Weekly Assignment 5

Munia Humaira
Student ID: 21116435
Algorithm Design and Analysis

October 10, 2024

Solution 1:

Worst-case space-efficiency analysis:

At each recursive step x is halved: $x \leftarrow x/2$. The recursion terminates when $x = 0$.
So the depth of the recursion is $\mathcal{O}(log_2 x)$.
Each recursive call creates a new variable r of size $2y$. Each call occupies $\mathcal{O}(y)$ space. The total space required will be the product of the two:

$$\mathcal{O}(log_2 x \cdot y)$$

which is the worst-case space-efficiency of the algorithm.

Solution 2:

Using the hint from the question I will use BFS, which explores all neighbors level by level. We need to find possible paths from u to reach v.

The pseudocode is:

```
    function atLeastThree(G, u, v):
1   paths ← []
2   queue ← []
3   path ← [u]
4   ENQUEUE(queue, path.copy())
5   while queue not equal to [] do
6       path ← DEQUEUE(queue)
7       last ← path[length(path) - 1]
8       if last = v then
9           paths ← paths and {path}
10          if length(paths) = 3 then
11              return paths
12      for each i in range(length(G[last])) do
13          if G[last][i] not in path then do
14              curr_path ← path.copy()
15              APPEND(curr_path, G[last][i])
16              ENQUEUE(queue, curr_path)
17   return None
```

A brief explanation of the pseudocode:

Initialization (lines 1-4):

paths: A list to store paths that reach v. queue: A queue to store paths during BFS exploration. The initial path [u] is created and enqueued.

Main BFS Loop (lines 5-11):

The loop continues while the queue is not empty. In each iteration, the current path is dequeued, and the last node in the path is examined. If the last node is v, the path is added to the paths list. Once 3 paths are found, the function returns the paths.

Neighbor Exploration (lines 12-16):

The algorithm checks each neighbor of the last node. If the neighbor is not already in the current path (to prevent cycles), the algorithm creates a new path by extending the current path with the neighbor and enqueues this new path for future exploration.

Termination (line 17):

If fewer than 3 paths are found, the function returns None.

This BFS checks for 3 distinct paths by repeatedly applying the algorithm and avoiding already used edges to ensure the paths differ.