STA-Algorithm

STA stands for Student-Topic-Assignment. It is a JAVA-algorithm and expects a CSV-matrix saving all students topic wishes as input, and outputs an optimal matching.

Documentation-Content

Folder Structure

In the root-folder *STA-Algorithm* you will find an **.idea-folder** and an **.iml-file**. These are working directories used for *IntelliJ*. They may come in handy to you, if you will also use this *IDE*, improving the algorithm. More important you will find three other folders in the root-directory:

- **src-folder:** here you will find the editable source-code of the algorithm in JAVA.
- **out-folder:** here you will find the compiled and executable code of the algorithm.
- **doc-folder:** here you will find an example CSV-file, which you may use as input for the algorithm. You will also find this instruction-document as .pdf, the source code of it (.docx) and the article “*Algorithmic Matching”,* a detailed guideline for understanding and improving the algorithm and its math.

Using the STA-Program

The program can be executed via command line. It expects 3 String-values as input.
Later two of them will be converted in integer, so please use only numbers.

The values are:
* int studentCount: The total number of students, that need to be assigned to a topic
* int topicCount: The total number of topics, students can wish to be assigned to
* String csvFile: A path to a CSV-file. In the file all weighted student’s topic wishes are saved.
The file needs to be specially formatted. Refer to the CSV-Section below.

To use the program via console please use following command:
>>>java mainSTA [studentCount] [topicCount] [csvFile]

The program will **sometimes** let you know, if something went wrong. Please check the CSV-file format in this case. Also check the order of the arguments you gave. Make sure that the int numbers are correct. Also make sure that you encapsulated the file path like this "[csvFile]" if it contains blanks. Note that the input validation is not perfect, so always make a quick check, if the output makes sense. If not review your input as well, like describesd.

Preparing the CSV-Data

The input csv-file, following presented in a excel-like view, should look like this:

	T1	T2	T3	…	…	TK
S1		1	2			
S2	1					
S3		3				
…						
…						
SN						

Every Row is a student (S1 - SN). *Note that the N equals the studentCount you pass the program*

Every Column is a topic (T1 - TK). *Note that the K equals the topicCount you pass the program*

The first row and the first column store indices. These are headers and will be ignored, their content may be anything. It is recommended to fill in the same numeration as sketched above: Start with 1 at the second column, go on with two at the third column etc. Likewise call the header of the second row 1, of the third row 2 and on. With this it will be very easy to understand the output because the student-IDs and topic-IDs will match the header indices. More information in the next section and an example picture of a CSV is given below.

In the table itself the priorities of topic wishes given by the students are stored. Empty content should be real-empty. Don't fill in something else, like "0".

From an excel file containing more data, like names of students or topics, simply remove all unnecessary rows and columns. You may need to readjust the matrix's position in the excel file then. Be sure that that the data is stored right to the left and upper border of the excel, like visualized:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		1	2	3	4	5	6	7	8	9	10	11	12	13
2	1									2				
3	2													
4	3													
5	4									2	1			
6	5													
7	6													
8	7													
9	8				4			3						
10	9													
11	10			8	9									

An example sheet. Be sure that the headers are right next to the table border (yellow lines)

After preparing the excel you simply need to export it as csv-file. The programs File-Reader uses Semicolons as separator, so if you create a csv file on your own or your editor asks you, chose that.

Interpreting the Output

The Output is the optimal matching, the algorithm found and will be presented as a list, where every list element starts on a new line and represents a match. A match is an assignment from topic to students. Structure of a match:

sX -- tY [PZ]

Interpretation:

- s stands for student, while the number X indicates the ID of the student. s1 for example is the very first student, the program found in the input CSV
- t stands for topic, while the number Y indicates the ID of the topic.
- P stands for priority, while the number Z indicates which priority of the student was chosen.
- *Remark: IDs of students and topics are set by the program. The ID represents the order of the students and topics in the csv-input. The first student (read from top to bottom) has ID 1, the second student has ID 2 etc. The first topic (read from left to right) has ID 1, the second topic has ID 2 etc.*

Example: s1 -- t16 [P1]
In words: The first student in the csv-file is assigned to his first priority, namely topic 16 – the topic at position 16 in the csv-file.

An extraction of a full matching, based on real data is given bellow:

```
Optimal matching:
 s1 -- t16 [P1]
 s2 -- t36 [P1]
 s3 -- t42 [P1]
 s4 -- t10 [P1]
 s5 -- t37 [P2]
 s6 -- t22 [P1]
 s7 -- t17 [P1]
 s8 --  no matching possible
 s9 -- t23 [P1]
 s10 -- t26 [P3]
```

You can see all suggested matches from student 1 to 10 here. Student 3 for example is assigned to topic 42 – his first priority. Student 8 couldn't be matched. Take that literal. With the information given as input it is not possible to find a match for the student. It indicates that you must increase the information amount, to find a match. How to do that is discussed in the next section.

At the very bottom of the output you find a weight-sum (or priority-sum). It is the sum of all effective priorities. If student s1 has his [P1]-topic and student s2 his [P3]-topic, the weight-sum of both is 4. Often the weight-sum is very high. That is because every *no matching possible* increases the weight-sum drastically. Just remember the following: A lower weight-number signalizes a better matching.

Suggestions to remove “no match possible”

The message “no match possible” indicates, that the algorithm needs more information to find a matching. You have several options to increase the information amount:

- Let students enter more priorities. Be aware that this will increase the number of students getting a low-priority-topic assigned.
- Increase the number of (popular) topics. The more choice a student has, the more likely it is, that students wishes won’t overlap. With that it is easier to find good matchings.
- Duplicate topics that may be assigned twice. Just duplicate every topic, that may be done by two students. In excel that is easily done: Copy the whole column of the topic in the excel and paste it in as a new topic at the very-right of the document. Make sure you note the new ID in the header and make a remark that this topic ID is a copy of another topic ID.
  If you have for example 42 topics and you wish to make topic 31 assignable twice, you simply copy column 31 and paste it in right next to the column storing topic 42. Now you name the newly created row 43, because this will be the ID the algorithm will give the topic. Make some remark as well, that topic 43 is a copy of topic 31.

Improving the Algorithm

A few things are not perfect yet, so they may be improved:

- There is almost no input validation nor error handling.

- The usability is not intuitive. One improvement would be, making the CSV-reader more flexible. In the combination with printing out the result in a CSV as well this could lead to a solution, where the course leader (at time of programming mister Fiedler) didn’t need to bother about student / topic IDs, but can see which student is assigned to which topic by name.

- There are other constraints for the course leader he has to involve, not only students priorities. One big requirement is that experts / professors minimum pensums are fulfilled, meaning that every expert has to lead a minimum number of topics. There are some concepts to solve this problem, that were developed already. However, it is needed to understand the mathematics behind the algorithm. The general math theory, the principle of the algorithm, and the specific concepts to solve the additional requirement are all explained in very detail in the article “*Algorithmic Matching - Mathematical approaches to optimally allocate topics to students”* by *Frederik Heck*. After reading it, you should be able to edit and improve the algorithm, such that it solves the experts pensum problem (in the article called extended STA-problem). You should also be able to make other adjustments.

- Please consider talking with the course leader, to find out if he has some other wishes, experience-based improvement suggestions, or error reports.

- This document is available as docx-document as well, so you may make improvements to it, if you wish.