Write Up:

I used a dataset that covers all anime titles, genre, and ratings and uses this information to analyze relationships between anime and see how influential these anime are. I made each vertex/node an anime title, and each edge is created between anime that share at least one genre. If further developed it could recommend anime based on anyone you choose. After setting up the graph, the algorithm figures out how many connections each anime has. It tells us how many other anime each anime is directly linked to through shared genres. This number helps us see which animes are the most important in the network. These are influential because having more connections might mean an anime is more popular or plays a key role within its genre groups. I also limited the amount of anime(vertices) to 1000 random anime as this dataset has over 180000 different anime and finding a common genre for each anime will take a very long time to do.

Help used:

I needed some help with creating a graph using just names and genres, as well as help reading csv files so I asked chatGPT for some assistance. I also used chatGPT to help with tests since I wasn't really sure on how to go about testing my code.

You

I have a csv file filled with different shows names, genres, and scores. How do I use this information to make a graph analysis algorithm making edges and such in rust

‹ 3 / 3 ›

ChatGPT

To create a graph analysis algorithm in Rust using the information from your CSV file containing show names, genres, and scores, you'll need to follow these general steps:

1. **Parse the CSV file**: You need to read the CSV file and extract the relevant information, such as show names, genres, and scores. You can use libraries like `csv`, `serde_csv`, or `rust-csv` to parse the CSV file.
2. **Create Nodes**: Each show in your CSV file will correspond to a node in your graph. You need to create nodes for each show, associating them with their respective attributes (name, genre, score).
3. **Create Edges**: Define the relationships between shows based on certain criteria. For example, you could create edges between shows that share the same genre or have similar scores.
4. **Construct the Graph**: Using a graph data structure, such as an adjacency list or adjacency matrix, you'll create the graph representation.
5. **Implement Graph Algorithms**: Once you have the graph constructed, you can implement various graph algorithms depending on your analysis needs, such as breadth-first search, depth-first search, shortest path algorithms, or clustering algorithms.

Here's a basic outline of how you can start implementing this in Rust:

First, add the dependencies to your `Cargo.toml` file:

toml

```toml
[dependencies]
serde = { version = "1.0", features = ["derive"] }
serde_csv = "1.1"
petgraph = "0.5"
```

Now, let's rewrite the code:

rust

```rust
use serde::Deserialize;
use std::error::Error;
use std::fs::File;
use petgraph::graph::{Graph, NodeIndex};
use petgraph::algo::connected_components;

#[derive(Debug, Deserialize)]
struct Show {
    name: String,
    genre: String,
    score: f64,
}

fn main() -> Result<(), Box<dyn Error>> {
    // Open CSV file
    let file = File::open("shows.csv")?;
    let mut rdr = csv::Reader::from_reader(file);

    // Create a graph
    let mut graph = Graph::new();
    let mut index_map = std::collections::HashMap::new();

    // Deserialize CSV records into Show structs and create nodes
    for result in rdr.deserialize() {
        let record: Show = result?;
        let node_index = graph.add_node(record.clone());
        index_map.insert(record.name, node_index);
    }

    // Create edges based on some criteria (e.g., shared genre)
    for (name, node_index) in &index_map {
        let show = graph[*node_index];
        for (_, other_index) in &index_map {
            let other_show = graph[*other_index];
            if show.genre == other_show.genre && name != &other_show.name {
                graph.add_edge(*node_index, *other_index, ());
            }
        }
    }

    // Example graph algorithm: Connected Components
    let components = connected_components(&graph);

    // Example output
    for (name, node_index) in index_map {
        let component_id = components[node_index.index()];
        println!("Show: {}, Component ID: {}", name, component_id);
    }

    Ok(())
}
```

In this version:

- We define a `Show` struct and derive the `Deserialize` trait from `serde` for automatic deserialization of CSV records into `Show` structs.

You

How could I use a shows rating to calculate what the most influential shows are based on genres

‹ 3 / 3 ›

ChatGPT

To determine the most influential shows based on genres using show ratings, you could employ a graph-based approach where shows are nodes and edges represent influence between shows of the same genre. Here's a general outline of how you could implement this:

1. **Construct the Graph**: Each show will be a node in the graph, and edges between shows will represent influence. The weight of each edge could be based on factors like similarity of ratings, number of shared viewers, or critical acclaim.
2. **Calculate Influence Scores**: Use a graph algorithm to calculate influence scores for each show based on its connections within the genre-specific subgraphs. Algorithms like PageRank or HITS (Hypertext Induced Topic Search) could be adapted for this purpose.
3. **Identify Most Influential Shows**: Once you have influence scores for each show, you can identify the most influential shows within each genre by sorting them based on their influence scores.

Here's a Rust code outline to get you started:

Here's a Rust code outline to get you started:

rust

Copy code

```rust
use petgraph::graph::{Graph, NodeIndex};
use petgraph::algo::pagerank;
use petgraph::visit::NodeFiltered;

// Define a struct to represent a show
#[derive(Debug, Clone)]
struct Show {
    name: String,
    genre: String,
    rating: f64,
}

fn main() {
    // Create a graph
    let mut graph = Graph::new();

    // Add shows as nodes to the graph
    let show1 = Show { name: "Show1".to_string(), genre: "Drama".to_string(), ra
    let show2 = Show { name: "Show2".to_string(), genre: "Drama".to_string(), ra
    let show3 = Show { name: "Show3".to_string(), genre: "Comedy".to_string(), r
    let show4 = Show { name: "Show4".to_string(), genre: "Comedy".to_string(), r

    let node1 = graph.add_node(show1.clone());
    let node2 = graph.add_node(show2.clone());
    let node3 = graph.add_node(show3.clone());
    let node4 = graph.add_node(show4.clone());

    // Add edges between shows of the same genre (e.g., based on shared viewersh
    graph.add_edge(node1, node2, ());
    graph.add_edge(node3, node4, ());

    // Calculate influence scores using PageRank algorithm
    let influence_scores = pagerank(&graph, 0.85, 0.001);

    // Filter nodes by genre and sort them based on influence scores
    let drama_nodes = NodeFiltered(&graph, |node| graph[node].genre == "Drama");
    let mut influential_drama_shows: Vec<_> = drama_nodes.map(|node| (node, infl
    influential_drama_shows.sort_by(|(_, score1), (_, score2)| score2.partial_cm

    // Print the most influential drama shows
    println!("Most influential Drama shows:");
    for (node, score) in influential_drama_shows {
        println!("{} - Influence Score: {}", graph[node].name, score);
    }
}
```

This code snippet is a simplified example. In a real-world scenario, you would read show data from a CSV file, calculate influence scores for each genre, and handle errors and edge cases appropriately. Additionally, you may need to fine-tune the graph construction and influence calculation based on your specific requirements and data characteristics.