Graphs II, Tries

Exam Prep 10

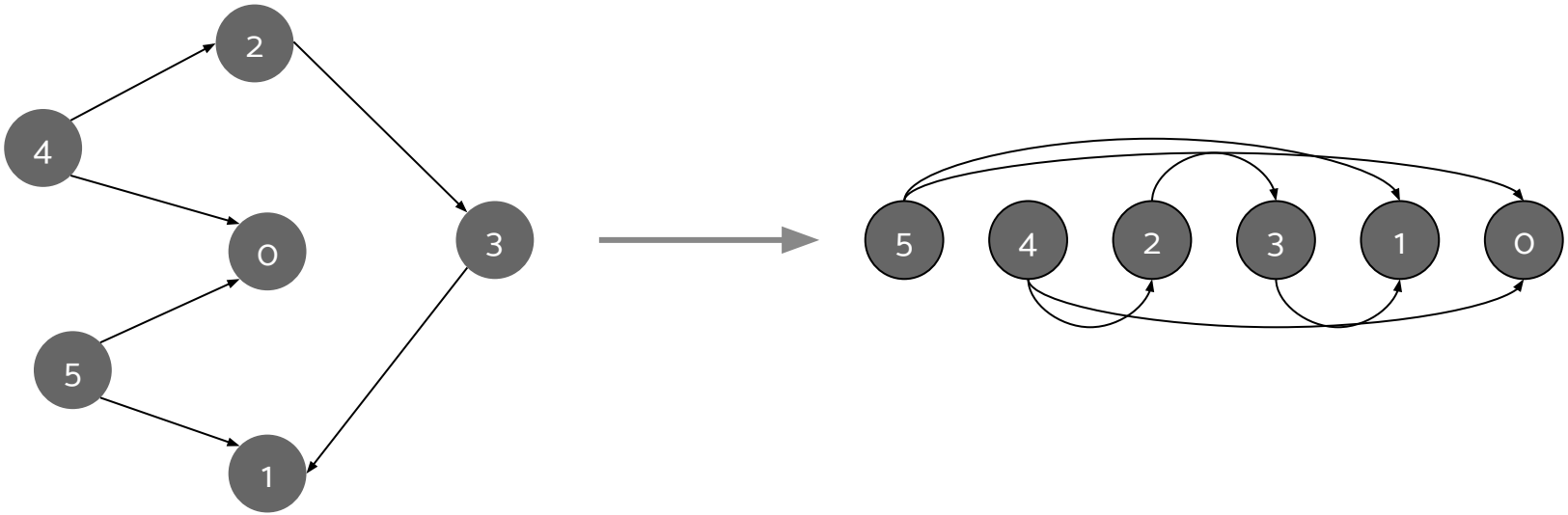
Announcements

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

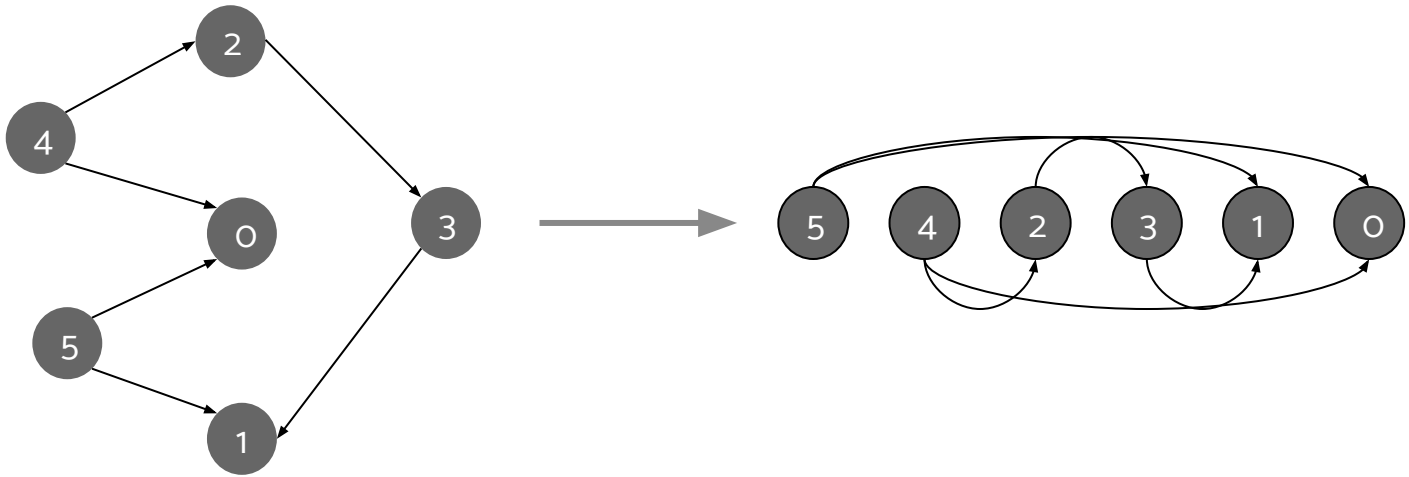
Topological Sort is a way of transforming a directed acyclic graph into a linear ordering of vertices, where for every directed edge u v, vertex u comes before v in the ordering.



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

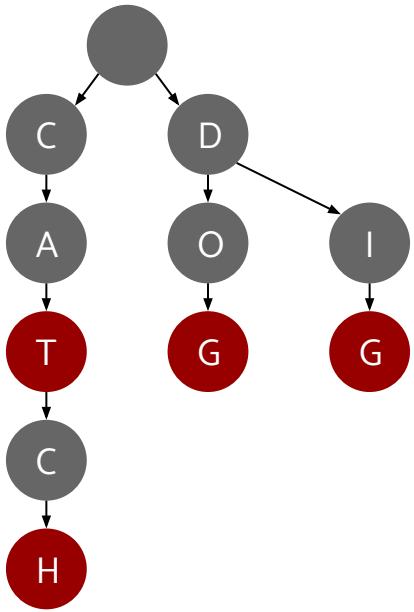
For a graph with V vertices and E edges:

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.

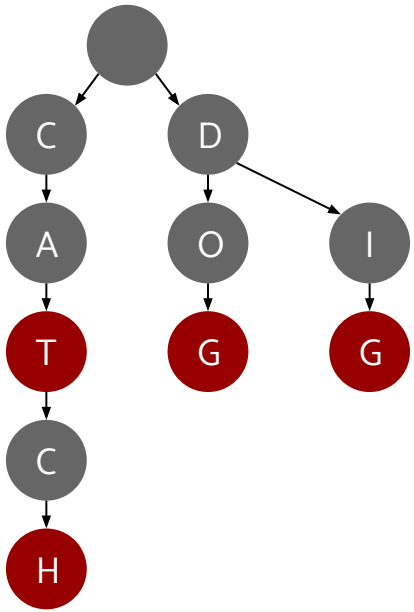
Each node in a trie is marked as being a word-end (a “key”) or not, so you can quickly check whether a word exists within your structure.



Trie Operations

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

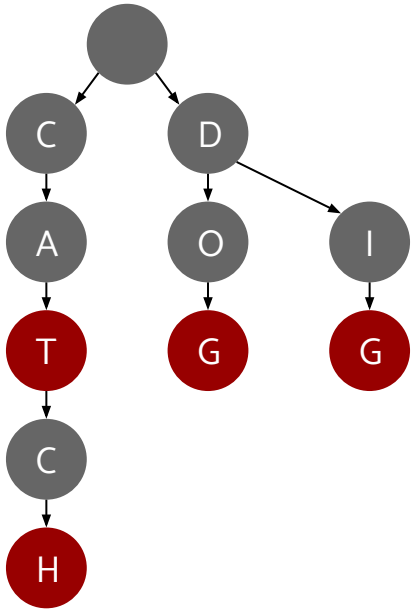
longestPrefixOf("catchall") → "catch"



Trie Operations

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

keysWithPrefix("ca") → "catch", "cat"



Worksheet

CS 61B Spring 2024

Graphs II, Tries

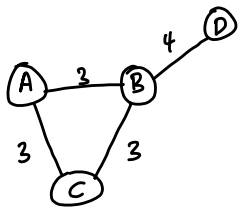
Exam-Level 10: April 1, 2024

1 Multiple MSTs

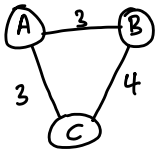
Recall a graph can have multiple MSTs if there are multiple spanning trees of minimum weight.

(a) For each subpart below, select the correct option and justify your answer. If you select "never" or "always," provide a short explanation. If you select "sometimes", provide two graphs that fulfill the given properties — one with multiple MSTs and one without. Assume G is an undirected, connected graph with at least 3 vertices.

Multiple



One



1. If **some** of the edge weights are **identical**, there will

   ◯ never be multiple MSTs in G.

   ☑ sometimes be multiple MSTs in G.

   ◯ always be multiple MSTs in G.

   Justification: Sometimes requires examples both ways

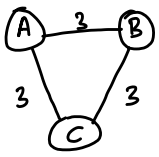
2. If **all** of the edge weights are **identical**, there will

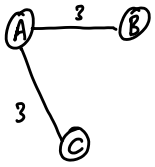
   ◯ never be multiple MSTs in G.

   ☑ sometimes be multiple MSTs in G.

   ◯ always be multiple MSTs in G.

   Justification: Same as previous



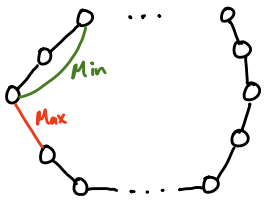


(b) Suppose we have a connected, undirected graph G with N vertices and N edges, where all the **edge weights are identical**. Find the maximum and minimum number of MSTs in G and explain your reasoning.

Minimum: ___3_____ smallest simple edge cycle is length 3

Maximum: ___N-1_____ any of the edges formed in the loop

Justification: Create a ring-like tree, then pick where to add the last edge



Insight: cycle of length ℓ has ℓ options for MST, pick one of the edges to remove ← # of edges, not vertices

(c) It is possible that Prim's and Kruskal's find **different** MSTs on the same graph `G` (as an added exercise, construct a graph where this is the case!). Given any graph `G` with integer edge weights, modify the edge weights of G to **ensure** that (1) Prim's and Kruskal's will output the same results, and (2) the output edges still form a MST correctly in the original graph. You may not modify Prim's or Kruskal's, and you may not add or remove any nodes/edges.
Hint: Look at subpart 1 of part a.

Yes, potentially there could be
Need to make the edges different, but they are integers
↳ Go through them randomly, and add $\frac{1}{|E|}$ to them

2 Topological Sorting for Cats

The big brain cat, Duncan, is currently studying topological sorts! However, he has a variety of **curiosities** that he wishes to satisfy.

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

Reverse edges, then postorder (can't preorder, since could explore wrong)

DFS runtime, so V+E runtime

(b) Duncan came up with another way to possibly do topological sorts, and he wants you to check him on its correctness and tell him if it is more efficient than our current way! Let's derive the algorithm.

1. First, provide a logical reasoning for the following claim (or a proof!): Every DAG has at least one source node, and at least one sink node.

Proof for both by contradiction
Sink: keep visiting vertices, after V times, must repeat → cycle, not DAG
Source: Reverse, apply sink reasoning

2. Duncan wishes to extend from the Graph class to create a DAG class. He wants to eventually add a method that enables topological sorting, but needs to write some helper methods first! Complete the following instance methods `computeInDegrees` and `findAllSourceNodes()`.

```
public class Graph {
    public Graph(int V) // Create empty graph with v vertices, numbered 0 to V - 1
    public void addEdge(int v, int w) // Adds edge from v to w
    Iterable<Integer> adj(int v) // Gets vertices adjacent to v
    int V() // Number of vertices
    int E() // Number of edges
}

public class DAG extends Graph {
    // Computes the number of incoming edges to a vertex
    public int[] computeInDegrees() {
1       int[] indegree = new int[V()];

2       for (int vertex = 0; vertex < V(); vertex ++) {

3           for (int adjVertex : adj(vertex)) {

4               indegree[adjVertex] += 1

            }
        }

        return indegree;
    }

    // Finds all source nodes in the graph
    public List<Integer> findAllSourceNodes(int[] indegree) {
        List<Integer> sources = new ArrayList<>();

1       for (int vertex = 0; vertex < V(); vertex ++) {

2           if (indegree[vertex] == 0) {

3               sources.add(vertex);

            }
        }

4       return sources;
    }
}
```

Order of Completion ↓

also length of indegree array

Runtime of `computeInDegrees` is: $\theta(V+E)$ - consider all vertices and edges

Runtime of `findAllSourceNodes` is: $\theta(V)$ - for loop

3. 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 class DAG extends Graph {
    public int[] computeInDegrees() { ... }
    public List<Integer> findAllSourceNodes(int[] indegree) { ... }

    public List<Integer> topologicalSort() {
        List<Integer> sorted = new ArrayList<>();

  1     int[] indegree = computeInDegrees();

        // Hint: add elements from another iterable here
  2     Queue<Integer> sources = new ArrayDeque<>(findAllSourceNodes(indegree));

  3     while (sources.length() > 0) {
            int source = sources.poll();

  4         sorted.add(source);

            for (int adjVertex : adj(source)) {

                indegree[adjVertex] --;

                if (indegree[adjVertex] == 0) {

                    sources.add(adjVertex)
                }
            }
        }
        return sorted;
    }
}
```

← two lines to create indegree array and source nodes! ↓ needs indegrees, so after

← also deletes the source

Runtime of `topologicalSort` is: V+E

4. Venti, the bard from Mondstadt is allergic to cats. He wanted to trick Duncan and created a DAG object, but it actually represents a graph with a cycle! How can you modify the method `topologicalSort()` above to detect whether the graph has a cycle?

Check if indegree array has nonzero elements once all sources are visited → if there is, then cycle

3 A Wordsearch

Given an N by N wordsearch and N words, devise an algorithm (using pseudocode or describe it in plain English) to solve the wordsearch 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.

If you are unfamiliar with wordsearches or want to gain some wordsearch solving intuition, see below for an example wordsearch. Note that the below wordsearch doesn't follow the precise specification of an N by N wordsearch with N words, but your algorithm should work on this wordsearch regardless.

Example Wordsearch:

C	M	U	H	O	S	A	E	D
T	R	A	T	H	A	N	K	A
O	C	Y	E	S	R	T	U	T
N	I	R	S	A	I	O	L	S
Y	R	R	M	T	N	N	H	R
Y	E	A	E	V	A	R	U	E
A	A	A	I	M	E	L	C	R
N	H	D	J	Y	U	A	C	I
T	Y	S	A	A	R	S	U	C
A	R	S	I	G	Y	E	S	A

ajay	anton
crystal	eric
grace	isha
luke	naama
rica	sarina
sherry	shreyas
sohum	sumer
tony	vidya

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.

1. Put words into Trie
2. Go through word search, letter by letter = N^2
3. For each letter, check if it's in Trie; if so, go in 8 directions and expand by $O(N-2 + 8) = O(N)$
   - word remainder
   - letter checks

$N^2 \cdot O(N) = N^3$