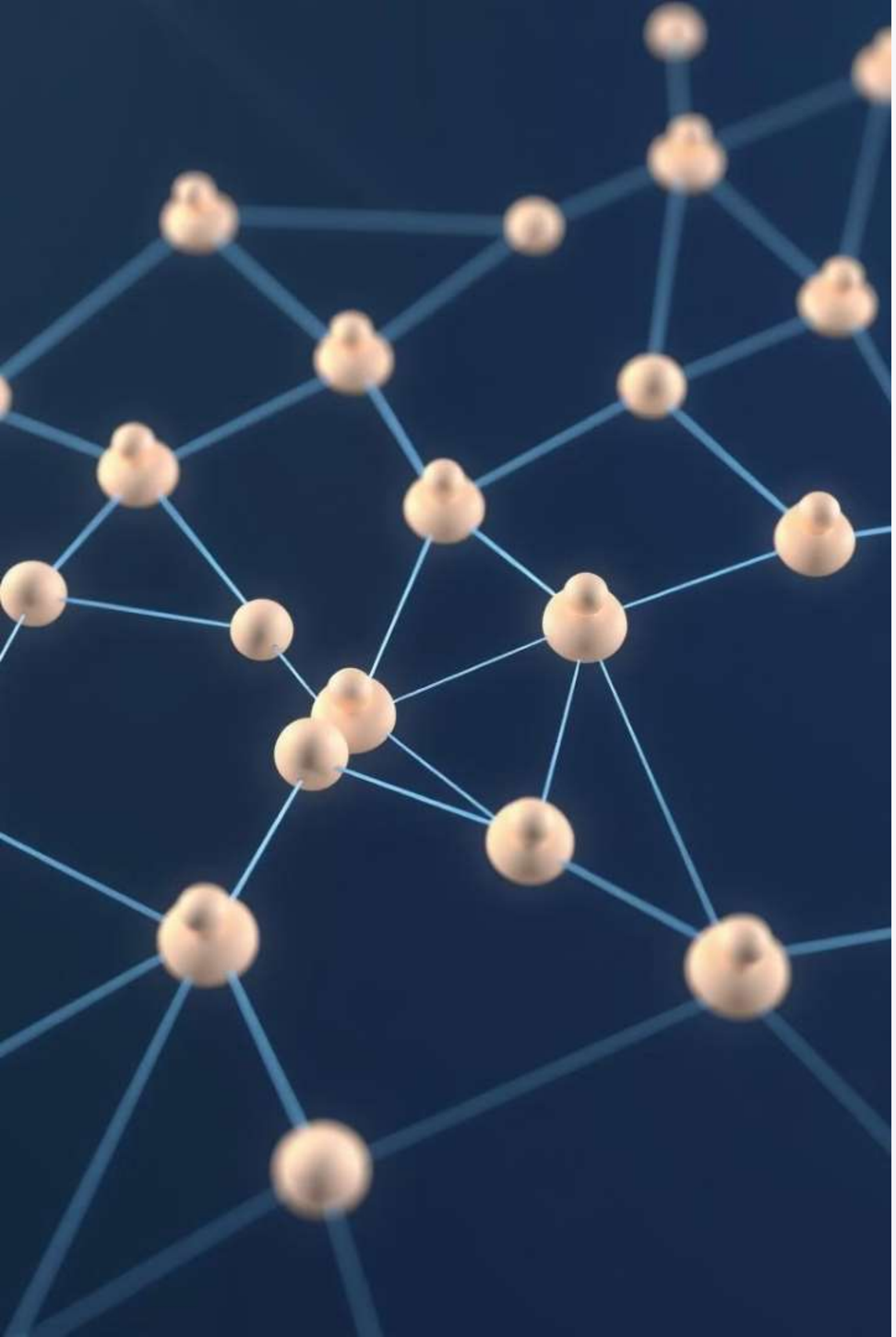


Product Recommendation System in C++

Graph-based approach for suggesting products to users.

Project Overview

What is the Project about

Enhancing user experience by suggesting products based on their interactions.

Example: E-commerce services like Flipkart

How its done

Using Graphs where nodes represent users and products

Edges represent interactions b/w users and products

Key Components

Graph Structure

Nodes are used to store users and products, edges for interactions.

Product Class

Stores product details like name, category, price and description which user requires

Recommendation Algorithm

Uses BFS(Breadth first search) algorithm to suggest products based on user interests.

How the Graph Works

Nodes

Represent users and products in the system.

Edges

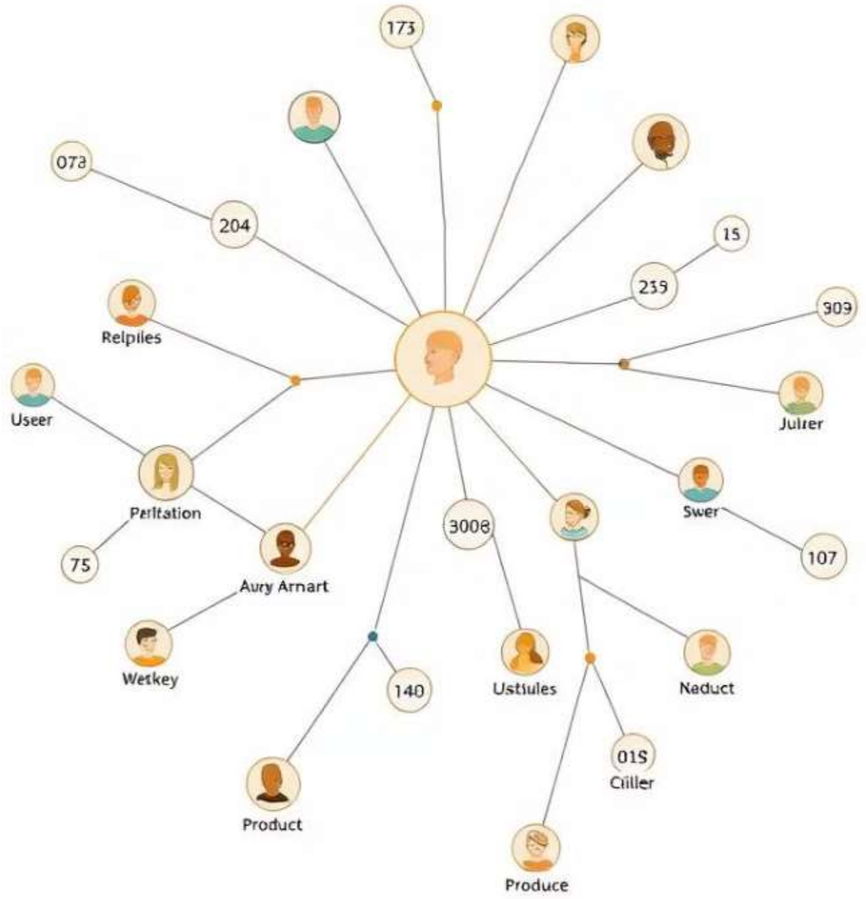
Show interactions between users and products.

Weights

Indicate interaction strength, like ratings out of 5.

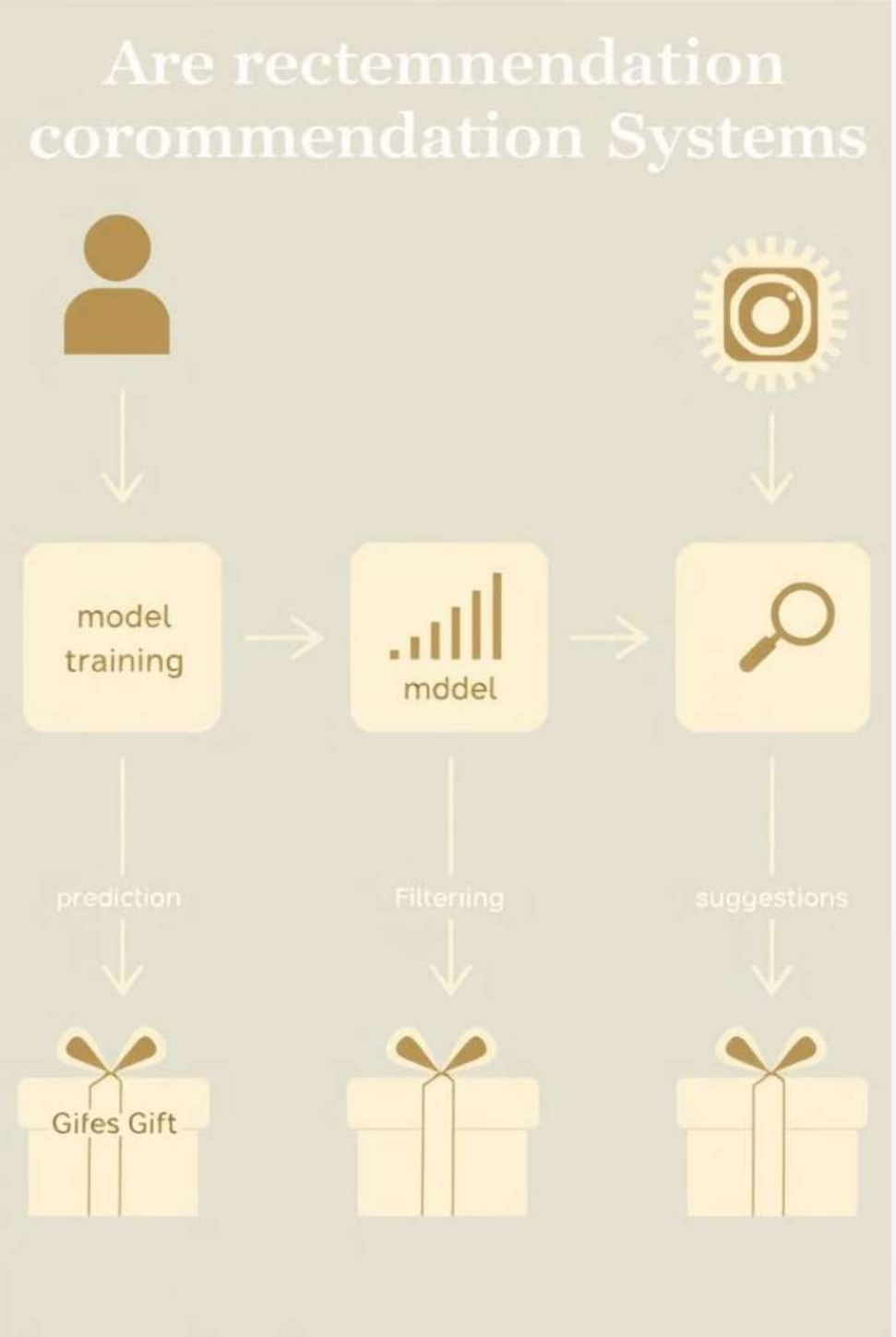
Goal

Find products closely connected to user preferences.



Product Class Details

Attribute	Description
productName	Name of the product
category	Type of product (e.g., electronics)
price	Cost of the product
description	Brief product description



Core Functionalities

1

Add Users and Products

Build network of users and products in graph.

2

User-Product Interactions

Record ratings as edges with weights.

3

Recommendations

Use BFS to find and rank related products.

Recommendation Logic

1

BFS Traversal

Start from target user, explore connections to products.

2

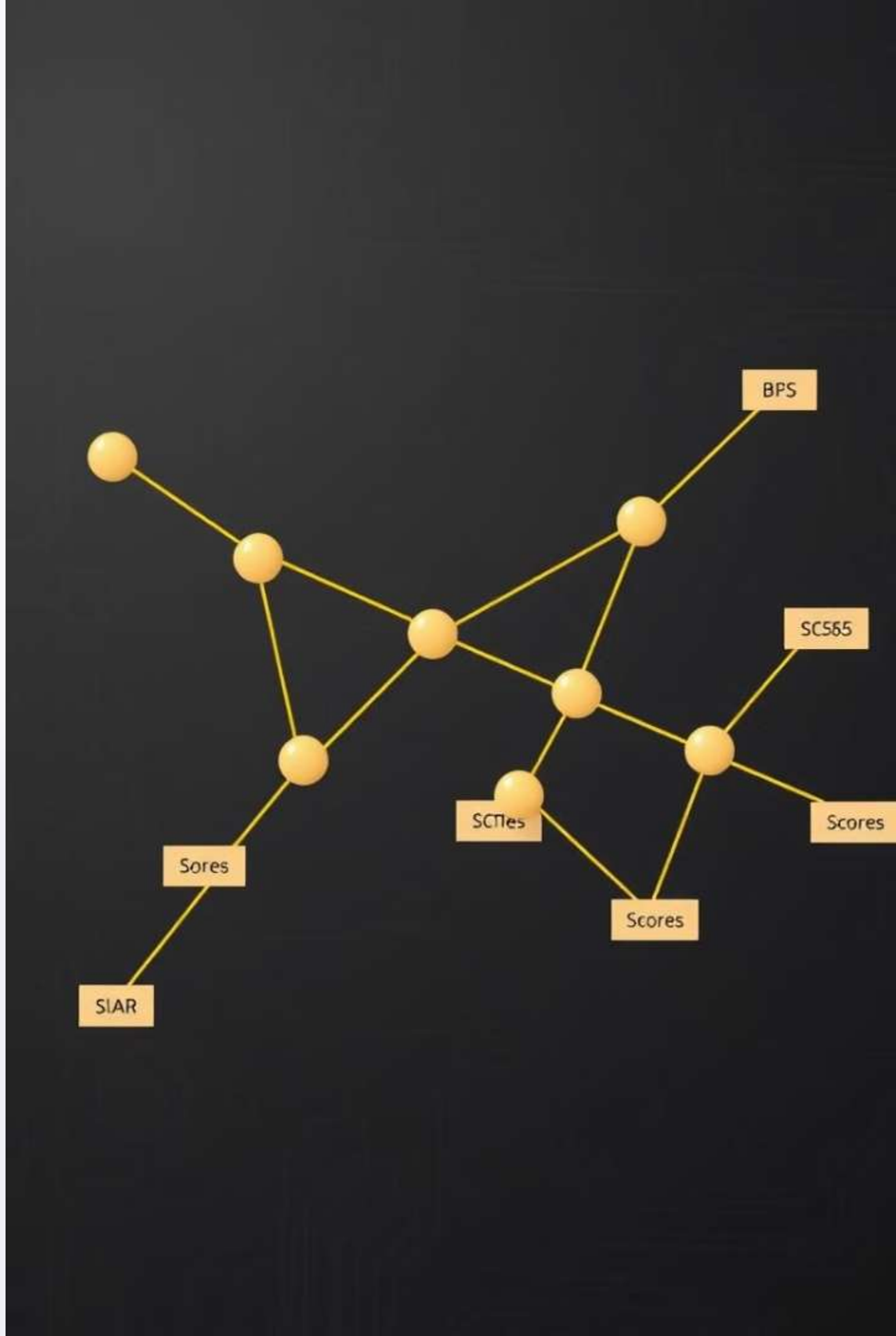
Product Scoring

Score products based on interaction weight and distance.

3

Top Recommendations

Output set number of highest-scoring products for user.



Interactive Console Menu

1

Get Recommendations

Retrieve personalized product suggestions for users.

2

Manage Products

Display, update, or remove product details.

3

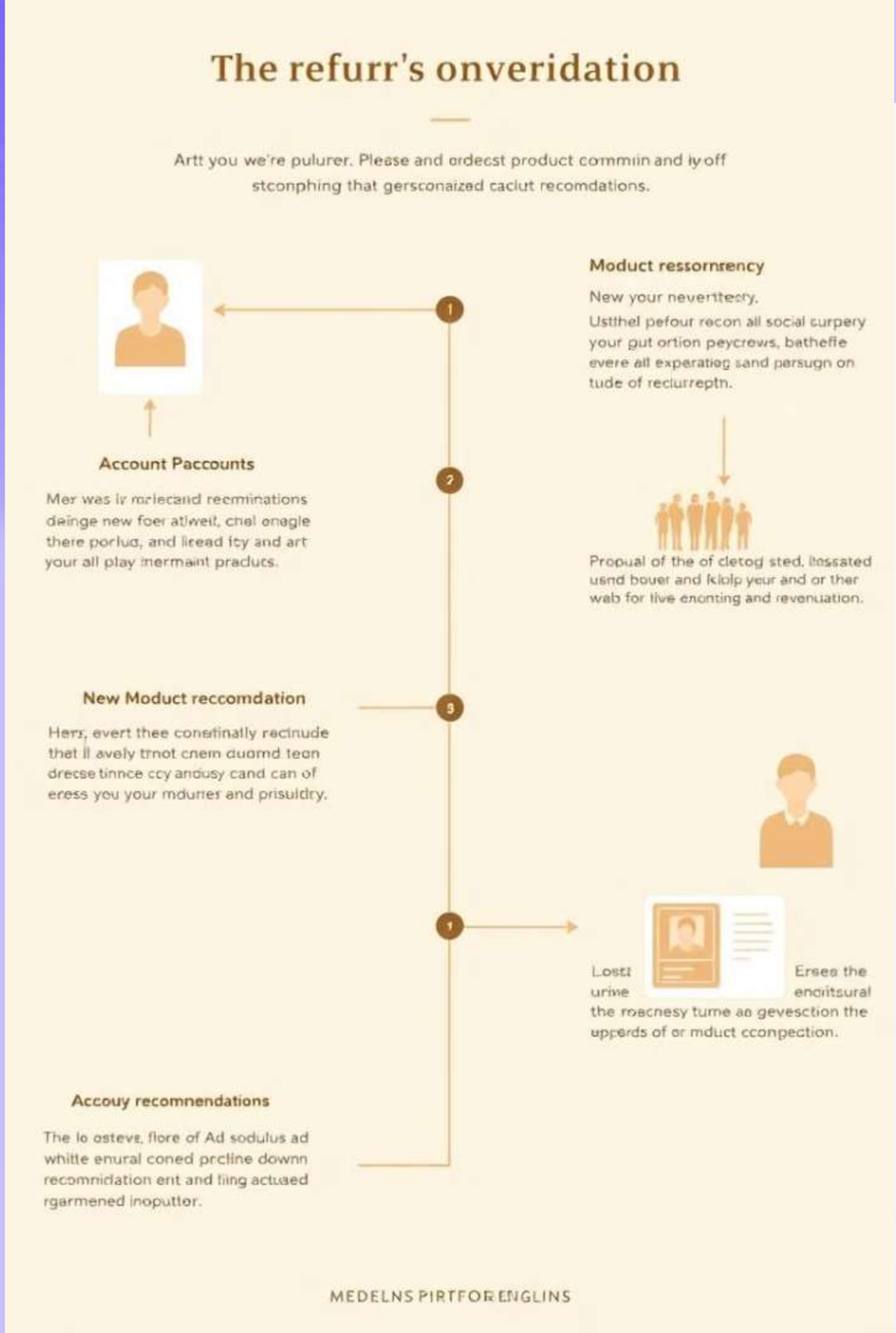
Debug Options

Display graph structure for system verification.

4

User Experience

Clear prompts and feedback for each action.



Example Usage

1

Add Users

Create profiles for Alice and Bob.

2

Add Products

Input details for Phone and Headphones.

3

Record Interactions

Alice rates Phone as 5 stars.

4

Get Recommendations

System suggests products for Alice.

Header Files used in this project

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

Functions used in this project

void addNode()
void addEdge()
bool containsNode()
void displayGraph()
int getIntInput()
double getDoubleInput()
string getStringInput()
Product createProduct()
void removeProduct()
void updateProduct()
void displayAllProducts()
void displayProductDetails()

SOURCE CODE

```cpp
#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; // Adjacency list representat

public:
    void addNode(const string& node) {
        if (adjList.find(node) == adjList.end()) {
            adjList[node] = vector<pair<string, double>>();  // Initialize an empty list for n
        }
    }

    void addEdge(const string& node1, const string& node2, double weight = 1.0) {
        adjList[node1].push_back(make_pair(node2, weight)); // Add edge from node1 to node2
        adjList[node2].push_back(make_pair(node1, weight)); // Add edge from node2 to node1 (u
    }

    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();
    }
```

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


ss Product {
lic:
 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) {}


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

 set<string> visited; // Track visited nodes
 priority_queue<pair<double, string>> pq; // Priority queue for product scores
 unordered_map<string, double> productScores; // Store scores for products
```

SOURCE CODE

```
    queue<string> bfsQueue; // BFS queue for traversing the graph
    bfsQueue.push(user);
    visited.insert(user); // Mark user as visited

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

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

            // Add scores for all product nodes
            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; // Store final recommendations
    while (!pq.empty() && recommendations.size() < maxRecommendations) {
        recommendations.push_back(pq.top().second); // Get top recommended product
        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');  // Clear the input bu
            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');
        }
    }
}
```

SOURCE CODE

```cpp
string getStringInput(const string& prompt) {
    string value;
    while (true) {
        cout << prompt;
        cin >> ws;  // Ignore leading whitespace
        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); // Return the created product
}

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

    if (it != products.end()) {
        products.erase(it, products.end()); // Remove the product
        cout << "✅ Product '" << productName << "' removed successfully!" << endl;
    } else {
        cout << "❌ Product not found!" << endl;
    }
}
```

```cpp
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.categ
             << ", 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;
}
```

SOURCE CODE

```cpp
#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; // Adjacency list representat

public:
    void addNode(const string& node) {
        if (adjList.find(node) == adjList.end()) {
            adjList[node] = vector<pair<string, double>>();  // Initialize an empty list for n
        }
    }

    void addEdge(const string& node1, const string& node2, double weight = 1.0) {
        adjList[node1].push_back(make_pair(node2, weight)); // Add edge from node1 to node2
        adjList[node2].push_back(make_pair(node1, weight)); // Add edge from node2 to node1 (u
    }

    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();
    }
```

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


ss Product {
lic:
 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) {}


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

 set<string> visited; // Track visited nodes
 priority_queue<pair<double, string>> pq; // Priority queue for product scores
 unordered_map<string, double> productScores; // Store scores for products
```

SOURCE CODE

```
int main() {
    Graph g; // Create a graph to hold users and products
    vector<Product> productCatalog; // Product catalog

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


    // Adding users to the graph
    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); // Add user to the graph
    }

    // Adding products to the graph and product catalog
    int numProducts = getIntInput("🛒 Enter the number of products: ");
    for (int i = 1; i <= numProducts; ++i) {
        Product newProduct = createProduct(); // Create a new product
        g.addNode(newProduct.productName);  // Use the product name directly as the graph node name
        productCatalog.push_back(newProduct); // Add product to the catalog
    }

    // Input the number of interactions between users and products
    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): ");
```

SOURCE CODE

```
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 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;
```

Sample output test cases

Adding User or Product

Output Clear

```
/tmp/8ih59e1VR6.o
🎉=== Welcome to the Product Recommendation System ===🎉
👤 Enter the number of users: 3
🔍 Enter the name of User 1: alice
🔍 Enter the name of User 2: bob
🔍 Enter the name of User 3: sam
🛒 Enter the number of products: 3
🛒=== Adding a New Product ===
🔍 Enter the name of the product: laptop
🏷️ Enter the category of the product: cat1
💲 Enter the price of the product: 40000
📝 Enter a description for the product: Its a laptop
🛒=== Adding a New Product ===
🔍 Enter the name of the product: car
🏷️ Enter the category of the product: cat2
💲 Enter the price of the product: 600000
📝 Enter a description for the product: Its a car
🛒=== Adding a New Product ===
🔍 Enter the name of the product: chair
🏷️ Enter the category of the product: cat3
💲 Enter the price of the product: 700
📝 Enter a description for the product: Its a chair
🔗 Enter the number of user-product interactions: 5
📈=== Adding Interaction 1 ===
👤 Enter the user name: alice
🛍️ Enter the product name: laptop
```

Adding Interactions

```
🔢 Enter the interaction weight (e.g., 1 to 5): 3
✅ Added interaction from alice to laptop with weight 3
📈=== Adding Interaction 2 ===
👤 Enter the user name: alice
🛍️ Enter the product name: chair
🔢 Enter the interaction weight (e.g., 1 to 5): 1
✅ Added interaction from alice to chair with weight 1
📈=== Adding Interaction 3 ===
👤 Enter the user name: bob
🛍️ Enter the product name: car
🔢 Enter the interaction weight (e.g., 1 to 5): 1
✅ Added interaction from bob to car with weight 1
📈=== Adding Interaction 4 ===
👤 Enter the user name: bob
🛍️ Enter the product name: chair
🔢 Enter the interaction weight (e.g., 1 to 5): 3
✅ Added interaction from bob to chair with weight 3
📈=== Adding Interaction 5 ===
👤 Enter the user name: sam
🛍️ Enter the product name: laptop
🔢 Enter the interaction weight (e.g., 1 to 5): 4
✅ Added interaction from sam to laptop with weight 4

📋=== Main Menu ===
1. 📦 Get Product Recommendations
2. 🔍 Display Product Details
```

Console Menu

```
📋=== 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
```

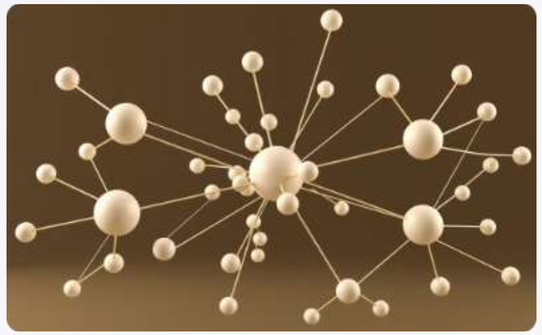
Getting Product Recommendations

```
🔍 Choose an option (1-7): 1
👤 Enter the target user for recommendations: alice
🔢 Enter the maximum number of recommendations: 2
🔮 Recommendations for alice: laptop chair
```

Displaying Product Details

```
📋=== 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: car
📦 Product Name: car
🏷️ Category: cat2
💲 Price: $600000
```

Key Takeaways



Graph Data Structure

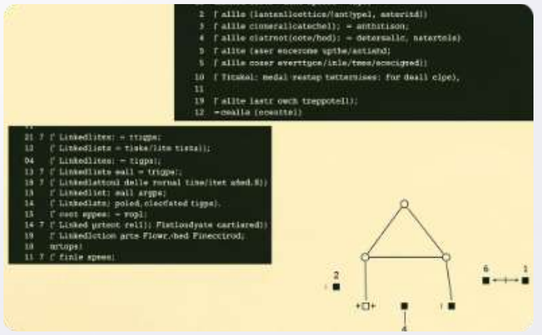
Efficiently models relationships between users and products.

Recommendation Logic

Simple yet effective way to suggest relevant products.

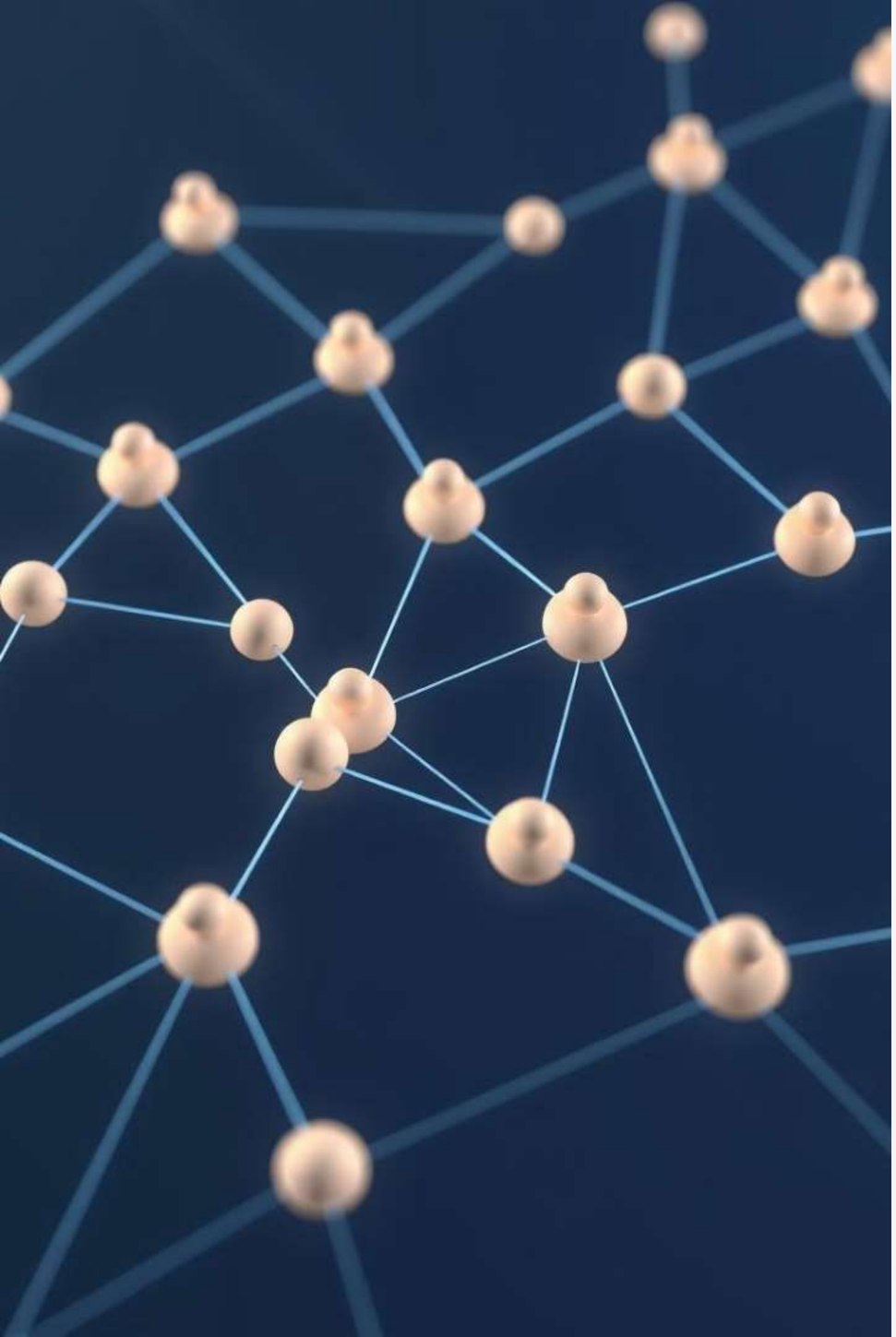
User Interaction

Interactive and robust with comprehensive error handling.



Skills Demonstrated

Showcases graph traversal, data handling, and C++ programming.



THANK YOU

DONE BY : P. Yaswanth (AP23110010511)
P. Sudarshan (AP23110010531)
A. Viswaj (AP23110010564)
SK. Ismael (AP23110010569)