IT University of Copenhagen

Algorithm Design 2

SAD 2 Project

Authors:

Daniel Stentsøe
Thorbjørn Serritslev Nielsen
Esben Philip Lind Kramer

Supervisors:

Rasmus Pagh
Francesco Silverstreni

December 16, 2013

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

1 Introduction

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2 Problem 1 - Most active actor pair

2.1 Problem definition

This problem is designing a tool capable of finding a pair of actors, who have appeared in the same movies more than any others, in an given interval of years. Additionally the tool should allow searching with an period of years as input, for instance finding the most active actor pairs over a period of 5 years, in the interval 1980-1990 (1980-1985, 1981-1986, etc.). The data for solving the problem is the IMDB dataset (REF).

2.2 Naive solution

We begin by presenting a greedy, naive solution for finding the most active actor pair.

Firstly, finding the solution for the problem demands that all actors from the movies of the interval, is selected from the database.
Our naive algorithm works by, for each movie, pairing up each actor with all other actors in the same movie. All found pairs are then added, with an attached counter, to an associative array. An existing element added to the array increases the attached counter.
A single pass of the array is then sufficient to produce the result.

An analysis of the running time gives an upper bound of $O(n^2 + m)$, where m is the number of movies in the database, and n is the number of actors retrieved.
The space requirement for the algorithm is $O(n^2)$.

2.3 Proposed solution

We present an algorithm which always find the correct solution, with a high chance of a reduced space requirement

In short, our algorithm works by repeatedly calling the Misra-Gries algorithm, with an increased value of the parameter, k. We will shortly explain this algorithm, and then return to discuss our solution.

2.3.1 Misra-Gries algorithm

Misra-Gries is a deterministic streaming-algorithm, capable of finding the most frequent elements in a stream - the so-called "heavy-hitters" - using a single pass of the data.
The motivation behind the algorithm is to be able to find the maintain a specific amount of elements The quality of the answer is controlled by a parameter, k. The algorithm maintains an array of maximum $k-1$ elements from the stream, in which a counter is associated with each element - to keep track of how many times that element has appeared.

When a new element, e_i, is added from the stream, one of the following actions occur:

- If the element already exists in $|k|$, increment e_i's counter.
- If the element does not exist in $|k|$, and an element that is monitored exist with a counter value of 0, replace that element with e_i, and set e_i's counter to 1.
- If the element does not exist in $|k|$, and if all monitored elements have counter values of more than zero, decrement all elements' counters by 1.

For a specific value of the stream (x_i), the Misra-Gries algorithm finds an approximated frequency of the element ($\hat{x}_i$), with a range of:

$$x_i - D <= \hat{x}_i <= x_i \tag{1}$$

Where D is the amount of decrements the algorithm performed. The largest amount of decrements possible can be found to be m/k (where m is the length of the stream).

The running time of the Misra-Gries algorithm have an upper bound time complexity of $O(n * k)$. However but a low space complexity of $O(k)$.

2.3.2 Algorithm for finding most active pair

In the context of our algorithm, we know that for two specific values of the stream (x_i and x_j), x_i will be guaranteed to be more frequent than x_j, if $x_i - D > x_j$. Comparing the two elements with the highest count from a single pass of Misra-Gries, $\hat{x_1}$ and $\hat{x_2}$, it is then possible to tell for certain if the element with the highest count is the most frequent element of the stream.

```
algorithm (movie_stream, initial_k)
while(true){
k <- initial_k
```

```
result = Misra-Gries(movie_stream, k)
if result.highest_elem > result.second_highest_elem + result.number_of_decrements then
return result.highest_elem

movie_stream.revert_stream
```

2.4 Implementation

We are writing the project in Java, and are getting our data from a MySQL database with the data from the provided IMDB imported. We are using ConnectorJ to retrieve data from MySQL into our workspace. The general architecture consists of a "data provider"-object, which reads data from our database and transform it into Pair-objects (an actor paired with another actor), and a "pair finder" class. The PairFinder class' job is to run a standard Misra-Gries algorithm and the class takes an initial k as input in its constructor. To emulate the stream and make sure that we do not load the entire set of data into memory, we only load one movie's actors at a time into the memory and the data provider then supplies a single pair at a time.

2.5 Results

Our pair object contains 8 byte for the object, and 3 times 4 bytes for the three integers (two actors, and a counter). This is of course 20 bytes, which times 3.2 million equates 64 million bytes, which is around 64 MB, which can be held in a standard Java VM with a little headroom to spare. This means that the amount of memory is directly related to our k: Memory consumption in megabytes (MC) = k / 1.000.000. This is again dependant on our M value from before. We will vary the M value to see how the result changes. We think that M = 0.5 is fairly low set, so the MC may be optimized some more. We currently assume that we know the maximum amount of pairs for a year, and set our values accordingly. This is not a given.

Changing our k to a too low value, will result in the right pair possibly being hidden in or forced out of the top pairs, with a frequency over 1/k.

We have discussed ways of changing our values dynamically based on the input. It may be possible to start with k = 3.2 million and decrease it when we encounter pairs that are already being monitored. This could work, since we can estimate M based on the frequency of the pairs in the stream. This still relies on the worst case 1.6 million per year, but this could be changed to the maximum allowed by the memory (k=available memory in bytes/20 bytes).

If we want to find the pair of actors who are in most movies over a subset of years from the input, we need to rethink our system. An immediate solution would be to maintain an array of k pairs for all subsets. We have discussed the possibility of combining reservoir sampling with Misra-Gries. Reservoir sampling gives us a probability that our output is correct, and since Misra-Gries will find the right answer for a large enough k, we could potentially change the k value to correspond to the sampling frequency.

Our current working solution to the above problem, is to query the SQL provider for each subset of years, and then comparing the results. However, the count is an estimate for each year, meaning that we can not be sure of which pair has the highest count when using this approach.

3 Problem 2 - WHAT IS THIS ONE CALLED??

3.1 Problem definition

3.2 Naive solution

3.3 Proposed solution

3.4 Implementation

3.5 Results

References

[1] Kurt, T. E., *Hacking Roomba*, Wiley Publishing. 2007.

[2] Lauesen, Søren, *User interface design: A software engineering perspective: The Virtual Window method*, IT University of Copenhagen. 2009.