Graphs II, Tries

Exam Prep 11

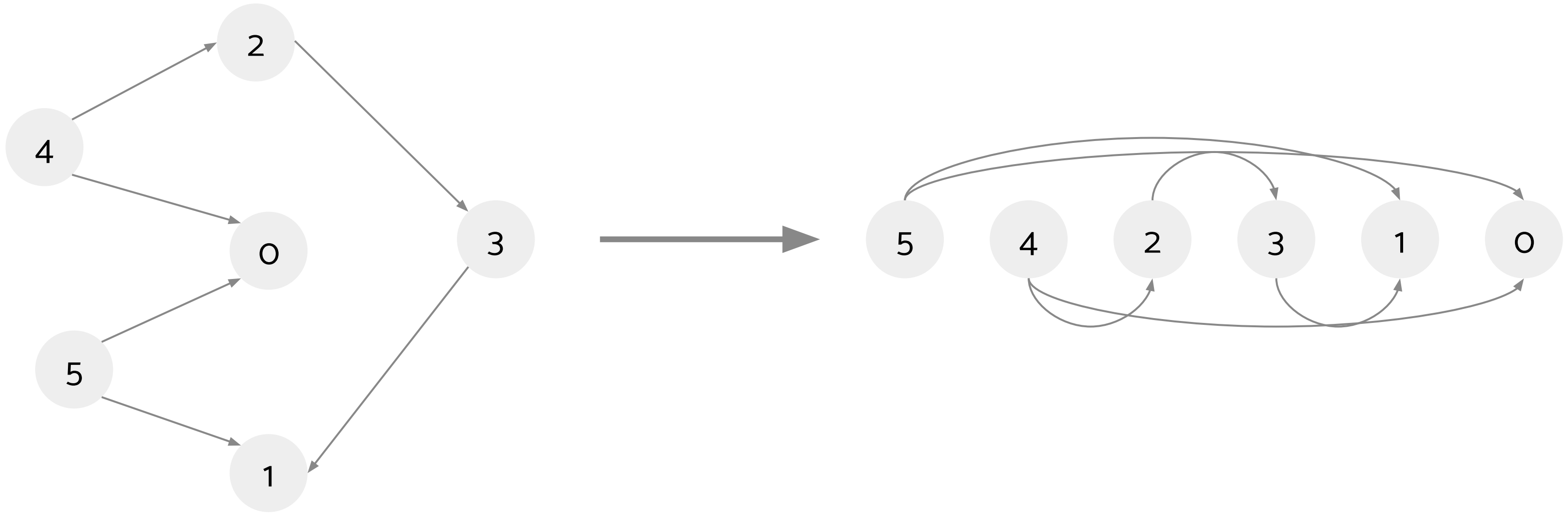
Announcements

- Week 10 Survey due Monday, Oct 30
- Proj 2b due Monday, Oct 30 (Tonight!!)
- Lab 11 (BYOW Intro) due Friday, Nov 3
- Tutor Review Session this Friday, Nov 3

Content Review

Topological Sort

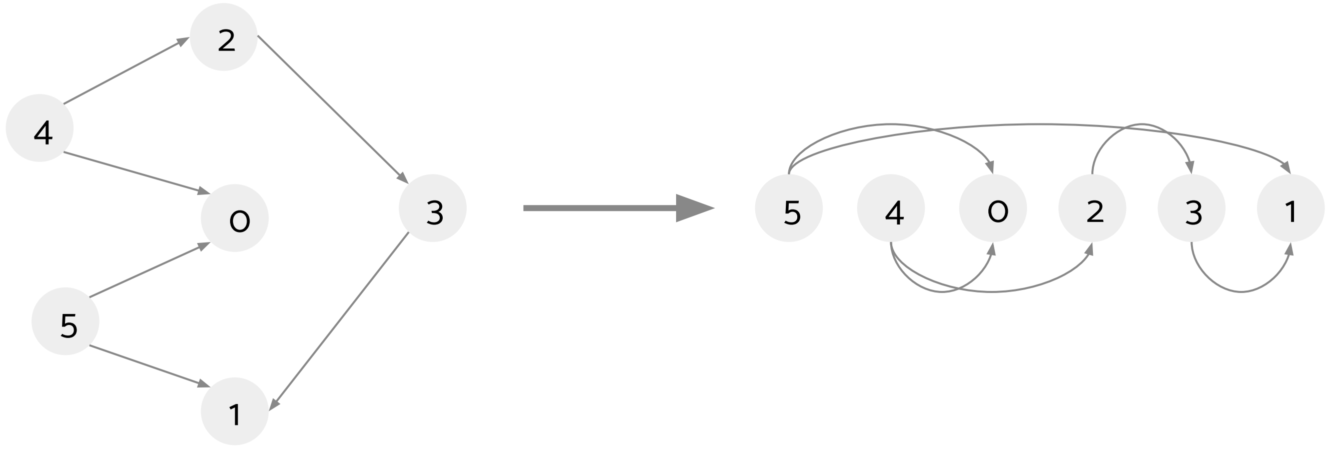
Topological Sort is a way of transforming a directed acyclic graph into a linear ordering of vertices, where for every directed edge u v, vertex u comes before v in the ordering.



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

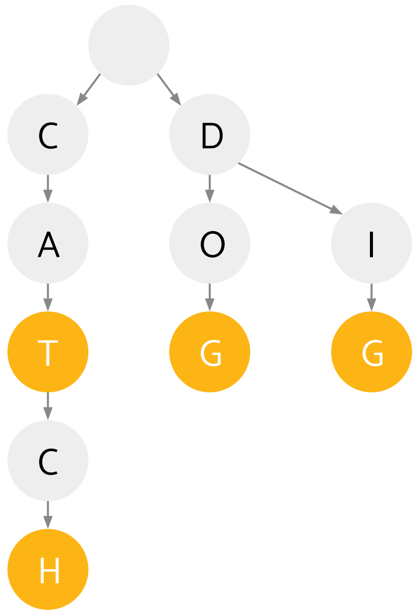
For a graph with V vertices and E edges:

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.

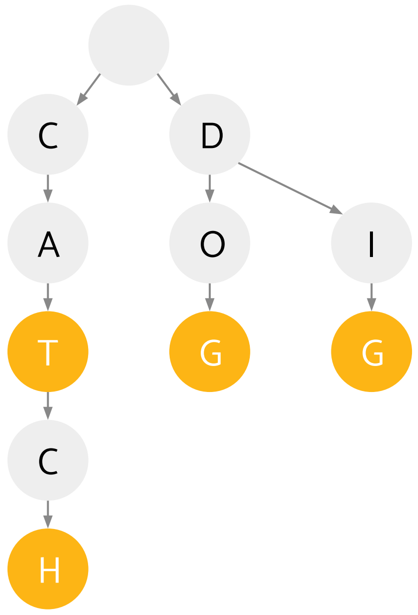
Each node in a trie is marked as being a word-end (a “key”) or not, so you can quickly check whether a word exists within your structure.



Trie Operations

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

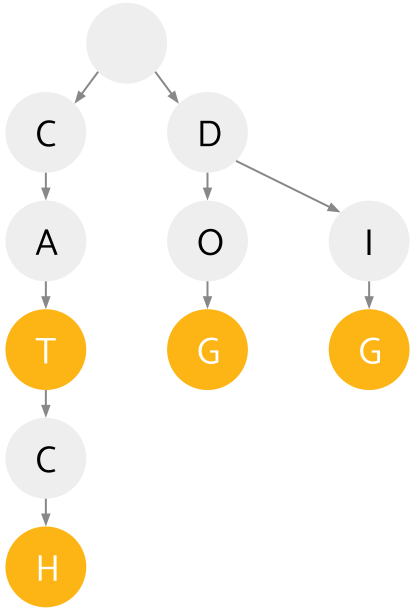
longestPrefixOf("catchall") → "catch"



Trie Operations

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

keysWithPrefix("ca") → "catch", "cat"



Worksheet

Graphs II, Tries

Exam-Level 11: October 30, 2023

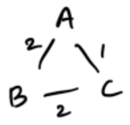
1 Multiple MSTs

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.

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.

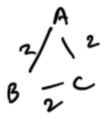
   

   Justification:

   For both: construct a tree → must be the only MST
   Can also construct examples that work - sometime

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.

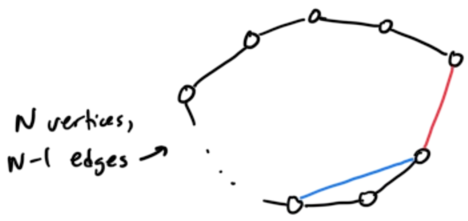
   Justification:

(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_____
Maximum: __N______

Justification:

Think of making a tree and determining where the last edge goes

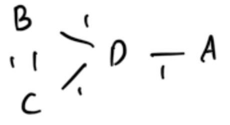


Case of 3 - can pick any edge of the 3 to remove

Case of N - can pick any edge in the loop to remove

(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.

Different based on order of consideration (i.e. tiebreaking scheme)



from A, kruskal's tiebreaking . Solution: no similar weight edges; add a small offset <1, $\frac{1}{E}$ to every edge that increases (avoids interfering if larger)

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 and do postorder
$\uparrow$
Indicates that all dependencies are finished

(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.

   True - that is the point of a DAG
   There has to be a sink - one "final node" with no edges going out else cycle
   Reverse and it's still a valid DAG - becomes a source

2. Next, describe an algorithm (in English or in pseudocode) for finding all of the source nodes in a graph.

   Reverse graph and DFS to find the sink nodes (which are source nodes)
   Other way tracks incoming edges per node (a little more unintuitive)

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, come up with an algorithm to topological sort. Describe it in words or using pseudocode. Is it more efficient than what we already have? *Hint: If it's easier for you, first consider one with quadratic runtime, then think about how you might save some computations to make it faster.*

   Technically Kahn's algorithm
   Idea leverages the in-degree array, removing items, updating array to track dependencies
   Quadratic: recompute sources every time

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. Add words to a trie - to use later on in checking if words are present
2. Iterate through the word search
   $\hookrightarrow N^2$
   3. Per letter and direction, check if continuing along that path is a word
      This involves using longest Prefix Of, with the key as the letters
      This is at most N - restricted by the size of the wordsearch for what words are possible

Approach to this question: N^3 is a hint, N^2 iteration

Figure out how to use Trie in N time