Jacob Karcz, 11.25.2016

Assignment 7, Written Part

1. How is the graph stored in the provided code? Is it represented as an adjacency matrix or as a list?

   The graphs in this program are stored as a list

2. Which of the 3 graphs (in main/prog) are connected? How can you tell?

   Graphs 2 and 3 because there is at least one path connecting all of the vertices in the graphs, you can start at any vertex and travel to any vertex in the graph. Graph 1 lacks the ability to visit any destination node from any arbitrary source node.

3. Imagine that we ran each depth-first and breadth-first searches in the other direction (from destination to source). Would the output change at all? Would the output change if the graphs were directed graphs?

   As long as the graph remains an undirected graph, there would be no change in the DFS and BFS results, however if the graphs were directed then the results would almost certainly change depending on the source and destination vertices chosen.

4. What are some pros and cons of DFS vs BFS? When would you want to use one over the other?

   DFS can be faster and consume less memory, however it can get stuck in an infinite path if it enters it, thus it may not always be successful in finding a path. Alternatively, a BFS would not get stuck in such infinite path because it radiates out from the origin moving one edge at a time, so it would reach the destination instead of getting stuck in an endless path, additionally, a BFS will always find a solution and it will return the shortest possible path (at the cost of speed and increased memory consumption).

   You would want to use BFS when the solution is close to the starting vertex or if solutions are rare and deep in the graph ADT, DFS is better for graphs where solutions are frequently (easily) encountered (even if they are deep). If solutions are rare and deep, then BFS will consume too much memory (while DFs will take up too much time. So, when timing is of no consequence but memory is at a premium, DFS is better (as long as no infinite paths exist), but when timing is the limiting factor and there is ample memory available, BFS would better serve the purpose.

5. What is the Big O execution time to determine if a vertex is reachable from another vertex?

   O(e+ v) execution time, where edges (e) or vertices (v) will dominate the algorithm depending on which one is greater.

Jacob Karcz, 11.25.2016

Assignment 7, Written Part

Tips

- Pay careful attention to the struct definitions. In particular the `Graph` struct contains an array of `Vertex` structs (all of the verices in the graph), while the `Vertex` struct contains an array of pointer to the `Vertex` structs that are neighbors of the vertex. These pointers point to the same vertices stored in the `Graph` struct's array.
- The edges in this graph are undirected, meaning that an edge from vertex1 to vertex2 and an edge from vertex2 to vertex1 are the same edge.
- A graph is "connected" if there is at least one path between every pair of vertices in the graph.
- If a `Vertex* vertex` had at least three neighbors, to access the third neighbor we would write `vertex->neighbors[2]`.
- You can check for memory leaks on flip by typing `make memcheckTests` or `make memcheckProg` on the command line for `tests` or `prog` respectively.

```
Graph 1 (graph1.txt)
---
Vertex count: 10
Edge count: 9
0 : 2 1
1 : 0 9 2
2 : 0 1 8
3 : 6
4 : 6
5 : 9
6 : 3 4 7
7 : 6
8 : 2
9 : 1 5

0 ... 0 : DFS path, BFS path
0 ... 1 : DFS path, BFS path
0 ... 2 : DFS path, BFS path
0 ... 3 : DFS no path, BFS no path
0 ... 4 : DFS no path, BFS no path
0 ... 5 : DFS path, BFS path
0 ... 6 : DFS no path, BFS no path
0 ... 7 : DFS no path, BFS no path
0 ... 8 : DFS path, BFS path
0 ... 9 : DFS path, BFS path
1 ... 1 : DFS path, BFS path
1 ... 2 : DFS path, BFS path
1 ... 3 : DFS no path, BFS no path
1 ... 4 : DFS no path, BFS no path
1 ... 5 : DFS path, BFS path
```

```
1 ... 6 : DFS no path, BFS no path
1 ... 7 : DFS no path, BFS no path
1 ... 8 : DFS path, BFS path
1 ... 9 : DFS path, BFS path
2 ... 2 : DFS path, BFS path
2 ... 3 : DFS no path, BFS no path
2 ... 4 : DFS no path, BFS no path
2 ... 5 : DFS path, BFS path
2 ... 6 : DFS no path, BFS no path
2 ... 7 : DFS no path, BFS no path
2 ... 8 : DFS path, BFS path
2 ... 9 : DFS path, BFS path
3 ... 3 : DFS path, BFS path
3 ... 4 : DFS path, BFS path
3 ... 5 : DFS no path, BFS no path
3 ... 6 : DFS path, BFS path
3 ... 7 : DFS path, BFS path
3 ... 8 : DFS no path, BFS no path
3 ... 9 : DFS no path, BFS no path
4 ... 4 : DFS path, BFS path
4 ... 5 : DFS no path, BFS no path
4 ... 6 : DFS path, BFS path
4 ... 7 : DFS path, BFS path
4 ... 8 : DFS no path, BFS no path
4 ... 9 : DFS no path, BFS no path
5 ... 5 : DFS path, BFS path
5 ... 6 : DFS no path, BFS no path
5 ... 7 : DFS no path, BFS no path
5 ... 8 : DFS path, BFS path
5 ... 9 : DFS path, BFS path
6 ... 6 : DFS path, BFS path
6 ... 7 : DFS path, BFS path
6 ... 8 : DFS no path, BFS no path
6 ... 9 : DFS no path, BFS no path
7 ... 7 : DFS path, BFS path
7 ... 8 : DFS no path, BFS no path
7 ... 9 : DFS no path, BFS no path
8 ... 8 : DFS path, BFS path
8 ... 9 : DFS path, BFS path
9 ... 9 : DFS path, BFS path

Graph 2 (graph2.txt)
---
Vertex count: 10
Edge count: 45
0 : 4 5 1 7 9 6 3 2 8
1 : 0 2 5 7 6 4 9 8 3
2 : 0 1 7 9 6 3 4 8 5
3 : 0 1 2 9 6 4 7 5 8
4 : 0 1 2 3 7 8 9 6 5
```

Jacob Karcz, 11.25.2016

Assignment 7, Written Part

```
5 : 0 1 2 3 4 9 7 6 8
6 : 0 1 2 3 4 5 9 8 7
7 : 0 1 2 3 4 5 6 9 8
8 : 0 1 2 3 4 5 6 7 9
9 : 0 1 2 3 4 5 6 7 8

0 ... 0 : DFS path, BFS path
0 ... 1 : DFS path, BFS path
0 ... 2 : DFS path, BFS path
0 ... 3 : DFS path, BFS path
0 ... 4 : DFS path, BFS path
0 ... 5 : DFS path, BFS path
0 ... 6 : DFS path, BFS path
0 ... 7 : DFS path, BFS path
0 ... 8 : DFS path, BFS path
0 ... 9 : DFS path, BFS path
1 ... 1 : DFS path, BFS path
1 ... 2 : DFS path, BFS path
1 ... 3 : DFS path, BFS path
1 ... 4 : DFS path, BFS path
1 ... 5 : DFS path, BFS path
1 ... 6 : DFS path, BFS path
1 ... 7 : DFS path, BFS path
1 ... 8 : DFS path, BFS path
1 ... 9 : DFS path, BFS path
2 ... 2 : DFS path, BFS path
2 ... 3 : DFS path, BFS path
2 ... 4 : DFS path, BFS path
2 ... 5 : DFS path, BFS path
2 ... 6 : DFS path, BFS path
2 ... 7 : DFS path, BFS path
2 ... 8 : DFS path, BFS path
2 ... 9 : DFS path, BFS path
3 ... 3 : DFS path, BFS path
3 ... 4 : DFS path, BFS path
3 ... 5 : DFS path, BFS path
3 ... 6 : DFS path, BFS path
3 ... 7 : DFS path, BFS path
3 ... 8 : DFS path, BFS path
3 ... 9 : DFS path, BFS path
4 ... 4 : DFS path, BFS path
4 ... 5 : DFS path, BFS path
4 ... 6 : DFS path, BFS path
4 ... 7 : DFS path, BFS path
4 ... 8 : DFS path, BFS path
4 ... 9 : DFS path, BFS path
5 ... 5 : DFS path, BFS path
5 ... 6 : DFS path, BFS path
5 ... 7 : DFS path, BFS path
5 ... 8 : DFS path, BFS path
```

```
5 ... 9 : DFS path, BFS path
6 ... 6 : DFS path, BFS path
6 ... 7 : DFS path, BFS path
6 ... 8 : DFS path, BFS path
6 ... 9 : DFS path, BFS path
7 ... 7 : DFS path, BFS path
7 ... 8 : DFS path, BFS path
7 ... 9 : DFS path, BFS path
8 ... 8 : DFS path, BFS path
8 ... 9 : DFS path, BFS path
9 ... 9 : DFS path, BFS path

Graph 3 (graph3.txt)
---
Vertex count: 10
Edge count: 9
0 : 7 8
1 : 5 2
2 : 1 3
3 : 2 8
4 : 5
5 : 1 4
6 : 8
7 : 0 9
8 : 0 3 6
9 : 7

0 ... 0 : DFS path, BFS path
0 ... 1 : DFS path, BFS path
0 ... 2 : DFS path, BFS path
0 ... 3 : DFS path, BFS path
0 ... 4 : DFS path, BFS path
0 ... 5 : DFS path, BFS path
0 ... 6 : DFS path, BFS path
0 ... 7 : DFS path, BFS path
0 ... 8 : DFS path, BFS path
0 ... 9 : DFS path, BFS path
1 ... 1 : DFS path, BFS path
1 ... 2 : DFS path, BFS path
1 ... 3 : DFS path, BFS path
1 ... 4 : DFS path, BFS path
1 ... 5 : DFS path, BFS path
1 ... 6 : DFS path, BFS path
1 ... 7 : DFS path, BFS path
1 ... 8 : DFS path, BFS path
1 ... 9 : DFS path, BFS path
2 ... 2 : DFS path, BFS path
2 ... 3 : DFS path, BFS path
2 ... 4 : DFS path, BFS path
2 ... 5 : DFS path, BFS path
```

```
2 ... 6 : DFS path, BFS path
2 ... 7 : DFS path, BFS path
2 ... 8 : DFS path, BFS path
2 ... 9 : DFS path, BFS path
3 ... 3 : DFS path, BFS path
3 ... 4 : DFS path, BFS path
3 ... 5 : DFS path, BFS path
3 ... 6 : DFS path, BFS path
3 ... 7 : DFS path, BFS path
3 ... 8 : DFS path, BFS path
3 ... 9 : DFS path, BFS path
4 ... 4 : DFS path, BFS path
4 ... 5 : DFS path, BFS path
4 ... 6 : DFS path, BFS path
4 ... 7 : DFS path, BFS path
4 ... 8 : DFS path, BFS path
4 ... 9 : DFS path, BFS path
5 ... 5 : DFS path, BFS path
5 ... 6 : DFS path, BFS path
5 ... 7 : DFS path, BFS path
5 ... 8 : DFS path, BFS path
5 ... 9 : DFS path, BFS path
6 ... 6 : DFS path, BFS path
6 ... 7 : DFS path, BFS path
6 ... 8 : DFS path, BFS path
6 ... 9 : DFS path, BFS path
7 ... 7 : DFS path, BFS path
7 ... 8 : DFS path, BFS path
7 ... 9 : DFS path, BFS path
8 ... 8 : DFS path, BFS path
8 ... 9 : DFS path, BFS path
9 ... 9 : DFS path, BFS path
==14578==
==14578== HEAP SUMMARY:
==14578==     in use at exit: 0 bytes in 0 blocks
==14578==   total heap usage: 4,935 allocs, 4,935 frees, 119,080 bytes
allocated
==14578==
==14578== All heap blocks were freed -- no leaks are possible
==14578==
==14578== For counts of detected and suppressed errors, rerun with: -v
==14578== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1 from 1)
```