Project Algorithms & Data Structures

Group 35

Ward de Muer & Garben Tanghe

**Content Based Filtering**

**Scenario 0:** A HashMap is useful because it provides key-value access to data.

Other options are stacks, queues and dictionaries.

HashMaps:

Advantages of using HashMaps are:

* Provides key-value access to data.
* Time complexity of search is O(1) when hashing with chaining or open addressing is used.

Disadvantages of using HashMaps are:

* If a wrong hashing algorithm is used, the performance will decrease dramatically and the HashMap might turn into a linked list.

Stacks:

Advantages of using stacks are:

* Type of data structure which can easily be implemented and in which elements can be stored without hardly any effort.

Disadvantages of using stacks are:

* Stacks work with the principle of ‘Last In, First Out’ (LIFO). If we need to search for a particular element in the stack, we are supposed to remove each element at the top until the required one has been found. But we are also obliged to keep track of all the removed elements, so the use of another stack to temporarily save all other elements until we find the one we’re searching for is necessary. Doing this, we are in need of extra memory and a dramatic increase in time complexity might occur if all of the elements have to be stored in another stack, and then need to be put back in the original stack.

Queues:

Advantages of using queues are:

* A queue is a First In, First Out (FIFO) system, which means that it keeps its order. If we would use a priority queue, we could weight each movie according to the similarity to movies which the user likes. This would allow us to move the movies with greater similarity and weight more towards the front of the queue, rather than always being added to the tail.

Disadvantages of using queues are:

* Like stacks, searching for elements will cost us more space and time. With priority queues, we start from the head of the queue and remove the elements and keep track of them in another queue until the one we are searching has been found. Unlike stacks, we are supposed to move all of the remaining elements of the original queue to the new one because once removed, we can’t put an element back at the front.
* Inserting new elements will also cost also additional memory space and time, because the elements would have to find their place in the queue, which would be behind the elements with a greater or equal weight.

Dictionaries:

Advantages of using dictionaries are:

Disadvantages of using dictionaries are:

**Scenario 1:** - The complexity of Dynamic1 is O(n5) in worst case.

- A worst case scenario is that all the letters in String s are the same. There will be may (equal) SquareSubsequences. The longest SquareSubsequence will have a length of n when n is even or n-1 if n is uneven, with n = s.length();

- The complexity of Dynamic2: not implemented

**Collaborative Filtering**

**Scenario 0:** Idem as scenario 0 in ContentBasedFiltering.

**Scenario 2:** The cosine distance is totally wrong when, 0 is a possible rating.

Let’s consider these 2 cases:

Case 1:

|  |  |  |
| --- | --- | --- |
|  | Movie A | Movie B |
| User A | 0 | 5 |
| User B | 0 | 5 |
| User C | 0 | 5 |
| User D | 0 | 5 |
| d(A, B) = 1 - = 1 - = 1 | | |

Case 2:

|  |  |  |
| --- | --- | --- |
|  | Movie A | Movie B |
| User A | 0 | 0 |
| User B | 1 | 0 |
| User C | 0 | 0 |
| User D | 0 | 1 |
| d(A, B) = 1 - = 1 - = 1 | | |

Attention: at least 1 rating should be non-zero!

In case 1, the distance couldn’t be much bigger and we do get the biggest distance possible.

In case 2, the distance is not big at all, but we get an equally big distance as in case 1.

**Scenario 3:** When a movie is not rated, there is a problem. We can solve it by giving that movie a rating of 2.5. If a user hast just watched one of the greatest movies he’s ever seen or one of a kind he never wants to witness again, he’s much more likely to give this movie a rating. Whereas if the user thinks the movie is rather mediocre, he might not have the needs to express his / her feelings about the movie. Giving this movie an average rating of 2.5 might be a good solution to get rid of movies which don’t have a rating.

**Scenario 4:** A solution for giving a movie a rating for each user is to give it a weighted average, based on other users who have seen the movie. The smaller the distance between 2 users, the bigger the weight and the other way around.

Another method is to use content based filtering. When a user didn’t rate a movie, find the closest movie that the user did rate and give it the same rating. When 2 movies are very similar, the same user should give it almost the same rating.

A similar method to this one, without using content based filtering is to just look at some of the similar movies to a specific movie, that the user did rate and calculate an average rating for it.

Working further from the last method, we could recommend only the movies with the highest rating by saving the newly created ratings in a fixed size priority queue.