Problem 1: K-means Clustering

In this problem, we will do just one step of the K-means clustering algorithm. Suppose points A, C, and E are randomly selected to be the initial centroids.

1) What are the initial clusters given these initial centroids?
   A B
   C D G
   E F H

2) Draw + signs to indicate where the updated centroids (cluster means) are after one step of the k-means algorithm:
   A+B
   C
   E
   F+ H

3) What are the updated clusters based on these new centroids?
   A B C E
   D G
   F H
Problem 2: Breadth First Search

This function returns the shortest path from start to end, and None if no path exists.

Modify this function to return a list of ALL paths (without cycles) from start to end

def BFS(graph, start, end, pathQueue = []):
    initPath = [start]
    pathQueue.append(initPath)
    visited = [start]
    while len(pathQueue) != 0:
        # Get and remove oldest temporary path in pathQueue
        tmpPath = pathQueue.pop(0)
        lastNode = tmpPath[len(tmpPath) - 1]

        # The end node has been reached so return the path taken to get here
        if lastNode == end:
            return tmpPath

        # Iterate through the children to explore all paths one step deeper
        for linkNode in graph.childrenOf(lastNode):
            if linkNode not in visited:
                visited.append(linkNode)
                newPath = tmpPath + [linkNode]
                pathQueue.append(newPath)

    return None
Solution
def BFS_allpaths(graph, start, end, pathQueue = []):
    initPath = [start]
    pathQueue.append(initPath)
    visited = [start]
    allpaths = []
    while len(pathQueue) != 0:
        # Get and remove oldest temporary path in pathQueue
        tmpPath = pathQueue.pop(0)
        lastNode = tmpPath[len(tmpPath) - 1]

        # The end node has been reached so add the path taken to get here
        if lastNode == end:
            allpaths.append(tmpPath)

        # Iterate through the children to explore all paths one step deeper
        for linkNode in graph.childrenOf(lastNode):
            if linkNode not in visited:
                visited.append(linkNode)
                newPath = tmpPath + [linkNode]
                pathQueue.append(newPath)

    return allpaths
Problem 3: Depth First Search

The code below (presented in lecture) returns the shortest path between start and end, or None if no path exists between start and end.

Modify the code below so that it returns true if a path exists from start to end, and false if no path exists.

def DFSS(graph, start, end, path = [], shortest = None):
    path = path + [start]
    if start == end:
        return path
    for node in graph.childrenOf(start):
        if node not in path: #avoid cycles
            if shortest == None or len(path)<len(shortest):
                newPath = DFS(graph,node,end,path,shortest)
                if newPath != None:
                    shortest = newPath
    return shortest

Solution

def DFS_haspath(graph, start, end, path = []):
    path = path + [start]
    if start == end:
        return True
    path_exists = False
    for node in graph.childrenOf(start):
        if node not in path: #avoid cycles
            if DFS_haspath(graph,node,end,path):
                path_exists = True
    return path_exists