МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2024
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2024
Proceedings of the Fifty-Third Spring Conference
of the Union of Bulgarian Mathematicians
Borovets, April 1–5, 2024

MATHEMATICS FOR BEGINNING PROGRAMMERS*

Krassimir Manev[1], Rumyana Karadjova[2]

[1]Journal "Mathematics and Informatics", Sofia, Bulgaria, e-mail: k.manev@azbuki.bg
[2]Sofia High School of Mathematics, Bulgaria, e-mail: rumikaradzhova@gmail.com

Computer programming is one of the most modern and promising professions of our time. To ensure sufficient manpower for the software industry, not only on a national but also on a global scale, it is necessary a systematic and comprehensive training of narrow specialists in the field. Start of the training of future programmers as early as possible is very important. The competitions (olympiads) in programming are a traditional form of early training for programmers with a very high qualification. Unfortunately, the knowledge of mathematics from the regular school classes, are rather not sufficient for the preparation of the participants in programming competitions. This research is based on task analysis from the national programming competitions for the very young students and aims to show what kind of mathematical knowledge and skills are required for preparation of successful competitors.
Keywords: competitive programming, education, tasks classification, mathematical skills.

МАТЕМАТИКА ЗА НАЧИНАЕЩИ ПРОГРАМИСТИ

Красимир Манев[1], Румяна Караджова[2]

[1]Списание „Математика и информатика", e-mail: k.manev@azbuki.bg
[2]Софийска математическа гимназия, e-mail: rumikaradzhova@gmail.com

Програмирането на компютри е една от най-модерните и перспективни професии на нашето време. За да се осигурят достатъчно кадри за софтуерната индустрия, не само в национален, но и в световен мащаб, е необходима систематична и всеобхватна подготовка на тесните специалисти в областта. От огромно значение е обучението на бъдещите програмисти да започва колкото може порано. Състезанията (олимпиадите) по програмиране са традиционна форма за ранно обучение на програмисти с много висока квалификация. За съжаление, знанията по математика, които редовният училищен курс съдържа, не винаги са достатъчни за подготовката на участниците в състезанията по програмиране. Това изследване е базирано на анализ на задачите от националните състезания по програмиране за ученици от най-младшата възрастова група и има за цел да покаже какви математически знания и умения са необходими за подготовката на успешни състезатели, за да се търсят най-добрите начини за постигане на такива знания и умения.
Ключови думи: състезателно програмиране, обучение, класификация на задачите, математически умения.

*https://doi.org/10.55630/mem.2024.53.025-035
2000 Mathematics Subject Classification: 68-00.

1. Въведение. Наскоро излезе от печат книгата [1], предназначена да подпомогне подготовката на учениците, които вземат участие в националните и регионалните състезания по програмиране в най-младшата състезателна група Е (до 5 клас, включително). След като в първата глава на книгата са представени състезанията по програмиране, в останалите глави подробно са разгледани няколко тематични области, в рамките на които се съставят състезателните задачи за тази група. Тематичните области са подредени хронологично, в реда по който би трябвало да се изучават в школите по състезателно програмиране, така че учениците да се подготвят своевременно за отделните състезания през годината – шестте национални (трите кръга на Националната олимпиада по информатика, Есенния, Пролетния и Летния национални турнири) и двете регионални (Есенния софийски и Пролетния софийски турнири).

За по-лесното му усвояване, материалът във всяка тематична глава е представен в няколко раздела, на които може да се разбие темата. Изложението във всеки раздел започва с въвеждане на необходимия теоретичен материал – както алгоритмичен, така и математически. След това материалът се илюстрира с решенията на няколко задачи от състезания. В края на всеки раздел са събрани задачите, давани на състезания у нас (от 2007 г. до наши дни), за решаването на които са необходими знанията и уменията, представени в раздела, с кратки указания за решаването им.

В тази статия обобщаваме нашия опит от преподаване на програмиране и подготовка на състезатели в ранна възраст, като представяме тази част от направения за целите на книгата анализ, която касае необходимите за начинаещите програмисти математически знания. Целта е да установим за кои от тях състезателите могат да разчитат на редовните часове по математика, а кои трябва непременно да бъдат преподавани в школите по състезателно програмиране (или в школите по математика, които много от състезателите по програмиране в тази възраст посещават).

2. Начални познания. Подготовката на състезатели по програмиране започва със заниманията в курс Увод в програмирането, където обучаваните се запознават със синтаксиса и семантиката на един език за програмиране (за състезанията по програмиране този език, без каквото и да било съмнение, трябва да е C/C++). Затова, изграждането на математическата култура на бъдещия програмист би трябвало да започне още от този курс. В този раздел представяме някои важни математически понятия, запознаването с които би трябвало да се направи паралелно, както в заниманията с програмиране, така и с математика.

На първо място ще спомена понятието *променлива*. Заради високата абстрактност на това понятие, използването му в обучението по математика става малко по-късно – в 6. – 7. клас. При изучаването на език за програмиране от процедурен тип, какъвто е C/C++, такова отлагане е **невъзможно**. За да може една програма да обработи някакви данни, в кода ѝ трябва да бъдат *декларирани* променливи, които са необходими за съхраняване на входните данни, както и променливи, в които да се съхраняват междинни резултати.

Трудността, с която неизбежно се сблъскват учениците в случая е *двойствеността* на понятието променлива – единство на *име* и *стойност*. Както в математиката, така и в програмирането, споменаваме една променлива с нейното име, а всъщност използваме стойността ѝ в предписаното пресмятане или обработка. За програмирането тази двойственост е от ключово значение. Тя ни позволява с един

и същ фрагмент код да правим различни обработки, като сменяме стойностите на променливите, чиито имена са споменати във фрагмента. Това намалява обема на необходимия код и прави програмите по-лесни за четене и изчистване от грешки. Практиката, в началния етап на обучение на програмисти входните данни да се задават в кода на програмата, скрива същността на понятието променлива и не е полезна.

Ако начинаещият програмист не разбира добре двойнствеността на понятието променлива, създаването на ефективен код, както е споменато по-горе, е затруднено. А по време на тестването на програмата, неспособността на ученика да си представя ясно как се сменят стойностите на променливите, чиито имена участват в някой фрагмент, води до невъзможност за ефективно тестване на програмата. Докато, ако си е изяснил добре двойствеността, включително и в заниманията с математика, тези процеси няма да го затруднят. Когато начинаещият програмист добре разбере двойнствеността на понятието променлива и упражнява систематично това разбиране при написване на програми, той няма да има никакъв проблем, когато в обучението по математика се наложи да решава задачи, в които данните са зададени не с константи, а с променливи.

Тук е мястото да обърнем внимание и на един вид променливи, употребата на които старателно се избягва, за дълго време, при обучението на начинаещи програмисти – променливите от тип *указател*. Няма нищо свръхестествено в този тип променливи, както обичайно се твърди. Просто стойностите на този вид променливи са адреси на полета от компютърната памет. Имената на масиви, както и имената на функции, употребата на които е неизбежна, дори в този ранен етап, са променливи от тип указател. Недоброто разбиране на същността на понятието променлива като цяло води до избягване на използването на променливи от този тип в ранните етапи на обучение на програмисти, а това влияе отрицателно на нивото на подготовката им.

След като начинаещите програмисти са се справили с разбирането на понятието променлива, на дневен ред излизат понятията *операция* и *израз*. Да наречем с понятието *(затворена) алгебрична система* двойката (Σ, Ω), където Σ е някакво множество от стойности, а Ω – множество от операции, чиито *аргументи*/*операнди* са стойности от Σ и резултатът от изпълнението на операция е стойност от Σ. Училищната математика традиционно се ограничава с *числовите* алгебрични системи, като постепенно разширява обхвата на множествата Σ и Ω. Отначало стойностите са целите положителни числа, а операциите – събиране и умножение. Добавянето на операцията изваждане води до добавянето на нулата и целите отрицателни числа в множеството от стойности, а на операцията деление – до добавяне в множеството на стойностите на дробните (рационалните числа) и това по същество е всичко, което може да се използва в програмирането от училищния курс по математика.

За обучението на програмисти това не е достатъчно. Още в ранните етапи на обучение начинаещият програмист неизбежно се сблъсква с втора алгебрична система – *двоичната логика*, в която стойностите са *истина* (`true`) и *неистина* (`false`), а операциите са *дизюнкция* (*или*), *конюнкция* (*и*) и *отрицание*. Преобладава мнението, че двоичната логика може да се преподава систематично едва от 8 клас нагоре, понеже затруднява учениците. А всъщност научаването на таблиците на трите логически операции е много по-лесно от научаване на таблицата за умножение на

десетичните цифри, да не говорим за трудността на операцията деление на цели числа, която се изучава в 4 клас.

Да добавим още, че двоичната логика е основата на математическите разсъждения, които изучаващите математика безусловно трябва да владеят при решаване на задачи. Освен това, тя по естествен начин се разширява до алгебричната система на *подмножествата* на някакво множество със същите операции, наричани *обединение* (вместо дизюнкция), *сечение* (вместо конюнкция) и *допълнение до универсалното* (вместо отрицание). А доброто познаване на логическите операции естествено се разширява до познаването на *побитовите логически операции* – мощен инструмент в арсенала на най-квалифицираните програмисти, които се занимават най-вече със създаването на сложни софтуерни продукти – операционни системи, компилатори и т.н.

Освен това, в бъдещата си работа начинаещите програмисти ще се сблъскат с много по-трудни за усвояване алгебрични системи, като *алгебрата на Клини* над множеството от думите (низовете) над крайна азбука, и най-вече с *обектно-ориентираното програмиране*, чиято същност е изграждането на различни алгебрични системи над множества от *обекти*, наричани *класове*, и оперции над обектите от класа, имплементирани в *методи*.

Когато се занимаваме с алгебрични системи, много важни са различните *свойства на операциите*, като наличието или отсъствието на комутативност, асоциативност и/или дистрибутивност на една или друга операция в числовите алгебрични системи. Отсъствието на съответен опит в изследването и използването на свойствата на операциите в една алгебрична система води до нейното неефективно прилагане. Затова по-обстойното запознаване с различни алгебрични системи, както в обучението на програмисти (където е неизбежно), така и в обучението по математика, сериозно би обогатило математическата култура на учениците.

Тук бихме искали да формулираме един принцип, който може и трябва да бъде активно използван при преподаването, особено в условия на ограничен хорариум – *доброто познаване на една* ***видова концепция*** *(каквато в случая е някоя от числовите алгебрични системи) трябва да бъде използвано за бързо въвеждане и на други видови концепции от същата* ***родова концепция*** *(каквато в случая е концепцията алгебрична система).* Ще наричаме този принцип *родово обобщаване.*

За някои читатели, дори запознати с програмирането, може да се окаже изненада, че езикът за програмиране C и наследилите от него синтаксиса си езици (като C++, C# и т.н.) предоставят неочаквано много операции над различни типове стойности – в C++ те са 67 (шестдесет и седем!!!). При това положение да се очаква, че само познаването на числовите алгебрични системи и евентуално двоичната логика са достатъчни за обучението на програмисти, е неоснователно.

Смисълът на операциите на една алгебрична система е в съставянето с тях и стойностите на системата (представени като константи или в променливи) на *изрази*. Понятието израз е сложна рекурсивна конструкция и в училищната математика дефинирането му се избягва. Но съставянето на правилно конструирани и смислени изрази е фундаментално за математиката и за програмирането. Както вече споменахме, C-подобните езици за програмиране предоставят голям брой операции, а това дава много възможности за съставяне на полезни изрази.

Освен обичайните *числови* (или *аритметични изрази*), най-познати за ученици-

те, и неизбежните за програмистите *логически изрази*, важни за бъдещите програмисти са и изразите, в които участват споменатите по-горе указатели – т.нар. *адресни изрази*, изразите с *побитови операции*, както и изразите в алгебрата на Клини, наричани *регулярни изрази*. Затова, дори без да се стига до формална дефиниция, изясняването на понятието израз е важно – както за информатиката, така и за математиката. За целта ще бъде полезно да се използва формулираният по-горе принцип на родово обобщаване, като всяка нова алгебрична система се въвежда бързо, чрез сравняване свойствата на нейните операции със свойствата на операциите в някоя позната алгебрична система.

Ключови за доброто разбиране на понятието израз от произволен вид са понятията *приоритет* и *асоциативност* на участващите в него операции, както и *употребата на скоби*, за определяне реда на прилагане на операциите. Докато с приоритета на аритметичните операции и употребата на скоби учениците в тази възраст се запознават още в началното училище, за асоциативност почти не се говори, тъй като при числовите изрази тя е само лява. Що се отнася до употребата на скоби, доброто познаване на този механизъм има два аспекта, всеки от които е важен за програмиста. Нашите наблюдения показват, че *разкриването на скоби* в израз, в алгебрична система в която има двойки операции, удовлетворяващи дистрибутивния закон, представлява проблем. При уменията за прилагане на дистрибутивните свойства в обратна посока – *изваждането на аргумент пред скоби* – състоянието е много по-лошо.

Нека разгледаме съвсем прост пример за неглижиране на изваждането пред скоби, който се отразява много зле на ефективността на програмите. В тялото на цикъл се налага да се пресметне изразът $ab+ac$. Записването на израза по този начин води до изпълняването на две умножения и едно събиране. В същото време за пресмятане на еквивалентния израз $a(b+c)$ е нужно само едно умножение. Предвид факта, че времето на процесора за изпълняване на едно умножение е много по-голямо от времето за изпълняване на едно събиране, ако тялото на цикъла трябва да се изпълни милион пъти, тази загуба на бързодействие е значителна.

3. Математика в ранния етап на решаване на състезателни задачи. Под ранен етап в подготовката на състезатели имаме предвид времето преди състезателите да са писали програми с използване на оператори за цикъл и масиви. В този етап задачите са силно ориентирани към математическата подготовка и математическите способности на състезателите. Затова се налага, по време на подготовката за първите състезания, сериозно да се обърне внимание на обогатяване и затвърдяване на знанията по математика.

Първата тема, с която обикновено се започва, е Частно и остатък при деление на цели числа. Най-общата математическа постановка в задачите по тази тема е: Зададено е множество от обекти с N елемента и тези елементи трябва да се разпределят в групи от по K елемента, като важен за решението на задачата е броят на получилите се групи. Възможни са два варианта – или се търси броят на „пълните групи“ с K елемента, който е частното от *целочисленото деление* N/K или броят на всички групи, включващ и възможната единствена група с по-малко от K елемента. За целта се използва операцията $N\%K$ – намиране на остатък при целочислено деление, която състезателите трябва да познават от уводния курс по програмиране. В случай, че остатъкът е ненулев, тогава трябва да се добави единица към частното N/K.

Темата Частно и остатък предлага много възможности да се проверят и усъвършенстват не само програмистките, но и математическите качества на състезателите. Като пример можем да посочим задача от състезание, за решението на която трябва да се намери израз за най-малкото четирицифрено число, което се дели на d и най-голямото трицифрено, което се дели на d. В тази възраст учениците познават понятието остатък от уроците по математика, без да му се обръща особено внимание, а то води по естествен път към дробните числа, както и към понятията *долна цяла част* и *горна цяла част*. Трудностите, които учениците изпитват при работа с дроби, води до избягване в тази възраст на задачи, изискващи използването на променливи от дробния тип `double`.

Следваща тема, която често се използва за съставяне на задачи за състезания по програмиране за разглежданата възрастова група, е Бройни системи, като най-често става дума за добре познатата на учениците десетична бройна система. Основен проблем при тези задачи може да бъде непознаването на операцията степенуване. Затова състезателите трябва да бъдат запознати с операцията *степенуване*, което в училищния курс става по-късно.

Две са базовите подзадачи в тази тема, които често се срещат като част от задачите по темата – отделянето на цифрите на число, записано в десетична позиционна бройна система, и намиране на числото по зададени цифри. След като учениците са се справили успешно с темата Частно и остатък и са се запознали с операцията степенуване, лесно можем да им покажем естествено решение на първата задача, тъй като най-младшата цифра на записа на едно число е остатъкът при деление на основата на системата, например 10, а с целочисленото деление на числото на 10 премахваме най-младшата му цифра.

Обратната задача изисква повече внимание. В задачите за тази възраст числата са с малко (3–5) цифри в десетична система, така че познавайки полиномиалния запис по степените на основата състезателите могат лесно да напишат нужния им аритметичен израз за възстановяването на числото. Но стремейки се към обобщаване е ясно, че това не е подход, когато броят на цифрите на числото и/или основата на системата са параметри на задачата. Тривиалният програмистки подход би бил, за всеки едночлен на полиномното представяне да се намери нужната степен на основата, да се умножи по съответната цифра и да се събере полученото с намерената до момента част от сумата. Добре познаващите операцията степенуване програмисти лесно ще съобразят, че всяка следваща степен се получава с умножаване на предходната по основата, и ще ускорят процеса на пресмятане като, при n-цифрено число, сведат броя на умноженията до $2n - 2$.

В този случай добрата математическа култура може да ни доведе до още по-добро решение. С прилагане на „трудния“ дистрибутивен закон за умножението спрямо събирането можем да покажем как представянето

$$N = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \cdots + a_0b^0$$

може да се преобразува в еквивалентното

$$N = (\ldots((0.b + a_{n-1}).b + a_{n-2}).b + \cdots).b + a_0$$

(*правило на Хорнер*), при което възстановяването на число от неговите цифри изисква само $n - 1$ умножения.

Следвайки принципа за родово обобщаване, лесно можем да пренесем опита от

десетична бройна система за другите бройни системи, които се използват в информатиката – двоичната, осмичната и шестнадесетичната (която използва като цифри и буквите A, B, C, D, E и F). Тези бройни системи много рядко биха се срещнали в задачи за тази възрастова група, но съответен опит би бил много полезен за състезателите в бъдеще. Тук е възможно още едно обобщение. Използването на букви като цифри в шестнадесетичната бройна система ни подсказва, че за цифри на бройна система може да използваме буквите на някаква азбука, например малките латински букви, съпоставяйки им поредни номера от 0 (за `a`) до 25 (за `z`). Така, низовете над азбуката може да бъдат разглеждани като числа записани в 26-тична позиционна бройна система – подход, които със сигурност ще е полезен на състезателите в бъдеще.

Трета тема в този ранен етап на подготовка е Подреждане на числа (или други обекти, които могат да са приравнят към числови, каквито са знаците на клавиатурната азбука) и в частност – намирането на минимален и максимален елемент на множество от стойности. В програмистката практика наричаме алгоритмите за подреждане *сортиране*. Лекотата, с която бихме се отнесли към тази тема, имайки предвид, че учениците добре познават сравняването на числови стойности, е недопустима. Сравнително простите задачи, давани на състезания на този етап, изискват подреждането на малък брой стойности, но чувството за важността на сортирането на зададените стойности, като първа стъпка към построяването на ефективно решение на задачата, трябва да се изработи още на този етап от обучението.

Друга тема, от която много често се дават задачи на състезания в тази възрастова група, е Мерни единици. Основната техника в този вид задачи е превръщане на всички стойности в най-малката мерна единица, спомената в задачата, извършване на нужните пресмятания в най-малката мерна единица и връщане на резултата в мерните единици, определени от условието. Затова, за тази тема е най-съществено да се знаят мерните единици и на колко по-малки единици се разделя една по-голяма, когато такова разделяне е възможно. Особеностите тук са мерните единици, които имат специфично, различно от десетичното разделяне Такива са, например, мерните единици за време (1 минута = 60 секунди, 1 час = 60 минути, 1 денонощие = 24 часа, 1 седмица = 7 денонощия, а броят на денонощията в един месец или една година е различен за различните месеци и години), както и единиците за обема на компютърни данни, при които разделянето е през кратни на 10 степени на двойката – 1 кибибайт = 2^{10} = 1024 байта, 1 мибибайт = 2^{20} байта = 1024 кибибайта, 1 гибибайт = 2^{30} байта = 1024 мибибайта и т.н.

4. Ad hoc задачи. Една специфична категория задачи, не само в тази възрастова група, а във всички състезания по програмиране, са така наречените *ad hoc* (от латински – специфичен, създаден само за целта). Това са задачи, за които е трудно или практически невъзможно да бъдат отнесени към някоя от добре очертаните теми. Ролята на тези задачи е много важна, тъй като те провокират творческото мислене на състезателите. Още повече, че в състезанията масово се използват ad hoc задачи, за да се проверят креативността на учениците и способността им да създават свои алгоритми, които не са научени от книгите. Разбира се, в решението на една ad hoc задача може да се срещнат като подзадачи алгоритмични стъпки, които състезателите трябва да са научили от книгите и заниманията в школите.

Анализът на използваните през годините състезателни ad hoc задачи за тази въз-

растова група показва, че едно важно за решаването на тези задачи умение, е умението **да се разбие множеството** от възможни входни данни на различни случаи и **да се изчерпят** различните случаи. Това ни води до необходимостта учениците в тази възрастова група да познават понятията *подмножество* (или *комбинация*), *вариация* и *пермутация* за неголеми базови множества – 3, 4, 5 елемента, така че елементарни познания по комбинаторика са безусловно необходими. В училищния курс за тези понятия става дума много късно. Преди години излезе от печат книгата [2], предназначена да запознае ученици от 3. и 4. клас с основни понятия на комбинатериката на достъпно за тях ниво. Авторката използваше с успех материала от тази книга в извънкласни форми. За съжаление книгата не получи необходимата популярност.

Други задачи, които на този етап наричаме ad hoc, защото не се предполага дълбоко познаване на необходимата математика, изискват съставяне и решаване на несложно *линейно уравнение* (или достатъчно проста *система от линейни уравнения*). Те имат пряка връзка с често използваните задачи от същия тип в обучението по математика и се оказват доста трудни за ученици дори в 7. клас. Ще илюстрираме трудностите, които изпитват повечето състезатели в тази възрастова група при решаване на тези не толкова сложни задачи с един пример. Количествата от четири продукта в склад са обвързани с прости линейни зависимости:

$$a + b + c + d = n; a + b = k; a = b; b + c = m,$$

където n, m и k са зададени. Задачата е да се намери количеството на всеки от продуктите. Както се вижда, системата е съвсем проста и не са нужни свръхусилия, за да се намерят формули за количествата: $a = b = k/2, c = m - b, d = n - a - b - c$. В състезанието са участвали 46 ученици, като 12 са написали решение за 30 т. (от възможни 100), 4 са написали решение за 12 т., а 30 не са получили точки.

По-подробното вглеждане в ad hoc задачите ще покаже и други математически умения, които трябва да се владеят, за да се постигне успех при решаването им, но основна, и решаваща в случая, е способността **да се четат и интерпретират правилно** *формални описания* (*дефиниции*) на обекти и събития и *твърдения* (*теореми*), задаващи свойствата на така дефинираните обекти. При много от учениците в тази възраст тази способност не е добре развита. Отказът да се заучават дефиниции и формулировки на твърдения, да се вниква в смисъла им и да се използват при решаване на задачи наблюдаваме дори и при студенти в информатичните специалности. Това ни отвежда към много сериозната дискусия за причините за т.нар. функционална неграмотност на някои ученици, за която съвършено неоснователно се обвинява прекаленото теоретизиране. А на практика такова няма!

5. Цикли и масиви. Когато в процеса на обучение състезателите от тази възрастова група достигнат до задачите, в които се изисква съставяне на цикли и използване на масиви, центърът на теоретичната подготовка се измества от математическите знания и умения, които са решаващи в началния етап на подготовката, към чисто програмистки умения и запознаване с основни алгоритми. Но това не означава, че в този етап на подготовка не са нужни математически знания и умения.

На първо място трябва да обърнем внимание на усложняването на един от проблемите, за които вече споменахме – усета за двойнственост на понятието променлива. Управляващите променливи на създаваните цикли имат много по-голяма динамика

на изменение на стойностите, отколкото при статичните, нециклични конструкции и затова проследяването на измененията в тях е по-трудно. Още по-трудно става проследяването на измененията в стойностите на елементите на масив, тъй като вече става дума за изменения на двойки от стойности – на индексната променлива, определяща мястото на елемента в масива, и на самия елемент. Затова в началото на статията определихме като ключова способността зад имената на променливите да се виждат ясно техните стойности в различните моменти от работата на програмата. А това умение може да бъде постигнато и с решаване на математически задачи, в които стойностите са зададени като променливи, а не като константи.

Един от критериите при оценяване на решенията на състезателни задачи, освен *коректността на резултата*, е времето, за което е получен. Непростото математическо понятие, което стои зад този критерий наричаме *сложност на алгоритъма по време* (в най-лошия случай). За да е успешен един състезател, трябва отрано да свикне да *оценява* сложността на намисления от него алгоритъм. За целта, първо трябва да съпостави на всяка комбинация от входни данни *размер* N на входа – например, броя на числата в един тестов пример. След което да намери функция на N, стойността на която е броят на операциите[1], които алгоритъмът ще изпълни при вход с размер N. Поведението на тази функция (растежа на стойностите ѝ, когато нараства N) е мярка за сложността по време (т.е. за бързодействието) на алгоритъма.

Когато програмите не използват цикли, лесно се вижда, че броят на изпълнените операции не се различава съществено при различните тестови примери и може да го ограничим с някаква константа – означаваме сложността на такива алгоритми с $O(1)$. Когато в програмата се използва цикъл, например най-често използвания в такива случаи

```
for(i = 0; i < N; i++) {тяло на цикъла}
```

и в тялото на цикъла няма друг цикъл, т.е. сложността му е $O(1)$, тогава лесно обясняваме, че сложността на цикъла е някаква линейна функция на N, която означаваме с $O(N)$. Когато в програмата има няколко отделни цикъла с линейна сложност, тогава сложността на цялата програма също е линейна, тъй като сума на няколко линейни функции на N е линейна функция. Както се вижда, за създаване на умения за оценяване сложността на използваните алгоритми от огромно значение е математическата култура на състезателя.

С въвеждането на задачи, изискващи влагане на цикли, нещата се усложняват. Причината е, че тялото на вътрешния от два вложени цикъла вече не е с константна сложност, както в класическите алгоритми за сортиране, и са необходими допълнителни усилия. Използвайки уменията за разглеждане на всички възможни случаи, придобити при решаване на ad hoc задачи, оценяваме сложността на вътрешния цикъл, като за размер на входа при него използваме стойността на управляващата променлива на външния цикъл при влизане във вложения, и сумираме получените стойности. При такова оценяване е важно състезателят да знае да намира сумата на първите няколко члена на прогресия, което в училищния курс се изучава много късно. Затова за начинаещите състезатели трябва да се обяснят правилата за сумиране

[1] Броят на изпълнените от алгоритъма операции, разбира се, не е най-точната мярка за бързодействие на алгоритъма, но за практически цели тя е напълно достатъчна.

при някои характерни прогресии – например, сумата на целите числа от 1 до N, на нечетните числа от 1 до $2K + 1$ или на първите N степени на двойката. Първата от тези суми може да се обясни на ученици дори в 5 клас по начина, приписван на невръстния Гаус.

Голямото предизвикателство при организирането на вложени цикли е правилното определяне на границите, в които се мени управляващата променлива на вложения цикъл, които са функции на управляващата променлива на външния цикъл. В [1] предложихме класическа техника за намиране на двата израза – разглеждане на стойностите на двете граници за първите няколко стойности на променливата на външния цикъл, построяване на **хипотеза** и проверка на хипотезата за следващи няколко стойности. Наблюдението на стойностите на търсена функция, изработване на хипотеза за израза, чрез който може да я пресмятаме и проверка на хипотезата (особено когато не разполагаме с време за доказателството ѝ), е важна математическа техника, която за съжаление рядко се упражнява в училище.

Работата с двумерни масиви отново ни отпраща към училищната математика, тъй като двумерният масив е опростен вариант на един от квадрантите на класическата Декартова *координатна система*. При работа с двумерни масиви се налага да се използват двойките от цели *координати* на елементите в двумерен масив – номер на ред и номер на стълб, което в училищния курс става не по-рано от 6-7 клас. При това трябва да се има предвид, че координатната система на двумерния масив е леко видоизменена спрямо традиционната – завъртяна е на 90° около началото в посока на часовата стрелка, т.е. първата координата е индексът на реда, а втората – индексът на стълба.

6. Делимост на числа. Това е последната тема, от която се дават задачи в тази възрастова група. Тя по естествен начин разширява и допълва темата Частно и остатък. Затова се предполага, че понятията *частно*, *остатък*, *дели се* (*е кратно*) *на*, *е делител/множител на*, са добре познати. Необходимо е освен това да се знаят популярните признаци за делимост на 2, 3, 5, 9, както и понятията *най-голям общ делител* (НОД) и *най-малко общо кратно* (НОК).

С понятието НОД е свързано запознаването с една от първите **циклични** алгоритмични техники с математическа основа – *алгоритъма на Евклид*. Сам по-себе си този алгоритъм (в ефективната му форма с намиране на частно и остатък, вместо с изваждане) не представлява програмистка трудност и лесно се научава. Но много по-важно в случая е разбирането на математическата му същност. Защото в следващ етап състезателите непременно трябва да се запознаят и с *разширения алгоритъм на Евклид* (за целите на решаване на линейни Диофантови уравнения), научаването на който не е елементарно и само доброто познаване на математиката зад алгоритъма на Евклид може да позволи на състезателите да могат да го пишат правилно, без да го научават наизуст.

На този етап много задачи за написване на програма са свързани с понятието *просто число*. По-лесни такива задачи са, например, да се провери дали едно число N е просто или да се намери разлагането му на прости множители. Ключово за тези задачи е сравнително простото твърдение, че проверката дали i е делител на N може да се ограничи до максималното i, такова че $i^2 \leq N$. Т.е. необходимо е поне интуитивно да се въведе понятието *квадратен корен*, изучаването на което става много по-късно. В по-сложни задачи състезателите трябва да могат да на-

мират простите числа в зададен интервал, за което е препоръчително да познават същността на техниката, известна като „сито на Ератостен“, или да намират броя на всички делители на зададено число с помощта на степените на простите множители на числото – нещо, което рядко се преподава в училище.

7. Заключение. При подготовката на състезатели по програмиране в най-младшата възрастова група се налага не само отлично владеене на езика за програмиране и операционната система, а и владеене на различни математически знания, както и умения да се прилагат тези знания за решаване на практически задачи, каквито са задачите в състезанията по програмиране. Считаме, че симбиозата на математически знания и програмистки умения, изграждана систематично при състезателите по програмиране от много години, е един от най-добрите примери за реализация на така модната напоследък концепция STEM, при това осъществена много преди тази концепция да е публично осъзната и популяризирана.

Безусловно, не всички нужни за целта знания и умения могат да бъдат получени в редовните часове по математика. Затова липсата им трябва да се компенсира със съответни занятия в извънкласните форми по програмиране. Необходимо е това да се прави интелигентно и с отчитане на особеностите на възрастта, най вече за най-малките състезатели. Важно е да се отбележи, че запознаването с математически понятия и тяхното използване за решаване на програмистки задачи, по изключително добър начин обосновава необходимостта от изучаване на абстрактни математически теории, и е най-точен отговор на обвиненията, че в часовете по математика се изучават ненужни неща, приложимостта на които е непонятна за учениците.

Анализът, направен за целите на тази статия, показва, че учениците в тази възраст, обучаващи се в математически гимназии, до голяма степен имат нужните за програмиране математически знания и умения, така че попълването им в школите по програмиране не е голям проблем. За учениците в регулярните училища недостигът от такива знания и умения е значителен и това до голяма степен предопределя недобрите им резултати в състезания по програмиране.

Надяваме се, че дискутираната тук проблематика може да стане повод за създаването на учебно пособие, което систематично и в подходяща за учениците от 4. – 5. клас форма да въвежда нужните за начинаещите програмисти математически понятия и твърдения, за които така или иначе не се намира място в учебните програми на българското училище или изучаването им се случва твърде късно, в сравнение с темпото, с което се подготвят учениците за участие в състезания. Заглавието на тази статия би било напълно съответно на очакваното съдържание на такова учебно пособие.

ЛИТЕРАТУРА

[1] Кр. Манев. Увод в състезателното програмиране. КЛМН, 2023 [Kr. Manev. Uvod v sastezatelnoto programirane. KLMN, 2023].

[2] З. Савова. Случайност + Математика. Изкуства, 2009 [Z. Savova. Sluchaynost + matematika. Izkustva, 2009].