Aidan Kiser
8 November 2023
COMP-3270

Programming Assignment 2 Results & Analysis

BFS Results:

Source Node	Target Node	Distance (Edges)	Time (ns)
N_0	N_1	1	19375
N_0	N_2	2	22084
N_0	N_3	3	26125
N_0	N_4	8	68167
N_0	N_5	3	27500
N_0	N_6	2	23334
N_0	N_7	3	28375
N_0	N_8	6	53000
N_0	N_9	7	61417
N_0	N_10	4	31875
N_0	N_11	5	39584
N_0	N_12	4	33542
N_0	N_13	5	41667
N_0	N_14	6	53875
N_0	N_15	5	38334
N_0	N_16	6	51375
N_0	N_17	5	46167
N_0	N_18	6	55125
N_0	N_19	7	63667
N_0	N_20	6	49750
N_0	N_21	7	58709
N_0	N_22	6	56917
N_0	N_23	7	65417
N_0	N_24	8	73417

DFS Results:

Source Node	Target Node	Distance (Edges)	Time (ns)
N_0	N_1	1	2417
N_0	N_2	2	3292
N_0	N_3	3	4417
N_0	N_4	8	24709
N_0	N_5	3	6709
N_0	N_6	2	6084
N_0	N_7	3	19542
N_0	N_8	6	22667
N_0	N_9	7	23917

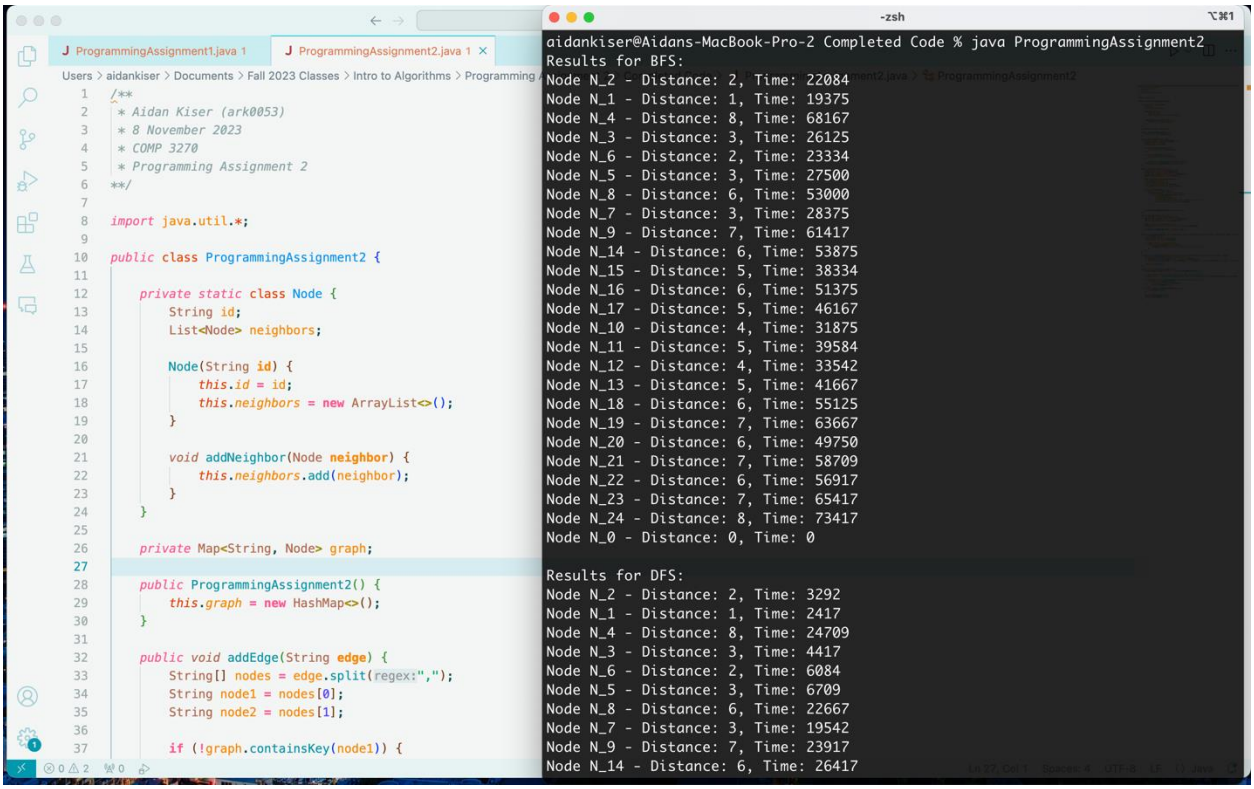
N_0	N_10	4	7625
N_0	N_11	5	12334
N_0	N_12	4	21250
N_0	N_13	5	22042
N_0	N_14	6	26417
N_0	N_15	5	8292
N_0	N_16	6	13292
N_0	N_17	5	32084
N_0	N_18	6	28250
N_0	N_19	7	28834
N_0	N_20	6	9375
N_0	N_21	7	10209
N_0	N_22	6	32709
N_0	N_23	7	36250
N_0	N_24	8	29667

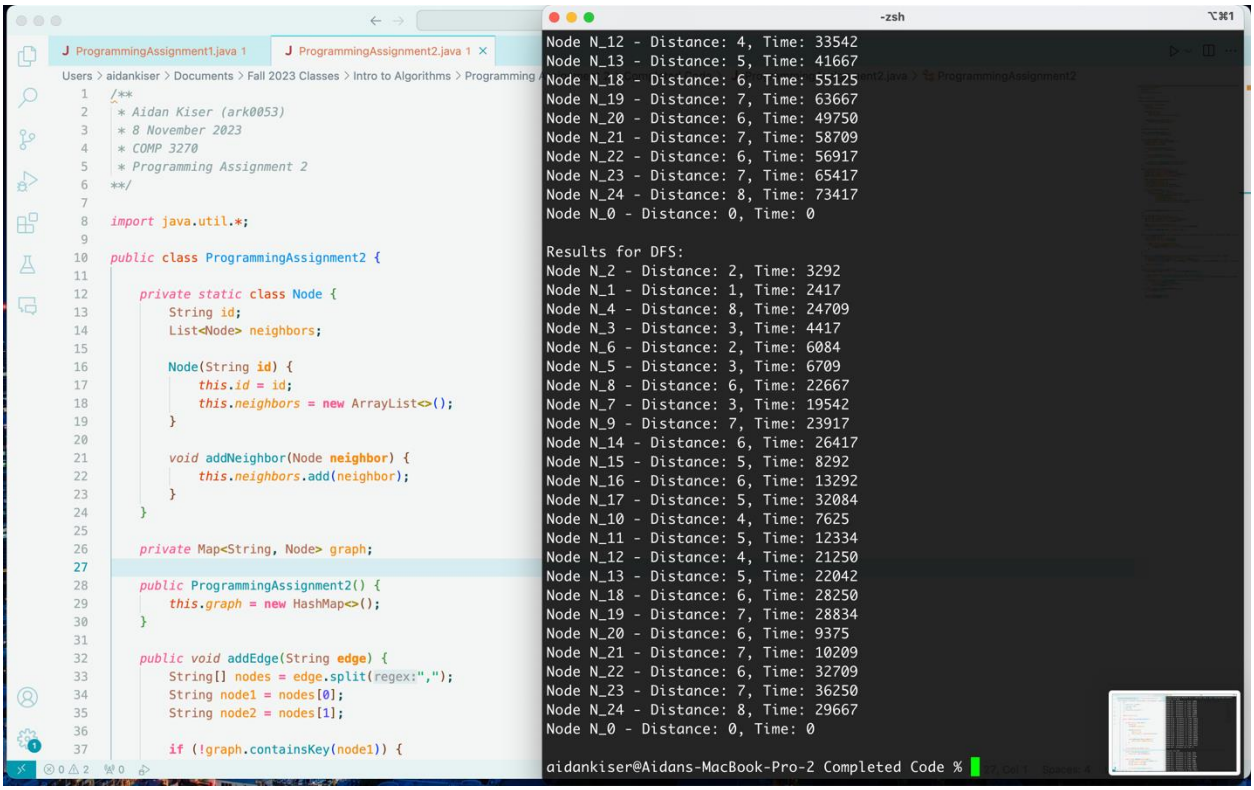
The data structure I chose to use for the graph is a Map. Specifically, the graph is stored as a Map<String, Node>, where each node has a unique identifier (String id) and a list of neighbors (List<Node> neighbors). In the adjacency list, each node only contains information about its neighbors, reducing the overall memory footprint. Traversing the graph is more efficient as well, as it directly provides information about a node's neighbors. The adjacency list representation strikes a balance between efficient memory usage and ease of traversal, making it suitable for scenarios where the graph is dynamic.

1. BFS explores the graph level by level, visiting all neighbors of the current node before moving on to their neighbors. This ensures that paths are explored in order of increasing distance from the starting node. Therefore, if the task is to find a path at a shallow depth, BFS is more suitable.

2. DFS explores as deeply as possible along one branch before backtracking. In the case where the end node is at a large depth, DFS might find the path more quickly than BFS because it goes deep into the graph before exploring other branches. This is advantageous when the goal is deep within the graph. However, it's important to note that DFS does not guarantee finding the shortest path, as it might find a path that goes deep before exploring alternative paths. If finding the shortest path is crucial, and memory usage is not a concern, BFS remains a better choice. It ensures that you find the shallowest path first due to its level-wise exploration strategy.

Below are screenshots of my code being run and the output:


ProgrammingAssignment1.java 1
ProgrammingAssignment2.java 1
Users > aidankiser > Documents > Fall 2023 Classes > Intro to Algorithms > Programming
/**
* Aidan Kiser (ark0053)
* 8 November 2023
* COMP 3270
* Programming Assignment 2
**/
import java.util.*;
public class ProgrammingAssignment2 {
private static class Node {
String id;
List<Node> neighbors;
Node(String id) {
this.id = id;
this.neighbors = new ArrayList<>();
}
void addNeighbor(Node neighbor) {
this.neighbors.add(neighbor);
}
}
private Map<String, Node> graph;
public ProgrammingAssignment2() {
this.graph = new HashMap<>();
}
public void addEdge(String edge) {
String[] nodes = edge.split(regex:",");
String node1 = nodes[0];
String node2 = nodes[1];
if (!graph.containsKey(node1)) {
-zsh
aidankiser@Aidans-MacBook-Pro-2 Completed Code % java ProgrammingAssignment2
Results for BFS:
Node N_2 - Distance: 2, Time: 22084
Node N_1 - Distance: 1, Time: 19375
Node N_4 - Distance: 8, Time: 68167
Node N_3 - Distance: 3, Time: 26125
Node N_6 - Distance: 2, Time: 23334
Node N_5 - Distance: 3, Time: 27500
Node N_8 - Distance: 6, Time: 53000
Node N_7 - Distance: 3, Time: 28375
Node N_9 - Distance: 7, Time: 61417
Node N_14 - Distance: 6, Time: 53875
Node N_15 - Distance: 5, Time: 38334
Node N_16 - Distance: 6, Time: 51375
Node N_17 - Distance: 5, Time: 46167
Node N_10 - Distance: 4, Time: 31875
Node N_11 - Distance: 5, Time: 39584
Node N_12 - Distance: 4, Time: 33542
Node N_13 - Distance: 5, Time: 41667
Node N_18 - Distance: 6, Time: 55125
Node N_19 - Distance: 7, Time: 63667
Node N_20 - Distance: 6, Time: 49750
Node N_21 - Distance: 7, Time: 58709
Node N_22 - Distance: 6, Time: 56917
Node N_23 - Distance: 7, Time: 65417
Node N_24 - Distance: 8, Time: 73417
Node N_0 - Distance: 0, Time: 0
Results for DFS:
Node N_2 - Distance: 2, Time: 3292
Node N_1 - Distance: 1, Time: 2417
Node N_4 - Distance: 8, Time: 24709
Node N_3 - Distance: 3, Time: 4417
Node N_6 - Distance: 2, Time: 6084
Node N_5 - Distance: 3, Time: 6709
Node N_8 - Distance: 6, Time: 22667
Node N_7 - Distance: 3, Time: 19542
Node N_9 - Distance: 7, Time: 23917
Node N_14 - Distance: 6, Time: 26417


ProgrammingAssignment1.java 1
ProgrammingAssignment2.java 1
Users > aidankiser > Documents > Fall 2023 Classes > Intro to Algorithms > Programming
/**
* Aidan Kiser (ark0053)
* 8 November 2023
* COMP 3270
* Programming Assignment 2
**/
import java.util.*;
public class ProgrammingAssignment2 {
private static class Node {
String id;
List<Node> neighbors;
Node(String id) {
this.id = id;
this.neighbors = new ArrayList<>();
}
void addNeighbor(Node neighbor) {
this.neighbors.add(neighbor);
}
}
private Map<String, Node> graph;
public ProgrammingAssignment2() {
this.graph = new HashMap<>();
}
public void addEdge(String edge) {
String[] nodes = edge.split(regex:",");
String node1 = nodes[0];
String node2 = nodes[1];
if (!graph.containsKey(node1)) {
-zsh
Node N_12 - Distance: 4, Time: 33542
Node N_13 - Distance: 5, Time: 41667
Node N_18 - Distance: 6, Time: 55125
Node N_19 - Distance: 7, Time: 63667
Node N_20 - Distance: 6, Time: 49750
Node N_21 - Distance: 7, Time: 58709
Node N_22 - Distance: 6, Time: 56917
Node N_23 - Distance: 7, Time: 65417
Node N_24 - Distance: 8, Time: 73417
Node N_0 - Distance: 0, Time: 0
Results for DFS:
Node N_2 - Distance: 2, Time: 3292
Node N_1 - Distance: 1, Time: 2417
Node N_4 - Distance: 8, Time: 24709
Node N_3 - Distance: 3, Time: 4417
Node N_6 - Distance: 2, Time: 6084
Node N_5 - Distance: 3, Time: 6709
Node N_8 - Distance: 6, Time: 22667
Node N_7 - Distance: 3, Time: 19542
Node N_9 - Distance: 7, Time: 23917
Node N_14 - Distance: 6, Time: 26417
Node N_15 - Distance: 5, Time: 8292
Node N_16 - Distance: 6, Time: 13292
Node N_17 - Distance: 5, Time: 32084
Node N_10 - Distance: 4, Time: 7625
Node N_11 - Distance: 5, Time: 12334
Node N_12 - Distance: 4, Time: 21250
Node N_13 - Distance: 5, Time: 22042
Node N_18 - Distance: 6, Time: 28250
Node N_19 - Distance: 7, Time: 28834
Node N_20 - Distance: 6, Time: 9375
Node N_21 - Distance: 7, Time: 10209
Node N_22 - Distance: 6, Time: 32709
Node N_23 - Distance: 7, Time: 36250
Node N_24 - Distance: 8, Time: 29667
Node N_0 - Distance: 0, Time: 0
aidankiser@Aidans-MacBook-Pro-2 Completed Code %