Analysis of Algorithms

BLG 335E

Project 2 Report

Atahan ÇOLAK
colakat21@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 26.11.2024

1. Implementation

1.1. Sort the Collection by Age Using Counting Sort

Counting Sort is a sorting technique that does not rely on comparisons, making it highly efficient when sorting integers within a limited range. In this implementation:

- I identified the maximum age in the dataset, which allowed me to initialize a counting array.
- The frequency of each age was recorded, and cumulative sums were used to determine the correct positions for each element in the sorted output.
- The algorithm has a time complexity of $O(n + k)$, where n represents the number of elements to sort and k is the range of the age values.
- I performed sorting in both ascending and descending orders by adjusting the placement of elements based on their counts.

The benchmarking results indicate good performance even with larger datasets:

- **Items_l**: 0.003189 seconds (ascending), 0.003139 seconds (descending)
- **Items_m**: 0.001292 seconds (ascending), 0.001295 seconds (descending)
- **Items_s**: 0.000544 seconds (ascending), 0.000595 seconds (descending)
- **Items_txt**: 0.00054 seconds (ascending), 0.000611 seconds (descending)

1.2. Calculate Rarity Scores Using Age Windows (with a Probability Twist)

The rarity score for each item was calculated based on its age and a probability factor related to similar items within a defined age window. The formula used is:

$$R = (1 - P)\left(1 + \frac{\text{age}}{\text{agemax}}\right)$$

where P represents the probability of finding similar items within the window, and $agemax$ is the maximum age value in the dataset. Items with lower P values are considered more rare, as they are less likely to be duplicated in the same age range.

1.3. Sort by Rarity Using Heap Sort

To sort the items by their rarity scores, I used Heap Sort due to its efficiency with a time complexity of $O(n \log n)$. The process involved:

- Constructing a max heap where the item with the highest rarity score would be at the root.
- Extracting the maximum element one by one and then maintaining the heap property by reordering the remaining elements.
- This method effectively ordered the items with the most rare ones appearing first.

The following are the benchmark results for the Heap Sort implementation:

- **Items_l**: 0.025931 seconds
- **Items_m**: 0.009309 seconds
- **Items_s**: 0.004141 seconds
- **Items_txt**: 0.004088 seconds

1.4. Defining & Analyzing Different Rarity Score Calculations

Various methods exist for determining rarity scores, each offering distinct advantages depending on the context. Below, I explore different approaches to calculating rarity and assess their potential impact on sorting and analysis.

1.4.1. Initial Rarity Calculation Approach

The primary rarity score used in this project is:

$$R = (1 - P)\left(1 + \frac{\text{age}}{\text{agemax}}\right)$$

This approach combines the probability of an item being similar to others in the specified window with its relative age. Lower probability values indicate greater rarity, and the age factor allows for older items to be prioritized as more rare.

1.4.2. Rarity Based on Age Frequency

A simpler way to calculate rarity is based on how frequently an age occurs in the dataset:

$$R_{\text{freq}} = \frac{1}{\text{frequency of age}}$$

This formula assigns a higher rarity score to items with less common ages. It does not rely on probabilities and offers a straightforward method for assessing rarity based solely on age distribution. However, this method may not fully capture uniqueness in datasets where age frequency alone does not represent the true rarity of an item.

1.4.3. Normalized Age as a Rarity Indicator

Another possible approach for defining rarity is to use normalized age values:

$$R_{\text{norm}} = \frac{\text{age}}{\text{agemax}}$$

Here, age is normalized to a value between 0 and 1, with older items having higher scores. While this method is simple, it may overlook the distribution of ages across the dataset, which could result in certain age groups being over- or underrepresented in terms of rarity.

1.4.4. Comparison of Rarity Calculation Methods

Each rarity calculation method has its merits:

- The **probability-based formula** provides a nuanced rarity score by combining both the item's age and the likelihood of finding similar items. It works well in datasets with a broad age range and varying distributions.
- The **frequency-based approach** offers a more direct measure of rarity but may not always reflect the full diversity of the dataset, particularly when items share the same age.
- The **normalized age method** is effective for prioritizing older items but may not account for the overall distribution of ages in a dataset.

Depending on the sorting objective, a combination of these methods could be used for more accurate rarity assessments.

1.5. Analysis of Results

The following observations can be made from the benchmarking results:

- **Counting Sort:** This algorithm proved to be faster than Heap Sort when sorting by age, as its linear time complexity gives it an edge for datasets with smaller age ranges.
- **Heap Sort:** While slower than Counting Sort, Heap Sort is well-suited for sorting by rarity, where comparison-based methods are required. The $O(n \log n)$ complexity is efficient enough for larger datasets where rarity needs to be calculated.

The difference in performance is especially noticeable for larger datasets, where Counting Sort excels in scenarios that involve a smaller range of values (such as age), while Heap Sort is more effective when dealing with more complex sorting criteria like rarity scores.