**Castillo, Anjelica M.**

**MCSCC123 - MSCS12S1**

**Cafe Menu using Naive and Tree Disjoint Sets**

This demonstrates a basic simulation simple cafe menu using the Disjoint Set (Union-Find) data structure. It clusters menu items based on simulated co-purchase patterns—identifying which items are frequently bought together in a simple cafe setup.

1. Naive Implementation

2. Optimized Tree-like Implementation

**# Cafe Menu Items**

menu\_items = [

'Burger', 'Fries', 'Ice Cream', 'Coke',

'Pasta', 'Cookies', 'Coffee', 'Milk Tea'

]

**# Simulated co-purchases (items often bought together)**

co\_purchases = [

('Burger', 'Fries'),

('Burger', 'Coke'),

('Pasta', 'Cookies'),

('Coffee', 'Milk Tea'),

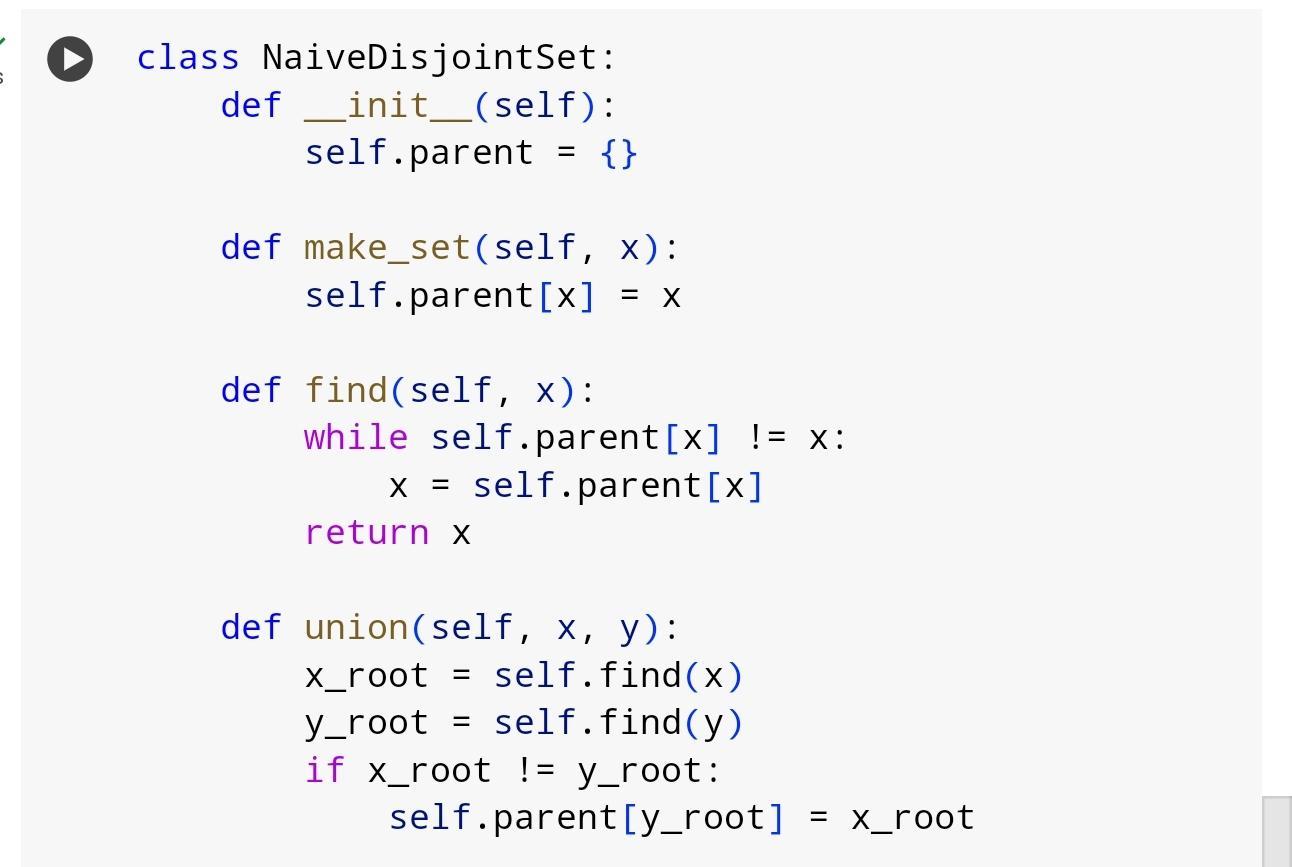
('Ice Cream', 'Cookies')

]

**1. Naive Disjoint Set Implementation**

This basic version uses a dictionary to store parent pointers without any optimization.

Code:

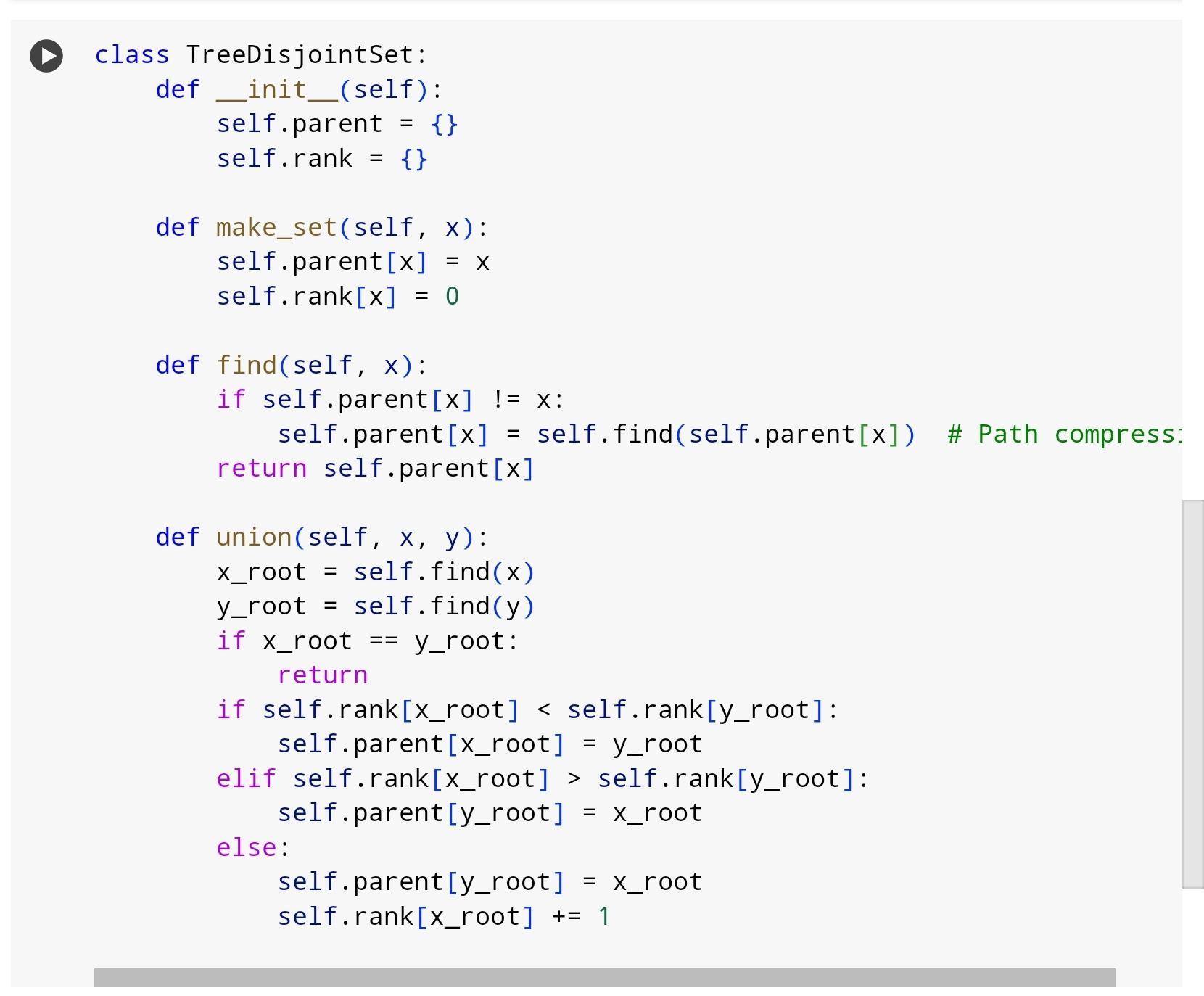


Pros: Easy to understand and implement.

Cons: Can become inefficient for large datasets (deep trees).

**2. Tree Disjoint Set**

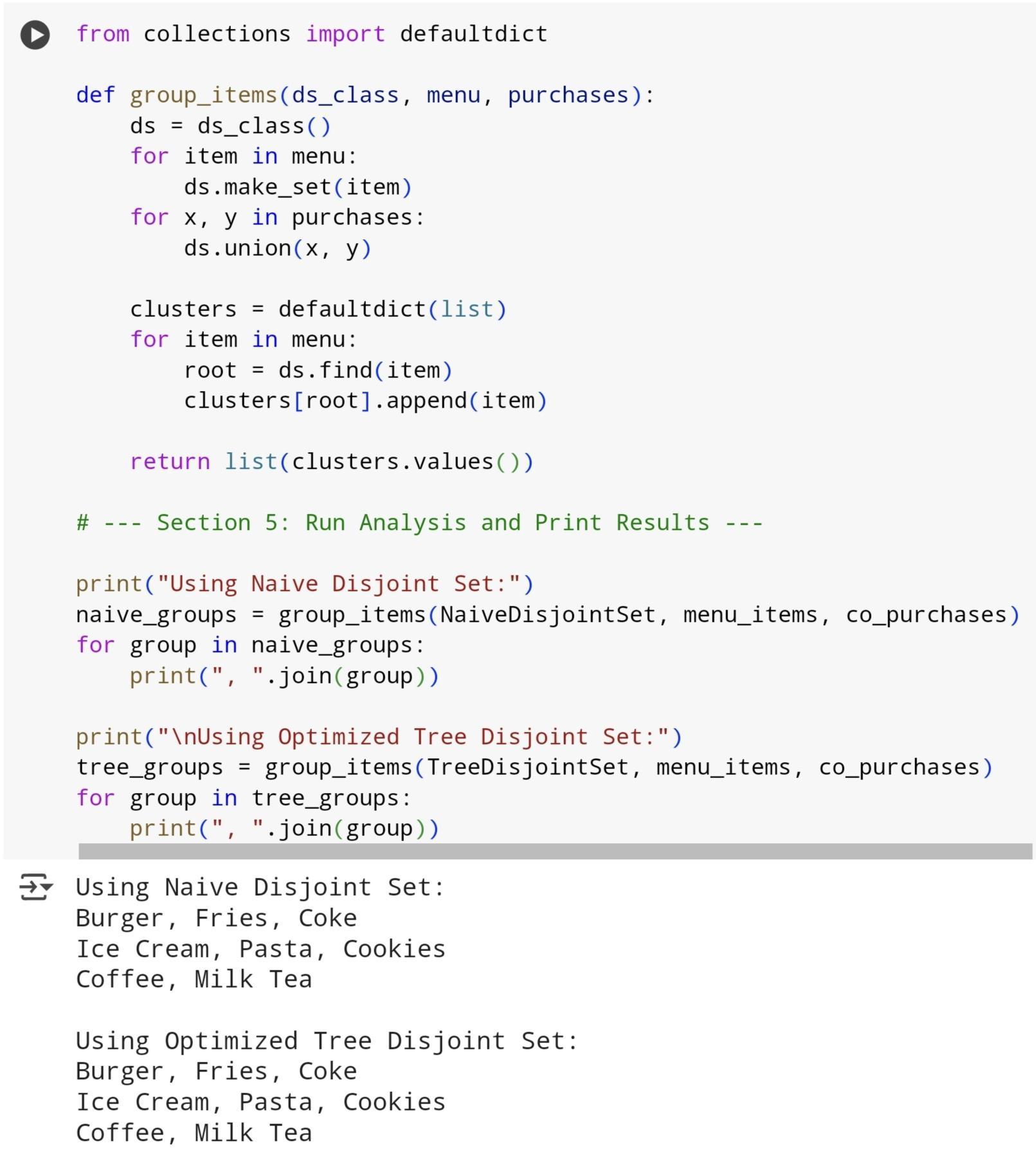
This version reduces lookup and union time to near constant using two techniques.



Pros: Efficient even for large numbers of items.

Cons: Slightly more complex to implement.

**3. Results**



**Conclusion**

Both the Naive and Optimized Tree-based Disjoint Set implementations successfully identified meaningful clusters of items based on simulated co-purchase patterns. Despite differences in internal structure and performance, the final groupings were consistent across both methods:

* **Burger, Fries, Coke** were grouped, suggesting a classic fast-food combo.
* **Ice Cream, Pasta, Cookies** formed a second group, possibly indicating meal-dessert pairings.
* **Coffee and Milk Tea** appeared together, reflecting common beverage preferences.

While the optimized version is more efficient for larger data, both approaches are effective for identifying frequently bought-together items and can support combo deals or menu planning in restaurants.