Luis Gascon, Ethan Webb, Femi Dosumu
COSC 336
May 7, 2023

Assignment 8

Exercise 1.

Describe in plain English (a short paragraph with at most 5-6 lines should be enough) an algorithm for the following task:

Input: A directed graph $G = (V, E)$, and a node $u \in V$.

Goal: Output 1 if there is a path from every $v \in G$ to u (so if u is reachable from any other node), and output 0 otherwise.

Your algorithm should have runtime $O(n + m)$. (Hint: Use an idea that we have seen for similar connectivity problems in directed graphs.)

Exercise 2.

We have seen that Dijkstra's algorithm can be implemented in two ways: Variant (a) uses an array to store the $dist[]$ values of the unknown nodes, and Variant (b) uses a MIN-HEAP to store these values.

(a) Suppose in your application $m \leq 3n$. Which variant gives a faster runtime? Justify your answer.

(b) Suppose in your application $m \geq n^2/3$. Which variant gives a faster runtime? Justify your answer.

Our algorithm starts off by initializing two integer arrays `dist` and `npath`.

```
int[] dist = new int[g.n];
int[] npath = new int[g.n];

for (int i = 0; i < g.n; i++) {
    dist[i] = Integer.MAX_VALUE;
    npath[i] = 0;
}
```

I've set the undiscovered nodes to have a distance of infinity, and in this case, the maximum integer value that Java supports.

The algorithm in its entirety.

```java
static void getShortestPath(Adj_List_Graph g, int s) {
  int[] dist = new int[g.n];
  int[] npath = new int[g.n];

  for (int i = 0; i < g.n; i++) {
      dist[i] = Integer.MAX_VALUE;
      npath[i] = 0;
  }

  Queue<Integer> queue = new LinkedList<Integer>();

  dist[s] = 0;
  npath[s] = 1;
  queue.add(s);
  while (!queue.isEmpty()) {
      int p = queue.poll();
      List<Integer> neighbors = g.adj.get(p);
      for (int v : neighbors) {
          if (dist[v] == Integer.MAX_VALUE) {
              dist[v] = dist[p] + 1;
              npath[v] = npath[p];
              queue.add(v);
          }

          else if (dist[v] == dist[p] + 1)
              npath[v] += npath[p];

      }
  }

  for (int index = 1; index < g.n; index++)
      System.out.printf("dist[%d] = %d \t npath[%d] = %d%n",
              index, dist[index], index, npath[index]);

}
```

Program output can be found on the next page.

```
G1 results:

dist[1] = 0    npath[1] = 1
dist[2] = 1    npath[2] = 1
dist[3] = 1    npath[3] = 1
dist[4] = 1    npath[4] = 1
dist[5] = 2    npath[5] = 2
dist[6] = 2    npath[6] = 1
dist[7] = 3    npath[7] = 3

G2 results:

dist[1] = 0    npath[1] = 1
dist[2] = 1    npath[2] = 1
dist[3] = 1    npath[3] = 1
dist[4] = 1    npath[4] = 1
dist[5] = 1    npath[5] = 1
dist[6] = 1    npath[6] = 1
dist[7] = 2    npath[7] = 5
dist[8] = 3    npath[8] = 5
dist[9] = 3    npath[9] = 5
```

To visually reflect one of our solutions, namely dist[7] and npath[7] of `G2`, we'll highlight the paths.