Lecture 22: Let’s Get Graphic – Graph Algorithms

✦ Today’s Agenda:
  ➤ What is a graph?
  ➤ Some graphs that you already know
  ➤ Definitions and Properties
  ➤ Implementing Graphs
  ➤ Topological Sort

✦ Covered in Chapter 9 of the textbook

What are graphs? (Take 1)

✦ Yes, this is a graph….

✦ But we are interested in a different kind of “graph”
Motivation for Graphs

- Consider the data structures we have looked at so far…
- **Linked list**: nodes with 1 incoming edge + 1 outgoing edge
- **Binary trees/heaps**: nodes with 1 incoming edge + 2 outgoing edges
- **Binomial trees/B-trees**: nodes with 1 incoming edge + multiple outgoing edges
- **Up-trees**: nodes with multiple incoming edges + 1 outgoing edge

Motivation for Graphs

- What is common among these data structures?
- How can you generalize them?
- Consider data structures for representing the following problems…
Course Prerequisites for CSE at UW

Nodes = courses
Directed edge = prerequisite

Representing the Floor Plan of a House

Nodes = rooms
Edge = door or passage
Representing Electrical Circuits

Nodes = battery, switch, resistor, etc.
Edges = connections

Representing Expressions in Compilers

$x_1 = q + y \times z$
$x_2 = y \times z - q$

Naive:
- $y \times z$ calculated twice

common subexpression eliminated:
- Nodes = symbols/operators
- Edges = relationships
Information Transmission in a Computer Network

Nodes = computers
Edges = transmission rates

Traffic Flow on Highways

Nodes = cities
Edges = # vehicles on connecting highway
Soap Opera Relationships

- Victor
- Ashley
- Brad
- Michelle
- Wayne
- Peter

Six Degrees of Separation from Kevin Bacon

- Apollo 13
- Gary Sinise
- Tom Hanks
- Robin Wright
- Forest Gump
- The Princess Bride
- Wallace Shawn
- Cary Elwes
- Toy Story
- Laurie Metcalf
- Desperately Seeking Susan
- After Hours
- Cheech Marin
- Rosanna Arquette

Based on:
CSE 326, 2000
R. Rao, CSE 373
Six Degrees of Separation from Kevin Bacon

Graphs: Definition

- A graph is simply a collection of nodes plus edges
  - Linked lists, trees, and heaps are all special cases of graphs
- The nodes are known as vertices (node = “vertex”)
- Formal Definition: A graph $G$ is a pair $(V, E)$ where
  - $V$ is a set of vertices or nodes
  - $E$ is a set of edges that connect vertices
Graph Example

- Here is a graph $G = (V, E)$
  - Each edge is a pair $(v_1, v_2)$, where $v_1, v_2$ are vertices in $V$
  
  $V = \{A, B, C, D, E, F\}$
  
  $E = \{(A,B), (A,D), (B,C), (C,D), (C,E), (D,E)\}$

![Graph Example Diagram]

Directed versus Undirected Graphs

- If the order of edge pairs $(v_1, v_2)$ matters, the graph is directed (also called a digraph): $(v_1, v_2) \neq (v_2, v_1)$

  ![Directed Graph Example Diagram]

- If the order of edge pairs $(v_1, v_2)$ does not matter, the graph is called an undirected graph: in this case, $(v_1, v_2) = (v_2, v_1)$

  ![Undirected Graph Example Diagram]
Graph Representations

- Space and time are measured in terms of:
  - Number of vertices = $|V|$ and
  - Number of edges = $|E|$ 
- There are two ways of representing graphs:
  - The **adjacency matrix** representation
  - The **adjacency list** representation

---

**Graph Representation: Adjacency Matrix**

The **adjacency matrix** representation: 

\[ M(v, w) = \begin{cases} 
1 & \text{if } (v, w) \text{ is in } E \\
0 & \text{otherwise} 
\end{cases} \]

\[
\begin{pmatrix}
A & B & C & D & E & F \\
A & 0 & 1 & 0 & 1 & 0 & 0 \\
B & 1 & 0 & 1 & 0 & 0 & 0 \\
C & 0 & 1 & 0 & 1 & 1 & 0 \\
D & 1 & 0 & 1 & 0 & 1 & 0 \\
E & 0 & 0 & 1 & 1 & 0 & 0 \\
F & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]
Adjacency Matrix for a Digraph

\[ M(v, w) = \begin{cases} 
1 & \text{if } (v, w) \text{ is in } E \\
0 & \text{otherwise}
\end{cases} \]

\[
\begin{pmatrix}
A & B & C & D & E & F \\
A & 0 & 1 & 0 & 1 & 0 \\
B & 0 & 0 & 1 & 0 & 0 \\
C & 0 & 0 & 0 & 1 & 1 \\
D & 0 & 0 & 0 & 0 & 1 \\
E & 0 & 0 & 0 & 0 & 0 \\
F & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Space = \( |V|^2 \)

Graph Representation: Adjacency List

The \textit{adjacency list} representation: For each \( v \) in \( V \),

\( L(v) = \text{list of } w \text{ such that } (v, w) \text{ is in } E \)

Space = ?
Graph Representation: Adjacency List

Adjacency List for a Digraph

Space = $a |V| + 2b |E|$
Adjacency List for a Digraph

Graph Algorithm #1: Topological Sort

Problem: Find an order in which all these courses can be taken. Example: 142 → 143 → 378 → 370 → 321 → 341 → 322 → 326 → 421 → 401

To take a course, all its prerequisites must be taken first.
Topological Sort

**Topological sorting problem:** given digraph $G = (V, E)$, find a linear ordering of its vertices such that:
for any edge $(v, w)$ in $E$, $v$ precedes $w$ in the ordering.

A B C F D E

On-board example:
Topo-Sort this digraph

Any linear ordering in which all the arrows go to the right is a valid solution

A B F C D E

R. Rao, CSE 373
Topological Sort

Topological sorting problem: given digraph $G = (V, E)$, find a linear ordering of its vertices such that: for any edge $(v, w)$ in $E$, $v$ precedes $w$ in the ordering.

Not a valid topological sort!

Step 1: Identify vertices that have no incoming edges

- The “in-degree” of these vertices is zero.
Topological Sort Algorithm #1

Step 1: Identify vertices that have no incoming edges

- If no such vertices, graph has cycle(s) (cyclic graph)
- Topological sort not possible – Halt.

![Example of a cyclic graph](image)

Topological Sort Algorithm #1

Step 1: Identify vertices that have no incoming edges

- Select one such vertex

Select

![Example of a cyclic graph](image)
Topological Sort Algorithm #1

**Step 2**: Delete this vertex of in-degree 0 and all its outgoing edges from the graph. Place it in the output.

Repeat Step 1 and Step 2 until graph is empty
Topological Sort Algorithm #1

Repeat Step 1 and Step 2 until graph is empty
Topological Sort Algorithm #1

Repeat Step 1 and Step 2 until graph is empty

Final Result:

\[ \rightarrow A \quad B \quad F \quad C \quad D \quad E \]

Topological Sort Algorithm #1: Analysis

For input graph \( G = (V,E) \), Run Time = ?

Break down into total time to:

\[
\rightarrow \quad \text{Find a vertex with in-degree 0} \\
\rightarrow \quad \text{Remove its edges} \\
\rightarrow \quad \text{Place vertex in output}
\]

Assume adjacency list representation

\[ A \quad B \quad C \quad D \quad E \quad F \]

R. Rao, CSE 373
Topological Sort Algorithm #1: Analysis

Calculate and store In-Degree of all vertices in an array
→ Find vertex with in-degree 0: Search this array
→ Remove its edges: Update this array

In-Degree array

For input graph \( G = (V,E) \), Run Time = ?

Break down into total time to:
→ Find vertices with in-degree 0: \(|V|\) vertices, each takes \(O(|V|)\) to search In-Degree array = \(O(|V|^2)\)
→ Remove edges: \(|E|\) edges
→ Place vertices in output: \(|V|\) vertices

Total time = \(O(|V|^2 + |E|)\)

Can we do better than quadratic time?

Can you think of a faster way to find vertices with in-degree 0?
Next Class: Faster Topological Sort and Finding shortest ways to get to your classrooms

To Do:
Read and enjoy chapter 9
Have a great weekend!