Assignment 10: Graph DFS With Stacks/Iteration

CS3305/W01 Data Structures

Casey Hampson

November 12, 2024

Pseudocode/Algorithm

We can take inspiration from the small listing in the Checkpoint Question 28.17. The thing that is wrong in that listing is that there is the possibility that we can visit the starting vertex twice. This happens because we push the starting vertex v onto the stack, then we pop it back off, visit it again, and continue going. Instead, we should only visit vertices that we search out from given some vertex that we pop. So:

```
   input : Vertex v
1  push v onto the stack;
2  while the stack is not empty do
3      pop a vertex u from the stack;
4      for each neighber n of u do
5          if n has not been visited then
6              add n to the stack and search order array;
7              store w as the parent of n;
```

With this, we still push the first vertex onto the stack only to immediately pop it back off, but by only "visiting" the vertices during the iteration of the popped vertex's neighbors, we eliminate any double counting issue.

Program Output

```
12 vertices are searched in this DFS order:
    Seattle Denver Kansas City Boston New York
    Chicago Atlanta Miami Dallas Houston Los Angeles
    San Francisco
parent of Seattle is Chicago
parent of San Francisco is Los Angeles
parent of Los Angeles is Dallas
parent of Denver is Chicago
parent of Kansas City is Chicago
parent of Chicago is New York
parent of Boston is Chicago
parent of New York is Chicago
parent of Atlanta is New York
parent of Miami is Atlanta
parent of Dallas is Atlanta
parent of Houston is Atlanta
```

Source Code

```
// Name:        Casey Hampson
// Class:       CS3305/W01
// Term:        Fall2024
// Intructor:   Sharon Perry
// Assignment: 10-DFS


import java.util.List;
import java.util.ArrayList;
import java.util.Stack;


class UnweightedGraphNonrecursiveDFS<V> extends UnweightedGraph<V> {
        /** Construct a graph from vertices and edges stored in arrays */
        public UnweightedGraphNonrecursiveDFS(V[] vertices, int[][] edges) {
                super(vertices, edges);
        }

        @Override
        public Tree dfs(int v) {
                Stack<Integer> stack = new Stack<>();

                List<Integer> search_order = new ArrayList<>();
                boolean[] visited = new boolean[this.vertices.size()];
                int[] parents = new int[this.vertices.size()];

                // initialize the arrays
                for (int i=0; i<this.vertices.size(); i++) {
                        parents[i] = -1;
                        visited[i] = false;
                }

                // push first vertex v onto the stack
                // so that in the first iteration of the while loop,
                // it starts with v
                stack.push(v);

                while (!stack.isEmpty()) {
                        // peek at the next vertex on the stack
                        // don't want to get rid of it yet
                        // since not all of its neighbors have been visited
                        int u = stack.pop();

                        // grab the list of edges that are adjacent to u
                        for (Edge e : this.neighbors.get(u)) {
                                if (!visited[e.v]) {
                                        // push the unvisited vertex to the stack
                                        // and ensure its parent it set accordingly
                                        stack.push(e.v);
                                        search_order.add(e.v);
                                        parents[e.v] = u;
                                        visited[e.v] = true;
```

```
                        }
                    }
                }

                // after this is all done, we can just return a new tree
                return new Tree(v, parents, search_order);
        }
}
```

```
public class A10
{
        public static void main(String[] args)
        {
                // grab the list of cities and stuff that were used before
                String[] vertices = {"Seattle", "San Francisco", "Los Angeles",
                                                    "Denver", "Kansas City", "Chicago", "Boston",
                                                    "Atlanta", "Miami", "Dallas", "Houston"};
                int[][] edges = {
                        {0, 1}, {0, 3}, {0, 5},
                        {1, 0}, {1, 2}, {1, 3},
                        {2, 1}, {2, 3}, {2, 4}, {2, 10},
                        {3, 0}, {3, 1}, {3, 2}, {3, 4}, {3, 5},
                        {4, 2}, {4, 3}, {4, 5}, {4, 7}, {4, 8}, {4, 10},
                        {5, 0}, {5, 3}, {5, 4}, {5, 6}, {5, 7},
                        {6, 5}, {6, 7},
                        {7, 4}, {7, 5}, {7, 6}, {7, 8},
                        {8, 4}, {8, 7}, {8, 9}, {8, 10}, {8, 11},
                        {9, 8}, {9, 11},
                        {10, 2}, {10, 4}, {10, 8}, {10, 11},
                        {11, 8}, {11, 9}, {11, 10}
                };

                // create the graph and grab the tree
                AbstractGraph<String> graph = new UnweightedGraphNonrecursiveDFS<>(vertices, edges);
                AbstractGraph<String>.Tree dfs = graph.dfs(graph.getIndex("Chicago"));

                // run the same code used in listing 28.9, TestDFS.java
                // the output will be different since the search order will
                // be a little different
                List<Integer> search_orders = dfs.getSearchOrder();
                System.out.print(dfs.getNumberOfVerticesFound() +
                                                    " vertices are searched in this DFS order:\n\t");
                for (int i=0; i<search_orders.size(); i++) {
                        if ((i+1)%6 == 0) System.out.print("\n\t");
                        System.out.print(graph.getVertex(search_orders.get(i)) + " ");
                }
                System.out.println();

                for (int i=0; i<search_orders.size(); i++) {
                        if (dfs.getParent(i) != -1) {
```

```
                                System.out.println("parent of " + graph.getVertex(i) +
                                    " is " + graph.getVertex(dfs.getParent(i)));
                            }
                        }
                    }
                }
```