CS101 Project 2

Atul Kharat

April 2024

1 Introduction

This project asks us to study the dataset which we made by one very fun activity during one of the tutorial classes. Everyone was asked to ask general questions to their peers, and note down the names of which they found the answer impressive. Every student in the class can have up to 30 impressions. We have to make a graph with this dataset, every student is a node and there exist a edge between them if someone found the other person impressive. We to have to make a directed graph with this data.
We have to find the Pagerank of this dataset that is we have find the most important node in this directed graph. We can achieve this by random walk method with teleportation.
Now, we have to find the missing links from the graph. Two nodes in the graph can be disconnected if none of them found each other impressive or there may be a case where they didn't meet, this are missing links. We have to find these connections.

2 Graph

We converted the existing excel file into a CSV file and deleted everything expect the entry numbers.
We read this CSV file by using pandas library, we first made a empty graph using the Networkx library. We added the primary nodes that is the first column of the file, which are the people who are taking the impression, Then we iterated through every row of the file, if a impression in a particular row is not empty, then we add a edge between the primary node and the impressive node. We get the following graph from this method.

3 Simulating Random Walk

Now, we define a function of random walk in the graph. First we make a dictionary and assign 0 to each node and as we visit the node, we increment the value by 1.

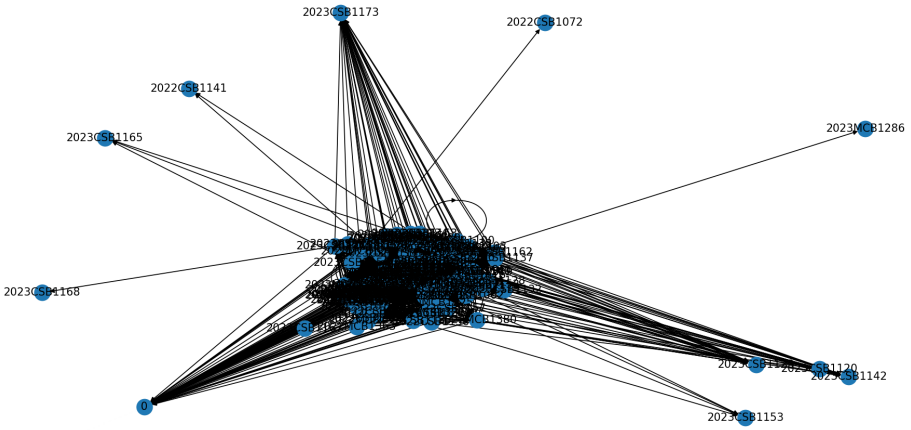

Figure 1: Graph

We randomly choose a node from the graph and then make a list of all the neighbours of the current node.
We now with a probability of 0.85 jump to a next node which is the neighbour of the current node or we move to a randomly selected node to avoid being stuck in a loop of nodes.
After running this operations many times, we rank the nodes according to the times we visited these nodes. By this we got the PageRank of the graph.

4 Missing links

We now have to find the missing links from the graph. First we will take all the nodes from the graph and make a adjacency matrix with those nodes.

A adjacency matrix is matrix representation of the connection between any two nodes of the graph.

If there exist a edge between two nodes, the corresponding entry in the matrix will set to 1 and 0 otherwise. We use Networkx library to make a adjacency matrix.

Now I found the entries which are 0 and also its transpose entry is 0. This connection may be a missing link.

Now, we delete the row and the column in which one of the entry lies and store it. This row and column contain every other entry excluding the missing link.

```python
def random_walk_simulation(graph, iterations=100000):
    #creating a dictionary to store the number of visits to each node
    node_visits = {node: 0 for node in graph.nodes}
    #starting from a random node

    current_node = random.choice(list(graph.nodes))
    #iterating over the number of iterations
    for i in range(iterations):
        #incrementing the visit count for the current node
        node_visits[current_node] += 1
        #getting the neighbors of the current node
        neighbors = list(graph.neighbors(current_node))
        #if there are neighbors then choosing a random neighbor
        if neighbors:
            #choosing a random neighbor
            p = random.uniform(0, 1)
            if p < 0.85:
                next_node = random.choice(neighbors)
            #if the next node is in the graph then moving to the next node
                current_node = next_node if next_node in graph.nodes else random.choice(list(graph.nodes))
        else:
            #if there are no neighbors then moving to a random node
            current_node = random.choice(list(graph.nodes))
    #sorting the nodes based on the number of visits
    sorted_visits = dict(sorted(node_visits.items(), key=lambda item: item[1], reverse=True))
    sorted_visits.pop('0', None)
    #returning the sorted visits
    return sorted_visits
```

Figure 2: Random Walk

Now, we use the inbuilt function in numpy library named lstsq to find the solution for the equation AX= R, where A is the adjacency matrix and R is the row matrix.

Now, we dot X with the column we stored to find the value of the missing link, it will give us the approximate value of the missing link.

We will have to set a value above which, we can say that it is a missing link otherwise they didn't find each other impressive. I had set this value as 0.5 in my code.

5 Cycle existance in a graph

In the section, I found if the graph has a cycle or not.
Firstly, I defined a function named cycle and kept a track of all visited nodes and the nodes existed in the path which I am traversing currently.

The visited set restrict us to repeat the dfs from already used node.
Now, I made a nested function for depth first search to find the cycles in the graph which takes any node as a argument.

In the next step, I see all the neighbours of the current node, if any neigh-

```python
def missing_link(graph):
    #getting the nodes of the graph
    n = sorted(graph.nodes())
    #setting k to the number of the nodes
    k = len(n)
    #making adjacency matrix of the graph
    m = nx.to_numpy_array(graph, n)
    #creating a list to store the missing links
    links=[]
    #iterating over every element of the adjacency matrix
    for i in range(len(n)):
        for j in range(len(n)):
    #if the element is not the diagonal element and the element is 0 and the element at the transpose of the element is 0
            if i!=j and m[i,j] == 0 and m[j,i] == 0:
                c = m.copy()
    #deleting the column which contain the element
                a1 = np.delete(c[i], j)
    #deleting the row which contain the element
                a2 = np.delete(c[:,j], i)
                c = np.delete(c,(i),axis = 0)
                c = np.delete(c,(j),axis = 1)
    #solving the linear equation, where the element is the linear combination of the other elements in column
                x = np.linalg.lstsq(c, a2, rcond = None)[0]
    #taking the dot product of the solution and the column
                k = np.dot(x, a1)
    #if the dot product is greater than 0.5 then adding the missing link to the list
                if k > 0.5:
                    links.append((n[i], n[j]))
    print("Missing links are:", links,"Number of missing links in the graph are:", len(links))
```

Figure 3: Missing link

```python
def cycle(graph):
    #visited keep track of visited vertices
    visited = set()
    #current keeps track of all the nodes in the current path, as soon as we find similar node in it, we have a cycle
    current = set()

    #defining a function to perform depth first search
    def dfs(node):
        #starting from a node, adding it to visited and current path
        visited.add(node)
        current.add(node)

        #now iterating over all the neighbours of the node
        for neighbour in graph.neighbors(node):
            #if the neighbour is in the current path, then we have a cycle
            if neighbour in current:
                return True
            #if the neighbour is not visited, recursively explore it
            elif neighbour not in visited:
                if dfs(neighbour):
                    return True

        #remove the node from the current path as we have explored all its neighbours and no cycle was found
        current.remove(node)
        return False

    #iterate over all vertices and start DFS from each unvisited vertex
    for node in graph.nodes():
        if node not in visited:
            if dfs(node):
                return True

    return False
```

Figure 4: Cycle existance

bour exist in the current set, they we have a cycle in the graph.

Otherwise recursively call the function on the neighbour till we find a cycle.

If we don't find a cycle from starting from a particular node, they we will remove it from the current set and return false for that particular node.

Now, we will call the same for every node till we find a cycle for a node. If we didn't found a cycle for any of the node, the parent function will return False.

6 Significance

Detecting cycles in a graph is significant for various reasons. If we need to check a given graph is a tree or not we will need to check that the graph has no cycles.

Checking if a graph has a cycle or not prevents us to get stuck in infinite loops.

Many graph algorithms, such as topological sorting, shortest path algorithms (like Dijkstra's), assume that the graph is acyclic. Detecting cycles allows algorithms to handle such cases appropriately or even modify the graph structure if necessary.