Graph Traversals
Final Project Mid-Project Check-ins

Mid-project check-ins this week (November 8th-12th)!

If you haven’t signed up yet do so today!

To sign up for a mid-project meeting time:

Learning Objectives

- Implement depth first search traversal on graphs
- Compare BFS and DFS utility
- Discuss how implementations change due to weight and direction
Graph Implementation: Adjacency List

Vertex Storage:

Edge Storage:
Traversal:

Objective: Visit every vertex and every edge in the graph.

Purpose: Search for interesting sub-structures in the graph.

We’ve seen traversal before ....but it’s different:

- Ordered
- Obvious Start
- Clear End

- Any Order
- Any Start
- End is not obvious
Traversals: BFS

Initialize queue

While queue not empty

Dequeue v

Process v

Assign labels to v and e

Add unvisited neighbors to queue

<table>
<thead>
<tr>
<th>v</th>
<th>d</th>
<th>P</th>
<th>Adjacent Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-</td>
<td>B C D</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>A</td>
<td>A C E</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>A</td>
<td>A B D E F</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>A</td>
<td>A C F H</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>B</td>
<td>B C G</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>C</td>
<td>C D G</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>E</td>
<td>E F H</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>D</td>
<td>D G</td>
</tr>
</tbody>
</table>
BFS Observations

**Obs. 1:** BFS can be used to count components.

**Obs. 2:** BFS can be used to detect cycles.

**Obs. 3:** In BFS, $d$ provides the shortest distance to every vertex.

**Obs. 4:** The runtime of BFS is $O(|V| + |E|)$
Traversal: DFS
Traversal: DFS

Diagram of a graph with labeled nodes and edges indicating discovery and back edges.
Traversal: DFS

Discovery Edge

Back Edge
def DFS_recur(self, s):
    self.V[s].visited = 1
    el = self.getEdges(s)
    for e in el:
        l = e.v2
        v = self.V[l]
        if v.visited == 0:
            e.label = 1
            e.parallel.label = 1
            self.DFS_recur(l)
        elif e.label == 0:
            e.label = 2
            e.parallel.label = 2
DFS Observations

**Obs. 1:** DFS can be used to count components.

**Obs. 2:** DFS can be used to detect cycles.

**Obs. 3:** In DFS, distance provides no clear meaning.
<table>
<thead>
<tr>
<th></th>
<th>DFS</th>
<th></th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros:</strong></td>
<td></td>
<td><strong>Pros:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Cons:</strong></td>
<td></td>
<td><strong>Cons:</strong></td>
<td></td>
</tr>
</tbody>
</table>
Graph Implementation: Edge List  \(|V| = n, |E| = m\)

Vertex Storage:

Edge Storage:
Graph Implementation: Adjacency Matrix

Vertex Storage:

Edge Storage:

<table>
<thead>
<tr>
<th></th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Adjacency List

class vertObj():
    def __init__(self, v):
        self.val = v
        self.eList = []

class edgeObj():
    def __init__(self, e):
        self.v2 = e
        self.parallel = None
“The Muddy City” by CS Unplugged, Creative Commons BY-NC-SA 4.0