Data Structures and Algorithm Projesi

Online Sipariş Yönetim Sistemi:

1. Proje Amacı

Bu proje, bir çevrimiçi perakende şirketinin siparişlerini verimli bir şekilde yönetebilmesi için bir ağaç veri yapısı kullanarak geliştirilmiştir. Sistem, her bir urunu bir düğüm olarak ele alır ve urun adi ile sipariş miktarını saklar.

2. Kullanılan Veri Yapısı

Proje, siparişlerin saklanması ve düzenlenmesi için Binary Search Tree (BST) veri yapısını kullanır. BST, verilerin alfabetik sıraya göre eklenmesini ve sorgulanmasını kolaylaştırır. Ağaç yapısı, hızlı ekleme, silme ve sorgulama işlemleri için uygundur.

3. Temel Fonksiyonlar

- **Sipariş Ekleme(Add Orders):** Birden fazla urun ve miktar tek bir komutla eklenebilir.

  Örnek "Elma:3, Muz:2."

```java
public void addOrder(String input) {  // Main function to add orders
    // Split input based on commas (Elma:3, Muz:2, Kiraz:1)
    String[] products = input.split(",");

    for (String product : products) {
        // Trim spaces and split by colon (Elma:3)
        String[] parts = product.trim().split(":");

        // Validate format
        if (parts.length != 2) {
            System.out.println("Invalid format for: " + product);
            continue;
        }

        String productName = parts[0].trim();
        int orderCount;

        try {
            orderCount = Integer.parseInt(parts[1].trim());
        } catch (NumberFormatException e) {
            System.out.println("Invalid quantity for: " + product);
            continue;
        }

        if (orderCount <= 0) { // Check for valid order count
            System.out.println("Order count must be greater than 0 for: " + productName);
            continue;
        }

        // Call recursive function to add orders into the tree
        root = addOrderRec(root, productName, orderCount);
        System.out.println("Added " + productName + " with count " + orderCount);
    }
```

```java
// Recursive function to add products to the tree
private Node addOrderRec(Node root, String productName, int orderCount) {
    if (root == null) { // Add new node if tree is empty
        return new Node(productName, orderCount);
    }
    if (productName.compareTo(root.productName) < 0) { // Go left for alphabetically
        root.leftChild = addOrderRec(root.leftChild, productName, orderCount);
    } else if (productName.compareTo(root.productName) > 0) { // Go right for alphabetically
        root.rightChild = addOrderRec(root.rightChild, productName, orderCount);
    } else { // If product already exists, increase order count
        root.orderCount += orderCount;
    }
    return root;
}
```

Bu fonksiyon, birden fazla urunu tek bir komutta eklemeye olanak tanır. Hatalı girişler kontrol edilir ve geçersiz olanlar atlanır.

- **Sipariş İptali(Cancel Order):** Birden fazla urun ayni anda iptal edilebilir.

  Örnek: “Elma, Muz”.

```java
// Function to cancel orders
public void cancelOrder(String input) {
    String[] products = input.split(",");
    for (String product : products) {
        String productName = product.trim(); // Remove spaces

        if (productName.isEmpty()) { // Skip empty names
            System.out.println("Invalid product name.");
            continue;
        }
        // Eğer root null'sa veya ürün zaten bulunamazsa:
        if (root == null || cancelOrderRec(root, productName) == null) {
            System.out.println("Product not found: " + productName);
        } else {
            root = cancelOrderRec(root, productName);
            System.out.println("Cancelled order for: " + productName);
        }
    }
}
```

```java
// Recursive function to cancel a product
private Node cancelOrderRec(Node root, String productName) {
    if (root == null) { // If the tree is empty return null
        return null;
    }

    if (productName.compareTo(root.productName) < 0) { // Search in the left subtree
        root.leftChild = cancelOrderRec(root.leftChild, productName);
    } else if (productName.compareTo(root.productName) > 0) { // Search in the right subtree
        root.rightChild = cancelOrderRec(root.rightChild, productName);
    } else { // Node to be deleted found
        if (root.leftChild == null) { // Replace with right child
            return root.rightChild;
        } else if (root.rightChild == null) { // Replace with left child
            return root.leftChild;
        }

        // Node has two children: Replace with smallest value in right subtree
        Node minNode = findMin(root.rightChild);
        root.productName = minNode.productName;
        root.orderCount = minNode.orderCount;
        root.rightChild = cancelOrderRec(root.rightChild, minNode.productName);
    }
    return root;
}
```

Bu fonksiyon, birden fazla urunu ayni anda iptal etmek için kullanılır. Ağaçtaki uygun düğümler kaldırılır.

- **Sorgulama(Query):** Belirli urun gruplarının birlikte kaç kez sipariş edildiği sorgulanabilir. Örnek: “Elma, Muz.”

```java
// Function to query how many times a product set was ordered together
public int queryProductSet(String input) {
    // Split and trim input to get product list
    String[] products = input.split(",");
    for (int i = 0; i < products.length; i++) {
        products[i] = products[i].trim(); // Remove spaces
    }

    // Sort products alphabetically
    Arrays.sort(products);

    // Start checking the path from the root node
    return checkPath(root, products, 0); // Recursive kontrol
}
```

```java
// Recursive function to check valid path in the tree
private int checkPath(Node node, String[] products, int index) {
    // Eğer node null ise veya ürün listesi bitmişse dur
    if (node == null || index >= products.length) {
        return 0;
    }

    // Eğer ürün bulunursa, sonraki ürüne geç
    if (node.productName.equals(products[index])) {
        // Son üründe sipariş sayısını döndür
        if (index == products.length - 1) {
            return node.orderCount;
        }

        // Sol ve sağ dallarda aramaya devam et
        int leftResult = checkPath(node.leftChild, products, index + 1);
        int rightResult = checkPath(node.rightChild, products, index + 1);

        // İki sonuçtan biri geçerliyse döndür
        return leftResult + rightResult;
    }

    // Ürün eşleşmezse, ağacın diğer dallarını kontrol et
    int leftSearch = checkPath(node.leftChild, products, index);
    int rightSearch = checkPath(node.rightChild, products, index);

    return leftSearch + rightSearch;
}
```

Bu fonksiyon, verilen urun grubunun birlikte sipariş edilme sayısını kontrol eder. Ürünler, ağaç üzerinde bir yol boyunca araştırılır.

- **Ağaç Yapısı Yazdırma(Print Tree):** Ağacın yapısı konsol üzerinden hiyerarşik olarak görüntülenebilir.

```java
private void printTreeRec(Node node,String prefix) {
    if(node!=null) {
        System.out.println(prefix + node.productName + " (" + node.orderCount + ")");
    }
    if(node.leftChild!=null) {//If the left child exists, adjust the prefix for the right child
        printTreeRec(node.leftChild, prefix.replace("└──", "    ") + "├── ");
    }
    if(node.rightChild!=null) {// Print the right child
        printTreeRec(node.rightChild,prefix.replace("└──", "    ") + "└── ");
    }
}

public void printTree() {
    if(root==null) {
        System.out.println("Tree is empty..");
    }
    else {
        System.out.println("root");
        printTreeRec(root,"└── ");
    }
}
```

Hazırlayan : Serhat Talha Taşan(220313035) ve Rıbar Bayram(220313013)