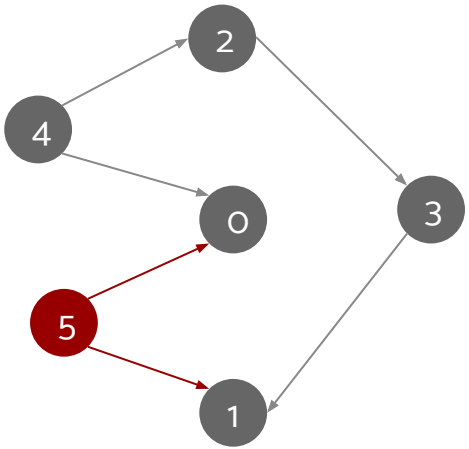
Incoming Edges
[1, 1, 1, 1, 0, 0]

counts[0] += 1
counts[2] += 1

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

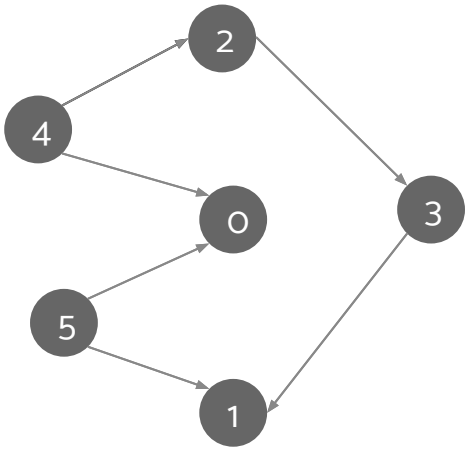
Incoming Edges
[2, 2, 1, 1, 0, 0]

counts[0] += 1
counts[1] += 1

2 Topological Sorting for Cats

Complete the following instance methods `computeInDegrees` and `findAllSourceNodes`.

Example:



Sources: 4, 5

Incoming Edges
[2, 2, 1, 1, 0, 0]

2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    ___________________________________________
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(_____________________);
    while (_______________________________) {
        int source = sources.poll();
        ____________________________________
        for (_______________________________________) {
            ____________________________________
            if (_____________________) {
                ______________________
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(_________________________);
    while (_________________________________) {
        int source = sources.poll();

        ____________________________________
        for (_______________________________________) {
            ____________________________________
            if (_________________________) {
                ________________________
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));
    while (_________________________________) {
        int source = sources.poll();

        ___________________________________
        for (_____________________________________) {
            ___________________________________
            if (_________________________) {
                ________________________
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));
    while (!sources.isEmpty()) {
        int source = sources.poll();

        ____________________________________
        for (_______________________________________) {
            ____________________________________
            if (________________________) {
                ________________________
            }
        }
    }
    return sorted;
}
```

2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));
    while (!sources.isEmpty()) {
        int source = sources.poll();
        sorted.add(source);
        for (_____________________________________) {
            ___________________________________
            if (_________________________) {
                ________________________
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));
    while (!sources.isEmpty()) {
        int source = sources.poll();
        sorted.add(source);
        for (int w : adj(source)) {
            ____________________________________
            if (_________________________) {
                ________________________
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));
    while (!sources.isEmpty()) {
        int source = sources.poll();
        sorted.add(source);
        for (int w : adj(source)) {
            indegree[w]--;
            if (_________________________) {
                ________________________
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));
    while (!sources.isEmpty()) {
        int source = sources.poll();
        sorted.add(source);
        for (int w : adj(source)) {
            indegree[w]--;
            if (indegree[w] == 0) {
                ----------------------
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node. Inspired by this fact, and using the previous parts, complete the `topologicalSort` method. What is its runtime?

```
public List<Integer> topologicalSort() {
    List<Integer> sorted = new ArrayList<>();
    int[] indegree = computeInDegrees();
    // Hint: add elements from another iterable here
    Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));
    while (!sources.isEmpty()) {
        int source = sources.poll();
        sorted.add(source);
        for (int w : adj(source)) {
            indegree[w]--;
            if (indegree[w] == 0) {
                sources.add(w);
            }
        }
    }
    return sorted;
}
```


2 Topological Sorting for Cats

Now, make the following observation: If we remove all of the source nodes from a DAG, we are guaranteed to have at least one new source node, since the new graph is still a DAG. Inspired by this fact, and using the previous parts, come up with an algorithm to topological sort. Is it more efficient?

Algorithm:

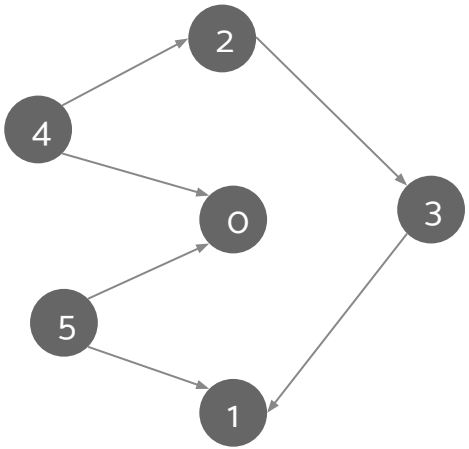
1. Create the indegree array from part 2.
2. Get all of the source nodes and add them to a queue.
3. While the queue is not empty:
    a. Remove a vertex from the queue.
    b. For all outgoing edges (u, v) of the vertex:
        i. indegree[v] -= 1
        ii. If indegree[v] == 0, add it to the queue.

Runtime: O(V + E)

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

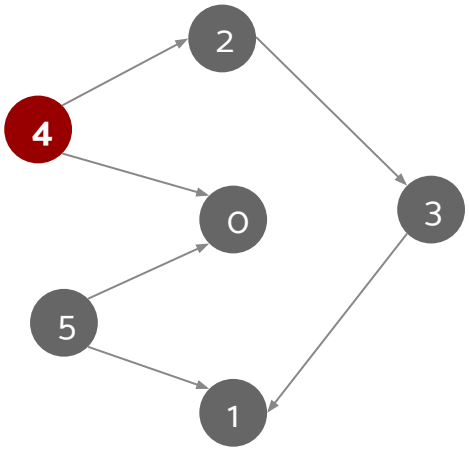
Indegrees
[2, 2, 1, 1, 0, 0]

Sources
{4, 5}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[1, 2, 0, 1, 0, 0]

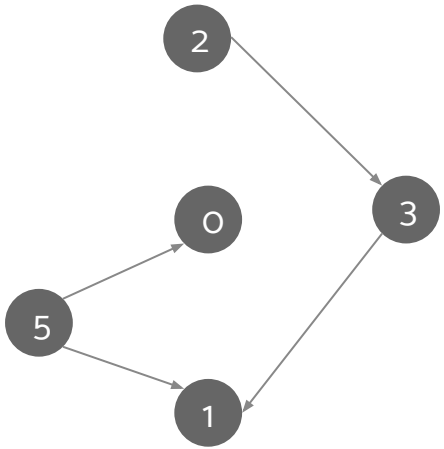
indegree[0] -= 1
indegree[2] -= 1

Sources
{4, 5}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[1, 2, 0, 1, 0, 0]

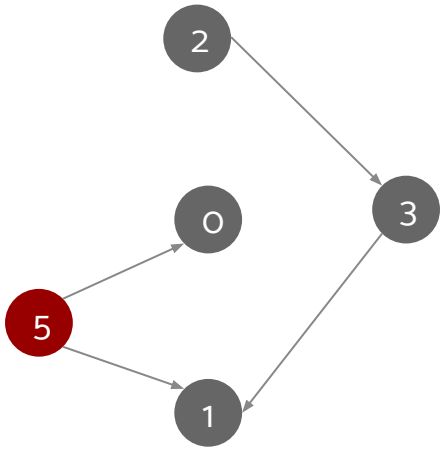
sources.add(2)

Sources
{5, 2}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4, 5]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[0, 1, 0, 1, 0, 0]

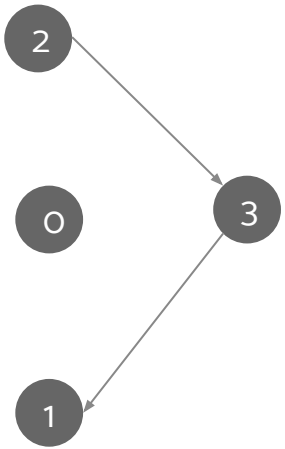
indegree[0] -= 1
indegree[1] -= 1

Sources
{5, 2}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4, 5]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[0, 1, 0, 1, 0, 0]

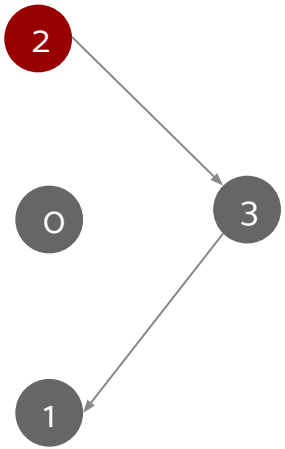
sources.add(0)

Sources
{2, 0}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4, 5, 2]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[0, 1, 0, 0, 0, 0]

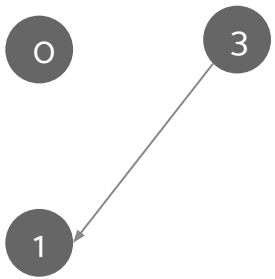
indegree[3] -= 1

Sources
{2, 0}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4, 5, 2]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[0, 1, 0, 0, 0, 0]

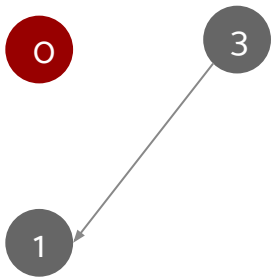
sources.add(3)

Sources
{0, 3}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4, 5, 2, 0]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[0, 1, 0, 0, 0, 0]

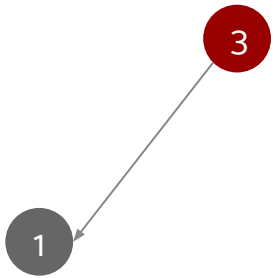
Sources
{0, 3}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:

Toposort
[4, 5, 2, 0, 3]



Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[0, 0, 0, 0, 0, 0]

indegree[1] -= 1

Sources
{3}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:

Toposort
[4, 5, 2, 0, 3]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

Indegrees
[0, 0, 0, 0, 0, 0]

sources.add(1)

Sources
{1}

2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:

Toposort
[4, 5, 2, 0, 3, 1]

Adjacency list
0: []
1: []
2: [3]
3: [1]
4: [0, 2]
5: [0, 1]

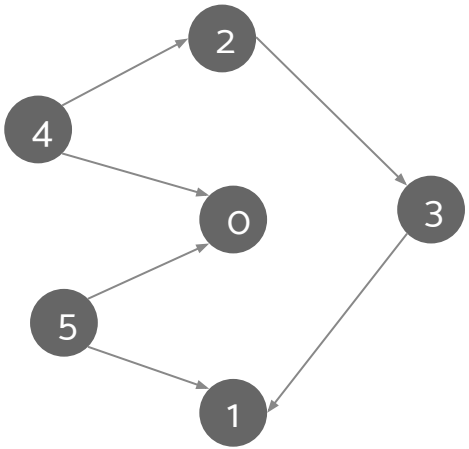
Indegrees
[0, 0, 0, 0, 0, 0]

Sources
{}

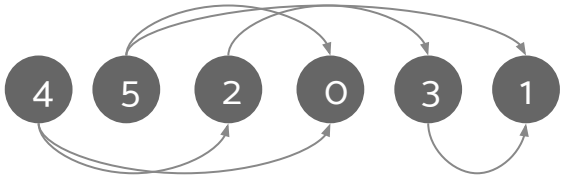
2 Topological Sorting for Cats

Next, describe an algorithm for finding all of the source nodes in a graph.

Example:



Toposort
[4, 5, 2, 0, 3, 1]



2 Topological Sorting for Cats

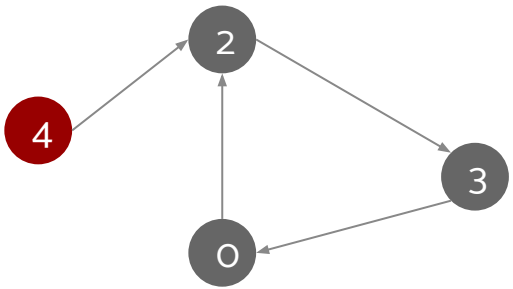
How can you modify the method `topologicalSort` above to detect whether the graph has a cycle?

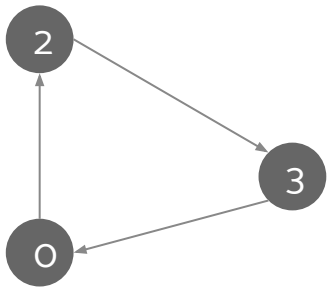
2 Topological Sorting for Cats

How can you modify the method `topologicalSort` above to detect whether the graph has a cycle?

If the graph has a cycle, at some point we will not be able to find any more sources, but there will still be things that we have not "removed" from the graph.

Therefore, we can check if there are any non-zero elements in the `indegree` array after the while loop. If there are, then the graph has a cycle.





3 Word Search

Given an N by N wordsearch and N words, devise an algorithm to solve the word-search in $O(N^3)$. For simplicity, assume no word is contained within another, i.e. if the word ”bear” is given, ”be” wouldn’t also be given.

Hint: Add the words to a `Trie`, and you may find the `longestPrefixOf` operation helpful. Recall that `longestPrefixOf` accepts a String key and returns the longest prefix of key that exists in the Trie, or `null` if no prefix exists.

3 Word Search

Algorithm:

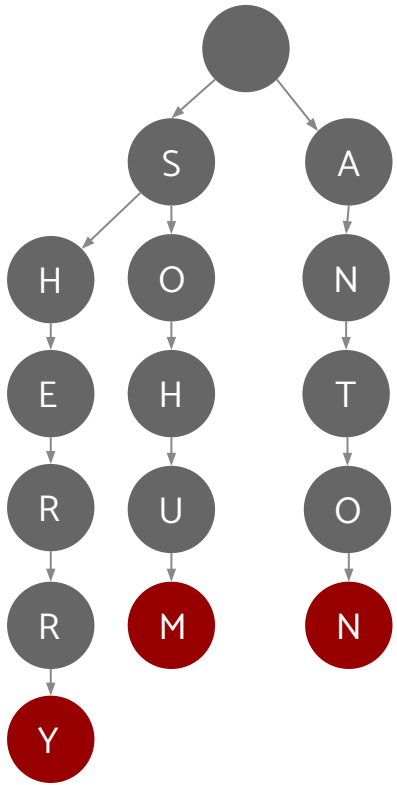
1. Add all words to a `Trie`.
2. For each letter in the word search:
    a. For each direction in [N, NE, E, SE, S, SW, W, NW]:
        i. Check `longestPrefixOf` in that direction
        ii. If `longestPrefixOf` returns a word, delete that word from our Trie.

3 Word Search

Example:

C M U H O **S** A E D
T R A T H A N K A

`longestPrefixOf("SOHUMC")`

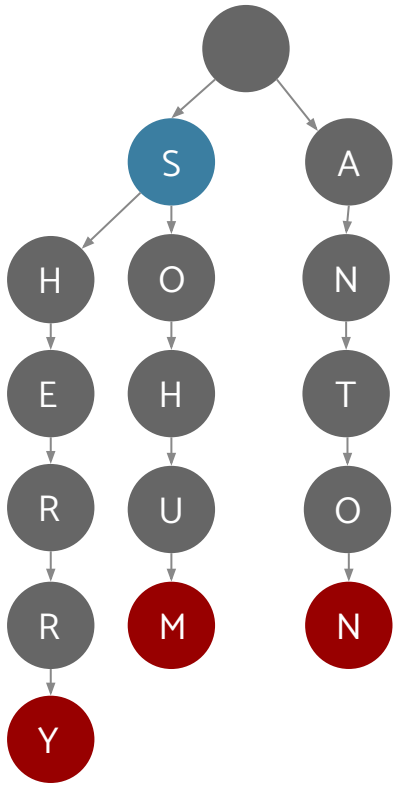


3 Word Search

Example:

C M U H O **S** A E D
T R A T H A N K A

`longestPrefixOf("SOHUMC")`

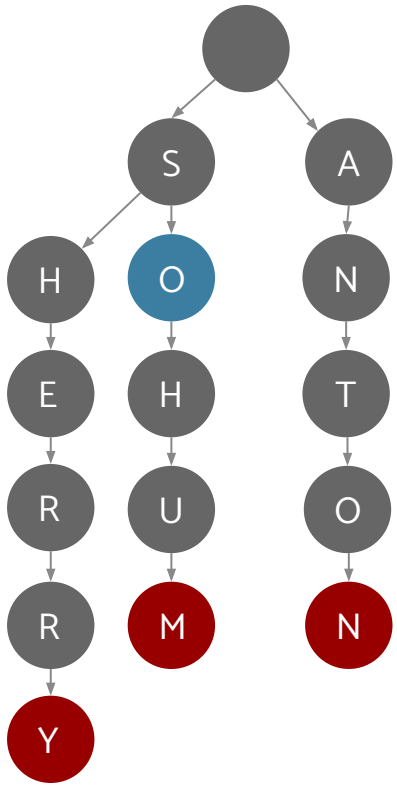


3 Word Search

Example:

C M U H **O** S A E D
T R A T H A N K A

`longestPrefixOf("SOHUMC")`

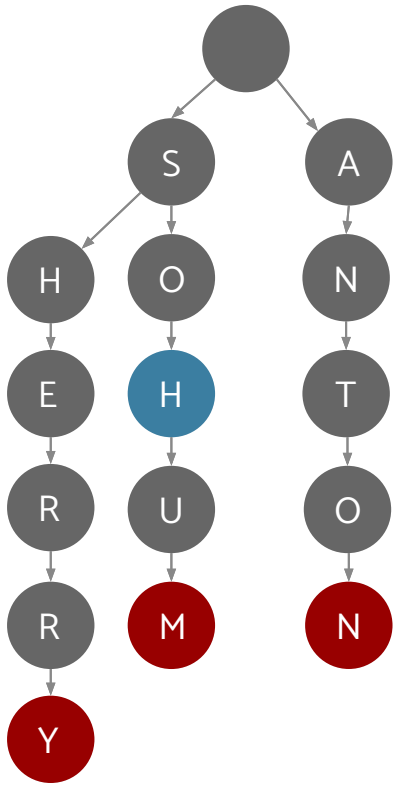


3 Word Search

Example:

```
C  M  U  H  O  S  A  E  D
T  R  A  T  H  A  N  K  A
```

`longestPrefixOf("SOHUMC")`

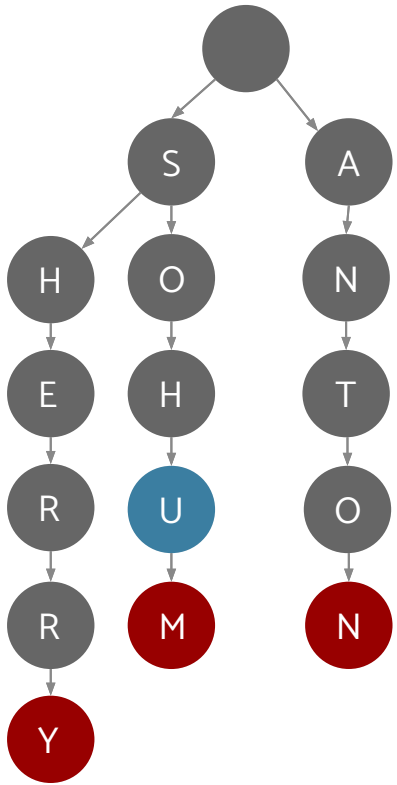


3 Word Search

Example:

C M **U** H O S A E D
T R A T H A N K A

`longestPrefixOf("SOHUMC")`

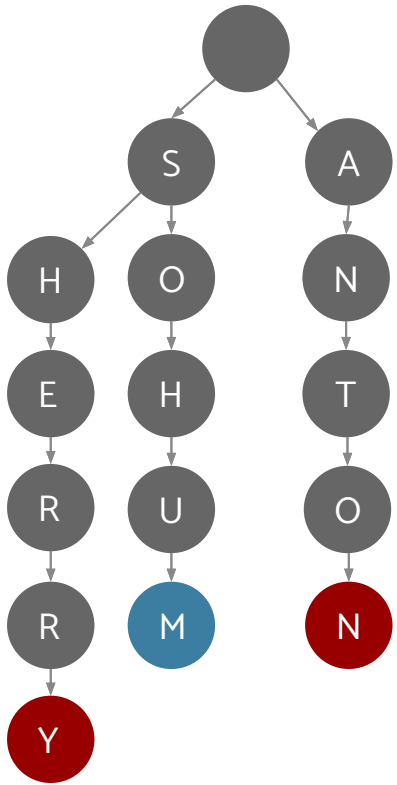


3 Word Search

Example:

C **M** U H O S A E D
T R A T H A N K A

longestPrefixOf(“SOHUMC”)
→ “SOHUM”



3 Word Search

Algorithm:

1. Add all words to a `Trie`.
2. For each letter in the word search:
    a. For each direction in [N, NE, E, SE, S, SW, W, NW]:
        i. Check `longestPrefixOf` in that direction
        ii. If `longestPrefixOf` returns a word, delete that word from our Trie.

Runtime: N^2 letters, and `longestPrefixOf` runs in O(L) time where L is the length of the string. L < N, so total runtime is $O(N^3)$.