investable periods hackerrank solution

investable periods hackerrank solution is a common challenge faced by many aspiring programmers and data scientists on the HackerRank platform. This problem typically involves finding the maximum number of non-overlapping intervals from a given set of intervals, each with a start and end time. Understanding how to efficiently solve the "Investable Periods" problem requires a grasp of interval scheduling algorithms. This article will delve into the intricacies of the investable periods HackerRank challenge, providing a comprehensive guide to understanding the problem, exploring efficient algorithmic approaches, and detailing a robust HackerRank solution. We will cover greedy strategies, dynamic programming concepts, and provide practical coding examples to help you conquer this frequently encountered algorithmic puzzle.

Introduction to Investable Periods

The "Investable Periods" problem on HackerRank is a classic example of an interval scheduling problem. It tests a candidate's ability to apply algorithmic thinking to optimize resource allocation, in this case, time. The core objective is to select the largest possible subset of activities (or "investable periods") such that no two selected activities overlap. This problem has direct applications in various real-world scenarios, from scheduling meetings and tasks to optimizing flight routes and managing project timelines. Mastering this problem not only enhances your problem-solving skills but also equips you with foundational knowledge for more complex algorithmic challenges.

Understanding the Investable Periods Problem

At its heart, the Investable Periods problem presents you with a list of intervals, where each interval is defined by a start time and an end time. You are tasked with choosing a maximum number of these intervals such that no

two chosen intervals share any common time. For instance, if you have an interval from 2 to 5 and another from 4 to 7, they overlap. If you have an interval from 2 to 5 and another from 5 to 7, they are considered non-overlapping (or touching at an endpoint, which is usually permissible in these problems). The goal is to maximize the count of selected non-overlapping intervals.

Input Format for Investable Periods

Typically, the input for the Investable Periods problem on HackerRank will consist of an integer 'n', representing the number of intervals, followed by 'n' lines, each containing two integers: the start time and the end time of an interval. The order of these intervals in the input might not be significant, but the start and end times themselves are crucial for determining overlaps.

Output Format for Investable Periods

The expected output is a single integer: the maximum number of non-overlapping intervals that can be selected from the given set.

Key Concepts for Solving Investable Periods

To effectively tackle the Investable Periods problem, understanding certain algorithmic concepts is paramount. These concepts form the bedrock of efficient solutions and allow you to devise strategies that scale well with larger inputs.

Interval Scheduling and its Variations

The Investable Periods problem falls under the broader category of interval scheduling. Variations of this problem might include weighted intervals (where each interval has a value) or finding the minimum number of resources needed to schedule all intervals. Understanding the basic interval scheduling problem provides a solid foundation for tackling these more complex variations.

Greedy Algorithms

Greedy algorithms make the locally optimal choice at each step with the hope of finding a global optimum. For interval scheduling problems, greedy strategies often prove to be very effective and efficient.

Dynamic Programming

Dynamic programming is an algorithmic technique that breaks down a problem into smaller overlapping subproblems, solves each subproblem once, and stores their solutions to avoid recomputation. While a greedy approach is often sufficient for the basic Investable Periods problem, dynamic programming can be used for more complex variations.

Greedy Approach to Investable Periods

The most intuitive and often the most efficient way to solve the Investable Periods problem is by employing a greedy strategy. The core idea behind the greedy approach is to make a choice that seems best at the moment without considering future consequences, and this often leads to the globally optimal solution for this specific problem.

The "Finish Time" Greedy Strategy

The most common and successful greedy strategy for Investable Periods involves sorting the intervals based on their finish times. The intuition here is to pick the interval that finishes earliest. By selecting the interval that finishes soonest, you leave the maximum amount of time available for subsequent intervals, thereby maximizing your chances of fitting in more non-overlapping intervals.

Steps for the Greedy Approach

1. Sort all the given intervals in ascending order based on their finish times.
2. Select the first interval (which has the earliest finish time) and add it to your set of chosen intervals.
3. Iterate through the remaining sorted intervals. For each interval, if its start time is greater than or equal to the finish time of the last selected interval, then select this interval and add it to your set. Update the "last selected interval's finish time."
4. The total count of selected intervals is the maximum number of non-overlapping investable periods.

Why the Greedy Approach Works

The greedy strategy of sorting by finish time is proven to be optimal for the unweighted interval scheduling problem. Consider any optimal solution. If it doesn't pick the interval with the earliest finish time first, you can always swap the first interval in the optimal solution with the interval that finishes earliest without creating any new overlaps, and without reducing the total number of intervals. This process can be repeated, demonstrating that a greedy choice leads to an optimal solution.

Dynamic Programming Approach to Investable Periods

While the greedy approach is sufficient for the standard Investable Periods problem, understanding how dynamic programming can solve it is also valuable, especially for variations or as a stepping stone to more complex problems.

Defining Subproblems

Let's sort the intervals by their finish times first. We can define a DP state `dp[i]` as the maximum number of non-overlapping intervals we can select from the first `i` intervals (after sorting).

Recurrence Relation

To calculate `dp[i]`, we have two choices for the i-th interval:

- **Exclude the i-th interval**: In this case, the maximum number of intervals is `dp[i-1]`.
- **Include the i-th interval**: If we include the i-th interval, we need to find the latest interval `j` (where `j < i`) such that interval `j` finishes before interval `i` starts. The number of intervals in this case would be `1 + dp[j]`.

So, the recurrence relation becomes: `dp[i] = max(dp[i-1], 1 + dp[j])`, where `j` is the index of the latest interval that does not overlap with the i-th interval. If no such `j` exists, then `dp[i] = max(dp[i-1], 1)`.

Base Case

The base case would be `dp[0] = 0` (no intervals, no selection).

HackerRank Solution Walkthrough

Let's walk through a typical implementation of the greedy approach for the Investable Periods problem on HackerRank. This will involve reading input, sorting, and applying the greedy selection logic.

Implementing the Greedy Strategy in Code

Most programming languages offer easy ways to sort data structures. You would typically store the intervals as pairs or tuples and then sort them.

Example Python Implementation (Conceptual)

Consider an input like:

3

1 4

2 5

5 7

The intervals are (1, 4), (2, 5), and (5, 7).

1. **Sort by finish time**: (1, 4), (2, 5), (5, 7)
2. **Select first**: (1, 4). Count = 1. Last finish time = 4.

3. **Consider (2, 5):** Start time 2 is NOT >= 4. Skip.

4. **Consider (5, 7):** Start time 5 IS >= 4. Select (5, 7). Count = 2. Last finish time = 7.

Final count = 2.

Handling Input and Output

Your code will need to correctly parse the input from standard input, likely reading the number of intervals and then the start and end times for each. The final result, the maximum count, should be printed to standard output.

Optimizing the Investable Periods Solution

While the greedy approach is already quite efficient, there are nuances to consider for optimal performance, especially in competitive programming contexts.

Time Complexity Analysis

The dominant factor in the greedy approach is the sorting step. If there are 'n' intervals, sorting takes O(n log n) time. The subsequent iteration to select non-overlapping intervals takes O(n) time. Therefore, the overall time complexity of the greedy solution is O(n log n).

Space Complexity Analysis

The space complexity is primarily determined by the storage of the intervals and any auxiliary space used during sorting. Typically, this is O(n) if you need to store all intervals, or O(1) if sorting can be done in-place and the input is processed iteratively.

Common Pitfalls and How to Avoid Them

When solving Investable Periods on HackerRank, certain mistakes are frequently made. Being aware of these can save you a lot of debugging time.

Incorrect Sorting Order

A common mistake is not sorting by finish times. Sorting by start times, for example, will not yield the optimal solution for this problem. Always prioritize sorting by finish times.

Off-by-One Errors

Carefully handle the comparison between the current interval's start time and the previous interval's finish time. Ensure you correctly implement the "greater than or equal to" condition to allow intervals that touch at their

endpoints.

Handling Empty Input

Your solution should gracefully handle cases where no intervals are provided (n=0). In such scenarios, the result should be 0.

Beyond Investable Periods: Related Algorithmic Problems

The skills honed by solving the Investable Periods problem are transferable to many other algorithmic challenges.

Maximum Number of Activities

This is essentially the same problem, often framed in terms of selecting the maximum number of activities that can be performed by a single person given their start and finish times.

Weighted Interval Scheduling

Here, each interval has an associated weight or value, and the goal is to find a non-overlapping set of intervals that maximizes the total weight. This problem typically requires dynamic programming.

Meeting Room Scheduling

This involves determining the minimum number of conference rooms required to schedule a given set of meetings. It's another variation where understanding interval overlaps is key.

Frequently Asked Questions

What is the typical time complexity of a brute-force approach to the Investable Periods problem?

A brute-force approach, which involves checking every possible subarray, would typically have a time complexity of O(n^3), where 'n' is the number of days. This is because you'd iterate through all possible start and end points (O(n^2)) and then sum the values within each subarray (O(n)).

How can a sliding window technique optimize the Investable Periods problem?

A sliding window technique can optimize the Investable Periods problem by maintaining a window of days and adjusting its start and end points. This

allows you to efficiently calculate the sum of the current window and check if it meets the criteria, often reducing the complexity to O(n).

What is the core idea behind Kadane's algorithm and its relevance to Investable Periods?

Kadane's algorithm is primarily used for finding the maximum subarray sum. While not directly solving Investable Periods (which has a target sum constraint), the concept of maintaining a running sum and resetting it when it becomes detrimental can be adapted. For Investable Periods, you might adapt it to track the current sum within the window and update counts based on whether the sum exceeds the target.

What data structures are commonly used in efficient solutions for Investable Periods?

Common data structures include prefix sums arrays to quickly calculate subarray sums, and often a simple integer variable to track the current sum within a sliding window. For certain variations or more complex constraints, a hash map or dictionary might be used to store counts of prefix sums.

What are the common edge cases or constraints to consider when solving Investable Periods?

Key edge cases include empty input arrays, arrays with all negative numbers, and cases where no valid investable periods exist. Constraints might involve the range of values in the input array (positive, negative, or both) and the size of the array itself.

How does the target sum (K) influence the choice of algorithm for Investable Periods?

The target sum (K) is crucial. If K is a fixed positive value, techniques like sliding window with two pointers are very effective. If K can vary or the problem involves finding periods with a sum equal to K, then techniques involving prefix sums and hash maps to count occurrences of `prefix_sum[i] - K` become relevant.

Are there any variations of the Investable Periods problem, and how do they affect the solution?

Yes, variations exist. For instance, finding the longest investable period, or finding the number of investable periods with a sum greater than or equal to K. These variations would require modifications to the core logic, perhaps by tracking the maximum length or adjusting the conditions within the sliding window or prefix sum approach.

Additional Resources

Here are 9 book titles related to "investable periods hackerrank solution," with each title starting with "" *and using only the "" tag for emphasis:*

1. The Art of Algorithmic Trading

This book delves into the foundational principles of building trading algorithms. It covers essential concepts like technical indicators, order execution, and risk management. Readers will learn how to design and implement strategies that can identify profitable trading opportunities within specific timeframes, a key element for solving problems like the Investable Periods challenge.

2. Quantitative Finance For Dummies
A comprehensive yet accessible introduction to the world of quantitative finance. It breaks down complex mathematical and statistical concepts relevant to financial markets into understandable terms. The book equips beginners with the knowledge to grasp market dynamics and develop systematic approaches to investment strategies.

3. Python For Finance: Mastering Data-Driven Finance
This practical guide focuses on using the Python programming language for financial analysis and trading. It explores libraries like Pandas and NumPy for data manipulation and explores how to implement quantitative models. It's an excellent resource for anyone wanting to translate algorithmic ideas into working code for financial applications.

4. High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Execution
While more advanced, this book offers insights into the speed and efficiency required for modern trading. It discusses strategies that capitalize on short-term market movements and the technical infrastructure needed. Understanding these concepts can inform the optimization of solutions for identifying optimal trading intervals.

5. Time Series Analysis For Machine Learning
This book provides a thorough grounding in analyzing data that changes over time, a fundamental aspect of financial markets. It covers various statistical and machine learning techniques for forecasting and pattern recognition. Mastering time series analysis is crucial for identifying trends and anomalies within potential investable periods.

6. Financial Markets and Trading: A Quantitative Approach
This title explores the mathematical underpinnings of financial markets and trading systems. It examines how to model market behavior and develop strategies based on quantitative analysis. The book's focus on systematic approaches directly relates to solving problems that require identifying structured opportunities.

7. Algorithmic Trading Strategies: Development of Robust Trading Systems
This book concentrates on the practical development and validation of trading algorithms. It emphasizes building systems that are resilient to changing market conditions. The methodologies discussed are directly applicable to creating robust solutions for the Investable Periods problem.

8. Introduction To Algorithmic Trading and Trading Systems
This introductory text provides a broad overview of the field of algorithmic trading. It covers the essential components of a trading system, from idea generation to backtesting. The principles laid out are foundational for tackling problems that involve automated decision-making in markets.

9. Python And Algorithmic Trading For Investing
This book bridges the gap between Python programming and practical investment strategies. It demonstrates how to leverage Python for analyzing market data and implementing trading ideas. For HackerRank challenges like Investable

Periods, this resource offers the practical coding skills needed to build and test solutions.

Investable Periods Hackerrank Solution

Investable Periods Hackerrank Solution

Related Articles

- intermediate accounting solutions manual spiceland
- introduction to management science 13th edition
- ionic bonding worksheet with answers

Back to Home