CS 471: Introduction to Artificial Intelligence
Assignment 1: Python (10 points)

Jacob Almon
9/12/2024

For this problem, you are only allowed to use standard python libraries. You may not use third party libraries or call any shell/bash functions.

You are given a list of tuples of the form (<float> x, <float> y, <float> r) (Let's call these c-tuples). Each c-tuple represents a circle on a rectangular coordinate space, with x and y being the coordinates of the center, and r being the radius. Assume that each c-tuple has a unique radius.

Let a cluster of circles be a group of circles where each circle in the group overlaps with at least one other circle in that group. A path is formed between two circles when they overlap. Define a cluster as a group of n circles, where each circle is reachable from every other circle through the formed paths.

Write a python script that does the following: Return True if the given circles form a cluster and return false if they don't form a cluster.

Below are some test cases.

Test case 1:

Input: [(1, 3, 0.7), (2, 3, 0.4), (3, 3, 0.9)]

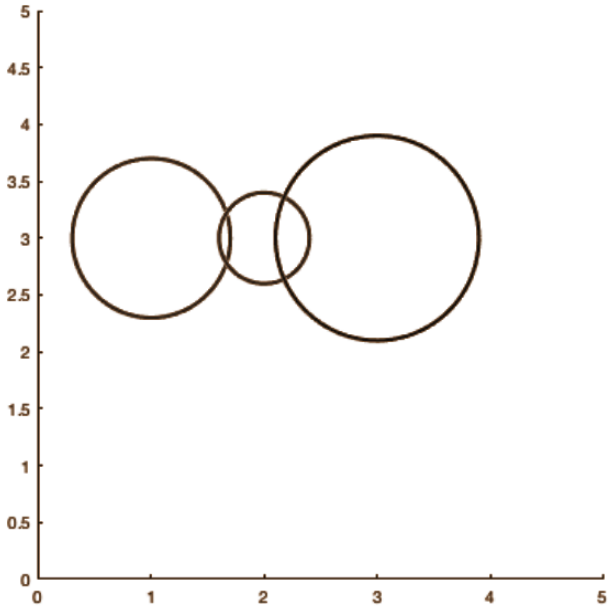

Figure 1: The three circles form the cluster. Output = True

Test case 2:

Input: [(1.5, 1.5, 1.3), (4, 4, 0.7)]

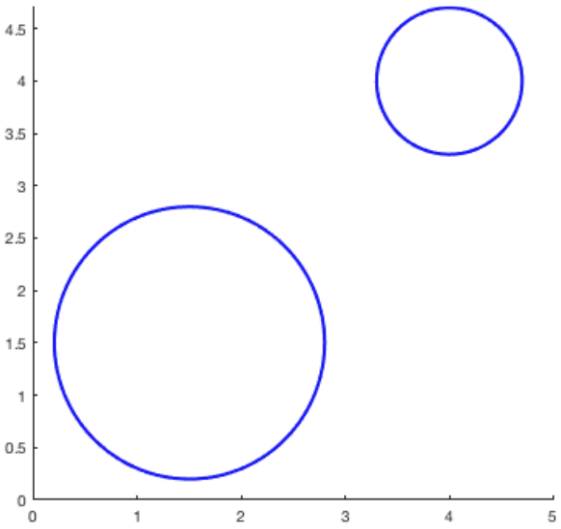

Figure 2: No clusters are found. Output = False

Test case 3:

Input: [(0.5, 0.5, 0.5), (1.5, 1.5, 1.1), (0.7, 0.7, 0.4), (4, 4, 0.7)]

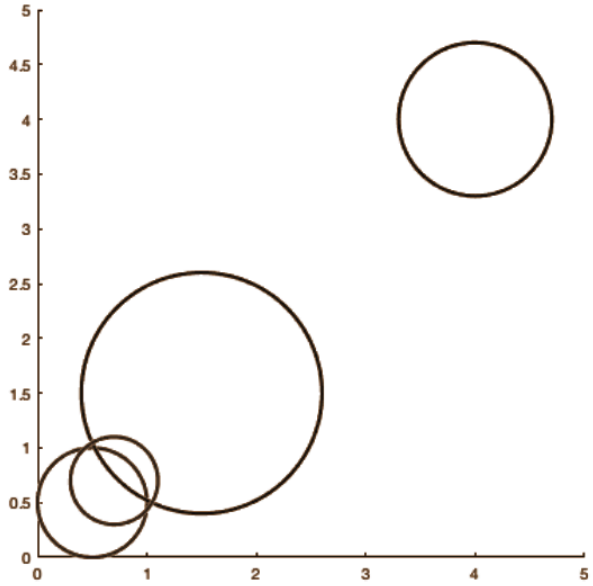

Figure 3: Given circles do not form a cluster. Output = False

Test case 4 (My Test Case):

Input: [(0.5, 0.5, 0.5), (0.5, 0.5, 0.8)]

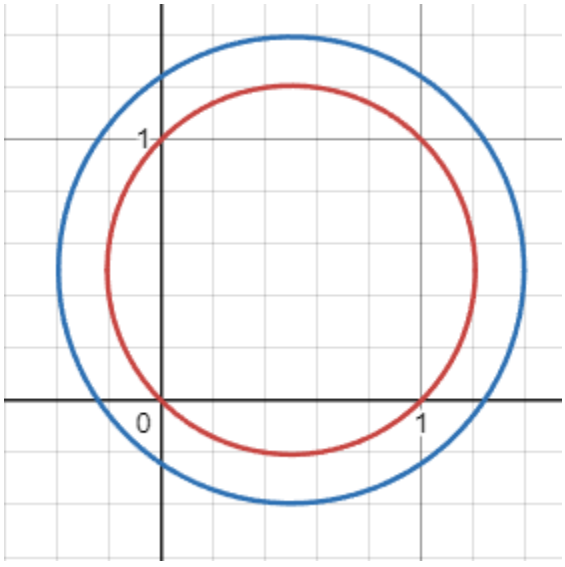

Figure 4: Given circles do form a cluster. Output = False

Along with the above three test cases, design a fourth test case of your choice. Share in detail the approach you used to solve the problem.

Hint: Sort the circles based on the radius and determine if the two adjacent circles intersect.

Grading rubric

Successful testing of test case 1: 2 points
Successful testing of test case 2: 2 points
Successful testing of test case 3: 2 points
Successful testing of test case 4 (designed by you): 2 points
Appropriate commenting and clear formatting of code: 2 points
---
Total score for this assignment: 10 points

Google Colab Link:
https://colab.research.google.com/drive/1u812Vip8GDdoy1hMDVhHCHzvH_klq5Pq?usp=sharing

GitHub Link (backup):
https://github.com/jacobalmon/CS-471/blob/main/Homework/Homework%201/cluster.py

To determine whether a collection of circles is clustered or not, we could perform a depth-first search (DFS) algorithm. But before we get into that, let's start with some basic edge cases:

- No Circles
    - If there are no circles, there are no clusters (False).
- Only 1 Circle
    - If there is 1 circle, it is a cluster of itself (True).

```
def isCluster(circles):
  # Get number of circles for later use.
  num_circles = len(circles)

# Edge Cases
  if num_circles == 0: # No Circles.
    return False
  elif num_circles == 1: # Only 1 Circle.
    return True
```

We now create an undirected graph of the cases where two circles are clustered. We can find this out by:

- Calculating the distance between each pair of circles using the distance formula: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- If the distance is less than or equal to the sum of both radii, then there is a cluster in those two circles. We can add edges to both vertices of the undirected graph.
- Otherwise, we check the next set of circles, until each case is run through.

```
# Creating Adjacency List (Graph) -> (dictionary of lists)
  adj_list = {}
  for i in range(num_circles):
    adj_list[i] = [] # initially empty.

  # Adds Overlaps into Adjacency List.
  for i in range(num_circles):
    for j in range(i + 1, num_circles):
      # Calculate the distance between circles i and j.
      x1, y1, r1 = circles[i]
      x2, y2, r2 = circles[j]
      distance = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
      # if the distance of said circles are less than or equal to the
      # sum of the radii, then there is a cluster of those two circles.
      if r1 + r2 >= distance >= abs(r1 - r2:
        adj_list[i].append(j)
        adj_list[j].append(i)
```

Now we have our undirected graph and we can perform a DFS algorithm. We can now check if all circles are clustered by seeing if every instance of the visited array is ‘True’. If so, then all circles have been visited in the graph, meaning they are all clustered (True). Otherwise, we return False.

```
def dfs(node, adj_list, visited):
  # Set current node to true, since we visited it.
  visited[node] = True
  # Loop through all adjacent nodes within our list.
  for adj_node in adj_list[node]:
    # if not visited, then we add it to the stack (recursive stack).
    # and perform dfs again.
    if not visited[adj_node]:
      dfs(adj_node, adj_list, visited)
```

…

```
# Perform DFS to check if all circles are clustering.
  visited = []
  for i in range(num_circles):
    visited.append(False)

  dfs(0, adj_list, visited)

  # every single instance of visited must be true for all circles to be
  # clustered.
  for i in visited:
    if i == False:
      return False

  return True
```

Test Case: 1

```
# Test Case 1.
  c_tuples1 = [(1, 3, 0.7), (2, 3, 0.4), (3, 3, 0.9)]
```

```
True
```

Test Case: 2

```
# Test Case 2.
  c_tuples2 = [(1.5, 1.5, 1.3), (4, 4, 0.7)]
```

```
False
```

Test Case: 3

```
# Test Case 3.
  c_tuples3 = [(0.5, 0.5, 0.5), (1.5, 1.5, 1.1), (0.7, 0.7, 0.4), (4, 4,
0.7)]
```

```
False
```

Test Case: 4

```
c_tuples4 = [(0.5, 0.5, 0.5), (0.5, 0.5, 0.8)]
```

```
False
```