CSCI 136:
Data Structures
and
Advanced Programming
Lecture 31
Graphs
Instructor: Dan Barowy
Williams

Your to-dos
1. Read before Fri: Bailey, Ch. 13.4.
2. Lab 10 (partner lab), due Tuesday 5/10 by 10pm.

Topics
Graphs

Announcements
Suresh Venkatasubramanian (White House; Brown U)
Friday, May 6 @ 2:35pm*
Computer Science Colloquium – Wege TCL 123
On Equity in Access
*Williams students, faculty and staff only.

Suresh Venkatasubramanian is a professor in computer science and data science, currently at the White House in the Office of Science and Technology Policy. His background is in theoretical computer science, and he's taken a long and winding path through many areas of data science. For almost the past decade, he's been interested in algorithmic fairness, and more broadly the impact of automated decision-making systems in society.
Graphs

Graph operations

Fundamental graph ADT operations

bool adjacent(Vertex u, Vertex v):
Given vertices u and v, are they adjacent?
(i.e., share an edge?)

adjacent(a, d) = true
adjacent(a, b) = false
adjacent(a, c) = false

Fundamental graph ADT operations

bool incident(Vertex v, Edge e):
Given vertex v and edge e, are they incident?
(i.e., is v an endpoint of edge e?)

incident(a, 1) = true
incident(a, 2) = false
**Fundamental graph ADT operations**

`vertices(1) = [a, b]`
`vertices(2) = [d, b]`

`Vertex[] vertices(Edge e):`
Given edge `e`, what are its end points?

`vertices(1) = [a, b]`
`vertices(2) = [d, b]`

`int degree(Vertex v):`
Given vertex `v` how many vertices are adjacent?

`degree(a) = 2`
`degree(c) = 0`

**Graph representations**

`Vertex[] neighbors(Vertex v):`
Given vertex `v` what other vertices are adjacent?

`neighbors(a) = [d, b]`
`neighbors(c) = []`
An **adjacency matrix** is a data structure for representing a finite graph. It consists of a **square matrix** (usually implemented as an array of arrays). In the simplest case, the elements of the matrix indicate whether an edge is present. Elements on the diagonal are defined as zero.

In an **undirected graph**, the adjacency matrix is **symmetric**.
In an **undirected graph**, the adjacency matrix is **symmetric**.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

In a **directed graph**, the adjacency matrix is **not symmetric** because edges are directed. A directed edge, from $\rightarrow$ to, is conventionally encoded in **row-major** form.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

In a **directed graph**, the adjacency matrix is **not symmetric** because edges are directed. A directed edge, from $\rightarrow$ to, is conventionally encoded in **row-major** form.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Adjacency matrix

In a directed graph, the adjacency matrix is not symmetric because edges are directed. A directed edge, from→to, is conventionally encoded in row-major form.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Adjacency list

An adjacency list is a data structure for representing a finite graph. It consists of a list of unordered lists.

```plaintext
[[c,d],[d,b],[a,b]]
```

Adjacency list

There are many variants on adjacency lists. The most common is the object-oriented adjacency list that stores a list of adjacent vertices in each vertex object.

```plaintext
a: [b]
b: [a,d]
c: [d]
d: [b,c]
```

Adjacency list

Object-oriented adjacency list:

```java
public class Vertex<T> {
    T label;
    List<Vertex<T>> neighbors = new SinglyLinkedList<>();
}
```

(strictly speaking, c and d are references to Vertex objects)
Adjacency list

This latter version is especially thrifty for directed graphs.

a: []
b: [a, d]
c: []
d: [c]

Activity

Write down both adjacency matrix and adjacency list representations for this graph.

Which one is better for this graph? Why? (think Big-O)

Recap & Next Class

Today:

Graph operations
Graph representations

Next class:

Heaps and priority queues