DEVELOPING AN ALGORITHM FOR GENERATING COMPUTERIZED TEST COMBINATIONS

Ivan KUYUMDZHIEV

JEL I210, C830, C880

Abstract

This article focuses on computerized testing as one the most effective methods of assessment in higher education. The study demonstrates problems of generating tests with random questions using computers. Proposed algorithm overcomes these difficulties taking into account predefined complexity of each question used in the test. Multiple tests of the productivity of the algorithm show the results of different scenarios and help assess its quality. The reason these findings are important is that they demonstrate an effective method for the computerized creation of tests that could be described as preadaptive.

In a crisis, more than ever before, it is necessary to improve the quality of higher education and raise the effectiveness of universities. Assessing students' knowledge is an essential part of the process of education. The assessment process involves several stakeholders among which students and lecturers come first. Students get an understanding of whether the knowledge and skills they have acquired are clearly visible and measurable at a certain point of time. For the lecturer this turns into a condition and mechanism for adjusting their own work in terms of educational content – whether particular topics are fully comprehended and utilized, which ones present more than the usual level of difficulty for the students, etc. (Desev et al, 1977, p. 446).

Another participant in the process of assessment are the higher education institutions. Checking students' level of knowledge (incl. admission exams, continuous assessment, term grades, state examinations) consumes a huge amount of resource – lecture halls occupancy and lecturers teaching load, printed teaching materials, etc., which means that even a slight improvement of efficiency would result in considerable economies on a global scale.

1 Department of Informatics, University of Economics – Varna, Bulgaria. e-mail: ivan_ognyanov@ue-varna.bg
Stakeholders also include legislators, employers and parents (Patrick et al., 2008). The government partially subsidizes institutions of higher education and sets various criteria for accreditation which also contain requirements on the effectiveness of assessment and the use of modern methods to achieve this (NAOA, 2011). Employers, on the other hand, need to feel confident that they are hiring well-prepared specialists, while parents, who usually pay for their offspring's education, view assessment as a measure of their progress.

Following from the above, raising the effectiveness of testing will result in the positive attitude of learners, employers and parents on the one hand, and lecturers, university governing bodies and government institutions on the other. Taking into consideration this fact, this article aims to develop and test the effectiveness of a module of a test system, which in itself represents an algorithm for generating test combinations.

1. Test format of examinations and problems with a computer-generated test combination under certain criteria

Using tests is the best known form of examination. Varieties can be defined based on the type of questions (true/false, short answer, fill in the blanks, matching, multiple choice questions (Gyurova, V. et al., 2006, p. 260)) or the method of scoring (scoring of correct answers only, withdrawal of points for wrong answers, bonuses for omitted answers, scoring for partial knowledge, questions with more than one correct answer, withdrawal of points for omitted answers (Lesage et al, 2013). A great deal of research has been carried out in search of an answer to the question "which is the most exact method of evaluation" and "which is the best test format", all reaching multidirectional results (Kastner and Stangla, 2011, Ventouras et al., 2010), but despite the scientific community arguments concerning the complexity of various types of test and the efficiency of evaluation methods, it is beyond doubt that performing computer-generated multiple-choice tests can potentially and to a large degree eliminate the subjective factor in assessment. The degree of objectivity depends on which part of the process is automated. With partially automated tests the choice of questions to include in the tests and/or assessing test scores is performed by a human. In order to minimize the subjective factor it is also necessary for both test generation and scoring to be automated.

The developed algorithm for generating computerized tests is part of a comprehensive testing system which reduces lecturers' involvement in the design of the examination to inputting a bank of test questions and choosing the criteria for test generation. For every question lecturers assign a weight (in the 1 to 5 range), which
allows for a precise calibration of the overall difficulty of the test. After the end of the examination, the system generates a reference sheet containing information on which questions were most often given wrong answers, and which were most often answered correctly, which in turn can be used as a foundation for correcting the questions' weights. After the complexity of each question is identified, the system will contain a reasonably varied database to be used for adding options in order to generate adaptive tests. With these tests a level of complexity is assigned to each question and the next question in the text depends on the answers to previous ones (Eskenasi et al., 1993). In a certain variant of adaptive test, if the learner answers correctly, the next question will be more difficult; should the answer be wrong, the next question can be less difficult. According to some researchers (Wainer et al., 2014) adaptive testing provides exact measurement of student knowledge, as students come with varying levels of preparedness. According to the same study the classical multiple-choice test assesses only the students with a medium level of knowledge and definitely does not do a favour to those of higher or lower levels of knowledge.

Despite the said advantages of computer-generated testing, the reduced expert involvement throughout the stages of creating the variants and their assessment suggests the development of a dedicated algorithm that is flexible enough to allow for test generation under preliminary set criteria. In order for a quality module to be created, the needs and requirements of each type of user must be studied.

At a conceptual level the work of the test-generating algorithm can be described in fig.1, where the following users of its functions can be identified – lecturers, system administrators, students/candidate-students, computer programmers.
Fig. 1. Interaction with the test-generating algorithm

Test parameters are set by lecturers and are input by system administrators. The duration and complexity (the sum total of the weights of the questions included) of the test, the students for which the test is active, as well as the time period when the test is accessible are characteristics of every test generated. Apart from the above, the following variants are available:

A) test generation according to a set total number of questions;
B) test generation according to a set total number of questions from a defined range of topics;
B) test generation by selecting a certain number of questions on each topic.

Needs analysis makes it clear that from the point of view of lecturers the system should generate a test containing a random selection of questions that meet the above criteria.

The next participants in the test-generating process are system administrators. They are the users who input the lecturer-set requirements and monitor the successful execution of the algorithm. As basic parameters of quality work of the system administrators identify the quick execution of the set requirements, as well as the high level of system dialog – with incorrect input, messages show not only the reason, but also the way to eliminate the problem.
After the test has been generated, it is accessible to the examined students for the stipulated time period. Students are passive users of the algorithm – they do not take part in its execution and their requirements for a fair distribution of questions depend on the criteria that have been set by the lecturer during test generation.

The test generating algorithm is part of the overall system of the testing centre and as such it interacts with other modules of this system, as the modules for reshuffling questions and answers, which in turn are a component of the test visualization module, etc. The requirements set by other programmers include fast response and a unified, predictable output from the execution, which can be used in the dependent modules.

A major problem in system development is the creation of an algorithm which should function efficiently with both small and large databases. The main factors affecting performance are: the number of questions in the test, the number of questions in the database that meet the weight requirements. The number of the latter largely depends on the interval of complexity of the end result test in setting the criteria – the larger the interval, the greater the number of questions that are able to meet the set criterion. From this standpoint, several combinations are possible in order to distribute the questions from a database:

k1) the database contains many questions with various weights and the user requires the creation of a test with a limited number of questions. For example, a choice of 10 questions out of 200 questions available in the database and grouped into 10 topics, with 20 questions on each topic;

k2) the database contains many questions, the user requires a large test, but the number of questions in the test is, again, a small percentage of what is available in the database. For example, generating a 100-question test where the database includes 10 topics with 70 questions on each topic;

k3) the number of database questions whose weights meet the users’ requirements is close to the number of questions necessary to generate the test;

K3.1) generating a large test of 150 questions where 20 topics with 10 questions on each topic are available in the database;

K3.2) generating a short test of 15 questions, of the type used for current assessment, where the database is small and only contains 10 topics with 2 questions on each topic.

Analysis of the above mentioned examples puts forward the following major problems:
1. A markedly uneven distribution of questions. If variant A) for test generation is used, there is a good chance that the random choice of questions will result in a distinctly uneven distribution of questions and topics – incl. the probability of the test covering questions on one topic only, and/or certain topics not being covered at all.

Let us suppose we have a bank of $u$ number of test topics, with each topic containing $v$ number of questions and let us randomly retrieve $w$ number of them. As the total number of questions equals $u \cdot v$, the number of all possible combinations for drawing the questions is

$$n = C_{(u,v)}^w = \frac{(u \cdot v)!}{w!(u \cdot v - w)!}$$  \hspace{1cm} (1)

Let us now calculate the probability of the questions drawn to be distributed in exactly $k$ number of topics. Evidently, $k \leq \min(u,v)$.

We number the topics in random order from 1 to $u$. Then every possible distribution of questions within the separate topics is described by a vector with dimensions $u$ topics with $k$ strictly positive components and $u - k$ zero components:

$$X = (x_1, x_2, ..., x_u)$$  \hspace{1cm} (2)

for which the following conditions are fulfilled:

$$\sum_{i=1}^u x_i = w \text{ and } x_i \leq v \text{ } (i = 1...u) .$$

If vector components (2) are arranged in descending order, then the vector appears as

$$X^*_k, s \subseteq S = (x_1, x_2, ..., x_k, 0, ..., 0) \text{, като } x_1 \geq x_2 \geq ... \geq x_k \geq 1,$$  \hspace{1cm} (3)

where $S = \{X^*_s\}$ is a finite set of vectors.

The set of vectors (2) includes all the possible permutations from the elements of all vectors (3). As these are permutations with repetitions, their number can be found using the formula

$$\bar{P}_{k, s \subseteq S} = \frac{u!}{(u - k)!k_{s,1}!k_{s,2}!...k_{s,t}!},$$  \hspace{1cm} (4)

where:
- $(u - k)$ - number of zero elements of (4);
- \( k_{s,j}, \ (j = 1...\ell) \) - number of elements equal to 1, 2, ..., \( \ell \), \( \sum_{j=1}^{\ell} k_{s,j} = w \).

The number of the different variants for retrieving the questions on the topics for each vector (3) will equal

\[
\left[ \prod_{i=1}^{u} \frac{v!}{x_i!(v-x_i)!} \right]_{s \in S}
\]  

(5)

Vectors (3) are random partitions of the number \( w \) into exactly \( k \) positive numbers. Their number is defined by recurrence equations (Hall, 1970). Algorithms for generating partitioning are provided by (Knuth, 2013).

Having in mind (4) and (5), then the number of all possible distributions of the \( w \) question into exactly \( k \) test sections is

\[
m_k = \sum_{s \in S} \frac{u!}{(u-k)!k_{s,1}!k_{s,2}!...k_{s,\ell}!} \left[ \prod_{i=1}^{u} C_{x_i}^v \right]_s.
\]  

(6)

Then the probability of the drawn questions being distributed in exactly \( k \) topics will be

\[
P_k = \frac{m_k}{n}.
\]  

(7)

Suppose, for example, we have a bank of 10 topics and each topic contains 20 test questions. The test contains 10 questions which are randomly retrieved from the whole set. We shall calculate the probability of grouping the questions into 7 topics. Vectors \( X^*_{7,s \in S} \) are:

\[
X^*_{7,1} = (4,1,1,1,1,1,0,0,0);
\]

\[
X^*_{7,2} = (3,2,1,1,1,1,0,0,0);
\]

\[
X^*_{7,3} = (2,2,2,1,1,1,0,0,0).
\]

Using (1), (6) and (7), we define

\[
n = 22 451 004 309 013 280,
\]

\[
m_7 = 8 363 040 000 000 000,
\]
\[ P_7 \approx 0.37250182 \ 151729 \].

Similarly we define (table 1)

**Table 1**

<table>
<thead>
<tr>
<th>Number of topics from which questions are selected ((k))</th>
<th>Number of implementations ((m_k))</th>
<th>Probability ((P_k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 topic</td>
<td>1847560</td>
<td>0.00000000008229</td>
</tr>
<tr>
<td>2 topics</td>
<td>65439360720</td>
<td>0.00000291476318</td>
</tr>
<tr>
<td>3 topics</td>
<td>8742192030000</td>
<td>0.00038938979787</td>
</tr>
<tr>
<td>4 topics</td>
<td>283500257040000</td>
<td>0.01262750891398</td>
</tr>
<tr>
<td>5 topics</td>
<td>2475475002000000</td>
<td>0.11026121450639</td>
</tr>
<tr>
<td>6 topics</td>
<td>7435792728000000</td>
<td>0.33120089532096</td>
</tr>
<tr>
<td>7 topics</td>
<td>8363040000000000000</td>
<td>0.37250182151729</td>
</tr>
<tr>
<td>8 topics</td>
<td>3436416000000000000</td>
<td>0.15306290768562</td>
</tr>
<tr>
<td>9 topics</td>
<td>4377600000000000000</td>
<td>0.01949845957778</td>
</tr>
<tr>
<td>10 topics</td>
<td>1024000000000000000</td>
<td>0.00045610431760</td>
</tr>
</tbody>
</table>

Results in the table show that on average 33 out of 100 generated tests would include questions from 6 topics only, which would exclude 40% of the studied material. Despite the slim probability, it is even possible for all questions to be selected from one and the same topic. This brings us to the conclusion that it is necessary to apply a mechanism which regulates the number of chosen questions for each topic using variant A) for test generation.

2. Impossibility to cover all combinations. The total number of possible combinations grows in parallel with the size of the database. Calculating each combination would lead to a huge waste of computational resources and time. The results of calculations for a number of combinations for the examples mentioned are shown in table 2.
### Table 2

<table>
<thead>
<tr>
<th>Database and test</th>
<th>Number of combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test of 10 questions chosen out of 200 questions</td>
<td>(2^{54.31})</td>
</tr>
<tr>
<td>Test of 100 questions chosen out of 700 questions</td>
<td>(2^{409.63})</td>
</tr>
<tr>
<td>Test of 150 questions chosen out of 200</td>
<td>(2^{158.31})</td>
</tr>
<tr>
<td>Test of 15 questions chosen out of 20</td>
<td>15504</td>
</tr>
</tbody>
</table>

While generating each combination, a check should be made on whether the sum of the weights of the questions in the combination matches the overall requirement for the difficulty of the test. The search for correct combinations can be compared to decoding an encrypted message. Thus for example for breaking the (once considered unbreakable) hash encryption algorithm SHA-1 \(2^{69}\) operations are enough (Schneier, 2005).

Despite the growing capacity of computer hardware, performing such a volume of calculations is difficult to achieve. A Chinese supercomputer developed in 2013 performs 33.86 trillion operations a second. If the calculating power of this computer were to be used the time it took to check the first three combinations would be as follows:

### Table 3

<table>
<thead>
<tr>
<th>Database and test</th>
<th>Time to process all combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test of 10 questions chosen out of 200 questions</td>
<td>11.05 minutes</td>
</tr>
<tr>
<td>Test of 100 questions chosen out of 700 questions</td>
<td>(2^{377.78}) years</td>
</tr>
<tr>
<td>Test of 150 questions chosen out of 200</td>
<td>(2^{126.46}) years</td>
</tr>
</tbody>
</table>

The impossibility of looping through all combinations creates a problem in cases where a very small number of combinations meet the set requirements.

### 2. Describing and testing an algorithm for generating a computerized test combination

In an attempt to solve the said problems there has been developed an algorithm with the following structure (fig. 2)
The probability of selected questions being concentrated in certain topics is eliminated by generating a combination of a random number of questions for each module, with the minimum being 1 (step 1). Due to the nature of random choice it is possible that the database does not include questions with suitable weights to meet the user's requirements for the overall complexity of the test. For example, if the topic only contains questions with the weight of 3, and the user demands a maximum complexity test, i.e. only with weight 5 questions. Nevertheless, no exclusion has been made of combinations that cannot be satisfied, as that corresponds with the issue of the impossibility to cover all variants.

In order to improve fast response and retain the probability for a random choice of test, the number of operations is reduced to whichever is less: either 200 000 or the total number of combinations*3 (step 6). As it was mentioned above, there is a good chance that the randomly chosen set of a number of questions does not contain a combination to match the requirements. On the other hand, it is possible for such a combination to exist, but not to be found even after 200 000 reshuffles. For this reason a recursion is added to the algorithm (step 7) and a reciprocal reshuffling quotient (RRQ) - N. RRQ stipulates what percentage of questions whose weights are closest to average should be chosen before reshuffling. For example, the requirement is to create a test of medium-level complexity and thus choose medium-level of complexity questions. At the beginning of the execution this quotient is zero, and with each following recursion it grows by 10%, so that execution time decreases, because the more questions are chosen based on their weights, the fewer possible combinations remain for random choice.
Despite the nearly 2 million operations, it is still possible not to find the right set of questions to meet the requirements. For this reason, a function has been created (step 8) for targeted search of questions with suitable complexity. Search mechanisms for certain questions, however, to some degree contradict the requirement for random choice, but are the only instrument in cases where the overall number of combinations is impossible to loop through.

Reports on the functioning of the system show that over 1000 tests were carried out in the last year. Test generation takes on average 0.0002 seconds, and maximum execution time is close to 0.001 seconds, which demonstrates a reasonably fast response of the algorithm. Despite the relatively large number of tests there has not been a single entrance into recursion. These findings result from the state of the database and the test requirements. At present there is a very small percentage of questions with a weight other than 3, while the requests for test generation demand a large interval of overall test complexity. Under such conditions every randomly drawn combination would match the set parameters.

The above mentioned results are not enough to determine the efficiency of the proposed algorithm and for this reason a test module has been created for an in-depth study of the capacity of the algorithm in various situations. The following hardware configuration has been used - Intel Core i3-3130M CPU @ 2.60GHz, HDD - HGST 7200RPM SATA-III 500GB, RAM - 4 GB DDR3-1600(800MHz) SDRAM. Test execution software includes OS Windows 7 Pro 32 bit with SP1, PHP 5.5.6 language and DBMS MySQL Community Server 5.6.14.

The basic parameters of the test module have been set so as to match the 4 cases discussed above, namely:

1) generating a test of 10 questions out of 200 questions available in the database;
2) generating a test of 100 questions out of 700 available in the database;
3.1) generating a test of 150 questions out of 200 available in the database;
3.2) generating a test of 15 questions out of 20 available in the database.

The questions are evenly distributed by topic. To check the system efficiency with different weight quotients, five sample databases with different quotients are created. Question complexity in these databases can be any number between - 1 and 4; 2 and 5; 1 and 5; 2 and 4; all questions have type 3 complexity.

For each combination (40 altogether) of the said types of test and weighted databases a condition is set for generating 4000 tests within a time frame of max. 10 minutes. In order to simulate various requirements for the overall complexity of the test, each batch of 20 tests receives requirements from the widest possible range (with
variant 1 overall complexity should be between 10 and 50) to the most limited one possible, with the increase of the upper limit and lowering the threshold taking place at each 2%. Thus each test that was first generated out of the batch of 20 has the easiest requirements, and every 20th test has the most difficult ones.

### 3. Results and conclusions

The conducted tests demonstrate that the average time for a single test generation is 0.110 seconds with 52118 tests generated.

#### Table 4

Results from the tests performed

<table>
<thead>
<tr>
<th>Type of test</th>
<th>Question weights in database between</th>
<th>Ave. number of iter.</th>
<th>Max. number of iter.</th>
<th>Ave. gen. time (sec)</th>
<th>Max. gen. time (sec)</th>
<th>Total numb. tests</th>
<th>Number exec. of targeted choice</th>
<th>Number cases of entrr. in recursion</th>
<th>Average number of recursions</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2)</td>
<td>1-4</td>
<td>1.20</td>
<td>8</td>
<td>0.000</td>
<td>0.016</td>
<td>760</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3.1)</td>
<td>1-4</td>
<td>70.27</td>
<td>987</td>
<td>0.660</td>
<td>199.475</td>
<td>779</td>
<td>-</td>
<td>10</td>
<td>0.0231</td>
</tr>
<tr>
<td>1)</td>
<td>1-4</td>
<td>2.47</td>
<td>9</td>
<td>0.0004</td>
<td>0.016</td>
<td>780</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2)</td>
<td>1-4</td>
<td>4.35</td>
<td>9</td>
<td>0.002</td>
<td>0.312</td>
<td>780</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3.2)</td>
<td>1-5</td>
<td>1.02</td>
<td>9</td>
<td>0.003</td>
<td>7.300</td>
<td>4000</td>
<td>2</td>
<td>2</td>
<td>0.01</td>
</tr>
<tr>
<td>3.1)</td>
<td>1-5</td>
<td>1.00</td>
<td>2</td>
<td>0.001</td>
<td>0.016</td>
<td>4000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1)</td>
<td>1-5</td>
<td>1.46</td>
<td>9</td>
<td>0.0003</td>
<td>0.016</td>
<td>4000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2)</td>
<td>1-5</td>
<td>1.00</td>
<td>1</td>
<td>0.001</td>
<td>0.016</td>
<td>4000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3.2)</td>
<td>2-4</td>
<td>1.00</td>
<td>1</td>
<td>0.000</td>
<td>0.016</td>
<td>4000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3.1)</td>
<td>2-4</td>
<td>1.00</td>
<td>1</td>
<td>0.001</td>
<td>0.016</td>
<td>4000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1)</td>
<td>2-4</td>
<td>1.17</td>
<td>9</td>
<td>0.0002</td>
<td>0.016</td>
<td>4000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2)</td>
<td>2-4</td>
<td>1.00</td>
<td>1</td>
<td>0.001</td>
<td>0.027</td>
<td>4000</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3.2)</td>
<td>2-5</td>
<td>1.04</td>
<td>4</td>
<td>0.0002</td>
<td>0.016</td>
<td>240</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3.1)</td>
<td>2-5</td>
<td>32.64</td>
<td>9</td>
<td>1.523</td>
<td>226.347</td>
<td>259</td>
<td>1</td>
<td>5</td>
<td>0.06</td>
</tr>
<tr>
<td>1)</td>
<td>2-5</td>
<td>2.39</td>
<td>9</td>
<td>0.0004</td>
<td>0.016</td>
<td>260</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2)</td>
<td>2-5</td>
<td>1.43</td>
<td>9</td>
<td>0.001</td>
<td>0.016</td>
<td>260</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
The highest average generation time (0.437 sec.) is spent on a test of 150 questions chosen from a database of 20 topics with 10 questions for each topic, and the lowest generation time (0.0003 sec.) went on a test of 10 questions with 10 topics, 20 questions on each topic. Results show that test generation time is directly related to the number of iterations and the number of recursions, and they, in turn depend not only on the size of the database and the size of the required test, but also on the correspondence between the overall complexity of the test and the weights of the individual questions in the database. The longest average execution time was spent on tests where the database questions have weights between 2 and 5 or between 1 and 4 i.e. databases of lower or higher than average complexity. The fifth database, comprised from weight 3 questions only, corresponds to the state of the database used in practice and proves the statement that execution speed and the small number of iterations are the result of setting wide search criteria and lack of diversity in the complexity of questions.

Results show that the test generation algorithm fulfils the set requirements. It runs at a satisfactory speed with both large and small databases. Conditions for improving its robust performance are as follows: inputting a set of questions of diverse complexity and creating tests with up to several dozen questions.

The use of recursion delays script execution but leads to the generation of a test with absolutely randomly chosen questions. As it was said before, despite the large number of reshuffles and recursions, in certain cases it is necessary to apply the function for targeted search of questions with suitable weights – this happened on three occasions out of over 52,000 cases in the tests we performed.

The use of the developed algorithm facilitates the work of the Test Center at Varna EU. Computer-generated tests from the developed module reduce the subjective factor and contain a fair and random distribution of questions by topic. At the same time, they raise the economic efficiency of carrying out examinations, as they reduce:

- the cost of photocopying a test;
- the time spent on performing the test;
- the time it takes to check and assess the results. The need of using software developed by an external organization is eliminated and an opportunity is cre-
ated for further development of the product – implementation of adaptive test system. Another prospect for the test center can be the potential for performing certificate exams for external organizations. The smooth performance of the algorithm over the last 12 months proves that it fulfils the requirements set by the users. Applying the algorithm for the purposes of admission exams and the conducted tests demonstrate its ability to also work with larger databases without a noticeable effect on execution time.

I wish to express my deep gratitude to the colleagues, without whom this article and the software it discusses would not exist: Georgi Zelenkov, Deyan Mihaylov, Milen Marinov, Petar Dimitrov.

Acknowledgements: Georgi Zelenkov, Deyan Mihaylov, Milen Marinov, Petar Dimitrov

End Notes

1. Calculated using the formula $c_n^k = \frac{v_n^k}{p_k}$

References


