FACULTY OF ENGINEERING
AND ARCHITECTURE

Algorithms and Data Structures

Java Project

GHENT UNIVERSITY

IDLab
INTERNET & DATA LAB

imec

Practical Information

Work in groups of 2 → Register on Minerva before 31/03

No collaboration between different groups

Hand in: Deadline 07/05, both for Java files and written report

Short informal talk about project during last lab session 15/05

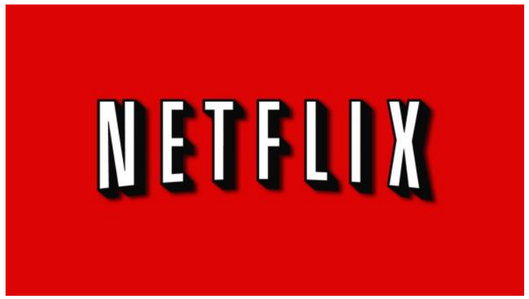
Questions can be asked during the second Java lab session

E-mail: leen.debaets@ugent.be

No questions, other than practical, will be answered after 01/05

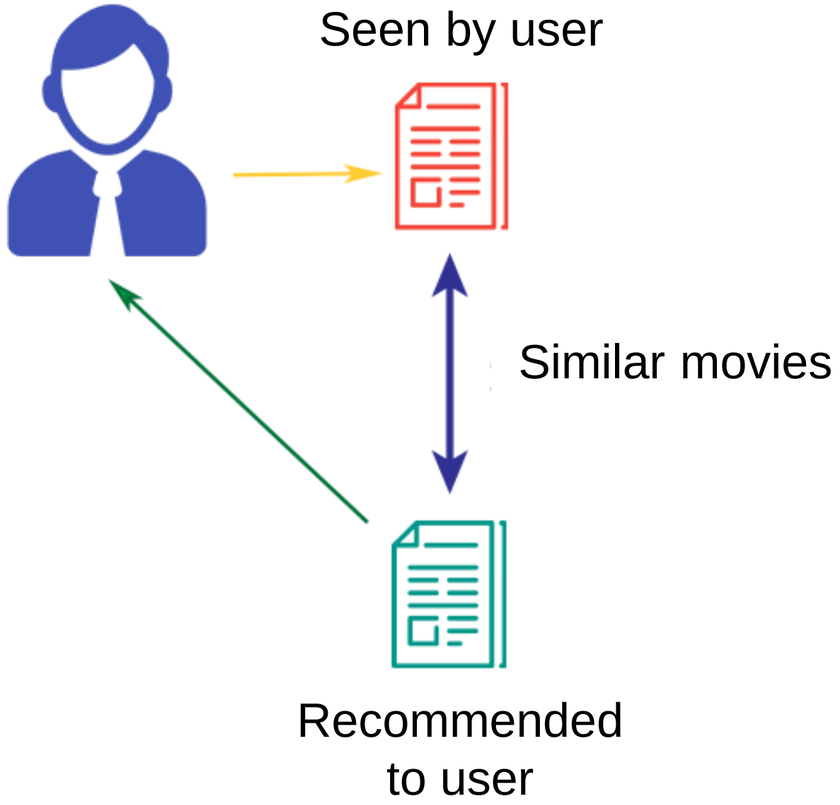
Recommender system

A recommender system is a system that seeks to predict which items a user likes and/or dislikes. This is done by estimating the ratings that a user would give to items.



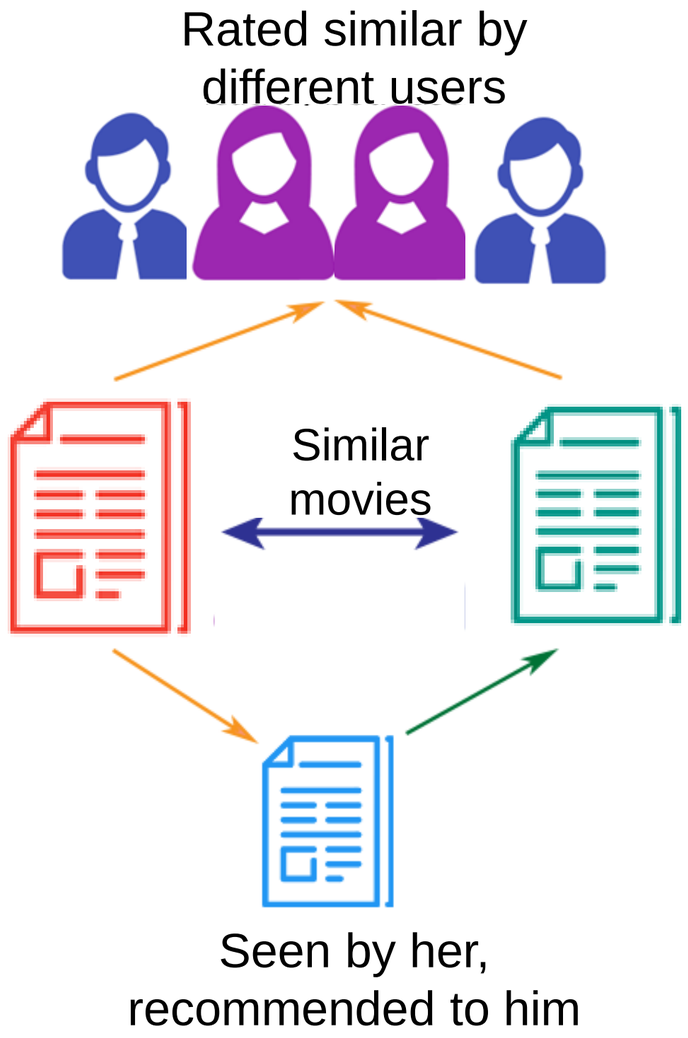
Recommender system: two groups

1) Content-based filtering:
   - Recommendation based on similar properties of the movies
   - E.g., we examine as property the genre of a movie and if we notice that a user watches romantic movies, we will recommend another romantic movie

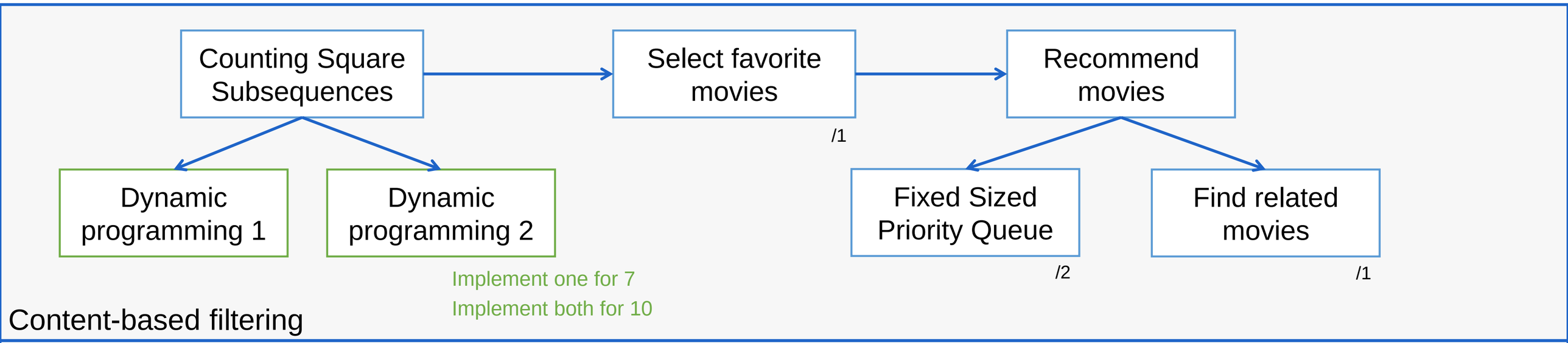


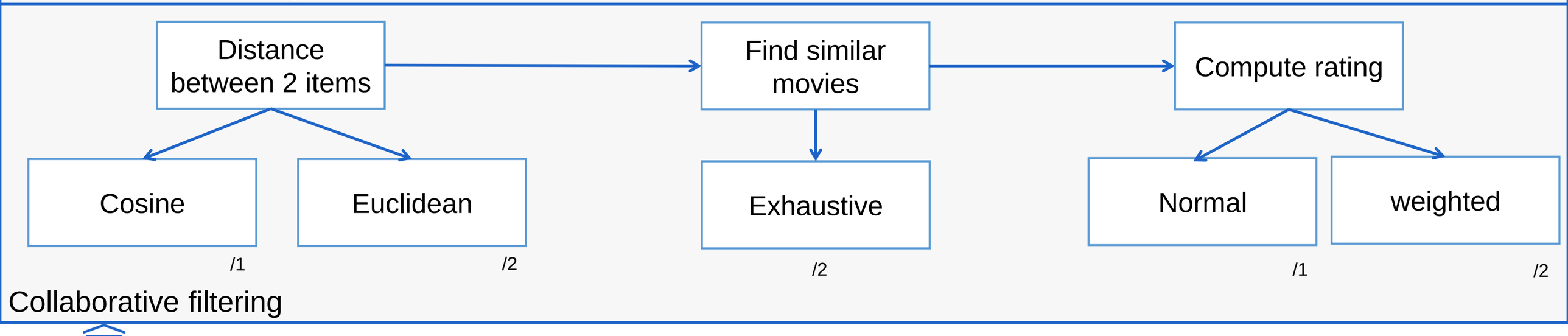
Recommender system: two groups

2) Collaborative filtering:
   - Recommendation based on similarity between users and/or items
   - E.g., if movie A is similar to movie B (based on ratings not on content) and user B likes movie A, then we will recommend movie B to him.



Project


Counting Square Subsequences
Select favorite movies
/1
Recommend movies
Dynamic programming 1
Dynamic programming 2
Implement one for 7
Implement both for 10
Fixed Sized Priority Queue
/2
Find related movies
/1
Content-based filtering


Distance between 2 items
Find similar movies
Compute rating
Cosine
/1
Euclidean
/2
Exhaustive
/2
Normal
/1
weighted
/2
Collaborative filtering

Content based filtering

In content based filtering, recommendation is based on **similar properties of the movies**. Examples of these properties can be the genre, the actors playing in a movie, the release year, …

We will however look at the title and search for the **amount of square subsequences** in it. Two movies are similar if they have a similar amount of square subsequences in their titles.

A string is called a **square string** if it can be obtained by concatenating two copies of the same string.
Eg., of square strings:

- *abab*
- *aa*

But these strings are no square strings:

- *aaa*
- *abba*

A **subsequence** of a string can be obtained by deleting zero or more characters from it, and maintaining the relative order of the remaining characters.

Counting square strings: example

Given string **baaba**, then the present sub square sequences are:

- aa (2, 3)
- aa (2, 5)
- aa (3, 5)
- bb (1, 4)
- baba (1-2, 4-5)
- baba (1-3, 4-5)

Counting square strings: dynamic programming 1

Two observations:

1. The length of a square string is always even.

2. Every square subsequence of length 2n is a combination of 2 square subsequences: one of length 2(n-1) and one of length 2

Step 1: find the square subsequences of size 2, it is important to keep the indices of the square subsequences.

Given string **baaba**, then the present square subsequences of size 2 are:

- aa (2, 3)
- aa (2, 5)
- aa (3, 5)
- bb (1, 4)

Counting square strings: dynamic programming 1

Step 2: combine the square subsequences of size 2 to form square subsequences of size 4

The combination of two square subsequences sss1 with indices (a, b) and sss2 with indices (c, d) is again a square subsequence with indices (a - c, b - d) if

- $a < c$
- $c < b$
- $b < d$

Given the square subsequences of size 2, then the square subsequences of size 4 are:

• aa	(2, 3)
• aa	(2, 5)
• aa	(3, 5)
• bb	(1, 4)
• baba	(1 - 2, 4 - 5)
• baba	(1 - 3, 4 - 5)

Counting square strings: dynamic programming 1

Step 2: combine the square subsequences of size 2 with the square subsequences of size 4 to form square subsequences of size 6

The combination of two square subsequences sss1 with indices (a, b) and sss2 with indices (c_1- c_2, d_1- d_2) is again a square subsequence with indices (a - c_1- c_2, b - d_1- d_2) if

- $a < c_1$
- $c_2 < b$
- $b < d_1$

Given the square subsequences of size 2 and, then the square subsequences of size 6 are:

- aa (2, 3) /
- aa (2, 5)
- aa (3, 5)
- bb (1, 4)

- baba (1 - 2, 4 - 5)
- baba (1 - 3, 4 - 5)

Counting square strings: dynamic programming 1

Step n: combine the square subsequences of size 2 with the square subsequences of size 2n to form square subsequences of size 2(n + 1)

The combination of two square subsequences sss1 with indices (a, b) and sss2 with indices (c_1- c_2 - … - c_n, d_1- d_2 -… - d_n) is again a square subsequence with indices (a - c_1- c_2 - … - c_n, b - d_1- d_2 -… - d_n) if

- $a < c_1$
- $c_n < b$
- $b < d_1$

This dynamic programming approach works, but it is not the most efficient.

Counting square strings: dynamic programming 2

Two tips:

1. Reformulate the problem as one for finding the amount of common subsequences
2. Take care to not count some square subsequences twice

Content based filtering: find similar movies

Once you have calculated the amount of square sequences in all movie titles, you can do the content based filtering:

1. Select one user
    1. Selects its highest rated movies

E.g.

Name	Rating	# square sequences
Movie b	4	1
Movie c	1	2
Movie d	4	3

The highest rated movies are b and d

    2. Search for movies similar to the highest rated movies: two movies are similar if they have a similar amount of square subsequences in their titles.

        E.g. If you have a Movie x, with 4 square subsequences in its title. Then its similarity to the highest rated movies b and d, is respectively 3 (|4-1|) and 1 (|4-3|). To obtain one number, we take the minimum.

Content based filtering: fixed sized priority queue

We only want a certain amount of recommendations (similar movies). To this end, a **fixed sized priority queue** can be used. A fixed sized priority queue is a priority queue with a fixed sized.

In this project, the priority is defined as the distance and the item with the highest priority is the item with the largest distance. So the element with the largest distance is on top of the queue and the one with the smallest distance at the bottom. Consequently, if we pop an element, the element with the largest distance is removed.

As we want to keep track of the n most similar items, the size of the fixed sized priority queue is n. So, if we want to add an item when the queue is full, we first need to pop an item and then push the new item.

Fixed sized priority queue: example with n = 3

Add item B with d(A,B) = 5

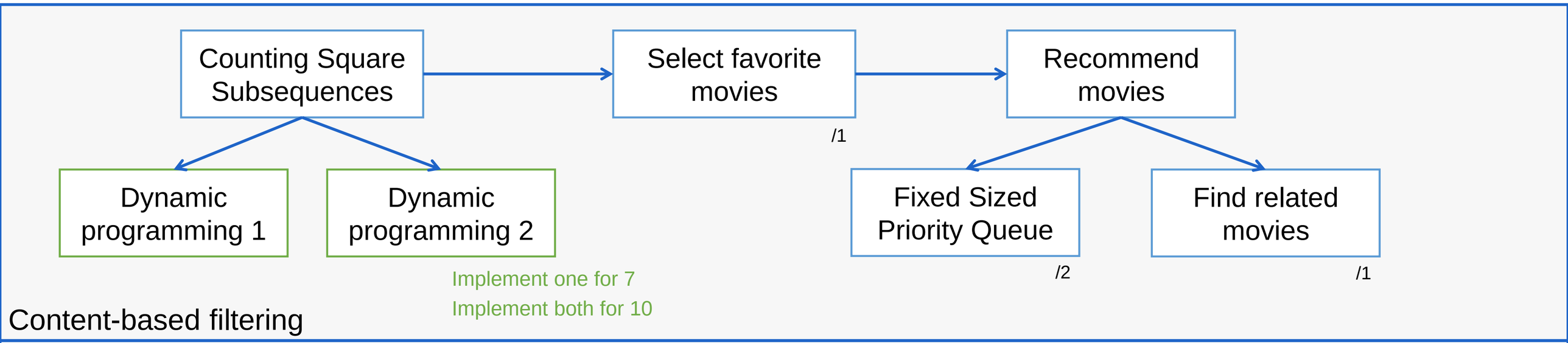
Add item C with d(A,C) = 10

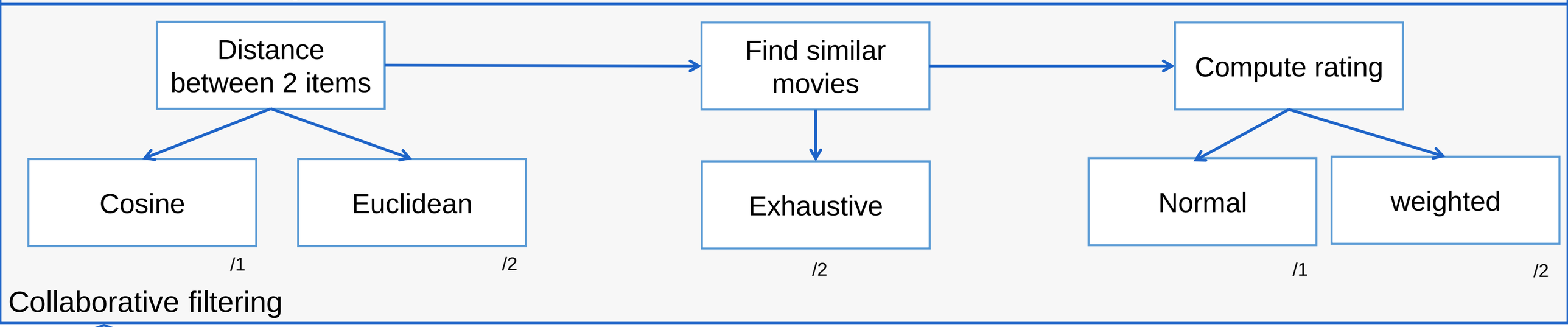
Add item D with d(A,D) = 7

Add item E with d(A,E) = 8

		B, 5
	C, 10	B, 5
C, 10	D, 7	B, 5
	D, 7	B, 5
E, 8	D, 7	B, 5

Project


Counting Square Subsequences
Select favorite movies
/1
Recommend movies
Dynamic programming 1
Dynamic programming 2
Fixed Sized Priority Queue
/2
Find related movies
/1
Implement one for 7
Implement both for 10
Content-based filtering


Distance between 2 items
Find similar movies
Compute rating
Cosine
/1
Euclidean
/2
Exhaustive
/2
Normal
/1
weighted
/2
Collaborative filtering

Collaborative filtering: calculate the distance between two movies

A movie is associated with a list containing its rating.

Now we want to calculate the distance between two movies. This is not straightforward, as two movies can be rated by a different set of users.

E.g.,

	Movie A	Movie B
User a		5
User b	1	2
User c	1	
User d		

Collaborative filtering: calculate the distance between two movies

Step 1: associate with the two movies, two arrays containing their ratings.

- If a user rated only one of the two movies, its rating for the other movie is 2.5
- If a user didn't rate both movie, you don't need to include these ratings in the arrays as they are both zero

E.g.,

	Movie A	Movie B
User a		5
User b	1	2
User c	1	
User d		

A = (2.5, 1, 1)

B = (5, 2, 2.5)

Collaborative filtering: calculate the distance between two movies

Step 2: calculate the distance between both vectors using the cosine distance

The distance between two movies $A = (A_1, A_2, \dots, A_n)$ and $B = (B_1, B_2, \dots, B_n)$ is calculated using the cosine distance:

$$\mathrm{d}(A,B) = 1 - \frac{\sum_i A_i * B_i}{\sqrt{\sum_i {A_i}^2} * \sqrt{\sum_i {B_i}^2}}, \text{ if ratings in common}$$

$$= +\infty, \qquad \text{otherwise}$$

Collaborative filtering: calculate the distance between two movies

$$d(A,B) = 1 - \frac{\sum_i A_i * B_i}{\sqrt{\sum_i A_i^2} * \sqrt{\sum_i B_i^2}}, \text{ if ratings in common}$$

$$= +\infty, \quad \text{otherwise}$$

	Movie A	Movie B
User a		5
User b	1	2
User c	1	

$$d(A,B) = 1 - \frac{2.5 * 5 + 1 * 2 + 1 * 2.5}{\sqrt{1 + 1 + 6.25} * \sqrt{25 + 4 + 6.25}}$$

Collaborative filtering: calculate the distance between two movies

$$\mathrm{d}(A,B) = 1 - \frac{\sum_i A_i * B_i}{\sqrt{\sum_i A_i{}^2} * \sqrt{\sum_i B_i{}^2}}, \text{ if ratings in common}$$

$$= +\infty, \qquad \text{otherwise}$$

	Movie A	Movie B
User a		5
User b	1	
User c	1	

$$d(A,B) = +\infty$$

Collaborative filtering: calculate the distance between two movies

Step 2: calculate the distance between both vectors using the Euclidean distance

The distance between two movies $A = (A_1, A_2, \ldots, A_n)$ and $B = (B_1, B_2, \ldots, B_n)$ is calculated using the Euclidean distance:

$$\mathrm{d}(A, B) = \sqrt{\sum_i (A_i - B_i)^2}, \text{ if ratings in common}$$

$$= +\infty, \qquad \text{otherwise}$$

Collaborative filtering:

For a movie, you want to find the most similar movies.

1. You calculate the distance from the selected movie to all other movies
2. Keep track of the closest movies (most similar ones) using the fixed sized priority queue

Collaborative filtering: calculate the rating

For a user and a movie, you want to calculate the rating.

1. Find the closest movies (see previous slide)
2. Select from the movies that were also rated by the user, average those ratings to get the rating

This is of course only interesting if the user didn’t rate the movie before.

Collaborative filtering: calculate the weighted rating

For a user and a movie, you want to calculate the rating.

1. Find the closest movies (see previous slide)
2. Select from the movies that were also rated by the user, average those ratings to get the ultimate rating in such a way that more similar movies have a higher weight in the average

Submission details

When: **07/05** (11:59 p.m.)

How: Submit using Dropbox functionality of Minverva “Alle cursusbeheerders”.

What: 1 zip file named **GroupX_LastName1_LastName2.zip** containing:

- The written report in pdf format
- The relevant Java files

Do not change the package structure!

Do not submit entire Eclipse or other project folders

Do not submit class file

Naming:

- X = number of group on Minerva
- Mention your group and names in the written report and the Java files

Some tips

Be sure to complete the tasks mentioned in the Java code.

You can add as much code as you want, but do not change existing code.

Do not neglect your written report. It can be short but needs to be precise. Answer the questions present in ContentBasedFiltering.java and CollaborativeFiltering.java. If asked what the time or memory complexity is, a high level reasoning is enough. You do not need to prove it.

Try to work together in your group and do not simply divide the workload.