Ministry of Education and Science of Ukraine

Odessa National Academy of Food Technologies

International Competition of Student Scientific Works

BLACK SEA SCIENCE 2021

Information Technology, Automation and Robotics

Proceedings

Odessa, ONAFT 2021

UDC 004.01/08

Editorial board:

Black Sea Science 2021: Proceedings of the International Competition of Student Scientific Works. Information Technology, Automation and Robotics. / Odessa National Academy of Food Technologies; B.Yegorov, M. Mardar, S.Kotlyk (editors-in-chief.) [*et al.*]. – Odessa: ONAFT, 2021. – 526 p.

These materials of International Competition of Student Scientific Works «Black Sea Science 2021» contain the works of the contest participants in the section «Information technologies, automation and robotics» (not winners).

The author of the work is responsible for the accuracy of the information.

Organizing committee:

Prof. Bogdan Iegorov, D.Sc., Rector of Odessa National Academy of Food Technologies, Head of the Committee

Prof. Maryna Mardar, D.Sc., Vice-Rector for Scientific and Pedagogical Work and International Relations of Odessa National Academy of Food Technologies, Deputy Head of the Committee

Prof. Stefan Dragoev, D.Sc., Vice-Rector for Scientific Work and Business Partnerships of University of Food Technologies (Bulgaria)

Prof. Baurzhan Nurakhmetov, D.Sc., First Vice-Rector of Almaty Technological University (Kazakhstan)

Prof. Mircea Bernic, Dr. habil., Vice-Rector for Scientific Work of Technical University of Moldova (Moldova)

Prof. Jacek Wrobel, Dr. habil., Rector of West Pomeranian University of Technology (Poland)

Prof. Michael Zinigrad, D.Sc., Rector of Ariel University (Israel)

Dr. Mei Lehe, Ph.D., Vice-President of Ningbo Institute of Technology, Zhejiang University (China)

Prof. Plamen Kangalov, Ph.D., Vice-Rector for Academic Affairs of "Angel Kanchev" University of Ruse (Bulgaria)

Dr. Alexander Sychev, Ph.D., Assoc. Professor of Sukhoi State Technical University of Gomel (Belarus)

Dr. Hanna Lilishentseva, Ph.D., Assoc. Professor, Head of the Department of Merchandise of Foodstuff of Belarus State Economic University (Belarus)

Prof. Heinz Leuenberger, Ph.D., Professor of the Institute of Ecopreneurship of University of Applied Sciences and Arts (Switzerland)

Prof. Edward Pospiech, Dr. habil., Professor of the Institute of Meat Technology of Poznan University of Life Sciences (Poland)

Prof. Lali Elanidze, Ph.D., Professor of the Faculty of Agrarian Sciences of Iakob Gogebashvili Telavi State University (Georgia)

Dr. V. Kozhevnikova, Ph.D., Senior Lecturer of the Department of Hotel and Catering Business of Odessa National Academy of Food Technologies, Secretary of the Committee

The jury for the section «Information technologies, automation and robotics»

Head of the jury:

Sergii Kotlyk – Ph.D., Associate Professor, Director of the P.M. Platonov Educational-Scientific Institute of Computer Systems and Technologies "Industry 4.0" of Odessa National Academy of Food Technologies (Ukraine)

Members of the jury:

Piotr Artiemjew - Dr hab., Associate Professor in Decision Systems of the Faculty of Mathematics and Computer Science,University of Warmia and Mazury in Olsztyn (Poland)

Francisco Antonio Augusto – Dr., International Relations Manager of Higher Institute of Information and Communication Technologies (Angola)

Andrey Kuprijanov – Ph.D., Associate Professor of the Department of Software for Computers and Automated Systems of Belarusian National Technical University (Belarus)

Simon Milbert – Vice-President of Xtra Information Management, Inc. (USA)

Ivan Palov – D.Sc., Professor of University of Ruse "Angel Kanchev" (Bulgaria)

Degla Gérard Hugues – Communications and Training Manager of "MAPCOM solutions informatiques" company group (Benin)

Nugzar Kereselidze - Academic Doctor of Informatics (Computer Science), Associate Professor of the Department of Natural Sciences, Mathematics, Technology and Pharmacy, Sukhumi State University (Georgia)

Etibar Seyidzade - Associate Professor of the Department of Computer and Information Technolohies, Baku Engineering University (Azerbaijan)

Vladimir Golenkov, D.Sc., Professor of the Department of Intelligent Information Technologies, Belarusian State University of Informatics and Radio Electronics (Belarus)

Zhanar Omirbekova - Ph.D., Associate Professor of the Department of Automation and Management, Satbayev University (Kazakhstan)

Ivan Palov - D.Sc., Professor of the Department of Power Supply and Electrical Equipment, University of Ruse "Angel Kanchev" (Bulgaria)

Siarhei Palavenia - Ph.D., Associate Professor, Head of the Department of Telecommunication Systems, Belarusian State Academy of Communications (Belarus)

Alexander Goloskokov - Ph.D., Professor of the Department of Software Engineering and Information Technology Management, National Technical University "Kharkiv Polytechnic Institute" (Ukraine)

Peter Nikolyuk - D.Sc., Professor of the Department of Computer Technology, Vasyl Stus Donetsk National University (Ukraine)

Vladimir Palagin - D.Sc., Professor, Head of the Department of Radio Engineering, Telecommunications and Robotics Systems, Cherkasy State Technological University (Ukraine)

Viktor Khobin – D.Sc., Professor, Head of the Department of Technological Processes Automation and Robotic Systems of Odessa National Academy of Food Technologies (Ukraine)

Valeriy Plotnikov – D.Sc., Professor, Head of the Department of Information Technology and Cybersecurity of Odessa National Academy of Food Technologies (Ukraine)

Sergii Artemenko – D.Sc., Professor, Head of the Department of Computer Engineering of Odessa National Academy of Food Technologies (Ukraine)

Fedir Trishyn - Ph.D., Associate Professor, Vice-Rector on Scientific and Educational Work, Odessa National Academy of Food Technologies (Ukraine)

Valerii Levinskyi – Ph.D., Associate Professor of the Department of Technological Processes Automation and Robotic Systems of Odessa National Academy of Food Technologies (Ukraine)

Viktor Yehorov – Ph.D., Supervisor of the Laboratory of Mechatronics and Robotics of Odessa National Academy of Food Technologies (Ukraine)

Pavlo Lomovtsev – Ph.D., Associate Professor of the Department of Information Technology and Cybersecurity of Odessa National Academy of Food Technologies (Ukraine)

Yurii Kornienko – Ph.D., Associate Professor of the Department of Information Technology and Cybersecurity of Odessa National Academy of Food Technologies (Ukraine)

Serhii Shestopalov – Ph.D., Associate Professor of the Department of Computer Engineering of Odessa National Academy of Food Technologies (Ukraine)

Anatoly Galiulin - Ph.D., Associate Professor, Acting Head of the Department of Electromechanics and Mechatronics, Odessa National Academy of Food Technologies (Ukraine)

Secretary of the jury:

Oksana Sokolova – Senior Lecturer of the Department of Information Technology and Cybersecurity of Odessa National Academy of Food Technologies (Ukraine)

APPLICATION OF THE METHOD OF GRADUAL FORMATION OF SETS OF ADMISSIBLE VALUES FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS

Author: ***Mariia Mushyn***
Advisor: ***Olexandr Shportko***
Academician Stepan Demianchuk International University of Economics and Humanities (Ukraine)

Abstract. *This article describes a method of gradual formation of sets of admissible values as an alternative to backtracking and account for changes methods. The mechanism of algorithms that apply these methods to solve combinatorial optimization problems is substantiated. Fragments of programs that implement these algorithms in C# programming language are given and the results of their testing in a remote computing environment are analyzed. Test results show that the implementation of the method of gradual formation of sets of admissible values cardinally reduces the execution time of programs, indicating its effectiveness.*

Keywords: *method of gradual formation of sets of admissible values, backtracking, taking changes into account method.*

I. INTRODUCTION

As known, combinatorics solves problems of selecting and permutating elements of a set in accordance with given conditions. It is used, for example, in cryptography, machine learning, competitive programming, or in solving problems from other spheres of life. The formulas of combinatorics can be simple for algorithmization, but their use for large input data can significantly slow down the work of separate programs and computing systems in general. That is why the development of existing and creation of new methods of optimization of combinatorial problems is currently an important scientific task that is carried out within the framework of combinatorial optimization [1].

II. LITERATURE ANALYSIS

Combinatorial optimization [1] considers optimization problems in which the set of feasible solutions is discrete or can be reduced to discrete. The purpose of such tasks is to find an optimal object from a finite set of objects when the well-known brute force search methods take an unacceptable amount of time to execute [1]. Traditionally, to speed up an exhaustive search, ***the method of taking into account changes*** [2] is used, which calculates the next variant of a solution, not from the very beginning, but only taking into account changes compared to the previous variant of a solution.

But the most famous method for solving combinatorial optimization problems is **backtracking** [3]. It implies that a partial solution expands and incomplete candidates for the solution is further analyzed. If the next candidate is unacceptable, the algorithm moves to another candidate or returns to another partial solution, or

continues the search in other ways. According to this method, all steps of finding the optimal solution are recorded so that in the case of changes that do not satisfy the constraints of the problem, it is possible to return to an acceptable candidate [4]. Implementations of this method are usually faster than a brute force search, which is not always possible in a reasonable time [2] because during their execution invalid candidates not added to the complete solution.

In general, nowadays, when programming solutions to combinatorial optimization problems, the main attention is paid to discarding inadmissible candidates, and, in our opinion, the computational complexity is not analyzed sufficiently [5]. It is in vain because measures of computational complexity "give an idea of the execution time of algorithms" [6, c. 57] and allow one to understand why one algorithm will work much faster than another and teach future programmers to take into account not only the correct solution of the problem but also its efficiency, not only the program execution time but also the memory usage. It is by analyzing the computational complexity in this article we will define the feasibility of using proposed algorithms.

In general, the idea of using sets of valid values has been used in programming for several years.

For example, it is expedient not to compare elements with each other to sort an array of integers in ascending order with a small values range but only to count the frequencies of individual values and write to the resulting array each value from smallest to largest as many times as found in the input array [7, 8]. It is clear that it is advisable to do this when the range of values is commensurate with the size of the array. We use the analysis of sets of admissible values to solve combinatorial optimization problems.

III. OBJECT, SUBJECT, AND METHODS OF RESEARCH

The object of research — methods for solving combinatorial optimization problems.

The subject of the work — the influence of improvements of combinatorial optimization algorithms on their computational complexity.

The purpose of the study is to describe and substantiate the feasibility of applying the method of gradual formation of sets of admissible values for solving combinatorial optimization problems.

The task of the work is to create programs that implement algorithms of different methods for solving one of the typical combinatorial problems and compare their effectiveness for different test cases to establish the feasibility of each of these methods.

In the process of creating algorithms, general scientific methods of analysis and synthesis were used. To solve the problem of combinatorial optimization, we used the method of backtracking, the method of taking into account changes and our proposed method of gradual formation of a set of admissible values.

It is clear that the developed programs can be tested on local workstations, but it should take into account computing environment characteristics and the ability to

adapt programs to known test cases. Therefore, we consider it appropriate to test the developed programs in remote computing environments. Such testing is a way to develop logical thinking and optimization skills in solving atypical problems. One of the sites that provide access to tasks and the remote computing environment is https://www.e-olymp.com. This work is written using its resources. The advantages of E-olymp are the following:

- a significant number of tasks from different sections;
- grouping problems by different levels of complexity;
- the ability to compare its results with the results of other users;
- 15 programming languages are supported;
- availability of training materials;
- ability to compare the runtime and the RAM usage for every single test using different approaches to solving each problem.

IV. RESULTS

4.1. Formulation of the problem for the approbation of combinatorial optimization methods

We will demonstrate the application of various methods for solving combinatorial optimization problems on the example of the problem "CD" (№ 1266) from the site https://www.e-olymp.com. The condition of this problem is formulated as follows:

You have a long drive by a car ahead. You have a tape recorder, but unfortunately, your best music is on CDs. You need to have it on tapes so the problem to solve is: you have a tape N minutes long. How to choose tracks from CD to get most out of tape space and have as short unused space as possible.

The program should find the set of tracks which fill the tape best and print it in the same sequence as the tracks are stored on the CD. **Input data** have any number of lines. Each one contains value N, (after space) number of tracks and durations of the tracks. The program should show for each line the string "**sum:**" and a sum of duration times.

Assumptions:

- number of tracks on the CD. does not exceed **100**
- no track is longer than N minutes
- length of each track is expressed as an integer number
- N is also integer ($0 \leq N \leq 200$).

Time limit 1 second for executing each test by program, memory limit is 122.81 Mi/B.

4.2. Using the exhaustive search method with elements of backtracking

To find the set of tracks that fill most of an available tape space, we may need to analyze all possible variants for their including for each test because any combination of tracks might be better. The execution can be completed before it only if a combination is found in which the length of the tape is fully occupied (equal to *N*).

According to the condition of the problem, the number of tracks is equal to S. Each of them can be in only two states - taken into account in the sum of the current combination or not. Therefore, **each variant of including of tracks can be written as an S-bit binary number**, and their amount is 2^S. To analyse each possible variant, we need to generate all S-bit binary numbers except 0 because this is the maximum sum by default. To calculate the tracks record length of following possible variant, we will convert its number into the binary numeration and analyze the individual bits. Therefore the computational complexity of the algorithm is 2^S*S. In the next sum, we will add only that lengths of tracks, for which the corresponding bits in the variant number are equal to 1. To solve the problem, we will output that maximum sum of variants, which is not greater than *N*.

Also, we should note that adding the length of the next track to the sum of the current version makes records longer than the length of the tape, it infringes the condition of the problem (because the track lengths are non-negative), and therefore the program can move to the next option and form a new sum. This, in fact, implements the idea of a backtracking algorithm.

For example, we trace the mechanism of formation of the maximal acceptable sum of tracks for a tape length 10 ($N = 10$) using four tracks ($S = 4$) with lengths 2, 4, 8, 4 (tab. 1). We see, for example, that setting the first bit on the right in the binary variant number leads to the adding to the current sum the length of the first track. In this case, the search for the maximum sum ends before the calculation all variants of occurrences of tracks ($2^4 = 16$), because the found maximal acceptable sum is equal to the length of the tape. If this did not happen, the program would continue to work until all variants are found. The maximal fill (10) was found after five search steps.

Table 1. The use of binary variants numbers representation to generate track combinations for $N = 10$, $S = 4$ and track lengths 2, 4, 8, 4

Variant's number in the decimal system	Variant's number in binary	Tracks are taken into account	The sum of the current version of included tracks	The maximum sum is known for now
1	0001	2	2	2
2	0010	4	4	4
3	0011	2, 4	6	6
4	0100	8	8	8
5	0101	2, 8	10	10

Here is the code of the program for solving the problem by exhaustive search with backtracking elements. The C# programming language was used because nowadays it is one of the main programming languages for applications.

```
while(true)
{string line = Console.ReadLine(); // Read the next line with the conditions of the problem
 if (String.IsNullOrEmpty(line)) return; // There are no more problems
 string[] str = line.Trim().Split(new char[] {' '});
 if (str.Length < 2)
  return; // The problem cannot be solved without the length of the tape or the number of tracks
 int n = Convert.ToInt32(str[0]); // The total length of the tape
 int s = Convert.ToInt32(str[1]); // Number of tracks
 long i, j, sum, maxSum = 0, remainder; // Remainder - the remainder of a division
 int[] track = new int[s];
 for (i = 0; i < s; i++)
  // Reading the length of each track
  track[i] = Convert.ToInt32(str[i + 2]);
•/* Generating all s-bit numbers, where the i-th bit will indicate whether (1) the i-th track is taken into account in the sum or not (0). This will get every possible variant of including their lengths */
 // Counting the number of all possible variants of tracks occurrence
 BigInteger variantsNumber = (BigInteger)1<<s; // 1<<s is equal to 2^ s
 // Conversion of each number of a variant of occurrences of tracks in a binary
// representation,  calculating the appropriate sum to find the maximal acceptable
 for (j = 1; j < variantsNumber; j++)
  {sum = 0; // Each iteration the sum for the next variant of track inclusion will be
// calculated
   remainder = j; // It is in order not to affect the counter value
   for (i = 0; i < s; i++) // The option of selecting tracks is interpreted as binary
// S-bit number, the length of which corresponds to the number of tracks (S).
    {if (remainder % 2 == 1) // If the i-th track is included in the sum
     {sum += track[i];
      if (sum > n) // Exceeding the total length of the tape is not allowed
       break; }
➢  remainder /= 2;
    } // We move on to the next track
   if (sum <= n && sum > maxSum) // If the amount found does not exceed
// the total length of the tape and greater than maximal of the previous ones
    {maxSum = sum;
     if (maxSum == n) break; }} // The best variant is already found
•Console.WriteLine("sum:" + maxSum.ToString()); }
```

The program testing results provided on the site e-olymp.com are shown in figure 1.

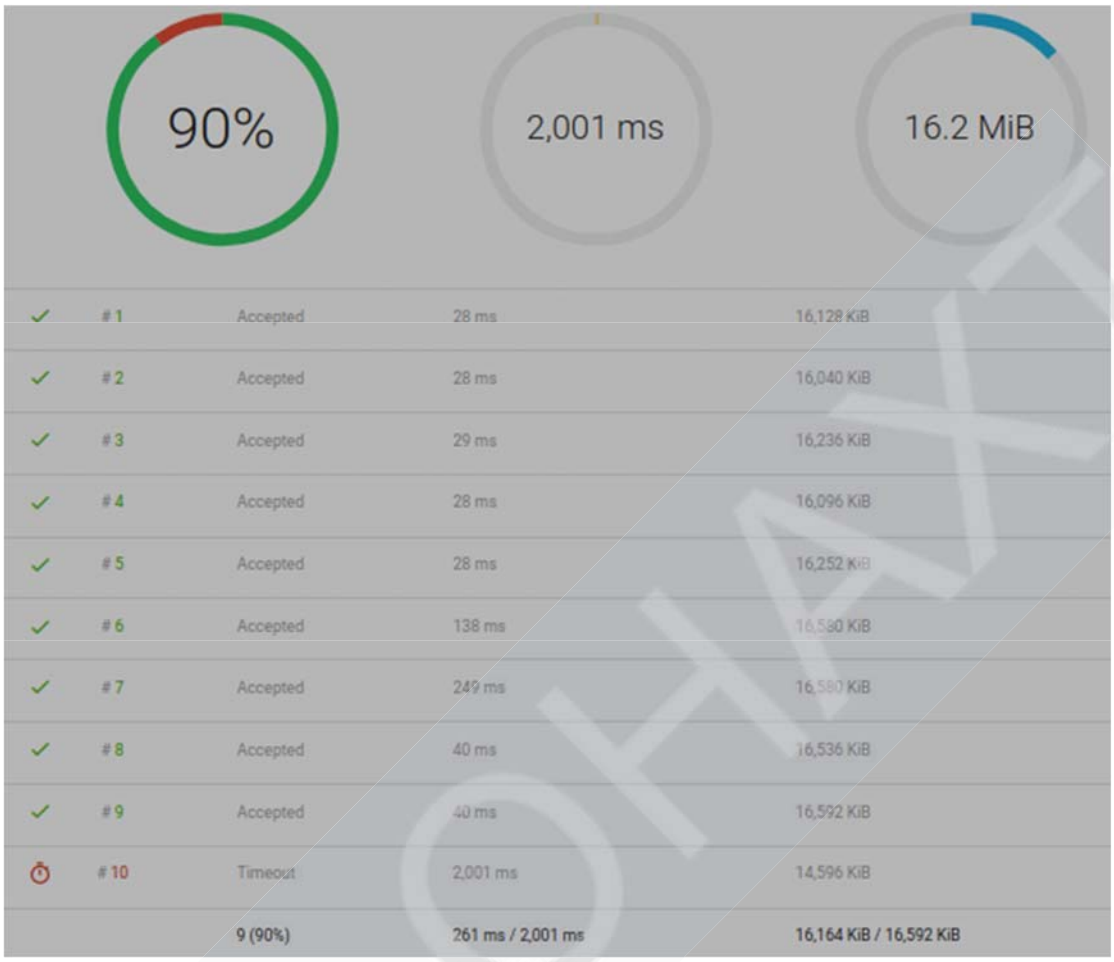


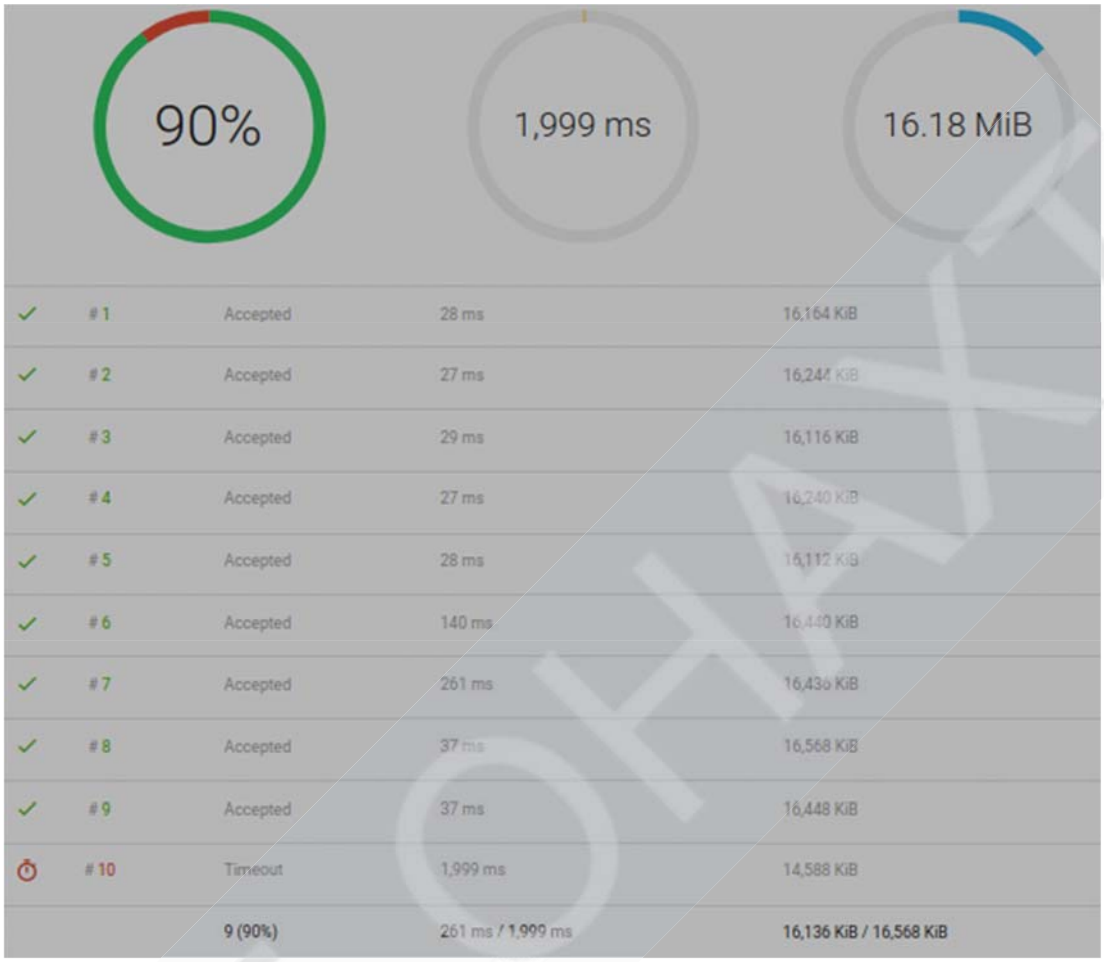
Fig 1. Results of testing the program that implements the exhaustive search

We see that the given program correctly execute nine tests, but exhausts the time limit when passing the last test.

4.3. Implicit application of the idea of a return search method

To speed up the above program, consider that the cycle for converting the variant number to bits of the binary representation is performed S times for each variant. It causes extra operations, as not all variant numbers are s-digit binary numbers. For our algorithm, this will mean that for the next version, there are no more tracks that are included in the current sum. In essence, this is an implicit application of the idea of a backtracking method. To implement it, we will add the next code to the previous program after the line marked with the symbol ➢:

```
if(remainder == 0) break; // No more tracks included in the sum
```



✓	#1	Accepted	28 ms	16,164 KiB
✓	#2	Accepted	27 ms	16,244 KiB
✓	#3	Accepted	29 ms	16,116 KiB
✓	#4	Accepted	27 ms	16,240 KiB
✓	#5	Accepted	28 ms	16,112 KiB
✓	#6	Accepted	140 ms	16,440 KiB
✓	#7	Accepted	261 ms	16,436 KiB
✓	#8	Accepted	37 ms	16,568 KiB
✓	#9	Accepted	37 ms	16,448 KiB
⏱	#10	Timeout	1,999 ms	14,588 KiB
		9 (90%)	261 ms / 1,999 ms	16,136 KiB / 16,568 KiB

Fig. 2. Results of testing the program that implements the exhaustive search with taking into account a length of a variant number in the binary number representation

This change significantly affects the runtime speed with a large S in cases of considering small variants numbers for including tracks (figure 2). We see that the execution of the last test still exhausts the time limit. The computational complexity of the algorithm is now $2^{S-1}*S$, ie halved, but practically, the execution time of the program is not significantly reduced, because the search interrupts with returns do not affect the increase in subtotals. If we could also reject some of the solutions when performing this check, it would be a classic backtracking implementation.

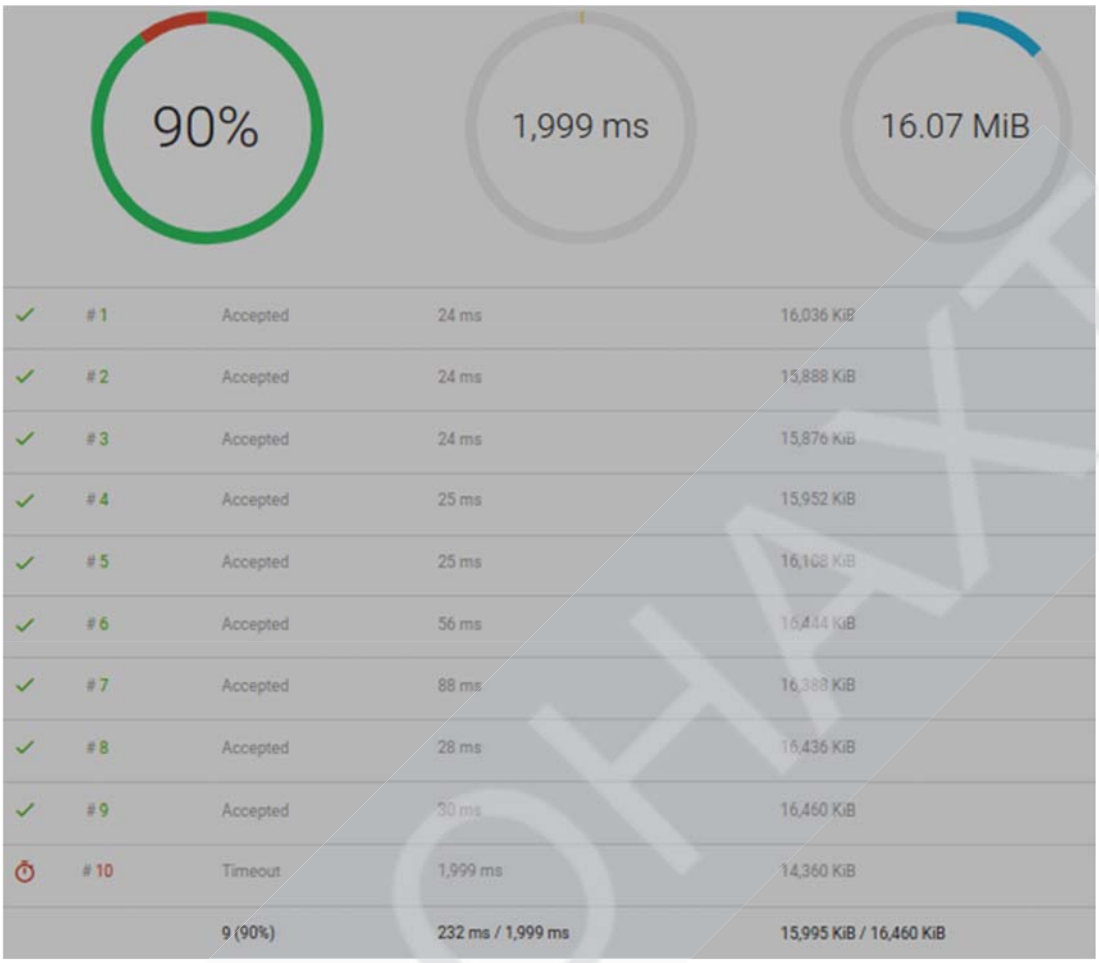
4.4. Taking changes into account method usage

We will implement the method of taking into account changes. When calculating the sum of the tracks lengths program will relate to the sum of the previous variant. The current sum should decrease relative to the previous one by the length of those tracks, in the binary representation of which the number instead of 1 became 0. And it should increase by the length of the first of the tracks, where instead

of 0 became 1. The order of selecting tracks bit numbers, but the values of individual bits will not be calculated each time, and save in the array occurrence of S + 1 element. This will avoid excessive divisions by two and solve the problem for any non-negative S. We will complete the search when 1 appears in the additional element of this array, ie all S-bit binary numbers will be searched. To implement this algorithm, replace the lines between the markers • in the code of the previous program with the following code:

```
byte[] occurrence = new byte[s+1]; /* i-th element of the array indicates is taken
into account (1) or not (0) the i-th track to the sum. Using this array instead of binary
variants numbers allows the program to process any number of tracks. Generating
the next variant of tracks inclusion in the array of occurrence corresponds to the
principle of increasing the number by 1 in the binary number system: adding one
converts all ones from right to left to zero, and the first right zero replaces to one.
Accordingly, the sum of track lengths relative to the previous sum will decrease by
the length of the track, if the occurrence array instead of 1 became 0 and increase by
the length of the track if the array occurrence instead of 0 became 1. Initially, all
values in the array are 0, i. e. the in first variant tracks inclusion no one track is
taken into account. The last variant of track inclusion corresponds to the occurrence
array, in which all elements are equal to 1, i. e. all tracks are taken into account.
Completion of the search of track selection options is performed when setting 1 in the
additional left position of the array (overflow of the option number). */
 while (occurrence[s] == 0) // While additional left position of the array is empty
  {j = 0; // track index, which changes in sum
   while(occurrence[j] != 0)
    {occurrence[j] = 0; // The track is no longer taken into account
     // therefore, the total sum decreases by the length of this track
     sum -= track[j++]; }
   occurrence[j] = 1; // Reached the track, which is already taken into account,
   //so the total duration increases by the length of this track
   if (j < s) sum += track[j];
   if(sum <= n && sum > maxSum) // The best variant is already found
    {maxSum = sum;
     if (maxSum == n) break; }}
```

Such changes to the program have sped its execution for all tests, but the last test execution still takes too much time (figure 3). In fact, this version of the program does not only process **adjacent unincluded** tracks on the right but all **adjacent unchanged** tracks on the right.



✓	#1	Accepted	24 ms	16,036 KiB
✓	#2	Accepted	24 ms	15,888 KiB
✓	#3	Accepted	24 ms	15,876 KiB
✓	#4	Accepted	25 ms	15,952 KiB
✓	#5	Accepted	25 ms	16,108 KiB
✓	#6	Accepted	56 ms	16,444 KiB
✓	#7	Accepted	88 ms	16,388 KiB
✓	#8	Accepted	28 ms	16,436 KiB
✓	#9	Accepted	30 ms	16,460 KiB
⏱	#10	Timeout	1,999 ms	14,360 KiB
		9 (90%)	232 ms / 1,999 ms	15,995 KiB / 16,460 KiB

Fig. 3. Results of testing the program that implements the taking into account changes method

4.5. Description and substantiation of the method of gradual formation of sets of admissible values

Let's ask the question: how does the set of possible sums of track lengths change after considering the next track? In other words: how to get the set of possible (acceptable after summation) lengths of tracks E_i, knowing the allowable lengths of the sums of previous tracks E_{i-1} and the length of the next track $track_i$ (i means that the tracks from the initial to the i-th inclusive)? If the next track $track_i$ is not taken into account, then the set of allowable sums of track lengths will stay the same, i. e. $E'_i=E_{i-1}$. If the next track is taken into account, then new acceptable lengths will appear, formed by increasing the previous lengths by the length of the next track. That is, $E''_i=\{e''_{i,j}' : e''_{i-1,j}+track_i\}$. Obviously that

$$E_i=E'_i \cup E''_i. \quad (1)$$

If no track is taken into account, then the sum of the track lengths is 0. If we consider the initial track, the sums of lengths is 0 (initial track is not taken into account) and $track_0$ (if the initial track is taken into account) is acceptable That is,

E_0={0; $track_0$}, and the following sets of allowable track sums are calculated according to (1).

For example, for the above-mentioned sequence of track lengths 2, 4, 8, 4, we consistently get the following sets of allowable sums:

E_0={0; 2} (zero track is taken into account or not taken into account).

E'_1={0; 2} (the first track is not taken into account), E''_1={4; 6} (the first track is taken into account), and, according to (1) $E_1=E'_1 \cup E''_1$={0; 2; 4; 6} (first track is taken into account or not taken into account).

E'_2={0; 2; 4; 6} (second track is not included), E''_2={8; 10; 12; 14} (second track is included), $E_2=E'_2 \cup E''_2$={0; 2; 4; 6; 8; 10; 12; 14}.

E'_3={0; 2; 4; 6; 8; 10; 12; 14}. (he third track is not taken into account), E''_3={4; 6; 8; 10; 12; 14; 16; 18} (the third track is taken into account), $E_3=E'_3 \cup E''_3$={0; 2; 4; 6; 8; 10; 12; 14; 16; 18}.

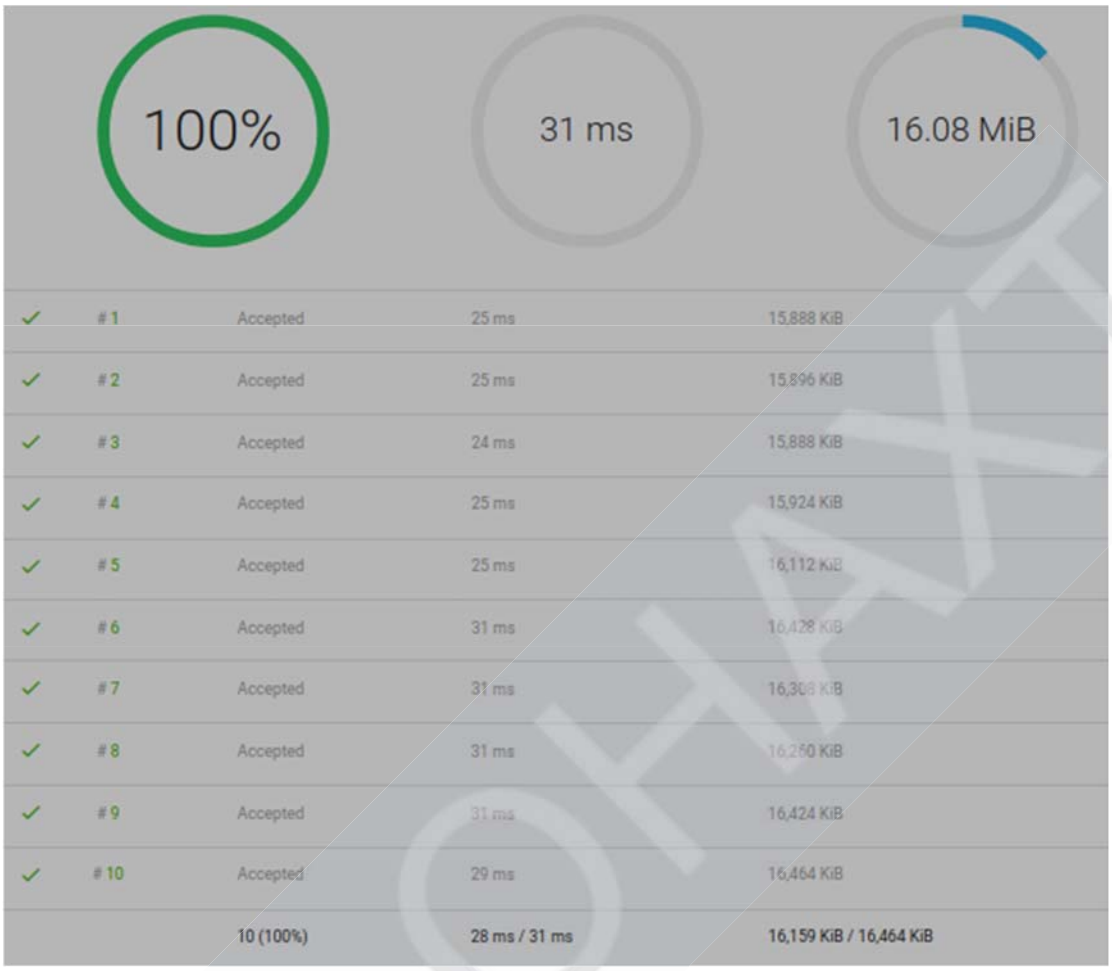
Since the allowable sums increase every time, it makes no sense to calculate those that are greater than *N* (in this example, the allowable amounts over 10 could be ignored). From the remaining allowable amounts, we have to choose the maximal one after considering all the tracks, or equal to N, if such sum found earlier (in this case, the allowable amount of 10 appears after considering the third track). On the example of the formation of the third set of admissible sums, we also see that the number of elements of the set does not exactly double each time, because it is possible to duplicate the elements in *E' and E''*.

We will implement a program version of the gradual formation of sets of acceptable sums. The set of admissible sums E_i *after consideration of the next element depends only on the set* E_{i-1}. Therefore it is not necessary to store all sets - iteratively is enough, having the previous set, to form the following set, and for the following iteration to assign to the previous set the following, and the following - the previous. Interestingly, the next set can be omitted - the valid values of any set will always be valid for subsequent valid sets. To store the previous and next sets of allowable sums on each iteration, we use arrays, where the value of 1 in each element will indicate its entry into the set of allowable values, and 0 - the absence of it in the set. In order to quickly reassign arrays, we will not swap their elements, but **links** to arrays. Given these considerations, the implementation of the method of gradual formation of sets of admissible values for the problem under study can be as follows:

```
byte[] nextSum = new byte[n+1], prevSum = new byte[n+1]; /* Previous and following elements of sets of admissible sums greater than N are not considered */
byte[] interim;
nextSum[0] = 1; // Before tracks analysing, only 0 sum is available
for(i = 0; i < s; i++) // Cycle on tracks
 {/* Each track may or may not be included in the sum. Accordingly, the possible sums of track lengths after considering the next track will be all previous sums (when the current track is not taken into account) and previous sums increased by the length of the current track. */
```

```
  interim = prevSum; /* An intermediate reference to the array is required so that prevSum is not lost during permutation */
  prevSum = nextSum;
  nextSum = interim; /* When we going consider the next track, the previous possible sums are taken from the available next ones and the next possible sums are calculated. For faster permutation of arrays their values are not reassigned, and references to them change places */
  for(j = 0; j <= n; j++) /* Cycle on possible sums of lengths of record on a tape. The index of the array and is the value of the sum. */
    {if(prevSum[j] == 1) // Found a valid sum relative to previous tracks
     {nextSum[j] = 1; /* The next track is not taken into account - the previous possible amount does not change */
      if(j+track[i] <= n) /* If the next track together with the possible previous sum can be placed on the tape */
       nextSum[j + track[i]] = 1; /* Adding the length of the next track to the previous possible sums allows you to determine the new possible sums without adding all the track lengths again. */
    }}}
 i = n;
 while(nextSum[i] == 0) i--; /* Find the maximum of the allowable sums of tracks. Search from the end, because the larger the index, the larger the sum */
```

These program test results are shown in figure 4. We see that all the tests execute in the allotted time and at the same time use the least RAM of the presented programs. Accelerating the solution of the problem is not accidental. After all, the computational complexity of the algorithm is now SN, which is much less than $2^{S-1}*S$, taking into account the limitations of our problem ($100*200 << 2^{99}*100$). In conclusion, we note that the method of gradual formation of sets of admissible values finds only the solution (extremum) of this problem of combinatorial optimization, but does not determine the tracks at which this solution is obtained. Other (perhaps discussed above) methods should be used to find these tracks, but even finding the extremum speeds up the solution of optimization problems in the classical case.



✓	# 1	Accepted	25 ms	15,888 KiB
✓	# 2	Accepted	25 ms	15,896 KiB
✓	# 3	Accepted	24 ms	15,888 KiB
✓	# 4	Accepted	25 ms	15,924 KiB
✓	# 5	Accepted	25 ms	16,112 KiB
✓	# 6	Accepted	31 ms	16,428 KiB
✓	# 7	Accepted	31 ms	16,308 KiB
✓	# 8	Accepted	31 ms	16,260 KiB
✓	# 9	Accepted	31 ms	16,424 KiB
✓	# 10	Accepted	29 ms	16,464 KiB
		10 (100%)	28 ms / 31 ms	16,159 KiB / 16,464 KiB

Fig. 4. Results of testing the program that implements the method of gradual formation of sets of admissible values

V. CONCLUSIONS

1. To determine the most effective way to solve the problem of combinatorial optimization, it is not enough to compare time on known test sets, but one should try to analyze their computational complexity.

2. The method of gradual formation of sets of admissible values is an effective alternative to methods of backtracking and taking into account changes in solving combinatorial optimization problems if the range of admissible values is discrete and the solution is similar to the method of dynamic programming.

3. To model sets of elements in programming, it is advisable to use logical arrays. Then, to reassign sets, it is sufficient to swapped by changing links to arrays, rather than reassign individual elements.

4. To speed up a solution of combinatorial optimization problems, it is not enough to bypass some options of complete search, and it is necessary to minimize the runtime of each variant, taking into account limitations of a problem.

VI. REFERENCES

1 Combinatorial optimization. Retrieved from https://en.wikipedia.org/wiki/Combinatorial_optimization [in English]. (2020, January 25).

2 Mnozhynnyi typ [Pruel type]. Retrieved from https://studfile.net/preview/7079855/page:12/ [in Ukranian]. (2020, January 27)

3 Poshuk z vertanniam [Backtracking]. Retrieved from https://uk.wikipedia.org/wiki/Пошук_з_вертанням [in Ukranian]. (2020, January 25).

4 Alhorytmy z povernenniam. Rozviazok zadachi pro rukh konia [Backtracking algorithms. Solving the problem about a horse movement]. Retrieved from https://studfile.net/preview/7079855/page:20/ [in Ukranian]. (2020, January 27).

5 Obchysliuvalna skladnist [Computational complexity]. Retrieved from https://znaimo.com.ua/Обчислювальна_складність#link0 [in Ukranian]. (2020, January 25).

6 Otsinka skladnosti alhorytmiv, abo shcho take O(log n) [Computational complexity of an algorithm evaluation or what is O(log n)]. Retrieved from https://echo.lviv.ua/dev/53 [in Ukranian]. (2020, January 27).

7 Knuth D. E. (1997). The Art of Computer Programming (Vol 2), (2nd ed.). Addison Wesley Longman.

8 Shportko A. V., Shportko L. V. (2019). Acceleration of large integer arrays sorting using ranges of values and frequencies of elements. Information Extraction and Processing, no. 47(123), 73-79. DOI:https://doi.org/10.15407/vidbir2019.47.073

International Competition of Student Scientific Works

BLACK SEA SCIENCE 2021

Information Technology, Automation and Robotics

Proceedings

Odessa National Academy of Food Technologies

The collection includes student works of the participants of the competition, which were not included in the number of prize-winners. The texts of the competitive works are published in the form in which they were submitted by the authors. The authors of the articles are responsible for the content and form of submission of the material.

Responsible for the issue: Sergii Kotlyk

Computer typesetting and layout: Oksana Sokolova

Odessa 2021