FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

ALGORITHMS AND DATA STRUCTURES

RECOMMENDER SYSTEM

This is the written report for the project of group 11

Lukas De Loose
Sean Deloddere

UNIVERSITEIT
GENT

1 CONTENT-BASED FILTERING

1. QUESTION: Before starting the data must be read, this is done by putting the data in HashMaps. Can you explain why this is interesting? Is there another way to store the data? What are the advantages and disadvantages of the method you propose?

A HashMap is a very efficient Abstract Data Type (ADT). It has a constant (O(1)) complexity for the dictionary operations: Insert, Delete and Search. In addition, it is also space efficient. We have O(n+m) for hashing with chaining and O(m) for hashing with direct addressing (n = number of elements and m = number of possible hash values).
In conclusion, hashing is a very flexible way to store and retrieve our data, especially in our case, where we can easily index our ratings by user or movie.

We could also use a linked list to store our data, where the index of the rating in our list would be the movie or user id. The greatest disadvantage to this method is that we would have a linear ((O(n)) complexity for Search operation, which are frequently used.
An advantage is that linked lists are per definition ordered.

2. QUESTION: Compute the complexity of the first dynamic approach. You will see it is not very efficient.

To compute the big O of the first dynamic approach we will take a look at the worst case scenario and apply the algorithm to it. Since the algorithm consists of several steps, we will take a look at it step by step and compute the complexity accordingly. In step 1, there are 2 for-loops, that run across the string. If we take a string of length n, that gives us a complexity of n^2. We take every square subsequence we find and put it in a list. In the worst case, that's a list of length of $\mathrm{n} * (\mathrm{n} - 1)$. In step 2, once again there are 2 for-loops, but this time we iterate the list we made in step 1. This means we get a complexity of $\mathrm{n} * (\mathrm{n} - 1) * \mathrm{n} * (\mathrm{n} - 1)$. Finally, in step n, we repeat step 2 until we've found all square subsequences. We can repeat step 2 a maximum of $\frac{n}{2}$ times, since we always use another 2 characters to make a new square subsequence and there are only n characters available in the string. All this in total gives us:

$$n^2 + n * (n-1) * n * (n-1) * \frac{n}{2} = O(n^5)$$

3. QUESTION: For the first dynamic approach, give me a worst case example.

A word that consists of only the same letter (for example: “aaaaa”).
This would give us the largest possible value for SquareSubsequences.size() in step n.

4. QUESTION: If you implemented the second approach, compute its complexity.

Dynamic 2 clarification:

We decided to base our new alternative algorithm to find the amount of square substrings on the algorithm used to find the longest common subsequence in 2 strings.
The algorithm starts of by making 2 square matrices, height and width 1 larger than the length of the string we are trying to find the amount of squared subsequences from. There will be 1 matrix that holds integers telling us something about the amount of square subsequences, this matrix is called ‘tabel’, and 1 matrix that will hold a direction which we will use when checking if we can find subsequences of a longer length, this one is called ‘pijlen’.

Next up we’ll fill both the first rows and the first columns with zeroes. This is done by 2 for-loops, 1 iterating the first row of both matrices, the other the first column.
Once that is done we have a double for-loop to iterate the rest of the matrices. This time however, instead of filling it up with 0’s we will put values in it. We can have three cases. The first case is when the character at position of i-1 is equal to the character at position j-1. In this case we put a 3 in our ‘pijlen’ matrix, which represents a diagonal arrow, and in our ‘tabel’ matrix we put the integer 1 larger than the one left and above it. If the characters are not the same however, we will just put the largest integer (above or to the left) in our ‘tabel’ matrix, and store an arrow accordingly in the ‘pijlen’ matrix.

So far it’s all very similar to the common subsequences algorithm. We haven’t actually counted any square subsequences though. To count the subsequences, we add a while loop into the first scenario (when the characters are equal). This means that whenever 2 characters are equal, and the while condition is met, this while loop runs back following the arrows stored in ‘pijlen’ to check if there are square subsequences of a larger length than 2. The i<jj is there to prevent us from counting double, and the pijlen[ii][jj]=0 is there to let us know when we are at the border of our matrices.

This algorithm does a pretty good job at finding all the square subsequences in most cases, however, there is a problem. In some cases when there are 2 identical square subsequences in terms of characters, but not in terms of indices, the while loop will end because it has reached a border, before finding every possible square subsequence. We have unfortunately not found a solution for this problem.

Answer to question:
The complexity is similar to that of the common subsequences algorithm, except here of course there is only 1 string so O(n*m) = O(n^2). On top of that there is a while-loop in the double for-loop. This makes the total running time O(n^3).

2 COLLABORATIVE FILTERING

1. QUESTION: Before starting the data must be read, this is done by putting the data in HashMaps. Can you explain why this is interesting? Is there another way to store the data? What are the advantages and disadvantages of the method you propose.

See answer at content-based filtering.

2. QUESTION: Can you give an example where the cosine distance is totally wrong? (So you see that the movies are not similar at all, but nevertheless the distance is zero

	Movie A	Movie B
user 1	1	5
user 2	1	5
user 3	1	5
user 4	1	5

It's obvious the movies are very different, movie A is rated very poorly, while movie B is a huge success. However, if we apply the cosine distance we get:

$$Distance = 1 - \frac{(5+5+5+5)}{\sqrt{(1^2+1^2+1^2+1^2)} * \sqrt{(5^2+5^2+5^2+5^2)}}$$

$$= 1 - \frac{20}{\sqrt{4} * \sqrt{100}}$$

$$= 1 - \frac{20}{20}$$

$$= 0$$

3. QUESTION: What happens if a movie is not rated? How would you solve it?

We would probably get a NullPointer exception in the calculation of the distance. We could solve this by excluding the movies with no rating, like we excluded likedMovie in relatedMoviesCollaborative. This would of course mean that this movie would never show up. To fix this we could also give it a mediocre rating (2.5 out of 5).

4. QUESTION: How would you use this to recommend only the movies with the highest rating?

For the input, you take a movie with a perfect rating. Then the higher the rating of other movies, the smaller the distance. Thus only the movies with the highest rating get returned.