Олександр Рудик

Програмування пошуку виграшної стратегії, логічних умовиводів і розрахунків з багатоцифровими числами

Київ 2001

УДК 372.800.2: 371.26.035.461
ББК 74.263.2
0–54

Рекомендовано Науково-методичною радою
КМіУВ ім. Б.Грінченка
(протокол No 2 від 12 березня 2001 року).

Рецензенти:
Ю.С.Рамський, професор, кандидат фізико-математичних наук;
Л.А.Федорів, вчитель-методист.

О.Рудик. Програмування пошуку виграшної стратегії, логічних умовиводів і розрахунків з багатоцифровими числами. — К: КМіУВ ім. Б.Грінченка, 2001. — 38 с.

Посібник складається з Передмови і трьох розділів — з детальним аналізом задач, які пропонувалися відповідно на I і II етапах олімпіади з інформатики в м. Києві в 1999 році і на III етапі у 2001 році. Розділи мають аналогічну структуру: рекомендації щодо умов проведення олімпіади, завдання для учнів, вказівки для перевірки для орєанізаторів олімпіади (опис тестів), авторське розв'язання з детальним аналізом алгоритмів і прикладами програм мовою програмування Turbo Pascal 7.0.

Для читача відкрито *весь* технологічний ланцюг створення олімпіадного завдання на прикладі задач із застосуванням теорії ігор, міркувань за законами формальної логіки й подання чисел масивами їхніх цифр ("довга арифметика").

Передмова

Олімпіада з основ інформатики та обчислювальної — найлегша й одночасно найважча серед інших учнівських одімпіад. Таке парадоксальне на перший погляд твердження зовсім не спроба заінтриєувати читача, а щира правда.

Справді, для успішного виступу на олімпіаді потрібно знати невелику кількість математичних моделей та мати навички швидкої програмної реалізації відповідних алгоритмів. Перерахуємо найголовніші:

- знаходження періоду підстановки, перебір всіх перестановок, розташувань чи комбінацій;
- оптимізація перебору, наприклад, методом динамічного програмування;
- невідома наперед кількість вкладених циклів без запису кожного такого циклу окремою єрупою операторів;
- використання рекурентних співвідношень і рекурсії;
- аналіз графів;
- використання тверджень теорії цілих чисел (відношення подільності, системи числення);
- аналіз "від кінця" (антагоністичних) ігор з повною інформацією;
- формально-логічні умовиводи;
- розпізнавання і створення образів примітивної графіки;
- використання методу координат і властивостей геометричних тіл.

В 1999 році на I–II етапах і в 2001 році на III етапі олімпіади з інформатики у м. Києві учням пропонувалися задачі, у яких вирішальну роль мали:

- створення алгоритму числових розрахунків з поданням числа масивом його цифр;
- реалізація гри й пошук виграшної стратегії;
- розв'язування логічних задач.

Мета посібника — ознайомити широке коло читачів (учнів, вчителів, студентів педагогічних вищих навчальних закладів) з умовами завдань, тестами для перевірки розв'язання та авторським баченням оптимального розв'язання таких задач. До трьох задач подано по два розв'язання.

Рекомендований спосіб роботи з посібником.

1. Ознайомтеся з умовами завдань певного етапу, упорядкуйте їх за зростанням складності задач. і саме в цій послідовності розв'язуйте задачі. Виключення може бути лише для випадку, коли Ви напевно знаєте розв'язання задачі, за розв'язання якої присуджується найбільша кількість балів.

2. Наскільки це можливо, чітко опишіть математич ну модель і алгоритм розв'язання задачі. Продумайте власну систему тестування. Для початківців бажано все це зафіксувати на папері. І чим менший досвід *успішного* розв'язування задач на ЕОМ, тим більша потреба у попередньому записі програми на папері й аналізу її дії. Особливо, якщо доступ до ПК утруднено. Пам'ятайте: алгоритм має бути зрозумілим не лише Вашим викладачам, але Вашим одноліткам. Лише тоді можна сподіватися, що він буде зрозумілим і Вам.

3. Усвідомивши модель і алгоритм розв'язання, створюйте програму, яка має задовольняти технічні умови задачі, успішно проходити тести, подані в умові задачі, та всі Ваші тести.

4. Проаналізуйте, які тести пропонуються автором, і в чому полягає істотна відмінність з Вашою системою тестування. Якщо такі відмінності є, подумайте, у чому неадекватність математичної моделі й неефективність алгоритму — Ваших чи авторських. Врахуйте: опрацювання авторських тестів авторською програмою триває секунди або долі секунди на IBM486 60MHz. Якщо знаєте як, удоскональте власну програму так, щоб вона пройшла всі тести.

5. Ознайомтеся з ідеєю авторського розв'язання, поданою перед програмою, і порівняйте з власними міркуваннями. Якщо знаєте як, удоскональте власну програму так, щоб вона пройшла всі авторські тести.

6. Ознайомтеся з поданою автором програмою мовою Turbo Pascal 7.0. алгоритми істотно не використовують особливостей саме цієї алгоритмічної мови, тому програму без труднощів можна перекласти будь-якою іншою мовою програмування, в тому числі, діалектами Basic'а. Зверніть увагу на прийоми програмування, до яких не змогли (не встигли, не захотіли) додуматися самі. Перефразуючи відомий вислів, можна сказати: чужа програма — пітьма. Але у цій пітьмі Ви обов'язково маєте знайти і зрозуміти алгоритм. Цього можна не робити, якщо Ви вже виробили власний стиль оформлення програм. Такий стиль, який допоміг Вам *успішно* подолати всі попередні етапи. є щонайменше 3 стадії вивчення літератури:

   - такі думки є;

- такі думки є, і вони правильні чи неправильні;
- такі думки є, вони правильні чи неправильні і зрозумілі переваги (недоліки) авторського викладу.

Пройдіть всі ці стадії.

7. Проведіть тренування в умовах проведення олімпіади (ніяких попередніх записів чи частин програм на магнітних носіях, 5 годин роботи на ПК), щоб перевірити навички використання відомих Вам алгоритмів.

8. Переходьте до завдань наступного етапу.

Найскладніші задачі було включено до умов завдвнь відбірково-тренувальних зборів команди міста Києва. Досвід проведення таких зборів засвідчив:

- реалістичність відтворення розв'язання за 1.5 години *підготовленим* учасником олімпіади (навіть 9-класником!);
- доцільність цих тренувальних завдань для підготовки учасника олімпіади на найвищому рівні.

1 I етап

1.1 Умови проведення

Олімпіаду рекомендуємо провести в один практичний тур тривалістю 5 (п'ять) астрономічних годин. Коректність даних перевіряти не потрібно.

1.2 Орієнтовні завдання

1. Система числення, 8 балів. Перший та другий рядки вхідного файлу NUMBER.DAT містять відповідно десяткові записи натуральних чисел a і b — не більше, ніж 80 цифр у кожному. Наприклад,

```
123
21
```

Створіть програму NUMBER.* , яка запише у файл NUMBER.RES запис числа a у системі числення з основою b. Всі цифри цієї системи числення записувати наступним чином: або цифра десяткової системи числення, або записаний у квадратних дужках десятковий запис числа, що має не менше 2-х цифр. Для поданого прикладу вхідного файлу файл NUMBER.RES має містити запис

```
5[18]
```

— див. рівність $123 = 5 \cdot 21 + 18$.

2. Спортивний прогноз, 10 балів. Перед фінальним забігом mn бігунів було висловлено m прогнозів щодо послідовності фінішування. У кожному прогнозі правильно вгадано лише n місць, причому кожне місце правильно вгадано лише в одному прогнозі. Перший рядок вхідного файлу PROGNOS.DAT містить у вказаному порядку натуральні числа m і n, де $mn<27$. Наступні m рядків цього самого файлу містять прогнози, в яких кожного учасника фінального забігу позначено однією з літер кирилиці чи латиниці. Наприклад,

```
2 2
ACBD
DBCA .
```

Рядки файлу PROGNOS.RES мають містити відповідь на питання: у якому порядку фінішували спортсмени (у кожному рядку — по одному можливому варіанту, причому у різних рядках — різні варіанти). Для поданого прикладу вхідного файлу PROGNOS.RES містить 2 рядки:

```
ABCD
DCBA .
```

3. Гра–1, 16 балів. На площині дано 4 *різні* точки $A_1(x_1; y_1)$, $A_2(x_2; y_2)$, $A_3(x_3; y_3)$, $A_4(x_4; y_4)$. Двоє гравців по черзі сполучають ці точки відрізками так, що жодна внутрішня точка проведеного відрізка не може належати іншому проведеному відрізку (наприклад, бути його кінцем). Програє той гравець, який не зможе зробити наступний хід. єдиний рядок файлу GAME1.DAT містить у вказаному порядку одноцифрові натуральні числа x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4. Створіть програму GAME1.*, яка:

- у файл GAME1.RES запише повідомлення "Виграє 1-ий гравець" або "Виграє 2-ий гравець", якщо відповідно 1-ий або 2-ий гравець може забезпечити собі виграш для даного розташування точок на площині, причому далі повідомлення або продовжується словами "за будь-яких обставин", або містить рекомендації щодо виграшної стратегії;

- зобразить на екрані монітора розташування даних точок координатної площини: на чорному тлі білі точки (кружечки), єдине мірило для осей абсцис і ординат, координатні осі не відображати, позначення кожної точки $A_j(x_j; y_j)$ числом j (в межах від 1 до 4) відповідно;

- у правому верхньому куті екрану дає повідомлення: "n-ий гравець! Задайте номери точок, які хочете сполучити відрізком" (n=1 або n=2).

Сполучення точок A_j і A_k гравцем задається за допомогою клавіатури набором чисел j і k через прогалину. Кожен хід першого гравця відображається рожевим

кольором, а другого — світло-зеленим. У разі неправильного задання ходу (сполучення точки з самою собою чи порушення умови стосовно відсутності спільних внутрішніх точок відрізків) гравцю надається можливість виправити свою помилку. Після останнього ходу на екрані з'являється повідомлення: "n-ий гравець! Ви виграли. Останній хід ..." і вказано номери точок, сполучених останнім ходом. Виконання програми припиняється після натискання клавіші Enter.

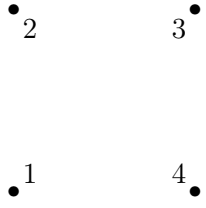
Наприклад, якщо файл GAME1.DAT містить запис

```
1 1  1 2  2 2  2 1
```

то файл GAME1.RES містить повідомлення

```
Виграє 1-ий гравець за будь-яких обставин
```

а безпосередньо перед грою на екрані монітора — зображення 4-х вершин квадрата.



Гра–2, (44 бали) (тренувальна вправа до II етапу).
На площині дано 5 *різних* точок $A_1(x_1;y_1)$, $A_2(x_2;y_2)$, $A_3(x_3;y_3)$, $A_4(x_4;y_4)$, $A_5(x_5;y_5)$. Єдиний рядок файлу GAME2.DAT містить у вказаному порядку одноцифрові натуральні числа x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5. Створіть програму GAME2.*, яка задовольняє всі вимоги задачі "Гра–1".

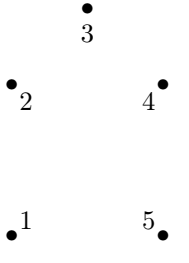
Наприклад, якщо файл GAME2.DAT містить запис

```
1 1  1 3  2 4  3 3  3 1
```

то файл GAME2.RES містить запис

```
Виграє 1-ий гравець за будь-яких обставин,
```

а перед запрошенням до гри на екрані монітора — зображення вершин опуклого 5-кутника.



1.3 Вказівки до перевірки

1. Система числення. За кожен тест — по 2 бали. Перевіряють:

- знання систем числення (тести 1–4);
- вміння подавати велике натуральне число масивом його цифр (тести 3–4);
- вміння оперувати з системами числення, основа яких більша за 10 (тести 2, 4).

1.1. Якщо файл NUMBER.DAT має вигляд

```
56789
7
```

то файл NUMBER.RES містить запис

```
324365
```

1.2. Якщо файл NUMBER.DAT має вигляд

```
12345
67
```

то файл NUMBER.RES містить запис

```
2[50][17]
```

1.3. Якщо файл NUMBER.DAT має вигляд

```
12345678901234567890123456789012345678901234567890
7
```

то файл NUMBER.RES містить запис

```
1435216644024312530032563366013546542565351446 3
```

1.4. Якщо файл NUMBER.DAT має вигляд

```
1234567890......1234567890
9876543210..9876543210
```

де перший рядок містить 8-кратне повторення послідовності цифр 1234567890, а другий — 4-кратне повторення послідовності цифр 9876543210, то файл NUMBER.RES містить запис

```
[124999998860937500014238281249822021 4843]
[886170186088617018608861701860886170186 0]
```

2. Спортивний прогноз. За кожен тест — по 2 бали. Перевіряють:

- чи враховано випадок очевидного розв'язання (тест 1);
- чи здійснено перебір всіх можливих варіантів з відкиданням тих варіантів, які суперечать умові задачі: у кожному проєнозі правильно вгадано лише n місць, причому кожне місце правильно вгадано лише в одному прогнозі (тести 2–5);
- чи записано *всі* розв'язки (тести 2, 5).

2.1. Якщо файл PROGNOSE.DAT має вигляд

```
1 26
ABCFEDGHILKJMNORQPSTUXWVYZ
```

то файл PROGNOSE.RES має вигляд

```
ABCFEDGHILKJMNORQPSTUXWVYZ
```

2.2. Якщо файл PROGNOSE.DAT має вигляд

```
3 2
АБВГДЄ
ВЄДАГБ
ДГЄБАД
```

то файл PROGNOSE.RES має вигляд

```
АЄВБГД
ДЄВГАБ
```

2.3. Якщо файл PROGNOSE.DAT має вигляд

```
3 3
AEDFHIGBC
DBGAEFIHC
EACGFDHBI
```

то файл PROGNOSE.RES має вигляд

```
ABCFEDGHI
```

2.4. Якщо файл PROGNOSE.DAT має вигляд

```
13 2
ZYWVUTSRQPONMLKJIHGFEDCBAX
YZXWUTSRQPONMLKJIHGFEDCBAV
YXWZVUSRQPONMLKJIHGFEDCBAT
YXWVUZTSQPONMLKJIHGFEDCBAR
YXWVUTSZRQONMLKJIHGFEDCBAP
YXWVUTSRQZPOMLKJIHGFEDCBAN
YXWVUTSRQPOZNMKJIHGFEDCBAL
YXWVUTSRQPONMZLKIHGFEDCBAJ
YXWVUTSRQPONMLKZJIGFEDCBAH
YXWVUTSRQPONMLKJIZHGEDCBAF
YXWVUTSRQPONMLKJIHGZFECBAD
YXWVUTSRQPONMLKJIHGFEZDCAB
YXWVUTSRQPONMLKJIHGFEDCZBA
```

то файл PROGNOSE.RES має вигляд

```
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

2.5. Якщо файл PROGNOSE.DAT має вигляд

```
2 13
ZYXWVUTSRQPONLKJIHGFEDCBAM
YXWVUTSRQPONZMLKJIHGFEDCBA
```

то файл PROGNOSE.RES має вигляд

```
ZYXWVUTSRQPONMLKJIHGFEDCBA
YXWVUTSRQPONZLKJIHGFEDCBAM
```

3. Гра-1. За кожен тест — по 4 бали, з них:

- за правильне зображення розташування точок на екрані монітора, запрошення робити наступний хід і зафарбування вказаних відрізків відповідним кольором — 1 бал;
- за дотримання правил гри (стосовно спільних точок відрізків), надання можливості гравцям виправити зроблені помилки і повідомлення переможцю — 2 бали;
- за правильну відповідь про виграшну стратегію (у файлі GAME1.RES) — 1 бал.

Дотримання правил гри і графічне подання перевіряти за сценарієм, вказаним у тесті:

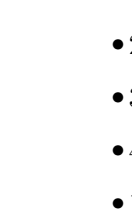
- у кожному рядку вказано хід I-го гравця і відповідь на нього II-го гравця (тут під ходом розуміється, насправді, послідовність ходів, серед яких правилам гри відповідає лише останній, і лише він повинен відображатися на екрані монітора);

- перемагає гравець, який робить останній хід, про що повинна сповістити виконувана програма.

3.1. Якщо файл GAME1.DAT має вигляд

```
1 1   1 4   1 3   1 2
```

то на екрані монітора на початку гри — зображення точок однієї вертикальної прямої.



Файл GAME1.RES при цьому має вигляд

```
Виграє 1-ий гравець,
якщо 1-им ходом з'єднає точки 1, 2.
```

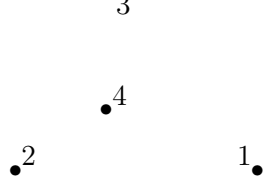
Припустимий також запис

```
Виграє 1-ий гравець,
якщо 1-им ходом з'єднає точки 3, 4.
```

Сценарій гри:
I: 1–1, 2–2, 3–3, 4–4, 5–5, 1–4; II: 4–1, 2–3;
I: 3–1, 2–4, 2–1, 3–4.
3.2. Якщо файл GAME1.DAT має вигляд

```
9 1   1 1   5 7   4 3
```

то на екрані монітора на початку гри — зображення точок, 4-та з яких є внутрішньою точкою трикутника, утвореного першими трьома.



Файл GAME1.RES при цьому має вигляд

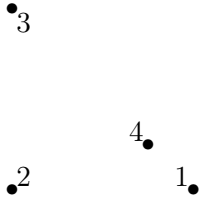
```
Виграє 2-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 1–2; II: 2–1, 2–3;
I: 3–2, 3–4; II: 4–3, 4–1;
I: 1–4, 3–1; II: 1–3, 4–2.

3.3. Якщо файл GAME1.DAT має вигляд

```
9 1   1 1   1 9   7 3
```

то на екрані монітора на початку гри — зображення точок, 4-та з яких є внутрішньою точкою відрізка, який з'єднує точки 1 і 3.



Файл GAME1.RES при цьому має вигляд

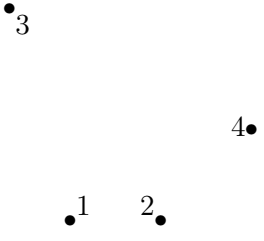
```
Виграє 1-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 1–3; II: 2–4, 2–1;
I: 3–2.

3.4. Якщо файл GAME1.DAT має вигляд

```
3 1   6 1   1 8   9 4
```

то на екрані монітора на початку гри — зображення вершин опуклого 4-кутника.



Файл GAME1.RES при цьому має вигляд

```
Виграє 1-ий гравець за будь-яких обставин.
```

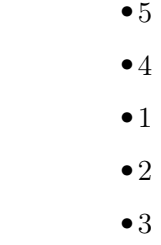
Сценарій гри:
I: 2–3; II: 1–4, 1–2;
I: 4–1, 2–1, 3–1; II: 4–2;
I: 3–4.

4. Гра–2 (див. загальні вказівки до гри–1 з 4-ма точками).

4.1. Якщо файл GAME2.DAT має вигляд

```
2 3   2 2   2 1   2 4   2 5
```

то на екрані монітора на початку гри маємо зображення точок однієї прямої.



Файл GAME2.RES при цьому має вигляд

```
Виграє 1-ий гравець,
якщо 1-им ходом з'єднає точки 3, 5.
```

Також можна першим ходом з'єднувати точки 2–4. Сценарій гри:
I: 1–1, 2–2, 3–3, 4–4, 5–5, 1–4; II: 5–3, 5–4;
I: 4–2, 3–4, 1–5, 2–5, 2–3; II: 1–3, 1–2.
4.2. Якщо файл GAME2.DAT має вигляд

```
7 3   1 1   9 1   7 7   7 5
```

то на екрані монітора на початку гри маємо: точки 1 і 5 є внутрішніми точками трикутника, утвореного точками 2, 3, 4, і лежать на вертикальній прямій, що проходить через точку 4.

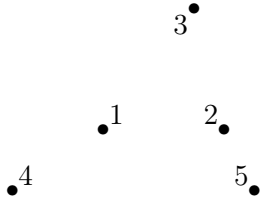
Файл GAME2.RES при цьому має вигляд

```
Виграє 1-ий гравець,
якщо 1-им ходом з'єднає точки 1, 5.
```

Також можна першим ходом з'єднувати точки 2–5 або 3–5. Сценарій гри:
I: 2–4; II: 4–1;
I: 2–5, 3–5, 1–2; II: 4–3;
I: 3–2; II: 3–1.
4.3. Якщо файл GAME2.DAT має вигляд

```
4 3   8 3   7 7   1 1   9 1
```

то на екрані монітора на початку гри маємо: точки 1 і 2 є внутрішніми точками трикутника, утвореного точками 3, 4, 5, і жодні три точки не належать одній прямій.



Файл GAME2.RES при цьому має вигляд

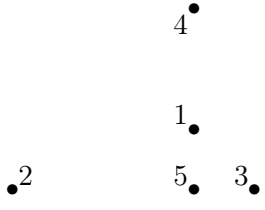
```
Виграє 1-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 2–4; II: 5–1, 2–5;
I: 2–4, 2–5, 4–5; II: 5–4, 1–2;
I: 2–1, 1–4; II: 3–4;
I: 3–1; II: 3–5;
I: 3–2.

4.4. Якщо файл GAME2.DAT має вигляд

```
7 3   1 1   9 1   7 7   7 1
```

то на екрані монітора на початку гри маємо: точка 1 є внутрішньою точкою трикутника, утвореного точками 2, 3, 4, точка 5 є внутрішньою точкою відрізка, який з'єднує точки 2–3, причому точки 1, 4 і 5 — на одній прямій.



Файл GAME2.RES при цьому має вигляд

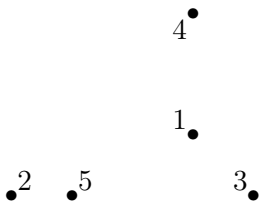
```
Виграє 1-ий гравець,
якщо 1-им ходом з'єднає точки 5, 4.
```

Сценарій гри:
I: 4–5; II: 1–2, 2–5;
I: 3–1, 3–5; II: 4–2;
I: 4–3.

4.5. Якщо файл GAME2.DAT має вигляд

```
7 3   1 1   9 1   7 7   3 1
```

то на екрані монітора на початку гри маємо: точка 1 є внутрішньою точкою трикутника, утвореного точками 2, 3, 4, точка 5 є внутрішньою точкою відрізка, який з'єднує точки 2–3, але точки 1, 4 і 5 не належать одній прямій.



Файл GAME2.RES при цьому має вигляд

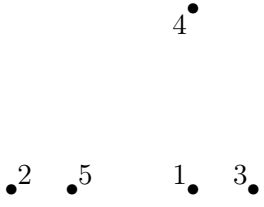
```
Виграє 2-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 4–5; II: 1–2, 2–5;
I: 3–1; II: 4–1;
I: 5–1; II: 5–3;
I: 4–2; II: 3–4.

4.6. Якщо файл GAME2.DAT має вигляд

```
7 1   1 1   9 1   7 7   3 1
```

то на екрані монітора на початку гри маємо: точки 1 і 5 є внутрішніми точками відрізка, який з'єднує точки 2, 3.



Файл GAME2.RES при цьому має вигляд

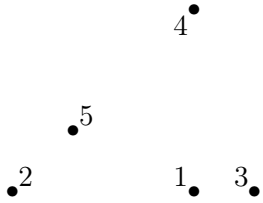
```
Виграє 1-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 2–3; II: 5–4, 1–4, 2–4;
I: 3–4.

4.7. Якщо файл GAME2.DAT має вигляд

```
7 1   1 1   9 1   7 7   3 4
```

то на екрані монітора на початку гри маємо: точки 1 і 5 є внутрішніми точками відрізків, які з'єднують точки 2–3 і 2–4 відповідно.



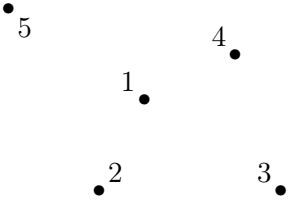
Файл GAME2.RES при цьому має вигляд

```
Виграє 1-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 2–4; II: 3–5, 3–2;
I: 1–4, 3–4.
4.8. Якщо файл GAME2.DAT має вигляд

```
4 3   3 1   7 1   6 4   1 5
```

то на екрані монітора на початку гри маємо: точки 2, 3, 4, 5 є вершинами опуклого 4-кутника, а точка 1 — середина відрізка, який сполучає точки 3–5.



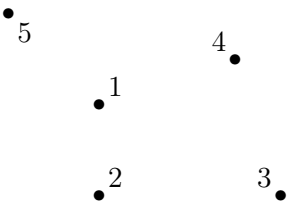
Файл GAME2.RES при цьому має вигляд

```
Виграє 1-ий гравець,
якщо 1-им ходом з'єднає точки 5, 3.
```

Сценарій гри:
I: 1–5; II: 3–1;
I: 2–4, 1–2; II: 4–2, 4–1;
I: 2–3; II: 3–5, 3–4;
I: 4–5; II: 2–5.
4.9. Якщо файл GAME2.DAT має вигляд

```
3 3   3 1   7 1   6 4   1 5
```

то на екрані монітора на початку гри маємо: точки 2, 3, 4, 5 є вершинами опуклого 4-кутника, а точка 1 — внутрішня точка цього 4-кутника, яка лежить зовні його діагоналей.



Файл GAME2.RES при цьому має вигляд

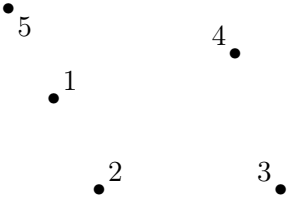
```
Виграє 2-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 1–5; II: 3–1;
I: 2–4, 1–2; II: 4–2, 4–1;
I: 2–3; II: 3–5, 3–4;
I: 4–5; II: 2–5.

4.10. Якщо файл GAME2.DAT має вигляд

```
2 3   3 1   7 1   6 4   1 5
```

то на екрані монітора на початку гри маємо: точки 2, 3, 4, 5 є вершинами опуклого 4-кутника, а точка 1 — середина сторони цього 4-кутника, яка сполучає точки 2 і 5.



Файл GAME2.RES при цьому має вигляд

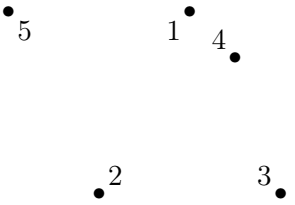
```
Виграє 1-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 2–5; II: 3–1, 1–4, 2–3;
I: 2–5, 5–3; II: 5–2, 5–4;
I: 4–3.

4.11. Якщо файл GAME2.DAT має вигляд

```
5 5   3 1   7 1   6 4   1 5
```

то на екрані монітора на початку гри маємо: точки 2, 3, 4, 1 і 5 — послідовні вершини опуклого 5-кутника.



Файл GAME2.RES при цьому має вигляд

```
Виграє 1-ий гравець за будь-яких обставин.
```

Сценарій гри:
I: 3–5; II: 2–1, 2–4, 2–3;
I: 2–5; II: 3–4;
I: 5–4; II: 1–4;
I: 1–5.

1.4 Розв'язання

1. Система числення. Цифри числа будь-якого натурального числа a у системі числення з основою b знаходяться послідовно, рахуючи справа наліво, як лишки від ділення на b: спочатку числа a, потім — його частки від ділення на b, потім — частки від ділення частки й т.д. до тих пір, поки остання частка не стане дорівнювати 0.

Повне розв'язання задачі вимагає реалізації "довгої арифметики": подана далі програма містить процедуру `divide` для ділення одного натурального числа на інше, для якої ділене, дільник, частка й остача подані масивами цифр.

```
const n_max=80;  type  carray=array[0..n_max] of shortint;
var o: text;  a,b,r,q: carray;  code: integer; s: string[1];
  ans: string;        la,lb,lr,lq,i,j,k: byte;

{Знаходження частки (lq цифр у масиві q) і остачі (lr цифр
у масиві r) від ділення одного числа (la цифр у масиві a)
                            на інше (lb цифр у масиві b)}
procedure divide; label NEXTi;                          begin
                                          if la<lb then begin
for j:=1 to la do r[j]:=a[j]; lr:=la;  lq:=1;  q[1]:=0  end
                                            else begin
for j:=1 to lb do r[j]:=a[j];          lq:=0; r[0]:=0; i:=0;
                           {Знаходження k - цифри частки q}
NEXTi: i:=i+1;  k:=0;                 while r[0]>=0 do  begin
                                  for j:=lb downto 1 do begin
       r[j]:=r[j]-b[j];                  if r[j]<0 then begin
       r[j]:=r[j]+10;  r[j-1]:=r[j-1]-1              end end;
                                  if r[0]>=0 then k:=k+1  end;
                                  for j:=lb downto 1 do begin
```

```
       r[j]:=r[j]+b[j];                if r[j]>9 then begin
       r[j]:=r[j]-10;  r[j-1]:=r[j-1]+1             end end;
if (k>0) or (i>1) then begin lq:=lq+1; q[lq]:=k end;
if    i+lb<=la    then                                  begin
for j:=1 to lb do r[j-1]:=r[j]; r[lb]:=a[i+lb];  goto NEXTi
                                                        end;
   lr:=lb;  while (r[lb-lr+1]=0) and (lr>1) do lr:=lr-1;
if lr< lb   then for j:=1 to lr do r[j]:=r[j+lb-lr]     end;
if lq= 0    then begin lq:=1; q[1]:=0 end               end;

begin{number}                                {Зчитування даних}
assign(o,'NUMBER.DAT');  reset(o);
readln(o,ans);  la:=length(ans);
    for j:=1 to la do  val(ans[j],a[j],code);
readln(o,ans);  lb:=length(ans);
    for j:=1 to lb do  val(ans[j],b[j],code);  close(o);

             {Знаходження запису у новій системі числення}
ans:='';  q[1]:=1;                     while q[1]>0 do begin
divide;    la:=lq;
if lr>1 then ans:=']'+ans;      for j:=lr downto 1 do begin
                               str(r[j],s);  ans:=s+ans end;
if lr>1 then ans:='['+ans; for j:=1 to la do a[j]:=q[j] end;

 assign(o,'NUMBER.RES');  rewrite(o);     {Запис відповіді}
writeln(o,ans);             close(o)                    end.
```

2. Спортивний прогноз. Задачу запозичено з книги Л.М.Лоповка "Збірник математичних задач логічного характеру", Київ, "Радянська школа", 1972, 151 с. Для успішного розв'язання задачі необхідно звернути увагу на таке:

- *опрацювання вхідних даних*;
- можливість розгляду всіх варіантів фінішування, тобто втілення *перебору всіх перестановок скінченної кількості елементів*;
- *оптимізацію перебору відкиданням продовження тих варіантів дешифрації, які неузгоджені з вхідними даними і умовою задачі.*

```
const mn_=26;                       label NEXTi, NEXTh;
var s: array[1..mn_] of string[mn_];               {Прогнози}
  v,h: array[1..mn_] of byte;       {Перестановки й гіпотези}
    m,                                 {Кількості: прогнозів}
    n,              {правильних відповідей в одному прогнозі}
   mn,                                              {бігунів}
    i,                              {Номер у списку перемежців}
   ii,  {Скільки разів правильно вгадано: результат бігуна i}
   kk,           {Скільки правильних відповідей у прогнозі k}
```

```
j,k,l: byte;  o: text;                                    BEGIN
                          assign(o,'PROGNOSE.DAT');   reset(o);
readln(o,m,n);  for j:=1 to m do readln(o,s[j]);    close(o);
                          assign(o,'PROGNOSE.RES'); rewrite(o);
mn:=m*n;  for j:= 1 to mn do v[j]:=j;
           i:=0;
NEXTi: inc(i);    h[i]:=0;
NEXTh:         inc(h[i]);            if h[i]>mn+1-i then begin
       dec(i);  if i=0 then begin close(o); halt end;
j:=v[h[i]]; v[h[i]]:=v[mn+1-i]; v[mn+1-i]:=j; goto NEXTh end;
                                              {Перевірка гіпотези}
ii:=0;  for k:=1 to m do if s[k][i]=s[1][v[h[i]]] then begin
inc(ii);  if ii>1 then goto NEXTh;     {Не було - не помилка}
kk:=1;
for l:=1 to i-1 do if s[k][l]=s[1][v[mn-l+1]] then inc(kk);
                                  if kk>n  then goto NEXTh end;
                                  if ii<>1 then goto NEXTh;
                          {Запам'ятовування зробленої гіпотези}
j:=v[h[i]];  v[h[i]]:=v[mn+1-i];  v[mn+1-i]:=j;
if i<mn then goto NEXTi;                       {Запис відповіді}
for j:=mn downto 1 do write(o,s[1][v[j]]);  writeln(o);
                                 {Відмова від гіпотези рівня i}
j:=v[h[i]]; v[h[i]]:=v[mn+1-i]; v[mn+1-i]:=j; goto NEXTh END.
```

3. Гра–1. Пошук стратегії потрібно істотно пов'язувати з кількістю точок (у поданій далі програмі виділено окремим блоком `if n=4 then begin ... end;`), а втілення гри — ні.

Пошук виграшної стратегії

Всі істотно різні для даної гри випадки розташування 4-х точок на площині перераховано у вказівках до перевірки, де й вказано оптимальну стратегію для кожного випадку.

Як же розпізнати всі ці випадки? Достатньо для кожної з 4-х даних точок розглянути її розташування стосовно трикутника, вершини якого — три інші точки. Для 4-х *різних* точок $A(x_1;y_1)$, $B(x_2;y_2)$, $C(x_3;y_3)$, $D(x_4;y_4)$ позначимо площі трикутників $s = S_{\triangle ABC}$, $s_1 = S_{\triangle DBC}$, $s_2 = S_{\triangle ADC}$, $s_3 = S_{\triangle ABD}$.

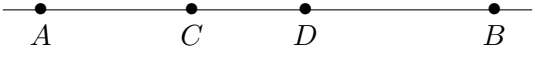


Рисунок 1: точки A, B, C і D розташовані на одній прямій тоді й лише тоді, коли $s = s_1 = s_2 = s_3$=0. Для того, щоб перемогти, першому гравцю достатньо сполучити точки A, B або C, D.

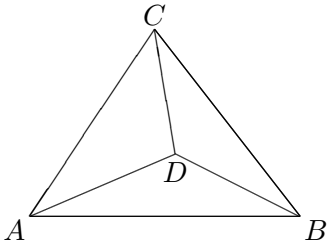


Рисунок 2: точка D — внутрішня точка $\triangle ABC$ тоді і лише тоді, коли одночасно $s_1 > 0$, $s_2 > 0$, $s_3 > 0$, $s = s_1 + s_2 + s_3$. Переможе 2-ий гравець, бо за будь-якого сценарію гри буде проведено 6 відрізків.

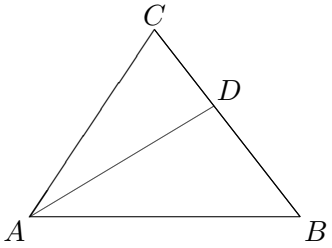


Рисунок 3: точка D — внутрішня точка сторони BC (AC чи AB), якщо s_1=0 (s_2=0 чи s_3=0 відповідно) і $s = s_1 + s_2 + s_3$. Переможе 1-ий гравець, бо за будь-якого сценарію гри буде проведено непарну кількість відрізків — 3 або 5.

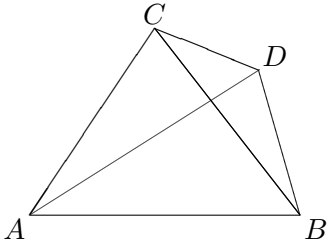


Рисунок 4: точка D знаходиться строго зовні $\triangle ABC$ тоді і лише тоді, коли $s < s_1 + s_2 + s_3$.

Чотири точки є вершинами опуклого чотирикутника тоді і лише тоді, коли кожна з них лежить зовні трикутника, утвореного трьома іншими вершинами. Тому потрібно передбачити можливість аналізу розташування кожної з 4-х точок відносно трикутника, утвореного трьома іншими точками як вершинами — див. використання масиву `ii` у поданій далі програмі.

Нераціонально користуватися формулою Герона для знаходження площі трикутника на координатній площині. Подамо формулу для знаходження площі

трикутника ABO за відомими координатами вершин $O(0;0)$, $A(a_1;a_2)$ та $B(b_1;b_2)$ Позначимо $a = |AO|$ і $b = |BO|$ — довжини двох сторін трикутника, φ_a і φ_b — кути між додатнім напрямом осі абсцис і цими сторонами, $\varphi = \angle AOB = |\varphi_b - \varphi_a|$ — кут між цими сторонами. Тоді

$$2S_{\triangle AOB} = ab\sin\varphi = ab|\cos\varphi_a \sin\varphi_b - \sin\varphi_a \cos\varphi_b| = |a_1b_2 - a_2b_1|,$$

де для доведення останньої рівності a потрібно помножити на тригонометричні функції φ_a, b — на тригонометричні функції φ_b. Будь-який трикутник координатної площини паралельним переносом можна перемістити таким чином, щоб одна з вершин співпала з початком координат. Тому подану формулу легко узагальнити на випадок довільного трикутника (див. означення функції `sq` поданої далі програми).

Реалізація гри

I_{x_i}, I_{y_i} — екранні координати i-тої точки — знаходилися за формулами

$$I_{x_i} = 2 + \left[N_x \cdot \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}\right],$$

$$I_{y_i} = 2 + \left[N_y \cdot \frac{y_{\max} - y_i}{y_{\max} - y_{\min}}\right],$$

де:

$x_{\min}$ і $x_{\max}$ — відповідно найменше та найбільше значення абсцис точок координатної площини, що відображаються на екрані монітора (у поданій далі програмі — `x0` і `x1`);

$y_{\min}$ і $y_{\max}$ — відповідно найменше та найбільше значення ординат точок координатної площини, що відображаються на екрані монітора (у поданій далі програмі — `y0` і `y1`);

N_x і N_y — розміри поля для гри у пікселях по горизонталі й вертикалі відповідно. У поданій далі програмі ці розміри вибрано меншими від максимально можливих `getmaxx` і `getmaxy`, щоб:

- кожну дану точку зображати кружечком з радіусом 2, а не екранною точкою;
- залишити місце на екрані для запрошення до наступного ходу та оголошення результату гри.

Гра вимагає ведення обліку відрізків, які не можна проводити наступними ходами (якщо вони вже проведені, або у результаті їх проведення буде порушено умову гри: жодна внутрішня точка проведеного відрізка не належить іншому проведеному відрізку).

Для n різних точок існує $n_1 = n(n-1)/2$ відрізків, що їх сполучають. Нехай $1 \le j < k \le n$. Відрізку, який сполучає j-ту й k-ту точки, поставимо

у відповідність натуральне число $l = j + (k-2)(k-1)/2$. Такий спосіб нумерації відрізків узгоджений з наступною нумерацією *різних* елементів симетричної матриці (квадратної таблиці) з нулями на діагоналі

$$\begin{array}{l} 0\ 1\ 2\ 4\ \ 7\ \ 11\ \ldots \\ \star\ 0\ 3\ 5\ \ 8\ \ 12\ \ldots \\ \star\ \star\ 0\ 6\ \ 9\ \ 13\ \ldots \\ \star\ \star\ \star\ 0\ 10\ 14\ \ldots\ , \\ \star\ \star\ \star\ \star\ \ 0\ \ 15\ \ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots \end{array}$$

де j — номер рядка, k — номер стовпчика. У поданій далі програмі елемент масиву булевих змінних `b[l]=true`, якщо проведення l-го відрізку порушує правила гри для розглядуваного сценарію гри.

Для n_1 відрізків існує $n_2 = n_1(n_1-1)/2$ пар *різних* відрізків. Нехай $1 \le l < l' \le n_1$. Парі відрізків з номерами l і l' поставимо у відповідність число $m = l + (l'-2)(l'-1)/2$. У поданій далі програмі елемент масиву булевих змінних `c[m]=true`, якщо проведення обох відрізків l і l' суперечить правилам гри. Мета переходу саме до лінійного масиву булевих змінних — 8-кратна економія оперативної пам'яті на `c` і 2-кратна на масиві `b`. Елементи масиву `c` визначаються за координатами точок (див. далі визначення неможливості проведення пари відрізків в одній грі).

Після проведення l-го відрізка серед всіх відрізків запам'ятовуємо ті, проведення яких під час наступних ходів суперечитиме правилам гри, а саме l-ий відрізок і всі ті відрізки l', для яких:

- або $l' < l$ і для $m = l' + (l-2)(l-1)/2$ `c[m]=true`;
- або $l < l'$ і для $m = l + (l'-2)(l'-1)/2$ `c[m]=true`.

Визначення неможливості проведення пари відрізків

Перш за все потрібно знайти точку перетину прямих, що є продовженням даних відрізків. Для цього використаємо традиційний апарат аналітичної геометрії.

$$(x - x_1)(y_2 - y_1) = (y - y_1)(x_2 - x_1)$$

— рівняння прямої, що проходить через *різні* точки $(x_1; y_1)$ та $(x_2; y_2)$.

Якщо $|a_{11}| + |a_{12}| > 0 < |a_{21}| + |a_{22}|$, то кожне рівняння системи

$$\begin{cases} a_{11}x + a_{12}y = b_1 \\ a_{21}x + a_{22}y = b_2 \end{cases}$$

задає пряму. Для довільного розв'язку цієї системи справджуються рівності

$$\begin{cases} x \cdot \Delta = \Delta_x \\ y \cdot \Delta = \Delta_y \end{cases}, \qquad \begin{cases} \Delta = a_{11}a_{22} - a_{21}a_{12} \\ \Delta_x = b_1a_{22} - b_2a_{12} \\ \Delta_y = a_{11}b_2 - a_{21}b_1 \end{cases}.$$

1. Якщо $\Delta \neq 0$, то розв'язок єдиний (прямі перетинаються у точці):

$$x = \Delta_x/\Delta, \qquad y = \Delta_y/\Delta.$$

   У цьому випадку для порушення правил гри при проведенні цих відрізків в одному сеансі гри необхідно й достатньо, щоб точка перетину належала одному відрізку й була внутрішньою точкою іншого.

2. Якщо $\Delta = 0$, але хоче б одне з чисел Δ_x та Δ_y відмінне від 0, то розв'язків система не має (прямі паралельні). У цьому випадку відрізки на цих прямих можна провести в одному сеансі гри.

3. Якщо $\Delta = \Delta_x = \Delta_y = 0$, то рівняння системи еквівалентні, а система має безліч розв'язків (прямі співпадають). У цьому випадку для порушення правил гри при проведенні цих відрізків в одному сеансі гри необхідньо й достатньо, щоб кінець одного з відрізків був внутрішньою точкою іншого.

```
uses  crt,graph;                              label ANSWER,GAME;
const n=4; n1=(n -1)*n  div 2;  kx=20; ky=12; c1=10; c2=12;
           n2=(n1-1)*n1 div 2;
var play:string[2];   s,s1,s2,s3,m,nx,ny,htext,vtext: word;
   a11,a12,a21,a22,b1,b2,d,dx,dy,detect,graphmode: integer;
     j,k,l,jj,kk,ll,x0,x1,y0,y1,x00,x11,y00,y11,w:    byte;
x,y,i: array[1..n] of shortint; b: array[1..n1] of boolean;
ix,iy: array[1..n] of word;     c: array[1..n2] of boolean;
xr,yr: real;       o: text;              online,f: boolean;
                      {Фунція - подвоєна площа трикутника,
                  вершини якого - точки i[k1], i[k2], i[k3].}
function sq (k1, k2, k3: byte): word;                 begin
sq:=abs((x[i[k2]]-x[i[k1]])*(y[i[k3]]-y[i[k1]])
       -(x[i[k3]]-x[i[k1]])*(y[i[k2]]-y[i[k1]]))       end;
begin{game1}
for j:=1 to n1 do b[j]:=false;  online:=false;
for d:=1 to n2 do c[d]:=false;       w:=1;
assign(o,'GAME1.DAT');          {Зчитування даних} reset(o);
for j:=1 to n do read(o,x[j],y[j]);                close(o);

     {Які пари відрізків не можна проводити в одній грі?..}
for j :=1 to n-1 do for k :=j +1 to n do              begin
a11:=y[k]-y[j];   a12:=x[j]-x[k];   b1:=a11*x[j]+a12*y[j];
if x[j]<x[k] then begin x0:=x[j];  x1:=x[k] end
             else begin x0:=x[k];  x1:=x[j] end;
if y[j]<y[k] then begin y0:=y[j];  y1:=y[k] end
             else begin y0:=y[k];  y1:=y[j] end;
l :=(k -2)*(k -1) div 2 + j;
for jj:=1 to n-1 do for kk:=jj+1 to n do              begin
ll:=(kk-2)*(kk-1) div 2 + jj;            if l<ll then begin
```

```
 m:=(ll-2)*(ll-1) div 2 + l;
a21:=y[kk]-y[jj]; a22:=x[jj]-x[kk]; b2:=a21*x[jj]+a22*y[jj];
if x[jj]<x[kk] then begin x00:=x[jj];  x11:=x[kk] end
               else begin x00:=x[kk];  x11:=x[jj] end;
if y[jj]<y[kk] then begin y00:=y[jj];  y11:=y[kk] end
               else begin y00:=y[kk];  y11:=y[jj] end;
d:=a11*a22-a12*a21;  dx:=b1*a22-a12*b2;  dy:=a11*b2-b1*a21;
                                        if d<>0 then begin
{Якщо продовження відрізків перетинаються в одній точці...}
xr:=dx/d;  yr:=dy/d;                                 c[m]:=
 ((x0<=xr) and (xr<=x1 ) and (y0 <=yr) and (yr<=y1 ) and
((x00< xr) and (xr< x11) or  (y00< yr) and (yr< y11)))   or
((x00<=xr) and (xr<=x11) and (y00<=yr) and (yr<=y11) and
((x0 < xr) and (xr< x1 ) or  (y0 < yr) and (yr< y1 )))  end

                   {Якщо відрізки належать одній прямій...}
else if (d=0) and (dx=0) and (dy=0) then c[m]:=
(x0<x00) and (x00<x1) or (x0<x11) and (x11<x1) or
(x00<x0) and (x0<x11) or (x00<x1) and (x1<x11) or
(y0<y00) and (y00<y1) or (y0<y11) and (y11<y1) or
(y00<y0) and (y0<y11) or (y00<y1) and (y1<y11)  end end end;

         {Визначення найменших і найбільших координат точок}
x0:=x[1];  x1:=x[1];
y0:=y[1];  y1:=y[1];                  for j:=2 to n do begin
if x[j]<x0 then x0:=x[j] else if x[j]>x1 then x1:=x[j];
if y[j]<y0 then y0:=y[j] else if y[j]>y1 then y1:=y[j]  end;
dx:=x1-x0;  dy:=y1-y0;
                      {Пошук стратегії}  if n=4 then begin
for j:=1   to n   do                                   begin
for k:=1   to j-1 do i[k]  :=k;
for k:=j+1 to n   do i[k-1]:=k; i[4]:=j;

                     {Встановлення розташування точки i[4]
відносно трикутника  з вершинами у точках i[1], i[2], i[3]:
   w=2, якщо i[4] - внутрішня точка трикутника, інакше w=1,
online справджується, якщо всі точки належать одній прямій.}
s:=sq(1,2,3); s1:=sq(4,2,3); s2:=sq(1,4,3); s3:=sq(1,2,4);

                  {Якщо всі точки належать одній прямій...}
if (s1=0) and (s2=0) and (s3=0) then                   begin
w:=1;  online:=true;
if x0<>x1 then   {Якщо пряма не горизонтальна}         begin
for l:=1 to n   do                               begin f:=true;
for k:=1 to n-l do if x[i[k]]>x[i[k+1]] then begin f:=false;
j:=i[k];  i[k]:=i[k+1];  i[k+1]:=j;                end;
                               if f then break end      end
```

```
          else   {Якщо пряма горизонтальна...}        begin
for l:=1 to n   do                              begin f:=true;
for k:=1 to n-l do if y[i[k]]>y[i[k+1]] then begin f:=false;
j:=i[k];  i[k]:=i[k+1];  i[k+1]:=j;             end;
                                  if f then break end      end;
                                               goto ANSWER end;
if (s=s1+s2+s3) then                                      begin
if (s1>0) and (s2>0) and (s3>0) then w:=2;  goto ANSWER end;
                                                        end end;
{Запис відповіді про результат гри за оптимальної стратегії}
ANSWER:  assign(o,'GAME1.RES');                    rewrite(o);
write(o,' Виграє ',w,'-ий гравець');
if online then writeln(o,', якщо 1-им ходом з‘єднає точки ',
                             i[1],', ',i[n],'.')
          else writeln(o,' за будь-яких обставин.');close(o);

                                  {Приготування екрану до гри}
detectgraph(detect,graphmode);
  initgraph(detect,graphmode,'c:\pascal\bgi');
nx:=((2*getmaxx) div 3);  ny:=getmaxy-4;
setbkcolor(0);  setcolor(15);  textcolor(c1);
if dx*ny<dy*nx then dx:=((dy*nx) div ny) else
if dx*ny>dy*nx then dy:=((dx*ny) div nx);
x1:=x0+dx;  y1:=y0+dy;                for j:=1 to n do begin
                                     {Зображення j-оï точки}
ix[j]:=2+((nx*(x[j]-x0)) div dx);
iy[j]:=2+((ny*(y1-y[j])) div dy);
fillellipse(ix[j],iy[j],2,2); str(j,play);
                                           {Підпис j-оï точки}
if 2*ix[j]<nx then begin b1:=ix[j]+3; htext:=0 end
              else begin b1:=ix[j]-3; htext:=2 end;
if 2*iy[j]<ny then begin b2:=iy[j]+3; vtext:=2 end
              else begin b2:=iy[j]-3; vtext:=0 end;
settextjustify(htext,vtext);  outtextxy(b1,b2,play)     end;
settextjustify(0,2);
outtextxy(nx+kx,0,   ' -ий гравець!');
outtextxy(nx+kx,ky,  'Задайте номери точок,');
outtextxy(nx+kx,ky*2,'які хочете сполучити');
outtextxy(nx+kx,ky*3,'відрізком '); setcolor(c1); play:='1';

                                                  {Новий хід}
GAME:  outtextxy(nx+kx,0,play);  gotoxy(67,3);  readln(j,k);
if (j>n) or (j<0) or (k<0) or (k>n) or (k=j) then goto GAME;
if j<k then l:=(k-2)*(k-1) div 2+j
       else l:=(j-2)*(j-1) div 2+k;
if b[l] then goto GAME else line(ix[j],iy[j],ix[k],iy[k]);
                                {Встановлення нових заборон}
```

```
b[l] :=true;                          for ll:=1   to l-1 do
b[ll]:=b[ll] or c[(l -2)*(l -1) div 2 +ll];
                                      for ll:=l+1 to n1  do
b[ll]:=b[ll] or c[(ll-2)*(ll-1) div 2 +l ];
                                           {Передача ходу}
for j:=1 to n1 do if not b[j] then                    begin
setcolor(0);  outtextxy(nx+20,0,play);    if play='2'
then begin play:='1'; setcolor(c1); textcolor(c1) end
else begin play:='2'; setcolor(c2); textcolor(c2) end;
goto GAME                                              end;
                                  {Кінець гри}  setcolor(0);
outtextxy(nx+kx,ky  ,'Задайте номери точок,');
outtextxy(nx+kx,ky*2,'які хочете сполучити ');
outtextxy(nx+kx,ky*3,'відрізком            '); setcolor(15);
outtextxy(nx+kx,ky  ,'Ви перемогли!');
outtextxy(nx+kx,ky*2,'Останній хід:');readln;closegraph;end.
```

Подана програма придатна для гри з більшою кількістю точок координатної площини: достатньо збільшити сталу `n`.

4. Гра–2. Як і в "Грі–1" з 4-ма точками, пошук стратегії залежить від кількості точок, а реалізація гри — ні. Всі істотно різні для даної гри випадки розташування 5-х точок на площині перераховано у вказівках до перевірки, де й вказано оптимальну стратегію для кожного випадку. Лише після того, як перевірено, що неможливо всі точки розташувати всередині трикутника з вершинами у трьох з даних точках, потрібно перевірити, чи можна їх розташувати всередині опуклого 4-кутника (див. другий цикл за змінними `j4`, `j5`). Програма для "Гри–2" відрізняється від програми для "Гри–1": додатковими змінними

```
s14,s24,s34,s15,s25,s35,s145,s245,s345: word;
j4,j5,w1,w2: byte;
```

записом відповіді

```
ANSWER:  assign(o,'GAME.RES');                    rewrite(o);
write(o,' Виграє ',w,'-ий гравець');
if w1>0 then writeln(o,', якщо 1-им ходом з‘єднає точки ',
                         w1,', ',w2,'.')
        else writeln(o,' за будь-яких обставин.'); close(o);
```

пошуком виграшної стратегії:

```
                       {Пошук стратегії}  if n=5 then begin
for j4:=1 to n-1 do for j5:=j4+1 to n do              begin
i[5]:=j5;  i[4]:=j4;  l:=0;
for k:=1 to n do          if (k<>j4) and (k<>j5) then begin
                                      l:=l+1;  i[l]:=k end;
       {Встановлення розташування точок i[4], i[5] відносно
        трикутника з вершинами у точках i[1], i[2], i[3]}
```

```
   s:=sq(1,2,3);
 s14:=sq(4,2,3);   s24:=sq(1,4,3);   s34:=sq(1,2,4);
 s15:=sq(5,2,3);   s25:=sq(1,5,3);   s35:=sq(1,2,5);
s145:=sq(1,4,5);  s245:=sq(2,4,5);  s345:=sq(3,4,5);
                  {Якщо всі точки належать одній прямій...}
if (s14=0) and (s24=0) and (s34=0) and
   (s15=0) and (s25=0) and (s35=0) then                begin
w:=1;  online:=true;
if x0<>x1 then   {Якщо пряма не горизонтальна}         begin
for l:=1 to n   do                              begin f:=true;
for k:=1 to n-l do if x[i[k]]>x[i[k+1]] then begin f:=false;
j:=i[k];  i[k]:=i[k+1];  i[k+1]:=j;               end;
                                if f then break end       end
          else   {Якщо пряма горизонтальна...}         begin
for l:=1 to n   do                              begin f:=true;
for k:=1 to n-l do if y[i[k]]>y[i[k+1]] then begin f:=false;
j:=i[k];  i[k]:=i[k+1];  i[k+1]:=j;               end;
                                if f then break end       end;
w:=1;  w1:=i[1];  w2:=i[n];                   goto ANSWER end;
                           if s>0 then                  begin
{Якщо точки i[4], i[5] знаходяться всередині трикутника ...}
if (s=s14+s24+s34) and (s=s15+s25+s35) then             begin

{Якщо точки i[4], i[5] - внутрішні точки трикутника ...}
if (s14>0) and (s24>0) and (s34>0) and
   (s15>0) and (s25>0) and (s35>0)     then             begin
{Якщо точки i[4], i[5] - на прямій, що містить вершину ...}
if (s145=0) or (s245=0) or (s345=0) then                begin
w:=1;  w1:=i[4];  w2:=i[5];               goto ANSWER     end

{Якщо немає прямої, що містить точки i[4],i[5] і вершину...}
                                     else               begin
w:=1;  w1:=0;  w2:=0;                     goto ANSWER end end;

           {Якщо точка i[4] внутрішня...}
if (s14>0) and (s24>0) and (s34>0) then                  begin
if (s35=0) and (s345=0) then                             begin
w:=1;  w1:=i[5];  w2:=i[3];               goto ANSWER      end;
if (s25=0) and (s245=0) then                             begin
w:=1;  w1:=i[5];  w2:=i[2];               goto ANSWER      end;
if (s15=0) and (s145=0) then                             begin
w:=1;  w1:=i[5];  w2:=i[1];               goto ANSWER      end;
w:=2;  w1:=0;     w2:=0;                  goto ANSWER      end;

           {Якщо точка i[5] внутрішня...}
if (s15>0)  and (s25>0) and (s35>0) then                 begin
if (s34=0) and (s345=0) then                             begin
```

```
w:=1;  w1:=i[4];  w2:=i[3];             goto ANSWER      end;
if (s24=0) and (s245=0) then                            begin
w:=1;  w1:=i[4];  w2:=i[2];             goto ANSWER      end;
if (s14=0) and (s145=0) then                            begin
w:=1;  w1:=i[4];  w2:=i[1];             goto ANSWER      end;
w:=2;  w1:=0;     w2:=0;                goto ANSWER      end;
w:=1;  w1:=0;     w2:=0;  goto ANSWER           end end end;

for j4:=1 to n-1 do for j5:=j4+1 to n do             begin
i[5]:=j5;  i[4]:=j4;  l:=0;
for k:=1 to n do          if (k<>j4) and (k<>j5) then begin
                                      l:=l+1;  i[l]:=k end;
      {Встановлення розташування точок i[4], i[5] відносно
       трикутника з вершинами у точках i[1], i[2], i[3]}
   s:=sq(1,2,3);
 s14:=sq(4,2,3);   s24:=sq(1,4,3);   s34:=sq(1,2,4);
 s15:=sq(5,2,3);   s25:=sq(1,5,3);   s35:=sq(1,2,5);
s145:=sq(1,4,5);  s245:=sq(2,4,5);  s345:=sq(3,4,5);
if (s=s14+s24+s34) and (s<s15+s25+s35) then           begin
if(s14>0) and (s24>0) and (s34>0) then if s145=0 then begin
w:=1;  w1:=i[1];  w2:=i[5];  goto ANSWER               end
                                    else if s245=0 then begin
w:=1;  w1:=i[2];  w2:=i[5];  goto ANSWER               end
                                    else if s345=0 then begin
w:=1;  w1:=i[3];  w2:=i[5];  goto ANSWER               end
                                    else               begin
w:=2;  w1:=0;     w2:=0;     goto ANSWER           end end;
if (s<s14+s24+s34) and (s=s15+s25+s35) then           begin
if(s15>0) and (s25>0) and (s35>0) then if s145=0 then begin
w:=1;  w1:=i[1];  w2:=i[4];  goto ANSWER               end
                                    else if s245=0 then begin
w:=1;  w1:=i[2];  w2:=i[4];  goto ANSWER               end
                                    else if s345=0 then begin
w:=1;  w1:=i[3];  w2:=i[4];  goto ANSWER               end
                                    else               begin
w:=2;  w1:=0;     w2:=0;     goto ANSWER       end end end;
w:=1;  w1:=0;     w2:=0;                                end;
```

Задача ускладнюється (за рахунок власне програмування) доповненням вимог до програми словами: “передбачає можливість вибору (за допомогою клавіатури після відповідного повідомлення на екрані монітора) режиму гри: двох людей-гравців чи гри з комп’ютером людини-гравця (в останньому випадку передбачається також можливість вибору права першого ходу)”. Очевидно, що складність математичної моделі істотно зростає за рахунок збільшення кількості точок.

2 II етап

2.1 Умови проведення

Олімпіада проводять в один практичний тур тривалістю 5 (п'ять) астрономічних годин. Коректність даних перевіряти не потрібно. З метою врахування рівня знань, умінь та навичок учасників II етапу, журі може долучити до поданих далі трьох завдань ще одне, за повне розв'язання якого присуджувати не більше 9 балів.

2.2 Завдання

1. Камінці, 12 балів. n ямок розташовано у ряд. Для натурального k в межах від 1 до n через j_k позначимо кількість камінців, які лежать у k-ій ямці, якщо рахувати ямки зліва направо. Нехай для деякого натурального $m > 1$ послідовність натуральних чисел l_1, l_2, ... , l_m зростає. Двоє гравців грають наступним чином:

- за один хід дозволяється перекласти з будь-якої ямки у сусідню праворуч (порядок розташування ямок для обох гравців однаковий) лише таку кількість камінців, що дорівнює одному з чисел l_1, l_2, ... , l_m;
- програє той, хто не може зробити хід (коли в усіх ямках, крім крайньої праворуч, кількість камінців менша за l_1).

1-ий рядок вхідного файлу STONES.DAT містить у вказаному порядку натуральні числа n та m, які лежать в межах від 2 до 5 включно. 2-ий рядок цього ж файлу містить у вказаному порядку послідовність n невід'ємних цілих чисел j_1, j_2, ... , j_n. 3-ій рядок цього ж файлу містить послідовність m натуральних чисел l_1, l_2, ... , l_m, що зростає, причому $l_1+l_2+\cdots+l_m < 246$. Назвемо *позицією* цієї гри розташування камінців у ямках і номер гравця, чия черга ходити. Відомо, що кількість всіх можливих позицій гри не перевищує 1000.

Створіть програму STONES.*, яка у 1-ий рядок вихідного файлу STONES.RES запише номер гравця (спочатку ходить 1-ий гравець, потім — 2-ий), який може гарантувати собі виграш. Якщо це буде 1, то кожен з наступних рядків цього самого файлу повинен містити по n невід'ємних цілих чисел, що є кількостями камінців у ямках після 1-го ходу, який веде до перемоги 1-го гравця. Потрібно подати у довільному порядку всі можливі варіанти, по 1 рядку на кожний варіант ходу.

Приклади файлів	I-ий варіант	II-ий варіант
STONES.DAT	3 2 4 0 1 2 3	3 2 2 2 1 2 3
STONES.RES	2	1 2 0 3

2. Десятковий дріб, 10 балів. Створіть програму FRACTION.*, яка подає відношення двох взаємно простих натуральних чисел a і b десятковим дробом. Перший і другий рядки вхідного файлу FRACTION.DAT містять відповідно десяткові записи натуральних чисел a і b, кожне — до 77 цифр, $b > 1$. Наприклад,

```
41
6
```

В єдиний рядок вихідного файлу FRACTION.RES потрібно записати подання нескоротного дробу a/b десятковим дробом (використовувати десяткову крапку): якщо можливо, то скінченним десятковим дробом, інакше період вказати у круглих дужках. Для поданого прикладу даних файл FRACTION.RES містить запис

```
6.8(3)
```

— див. подання 41/6=6.8333333... .

3. Офіцери, 9 балів. За даними розвідки серед 6-ти офіцерів A, B, C, D, E, F є 3 полковники, 2 майори і 1 капітан. Єдиний рядок вхідного файлу OFFICERS.DAT містить у вказаному порядку 8 літер латиниці a_1, a_2, ... a_8, не розділених прогалинами, якими позначено офіцерів (деякі літери з переліку A, B, C, D, E, F можуть повторюватися, а деякі — не зустрічатися взагалі). Відомо (у кожному з поданих далі *простих* речень офіцери, про яких іде мова, різні), що a_1 зобов'язаний першим вітати a_2 (майор зобов'язаний першим вітати полковника, а капітан — і полковника, і майора). a_3 і a_4 — в одному званні. a_5 і a_6 — в одному роді військ. Полковник, майор і a_7 — танкісти. a_8 і капітан — артилеристи. Один з офіцерів — зв'язківець. Створіть програму OFFICERS.*, яка у файл OFFICERS.RES запише, хто є хто серед офіцерів. Потрібно через кому перерахувати всіх офіцерів у алфавітному порядку, вказавши через дефіс перші літери звання та роду військ) — по 1 рядку на кожен варіант відповіді. Наприклад, якщо файл OFFICERS.DAT має вигляд

```
ABDECFDB
```

то файл OFFICERS.RES має 2 рядки

```
A к-а, B м-а, C м-т, D п-т, E п-з, F п-т.
A к-а, B м-а, C п-т, D п-т, E п-з, F м-т
```

Останній рядок читають так: A — капітан-артилерист, B — майор-артилерист, C — полковник-танкіст, D — полковник-танкіст, E — полковник-зв'язківець, F — майор-танкіст. Якщо розвідувальні дані хибні (призводять до логічних суперечностей), то файл OFFICERS.RES порожній.

2.3 Вказівки до перевірки

1. Камінці. Перевіряється можливість здійснення аналізу всіх позицій гри з метою встановлення, які з них виграшні, а які програшні. За кожен тест — 2 бали у разі успішного виконання *всіх* попередніх тестів. Зауважимо:

- тести 1–2 перевіряють правильність аналізу позицій гри Баше з одними правилами гри;
- тести 3–4, подані в умові задачі як приклади, стосуються ігор з одиними правилами гри, причому початкова виграшна позиція тесту 4 отримується за один хід з початкової програшної позиції тесту 3.

Тест	STONES.DAT	STONES.RES
1.1	2 5 233 0 1 2 3 4 5	1 228 5
1.2	2 5 234 0 1 2 3 4 5	2
1.3	3 2 4 0 1 2 3	2
1.4	3 2 2 2 1 2 3	1 2 0 3
1.5	4 3 9 0 6 0 2 3 5	1 6 3 6 0 9 0 3 3
1.6	5 5 11 0 0 3 0 3 5 7 9 11	1 8 3 0 3 0 6 5 0 3 0 11 0 0 0 3

2. Десятковий дріб. За кожен тест — по 1 балу.
2.1. Якщо файл FRACTION.DAT має вигляд

```
19
8
```

то файл FRACTION.RES містить запис

```
2.375
```

2.2. Якщо файл FRACTION.DAT має вигляд

```
13
25
```

то файл FRACTION.RES містить запис

```
0.52
```

2.3. Якщо файл FRACTION.DAT має вигляд

```
17
12
```

то файл FRACTION.RES містить запис

```
1.41(6)
```

2.4. Якщо файл FRACTION.DAT має вигляд

```
347
115
```

то файл FRACTION.RES містить запис

```
3.0(1739130434782608695652)
```

2.5. Якщо файл FRACTION.DAT має вигляд

```
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901
7
```

то файл FRACTION.RES містить такий запис.

```
17636684144620811271604938270017636684144620811271604938270017636684 14.(428571)
```

2.6. Якщо файл FRACTION.DAT має вигляд

```
1234567890123456789012345678901234567890000000000000000000000000000000001
9999999999999999999999999999999
```

то файл FRACTION.RES містить наступний запис.

```
12345678901234567890123456789024691357 80.(12345678901234567890246913578 1)
```

2.7. Якщо файл FRACTION.DAT має вигляд

```
12345678901234567890123456789012345678901234567890123456789012345678901
57896044618658097711785492504343953926 6 34992332820282019728792003956564819968
```

то файл FRACTION.RES містить такий запис.

```
0.000000213238727836269817821328358863234 0
  116797151114711981301262196368756562861 7
  006061358884293131618039653176316658457 2
  256261381065869249014388218673632195914 5
  518067219536958092223097630518678170876 4
  294758876609112846489679782280290964990 8
  542633056640625
```

2.8. Якщо файл FRACTION.DAT має вигляд

```
12345678901234567890123456789012345678901234567890123456789012345678901
18173039004871896699856737152270336
```

(у перших двох рядках тексту подано символи першого рядка вхідного файлу, у третьому — символи другого рядка — десятковий запис числа $7 \cdot 2^{111}$), то файл FRACTION.RES містить такий запис.

```
67934036227649606766755496118493117 1.
96161941332267453268320439114596268308 1
24440736248117718928149363740210041142 01
86836559358718139784676688058 03(571428)
```

2.9. Якщо файл FRACTION.DAT має вигляд

```
12345678901234567890123456789012345678901234567890123456789012345678901
51650883406386744524800000000000000000000
```

то файл FRACTION.RES містить наступний запис.

```
23902164081298167563098124767 8.
20421460673393028461312517131845537427 46
98138873611925297219776049522416932242 25
725446(428571)
```

2.10. Якщо файл FRACTION.DAT має вигляд

```
12345678901234567890123456789012345678901234567890123456789012345678901
10000000000000000000000000000000000000000
```

то файл FRACTION.RES містить такий запис.

```
12345678901234567890123456789 012.3456789012345678901234567890123456789 01
```

3. Офіцери. За кожен тест — по 3 бали, причому бали за 2-ий і 3-ій тести присуджуються лише, якщо успішно пройдено попередні тести (відповідно 1-ий та 1-ий і 2-ий одночасно). Перевіряють уміння:

- перейти від символьних змінних до натуральних і орєанізувати перебір за всіма можливими значеннями останніх;
- записати твердження сюжетної задачі за допомогою систем і сукупностей нерівностей між відповідними натуральними змінними.

3.1. Якщо файл OFFICERS.DAT має вигляд

```
FECBDACE
```

(з поданих в умові даних отримується заміною літер A, B, C, D, E, F на F, E, D, C, B, A відповідно), то файл OFFICERS.RES має вигляд

```
A м-т, B п-з, C п-т, D п-т, E м-а, F к-а.
A п-т, B п-з, C п-т, D м-т, E м-а, F к-а.
```

3.2. Якщо файл OFFICERS.DAT має вигляд

```
ABCDABCD
```

то файл OFFICERS.RES має вигляд

```
A м-т, B п-т, C п-т, D п-а, E к-а, F м-з.
A м-т, B п-т, C п-т, D п-а, E м-з, F к-а.
```

3.3. Якщо файл OFFICERS.DAT має вигляд

```
FEDCBABA
```

то файл OFFICERS.RES порожній, тобто розвідувальні дані хибні.

2.4 Розв'язання

1. Камінці. Зауважимо, що гра закінчиться за скінченну кількість ходів для будь-якого початкового розташування камінців і для будь-яких ходів суперників. Скористаємося стандартним підходом теорії ігор[1] — подамо всі можливі сценарії гри *графом гри*, який складається з:

- вершин (*позицій*) — четвірок невід'ємних цілих чисел: перші три числа дорівнюють кількостям камінців у відповідних ямках, а четверте число є номером гравця, який робить хід;

[1]Теорія ігор — теорія математичних моделей прийняття оптимальних рішень в умовах конфліктів інтересів.

- дуг — стрілок, які вказують, яку позицію з якої можна отримати за один хід.

Вершину, на яку не вказує жодна стрілка, назвемо *початковою* позицією (початком гри). Вершину, з якої не виходить жодна стрілка, назвемо *заключною* позицією (кінцем гри). Назвемо позицію *виграшною,* якщо гравець, чия черга ходу в цій позиції, може вибрати такий хід, який забезпечить йому перемогу (за умови правильного вибору й усіх наступних ходів). Наприклад, такими є ті позиції, з яких наступним ходом можна отримати заключні позиції. Натомість позицію назовемо *програшною,* якщо вона є заключною (неможливо зробити хід), або будь-який хід у цій позиції веде до виграшної позиції.

Схема аналізу графа гри.

1. Визначити всі заключні позиції і запам'ятати їх як програшні.
2. Визначити й запам'ятати всі виграшні позиції, з яких визначені на попередньому кроці програшні позиції досягаються за 1 хід.
3. Визначити й запам'ятати всі програшні позиції, з яких визначені на попередньому кроці виграшні позиції досягаються за 1 хід.
4. Повторювати виконання пунктів 2–3 до тих пір, поки не буде встановлено, якою є початкова позиція.

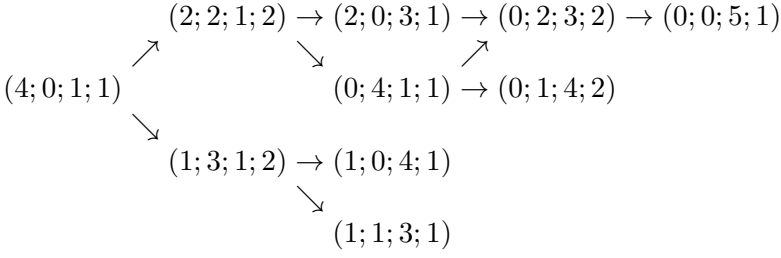


Рисунок 5: граф гри для запропонованих в умові задачі II.1 правил гри і першого набору вхідних даних.

Проведемо за цією схемою аналіз графа гри для запропонованих в умові задачі II.1 правил гри і першого набору вхідних даних.

1. Заключними, а тому програшними, є позиції (0;0;5;1), (0;1;4;2), (1;0;4;1), (1;1;3;1).
2. Виграшними є позиції (0;2;3;2), (0;4;1;1), (1;3;1;2), бо з них наступним ходом отримується одна з перерахованих заключних позицій.

3. Програшною є позиція (2;0;3;1), бо наступним ходом можна отримати лише виграшну позицію (0;2;3;2). Про програшність початкової позиції (4;0;1;1) робити висновок рано, бо ще не встановлено виграшність позиції (2;2;1;2).

2′. Виграшною є позиція (2;2;1;2), бо з неї наступним ходом можна отримати програшну позицію (2;0;3;1).

3′. Програшною є позиція початкова позиція (4;0;1;1), бо наступним ходом можна отримати лише одну з двох виграшних позицій (2;2;1;2) і (1;3;1;2).

$$(2;2;1;1) \to (2;0;3;2) \to (0;2;3;1) \to (0;0;5;2)$$
$$\searrow \qquad\qquad \nearrow$$
$$(0;4;1;2) \to (0;1;4;1)$$

Рисунок 6: граф гри для запропонованих в умові задачі II.1 правил гри і другого набору вхідних даних.

Для ігор з повною інформацією[2] з одним переможцем[3] із скінченною кількістю позицій без випадкового втручання[4] такий алгоритм за скінченну кількість кроків дозволяє провести повний аналіз всіх позицій (в тому числі, й початкової).

Наочність і мала кількість ходів інколи роблять можливим проведення аналізу гри й пошуку оптимальної стратегії без формалізації за допомогою графа гри. Наприклад, у розв'язанні задач I.3–4 про сполучення відрізками даних точок координатної площини.

Використання графа гри можна поширити на ігри, результат яких для кожного гравця описується певною *функцією виграшу* на багатоелементну множину чисел, а не є тільки виграшем або програшем. У цьому випадку аналіз позицій передбачає встановлення, який результат матиме гра, якщо вона почнеться з даної позиції, а кожен гравець бажає зробити свій виграш якомога більшим. Але й у цьому випадку аналіз позицій потрібно починати від заключних позицій. Приклад такого узагальнення подано далі у розв'язанні задачі III.6 "Приватні інтереси".

Для багатьох відомих популярних ігор, наприклад, шахів, аналіз графа гри здійснити, практично, неможлививо завдяки неосяжній (для сучасної обчислювальної техніки) кількості позицій. Традиційно, в розв'язаннях *математичних* задач про ігри (Баше, нім[5], фан-тан[6], цзяншицзи[7]):

[2]Гра з повною інформацією — гра, у якій кожен гравець у момент вибору ходу знає, які ходи робили його суперники, і в якій позиції перебуває гра.

[3]У такій грі виграє лише один гравець, а решта гравців програють.

[4]Тобто, коли наступна позиція визначається виключно вибором гравців. Таким чином, не передбачено використання гральних кубиків, карт, рулетки тощо.

[5]Ж.Арсак. Программирование игр и головоломок. — М.: "Наука", 1990, 223 с.

[6]В.А.Вишенський. Гра фан-тан // У світі математики, 1995, том 1, випуск 2, с. 69–74.

[7]В.А.Вишенський. Гра цзяншицзи // У світі математики, 1996, том 2, випуск 1, с. 75–81.

- описується структура графа гри (інколи неявно, навіть, без згадування геометричної ілюстрації);
- висловлюється й доводиться (як правило, методом математичної індукції) гіпотеза про критерій виграшності позиції.

Такий підхід непридатний для даної задачі, враховуючи висунуті технічні умови. Розв'язати дану задачу означає написати (мовою програмування) алгоритм:

- зчитування вхідних даних;
- побудови графа гри, вершини якого у пам'яті ЕОМ зберігається як масиви невід'ємних чисел;
- аналізу графа гри (за описаною раніше схемою) з метою встановлення виграшності чи програшності позицій;
- у разі виграшності початкової позиції встановлення програшних позицій, які можна отримати після 1-го ходу;
- запису відповіді у вихідний файл

для певного *класу ігор*.

Для зручності читачів прокоментуємо подану далі програму. З метою економії оперативної пам'яті ЕОМ запам'ятовують загальну кількість кульок в усіх ямках на початку гри (див. використання ідентифікатора `sum`), а от кількість кульок у останній ямці кожної позиції не зберігають у пам'яті ЕОМ. Під час побудови графа гри:

`i0` та `i1` — відповідно нижня та верхня межі зміни номерів позицій, з яких за один хід утворюються нові позиції (початковій позиції відповідає число 0);

`inew` — кількість вже знайдених позицій (без початкової).

У поданій далі програмі ідентифікатори масивів використано наступним чином:

`l[k]` — `k`-ий елемент послідовності, що задає правила гри і зчитується з 3-го рядка вхідного файлу;

`state[k,i]` — кількість кульок у `k`-ій ямці в `i`-ій позиції;

`first[i]` — справджується, якщо в `i`-ій позиції ходить перший гравець;

`newstate[k]` — кількість кульок у `k`-ій ямці позиції, яку отримано з уже відомих за один хід, але ще не встановлено, чи є ця позиція насправді новою;

`where[0,i]` — кількість позицій, для які можна отримати з `i`-ої за один хід;

`where[k,i]` для `k` в межах від 1 до `where[0,i]` включно — номери всіх позицій, для які можна отримати з `i`-ої за один хід;

`undone[i]` — справджується, якщо на момент виконання частини програми ще не встановлено виграшність чи програшність `i`-ої позиції;

`win[i]` — справджується за умови `undone[i]=false`, якщо встановлено виграшність `i`-ої позиції.

```
const n_max=5; m_max=5; n_states=1000;
var i,i0,i1,inew,k,ks: word; o: text;  km,kn,m,n,sum: byte;
    new:boolean; l: array[1..m_max] of byte;
          newstate: array[1..n_max] of byte;
             state: array[1..n_max,0..n_states] of byte;
undone, win, first: array[0..n_states] of boolean;
             where: array[0..m_max*(n_max-1),
                          0..n_states] of word;
begin{stones}                                {Зчитування даних}
assign(o,'STONES.DAT'); reset(o); readln(o,n,m);
for i:=1 to n do read(o,state[i,0]);  readln(o);
for i:=1 to m do read(o,l[i]);          close(o);    sum:=0;
for i:=1 to n do sum:=sum+state[i,0];  n:=n-1;
first[0]:=true; undone[0]:=true;       {Побудова графа гри}
inew:=0;  i0:=0;  i1:=0;           while  i0 <= i1  do begin
                                   for i:=i0 to i1  do begin
                                   for kn:=1 to n   do begin
km:=1;
new:=state[kn,i]>=l[km];                   while new do begin
{Хід - з ямки k у ямку (kn+1) перекладено l[km] камінців}
for k:=1 to n do newstate[k]   :=state[k   ,i];
                 newstate[kn]  :=state[kn  ,i]-l[km];
if kn<>n then    newstate[kn+1]:=state[kn+1,i]+l[km];

{Чи і позиція новою?}  for ks:=inew downto 1 do
                        if first[ks]=not first[i] then begin
new:=false;
for k:=1 to n do if newstate[k]<>state[k,ks] then     begin
new:=true;      break                                   end;
if not new then break                                   end;
                      {Якще це справді нова позиція...}
                                        if new then begin
inew:=inew+1;  first[inew]:=not first[i];
for k:=1 to n do state[k,inew]:=newstate[k];
where[0,i]:=where[0,i]+1;  where[where[0,i],i]:=inew;

{Чи це кінцева, а тому програшна позиція?...}
   undone[inew]:=false;
for k:=1 to n do if newstate[k]>=l[1] then            begin
```

```
   undone[inew]:=true; break                                  end;
if not undone[inew] then win[inew]:=false                     end

       {Якщо це вже розглянута ks-та позиція...} else begin
where[0, i]:=where[0, i]+1; where[where[0, i], i]:=ks   end;
if km<m then begin km:=km+1; new:=state[kn,i]>=l[km] end
        else new:=false                              end end end;
                          i0:=i1+1;  i1:=inew;              end;
{Аналіз графа гри}
                               while inew>0        do begin
                              for i:=inew downto 0 do begin
if undone[i] or (i=0) then                                 begin

{Чи можливо встановити програшність i-oї позиції?}
for k:=1 to where[0,i] do if
not undone[where[k,i]] and not win[where[k,i]] then   begin
               undone[i]:=false;  win[i]:=true;  break  end;
{Чи можливо встановити виграшність  i-oї позиції?}
                                    if undone[i] then begin
   new:=true;
for k:=1 to where[0,i] do if undone[where[k,i]] then  begin
   new:=false;  break                                    end;
if new then                                             begin
       undone[i]:=false;  win[i]:=false         end end end;
if not undone[i] and (i=inew-1) then inew:=i        end end;
                                  {Запис відповіді}
assign(o,'STONES.RES');  rewrite(o);   if win[0] then begin
     writeln(o,'1');
for i:=1 to where[0,0] do if not win[where[i,0]] then begin
m:=sum;                               for kn:=1 to n do begin
m:=m-state[kn,i];  write  (o,state[kn,i]:4)              end;
                   writeln(o,m:4)                    end end
else writeln(o,'2');  close(o)                             end.
```

Порівняйте дану програму з іншим варіантом розв'язання, в якому використано структуру множин.

```
const np_=1000;  np256= np_ div 256;
      l_=5;  n_=4;     nw_=np_*l_*n_;
type set256=array[0..np256] of set of byte;
var p0,           {Позиції, знайдені: на попередньому етапі}
    p1: array[1..np_] of word;          {на поточному етапі}
     p: array[1..n_] of byte;  {Позиція, в яку зроблено хід}
     l: array[1..l_] of byte;          {Фрагмент правил гри}
   pos: array[1..np_,1..n_] of byte;       {Всі позиції гри}
 known,                 {Номери позицій: виграшність відома}
   win: set256;                                    {виграшні}
    np,                             {Кількість: всіх позицій}
```

```
    mp,
    n0,                    {знайдених на попередньому кроці}
    n1,                           {які визначаються вперше}
a,i,j,k: word;
     n,                      {Кількість ямок, зменшена на 1}
     m: byte;           {Кількість елементів посліловності l}
     o: text;                                    {Файл даних}
    nw: array[0..np_] of word;       {Вказавники на масив w:
         w[k] для k від nw[j-1]+1 до nw[j] включно - номери
         позицій, у які можна зробити хід з позиції номер j}
     w: array[1..nw_] of word;
   yet: boolean;
              {Номер позиції p при визначенні всіх позицій}
function FIND: word;   var yet: boolean; j,k: word;    begin
                k:=0;
repeat inc(k);  j:=0;
repeat inc(j);       yet:=p[j]=pos[k,j]
 until  (n=j) or not yet;
 until (k=np) or     yet;     if yet then find:=k else begin
                  inc(n1);  inc(np);  p1[n1]:=np;  find:=np;
                  for j:=1 to n do pos[np,j]:=p[j]  end end;

                    {Позиція p після ходу (j,i) з позиції k}
procedure STEP (k: word; j,i: byte);    var a: byte;   begin
                           for a:=1 to n do p[a]:=pos[k,a];
     p[j]:=p[j]-l[i];  if j<>n then p[j+1]:=p[j+1]+l[i]  end;
function inset  (j: word;      s: set256): boolean;    begin
         inset:=(j mod 256) in s[j div 256]              end;
                                                       BEGIN
                                      {Зчитування даних}
assign(o,'STONES.DAT');  reset(o);  readln(o,n,m);  dec(n);
for j:=1 to n do read(o,pos[1,j]);  readln(o,a);
for j:=1 to m do read(o,l[j]);       close(o);
assign(o,'STONES.RES');  rewrite(o);
for j:=1 to n do a:=a+pos[1,j];
                          {Визначення всіх можливих позицій}
       nw[0]:=0;  n0:=1;  p0[1]:=1;  n1:=0;  np:=1;
repeat                                for k:=1 to n0 do begin
       nw[p0[k]]:=nw[p0[k]-1];
                          for j:=1 to n do for i:=1 to m do
       if pos[p0[k],j]>=l[i] then                      begin
inc(nw[p0[k]]);  STEP(p0[k],j,i); w[nw[p0[k]]]:=find end end;
       n0:=n1;  n1:=0;  for k:=1 to n0 do p0[k]:=p1[k];
until  n0=0;                       for j:=0 to np256 do begin
                         known[j]:=[];  win[j]:=[0..255] end;
{Аналіз графа гри: кінцеві позиції програшні}
mp:=0;                                for k:=1 to np do begin
```

```
       j:=0;  repeat inc(j) until (pos[k,j]>=l[1]) or (n=j);
                                  if pos[k,j]< l[1] then begin
inc(mp);          include(known[k div 256],k mod 256);
                  exclude(  win[k div 256],k mod 256) end end;
{Аналіз графа гри від кінцевих позицій}
repeat for k:=np downto 1 do if not inset(k,known)then begin
            j:=nw[k-1];  yet:=true;
repeat inc(j);
       if inset(w[j],known) then yet:=yet and inset(w[j],win)
                            else yet:=false;
until (j=nw[k]) or inset(w[j],known) and not inset(w[j],win);
       if yet then                                        begin
       inc(mp);  include(known[k div 256],k mod 256);
                 exclude(  win[k div 256],k mod 256)     end
else if inset(w[j],known) and not inset (w[j],win)then begin
       inc(mp);  include(known[k div 256],k mod 256) end end
until mp=np;                                    {Запис відповіді}
if not (1 in win[0]) then writeln(o,'2')    else       begin
                          writeln(o,'1');
for j:=1 to nw[1] do if not inset(w[j],win) then       begin
                           k:=a;      for i:=1 to n do begin
write(o,pos[w[j],i],' ');  k:=k-pos[w[j],i]              end;
                  writeln(o,k)                       end end;
                                                  close(o) END.
```

Подані програми можна дещо удосконалити:

- використовувати замість двовимірних масивів одновимірні для того, щоб обмежувати не кількість ямок та кількість позицій, лише їх добуток;
- використати динамічний розподіл пам'яті.

Після здійсненого аналізу позицій легко запрограмувати дії гравця, що прагне виграшу на основі цього аналізу.

2. Десятковий дріб. Ціла частина шуканого дробу знаходиться як частка від ділення a на b. Цифри після десяткової крапки знаходяться традиційним діленням "у стовпчик" (з остачею). При цьому кількість цифр між десятковою крапкою (комою) та першою цифрою періода дорівнює найбільшому з чисел n_2 та n_5, які є відповідно степенями 2 та 5 у канонічному розкладі знаменника b на прості множники. Повне розв'язання задачі вимагає реалізації "довгої арифметики". Процедуру `divide` (ділення з остачею натуральних чисел, поданих масивами цифр) запозичено з розв'язання задачі I.1 "Система числення" без будь-яких змін.

```
const n_max=78;    type  carray=array[0..n_max] of shortint;
var  a,b,q,r,r0,b0:carray; code:integer; o:text; ans:string;
n2,n5,n0,i,j,k,lb0,lr0,la,lb,lr,lq: byte;   period: boolean;
```

```
procedure divide(a,b: carray; la,lb: byte);     label NEXTi;
{Знаходження частки (lq цифр у масиві q) і остачі
(lr цифр у масиві r) від ділення одного числа
(la цифр у масиві a) на інше (lb цифр у масиві b)}    begin
                                          if la<lb then begin
for j:=1 to la do r[j]:=a[j]; lr:=la;  lq:=1;  q[1]:=0  end
                                            else begin
for j:=1 to lb do r[j]:=a[j];          lq:=0; r[0]:=0; i:=0;
                           {Знаходження k - цифри частки q}
NEXTi: i:=i+1;  k:=0;               while r[0]>=0 do  begin
                                  for j:=lb downto 1 do begin
       r[j]:=r[j]-b[j];                 if r[j]<0 then begin
       r[j]:=r[j]+10;  r[j-1]:=r[j-1]-1             end end;
                                  if r[0]>=0 then k:=k+1  end;
                                  for j:=lb downto 1 do begin
       r[j]:=r[j]+b[j];                 if r[j]>9 then begin
       r[j]:=r[j]-10;  r[j-1]:=r[j-1]+1             end end;
if (k>0) or (i>1) then begin lq:=lq+1; q[lq]:=k end;
if    i+lb<=la    then                                  begin
for j:=1 to lb do r[j-1]:=r[j]; r[lb]:=a[i+lb];  goto NEXTi
                                                      end;
   lr:=lb;  while (r[lb-lr+1]=0) and (lr>1) do lr:=lr-1;
if lr< lb   then for j:=1 to lr do r[j]:=r[j+lb-lr]     end;
if lq= 0    then begin lq:=1; q[1]:=0 end               end;
begin{fraction}                          {Зчитування даних}
assign(o,'FRACTION.DAT');  reset(o);
readln(o,ans);  la:=length(ans);  for j:=1 to la do
     val(ans[j],a[j],code);
readln(o,ans);  lb:=length(ans);  for j:=1 to lb do begin
     val(ans[j],b[j],code);        b0[j]:=b[j]        end;
close(o);       lb0:=lb;  n0:=1;

{Знаходження n2 - степеня 2 у розкладі a на прості множники}
n2:=0;  r0[1]:=2;           lr:=1;      r[1]:=0;
                      while (lr =1) and (r[1] =0) do   begin
divide(b,r0,lb,n0);     if (lr =1) and (r[1] =0) then begin
n2:=n2+1;  lb:=lq;   for j:=1 to lb do b[j]:= q[j]  end end;
           lb:=lb0;  for j:=1 to lb do b[j]:=b0[j];

{Знаходження n5 - степеня 5 у розкладі a на прості множники}
n5:=0;  r0[1]:=5;           lr:=1;      r[1]:=0;
                      while (lr =1) and (r[1] =0) do   begin
divide(b,r0,lb,n0);     if (lr =1) and (r[1] =0) then begin
n5:=n5+1;  lb:=lq;   for j:=1 to lb do b[j]:= q[j]  end end;
           lb:=lb0;  for j:=1 to lb do b[j]:=b0[j];
```

```
                {Знаходження n0 - кількості цифр до періода}
if n2<n5 then n0:=n5 else n0:=n2;
assign(o,'FRACTION.RES');  rewrite(o);

                          {Знаходження і запис цілої частини}
divide(a,b,la,lb);
for j:=1 to lq do write (o,q[j]);  write(o,'.');
    {Знаходження і запис цифр після крапки, але до періода}
                                  for lr0:=1 to n0 do begin
for j:=1 to lr do a[j]:=r[j];  la:=lr+1;  a[la]:=0;
divide(a,b,la,lb);  write(o,q[1])                       end;

{Знаходження періоду}      if (lr>1) or (r[1]>0) then begin
write(o,'(');  lr0:=lr;  for j:=1 to lr do r0[j]:=r[j];
period:=true;                           while period do begin
for j:=1 to lr do a[j]:=r[j];  la:=lr+1;  a[la]:=0;
divide(a,b,la,lb);   write(o,q[1]);    if lr=lr0 then begin
                     j:=0; while period and (j<lr) do begin
                     j:=j+1;     period:=r[j]=r0[j]     end;
                                 period:=not period end end;
write(o,')')                                           end;
                                              close(o)  end.
```

Задачу можна дещо ускладнити, не вимагаючи взаємної простоти чисел a і b. Така задача зводиться до розглянутої, якщо:

- використавши алгоритм Евкліда, знайти НСД(a, b) — найбільший спільний дільник чисельника і знаменника дробу;
- поділити a і b на НСД(a, b).

3. Офіцери. Задачу запозичено із книги Л.М. Лоповка "Збірник математичних задач логічного характеру", Київ, "Радянська школа", 1972, 151 с. Для успішного розв'язання задачі необхідно звернути увагу на наступне.

Перехід від літер A, B, C, D, E, F до чисел 1, 2, 3, 4, 5, 6 відповідно під час опрацювання вхідного файлу для того, щоб полегшити написання алгоритму перебору всіх можливих варіантів розподілу серед офіцерів військових звань та родів військ.

Аналіз умови задачі, перетворення її на еквівалентну систему тверджень. Згідно з умовою задачі, серед офіцерів є:

- один капітан-артилерист (нехай йому відповідає число i_1 в межах від 1 до 6 включно);
- два майори, серед яких щонайменше один танкіст (нехай йому відповідає число i_2), а другий — з поки що невідомого роду військ (нехай йому відповідає число i_3)

$$i_2 \neq i_1 \wedge i_3 \neq i_1 \wedge i_3 \neq i_2; \tag{1}$$

- три полковники, яким відповідають числа, відмінні від i_1, i_2, i_3;
- один зв'язківець, якому відповідає число j

$$j \neq i_1 \wedge j \neq i_2 \wedge j \neq a_5 \wedge j \neq a_6 \wedge j \neq a_7 \wedge j \neq a_8; \tag{2}$$

- два артилеристи, одному з яких відповідає число i_1, а другому — a_8

$$i_1 \neq a_7 \wedge i_1 \neq a_8 \wedge i_2 \neq a_8; \tag{3}$$

- три танкіста, яким відповідають числа, відмінні від i_1, j, a_8.

Серед танкістів, щонайменше, один майор, якому відповідає число i_2:

$$i_2 \neq a_7, \tag{4}$$

і один полковник, якому відповідає число k, відмінне від i_1, i_2, i_3, j, a_7, a_8.

$$i_3 = j \vee i_3 = a_7 \vee i_3 = a_8. \tag{5}$$

Умову "a_1 зобов'язаний першим вітати a_2" можна записати як

$$a_1 = i_1 \vee \begin{cases} a_1 = i_2 \vee a_1 = i_3 \\ a_2 \neq i_1 \wedge a_2 \neq i_2 \wedge a_2 \neq i_3 \end{cases}. \tag{6}$$

Умову "a_3 і a_4 — в одному званні" можна записати як

$$\begin{cases} a_3 = i_2 \\ a_4 = i_3 \end{cases} \vee \begin{cases} a_3 = i_3 \\ a_4 = i_2 \end{cases} \vee \left(\begin{cases} a_3 \neq i_1 \\ a_3 \neq i_2 \\ a_3 \neq i_3 \end{cases} \wedge \begin{cases} a_4 \neq i_1 \\ a_4 \neq i_2 \\ a_4 \neq i_3 \end{cases} \right) \tag{7}$$

— два офіцери мають однакове звання, якщо вони майори або полковники. Умову "a_5 і a_6 — в одному роді військ" можна записати як

$$\begin{cases} a_5 = i_1 \\ a_6 = a_8 \end{cases} \vee \begin{cases} a_5 = a_8 \\ a_6 = i_1 \end{cases} \vee \left(\begin{cases} a_5 \neq i_1 \\ a_5 \neq j \\ a_5 \neq a_8 \end{cases} \wedge \begin{cases} a_6 \neq i_1 \\ a_6 \neq j \\ a_6 \neq a_8 \end{cases} \right) \tag{8}$$

— два офіцери в одному роді військ, якщо вони артилеристи або танкісти. Умову "полковник, майор і a_7 — танкісти" можна записати як

$$a_7 \neq i_1 \wedge a_7 \neq j \wedge a_7 \neq a_8. \tag{9}$$

Для узгодження з даними вхідного файлу OFFICERS.DAT необхідньо й достатньо справдження кожного з тверджень (1–9). У поданій далі програмі таку перевірку здійснено з використанням складних умовних операторів if ... then if ... then ... на початку відповідних циклів. Мета — зменшити кількість виконуваних дій.

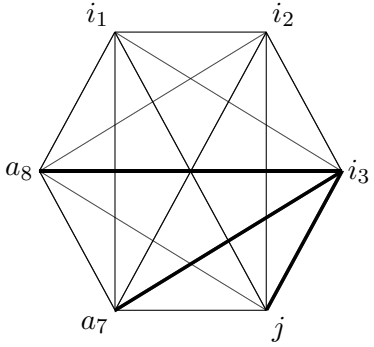


Рисунок 7: графічне подання нерівностей між числами i_1, i_2, i_3, j, a_7, a_8 (тонкими лініями) та можливих рівностей (широкими лініями). Виконання однієї з цих можливих рівностей призводить до існування k в межах від 1 до 6, відмінного від i_1, i_2, i_3, j, a_7, a_8.

```
var  s: char; a:array[1..8] of byte; b: array[1..6] of char;
     o: text;      i1,i2,i3,j,k: byte;                 begin
b[1]:='A';b[2]:='B';b[3]:='C';b[4]:='D';b[5]:='E';b[6]:='F';
assign(o,'OFFICERS.DAT'); reset(o); for j :=1 to 8 do begin
read(o,s);      for i1:=1 to 6 do if s=b[i1] then     begin
                                      a[j]:=i1; break  end end;
close(o);   assign(o,'OFFICERS.RES');   rewrite(o);
if a[7]<>a[8]  then
for i1:=1 to 6 do if (i1<>a[7]) and (i1<>a[8]) then
for i2:=1 to 6 do if (i2<>i1)   and (i2<>a[7])
                                and (i2<>a[8]) then
for i3:=1 to 6 do if (i3<>i1)   and (i3<>i2)   then
if (a[1] =i1) or ((a[1] =i2) or (a[1] =i3)) and
   (a[2]<>i1) and (a[2]<>i2) and (a[2]<>i3)    then
if (a[3] =i2) and (a[4] =i3) or
   (a[3] =i3) and (a[4]=i2)  or
   (a[3]<>i1) and (a[3]<>i2) and (a[3]<>i3) and
   (a[4]<>i1) and (a[4]<>i2) and (a[4]<>i3) then
for j:=1 to 6 do if (i3=j) or (i3=a[7]) or (i3=a[8]) then
if (j<>i1) and (j<>i2) and (j<>a[5]) and (j<>a[6])
                       and (j<>a[7]) and (j<>a[8])   then
if (a[5] =i1) and (a[6]=a[8]) or (a[5] =a[8]) and (a[6]=i1)
or (a[5]<>i1) and (a[5]<>j)  and (a[5]<>a[8]) and
   (a[6]<>i1) and (a[6]<>j)  and (a[6]<>a[8]) then
for k:=1 to 6 do                                      begin
             write(o,b[k],' ');
if k=i1 then write(o,'a-')
        else if (k=i2) or (k=i3)  then write(o,'¬-')
```

```
                                  else write(o,'¯-');
if k=j then write(o,'§')
       else if (k=i1) or (k=a[8]) then write(o,' ')
                                  else write(o,'â');
if k<6 then write(o,', ') else writeln(o,'.')           end;
close(o)                                                end.
```

Відверто кажучи, автор не сподівався на аналогічне оптимальне розв'язання з боку учасників олімпіади. Тим більше, що важко тестувати у присутності учасників задачі, ефективність розв'язання яких (на поданих тестах) вимірюється мікросекундами. Ймовірнішим видається (подано у порядку наростання складності програмування):

- перебір $(3^6)^2$=531441 варіантів розташування 3 родів військ і 3 звань серед 6 офіцерів з наступною перевіркою узгодженості з умовою задачі;
- перебір $(6!/(1!\cdot 2!\cdot 3!))^2$=3600 варіантів розташування з повторенням 3 родів військ і 3 звань серед 6 офіцерів з наступною перевіркою узгодженості з умовою задачі;
- перебір варіантів розташування з повторенням 3 родів військ і 3 звань серед 6 офіцерів з оптимізацією перебору за рахунок перевірки узгодженості з умовою задачі.

Останній підхід реалізовано у такій програмі.

```
const abc:string[6]='ABCDEF';  title: string[3] ='кмп';
                               force: string[3] ='зат';
var o: text;   a: string[8];                    {Вхідні дані}
    f,              {f[j] - номер в алфавітному порядку a[j]}
    h: array[1..12] of byte;                  {Масив гіпотез:
h[2*j-1]=1,2,3 -        капітан, майор, полковник відповідно,
h[2*j]  =1,2,3 - танкіст, артилерист, зв'язківець відповідно}
  i,                                          {Рівень гіпотези}
j,k,                                      {Лічильники персонажів}
  l,                   {Номер персонажа в алфавітному порядку}
  m: byte;                  {=0 для роду військ, 1 для звання}
  yes: boolean;  label NEXT,NEXTi, NEXTh;              BEGIN
assign(o,'OFFICERS.DAT');    reset(o);  read(o,a);  close(o);
assign(o,'OFFICERS.RES');  rewrite(o);
for j:=1 to 8 do f[j]:=pos(a[j],abc);
           i:=0;
NEXTi: inc(i);  h[i]:=0;
NEXTh:      inc(h[i]);  if h[i]>3 then                  begin
       dec(i);          if i=0    then begin
                         close(o);  halt end; goto NEXTh end;
if (i<3) and not (abc[1] in [a[7],a[8]]) then goto NEXT;
```

```
l:=(i+1) div 2;  m:=i mod 2;  k:=0;  case m of  {Чисельність}
0: for j:=1 to l do if h[i]=h[2*j]   then inc(k);
1: for j:=1 to l do if h[i]=h[2*j-1] then inc(k) end;
if k>h[i] then goto NEXTh;                  case m of 1: begin
                           {Звання: a[1] вітаї a[2]}
if (abc[l] in [a[1],a[2]]) and (f[1]<=l) and (f[2]<=l) then
if h[2*f[1]-1]>=h[2*f[2]-1] then goto NEXTh;
                                  {a[3] і a[4] в одному званні}
if (abc[l] in [a[3],a[4]]) and (f[3]<=l) and (f[4]<=l) then
if h[2*f[3]-1]<>h[2*f[4]-1] then goto NEXTh              end;
                                                      0: begin
{Рід військ:   a[5],a[6] в одному роді військ}
if (abc[l] in [a[5],a[6]]) and (f[5]<=l) and (f[6]<=l) then
if h[2*f[5]]<>h[2*f[6]] then goto NEXTh;
                                                  {a[7] танкіст}
if (abc[l]=a[7]) and (h[2*l]<>3) then goto NEXTh;
                                               {a[8] артилерист}
if (abc[l]=a[8]) and (h[2*l]<>2) then goto NEXTh     end end;

NEXT: if i<12 then goto NEXTi;
           j:=0;           {Полковник, майор і a[7] танкісти}
repeat inc(j); yes:=(h[2*j-1]=3) and (h[2*j]=3) and (j<>f[7])
until (j=6) or yes;  if not yes then goto NEXTh;
           j:=0;
repeat inc(j); yes:=(h[2*j-1]=2) and (h[2*j]=3) and (j<>f[7])
until (j=6) or yes;  if not yes then goto NEXTh;
j:=0;                          {a[8] і капітан - артилеристи}
repeat inc(j); yes:=(h[2*j-1]=1) and (h[2*j]=2) and (j<>f[8])
until (j=6) or yes;               if not yes then goto NEXTh;
                       {Відповідь}     for j:=1 to 6 do begin
write(o,abc[j],' ',title[h[2*j-1]],'-',force[h[2*j]]);
if j<6 then write(o,', ') else writeln(o,'.')            end;
                                                goto NEXTh END.
```

3 III етап

3.1 Умови проведення

Олімпіаду проведено у 2 (два) практичних тури тривалістю 5 (п'ять) астрономічних годин кожний.

3.2 Завдання I туру

1. Хімія, 20 балів. Файл CHEMIE.DAT містить формули субстратів хімічної реакції, розділені знаком + і записані ліворуч від знаку =, і продуктів цієї самої реакції, розділені знаком + і записані праворуч від знаку =. Кількість всіх

речовин (субстратів і продуктів) не перевищує 10. Позначення всіх хімічних елементів (1-ий рядок файлу CHEMIE.DAT) починаються з великих літер латиниці. Більшу за 1 кількість атомів хімічного елемента у сполуці вказують знизу праворуч (2-ий рядок CHEMIE.DAT). В одній формулі позначення одного й того самого елемента може зустрічатися не один раз. Якщо сполука містить кілька однакових груп атомів, то у хімічній формулі відповідний запис виділяють круглими дужками, а кількість повторень вказується праворуч знизу від правої круглої дужки. Прогалини у першому рядку CHEMIE.DAT зустрічаються лише над числами другого рядка, але перший рядок не закінчується прогалинами.

Створіть програму CHEMIE.*, яка:

- визначити *найменші натуральні* коефіцієнти, які не перевищують 30, і які потрібно написати перед формулою кожної речовини (у результаті хімічної реакції кількість атомів будь-якого хімічного елемента не змінюється);
- запише у файл CHEMIE.RES правильно складене рівняння хімічної реакції (вхідні дані гарантують єдиність).

CHEMIE.DAT Приклади файлів CHEMIE.RES

```
Ba(OH) +HCl=BaCl +H O                Ba(OH) +2HCl=BaCl +2H O
      2         2  2                       2          2   2
```

2. Многогранник, 20 балів. Перший рядок вхідного файлу SOLID.DAT містить натуральне число n — кількість вершин многогранника, поверхню якого можна неперервно і взаємно однозначно відобразити на сферу. Для j в межах від 1 до n включно $(j+1)$-ий рядок цього самого файлу містить у порядку зростання номери вершин, які з'єднані з j-тою вершиною ребрами. Відомо, що: вершин не більше, ніж 2000; ребер не більше, ніж 4000; лише в одній грані кількість ребер перевищує 8.

Створіть програму SOLID.*, яка проведе аналіз даних:

- єдиний рядок вихідного файлу SOLID.RES міститиме у вказаному порядку невід'ємні натуральні числа n_3, n_4, ... n_m, — кількості 3-, 4-, ... m-кутних граней відповідно, де m — найбільша кількість вершин однієї грані многокутника;
- файл l.RES міститиме n_l рядків;
- кожний рядок вихідного файлу l.RES міститиме натуральні числа i_1, i_2, ... , i_l — перелік номерів вершин l-кутної грані у порядку обходу її за периметром, причому різним граням відповідатимуть різні рядки, а із збільшенням номера рядка величина $i_1n^2 + i_2n + i_3$ зростатиме.

Приклади файлів для тетраедра

SOLID.DAT	SOLID.RES	3.RES
4 2 3 4 1 3 4 1 2 4 1 2 3	4	1 2 3 1 2 4 1 3 4 2 3 4

3. Приватні інтереси, 20 балів. На клітинках прямокутної таблиці, що має m рядків та n стовпчиків, розкладено монети. Двоє гравців по черзі рухають пішак або праворуч, або вгору на одне поле за один хід. Гру розпочинає перший гравець з крайнього нижнього лівого поля. Гра припиняється після досягнення пішаком першого (якщо рахувати згори вниз) рядка чи останнього (якщо рахувати зліва направо) стовпчика таблиці. Кожен гравець:

а) не має можливості спілкуватися з суперником;

б) знає розподіл монет перед початком гри і розташування пішака у будь-який момент гри;

в) забирає монети з клітин, на які ставить пішак;

г) ходить праворуч лише тоді, коли альтернатива (хід вгору) приносить менший остаточний зиск;

д) намагається отримати найбільшу суму грошей;

е) знає і враховує намагання суперника;

є) знає, що суперник діє аналогічно.

Створіть програму PRIVAT.*, яка передбачить результат гри та рух пішака.

Перший рядок вхідного файлу PRIVAT.DAT містить у вказаному порядку натуральні числа m та n. Для j у межах від 1 до m включно $(j+1)$-ий рядок цього самого файлу містить невід'ємні цілі числа — сукупні вартості монет у клітинках j-го рядка таблиці у тому самому порядку, як вони (клітинки цього рядка) розташовані у таблиці зліва направо наліво. Кількість всіх чисел файлу PRIVAT.DAT не перевищує 12346, максимальний виграш не перевищує 65432 (золотих).

Перший рядок вихідного файлу PRIVAT.RES має містити два невід'ємних цілих числа — виграші (кількості зібраних монет) першого та другого гравців відповідно. Другий рядок цього самого файлу має містити послідовність символів, що описують напрям ходів пішака (від першого до останнього) для досягнення цього результату: u — вгору (від англійського up), r — праворуч (від англійського right). Наприклад,

PRIVAT.DAT PRIVAT.RES

```
4 4
1 1 2 0                                    19 11
3 7 8 9                                    uurrr
3 0 7 1
0 3 3 1
```

3.3 Завдання II туру

4. Вечірка, 24 бали. Створіть програму PARTY.*, яка допоможе встановити, як розсілися за круглим столом студенти (обличчям до центру столу) й фах кожного з учасників вечірки.

Кожен рядок вхідного файлу PARTY.DAT складається з 3-х частин, розділених прогалинами:

1) ім'я або фах учасника вечірки;

2) символ для позначення його розташування відносно того учасника вечірки, ім'я або фах якого вказані наприкінці твердження:
   < — читати "ліворуч від" (якщо дивитися в центр столу з місця учасника вечірки, вказаного у частині 1),
   > — читати "праворуч від",
   | — читати "навпроти";

3) ім'я або фах учасника вечірки.

Всі рядки, крім останнього, закінчуються комою, а останній — крапкою. Всі літери у словах файлу PARTY.DAT малі, крім перших літер імен. Апостроф не використвно. Кількість учасників вечірки, кількість літер у кожному слові та кількість рядків PARTY.DAT не перевищують 25. Імена та фахи всіх учасників перераховано у файлі PARTY.DAT.

Кожний рядок файлу PARTY.RES має містити по одному, відмінному від інших, варіанту розташування студентів за столом, який не суперечить умові задачі. Потрібно вказати імена учасників вечірки в порядку обходу столу проти руху годинникової стрілки, якщо дивитися згори: кожний наступний учасник сидить праворуч від попереднього. Починати перелік потрібно з імені студента, яке буде першим у списку імен, впорядкованих у алфавітному порядку української абетки: АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ. Після кожного імені (через прогалину) потрібно вказати в дужках фах студента. Ім'я кожного наступного учасника потрібно відділити комою і прогалиною, а в кінці рядка поставити крапку.

Наприклад, якщо файл PARTY.DAT має вигляд

```
філолог | Кирило,
```

```
Кирило < історик,
біолог > Віталій,
Федір > Євген,
хімік > Федір.
```

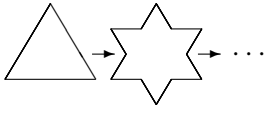
то файл PARTY.RES має містити запис

```
Віталій (хімік), Кирило (біолог), євген (історик), Федір (філолог).
```

Всі слова файлів PARTY.DAT і PARTY.RES — у називному відмінку. Всі імена та фахи — різні

5. Багатокутник, 12 балів. Створіть програму POLYGON.*, яка:

- зображує сторони рівностороннього трикутника (білі на чорному тлі, одна сторона — горизонтальна і розташована знизу);
- після натискання клавіші Enter кожний відрізок попереднього малюнка ділиться на 3 рівних частини, середня частина замінюється ламаною з двох ланок тієї самого довжини, причому нова вершина ламаної лежить зовні області, обмеженої ламаною попереднього зображення. Зображення має максимальні розміри (на весь екран).



Після 6-го натискання клавіші Enter робота програми припиняється.

6. Корінь квадратний, 34 бали. Щоб добути квадратний корінь з десяткового дробу (наприклад, 65432.109), його запис розбивають праворуч і ліворуч від десяткової крапки на *грані*, що містять по 2 цифри. У крайній лівій грані може виявитися і одна цифра (6).

$\sqrt{}$ 65432.109 = 255.7970...

4
4*5* _ 254
5 225
50*5* _ 2932
5 2525
510*7* _ 40710
7 35749
5114*9* _ 496190
9 460341
51158*7* _ 3584900
7 3581109
5115940 379100
0

Перша цифра кореня (2) знаходиться як найбільша з усіх тих цифр, квадрати яких не перевищують першої грані (6). Потім від першої грані віднімають квадрат першої цифри кореня (4), і до різниці приписують праворуч (*зносять*) наступну грань (54). Ліворуч від отриманого числа (254) пишуть подвоєну першу цифру кореня (4), а за наступну цифру (*5*) беруть найбільшу з тих, добуток яких на число, утворене приписуванням їх до подвоєної першої цифри праворуч, ($45 \cdot 5 = 225$) не перевищує результату приписування до попередньої різниці відповідної грані (254). Цю цифру (5) записують після першої цифри кореня і т.д. Десяткова крапка у корені ставиться тоді, коли зноситься перша грань праворуч від десяткової крапки числа.

Створіть програму ROOT.*, яка вираховує корінь квадратний додатного дійсного числа. Єдиний рядок вхідного файлу ROOT.DAT містить у вказаному порядку скінченний або періодичний додатний десятковий дріб (всього — до 1000 символів разом з *десятковою крапкою*) a та натуральне число n, яке не перевищує 100. єдиний рядок файлу ROOT.RES має містити подання $\sqrt{a}$ скінченим чи періодичним десятковим дробом, якщо це можливо, інакше — корінь квадратний числа a з недостачею з точністю до n цифр після десяткової крапки.

ROOT.DAT		ROOT.RES
25	6	5
0.69(4)	5	0.8(3)
65432.109	4	255.7970

3.4 Вказівки до перевірки

1. Хімія. За успішне проходження кожного тестів 1–13 присуджують по 1 балу, за тест 14 — 7 балів. Перевіряється правильність відповіді, якщо:

- всі коефіцієнти дорівнюють 1, тобто вихідний файл — копія вхідного — тести 1–4;
- всі сполуки утворено одним елементом — тест 7;
- субстратів більше одного — тести 1, 2, 5, 6, 9, 11, 14;
- продуктів більше одного — тести 3, 4, 8, 10, 11–14;
- є індекси знизу — тести 1–14;
- формули містять позначення групи в дужках — тести 4, 8, 10, 11, 14;
- формули містять позначення групи в дужках, до якої входить кілька атомів одного й того самого елемента, на що вказує індекс знизу — тести 8, 11, 14;
- кількість субстратів більша за 2 — тест 12;

- кількість продуктів більша за 2 — тести 13, 14;
- формули містять кілька позначень одного й того ж елемента — тест 10;
- неможливо встановити коефіцієнти рівняння лише на підставі аналізу кількості атомів серед субстратів і продуктів *для кожного елемента окремо*, тобто необхідно провести перебір — тест 14.

1.1 Fe+S=FeS

1.2 $S+O_2=SO_2$

1.3 $SrCO_3=SrO+CO_2$

1.4 $Hg(OH)_2=HgO+H_2O$

1.5 $H_2+Cl_2=2HCl$

1.6 $5S+2P=P_2S_5$

1.7 $2O_3=3O_2$

1.8 $Fe_2(SO_4)_3=Fe_2O_3+3SO_3$

1.9 $Mg_2Si+4HCl=2MgCl_2+SiH_4$

1.10 $Al_2(OH)_4SO_4=2H_2O+Al_2O_2SO_4$

1.11 $Ca(HCO_3)_2+Ca(OH)_2=2CaCO_3+2H_2O$

1.12 $2Sn(OH)_2+O_2+2H_2O=2Sn(OH)_4$

1.13 $4HNO_3=4NO_2+2H_2O+O_2$

1.14 $8Al+30HNO_3=8Al(NO_3)_3+3N_2O+15H_2O$

Зауважимо, що всі подані рівняння (подано лише вміст файлу CHEMIE.RES) описуються хімічні реакції, які насправді відбуваються за певних умов — див. А.Т.Пилипенко, В.Я.Починок, И.П.Середа, Ф.Д.Шевченко. Справочник по элементарной химии (под общей редакцией академика АН УССР А.Т.Пилипенко) / Киев, "Наукова думка", 1977, 542 с.

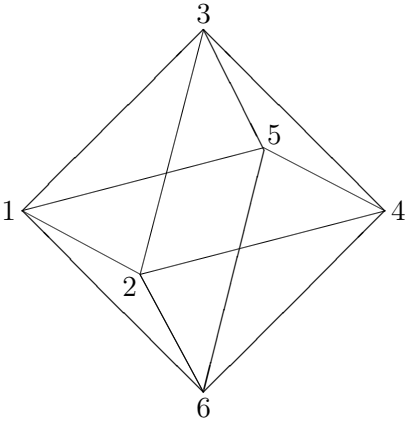
2. Многогранник. За тести 1–7 присуджується по 2 бали, за тест 8 — 6 балів. Перевіряють правильність відповіді, якщо:

- всі грані — трикутники (тести 1, 6);
- всі грані — чотирикутники (тест 2);

- всі грані — п'ятикутники (тест 3);
- грані мають різну кількість вершин (тести 4–6, 8);
- є вершини, до яких сходиться більше, ніж 3 ребра (тести 5–8);
- до вершин сходиться різна кількість ребер (тести 7–8);
- кількість вершин многогранника настільки велика, що неможливо розглянути матрицю суміжності й потрібно раціонально використати оперативну пам'ять (тест 8).

У описі кожного тесту подано назву многогранника, деякі його властивості, його паралельну проекцію на площину (видимі ребра виділено лініями більшої товщини), на якій вказано номери вершин, вхідний та вихідні файли. Многогранники тестів 1–3 називаються правильними, тестів 1–6 — однорідними (про класифікацію та технологію виготовлення моделей багатогранників див., наприклад, М.Венninджер. Модели многогранников. — М.: "Мир", 1974, 236 с.).

2.1. Восьмигранник (октаедр), у якого всі 8 граней — правильні трикутники. До кожної з 6 вершин сходиться 4 ребра (грані). Восьмигранник має 12 ребер. Вершини такого октаедра є центрами граней деякого куба. Восьмигранник утворюється "склеюванням основ" двох правильних 4-кутних пірамід.



SOLID.DAT

```
6
2 3 5 6
1 3 4 6
1 2 4 5
2 3 5 6
1 3 4 6
1 2 4 5
```

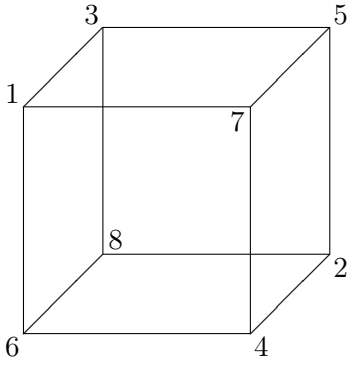
SOLID.RES

```
8
```

3.RES

```
1 2 3
1 2 6
1 3 5
1 5 6
2 3 4
2 4 6
3 4 5
4 5 6
```

2.2. Куб — шестигранник (гексаедр), у якого всі 6 граней — квадрати. До кожної з 8 вершин сходиться 3 ребра (грані). Куб має 12 ребер. Вершини куба є центрами граней деякого октаедра.



SOLID.DAT

```
8
3 6 7
4 5 8
1 5 8
2 6 7
2 3 7
1 4 8
1 4 5
2 3 6
```

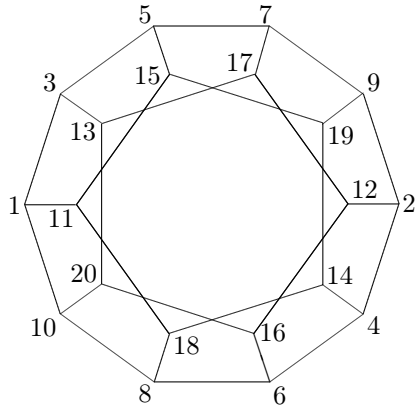
SOLID.RES

```
0 6
```

4.RES

```
1 3 5 7
1 3 8 6
1 6 4 7
2 4 6 8
2 4 7 5
2 5 3 8
```

2.3. Дванадцятигранник (додекаедр), у якого всі 12 граней — правильні п'ятикутники. До кожної з 20 вершин сходиться 3 ребра (грані).



SOLID.DAT

```
20
 3 10 11
 4  9 12
 1  5 13
 2  6 14
 3  7 15
 4  8 16
 5  9 17
 6 10 18
 2  7 19
 1  8 20
 1 15 18
 2 16 17
 3 17 20
 4 18 19
 5 11 19
 6 12 20
 7 12 13
 8 11 14
 9 14 15
10 13 16
```

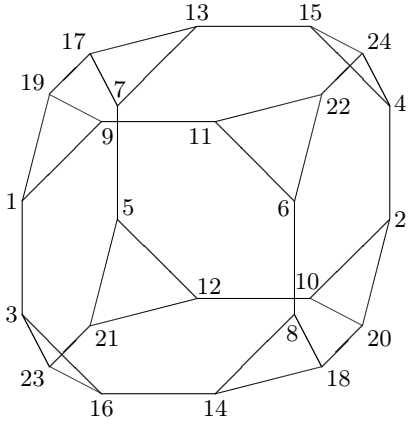
SOLID.RES

```
0 0 12
```

5.RES

```
 1  3  5 15 11
 1  3 13 20 10
 1 10  8 18 11
 2  4  6 16 12
 2  4 14 19  9
 2  9  7 17 12
 3  5  7 17 13
 4  6  8 18 14
 5  7  9 19 15
 6  8 10 20 16
11 15 19 14 18
12 16 20 13 17
```

2.4. Зрізаний куб — 14-гранник, у якого 8 граней — правильні 3-кутники, 6 граней — правильні 8-кутники. До кожної з 24 вершин сходиться 3 ребра (грані).



SOLID.DAT

```
24
 3  9 19
 4 10 20
 1 16 23
 2 15 24
 7 12 21
 8 11 22
 5 13 17
 6 14 18
 1 11 19
 2 12 20
 6  9 22
 5 10 21
 7 15 17
 8 16 18
 4 13 24
 3 14 23
 7 13 19
 8 14 20
 1  9 17
 2 10 18
 5 12 23
 6 11 24
 3 16 21
 4 15 22
```

SOLID.RES

```
8 0 0 0 0 6
```

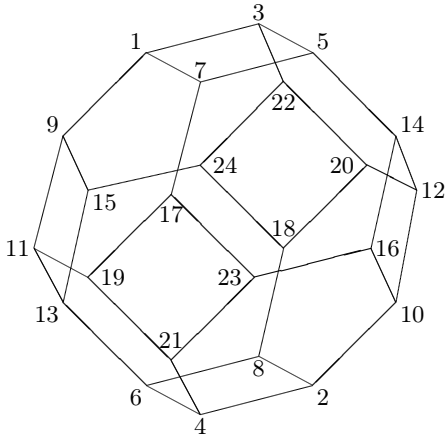
3.RES

```
 1  9 19
 2 10 20
 3 16 23
 4 15 24
 5 12 21
 6 11 22
 7 13 17
 8 14 18
```

8.RES

```
 1  3 16 14  8  6 11  9
 1  3 23 21  5  7 17 19
 2  4 15 13  7  5 12 10
 2  4 24 22  6  8 18 20
 9 11 22 24 15 13 17 19
10 12 21 23 16 14 18 20
```

2.5. Зрізаний октаедр — 14-гранник, у якого 6 граней — квадрати, 8 граней — правильні 6-кутники. До кожної з 24 вершин сходиться 3 ребра (грані).



SOLID.DAT

```
24
 3  7  9
 4  8 10
 1  5 22
 2  6 21
 3  7 14
 4  8 13
 1  5 17
 2  6 18
 1 11 15
 2 12 16
 9 13 19
10 14 20
 6 11 15
 5 12 16
 9 13 24
10 14 23
 7 19 23
 8 20 24
11 17 21
12 18 22
 4 19 23
 3 20 24
16 17 21
15 18 22
```

SOLID.RES

```
0 6 0 8
```

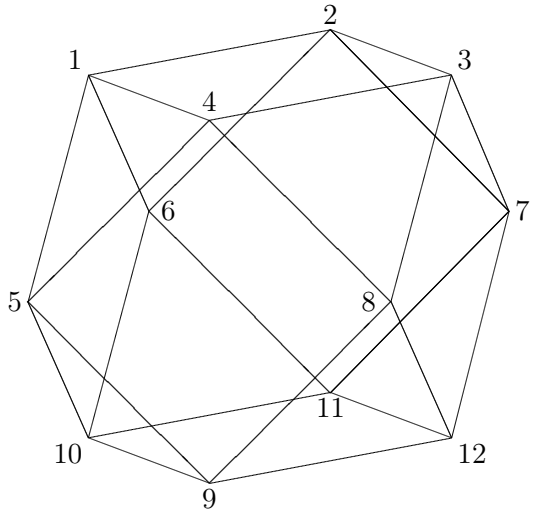
4.RES

```
 1  3  5  7
 2  4  6  8
 9 11 13 15
10 12 14 16
17 19 21 23
18 20 22 24
```

6.RES

```
1 3 22 24 15  9
1 7 17 19 11  9
2 4 21 23 16 10
2 8 18 20 12 10
3 5 14 12 20 22
4 6 13 11 19 21
5 7 17 23 16 14
6 8 18 24 15 13
```

2.6. Кубооктаедр — 14-гранник, у якого 6 граней — квадрати, 8 граней — правильні 3-кутники. До кожної з 12 вершин сходиться 3 ребра (грані). Утворено перетином куба й октаедра.



SOLID.DAT

```
12
 2 4  5  6
 1 3  6  7
 2 4  7  8
 1 3  5  8
 1 4  9 10
 1 2 10 11
 2 3 11 12
 3 4  9 12
 5 8 10 12
 5 6  9 11
 6 7 10 12
 7 8  9 11
```

SOLID.RES

```
8 6
```

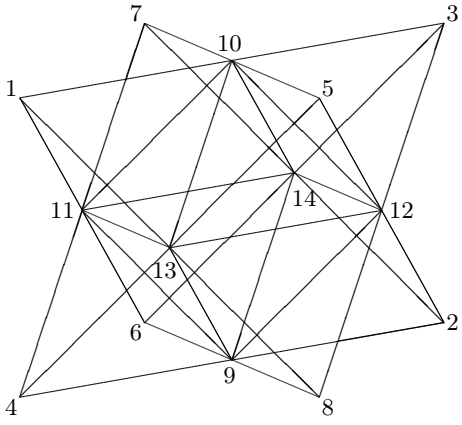
3.RES

```
1  2  6
1  4  5
2  3  7
3  4  8
5  9 10
6 10 11
7 11 12
8  9 12
```

4.RES

```
1  2  3  4
1  5 10  6
2  6 11  7
3  7 12  8
4  5  9  8
9 10 11 12
```

2.7. Зірчатий октаедр (stella octangula Кеплера) — неопуклий 24-гранник, всі грані якого — правильні 3-кутники. До кожної з 14 вершин сходиться або 3, або 8 ребер (граней). Утворено об'єднанням двох тетраедрів або продовженням граней октаедра.



SOLID.DAT

```
14
10 11 13
 9 12 14
10 12 14
 9 11 13
10 12 13
 9 11 14
10 11 14
 9 12 13
 2  4  6 8 11 12 13 14
 1  3  5 7 11 12 13 14
 1  4  6 7  9 10 13 14
 2  3  5 8  9 10 13 14
 1  4  5 8  9 10 11 12
 2  3  6 7  9 10 11 12
```

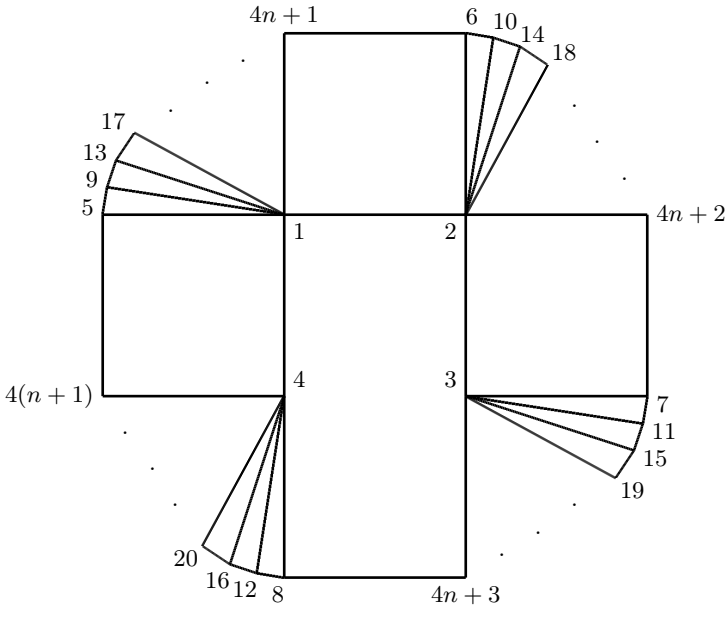
SOLID.RES

```
24
```

3.RES

```
1 10 11
1 10 13
1 11 13
2  9 12
2  9 14
2 12 14
3 10 12
3 10 14
3 12 14
4  9 11
4  9 13
4 11 13
5 10 12
5 10 13
5 12 13
6  9 11
6  9 14
6 11 14
7 10 11
7 10 14
7 11 14
8  9 12
8  9 13
8 12 13
```

2.8. Призматоїд — багатогранник, дві грані якого (називаються основами, для тесту обрано квадрат і $4n$-кутник), належать паралельним площинам, а решта є трикутниками, трапеціями або паралелограмами (для тесту обрано $4(n-1)$ трикутник і 4 прямокутника), причому у трикутників одна сторона, у трапецій основи, в паралелограмів протилежні сторони є сторонами основ призматоїда. На малюнку подано вид з боку меншої основи (квадрата), з якого (боку) видно всі ребра. Для тесту обрано $n = 399$.



Для подання вхідного та вихідних файлів використаємо позначення $m....n$ для арифметичної прогресії натуральних чисел, яка починається числом m, закінчується числом n, а кожен член прогресії більший за попередній на 4. У файлах присутні також послідовності груп 4-х рядків. Загальний вигдяд групи подано між двома рядками крапок, причому перед першим рядком крапок подано першу таку групу, після другого — останню.

SOLID.DAT

1600

2	4	5....1597
1	3	6....1598
2	4	7....1599
1	3	8....1600
1	9	1600
2	10	1597
3	11	1598
4	12	1599
1	5	13
2	6	14
3	7	15
4	8	16

.....

1	$4k-3$	$4k+5$
2	$4k-2$	$4k+6$
3	$4k-1$	$4k+7$
4	$4k$	$4(k+2)$

.....

1	1589	1597
2	1590	1598
3	1591	1599
4	1592	1600
1	6	1593
2	7	1594
3	8	1595
4	5	1596

3.RES

1	5	9

.....

1	$4k+1$	$4k+5$

.....

1	1593	1597
2	6	10

.....

2	$4k+2$	$4k+6$

.....

2	1594	1598
3	7	11

.....

3	$4k+3$	4k+7

.....

3	1595	1599
4	8	12

.....

4	$4k$	$4(k+1)$

.....

4	1596	1600

4.RES

1	2	3	4
1	2	6	1597
1	4	1600	5
2	3	7	1598
3	4	8	1599

1596.RES містить запис 5....1597 6....1598 7....1599 8....1600,

а SOLID.RES — 1592 5 $\underbrace{0\ \ldots\ 0}_{1591\ \text{i}}$ 1,

тобто серед граней призматоїда 1592 трикутника, п'ять чотирикутників і один 1596-кутник.

Примітка: вхідний файл SOLID.DAT створено за допомогою поданої далі програми

```
const n=399;  var j,k: word;  o: text;                    begin
assign(o,'solid.d_8');  rewrite(o);  writeln(o,4*(n+1));
write(o,'2 4'); for j:=1 to n do write(o,' ',4*j+1); writeln(o);
write(o,'1 3'); for j:=1 to n do write(o,' ',4*j+2); writeln(o);
write(o,'2 4'); for j:=1 to n do write(o,' ',4*j+3); writeln(o);
```

```
write(o,'1 3'); for j:=1 to n do write(o,' ',4*j+4); writeln(o);
writeln(o,'1  9 ',4*n+4);  writeln(o,'2 10 ',4*n+1);
writeln(o,'3 11 ',4*n+2);  writeln(o,'4 12 ',4*n+3);
                                        for j:=9 to 4*n do  begin
k:=j mod 4; if k=0 then k:=4;  writeln(o,k,' ',j-4,' ',j+4) end;
writeln(o,'1 6 ',4*n-3);  writeln(o,'2 7 ',4*n-2);
writeln(o,'3 8 ',4*n-1);  writeln(o,'4 5 ',4*n);  close(o)  end.
```

3. Приватні інтереси. За успішне проходження кожного тесту присуджують по 5 балів. Тести 1–3 перевіряють правильність відповіді навіть при неправильному тлумаченні умови задачі й використанні "жадібного" алгоритму здобуття максимального зиску на кожному кроці. Тести 1–2 перевіряють:

- оптимальне використання оперативної пам'яті (поле для гри має екстремальні розміри й потрібно подати двовимірний масив одновимірним);
- наскільки послідовно ("аж до абсурду") при правильному трактуванні умови проводиться аналіз сценаріїв гри "з кінця" на основі інтуїтивних уявлень про раціональну поведінку гравців, описаних в умові задачі.

3.1 Використано позначення $j.k$ для послідовності всіх натуральних чисел від j до k включно.

PRIVAT.DAT

```
2 6172
2.6166 6167 6168 6169 6170 6171 6172 0
0.6164 6165 6166 6167 6168 6169 6170 6171,
```

PRIVAT.RES

```
2 0
 u
```

3.2 Використано позначення для групи рядків одного виду (аналогічно опису тесту 2.8 попередньої задачі).

PRIVAT.DAT

```
6172 2
6171 0
6170 6172
6169 6171
.....
k k+2
.....
0 2
```

PRIVAT.RES

```
2 0
 r
```

3.3 Тест відтворює приклад, поданий в умові задачі, з точністю до множника 7.

PRIVAT.DAT

```
4 4
 7  7 14  0
21 49 56 63
21  0 49  7
 0 21 21  7
```

PRIVAT.RES

```
133 77
uurrr
```

3.4 Якщо перший рядок файлу PRIVAT.DAT містить запис `98 89`, а k-те число j-го рядка цього самого файлу — результат округлення до найближчого цілого числа виразу $200|2\sin k^2 + 3\cos(j-1)^2|$, де j=2, 3, ... , 99, k=1, 2, ... , 89, то файл PRIVAT.RES має вигляд

```
59621   60885
ruurrurrrurrrrrrrrrruuuuuruurrrurrurrrruuuuuuuuuur
rrruuruuuruuruurrrrruurruurrrrrrrurrruuuuuuurruuru
rururruuururuurrurrruuuuurrrurrrruuuuuurruuuuuurrr
rrrrurrruurrurruuuuuuuuuuuuuuuuuuur
```

— для зручності читача другий рядок (184 символи) подано трьома рядками по 50 символів і одним з 34 символами.

4. Вечірка. За успішне проходження тестів 1–3 присуджують по 5 балів, за тест 4 — 9 балів. У разі недотримання алфавітного порядку знімають 1 штрафний бал за кожен тест окремо. У разі баготократного переліку одного й того самого варіанту розміщення студентів за столом також знімається 1 штрафний бал за кожен тест окремо. Всі тести перевіряють можливість отримання правильного варіанту відповіді; тест 2 — можливість отримання *всіх* варіантів правильної відповіді; тест 3 — дотримання алфавітного порядку з урахуванням того, що кодування літер української абетни не відповідає алфавітному порядку; тест 4 — ефективність алгоритму. єдиний рядок вихідного файлу PARTY.RES подано далі в тексті кількома рядками, щоб не використовувати дуже дрібний шрифр.

4.1 Нехай файл PARTY.DAT має вигляд

```
Федір | мистецвознавець,
Олексій < етнограф,
бібліограф > Олександр,
Олесь > етнограф,
менеджер | Олексій.
```

Тоді файл PARTY.RES має вигляд

```
Олександр (мистецвознавець),
Олексій (бібліограф),
Федір (етнограф),
Олесь (менеджер).
```

4.2 Нехай файл PARTY.DAT має вигляд

```
Федір | мистецвознавець,
Олесь > етнограф,
бібліограф > Олександр,
менеджер | Олексій.
```

Тоді файл PARTY.RES має вигляд

```
Олександр (мистецвознавець),
Олексій (бібліограф),
Федір (етнограф),
Олесь (менеджер).

Олександр (менеджер),
Федір (бібліограф),
Олексій (етнограф),
Олесь (мистецвознавець).
```

4.3 Нехай файл PARTY.DAT має вигляд

```
Тарас > гідролог,
Павло > метеоролог,
інженер < іван,
іван < метеоролог.
```

Тоді файл PARTY.RES має вигляд

```
іван (гідролог),
Тарас (метеоролог),
Павло (інженер).
```

4.4 Нехай файл PARTY.DAT має вигляд

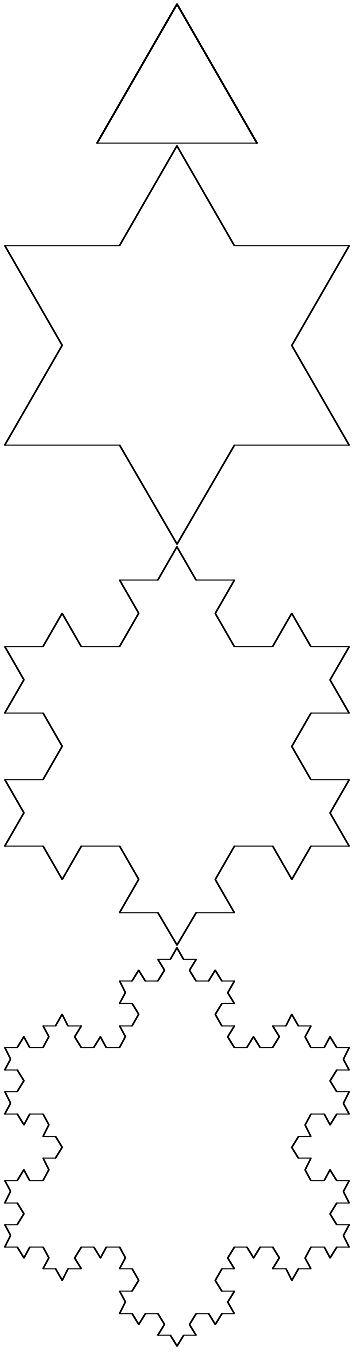
```
Анрі | історик,
фізик | Людовік,
Ежен | психолог,
біолог | Ганс,
Франсуа | стоматолог,
геолог | Аарон,
Жак | хірург,
журналіст | ігор,
Жан < хімік,
географ > ів,
Антоні > філолог,
Ремі < юрист,
Ганс > психолог,
Мохамед < терапевт,
археолог > Дмитро,
Етьен < філолог,
історик < юрист,
філософ > Жером,
археолог > хірург,
хімік | психолог,
географ | Мохамед,
стоматолог < терапевт,
ігор < математик,
математик | історик,
Аарон < хірург.
```

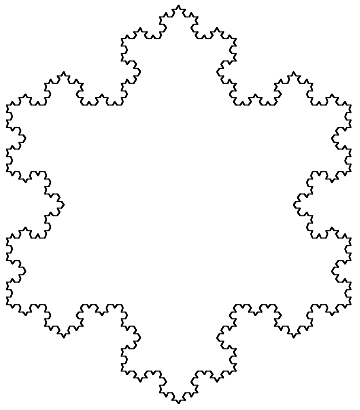
Тоді файл PARTY.RES має вигляд

```
Аарон (терапевт),
Дмитро (хірург),
ігор (археолог),
Анрі (математик),
Жан (фізик),
Ежен (хімік),
ів (біолог),
Франсуа (географ),
Етьен (геолог),
Жак (філолог),
Антоні (журналіст),
Ремі (історик),
Людовік (юрист),
Жером (психолог),
Ганс (філософ),
Мохамед (стоматолог).
```

5. Багатокутник. За правильне зображення кожного багатокутника — по 2 бали. За те, що зображення по

вертикалі не займає весь екран, віднімають 1 бал. Наступні рисунки подають зображення $3 \cdot 4^n$-кутників для $n = 0$, 1, 2, 3, 4, які мають бути на екрані.





Деталі зображення $3 \cdot 4^5$-кутника — останнього, який виникає на екрані — *для даних розмірів сторінки* майже непомітні, тому рисунок не подано.

6. Квадратний корінь. За правильне проходження всіх тестів, крім 11 і 12, присуджують по 1 балу, за тести 11–12 — по 6 балів. Перевіряють правильність відповіді, якщо квадратний корінь є:

- натуральним числом — тести 1–4;
- скінченним десятковим дробом — тести 5–8;
- періодичним десятковим дробом — тести 9–12.

Тести 13–24 отримано незначним "спотворенням" тестів 1–12. Перевіряють можливість отримання правильної відповіді, якщо квадратний корінь є ірраціональним числом, а перше число файлу ROOT.DAT є:

• натуральним числом — тести 13–16;

• скінченним десятковим дробом — тести 17–20;

• періодичним десятковим дробом — тести 21–24.

Перевіряють можливість подання числа

масивом його цифр — тести 3–4, 7–8, 11–12, 15–16, 19–20, 23–24.
Перевіряють правильність роботи алгоритму для чисел, у яких кількість цифр перед десятковою крапкою:

- непарна — тести з непарними номерами;
- парна — тести з парними номерами.

6.1 Якщо ROOT.DAT містить `196 4`, то файл ROOT.RES містить `14`.

6.2 Якщо ROOT.DAT містить `81 11`, то файл ROOT.RES містить `9`.

6.3 Якщо ROOT.DAT містить запис
`1524157877515647915714744191487952956552463714689 2`,
то файл ROOT.RES містить `1234567891011121314151617`.

6.4 Якщо ROOT.DAT містить запис
`5128728465153643380363614061852848487367778997104 1 5`,
то файл ROOT.RES містить запис `7161514131211110987654321`.

6.5 Якщо ROOT.DAT містить `371.7184 4`, то файл ROOT.RES містить `19.28`.

6.6 Якщо ROOT.DAT містить `8430.9124 4`, то файл ROOT.RES містить `91.82`.

6.7 Якщо ROOT.DAT містить запис[8]
`14965471632360523481656817311375311163234.`
`607165212148126073851854098765422222222000000001 4`,
то файл ROOT.RES містить запис
`122333444455555666666.777777788888888999999999`.

6.8 Якщо ROOT.DAT містить запис
`99999999977777777556790101259237039407387656570 3.`
`98768177802520027187684076595087928423484 1 4`,
то файл ROOT.RES містить запис
`999999999888888887777777.666666555554444333221`.

6.9 Якщо ROOT.DAT містить запис `0.07(1) 4`, то файл ROOT.RES містить запис `0.2(6)` — див. рівність $\sqrt{16/225}$=4/15.

6.10 Якщо ROOT.DAT містить запис `69.(4) 4`, то файл ROOT.RES містить запис `8.(3)` — див. рівність $\sqrt{625/9}$=25/3.

6.11 Якщо ROOT.DAT містить запис

[8]Щоб не використовувати занадто дрібний шрифт, тут і в поданні деяких тестів далі десятковий дріб розбито на частини, подані кількома рядками тексту.

57217584514360514066137 12.
72290783295719092364557641 86(
13017751479289940828402366863905325443786982248520
71005917159763313609467455 62) 4,
то файл ROOT.RES містить запис
2392019743111.67661955230798(769230)
— див. рівність

$$\sqrt{\frac{3602270701127844630620321073276638126988955 2001}{6295740604400634765625}} = \\ = \frac{189796488406077856776001}{79345703125} = \\ = \frac{(17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43)^2}{13 \cdot 5^{14}}.$$

6.12 Якщо ROOT.DAT містить запис
5691996867857144.
22522869818261011935063320691528355226935124113262
67333792749921169794973499169005258222(
43752750745757738764731771724778717785710792703799
69680668981368282067582766883466184165484864785564
08626338696268766198836128906058975989045919115849
18577925570932563939556946549953542960535967528974
52198151498850799550100249400948701648002347303046
60374590444520514450584380654310724240794170864100
93403100396107389114382121375128368135361142354149
34715634016333317032617731918431219130519829820529
12122842192772262702332632402562472492542422612352
68228275221282214289207296200303193310186317179324
17233116533815834515135214435913736613037312338011
63871093941024010954080884150814220744290674360604
43053450046457039464032471025478018485011492004498
99750599051298351997652696953396254095554794855494
15619345689275759205829135899065968996038926108856
17878624871631864638857645850652843659836666829673
82268081568780869480170179470878771578072277372976
67367597) 2,
то файл ROOT.RES містить запис
75445323.
69774248788347104554641763907629293161672312(062937)
— див. рівність

$$\sqrt{\frac{3602270701127844630620321073276638126988955 2001}{6328658965836685310353047814144}} =$$

$$= \frac{189796488406077856776001}{2515682604351488} =$$

$$= \frac{(17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43)^2}{11 \cdot 13 \cdot 2^{44}}.$$

6.13 Якщо ROOT.DAT містить `195 4`, то файл ROOT.RES містить `13.9642`.

6.14 Якщо ROOT.DAT містить `82 4`, то файл ROOT.RES містить `9.0553`.

6.15 Якщо ROOT.DAT містить запис
`1524157877515647915714744191487952956552463714690 44`,
то файл ROOT.RES містить запис
`123456789101112131415161 7.`
`00000000000000000000000040500000335380168066.`

6.16 Якщо ROOT.DAT містить запис
`5128728465153643380363614061852848487367778997104 0 44`,
то файл ROOT.RES містить запис
`7161514131211110987654320.`
`99999999999999999999999993018236215985192901.`

6.17 Якщо ROOT.DAT містить запис `371.7183 9`,
то файл ROOT.RES містить запис `19.279997406`.

6.18 Якщо ROOT.DAT містить запис `8430.9125 9`,
то файл ROOT.RES містить запис `91.820000544`.

6.19 Якщо ROOT.DAT містить запис
`14965471632360523481656817311375311163234.`
`607165212148126073851854098765422222222000000002 99`,
то файл ROOT.RES містить запис
`122333444455555666666.777777788888888999999999`
`00000000000000000000000000000000000000000000000`
`40871897478669655504576911263 76.`

6.20 Якщо ROOT.DAT містить запис
`99999999977777777556790101259237039407387656570 3.`
`98768177802520027187684076595087928423484 99`,
то файл ROOT.RES містить запис
`99999999988888887777777.666666555554444333220`
`99999999999999999999999999999999999999999999999`
`49999999994444444382715993702962.`

6.21 Якщо ROOT.DAT містить запис `0.07(5) 5`, то файл ROOT.RES містить запис `0.27487` — $17/225$ не є квадратом раціонального числа.

6.22 Якщо ROOT.DAT містить запис `69.(5) 5`, то файл ROOT.RES містить запис `8.33999` — $626/9$ не є квадратом раціонального числа.

6.23 Якщо ROOT.DAT містить запис
```
57217584514360514066137l2.
722907832957190923645735256l
(60946745562130177514792899408284023668б
39053254437869822485207100591715976331З) 50,
```
то файл ROOT.RES містить запис
```
2392019743111.
67661955230798769230769230769230772550924684887035
```

$$\frac{360227070112784463062032107327663812698895520 02}{6295740604400634765625} =$$
$$= \frac{(17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43)^4 + 1}{13^2 \cdot 5^{28}}$$

не є квадратом раціонального числа.

6.24 Якщо ROOT.DAT містить запис
```
5691996867857144.
225228698182610119350633206915599574986137906921l7
056830552060729588716555454368503631550
747224803168859112915056971001026945082889138833l9
477725072130666536260941855347449753044158638564235
296982737542178101618661059220499779940339380898825
145826201770257714313658369602425546481490537434595
337864932270526676121081715487309892904298498704095
310968751528192087632647073206513765954325394884835
544427600371656315712259768203824147880091936035995
198004792410386815981221575627170032764438358843955
324954765514206073646633087192527751968311408870845
943028998973054917110861166805222749278693334637395
058144652550246955841361435767030172624578218983815
338940779500220059660619101178541737982297422856865
341630397574453518509462565406621350677294733238785
918284512690107095701501295906890312484718079123675
352926793486234045674605115164555723996283436842875
740231796175852119908063964008019952075896131840185
778424372829967235561641156046750452344857939263535
```

36691280) 100,
то файл ROOT.RES містить запис
75445323.
69774248788347104554641763907629293161881750303306
17542047175993933406248534674318015991887871968157

$$\frac{360227070112784463062032107327663812698895520 03}{63286589658366853103530478141 44} =$$

$$= \frac{(17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43)^4 + 2}{(11 \cdot 13 \cdot 2^{44})^2}$$

не є квадратом раціонального числа.

3.5 Розв'язання

1. Хімія. Успішне розв'язання задачі передбачає здійснення наступних кроків (див. зміст назв змінних поданої далі програми).

Зчитування вхідних даних: перший рядок CHEMIE.DAT подаємо рядковою змінною `up` довжини `nup`, другий рядок — рядковою змінною `low` довжини `nlow`.

Опрацювання вхідних даних:

`latinup` — множина великих літер латиниці;
`latinlow` — множина малих літер латиниці;
`j` — номер символу першого та другого рядків вхідного файлу, що аналізують;
`n` — номер сполуки (тут і надалі — рахуючи зліва направо у записі рівняння хімічної реакції), символ якої `up[j]` аналізують (після опрацювання вхідних даних — кількість всіх сполук);
`ne` — кількість різних хімічних елементів, позначення яких зчитано з вхідного файлу;
`el[k]` — позначення хімічного елементу, яке зустрілося `k`-им у рівнянні реакції (повторення не враховуюся);
`a[n,k]` — кількість атомів елементу `el[k]` у `n`-ій сполуці;
`m[n]` — номер символу рядка `up`, з якого починається формула `n`-ої сполуки;
`group` — булева змінна, яка набирає значення `true` тоді й лише тоді, коли символ `up[j]` належить групі, виділеній круглими дужками;
`l[k]` — кількість входження `k`-го елементу у розглядувану групу;
`s` — рядкова змінна, що зберігає запис кратності входження групи атомів у сполуку або окремого атома; використовується також для аналізу позначення хімічного елемента;
`No` — номер елементу, який розглядається.

Перебір всіх можливих значень коефіцієнтів з оптимізацією перебору:

`c[k]` — коефіцієнт, який потрібно поставити перед формулою `k`-ої сполуки в рівнянні хімічної реакції;

`lr` — для `k>ns` сума доданків вигляду `c[j]*a[j,No]`, взятих із знаком +, якщо `j` неперевищує `ns`, та із знаком − у протилежному випадку.

Для того, щоб отримати розв'язання, перебір потрібно починати з найменших значень коефіцієнтів, тобто з 1. Справдження рівності `lr=0` для всіх елементів, якщо `k=n` — це умова коректності рівняння. Оскільки `lr` не зростає із зростанням `k>ns`, то перебір продовжень вибраної послідовності коефіцієнтів потрібно обірвати, як тільки но `lr<0`. *Формування відповіді* "внесенням" коефіцієнтів рівняння у рядкову змінну `up` та відповідної кількості прогалин у `low`.

Запис відповіді у файл CHEMIE.RES рядками `up` і `low`.

```
const latinup=['A'..'Z'];latinlow=['a'..'z'];max=10;cmax=30;
label NEXTc,NEXTj,NEXTk; var a:array[1..max,1..max] of byte;
o:text;  group:boolean;  c,m,l:array[1..max]        of byte;
lr:integer; up,low,s:string;el:array[1..max] of string[2];
j,k,n,ns,ne,nup,nlow,No,q: byte;
                                {НСД двох натуральних чисел}
function gcd (a,b: byte): byte;  var c,d,f: byte;     begin
c:=a; d:=b;  while d<>0 do begin f:=c mod d; c:=d; d:=f end;
gcd:=c;                                                 end;

begin{chemie}        m[1]:=1;  for k:=1 to max do     begin
             for j:=1 to max do a[k,j]:=0; l[k]:=0;  end;
{Зчитування даних}        assign(o,'CHEMIE.DAT');  reset(o);
               readln(o,up);     readln(o,low);    close(o);
            nup:=length(up); nlow:=length(low); n:=1; ne:=0;
{Опрацювання даних}     group:=false;  j:=0;  NEXTj: j:=j+1;
if up[j]='+' then begin        n:=n+1; m[n]:=j+1 end else
if up[j]='=' then begin ns:=n; n:=n+1; m[n]:=j+1 end else
if up[j]='(' then                                       begin
                group:=true; for k:=1 to ne do l[k]:=0 end
                                                         else
if up[j]=')' then                                       begin
s:=''; while j<nlow do if (low[j+1]<>' ') and (low[j+1]<>'')
then begin j:=j+1; s:=s+low[j] end else break;
val(s,q,lr);  group:=false;
for k:=1 to ne do a[n,k]:=a[n,k]+q*l[k]                   end
                                            else        begin
{Аналіз s - позначення елемента}
if (up[j] in latinup) and (up[j+1] in latinlow) then
begin s:=up[j]+up[j+1]; j:=j+1 end  else  s:=up[j];
   No:=0;  for k:=1 to ne do  if s=el[k] then begin
   No:=k;                                 break  end;
if No =0 then begin ne:=ne+1; el[ne]:=s; No:=ne end;

{Визначення q - кратності входження у фрагмент формули}
```

```
q:=1;   if (j=nup) and (j<nlow) or (up[j+1]=' ') then begin
s:=''; while j<nlow do if (low[j+1]<>' ') and (low[j+1]<>'')
then begin j:=j+1; s:=s+low[j] end else break;
val(s,q,lr)                                                end;
if group then  l[No]:=l[No]+q  else  a[n,No]:=a[n,No]+q end;
if j<nup then goto NEXTj;
       k:=0;             {Перебір всіх можливих комбінацій}
NEXTk: k:=k+1; c[k]:=0;
NEXTc:         c[k]:=c[k]+1;
if c[1]>cmax then halt else
if c[k]>cmax then begin k:=k-1; goto NEXTc end;
if k>ns then                          for No:=1 to ne do begin
lr:=0;  for j:=1    to ns do lr:=lr+c[j]*a[j,No];
        for j:=1+ns to k  do lr:=lr-c[j]*a[j,No];
                        if lr<0 then goto NEXTc           end;
if k<n  then goto NEXTk;              for No:=1 to ne do begin
lr:=0;  for j:=1    to ns do lr:=lr+c[j]*a[j,No];
        for j:=1+ns to n  do lr:=lr-c[j]*a[j,No];
if lr<>0 then goto NEXTc                                   end;
                                    {Формування відповіді}
for k:=n downto 1 do if c[k]>1 then                     begin
str(c[k],s);  j:=length(s);
 up:=copy(up ,1,m[k]-1)+s+copy(up ,m[k],nup);   nup:=nup+j;
low:=copy(low,1,m[k]-1)  +copy('         ',1,j)
                         +copy(low,m[k],nlow); nlow:=nlow+j
                                                           end;
{Запис відповіді}   assign(o,'CHEMIE.RES');  rewrite(o);
                writeln(o,up); writeln(o,low); close(o) end.
```

2. Многогранник.

Зчитування і запам'ятовування вхідних даних передбачає використання таких змінних:

`n` — кількість всіх вершин;

`ne` — подвоєна кількість всіх ребер;

`b[k]` для `k` в межах від `a0[j-1]+1` до `a0[j]` включно (одразу після зчитування даних) — монотонна послідовність номерів вершин, які з'єднано з `j`-ою вершиною ребром.

Інформацію про те, які вершини сполучено ребрами, небажано зберігати у вигляді *дво*вимірного масиву (таблиці), в якій перший індекс `j` — номер вершини, а величина самого елемента — номер іншої вершини, яку з'єднано з `j`-ою вершиною ребром. У цьому випадку проходження останнього тесту неможливе — не вистачить ресурсів пам'яті.

Виділення граней многогранника здійснюється обходом граней по периметру в порядку збільшення кількості вершин грані `l`: спочатку — трикутники, потім — чотирикутники, потім — п'ятикутники і т.д. Робимо перебір всіх послідовностей номерів вершин `v[j]` багатогранника, в яких:

- кожну наступну вершину `v[j1]` з'єднано з попередньою `v[j]` ребром (вибираємо `v[j1]:=b[d[j]]`, де ціла змінна `d[j]` більша за `a0[v[j]-1]`, але не перевищує `a[v[j]]` — зміст елементів масиву `a0` описано раніше, на початку перебору масиви `a0`, `a` ідентичні, а умову зміни величини елемента `a` описано далі);

- останню вершину `v[l]` з'єднано ребром з першою вершиною `v[1]` (шукаємо `d[l]` з умови `v[1]=b[d[l]]`);

- номер першої вершини — найменший з номерів усіх вершин;

- номер другої вершини — менший від номера останньої вершини;

- жоден номер не зустрічається у послідовності двічі.

Якщо виявлено *першу* грань, яка містить ребро, що з'єднує вершини `v[j]` і `b[d[j]]`, то значення булевої змінної `c[d[j]]` змінюємо з `false` на `true`. Якщо виявлено *другу* грань, яка містить це ребро (коли `c[d[j]]=true`), то таке ребро вилучаємо з подальшого розгляду — величину `a[v[j]]` зменшуємо на 1 з відповідним зсувом значень частини елементів масиву `b`. В обох випадках `nl` — кількість виявлених граней, що мають `l` вершин — збільшуємо на 1. Після розгляду всіх ламаних довжини `l` величину `nl` записуємо у файл SOLID.RES, а величину `ne` зменшуємо на `l*nl`.

Якщо $l = 8$, перевіряємо умову: чи всі ребра, які залишилися, пройдено по одному разу (з кожної вершини виходить або два таких ребра, або жодного). Якщо це так, покладемо `l:=ne`, записавши відповідну кількість нулів у файл SOLID.RES — згідно умови задачі лише в одній грані кількість вершин перевищує 8. Такий "стрибок" позбавляє необхідності робити перебір великої кількості ламаних, які не обмежують грані. Лише так можна успішно пройти тест 8.

Технічні умови (структура вхідного та вихідних файлів) максимально сприяють простоті алгоритму. Ускладнення задачі зняттям вимоги монотонності послідовностей чисел у рядках SOLID.DAT легко долають традиційними методами упорядкування. Зняття обмеження на кількість граней з великою кількістю ребер — серйозніша проблема. У цьому випадку для підвищення ефективності алгоритму доречно виділити незамкнені ламані, з вершин яких, крім першої та останньої, виходять ребра, які або належать цій ламаній, або вже "пройдено" двічі. Як тільки не залишиться жодної вершини многранника зовні таких ламаних, то можна здійснити відповідний "стрибок" щодо кількості вершин грані `l`.

Послідовність розгляду кількості вершин грані (у порядку зростання) гарантує, що до граней не буде зараховано просторовий многокутник (замкнену ламану), всі сторони якого і хоча б одна з діагоналей — ребра даного багатогранника.

Описаний алгоритм повністю придатний для опуклих однорідних многогранників[9] незалежно від нумерації вершин. Але для поданої далі нумерації вершин зірчатого октаедра такий алгоритм при відсутності блоку аналізу, чи є виявлена замкнена ламана гранню, зациклюється і не дає потрібної відповіді.

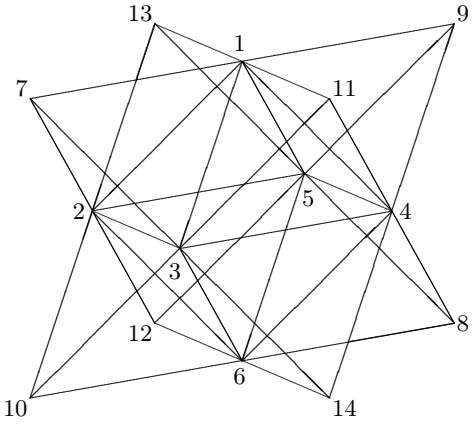


Рисунок 8: нумерація вершин stella octangula Кеплера, для якої треба аналізувати, чи є виявлена замкнена ламана гранню.

SOLID.DAT

```
14
 2 3 4 5 7  9 11 13
 1 3 5 6 7 10 12 13
 1 2 4 6 7 10 11 14
 1 3 5 6 8  9 11 14
 1 2 4 6 8  9 12 13
 2 3 4 5 8 10 12 14
 1 2 3
 4 5 6
 1 4 5
 2 3 6
 1 3 4
 2 5 6
 1 2 5
 3 4 6
```

SOLID.RES

```
24
```

3.RES

```
1 2  7
1 2 13
1 3  7
1 3 11
1 4  9
1 4 11
1 5  9
1 5 13
2 3  7
2 3 10
2 5 12
2 5 13
2 6 10
2 6 12
3 4 11
3 4 14
3 6 10
3 6 14
4 5  8
4 5  9
4 6  8
4 6 14
5 6  8
5 6 12
```

Аналіз, чи є виділена замкнена ламана гранню, у поданій далі програмі здійснено таким чином.

[9]Многогранник називається однорідним, якщо всі його грані — правильні многокутники — розташовані однаковим чином біля кожної вершини.

1. Виділяємо вершину з найменшим номером `k1`, що не є вершиною ламаної.

2. Для всіх j, що є номерами виділеної вершини або вершин ламаної, покладемо `e[j]:=false`, для всіх інших значень j в межах від 1 до n включно покладемо `e[j]:=true`. Тут і надалі `e[j]` не справджується тоді й лише тоді, коли вершина j належить ламаній, або для неї встановлено існування маршруту ребрами многогранника з виділеної вершини без перетину ламаної.

3. Знаходимо всі вершини, які можна досягнути ребрами многогранника без перетину ламаної з виділеної вершини. У поданій програмі:

   `nf0` і `nf1` — кількості таких вершин, виявлених відповідно на попередньому і поточному кроках дослідження;

   `f[j]` для `j` в межах від 1 до `nf0` або в межах від `(nf0+1)` до `(nf0+nf1)` — номери таких вершин, виявлених відповідно на попередньому або поточному кроках.

   Для такого пошуку зручно використати масив змінних `b0`, що є копією масиву `b` по закінченню роботи блоку *зчитування і запам'ятовування вхідних даних*. Пошук припиняємо, якщо `nf1=0` після здійснення певного кроку пошуку нових вершин. Іншими словами, коли повністю виділено зв'язну складову графа-многогранника по один бік від досліджуваної ламаної.

4. Здійснюємо пошук вершини, яка лежить з виділеною точкою на поверхні многогранника по різні боки від ламаної. іншими словами, шукаємо j у межах від 1 до n таке, що справджується `e[j]`. Ламана є гранню тоді й лише тоді, коли такої вершини j немає.

Можливе розв'язання задачі й без такого аналізу:

- здійснюємо перебір всіх послідовностей граней, у яких кількість вершин кожної наступної грані не менша за кількість вершин попередньої грані;

- при вилученні ребер з розгляду (при прийнятті гіпотези про наявність грані з цими ребрами) запам'ятовуємо рівень гіпотези (номер члена послідовності граней);

- при відмові від гіпотези про наявність певної грані:

  * долучаємо в розгляд усі її вилучені раніше ребра з інформацією про їх однократне використання і вважаємо, що всі її невилучені раніше ребра не належать жодній із граней розглядуваної послідовності;

* переходимо до наступної щодо лексикографічного порядку гіпотези про наявність грані з такою ж кількістю вершин. За відсутності таких вершин збільшуємо кількість вершин грані на 1 доти, поки не знайдемо замкнену ламану з невилучених ребер відповідної довжини.

Такий підхід істотно ускладнить програму і збільшить час обчислень. На думку автора втілення будь-якого з цих підхів непосильне для учасників олімпіади, не знайомих з використанням елементів теорії графів у програмуванні стереометричних задач. Тому тест 9, поданий в описі розв'язання, не виносився на перевірку. Задача 2 "Многогранник" була єдиною задачею, за розв'язання якої всі учасники олімпіади отримали 0 балів навіть за подані тести 1–8. Це можна пояснити очевидністю алгоритму задачі 1 і жадібного алгоритму для *неправильного* трактування умови задачі 3 при природньому бажанні всіх учасників здобути максимальну кількість балів.

```
const nedge=4000;  nvertex=2000;
label NEXTd,NEXTj,NEXTv1,NOMORE;
  var    c: array[1..2*nedge] of boolean;    s: string;
         e: array[0..nvertex] of boolean;
      b,b0: array[1..2*nedge] of word;    o,ol: text;
a,a0,d,f,v: array[0..nvertex] of word;
i,j,j1,k,k1,l,nl,m,m1,n,ne,nf0,nf1: word;             begin
{$I-} for j:=1 to nedge do c[j]:=false;  a0[0]:=0; v[0]:=0;
assign(o,'SOLID.DAT');  reset(o);  readln(o,n);
       ne:=0;                          for j:=1 to n do begin
repeat ne:=ne+1; read(o,b[ne]) until eoln(o); a0[j]:=ne;
                                                 a[j]:=ne end;
for j:=1 to ne do b0[j]:=b[j];
l:=2;             assign(o,'SOLID.RES'); rewrite(o); repeat
l:=l+1; str(l,s); assign(ol,  s+'.RES'); rewrite(ol);
        nl:=0;  v[1]:=0;
NEXTv1: repeat  v[1]:=v[1]+1
         until (v[1]>n) or (a[v[1]]>a0[v[1]-1]);
            if  v[1]>n then goto NOMORE;
       j:=0;  j1:=1;
NEXTj: j:=j1; j1:=j1+1;  d[j]:=a0[v[j]-1];
NEXTd: repeat             d[j]:=d[j]+1;
                       if d[j]<=a[v[j]] then v[j1]:=b[d[j]]
        until
  (d[j]>a[v[j]]) or (a[v[j1]]>a0[v[j1]-1]) and (v[1]<v[j1]);
if d[j]>a[v[j]] then                                  begin
j:=j-1;j1:=j1-1;if j1=1 then goto NEXTv1 else goto NEXTd end;

k:=0;repeat k:=k+1 until v[k]=v[j1]; if j1>k then goto NEXTd;
if j1<l then goto NEXTj;       if v[l]<v[2]  then goto NEXTd;

       {Перевірка замкнутості ламаної v[1], v[2], ... , v[l]}
```

```
       k:=a0[v[l]-1];
repeat k:=k+1 until (k>a[v[l]]) or (b[k]=v[1]);
                  if k>a[v[l]] then goto NEXTd; d[l]:=k;
                                {Чи є ламана гранню?}
for k:=1 to n do e[k]:=false;                          k1:=1;
for k:=1 to l do e[v[k]]:=true;  while e[k1] do inc(k1);
       nf0:=1; f[1]:=k1; for k:=1+k1 to n do e[k]:=true;
                         for k:=1 to l do e[v[k]]:=false;
repeat nf1:=0;           for k:=1 to nf0 do
for k1:=a0[f[k]-1]+1 to a0[f[k]] do if e[b0[k1]] then begin
   inc(nf1);      f[nf0+nf1]:=b0[k1];  e[b0[k1]]:=false end;
               for k:=1 to nf1 do f[k]:=f[nf0+k];  nf0:=nf1
 until nf1=0;      k:=0;  repeat inc(k) until (k>n) or e[k];
                                    if k<=n then goto NEXTd;
{Запис відповіді й вилучення з подальшого розгляду грані}
for k:=1 to l do write(ol,v[k]:6);  writeln(ol);
for k:=1 to l do if c[d[k]] then                       begin
                 {Якщо ребро [v[k];b[d[k]]] пройдено вперше}
a[v[k]]:=a[v[k]]-1;          for m:=d[k] to a[v[k]] do begin
                             b[m]:=b[m+1];  c[m]:=c[m+1] end;
d[k]:=d[k]-1                                             end
                             else c[d[k]]:=true;
for k:=1 to l do                                       begin
if k=1 then k1:=v[l] else k1:=v[k-1];  m:=a0[v[k]-1];
repeat m:=m+1 until b[m]=k1;              if c[m] then begin
                        {Якщо ребро [v[k];k1] пройдено двічі}
a[v[k]]:=a[v[k]]-1;           for m1:=m to a[v[k]] do begin
                          b[m1]:=b[m1+1]; c[m1]:=c[m1+1] end;
if d[k]>=m then d[k]:=d[k]-1                             end
                                       else c[m]:=true  end;
        nl:=nl+1;  goto NEXTd;
NOMORE:      close(ol);  write(o,' ',nl);
if nl=0 then erase(ol) else ne:=ne-l*nl;  if l=8 then begin
k:=0;         for m:=0 to n-1 do if a[m+1]>a0[m] then begin
if (a[m+1]<>a0[m]+2) or (not c[a0[m]+1]) or (not c[a0[m]+2])
then begin k:=0; break end else k:=k+1                   end;
        if k >0 then                                   begin
for m:=l+1 to k-1 do write(o,' 0'); l:=k-1           end end
                                                 until ne=0;
             close(o)                                    end.
```

З використанням структури множини програма створюється набагато швидше і простіше.

```
label NEXTp1, NEXTi, NEXTh, NOMORE;
const nv_=2000;              {Верхні межі кількості: вершин}
      ne_=4000;                                      {ребер}
                nv_256=nv_ div 256;
```

```
var c: array[0..nv_256] of set of byte;
    v: array[1..ne_*2] of word;             {Суміжні вершини}
   kr: array[1..ne_*2] of 0..2;{Кратність використання ребра}
    n,                                      {Вказівники на v}
    h,                                  {Гіпотези-вказівники на v}
    f,                  {Цикл та вершини по один бік від нього}
    p: array[0..nv_] of word;                          {Грань}
 o,ol: text;                           {Файли даних і відповіді}
   nv,                                    {Кількості: вершин}
   ne,                                           {всіх ребер}
   ke,                                     {нерозглянутих ребер}
    l,                                           {вершин грані}
   nl,                                       {l-кутних граней}
         {Кількості вершин по один бік від циклу, знайдені:}
   n0,                                    {до попереднього кроку}
   n1,                                    {на попередньому кроці}
   n2,                                          {на даний момент}
   n3,                                                  {n2-n1}
 n256,                                             {nv div 256}
                         i,j,k: word;  s: string[4];  BEGIN
assign(o,'SOLID.DAT');  reset(o);  readln(o,nv);
n[0]:=0;                                for j:=1 to nv do begin
n[j]:=n[j-1];                    while not seekeoln(o) do begin
inc(n[j]);  read(o,k);  v[n[j]]:=k;  kr[n[j]]:=0         end;
                                                  readln(o) end;
 close(o); ne:=n[nv] div 2;  ke:=ne;  n256:=nv div 256;
          l:=2;          assign(o,'SOLID.RES'); rewrite(o);
repeat inc(l);  str(l,s); assign(ol,  s+'.RES'); rewrite(ol);
         nl:=0;     p[1]:=0;
NEXTp1:    i:=1; inc(p[1]);  if p[1]>nv-l+1 then goto NOMORE;
NEXTi: inc(i);  h[i]:=n[p[i-1]-1];
NEXTh: while   (h[i]=n[p[i-1]]) and (2<i) do dec(i);
       if       h[i]=n[p[i-1]] then goto NEXTp1;
       repeat inc(h[i]) until (h[i]=n[p[i-1]])or(kr[h[i]]<2);
       if kr[h[i]]=2 then goto NEXTh;
                       {Перевірка гіпотези: порядок перебору}
       if (i=2) and (p[1]>v[h[2]]) or
          (i=l) and (p[2]>v[h[l]]) then goto NEXTh;
       if i<=l then        {відсутність самоперетинів} begin
       k:=0;  repeat inc(k) until (k=i-1) or (p[k]=v[h[i]]);
       if p[k]=v[h[i]] then goto NEXTh                 end;
          p[i]:=v[h[i]];                  {Прийняття гіпотези}
       if i<l+1        then goto NEXTi;
       if p[l+1]<>p[1] then goto NEXTh; {Чи ламана замкнена?}
       for k:=0 to n256 do c[k]:=[];    {Чи є цикл гранню?}
       for k:=1 to l do include(c[p[k] div 256],p[k] mod 256);
       k:=0;  repeat inc(k)
```

```
              until not ((k mod 256) in c[k div 256]);
       include(c[k div 256],k mod 256);
       n0:=0;  n1:=1;  n2:=1;  f[1]:=k;
repeat for k:=n0+1 to n1 do for j:=n[f[k]-1]+1 to n[f[k]] do
       if not ((v[j] mod 256)in(c[v[j] div 256])) then begin
       include(c[v[j] div 256],v[j] mod 256);
                                  inc(n2);  f[n2]:=v[j] end;
       n3:=n2-n1;  n0:=n1;  n1:=n2;
 until n3=0;  k:=0;
repeat inc(k)
 until not ((k mod 256) in c[k div 256]) or (nv=k);
    if not ((k mod 256) in c[k div 256]) then goto NEXTh;
       inc(nl);                {Запис грані, вилучення ребер}
       for j:=1 to l do write(ol,p[j],' ');      writeln(ol);
       for j:=2 to l+1 do inc(kr[h[j]]);
       for j:=2 to l+1 do                                begin
       k:=n[p[j]-1];  repeat inc(k) until v[k]=p[j-1];
       inc(kr[k]);  if kr[k]=2 then dec(ke)              end;
       goto NEXTh;        {Запис кількості граней l-кутників}
NOMORE: write(o,nl,' ');  close(ol);
        {Чи не виходить з кожної вершини більше, ніж 2 ребра}
until (ke=0) or (l=8);                    if ke=0 then begin
                                         close(o);  halt end;
for j:=l+1 to ke-1 do write(o,'0 ');  write(o,'1'); close(o);
str(ke,s);  assign(ol,  s+'.RES');  rewrite(ol);
   k:=0;  repeat inc(k)    until kr[k]=1;    p[2]:=v[k];
p[1]:=0;  repeat inc(p[1]) until n[p[1]]>=k;
                                     for j:=3 to ke do begin
k:=n[p[j-1]-1];  repeat inc(k)
                  until (kr[k]=1) and (p[j-2]<>v[k]);
                                        p[j]:=v[k]      end;
        for j:=1 to ke do write(ol,p[j],' ');  close(ol) END.
```

3. Приватні інтереси. Запропонована задача — своєрідне узагальнення задачі "Гроші на шахівниці" (3-тя задача I етапу олімпіади 1998-99 навчального року) до задачі теорії ігор. Для клітини в j-му рядку та k-му стовпчику, позначимо через:

(j,k) — саму клітинку;

$a(j,k)$ — кількість золотих, розташованих на цій клітинці;

$c(j,k)$ — найбільшу кількість золотих, які може зібрати гравець, починаючи з того моменту, як він *ставить* пішак на дану клітинку, тобто разом з монетами даної клітинки;

$(j,k)'$ — клітину, на яку пішак подаде наступним ходом з даної клітини (в j-му рядку й k-му стовпчику);

$c(j,k)'$ — найбільшу кількість грошей, які може зібрати гравець, починаючи з того моменту, як він *ставить* пішак на клітинку $(j,k)'$. Якщо клитинка $(j;k)'$ невизначена (наприклад, коли клітинка $(j;k)$ розташована у першому рядку чи останньому стовпчику), вважаємо $c(j,k)'$=0.

$$c(1;j)=a(1;j), \qquad c(j;n)=a(j;n), \qquad c(2;n-1)=a(2;n-1),$$

— гра припиняється одразу після досягнення пішаком 1-го рядка чи останнього n-го стовпчика. З клітинки (j,k) пішак піде:

- праворуч у клітину $(j,k+1)$, якщо $c(j-1,k)<c(j,k+1)$, тоді $c(j,k)=a(j,k)+c(j,k+1)'$;
- вгору у клітинку $(j-1,k)$, якщо $c(j-1,k)\geq c(j,k+1)$, тоді $c(j,k)=a(j,k)+c(j-1,k)'$.

Таким чином, за відомими величинами $a(j,k)$ послідовно знаходяться величини $c(j,k)$ — спочатку в першому рядку, потім — у кожному наступному рядку справа наліво (у порядку спадання номера рядка). При цьому попередньо знаходять напрям руху пішака з клітин таблиці.

Проілюструємо знаходження результату гри для поданої в умові задачі таблиці значень $a(j,k)$, вказавши індексами знизу праворуч порядок обчислення величин $c(j,k)$ і напряму руху пішака з клітин (j,k)

$$\begin{pmatrix} 1 & 1 & 2 & 0 \\ 3 & 7 & 8 & 9 \\ 3 & 0 & 7 & 1 \\ 0 & 3 & 3 & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 1_1 & 1_1 & 2_1 & 0_1 \\ 11_3 \to & 16_2 \to & 8_1 \to & 9_1 \\ \uparrow & \uparrow & \uparrow & \\ 19_6 & 8_5 & 16_4 & 1_1 \\ \uparrow & \uparrow & \uparrow & \\ 11_9 & 19_8 & 11_7 & 1_1 \end{pmatrix}.$$

Подане розв'язання без труднощів узагальнюється на випадок:

- збільшення кількості гравців;
- ускладнення простору можливих станів (тор, лист Мьобіуса, множина клітинок неквадратної форми на площині чи на поверхні многогранника, багатовимірний простір).

Задача ілюструє проблему теорії ігор з багатьма переможцями (без випадкового втручання) з повною інформацією і без коаліцій (гравці не можуть домовитися про результат гри). Проблема полягає в неочевидності того, що вважати розумною поведінкою гравця. Все, що на перший погляд описує таку поведінку, подано в умові задачі (див. пункт д–є). Запропонований алгоритм лише втілює ці принципи. До чого це призвело? До того, що маючи прекрасну можливість розійтися мільйонерами, гравці розходяться: один — з двома золотими, а другий — з вітром у кишенях (тести 1–2). і лише тому, що кожний гравець:

- намагається отримати найбільший зиск;
- знає, чого прагне суперник;
- знає, що суперник знає, чого прагне гравець;
- знає, що суперник знає, що гравець знає, чого прагне суперник...

Саме така низка припущень призводить до можливості розглядати граф гри "від кінця". Зауважимо, що кількість простих речень у найдовшому припущенні дорівнює найбільшій кількості можливих послідовних ходів.

Пояснимо зміст сказаного, детально розібравши гру "з кінця" для тесту 1, коли вхідний файл має вигляд

```
2  6172
2.6166 6167 6168 6169 6170 6171 6172 0
0.6164 6165 6166 6167 6168 6169 6170 6171 .
```

1. Якщо пішак знаходиться на передостанній клітинці другого рядка, що містила 6170 золотих, то наступний хід буде зроблено вгору, бо $6172 > 6171$.
2. Якщо пішак знаходиться на тій клітинці другого рядка, що містила 6169 золотих, то наступний хід буде зроблено вгору. Інакше ще за один хід суперник припинить гру, і гравець отримає 6170 золотих замість 6171.
3. Якщо пішак знаходиться на тій клітинці другого рядка, що містила 6168 золотих, то наступний хід буде зроблено вгору. інакше ще за один хід суперник припинить гру (див. попереднє міркування), і гравець отримає 6169 золотих замість 6170...

Міркування можна продовжити до тих пір, поки не дійдемо висновку, що гру буде припинено першим же ходом. Тест 2 ідентичний тесту 1 з точністю до симетрії.

У поданій далі програмі:

- використано подання прямокутної таблиці лінійним масивов — клітинці (j, k) відповідає елемент лінійного масиву з індексом

$$l = (j-1)n + k - 1,$$

де j лежить в межах від 1 до m включно, k — в межах від 1 до n включно, l — в межах від 0 до $mn-1$ включно. Руху праворуч відповідає збільшення l на 1, руху вгору — зменшення l на величину n;

- використано булеву змінну `b[l]`, величина якої дорівнює `true`, якщо з відповідної клітини `l:=(j-1)*n+k-1` пішак іде праворуч.

```
const nmax=12345;
var b: array[0..nmax] of boolean;  j,k,l,m,n: word;
  a,c: array[0..nmax] of word;             o: text;  begin
                              {Зчитування даних}
assign(o,'PRIVAT.DAT');  reset(o);  readln(o,m,n);
for l:=0 to m*n-1 do read(o,a[l]);   close(o);
a[(m-1)*n]:=0;  a[n-1]:=0;
                                  {Знаходження масивів b і c}
for l:=0 to m*n-1 do c[l]:=a[l];
for j:=2 to m do for k:=n-1 downto 1 do                 begin
l:=(j-1)*n+k-1;  b[l]:=c[l-n]<c[l+1];    if b[l] then begin
if not b[l+1] and (k<n-1) then c[l]:=c[l]+c[l-n+1] else
if     b[l+1] and (k<n-1) then c[l]:=c[l]+c[l+2]        end
                                                   else begin
if     b[l-n] and (j>2) then c[l]:=c[l]+c[l-n+1] else
if not b[l-n] and (j>2) then c[l]:=c[l]+c[l-2*n]    end end;

{Визначення послідовності напрямків руху і запис відповіді}
assign(o,'PRIVAT.RES');  rewrite(o);  l:=(m-1)*n;
if b[l] then write(o,c[l+1]) else write(o,c[l-n]);
writeln(o,'   ',c[l]);
repeat if b[l] then begin write(o,'r'); l:=l+1 end
               else begin write(o,'u'); l:=l-n end
 until (l<n) or (l mod n =n-1);   close(o);   halt      end.
```

4. Вечірка. Запропонована задача навіяна відомими задачами для розвитку алгоритмічного мислення (див. Л.М. Лоповок, "Збірник математичних задач логічного характеру", Київ, "Радянська школа", 1972, 151 с.). Для її успішного розв'язання необхідно звернути увагу на таке (після *назв етапів* розв'язання подано опис використання змінних відповідною частиною поданої далі програми).

- *Опрацювання вхідного файлу:*
  `line` — зчитаний рядок вхідного файлу;
  `m` — кількість умов (рядків файлу PARTY.DAT);
  `n` — кількість учасників вечірки, тобто кільність різних імен (фахів), записаних у файлі PARTY.DAT;
  `n2:=n div 2`,
  `r[1]`, `r[2]` — відповідно перше і друге слова умови;
  `rel[i]` — символ умови `i` (`<`, `>` чи `|`);
  `b[j,i]=1` чи `b[j,i]=2`, якщо `j`-те слово умови `i` є іменем або назвою фаху відповідно;
  `nold[1]` та `nold[2]` — відповідно кількості *різних* імен та фахів, які вже зчитано з вхідного файлу;
  `old[1,k]` і `old[2,k]` — відповідно ім'я та фах, які були `k`-ими у порядку зчитування *різних* імен чи спеціальностей з вхідного файлу.

- *Визначення першого імені у записі відповіді шляхом кодування імен і порівняння рядкових змінних* (небхідність кодування викликана неузгодженністю кодів літер української абетки алфавітному порядку):
`kyr1, kyr2` — відповідно переліки великих і малих літер української абетки у алфавітному порядку;
`cod1, cod2` — "літерні" коди української абетки, чиї машинні коди узгоджені з алфавітним порядком *української* абетки;
`namecode[k]` — "літерний"код імені `old[1,k]`;
`first=old[1,nfirst]` — перше ім'я вихідного файлу.

- *Здійснення формально логічних висновків з умови задачі* — див. частину програми, прокоментовану словами

Використання умови i:

`old[1,dd[1,i]]`, `old[2,dd[2,i]]` — відповідно ім'я та фах, записані у відповіді на `i`-му місці;
`w[b[k,i],c[k,i]]=l`, якщо `k`-те слово умови `i` буде `l`-им іменем (`b[k,i]=1`) чи фахом (`b[k,i]=1`) у відповіді.

- Можливість розгляду всіх розташувань учасників вечірки за столом та всіх варіантів фахової спеціалізації кожного з них забезпечено *припущеннями різних рівнів*:
`level[i]=0`, якщо умову `i` не використано;
`k0` — кількість умов, використаних до останнього аналізу розташування учасників;
`k` — кількість всіх використаних умов;
`assume[levelnow]=j`,якщо припущення рівня `levelnow` робилять стосовно `j`-го імені чи фаху у відповіді;
`level[i]=levelnow`, якщо умову `i` використано для отримання безпосереднього наслідку умов вхідного файлу PARTY.DAT і припущень всіх рівнів в межах від 1 до `levelnow` включно. Несуперечливість припущенн цих рівнів перевіряється для всіх можливих припущень рівнів, що більші за `levelnow`.

- *Оптимізація перебору відкиданням продовження тих варіантів розташування, які неузгоджені з вхідними даними й умовою задачі* — див. змінну `contra`. При виявленні суперечностей:

  i) всі висновки з останнього припущення ігноруємо;

  ii) робимо наступне (з усіх можнивих) припущення рівня `levelnow` і продовжуємо аналіз можливого розташування учасників;

  iii) якщо крок ii) здійснити неможливо, то рівень припущення `levelnow` зменшуємо на 1;

iv) якщо `level>1`, то усуваємо суперечності, тобто виконуємо пункти i–iv), інакше припиняємо роботу.

Таким чином повністю відтворено алгоритм розв'язання логічних задач, що є традиційними на учнівських олімпіадах з математики навіть для учнів 8–9 класів.

```
const max=25;   label ANALYS,NEXT,NEXTLINE,NEXTASS,WRONG;
var rel: array     [1..max] of char;
b,c,d,w: array[1..2,1..max] of byte;
    old: array[1..2,1..max] of string;
   nold: array[1..2]        of byte;
   code: array     [1..max] of string;
      r: array[1..2]        of string;   contra: boolean;
  level,assume: array[1..max] of byte;
  kyr1,kyr2,cod1,cod2,line: string;         o: text;
i,j,k,l,m,n,k0,n2,nfirst,levelnow: byte; ll:shortint;
begin{program}                             {Таблиця кодування}
kyr1:='АБВГҐДЕЇЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ';
kyr2:='абвгґдеїжзиіїйклмнопрстуфхцчшщьюя';      nold[1]:=0;
cod1:='1234567ABCDEFGHIJKLMNOPQRSTUVWXYZ';      nold[2]:=0;
cod2:='1234567abcdefghijklmnopqrstuvwxyz';
                                           {Зчитування даних}
          i:=0;    assign(o,'PARTY.DAT');   reset(o);
NEXTLINE: i:=i+1;  readln(o,line);
l:=1;   r[1]:='';    while  line[l]<>' '  do begin
        r[1]:=r[1]+line[l];  l:=l+1            end;
        rel[i]:=line[l+1];
l:=l+3; r[2]:='';    while (line[l]<>'.')
                       and (line[l]<>',') do begin
        r[2]:=r[2]+line[l];  l:=l+1            end;
                                      for j:=1 to 2 do begin
b[j,i]:=2;  for k:=1 to 33 do if r[j][1]=kyr1[k] then begin
b[j,i]:=1;                                        break end;
for k:=1 to nold[b[j,i]] do if old[b[j,i],k]=r[j] then begin
c[j,i]:=k;                                        break end;
                                      if c[j,i]=0 then begin
nold[b[j,i]]:=nold[b[j,i]]+1;
 old[b[j,i],  nold[b[j,i]]]:=r[j];
     c[j,i]:= nold[b[j,i]]                          end end;
if line[l]=',' then goto NEXTLINE;  close(o);  m:=i;
                       n:=nold[1];  n2:=n div 2;
{Кодування ўмен}       for j:=1 to n do               begin
for i:=1 to 33 do if old[1,j][1]=kyr1[i] then         begin
code[j]:=cod1[i];                                  break end;
               l:=length(old[1,j]);  for k:=2 to l do begin
for i:=1 to 33 do if old[1,j][k]=kyr2[i] then         begin
code[j]:=code[j]+cod2[i];                  break end end end;
```

```
{Хто перший у перелŭку?}    l:=length(old[1,1]);  nfirst:=1;
                                      for k:=2 to n do begin
ll:=length(old[1,k]); if ll<l then j:=ll else j:=l; i:=1;
while (i<j) and (code[k][i]=code[nfirst][i]) do i:=i+1;
              if code[k][i]<code[nfirst][i] then      begin
                                   nfirst:=k;  l:=ll  end end;
for j:=1 to n do for k:=1 to 2 do begin
              d[k,j]:=0;  w[k,j]:=0 end;   d[1,1]:=nfirst;
for i:=1 to m do level[i]:=0;              w[1,nfirst]:=1;
      assign(o,'PARTY.RES');  rewrite(o);  contra:=false;
                                   levelnow:=1;
NEXT: k0:=0; for i:=1 to m do if level[i]>0 then k0:=k0+1;
      k:=k0; for i:=1 to m do if level[i]=0 then      begin

{Використання умови i}
                     l:=w[b[1,i],c[1,i]];  if l>0 then begin
if rel[i]='<' then ll:=l+1 else
if rel[i]='>' then ll:=l-1 else if rel[i]='|' then ll:=l-n2;
if ll<1 then ll:=ll+n else if ll>n then ll:=ll-n;
level[i]:=levelnow;   if d[b[2,i],ll]=0 then          begin
w[b[2,i],c[2,i]]:=ll;    d[b[2,i],ll]:=c[2,i];  k:=k+1  end
                                        else contra:=
(w[b[2,i],c[2,i]]<>ll) or (d[b[2,i],ll]<>c[2,i])        end;
l:=w[b[2,i],c[2,i]];   if (l>0) and (not contra) then begin
if rel[i]='>' then ll:=l+1 else
if rel[i]='<' then ll:=l-1 else if rel[i]='|' then ll:=l-n2;
if ll<1 then ll:=ll+n else if ll>n then ll:=ll-n;
level[i]:=levelnow;   if d[b[1,i],ll]=0 then          begin
w[b[1,i],c[1,i]]:=ll;    d[b[1,i],ll]:=c[1,i];  k:=k+1  end
                                        else contra:=
(w[b[1,i],c[1,i]]<>ll) or (d[b[1,i],ll]<>c[1,i])        end;
          if     contra then break                      end;
          if not contra then goto ANALYS else contra:=false;
WRONG:  for i:=1 to m do  if level[i]=levelnow   then begin
                            level[i]:=0;
        for k:=1 to 2 do  if w[b[k,i],c[k,i]]<>0 then begin
d[b[k,i],w[b[k,i],c[k,i]]]:=0;  w[b[k,i],c[k,i]]:=0 end end;
NEXTASS: i:=1;  while (level[i]<>0) and (i<=m) do i:=i+1;
                                      if i>m then     begin
if levelnow=2 then halt;levelnow:=levelnow-1;goto WRONG end;
         k:=b[1,i];     j:=assume[levelnow]+1;
while (d[k,j]<>0) and (j<=n) do j:=j+1;   if j>n then begin
if levelnow=2 then halt;levelnow:=levelnow-1;goto WRONG end;
{Нове припущення щодо умови i}   d[k,j]:=c[1,i];
           assume[levelnow]:=j;  w[k,c[1,i]]:=j;  goto NEXT;
{Якщо суперечностей нема...} ANALYS: if k>k0 then goto NEXT
```

```
                                  else if k<m  then     begin
{Якщо треба робити припущення...}
                                    levelnow:=levelnow+1;
j:=1;  while (d[1,j]<>0) and (d[2,j]<>0) do j:=j+1;
                    assume[levelnow]:=j-1;  goto NEXTASS end;
{Відповідь}         append(o);         for j:=1 to n do begin
          write(o,old[1,d[1,j]],' (',old[2,d[2,j]],')');
          if j<n then write(o,', ') else writeln(o,'.') end;
close(o);  goto WRONG                                    end.
```

Задачу можна ускладнити, запровадивши додаткові характеристики учасників. Наприклад, додатково до імен та фахів розглядати національність, темперамент, уподобання тощо. Це спричинить ускладнення опрацювання даних (великі та малі літери на початку слів гарантують коректність розподілу лише за двома характеристиками).

Долучення умов типу "При обході столу проти руху годинникової стрілки (якщо дивитися згори), між Тарасом і математиком сидить ще троє учасників вечірки" викличе незначні зміни в опрацюванні даних та перевірці узгодженності з умовою задачі. У вхідному файлі такі умови можна подати наступним чином

```
Тарас <<<< математик .
```

Задача істотно ускладнюється вимогою того, щоб умови вхідного та вихідного файлів формулювалися українською мовою (із зміною закінчень при відмінюванні, наявністю прийменників, наявністю присудків тощо).

5. Багатокутник. Побудуємо математичну модель задачі. Запровадимо декартову систему координат, у якій координати вершин початкового трикутника дорівнюють $(-1;0)$, $(1;0)$ та $(0,\sqrt{3})$. Для цілого k в межах від 0 до 5 включно позначимо

$$\vec{a}_k = \vec{a}_k \left(\cos\frac{k\pi}{3};\ \sin\frac{k\pi}{3} \right).$$

Зафіксуємо такий порядок обходу периметра отримуваних многокутників, щоб внутрішня область багатокутників залишалася ліворуч.

Нехай після n перетворень (натискань клавіші Enter) $(x;y)$ та $(x';y')$ — послідовні вершини отриманого $3{\cdot}4^n$-кутника ($n = 0, 1, 2, 3, 4$). Під час $(n{+}1)$-го перетворення відрізок, що сполучає точки $(x;y)$ та $(x';y')$, замінюється ламаною з 4-х ланок, що сполучає послідовно точки $(x;y)$, $(x_1;y_1)$, $(x_2;y_2)$, $(x_3;y_3)$, $(x';y')$, де

$$x_1 = \frac{2x+x'}{3}, \quad y_1 = \frac{2y+y'}{3},$$
$$x_3 = \frac{x+2x'}{3}, \quad y_3 = \frac{y+2y'}{3},$$

всі ланки цієї ламаної втричі коротші за відрізок, що сполучає точки $(x;y)$ і $(x';y')$, а точка $(x_2;y_2)$ лежить зовні багатокутника, отриманого в результаті попередніх n перетворень.

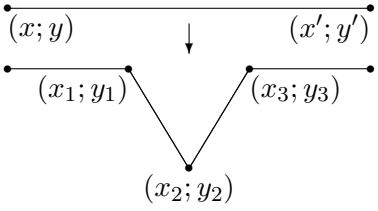


Рисунок 9: заміна відріка ламаною з 4-х ланок.

Якщо $\vec{a}_k$, $\vec{a}_{k_1}$, $\vec{a}_{k_2}$, $\vec{a}_{k_3}$ — напрямки руху (при обході периметру) з точок $(x;y)$, $(x_1;y_1)$, $(x_2;y_2)$ і $(x_3;y_3)$ відповідно, то

$$k_1 \overset{6}{\equiv} k-1, \qquad k_2 \overset{6}{\equiv} k+1, \qquad k_3 = k_1,$$

де $\overset{6}{\equiv}$ — відношення рівності остач при діленні на 6.

У поданій далі програмі спочатку знаходяться всі декартові координати вершин багатокутників, потім — їх екранні координати. Точки занумеровано у порядку обходу периметру останнього $3\cdot 4^5$-кутника, тому при побудові зображення $3\cdot 4^i$-кутника лічильник точок j змінюється з кроком 4^{5-i}.

```
uses graph;              const  nstep=5;  npoint=3*4*4*4*4*4;
var  x0,x1,y0,y1,dx,dy,d:real; i,j,j1,j2,j3,l,ll,nx,ny:word;
ix,iy:array[0..npoint] of word;   detect,graphmode: integer;
 x, y:array[0..npoint] of real;  co,si: array[0..5] of real;
    k:array[0..npoint] of byte;  n4:array[0..nstep] of word;
                                                      begin
{Напрями}                                 for j:=0 to 5 do begin
               co[j]:=cos(j*pi/3);  si[j]:=sin(j*pi/3)  end;
{Степені 4} n4[0]:=1; for j:=1 to nstep do n4[j]:=n4[j-1]*4;

{Координати вершин початкового трикутника}
x[0]:=-1;  x[n4[nstep]]:=1;  x[n4[nstep]*2]:=0;
y[0]:= 0;  y[n4[nstep]]:=0;  y[n4[nstep]*2]:=sqrt(3);
k[0]:= 0;  k[n4[nstep]]:=2;  k[n4[nstep]*2]:=4;
x[npoint]:=-1;  y[npoint]:=0;
x0:=-1;  y0:=-1/sqrt(3);  dx:=2;          {Границі малюнка}
x1:= 1;  y1:=sqrt(3);     dy:=4/sqrt(3);
 d:=2;                          for i:=0 to nstep-1 do begin
{Координати точок, що виникають після i-го перетворення}
 d:=d/3;  l:=n4[nstep-i];
 j:=0;   ll:=n4[nstep-i-1];       while j<npoint-1 do begin
j1:=j+ll;     x[j1]:=(2*x[j]+x[j+l])/3;
```

```
              y[j1]:=(2*y[j]+y[j+l])/3;
              if k[j]=0 then k[j1]:=5 else k[j1]:=k[j]-1;
j2:=j+ll*2;   x[j2]:=x[j1]+d*co[k[j1]];
              y[j2]:=y[j1]+d*si[k[j1]];
              if k[j]=5 then k[j2]:=0 else k[j2]:=k[j]+1;
j3:=j+ll*3;   x[j3] :=(x[j]+2*x[j+l])/3;
k[j3]:=k[j];  y[j3] :=(y[j]+2*y[j+l])/3;  j:=j+l    end end;
                                          {Приготування екрану}
detectgraph(detect,graphmode);
  initgraph(detect,graphmode,'c:\pascal\bgi');
nx:=getmaxx;  ny:=getmaxy;  setbkcolor(0);  setcolor(15);
if dx*ny<dy*nx then dx:=(dy*nx)/ny else
if dx*ny>dy*nx then dy:=(dx*ny)/nx;  x1:=x0+dx;  y1:=y0+dy;
                                  for j:=0 to npoint do begin
{Екранні координати j-о точки}
ix[j]:=trunc(((x[j]-x0)/dx)*nx);
iy[j]:=trunc(((y1-y[j])/dy)*ny);                          end;
                                   for i:=0 to nstep do begin
j:=0;  cleardevice;                   while j<npoint do begin
l:=j+n4[nstep-i];  line(ix[j],iy[j],ix[l],iy[l]);  j:=l end;
                                                  readln end;
closegraph                                                end.
```

6. Корінь квадратний. Запропоноване далі розв'язання ґрунтується на поданні десяткового дробу масивом його цифр (так звана"довга арифметика"). Для періодичного десяткового дробу позначимо через:

k_0 — кількість цифр до десяткової крапки (якщо рахувати зліва направо);

k_1 — кількість цифр до періоду;

k_2 — кількість всіх цифр у записі;

$\{a_j\}_{j=1}^{k_2}$ — послідовність всіх його цифр (якщо рахувати зліва направо).

Додатній *скінченний* десятковий дріб ($k_1{=}k_2$) є відношенням натуральних чисел

$$\overline{a_1\cdots a_{k_0}.a_{k_0+1}\cdots a_{k_1}} = \overline{a_1\cdots a_{k_1}} : 10^{k_1-k_0}.$$

Тут і надалі риска над послідовністю цифр вказує на запис у десятковій системі числення, а не на добуток цифр.

Додатній *періодичний* десятковий дріб також є відношенням натуральних чисел

$$\overline{a_1\cdots a_{k_0}.a_{k_0+1}\cdots a_{k_1}(a_{k_1+1}\cdots a_{k_2})} =$$

$$= (\overline{a_1\cdots a_{k_0}a_{k_0+1}\cdots a_{k_1}} +$$

$$+\overline{a_{k_1+1}\cdots a_{k_2}}\cdot(10^{k_1-k_2}+10^{2(k_1-k_2)}+\cdots))\cdot 10^{k_0-k_1}$$

$$=\left(\overline{a_1\cdots a_{k_1}}+\overline{a_{k_1+1}\cdots a_{k_2}}\cdot\frac{10^{k_1-k_2}}{1-10^{k_1-k_2}}\right)\cdot 10^{k_0-k_1}=$$

$$=\left(\overline{a_1\cdots a_{k_1}}\cdot(10^{k_2-k_1}-1)+\overline{a_{k_1+1}\cdots a_{k_2}}\right):$$
$$:\left((10^{k_2-k_1}-1)\cdot 10^{k_1-k_0}\right).$$

Поділивши чисельник і знаменник отриманого дробу на їх найбільший спільний дільник, отримаємо подання нескоротним дробом.

Нескоротний дріб є квадратом раціонального числа[10] тоді й лише тоді, коли його чисельник і знаменник є квадратами (взаємно простих) цілих чисел.

Десятковий *періодичний* дріб не є квадратом раціонального числа, якщо в нього кількість цифр від десяткової крапки до періоду (k_1-k_0) — непарне число (див. розв'язання задачі II.2).

Таким чином, задача має наступний *алгоритм* розв'язання.

1. Зчитуємо десятковий запис дробу, встановлюючи величини k_0, k_1, k_2 і послідовність цифр у записі $\{a_j\}_{j=1}^{k_2}$.

2. Якщо (k_1-k_0) — парне ціле число, то:

   - подаємо даний десятковий дріб нескоротним дробом, використавши алгоритм Евкліда знаходження найбільшого спільного дільника чисельника і знаменника дробу;
   - встановлюємо, чи є отримані чисельник і знаменник одночасно квадратами натуральних чисел;
   - якщо відповідь ствердна, подаємо відношення квадратних коренів чисельника і знаменника періодичним десятковим дробом, записуємо відповідь і припиняємо виконання алгоритму.

3. Знаходимо шуканий квадратний корінь згідно описаного алгоритму;

4. Записуємо відповідь і припиняємо виконання алгоритму.

Таким чином, *повне* розв'язання даної задачі містить у собі підзадачу подання нескоротного дробу десятковим дробом. Відповідну частину програми запозичено з розв'язання задачі II.2 "Десятковий дріб" (там же див. опис відповідних змінних). Звичайно, це найгроміздкіша частина програми, без якої неможливо пройти успішно тести 11–12. Зауважимо, що вилучення процедур `divide, gcd` і групи операторів

[10]Кожне раціональне число можна подати як нескоротним дробом з цілим чисельником і натуральним знаменником, так і періодичним десятковим дробом.

`if (k2>k1)and(k1-k0=2*(k1-k0) div 2)) then begin...end;`
істотно скорочує код. Для успішного проходження тестів 9–10 у цьому випадку потрібно алгоритм з поданням чисел масивами цифр замінити алгоритмом з використанням базових типів мови програмування.

Подана програмі містить мітки операторів зчитування:

`EMPTY` — прогалин;

`BEFORE` — цифр до десяткової крапки;

`AFTER` — цифр після десяткової крапки, але перед періодом;

`BRACKETS` — цифр періоду;

`START` — кількості цифр після крапки, якщо корінь квадратний не є раціональним числом.

У поданій програмі ідентифікатори (описано лише частину) мають наступний зміст:

`k0` — кількість цифр до десяткової крапки (якщо рахувати зліва направо) у першому числі файлу ROOT.DAT;

`k1` — кількість цифр до періоду у цьому ж числі;

`k2` — кількість всіх цифр у записі першого числа файлу ROOT.DAT;

`a` — масив цифр першого числа файлу ROOT.DAT, з якого треба добути квадратний корінь;

`m0` — кількість цифр до десяткової крапки (якщо рахувати зліва направо) у шуканому числі;

`m1` — кількість цифр до періоду у цьому ж числі;

`num` — масив цифр чисельника нескоротного дробу, з якого потрібно добути квадратний корінь;

`den` — масив цифр знаменника цього ж дробу;

`q` — масив цифр частки, знайденої за допомогою процедури `divide`;

`r` — масив цифр лишку, знайденої за допомогою процедури `divide`;

`l*` — кількість цифр числа, які (цифри) подаються масивом `*`.

```
const max=1000;   label AFTER,BEFORE,BRACKETS,EMPTY,START;
type  carray=array[0..max] of integer;    var i,j,k,la,lb,
lc,ld,lr,lq,lnum,lden,nroot,k0,k1,k2,m0,m1,n0,n2,n5: word;
a,b,c,d,q,r,num,den: carray;    code: integer;    o: text;
                      period,zero: boolean;  s: string[1];
procedure root
(a:carray;var b:carray;k0,k1,k2,nroot:word;var m0,m1:word);
{Знаходження кореня квадратного (m1 цифр у масиві b, з яких
m0 -  до десяткової крапки)  десяткового  дробу  (k2 цифр у
масиві a, з яких k0 - до крапки, k1 - до періоду)}
                                    label NEXTb;     begin
        m0:=k0 div 2;  n5:=k2-k1;
if k0=2*m0 then begin k:=2;  i:=10*a[1]+a[2]    end
           else begin k:=1;  i:=a[1];  m0:=m0+1 end;
b[1]:=trunc(sqrt(i));          r[1]:=0;  r[k]:=i-b[1]*b[1];
               if (m0=1) and (k=2) and (r[2]=0) then begin
m1:=m0;   zero:=true;  exit                              end;
m1:=m0+nroot; if (k2=k1) and ((k1-k0) div 2 > nroot)  then
m1:=m0+((k1-k0) div 2);           for i:=2 to m1 do  begin
k:=k+1;  if k<=k1 then r[k]:=a[k] else
         if k2>k1 then r[k]:=a[k1+1+(k-k1-1) mod n5]
                  else r[k]:=0;
k:=k+1;  if k<=k1 then r[k]:=a[k] else
         if k2>k1 then r[k]:=a[k1+1+(k-k1-1) mod n5]
                  else r[k]:=0;            b[i]:=10;
NEXTb: b[i]:=b[i]-1; q[k]:=r[k]-b[i]*b[i];
for j:=1     to k-i do q[j]:=r[j];
for j:=k-i+1 to k-1 do q[j]:=r[j]-2*b[j+i-k]*b[i];
for j:=k   downto 2 do while q[j]<0 do                  begin
                      q[j]:=q[j]+10;  q[j-1]:=q[j-1]-1 end;
if (q[1]<0) and (b[i]>0) then goto NEXTb;
   zero:= (k>=k1) and (k2=k1);
if zero then for j:=1 to k do zero:=zero and (q[j]=0);
if zero then begin m1:=i; break end;
for j:=1 to 2*i do r[j]:=q[j]                        end end;

procedure divide(a,b: carray; la,lb: word);     label NEXTi;
{Знаходження частки (lq цифр у масиві q) і остачі
(lr цифр у масиві r) від ділення одного числа
(la цифр у масиві a) на інше (lb цифр у масиві b)}    begin
                                        if la<lb then begin
for j:=1 to la do r[j]:=a[j]; lr:=la;  lq:=1;  q[1]:=0  end
                                                 else begin
for j:=1 to lb do r[j]:=a[j];          lq:=0; r[0]:=0; i:=0;
                           {Знаходження k - цифри частки q}
NEXTi: i:=i+1;  k:=0;                while r[0]>=0 do  begin
                                  for j:=lb downto 1 do begin
```

```
       r[j]:=r[j]-b[j];                if r[j]<0 then begin
       r[j]:=r[j]+10;  r[j-1]:=r[j-1]-1             end end;
                              if r[0]>=0 then k:=k+1  end;
                              for j:=lb downto 1 do begin
       r[j]:=r[j]+b[j];                if r[j]>9 then begin
       r[j]:=r[j]-10;  r[j-1]:=r[j-1]+1             end end;
if (k>0) or (i>1) then begin lq:=lq+1; q[lq]:=k end;
if    i+lb<=la    then                                  begin
for j:=1 to lb do r[j-1]:=r[j]; r[lb]:=a[i+lb];  goto NEXTi
                                                      end;
   lr:=lb;  while (r[lb-lr+1]=0) and (lr>1) do lr:=lr-1;
if lr< lb   then for j:=1 to lr do r[j]:=r[j+lb-lr]     end;
if lq= 0    then begin lq:=1; q[1]:=0 end               end;

procedure gcd(var a,b:carray; var la,lb:word); {Знаходження
найбільшого спільного дільника (la  цифр у масиві  a  після
виконання підпрограми) натуральних чисел (la  цифр у масиві
a i lb цифр у масиві b на початку виконання програми)}begin
lr:=2;              while not ((lr=1) and (r[1]=0)) do begin
divide(a,b,la,lb);
la:=lb;  for j:=1 to la do a[j]:=b[j];
lb:=lr;  for j:=1 to lb do b[j]:=r[j]               end end;

begin{program}              {Зчитування і опрацювання даних}
assign(o,'ROOT.DAT');  reset(o);  k0:=0;  k1:=0;  k2:=0;
EMPTY:  read(o,s);  if s=' ' then goto EMPTY;
                    i:=1; val(s,a[i],code);
BEFORE: read(o,s);
if  s=' '  then  begin  k0:=i;  goto START  end else
if  s='.'  then  begin  k0:=i;  goto AFTER  end else
begin i:=i+1; val(s,a[i],code); goto BEFORE end;
AFTER:  read(o,s);
if  s=' '  then  begin  k1:=i;  goto START    end else
if  s='('  then  begin  k1:=i;  goto BRACKETS end else
begin i:=i+1; val(s,a[i],code); goto AFTER    end;
BRACKETS: read(o,s);
if  s=')'  then  begin  k2:=i;  goto START    end else
begin i:=i+1; val(s,a[i],code); goto BRACKETS end;
START: readln(o,nroot);  close(o);  if k1=0 then k1:=k0;
                                    if k2=0 then k2:=k1;
assign(o,'ROOT.RES');  rewrite(o);
if (k2>k1) and (k1-k0=2*((k1-k0) div 2)) then         begin
{Якщо десятковий дріб періодичний,
а кількість цифр від крапки до дужки парна...}
for j:=1 to k2 do num[j]:=a[j];            {Чисельник}
for j:=k2 downto k2-k1+1 do                           begin
num[j]:=num[j]-a[j+k1-k2];           if num[j]<0 then begin
```

```
num[j]:=num[j]+10;  num[j-1]:=num[j-1]-1            end end;
i:=0; for j:=1 to k2 do if num[j]<>0 then break else i:=j;
if i<>0 then for j:=1 to k2-i do num[j]:=num[j+i];
lnum:=k2-i;  ld:=lnum;  lden:=k2-k0;  lc:=lden;
                                                  {Знаменник}
for j:=1       to k2-k1 do den[j]:=9;
for j:=k2-k1+1 to lden  do den[j]:=0;
for j:=1 to lnum do d[j]:=num[j];
for j:=1 to lden do c[j]:=den[j];    gcd(d,c,ld,lc);
divide(num,d,lnum,ld);
lnum:=lq;  for j:=1 to lq do num[j]:=q[j];
divide(den,d,lden,ld);
lden:=lq; for j:=1 to lq do den[j]:=q[j];
root(num,c,lnum,lnum,lnum,0,lc,m1);     if zero then begin
root(den,d,lden,lden,lden,0,ld,m1);     if zero then begin
{Якщо корінь квадратний і раціональним числом...}
la:=lc;  for j:=1 to la do a[j]:=c[j];
lb:=ld;  for j:=1 to lb do b[j]:=d[j];

{Знаходження n2 - степеня 2 у розкладі a на прості множники}
n2:=0;  den[1]:=2;          lr:=1;      r[1]:=0;
                      while (lr =1) and (r[1] =0) do   begin
divide(b,den,lb,1);     if (lr =1) and (r[1] =0) then begin
n2:=n2+1;  lb:=lq;   for j:=1 to lb do b[j]:= q[j]  end end;

{Знаходження n5 - степеня 5 у розкладі a на прості множники}
n5:=0;  den[1]:=5;          lr:=1;      r[1]:=0;
                      while (lr =1) and (r[1] =0) do   begin
divide(b,den,lb,1);     if (lr =1) and (r[1] =0) then begin
n5:=n5+1;  lb:=lq;   for j:=1 to lb do b[j]:= q[j]  end end;

               {Знаходження n0 - кількості цифр до періода}
if n2<n5 then n0:=n5 else n0:=n2;
                     {Знаходження і запис цілої частини}
divide(c,d,lc,ld);
for j:=1 to lq do write(o,q[j]);  write(o,'.');

    {Знаходження і запис цифр після крапки, але до періода}
                                  for la:=1 to n0 do begin
for j:=1 to lr do c[j]:=r[j];  lc:=lr+1;  c[lc]:=0;
divide(c,d,lc,ld);  write(o,q[1])                     end;

{Знаходження періоду}      if (lr>1) or (r[1]>0) then begin
write(o,'(');  la:=lr;  for j:=1 to lr do a[j]:=r[j];
period:=true;                          while period do begin
for j:=1 to lr do c[j]:=r[j];  lc:=lr+1;  c[lc]:=0;
divide(c,d,lc,ld);   write(o,q[1]);     if lr=la then begin
```

```
                    j:=0; while period and (j<lr) do begin
                    j:=j+1;     period:=r[j]=a[j]      end;
                                period:=not period end end;
writeln(o,')')                                         end;
                                close(o);  halt end end end;
root(a,b,k0,k1,k2,nroot,m0,m1);
if (k2=k1) and not zero then m1:=m0+nroot;
for j:=1 to m0 do write(o,b[j]);        if m1>m0 then begin
write  (o,'.');  for j:=m0+1 to m1 do write(o,b[j])     end;
                                close(o)               end.
```