the organized shop hackerrank solution

The organized shop hackerrank solution is a popular challenge on the HackerRank platform that tests a developer's problem-solving skills, particularly in the context of data structures and algorithms. This challenge simulates a scenario in which a shopkeeper wants to keep track of the items in their store, focusing on how to efficiently organize and retrieve information about the items. In this article, we will explore the problem statement, analyze the requirements, and provide a structured solution, breaking down the components necessary to achieve the task effectively.

Problem Statement

The organized shop problem typically presents a scenario where a list of items is provided, characterized by their names and prices. The goal is to determine how to best organize these items so that they can be efficiently accessed and manipulated. The main objectives are:

1. Organizing Items: Items must be stored in a way that allows for quick retrieval.
2. Handling Queries: The solution should allow for querying the items based on certain criteria, such as name or price.
3. Efficiency: The solution must be efficient in terms of both time and space complexity.

To illustrate, consider the following example:

- Input: A list of items, e.g., `["apple", "banana", "orange", "apple", "banana"]`
- Output: A structured representation of items, such as a dictionary where the keys are item names and the values are their respective counts or prices.

Understanding Requirements

Before diving into the solution, it’s essential to dissect the problem requirements further. The organized shop hackerrank solution can be approached with the following considerations:

Data Structure Selection

Choosing the right data structure is crucial for optimizing performance. Here are several options:

- List: Useful for maintaining order but has slow search times (O(n)).
- Dictionary (HashMap): Provides average O(1) time complexity for lookups, making it ideal for counting occurrences or retrieving prices.
- Sets: Useful for storing unique items but does not maintain order.

Input and Output Format

Understanding the input format and the desired output is critical for developing an efficient solution:

- Input: The first line usually indicates the number of items. Subsequent lines list the items with their names and prices.
- Output: Organized output could include a count of each item or a price list.

Query Handling

The solution should also be able to handle queries effectively. This could involve:

- Finding the price of a specific item.
- Counting the occurrences of an item.
- Listing items based on certain criteria (e.g., price range).

Detailed Solution Approach

Now let’s break down a structured approach to implement the organized shop hackerrank solution.

Step 1: Reading Input

The first step involves reading the input data. Here’s a sample implementation in Python:

```python
def read_input():
n = int(input("Enter number of items: "))
items = []
for _ in range(n):
item = input().strip().split()
items.append((item[0], int(item[1]))) (name, price)
return items
```

Step 2: Organizing Data

Next, we will organize the data using a dictionary:

```python
def organize_items(items):
organized_shop = {}

for item, price in items:
if item in organized_shop:
organized_shop[item]['count'] += 1
else:
organized_shop[item] = {'count': 1, 'price': price}

return organized_shop
```

Step 3: Handling Queries

Once the data is organized, we can handle various queries. Below are examples of how to implement some query functions:

Query 1: Get Price of an Item

```python
def get_price(organized_shop, item_name):
if item_name in organized_shop:
return organized_shop[item_name]['price']
return "Item not found"
```

Query 2: Count of Item Occurrences

```python
def count_item(organized_shop, item_name):
if item_name in organized_shop:
return organized_shop[item_name]['count']
return 0
```

Step 4: Putting It All Together

Now that we have the core functions, we can create a main function to tie everything together:

```python
def main():
items = read_input()
organized_shop = organize_items(items)
```

Example Queries

```
print(get_price(organized_shop, "apple")) Output: price of apple
print(count_item(organized_shop, "banana")) Output: count of bananas

if __name__ == "__main__":
main()
```

Performance Considerations

When evaluating the organized shop hackerrank solution, it's important to consider its performance:

Time Complexity

1. Reading Input: O(n) where n is the number of items.
2. Organizing Items: O(n) since we iterate through the list once.
3. Query Operations: O(1) for retrieval from the dictionary.

Space Complexity

The space complexity is mainly determined by the size of the dictionary used to store items, which is O(m) where m is the number of unique items.

Testing the Solution

To ensure the solution works effectively, it's crucial to test it with various scenarios:

- Basic Functionality: Test with a simple list of items.
- Edge Cases: Consider scenarios with duplicate items, no items, or non-existent queries.
- Performance Testing: Assess how the solution performs with a large number of items.

Example Test Cases

1. Basic Case:

Input:

```
5
apple 100
banana 50
apple 100
orange 30
banana 50
```

Expected Output:

```
Price of apple: 100
Count of banana: 2
```

2. Edge Case:

Input:

```
0
```

Expected Output:

```
Item not found
```

Conclusion

The organized shop hackerrank solution provides an excellent opportunity to practice data handling and algorithmic thinking. By following a structured approach to input reading, data organization, and query handling, developers can create an efficient solution that meets the problem's requirements. The

key takeaway is the importance of selecting the right data structures and understanding the complexity of operations to build an optimal solution. Whether you're preparing for coding interviews or honing your skills, tackling challenges like this one is invaluable in your programming journey.

Frequently Asked Questions

What is the main objective of the 'Organized Shop' problem in HackerRank?

The main objective is to determine the maximum number of items that can be purchased from a shop with given constraints on prices and available budget.

What are the key parameters required to solve the 'Organized Shop' problem?

Key parameters include the list of item prices, the total budget available for purchases, and possibly the maximum number of items that can be bought.

Which algorithmic approach is commonly used to solve the 'Organized Shop' problem?

A greedy algorithm approach is commonly used, where items are sorted by price and the algorithm iteratively purchases the cheapest items until the budget is exhausted.

How can edge cases, such as an empty item list or a budget of zero, affect the solution to the 'Organized Shop' problem?

Edge cases like an empty item list should return zero items purchased, while a budget of zero means no items can be bought regardless of the item prices.

What data structures can be useful in implementing the solution for the 'Organized Shop' problem?

An array or list can be used to store item prices, and a sorting algorithm can be applied to arrange the prices in ascending order for efficient purchasing.

Are there any optimizations that can be made for the 'Organized Shop' problem solution?

Yes, optimizations can include early termination if the remaining budget is less than the lowest priced item, and using efficient sorting algorithms to minimize time complexity.

The Organized Shop Hackerrank Solution

The Organized Shop Hackerrank Solution

Related Articles

- the nourishing traditions of baby child care
- the life cycle of amphibians
- the martian andy weir

Back to Home