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?

2) Draw + signs to indicate where the updated centroids (cluster means) are after one step of the k-means algorithm.

3) What are the updated clusters based on these new centroids?

Problem 2: Breadth First Search

This function (presented in lecture) 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.
Dec. 5, 2014 Recitation 5
PSet 5 & Finger Exercise 5 due TONIGHT 11:59pm
Course Evaluations due Mon Dec 15 @ 9am (right before your final)

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

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 DFS(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
            pathQueue.append(newPath)
    return shortest