Product recommendation System

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

P.Yashwenth Kumar | AP23110010511

P.Sudarshan | AP23110010531

Ismael | AP23110010569

A.Viswaj | AP23110010564

Under the Guidance of

Mrs.Kavitha Rani Karnena

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

November,2024

Certificate

Date: 20/11/2024

This is to certify that the work present in this Project entitled

"**Product Recommendation System**" has been carried out by **[Name of the Candidate]** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. / Dr. [Name]

Designation,

Affiliation.

Co-supervisor

(Signature)

Prof. / Dr. [Name]

Designation,

Affiliation.

Acknowledgements

There are specific people and references to which the following work is dedicated. They contributed to its creation in one way or another.

Graph Theory and Algorithms:

The project lays on the basics of graphs beginning with defining its main components to using simple graph traversal algorithms like Breadth-First Search (BFS) for use in the projects.

Thanks to sources of theory, as well as tutorials on graph data structures and their practical aspects.

Product Recommendation Systems:

The idea of these recommendation systems comes from existing e-commerce sites such as Amazone and Flipkart where a graph linking users with products exists.

Standard C++ Libraries:

Utilized standard C++ containers such as unordered_map, vector, and priority_queue to provide an effective and modular solution.

Problem-Solving Communities:

To such resources belong GeeksforGeeks, Stack Overflow, and Cplusplus.com which played crucial parts in debugging, editing and optimization of the code.

Open Source Spirit:

The design of the program emphasized modularity as well as extendability due to the core principle of the open-source community; reusability and teamwork.

Development Tools:

Great appreciation to visual studio code and other debugging tools which made the coding process easy and smooth.

Table of Contents

Abstract

The Product Recommendation System Using Graphs is the C + + oriented software application that advises its users on what products to buy depending on their activity and also employs graph-based data and

equitable solutions. The proposition embodies the concept of a customer and a product as nodes in a graph, while any purchase or average raking rises as a graph edge. Owing to the BFS (breadth-first search) approach and the incorporation of weighted edges, the adequate product recommendation system software can determine products that individual customers require.

Also, the system features complete product management facilities and users can create, edit, seek, and delete any product. It is also complemented with a user friendly menu that eases the operation of the system. Targeted at supporting expansion and development of the project, the project illustrates the application of graph theory in solving practical problems in recommendation engines as we find in online shopping and other platforms. This project serves as a groundwork for more complex systems operating such as collaborative filtering or ML-based recommendation system for better targeting.

Statement of Contributions

1. Yashwenth :

-Overall design and structure was fully conceptualized by him for the entire project.

-Graph-based recommendation engines employing Breadth-First Search (BFS) and weighted edges were developed.

-Core functionalities of graph creation, user-product involvement and recommendation logic were put in place.

-Did some debugging and code optimization as regards the functions to enhance the performance of the code.

2. Sudarshan:

-Created and deployed the **Product Management System**, with modules for adding, editing, deleting and displaying products.

-Developed input validation functions to improve user experience and avoid erroneous acts with the system.

-Performed thorough tests on the features of the product catalogue and ensured that the product worked well under numerous stress situations.

3. Ismael:

-Participated in the user interface design activities including the task of developing a menu driven system to allow easy interaction.

-Assisted in the integration of the product catalogue with the graph structure to render data more manageable.

-Organized people's feedback loops for features enhancement and usability improvement.

4. Vishwaj:

-Graph structure debugging feature was developed and incorporated into the system to allow analysis and validation of the structure.

-Wrote full documentation

Abbreviations

OOP: Object-Oriented Programming

C++: C Plus Plus

UI: User Interface

STD: Standard
PRS : Product Recommendation System
BFS : Breadth-First Search
UI : User Interface
CRUD : Create, Read, Update, Delete
PQ : Priority Queue
ID : Identification (e.g., User ID, Product ID)
IDE : Integrated Development Environment
DB : Database
UX : User Experience

Introduction

Welcome to our Product Recommendation System project! We're on a mission to make every shopping experience feel personalized and special.

Our Goals:

- **Graph-Based Data Model**: We use a graph to show relationships between users and products.
- **Core Functions**: Add, remove, and update products, and track user interactions.
- **Smart Recommendations**: Our algorithm suggests products based on user preferences.
- **User-Friendly Interface**: Easy to navigate, ensuring a smooth experience.

Why It Matters:

- **E-commerce**: Think of a shopping site that knows your taste.
- **Media Streaming**: Get movie or music recommendations based on your history.
- **Social Networks**: Find content and friends that match your interests.
- **Education**: Personalized course suggestions for your learning path.

This project combines tech and personalization to create something really useful and exciting. Ready to dive in? Let’s go!

Methodology:

The development of the Product Recommendation System Using Graphs was guided by a clear vision: to create a user-friendly, efficient, and scalable solution that can recommend products based on user interactions. Here’s how the project was approached:

1. Understanding the Problem
We started by identifying the core objective: to provide personalized product recommendations by analyzing relationships between users and products. Real-world examples, like e-commerce platforms, helped us define what the system needed—such as a way to capture user interactions and prioritize products effectively.

2. Designing the System
To make the system work seamlessly, we broke it down into three main components:
- Graph-Based Representation:
  - Users and products are represented as nodes in a graph, with weighted edges connecting them based on their interactions (e.g., ratings or purchases). This structure captures the relationships effectively.
- Product Management:
  - A catalog stores product details like name, category, price, and description. Users can add, update, view, or remove products easily.
- User Interface:
  - A simple menu-driven interface allows users to interact with the system intuitively, whether they're looking for recommendations or managing products.

3. Building the System
The actual coding was divided into manageable parts:
- Graph Operations:
  - We implemented an adjacency list to efficiently store and access the graph. This allowed us to add users, products, and their interactions as nodes and edges.
- Recommendation Engine:
  - Using a Breadth-First Search (BFS), we explored user-product connections in the graph and prioritized products based on their interaction weights. A priority queue helped rank the products and return the top recommendations.
- Product Management Features:

- Functions for creating, updating, deleting, and listing products ensured the catalog could be managed dynamically.
- Interactive Menu:
- A menu system tied everything together, making it easy for users to perform various actions like viewing recommendations or visualizing the graph.

4. Testing and Debugging
We made sure the system was reliable and user-friendly by testing it thoroughly:
-Functionality: Checked whether features like adding nodes, generating recommendations, and managing products worked as intended.
- Edge Cases: Handled scenarios like empty graphs, missing users, or invalid inputs gracefully.
- Performance: Ensured the system could scale as the number of users and products grew.

Implementation:

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <queue>
#include <set>
#include <string>
#include <limits>
#include <fstream>
#include <algorithm>

using namespace std;
```

```
// Graph structure to hold users and products
class Graph {
   unordered_map<string, vector<pair<string, double>>> adjList;

public:
   void addNode(const string& node) {
      if (adjList.find(node) == adjList.end()) {
         adjList[node] = vector<pair<string, double>>();
      }
   }

   void addEdge(const string& node1, const string& node2, double
weight = 1.0) {
      adjList[node1].push_back(make_pair(node2, weight));
      adjList[node2].push_back(make_pair(node1, weight));
   }

   const vector<pair<string, double>>& getNeighbors(const string&
node) const {
      return adjList.at(node);
   }

   bool containsNode(const string& node) const {
      return adjList.find(node) != adjList.end();
   }

   void displayGraph() const {
      for (const auto& node : adjList) {
         cout << node.first << ": ";
         for (const auto& edge : node.second) {
            cout << "(" << edge.first << ", " << edge.second << ") ";
         }
         cout << endl;
      }
   }
};
```

```
class Product {
public:
   string productName;
   string category;
   double price;
   string description;

   Product(const string& name, const string& category, double price,
const string& description)
      : productName(name), category(category), price(price),
description(description) {}
};

vector<string> recommendProducts(const Graph& g, const string&
user, int maxRecommendations) {
   if (!g.containsNode(user)) {
      cout << "User not found!" << endl;
      return {};
   }

   set<string> visited;
   priority_queue<pair<double, string>> pq;
   unordered_map<string, double> productScores;

   queue<string> bfsQueue;
   bfsQueue.push(user);
   visited.insert(user);

   while (!bfsQueue.empty()) {
      string currentUser = bfsQueue.front();
      bfsQueue.pop();

      for (const auto& neighbor : g.getNeighbors(currentUser)) {
         string neighborNode = neighbor.first;
         double weight = neighbor.second;
```

```
            if (g.containsNode(neighborNode)) {
                productScores[neighborNode] += weight;
            } else if (visited.find(neighborNode) == visited.end()) {
                bfsQueue.push(neighborNode);
                visited.insert(neighborNode);
            }
        }
    }

    for (const auto& score : productScores) {
        pq.push({score.second, score.first});
    }

    vector<string> recommendations;
    while (!pq.empty() && recommendations.size() <
maxRecommendations) {
        recommendations.push_back(pq.top().second);
        pq.pop();
    }

    return recommendations;
}

int getIntInput(const string& prompt) {
    int value;
    while (true) {
        cout << prompt;
        cin >> value;
        if (!cin.fail() && value > 0) {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            return value;
        } else {
            cout << "Invalid input. Please enter a positive integer.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
```

```
}

double getDoubleInput(const string& prompt) {
    double value;
    while (true) {
        cout << prompt;
        cin >> value;
        if (!cin.fail() && value >= 0) {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            return value;
        } else {
            cout << "Invalid input. Please enter a non-negative number.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
    }
}

string getStringInput(const string& prompt) {
    string value;
    while (true) {
        cout << prompt;
        cin >> ws;
        getline(cin, value);
        if (!value.empty()) {
            return value;
        } else {
            cout << "Input cannot be empty. Please try again.\n";
        }
    }
}

Product createProduct() {
    cout << "Adding a New Product" << endl;
    string name = getStringInput("Enter the name of the product: ");
    string category = getStringInput("Enter the category of the product:
");
```

```
    double price = getDoubleInput("Enter the price of the product: ");
    string description = getStringInput("Enter a description for the
product: ");

    return Product(name, category, price, description);
}

void removeProduct(vector<Product>& products, const string&
productName) {
    auto it = remove_if(products.begin(), products.end(), [&](const
Product& p) {
        return p.productName == productName;
    });

    if (it != products.end()) {
        products.erase(it, products.end());
        cout << "Product '" << productName << "' removed successfully!"
<< endl;
    } else {
        cout << "Product not found!" << endl;
    }
}

void updateProduct(vector<Product>& products, const string&
productName) {
    for (auto& product : products) {
        if (product.productName == productName) {
            cout << "Updating Product: " << productName << endl;
            product.category = getStringInput("Enter the new category: ");
            product.price = getDoubleInput("Enter the new price: ");
            product.description = getStringInput("Enter the new description:
");
            cout << "Product updated successfully!" << endl;
            return;
        }
    }
    cout << "Product not found!" << endl;
```

```
}

void displayAllProducts(const vector<Product>& products) {
   cout << "Available Products" << endl;
   for (const auto& product : products) {
      cout << "Product Name: " << product.productName << ",
Category: " << product.category
           << ", Price: $" << product.price << endl;
   }
}

void displayProductDetails(const vector<Product>& products, const
string& productName) {
   for (const auto& product : products) {
      if (product.productName == productName) {
         cout << "Product Name: " << product.productName << endl;
         cout << "Category: " << product.category << endl;
         cout << "Price: $" << product.price << endl;
         cout << "Description: " << product.description << endl;
         return;
      }
   }
   cout << "Product not found!" << endl;
}

int main() {
   Graph g;
   vector<Product> productCatalog;

   cout << "Welcome to the Product Recommendation System" << endl;

   int numUsers = getIntInput("Enter the number of users: ");
   for (int i = 1; i <= numUsers; ++i) {
      string userName = getStringInput("Enter the name of User " +
to_string(i) + ": ");
      g.addNode(userName);
   }
```

```
    int numProducts = getIntInput("Enter the number of products: ");
    for (int i = 1; i <= numProducts; ++i) {
        Product newProduct = createProduct();
        g.addNode(newProduct.productName);
        productCatalog.push_back(newProduct);
    }

    int numInteractions = getIntInput("Enter the number of user-product
interactions: ");
    for (int i = 0; i < numInteractions; ++i) {
        cout << "Adding Interaction " << (i + 1) << endl;
        string userName = getStringInput("Enter the user name: ");
        string productName = getStringInput("Enter the product name: ");
        double weight = getDoubleInput("Enter the interaction weight (e.g.,
1 to 5): ");

        if (g.containsNode(productName)) {
            g.addEdge(userName, productName, weight);
            cout << "Added interaction from " << userName << " to " <<
productName << " with weight " << weight << endl;
        } else {
            cout << "Product not found!" << endl;
        }
    }

    while (true) {
        cout << "\nMain Menu" << endl;
        cout << "1. Get Product Recommendations" << endl;
        cout << "2. Display Product Details" << endl;
        cout << "3. Display All Products" << endl;
        cout << "4. Remove a Product" << endl;
        cout << "5. Update Product Details" << endl;
        cout << "6. Display Graph (for debugging)" << endl;
        cout << "7. Exit" << endl;

        int choice = getIntInput("Choose an option (1-7): ");
```

```
    switch (choice) {
        case 1: {
            string targetUser = getStringInput("Enter the target user for
recommendations: ");
            int maxRecommendations = getIntInput("Enter the maximum
number of recommendations: ");
            vector<string> recommendations = recommendProducts(g,
targetUser, maxRecommendations);

            cout << "Recommendations for " << targetUser << ": ";
            if (recommendations.empty()) {
                cout << "No recommendations found." << endl;
            } else {
                for (const auto& product : recommendations) {
                    cout << product << " ";
                }
                cout << endl;
            }
            break;
        }
        case 2: {
            string productName = getStringInput("Enter the product name
to view details: ");
            displayProductDetails(productCatalog, productName);
            break;
        }
        case 3: {
            displayAllProducts(productCatalog);
            break;
        }
        case 4: {
            string productName = getStringInput("Enter the product name
to remove: ");
            removeProduct(productCatalog, productName);
            break;
        }
        case 5: {
```

```
            string productName = getStringInput("Enter the product name
    to update: ");
            updateProduct(productCatalog, productName);
            break;
          }
          case 6: {
            cout << "Graph Structure" << endl;
            g.displayGraph();
            break;
          }
          case 7: {
            cout << "Exiting the program. Thank you!" << endl;
            return 0;
          }
          default: {
            cout << "Invalid choice. Please select a valid option." << endl;
          }
        }
      }

      return 0;
    }
```

Output:

Welcome to the Product Recommendation System:

Enter the number of users: 2

Enter the name of User 1: Raj

Enter the name of User 2: Priya

Enter the number of products: 2

Adding a New Product

Enter the name of the product: Laptop

Enter the category of the product: Electronics

Enter the price of the product: 75000

Enter a description for the product: High-performance laptop

Adding a New Product

Enter the name of the product: Mobile

Enter the category of the product: Electronics

Enter the price of the product: 20000

Enter a description for the product: Smartphone with latest features

Enter the number of user-product interactions: 2

Adding Interaction 1

Enter the user name: Raj

Enter the product name: Laptop

Enter the interaction weight (e.g., 1 to 5): 5

Added interaction from Raj to Laptop with weight 5

Adding Interaction 2

Enter the user name: Priya

Enter the product name: Mobile

Enter the interaction weight (e.g., 1 to 5): 4

Added interaction from Priya to Mobile with weight 4

Main Menu

1. Get Product Recommendations

2. Display Product Details

3. Display All Products

4. Remove a Product

5. Update Product Details

6. Display Graph (for debugging)

7. Exit

Choose an option (1-7): 1

Enter the target user for recommendations: Raj

Enter the maximum number of recommendations: 2

Recommendations for Raj: Laptop

Updating and Removing Products:

Input

1. Choose option: 4
2. Product to remove: Mobile
3. Choose option: 5
4. Product to update: Laptop
5. New details:
    - Category: Computers
    - Price: 68000.00
    - Description: Updated high-performance laptop

Main Menu

1. Get Product Recommendations

2. Display Product Details

3. Display All Products

4. Remove a Product

5. Update Product Details

6. Display Graph (for debugging)

7. Exit

Choose an option (1-7): 4

Enter the product name to remove: Mobile

Product 'Mobile' removed successfully!

Main Menu

1. Get Product Recommendations
2. Display Product Details
3. Display All Products
4. Remove a Product
5. Update Product Details
6. Display Graph (for debugging)
7. Exit

Choose an option (1-7): 5

Enter the product name to update: Laptop

Updating Product: Laptop

Enter the new category: Computers

Enter the new price: 68000

Enter the new description: Updated high-performance laptop

Product updated successfully!

Display All Products and Product Details:

Input

1. Choose option: 3 to display all products
2. Choose option: 2 to display product details
3. Product name: Laptop

Main Menu

1. Get Product Recommendations
2. Display Product Details

3. Display All Products

4. Remove a Product

5. Update Product Details

6. Display Graph (for debugging)

7. Exit

Choose an option (1-7): 3

Available Products

Product Name: Laptop, Category: Computers, Price: ₹68000.00

Main Menu

1. Get Product Recommendations

2. Display Product Details

3. Display All Products

4. Remove a Product

5. Update Product Details

6. Display Graph (for debugging)

7. Exit

Choose an option (1-7): 2

Enter the product name to view details: Laptop

Product Name: Laptop

Category: Computers

Price: ₹68000.00

Description: Updated high-performance laptop

Discussion:

Building a Product Recommendation System has been quite the journey, and there's a lot to reflect on. This project was not just about writing code; it involved solving real-world problems and understanding how people interact with products and technology.

Key Components:

1.Graph Structure:
- We started by designing a graph to represent relationships between users and products. Each user and product became a node in this graph, and their interactions were the connecting edges. These connections help us understand who likes what, and how much they like it.

2.Product Management:
- Managing the products was crucial. We needed to add, remove, and update product details to keep our recommendations relevant and accurate. This ensured that users always had up-to-date information on their favorite items.

3.User-Product Interactions:
- Capturing interactions between users and products was essential. These interactions, represented by weighted edges in the graph, allowed us to gauge the strength of a user's preference for a product. The more a user interacted with a product, the higher the weight, making our recommendations smarter.

4.Recommendation Algorithm:
- Our recommendation algorithm used Breadth-First Search (BFS) to traverse the graph and gather data on potential product recommendations. By prioritizing products with higher interaction weights, we could suggest items that were likely to interest the user.

Challenges Encountered:

1.Data Representation:
- Designing a graph that accurately mirrored the complex relationships and interactions between users and products wasn't easy. We had to make sure that our data structures and algorithms allowed for efficient traversal and manipulation of the graph.

2.Balancing Recommendations:
- One of the tricky parts was balancing highly rated products against those with frequent interactions. We had to tweak our weighting mechanism multiple times to ensure it accurately reflected user preferences without bias.

3.Scalability:
- As our dataset grew, we had to make sure our recommendation algorithm scaled well. Optimizing BFS traversal and keeping our graph operations efficient were critical to handle larger volumes of data.

4.User Engagement:
- Ensuring our recommendations were relevant and engaging was paramount. We needed to account for a wide range of user preferences and provide a diverse set of product suggestions to keep users interested and satisfied.

Real-World Applications

1.E-commerce Platforms:
- Imagine this system on an e-commerce platform like Flipkart or Amazon, tailoring product suggestions to each user based on their interactions. This would enhance customer satisfaction and boost sales.

2.Media Streaming Services:
- For platforms like Netflix or Spotify, a similar approach could recommend movies, TV shows, or music based on user viewing or listening habits, improving user engagement and retention.

3.Social Networks:
- Social networks like Facebook or Instagram could use this to recommend content, friends, or groups based on user interactions, creating a more personalized and engaging experience.

4.Educational Platforms:
- On sites like Coursera or edX, this system could suggest courses or resources tailored to individual learning patterns, enhancing the educational experience.

Conclusion:

Creating this Product Recommendation System was both challenging and rewarding. We learned the importance of efficient data structures and algorithms in developing a scalable and effective recommendation system. Moving forward, exploring advanced techniques like machine learning could further improve the accuracy and relevance of our recommendations.

References

"Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein:

1. A comprehensive guide to algorithms, including graph algorithms which are fundamental for this project.

2. **ChatGpt** :
    - Used extensively for researching C++ functions, debugging issues, and exploring various programming-related resources to implement features like user authentication and product display in the Product recommendation system.