FinacPlus Software Assignment – 2024

Submitted by

Rajeshwar Reddy Manthena

My LinkedIn Profile

Github Repository of Assignment

Coding Task Challenge

1. **Write a special cipher that is a combination of Caesar's cipher followed by a simple RLE. The function should receive a string and the rotation number as parameters. Input: special Cipher("AABCCC", 3) Output: D2EF3**

Code:

```
import java.util.*;
public class SpecialCipher{
   public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    String input = sc.nextLine();
    int rotation = sc.nextInt();
    input = CaesarsCipher(input,rotation);
    System.out.println(SimpleRLE(input));
   }
   private static String CaesarsCipher(String input,int k){
      StringBuilder result = new StringBuilder();
      for(char c : input.toCharArray()) {
         if (Character.isLetter(c)) {
            char base = Character.isLowerCase(c) ? 'a' : 'A';
            char newChar = (char) (((c - base + k) % 26) + base);
            result.append(newChar);
         } else {
            result.append(c);
         }
      }
      return result.toString();
   }
   private static String SimpleRLE(String input){
      StringBuilder result = new StringBuilder();
      if (input == null || input.isEmpty()) {
         return result.toString();
      }
      char prevChar = input.charAt(0);
      int count = 1;
      for (int i = 1; i < input.length(); i++) {
         char currentChar = input.charAt(i);
         if (currentChar == prevChar) {
            count++;
         } else {
            result.append(prevChar);
           if(count>1) result.append(count);
            count = 1;
            prevChar = currentChar;
         }
      }
      result.append(prevChar);
      if(count>1)result.append(count);
      return result.toString();

   }}
```

[Code link]

2. **Write a program that finds the most optimized set of 6 units to shop with for values fewer than 100. Example: Units used are 1, 2, 5, 10, 20, 50 1: 1 (1 unit used) 2: 2 (1 unit used) 3: 1+2 (2 units used) 4: 2+2 (2 units used) ... 98: 1+2+5+20+20+50 (6 units used) 99: 2+2+5+20+20+50 (6 units used) AVG of units = 3.4**

Code:

```
import java.util.*;
public class OptimizedShopping {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int target = sc.nextInt();
      int[] units = {1, 2, 5, 10, 20, 50};
      List<Integer> result = findOptimizedSet(units, target);
      System.out.println("Optimized set: " + result);
      System.out.println("Average units: " + calculateAverage(result));
   }
   public static List<Integer> findOptimizedSet(int[] units, int target) {
      int[] dp = new int[target + 1];
      int[] parent = new int[target + 1];
      Arrays.fill(parent, -1);
      dp[0] = 0;
      for (int i = 1; i <= target; i++) {
         dp[i] = Integer.MAX_VALUE;
         for (int j = 0; j < units.length; j++) {
            if (i >= units[j] && dp[i - units[j]] + 1 < dp[i]) {
               dp[i] = dp[i - units[j]] + 1;
               parent[i] = j;
            }
         }
      }
      List<Integer> result = new ArrayList<>();
      int remaining = target;
      while (parent[remaining] != -1) {
         int index = parent[remaining];
         result.add(units[index]);
         remaining -= units[index];
      }
      return result;
   }
   public static double calculateAverage(List<Integer> set) {
      if (set.isEmpty()) return 0.0;
      double sum = 0.0;
      for (int num : set) {
         sum += num;
      }
      return sum / set.size();
   }
}
```

Code link

3. **Imagine you work for amazon, what is the Meta data information you will store for an item in your Database. For E.g. the item is a shirt, once you have stored the Meta data how will use the information?**

I have listed below the metadata of the item. I will store it for better functionality. I have used an example to demonstrate it better.

The table provides an overview of the metadata attributes typically I will store for an item on Amazon, using a shirt as an example. Each row represents a specific metadata attribute, while the columns describe the attribute name, its purpose, and an example value.

The way I will use the information:

Once I store the metadata for an item, it can be used in a variety of ways to improve the customer experience, manage inventory, and optimize sales. Here I listed some of them:

- **Make products easier to find** by including relevant keywords in the metadata, I can ensure your products show up higher in search results when customers are looking for something specific.
- **Help customers choose the perfect product** with detailed descriptions, accurate sizing information, and high-quality images in the metadata I can provide customers all the details they need to make informed decisions.
- **Show off products in the best light** by compelling product descriptions and captivating images based on the metadata I will make products stand out on the Amazon marketplace.
- **Keep track of my inventory** with stock levels stored in the metadata, I can always know exactly how much of each item I have in stock, preventing overselling and ensuring smooth customer fulfillment.
- **I can understand what's selling well** by reviewing sales rank and review data in the metadata I can identify popular products and areas for improvement. This allows me to optimize pricing, promotions, and potentially even develop new products based on customer preferences.
- **Can Predict future demand** when we analyze historical sales data and product attributes to forecast future demand and adjust inventory levels accordingly. This helps us avoid stockouts and capitalize on peak selling seasons.
- **Personalize the shopping experience** by leveraging the metadata to tailor marketing campaigns and product recommendations based on a customer's past purchases and browsing behavior. This creates a more personalized shopping experience that can lead to increased sales.
- **Gain valuable insights** by analyzing the vast amount of metadata and gain valuable insights into customer behavior and market trends. This knowledge can inform product development strategies and marketing efforts.

The Metadata I will store with an example:

Attribute	Description	Example
ASIN	Amazon Standard Identification Number	B001234567
SKU	Stock Keeping Unit	ABC-SHIRT-123
UPC	Universal Product Code	123456789012
Brand	Name of the shirt's manufacturer	Wrangler
Title	Descriptive product name	Men's Long-Sleeve Western Snap Button Shirt
Description	Detailed information about the shirt	100% cotton, relaxed fit, two chest pockets, pearl snaps. Perfect for casual wear or western-themed events.
Category	Product taxonomy	Clothing > Men's Clothing > Shirts > Western Shirts
Size	Available sizes	S, M, L, XL, XXL
Color	Available color options	Black, Blue Denim
Price	Current selling price	2999
Stock	Available inventory levels	In stock (all sizes)
Sales Rank	Ranking of the shirt compared to other similar products	#234 in Men's Western Shirts
Reviews	Average rating and number of customer reviews	4.5 stars (out of 5) from 1,234 reviews
Material	Fabric composition	100% Cotton
Sleeve Length	Short sleeve, long sleeve, etc.	Long Sleeve
Neck Style	Crew neck, V-neck, etc.	Spread Collar
Fit	Regular fit, slim fit, etc.	Relaxed Fit
Care Instructions	Washing and drying instructions	Machine wash cold, tumble dry low
Search Keywords	Relevant keywords to improve search ranking	western shirt, cowboy shirt, snap button shirt, Wrangler shirt
Images	High-quality product images from various angles	
Dimensions	Measurements of the shirt (e.g., chest width, sleeve length)	Available in a size chart linked in the description
Country of Origin	Where the shirt was manufactured	Made in Vietnam
Warranty Information	Any warranty offered by the manufacturer	Limited lifetime warranty on snaps

4. **Represent below problem in a high-level design diagram.**
   Have a set of 250 users.
   Each user has at least one account with assets, i.e. assets can be stocks or mutual funds
   Each user will see his portfolio real time at any time of the day
   Prices come from different sources.

   - **Design a platform to create, calculate and maintain the portfolios of these users**
   - **Reliably and should scale.**
   - **The portfolios should be updated as soon as the source provides data for the platform.**
   - **The data gets refreshed every 10 mins.**

To design a platform for creating, calculating, and maintaining the portfolios of 250 users with real-time updates and scalability, consider the following architecture:

System Components

Frontend:

- **User Interface**: A web-based interface that allows users to view their portfolios in real-time. This can be built using HTML, CSS, and JavaScript.

Backend:

- **API Gateway**: Handle incoming requests from the frontend and route them to the appropriate services. This can be implemented using a RESTful API.
- **Portfolio Service**: Responsible for creating, updating, and retrieving user portfolios. This service should handle the following tasks:
  - Portfolio Creation: Initialize a new portfolio for each user with the initial assets.
  - Portfolio Update: Update the portfolio with new assets, prices, and other relevant data.
  - Portfolio Retrieval: Retrieve the current portfolio for a user.
- **Price update service:** Collect and aggregate data from various sources (e.g., stock exchanges, financial institutions) to update portfolio values. This service should:
  - Data Collection: Fetch data from different sources at regular intervals (every 10 minutes).
  - Data Processing: Process the collected data to calculate the current portfolio values.
  - Data Storage: Store the processed data in a database for retrieval by the Portfolio Service.
- **Database:**
  - Relational Database: Use a relational database management system like MySQL or PostgreSQL to store user data, portfolio information, and historical data.
  - NoSQL Database: Consider using a NoSQL database like MongoDB for storing large amounts of aggregated data.

System Architecture

- Data Flow:
  - Portfolio Update: The Portfolio Service retrieves the latest data from the database and updates the user portfolios accordingly.
- Scalability:
  - Load Balancing: Use load balancing techniques to distribute incoming requests across multiple instances of the API Gateway and Portfolio Service.
  - Caching: Implement caching mechanisms to reduce the load on the system by storing frequently accessed data.
- Real-Time Updates:
  - Long Polling: Implement long polling techniques to fetch updates from the backend at regular intervals (every 10 minutes).
- Security Considerations
  - Authentication and Authorization: Implement robust authentication and authorization mechanisms to prevent unauthorized access to user portfolios.
  - Data Validation: Validate all user input and ensure that the data is accurate and consistent.

High Level Design Diagram