Dynamic Programming - Knapsack Problem

Topics: #dynamic-programmming #algorithms
LinkedIn: Md. Ziaul Karim
Find me here:

Problem Statement: Visiting London

Suppose you are going to London for a nice vacation. You have two days there and a lot of things you want to do. You can't do everything, so you make a list.

Attraction	Duration (days)	Rating
WESTMINSTER ABBEY	0.5	7
GLOBE THEATER	0.5	6
NATIONAL GALLERY	1	9
BRITISH MUSEUM	2	8
ST. PAUL'S CATHEDRAL	0.5	8

Your task is to determine the most effective strategy for utilizing your available time to maximize the overall value, taking into account the ratings of the places you intend to visit.

Note: This is an exercise from *Chapter 9: Dynamic Programming* of a book titled: *Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People* *by Aditya Y. Bhargava.*

*Turn to the next page

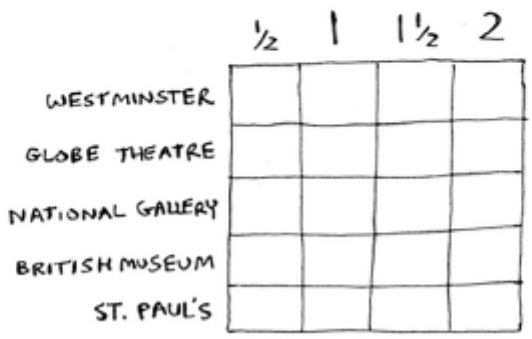
Explanation:

Taking a page directly from the book:

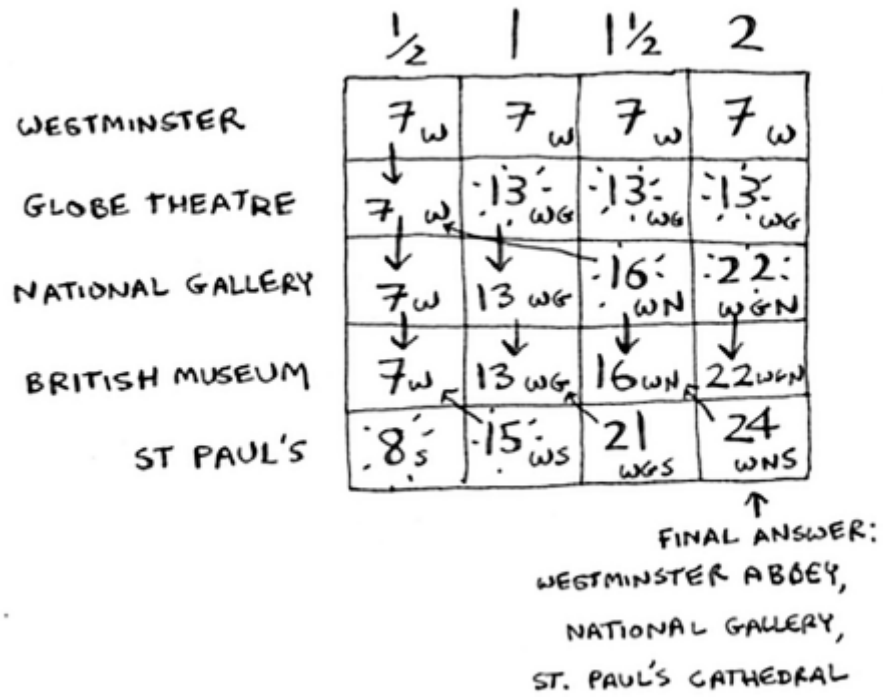
For each thing you want to see, you write down how long it will take and rate how much you want to see it. Can you figure out what you should see, based on this list?

It's the knapsack problem again! Instead of a knapsack, you have a limited amount of time. And instead of stereos and laptops, you have a list of places you want to go. Draw the dynamic-programming grid for this list before moving on.

Here's what the grid looks like.



Did you get it right? Fill in the grid. What places should you end up seeing? Here's the answer.



*Turn to the next page

Solution

```
def maximize_rating(attractions: list, available_time:float)->tuple:

    n = len(attractions)  # Number of attractions

    # Create a 2D table to store the maximum rating for different time
    constraints with each cell containing (rating: int, place(s): str)

    dp = [[(0.0,'')] * (int(available_time * 2) + 1) for _ in range(n + 1)]
    # Now we visit all the cells in the table
    for i in range(1, n + 1): #rows

        for t in range(1, int(available_time * 2) + 1): #columns

            place, duration, rating = attractions[i - 1]

            if duration <= t / 2:
                candidates1 = dp[i - 1][t] #previous

                candidates2 = (
                dp[i - 1][int(t - 2 * duration)][0]+rating,
                (
                dp[i - 1][int(t - 2 * duration)][1] +", "+ place
                ).strip(", ")) #current

                dp[i][t] = max(candidates1,candidates2)

            else:
                dp[i][t] = dp[i - 1][t]

    return dp[n][-1] #bottom right corner of the table

if __name__ == "__main__":
    # Define the attractions and available time
    # This is the format: (place:str, duration:float, rating:int)
    attractions = [
        ("WESTMINSTER ABBEY", 0.5, 7),
        ("GLOBE THEATER", 0.5, 6),
        ("NATIONAL GALLERY", 1, 9),
        ("BRITISH MUSEUM", 2, 8),
        ("ST. PAUL'S CATHEDRAL", 0.5, 8),
    ]

    available_time = 2.0  # 2 days

    # Find the optimal set of attractions and their total rating
    max_rating, places_selected = maximize_rating(attractions, available_time)
    print(f"Max Rating: {max_rating} \nOptimal set of Places to visit:\n{places_selected}")
```

Output

```
Max Rating: 24.0
Optimal set of Places to visit:
WESTMINSTER ABBEY, NATIONAL GALLERY, ST. PAUL'S CATHEDRAL
```