

**Dataset:** <https://snap.stanford.edu/data/ego-Facebook.html>

**Data:** This is a set of over 4000 users, and it represents the connections between accounts. In the .txt file, there is a number (representing a user) followed by another number (representing another user they follow). In my graph, vertices represent the users and outgoing edges represent a follow. It is directed and unweighted.

**Experiments and Results:** Some things I wanted to see for this data are descriptions on the number of connections, and the density of the dataset. To do this, I found the average number of neighbors for each vertex, the distribution of degrees (number of users someone is following) and the average distance between vertices (which translates to density/degrees of separation). I used a module for the graph struct and its associated types, and a different file to actually conduct the experiments.

After all was written and executed, the output came to be this:

```
Average number of neighbors: 21
There are 281713 pairs that are farther than 6 degrees of separation.
Average distance between Vertices: 4.33774423847196
Distribution of Degrees: [(0, 376), (1, 323), (2, 208), (3, 205), (4, 180), (5, 158), (6, 148), (7, 152), (8, 133), (9, 118), (11, 108), (10, 105), (14, 87), (13, 86), (16, 81), (12, 80), (15, 70), (19, 54), (18, 52), (20, 51), (17, 50), (24, 44), (22, 43), (21, 43), (27, 39), (23, 39), (28, 36), (29, 33), (31, 33), (26, 32), (25, 32), (33, 29), (30, 29), (37, 24), (32, 21), (36, 20), (45, 20), (34, 20), (42, 18), (44, 17), (35, 17), (47, 16), (38, 16), (48, 16), (41, 15), (39, 15), (60, 15), (54, 14), (40, 14), (58, 13), (69, 12), (52, 12), (53, 11), (49, 11), (66, 11), (61, 11), (43, 11), (46, 11), (55, 11), (51, 10), (56, 10), (63, 10), (62, 10), (65, 9), (59, 8), (85, 8), (64, 8), (76, 8), (74, 8), (78, 8), (77, 8), (73, 7), (90, 7), (57, 7), (98, 7), (50, 7), (75, 6), (67, 6), (89, 6), (82, 6), (72, 6), (68, 5), (71, 5), (70, 5), (92, 5), (93, 5), (118, 5), (87, 5), (86, 5), (100, 5), (80, 5), (99, 5), (133, 4), (79, 4), (83, 4), (91, 4), (108, 4), (104, 4), (123, 4), (105, 4), (119, 4), (109, 4), (117, 3), (115, 3), (164, 3), (111, 3), (94, 3), (124, 3), (125, 3), (147, 3), (81, 3), (97, 3), (126, 3), (132, 3), (114, 3), (121, 3), (143, 3), (163, 3), (110, 3), (88, 3), (122, 3), (183, 2), (130, 2), (185, 2), (84, 2), (152, 2), (162, 2), (160, 2), (95, 2), (137, 2), (140, 2), (112, 2), (106, 2), (149, 2), (138, 2), (136, 2), (161, 2), (131, 2), (129, 2), (102, 2), (179, 1), (187, 1), (170, 1), (146, 1), (748, 1), (215, 1), (107, 1), (150, 1), (153, 1), (778, 1), (101, 1), (165, 1), (542, 1), (225, 1), (120, 1), (144, 1), (156, 1), (189, 1), (174, 1), (139, 1), (159, 1), (148, 1), (113, 1), (1043, 1), (154, 1), (207, 1), (191, 1), (151, 1), (168, 1), (347, 1)]
```

The average number of neighbors(follows) is 21, the average distance between vertices is 4.3377, and the distribution of degrees follows a sort of decreasing power law. Most users in this dataset were actually following 0-10 people, and as the number of outedges increases, the frequency falls off rapidly. One thing I found interesting was that there were 281,713 pairs that had a degree of separation between them that was greater than 6. This goes against Stanley Milgram's theory, and could be because of the way this data was sampled.

In this Stanford lecture (<https://web.stanford.edu/class/cs124/lec/socialnetworks21.pdf>), it states that the average distance between two people on social networks was 3.47 intermediaries, or 4.74 hops. My result came very close to that. It also states that "Many nodes have a very small degree— one or two connections." This is also consistent with my findings.

**Code:** The very first thing I had to do was get the data from the .txt file into a usable format.

```
pub fn read_file(path: &str) -> Vec<(usize, usize)> {
    let mut result: Vec<(usize, usize)> = Vec::new();
    let file = File::open(path).expect("Could not open file");
    let buf_reader = std::io::BufReader::new(file).lines();
    for line in buf_reader {
        let line_str = line.expect("Error reading");
        let v: Vec<str> = line_str.trim().split(' ').collect();
        let x = v[0].parse::<usize>().unwrap();
```

This function takes the data from the .txt file, and reads it line by line converting each entry into a tuple.

These tuples are then put into a vector for later conversion.

After that, I created the necessary types and structs to convert the array of tuples into a graph.

The graph struct has 2 features: n, which is the number of nodes, and outedges, which is a list of adjacency lists. Each adjacency list represents a node, with each item in the list representing an outedge.

```
pub type Vertex = usize;
pub type Edge = (Vertex, Vertex);
pub type AdjacencyList = Vec<Outedge>;

#[derive(Debug, Copy, Clone)]
pub struct Outedge {
    pub vertex: Vertex
}

#[derive(Debug)]
pub struct Graph {
    pub n: usize,
    pub outedges: Vec<AdjacencyList>,
}

//implementing different functions for the Graph struct
impl Graph {
    //creating the directed graph, which is represented by number of vertices followed by an array
    //each array represents a vector and its outgoing edges
    pub fn create_directed(n:usize,edges:&Vec<Edge>) -> Graph {
        let mut outedges = vec![vec![];n];
        for (u, v) in edges {
            outedges[*u].push(Outedge{vertex: *v});
        }
        Graph{n,outedges}
    }
}
```

To find further information about the connections between people, I implemented functions to the graph struct that used the outedges. The function numneighbors uses the length of each list in a graph's outedges to find it's number of connections. This later assists the averagenumneighbors and degreedistribution functions. For averagenumneighbors, I just had to iterate through the vectors in my graph and sum them up, dividing them by n in the end. For the degreedistribution function, I used a hashmap to store the number of vertices that have a specific number of connections. It uses a key-value pair, where the key is the degree of a vertex and the value is the number of vertices with that degree. Once all values are iterated over, the hashmap is converted into a list of tuples, and is sorted high to low by the frequency of a degree.

In order to find the average distance, I had to use a BFS function to visit each node from a source and find each nodes respective distance. In my bfs function, I used VecDeque to create a queue, and added all of the neighbors of the source node into the queue. It visits each neighbor, adding them to the queue and the distance to the output along the way. The distance is first

initialized to `usize::MAX` to show that a neighbor has not been visited. Once it iterates through all neighbors, it returns a vector of distances, where the index is the vertex visited. Once I had that, I could find the average of an entire graph by iterating through all the vertices, summing up their distances, and dividing it by the number of pairs.

### Testing:

I tested the `createdirected`, `averagedistance`, and `degreedistribution` functions. This was the code and output:

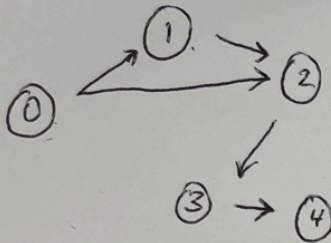
```
35  #[test]
36  fn test_createdirected() {
37      let testn = 3;
38      let testedges = vec![(0,1),(0, 2), (1, 2), (2, 3)];
39      let testgraph = graph::Graph::create_directed(testn,&testedges);
40      assert_eq!(testgraph.n,testn);
41      assert_eq!(testgraph.outedges[0],vec![Outedge { vertex: 1 }, Outedge { vertex: 2 }]);
42  }
43
44  #[test]
45  fn test_degreedistribution() {
46      let testedges = vec![(0, 1), (0, 2), (1,0),(1,2),(2,1)];
47      let testgraph = Graph::create_directed(3, &testedges);
48      let distribution = testgraph.degreedistribution();
49      assert_eq!(distribution.len(), 2);
50      assert_eq!(distribution[0], (2, 2));
51      assert_eq!(distribution[1], (1, 1));
52  }
53
54  #[test]
55  fn test_averagedistance() {
56      let edges = vec![(0, 1), (0, 2), (1, 2), (2, 3), (3, 4)];
57      let graph = Graph::create_directed(5, &edges);
58      let average_distance = graph.averagedistance();
59      let correct:f64 = (17 as f64/10 as f64);
60      assert_eq!(average_distance, correct);
61  }
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
running 3 tests
test testcreatedirected ... ok
test test_averagedistance ... ok
test test_degreedistribution ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

The first two were easy to test, but for the last one, to ensure it was correct, I found the result on paper:



The diagram shows a directed graph with 5 nodes labeled 0, 1, 2, 3, and 4. Node 0 has two outgoing edges: one to node 1 and one to node 2. Node 1 has one outgoing edge to node 2. Node 2 has one outgoing edge to node 3. Node 3 has one outgoing edge to node 4.

pairs	$v_0$	$v_1$	$D$
1	0	1	1
2	0	2	1
3	0	3	2
4	0	4	3
5	1	2	1
6	1	3	2
7	1	4	3
8	2	3	1
9	2	4	2
10	3	4	1

17  $\Rightarrow \frac{17}{10}$

#### Git Disclaimer:

While I used git add throughout my work on this project, I completely forgot to commit them. This is why there are only a couple commits from the same day.

#### Sources:

I used ChatGPT to debug the degreedistribution, bfs, and averagedistance functions, and I used lecture notes for the read file function, create directed function, and the bfs function.

<https://doc.rust-lang.org/std/collections/struct.VecDeque.html>

<https://doc.rust-lang.org/std/collections/struct.HashMap.html>