Data Structures with C++ : CS189

Lecture 10-1: Graphs Algorithms

Class Structure

- Phase 1: Do List and Vector from scratch
- Phase 2: Learn about trees and optimize them
- Phase 3: Learn about Graph algorithms and debug them
- Code, Optimize, Debug. That's your whole job when you graduate

Graphs

- A Directed Acyclical Graph (DAG) is a set of nodes (vertexes) connected by arrows (edges) that don't make a loop.
  - And "Draw a picture" is again essential
- How should we represent this in code?
  - Need to support arbitrary number of nodes and arbitrary number of edges
  - Need to be able to find all edges leaving a vertex as fast as possible
    - Remember, to pick a structure you need to know what you are using it for
  - We don't want to waste too much space in overhead

Two Solutions

- 2D array! (Vector of vectors)
  - mData[x][y] is true if there is an edge leading from x to y.
  - n^2 memory used, and 90% empty
  - O(1) lookup for "Are X and Y connected"
  - O(n) lookup for "All X point at"
- Map!
  - Vertex is key, list of edges out is Value
  - Use unordered version since order doesn't matter
  - O(n) lookup for "Are X and Y connected"
  - O(1) lookup for "All X point at"

Which is Best?

- The answer is "It depends"
  - A recurring theme in this class
- You need to analyze all the choices based on what your program and data need
- We are using choice B there. An unordered map of vertices to lists of edges
  - Book examples use the 2D array
- We can look at just the graph structure I provided now

Has Loop

Topological Sort

- Imagine vertexes are tasks, and arrows are dependencies
  - X -> Y means that X must be done before Y can be done. Y depends on X.
    - Each graph algo answers a different problem
- Is there a way I could finish every task, if I can only finish a task that is depending on nothing?
- If the graph has a loop, then no I can't
  - Topological Sort is the term for the order you can finish all tasks successfully

Graph vs Brain

- If I draw a graph on the board, your eyes and brain could find a loop in moments
- Computer doesn't have spacial awareness, and graphs don't exist in 2D space anyway
  - Graphs don't have a shape either. Just points and arrows
  - Half of the brain power to solve this is done the moment I draw it on the board

Has Loop Algorithm

- Find a node that nobody is pointing at and delete it
- If every node ends up deleted, then there is no loop, and the order you deleted them is your topological sort
- If vertices are left over, they must be a loop
- #1 use of this algorithm is in Operating Systems
  - Windows decides order to do tasks
  - Windows must decide if there is ever a deadlock

Guidelines for Graphs

- There are three approaches to each of the algorithms in the next few weeks
- External: Keep a data struct (map) local to the algorithm that keeps score of how many "ins" each vertex has
- Internal: Add a variable to each vertex so it can track how many vertices are pointing at it
- Copy: Make a copy of the whole graph and literally remove vertices as you go

Implementation

- We can look at the structure for HasLoop now
- Note how HasLoop uses all three of the techniques and combines them. You wouldn't do that in real life.
- Thinking past the algo in to C++, we can optimize this using our data structures
    - Every round through the algo, we are looking for a node with no ins
    - So keep a side list of all the current zeroes - you know the moment you change a one to a zero

End

Remember these algos are directly relatable to specific jobs now