1105. Filling Bookcase Shelves

Medium Array Dynamic Programming Leetcode Link

Problem Description

In this problem, we're given an array `books` where each `books[i]` represents the `ith` book with its thickness and height as `[thicknessi, heighti]`. We also have a bookshelf of a fixed width, `shelfWidth`. The goal is to arrange the books in such a way that they all fit on the shelves while maintaining their original order and minimizing the total height of the bookshelf.

The process for placing books onto the bookshelf is as follows:

- Books are taken from the array in the same order they are given and placed onto the shelf.
- A group of books can be placed on the same shelf if their total thickness does not exceed the shelf width.
- Once a shelf is filled (or can't fit the next book without exceeding `shelfWidth`), a new shelf is started above it.
- The height of each shelf is determined by the tallest book on that shelf.
- The process continues until all books are placed on the shelf.
- The total height of the bookshelf is the sum of the heights of each individual shelf.

The challenge is to determine the least total height of the bookshelf after all the books are placed according to the rules.

Intuition

For the solution, dynamic programming is used to keep track of the optimal height at each step. The key idea is to iteratively determine the minimum height required to arrange all books up to the current one. Here's a step-by-step breakdown:

1. Initialize a dynamic programming array `f` of `n + 1` elements, where `n` is the number of books.
2. For each book, calculate the minimum height if this book starts a new shelf by adding its height to the minimum height found for all previous books.
3. Then, for the same book, consider the possibility of placing it on the same shelf as the previous book(s). For each cluster of books that can fit together on the same shelf without exceeding `shelfWidth`, calculate the minimum height needed if these books shared a shelf.
4. To do this, iterate backwards from the current book and keep adding the thicknesses of each previous book until the total thickness exceeds `shelfWidth`.
5. As you go along, keep track of the maximum height of the books in the current shelf arrangement since that determines the height of the shelf.
6. Update the minimum height for the current set of books as the lesser of the existing minimum height or the height required if these books share a shelf, which is the sum of maximum book height on this shelf and the minimum height found before starting this shelf.
7. After calculating for all books, the last element in the `f` array will contain the minimum possible height of the bookshelf.

The optimization comes from recognizing that the best height for a set of books up to the ith book is either on a new shelf or combined with some recent books on the same shelf to minimize the height added. By comparing each possibility, we guarantee we're finding the optimal combination at each step.

Solution Approach

The solution provided uses dynamic programming as its primary algorithmic strategy. This involves breaking a larger problem down into smaller subproblems and storing the results of those subproblems to avoid redundant computations. Here's how the algorithm works step by step:

- An array `f` of length `n + 1` is initialized with zeroes, where `n` is the number of books. The array `f` will hold the minimum height of the bookshelf when the last book placed is the `i`th book (`f[i]`).

- We loop through each book, and for each book `(w, h)`:

  - We set `f[i]` to `f[i - 1] + h` by default, assuming the book `(w, h)` starts a new shelf, where `i` is the current book's index in one-based notation. This means we add the height of this book to the total height we had before adding this book.
  - We then consider the possibility of placing each book in reverse order onto the same shelf, starting by adding the second last book and so on, until we either run out of books or the total thickness would exceed the shelfWidth. For each combination, we update the current maximum height of the books on the shelf and check if the minimum total height `f[i]` can be improved by placing the current book on the same shelf as the previous one(s).
- Within the inner loop, we are performing the following computations:

  - Increment the total thickness `w` by the thickness of the previous book (starting from the current book and moving backwards).
  - If `w` exceeds `shelfWidth`, we break out of the loop as we can't add any more books to the current shelf.
  - Otherwise, we update the current maximum height `h` to be the maximum of the current `h` and the height of the book we're trying to add to the shelf.
  - Then, we calculate the minimum total height `f[i]` by comparing its current value with the sum of the maximum shelf height `h` and the minimum height `f[j - 1]` that can be achieved before adding the books currently considered on the shelf. We then select the minimum of those two values.
- This loop continues for all books, and once we have considered all possible placements for each book, the last element of `f`, `f[n]`, holds the minimum total height of the bookshelf, which is the final answer.

The data structure used is a simple array (`f`), which holds the best solution to subproblems (best bookshelf height given a certain number of books). By iterating over the possible placements for each book, we are effectively employing a bottom-up dynamic programming approach.

In summary, the algorithm builds upon previous solutions to maintain the minimum height required at each step, while considering new book placements, to achieve an overall optimal arrangement with respect to the total height.

Example Walkthrough

Let's illustrate the solution approach with a small example. Suppose we have an array of books:

```
books = [[1, 1], [2, 3], [2, 3], [1, 2]], shelfWidth = 4
```

and we want to arrange the books on a bookshelf with a shelf width of `4`. We follow the solution approach using dynamic programming.

1. We initialize an array `f` with length `n + 1` where `n` is the number of books. In our case, `n=4`, so `f = [0, 0, 0, 0, 0]`. This array will store the minimum total height after the `i`th book has been placed.

2. Start with the first book `[1, 1]`. If it starts a new shelf, the total height is just the height of this book, so `f[1] = 1`.

3. Now place the second book `[2, 3]`. If it starts a new shelf, the minimum total height after this book is `f[1] + height(second book) = 1 + 3 = 4`, so `f[2] = 4`. However, since its thickness (2) plus the thickness of the first book (1) does not exceed the shelfWidth (4), we also check if we can place it on the same shelf as the first book. This would keep the shelf height as the maximum height of the first two books, which is `3`. So, `f[2]` remains `3` since `3` < `4`.

4. Now consider the third book `[2, 3]`. If it starts a new shelf, then the minimum total height is `f[2] + height(third book) = 3 + 3 = 6`. But can it fit with the second book on the same shelf? The total thickness would be `2(second book) + 2(third book) = 4`, which is equal to `shelfWidth`, and the height would be the taller book's height, which is `3`, so the total height is still `3 + 3 = 6`. Now, can we fit the first and third books together instead? The total thickness would be `1(first book) + 2(third book) = 3`, which is less than `shelfWidth`, and we find the max height among the first and third books to be `3`. So, the minimum height becomes `f[3] = 1(original height of the first shelf with only the first book) + 3(height of the third book) = 4`. Since `4` < `6`, we update `f[3] = 4`.

5. Finally, the fourth book `[1, 2]`. Starting a new shelf would result in `f[3] + height(fourth book) = 4 + 2 = 6`. Can it fit on the same shelf as the third? No, because `2(third book) + 1(fourth book) = 3`, and if we include the second book, the total thickness `2(second book) + 2(third book) + 1(fourth book) = 5` exceeds the shelf width. So, it must start a new shelf. As the fourth book is shorter than the third, it doesn't alter our already calculated `f[3]`. However, we check if the first and fourth book can share a shelf, and they can as their combined thickness is `2`, and the shelf height will be determined by the fourth book `2`. So, `f[4]` would be `1(height of the first shelf) + 2(height of the fourth book) = 3`. The total minimum height `f[4]` is `3`.

So, by following the dynamic programming approach, we've determined that the books can be arranged in the smallest total height which is `3`. The arrangement would be: first and fourth book on the bottom shelf and second and third books on the shelf above.

Python Solution

```python
class Solution:
    def minHeightShelves(self, books: List[List[int]], shelf_width: int) -> int:
        # Number of books
        num_books = len(books)

        # List to store the minimum height at each point
        min_height = [0] * (num_books + 1)

        # Loop through each book
        for i, (width, height) in enumerate(books, 1):
            # Initially, put the book on a new shelf
            min_height[i] = min_height[i - 1] + height

            # Initialize the total width of the current shelf
            total_width = width

            # Try placing previous books on the same shelf and check
            # if it can minimize the total height of the shelves
            for j in range(i - 1, 0, -1):
                # Add the width of the book j to the total width
                total_width += books[j - 1][0]

                # If the total width exceeds the shelf width, we can't place more books on the same shelf
                if total_width > shelf_width:
                    break

                # Update the height of the current shelf to be the maximum book height on the shelf
                height = max(height, books[j - 1][1])

                # Update the minimum height if placing the book on the current shelf results in less height
                min_height[i] = min(min_height[i], min_height[j - 1] + height)

        # After placing all the books, the last element in min_height stores the minimized total height
        return min_height[num_books]

```

Java Solution

```java
class Solution {
    public int minHeightShelves(int[][] books, int shelfWidth) {
        int numOfBooks = books.length;
        // 'dp' array will store the minimum height for the first 'i' books
        int[] dp = new int[numOfBooks + 1];

        for (int i = 1; i <= numOfBooks; ++i) {
            // Initial width and height for the current book
            int currentWidth = books[i - 1][0];
            int currentHeight = books[i - 1][1];

            // Place current book on a new shelf
            dp[i] = dp[i - 1] + currentHeight;

            // Try placing previous books with the current one on the same shelf
            for (int j = i - 1; j > 0; --j) {
                currentWidth += books[j - 1][0]; // Accumulate width

                // If accumulated width exceeds shelfWidth, stop trying to place more books
                if (currentWidth > shelfWidth) {
                    break;
                }

                // Update the current height to be the tallest book in the current placement
                currentHeight = Math.max(currentHeight, books[j - 1][1]);
                // Calculate the minimum height if this set of books were on the same shelf,
                // comparing with the previous minimum height and updating it if necessary
                dp[i] = Math.min(dp[i], dp[j - 1] + currentHeight);
            }
        }

        // Return the minimum height to place all the books
        return dp[numOfBooks];
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    int minHeightShelves(vector<vector<int>>& books, int shelfWidth) {
        // Total number of books
        int numBooks = books.size();

        // Initialize the dynamic programming array which will store the minimum height for the first i books
        vector<int> dp(numBooks + 1);

        // Base case: no books means the shelves have a height of 0
        dp[0] = 0;

        // Loop through each book
        for (int i = 1; i <= numBooks; ++i) {
            // Width and height of the current book
            int currentWidth = books[i - 1][0], currentHeight = books[i - 1][1];

            // By default the minimum height after adding this book is its height plus the height of previous shelves without this b
            dp[i] = dp[i - 1] + currentHeight;

            // Try to place previous books on the same shelf with the current book to minimize the total height
            for (int j = i - 1; j > 0; --j) {
                // Accumulate the width with the previous book
                currentWidth += books[j - 1][0];

                // Break the loop if adding another book exceeds the shelf width
                if (currentWidth > shelfWidth) {
                    break;
                }

                // Update the current shelf's height based on the tallest book on this shelf
                currentHeight = max(currentHeight, books[j - 1][1]);

                // Determine the minimum height if this shelf configuration is used (place as many books as possible on the current s
                dp[i] = min(dp[i], dp[j - 1] + currentHeight);
            }
        }

        // The last element in dp will have the minimum height by placing all books
        return dp[numBooks];
    }
};

```

Typescript Solution

```typescript
function minHeightShelves(books: number[][], shelfWidth: number): number {
    const numBooks = books.length;
    // Array 'dp' to store the minimum height at each index
    const dp = new Array(numBooks + 1).fill(0);

    for (let i = 1; i <= numBooks; ++i) {
        let [currentWidth, currentHeight] = books[i - 1];
        // Initialize the height of the current shelf
        dp[i] = dp[i - 1] + currentHeight;

        // Iterate through the previous books to find minimum height for the current shelf
        for (let j = i - 1; j > 0; --j) {
            // Accumulate width of books on the current shelf
            currentWidth += books[j - 1][0];
            if (currentWidth > shelfWidth) {
                // If width exceeds the shelf width, stop the loop
                break;
            }

            // Calculate the max height on the current shelf
            currentHeight = Math.max(currentHeight, books[j - 1][1]);
            // Update 'dp' with the minimum height by comparing it with the height of the previous arrangement plus current shelf hei
            dp[i] = Math.min(dp[i], dp[j - 1] + currentHeight);
        }
    }

    // Return the minimum height needed to place all books
    return dp[numBooks];
}

```

Time and Space Complexity

The given Python function calculates the minimum height of a shelf that can contain all the books, given a list of book dimensions and the width of the shelf. It uses dynamic programming to solve the problem.

Time Complexity:

The outer loop iterates over each book (`n` iterations, where `n` is the number of books). Inside the outer loop, there is an inner loop that iterates backward from the current book index `i` to 1. This inner loop runs at most `i` times, and since it decreases progressively, it results in a series that sums up roughly to `n(n + 1)/2` in the worst case. The operations within the inner loop are constant-time operations.

Consequently, the overall time complexity is `O(n^2)`.

Space Complexity:

For space, an array of size `n + 1` is used to store the running minimum heights, where `n` is the number of books. No other space-consuming structures are used. Thus, the space complexity is linear.

Hence, the space complexity is `O(n)`.