Opleiding Informatica

Analysis of submissions of the Dutch Olympiad for Informatics with a special focus on heuristics

Martijn Strating

Supervisors:
Felienne Hermans & Alaeddin Swidan

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

02/07/2019

Abstract

We have studied some programming problems from the Olympiad in Informatics, a programming competition for secondary education students, in order to see what contestants had difficulties with. As a basis, we used the ideas of the mathematician George Polya: solving problems using four steps. In the submissions, we looked for common ways of solving the Olympiad problems, as well as common made mistakes. Subsequently we arranged these in Polyas steps. It was shown that in all steps there was room for the contestants for improvements. For future work, more comprehensive results could be achieved if this research would be conducted on a larger scale.

Contents

1 Introduction

1.1 About the Olympiad

The International Olympiad in Informatics (IOI) is a yearly held programming competition for secondary education students. In terms of age, contestants at IOI'n (held in year n) must have been born on or after 1$^{\text{st}}$ July, in the year $n - 20$ per participating country. [IOI] Per country, four best contestants are selected after a selection that consists of three rounds. Then, the best participants compete in an international tournament. The number of contestants of the IOI is small compared to other Olympiads. For example, in 2019 there were 319 participants in the Netherlands (a record), whereas the number of participants of the Olympiad in Mathematics was 8150. [wis] One of the main goals of the Olympiad is to encourage and motivate people to do Informatics. On the other hand, the Olympiad does not want to lower the level, in such a way that the given problems can be solved without thorough knowledge and a creative approach. The difficulty of an Olympiad depends mainly on the level of the questions. In order to assess the difficulty, one has to examine multiple factors, such as the average grade of a question, but also how exactly the answers have been coded. Questions are now graded according to their performance on various test cases. However, a scrutinized look at the submissions themselves can reveal useful insights. There are four types of questions, see 1. In this research, we focus on questions of type A and C.

Type	Description	Amount	Points
A	Preliminary exercises	5	40
B	Theoretical exercises	4	50
C	Advanced exercises	2/3	100
D	Program a game	1	100

Table 1: Overview Olympiad questions

1.2 Overview

Firstly, we report on heuristics, the study of methods of problem-solving (especially the work of George Polya). It is a viewpoint that is much used in computer science education (more about that later) and we hope it gives us a useful framework. Problem-solving is seen as a vital, important part of programming. [PK17] Secondly, we analyze the answers of four Olympiad problems. A database was obtained from the Dutch Informatics Olympiad, which contains data about the previous made Olympiad questions. This includes, among others, scores, test cases and the submissions themselves. An EER diagram of the database can be found in Appendix 5. We hope to find insights in why contestants find questions hard, and if we can recognize patterns in the submissions with Polyas ideas in mind.

2 About Polya

When we analyze the submissions to the Olympiad, it is useful to have a good understanding of the problem of which the submission is the answer. Also helpful seems a guiding principle. One of the most important books of problem solving is How to Solve It, which was published in 1945 by the Hungarian mathematician George Polya, a professor at Stanford University. It is a book on heuristic, a study about how to solve problems (not to be confused with the heuristics that are used in algorithms like A* that are techniques to find a quicker solution). It is written on a high level of abstraction, with purpose of its content being applicable to the most diverse array of problems. We assume that its content, by this generality, is useful for our study.
According to Polya, one must adhere to four steps when solving a problem. It was his intention to keep everything simple, to state plain commons sense in general terms The steps are:

1. Understanding the problem
2. Devising a plan
3. Carrying out the plan
4. Looking back

2.1 Summary of 'How to Solve It'

How to Solve It is based on the four steps mentioned earlier. In this section we will summarize each step. Later on we will make a comparison between the book and solving programming exercises.

2.1.1 Understanding the problem

In this phase, Polya advises to start with asking three questions:

1. What is the unknown?
2. What are the data?
3. What is the condition?

Additionally, he advises the student to draw a figure, as visualizing the problem leads to a better understanding. 'Figure'is a broad notion: it can be a triangle, but also a number of blank spaces to indicate the length of a word. Furthermore, Polya argues that the student should *desire* the answer: if the teacher wants the student to be interested, he should make his questions relevant, not too easy or too difficult. He states that the question should be presented 'natural'and 'interesting'.

2.1.2 Devising a plan

This is considered to be the most important and difficult part of the four devised steps. (Or as Polya writes it, 'long and tortuous'.) In this stage, the solver must first "find the connection between the data and the unknown. [The solver] may be obliged to consider auxiliary problems if an immediate connection cannot be found", and then "(. . .) should obtain eventually a plan of the solution."

Additionally, the solver could restate the problem; they should not, like an insect, try many times hopelessly flying through a windowpane. Instead, they ought to enrich their conception of the problem. Polya mentions different ways to achieve this, such as seeing the problem from different angles; setting up new, related questions; or ‘mobilization’- considering multiple theories that could be useful.
Because of the continuous process of evolving knowledge about the problem, there could arise the need of introducing auxiliary elements. We find that Polya is somewhat unclear in his explanation when to use these auxiliary elements. On the one hand, he writes that the solvers can add auxiliary elements, even when they do not know how or when to use them. On the other hand, he writes that they should not just be included unhesitatingly.

2.1.3 Carrying out the plan

For Polya, carrying out the plan is ‘boring’process. He says that every step of the plan must be meticulously performed and checked.

2.1.4 Examining the plan

Polya writes that it is important and instructive for the students to look back upon the obtained solution, to ”consolidate their knowledge and develop their ability to solve problems”. This is to be achieved by verification of the answer. Furthermore the plan should be revisited to check if it is possible to achieve the result in a shorter way. Polya also devises to look for related problems that could now be solved or better understood. There remains always something to do.

2.2 ‘How to Solve It’ translated to Olympiad problems

In this section, we give a link between Polyas steps and programming. Polya often uses abstract words like unknown, data, plan, auxiliary element, figure. These words can often be replaced by more concrete cases. In figure 1, we show how the question ’What is the unknown?’ can be used as a basis for problem-solving in many different fields.

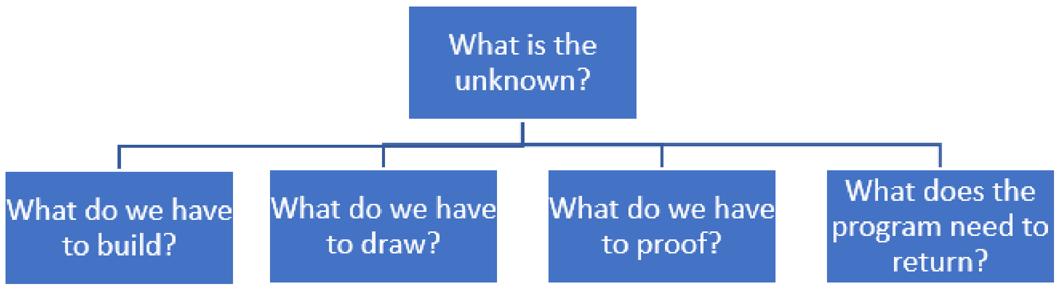

What is the unknown?
What do we have to build?
What do we have to draw?
What do we have to proof?
What does the program need to return?

Figure 1: An abstract question can serve as a basis for more specific questions

2.2.1 Understanding the problem

In programming terminology, Polyas three questions could be restated as:

1. What does the program need to return?
2. What is the input?
3. What should the returned abide by?

Drawing a figure does not have a programming equivalent, but it can still be used. We will give an example while discussing one of the problems.

2.2.2 Devising a plan

Devising a plan looks like writing an answer in pseudocode. We find Polyas idea of looking for related problems remarkable. This is a method that is already used by Computer Scientists often, when one observes the popularity of websites like StackOverflow, which is a question and answer website dedicated to programming. (StackOverflow is the 50st most used website in the world. Users visit it 5.20 hours a day on average.[Ale]) Polyas auxiliary elements could be compared to writing functions in code, that could solve or help to solve a subproblem. Additionally, classes can be seen as auxiliary elements, that help the programmer to store related data in an efficient and comprehensible way.

2.2.3 Carrying out the plan

Carrying out the plan is compared to coding analogous to running the code. Albeit that running code costs in general a lot less effort than making a mathematical calculation by hand. Programmers can keep in mind, that because of this, they can easier test the code they have written so far and recognize early mistakes.

2.2.4 Examining the plan

Examining the plan could be compared to cleaning up the written code. It encapsulates eliminating redundant variables, functions, white lines and other elements; removing comments that are not necessary; and making the solution as simple as possible. It is not a point of focus in this research, but it would be interesting to see how the results of an Olympiad problem are if it has a maximum number of lines.
Testing is also a part of examining the plan. Problems of the Olympiad always have strict specifications. For example, from 2019's question A5: Write a program that reads a number N from standard input ($2 < N < 27$). Probably the cases 2 and 27 do not work or give special results. The answers to the problems are always checked by means of multiple test cases, to examine whether the program really works. If the program does not return the correct results on certain test cases, it does not receive all the points.

2.3 Intermezzo: Problem presentation in the Olympiad

To continue on Polyas idea of making a question desirable, we consider some questions of the Olympiad. They often concern puzzles. Recently, puzzle-based learning was investigated by Falkner, Sooriamurthi and Michalewicz, as a way to attract more students to study computer science, with a focus on solving problems, to deal with the the shortage of a skilled information technology workforce, in a program that what was based on Polyas principles. [FSM10] Using puzzles, they wanted to, next to motivate the students, make sure that they boost their problem-solving skills, without having all the questions all the time being focused on previously explained material, like in textbooks, but more in a general way. Their experiments were well-received and indicated improvement in problem-solving skills.
We have seen that the appearance of the Olympiad questions has changed through the years. This concerns their subject and lay-out. We would like to zoom in on the look of the questions of type A and make a small comparison between the appearance of images next to the questions in general, and the average scores. We choose type A questions, because those are the preliminary questions, that are made by the most people. However, there does not seem to be any connection. An overview of our results is given in 2.

Year	Number of images	Average points type A questions (out of 10)
2015	0	8.9
2016	2	8.3
2017	2	8.2
2018	4	8.2

Table 2: Images in type a questions

2.4 Related Work

'How to Solve It'has proved to be influential to the development of computer science education, especially since the 80's. From then on, he was seen to have put the field of heuristic back on track[New83]. Nowadays How to Solve It is probably the most cited book in the field of mathematical- and computer science education, (11537 citations) and has also been influential to fields as Artificial Intelligence[New83] and behavioral sciences[Kap17].
Gradually his ideas were more incorporated into university curriculums, for example in a curriculum at Carnegie-Mellon[DKM98] and a teaching reform initiative in New Jersey in 1993, which describes a new teaching method for beginning computer scientists. It devises class sessions which are problem driven and which use a strict ordering of classical activities. We find it remarkable, because it puts a lot of focus on the understanding of the given problems in class. Students are only allowed to code if they first completely thought out their plan - confer how people had to use punched cards in the 50s.
This way of programming is also acknowledged by Dalbey and Linn, who call the understanding phase specification, in a literature review about the demands and requirements of programming: "Most researchers describe programming as consisting of a number of identifiable stages.[LD85] (. . .) In general, their analyses describe the following tasks : specification, design, coding, and debugging."

We see Polya's influence in the four task, that look like Polya's four steps.
Allan and Kolesar from Utah University also recommend the approach, saying that otherwise for many novices learning programming, rather than learning the basic concepts of the field, their energies are devoted to learning syntax.[AK96] Furthermore, the link between Plyas four steps and programming was almost literally stated by Pea and Kurland, in a paper about the cognitive effects of learning computer programming to children: "a set of activities is involved in programming for either novices or experts, which constitutes phases of the problem solving process," as been described by Polya.
Since 2017, Polyas problem-solving strategies are included in the official syllabus of the IOI. This syllabus states the participant requirements. [IOI] However, lately there also rises some criticism. Bret Victor warns in a speech about immediate connection, that the contemporary conception of programming, in which a typed-in program needs to be handed to a compiler, looks in essence a lot like during the period when the earlier mentioned punched cards were used. While, he argues, modern computers make a much higher level of interactivity possible. Programming does not need to be as abstract as it is now. At the moment programmers need to be well able to visualize the code they type, before they compile. But that is not necessary. Using modern Integrated Development Environments (IDE's), it possible to have a more immediate connection between the programmer: to have immediately, while typing, changing results on a separate window. In this way, Polyas second and third steps would be more intertwined. [Vic]

2.5 Conclusions concerning Polya

Polya's ideas still have a considerable influence on Computer Science Education. We have seen that his idea of the four-step-plan is quickly mapped to programming and it is a basis for a lot of research on what is the best way to learn novices to program. That is why we think How to solve it is a good guidance to our research. When we are going to examine the Olympiad submissions, we try to find patterns in the code that indicate a Polyan way of thinking. When we observe incorrect submissions, we try to find in which step probably the mistake was made.

3 Implementation

Olympiad questions made in C++ and Python. MySQL Workbench and XAMPP for writing queries and retrieving information from the database. Useful queries can be found in Appendix B Jupyter Notebook, Python, libraries Pandas and Numpy, for further analysis.

4 Analyzing the questions

4.1 Short remarks

We limit ourselves to questions of type A and C, because we do not have access to the answers of type B, as they do mostly not contain code. For example, a type B question is simulate a Dijkstra shortest-path search by hand. We are not interested in questions of type D (programming a game) either, because we believe that students that are able to program a game, already have reasonable programming skills.
It appears that lots of the submissions look like each other. Sometimes a person has handed in more than ten submissions, while only one was correct. We do not know the procedure behind this system of re-attempting, but for problems in which this happens often, it results in misleading answers of the queries. It is very likely that when a contestant has at least one answer right, he understands the question, and it is not necessary to take all his previous failures into account. So we select only the best submission per problem per contestant. The average score of all problems was 0.74.
We first made the questions that we were about to examine ourselves, to have a better understanding of the problem.
We decided to focus on four specific problems: per type of question, one with a high score (bigger than 0.8) and one with a low score (smaller than 0.7), see below 3

Type of question	Average points (from 10)	Name	Year
-	8.4	Meest waardevolle object	-
A	6.4	Verkiezing	2015
C	9.4	Delercode	2016
C	5.4	Waslijnen	2015

Table 3: Analyzed problems and their scores

We separated the incorrect submissions on where we think they would fit best in the four steps of problem-solving. The correct submissions are categorized in a way specific to the problem.
We do not assign mistakes to the carrying-out-phase, as we consider that phase being the task of the compiler, not of the student.

4.2 'Meest waardevolle object'

According to our first queries, this appeared to be the best made Olympiad problem so far. It had a score of 0.89 out of 1. It is from an Olympiad older than 2015. We have no access to problems before 2015, which means we do not know what the problems are. However, this question is still interesting. Firstly, because the solutions are short, so it is easy to process them fast. Secondly, because it was clear what the problem was: to return the highest number of a series of numbers. Probably it was an introductory problem. We encountered four ways the students used solving the problem.

- max var use a variable that represents the highest value so far encountered (initialize it at a very low value). Check per newly observed value if it is higher. If so, update its value.
- max func save all the inputted data in an array. Then use a built-in max() function to let it determine the maximum value.
- Sort save all the inputted data in an array. Sort it in ascending order, and return the last value.
- Max array first save all the data in an array, and then walk through the array like in max var.

Answers

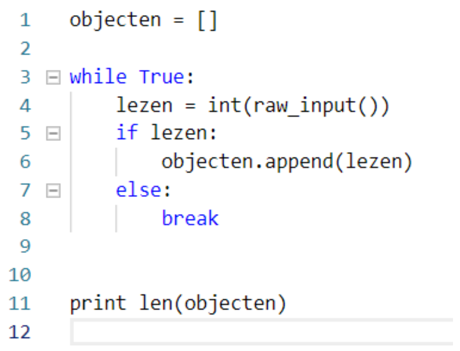
We summarized our findings in table 4. The four different algorithms were used in various languages. We counted how many the algorithms were used per language. We put the numbers of used ways to get the solution vertically against the used language horizontally. The mistakes were of various

	Python	C++	C	Java	C#	Pascal	PHP	Total
max var	12	23	7	8	1	1	1	53
max func	16	3		3	3		2	27
sort	1	3	1	1				6
max array	1	6	1	2	2			12
wrong	3	4	1	1	1			10
Total	30	35	9	14	6	1	3	98

Table 4: Overview 'Het meest waardevolle object'

kinds. We put them here below.

1. Use of a wrong helper function. (`len()` or `size()` instead of `max()`). Examples:



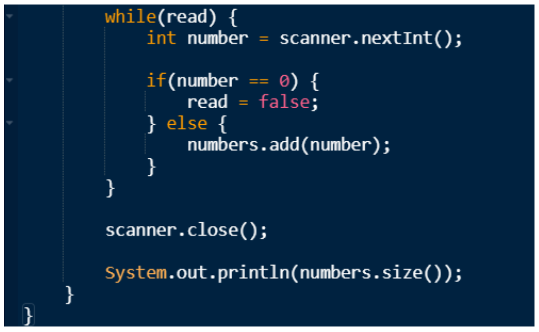
```python
objecten = []

while True:
    lezen = int(raw_input())
    if lezen:
        objecten.append(lezen)
    else:
        break


print len(objecten)

```

(a) Python



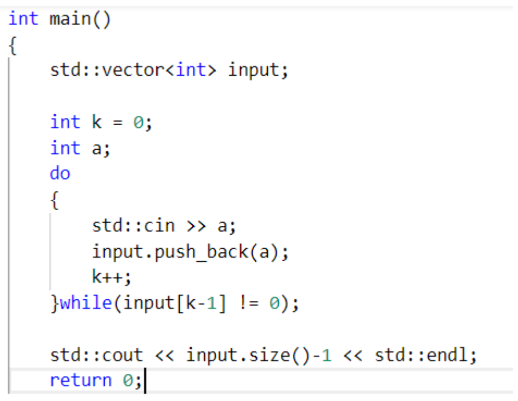
```java
        while(read) {
            int number = scanner.nextInt();

            if(number == 0) {
                read = false;
            } else {
                numbers.add(number);
            }
        }

        scanner.close();

        System.out.println(numbers.size());
    }
}
```

(b) Java



```cpp
int main()
{
    std::vector<int> input;

    int k = 0;
    int a;
    do
    {
        std::cin >> a;
        input.push_back(a);
        k++;
    }while(input[k-1] != 0);

    std::cout << input.size()-1 << std::endl;
    return 0;
```

(c) C++

Figure 2: Illustrations of wrong helper functions.

Probably these students had a wrong understanding of these helper-functions.

2. Output sent to a file instead of std::output

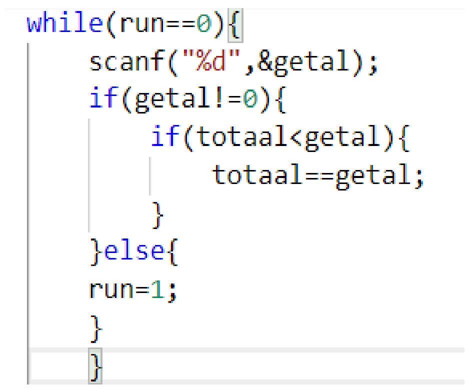
3. C#: Use of the wrong object. Debug instead of Console. (Debug does not write to std::output)

4. Confusion between assignment- and equal-to-operators. See figure 3 This is syntactically correct, but not semantically.

5. Mixing up between string and int-datatypes, for example A (int) >0 (string) see line 7 in the example below (4)

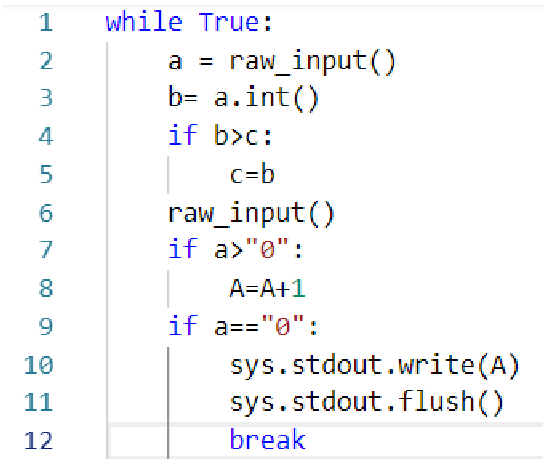
6. In Python: Not save the outcome of `input()`, which returns the input as string, but use `input()` again. Also, see 4

7. No code to process the input. See figure 5 This student forgot to let x be a value from standard input. This could be prevented by testing.



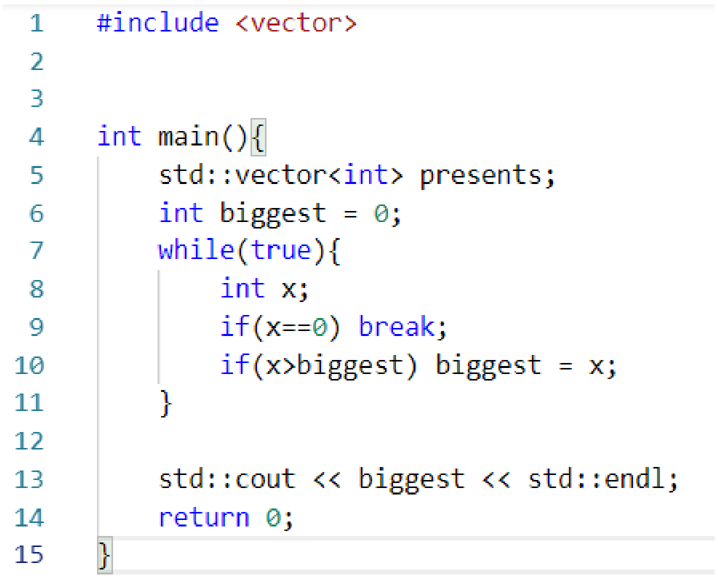
```
while(run==0){
    scanf("%d",&getal);
    if(getal!=0){
        if(totaal<getal){
            totaal==getal;
        }
    }else{
    run=1;
    }
}
```

Figure 3: == instead of =



```
while True:
    a = raw_input()
    b= a.int()
    if b>c:
        c=b
    raw_input()
    if a>"0":
        A=A+1
    if a=="0":
        sys.stdout.write(A)
        sys.stdout.flush()
        break
```

Figure 4: Unclear comparison with string types



```
#include <vector>


int main(){
    std::vector<int> presents;
    int biggest = 0;
    while(true){
        int x;
        if(x==0) break;
        if(x>biggest) biggest = x;
    }

    std::cout << biggest << std::endl;
    return 0;
}
```

Figure 5: No code to process the input

Discussion

Not all students understand what the problem was, Polya's step one. Some returned the length of the input-array instead of the maximum value (mistake 1). This could be helped by asking *What is the unknown?* - the maximum value of the array. Some programs did not have any output (mistake 2 and 3). They could have asked: *What is the condition?* - The maximum value should be printed. There were no signs of trouble with step two, devising a plan, and step three, carrying out the plan. But some students missed step four, the looking-back-phase (mistake 4-7). All these algorithms returned the wrong answers, but they could be repaired by small changes. For example by changing == into = (mistake 4). Or by changing `int x` into `x = int(input())`.

4.3 'Verkiezing'

This problem simulates an election. Every voter has a favorite list. The winning candidate should have more than half of the votes. If there is no winner, the candidate with the least number of votes must leave. If there are two or more candidates, the last candidate must leave. If he is on a favorite list, the next candidate on his list will be selected. We made this question ourselves, and encountered some things that contestants could find difficult:

- It is a long question, there are relatively many rules the program has to abide by.
- When all code is put in the same function, (for example the main function), that function could consist of many lines and statements, and the program could be hard to read. Use of functions could make things clearer.
- When arrays are used, it is wise to use 1-indexing instead of 0-indexing which means that the first element of the array is at position 1. It is convenient to use extra arrays, one to count the votes per contestant and one to keep track of the contestants that where voted out. When 0-indexing is used, everything becomes more complicated, when for example contestant 1 is placed on position 0, contestant 2 on position 1 etc.

There were 62 submissions. We decided to limit ourselves to analyzing the submissions made in Python. In table 5, an overview is shown of how the question was made in different languages.

	Mean score (out of 40)	Median (out of 40)	Amount of submissions
Python	23,7	20	27
C++	27,14	40	14
HS	35	35	2
PHP	27,5	35	4
C#	23,33	20	3
Java	26,25	30	8
C	35	40	4
			Total: 62

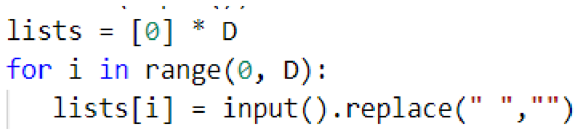
Table 5: Overview 'Verkiezing'

It is noticeable that the medians of C++ and C were of 40 points, the maximum score. The relatively low mean score at C++ indicates that the C++ programmers that did not have a perfect score, immediately had a comparatively much lower score, and probably had difficulties with the array-management. Haskell has a very good score, but only 2 submissions.

Answers

Solutions to this problem all look like each other. Instead of the previously described waslijnen-problem, which leaves much space for the solver, here all contestants have to follow the same steps, namely, the rules of the election. That is why we decided not to classify the good solutions.
27 Submissions were made in Python. 10 of them had a perfect score.
We put the kind of submissions that were downgraded below.

1. Not giving right answers on all test cases. For example, a contestant used the following code to put the input in a list 6: However, this program only works for < 10 candidates. According



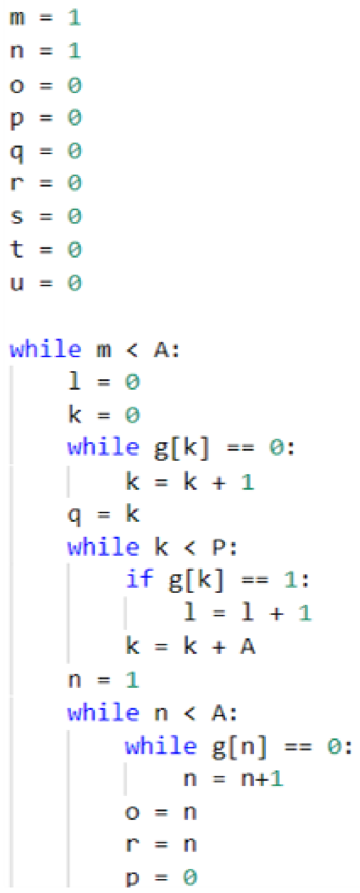
```
lists = [0] * D
for i in range(0, D):
    lists[i] = input().replace(" ","")
```

Figure 6: Wrong input handling

   to the problem statement, there can be up to 20 candidates. Here, the replace-function deletes spaces between numbers. So candidate 10 would not be separated as the number 10, but as 1 and 0. There were nine submissions that worked for the example case, but not for some test cases. Not working on all test cases sometimes overlaps with other types of mistakes (see mistake 3 and mistake 5).

2. Working well but did not get all the points for an unclear reason.

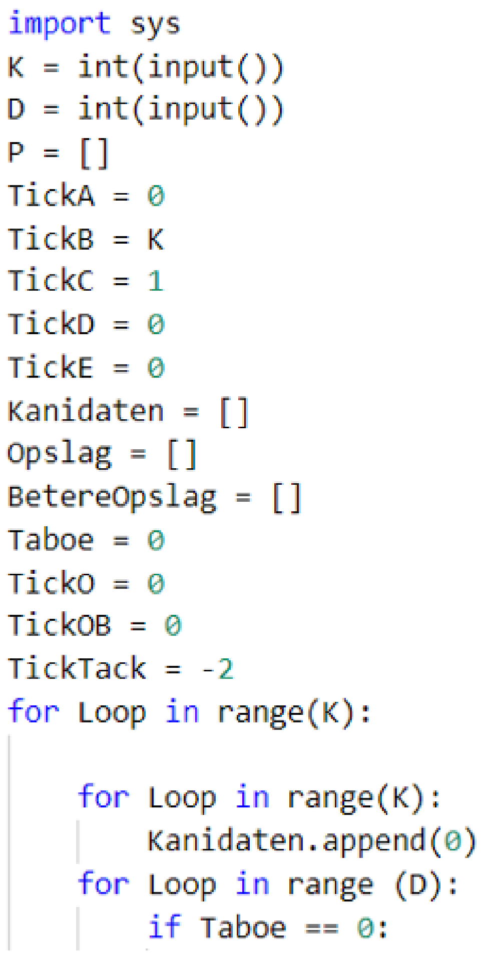
3. Incomprehensible code as result of bad variable names. Two examples 7:



```
m = 1
n = 1
o = 0
p = 0
q = 0
r = 0
s = 0
t = 0
u = 0

while m < A:
    l = 0
    k = 0
    while g[k] == 0:
        k = k + 1
    q = k
    while k < P:
        if g[k] == 1:
            l = l + 1
        k = k + A
    n = 1
    while n < A:
        while g[n] == 0:
            n = n+1
        o = n
        r = n
        p = 0
```

(a)



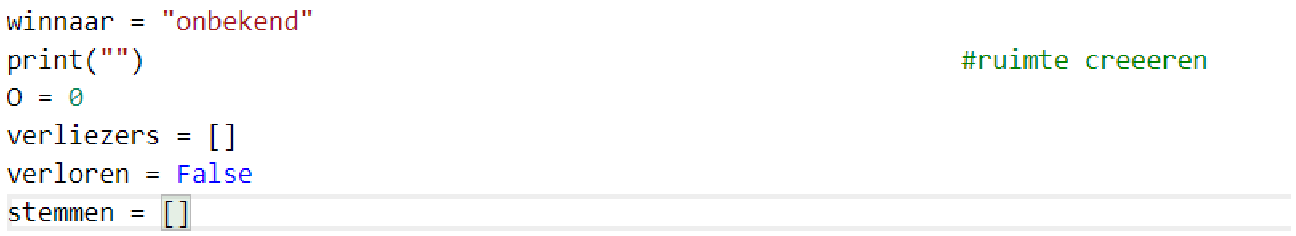
```
import sys
K = int(input())
D = int(input())
P = []
TickA = 0
TickB = K
TickC = 1
TickD = 0
TickE = 0
Kanidaten = []
Opslag = []
BetereOpslag = []
Taboe = 0
TickO = 0
TickOB = 0
TickTack = -2
for Loop in range(K):

    for Loop in range(K):
        Kanidaten.append(0)
    for Loop in range (D):
        if Taboe == 0:
```

(b)

Figure 7: Illustrations of incomprehensible variable names.

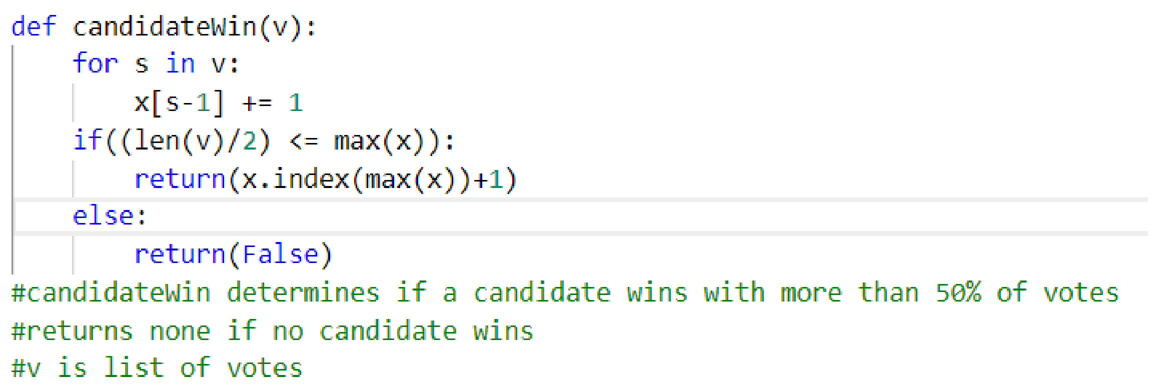
4. Wrongly formatted output. Printing of an extra line (`print()`) in figruimtecre.



```
winnaar = "onbekend"
print("")                                               #ruimte creeeren
O = 0
verliezers = []
verloren = False
stemmen = []
```

Figure 8: Printing of an extra line

5. 5. Incorrect application of the election-rules. For example, removing of the contestant with the smallest number (instead of the highest number). Or removing the first contestant with less votes than the maximum votes (this strategy works accidentally on the test case). Or, declaring a winner when he has half of the votes (in stead of more than half). Example 9:

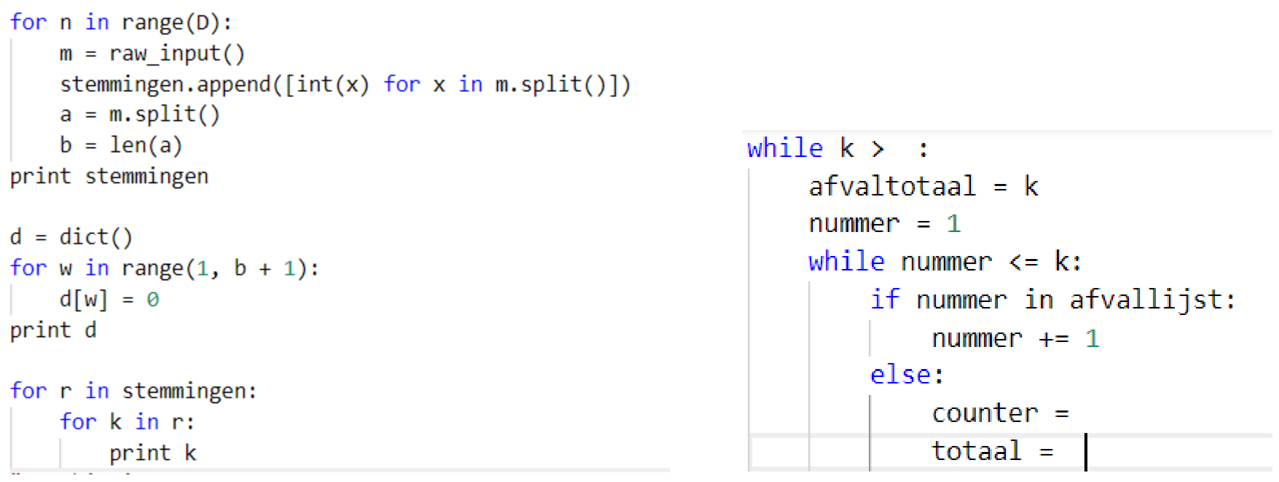


```python
def candidateWin(v):
    for s in v:
        x[s-1] += 1
    if((len(v)/2) <= max(x)):
        return(x.index(max(x))+1)
    else:
        return(False)
#candidateWin determines if a candidate wins with more than 50% of votes
#returns none if no candidate wins
#v is list of votes
```

Figure 9: Wrong winner declaration

It should be `if((len(v)/2) < max(x))`.

6. Incomplete code. For example 10:



```python
for n in range(D):
    m = raw_input()
    stemmingen.append([int(x) for x in m.split()])
    a = m.split()
    b = len(a)
print stemmingen

d = dict()
for w in range(1, b + 1):
    d[w] = 0
print d

for r in stemmingen:
    for k in r:
        print k
```

(a) Multiple print-statements

```python
while k >  :
    afvaltotaal = k
    nummer = 1
    while nummer <= k:
        if nummer in afvallijst:
            nummer += 1
        else:
            counter =
            totaal =
```

(b) Empty declarations

Figure 10: Illustrations of incomplete code

7. Submission of a different problem

Discussion

There were some submissions that did not show a good understanding of the problem. For example, they printed extra lines (mistake 4). This could have been prevented if they had asked: What is the unknown? because the program only needs to output the number of the winner. The incorrect application of the election-rules could have been avoided by asking What is the condition? (mistake 5), also part of step one. In step two, devising a plan, the problem-solvers should also be aware, that subsequently they have to check their results. When submissions have bad variable names (mistake 3), this becomes difficult. A few algorithms passed over step four: they did not give good results on all test cases (mistake 1). The contestants did not have many difficulties with the indexing of the arrays, as we hypothesized. We think an average score of 0.64 is quite low for an introductory question. However, most mistakes were made not because of the difficulty of the problem, but because of misunderstanding of the question, and the negligence of test cases.

4.4 'Waslijnen'

In this problem, there are several washing lines and some pieces of clothing. The question is: how to hang up all the garments while using the least amount of washing lines. The program has a time limit of 2 seconds. It is a difficult problem; it is practically a restatement of the bin-packing problem, which is NP-complete. [KK82] [Fil07]
The time limit forces the student to use a different way of solving than brute-forcing. Fortunately, there are some restrictions on the length of the washing lines (B) and width of the garments (N): it should work for cases where $5 < B < 1000$ and $3 < N < 20$. Furthermore, it should work for a group where $5 < B < 65$ en $20 < N < 40$.
To solve this problem, it would be useful for the students to know about NP-completeness, the bin packing problem, or algorithms like dynamic programming. If the student does not know about this, he could find it difficult where to start, 'to devise a plan', and to search for related problems. On the other hand, because it forces the students to think without foreknowledge or textbooks, the students have to come up an answer by themselves, like they should also be able to when they have finished their studies. [FSM10]
The mean score of this problem was 0.55 out of 1.

	Mean score (out of 100)	Median score (out of 100)	Amount of submissions
Python	47,27	30	26
C++	66,15	70	13
Haskell	93,33	100	3
PHP	65	65	2
Pascal	20,9	30	12
C#	70	70	2
Java	74,29	100	7
C	96,66	100	3
			Total: 68

Table 6: Scores of 'Waslijnen' in different programming languages

We decided to focus on the submissions made in Python.

Answers

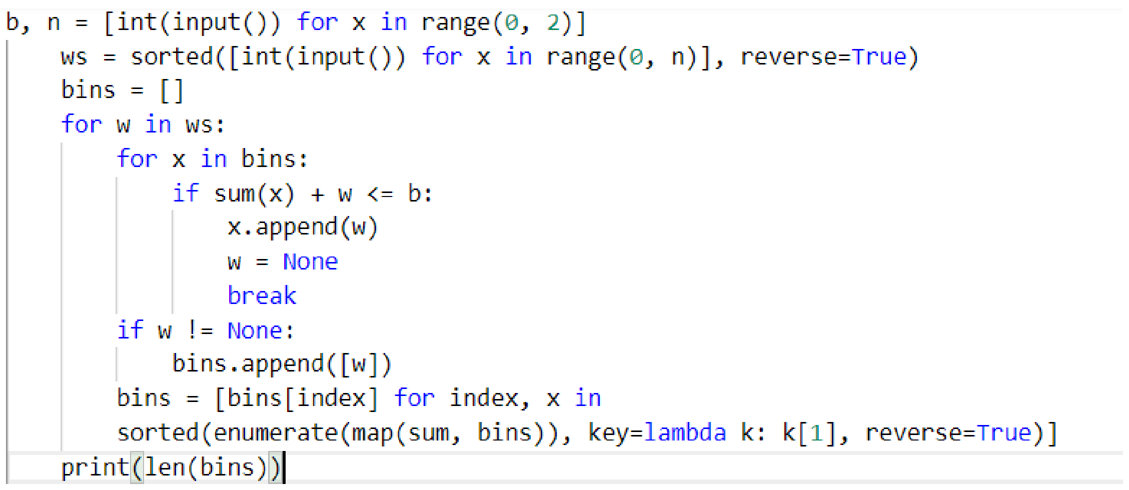
We have discerned multiple ways the students used to solve this problem, summarized in table 7

1. Next fit decreasing: try to fill up the lines one for one, by taking always the smallest remaining garment and hang that on the current line. This was used by one student, who made a small mistake by writing if `sum(clothesline)+min(nb) < int(B):` instead of : `if sum(clothesline)+min(nb) <= int(B):` so he could never fill up his lines completely.

Method	Used by _ contestants	Points
1. Next fit decreasing	1	0
2. First fit decreasing	6	100
3. Combinations	3	60/30
4. Random	1	70
5. Trick	7	20/30
6. Incomplete	4	0
7. Other	3	100
	Totaal: 25	

Table 7: Overview 'Waslijnen'

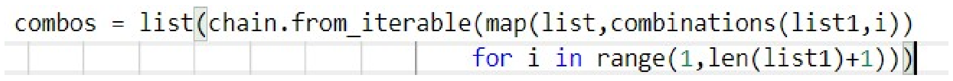
2. First fit decreasing: Sort the garments in descending order. After placing one on a line, sort all the lines with garments in descending order. So, every time the largest garment left is placed in the largest possible bin. However, this approach does not work all the time. For example, when you take the following six garments, with a washing line length of 200: 100, 60, 60, 60, 50, 50, this algorithm gets you 3 washing lines, with bins (100, 60) (60, 60, 50) (50) but the optimal solution is 2, with bins (100, 50, 50), (60, 60, 60). Thus, this solution is not an optimal solution. Nonetheless, it works for all the test cases, so contestants that used this approach generally got all the points. As an example, see 11.



```
b, n = [int(input()) for x in range(0, 2)]
    ws = sorted([int(input()) for x in range(0, n)], reverse=True)
    bins = []
    for w in ws:
        for x in bins:
            if sum(x) + w <= b:
                x.append(w)
                w = None
                break
        if w != None:
            bins.append([w])
        bins = [bins[index] for index, x in
        sorted(enumerate(map(sum, bins)), key=lambda k: k[1], reverse=True)]
    print(len(bins))
```

Figure 11: Example First-fit decreasing algorithm

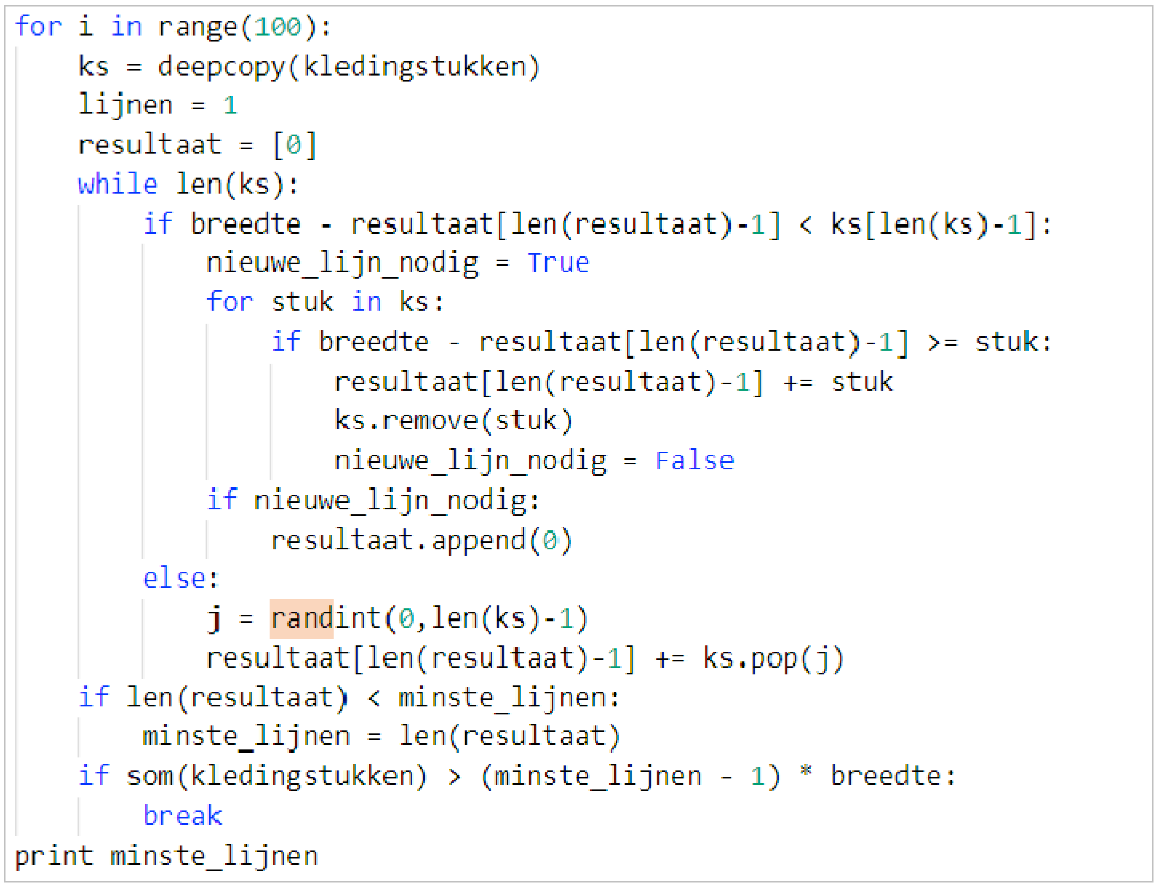
3. Combinations: This solution uses a combinations-function to list all combinations of garments from 1 to the total number of garments. The idea is, that among these combinations are the right solutions. See figure 12Keep up a counter. While there are garments left, look for the greatest fitting combination, delete all the garments belonging to that combination, add 1 to the counter. However, when the number of garments increases, so does the number of combinations. Then this algorithm does not stay in the time limit.



```
combos = list(chain.from_iterable(map(list,combinations(list1,i))
                                  for i in range(1,len(list1)+1)))
```

Figure 12: Characteristic detail of a ’combinations’-algorithm

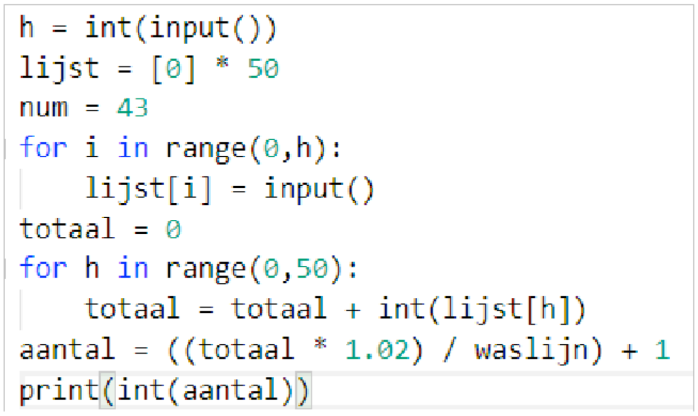
4. Random: Build up solutions 100 times, each time beginning with adding random garments until their sum is bigger than B the biggest value. Then add up the remaining clothes, beginning with the smallest. When the next garment does not fit anymore, use the same strategy for the remaining clothes. Every time the present solution is compared with the best so far, and eventually it replaces the best. This algorithm could, because of its application of randomness, give an optimal solution for every case, but also fail. See figure 13



```
for i in range(100):
    ks = deepcopy(kledingstukken)
    lijnen = 1
    resultaat = [0]
    while len(ks):
        if breedte - resultaat[len(resultaat)-1] < ks[len(ks)-1]:
            nieuwe_lijn_nodig = True
            for stuk in ks:
                if breedte - resultaat[len(resultaat)-1] >= stuk:
                    resultaat[len(resultaat)-1] += stuk
                    ks.remove(stuk)
                    nieuwe_lijn_nodig = False
            if nieuwe_lijn_nodig:
                resultaat.append(0)
        else:
            j = randint(0,len(ks)-1)
            resultaat[len(resultaat)-1] += ks.pop(j)
    if len(resultaat) < minste_lijnen:
        minste_lijnen = len(resultaat)
    if som(kledingstukken) > (minste_lijnen - 1) * breedte:
        break
print minste_lijnen
```

Figure 13: Example of a ’random’-algorithm

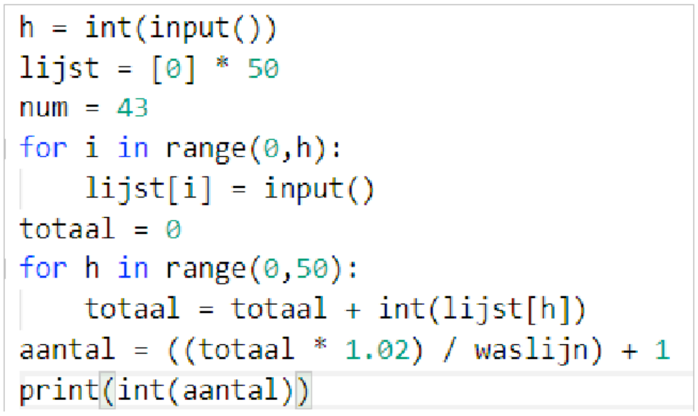
5. Trick: This solution divides the total length of all the clothes by their quantity and rounds up. There were some variants, see for example figure 14, in which the total is multiplied by 1.02.



```
h = int(input())
lijst = [0] * 50
num = 43
for i in range(0,h):
    lijst[i] = input()
totaal = 0
for h in range(0,50):
    totaal = totaal + int(lijst[h])
aantal = ((totaal * 1.02) / waslijn) + 1
print(int(aantal))
```

Figure 14: Example of a 'trick'-algorithm

6. Incomplete: these were incomplete solutions, often containing comments and too many print statements. See figure 15, which for example contains redundant print-statements.



```
h = int(input())
lijst = [0] * 50
num = 43
for i in range(0,h):
    lijst[i] = input()
totaal = 0
for h in range(0,50):
    totaal = totaal + int(lijst[h])
aantal = ((totaal * 1.02) / waslijn) + 1
print(int(aantal))
```

Figure 15: Example of an incomplete solution

7. 7. Other: These were mostly adjustments of the first fit decreasing algorithm. One solution was for example: First search for the biggest suitable value, then check whether there are not two possible values bigger. If no garment can be added, take a new line and repeat until all the garments are hung up.

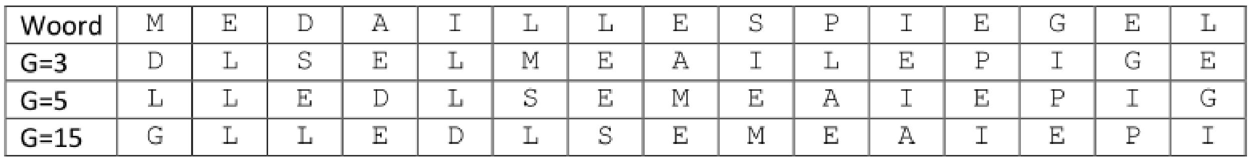
Discussion

All of the students who sent in a working program passed Polyas step one. They understood what the problem was. They were aware that they could not have found an optimal solution. The most trouble here contestants had was to devise a plan. Except for them who used the trick or whose solutions were incomplete, the contestants solved a subproblem: how to optimally fill one line? Polya calls this decomposing. They found out different ways: By continuously adding the largest garment possible (first fit decreasing); by searching through all possible combinations of the clothes and see which ones have the least space left (combinations); or by trying randomly (randomly).

Then they built up their algorithms accordingly. But when all contestants (except the random algorithm) had tested more, Polyas step four, they would have found out that their algorithms did not work for all cases.

4.5 'Delercode'

The question of type c with the highest average score was delercode: 0.95 to 1. The objective is to revert a series of swaps that change a word of length N to another, following a specific rule: for numbers G from 2 to N, if N is divisible by G, put all letters on positions G and multiples first, then follow up the rest. There is given one example 16:



Woord	M	E	D	A	I	L	L	E	S	P	I	E	G	E	L
G=3	D	L	S	E	L	M	E	A	I	L	E	P	I	G	E
G=5	L	L	E	D	L	S	E	M	E	A	I	E	P	I	G
G=15	G	L	L	E	D	L	S	E	M	E	A	I	E	P	I

Figure 16: Example 'Delercode'

Below is a table 8 that shows how the question was made in different programming languages.

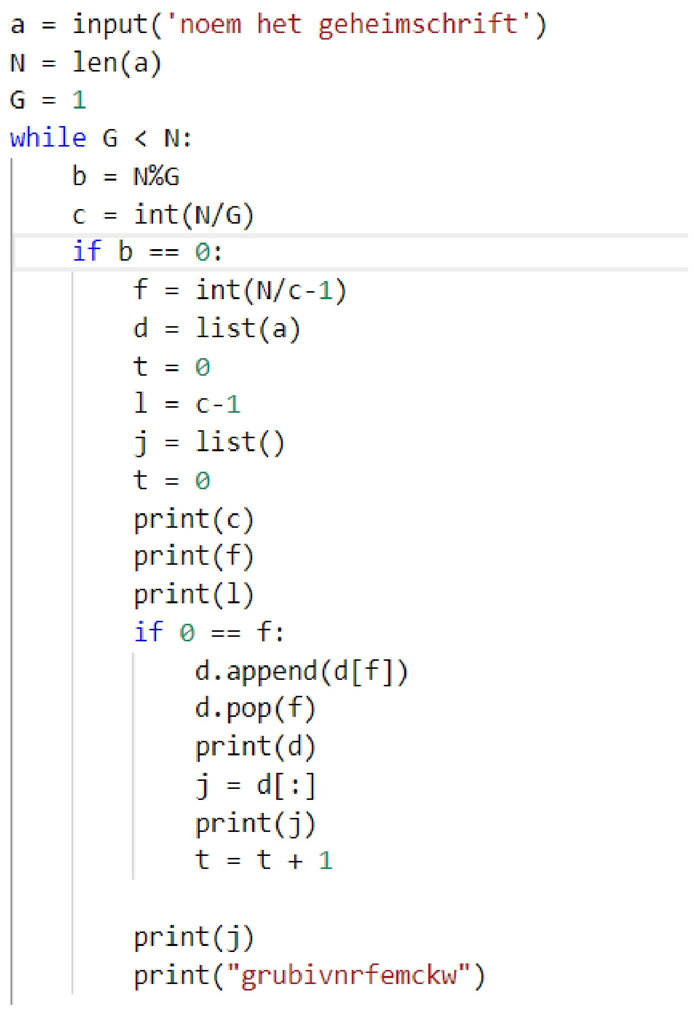
Language	Mean (out of 100)	Median (out of 100)	Amount of submissions
Python	89.15	100.0	50
C++	99.58	100.0	24
C	94.29	100.0	7
Java	100.0	100.0	15
C#	100.0	100.0	6
PHP	100.0	100.0	3
Haskell	100.0	100.0	3

Table 8: Overview 'Delercode'

Answers

Only eight submissions of 110 were not given all the points, four of those had zero points. Because this problem is in essence to program the retrograde of a set of given rules, the good submissions do not differ much: for all contestants these rules are the same, and have to be used in the same order. Below we have written down the made mistakes.

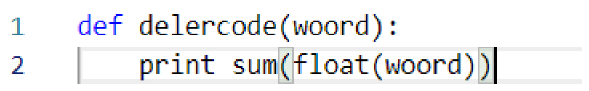
1. 1. Incomplete code. Two examples:



```
a = input('noem het geheimschrift')
N = len(a)
G = 1
while G < N:
    b = N%G
    c = int(N/G)
    if b == 0:
        f = int(N/c-1)
        d = list(a)
        t = 0
        l = c-1
        j = list()
        t = 0
        print(c)
        print(f)
        print(l)
        if 0 == f:
            d.append(d[f])
            d.pop(f)
            print(d)
            j = d[:]
            print(j)
            t = t + 1

        print(j)
        print("grubivnrfemckw")
```

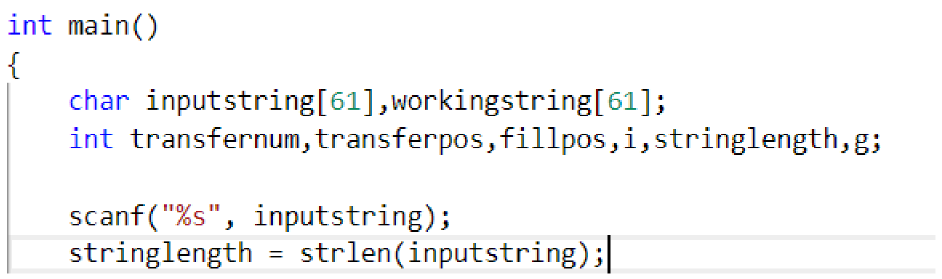
(a) This algorithm shows signs of a plan: there is a loop with goes from N to 1, and a N mod G operation, which makes sure only the right indices are selected. c is made the number that divides N, so also the right divisors are selected. But then there is no follow-up.



```
def delercode(woord):
    print sum(float(woord))
```

(b) b) This submission consisted of this function only.

2. Not working on all test cases. Example 17:



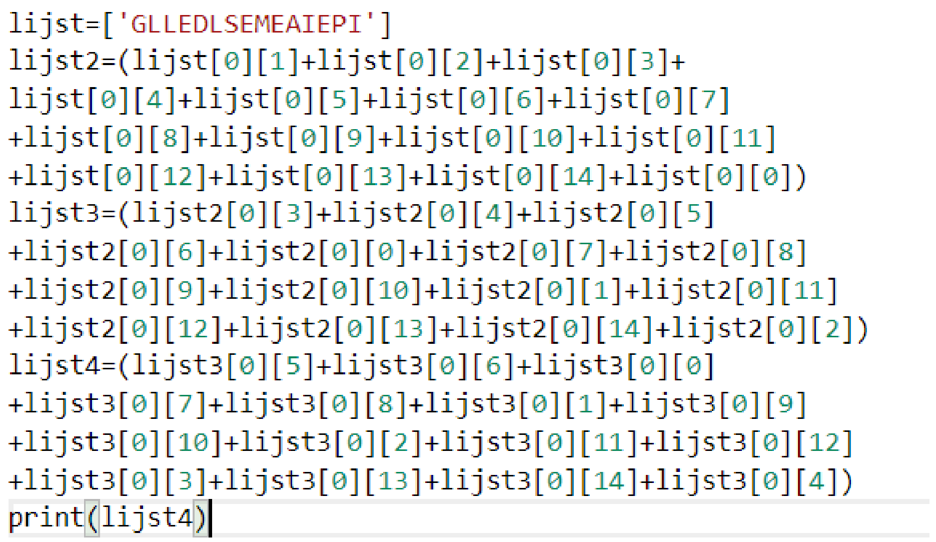
```
int main()
{
    char inputstring[61],workingstring[61];
    int transfernum,transferpos,fillpos,i,stringlength,g;

    scanf("%s", inputstring);
    stringlength = strlen(inputstring);
```

Figure 17: No array instantiating

   The student forgot to zero his arrays. The arrays inputstring and workingstring are not zero-instantiated, which for certain test-cases leads to the occurrence of random extra symbols in his output-string.

3. Hardcoding the rules.



```
lijst=['GLLEDLSEMEAIEPI']
lijst2=(lijst[0][1]+lijst[0][2]+lijst[0][3]+
lijst[0][4]+lijst[0][5]+lijst[0][6]+lijst[0][7]
+lijst[0][8]+lijst[0][9]+lijst[0][10]+lijst[0][11]
+lijst[0][12]+lijst[0][13]+lijst[0][14]+lijst[0][0])
lijst3=(lijst2[0][3]+lijst2[0][4]+lijst2[0][5]
+lijst2[0][6]+lijst2[0][0]+lijst2[0][7]+lijst2[0][8]
+lijst2[0][9]+lijst2[0][10]+lijst2[0][1]+lijst2[0][11]
+lijst2[0][12]+lijst2[0][13]+lijst2[0][14]+lijst2[0][2])
lijst4=(lijst3[0][5]+lijst3[0][6]+lijst3[0][0]
+lijst3[0][7]+lijst3[0][8]+lijst3[0][1]+lijst3[0][9]
+lijst3[0][10]+lijst3[0][2]+lijst3[0][11]+lijst3[0][12]
+lijst3[0][3]+lijst3[0][13]+lijst3[0][14]+lijst3[0][4])
print(lijst4)
```

Figure 18: Hard-coding

4. A solution to a different problem

Discussion

All the contestants seem to have understood the problem. Not everyone was able to devise a working plan. This is the case for both the two incomplete examples (mistake 1) and the hardcoded example (mistake 3). They could have asked for related problems, or ask: Could you solve a part of the problem? For example, to build up a word from another word by putting the letters on even positions first, and the letters on odd positions second. Solving a part of the problem could be useful if some parts of the solution seem more accessible than others. The contestants could also have drawn a figure, showing where the letters have to go, for example like in figure 19 with a word

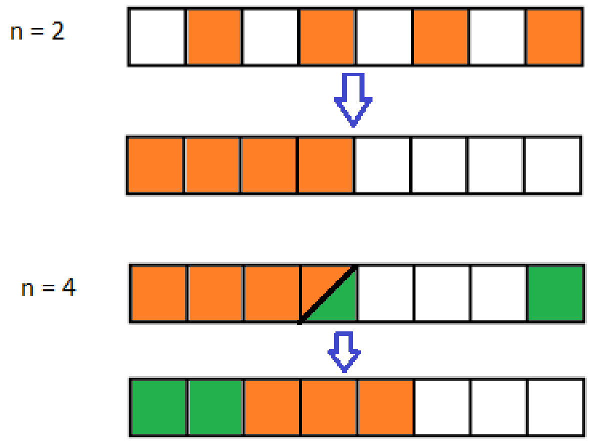


Figure 19: Drawing a figure like this could give more insight in the question

of eight letters. The students then might see that they have to reverse this process from the bottom to the top. Polyas step four, looking back, was neglected by submissions as mistake 2, which do not work on all test cases.

This problem was made much better than the other type c question, waslijnen. It had an average score of 9.4 against 5.4. We argue that this question was too easy. Firstly, because of the scores. Secondly, because while Delercode is about translating a procedure into code, thus limiting the students in their approach, waslijnen is an open problem, requiring a more creative approach. This is confirmed by waslijnen having more different solutions.

Delercode was also made better than the two problems of type a. When we compare it to Verkiezing (which had an average score of 0.64), the problems look like each other, because they both have to do with coding a set of requirements. We find it remarkable that there were made so much more mistakes in the test cases in Verkiezing. There was only one submission in Delercode that did not work on all test cases, and that was because of uninitialized arrays. When a contestant has a solution to one test case, it most likely works for all test cases. That was different in Verkiezing. In all specifications (to declare a winner when one candidate has more than half of the votes, to eliminate the last candidate with the least votes, to let the maximum number of candidates be 20 and participants 1000) there was at least made one mistake.

5 Conclusion and future prospects

In this paper, we have studied the work of George Polya about heuristics. In order to solve a problem, Polya proposes to adhere to four steps: understanding the problem, devising a plan, carrying out the plan and looking back. We put these four steps in a context closer to informatics. Afterwards we observed four problems of the IOI, and looked for patterns in the submissions that were commensurable with the four steps.
In the submissions of problems of type c, the advanced problems, it was observed that the contestants did not encounter problems with understanding the problem. However, we did in Het meest waardevolle object and Verkiezing, both problems that were meant to be introductory exercises. Mostly because the contestants made small mistakes, such as having more output than what is specified. It is interesting that people seem to make more little mistakes when making easier questions. This could be investigated further.
Devising a plan appeared to be the most difficult in the type c problem Waslijnen, as expected. The solutions were more varied and there were more incomplete solutions compared to questions of type a.
In all problems there were contestants that did not look back, as multiple submissions failed to give good results on all the test cases. Further investigation is necessary to see if better results would be obtained, if contestants are explicitly warned to check the border cases.
Waslijnen, a problem of type c, did not have a high score because the problem was in essence unsolvable. However, we think that the problem was also valuable, as the solvers came up with diverse and creative answers.
We conclude that the problem-solving method of Polya proved to be a model that was well able to cover up the strategies and mistakes we observed. In the future, analyses as conducted in this paper could be carried out on a larger scale. Submissions in other programming languages of the same problems that we have observed could be explored as well. Subsequently it would be interesting to analyze other problems. Furthermore, there could be more comparisons between questions of type a and type c, to be able to predict whether a question is difficult or not.

References

[AK96] VH Allan and MV Kolesar. Teaching computer science: A problem solving approach that works. 1996.

[Ale] Alexa.com. Stackoverflow.com. https://www.alexa.com/siteinfo/stackoverflow.com. Accessed: 2019-06-27.

[DKM98] FadiP Deek, Howard Kimmel, and James A McHugh. Pedagogical changes in the delivery of the first-course in computer science: Problem solving, then programming. *Journal of Engineering Education*, 87(3):313–320, 1998.

[Fil07] Carlo Filippi. On the bin packing problem with a fixed number of object weights. *European Journal of Operational Research*, 181(1):117–126, 2007.

[FSM10] Nickolas Falkner, Raja Sooriamurthi, and Zbigniew Michalewicz. Puzzle-based learning for engineering and computer science. *Computer*, (4):20–28, 2010.

[IOI] IOI Informatics. Syllabus. https://ioinformatics.org/files/regulations18.pdf. Accessed: 2019-06-28.

[Kap17] Abraham Kaplan. *The conduct of inquiry: Methodology for behavioural science*. Routledge, 2017.

[KK82] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 312–320, Nov 1982.

[LD85] Marcia C Linn and John Dalbey. Cognitive consequences of programming instruction: Instruction, access, and ability. *Educational Psychologist*, 20(4):191–206, 1985.

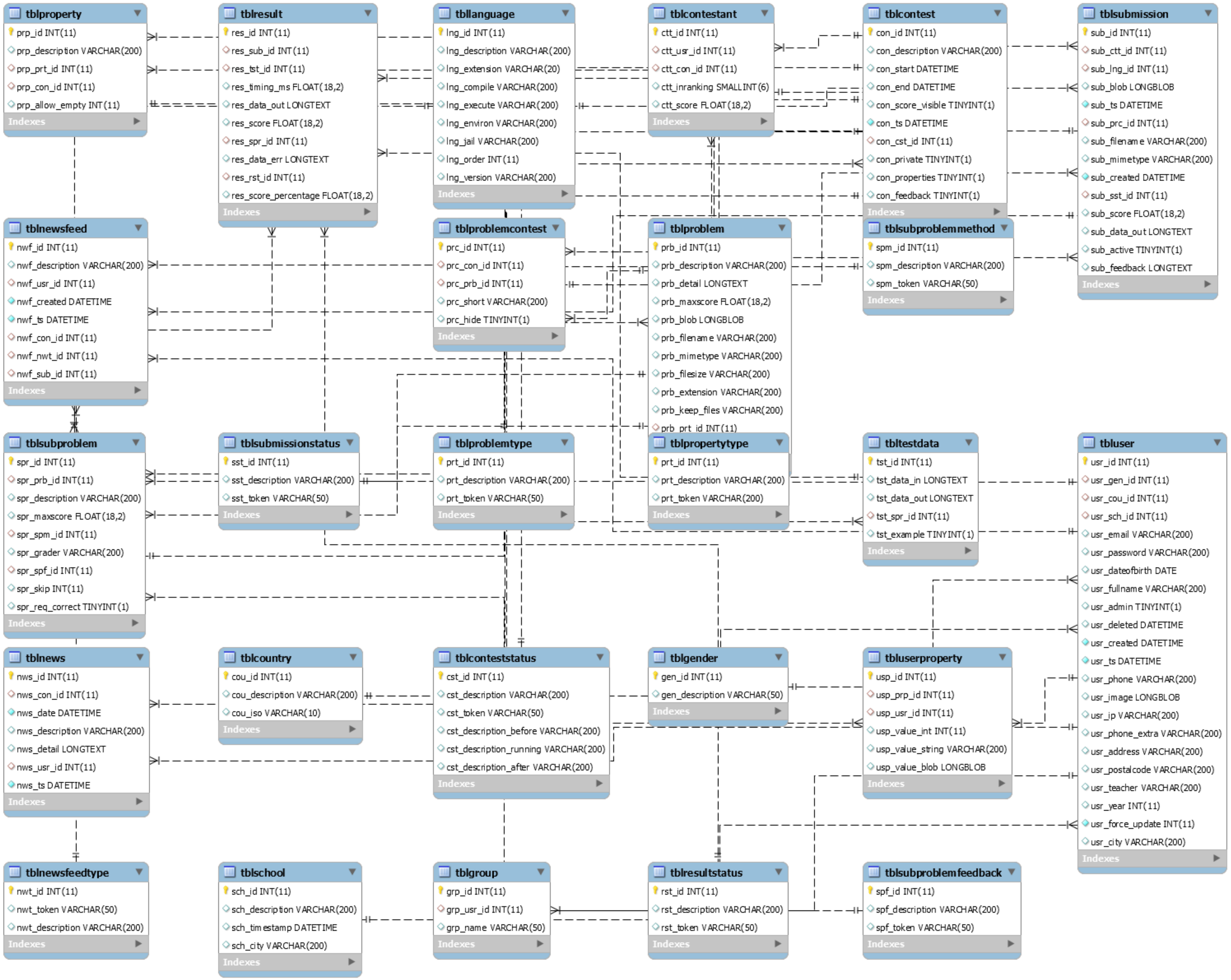
[New83] Allen Newell. The heuristic of george polya and its relation to artificial intelligence. *Methods of heuristics*, pages 195–243, 1983.

[PK17] Sarantos Psycharis and Maria Kallia. The effects of computer programming on high school students reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, 45(5):583–602, 2017.

[Vic] Bret Victor. Inventing on principle. https://www.youtube.com/watch?v=8QiPFmIMxFc. Accessed: 2019-06-25.

[wis] Wiskunde Olympiade. https://www.wiskundeolympiade.nl/. Accessed: 2019-06-28.

A EER diagram Olympiad Database


tblproperty
prp_id INT(11)
prp_description VARCHAR(200)
prp_prt_id INT(11)
prp_con_id INT(11)
prp_allow_empty INT(11)
Indexes
tblresult
res_id INT(11)
res_sub_id INT(11)
res_tst_id INT(11)
res_timing_ms FLOAT(18,2)
res_data_out LONGTEXT
res_score FLOAT(18,2)
res_spr_id INT(11)
res_data_err LONGTEXT
res_rst_id INT(11)
res_score_percentage FLOAT(18,2)
Indexes
tbllanguage
lng_id INT(11)
lng_description VARCHAR(200)
lng_extension VARCHAR(20)
lng_compile VARCHAR(200)
lng_execute VARCHAR(200)
lng_environ VARCHAR(200)
lng_jail VARCHAR(200)
lng_order INT(11)
lng_version VARCHAR(200)
Indexes
tblcontestant
ctt_id INT(11)
ctt_usr_id INT(11)
ctt_con_id INT(11)
ctt_inranking SMALLINT(6)
ctt_score FLOAT(18,2)
Indexes
tblcontest
con_id INT(11)
con_description VARCHAR(200)
con_start DATETIME
con_end DATETIME
con_score_visible TINYINT(1)
con_ts DATETIME
con_cst_id INT(11)
con_private TINYINT(1)
con_properties TINYINT(1)
con_feedback TINYINT(1)
Indexes
tblsubmission
sub_id INT(11)
sub_ctt_id INT(11)
sub_lng_id INT(11)
sub_blob LONGBLOB
sub_ts DATETIME
sub_prc_id INT(11)
sub_filename VARCHAR(200)
sub_mimetype VARCHAR(200)
sub_created DATETIME
sub_sst_id INT(11)
sub_score FLOAT(18,2)
sub_data_out LONGTEXT
sub_active TINYINT(1)
sub_feedback LONGTEXT
Indexes
tblnewsfeed
nwf_id INT(11)
nwf_description VARCHAR(200)
nwf_usr_id INT(11)
nwf_created DATETIME
nwf_ts DATETIME
nwf_con_id INT(11)
nwf_nwt_id INT(11)
nwf_sub_id INT(11)
Indexes
tblproblemcontest
prc_id INT(11)
prc_con_id INT(11)
prc_prb_id INT(11)
prc_short VARCHAR(200)
prc_hide TINYINT(1)
Indexes
tblproblem
prb_id INT(11)
prb_description VARCHAR(200)
prb_detail LONGTEXT
prb_maxscore FLOAT(18,2)
prb_blob LONGBLOB
prb_filename VARCHAR(200)
prb_mimetype VARCHAR(200)
prb_filesize VARCHAR(200)
prb_extension VARCHAR(200)
prb_keep_files VARCHAR(200)
prb_prt_id INT(11)
tblsubproblemmethod
spm_id INT(11)
spm_description VARCHAR(200)
spm_token VARCHAR(50)
Indexes
tblsubproblem
spr_id INT(11)
spr_prb_id INT(11)
spr_description VARCHAR(200)
spr_maxscore FLOAT(18,2)
spr_spm_id INT(11)
spr_grader VARCHAR(200)
spr_spf_id INT(11)
spr_skip INT(11)
spr_req_correct TINYINT(1)
Indexes
tblsubmissionstatus
sst_id INT(11)
sst_description VARCHAR(200)
sst_token VARCHAR(50)
Indexes
tblproblemtype
prt_id INT(11)
prt_description VARCHAR(200)
prt_token VARCHAR(50)
Indexes
tblpropertytype
prt_id INT(11)
prt_description VARCHAR(200)
prt_token VARCHAR(200)
Indexes
tbltestdata
tst_id INT(11)
tst_data_in LONGTEXT
tst_data_out LONGTEXT
tst_spr_id INT(11)
tst_example TINYINT(1)
Indexes
tbluser
usr_id INT(11)
usr_gen_id INT(11)
usr_cou_id INT(11)
usr_sch_id INT(11)
usr_email VARCHAR(200)
usr_password VARCHAR(200)
usr_dateofbirth DATE
usr_fullname VARCHAR(200)
usr_admin TINYINT(1)
usr_deleted DATETIME
usr_created DATETIME
usr_ts DATETIME
usr_phone VARCHAR(200)
usr_image LONGBLOB
usr_ip VARCHAR(200)
usr_phone_extra VARCHAR(200)
usr_address VARCHAR(200)
usr_postalcode VARCHAR(200)
usr_teacher VARCHAR(200)
usr_year INT(11)
usr_force_update INT(11)
usr_city VARCHAR(200)
Indexes
tblnews
nws_id INT(11)
nws_con_id INT(11)
nws_date DATETIME
nws_description VARCHAR(200)
nws_detail LONGTEXT
nws_usr_id INT(11)
nws_ts DATETIME
Indexes
tblcountry
cou_id INT(11)
cou_description VARCHAR(200)
cou_iso VARCHAR(10)
Indexes
tblconteststatus
cst_id INT(11)
cst_description VARCHAR(200)
cst_token VARCHAR(50)
cst_description_before VARCHAR(200)
cst_description_running VARCHAR(200)
cst_description_after VARCHAR(200)
Indexes
tblgender
gen_id INT(11)
gen_description VARCHAR(50)
Indexes
tbluserproperty
usp_id INT(11)
usp_prp_id INT(11)
usp_usr_id INT(11)
usp_value_int INT(11)
usp_value_string VARCHAR(200)
usp_value_blob LONGBLOB
Indexes
tblnewsfeedtype
nwt_id INT(11)
nwt_token VARCHAR(50)
nwt_description VARCHAR(200)
Indexes
tblschool
sch_id INT(11)
sch_description VARCHAR(200)
sch_timestamp DATETIME
sch_city VARCHAR(200)
Indexes
tblgroup
grp_id INT(11)
grp_usr_id INT(11)
grp_name VARCHAR(50)
Indexes
tblresultstatus
rst_id INT(11)
rst_description VARCHAR(200)
rst_token VARCHAR(50)
Indexes
tblsubproblemfeedback
spf_id INT(11)
spf_description VARCHAR(200)
spf_token VARCHAR(50)
Indexes

B SQL queries

```
    -- number of submissions per problem.
select prc_con_id, prb_id, prb_description, count(sub_id) as number_of_instances from tblsu
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    group by prb_description;

    -- all submissions per problem (interessant, 123[waslijnen], 22[meest waardevolle obje
select sub_ctt_id, sub_prc_id, prb_id, prb_description, res_tst_id, convert(sub_blob using 
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblresult on sub_id=res_sub_id
    where prb_id=122 group by sub_ctt_id order by sub_ctt_id ;



--Some problems do not have test data, like this one, "Meer".
--(It appears that this is the case for all b-problems)
select prc_short, res_tst_id, convert(sub_blob using utf8), sub_filename from tblsubmissio
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblresult on sub_id=res_sub_id
     inner join tblsubproblem on spr_prb_id=prb_id
    -- inner join tbltestdata on tst_spr_id=spr_id
    where prb_id=253 order by sub_ctt_id desc limit 10;



    -- All submissions per problem and test data
select sub_ctt_id, prb_id, prb_description, prc_short, spr_id, convert(sub_blob using utf8
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblresult on sub_id=res_sub_id
    inner join tblsubproblem on spr_prb_id=prb_id
    inner join tbltestdata on tst_spr_id=spr_id
    where prb_id=122 order by sub_ctt_id, tst_spr_id;



    -- Best made problems based on sub_problems
    select prb_id, prc_short, count(sub_id) as number_of_subs, prb_description, max(res_sc
    inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblresult on sub_id=res_sub_id
    where sub_blob is not null and prc_short != "D" group by prb_description order by avgma

   -- Best scores per contestant for a specific problem
   -- (to be filled in at 'prb-id')

   select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub_b
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    where prb_id = 172 and sub_filename not like '%.exe' group by sub_ctt_id, prb_id order




    -- 20 best submissions
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, convert(sub_blob using
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    where prb_id in ( select * from (
select prb_id from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
inner join tblproblem on prc_prb_id=prb_id
where sub_blob is not null and prc_short != "D" group by prb_description order by avg(sub_s


-- 20 best subs, their descriptions and short names (like A2, C1)
    select distinct prb_description, prc_short
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    where prb_id in ( select * from (
select prb_id from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
inner join tblproblem on prc_prb_id=prb_id
where sub_blob is not null and prc_short != "D" group by prb_description order by avg(sub_s

         -- 20 best subs, sub_id's
    select sub_id
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    where prb_id in ( select * from (
select prb_id from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
inner join tblproblem on prc_prb_id=prb_id
where sub_blob is not null and prc_short != "D" group by prb_description order by avg(sub_s


 -- 30 worst subs
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub_
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    where prb_id in ( select * from (
select prb_id from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
inner join tblproblem on prc_prb_id=prb_id
where sub_blob is not null and prc_short != "D" group by prb_description order by avg(sub_s
    and sub_filename not like '%.exe' and sub_filename not like '%.pdb' and sub_filename no
    and sub_filename not like '%.jar' and sub_filename not like '%.8xp' and sub_filename no
    group by sub_ctt_id, prb_id;





    -- New score average
    select avg(woeloe) from
    (select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, wo, sub_fil
    from (
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub_
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    group by sub_ctt_id, prb_id) as blub
    group by prb_id) as blubi ;

    -- Mean 2019 A questions
    select avg(woeloe) from
    (select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, wo, sub_fil
    from (
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub_
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblcontest on prc_con_id=con_id
    where prc_short like "A%"
    and con_id between 51 and 54
    group by sub_ctt_id, prb_id) as blub
    group by prb_id) as blubi ;

       -- Mean 2018 A questions
    select avg(woeloe) from
    (select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, wo, sub_fi
    from (
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblcontest on prc_con_id=con_id
    where prc_short like "A%"
    and con_id between 46 and 50
    group by sub_ctt_id, prb_id) as blub
    group by prb_id) as blubi ;


          -- mean 2017 A questions
    select avg(woeloe) from
    (select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, wo, sub_fi
    from (
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblcontest on prc_con_id=con_id
    where prc_short like "A%"
    and con_id between 38 and 41
    group by sub_ctt_id, prb_id) as blub
    group by prb_id) as blubi ;

             -- mean 2016 A questions
    select avg(woeloe) from
    (select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, wo, sub_fi
    from (
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblcontest on prc_con_id=con_id
    where prc_short like "A%"
    and con_id between 32 and 37
    group by sub_ctt_id, prb_id) as blub
    group by prb_id) as blubi ;


                -- mean 2015 A questions
    select avg(woeloe) from
    (select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, wo, sub_fil
    from (
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub_
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    inner join tblcontest on prc_con_id=con_id
    where prc_short like "A%"
    and con_id between 28 and 31
    group by sub_ctt_id, prb_id) as blub
    group by prb_id) as blubi ;

    -- Average scores submissions, the best submission per contestant
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, wo, sub_file
    from (
    select sub_id, sub_ctt_id, sub_prc_id, prb_id, prb_description, prc_short, convert(sub_
from tblsubmission
inner join tblproblemcontest on sub_prc_id=prc_id
    inner join tblproblem on prc_prb_id=prb_id
    group by sub_ctt_id, prb_id) as blub
    group by prb_id;
```