Dyanmic 2

```
public static int amountOfSquareSubSequences(String s){

	int l=s.length();
	int teller=0;
	int ii;
	int jj;
	int[][] tabel=new int[l+1][l+1];
	int[][] pijlen= new int[l+1][l+1];

	//0 is return, 1 is links, 2 is boven, 3 is diagonaal
	for(int i=0;i<=l;i++){
		tabel[i][0]=0;
		pijlen[i][0]=0;
	}
	for(int i=0;i<=l;i++){
		tabel[0][i]=0;
		pijlen[0][i]=0;
	}
	for(int i=1;i<=l;i++){
		for(int j=1;j<=l;j++){
			if(s.charAt(i-1)==s.charAt(j-1)){
				tabel[i][j]=tabel[i-1][j-1]+1;
				pijlen[i][j]=3;
				ii=i;
				jj=j;
				while(pijlen[ii][jj]!=0 && i<jj){
					switch(pijlen[ii][jj]){
					case 1:ii--;break;
					case 2:jj--;break;
					case 3:teller++;ii--;jj--;break;
					}


				}
			}
			else if(tabel[i-1][j]>=tabel[i][j-1]){
				tabel[i][j]=tabel[i-1][j];
				pijlen[i][j]=2;
			}
			else{
				tabel[i][j]=tabel[i][j-1];
				pijlen[i][j]=1;
			}
		}
	}

	  return teller;
    }
```

Dynamic 1

```
public static int amountOfSquareSubSequences(String s){

	ArrayList<SquareSubsequence> SquareSubsequences = new ArrayList<>(); //To
store the found SquareSubsequences

	//Step 1, find the square subsequences of size 2
	for (int i =0; i<s.length()-1;i++){
		for (int j = i+1; j<s.length();j++){
			if (s.charAt(i)==s.charAt(j)){
				String halfstring = s.substring(i, i+1)+
s.substring(j,j+1);
			ArrayList<Integer> leftIndices = new ArrayList<>();
			ArrayList<Integer> rightIndices = new ArrayList<>();
			leftIndices.add(i);
			rightIndices.add(j);
			SquareSubsequence two_letter = new
SquareSubsequence(halfstring,leftIndices,rightIndices);
				SquareSubsequences.add(two_letter);
			}
		}
	}

	//Step 2, combine the sss of size 2 to form sss of size 4
	int size_2 = SquareSubsequences.size(); // size changes in loop

	for (int i = 0; i<size_2-1;i++){
		int a = SquareSubsequences.get(i).getLeftIndices().get(0);
			int b = SquareSubsequences.get(i).getRightIndices().get(0);

		for (int j = i+1; j<size_2;j++){
			int c = SquareSubsequences.get(j).getLeftIndices().get(0);
			int d = SquareSubsequences.get(j).getRightIndices().get(0);

			if (a < c && c < b && b <d){

			    ArrayList<Integer> leftIndices = new ArrayList<>();
			    ArrayList<Integer> rightIndices = new ArrayList<>();
			      leftIndices.add(a);
			      leftIndices.add(c);
			      rightIndices.add(b);
			      rightIndices.add(d);

				String halfstring = s.substring(a, a+1)+
s.substring(c,c+1) + s.substring(b, b+1) + s.substring(d, d+1);
				SquareSubsequence two_letter = new
SquareSubsequence(halfstring,leftIndices,rightIndices);
				SquareSubsequences.add(two_letter);

			}
		}
	}

	//Step n: combine the sss of size 2 with the sss of size 2n to form sss of
size 2(n+1)

	for (int i = size_2;i<SquareSubsequences.size();i++){
		int c1 = SquareSubsequences.get(i).getLeftIndices().get(0);
		int d1 = SquareSubsequences.get(i).getRightIndices().get(0);
```

```
            int cn = SquareSubsequences.get(i).getLeftIndices().get(1);
            int dn = SquareSubsequences.get(i).getRightIndices().get(1);

            for (int j = 0; j< size_2; j++){ //sss of length 2
                  int a = SquareSubsequences.get(j).getLeftIndices().get(0);
                  int b = SquareSubsequences.get(j).getRightIndices().get(0);


                  if (a < c1 && cn < b && b < d1){
                        ArrayList<Integer> leftIndices = new ArrayList<>();
                      ArrayList<Integer> rightIndices = new ArrayList<>();
                        leftIndices.add(a);
                        leftIndices.add(cn);
                        rightIndices.add(b);
                        rightIndices.add(dn);


                        int stringlength =
SquareSubsequences.get(i).getHalfString().length();
                        String leftstring =
SquareSubsequences.get(i).getHalfString().substring(0, stringlength/2);
                        String rightstring =
SquareSubsequences.get(i).getHalfString().substring(stringlength/2, stringlength);


                        String halfstring = s.substring(a, a+1) + leftstring +
s.substring(b,b+1)+rightstring;
                        SquareSubsequence new_sss = new
SquareSubsequence(halfstring,leftIndices,rightIndices);
                        SquareSubsequences.add(new_sss);

                  }
            }
      }
        return SquareSubsequences.size();
    }
```

Rating

```
public static HashMap<User, Double> ratingBasedOnSimilarMovies(Movie a,
FixedSizedPriorityQueue similarToA, HashMap<Integer,ArrayList<Rating>>
ratingsIndexedByMovie) throws Exception{

        HashMap<User, Double> ratingsMovie = new HashMap<>();
        HashMap<User, ArrayList<Double>> ratingsPerUser = new HashMap<>();
        HashMap<Integer, User> Users= new HashMap<>();
        ArrayList<User> usersThatRatedA = new ArrayList<>();

        for (int i = 0; i < ratingsIndexedByMovie.get(a.getId()).size(); i++){
//Iterate over all ratings of a and get the users

      usersThatRatedA.add(ratingsIndexedByMovie.get(a.getId()).get(i).getUser());
        }

        //Take out movies from fspq
        while (similarToA.size()>0){
             Movie b = (Movie)similarToA.remove().getValue();

             //Take out all the ratings of that movie
             ArrayList<Rating> ratingsOfB = ratingsIndexedByMovie.get(b.getId());

             for (int i = 0; i< ratingsOfB.size(); i++){
                   if (!usersThatRatedA.contains(ratingsOfB.get(i).getUser())){
//If User already rated a, do nothing

                         if (ratingsPerUser.get(ratingsOfB.get(i).getUser()) ==
null){ //Check if User already in HashMap, if not, make new ArrayList, put Rating
in ArrayList and add ArrayList to HashMap
                         ArrayList <Double> theRatings = new ArrayList<Double>();
                         theRatings.add(ratingsOfB.get(i).getRating());
                         ratingsPerUser.put(ratingsOfB.get(i).getUser(),
theRatings);

      Users.put(ratingsOfB.get(i).getUser().getId(),ratingsOfB.get(i).getUser());
                   }

                   else { //Add the rating to ArrayList in HashMap

      ratingsPerUser.get(ratingsOfB.get(i).getUser()).add(ratingsOfB.get(i).getRat
ing());
                   }
             }

             }
        }

        double size = ratingsPerUser.size();

        for (int i = 0; i < size;i++){

             if (Users.get(i) != null){ //So if User is in the HashMap
                   double averageRating = 0;
                   for (int j = 0; j < ratingsPerUser.get(Users.get(i)).size();
j++){
                         averageRating +=
ratingsPerUser.get(Users.get(i)).get(j);
                   }
```

```
                    averageRating /= ratingsPerUser.get(Users.get(i)).size();
                    ratingsMovie.put(Users.get(i), averageRating);

                }
                else { //Else we might miss keys, for example if user with id 2
doesn't exist, then we would miss the last user.
                    size++;
                }
            }

            return ratingsMovie;
        }
```

Weighted rating

```
public static HashMap<User, Double> ratingBasedOnSimilarMoviesWeighted(Movie a,
FixedSizedPriorityQueue similarToA, HashMap<Integer,ArrayList<Rating>>
ratingsIndexedByMovie) throws Exception{
      //remark: We think there is an error in the solution file, because now
movies with a bigger distance (less similar), have a bigger weight.
      //            This should not be the case. Easily fixed in
ratingBasedOnSimilarMoviesWeighted by multiplying by (5-distance) at ***Should be
5-distance***

        HashMap<User, Double> ratingsMovie = new HashMap<>();
        HashMap<User, ArrayList<Double>> ratingsPerUser = new HashMap<>();
        HashMap<Integer, User> Users= new HashMap<>();
        ArrayList<User> usersThatRatedA = new ArrayList<>();
        ArrayList<Rating> ratingsOfA = ratingsIndexedByMovie.get(a.getId());
        double totalDistance = 0;

        for (int i = 0; i < ratingsOfA.size(); i++){ //Iterate over all ratings of
a and get the users
             usersThatRatedA.add(ratingsOfA.get(i).getUser());
        }

        //Take out movies from fspq
        while (similarToA.size()>0){
             Movie b = (Movie)similarToA.remove().getValue();

             //Take out all the ratings of that movie
             ArrayList<Rating> ratingsOfB = ratingsIndexedByMovie.get(b.getId());

             for (int i = 0; i< ratingsOfB.size(); i++){


                  if (!usersThatRatedA.contains(ratingsOfB.get(i).getUser())){
//If User already rated a, do nothing

                       if (ratingsPerUser.get(ratingsOfB.get(i).getUser()) ==
null){ //Check if User already in HashMap, if not, make new ArrayList, put Rating
in ArrayList and add ArrayList to HashMap
                       ArrayList <Double> theRatings = new ArrayList<Double>();
                       //Calculate distance between movie a and movie that's
rated
                       double distance = distanceBetweenTwoMovies(ratingsOfB,
ratingsOfA, "euclidean");
                       totalDistance += distance;
                       theRatings.add(ratingsOfB.get(i).getRating()*distance);
//*** Should be 5-distance ***
                       ratingsPerUser.put(ratingsOfB.get(i).getUser(),
theRatings);

      Users.put(ratingsOfB.get(i).getUser().getId(),ratingsOfB.get(i).getUser());
                  }

                  else { //Add the rating to ArrayList in HashMap
                       double distance = distanceBetweenTwoMovies(ratingsOfB,
ratingsOfA, "euclidean");
                       totalDistance += distance;

      ratingsPerUser.get(ratingsOfB.get(i).getUser()).add(ratingsOfB.get(i).getRat
ing()*distance); //*** Should be 5-distance  ***
```

```
                    }
                }

                }
            }

            double size = ratingsPerUser.size();

            for (int i = 0; i < size;i++){

                if (Users.get(i) != null){ //So if User is in the HashMap
                    double averageRating = 0;
                    for (int j = 0; j < ratingsPerUser.get(Users.get(i)).size();
j++){
                        averageRating +=
ratingsPerUser.get(Users.get(i)).get(j);
                    }
                    averageRating /= totalDistance;
                    ratingsMovie.put(Users.get(i), averageRating);

                }
                else { //Else we might miss keys, for example if user with id 2
doesn't exist, then we would miss the last user.
                    size++;
                }
            }

            return ratingsMovie;
        }
```