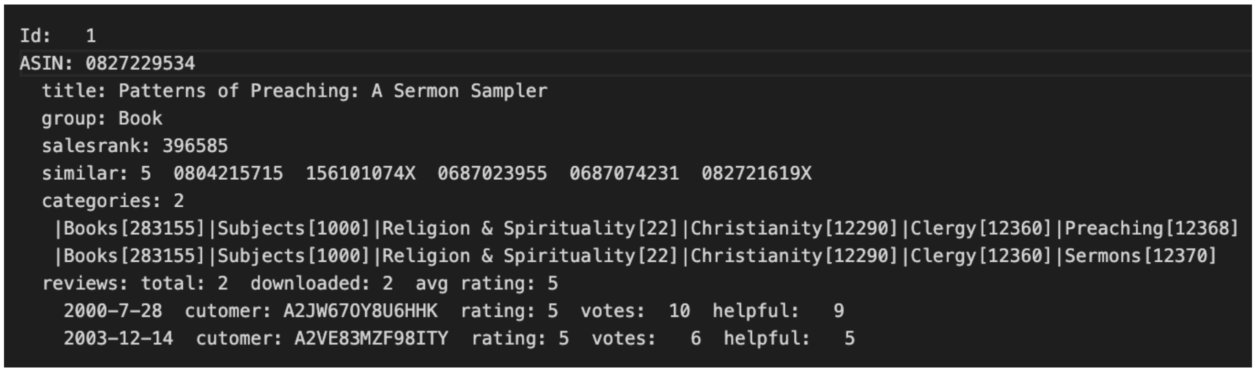
Amazon Similar Products

Rust Graph Analysis by Anmar Abdi for DS210

Dataset: https://snap.stanford.edu/data/amazon-meta.html

I would always be on amazon and wonder how they would come up with the products in the section “Customers Also Bought…” So for my final project I wanted to use the above data set of approximately 500k books from amazon to understand the connections between similar purchases.

The data set contains a bunch of data on each book in the following format:



```
Id:   1
ASIN: 0827229534
  title: Patterns of Preaching: A Sermon Sampler
  group: Book
  salesrank: 396585
  similar: 5  0804215715  156101074X  0687023955  0687074231  082721619X
  categories: 2
   |Books[283155]|Subjects[1000]|Religion & Spirituality[22]|Christianity[12290]|Clergy[12360]|Preaching[12368]
   |Books[283155]|Subjects[1000]|Religion & Spirituality[22]|Christianity[12290]|Clergy[12360]|Sermons[12370]
  reviews: total: 2  downloaded: 2  avg rating: 5
    2000-7-28  cutomer: A2JW67OY8U6HHK  rating: 5  votes:  10  helpful:   9
    2003-12-14  cutomer: A2VE83MZF98ITY  rating: 5  votes:   6  helpful:   5
```

For my graph building and analysis purposes I only needed the “Id:”, “ASIN:”, “title:”, and “similar:” So I first created a public Struct called Product in my parser.rs file in order to store the data. Then I created a vector to store all these products after being parsed. I then created a function to parse each line of the data to populate the struct and products vector. Everytime a new product ID is encountered, the previous ‘current_product’ is added to the products vector and a new ‘Product’ is initialized. The function then extracts the ASIN, book title, and similar products. I then had to ensure that the last product was added to the vector since there's no “ID” after it. It then returns the vector of parsed Products.

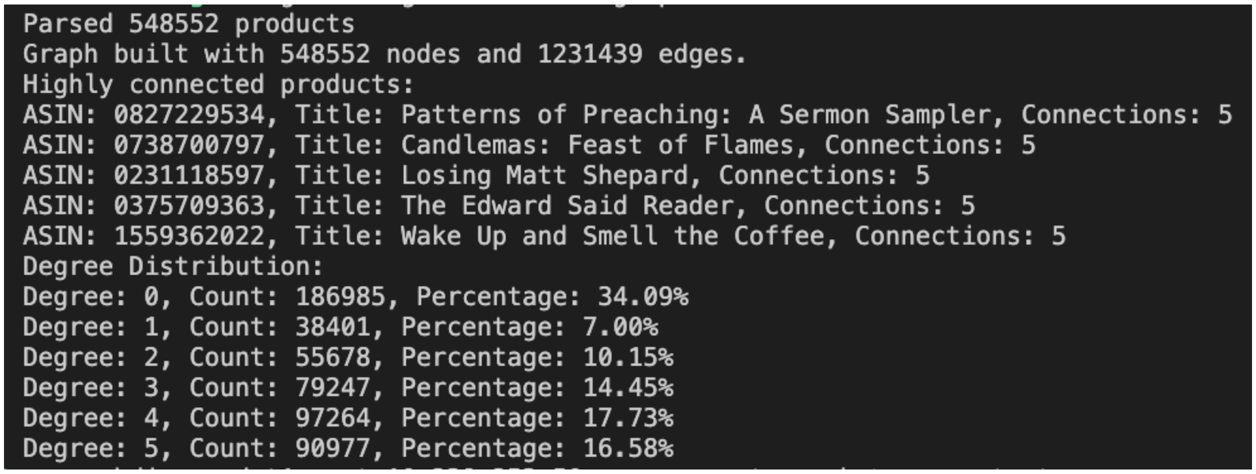
In graph.rs, the function build_graph creates a directed graph where each node is a book(Title and ASIN), and edges indicate similarity relationships. I used the petgraph crate because chatGPT said it would make graph building easier and I looked into it more and it was right. To make locating nodes more efficient I implemented mapping to relate each book’s ASIN to its corresponding node index in the graph.

I also include my graph analysis in this file. First I created a function to find highly connected nodes. It iterates through all the nodes and calculates the number of connections/or degrees for each node. It then sorts them by number of degrees in descending order to find the most connected books then returns the top 5.

The max number of nodes was 5 and there was a big number of books with this amount so I decided to analyze the degree distribution in the next function. This function counts the number of nodes with each degree then calculates what percentage of the total nodes each degree count represents.

In my main.rs file I have my main function that calls all the functions with the different modules. I also include some error handling to identify if the parsing fails. It first parses the data, builds the graph, then calls the 2 functions for analyzing the graph.

Output:



```
Parsed 548552 products
Graph built with 548552 nodes and 1231439 edges.
Highly connected products:
ASIN: 0827229534, Title: Patterns of Preaching: A Sermon Sampler, Connections: 5
ASIN: 0738700797, Title: Candlemas: Feast of Flames, Connections: 5
ASIN: 0231118597, Title: Losing Matt Shepard, Connections: 5
ASIN: 0375709363, Title: The Edward Said Reader, Connections: 5
ASIN: 1559362022, Title: Wake Up and Smell the Coffee, Connections: 5
Degree Distribution:
Degree: 0, Count: 186985, Percentage: 34.09%
Degree: 1, Count: 38401, Percentage: 7.00%
Degree: 2, Count: 55678, Percentage: 10.15%
Degree: 3, Count: 79247, Percentage: 14.45%
Degree: 4, Count: 97264, Percentage: 17.73%
Degree: 5, Count: 90977, Percentage: 16.58%
```

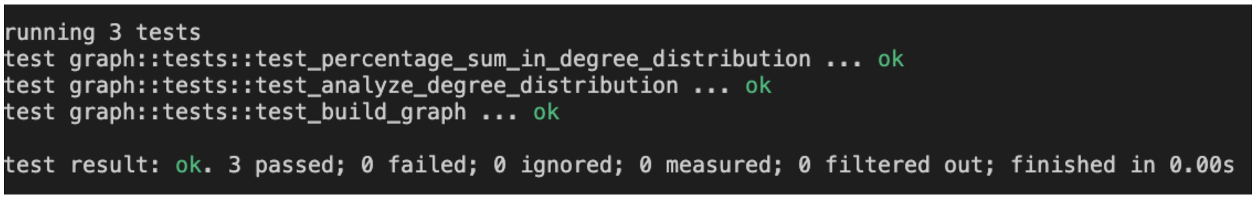
Tests

I made 3 tests for this program.

The first tests to see if the graph is built correctly with the expected number of nodes and edges by creating a small set of Product instances, building the graph, then checking if the number of nodes and edges matches the expected value.

The second tests to check if the analyze_degree_distribution function is working by seeing if it can correctly identify the degree of a particular node in the graph. This is to verify the correctness of individual degree calculations.

The third test is designed to check the same function but to see if the sum of all the degree percentages is approximately 100%. Since there can be some issues with precision I used the float-cmp dependency to check if it equals 100% with a small tolerance level.



```
running 3 tests
test graph::tests::test_percentage_sum_in_degree_distribution ... ok
test graph::tests::test_analyze_degree_distribution ... ok
test graph::tests::test_build_graph ... ok

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