https://doi.org/10.53656/math2023-2-4-exa

Educational Technologies
Образователни технологии

АЛГОРИТМИЧНИТЕ ЗАДАЧИ ОТ ДЪРЖАВНИЯ ЗРЕЛОСТЕН ИЗПИТ ПО ПРОФИЛИРАЩ ПРЕДМЕТ ИНФОРМАТИКА ЗА 2022 Г.

Д-р Красимир Манев
Сп. „Математика и информатика“

Резюме. През учебната 2021/2022 година се проведе за пръв път държавен зрелостен изпит за учениците, изучавали профилиращия предмет информатика в българското средно училище. В статията се разглеждат задачите от двете сесии на изпита, които трябва да проверят уменията на зрелостниците да избират/създават алгоритми. Подробно са разгледани условията, на които трябва да отговаря една задача от този тип, и доколко задачите от двете изпитни теми удовлетворяват тези условия.

Ключови думи: матура по информатика; състезателна задача; формат на входа и изхода; алгоритъм; сложност на алгоритъм

1. Въведение

В края на учебната 2021/2022 година се проведе за първи път държавен зрелостен изпит (ДЗИ) за учениците, които изучаваха профилиращия предмет информатика в българското средно училище. Изпитът се състоеше от две части. В първата част, с продължителност 90 минути, зрелостниците трябваше да отговарят на 24 тестови въпроса, 16 от които с избираем отговор и 8 със свободен отговор. Във втората, с продължителност 150 минути, трябваше да решават 4 задачи. Три от задачите изискваха написване на програма на изучавания език за програмиране (C# или Java), а четвъртата – съставяне на последователност от заявки на езика за структурирано програмиране SQL. Една от задачите за програмиране е с алгоритмичен характер, а другите две изискват създаване на клас от обекти с посочени в заданието характеристики (атрибути) и методи.

Проведени бяха две изпитни сесии: една през май – юни и една през август – септември. На анализа и оценката на тестовата част от първата сесия на изпита е посветена статията (Atanasov et al. 2023). В тази статия ще разгледаме алгоритмичните задачи от втората част в двете изпитни сесии.

2. Формулировка на задачата от изпитната сесия май – юни

Алгоритмичната задача за първата сесия беше формулирана по следния начин[1].

Задача 25. Да се реализира програма, която въвежда от клавиатурата брой елементи в редица, и непразна редица от толкова на брой цели положителни числа. На стандартния изход да се извежда информация колко пъти се среща всяко от числата в редицата. В изведената информация за броя на срещания НЕ трябва има повторения. Редът на извеждане на числата няма значение. Първото въведено число е брой на числата в редицата. В програмата да бъдат направени необходимата валидация и прихващане на изключения.

Пример:

Примерен вход	Примерен изход
6 1 5 5 1 2 1	число: 1, брой: 3 число: 5, брой: 2 число: 2, брой: 1

Водени от опита си в състезанията по програмиране (а ДЗИ е вид състезание и аналогиите със състезателното програмиране могат да са само от полза), ще си позволим да предложим нова формулировка на задачата, която смятаме за по-подходяща за решаване в изпитна обстановка. За задачите от състезания по програмиране (обикновено наричани олимпиади) – за ученици, студенти или професионалисти, от много години е възприета следната структура на условието:

А. формулировка на задачата;

Б. описание на входните данни;

В. описание на очаквания изход;

Г. ограничения за входните данни;

Д. един или няколко примера за входни данни и очакваните за тези примери изходи;

Е. обяснения (ако авторът на задачата сметне за необходимо).

За програмите, решаващи задачи с алгоритмичен характер, е задължително да четат входните данни от стандартния вход и да извеждат резултата на стандартния изход, за да може да се проверяват автоматично. Това правило в случая е спазено.

В условието на предложената за ДЗИ задача няма ясни формулировки на посочените по-горе части А, Б и В. За формàта на входните данни и очаквания резултат изпитваният може да съди само от примера, но нищо в условието на задачата не го задължава да се съобрази с този пример. Това е потенциална възможност за различен подход при оценяването от различните проверяващи. Вижда се, че в описанието на входните данни липсва условието всяко от

числата на редицата да е на отделен ред, а това, в C# например, ще доведе до възможност за две различни по трудност решения. Като част Г е формулирано някакво изискване, което се опитва да компенсира липсата на това, което обикновено наричаме ограничения за входните данни, но не е ясно с какво допринася за яснотата на поставената задача. Изискването за контрол на входните данни и прихващането на изключения също може да се интерпретира по различни начини.

И докато недобрата формулировка на частите Б и В може да бъде компенсирана от проверяващите с уточняване как да постъпват при различните интерпретации за форматирането на входа и изхода, и не би създала кой знае колко сериозни затруднения при оценяването, отсъствието на частта Г оставя възможности за сериозно различаващи се интерпретации от страна на участниците в изпита.

Отсъствието на ограничение за броя N на числата в редицата не е проблем при деклариране на масива, който ще е нужен (а задачата не може да бъде решена без използване на масив), тъй като езиците за програмиране позволяват да се задели необходимата за масива памет по време на изпълнение. Очевидно авторът е сметнал, че така проверява важно умение на учениците. Неприятното е, че това отсъствие ограничава до голяма степен стремежа да се търси ефективно решение, както е показано по-долу. Послабо подготвените ученици няма да се замислят и ще напишат първия хрумнал им алгоритъм (ad hoc, както наричаме такива алгоритми), който ще е неефективен. И ще спечелят! Защото добре подготвените ще се замислят какъв трябва да е алгоритъмът на програмата им, за да работи за разумно време при големи стойности на N. Така те ще загубят време за изработване на такъв ефективен алгоритъм, което няма да им донесе повисок резултат.

Така поставена, задачата не проверява много важна компетентност – умението на ученика да подбира **добър алгоритъм** според ограничението за размера на входа. Тя не поставя никакво предизвикателство към изпитваните и нищо чудно повечето ученици да са написали тривиалното и най-бавно решение, каквото е и предложеното от автора на задачата. Добре е поне, че в забележка е добавено указание към проверяващите да допускат и други алгоритми! Обратното би било недопустимо.

Отсъствието на ограничение за големината на числата в редицата също не е добра практика. Езикът за програмиране предлага няколко целочислени типа и по-добре подготвените ще се замислят кой от тях да използват, тъй като във формулировката не е указано нищо. Разбира се, решението ще е да изберат най-мощния от целочислените типове. По-неприятното е, че знаейки големината на зададените числа, изпитваният би могъл да състави много по-ефективен алгоритъм, както ще покажем по-нататък.

Още забележки към формулировката на задачата ще посочим в следващия раздел. Засега една по-добра формулировка, според традицията в състезателното програмиране, би била следната.

А. Дадена е редица от N цели положителни числа a_1, a_2, …, a_N. Напишете програма, която да намира броя на срещанията на всяко от числата в редицата. В програмата да бъде направена необходимата валидация на входните данни и прихващане на изключения.

Б. На първия ред на стандартния вход ще бъде зададен броят N на числата в редицата, а на всеки от следващите N реда – по едно от числата.

В. За всяко от числата, което се среща в редицата, на един ред на стандартния изход програмата трябва да изведе низа `"число: "`, числото a, за което се отнася редът, запетая, интервал, низа `"брой: "` и намерения брой срещания на a в редицата (виж примера). Редът, по който ще са подредени числата в изхода, е без значение.

Г. Ограничения: $3 <= N <= 1000000$, $1 <= a_i < 1000000$. (**Забележка:** тъй като в задачата не са зададени никакви ограничения, предложените стойности са примерни.)

Примерът от оригиналната формулировка е добър, а обяснения за тази задача, изглежда, не са необходими.

В допълнение е важно да кажем, че в състезанията по програмиране за всяка задача се определят *ограничение за използваната памет* и *ограничение на времето,* което програмата може да изразходи за решаване на един тестов пример. Ограничението за паметта се използва в ситуации, когато авторът би искал да оцени по-високо решения, които използват пестеливо паметта в решението на задачи, някои от алгоритмите за решаване на които изискват голямо количество памет. Този параметър е по-скоро несъществен за задачите, които ще се предлагат на държавните зрелостни изпити.

Не така стоят нещата с ограничението по време. В състезанията по програмиране, където проверката на решенията се извършва автоматично от специализирани софтуерни системи (Manev et al. 2009), това е средството да се оцени качеството *бързодействие* на използвания алгоритъм, което в теорията наричаме *сложност по време*. За подценяващите ролята на тази характеристика на един алгоритъм бих предложил да се опитат да сортират 1 000 000 цели числа с простия и лесен за имплементиране „алгоритъм на мехурчето“. При начина, по който е формулирана задачата, и начина, по който са оценявани решенията на проведените сесии на ДЗИ – ръчно и на компютри с различно бързодействие, прилагането на критерия сложност по време е невъзможно. В същото време, не намираме никакви сериозни основания проверката на алгоритмичната задача да не бъде направена автоматично – така ще се спестят много усилия на проверяващите и ще бъде изключена всяка субективност.

3. Авторското решение на задачата от май – юни

В публикуваната от МОН изпитна тема с отговори на въпросите и решения на задачите[1] е предложено следното решение на C# на задача 25 (надяваме се, че читателят ще може да отнесе следващите разсъждения и към авторското решение на Java):

```
using System;
class Problem25
{  static int[] readArray()
   {  Console.WriteLine(„Въведете брой елементи“);
      int n;
      do
      {  n = Convert.ToInt32(Console.ReadLine());
         if (n < 1)
           Console.WriteLine(„Моля, въведете положително число!“);
      } while (n < 1);
      int[] result = new int[n];
      for (int i = 0; i < n; i++)
      {  do
         {  Console.WriteLine(„Моля, въведете елемент „ + (i + 1));
            result[i] = Convert.ToInt32(Console.ReadLine());
            if (result[i] < 1)
            Console.WriteLine(„Моля, въведете положително число!“);
         } while (result[i] < 1);
      }
      return result;
   }
   static void Main(string[] args)
   {  try
      {  int[] array = readArray();
         for (int i = 0; i < array.Length; i++)
         {  // това първо срещане ли е?
            int j = 0;
            while (j < array.Length && array[i] != array[j]) j++;
            if (j == i)
            {  int count = 1;
               for (j++; j < array.Length; j++)
                 if (array[i] == array[j]) count++;
               Console.WriteLine(„Число: „ + array[i] +
                                 „, брой срещания: „ + count);
            }
         }
```

```
        }
        catch (FormatException)
        {  Console.WriteLine(„Некоректно въведено число“);  }
    }
}
```

Нека разгледаме това решение.

3.1. Въвеждане на данните и контрол на входа

Да разгледаме първо статичната функция `readArray(),` предназначена да въвежда и контролира правилността на входните данни. В нея правят впечатление „подсказките“ за това какво трябва да се въвежда, характерни за обучението в уводния курс по програмиране. Те ни подсказват, че се очаква програмите на зрелостниците да се проверяват ръчно, като проверяващият стартира програмата, получава подсказващото съобщение и въвежда данни от клавиатурата. В случай, че е въвел некоректни данни, се извежда съответният текст, проверяващият въвежда коректна стойност, след това получава отново подсказка, въвежда стойност и т.н.

Първото възражение е, че нито извеждането на подсказващи текстове, нито какъв текст трябва да изведе програмата при грешни данни, не е заложено във формулировката на задачата. Така изпитваният може изобщо да не постави подсказващ текст или да постави текст по свой избор и различните програми щяха да имат различно поведение върху едни и същи тестови примери. Второто възражение произлиза от споменатия вече факт, че във формулировката не е посочено и ограничение за стойността на *N*. Тези две особености изключват възможността за автоматизирано тестване на програмите. При тази постановка дори проверката с тестове с *N* от порядъка 20 – 30 би било истинско предизвикателство за проверяващия и той вероятно ще се ограничи с тестове с *N* в интервала 5 – 10.

Ако изпитването се отнасяше до уводен курс по програмиране, такъв подход може да се приеме. В случая, обаче, става въпрос за ДЗИ по профилиращия предмет информатика и би трябвало изискванията към изпитващите да са далеч по-високи, тъй като се очаква завършилите съответния профил да се занимават професионално с програмиране.

Поставената задача съвсем спокойно можеше да се формулира в следната много практична и полезна форма: „На стандартния вход са зададени номерата на обувките на *N* граждани. Напишете програма за намиране на честотите на отделните номера на обувки, носени от тези граждани“. Такава програма би била полезна за производители на обувки, за да преценят при пускане в производство на нов модел по колко екземпляра от всеки номер да произведат. Можем да си представим, че в този случай стойността на *N* може да достигне много големи величини (например 1 000 000, както беше посочено в нашата

примерна формулировка по-горе). Как би могла да бъде тествана съответната програма, ако статичната функция за обработка на входа е направена по начина в примерното решение? Отговорът е: „Няма как!“. Проверката в този случай трябва да се направи автоматично, като не само не трябва да се изискват никакви подсказки, а даже да се забрани на програмата да извежда такива. Освен това, за целите на автоматичната проверка задължителните текстови съобщения при откриване на грешки трябва да се специфицират в явен вид в условието.

3.2 Предложеният алгоритъм

В примерното решение е реализиран следният алгоритъм: за *i*-тия елемент на масива, със сравняване с предхождащите го, се проверява дали това е първото срещане на стойността a_i в този елемент. Ако това не е първо срещане, се преминава към следващ елемент на масива, а ако е първо, се преглежда остатъкът от масива и се намира и извежда броят на срещанията на стойността a_i.

Да приложим традиционния, сравнително прост подход за оценяване сложността на алгоритъма *в най-лошия случай*. За целта ще намерим, като функция на *размера на входните данни N*, броя на изпълнените сравнения на две цели числа при работа на програмата над вход с размер *N*. Не е трудно да се прецени, че най-голям брой стъпки програмата ще направи, ако всички числа в редицата са различни. В такъв случай, след като установи, че стойността a_i се среща за пръв път, алгоритъмът ще трябва да прегледа масива докрай, за да провери има ли и други срещания. Така за всеки от *N*-те елемента на масива алгоритъмът ще сравни стойността му с останалите $N - 1$ елемента. Общият брой сравнения ще бъде $N.(N - 1)$, а заедно със съпътстващите операции – $a.N.(N - 1)$ операции, за някаква константа *a*. Ясно е, че за въвеждане на входните данни ще са необходими $b.N$ операции за някаква константа *b,* или общо $b.N + a.N.(N - 1)$ операции. Означаваме този брой с $O(N^2)$, което, най-просто казано, означава, че във функцията за броя на реално извършваните от алгоритъма операции при вход с размер *N* най-бързо растящият едночлен е $c.N^2$ за някаква константа *c*.

Защо правим тези оценки? Да забравим за подсказките и невъобразимото количество време, което би се загубило за извеждане на екрана на 1 000 000 подсказки. Да допуснем, че компютърът, на който проверяваме решението, изпълнява *c* милиона операции за секунда. Тогава, за да проверим правилно ли работи програмата за тестов пример с размер 1 000 000, ще е нужно време 100 000 секунди, или повече от 27 часа – и това за един тестов пример. А както подсказахме с една практична постановка на задачата, може да се наложи програмата да се изпълнява за различни примери с подобен размер – отделни тестове за различните възрастови групи на населението, различни тестове за мъжки и женски обувки и т.н.

3.3. Форматиране на изхода

Както вече споменахме, при отсъствие на описание как трябва да се форматират резултатите от работата на програмата, единствена възможност за изпитваните е да следват примерния изход за зададения във формулировката на задачата примерен вход. Това, което се вижда в предложеното решение, е, че то използва различно форматиране от това, показано в примера – `Число` вместо `число и брой срещания` вместо `брой`. Очевидно за автора на задачата не е било важно да се спазва някакъв формат на изхода. А това прави автоматичната проверка на работите невъзможна.

Друга сериозна пречка не просто за автоматизиране на процеса на тестване, а изобщо за тестването, е явното указание за това, че редът, по който трябва да се изведат намерените резултати, е без значение. Предположението на автора вероятно е, че така улеснява изпитваните и всички програми ще извеждат резултата в реда на първото срещане на всяко от числата. Как обаче това може да се отрази на проверката? Как проверяващият при ръчно тестване ще провери правилността на изхода, ако следвайки „облекчението“, ученикът е написал програмата така, че извежда резултатите в различен ред – например в реда на последното срещане на всяко от числата или пък (защо не) в сортиран ред – от най-малко към най-голямо или обратно. По-долу ще покажем, че за последното има сериозни основания и можем да очакваме добре подготвените ученици да го направят.

4. Алтернативни решения

Едно възможно алтернативно решение, което трябва да е напълно по силите на изучаващите профилиращия предмет информатика, стига да са добре овладели възможностите, които компютърната памет им представя за съхраняване на междини данни, е следното. Лесно се вижда, че сложността на алгоритъма натежава от проверката дали за поредното число в редицата това е неговото първо срещане. За целта се проверява има ли го това число в предхождащите елементи на масива. Вместо това много по-ефективно е да се запомни фактът, че числото вече е срещано, и да се опрости проверката. Тъй като е известно, че числата са положителни, това може да се направи, като при преброяване на срещанията на числото a_i всяко следващо негово срещане в редицата се замени с 0. Получаваме следния по-прост код за главната функция:

```
static void Main(string[] args)
 {  try
    {  int[] array = readArray();
       for (int i = 0; i < array.Length; i++)
       {  if (array[i] != 0) // това първо срещане ли е?
          {  int count = 1;
             for (int j = i + 1; j < array.Length; j++)
```

```
                if (array[i] == array[j]) { count++; array[j] = 0; }
             Console.WriteLine(„число:“ + array[i] + „, брой : „ + count);
          }
       }
    }
    catch (FormatException)
    {  Console.WriteLine(„Некоректно въведено число“);  }
}
```

Трябва да отбележим, че при тази промяна в алгоритъма всички възможни случаи вече са еднакво „лоши“ и сложността му в най-лошия случай остава $O(N^2)$, но бързодействието му в средния случай ще се подобри значително заради избягването на сложната проверка за първо срещане.

Друго възможно алтернативно решение, което е напълно по-силите на изучавалите профилиращия предмет информатика, защото изисква само владеенето на елемент от учебната програма, е следното: да се сортират елементите на масива, при което всички еднакви стойности ще се подредят една след друга в сортирания масив и след това, с един пас през масива, да се намери за всяко число търсеният брой. След като в условието не се изисква извеждане на числата в определен ред, такова решение е напълно допустимо:

```
static void Main(string[] args)
{  try
   {  int[] array = readArray();
      Array.Sort(array);
      int current = array[0], i = 1, count = 1;
      while (i < array.Length)
      {  if (array[i] == current) { count++; i++; }
         else
         {  Console.WriteLine(„число: „ + current + „, брой : „ + count);
            current = array[i]; i = i + 1; count = 1;
         }
      }
      Console.WriteLine(„число: „ + current + „, брой : „ + count);
   }
   catch (FormatException)
   {  Console.WriteLine(„Некоректно въведено число“);  }
}
```

Да оценим сложността на това решение на C#. В средите от фамилията Visual Studio на Microsoft, за която можем да предположим, че се използва масово, стандартният метод за сортиране `Array.Sort` използва алгоритъма QuickSort. Сложността на този добре известен алгоритъм в най-лошия случай е $O(N^2)$, но също така добре известно е, че той работи достатъчно бързо

за естествено срещащи се множества от числа (т.е. най-лош случай за него може да бъде създаден изкуствено и това не е никак лесно, при използваните техники за рандомизиране на пивота). Така за оценките в практиката се използва неговата сложност в средния случай, която е $O(N.\log N)$. Само за да се почувства разликата, ако един алгоритъм със сложност $O(N^2)$ прави върху вход с размер 1000000 около 10^{12} стъпки, то алгоритъмът със сложност $O(N.\log N)$ прави около 2.10^7 стъпки.

И така, този вариант на решението ще направи $c.N.\log N$ стъпки за сортиране на числата и $d.N$ стъпки за обхождане на сортирания масив. Т.е. сложността му е $O(N.\log N)$ и може да реши задачата при входове с размери N в рамките на секунди.

И това не е всичко, което може да се направи в подобна задача. Да допуснем, че авторът беше поставил в условието споменатото по-горе ограничение за големината на числата в редицата. Например, числата в редицата са по-малки от K. В примера за практичен вариант на задачата, с търсене на честотата на различните номера обувки, такова естествено ограничение би било не повече от 50-60. И това е в сила за почти всеки практичен вариант на задачата. При това положение възможен алгоритъм би бил, вместо стандартното сортиране, да се използва *Сортирането с броене*[3]. В един масив от броячи `count[K]` по време на четене на числата от входа намираме броя на всяко от тях, като за всяко прочетено `a` увеличаваме брояча му – `count[a]++.` След това с един цикъл по броячите за всяко `count[a],` различно от 0, извеждаме `a` и `count[a].` Ако приемем условно ограничението $a_i < 1\ 000\ 000$ за големината на числата от предложеното примерно условие, получаваме следната програма:

```
static int[] count = new int[1000000];
static void readArray()
{   int n, a;
    do
    {  n = int.Parse(Console.ReadLine());
         if (n < 1) Console.WriteLine(„Въведете положително число!“);
    } while (n < 1);
    for (int i = 0; i < n; i++)
    {  do
       {  a = int.Parse(Console.ReadLine());
          if (a < 1) Console.WriteLine(„Въведете положително число!“);
       } while (a < 1);
       count[a]++;
    }
    return;
 }
```

```
static void Main(string[] args)
{   try
    {  readArray();
       for(int i = 0; i < 1000000; i++)
         if (count[i] !=0)
           Console.WriteLine(„число: „ + i + „„, брой : „ + count[i]);
    }
    catch (FormatException)
    {  Console.WriteLine(„Некоректно въведено число“);  }
}
```

Да намерим сложността на този алгоритъм в най-лошия случай. Той се състои от един цикъл със сложност $O(N)$ за прочитане на данните и обновяване на броячите и един цикъл със сложност $O(K)$ за обхождане на масива от броячи и извеждане на резултата. Така сложността му в най-лошия случай ще бъде $O(N + K) = O(M)$, където $M = \max\{N,K\}$. Така достатъчно големи екземпляри на задачата ще бъдат решени за по-малко от секунда.

5. Задачата от изпитната сесия през август – септември

В желанието си да анализираме задачите с алгоритмичен характер от ДЗИ по профилиращия предмет информатика, да разгледаме и „алгоритмичната“ задача в темата от сесията през август – септември[2].

Задача 25. Създайте проект с име zad25. Вашата задача е да напишете програма, която прочита от стандартния вход две цели числа a и b. Програмата да намира и извежда на стандартния изход решенията на квадратното неравенство $a.x^2 < b$. **Упътване:** могат да се използват функциите `Math.Sqrt` (C#) и `java.lang.Math.sqrt` (Java).

Примери:

Вход		Изход
a	*b*	
2	18	Решенията са (-3.00; 3.00)
–4	–1	Решенията са (-inf; -0.50) U (0.50; +inf)
2	0	Няма реални решения
–3	15	Всички реални числа са решения
4	t	Некоректно въведено число

Повечето от забележките, които отправихме към формулировката на задачата от сесията през май – юни са валидни и тук. За формàта на входните данни не е казано нищо и не става ясно дали ще са на един ред, или на два реда (за програмите, написани на C#, това е важно, а освен това изучавалите профилиращ предмет информатика би трябвало да могат да отделят двата параметъра на задачата, когато са прочетени в един низ). За двете числа

на входа не е казано от какъв вид са – във всички примери двете числа са цели. Но дали това трябва да е така? Защо при решаване на квадратно неравенство съответните коефициенти да не може да са дробни? Този въпрос би си задал и изпитваният и това би породило колебания.

Форматът на изхода също не е фиксиран и единствено от примерите изпитваният трябва да реши какво да извежда. Но след като в условието този формат не е фиксиран, може да се очакват най-разнообразни изходи, което би затруднило проверяващите. Фразата „Решенията са <*интервал*>“ е граматически некоректна. Правилно би било да се формулира „Решенията са числата в интервала <*интервал*>“. Буквата U, използвана в случая, в който решенията са в обединението на два интервала, както и означаването на безкрайността с inf също може да доведат до объркване, ако не са споменати изрично. Не е специфицирано също, когато решенията са в обединението на два интервала, трябва ли левият от двата интервала да е ляв аргумент на обединението, или там може да стои десният интервал. Но най-важно според нас е, че тъй като краищата на интервалите ще са дробни числа, а не е посочено с колко цифри след десетичната запетая трябва да се изведат, проверяващите може да се сблъскат с голямо разнообразие от форматирания на дробните числа. Така формулирана, задачата, разбира се, не може да бъде проверявана автоматично.

Колкото и да са важни направените по-горе забележки, трудностите, които те биха причинили на изпитваните и/или на проверяващите, са сравнително преодолими. Големият минус, според нас, на предлаганата задача е, че **тя не е алгоритмична.** Това е задача, която не проверява подготовката на изпитваните по програмиране, а **знанията им по математика**. Тук изобщо не може да говорим за сложност на алгоритъм, тъй като размерът на входа за всички възможни тестове е 2, а **числениият метод** (така е по-правилно да наречем процедурата, с която се решава задачата), който е необходим, е с константна сложност. Програмното решение на задачата дори не съдържа цикъл, а ще е съставено от няколко вложени `if-else` оператора. При условие, че като примери са посочени всички случаи, които трябва да бъдат разгледани, работата, която трябва да извърши изпитваният, не е много. Затова тази задача не представлява интерес от гледна точка на информатиката.

6. Заключение

Държавният зрелостен изпит по профилиращия предмет информатика през учебната 2021/2022 година е първи по рода си. Естествено е, че при липса на какъвто и да било опит за съставяне на изпитните теми не всички задачи от темите да са най-подходящите. Както посочихме вече, алгоритмичната задача от сесията през май – юни е много подходяща и при по-

добро формулиране на условието тя би изпълнила напълно ролята си – да се проверят възможностите на изпитваните да структурират добре данните, да използват правилно паметта за съхраняване на полезни за алгоритъма междинни резултати, както и да съставят/подбират добри, ефективни, алгоритми.

Задачата от изпитната сесия през август – септември не притежава необходимите качества. Тя не проверява нищо от споменатото по-горе и такива задачи е по-добре да не бъдат използвани в темите за ДЗИ за профилиращия предмет информатика.

БЕЛЕЖКИ

1. Държавен зрелостен изпит за профил „Информатика“, май 2022 г. https://web.mon.bg/upload/30769/2DZI_INFORMATIKA_V1.pdf, посетен на 22.03.2023
2. Държавен зрелостен изпит за профил „Информатика“, август 2022 г. https://web.mon.bg/upload/32862/2DZI_INFORMATIKA.pdf, посетен на 22.03.2023.
3. Изучаването на сортирането с броене не е предвидено в учебната програма за профилиращия предмет информатика, но е лесно за разбиране и би могло да се възложи на учениците за домашна работа да се запознаят с него.

ЛИТЕРАТУРА

АТАНАСОВ, Д., МАНЕВ, КР., СТОИМЕНОВА, В., ВОЙНОВА, Р., 2023. Тестовите задачи от държавния зрелостен изпит за профилиращ предмет информатика през учебната 2021/2022 година, *Математика и информатика,* том LXVI, № 1, 50 – 66.

REFFERENCES

ATANASOV, D., MANEV, KR., STOIMENOVA, V., VOYNOVA, R., 2023. The Test Tasks from the State Graduation Examination for the Informatics Profile During the Academic Year 2021/2022. *Mathematics and Informatics,* v. LXVI, №1, 50 – 66. [in Bulgarian]

MANEV, KR,, SREDKOV, M., BOGDANOV, TS., 2009. Grading Systems for Competitions in Programming. Proc. of the XXXVIII Spring Conference of the Union of Bulgarian Mathematicians, 103 - 116. http://www.math.bas.bg/smb/2009_PK/tom_2009/pdf/103-116.pdf

THE ALGORITHMIC TASKS FROM THE STATE GRADUATION EXAMINATION IN THE PROFILING SUBJECT OF INFORMATICS FOR 2022

Abstract. In the academic year 2021/2022, the State Matriculation Exam (the examination taken at the end of Secondary education to qualify for entry into University) was held for the first time for the students studying the profiling subject of Informatics in the Bulgarian secondary school. The article examines the tasks from the two sessions of the exam, which are supposed test the ability of high school students to select/create algorithms. The conditions to which a task of this type must meet and to what extent the tasks of the two exam topics satisfy these conditions are examined in detail.

Keywords: matriculation exam; competitive task; input and output format; algorithm; algorithm complexity

✉ **Dr. Krassimir Manev**
ORCID iD: 0000-0002-0284-312X
Journal Mathematics and Informatics
125, Tsarigradsko shose blvd., bl. 5
1113 Sofia, Bulgaria
E-mail: k.manev@azbuki.bg