CS 225

Data Structures

October 26 – Graph Traversals

G Carl Evans
mp_mazes AMA

Today at 7pm on Zoom

Meeting ID: 820 2669 7494
Passcode: 139341

https://illinois.zoom.us/j/82026697494?pwd=OVM0SWdPTCtKSjZ1TXBKUE10MmJYQT09
Graphs

To study all of these structures:
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms
BFS(G):

Input: Graph, G
Output: A labeling of the edges on
G as discovery and cross edges

foreach (Vertex v : G.vertices()):
  setLabel(v, UNEXPLORED)

foreach (Edge e : G.edges()):
  setLabel(e, UNEXPLORED)

foreach (Vertex v : G.vertices()):
  if getLabel(v) == UNEXPLORED:
    BFS(G, v)

BFS(G, v):

Queue q
setLabel(v, VISITED)
q.enqueue(v)

while !q.empty():
  v = q.dequeue()
  foreach (Vertex w : G.adjacent(v)):
    if getLabel(w) == UNEXPLORED:
      setLabel(v, w, DISCOVERY)
    setLabel(w, VISITED)
    q.enqueue(w)
  elseif getLabel(v, w) == UNEXPLORED:
    setLabel(v, w, CROSS)
Traversal: BFS

<table>
<thead>
<tr>
<th>v</th>
<th>d</th>
<th>P</th>
<th>Adjacent Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Traversal: BFS

A

B

C

D

E

F

G

H

<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>C B D</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>A C E</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>B A D E F</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>A C F H</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>B C G</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td>C D G</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td>E F H</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td>D G</td>
</tr>
</tbody>
</table>
Traversal: BFS

<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>C B 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>B A 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>C</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>

Traversed order: G H F E D B C A
BFS Analysis

**Q:** Does our implementation handle disjoint graphs? If so, what code handles this?
  • *How do we use this to count components?*

**Q:** Does our implementation detect a cycle?
  • *How do we update our code to detect a cycle?*

**Q:** What is the running time?
BFS(G):
  Input: Graph, G
  Output: A labeling of the edges on G as discovery and cross edges
  foreach (Vertex v : G.vertices()):
      setLabel(v, UNEXPLORED)
  foreach (Edge e : G.edges()):
      setLabel(e, UNEXPLORED)
  foreach (Vertex v : G.vertices()):
      if getLabel(v) == UNEXPLORED:
          BFS(G, v)

BFS(G, v):
  Queue q
  setLabel(v, VISITED)
  q.enqueue(v)
  while !q.empty():
      v = q.dequeue()
      foreach (Vertex w : G.adjacent(v)):
          if getLabel(w) == UNEXPLORED:
              setLabel(v, w, DISCOVERY)
              setLabel(w, VISITED)
              q.enqueue(w)
          elseif getLabel(v, w) == UNEXPLORED:
              setLabel(v, w, CROSS)
Running time of BFS

While-loop at :\textbf{19}?

For-loop at :\textbf{21}?

<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>C B 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>B A 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>C</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**

**Q:** What is a shortest path from **A** to **H**?

**Q:** What is a shortest path from **E** to **H**?

**Q:** How does a cross edge relate to **d**?

**Q:** What structure is made from discovery edges?
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:** In BFS, the endpoints of a cross edge never differ in distance, \(d\), by more than 1:
\[
|d(u) - d(v)| = 1
\]
Traversal: DFS
BFS(G):
  Input: Graph, G
  Output: A labeling of the edges on G as discovery and cross edges
  foreach (Vertex v : G.vertices()):
    setLabel(v, UNEXPLORED)
  foreach (Edge e : G.edges()):
    setLabel(e, UNEXPLORED)
  foreach (Vertex v : G.vertices()):
    if getLabel(v) == UNEXPLORED:
      BFS(G, v)

BFS(G, v):
  Queue q
  setLabel(v, VISITED)
  q.enqueue(v)
  while !q.empty():
    v = q.dequeue()
    foreach (Vertex w : G.adjacent(v)):
      if getLabel(w) == UNEXPLORED:
        setLabel(v, w, DISCOVERY)
        setLabel(w, VISITED)
        q.enqueue(w)
      elseif getLabel(v, w) == UNEXPLORED:
        setLabel(v, w, CROSS)
DFS(G):
Input: Graph, G
Output: A labeling of the edges on G as discovery and back edges

foreach (Vertex v : G.vertices()):
    setLabel(v, UNEXPLORED)

foreach (Edge e : G.edges()):
    setLabel(e, UNEXPLORED)

foreach (Vertex v : G.vertices()):
    if getLabel(v) == UNEXPLORED:
        DFS(G, v)

DFS(G, v):
Queue q
setLabel(v, VISITED)
q.enqueue(v)

while !q.empty():
    v = q.dequeue()
    foreach (Vertex w : G.adjacent(v)):
        if getLabel(w) == UNEXPLORED:
            setLabel(v, w, DISCOVERY)
            setLabel(w, VISITED)
            DFS(G, w)
        elseif getLabel(v, w) == UNEXPLORED:
            setLabel(v, w, BACK)
Traversal: DFS
Traversals: DFS

Diagram:

Nodes: A, B, C, D, E, F, G, H, K, J

Edges:
- Discovery Edge
- Back Edge
BFS(G):

Input: Graph, G
Output: A labeling of the edges on G as discovery and cross edges

foreach (Vertex v : G.vertices()):
  setLabel(v, UNEXPLORED)

foreach (Edge e : G.edges()):
  setLabel(e, UNEXPLORED)

foreach (Vertex v : G.vertices()):
  if getLabel(v) == UNEXPLORED:
    BFS(G, v)

BFS(G, v):

Queue q
setLabel(v, VISITED)
q.enqueue(v)

while !q.empty():
  v = q.dequeue()
  foreach (Vertex w : G.adjacent(v)):
    if getLabel(w) == UNEXPLORED:
      setLabel(v, w, DISCOVERY)
      setLabel(w, VISITED)
      q.enqueue(w)
    elseif getLabel(v, w) == UNEXPLORED:
      setLabel(v, w, CROSS)
DFS(G):
Input: Graph, G
Output: A labeling of the edges on G as discovery and back edges

dfs(G):
foreach (Vertex v : G.vertices()):
  setLabel(v, UNEXPLORED)
foreach (Edge e : G.edges()):
  setLabel(e, UNEXPLORED)
foreach (Vertex v : G.vertices()):
  if getLabel(v) == UNEXPLORED:
    dfs(G, v)
Running time of DFS

Labeling:
• Vertex:

• Edge:

Queries:
• Vertex:

• Edge:
Minimum Spanning Tree Algorithms

**Input:** Connected, undirected graph $G$ with edge weights (unconstrained, but must be additive)

**Output:** A graph $G'$ with the following properties:
- $G'$ is a spanning graph of $G$
- $G'$ is a tree (connected, acyclic)
- $G'$ has a minimal total weight among all spanning trees
Kruskal’s Algorithm

The algorithm proceeds as follows:

1. Sort all the edges from the smallest weight to the largest weight.
2. Pick the smallest edge. If it forms a cycle, discard it. Otherwise, include it to form a tree.
3. Pick the next smallest edge. If it forms a cycle with the edges already in the tree, discard it. Otherwise, include it to form a larger tree.
4. Repeat step 3 until there are (V-1) edges in the tree, where V is the number of vertices.

The edges used in the example are:

- (A, D)
- (E, H)
- (F, G)
- (A, B)
- (B, D)
- (G, E)
- (G, H)
- (E, C)
- (C, H)
- (E, F)
- (F, C)
- (D, E)
- (B, C)
- (C, D)
- (A, F)
- (D, F)
Kruskal’s Algorithm

(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)
Kruskal’s Algorithm

KruskalMST(G):
1. DisjointSets forest
2. foreach (Vertex v : G):
   3.   forest.makeSet(v)
4. 
5. PriorityQueue Q    // min edge weight
6. foreach (Edge e : G):
   7.   Q.insert(e)
8. 
9. Graph T = (V, {})
10. 
11. while |T.edges()| < n-1:
12.   Vertex (u, v) = Q.removeMin()
13.   if forest.find(u) == forest.find(v):
14.     T.addEdge(u, v)
15.     forest.union( forest.find(u),
16.                   forest.find(v) )
17. 
18. return T

(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)
Kruskal's Algorithm

<table>
<thead>
<tr>
<th>Priority Queue:</th>
<th>Heap</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building</td>
<td>:7-9</td>
<td></td>
</tr>
<tr>
<td>Each removeMin</td>
<td>:13</td>
<td></td>
</tr>
</tbody>
</table>

```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G):
    forest.makeSet(v)

  PriorityQueue Q    // min edge weight
  foreach (Edge e : G):
    Q.insert(e)

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) == forest.find(v):
      T.addEdge(u, v)
      forest.union( forest.find(u), forest.find(v) )
  return T
```
Kruskal’s Algorithm

```
Priority Queue:    Total Running Time
Heap
Sorted Array

KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G):
    forest.makeSet(v)
  PriorityQueue Q    // min edge weight
  foreach (Edge e : G):
    Q.insert(e)
  Graph T = (V, {})
  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) == forest.find(v):
      T.addEdge(u, v)
      forest.union( forest.find(u),
                    forest.find(v) )
  return T

Priority Queue:
Total Running Time
Heap
Sorted Array

Heap
Sorted Array
```

Heap
Sorted Array