CSCI 241
Lecture 18
Intro to Graphs; Graph Representation; Graph Traversals
Announcements

• A2 grades are out.
  • Please pull the grading branch to see your feedback.
  • As usual, you can resubmit once for half unit test credit back.

• A3 is due Wednesday
  • My solution has 111 more lines than the skeleton.
  • The concepts-to-code ratio is high.

• A4 out Wednesday, due the following Wednesday
  • We’ll cover the algorithm on Wednesday and Friday.
Happenings

Wednesday, 3/6 – Peer Lecture Series: VIM Workshop – 5 pm in CF 420

Wednesday, 3/6 – Cybersecurity Lecture Series: Cyber and Physical Security Standards in the Power Industry – 5 pm in CF 105

Thursday, 3/7 – Elevator Speech Workshop – 6 pm in CF 316

Friday, 3/8 – AWC Women in Industry Panel – 4 pm in AW 210
Goals

• Know the definition of a graph and basic associated terminology:
  • Node/vertex; edge/arc; directed, undirected; adjacent; (in/out-)degree; path; cycle;

• 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 implement breadth-first and depth-first graph traversals.
THESE AREN'T THE GRAPHS YOU'RE LOOKING FOR
Graph: a bunch of points connected by lines. The lines may have directions, or not.
This is a graph:

The internet's undersea world

[Description of the graph: A world map with various cables linking continents, representing the undersea communication network.]
The edges are made of these:
Social Networks
(before they were cool)
Social Networks
(before they were cool)
The USA as a graph:

- Neighboring states are connected by edges.
Electrical circuit
A bigger electrical circuit
This is not a graph:

**it is a cat.**
This is a graph that can recognize cats.
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 vertex or node

- Elements in \(E\) are called edges or 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 a just like a digraph, but
  - $E$ is a set of unordered pairs $(u, v)$ where $u, v$ are 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: Paths, Cycles

- **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 (nodes must be ordered source then sink).

- **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.
Representing Graphs: Adjacency Lists

```java
public class GraphNode {
    // fields storing information
    // about this node
    List<GraphNode> neighbors;
}
```

Node: 1 → 2 → 4
      2 → 3
      3
      4 → 2 → 3

Diagram:
```
1 ——— 2 ——— 4
  |      |      |
  |      |      |
  |      |      |
  |      |      |
  |      |      |
  |      |      |
  |      |      |
  |      |      |
  |      |      |
      3 ——— 4
```
Representing Graphs: Adjacency Matrix

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

Adjacency lists:

<table>
<thead>
<tr>
<th>Node</th>
<th>Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2, 3</td>
</tr>
</tbody>
</table>
Representing Graphs: Adjacency Matrix

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

Adjacency lists:

<table>
<thead>
<tr>
<th>Node</th>
<th>Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>→ 2 → 4</td>
</tr>
<tr>
<td>2</td>
<td>→ 3</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>→ 2 → 3</td>
</tr>
</tbody>
</table>

Adjacency Matrix:

```
   1 2 3 4
1 0 1 0 1
2 0 0 1 0
3 0 0 0 0
4 0 1 1 0
```
Let \( n = |V| \) and \( m = |E| \); let \( d(u) \) = degree of \( u \)

- ABCD: How much space does it take to store \( G \) as an adjacency list vs. adjacency matrix?
  
  A. List: \( O(n^2 + e) \); Matrix: \( O(n^2) \)
  
  B. List: \( O(n + e) \); Matrix: \( O(n + e) \)
  
  C. List: \( O(n^2) \); Matrix: \( O(n + e^2) \)
  
  D. List: \( O(n + e) \); Matrix: \( O(n^2) \)
Let \( n = |V| \) and \( m = |E| \); let \( d(u) = \text{degree of } u \).

- ABCD: How much space does it take to store \( G \) as an adjacency list vs. adjacency matrix?
  
  A. List: \( O(n^2 + e) \); Matrix: \( O(n^2) \)
  B. List: \( O(n + e) \); Matrix: \( O(n + e) \)
  C. List: \( O(n^2) \); Matrix: \( O(n + e^2) \)
  D. List: \( O(n + e) \); Matrix: \( O(n^2) \)
Let \( n = |V| \) and \( m = |E| \); let \( d(u) \) = degree of \( u \).

ABC: What's the runtime of iterating over all edges?

- A. List: \( O(n^2) \); Matrix: \( O(n^2) \)
- B. List: \( O(n+e) \); Matrix: \( O(n^2) \)
- C. List: \( O(n + e) \); Matrix: \( O(n + e) \)
- D. List: \( O(n+e) \); Matrix: \( O(n^2 + e) \)
Let $n = |V|$ and $m = |E|$; let $d(u) =$ degree of $u$.

**ABCD:** What’s the runtime of iterating over all edges?

A. List: $O(n^2)$; Matrix: $O(n^2)$
B. List: $O(n+e)$; Matrix: $O(n^2)$
C. List: $O(n + e)$; Matrix: $O(n + e)$
D. List: $O(n + e)$; Matrix: $O(n^2 + e)$
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 \( n^2 \) 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 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 \( n^2 \) is close to \( n^2 \), you need \( n^2 \) storage anyway.
Adjacency Matrix vs Adjacency List

• Reminder: \( n = |V| \) and \( m = |E| \); let \( d(u) \) = 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:

• Search/traversal: search for a particular node or traverse all nodes (Lab 9)
  • Breadth-first
  • Depth-first

• Shortest Paths (A4)

• Spanning trees
Graph Algorithms

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

- **Search/traversal**: search for a particular node or traverse all nodes (Lab 9)
  - Breadth-first
  - Depth-first
- **Shortest Paths** (A4)
- **Spanning trees**