Design and Analysis of Algorithms — Lab

R S Milton, C Aravindan, T T Mirnalinee
Department of CSE, SSN College of Engineering

Session 3: Design and Analysis
8 January 2020

1 Out-of-Order Rankings

Two visitors, A and B to Chennai Book Fair rank a set of books. How do we quantify the similarity (or dissimilarity) between the rankings given by the two visitors? If the rankings are exactly the same, then the dissimilarity measure should be low. When they are exactly the opposite (one ranking is the reverse of the other), the dissimilarity must be high. In other cases, the measure should be a smooth interpolation between these two extremes.

We will describe a method of quantify the dissimilarity. If A ranks a book as i, but B ranks that book as j, then $B[j] = i$. Example: Suppose there are 5 books, and `B = [3, 1, 2, 5, 4]`. B's first-ranked book is A's third-ranked book, B's second-ranked book is A's first-ranked book, and so on.

We will measure dissimilarity as the number of pairs that are out of order. In our example, the dissimilarity is 3, since there are three pairs (3, 1), (3, 2), (5, 4) that are out of order. The number of out-of-order pairs provide a measure that smoothly interpolates between complete agreement (0 pairs) and complete disagreement ($\binom{n}{2}$).

In an array $A[1 : n]$, a pair of indices (i, j) is out-of-order if $i < j$ and $A[i] > A[j]$. The number of out-of-order pairs in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward). Given an array of n distinct numbers, count the number of out-of- the-order pairs in that sequence.

Example:

Array	Number of out-of-order pairs
1, 2, 3, 4, 5	0
2, 4, 1, 3, 5	3
3, 1, 2, 5, 4	3
4, 5, 2, 1, 3	7
5, 4, 3, 2, 1	10

1. Design an algorithm using brute-force/exhaustive enumeration of all possible pairs of labels. Implement the algorithm in Python.
2. Analyse the time complexity the algorithm and express it in asymptotic notation.
3. (OPTIONAL) Perform empirical analysis of the program to verify the time complexity.
4. Design an improved version (time complexity should be less) using any of the design techniques discussed so far. Implement the algorithm in Python.
5. Analyse the time complexity of the improved algorithm and express it in asymptotic notation.
6. (OPTIONAL) Perform empirical analysis to verify the time complexity of the improved algorithm.