CSCI 241

Lecture 20
Graph Representations, Topological Sort
Announcements

• A3 has phases!

• They’re all out now.

• My solution has 111 more lines of code than the skeleton.

• Don’t procrastinate: There’s a lot of conceptual stuff in those 111 lines.
A3 has 3 phases.

1. Write a min-heap to implement a priority queue with operations:
   - boolean add(V value, P priority)
   - V peek();
   - V poll();

2. Write a hash table.

3. Use the hash table to augment the heap, making the following operations efficient:
   - boolean contains(V v);
   - void changePriority(V v, P newP);
Goals

• Know some additional graph terminology:
  • Tree (graph definition), subgraph, planarity

• Understand how to represent a graph using:
  • adjacency list
  • adjacency matrix

• Be able to implement and analyze the runtime of simple graph operations on adjacency matrices and adjacency lists.

• Know how to detect whether a directed graph is acyclic using Topological Sort
Graph: a bunch of points connected by lines. The lines may have directions, or not.
Graphs: Abstract View

- $K_5$
- $K_{3,3}$
Graphs, Formally

• A directed graph (digraph) is a pair \((V, E)\) where:
  • \(V\) is a (finite) set
  • \(E\) is a set of ordered pairs \((u, v)\) where \(u, v\) are in \(V\)
  • Often (not always): \(u \neq v\) (i.e. no edges from a vertex to itself)

• An element in \(V\) is called a \textit{vertex} or \textit{node}

• Elements in \(E\) are called \textit{edges} or \textit{arcs}

• \(|V| = \text{size of } V\) (traditionally called \(n\))

• \(|E| = \text{size of } E\) (traditionally called \(m\))
An example directed graph

\[ V = \{A, B, C, D, E\} \]
\[ E = \{(A, C), (B, A), (B, C), (C, D), (D, C)\} \]
\[ |V| = 5 \]
\[ |E| = 5 \]
Graphs, Formally

- An **undirected graph** is just like a digraph, but
  - $E$ is a set of unordered pairs $(u, v)$ where $u, v \in V$
  - $V = \{A, B, C, D, E\}$
  - $E = \{\{A, C\}, \{B, A\}, \{B, C\}, \{C, D\}\}$
  - $|V| = 5$
  - $|E| = 4$

- An **undirected graph** can be converted to an equivalent **directed** graph:
  - Replace each undirected edge with two directed edges in opposite directions

- A **directed** graph can’t always be converted to an **undirected** graph.
Graph Terminology: Adjacency, Degree

- Two vertices are **adjacent** if they are connected by an edge.
- Nodes $u$ and $v$ are called the **source** and **sink** of the **directed** edge $(u, v)$.
- Nodes $u$ and $v$ are **endpoints** of an edge $(u, v)$ (directed or undirected).
- The **outdegree** of a vertex $u$ in a **directed** graph is the number of edges for which $u$ is the source.
- The **indegree** of a vertex $v$ in a **directed** graph is the number of edges for which $v$ is the sink.
- The **degree** of a vertex $u$ in an **undirected** graph is the number of edges of which $u$ is an endpoint.
Graph Terminology

- A **path** is a sequence of vertices where each consecutive pair are adjacent.
- In a directed graph, paths must follow the direction of the edges.
- A **cycle** is a path that ends where it started, e.g.: $x, y, z, x$
- A graph is **acyclic** if it has no cycles.
- A graph is **connected** if there is a path between every pair of nodes.
Graph Terminology

• A **path** is a sequence of vertices where each consecutive pair are adjacent.

• In a directed graph, paths must follow the direction of the edges.

• A **cycle** is a path that ends where it started, e.g.: x, y, z, x

• A graph is **acyclic** if it has no cycles.

• A graph is **connected** if there is a path between every pair of nodes.
Trees vs Graphs

- Trees are graphs!
- A tree is an **undirected graph** with exactly 1 **path** between all pairs of **nodes**.
- Implication: no **cycles**!

Many problems are easy in trees and harder in graphs.
Graph terminology: Lightning Round!

A: No  B: Yes

• Is graph G acyclic?

• Is there a path from 3 to 5 in graph H?

• Is graph H directed?

• Is (1,2) an edge in H?
Graph terminology: Lightning Round!

- What’s the degree of node 5 in graph G?
  A: 1  B: 2  C: 3  D: 4

- What is $|V|$ in graph G?
  A: 3  B: 4  C: 5  D: 6

- What is $|E|$ in graph H?
  A: 4  B: 5  C: 6  D: 7

- Is H connected?
  A: no  B: yes
Representing Graphs: Adjacency Lists

```java
public class GraphNode {
    // fields storing information about this node

    List<GraphNode> neighbors;
}
```

Node: Neighbors:

1 → 2 → 4
2 → 3
3
4 → 2 → 3
1 2 3 4
Representing Graphs: Adjacency Matrix

```java
public class Graph {
    boolean[][][] adjacent; // size n x n
}
```

**Adjacency lists:**
- **Node:** 1, 2, 3, 4
- **Neighbors:**
  - 1 → 2, 4
  - 2 → 3
  - 3
  - 4 → 2, 3

**Adjacency Matrix:**
```
    1 2 3 4
1 1 0 1 0 1
2 0 0 1 0
3 0 0 0 0
4 0 1 1 0
```
Adjacency Matrix vs Adjacency List

• Reminder: $n = |V|$ and $m = |E|$; let $d(u) = \text{degree of } u$

• Adjacency matrix:
  
  • Storage space: $O(n^2)$
  
  • Iterate over edges: $O(n^2)$ time
  
  • Check if there’s an edge from $u$ to $v$: $O(1)$
  
  • Good for dense graphs

  • e.g., if $|E|$ is close to $n^2$, you need $n^2$ storage anyway.
Adjacency Matrix vs Adjacency List

• Reminder: $n = |V|$ and $m = |E|$; let $d(u) = \text{degree of } u$

• Adjacency list:
  • Storage space: $O(n + e)$
  • Iterate over edges: $O(n + e)$ time
  • Check if there’s an edge from $u$ to $v$: $O(d(u))$
  • Good for more sparse graphs:
    • e.g., if $|E|$ is close to $n$, $n + e \approx 2n$, which is $O(n)$
Graph Algorithms

You can take entire graduate-level courses on graph algorithms. In this class:

• Topological Sort: detect cycles in a directed graph

• Search/traversal: search for a particular node or traverse all nodes
  • Breadth-first
  • Depth-first

• Spanning trees

• Shortest Paths
Graph Algorithms

You can take entire graduate-level courses on graph algorithms. In this class:

• Topological Sort: detect cycles in a directed graph

• Search/traversal: search for a particular node or traverse all nodes
  • Breadth-first
  • Depth-first

• Spanning trees

• Shortest Paths
DAGs

• A **DAG**, or **Directed Acyclic Graph** is a… graph that is directed and acyclic.

![Directed Acyclic Graph Diagram]
Is this a DAG?

• How do we tell if a directed graph is acyclic?
  
  • If a node has indegree 0, it can’t be part of a cycle.
  
  • Edges coming from that node also can’t be part of a cycle.

Algorithm:

while there is a node with indegree 0:
  
  delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG
Is this a DAG?

• How do we tell if a directed graph is acyclic?
  • If a node has indegree 0, it can’t be part of a cycle.
  • Edges coming from that node also can’t be part of a cycle.

Algorithm:

while there is a node with indegree 0:
  delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG
Is this a DAG?

• How do we tell if a directed graph is acyclic?
  • If a node has indegree 0, it can’t be part of a cycle.
  • Edges coming from that node also can’t be part of a cycle.

Algorithm:

while there is a node with indegree 0:
  delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG
Is this a DAG?

- How do we tell if a directed graph is acyclic?
  - If a node has indegree 0, it can’t be part of a cycle.
  - Edges coming from that node also can’t be part of a cycle.

Algorithm:

while there is a node with indegree 0:
  delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG
Is this a DAG?

• How do we tell if a directed graph is acyclic?
  • If a node has indegree 0, it can’t be part of a cycle.
  • Edges coming from that node also can’t be part of a cycle.

Algorithm:

while there is a node with indegree 0:
  delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG
Is this a DAG?

• How do we tell if a directed graph is acyclic?
  • If a node has indegree 0, it can’t be part of a cycle.
  • Edges coming from that node also can’t be part of a cycle.

Algorithm:

while there is a node with indegree 0:
  delete the node and all edges coming from it

if the graph is empty, the original graph was a DAG
Topological Sort

Topological sort (or toposort):

\[ i = 0 \]

while there is a node with indegree 0:

- delete* the node and all edges coming from it
- label* the deleted node i
- increment i

if the graph is empty, the original graph was a DAG
Topological Sort

Topological sort (or toposort):

\[ i = 0 \]

while there is a node with indegree 0:

delete* the node and all edges coming from it

label* the deleted node i

increment i

if the graph is empty, the original graph was a DAG

*This is pseudocode: we probably don’t want to actually modify the graph. We’ll need to store extra data with nodes and edges, and possibly overlay additional data structures to make it efficient."
Topological Sort

• Here are the labels we applied to the example graph:

• Property: all edges go from a lower-numbered node to a higher-numbered node.

• Useful for dependency resolution, job scheduling,

• Ordering is not necessarily unique: could have chosen from among multiple nodes with indegree 0.
Tensorflow Computation Graphs

Computation graph implementing the equation $z = 2 \times (a - b) + c$

- $a, b, c$: input tensors (scalar)
- $r_1, r_2$: intermediate result tensors
- $z$: tensor of the final result
The same graph can be drawn (infinitely!) many different ways.

\[ V = \{1, 2, 3, 4, 5, 6\} \]
\[ E = \{(1, 2), (2, 5), (3, 5), (4, 5), (5, 6)\} \]
Planarity

- If a graph can be drawn without crossing edges, it is **planar**.
Detecting Planarity

• It’s tricky! There’s a theorem that says a graph is **planar** if and only if it contains one of these as a **subgraph**:

![K_5](image1)  
![K_{3,3}](image2)

A **subgraph** of a graph is a graph whose vertex and edge sets are subsets of the larger graph’s.

• Elements of the edge subset can only contain nodes in the vertex subset.
Midterm

Grades will be released on Gradescope and Canvas after class today.

Exam is out of 50 points.
Guidelines for interpretation:

43+: A range

35+: B range

<35: C range

We’re not perfect. Check your exam for grading errors!