CS225 Final Project | Results

Team Members: Ilana Kaven, Adam Shore, Charles Kassmir, Emily Kaven

---

BFS

```
TEST_CASE("A test case BFS", "[case-1-BFS]") {  //correct size means every node was traversed and marked as visited
    AdjacencyMatrix AM("tests/TestData.txt");
    vector<bool> test = AM.bfs(0);
    for(size_t i= 0; i < 11; i++) {
        REQUIRE(test[i]);
    }

}

TEST_CASE("A test case BFSFB", "[case-1-BFS]") {  //correct size means every node was traversed and marked as visited
    AdjacencyMatrix AM("data/FacebookDataset.txt");

    vector<bool> test = AM.bfs(0);
    for(size_t i= 0; i < 11; i++) {
        REQUIRE(test[i]);
    }
}
```

For BFS, this algorithm had a goal to return all the nodes that were visited in the adjacency matrix(taken from the Facebook dataset) given a beginning start index. The algorithm works by adding nodes adjacent to a given node, and continuing the process by updating the node as visited and iterating through the entire dataset, marking the nodes as visited. While iterating through the dataset, if the node you are on is not yet visited, the bfs algorithm code works by adding the node to the queue and marking node as visited. We tested the bfs algorithm on the Facebook dataset as well as our own smaller dataset to ensure the effectiveness and proper results of the bfs algorithm. The test case checks the correct side of the boolean vector, ensuring that the node, each Facebook user, was traversed and marked as visited.

Dijkstra's Algorithm

```
77  TEST_CASE("Test dijkstras", "[case-3-data]") {  //vector has correct distance from source
78
79      vector<int> correct;
80      AdjacencyMatrix::Vertex src;
81      src.v = 0;
82
83      AdjacencyMatrix AM("tests/TestData.txt");
84
85      vector<int> output = AM.dijkstras(AM.getGraph(), src);
86
87      REQUIRE(output.at(0) == 0);
88      REQUIRE(output.at(1) == 1);
89      REQUIRE(output.at(2) == 1);
90      REQUIRE(output.at(3) == 1);
91      REQUIRE(output.at(4) == 1);
92      REQUIRE(output.at(5) == 2);
93      REQUIRE(output.at(6) == 3);
94
95
96  }
97
98  TEST_CASE("Test dijkstrasFB", "[case-3-data]") {  //vector has correct distance from source
99
00      vector<int> correct;
01      AdjacencyMatrix::Vertex src;
02      src.v = 0;
03
04      AdjacencyMatrix AM("data/FacebookDataset.txt");
05
06      vector<int> output = AM.dijkstras(AM.getGraph(), src);
07
08      REQUIRE(output.at(0) == 0);
09      REQUIRE(output.at(2) == 1);
10
11  }
```

Dijkstra's Algorithm takes in parameters of a graph and a source vertex. It then returns a vector of integers that represent the distance between the source vertex and the vertex stored at the corresponding index in the graph. In order to do this the algorithm utilizes a matrix to track which vertices have been visited. Using this, the algorithm loops through the indices and searches for the non-visited vertices and updates the minimum

distance and index for the closest vertex. This is then used to store the distance in the return vector. This is repeated until every index is marked as visited. To test our algorithm we made our own text file consisting of a data set that we made. We then used the adjacency matrix constructor to turn the file into a matrix. This was then used to test our algorithm. We created our own requirements for different vertices on the matrix in order to accurately assess if the algorithm works as intended. Additionally, we utilized the Facebook dataset in order to further test our algorithm. We successfully passed each of our custom test cases, indicating that our algorithm worked as intended.

Page Rank

```
113  TEST_CASE("Test pagerank", "[case-4-data]") {  //vector of ranks is created and most conected node is returned
114      vector<double> correct;
115      AdjacencyMatrix AM("tests/TestData.txt");
116
117
118
119      correct = AM.pageRank(100, 0.85);
120      double highest = 0;
121      double highest_idx = 0;
122
123      for (unsigned i = 0; i < correct.size(); i++) {
124          if (correct.at(i) > highest) {
125              highest = correct.at(i);
126              highest_idx = i;
127          }
128      }
129
130      REQUIRE(highest_idx == 0);
131
132
133  }
134
135  TEST_CASE("Test pagerank2", "[case-4-dataFB]") {  //vector of ranks is created and most connected node is returned
136      vector<double> correct;
137      AdjacencyMatrix AM("data/FacebookDataset.txt");
138
139
140
141      correct = AM.pageRank(100, 0.85);
142      double highest = 0;
143      double highest_idx = 0;
144
145      for (unsigned i = 0; i < correct.size(); i++) {
146          if (correct.at(i) > highest) {
147              highest = correct.at(i);
148              highest_idx = i;
149          }
150      }
151
152      REQUIRE(highest_idx == 1912);
153
154  }
```

For the page rank algorithm, the goal is to return a vector of each vertice's relative connectedness with each index corresponding to a given vertex. We did this by initializing our return vector with an even distribution of connectedness, each index is initialized to 1/N(total number of vertices). We use our adjacency matrix and the damping factor parameter to create an M_hat matrix. This matrix will be multiplied by

our return vector to update the return vector by rewarding the vertices with higher connectivity. Finally, we normalize the return vector, so it sums to one. This allows each element of the return vector to be a relative connectivity for each index corresponding vertex.

We test the page rank algorithm using our own text file and our chosen facebook-data file. We created our test file so that the vertex 0 is the most connected. We then run the page rank algorithm, and iterate through the return vector to make sure that the value at index 0 is the highest. We use a similar method for testing the facebook data, except we found the most connected node. This was node 1912, and we tested to make sure that index contained the highest element.

Answer to Leading Question

Our one leading question was to determine which nodes have direct(first degree), second degree, and higher connections to each other. Our Dijkstra's algorithm was used to answer this question. This algorithm takes in a source node and returns the degree of connections to this node. So, we were able to pass in a source node and determine the connections between all other nodes in the dataset. We also wanted to determine which node(Facebook user) is the most connected to any other users in the network in order to determine which user would likely be recommended as a friend to connect with on Facebook. We used our Page Rank algorithm to determine which node(Facebook) user is the most connected to all other users in the Facebook dataset.
