Hands On 3

Competitive Programming and Contests

Leonardo Crociani (Mat. 615392)

Contents

1 Hands On 3

1.1 Introduction

I have implemented solutions for both problems using dynamic programming. Initially, I designed a recursive algorithm which was quite challenging and took me several hours. However, since the time complexity of the recursive solution was incorrect, I switched to dynamic programming.

1.2 Holiday Planner

To develop the dynamic programming version, I first determined the base cases. For a single city, the maximum attraction that can be visited is simply the sum of all the attractions in that city.

For `n` cities, once the optimal solution for the previous `n-1` cities is known, the optimal solution for the `d-th` day in the `n-th` city can be found by considering the maximum of the combinations: `x` days in the current city plus `d-x` days in the optimal solution for the previous cities. At the end of the `d-th` day computation for the `n-th` city, we should compare the outcome with the previous optimal day, to determine whether it would be better to use that solution instead.

The memoization matrix, indexed by `i-th city` and `j-th day`, has a dimension of:

$$O(n * D) \tag{1}$$

Where `n` is the number of cities and `D` is the number of days. To insert a new item in the matrix, we need to scan a row, which is `D` days long.

So the overall time complexity is:

$$O(n * D^2) \tag{2}$$

1.3 Design a Course

For this problem, the approach used to design the efficient solution has been the following:

- Sort the topics on increasing order of beauty. In case two topics have the same beauty, they are sorted based on the decreasing order of difficulty.
- Reduce the problem to a `LIS` calculation on the difficulty. The previous sorting ensures that the beauty is increasing while we select increasingly difficult topics. We can use the efficient solution we have seen in class for this step.

Both the sorting and the LIS calculation take

$$O(n * log(n)) \tag{3}$$

That is also the overall time complexity of the solution.

1.4 Source code and tests

In the .zip I've provided, you can find `main.rs` (used for testing) and `lib.rs` (the actual implementation of the data structures and algorithms) in `/hands-on/3/`. To run the tests, open a terminal in the code folder and execute `cargo run`. Please note that you will need to initialize the code folder by running `cargo new <any_name>` first. The code expects to find a folder named `tests` with two subfolders inside: `1` and `2`. Each of these subfolders should contain the pairs of .txt test files.

1.5 References

The Longest Increasing Subsequence implementation takes strong inspiration by Tushar Roy.