ДОСВІД ПРОВЕДЕННЯ ОЛІМПІАДИ З ІНФОРМАТИКИ В ДОНЕЦЬКІЙ ОБЛАСТІ

Неспірний В.М., Глухов В.О.

На ІІ етапі Всеукраїнської олімпіади з інформатики в Донецькій області учням 8–9-их класів пропонується для розв'язання 5 теоретичних завдань, а учням 10–11-их класів — 3 практичних завдання. Завдання теоретичного туру виконуються на навчальній алгоритмічній мові або на деякій мові програмування, практичні завдання — на мові програмування Pascal, C або C++.

Кожна задача, запропонована на олімпіаді в 8-их і 9-их класах, оцінюється по 60 балів.

Результатом роботи учасника для кожної задачі є алгоритм її розв'язку. Учень повинен самостійно визначити та описати аргументи (вхідні дані) та результати (вихідні дані), а потім скласти та записати правильний алгоритм формування цих результатів.

Під час перевірки робіт оцінюються такі складові — ідея розв'язку, її правильність та ефективність, правильність її реалізації у вигляді алгоритму або програми та синтаксична коректність запису алгоритму на вибраній учасником мові програмування. Словесне формулювання ідеї не є обов'язковим для учня, а у запропонованих учнями розв'язках ідея не завжди очевидна. У цьому випадку, того щоб її зрозуміти й оцінити алгоритм, члени журі здійснюють так зване «прокручування» алгоритму на мінімальних наборах даних, тобто задають деякі аргументи, для яких неважко відразу знайти результат, і потім покроково виконують усі команди алгоритму.

Як правило, ефективність алгоритму (сюди включається його складність за часом та за пам'яттю, розмір самого алгоритму) не є ключовим критерієм в оцінці теоретичного завдання. У будь-якому випадку правильний алгоритм з високою часовою складністю отримає більшу кількість балів, ніж неправильний.

Правильно описана ідея алгоритму без реалізації оцінюється в 10% від ваги задачі.

Для 10–11-их класів задачі, запропоновані на олімпіаді, оцінюються по 100 балів. Результатом роботи учасника для кожної задачі є файл із текстом програми на одній з дозволених мов програмування. Кожен учень отримує «Пам'ятку учасника практичного туру», де докладно описуються технічні вимоги, яким повинна задовольняти робота. Усі вони є важливими, оскільки перевірка практичних завдань здійснюється у автоматичному режимі за допомогою спеціальної програми.

Програма учасника компілюється та запускається на заздалегідь підготовлених журі тестових файлах, після чого результати, що видала програма, перевіряються на коректність. Останню функцію виконують також спеціально підготовлені програми — так звані валідатори. Їх використання важливе, коли задача допускає декілька правильних відповідей.

Як правильно оцінити те або інше рішення?

У роботі Е.В. Андрєєвої [1] описано етап складання тестів для автоматизованої системи перевірки рішень, відзначається користь застосування даної системи перевірки рішень, що є «незалежним арбітром» і створює в учнів почуття об'єктивності під час перевірки. Більш докладний опис автоматизованої системи тестування міститься в роботі В.А. Матюхіна [4]. Відділом інформаційних технологій Донецького обласного інституту післядипломної педагогічної освіти з 2000 року розв'язки задач всіх учасників перевіряються в автоматичному режимі спеціальною програмою.

Під час складання задач з інформатики в ідеальному випадку хотілось би, щоб у них були відбиті всі розділи науки інформатики — тоді вони стануть досить гарними тестами на знання інформатики.

Важливо сформулювати, наскільки ефективний алгоритм повинен бути реалізований у процесі розв'язання. У багатьох завдань є перебірний розв'язок, що полягає в перегляді всіх можливих варіантів і вибору з них найкращого. Про такий розв'язок, на перший погляд, легко здогадатися, але його реалізація нерідко вимагає програмістських зусиль. Задачі з перебірними розв'язками є досить важливими під час вивчення інформатики. Необхідно врахувати, що для одного й того ж завдання може бути кілька ефективних розв'язків, що значно різняться між собою. У постановці завдання необхідно визначити, реалізація якого алгоритму буде повним розв'язком завдання. Один із способів довести це до учня — встановити обмеження на час роботи програми.

Під час розв'язування завдань учень повинен скористатися теоретичними знаннями математичних основ інформатики, а потім реалізувати або модифікувати відомі йому алгоритми й вибрати прийнятні структури даних. Список основних алгоритмів, які застосовуються на олімпіадах з інформатики, можна знайти, зокрема, у книжці [2].

Узагальнюючи вищесказане, сформулюємо такі формальні вимоги до завдання з інформатики:

- у завданні повинна бути неявно сформульована, але матися на увазі єдино првильна математична модель;

- завдання повинно бути складене так, що для його розв'язку учень повинен володіти знаннями теоретичних і практичних основ інформатики;
- в умові завдання повинні бути чітко зазначені обмеження на всі вхідні дані, а також формати введення та виведення даних;
- повинні бути встановлені обмеження на час роботи програми, а також обмеження на кількість використаної пам'яті, виходячи з вимог до ефективності алгоритму й використання пам'яті;
- розв'язком завдання повинен бути текст програми. Правильність розв'язку визначається на наборі тестів, що складаються з наборів вхідних даних і еталонних відповідей до них (або програми, що перевіряє).

Окрім вищесказаних вимог до складання завдань з інформатики, необхідно враховувати такі педагогічні завдання:

- учень повинен продемонструвати творчий підхід, самостійно відбираючи ті знання й уміння, отримані в ході вивчення інформатики, які будуть потрібні йому для розв'язку завдання;
- завдання повинно мати кілька розв'язків, деякі з яких є неповними, що не задовольняють всі критерії. Такий підхід дозволяє забезпечити варіативність у рамках одного завдання: учень сам вибирає посильний рівень складності, а виходить, завдання може бути запропоноване групі учнів різного рівня підготовки й оцінене залежно від ступеня відповідності розв'язку формальним вимогам;
- у завданні бажані елементи міжпредметної інтеграції, тобто посилання на знання, отримані учнями на інших предметах і необхідні для розуміння умови й розв'язку завдання.

Наведемо деякі завдання олімпіади 2006–2007 навчального року та варіанти їх розв'язання.

Задача 1 (8 клас, задача 3). Задані 4 дійсних числа a, b, c, d. Скласти алгоритм, що визначає, чи можна побудувати чотирикутник зі сторонами, рівними цим числам, усі кути якого мають градусну міру строго більше 0 і менше 180 градусів. Постарайтеся, щоб у вашому алгоритмі виконувалося якнайменше операцій порівняння й арифметичних операцій.

Розв'язок. Спочатку розглянемо задачу про можливість побудови трикутника за заданими сторонами a, b, c. Відоме наступне твердження.

Твердження. Трикутник (невироджений) можна побудувати тоді й тільки тоді, коли всі сторони додатні та задовольняють нерівності трикутника — сума кожних двох сторін більше третьої.

Доведення. (Необхідність). Нехай трикутник зі сторонами a, b, c може бути побудований. Побудуємо його й позначимо його вершини A, B, C. Доведемо, наприклад, що $a + b > c$. Оскільки трикутник невироджений, то C не лежить на прямій AB. Опустимо перпендикуляр CD на цю пряму (рис. 1).

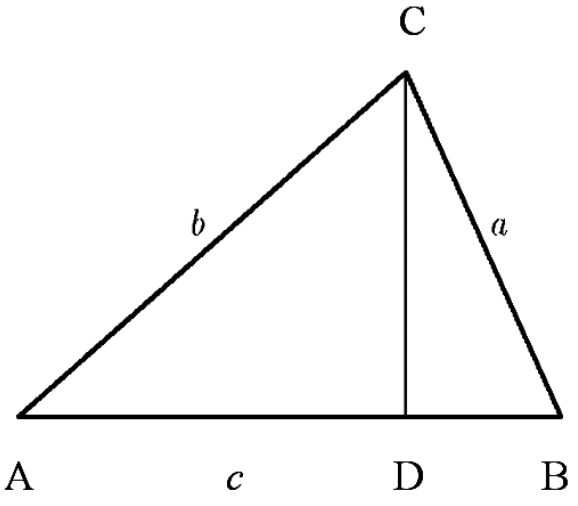

Рис. 1

Якщо обидва кути при основі трикутника ($\angle BAC$ і $\angle ABC$) гострі (або один з них прямий), то точка D буде належати відрізку AB і, тому $AB = AD + DB$. У випадку, коли один з кутів тупий, точка D потрапить на продовження відрізка AB і тому $AB < AD + DB$. У кожному випадку $AB \le AD + BD$. Оскільки похила завжди більша своєї проекції, маємо $AD < AC$, $BD < BC$. Звідси випливає, що $c = AB < AC + BC = a + b$. Аналогічно, доводяться нерівності й для інших пар сторін.

Достатність. Нехай задані три додатні сторони a, b, c, що задовольняють нерівності трикутника. Доведемо, що існує трикутник із заданими сторонами. Доведення буде конструктивним, тобто ми вкажемо спосіб побудови такого трикутника.

Відкладаємо довільним чином відрізок $AB = c$. З точки A як центра проведемо коло радіуса b, а з точки B — коло радіуса a (рис. 2).

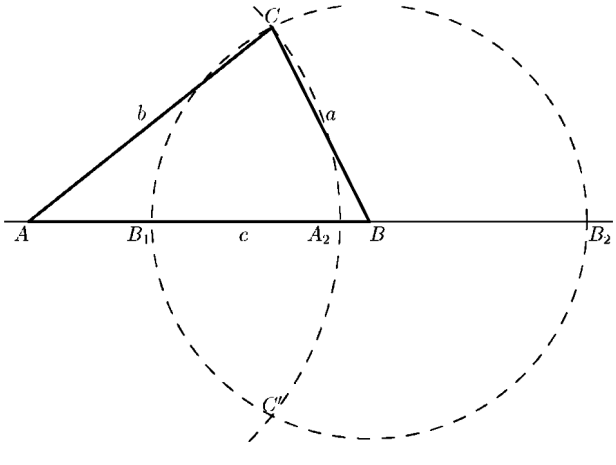

Рис. 2

Позначимо A_1, A_2 — точки перетину першого з прямою AB, а B_1, B_2 — другого кола. Тоді $AB_1 = |AB - BB_1| = |a - c|$ (знак модуля взятий, тому що при $a > c$ вираз може бути від'ємним, у цьому випадку точка B_1 розташована ліворуч від A). З нерівностей трикутника $c < a + b$ і $a < b + c$ випливає, що $c - a < b$ і $a - c < b$, а тому $AB_1 = |a - c| < b$, тобто відстань від B_1 до центра першого кола менше його радіуса, а звідси, ця точка лежить всередині кола. З іншого боку, $AB_2 = AB + BB_2$

$= c + a > b$, і тому, точка B_2 лежить поза кругом. Оскільки коло — це неперервна крива, друге коло повинно перетнути перше, а в силу замкненості точок перетину буде дві, і вони будуть лежати в різних півплощинах відносно прямої AB. Позначимо точки перетину цих кіл C і C'. Неважко бачити, що $\triangle ABC$ (так само як і $\triangle ABC'$) має сторони рівні a, b, c. Дійсно, $AB = c$ за побудовою, $AC = b$ як радіус першого кола, $BC = a$ як радіус другого кола. Тим самим твердження доведено.

Помітимо, що при фіксованому виборі місця розташування відрізка $AB = c$, вибору від якого кінця який відрізок (a або b) відкладати, та півплощини, у якій буде розташовуватися третя точка, трикутник будується однозначно.

```
алг трикутник (дійс a, b, c, літ s)
      арг a, b, c
      рез s
поч
      якщо a>0 і b>0 і c>0 і a+b>c і a+c>b і b+c>a то
            s := «можна побудувати»
      інакше
            s := «не можна побудувати»
      все
кін
```

У цьому алгоритмі виконується 6 операцій порівняння й 3 операції додавання. Помітимо, що перевірка додатності сторін є зайвою. Дійсно, нехай виконуються нерівності $a + b > c$ і $a + c > b$. Тоді ми можемо їх скласти, одержимо $2a + b + c > c + b$. Потім віднімемо від обох частин $b + c$ і розділимо на додатне число 2. Отримаємо $a > 0$. Аналогічно можна показати, що $b > 0$ і $c > 0$. Це дозволяє нам позбутися трьох операцій порівняння.

Якби ми знали більшу зі сторін, то достатньо було б перевірити, що сума двох менших сторін перевищує більшу, тобто обійтися всього однією операцією порівняння й однією додавання. Можна реалізувати й такий алгоритм, однак нам буде потрібно кілька операцій порівняння для знаходження найбільшого числа.

```
алг трикутник (дійс a, b, c, літ s)
      арг a, b, c
      рез s
поч
      дійс max
      max := a
      якщо b > max то
            max := b
      все
      якщо c > max то
            max := c
      все
      якщо a + b + c > max + max то
            s := «можна побудувати»
      інакше
            s := «не можна побудувати»
      все
кін
```

Оскільки ми заздалегідь не знаємо, яка зі сторін найбільша, то ми просто додаємо її до обох частин нерівності:

сума двох менших сторін < більша сторона.

В отриманому алгоритмі використовуються все ті ж 3 операції порівняння й 3 додавання. Тобто він нічим не кращий за попередній (якщо з нього вилучити перевірку додатності). Однак такий підхід буде більш ефективним для багатокутників з більшою кількістю сторін.

Твердження. N-кутник можна побудувати тоді і лише тоді, коли всі сторони додатні й кожна сторона менша суми інших $N-1$ сторін.

Доведення. Скористаємося методом математичної індукції. Для $N=3$ твердження вже доведене. Нехай твердження виконане для всіх $N<k$, доведемо, що із цього випливає виконання його для $N=k$.

Необхідність. Нехай побудовано k-кутник зі сторонами $a_1, a_2, \ldots, a_k$. З'єднаємо вершини A_{k-1} і A_1 діагоналлю x. За припущенням індукції для сторін (k−1)-кутник $A_1A_2\ldots A_{k-1}$ виконується нерівність $x<a_1+a_2+\ldots+a_{k-2}$. А з трикутника $A_1A_{k-1}A_k$ маємо $a_k<x+a_{k-1}$ або (з урахуванням попередньої нерівності): $a_k<a_1+a_2+\ldots+a_{k-1}$. Аналогічно, доводиться нерівність і для інших сторін (рис. 3).

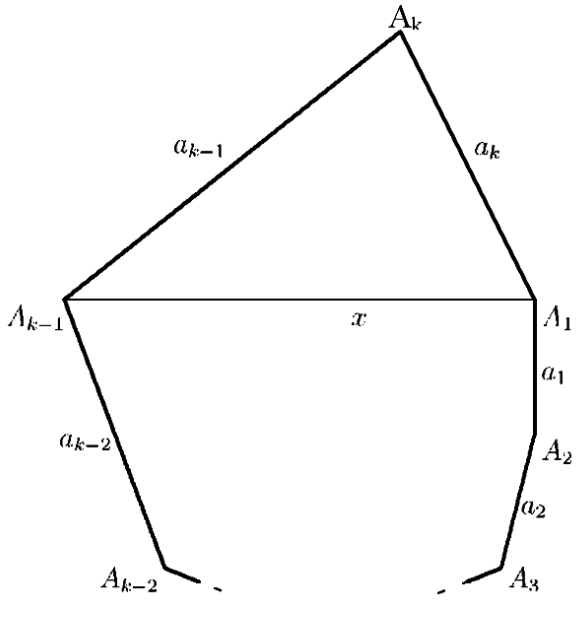

Рис. 3

Достатність. Нехай у нас є k сторін $a_1, a_2, \ldots, a_k$. Для визначеності будемо вважати, що a_k — найбільша сторона. Тоді побудуємо трикутник зі сторонами a_{k-1}, a_k, x, і $(k-1)$-кутник зі сторонами $a_1, a_2, \ldots, a_{k-2}$, x, де x — деяке дійсне число, яке ми далі визначимо таким чином, щоб відповідні побудови були можливі. Якщо сполучити сторони x отриманого трикутника й $(k-1)$-кутника так, щоб вони були розташовані в різних півплощинах відносно прямої, що проходить через сторону x, то одержимо k-кутник з потрібними сторонами. Тепер вкажемо, як слід вибирати

значення x. Оскільки всі сторони a_i додатні, то для того щоб існував трикутник, необхідно і достатньо, щоб x задовольняло нерівність

$$a_k - a_{k-1} < x < a_k + a_{k-1}. \quad (1)$$

Для того ж, щоб можна було побудувати (k−1)-кутник, за припущенням індукції необхідно і достатньо, щоб

$$x>0,$$
$$a_1+a_2+ \dots +a_{k-2}>x,$$
$$x+(a_1+a_2+ \dots +a_{k-2}-l)>l,$$

де l — найбільша серед сторін $a_1, a_2,\dots, a_{k-2}$. Ці нерівності відносно x можна записати у вигляді

$$\max\{0, 2l-(a_1+a_2+ \dots +a_{k-2})\}<x<a_1+a_2+ \dots +a_{k-2}. \quad (2)$$

У нерівності (2) ліва границя менша за l, а права — більша. Тому множина його розв'язків непорожня. У силу умови твердження для (k−1)-кутника $a_1+a_2+\dots+a_{k-2} > a_k-a_{k-1}$. Тому права границя нерівності (2) більша лівої границі нерівності (1), і тому загальний розв'язок нерівностей (1) та (2) існує і може бути записаний як

$$\max\{2l-(a_1+a_2+ \dots +a_{k-2}),\ a_k-a_{k-1}\}<x<\min\{a_1+a_2+ \dots +a_{k-2},\ a_k+a_{k-1}\}.$$

За x можна взяти, наприклад, середину цього відрізка, але чим ближче x буде до правої границі цієї нерівності, тим менше шансів, що буде побудований неопуклий або вироджений k-кутник. Твердження повністю доведене.

Слід зазначити, що для багатокутника з числом сторін більше трьох, умова додатності сторін не випливає з нерівності, визначеної твердженням. Наприклад, сума будь-яких трьох чисел з −1, 3, 4, 5 більше четвертого, однак серед них є від'ємне число −1. Тому в алгоритмі треба шукати не тільки максимальне значення, але й мінімальне, перевіряючи його знак. Для пошуку одночасно мінімального й максимального значення, можна використати попарні порівняння двох чисел, порівнюючи менше з них з поточним мінімальним, а більше з поточним максимальним. Це дає виграш за часом 25% порівняно із знаходженням мінімуму й максимуму окремо один від одного.

```
алг чотирикутник (дійс a, b, c, d, літ s)
    арг a, b, c, d
    рез s
поч
    дійс min, max
    якщо a < b то
      min := a; max := b
    інакше
      min := b; max := a
    все
    якщо c < d то
      якщо c < min то min := c
      якщо d > max то max := d
    інакше
      якщо d < min то min := d
      якщо c > max то max := c
    все
    якщо min > 0 і a + b + c + d > max + max то
      s := «можна побудувати»
    інакше
      s := «не можна побудувати»
    все
кін
```

Наведений алгоритм виконує 6 операцій порівняння й 4 додавання (пряма перевірка умов твердження зажадала б 8 операцій порівняння й 8 додавання).

Задача 2 (9 клас, задача 3). На одній вулиці стоять один напроти одного два магазини. На кожному з них є своя вивіска, на якій написана назва відповідного магазину. Усе було добре, доки одного разу вночі місцевий хуліган Стьопа не розламав ці вивіски точно навпіл і поміняв місцями другі половини. Відомо, що в кожній з вивісок було парне число букв. Вам задано два рядки S_1 і S_2, що містять тексти вивісок, що вийшли. Складіть алгоритм, що визначає, чи могли такі вивіски вийти в результаті дій Стьопи, і якщо так, то знайдіть всі можливі варіанти назв магазинів.

Розв'язок. Нехай R_1 — назва першого магазину, R_2 — назва другого магазину, $2k_i$ — кількість символів у R_i. Тоді повинні виконуватися такі рівності:

$$R_1[1{:}k_1] + R_2[(k_2+1){:}2k_2] = S_1;$$
$$R_2[1{:}k_2] + R_1[(k_1+1){:}2k_1] = S_2.$$

Звідси видно, що кожен з рядків S_1 і S_2 повинен мати ту саму довжину k_1+k_2. Оскільки відповідні підрядки в записаних рівностях не перетинаються, то рівність довжин S_1 і S_2 є й достатньою умовою можливості відновлення вивісок. Однак відновлюються вони неоднозначно. Взявши довільне значення k_1 з діапазону $0\le k_1\le$ довж(S_1) = довж(S_2), ми одержимо один варіант відновлення. Усі ці варіанти будуть різні, оскільки будуть виходити різні довжини вивісок.

Складемо алгоритм, що буде у змінній N повертати кількість варіантів відновлення, і зберігати результати відновлення в таблицях літерних величин $R_1[1{:}N]$ і $R_2[1{:}N]$:

```
алг вивіски (літ S1, S2, ціл N, літ таб R1[1:N], R2[1:N])
    арг S1, S2
    рез N, R1, R2
поч
    ціл k1, l
    N := 0
    l := довж (S1)
    якщо l=довж(S2) то
      для k1 від 0 до l
      пц
            N := N + 1
            R1[N] := S1[1:k1] + S2[l-k1+1:l]
            R2[N] := S2[1:l-k1] + S2[k1+ 1:l]
      кц
    все
кін
```

Задача 3 (10 клас, задача 3). Деякі сучасні композитори люблять використовувати у своїх творах мотиви, що вже стали популярними. Щоб припинити появу на естраді таких псевдонових пісень, при Міністерстві Культури був створений Комітет по боротьбі з пла-

гіатом і рімейками в сучасній популярній музиці. Саме сьогодні в Комітет надійшов новий твір Кіркора Філіппова, у якому, як підозрює голова Комітету, зустрічається мотив з однієї відомої пісні Орбіни Крістікайте. Однак, оскільки музичні твори й мотиви можуть бути достатньо великими, без комп'ютера перевірити пісню на плагіат досить важко.

Завдання. Напишіть програму MELODY, що перевіряє, чи зустрічається у заданому творі як фрагмент певний мотив (можливо транспонований).

Вхідні дані. У першому рядку текстового файлу MELODY.DAT записана кількість тестів. Перший рядок кожного тесту містить довжину (кількість нот) шуканого мотиву M ($1 \le M \le 1000$). У наступному рядку записана послідовність M нот мотиву. Кожна нота є цілим числом з діапазону — 100..100, і визначає висоту ноти (наскільки півтонів вона вища, ніж нота «до першої октави»). У третьому рядку задана довжина музичного твору N ($0 \le N \le 1000000$). І, нарешті, останній (четвертий) рядок тесту є послідовністю нот твору.

У 50% тестових файлів значення довжини мотиву й твору будуть задовольняти обмеженням $1 \le M \le 50$, $1 \le N \le 10000$, у 75% — $1 \le M \le 50$, $1 \le N \le 1000000$.

Вихідні дані. У вихідний файл MELODY.SOL потрібно вивести для кожного тесту число 0, якщо в творі не зустрічається шуканий мотив (навіть транспонований), або число 1, якщо твір містить мотив.

«Транспонований» означає, що деяка додатна або від'ємна стала додається до кожної ноти мотиву. Наприклад, для мотиву (0,5,10,6) транспонованим є (5,10,15,11), а також (–2,3,8,4).

Приклад вхідних і вихідних даних

MELODY.DAT	*MELODY.SOL*
3 5 1 2 3 4 5 10 1 -1 1 2 3 4 5 0 1 2 3 4 5 7 7 -1 -2 -1 1 4 5 8 2 1 6 4 4 3 2 1	1 1 0

Розв'язок. Нехай мелодія записана в масиві Melody[1..N], а мотив — Motive[1..M]. Очевидний розв'язок задачі полягає в тому, що для кожної позиції в мелодії перевіряється збіг M нот мелодії з M-нотним мотивом, можливо транспонованим:

```
s:='не знайдений';
for i:=0 to N–M do
begin
  s:='знайдений';
  shift:=Melody[i+1]–Motive[1];
  for j:=2 to M do
    if Melody[i+j]<>Motive[j]+shift then
    begin
      s:='не знайдений';
      break
    end
  if s='знайдений' then
    break
end;
```

Тут визначається змінна shift, яка означає єдиний такий можливий зсув мелодії вгору або вниз, що збігаються перша нота мотиву й перша нота розглянутого фрагмента мелодії. Ясно, що наведений алгоритм буде мати складність $O(M(N-M))$.

Однак, можна уникнути численних додавань, які виконуються у внутрішньому циклі. Для цього досить помітити, що в мотиві й транспонованому мотиві повинні збігатися інтервали (різниці) між послідовними нотами. Тоді, якщо ми перетворимо мелодію й мотив до різницевого вигляду, задача зведеться до перевірки точного входження мотиву в мелодію. Наприклад, для другого тесту з прикладу вхідних даних мотив (4, 5, 7) перетвориться до вигляду (1, 2), а мелодія (–1, –2, –1, 1, 4, 5, 8) — до (–1, 1, 2, 3, 1, 3). Звідси видно, що мотив входить у мелодію, починаючи із другої позиції:

```
for i:=1 to N–1 do
  MelodyDiff[i]:=Melody[i+1]–Melody[i];
for j:=1 to M–1 do
  MotiveDiff[j]:=Motive[j+1]–Motive[j];
  s:='не знайдений';
for i:=0 to N–M do
begin
  s:='знайдений';
  for j:=1 to M–1 do
    if MelodyDiff[i+j]<>MotiveDiff[j] then
    begin
      s:='не знайдений';
      break
    end
  if s='знайдений' then
    break
end;
```

Тут є один важливий момент. У той час, як ноти мелодії й мотиву лежать у межах від –100 до 100 і для них можна використовувати тип shortint, різниці ж можуть приймати значення від –200 до 200 і тому для них буде потрібен більший тип. У цьому алгоритмі можна обійтися без вихідних масивів Melody і Motive, а масиви різниць одержувати відразу під час читання даних.

Якщо писати програму на Turbo Pascal, то виникає проблема, як розмістити у пам'яті масив різниць мелодії MelodyDiff. При обмеженні по пам'яті 64 Kb, що накладає на сегмент даних Turbo Pascal, найбільше, що ми зможемо розмістити в ньому — порядку 30000 елементів. Однак можна помітити, що нам не потрібний відразу весь масив MelodyDiff, а лише M його елементів, і коли ми переходимо після перевірки i-ої позиції до $(i+1)$-позиції, то i-а нота нам більше не знадобиться. Таким чином, у разі чергової перевірки ми можемо зсунути M елементів вліво, а на позицію, що звільнилася, прочитати чергову ноту.

Тепер наш алгоритм буде працювати для будь-яких N, однак його самою критичною за швидкістю частиною, буде зсув елементів масиву MelodyDiff,

що не має безпосереднього відношення до поставленої задачі. Цієї частини можна уникнути, якщо ми будемо не зсувати елементи масиву, а тільки порівнювати із зсувом. Основний цикл програми буде таким:

```
...
curpos:=M–1;
for i:=0 to N–M do
begin
  read(fIn, curnote);
  MelodyDiff[curpos]:=integer(curnote)–prevnote;
  prevnote:=curnote;
  found:=true; j2:=M–curpos;
  for j:=1 to curpos do
  begin
    if MotiveDiff[j2]<>MelodyDiff[j] then
    begin
      found:=false;
      break
    end;
    inc(j2);
  end;
  if found then
  begin
    j2:=1;
    for j:=curpos+1 to M–1 do
    begin
      if MotiveDiff[j2]<>MelodyDiff[j] then
      begin
        found:=false;
        break
      end;
      inc(j2);
    end;
  end;
  if found then
     break;
  if curpos=M–1 then
     curpos:=1
  else
     inc(curpos);
end;
readln(fIn);
...
```

Отримана програма буде працювати досить добре для випадкових тестів, але якщо частина мотиву буде зустрічатися досить часто в мелодії (наприклад, для мотиву мотив (0, 0,..., 0, 1), такою буде мелодія, що вся складається з однакових нот), буде потрібно порядку O(MN) операцій порівняння, що неприйнятно для заданих обмежень на M і N. У такому випадку краще застосувати алгоритм Кнута-Мориса-Пратта (див. [3], параграф 34.4):

```
...
pi[1]:=0; q:=0;
for j:=2 to M–1 do
begin
   while (q>0)and(MotiveDiff[q+1]<>MotiveDiff[j])
    do q:=pi[q];
   if MotiveDiff[q+1]=MotiveDiff[j] then
    inc(q);
    pi[j]:=q;
end;
readln(fIn, N);
if N=0 then
begin
  writeln(fOut, '0');
  continue
end;
read(fIn, prevnote);
found:=(M=1); q:=0;
for i:=1 to N–1 do
begin
  read(fIn, curnote);
  MelodyDiffi:=integer(curnote)-prevnote;
  prevnote:=curnote;
   while (q>0) and (MotiveDiff[q+1]<>MelodyDiffi)
     do q:=pi[q];
  if (MotiveDiff[q+1]=MelodyDiffi) then
     inc(q);
  if (q>=M–1) then
  begin
     found:=true;
     break;
  end;
end;
readln(fIn);
...
```

Обчислювальна складність цього алгоритму O(M+N).

Наведені вище задачі є типовими олімпіадними задачами. Учасників олімпіад учать свідомо або підсвідомо будувати свої міркування від простого до складного.

Ці задачі можна використати й у навчанні інформатики. Залежно від рівня підготовки учнів можуть варіюватися час на виконання завдання, даватися підказки, організовуватися групова робота над задачами й т. д.

Література

1. *Андреева Е. В.* Принципы проверки учебных и олимпиадных задач по информатике // Информатика. — 2001. — №34.
2. *Кирюхин В.М.* Информатика. Всероссийские олимпиады. — М.: Просвещение, 2008.
3. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. — М.: 2005. МЦНМО, 2001. — 960 с., 263 ил.
4. *Матюхин В.А.* Автоматизация проверки решений школьников при изучении программирования // Вестник Вятского государственного педагогического университета. Вып. «Информатика». — Киров, 2002.
5. http://www/csin.ru/info/informatika.
6. http://homepages.compuserve.de/chasluebeck/index.htm.
7. *Гуржій А.М., Бондаренко В.В., Співаковський О.В., Ягіяєв Ш.І.* Всеукраїнські та міжнародні олімпіади з інформатики в задачах та рішеннях: Посібник./ За редакцією А.М. Гуржія: Видання друге, доповнене і перероблене. — Херсон: Айлант, 2007. — 572 с. іл.
8. *Пасіхов Ю., Сімонов К., Кравець Г. та ін*. Всеукраїнські Інтернет-олімпіади з інформатики NetOI. — Вінниця: УНІВЕРСУМ-Вінниця, 2006. — 152 с.
9. *Порублев И.Н., Ставровский А.Б.* Алгоритмы и программы. Решение олимпиадных задач. — М.: ООО «И.Д. Вильямс», 2007. — 480 с.: ил.