A, Overview:
The dataset I am analyzing is the Higgs Boson graph dataset from Stanford Snap. The Higgs Boson dataset describes who tweeted what at what time about the Higgs Boson discovery, essentially showing who found out first and who told who in the span of a week.

The questions I wanted to ask were:

- Who found out first and who found out last in the dataset? What time did they find out?
- What is the farthest distance from the first user? Aka using BFS to find the max distance. Essentially what I'm asking here is how many interactions did it take to get from one user to another? I'm going to do this for the first user
- Who tweeted the most about higgs boson?
- For each layer do people tend to talk and tweet about the particle more or less?

Link to dataset: https://snap.stanford.edu/data/higgs-twitter.html

B, Data Processing: The way I read my dataset was by reading the entire thing at once using the txt file given in the last link on the Stanford Snap website. I used the same method as the hw7 method for getting graphs by splitting by space and putting it all into parts. The only difference is that my dataset had two extra parts, timestamp and tweet type. It was also a gz file so I had to use gz decoder so that I could read in the graph. I didn't have to clean or transform the dataset.

C, Code Structure:

1. Modules
    - To put it simply, I put my read functions and all of my helper functions into mod and all of my big functions into main.
    - I put my read function there because it isn't something I have to look at after I make it.
    - Convert timestamp: this function also needs to import a bunch of stuff like chrono so I figured that would be better stored in projectutils.
    - Print timestamp: it is just a helper function for printing (since the first and last functions are essentially the same) so it is just for the side
    - Top ten influencers: Top ten influencers is just an extension of most tweets so if I put those two in main it would look a bit cluttered
    - Layer map count: This was just a function for me to see how many layers the dataset goes. It doesn't really help with any of the actual data analysis so I put it in projectutils.
2. Key Types:
    - Dataset: HashMap<usize, Vec<(usize, usize, String)>>
        - The key is the user id which is a number from one to a million
        - In the vec there is target which is the target id for connecting the graph, the timestamp, which is when that user tweeted, and interaction type such as MT, RT, or RE meaning mention, retweet, or reply.
        - Output: hashmap of the graph

- Layer Map: HashMap<usize, usize>
    - The first usize is the user id and the second is the layer number
- Tweet Map: HashMap<usize, u32>
    - The usize is the user id and the u32 is the number of interactions aka the number of outgoing edges
- Interactions per layer: HashMap<usize, usize>
    - The first usize is the layer number and the second is the average number of interactions

3. Key functions
    - Read higgs dataset (in mod)
        - The function takes in the txt file as a string. It then decodes the gz file with GZ decoder. In the loop which goes through each row I split by space and then there is 4 parts so I parse each part except for the string. Then I insert into my hashmap with part 0 being the key and part 1, part 2, and part 3 all being in a tuple to store in a vec.
        - Input: the txt gz file from stanford snap
        - Output: hashmap with the dataset
    - Find first timestamp
        - The goal of this function is to find who tweeted about Higgs Boson first and to do so I have to look at the timestamp. I set result to Max because I'm going to progressively work my way down as I go through the dataset. So to do that I do a nested loop so I can iterate through the timestamp and do a simple if this is less than the minimum make that the minimum type of function. I then return the minimum timestamp and the data associated with it such as who sent it and who it is being sent to.
        - Input: the dataset
        - Output: The source userid, the target userid, the unix time, the interaction type
    - Find last timestamp
        - Same thing as first timestamp it just finds the last timestamp so it sets a minimum and works it way up
        - Input: the dataset
        - Output: The source userid, the target userid, the unix time, the interaction type
    - Convert timestamp
        - I found through some googling that the timestamp data is in unix data and obviously that is very hard to understand. I used the chrono crate and using datetime functions I transformed it into normal year date, month, etc. Much of this code was from stack overflow but it is relatively simple.
        - Input: a unix timestamp
        - Output: A timestamp with the year, month, day, hour
    - Higgs BFS:
        - The goal of this function is to see the how far the layers go from that user with the first timestamp. My BFS function takes in the dataset and

establishes the distance for the start node as zero and the I put the start node at the back of the queue. So while the queue goes im searching for a target at each layer and if there is a target I add 1 to the distance integer and it keeps going until the queue is empty.
    - Input: dataset, start node
    - Output: The Max user, the max distance as an integer
- Most Tweets:
    - Most tweets essentially counts the outgoing edges for each user. Then when I go through the loop I just count up the vectors because each vector represents an outgoing edge in the hashmap. I then do a max value function that starts at 0 and works its way up. Surprisingly, one user had 656 interactions!
    - Input: the dataset
    - Output: Userid for the max user, the number of tweets for the max user
- Top ten influencers
    - This essentially uses the most tweets formula for counting up the vector but it just sorts the hashmap into the top 10 users. It does so by collecting the hashmap into a vector and then does .take(10) to output the first 10 users.
    - Input: The dataset
    - Output: The top 10 user ids and the top 10 tweets
- Find layers
    - The goal of this function is to find what layer a person is at so that I know where they are in the whole spreading information process. It establishes the layer map which is explained above and then it finds all receivers that are mentioned, who retweeted, or who replied as we can know they are at a different layer. We also know that anyone that is not a receiver started a conversation which means we can inert them into layer 0 of the layer map. Then I do bfs accounting for the layer each user is at.
    - Input: the dataset
    - Output: Userid and layer in hashmap format
- Layer map count
    - This is a helper function to help me visualize how many users are in each layer. This doesn't really help with the data analysis or anything it just helps with my process. It helped me find that a lot of people were starting the conversation, not continuing it. I am just inserting a hashmap counting up for each layer.
    - Input: dataset
    - Output: Hashmap of the layer and a vec of all the user ids
- Average interactions per layer
    - Input: the dataset, the layer map
    - Output: a hashmap of the layer for the key, the average as an integer
    - The purpose of this function was to see the frequency by which people had outgoing edges for each layer. The whole point of that is to see which

layer was most influential in the spread. To find this I had to create a hashmap called users in layer which found every user for each layer and put all the users into a vec. Essentially it is a consolidated layer map. Then for each user I use the most tweets method of counting up the individual vectors and I add that up to a total and then I divide the total by the number users for that layer. Then I add the layer and the average into a new hashmap.

D. Cargo Tests

running 4 tests
test tests::tests::test_first_timestamp ... ok
test tests::tests::test_most_tweets ... ok
test tests::tests::test_number_of_nodes ... ok
test tests::tests::test_average_interactions ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 4.66s

My checks:

- Test first timestamp: this checks if the first timestamp is the actual first timestamp. This is important because the first user is really important to my data analysis and what I want to understand in terms of the social dynamics.
- Test most tweets: this checks my most tweets function verifying the number of tweets and the user id. This makes sure I know who tweeted the most which helps me further understand the social dynamics.
- Test average interactions: This checks if average interactions matches with the averages that should be out there. I chose to test this because average interactions is the final touch on my data analysis.
- Test number of nodes: I wanted to check if I was fully reading in my part of the dataset. The internet said the specific file I was looking at had 304691 nodes so I made a test to verify that.

E. Results:

Find first timestamp:
For who found out first
The sender is 223789
The receiver is 213163
The timestamp is 2012-07-01 00:02:52.000000000
The type is MT

Find last timestamp:
For who found out last

The sender is 23996
The receiver is 253
The timestamp is 2012-07-07 23:59:53.000000000
The type is RT

Max distance (for the original node 223789):
The farthest node is 235700 which has distance 23
Most tweets
User 89805 has the most interactions with 656

Top ten influencers
Top ten: [(89805, 656), (26158, 199), (53508, 189), (181190, 185), (81405, 184), (64911, 178), (162402, 163), (38535, 162), (81181, 158), (9021, 156)]

Layer Map Count
Layer map: {6: 2, 5: 17, 0: 244714, 4: 142, 2: 6380, 3: 970, 1: 45680}

Average interactions per layer:
Interactions per layer: {0: 1, 1: 2, 2: 3, 3: 3, 4: 2, 5: 3, 6: 0}

F. Usage Instructions

- How to build and run
    - Cargo build
    - Cargo run
    - Output time: 0.05s
- No special command line arguments

G. AI Assistance

```rust
let mut target_vec = vec![];
target_vec.push(target, timestamp, interaction_type);
graph.entry(source).or_insert(target_vec);
```

This is wrong because:

- `.push()` takes a **single item**.
- You're passing **3 separate arguments** to `.push()`.
- Also, `.or_insert` only inserts if key is missing, but doesn't **append** if alread

What you should do is:

rust

```rust
graph.entry(source)
    .or_insert_with(Vec::new)
    .push((target, timestamp, interacti type.to_string()));
```

```rust
    if let Some(neighbors) = dataset.get(&v) {
        for (neighbor, _timestamp, _interaction) in neighbors {
            if !distance.contains_key(neighbor) {
                distance.insert(*neighbor, distance[&v] + 1);
                queue.push_back(*neighbor);
```

In the code:

rust Copy Edit

```rust
let mut receivers = std::collections::HashSet::new();

for vec in graph.values() {
    for (target, _timestamp, _interaction) in vec {
        receivers.insert(*target);
    }
}
```

🔥 How to fix it:

NEVER use `dataset[&user]` directly unless you are 100% sure the key exists.

Instead, use `.get(&user)` and check safely:

rust Copy Edit

```rust
if let Some(interactions) = dataset.get(&user) {
    total_interactions += interactions.len();
} else {
    // user has 0 interactions, do nothing
}
```

- I had trouble trying to enter a new vec into a hashmap so I needed help with the formatting. There are parentheses around the vec because that is correlated with the or_insert and I couldn't exactly figure that out at first. The push is just an extra add on function like a .map or something like that.
- I needed some help with completing the bfs so I asked chat for help. What the code is saying is that if there is a target for that node then you can add 1 to the distance because you know you can go another layer in the graph. Then it tacks that on to the hashmap.
- I was very stuck on find layers function so I asked chat for ideas and it gave me receivers. Basically receivers figures out if a user has an edge pointing at them and if it does it adds that to the hashset. The reason it uses a hashset is because my dataset is so big so combing through a vec would be really inefficient.

- I thought you could just access the user directly without an if let statement and then I also needed help with accessing the user. I forgot you can use the .get() statement which gets the value of a key and not accessing something array style.