Київський міжрегіональний інститут удосконалення вчителів
імені Бориса Грінченка

Олександр Рудик

Олімпіада з основ інформатики та обчислювальної техніки 1998–99 навчального року в Київській області

Київ 1999

УДК 372.800.2: 371.26.035.461
ББК 74.263.2
0–54

Рекомендовано Науково-методичною радою
КМІУВ ім. Б.Грінченка
(протокол No 7 від 16 березня 1999 року).

Рецензенти:
А.Ф.Верлань, професор, доктор технічних наук;
В.Б.Распопов, доцент, кандидат фізико-математичних наук.

О.Рудик. Олімпіада з основ інформатики та обчислювальної техніки 1998–99 навчального року в Київській області. — К: КМІУВ ім. Б.Грінченка, 1999. — 112 с.

Посібник складається з Передмови і трьох розділів — з детальним аналізом задач, які пропонувалися відповідно на I–III етапах проведення олімпіади з ОІОТ в Київській області в 1998-99 навчальному році. Кожний розділ має аналогічну структуру: рекомендації щодо умов проведення олімпіади, завдання для учнів, вказівки для перевірки для організаторів олімпіади (опис тестів), авторське розв'язання з детальним аналізом алгоритмів і прикладами програм мовою програмування Pascal 7.0.

Посібник дає змогу учням і вчителям стандартизувати на високому, методично-виваженому рівні підготовку до олімпіади. Книжку написано лаконічною і водночас зрозумілою мовою. Робота над посібником буде корисною учням, які захоплюються математикою і бажають розширити свої уявлення про математичні моделі. Він також буде корисним майбутнім програмістам, бо на прикладах, детально описаних у посібнику, можна з успіхом навчитися культурі програмування.

ISBN 966–7548–00–7

Передмова

Олімпіада з основ інформатики та обчислювальної — найлегша й одночасно найважча серед інших учнівських одімпіад. Таке парадоксальне на перший погляд твердження зовсім не спроба заінтригувати 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 з них, вказавши у дужках римськими цифрами номер етапу, а арабськими — номер задачі відповідного етапу олімпіади 1998–99 навчального року, розв'язання якої (задачі) вимагає відповідних знань, умінь і навичок:

- як реалізувати найпростіші задачі перебору — знайти період підстановки (I.2), перебрати всі перестановки, розташування чи комбінації (II.2, III.6);
- як оптимізувати перебір, наприклад, методом динамічного програмування (I.3, III.1, III.6);
- як програмно реалізувати невідому кількість вкладених один в одного циклів без запису кожного такого циклу окремою групою операторів (II.2, III.6);
- як використовувати рекурсію (I.3);
- елементи теорії графів (III.5);
- елементи теорії цілих чисел: відношення подільності, системи числення (I.2, II.1–2, III.6);
- елементи теорії ігор (аналіз "від кінця" ігор з повною інформацією) хоча б на прикладі ігор Баше й нім;
- елементи математичної логіки (III.6);
- розпізнавання або створення образів примітивної графіки (II.3, III.3);
- як програмно реалізувати метод координат, використовувати властивості геометричних тіл (II.1, III.2, III.4).

В 1999–2000 навчальному році на I–III етапах планується пропонувати учням задачі, у яких вирішальну роль матиме:

- створення алгоритму числових розрахунків з поданням числа масивом його цифр;
- реалізація гри й пошук виграшної стратегії;

- створення алгоритму розв'язування логічних задач.

Для підготовленого учасника олімпіади виконання олімпіадного завдання зводиться до:

- розпізнавання того, яким чином умова завдання за потоком слів ховає якусь з моделей або їх поєднання;

- демонстрації набутих навичок програмування.

Намагання залишити кожній дитині можливість відкрити щось нове, залишаючись у межах знань учнів середніх шкіл майже нездійсненне. Адже алгоритмічні задачі розглядалися і розв'язувалися протягом тисячоліть, задовго до створення обчислювальної техніки. За дійсно нові, важливі для суспільства математичні моделі та алгоритмиприсуджують наукові ступені, державні та міжнародні премії!

Серед поданих далі завдань I–III етапів є завдання минулих років олімпіад різного рівня із зміненими технічними умовами і викладом сюжету. Переслідуємо суто виховну мету. Навіть маючи на руках тести для перевірки та знаючи ідею розв'язання, не всім вистачає волі подолати шлях до повного розв'язання, навіть не будучи обмеженим у часі. У аналогічних ситуаціях шкільний вчитель пропонує на контрольній роботі задачі домашнього завдання.

Нажаль, чинні програми курсу ОІОТ не передбачають повноцінного ознайомлення з переліченими раніше темами. Щиро кажучи, вивчення теми "Основи алгоритмізації і програмування" грунтується на ілюстрації дії операторів алгоритмічних мов програмування, а не на розв'язуванні власне алгоритмічних завдань. Навіть найпростіші завдання "визначити вид трикутника за градусною мірою двох його кутів" чи "описати у довільній формі алгоритм розв'язання рівняння" $ax^2 + bx + c = 0$ викликають масу труднощів у випускників наших шкіл.

Наступний графік подає результати IV етапу Всеукраїнської учнівської олімпіади з основ інформатики та обчислювальної техніки в 1998 році. По вертикалі відкладено бали (шкала рівномірна), по горизонталі — номери учасників у робочому протоколі журі. Рівні 115 балів, 88 балів і 48 балів відділяють результати призерів, які отримали дипломи різного ступеня. 11/12 учасників не спромоглися набрати й половини від максимального досягнутого результату 260 балів з 500 балів, а ті що змогли, отримали дипломи I ступеня. І це незважаючи на те, що орієнтовні завдання Міністерства освіти, за якими проводило III етап багато областей, були (на думку автора) складнішими від запропонованих на єдиному турі IV етапу.

Мета посібника — ознайомити широке коло читачів (учнів, вчителів, студентів педагогічних університетів) з типовими й оригінальними умовами завдань, тестами для перевірки розв'язання та авторським баченням оптимального розв'язання. Цим автор прагне створити передумови успішного опанування методами розв'язування алгоритмічних задач, можливо, і самостійного.

Рекомендований спосіб роботи з посібником.

1. Ознайомтеся з умовами завдань певного етапу, впорядкуйте їх за зростанням складності задач. І саме в цій послідовності розв'язуйте задачі. Виняток 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 математичну модель (співвідношення між параметрами) та алгоритм розв'язання задачі. Продумайте власну систему тестування. Для початківців бажано все це зафіксувати на папері. І чим менший досвід *успішного* розв'язування алгоритмічних 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. Після усвідомлення моделі й алгоритму розв'язання вмикайте ПК і беріться за програмну реалізацію. Створений Вами варіант програми 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. Зверніть увагу на вимоги до часу роботи програми. Для IBM386 25MHz компіляція і виконання авторської програми потребують:

   - для всіх тестів задач I-го етапу — до 2 секунд;
   - для тестів задач II-го етапу — до 3 секунд, крім 10-го тесту 3-ої задачі, який потребує до 10 секунд;
   - для тестів задач III-го етапу — до 5 секунд, крім 2-го тесту 5-ої задачі, який потребує до 12 секунд.

   Якщо знаєте як, удоскональте власну програму так, щоб вона змогла пройти 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 змогла пройти 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. 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. Пам'ятайте, що 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 зрозуміло, в чому 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.

Успіхів Вам, шановні читачі!

1 I етап

1.1 Умови проведення

Олімпіаду рекомендуємо провести в один практичний тур тривалістю 5 (п'ять) астрономічних годин (після ознайомлення з умовами, переписування умов.) Коректність даних перевіряти не потрібно.

1.2 Орієнтовні завдання

1. Нова пошта, 10 балів. Містечко Друатвіль (французькою droite — пряма, ville — місто) розташоване у гірській ущелині вздовж шляху, навколо якого майже непрохідні гори. Мешканці містечка задумали побудувати нове приміщення пошти таким чином, щоб якнайменше стоптувати черевики. Створіть програму DROITE.* , яка визначить місце розташування нового приміщення пошти, для якого сума добутків відстаней від пошти до осель вздовж шляху на кількість мешканців цих осель найменша.

Вхідний файл DROITE.DAT у першому рядку містить натуральне число n — кількість осель у місті. Для k в межах від 1 до n включно $(k+1)$-ий рядок цього самого файлу містить (у вказаному порядку) два натуральних числа: x_k — відстань від пам'ятного знаку на в'їзді у місто до k-ої оселі — й m_k — кількість її мешканців. Всі числа вхідного файлу і загальна кількість мешканців міста не перевищують 65432, а кількість помешкань n не перевищує 234. Наприклад,

```
3
10 3
20 2
30 3
```

Вихідний файл DROITE.RES має містити відповідь українською мовою (чи мовою навчання, граматичні помилки та літературний стиль на кількість балів не впливають). Наприклад,

```
Шукане місце - за 20 кроків від знаку.
```

2. Сновида, 10 балів. У президента Першого національного банку майора Томаса Б. Кінґмена (героя оповідання О'Генрі "Товариші із Сан-Розаріо") з'явилася звичка вночі перекладати вміст сейфів, у яких клієнти зберігають свої коштовності. Створіть програму SLEEP.*, яка допоможе вирахувати, через скільки діб всі коштовності *вперше* повернуться на свої місця, якщо:

- щоночі майор Кінґмен перекладатиме вміст сейфів однаковим чином;
- відомо, що десятковий запис шуканого числа містить не більше 80-ти цифр.

Перший рядок вхідного файлу SLEEP.DAT містить одне натуральне число n — кількість сейфів банку, що не перевищує 600. Другий рядок цього ж файлу містить послідовність n різних натуральних чисел в межах від 1 до n включно. k-ий член цієї послідовності — номер сейфу, куди майор перекладає вміст k-го сейфу першої ночі. Наприклад,

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

Єдиний рядок вихідного файлу SLEEP.DAT має містити запис у десятковій системі числення шуканого числа діб (починаючи з першої позиції). Наприклад,

```
6
```

3. Гроші на шахівниці, 10 балів. На кожній клітинці шахівниці розміру $m \times n$ клітин випадковим чином розкладено монети, загальна вартість яких не перевищує \$3333. Пішак рухається або праворуч, або вгору (на одне поле за один хід) від нижнього лівого поля до верхнього правого. З усіх клітин, на яких побував пішак, знімають монети. Створіть програму MONEY.*, яка допоможе таким чином зібрати найбільшу кількість монет.

Перший рядок вхідного файлу MONEY.DAT містить у вказаному порядку натуральні числа m і n, менші за 22. Для j в межах від 1 до m включно $(j+1)$-ий рядок цього ж файлу містить сукупні вартості монет у клітинках j-го рядка шахівниці, якщо рахувати рядки згори донизу, у тому ж порядку, як вони (клітинки цього рядка) розташовані на шахівниці. Наприклад,

```
2 3
1 2 3
6 8 2
```

Перший рядок вихідного файлу MONEY.RES має містити найбільшу кількість монет, яку можна зібрати таким чином. Наступний рядок цього ж файлу містить $n + m - 2$ символи, що описують напрям ходів пішака (від першого до останнього): u — вгору (від анєлійського up), r — праворуч (від анєлійського right), які дають можливість зібрати найбільше грошей, і в якій хід праворуч здійснюється якомога пізніше на кожній ділянці шляху. Наприклад,

```
19
rur
```

1.3 Вказівки до перевірки

Якщо є можливість, перевірку тестів треба здійснити у присутності учнів. Тоді необхідність у апеляції відпадає взагалі.

1. Нова пошта. За перші 6 тестів — по 1 балу, за 7-й і 8-ий — по 2. Перевіряють:

- можливість дати правильну відповідь, коли шукана множина одноелементна (1-ий, 2-ий, 3-ій та 7-ий тести) або є відрізком (4-ий, 5-ий, 6-ий та 8-ий тести);
- впорядкування невпорядкованих масивів даних (2-ий, 4-ий, 7-ий та 8-ий тести);
- агрегування даних, тобто врахування того, що різні будівлі можуть мати однакову відстань до знаку (3-ій, 6-ий, 7-ий та 8-ий тести);
- проведення розрахунків без обчислення функції, для якої знаходяться точки з найменшими значеннями, або використання змінних із збільшеною кількістю цифр (7-ий та 8-ий тести).

1.1–3. Якщо файл DROITE.DAT має вигляд

або
```
5
1 4
2 5
6 2
8 4
9 5
```
,
або
```
5
9 5
1 4
6 2
2 5
8 4
```
,
або
```
9
1 2
1 2
2 2
2 3
6 2
8 3
8 1
9 4
9 1
```
,

то DROITE.RES має містити висловлювання, еквівалентне наступному.

Шукане місце — за 6 кроків від знаку.

1.4–6. Якщо файл DROITE.DAT має вигляд

або
```
5
1 4
2 5
6 2
8 4
9 7
```
,
або
```
5
8 4
1 4
9 7
2 5
6 2
```
,
або
```
9
9 1
1 2
8 3
1 2
9 4
2 2
6 2
2 3
8 3
```
,

то DROITE.RES має містити висловлювання, еквівалентне наступному.

Шукане місце — в межах від 6 до 8 кроків від знаку.

1.7. Файл DROITE.DAT для 7-го тесту створюють таким чином. Створюємо файл, в якому для j в межах від 1 до 200 включно j-ий рядок містить два числа: 65000+j та j. Далі переставляємо рядки цього файлу: після рядків, які закінчуються числами 10, 20, 30, 40, 50, 60, 70, 90 і 100 вставляємо групи по 10 рядків, які закінчуються числами 101–110, 111–120, 121–130, 131–140, 141–150, 151–160, 161–170, 171–180, 181–190 і 191–200 відповідно. Перші 30 рядків новоствореного файлу копіюємо в кінець файлу, і вставляємо перший рядок, який містить єдине число 230. Отримуємо файл вигляду (подано у 3 стовпчики, де 2-ий є продовженням 1-го, а 3-ій — продовженням 2-го, крапками позначено групи по 8 рядків, кожне число яких на 1 більше від відповідного числа попереднього рядка).

230	65140 140	
65001 1	65041 41	65180 180
.....	.....	65081 81
65010 10	65050 50	.....
65101 101	65141 141	65090 90
.....	.....	65181 181
65110 110	65150 150	.....
65011 11	65051 51	65190 190
.....	.....	65091 91
65020 20	65060 60	.....
65111 111	65151 151	65100 100
.....	.....	65191 191
65120 120	65160 160	.....
65021 21	65061 61	65200 200
.....	.....	65001 1
65030 30	65070 70	.....
65121 121	65161 161	65010 10
.....	.....	65101 101
65130 130	65170 170	.....
65031 31	65071 71	65110 110
.....	.....	65011 11
65040 40	65080 80	.....
65131 131	65171 171	65020 20
.....	.....	

Файл DROITE.RES для 7-го тесту має містити висловлювання, еквівалентне наступному.

Шукане місце — за 65137 кроків від знаку.

1.8. Файл DROITE.DAT 8-го тесту отримується з відповідного файлу для 7-го файлу дописуванням останнього рядка

і виправленням першого числа (першого рядка) з 230 на 231. У цьому випадку файл DROITE.RES має містити висловлювання, еквівалентне наступному. Шукане місце — в межах від 65137 до 65138 кроків від знаку.

2. Сновида. За 1-ий та 2-ий тест — по 1 балу, за 3-ій — 2 бали, за 4-ий і 5-ий — по 3 бали. Перевіряють:

- можливість подати перестановку добутком циклів;
- знаходження періоду перестановки як найменшого спільного кратного довжин знайдених циклів (3-ій та 5-ий тести);
- подання шуканого числа масивом його цифр ("довга арифметика", 4-ий та 5-ий тести).

2.1. Якщо файл SLEEP.DAT має вигляд

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

то файл SLEEP.RES має містити число 1.

2.2. Якщо файл SLEEP.DAT має вигляд

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

то файл SLEEP.RES має містити число 60 — добуток взаємно простих чисел 4, 3 і 5.

2.3. Якщо файл SLEEP.DAT має вигляд

```
25
2 3 4 1 6 7 8 9 10 5
12 13 14 15 16 17 18 19 20 21 22 23 24 25 11 ,
```

то файл SLEEP.RES має містити число 60 — найменше спільне кратне чисел 4, 6 і 15.

2.4. Якщо файл SLEEP.DAT має вигляд

```
501
      2    1   4   5    3   7..10    6  12..17   11
 19..28   18  30..41   29  43..58   42
 60..77   59  79..100  78 102..129 101 131..160 130
162..197 161 199..238 198 240..281 239 283..328 282
330..381 329 383..440 382 442..501 441
```

($m..n$ тут позначає послідовність послідовних натуральних чисел в межах від m до n включно, розділених пропусками), то файл SLEEP.RES має містити число 117288381359406970983270 — добуток простих чисел 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59 і 61.

2.5. Файл SLEEP.DAT для 5-го має наступний вигляд

```
552
  2   3    1   5..9     4  11..17   10
 19..28   18  30..41   29  43..58   42
 60..77   59  79..100  78 102..129 101 131..160 130
162..197 161 199..238 198 240..281 239 283..328 282
330..379 329 381..432 380 434..491 433 493..552 492 .
```

У цьому випадку файл SLEEP.RES має містити число 13404386441075082398088 — найменше спільне кратне чисел 3, 6, 8, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 51, 53, 59 і 61.

3. Гроші на шахівниці. За 1-ий та 2-ий тест — по 4 і 6 балів відповідно. Якщо відповідь правильна лише наполовину (з поданою відповіддю співпадає лише перший або другий рядок файлу MONEY.DAT) то присуджується половина балів — по 2 і 3 бали відповідно. 2-ий тест перевіряє оптимальність перебору — застосування методу динамічного програмування.
3.1. Якщо файл MONEY.DAT має вигляд

```
5 5
   1   1  22   6   7
  17   8   5  10  11
   3  12   2  22   2
   8  23  10  20   9
  18  22  18   8   5,
```

то файл MONEY.RES має містити записи

```
143
rurruuru
```

3.2. Якщо файл MONEY.DAT має вигляд

```
20 20
 1  4 86 21 28 67 32 17 37 43  9 48  7 84  6 30 91 37 77 33
70 84 72 31 17 33 47 25 82 28 48 15 87 29 77 97 49 88 82  3
14 15 50  3 59  1 77 65 77 71 56 21 68 59 95 64 99 25 67 30
 9 77 50 88 51 57 95 68 34  1 70 98 77 75 20 15 91 78 59 86
69 29 10 63 28 87 16 28 54 96 18 16 27 18 58 50 29 16 61 74
```

```
74  8  8  6 11 79 80 38 61 44 74 49 34 82 81 14 10  7  6 20
67 23 57 26 37 83 32 32 65 69 90 32 48 30 76 74 55 72 41 54
59 88 67 12 47 40 12  6 55 67  8 92 84 68 92 28 70 73 50 66
16 23 94 37 13 98 31  6 11 86 55 70 10 67  2 31 90 53 33 44
90 35 36 49  8 85 76 18 92 49 13 89 40 21 80 59 11 62 92 54
33 45 23 19 69 99 63 69 54 10 74 15 24 88 71 44 98 16  4 19
 1 38 40 37  9 45 50 46 67 81 57 26 37 58 87 61 26 69 56 87
36 37 62 78 46 21 70 83 75 90 19 77 27 30 70 46 96 19  2 23
90 32 86 76 96 28 40 83  3 25  6 51 42  1 15 71 31 40 55 26
15 30  8 70 33 79 65 57 89  3 26 88 68  4 16 16 41 42 15  1
53 76 57 23 86  7 49  8 37 66 21 80 52 42  4 99 16  5  6 11
80 95 12 22 76 98 50  4 59 80 92 51 66 99 77 71 89 99 73 78
 2 37 32  8 78 26 85 91  5 89 92 49 69 38 42 74 69 34 63 51
96 11 99  3 62 36 51 64 81 63 95 50 49 99 24 20 44 69 76  4
61 80 46 17 63  8 96 78 87 43  1 22 58 88  6 47 93 87 86 72,
```

то файл MONEY.RES має містити записи

```
2711
rrurruuuurrruurrurururuuurrruuuuuurrrrur
```

1.4 Розв'язання

1. Нова пошта. Не обмежуючи загальності міркувань, можемо вважати, що послідовність $\{x_k\}$ монотонно зростає. Якщо це не так, то спочатку відповідними чином потрібно впорядкувати значення $\{x_k\}$ та $\{m_k\}$ так, щоб нова послідовність $\{x_k\}$ була неспадною. Якщо ця послідовність має однакові члени — кілька будинків (наприклад, два будинки, розташовані один навпроти одного по різні боки від шляху) знаходяться на однаковій відстані від знаку, зменшимо кількість помешкань, збільшуючи відповідним чином кількість їх мешканців.

Графік функції $f(x) = m_1|x - x_1| + m_2|x - x_2| + \cdots + m_n|x - x_n|$ складається з двох променів і $(n-1)$-го відрізка. Кутовий коефіцієнт (тангенс кута нахилу до додатнього напряму осі абсцис) таких променів чи відрізків для аргументу x, відмінного від всіх членів послідовності $\{x_k\}$, дорівнює

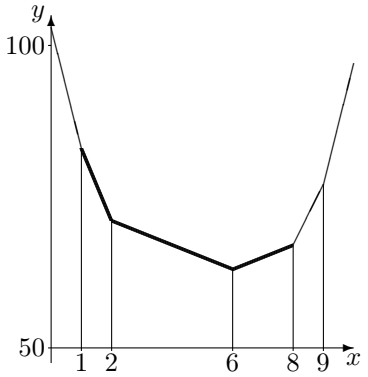
$$g(x) = \sum_{x_j<x} m_j - \sum_{x<x_k} m_k$$

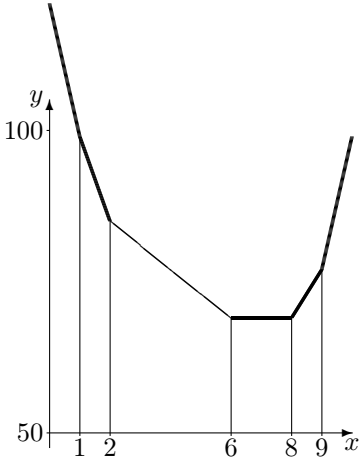
— різниці суми всіх значень m_j, для яких $x_j<x$, і суми всіх значень m_k, для яких $x<x_k$. Функція $g(x)$ не спадає, якщо x зростає. Можливі два різних випадки.

1. Існує точка x_k, в якій функція $g(x)$ міняє знак з $-$ на $+$ при зростанні x. Тоді (глобальний строгий) мінімум функція $f(x)$ досягає саме у цій точці.

2. Існує інтервал $(x_k; x_{k+1})$, на якому функція $g(x)$ дорівнює 0, а відповідна частина графіка $y{=}f(x)$ — горизонтальний проміжок. Тоді (глобальний нестрогий) мінімум функція $f(x)$ досягає у кожній точці відрізка $[x_k; x_{k+1}]$.

Подамо графіки функції $y{=}f(x)$, побудованої за даними тестів 1 (ліворуч) та 4 (праворуч).





Зверніть увагу на те, що обчислення значень функції поданою програмою не передбачено.

Відмова встановити формат відповіді зумовлена небажанням розкривати ідею розв'язання задачі.

```
const n_max=234;
var   n,k,l,s,s0,s1: word;  x,m: array[1..n_max] of word;
      o: text;              flag: boolean;
begin{droite.pas}
                                          {Зчитування даних}
assign(o,'DROITE.DAT');reset(o);readln(o,n);s:=0;
for k:=1 to n do begin readln(o,x[k],m[k]); s:=s+m[k]; end;
close (o);
               {Впорядкування масиву x "методом бульбашки"}
for l:=1 to n   do                       begin flag:=true;
for k:=1 to n-l do if x[k]>x[k+1] then begin flag:=false;
s0:=m[k]; m[k]:=m[k+1]; m[k+1]:=s0;
s1:=x[k]; x[k]:=x[k+1]; x[k+1]:=s1      end;
if flag then break;                      end;
l:=1;        {Перехід до строго монотонної послідовності x}
for  k:=2 to n do
if x[k]=x[l] then               m[l]:=m[k]+m[l]
             else begin l:=l+1; m[l]:=m[k]; x[l]:=x[k] end;
                                {Визначення шуканого місця}
                        k:=1;   s0:=   m[1]; s1:=s -m[1];
while s0<s1 do   begin k:=k+1; s0:=s0+m[k]; s1:=s1-m[k] end;
```

```
                                                {Запис відповіді}
assign(o,'DROITE.RES'); rewrite(o);
    if s0=s1 then begin writeln(o,'Шукане місце - в межах ',
'від ',x[k],' до ',x[k+1],' кроків від знаку.');   Close(o);
halt                end;
   if s0>s1 then begin writeln(o,'Шукане місце - за ',x[k],
' кроків від знаку.');  Close(o);
halt                end;                                    end.
```

2. Сновида. Кожну перестановку вмісту n сейфів можна подати таблицею

$$\begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ f(1) & f(2) & f(3) & \cdots & f(n) \end{pmatrix},$$

де f — деяке взаємно однозначне відображення множини всіх натуральних чисел в межах від 1 до n включно[1] на себе[2]. Тоді другий рядок вхідного файлу містить (у вказаному порядку) послідовність $f(1)$, $f(2)$, ... , $f(n)$.

Перестановку, яка:

- на місце i_1-го елемента ставить елемент з порядковим номером i_2=$f(i_1)$;
- на місце i_2-го елемента ставить елемент з порядковим номером i_3=$f(i_2)$;
  .....
- на місце i_{k-1}-го елемента ставить елемент з порядковим номером i_k=$f(i_{k-1})$;
- на місце i_k-го елемента ставить елемент з порядковим номером i_1,

а всім іншим елементам ставить у відповідність самих себе, назвемо циклічною підстановкою k елементів чи циклом довжини k з елементами i_1, i_2, ... , i_k і позначимо через $(i_1\ i_2\ \cdots\ i_{k-1}\ i_k)$.

Для того, щоб у результаті застосування l разів такої підстановки (циклу довжини k) отримати *одиничне чи тотожнє відображення*[3], необхідно і достатньо, щоб l було кратним k.

Будь-яку перестановку можна подати добутком (послідовним виконанням) циклів, що не мають спільних елементів. Нехай l_1, l_2, l_3, ... , l_j — довжини таких циклів. Тоді шукана найменша кількість застосувань перестановки, що призводить до одиничного відображення, є найменшим спільним кратним довжин цих циклів.

При розв'язанні задачі треба також врахувати необхідність використання "довгої арифметики", тобто подання числа масивом його цифр, для того, щоб успішно пройти всі тести (див. останні рядки програми.)

[1]Таку множину позначають через $\{1, 2, \ldots, n\}$.

[2]Таке відображення називають підстановкою.

[3]Тобто таке, що відображає кожен елемент сам у себе.

```
const n_max=600;
var i,j,k,km,n,nd,m0,m1: word;
 m,l: array[1..n_max] of word;     r: real;
 b:   array[1..n_max] of boolean;  o: text;begin{sleep.pas}
                                          {Зчитування даних}
assign(o,'SLEEP.DAT');  reset(o);  readln(o,n);
for k:=1 to n do begin read(o,m[k]); b[k]:=true; end;
close (o);
              {Знаходження порядків циклічних перестановок}
nd:=0; j:=0; km:=0;
while nd<n do begin  km:=km+1;  k:=km;  j:=j+1;  l[j]:=0;
while b[k] do
begin b[k]:=false;  k:=m[k];  l[j]:=l[j]+1;  nd:=nd+1 end;
if nd=n then break
        else while not b[km+1] do km:=km+1            end;
    {Подання шуканого числа добутком взаємно простих чисел}
for i:=1   to j-1 do if l[i]>1 then                  begin
for k:=i+1 to j   do if l[k]>1 then                  begin
    m0:=l[i];       m1:=l[k];  km:=1;  while km>0 do begin
km:=m0 mod m1;  m0:=m1;    m1:=km                      end;
l[k]:=l[k] div m0                                 end end;
r:=0;
           {Знаходження десяткового запису шуканого числа}
for k:=1 to j do r:=r+Ln(l[k]);      n:=Trunc(r/Ln(10))+1;
for k:=2 to n do m[k]:=0;  m[1]:=1;  nd:=1;
for k:=1 to j do if l[k]>0 then begin i:=0;  m0:=0;
while (i<nd) or (m0>0) do begin i:=i+1; m1:=m0+m[i]*l[k];
m[i]:=m1 mod 10;  m0:=(m1-m[i]) div 10 end;
nd:=i                              end;
                                          {Запис відповіді}
assign(o,'SLEEP.RES'); rewrite(o);
for k:=2 to nd do write (o,m[i-k+2]); writeln(o,m[1]);
close (o)                                             end.
```

3. Гроші на шахівниці. Для клітинки у k-му рядку, якщо рахувати згори донизу, та j-му стовпчику, якщо рахувати зліва направо, позначимо через:

a_{kj} — кількість монет, розташованих на цій клітинці;

c_{kj} — найбільшу кількість монет, які можна зібрати з клітинок шахівниці, якщо розпочати рух (праворуч і вгору) з цієї клітинки до клітинки у першому рядку і останньому стовпчику.

Тоді:

- $c_{1n} = a_{1n}$;
- для всіх k в межах від 2 до m включно

$$c_{kn} = c_{(k-1)\ n} + a_{kn}$$

(розгляньте клітинки n-го стовпчика, в яких рух можна здійснювати лише вгору);

- для всіх j в межах від 1 до $n-1$ включно

$$c_{1j} = c_{1\ (j+1)} + a_{1j}$$

(розгляньте клітинки 1-го рядка, в якому рух можна здійснювати лише праворуч);

- для решти значень $k>1$ та $j<n$, якщо $c_{(k-1)\ j}>c_{k\ (j+1)}$, то

$$c_{kj} = c_{(k-1)\ j} + a_{kj}$$

і з клітини у k-му рядку j-го стовпчика перший крок треба робити вгору, інакше

$$c_{kj} = c_{k\ (j+1)} + a_{kj}$$

і перший крок треба робити праворуч.

Таким чином, за відомою матрицею[4] $||a_{kj}||$ легко побудувати матрицю $||c_{kj}||$ і вказати траєкторію руху пішака. Наприклад, для поданої в умові задачі матриці

$$||a_{jk}|| = \begin{pmatrix} 1 & 2 & 3 \\ 6 & 8 & 2 \end{pmatrix}$$

матриця $||c_{jk}||$ і напрям руху пішака знаходяться у наступним чином

$$\begin{pmatrix} * & * & 3 \\ * & * & * \end{pmatrix} \to \begin{pmatrix} * & 5 & 3 \\ * & * & 5 \end{pmatrix} \to \begin{pmatrix} 6 & 5 & 3 \\ * & 13 & 5 \end{pmatrix} \to \begin{pmatrix} 6 & 5 & 3 \\ 19 & 13 & 5 \end{pmatrix},$$

$$\begin{pmatrix} \to & \to & \bullet \\ * & * & \uparrow \end{pmatrix} \to \begin{pmatrix} \to & \to & \bullet \\ * & \to & \uparrow \end{pmatrix} \to \begin{pmatrix} \to & \to & \bullet \\ \to & \to & \uparrow \end{pmatrix}.$$

```
const nmax=20;
var a,c: array[1..nmax,1..nmax] of word;
      b: array[1..nmax,1..nmax] of boolean;
      j,k,l,m,n: integer;          o: text;begin{money.pas}
                                          {Зчитування даних}
assign(o,'MONEY.DAT');  reset(o);  readln(o,m,n);
for k:=1 to m do for j:=1 to n do read(o,a[k,j]); close(o);

c[1,n]:=a[1,n];                  {Знаходження матриць b i c}
for k:=2 to m do
begin c[k,n]:=c[k-1,n]+a[k,n]; b[k,n]:=true  end;
for j:=n-1 downto 1 do
begin c[1,j]:=c[1,j+1]+a[1,j]; b[1,j]:=false end;
```

[4]Матриця — прямокутною таблиця чисел.

```
for k:=2 to m do for j:=n-1 downto 1 do if c[k-1,j]>c[k,j+1]
then begin c[k,j]:=c[k-1,j]+a[k,j]; b[k,j]:=true  end
else begin c[k,j]:=c[k,j+1]+a[k,j]; b[k,j]:=false end;

{Визначення послідовності напрямків руху і запис відповіді}
assign(o,'MONEY.RES');  rewrite(o);   writeln(o,c[m,1]);
j:=1;  k:=m;             for l:=1 to m+n-2 do
if b[k,j] then begin write(o,'u'); k:=k-1 end
          else begin write(o,'r'); j:=j+1 end; close(o);
halt                                        end.
```

2 II етап

2.1 Умови проведення

Олімпіаду рекомендуємо провести в один практичний тур тривалістю 5 астрономічних годин (після ознайомлення з умовами, переписування умов.) Коректність даних перевіряти не потрібно.

2.2 Завдання

1. Мотель, 60 балів. Рівнину перетинають швидкісні автостради. Створіть програму MOTEL.*, яка вибере найвигідніше місце для мотелю, від якого сума відстаней до всіх автострад найменша.

Перший рядок вхідного файлу MOTEL.DAT містить натуральне число n — кількість автострад, які надалі вважаємо різними прямими лініями. Для j в межах від 1 до n включно $(j+1)$-ий рядок цього ж файлу містить 4 натуральних числа — декартові координати в одній і тій же системі координат двох різних точок на j-ій прямій, відстань між якими виражається натуральним числом. Наприклад,

```
2
0  0  4  3
1  1  0  1
```

Кількість прямих n лежить в межах від 2 до 10, а всі інші числа вхідного файлу MOTEL.DAT не перевищують 123.

Перший рядок вихідного файлу MOTEL.RES має містити найменшу можливу суму відстаней від автострад до мотелю.

Наступні рядки цього ж файлу мають містити відповідь українською мовою чи мовою навчання (орфографічні помилки та літературний стиль на кількість балів не впливають) на питання про множину точок оптимального розташування

мотелю. Шукану множину треба подати об'єднанням якомога меншої кількості лінійно зв'язних[5] фігур. Наприклад,

```
0
Шукана множина містить єдину точку (4/3;1).
```

Будь-яке число у відповіді треба подати нескоротним дробом з цілим чисельником і натуральним знаменником, розділеними знаком ділення /. Якщо знаменник дорівнює 1, то знак ділення і знаменник не записувати.

Математична довідка.

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

Якщо $ax+by+c=0$ — *рівняння деякої прямої, то знак лівої частини цього рівняння сталий на кожній з півплощин, на які декартова площина розбивається даною прямою.*

Відстань від точки $(x_0;y_0)$ *до прямої, заданої рівнянням* $ax+by+c=0$, *дорівнює* $|ax_0+by_0+c| : \sqrt{a^2+b^2}$.

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

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

Якщо $\Delta\neq 0$, *то розв'язок єдиний (прямі перетинаються у точці); якщо* $\Delta=0$, *але хоча б одне з чисел* Δ_x *та* Δ_y *відмінне від 0, то розв'язків система не має (прямі паралельні); якщо* $\Delta=\Delta_x=\Delta_y=0$, *то рівняння системи еквівалентні, а система має безліч розв'язків (прямі співпадають).*

2. Латиниця, 30 балів. Латиниця (латинська абетка) має 26 літер: *a b c d e f g h i j k l m n o p q r s t u v w x y z* (вказано в алфавітному порядку).

Перший рядок вхідного файлу LATIN.DAT містить у вказаному порядку два натуральних числа k та l, де $k<26$, $l<255$. Другий рядок цього самого файлу містить послідовність k літер латиниці (використано лише малі літери) без прогалин між літерами. Наприклад,

```
4 3
bcdx
```

Створіть програму LATIN.*, яка у вихідний файл LATIN.RES записує в алфавітному порядку l "слів" по k літер у кожному — послідовностей по k літер латиниці, в яких:

[5] Фігуру називають лінійно зв'язною, якщо будь-які дві її точки можна з'єднати неперервною прямою, кожна точка якої належить цій фігурі.

i) не зустрічається літера a;

ii) жодна літера не повторюється;

iii) всі літери зустрічаються у тому порядку (при зчитуванні зліва направо), як і в абетці.

Кожен рядок файлу LATIN.RES має містити лише одне таке "слово". При цьому неможливо вставити ніяке інше слово з переліченими властивостями (i–iii):

- між послідовними "словами" файлу LATIN.RES;
- між єдиним "словом" файлу LATIN.DAT і першим файлу LATIN.RES,

не порушивши алфавітного порядку. Наприклад, для поданого раніше файлу LATIN.DAT файл LATIN.RES має вигляд

```
bcdy
bcdz
bcef
```

Якщо таких "слів" менше, ніж l, то записують всі можливі такі "слова". Якщо таких "слів" немає взагалі, то в єдиний рядок файлу LATIN.RES записують повідомлення

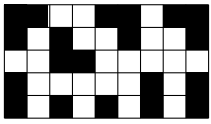
```
Таких слів немає!
```

Наприклад, це потрібно зробити, коли LATIN.DAT має вигляд

```
4 5
xbcd
```

Врахуйте, що "слово" другого рядка LATIN.DAT може не задовольняти умови (i–iii).

3. Тест, 60 балів. У дитячому садку проводиться тестування. Дітям показують білі аркуші паперу прямокутної форми, які поділено на рівні квадрати горизонтальними та вертикальними лініями. Частину квадратів зафарбовано чорною фарбою, а частину ні. Фігурою на такому малюнку називають сукупність чорних квадратів, для довільної пари яких центри квадратів можна з'єднати ламаною, що повністю міститься у чорних квадратах і не містить жодної вершини квадрата. Різними фігурами називаються фігури, які неможливо сумістити послідовним застосуванням паралельних переносів, поворотів на 90° і симетрії відносно вертикальної чи горизонтальної прямої. Дітям пропонують встановити, скільки всього фігур зображено на малюнку, і скільки різних фігур зображено на цьому малюнку. Наприклад, для малюнку



правильна відповідь: 9 і 3 відповідно.

Створіть програму TEST.*, яка дає відповідь на поставлене питання.

Перший рядок вхідного файлу TEST.DAT містить (у вказаному порядку) два натуральних числа m і n, які не перевищують 60 — кількості квадратів, які розташовані на малюнку по вертикалі та горизонталі відповідно. Наступні m рядків по n символів цього самого файлу містять подання малюнку, в якому символ " " (прогалина) означає білий квадрат, а "Ш" — чорний. Наприклад, для поданого раніше малюнку маємо таке.

```
5 9
ШШШ   ШШШ ШШШ
Ш Ш   Ш   Ш
  ШШШ
Ш       Ш Ш
Ш Ш Ш Ш Ш
```

Єдиний рядок вихідного файлу TEST.RES має містити у вказаному порядку два натуральних числа: кількість всіх фігур та кількість різних фігур. Відомо, що обидва шуканих числа не перевищують 127. Наприклад,

```
9 3
```

2.3 Вказівки до перевірки

Якщо є можливість, перевірку тестів треба здійснити у присутності учнів. Тоді необхідність у апеляції відпадає взагалі.

1. Мотель. За успішне проходження кожного з 12-ти тестів — по 5 балів. Якщо:

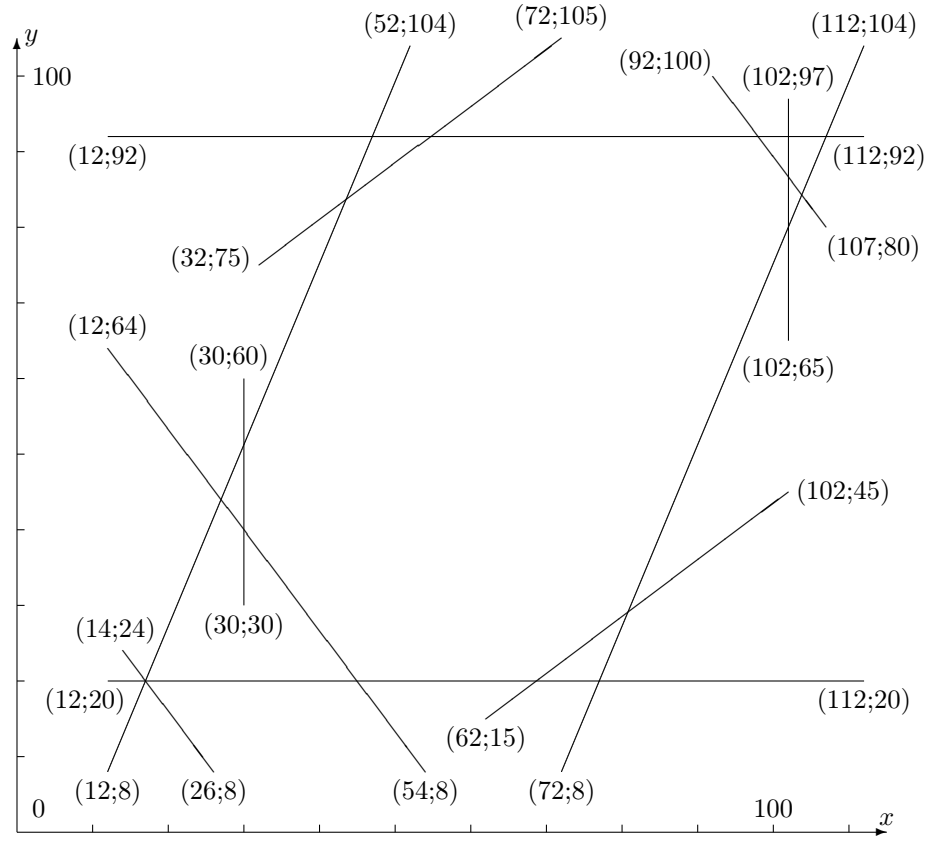
- не знайдено хоча б наближено шукану відстань;
- не виконано умову щодо подання шуканої відстані нескоротним дробом;
- не вказано вид шуканої множини або кілька разів перераховано одні й ті ж вершини многокутників;
- не згадано про опуклість многокутників;
- не виконано умову щодо подання координат многокутника нескоротним дробом,

то знімається по 1 балу за кожен з перелічених недоліків (окремо для кожного тесту), якщо решту вимог щодо відповіді виконано.

Не вимагається вказувати координати вершин многокутників у порядку обходу за периметром. Аварійне завершення в результаті виходу за межі допустимих значень змінних (тести 5–10) можливе, якщо невдало реалізовано арифметику раціональних чисел — скорочення спільних дільників чисельника і знаменника відкладається до останнього кроку, а знаменник суми дробів знаходиться як добуток знаменників доданків, а не як їх найменше спільне кратне.

Тести створено, враховуючи:

- відомості математичної довідки;
- раціональність тригонометричних функцій кутів, утворених прямими з віссю абсцис (вимога умови задачі — див. слова про натуральність відстані між точками з натуральними координатами);
- те, що сума відстаней від будь-якої точки площини до двох паралельних прямих на площині найменша для точок, які лежать між даними прямими або на них.



На поданому малюнку зображено відрізки прямих і вказано натуральні координ кінців цих відрізків, які є в умовах тестів. Для того, щоб не захаращувати малюнок, не зображено лише відрізок з тесту 9, який з'єднує точки (99;96) і (105;88), і серединою якого є точка (102;92).

1.1 Якщо MOTEL.DAT має вигляд

```
 3
 12   8   52 104
 12  20  112  20
 14  24   26   8 ,
```

то MOTEL.RES має вигляд

```
0
Шукана множина містить єдину точку (17;20).
```

1.2 Якщо MOTEL.DAT має вигляд

```
 3
 12   8   52 104
 72   8  112 104
 92 100  107  80 ,
```

то MOTEL.RES має вигляд

```
720/13
Шукана множина - відрізок, що з‘єднує точки
(913/14;950/7) і (1453/14;590/7).
```

1.3 Якщо MOTEL.DAT має вигляд

```
6
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92
 14  24   26   8
 12  64   54   8 ,
```

то MOTEL.RES має вигляд

```
9736/65
Шукана множина - трикутник з вершинами
(17;20), (27;44), (45;20).
```

1.4 Якщо MOTEL.DAT має вигляд

```
4
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92 ,
```

то MOTEL.RES має вигляд

```
1656/13
Шукана множина - опуклий 4-кутник з вершинами
(17;20), (47;92), (77;20), (107;92).
```

1.5 Якщо MOTEL.DAT має вигляд

```
6
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92
 92 100  107  80
 14  24   26   8 ,
```

то MOTEL.RES має вигляд

```
3060/13
Шукана множина - опуклий 5-кутник з вершинами
(17;20), (47;92), (77;20), (1453/14;590/7), (98;92).
```

1.6 Якщо MOTEL.DAT має вигляд

```
6
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92
 92 100  107  80
 12  64   54   8 ,
```

то MOTEL.RES має вигляд

```
13844/65
Шукана множина - опуклий 6-кутник з вершинами
(47;92), (27;44), (77;20), (1453/14;590/7),
(45;20), (98;92).
```

1.7 Якщо MOTEL.DAT має вигляд

```
8
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92
 92 100  107  80
 14  24   26   8
 32  75   72 105
 62  15  102  45 ,
```

то MOTEL.RES має вигляд

```
3918/13
Шукана множина - опуклий 7-кутник з вершинами
(17;20), (1436/33;920/11), (1453/14;590/7),
(2666/33;320/11), (206/3;20), (98;92), (164/3;92).
```

1.8 Якщо MOTEL.DAT має вигляд

```
8
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92
 92 100  107  80
 12  64   54   8
 32  75   72 105
 62  15  102  45 ,
```

то MOTEL.RES має вигляд

```
18134/65
Шукана множина - опуклий 8-кутник з вершинами
(27;44), (1436/33;920/11), (1453/14;590/7),
(2666/33;320/11), (45;20), (206/3;20),
(98;92), (164/3;92).
```

1.9 Якщо MOTEL.DAT має вигляд

```
10
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92
 99  96  105  88
```

```
 12  64   54   8
 32  75   72 105
 62  15  102  45
 30  30   30  60
102  65  102  97 ,
```

то MOTEL.RES має вигляд

```
23022/65
Шукана множина - опуклий 9-кутник з вершинами
(1436/33;920/11), (30;256/5), (2666/33;320/11),
(102;80), (45;20), (206/3;20), (102;92),
(164/3;92), (30;40).
```

1.10 Якщо MOTEL.DAT має вигляд

```
10
 12   8   52 104
 72   8  112 104
 12  20  112  20
 12  92  112  92
 92 100  107  80
 12  64   54   8
 32  75   72 105
 62  15  102  45
 30  30   30  60
102  65  102  97 ,
```

то MOTEL.RES має вигляд

```
22814/65
Шукана множина - опуклий 10-кутник з вершинами
(1436/33;920/11), (30;256/5), (2666/33;320/11),
(102;80), (45;20), (206/3;20), (98;92),
(164/3;92), (102;260/3), (30;40).
```

1.11 Якщо MOTEL.DAT має вигляд

```
3
 92 100  107  80
 14  24   26   8
 12  64   54   8 ,
```

то MOTEL.RES має вигляд

```
108
Шукана множина - пряма 4*x+3*y-240=0.
```

1.12 Якщо MOTEL.DAT має вигляд

```
2
 12   8   52 104
 72   8  112 104 ,
```

то MOTEL.RES має вигляд

```
720/13
Шукана множина - частина площини між паралельними
прямими 12*x-5*y-104=0 i 12*x-5*y-824=0
разом з цими прямими.
```

В тестах 11–12 допускається замість рівнянь прямих вказувати точки, через які проходять ці прямі.

2. Латиниця. За успішне проходження тестів 1, 2, 4, 5, 7, 8 — по 3 бали, за успішне опрацювання тестів 3, 6 та 9 — по 4 бали.

Тести 1, 4, 7 перевіряють "паталогічний" випадок слів, які складаються лише з однієї літери.

Тести 1–3 перевіряють, чи враховано можливість відсутності потрібних "слів". Якщо файл LATIN.DAT — один з 3-х наступних:

2.1
```
1 11
z
```

2.2
```
4 7
xaaa
```

2.3
```
23 10
ocddddddddddddddddddddf
```

то файл LATIN.RES має вигляд

```
Таких "слів" немає!
```

Тести 4–6 перевіряють, чи враховано можливість присутності літери `a` напочатку другого рядка файлу LATIN.DAT.

2.4 Якщо файл LATIN.DAT має вигляд

```
1 27
a ,
```

то файл LATIN.RES містить всі, крім першої, літери латиниці, записані у алфавітному порядку: `b c d e f g h i j k l m n o p q r s t u v w x y z`.

2.5 Якщо файл LATIN.DAT має вигляд

```
4 5
abba ,
```

то файл LATIN.RES має вигляд

```
bcde
bcdf
bcdg
bcdh
bcdi .
```

2.6 Якщо файл LATIN.DAT має вигляд

```
23 10
abbaabbaabbaabbaabbaabb ,
```

то файл LATIN.RES має вигляд

```
bcdefghijklmnopqrstuvwx
bcdefghijklmnopqrstuvwy
bcdefghijklmnopqrstuvwz
bcdefghijklmnopqrstuvxy
bcdefghijklmnopqrstuvxz
bcdefghijklmnopqrstuvyz
bcdefghijklmnopqrstuwxy
bcdefghijklmnopqrstuwxz
bcdefghijklmnopqrstuwyz
bcdefghijklmnopqrstuxyz .
```

Тести 7–9 перевіряють правильне визначення потрібних "слів" для даних, що не мають перерахованих раніше "паталогій".
2.7 Якщо файл LATIN.DAT має вигляд

```
1 2
o ,
```

то файл LATIN.RES має вигляд

```
p
q .
```

2.8 Якщо файл LATIN.DAT має вигляд

```
4 7
vutr ,
```

то файл LATIN.RES має вигляд

```
vwxy
vwxz
vwyz
vxyz
wxyz .
```

2.9 Якщо файл LATIN.DAT має вигляд

```
23 10
bcddddddddddddddddddddf ,
```

то файл LATIN.RES має такий же вигляд, як і для тесту 6

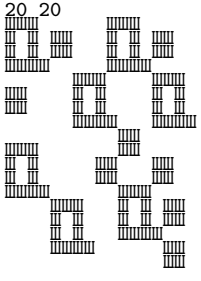
```
bcdefghijklmnopqrstuvwx
bcdefghijklmnopqrstuvwy
bcdefghijklmnopqrstuvwz
bcdefghijklmnopqrstuvxy
bcdefghijklmnopqrstuvxz
bcdefghijklmnopqrstuvyz
bcdefghijklmnopqrstuwxy
bcdefghijklmnopqrstuwxz
bcdefghijklmnopqrstuwyz
bcdefghijklmnopqrstuxyz .
```

Тести 3, 6, 9 неможливо успішно опрацювати, якщо "в лоб" використано систему числення з основою 26.

3. Тест. За успішне проходження кожного з 10-ти тестів — по 6 балів. Якщо неправильно вказано кількість фіцур, то знімають 2 бали, якщо неправильно вказано кількість різних фігур, то знімають 4 бали (за кожен тест окремо). Для тестів 1–9 аналізують малюнок 20×20 клітинок, а тест 10 — 60×60. Тести 1–8 перевіряють правильність порівняння фігур для певної симетрії, тест 9 — правильність виділення фігури у випадку, коли треба аналізувати весь малюнок. Тест 10 перевіряє все те, що й тести 1–9, і також правильність використання типів змінних.

3.1 Якщо файл TEST.DAT має вигляд

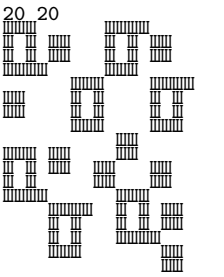
```
20 20
```



,

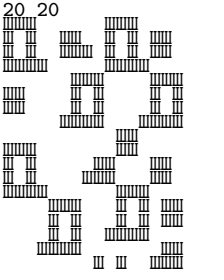
то TEST.RES містить 15 і 2.

3.2 Якщо файл TEST.DAT має вигляд



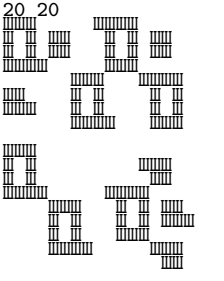
то TEST.RES містить 16 і 3.

3.3 Якщо файл TEST.DAT має вигляд



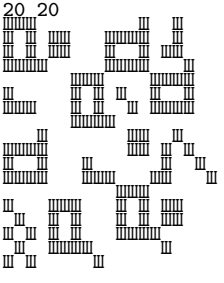
то TEST.RES містить 17 і 4.

3.4 Якщо файл TEST.DAT має вигляд



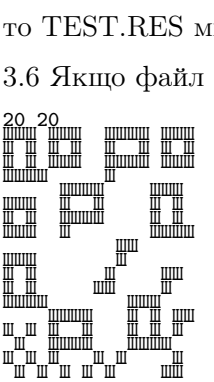
то TEST.RES містить 13 і 3.

3.5 Якщо файл TEST.DAT має вигляд



то TEST.RES містить 29 і 4.

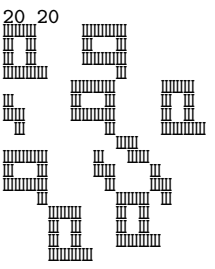
3.6 Якщо файл TEST.DAT має вигляд



,

то TEST.RES містить 27 і 4.

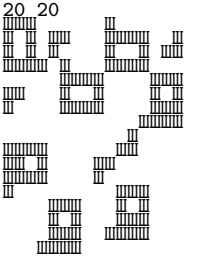
3.7 Якщо файл TEST.DAT має вигляд



,

то TEST.RES містить 11 і 2.

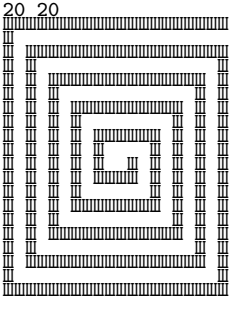
3.8 Якщо файл TEST.DAT має вигляд



,

то TEST.RES містить 12 і 3.

3.9 Якщо файл TEST.DAT має вигляд

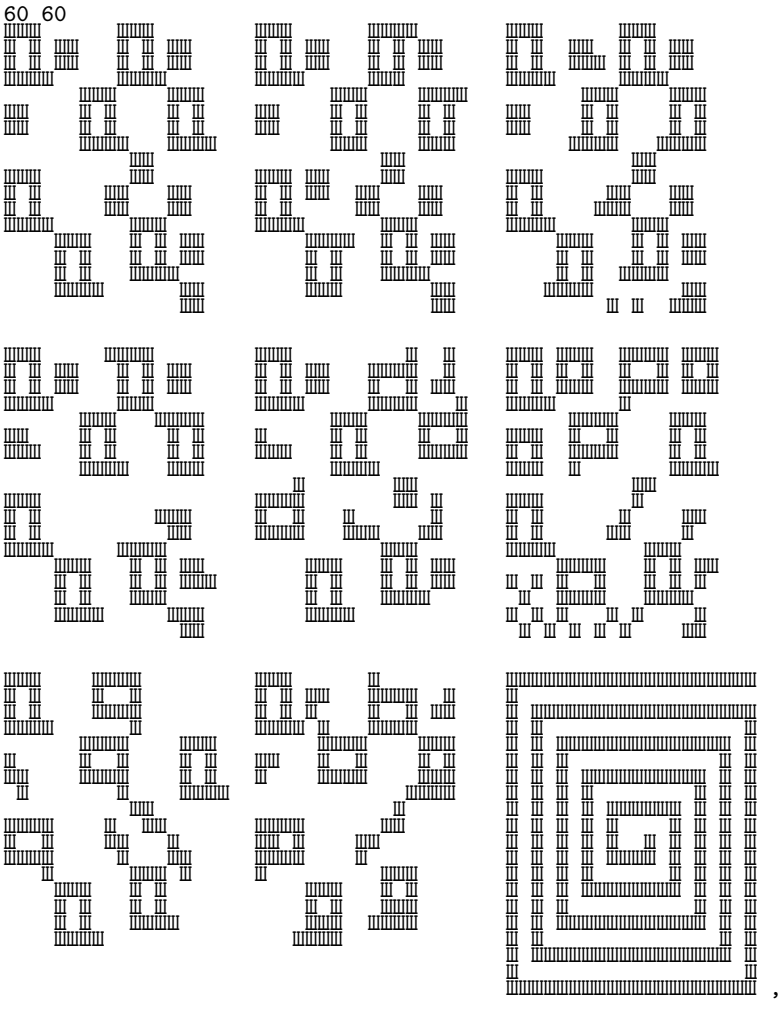


,

то TEST.RES містить 1 і 1.

3.10 Якщо файл TEST.DAT має вигляд

60 60



,

то TEST.RES містить числа 126 і 11.

2.4 Розв'язання

1. Мотель. Скінчена кількість даних прямих розбиває площину на опуклі "многокутники" (деякі з них необмежені), внутрішні точки яких не належать жодній з цих прямих. Для всіх точок таких "многокутників" сума відстаней до всіх даних прямих лінійним чином залежить від координат (див. математичну довідку) і тому:

- або стала;

- або досягає найменшого значення на одній з вершин і лише на ній;
- або досягає найменшого значення на всіх точках однієї із сторін і лише на них.

Якщо хоча б дві прямі перетинаються, то найменша сума відстаней до всіх прямих не може досягатися у всіх точках променя, який обмежує один з "многокутників".

Нехай напрям "вгору" визначається додатним напрямом осі аплікат z тривимірної системи координат. Будемо казати, що функція f з декартової площини у числову пряму *нестрого опукла вниз*, якщо *всі точки відрізка, який з'єднує будь-які дві точки графіка функції, розташовані не нижче самого графіка.*

Поверхня $\{(x;y;z)|\ x,y \in(-\infty;+\infty),\ z=f(x,y)\}$, що є графіком функції

$$f(x,y)=\frac{|ax+bx+c|}{\sqrt{a^2+b^2}},$$

складається з двох півплощин і прямої, що їх обмежує і задається рівняннями $ax+by+c=0=z$ — див. математичну довідку до задачі про геометричний зміст функції. Очевидно, що функція $f(x,y)$ нестрого опукла вниз.

Координати будь-якої точки відрізка, що сполучає точки $(x_1;y_1;f(x_1,y_1))$ та $(x_2;y_2;f(x_2,y_2))$ на графіку, можна подати у вигляді

$$\begin{cases} x=\alpha x_1+(1-\alpha)x_2 \\ y=\alpha y_1+(1-\alpha)y_2 \\ z=\alpha f(x_1,y_1)+(1-\alpha)f(x_2,y_2) \end{cases},$$

причому кожній точці відрізка можна поставити у взаємно однозначну відповідність деяке $\alpha\in[0;1]$. Таким чином, *функція f нестрого опукла вниз тоді і лише тоді, коли для довільних точок декартової площини $(x_1;y_1)$ та $(x_2;y_2)$ і довільного $\alpha\in[0;1]$ справджується нерівність*

$$f(\alpha x_1+(1-\alpha)x_2,\alpha y_1+(1-\alpha)y_2)\le\alpha f(x_1,y_1)+(1-\alpha)f(x_2,y_2).$$

Очевидно, що сума будь-якої кількості нестрого опуклих вниз функцій, визначених на опуклій множині, є нестрого опуклою вниз.

Якщо з'єднати будь-які дві точки, у яких нестрого опукла функція набирає найменше значення, відрізком, то в будь-якій точці цього відрізка функція також набирає найменше значення. Іншими словами, *множина, на якій нестрого опукла функція набирає найменше значення, є опуклою.*

З поданих міркувань випливає наступний алгоритм розв'язання задачі.

1. Зчитуємо натуральне n з вхідного файлу.
2. Для j в межах від 1 до n включно

- зчитуємо з вхідного файлу координати точок $(x_1; y_1)$ і $(x_2; y_2)$ на j-ій прямій;
- визначаємо цілі коефіцієнти $a_j = y_2 - y_1$, $b_j = x_1 - x_2$ $c_j = y_1(x_2 - x_1) - x_1(y_2 - y_1)$, рівняння прямої $a_j x + b_j x + c_j = 0$;
- знаходимо найбільший спільний дільник чисел a_j, b_j та c_j і ділимо на нього ці числа;
- знаходимо натуральне[6] число $d_j = \sqrt{a_j^2 + b_j^2}$.

3. Для кожної пари прямих знаходимо точку перетину, якщо тільки вона існує, і для неї визначаємо суму відстаней до всіх даних прямих. Якщо величина суми відстаней — найменша з вже знайдених, а точка перетину прямих з такими координатами ще не розглядалася[7], то запам'ятовуємо значення координат знайденої точки. Якщо точок перетину даних прямих немає, тобто всі прямі паралельні, переходимо до виконання пункту 5.

4. В залежності від кількості точок, знайдених на попередньому етапі, записуємо у вихідний файл відповідь:
   - або "Шукана множина містить єдину точку" і вказуємо її координати;
   - або "Шукана множина — відрізок, що з'єднує точки" і вказуємо їх координати;
   - або "Шукана множина — опуклий многокутник з вершинами " і подаємо перелік координат вершин,

   після чого припиняємо виконання алгоритму.

5. У випадку паралельних прямих знаходимо суму відстаней від кожної прямої до решти прямих.

6. Визначаємо, для якої прямої знайдена відстань найменша.

7. Якщо така пряма єдина, припиняємо виконання алгоритму, записавши у вихідний файл повідомлення "Шукана множина — пряма, рівняння якої" і записуємо рівняння прямої.

8. Якщо таких прямих дві, припиняємо виконання алгоритму, попередньо записавши у вихідний файл повідомлення "Шукана множина — смуга, обмежена паралельними прямими, заданих рівняннями" і рівняння прямих.

[6]Згідно умови задачі, відстань між точками $(x_1; y_1)$ і $(x_2; y_2)$ виражається натуральним числом.

[7]Перетин трьох чи більше заданих прямих не суперечить умові задачі. Тому треба уникнути повторного переліку точок.

Запропонована задача розвиває ідею задачі I.1 і є частковим випадком задач лінійного програмування. Справді, розглянемо множину точок декартового простору, розташованих не нижче графіка функції $z{=}f(x,y)$. Тут $f(x,y)$ — відстань від точки $(x;y)$ до даних прямих, які задано рівняннями

$$a_j x + b_j x + c_j = 0, \qquad j = 1,\ 2,\ \ldots,\ n.$$

Цю множину однозначно задано сукупністю 2^n лінійних нерівностей

$$z \geq \sum_{j=1}^{n} \sigma_j \frac{a_j x + b_j y + c_j}{\sqrt{a_j^2 + b_j^2}},$$

де знаки $\sigma_j{=}\pm 1$ вибираються незалежно. Ў завдання полягає у тому, щоб знайти всі ті точки $(x;y;z)$ цієї множини, в яких

$$z = \vec{k}(0;0;1) \cdot \vec{v}(x;y;z)$$

набирає найменше значення.

```
const n_max=10;
var              a,b,c,d,x,y,z: array[1..n_max] of longint;
       o: text;  del,x1,y1,z1,x2,y2,pmin,p,qmin,q: longint;
i,j,k,l,m,m0,m1,n: byte; differ: boolean;     f,fmin: real;
{Функція - найбільший спільний дільник двох її аргументів}
function gcd(i,j:longint):longint; var k,l,m:longint; begin
if i=0 then             gcd:=abs(j)
       else if j=0 then gcd:=abs(i)
                       else                                 begin
       m:=1;           k:=abs(i); l:=abs(j); while m>0 do begin
       m:=k mod l; k:=l;      l:=m                            end;
                            gcd:=k                        end end;

{Процедура запису рівняння прямої у файл o}
procedure equation(a,b,c:longint);                        begin
if a<>0 then              write (o,    a,'*x');
if b< 0 then              write (o,    b,'*y');
if b> 0 then if a=0 then write (o,    b,'*y')
                    else write (o,'+',b,'*y');
if c< 0 then              write (o,    c,'=0 ')
        else if c>0 then write (o,'+',c,'=0 ')
                    else write (o,'=0 ')                    end;

{Процедура запису раціональних координат точки у файл o}
procedure fraction(x,y,z:longint);   var k:longint;   begin
k:=gcd(x,z);  write(o,'(',x div k);  k:=z div k;
if  k>1  then write(o,'/',k);
```

```
k:=gcd(y,z);  write(o,';',y div k);  k:=z div k;
if  k>1  then write(o,'/',k,')') else write(o,')')      end;

begin{motel.pas}  {Опрацювання даних: знаходження a[j],b[j],
                   c[j] - цілих коефіцієнтів рівнянь прямих}
assign(o,'MOTEL.DAT');  reset (o);  readln(o,n);
for j:=1 to n do                                        begin
read(o,x1,y1,x2,y2);  a[j]:=y2-y1;  b[j]:=x1-x2;
c[j]:=y1*(x2-x1)-x1*(y2-y1);  p:=gcd(a[j],gcd(b[j],c[j]));
if (a[j]<0) or ((a[j]=0) and (b[j]<0)) then p:=-p;
a[j]:=a[j] div p;  b[j]:=b[j] div p;  c[j]:=c[j] div p;
d[j]:=trunc(sqrt(a[j]*a[j]+b[j]*b[j]));                 end;
close (o); pmin:=-1;
                        {Знаходження точок перетину прямих
(x1 та y1 - відповідно чисельники абсциси і ординати точки,
z1 - їх спільний знаменник) і сум відстаней від усіх прямих
до знайдених точок (p, q - відповідно чисельник і знаменник
суми, pmin та qmin - чисельник і знаменник найменшої суми)}
for j:=1 to n-1 do for k:=j+1 to n do                   begin
del:= a[j]*b[k]-a[k]*b[j];              if del<>0 then begin
x1 :=-c[j]*b[k]+c[k]*b[j];
y1 :=-a[j]*c[k]+a[k]*c[j];              if del< 0 then begin
x1 :=-x1;  y1:=-y1;  del:=-del                            end;
z1 :=gcd(x1,gcd(y1,del));
x1:=x1 div z1; y1:=y1 div z1; z1:=del div z1; p:=0; q:=1;
for i:=1 to n do                                        begin
del:=gcd(d[i]*z1,q);
p:=abs(a[i]*x1+b[i]*y1+c[i]*z1)*(q div del)
  +p*((d[i]*z1) div del);
q:=q*((d[i]*z1) div del);
del:=gcd(p,q); p:=p div del; q:=q div del;  f:=p/q;
if pmin>=0 then if fmin<f then break                     end;
if pmin=-1 then                                         begin
               pmin:=p;   qmin:=q;   fmin:=f;  m:=1;
               x[1]:=x1;  y[1]:=y1;  z[1]:=z1             end
          else
if fmin=f  then    {Врахування можливості перетину}     begin
                {в одній точці більше, ніж 2-х прямих}
     l:=0;   differ:=true; while (l<m) and differ do    begin
     l:=l+1; differ:=differ and ((x[l]<>x1)
                or (y[l]<>y1) or (z[l]<>z1))              end;

                                        if differ then begin
               m:=m+1; x[m]:=x1; y[m]:=y1; z[m]:=z1 end end
          else
if fmin>f  then                                         begin
               pmin:=p;   qmin:=q;   fmin:=f;  m:=1;
```

```
              x[1]:=x1;  y[1]:=y1;  z[1]:=z1   end end end;
if pmin>=0 then                                          begin

{Запис відповіді, якщо хоча б дві прямі перетинаються...}

assign(o,'MOTEL.RES');  rewrite(o);  write (o,pmin);
if qmin>1 then writeln (o,'/',qmin) else writeln(o);
write(o,'Шукана множина ');
if m=1 then                                              begin
write  (o,'містить єдину точку ');
fraction(x[1],y[1],z[1]);  writeln(o,'.')                  end
       else if m=2 then                                  begin
writeln (o,'- відрізок, що з'єднує точки ');
fraction(x[1],y[1],z[1]);  write  (o,' i ');
fraction(x[2],y[2],z[2]);  writeln(o,'.')                  end
                  else if m>2 then                       begin
if m=3 then writeln(o,'- трикутник з вершинами')
       else writeln(o,'- опуклий ',m,'-кутник з вершинами');
for j:=1 to m-1 do                                       begin
fraction(x[j],y[j],z[j]);  write  (o,', ')                 end;
fraction(x[m],y[m],z[m]);  writeln(o,'.')              end end
           else                                          begin
                  {Якщо всі прямі паралельні ...}

reset(o);  readln(o,n); m1:=0;       for j:=1 to n do begin
read (o,x1,y1,x2,y2);   p:=0;  q:=1; for i:=1 to n do begin
p:=p*d[i]+abs(a[i]*x1+b[i]*y1+c[i])*q; q:=q*d[i];
del:=gcd(p,q);  p:=p div del;  q:=q div del;  f:=p/q    end;
if pmin<0 then begin pmin:=p; qmin:=q; fmin:=f; m0:=j end
          else
if fmin>f then begin pmin:=p; qmin:=q; fmin:=f; m0:=j end
          else if fmin=f then m1:=j                      end;
close(o);
                                              {Запис відповіді}
assign(o,'MOTEL.RES');  rewrite(o);  write (o,pmin);
if qmin>1 then writeln (o,'/',qmin) else writeln(o);
write(o,'Шукана множина ');
if m1=0 then                                             begin
              write  (o,'- пряма ');
              equation(a[m0],b[m0],c[m0]);
              writeln(o,'.')                               end
        else                                             begin
writeln (o,'- частина площини між паралельними прямими ');
equation(a[m0],b[m0],c[m0]);  write (o,' i ');
equation(a[m1],b[m1],c[m1]);
writeln (o,'разом з цими прямими.')                  end end;
close (o); halt                                          end.
```

Для того, щоб не вийти за границі зміни цілих змінних, у поданій програмі замість "точного" порівняння цілих величин `p*qmin` та `pmin*q` порівнюються величини дробів `f:=p/q` та `fmin:=pmin/qmin`.

Намагання розширити можливості програми для роботи з довільними цілими числами вхідного файлу призводить до необхідності подання цілих чисел масивами їх цифр або до зміни мови програмування на ту, яка найбільше придатна до даної задачі і дозволяє безпосередньо працювати з раціональними числами, не турбуючись про кількість цифр чисельника і знаменника. Наприклад, для IBM286 це може бути мова аналітичних обчислень Reduce, для IBM486 — Matematica.

2. Латиниця. Після встановлення взаємно однозначної відповідності між літерами й числами, записаними у системі числення з основою 26 (див. функції на початку поданої далі програми) і першого запису у вихідний файл задача еквівалентна переліку у певному порядку $(l-1)$-ої k-елементної підмножини множини всіх натуральних чисел в межах від 1 до 25 включно. Програма коротка, прозора і майже не вимагає додаткових пояснень. Зауважимо лише, що у поданій програмі:

`m` — номер літери у записі числа, яку треба міняти при переході до наступного "слова";

`change` — булева змінна, яка набирає значення `true`, якщо "слово" файлу LATIN.DAT не задовольняє умови (i–iii).

```
const n_max=25;                          label NEXT;
var num: array[0..n_max] of byte;  change: boolean;
    sym: array[0..n_max] of char;  i,j,k,l,m: byte;
      o: text;
           {Функція-літера образ аргумента-числа}
function s(n:byte): char;                   begin
if n= 0 then s:='a' else if n= 1 then s:='b' else
if n= 2 then s:='c' else if n= 3 then s:='d' else
if n= 4 then s:='e' else if n= 5 then s:='f' else
if n= 6 then s:='g' else if n= 7 then s:='h' else
if n= 8 then s:='i' else if n= 9 then s:='j' else
if n=10 then s:='k' else if n=11 then s:='l' else
if n=12 then s:='m' else if n=13 then s:='n' else
if n=14 then s:='o' else if n=15 then s:='p' else
if n=16 then s:='q' else if n=17 then s:='r' else
if n=18 then s:='s' else if n=19 then s:='t' else
if n=20 then s:='u' else if n=21 then s:='v' else
if n=22 then s:='w' else if n=23 then s:='x' else
if n=24 then s:='y' else if n=25 then s:='z'  end;
           {Функція-число образ аргумента-літери}
function n(s: char):byte;                   begin
if s='a' then n:= 0 else if s='b' then n:= 1 else
```

```
if s='c' then n:= 2 else if s='d' then n:= 3 else
if s='e' then n:= 4 else if s='f' then n:= 5 else
if s='g' then n:= 6 else if s='h' then n:= 7 else
if s='i' then n:= 8 else if s='j' then n:= 9 else
if s='k' then n:=10 else if s='l' then n:=11 else
if s='m' then n:=12 else if s='n' then n:=13 else
if s='o' then n:=14 else if s='p' then n:=15 else
if s='q' then n:=16 else if s='r' then n:=17 else
if s='s' then n:=18 else if s='t' then n:=19 else
if s='u' then n:=20 else if s='v' then n:=21 else
if s='w' then n:=22 else if s='x' then n:=23 else
if s='y' then n:=24 else if s='z' then n:=25  end;
begin{latin.pas}
                                {Зчитування даних}
assign(o,'LATIN.DAT');  reset(o);  readln(o,k,l);
for j:=1 to k do read(o,sym[j]);      close(o);
assign(o,'LATIN.RES');  rewrite(o);

                   {Визначення літер першого "слова"}
for j:=1 to k do num[j]:=n(sym[j]);  change:=false;
   if num[1]=0 then begin num[1]:=1; change:=true end;
for j:=1 to k-1 do           if num[j+1]<=num[j] then begin
change:=true;for i:=j to k-1 do num[i+1]:=num[i]+1      end;

         {Запис повідомлення про відсутність "слів"
                           або 1-го шуканого "слова"}
if  (num[k]>n_max) or
   ((num[k]=n_max) and (not change)) then             begin
writeln(o,'Таких "слів" немаї!');  close(o);  halt      end
                else if change then                   begin
                                    for i:=1 to k do  begin
                sym[i]:=s(num[i]);  write(o,s(num[i]))  end;
                                    writeln(o); j:=1    end
                                 else             j:=0;  m:=k;
                           {Запис решти можливих слів}
while j<l do                                          begin
      j:=j+1; NEXT: num[m]:=num[m]+1;  sym[m]:=s(num[m]);
if num[m]>n_max-k+m then                    begin m:=m-1;
if m=0 then begin close(o); halt end
       else  goto NEXT                        end
                    else for i:=m+1 to k do           begin
                  num[i]:=num[i-1]+1; sym[i]:=s(num[i]) end;
for i:=1 to k do write(o,sym[i]);  writeln(o);
if num[k]<n_max then m:=k                               end;
close(o);  halt                                         end.
```

3. Тест. Малюнок зручно зберігати і опрацьовувати як таблицю логічних змінних b, у якій істинному елементу відповідає зафарбований квадрат, а хиб-

ному — незафарбований. Всі елементи 0-го рядка й 0-го стовпчика зручно покласти хибними для того, щоб спростити частину програми, що визначає номер фігури, якій належить кожен зафарбований квадрат.

Перед розбиттям малюнку на фігури кількість фігур N_{fig} покладаємо рівною 0. Аналіз малюнку здійснюємо послідовним розглядом рядків клітин згори донизу, а в кожному рядку — зліва направо. При цьому розгляді для кожної зафарбованої клітини можливі лише наступні випадки.

1. Найближчі сусідні квадрати ліворуч та згори зафарбовані і належать одній фігурі. Тоді розглядувана клітина належить цій же фігурі.

2. Найближчі сусідні квадрати ліворуч та згори зафарбовані і належать різним[8] w_0-ій та w_1-ій фігурам, де $w_0 < w_1$. Тоді:

   - запам'ятовуємо масивом `w` те, що дана клітина належить w_0-ій фігурі;
   - для кожної вже розглянутої зафарбованої клітини, що належить w_1-ій фігурі, вважатимемо з даного моменту, що вона належить w_0-ій фігурі;
   - для кожної вже розглянутої зафарбованої клітини, що належать w-ій фігурі, де $w_1 < w$, вважатимемо з даного моменту, що вона належать $(w-1)$-ій фігурі.

3. Лише один з найближчих сусідніх квадратів ліворуч та згори зафарбовано. Тоді розглядувана клітина належить тій же фігурі, що і згаданий сусідній зафарбований квадрат.

4. Найближчі сусідні квадрати ліворуч та згори не зафарбовано. Тоді кількість фігур N_{fig} збільшуємо на 1 і вважаємо, що розглядувана клітина належить фігурі з номером N_{fig}.

Далі для k в межах від 1 до N_{fig} знаходимо кількість клітин k-ої фігури і найменший та найбільший номери рядків і стовпчиків, які містять клітини k-ої фігури.

Для k в межах від 1 до $N_{fig} - 1$ включно і для k_k в межах від $k-1$ до N_{fig} включно k-ту і k_k-ту фігури порівнюємо лише за умови, що вони мають однакові додатні кількості клітин і габарити. Якщо фігури можна сумістити одним з наступних способів:

- паралельним переносом;
- паралельним переносом і симетрією відносно горизонтальної прямої;
- паралельним переносом і симетрією відносно вертикальної прямої;

[8] Маємо на увазі — на даний момент аналізу малюнку. Тобто з урахуванням не всіх клітин на малюнку, а лише тих, які розглянуто на даний момент.

- паралельним переносом і поворотом на 180°;
- паралельним переносом і поворотом на 90°;
- паралельним переносом і поворотом на 270°;
- паралельним переносом, поворотом на 90° і симетрією відносно горизонтальної прямої;
- паралельним переносом, поворотом на 270° і симетрією відносно горизонтальної прямої,

то виключаємо k_k-у фієуру з переліку *різних стосовно геометрії Евкліда* фігур. Поданим переліком вичерпуються всі можливі перетворення рівних фігур, означених умовою тесту. Щоб це проілюструвати, подаємо малюнок, на якому образи всіх перетворень для крайньої ліворуч фігури подано у тому порядку, як їх перелічено вище.

Лише на перший погляд програма видається громіздкою (136 рядків тексту). Але зауважте: 8 блоків перевірки рівності фігур (кожен по 2 рядки коментарів і 6 рядків операторів) майже ідентичні, тому процес створення програми можна значно полегшити, використовуючи можливості копіювання блоків текстовими редакторами. Зменшення кількості блоків до 2 (в циклах за змінними, що набирають 4 значення) можливе і виправдане з академічної точки зору. Така модифікація програми на олімпіаді — це змарнований час, який не підвищує швидкість опрацювання даних і не зменшує вимоги до пам'яті. Цікаво було б використати динамічної структури даних для опису фігур у тих випадках, коли навіть таблицю логічних елементів b неможливо зберігати у пам'яті комп'ютера. Однак такий алгоритм недосяжний для більшості учасників II етапу, серед яких, доречі, багато володіють лише одним з діалектів Basic'а. Тому тести не передбачають обов'язкової реалізації цієї ідеї.

```
const  m_max=60;  n_max=60;  fig_max=127;      label NEXT;
var       i,j,k,ii,jj,kk,m,n,t0,t1,du,duk,rl,rlk: shortint;
l,r,d,u,num: array[1..fig_max] of byte;   nfig,w0,w1: word;
   w: array[1..m_max,1..n_max] of word;  s: char;  o: text;
   b: array[0..m_max,0..n_max] of boolean;       t:boolean;
begin{test.pas}
             {Зчитування даних і запам'ятовування малюнку
 масивом b з порожніми (false) першими стовпчиком і рядком}
assign(o,'TEST.DAT');  reset(o);  readln(o,m,n);
for i:=0 to m do for j:=0 to n do b[i,j]:=false;
for i:=1 to m do                                          begin
for j:=1 to n do                                          begin
```

```
                read(o,s);  w[i,j]:=0;
if s='Ш' then b[i,j]:=true else b[i,j]:=false           end;
readln(o)                                               end;
close (o);  nfig:=0;
      {Знаходження кількості фігур nfig та запам'ятовування
  масивом w того, якій фігурі належить відповідний квадрат}
for i:=1 to m do for j:=1 to n do                      begin
if b[i,j] then if b[i,j-1] and b[i-1,j] then
if w[i,j-1]=w[i-1,j] then w[i,j]:=w[i-1,j]
                     else                              begin
if w[i,j-1]<w[i-1,j]
   then begin w1:=w[i-1,j]; w0:=w[i,j-1] end
   else begin w0:=w[i-1,j]; w1:=w[i,j-1] end;
for ii:=1 to i-1 do for jj:=1 to n do
if w[ii,jj]=w1 then w[ii,jj]:=w0 else
if w[ii,jj]>w1 then w[ii,jj]:=w[ii,jj]-1;
for jj:=1 to j-1 do
if w[i,jj]=w1 then w[i,jj]:=w0 else
if w[i,jj]>w1 then w[i,jj]:=w[i,jj]-1;
nfig:=nfig-1;      w[i,j ]:=w0                          end
                                      else
if b[i,j-1] then                  w[i,j]:=w[i,j-1]
            else if b[i-1,j] then w[i,j]:=w[i-1,j]
                             else
begin nfig:=nfig+1; w[i,j]:=nfig end                   end;

          {Запам'ятовування меж фігур масивами d, u, l, r}
for k:=1 to nfig do                                    begin
l[k]:=0;  r[k]:=0;  d[k]:=0;  u[k]:=0;  num[k]:=0;
j:=0;    while   u[k]=0             do begin j:=j+1;
i:=0;    while ((u[k]=0) and (i<n)) do begin i:=i+1;
if w[j,i]=k then u[k]:=j             end end;
j:=0;    while   l[k]=0             do begin j:=j+1;
i:=0;    while ((l[k]=0) and (i<m)) do begin i:=i+1;
if w[i,j]=k then l[k]:=j             end end;
j:=m+1;  while   d[k]=0             do begin j:=j-1;
i:=0;    while ((d[k]=0) and (i<n)) do begin i:=i+1;
if w[j,i]=k then d[k]:=j             end end;
j:=n+1;  while   r[k]=0             do begin j:=j-1;
i:=0;    while ((r[k]=0) and (i<m)) do begin i:=i+1;
if w[i,j]=k then r[k]:=j             end end           end;

   {Запам'ятовування кількості квадратів фігур масивом num}
for i:=1 to m do for j:=1 to n do
if w[i,j]>0 then num[w[i,j]]:=num[w[i,j]]+1;

                    {Встановлення кількості різних фігур}
```

```
for k:=1 to nfig-1 do                                    begin
du:=d[k]-u[k]; rl:=r[k]-l[k];
if num[k] >0      then for kk:=k+1 to nfig do
if num[k]=num[kk] then                                   begin
duk:=d[kk]-u[kk];  rlk:=r[kk]-l[kk];  t:=false;
if ((duk=du) and (rlk=rl)) then                          begin

                          {Перевірка рівності фігур: зсув}
t0:=u[kk]-u[k];  t1:=l[kk]-l[k];         t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; ii:=t0+i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; jj:=t1+j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false   end end;
if t then goto NEXT;
                                {Перевірка рівності фігур:
                     зсув і симетрія відносно горизонталі}
t0:=d[kk]+u[k];  t1:=l[kk]-l[k];         t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; ii:=t0-i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; jj:=t1+j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false   end end;
if t then goto NEXT;
                                {Перевірка рівності фігур:
                      зсув і симетрія відносно вертикалі}
t0:=u[kk]-u[k];  t1:=r[kk]+l[k];         t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; ii:=t0+i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; jj:=t1-j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false    end end;
if t then goto NEXT;

              {Перевірка рівності фігур: зсув і 1/2 оберта}
t0:=d[kk]+u[k];  t1:=r[kk]+l[k];         t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; ii:=t0-i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; jj:=t1-j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false   end end;
if t then goto NEXT                                    end;

if ((duk=rl) and (rlk=du)) then                          begin

              {Перевірка рівності фігур: зсув і 1/4 оберта}
t0:=d[kk]+l[k];  t1:=l[kk]-u[k];         t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; jj:=t1+i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; ii:=t0-j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false   end end;
```

```
if t then goto NEXT;

              {Перевірка рівності фігур: зсув і 3/4 оберта}
t0:=u[kk]-l[k];  t1:=r[kk]+u[k];          t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; jj:=t1-i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; ii:=t0+j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false   end end;
if t then goto NEXT;
                                  {Перевірка рівності фігур:
          зсув, 1/4 оберта і симетрія відносно горизонталі}
t0:=u[kk]-l[k];  t1:=l[kk]-u[k];          t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; jj:=t1+i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; ii:=t0+j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false   end end;
if t then goto NEXT;
                                  {Перевірка рівності фігур:
          зсув, 3/4 оберта і симетрія відносно горизонталі}
t0:=d[kk]+l[k];  t1:=r[kk]+u[k];          t:=true; i:=u[k]-1;
while (t and (i<d[k])) do begin i:=i+1; jj:=t1-i; j:=l[k]-1;
while (t and (j<r[k])) do begin j:=j+1; ii:=t0-j;
if ((w[i,j]= k) and (w[ii,jj]<>kk))
or ((w[i,j]<>k) and (w[ii,jj]= kk)) then t:=false   end end
                                                        end;
              NEXT:  if t then num[kk]:=0          end end;

                        {Підрахунок кількості різних фігур}
k:=0;  for i:=1 to nfig do if num[i]>0 then k:=k+1;

                                          {Запис відповіді}
assign(o,'TEST.RES');  rewrite(o);   writeln (o,nfig,' ',k);
close (o);  halt                                        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. Банківська справа, 20 балів. Молодий ломбардець Гуччо Бальйоні (герой серії творів французького письменника Моріса Дрюона) часто виконував таємні доручення сучасних йому можновладців Франції, Англії та Італії. Не маючи можливості протягом трьох років безпосередньо самому займатися справами, він вирішив отримати зиск, вклавши гроші у справи паризьких банкірів. Виявилося, що розмір винагороди (% зиску) різний у різних банкірів. Звичайно, чим більшу суму, він дасть банкіру, тим більше отримає через 3 роки. І не менше, ніж дав банкіру. Але в кожного банкіра % зиску різний для різних сум. Нажаль, за розрахунками Гуччо, йому ніколи не отримати більше 1000 ліврів (середньовічних французьких монет). Створіть програму PROFIT.*, яка домоможе молодому ломбардцю отримати якнайбільші статки і взяти шлюб з коханою Марі де Крессе.

Перший рядок вхідного файлу PROFIT.DAT у вказаному порядку містить 2 натуральних числа: m — кількість ліврів, яку молодий Гуччо може використати для збагачення, та n — кількість паризьких банкірів, де $m < 100$, $n < 16$. Для j в межах від 1 до m $(j+1)$-ий рядок цього ж файлу містить послідовність n натуральних чисел. k-ий член цієї послідовності — це кількість монет, яку отримає Гуччо через три роки від k-го банкіра, віддавши йому j монет перед від'їздом. Наприклад,

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

Перший рядок вихідного файлу PROFIT.RES має містити найбільшу кількість ліврів, яку може мати молодий ломбардець через 3 роки, використавши m ліврів належним чином. Другий рядок цього ж файлу має містити послідовність n невід'ємних цілих чисел. k-ий член цієї послідовності — це кількість ліврів, яку має дати Гуччо k-му банкіру, щоб отримати максимальний зиск від своїх капіталовкладень (потрібно навести хоча б один з варіантів розподілу). Наприклад,

```
14
0 0 1 2 0 0 0 0 0 0
```

2. Криниця, 20 балів. Будівельники отримали завдання спорудити огорожу, яка на плані має вигляд замкненої ламаної без самоперетинів, і впорядкувати криницю. Створіть програму WELL.*, яка допоможе з'ясувати:

- яка відстань від криниці до огорожі;
- чи буде криниця знаходитися всередині огородженої ділянки.

Вхідний файл WELL.DAT містить лише натуральні числа, які не перевищують 30. 1-ий рядок цього файлу містить n — кількість прямолінійних частин огорожі, довжина кожної з яких виражається натуральним числом. 2-ий рядок цього самого файлу містить послідовність n абсцис (у деякій декартовій системі координат) всіх початків (кінців) прямолінійних ділянок огорожі в порядку обходу її периметру. 3-ій рядок вхідного файлу містить послідовність n ординат всіх початків (кінців) прямолінійних ділянок огорожі в тому самому порядку, що й 2-ий рядок файлу. 4-ий рядок цього самого файлу містить координати криниці (у тій самій системі координат).

1-ий рядок вихідного файлу WELL.RES має містити *відстань* від криниці до огорожі, подану:

- або цілим числом;
- або коренем квадратним натурального числа, яке не є точним квадратом іншого натурального числа, у вигляді `sqrt(...)`;
- або нескоротним дробом, чисельник і знаменник якого — натуральні числа. Якщо знаменник цього дробу дорівнює 1, то риску дробу / і знаменник записувати непотрібно.

2-ий рядок цього ж файлу має містити один з прийменників англійської мови: `in, on, out,` якщо криниця буде знаходитися відповідно всередині огорожі, на одній з ланок огорожі чи зовні її.

Наприклад, нехай перші три рядки вхідного файлу WELL.DAT мають вигляд

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

Якщо 4-ий рядок цього ж файлу містить запис або `2 2`, або `3 1`, або `5 6`, то файл WELL.RES має відповідно вигляд

або `1/5` `in` , або `0` `on` , або `sqrt(2)` `out` .

3. Лабіринт, 20 балів. Створіть програму LABIRINT.*, яка допоможе герою твору Джерома К. Джерома "Троє у човні, не рахуючи собаки" Гаррісу якнайшвидше вийти з Гемптон-Кортського лабіринту. Лабіринт на плані має вид прямокутника, сторони якого спрамовані із заходу на схід або з півночі на

південь (наскільки це можливо, враховуючи форму Землі). Розміри лабіринту в ярдах (англійська міра довжини, приблизно 91,44 см) вздовж паралелі та мередіану виражаються натуральними числами n та m відповідно, а його загальна площа mn не перевищує 10000 (квадратних ярдів). Всю територію лабіринту поділено на квадрати з довжиною сторони 1 ярд. Частину квадратів засаджено кущами з колючками, які можна лише обійти.

Перший рядок вхідного файлу LABIRINT.DAT містить два натуральних числа m і n — розміри лабіринту в ярдах. Наступні m рядків по n символів цього самого файлу містять схему лабіринту, в якій символ " " (прогалина) означає вільний квадрат поверхні, літера кирилиці Ш — квадрат з кущем, хрестик (знак додавання) + — початкове розташування Гарріса.

Наприклад,

```
9 9
ШШШШШШШШШ
Ш       Ш
Ш ШШШШШШШ
Ш Ш   +   Ш
Ш Ш Ш Ш Ш Ш
Ш Ш Ш Ш Ш Ш
Ш   Ш     Ш
ШШШШШШШШШ
```

Виходу з лабіринту відповідають елементи 1-го та m-го рядка і 1-ий та n-ий елемент кожного рядка, якщо вони відмінні від "Ш". Рух у лабіринті здійснюють у напрямку сторін світу на вільний від кущів сусідній квадрат. Рухові на північ відповідає рух вгору на один символ на схемі, поданій файлом LABIRINT.DAT, на південь, захід чи схід — відповідно вниз, ліворуч чи праворуч на схемі.

Перший рядок вихідного файлу LABIRINT.RES має містити ціле число — найменшу кількість кроків довжиною 1 ярд, які має зробити Гарріс, щоб вийти з лабіринту. Наступний рядок цього самого файлу має містити опис такого найкоротшого шляху — опис кроків від першого до останнього, у якому літера n позначає крок на північ (англійською north), s — на південь (south), e — на схід (east), w — на захід (west). Якщо Гарріс вже знаходяться на виході, то перший рядок файлу LABIRINT.RES містить лише число 0, а другий рядок відсутній. Якщо вийти з лабіринту неможливо (не будемо обговорювати, яким чином у цьому випадку Гарріс потрапив у відповідне місце лабіринту), то єдиний рядок вихідного файлу містить єдине число -1. Для поданого LABIRINT.DAT файл LABIRINT.RES має вигляд

```
22
wwwsssswwnnnnnneeeeeee
```

3.3 Завдання II туру

4. Куб, 24 бали. Відстанню на поверхні многоєранника між двома точками назвемо найменшу довжина ламаної, що з'єднує дві дані точки, і всі ланки

якої (ламаної) належать поверхні багатоєранника. Нехай у декартовій системі координат координати вершин куба дорівнюють 0 або деякому натуральному числу a, яке не перевищує 15. Створіть програму CUBE.*, яка вираховує *квадрат відстані на поверхні куба* між точками з цілими координатами.

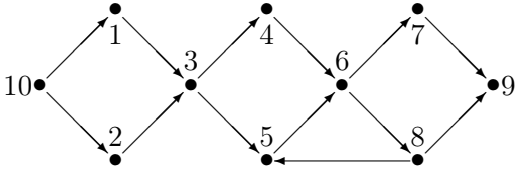
Перший рядок вхідного файлу CUBE.DAT містить у вказаному порядку 2 натуральних числа — a і n. Для j в межах від 1 до n включно $(j+1)$-ий рядок цього самого файлу містить 6 цілих невід'ємних чисел — відповідно координати (порядок усталений: спочатку абсциса x, потім — ордината y, і лише потім — апліката z) точки, що *перебувають на грані куба і в координатній площині xy*, і довільної точки на поверхні куба. Наприклад,

```
10 2
1 2 0  2 1 0
9 5 0 10 7 1
```

Вихідний файл CUBE.RES утворюється з CUBE.DAT дописуванням у кожен рядок, починаючи з 2-го, шуканого квадрату довжини ламаної, що з'єднує точки, чиї координати записано у цьому ж рядку. Для поданого прикладу вхідного файлу CUBE.DAT файл CUBE.RES має такий вигляд.

```
10 2
1 2 0  2 1 0  2
9 5 0 10 7 1  8
```

5. Вуличні перегони, 30 балів. Організаційний комітет велосипедних перегонів Лазуровий Берег–99 звернувся до жандармерії Сан-Тропе дозволити провести велосипедні перегони вулицями цього курортного містечка таким чином, щоб кожен учасник мав би певну свободу у виборі шляху до фінішу. Жандармерія у відповідь надіслала в організаційний комітет план проведення таких перегонів. Один з можливих варіантів такого плану подано наступним малюнком, на якому пункти-перехрестя позначено кругами • і занумеровано натуральними числами в межах від 1 до n=10, а окремі ділянки перегонів — вулиці з одностороннім рухом — позначено стрілками.



На цьому плані:

- стартом — пунктом, з якого можна досягнути будь-який пункт перегонів, і який неможливо досягнути з будь-якого іншого пункту — є пункт 10;
- фінішом — пунктом, який можна досягнути з будь-якого іншого пункту, і з якого неможливо досягнути жоден інший пункт — є пункт 9.

Деякі пункти неможливо уникнути на шляху від старту до фінішу, не рахуючи останніх. Для поданого на малюнку плану це пункти 3 і 6.

Деякі з пунктів, які неможливо уникнути на шляху від старту до фінішу, розбивають план перегонів на два плани без спільних стрілок, але з єдиним спільним пунктом, що є фінішом для одного плану і стартом для іншого. Для поданого на малюнку плану це пункт 3.

Створіть програму RACE.*, яка допоможе членам організаційного комітету провести аналіз поданого плану (дивись далі вимоги до вихідного файлу).

Кількість рядків вхідного файлу RACE.DAT дорівнює n — кількості всіх пунктів перегонів (не перевищує 111). Для j в межах від 1 до n включно j-ий рядок цього самого файлу містить номери кінцевих пунктів тих стрілок, які виходять з j-го пункту. Наприклад, для поданого плану файл RACE.DAT має наступний вигляд.

```
3
3
4 5
6
6
7 8
9
5 9


1 2
```

Якщо якийсь рядок містить лише погалини (у поданому прикладі це 9-ий рядок), то з відповідного пункту не виходить жодна стрілка, тобто цей пункт — фініш для запропонованого плану.

Перший рядок вихідного файлу RACE.RES має містити у вказаному порядку номери пунктів, що є стартом і фінішом. Другий рядок — кількість пунктів, які неможливо уникнути на шляху від старту до фінішу та номери цих пунктів у порядку зростання. Третій рядок — кількість пунктів, якими можна розбити поданий план перегонів на окремі плани, та номери цих пунктів у порядку зростання. Наприклад,

```
10 9
2 3 6
1 3
```

6. Криптограма, 22 бали. Перший рядок вхідного файлу CRIPTO.DAT містить натуральне число, яке є основою системи числення і лежить в межах від 5 до 10 включно. Другий рядок цього ж файлу містить зашифрований запис дії додавання, в якому всі доданки розділено знаком +, перед сумою стоїть знак =, а кожну цифру замінено на літеру, причому:

- однакові цифри замінено на однакові літери;
- різні цифри замінено на різні літери;
- розрізняються великі та малі літери.

Можуть використовуватися як літери латиниці, так і літери кирилиці. Кількість доданків не перевищує 6, запис кожного з них містить не більше 9 цифр, але й не менше 2 цифр. Наприклад,

```
10
ten+ten+forty=sixty
```

Розділення доданків, суми, знаків + і = додатковими пробілами не передбачено.

Створіть програму CRIPTO.*, яка запише у вихідний файл CRIPTO.RES всі способи дешифрації поданого запису дії додавання (по одному у кожному рядку без повторення). Для поданого прикладу вхідного файлу CRIPTO.DAT файл CRIPTO.RES має такий вигляд.

```
850+850+29786=31486
```

3.4 Вказівки до перевірки

Перевірку тестів здійснювати у присутності учнів після того, як усі роботи буде скопійовано на дискети. Кожна робота перевіряється не менше, ніж 2-ма членами журі у присутності учасника олімпіади — автора роботи. і ці члени журі, і учасник мають бути з різних районів. Для зручності роботи журі члени журі можуть редагувати у тексті програми розширення вхідного файлу, і лише його. Це розширення має вигляд d_n для n — номеру тесту в межах від 1 до 9 включно (d_1, d_2, ... , d_9), або dm для m — номеру тесту, який перевищує 9 (d10, d11, d12 тощо). Члени журі заповнюють додаток до протоколу журі результатами перевірки олімпіадної роботи і підписують його разом з учасником олімпіади.

1. Банківська справа. За успішне проходження 1-го тесту — 8 балів, 2-го — 12 балів. Перевіряється знаходження оптимального розв'язку (тести 1–2) і вміння застосовувати метод динамічного програмування (тест 2). Якщо неправильно вказано максимальну суму, яку отримає Гуччо після повернення, або для вказаного розподілу капіталу максимальний зиск не досягається, то знімається половина балів за тест. 2-ий рядок файлу PROFIT.RES може відрізнятися від того, який подано як відповідь. Важливо лише, щоб сума відповідних чисел файлу PROFIT.DAT дорівнювала першому числу файлу PROFIT.RES поданої відповіді.

1.1 Якщо PROFIT.DAT має вигляд

```
5 5
   1   2  52  13  17
  42  22  62  36  43
  47  51  67  87  47
  65 107  90 134  67
 107 158 134 153  77 ,
```

то PROFIT.RES може мати вигляд

```
186
0 0 1 4 0 .
```

1.2 Якщо PROFIT.DAT має вигляд

```
99 15
   1   1  13   4   5  11   5   3   6   7   2   8   2  13   1
   6  15  19  16  10  22  18  14  11  10   7  15   6  26   6
  14  18  33  21  22  37  26  28  24  11  10  18  14  27  15
  15  30  43  33  33  46  30  39  33  26  20  33  18  38  20
  17  42  51  47  41  55  45  50  39  27  31  48  30  50  23
  20  56  63  56  54  66  50  52  49  32  45  51  35  59  38
  23  59  68  59  63  74  55  55  59  44  57  53  37  60  40
  35  72  74  69  70  86  63  61  72  57  60  55  39  61  43
  46  76  83  73  76  99  68  66  82  68  74  60  47  66  55
  58  85  94  80  85 108  82  77  84  76  81  62  48  75  66
  60  99 107  91  99 113  93  89  92  86  84  66  63  81  68
  75 104 108  93 113 122 104  91 103  87  89  80  72  86  75
  89 110 114 101 115 135 116  94 117  95  91  94  79  90  88
  98 112 124 115 124 140 123  98 120 106 106 104  90  99  90
 110 115 128 129 135 147 138 101 121 109 107 110  96 105  92
 117 123 135 140 148 156 142 107 130 123 117 114 107 114 106
 123 129 145 152 155 160 153 120 142 137 120 126 112 119 117
 130 144 148 153 159 174 158 133 154 152 125 132 125 120 121
 131 152 155 154 162 185 163 139 163 156 128 137 127 131 126
 143 162 164 168 163 189 177 150 164 159 131 144 134 134 127
 151 174 173 172 176 190 185 152 170 169 135 157 142 141 128
 166 177 174 173 178 203 200 154 174 181 150 165 143 150 141
 180 185 184 188 190 214 214 169 185 193 151 171 148 152 153
 184 198 198 189 204 228 222 180 191 200 163 182 154 162 161
 199 200 213 190 214 234 230 190 204 210 178 190 162 177 165
 202 207 224 202 215 244 243 197 207 220 180 205 174 191 172
 203 211 233 216 216 252 257 211 220 231 195 209 177 201 185
 206 212 246 225 224 259 260 223 234 233 204 221 181 209 192
 214 223 258 230 225 269 265 224 241 237 206 231 194 215 197
 222 237 269 237 227 282 269 225 247 251 219 238 198 222 203
 232 244 279 249 240 289 278 235 252 255 226 249 208 225 218
 233 249 293 257 248 303 289 242 261 258 241 257 211 230 228
 246 257 296 264 251 308 299 255 274 261 244 269 216 236 232
 247 271 307 266 261 309 309 264 286 264 258 282 226 244 244
 259 274 313 280 269 323 320 269 295 266 264 288 233 253 258
 270 286 328 286 271 330 325 283 299 272 266 296 241 262 262
 273 293 343 294 280 336 331 295 313 283 275 304 255 266 263
 288 301 358 298 293 338 338 296 328 289 286 317 268 272 273
 293 314 361 304 300 340 353 297 331 291 287 327 273 285 275
 304 321 369 312 302 355 358 312 336 303 300 334 281 290 286
 310 328 376 327 308 364 369 320 339 308 307 342 287 297 287
 313 343 385 334 322 377 378 327 344 321 316 347 297 311 291
```

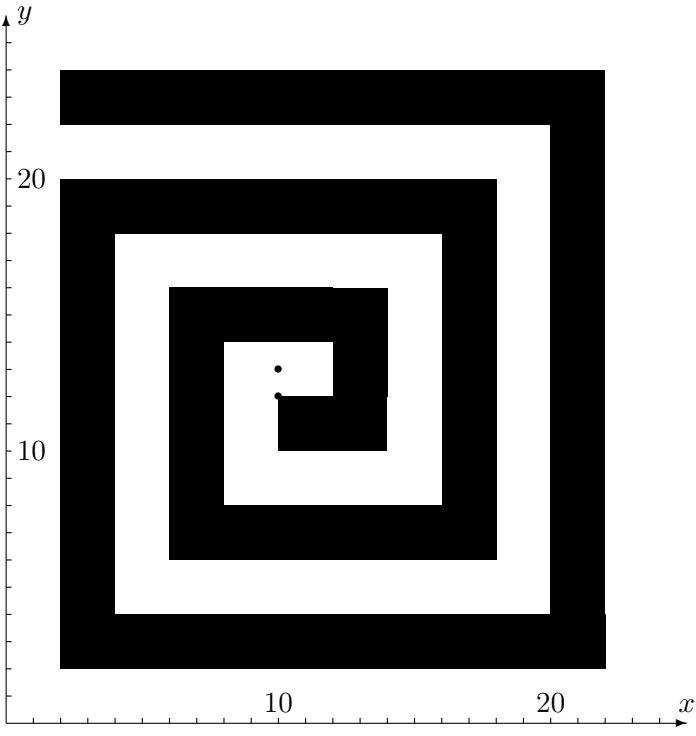
```
323 344 393 347 336 381 393 338 355 326 330 351 305 317 304
333 345 400 351 344 389 397 347 365 332 339 363 316 320 318
340 356 409 360 352 400 411 356 375 345 346 364 331 321 329
354 360 419 375 364 401 413 362 389 349 361 373 346 333 335
355 361 429 387 372 409 425 377 394 352 364 388 348 339 338
362 369 431 393 377 414 427 382 407 356 370 397 349 343 344
364 381 438 394 388 422 434 393 408 358 374 405 362 353 348
377 389 445 395 399 433 441 397 417 364 376 418 371 366 349
388 400 458 401 412 446 455 399 421 372 388 430 378 372 352
402 408 461 415 422 455 457 406 422 383 391 433 380 378 362
409 417 466 426 429 457 468 421 426 396 398 444 395 385 363
421 419 475 436 443 467 480 426 436 410 403 448 397 391 364
426 433 478 438 457 473 483 430 444 424 410 451 399 401 369
432 442 491 451 460 485 495 441 447 430 415 465 406 405 373
433 455 504 452 463 493 500 451 456 439 422 477 409 416 377
436 469 518 454 467 507 511 453 463 446 429 489 423 423 391
445 470 532 463 473 521 525 462 473 452 431 491 438 431 406
450 477 543 471 474 523 537 465 487 456 440 494 448 441 421
465 481 544 475 477 538 552 478 490 464 445 504 456 446 426
477 483 555 489 478 544 566 486 497 476 447 515 470 451 428
491 496 557 493 493 550 579 498 512 478 448 520 481 455 435
493 507 558 503 508 561 590 507 518 489 457 526 486 461 437
498 519 567 512 509 572 598 512 523 498 461 534 500 468 447
503 528 578 516 517 582 603 521 532 505 463 542 512 479 451
510 543 590 529 524 583 617 530 540 512 472 555 514 480 461
513 544 601 543 537 584 620 543 541 527 473 557 529 485 476
514 545 602 547 542 596 626 556 553 542 479 569 533 500 482
528 550 608 562 551 601 633 561 564 556 488 582 534 504 490
540 553 620 574 563 612 644 571 569 561 492 586 541 515 497
544 557 630 578 571 624 657 583 584 575 504 589 548 518 500
559 558 632 592 584 638 661 585 594 588 517 600 549 530 508
564 559 644 599 585 644 672 600 603 594 522 614 557 545 516
570 564 646 601 586 655 678 610 609 598 525 617 568 559 525
571 569 661 606 596 670 680 619 621 604 538 623 577 564 526
578 577 673 607 598 683 689 628 634 618 541 638 579 574 536
586 579 686 610 609 686 702 637 636 625 542 652 591 588 544
599 582 693 619 613 690 716 644 642 636 544 653 592 590 552
613 585 694 630 628 691 718 646 644 640 554 662 603 600 558
619 599 701 637 640 702 724 655 658 654 561 670 607 605 565
626 607 715 652 651 707 729 666 661 658 565 677 617 606 570
641 619 718 658 665 711 744 672 669 659 571 679 620 617 583
647 620 733 663 680 718 751 682 671 668 578 681 633 619 585
653 624 740 665 691 726 766 692 679 682 590 684 648 634 593
658 635 751 676 698 729 778 694 694 687 599 696 663 635 599
668 642 753 684 711 737 788 702 699 695 609 710 676 637 613
670 656 761 697 717 747 801 709 707 702 622 721 677 641 620
683 660 776 701 724 758 802 711 715 703 631 728 682 642 623
689 675 778 703 738 770 817 713 730 712 640 740 687 644 625
695 688 791 715 743 777 827 715 734 714 642 754 693 659 633
698 702 794 726 751 784 832 721 748 723 655 755 707 665 640
706 708 802 740 766 787 839 725 756 732 668 765 710 674 645
709 723 808 754 767 789 849 734 766 742 670 778 717 680 657
719 724 820 766 777 799 862 740 775 743 671 788 723 695 667
729 738 821 769 788 810 864 750 787 755 676 790 726 706 676
734 752 833 778 799 813 870 762 795 763 683 798 734 715 685
738 758 838 786 802 823 878 771 797 772 684 809 747 723 691
744 764 846 791 817 831 890 782 812 779 696 813 748 733 703 ,
```

то PROFIT.RES може мати вигляд

```
960
0 0 38 0 12 10 27 5 0 0 0 5 0 2 0 .
```

2. Криниця. За успішне проходження кожного з 10-ти тестів — по 2 бали. Якщо неправильно вказано відстань або розташування криниці по відношенню до оргорожі, то знімається по 1 балу за кожен з цих недоліків (окремо для кожного тесту).

Тести 1–3 перевіряють правильність встановлення взаємного розташування криниці й огорожі для випадку, коли огороджена ділянка не є опуклою. Поданий далі малюнок містить зображення огородженої ділянки, зафарбоване чорним кольором, і точок розташування криниці для тестів 1–3.



2.1 Якщо WELL.DAT має вигляд

```
24
10 12 12  8  8 16 16  4  4 20 20  2  2 22 22  2  2 18 18  6  6 14 14 10
12 12 14 14  8  8 18 18  4  4 22 22 24 24  2  2 20 20  6  6 16 16 10 10
11 11 ,
```

то WELL.RES має вигляд

```
1
in .
```

2.2 Якщо WELL.DAT має вигляд

```
24
10 12 12  8  8 16 16  4  4 20 20  2  2 22 22  2  2 18 18  6  6 14 14 10
12 12 14 14  8  8 18 18  4  4 22 22 24 24  2  2 20 20  6  6 16 16 10 10
10 12 ,
```

то WELL.RES має вигляд

```
0
on .
```

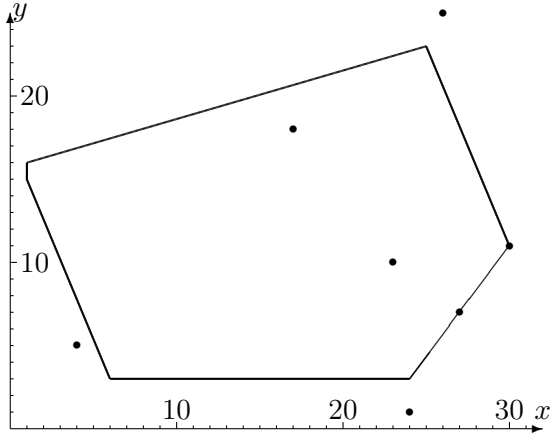
2.3 Якщо WELL.DAT має вигляд

```
24
10 12 12  8  8 16 16  4  4 20 20  2  2 22 22  2  2 18 18  6  6 14 14 10
12 12 14 14  8  8 18 18  4  4 22 22 24 24  2  2 20 20  6  6 16 16 10 10
10 13 ,
```

то WELL.RES має вигляд

```
1
out .
```

У тестах 4–10 робиться наголос на перевірці правильності визначення шуканої відстані і виконання технічних умов щодо її подання. Наступний малюнок містить зображення огорожі (опуклий 6-кутник), і точок розташування криниці для тестів 4–10.



2.4 Якщо WELL.DAT має вигляд

```
6
 1  1  6 24 30 25
16 15  3  3 11 23
24  1 ,
```

то WELL.RES має вигляд

```
2
out .
```

2.5 Якщо WELL.DAT має вигляд

```
6
 1  1  6 24 30 25
16 15  3  3 11 23
30 11 ,
```

то WELL.RES має вигляд

```
0
on .
```

2.6 Якщо WELL.DAT має вигляд

```
6
 1  1  6 24 30 25
16 15  3  3 11 23
27  7 ,
```

то WELL.RES має вигляд

```
0
on .
```

2.7 Якщо WELL.DAT має вигляд

```
6
 1  1  6 24 30 25
16 15  3  3 11 23
23 10 ,
```

то WELL.RES має вигляд

```
5
in .
```

2.8 Якщо WELL.DAT має вигляд

```
6
 1  1  6 24 30 25
16 15  3  3 11 23
26 25 ,
```

то WELL.RES має вигляд

```
sqrt(5)
out .
```

2.9 Якщо WELL.DAT має вигляд

```
6
 1  1  6 24 30 25
16 15  3  3 11 23
 4  5 ,
```

то WELL.RES має вигляд

```
14/13
out .
```

2.10 Якщо WELL.DAT має вигляд

```
6
 1  1  6 24 30 25
16 15  3  3 11 23
17 18 ,
```

то WELL.RES має вигляд

```
64/25
in .
```

3. Лабіринт. За успішне проходження тестів 1–2 — по 3 бали, тесту 3 — 2 бали, тестів 4–6 — по 4 бали. Бали за тест 6 нараховуються лише за умови успішного проходження всіх попередніх тестів.

Тести 1–2 перевіряють можливість роботи програми з масивами різноманітної розмірності.

3.1 Якщо LABIRINT.DAT має вигляд

```
2000 5
ШШШШШШШ
Ш     Ш
.....
Ш  .  Ш
Ш Ш Ш Ш
Ш+Ш Ш Ш
ШШШШ Ш
```

(тут крапками ... позначено 1994 рядки, кожен з яких містить по 5 символів: "Ш Ш"), то LABIRINT.RES має вигляд

```
7
nneesss .
```

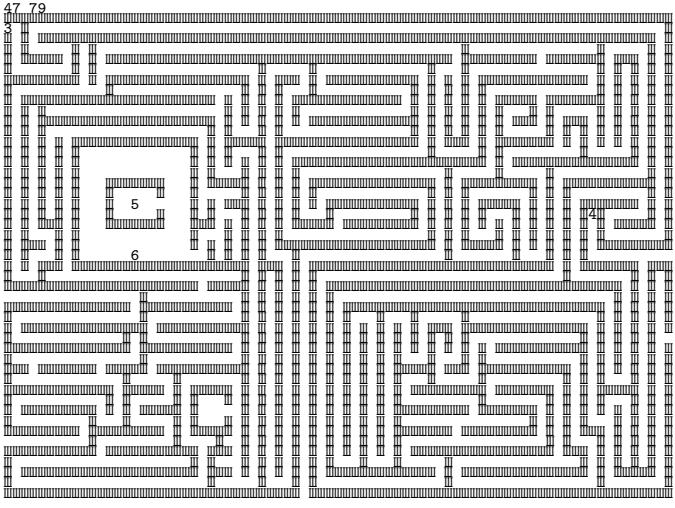
3.2 Якщо LABIRINT.DAT має вигляд

```
5 2000
ШШШШ...ШШШШШШШШШШШ
Ш   ...        +Ш
Ш   ...  ШШШШШШШШ
Ш   ...
ШШШШ...ШШШШШШШШШШШ,
```

(тут крапками ... позначено послідовність 1989 символів: у 2-му та 6-му рядку — "Ш", у 3–5-ому — " "), то LABIRINT.RES має вигляд

```
13
wwwwwsseeeeee .
```

Для тестів 1–2 вигляд файлу LABIRINT.DAT не залежить від тієї частини файлу LABIRINT.DAT, яку позначено крапками. Далі подано загальний вигляд файлу LABIRINT.DAT для тестів 3–6, який відповідає лабіринту розміру 47×79 клітин. На ньому цифрами 3–6 позначено початкове розташування (символ "+" для відповідного тесту, для всіх інших тестів на цьому місці — пропуск " ".) Для тесту 6 на місці цифри "3" має бути символ "Ш".



3.3 Файл LABIRINT.RES містить тільки запис числа 0. 3.4 Файл LABIRINT.RES має вигляд

```
13
ssssseeeeeeee .
```

3.5 Файл LABIRINT.RES має вигляд (для зручності читача другий рядок файлу розбито на частини по 50 символів).

```
986
eeeesssesesеenneennnnwwnnnnnnwwwwwwwwwwwwwwwwssssss
sssssssssseeeeeeeeeeeeeeeeesseeeessswwwwwwwwwwsseeeee
eeeeessswwwwwwwwwwsseessswwwwsseeeessswwwwwwwwsswwww
wwwwwsseeeeeeeeeeeeeeeeeeeeeeennnnwwnnnnnneeeeeesss
sssssssseennnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnwwssswwnnn
nnnnneessssseennnnnnwwwwwwwwwwwwwwwwwwwneeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeesswwsssssssseeeennn
```

```
nnnnneeeeeeeeeeeeeeesswwwwwwwwwwwwsssssssswwwwssssss
sseeeeeeennnnwwsswwnnnneeeeeeeesssssseennnnnnnnwwwwn
neeeeeennnneeeessssseennnnnnnnnneeeessssssssssssswww
wwwwwwssssssssssswwwwwwwwwwwwwwwwwwwwwwwwwwssssss
sssssssssssssseeeeeeeeeeeeeennwwwwnneeeeeeeeeeeee
eeennnnwwwwwwnneeeeeeeessssssswwwwwwwwwwwwsseeeee
eeeeeeeeeeennnnwwnnnnnnnnwwwwwwwwwwnnwssssswwwwne
ennnnwwwwsssswwnnnnwwsssssssssssssswwnnnnnnnnnnnn
nwwsssssssssssssswwnnnnnnnnnnnnnnnneeeeeeeeeeeeee
eeeeeeeeeeeeeeeessssssseennnnnnnnnnwwwwwnnnnnnn
neeeeeeeesswwwwsseeeeeennnnnnnnnnnnnnnnnnnwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwsseeeessswwwwwwnnnnw
```

3.6 Файл LABIRINT.RES містить тільки запис числа -1.

4. Куб. За успішне проходження всього тесту — 24 бали. За відмінності у будь-якому рядку файлу CUBE.RES, починаючи з 2-го рядка, знімається 1 бал (за кожен рядок окремо). Нагадаємо, що файл CUBE.RES отримується з CUBE.DAT дописуванням у кожний рядок, починаючи з 2-го, після координат двох точок квадрату найкоротшої ламаної, що їх сполучає. CUBE.RES має вигляд

```
 13 24
 11  4  0 13  7  2  25
  1 10  0 13 12 11 365
  1  3  0 13  1 12 394
  4  2  0  7  0  2  25
 10 12  0 12  0 11 365
  3 12  0  1  0 12 394
  2  4  0  0  7  2  25
 12 10  0  0 12 11 365
 12  3  0  0  1 12 394
  9 11  0  6 13  2  25
  3  1  0  1 13 11 365
 10  1  0 12 13 12 394
 12  7  0 11  6 13 257
 12 11  0 11 12 13 241
 12  3  0 10  1 13 261
  7  1  0  6  2 13 257
 11  1  0 12  2 13 241
  3  1  0  1  3 13 261
  1  7  0  2  6 13 257
  1 11  0  2 12 13 241
```

```
 1  3  0  3  1 13 261
 6 12  0  7 11 13 257
 2 12  0  1 11 13 241
10 12  0 12 10 13 261 .
```

Рядки 2–13 перевіряють правильність обчислень для ламаної, що сполучає внутрішню точку основи на координатній площині $\{z = 0\}$ з внутрішньою точкою бічної грані. Для кожної бічної грані розглянуто 3 випадки: один, коли ламана не перетинає бічне ребро, і два, коли ламана перетинає бічне ребро. Останні 12 рядків перевіряють правильність обчислень для ламаної, що сполучає внутрішню точку основи на координатній площині $\{z = 0\}$ з внутрішньою точкою протилежної грані, що належить площині $\{z = a\}$. Розглянуто всі можливі варіанти перетину бічних ребер і ребер основи.

5. Вуличні перегони. За успішне виконання кожного тесту — по 10 балів, з них:

1 — за правильну відповідь про старт (1-ше число 1-го рядка RACE.RES);

1 — за правильну відповідь про фініш (2-ге число 1-го рядка RACE.RES);

1 — за правильну кількість пунктів, які неможливо уникнути (1-ше число 2-го рядка RACE.RES);

3 — за правильний перелік пунктів, які неможливо уникнути, причому за кожну помилку (не вказано якийсь пункт або вказано пункт, який можна уникнути) знімається по 1 балу з 3-х (див., починаючи з 2-го, числа 2-го рядка RACE.RES);

1 — за правильну кількість пунктів, які розбивають даний план (1-ше число 3-го рядка RACE.RES);

3 — за правильний перелік пунктів, які розбивають даний план, причому за кожну помилку (не вказано якийсь пункт або вказано пункт, який не розбиває план) знімається по 1 балу з 3-х (див., починаючи з 2-го, числа 3-го рядка RACE.RES).

5.1 Якщо файл RACE.DAT містить записи

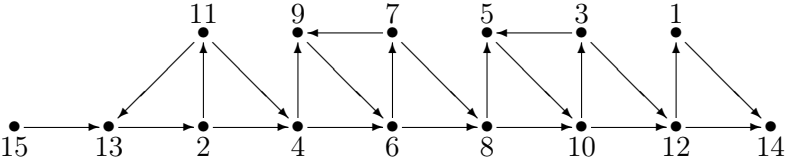
```
14
11  4
12  5
 9  6
10
 7  8
 8  9
 5 10
```

```
 6
 3 12
 4 13
 1 14
 2


13 ,
```

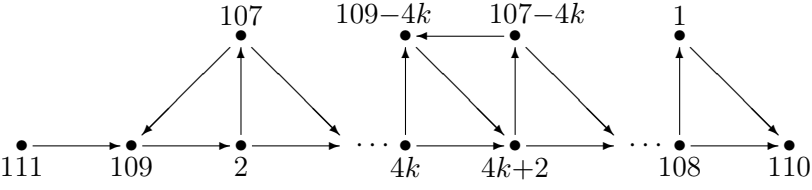
то файл RACE.RES має вигляд

```
 15  14
  7   2   4   6   8  10  12  13
  3   4   8  12   .
```

Таким чином, проводиться аналіз наступного плану.



5.2 Тест 2 перевіряє ефективність алєоритму за рахунок вилучення з подальшого розгляду тих пуктів, які разом із стартом (фінішом) знаходяться по різні боки від вже знайденого пункту, який не можливо уникнути на шляху від старту до фінішу. План тесту 2 утворено з плану тесту 1 багатократним повторенням деякої частини плану, і новою нумерацією пунктів. Графічно такий план можна подати наступним чином.



Тут крапками виділено структуру (вершини графа • та дуги, позначені стрілками) яка повторюється зліва направо для натурального k, яке зростає від 1 до 26 включно. Отже, нехай файл RACE.DAT містить 111 рядків і має вигляд

```
110
107          4
108          5
105          6
106
. . . . . .
```

$(111 - 4k)$ $4k$
$(112 - 4k)$ $(4k + 1)$

$(109 - 4k)$ $(4k + 2)$
$(110 - 4k)$

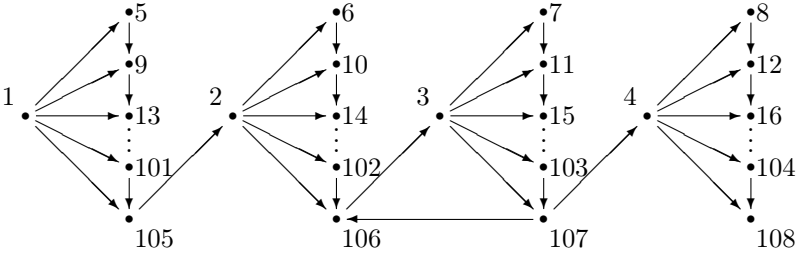
```
. . . . . .
3           108
4           109
1           110
2


109 ,
```

де крапками виділено 4 рядки, які послідовно зустрічаються у файлі RACE.DAT для натурального k, що зростає від 2 до 26 включно. Тоді файл RACE.RES має вигляд

```
111 110
 55   2..108 109
 27   4....108  ,
```

де через `2..108` позначено послідовність всіх парних чисел в межах від 2 до 108 включно, через `4....108` — послідовність всіх чисел в межах від 4 до 108 включно, які діляться на 4 без остачі.

5.3 Тест 3 перевіряє ефективність алєоритму: якщо пункт неможливо уникнути на шляху від старту до фінішу, то найменша кількість послідовних ланок від старту до фінішу дорівнює сумі мінімальних кількостей послідовних ланок від даного пункту до старту і до фінішу. План тесту утворено багатократним повторенням деякої структури, номери всіх пунктів якої мають одну й той самий лишок від ділення на 4, і проведенням додаткової стрілки від пункту 107 до пункту 106. Графічно план для цього тесту можна подати таким чином.



Отже, нехай файл RACE.DAT містить 108 рядків і має такий вигляд.

```
 5  9 13....101 105
 6 10 14....102 106
 7 11 15....103 107
 8 12 16....104 108
 9
...
```

```
108
2
3
4 106
```

Тут через `13....101`, `14....102`, `15....103` і `16....104` позначено записи послідовних членів арифметичних проєресій з різницею прогресії 4 і вказаними першим та останнім членами. Для j в межах від 5 до 104 включно j-ий рядок файлу RACE.DAT містить єдине число j+4. У поданні вхідного файлу відображено лише 5-ий та 104-ий рядок, а решту позначено трикрапкою. 108-ий рядок файлу RACE.DAT не містить жодного числа. Тоді файл RACE.RES має вигляд

```
1 108
6   2   3   4 105 106 107
3   2   4 105 .
```

6. Криптограма. За кожен варіант дешифрації будь-якого тесту — 2 бали. У разі повної дешифрації:

- за тести 1 і 2 присуджують по 2 бали;
- за тест 3 присуджують 6 балів;
- за тести 4, 5 і 6 присуджують по 4 бали.

Різними тестами перевіряється правильність дешифрації для різних основ числення. Обмеження на час дають можливість перевірити ефективність алгоритму: чи оптимізовано перебір відкиданням продовження тих варіантів дешифрації, які неузгоджені з вхідними даними й умовою задачі.

Тест	CRIPTO.DAT	CRIPTO.RES
6.1	5	
	сон+сон=ніч	204+204=413
6.2	6	
	one+two+two+two+two=nine	342+513+513+513+513=4042
6.3	7	
	дім+дім+дім+сад=двір	152+152+152+631=1450
		134+134+134+501=1236
		154+154+154+231=1056
6.4	8	
	drei+drei+drei=neun	2315+2315+2315=7147
		2465+2465+2465=7637
6.5	9	
	ТЕМП+ТРУД=УДАЧА	6508+6714=14323

6708+6514=14323

6.6 10

ЛіНіЯ+ЛіНіЯ=ФіЄУРА 74543+74543=149086

74345+74345=148690

3.5 Розв'язання

1. Банківська справа. Порівняно із задачею I.1 (про монети на шахівниці) розв'язання даної задачі повніше ілюструє *метод динамічного програмування*. Опишемо стисло ідею цього методу.

Розглянемо процес прийняття рішень q_j, який відбувається скінченну кількість разів j=1, 2 , ... , n, а прийняте рішення q_j разом із станом системи p_j однозначно визначають p_{j+1} — стан системи на момент прийняття наступного рішення. Поняття системи як множини станів та можливих рішень конкретизують при описі математичної моделі окремої задачі. Надалі ми обмежимося випадком, коли всі розглядувані величини — невід'ємні цілі числа. З процесом керування (прийняття рішень) пов'язана *функція зиску* (прибутку, виграшу — залежно від змісту задачі) $F(p_1, p_2, \ldots p_n; q_1, q_2, \ldots q_n)$, а мета процесу прийняття рішень $\{q_j\}_{j=1}^n$ — у досягненні найбільшого значення функції F. Якщо ж F описує втрати, то кажуть, що вона є *функцією втрат (ризику)*, а мета керування — зробити втрати найменшими. Не обмежуючи загальності міркувань, обмежимося надалі розглядом функції зиску, яка протилежна до функції витрат.

Переконаємося, що у цих досить загальних термінах описується запропонована задача. Означимо наступні невід'ємні цілі числа:

b_{jk} — кількість ліврів, яку Гуччо отримає від j-го банкіра після повернення з подорожі, вручивши йому k ліврів перед своїм від'їздом з Парижу;

$p_1 = m$ — загальна кількість ліврів Гуччо;

q_1 в межах від 0 до p_1 — кількість ліврів, які Гуччо вирішить віддати 1-му банкіру;

$p_2 = p_1 - q_1$ — кількість ліврів, яка залишиться для розподілу між 2-им, 3-ім, ... , n-им банкірами;

q_2 в межах від 0 до p_2 — кількість ліврів, які Гуччо вирішить віддати 2-му банкіру;

.....

$p_{k+1} = p_k - q_1$ — кількість ліврів, яка залишиться для розподілу між $(k+1)$-им, $(k+2)$-им, ... , n-им банкірами;

q_{k+1} в межах від 0 до p_{k+1} — кількість ліврів, які Гуччо вирішить віддати $(k+1)$-му банкіру;

.....

$p_n = p_{n-1} - q_{n-1}$ — кількість ліврів, яка залишиться для того, щоб віддати n-му банкіру;

q_n в межах від 0 до p_n — кількість ліврів, які Гуччо вирішить віддати n-му банкіру.

Тоді, повернувшись у Париж, Гуччо отримає від банкірів,

$$b_{1q_1} + b_{2q_2} + \cdots + b_{nq_n}$$

ліврів. Крім цього, у нього на руках буде $m - q_1 - q_2 - \cdots - q_n$ ліврів, які він не роздав банкірам. Разом це складе

$$F = (b_{1q_1} - q_1) + (b_{2q_2} - q_2) + \cdots + (b_{nq_n} - q_n) + m$$

ліврів.

Головна й визначальна ідея методу динамічного програмування: замість того, щоб розглядати ізольовано задачу пошуку оптимального рішення для заданих p_1 та n, таку задачу трактують як крок у послідовності пошуку максимумів. Цей метод швидкого знаходження оптимального рішення придатний для адитивної[9] функції зиску, яка для скінченного детермінованого процесу має вигляд

$$F(p_1, p_2, \ \ldots\ , p_n; q_1, q_2, \ \ldots\ , q_n) = g(p_1, q_1) + \cdots + g(p_n, q_n).$$

Доведення існування максимуму функції зиску — це окрема і досить складна задача. Ми надалі обмежимося розглядом випадку, коли множина всіх рішень q_j — скінченна (як у запропонованій задачі), і тому такий максимум обов'язково існує.

Означимо функцію найбільшого значення зиску на j-му кроці прийняття рішень для початкового стану p_1

$$\begin{aligned}
f_1(p_1) &= \max_{q_1} g(p_1, q_1), \\
f_j(p_1) &= \max_{q_1} \max_{q_2} \cdots \max_{q_j} (g(p_1, q_1) + g(p_2, q_2) + \cdots + g(p_j, q_j)) \\
&= \max_{q_1} \left(g(p_1, q_1) + \max_{q_2} \cdots \max_{q_j} g(p_2, q_2) + \cdots + g(p_j, q_j) \right) \\
&= \max_{q_1} \left(g(p_1, q_1) + f_{j-1}(p_2) \right).
\end{aligned}$$

Останню рівність можна прокоментувати таким чином. *Для довільного початкового стану та початкового рішення всі наступні рішення оптимального керування мають бути оптимальним керуванням для стану, отриманого в результаті першого рішення.* Отримане рекурентне співвідношення для функції

[9] Адитивна — пов'язана з додаванням. Від латинського additivus — придатковий.

f використано у поданій далі програмі. А вже за відомими значеннями функції f відтворюють оптимальне керування $\{q_j\}_{j=1}^{n}$, яке може бути не єдиним.

Описане перетворення задачі вибору точки у n-вимірному просторі всіх можливих рішень на послідовність n виборів точки в одновимірному просторі має вкрай важливе значення: метод динамічного програмування розглядає не всі можливі продовження, а лише ті, які відповідають оптимальним продовженням із нового стану. Таким чином, відкидають значну кількість варіантів керування, які не задовольняють вимоги оптимальності, і задача стає посильною для наявної у школах обчислюваної техніки навіть за жорстких обмежень на час роботи програми.

І ще одне зауваження стосовно програмно реалізованого алгоритму. Згідно умови задачі послідовність $\{b_{nk}\}_{k=0}^{m}$ монотонно зростає. Тобто для довільного натурального l, яке не перевищує $m-k$, справджується нерівність:

$$b_{n\ (k+l)} \geq b_{nk} + l.$$

Отже, за оптимального управління $q_n = p_n$, тобто всі m ліврів розподілено між банкірами. За цієї умови задача оптимального керування полягає в досягненні найбільшого значення функції

$$F(p_1, p_2, \ \dots\ , p_n; q_1, q_2, \ \dots\ , q_n) = b_{1q_1} + b_{2q_2} + \cdots + b_{nq_n}.$$

```
const n_max=15; m_max=99;
var   b,f: array[1..n_max,0..m_max] of word;
        q: array[1..n_max,0..m_max] of byte;
        x: array[1..n_max]          of byte;
        p: word; o:text;  i,j,k,l,m,n: byte;
begin{profit.pas}
assign(o,'PROFIT.DAT');  reset(o);  readln(o,m,n);
for i:=1 to n do b[i,0]:=0;
for j:=1 to m do                                        begin
for i:=1 to n do read(o,b[i,j]);  readln(o)                end;
close(o);
for j:=0 to m do begin f[1,j]:=b[1,j]; q[1,j]:=j end;
for i:=2 to n do for j:=0 to m do                       begin
p:=0;  l:=0;
for k:=0 to j do if (f[i-1,j-k]+b[i,k]) > p then        begin
p:=f[i-1,j-k]+b[i,k];              l:=k;                  end;
f[i,j]:=p;                 q[i,j]:=l;                     end;
p:=f[n,m];
for i:=n downto 1 do begin x[i]:=q[i,m]; m:=m-x[i] end;
assign(o,'PROFIT.RES');  rewrite(o);  writeln(o,p);
for i:=1 to n do write(o,x[i],' ');   writeln(o);  close(o);
end.
```

2. Криниця. Розв'язання даної задачі частково опирається на апарат аналітичної геометрії, поданий у математичній довідці до задачі II.1. Додатково нагадаємо ще й таке.

Відстанню від точки до фігури називають точну нижню грань відстаней від даної точки до точок даної фігури. Для даної ламаної це найменша з відстаней від даної точки до ланок ламаної.

Нехай в $\triangle ABC$ $BC = a$, $AC = b$, $AB = c$, d — відстань від вершини C до основи (відрізка, а не прямої) AB. Тоді:

- якщо $\angle A \geq 90^\circ$, що (згідно теореми косинусів) еквівалентно виконанню нерівності $a^2 \geq b^2 + c^2$, то $d = AC$;
- якщо $\angle B \geq 90^\circ$, що еквівалентно виконанню нерівності $b^2 \geq a^2 + c^2$, то $d = AB$;
- в усіх інших випадках, тобто, якщо справджуються обидві рівності $\angle A < 90^\circ$ і $\angle B < 90^\circ$ чи еквівалентні їм нерівності $a^2 < b^2 + c^2$, $b^2 < a^2 + c^2$, d дорівнює відстані від точки C до прямої AB.

Точка знаходиться на ламаній тоді і лише тоді, коли відстань між ними дорівнює 0.

Для того, щоб визначити взаємне розташування ламаної і точки поза нею, достатньо розглянути рух вектора $\vec{v}$, спрямованого з даної точки C до іншої точки, що рухається ланками ламаної, послідовно проходячи всі вершини ламаної, і повертається у початковий стан. Тоді вектор $\vec{v}$ здійснить оберт на кут $\pm 2\pi$ чи 0 залежно від того, де розташована точка C: всередині чи зовні ламаної відповідно.

Нехай вектори $\overrightarrow{CA}(x_1; y_1)$ і $\overrightarrow{CB}(x_2; y_2)$ утворюють з додатнім напрямом осі абсцис кути φ_1 і φ_2 відповідно, $\Delta\varphi = \varphi_2 - \varphi_1$. Тут, як і раніше, A і B — послідовні вершини ламаної, яка у даному випадку, не містить точку C. Тоді

$$\cos\varphi_1 = \frac{x_1}{|AC|}, \qquad \sin\varphi_1 = \frac{y_1}{|AC|}, \qquad |AC| = \sqrt{x_1^2 + y_1^2},$$

$$\cos\varphi_2 = \frac{x_2}{|BC|}, \qquad \sin\varphi_2 = \frac{y_2}{|BC|}, \qquad |BC| = \sqrt{x_2^2 + y_2^2}.$$

Згідно теорем додавання (тригонометричних функцій)

$$\cos\Delta\varphi = \cos\varphi_1\cos\varphi_2 + \sin\varphi_1\sin\varphi_2 = \frac{x_1x_2 + y_1y_2}{|AC| \cdot |BC|},$$

$$\sin\Delta\varphi = \cos\varphi_1\sin\varphi_2 - \sin\varphi_1\cos\varphi_2 = \frac{x_1y_2 - y_1x_2}{|AC| \cdot |BC|}.$$

За координатами векторів $\overrightarrow{CA}(x_1; y_1)$ і $\overrightarrow{CB}(x_2; y_2)$ легко визначити всі тригонометричні функції $\Delta\varphi$. Наприклад,

$$\operatorname{tg}\,\Delta\varphi = \frac{\sin\Delta\varphi}{\cos\Delta\varphi} = \frac{x_1y_2 - y_1x_2}{x_1x_2 + y_1y_2},$$

якщо тільки відповідний знаменник відмінний від 0. У поданій далі програмі використано низку умовних операторів, що використовують формули зведення та враховують властивості функції arctg x. Причина зовсім не в тому, що серед стандартних функцій мови Pascal наявна лише одна обернена тригонометрична функція — арктангенс. Реалізація функцій arcsin x і arccos x не врятує ситуацію, бо неможливо уникнути аналізу знаків $\cos\Delta\varphi$ та $\sin\Delta\varphi$.

Опишемо ще один спосіб визначення взаємного розташування ламаної і точки C поза нею. Вибирамо точку D, про яку напевно відомо, що вона лежить зовні ламаної. Наприклад, її координати на 1 більші від відповідно найбільшої абсциси і ординати вершини ламаної. Далі знаходимо кількість всіх точок перетину відрізка, який з'єднує точки C і D, з ланками ламаної під кутами, відмінними від 0. Зверніть увагу, точки перетину відрізків, а не прямих! Якщо знайдене число парне, то обидві точки C і D зовні ламаної, якщо непарне, то точка C розташована всередині ламаної.

Останній спосіб видається менш привабливим. Більше того, саме втілений у поданій далі програмі перший спосіб має безпосереднє продовження у теорії функцій комплексної змінної.

```
const n_max=33;          label FINISH;            var a,b,c,
i,k,n,x0,y0,x1,y1,x2,y2,d2,d20,den,den0,num,num0: integer;
o:text;   f,d,d0:real;    x,y: array[0..n_max] of integer;

{Функція, значення якої - найбільший спільний дільник
         двох її аргументів (невід'ємних цілих чисел)}
function gcd(i,j:integer):integer; var k,l,m:integer; begin
if i=0 then gcd:=abs(j)
       else if j=0 then gcd:=abs(i)
                   else begin k:=abs(i); l:=abs(j); m:=1;
while m>0 do begin m:=k mod l; k:=l;  l:=m     end;
gcd:=k                     end                 end;

begin{well.pas}                            {Зчитування даних}
f:=0;    assign(o,'WELL.DAT');    reset (o);  readln(o,n);
for i:=1 to n do read(o,x[i]);    readln(o);
for i:=1 to n do read(o,y[i]);    readln(o);
readln(o,x0,y0);    close (o);    x[0]:=x[n];  y[0]:=y[n];
for i:=1 to n do                                    begin
                   {Визначення кута між векторами,
спрямованими з точки C(x0;y0) до послідовних вершин
ламаної  A(x[i-1];y[i-1])  i  B(x[i];y[i])}
x1:=x[i-1]-x0; y1:=y[i-1]-y0; x2:=x[i]-x0; y2:=y[i]-y0;
b:=x1*y2-x2*y1;                 a:=x1*x2+y1*y2;
if (a=0) and (b=0) then begin d0:=0; goto FINISH end;
if a>0 then f:=f+arctan(b/a)
       else if a=0 then if b>0 then f:=f+pi/2
                               else f:=f-pi/2;
```

```
            if a<0 then                                  begin
if b>0 then             f:=f+pi/2+arctan(-a/b)
       else if b<0 then f:=f-pi/2-arctan( a/b)
                       else f:=f+pi                        end;

                     {Визначення відстані від C(x0;y0)
до відрізка з кінцями A(x[i];y[i]) і B(x[i-1];y[i-1])}
a:=x1*x1+y1*y1;  b:=x2*x2+y2*y2;
c:=(x[i]-x[i-1])*(x[i]-x[i-1])+(y[i]-y[i-1])*(y[i]-y[i-1]);
if a>=b+c then begin d2:=b; d:=sqrt(b); den:=0 end else
if b>=a+c then begin d2:=a; d:=sqrt(a); den:=0 end else

{Якщо шукана відстань - висота трикутника ABC...}     begin
a:=y[i]-y[i-1];  b:=x[i-1]-x[i];
c:=y[i-1]*(x[i]-x[i-1])-x[i-1]*(y[i]-y[i-1]);
k:=gcd(a,gcd(b,c));  a:=a div k;  b:=b div k;  c:=c div k;
num:=abs(a*x0+b*y0+c);  den:=trunc(sqrt(a*a+b*b));
                                        k:=gcd(num,den);
num:=num div k;          den:=den div k;  d:=num/den;    end;
if i=1  then begin num0:=num; den0:=den; d0:=d; d20:=d2 end
        else
if d<d0 then begin num0:=num; den0:=den; d0:=d; d20:=d2 end;
if d0=0 then goto FINISH                                 end;
FINISH:  assign(o,'WELL.RES');   rewrite (o);
if frac(d0)=0 then writeln(o,trunc(d0)) else
if     den0=0 then writeln(o,'sqrt(',d20,')')
              else writeln(o,num0,'/',den0);   f:=abs(f)/pi;
if d0=0  then writeln(o,'on' ) else
if f<0.5 then writeln(o,'out')
         else writeln(o,'in ');    close(o);    halt   end.
```

3. Лабіринт. Пошук найшвидшого шляху назовні лабіринту здійснюємо таким чином.

Зчитуємо значення m та n. Зчитуючи далі символи вхідного файлу, створюємо прямокутну таблицю чисел розміру $m \times n$, у якій елемент у i-му рядку j-го стовпчика дорівнює 0, 1 або 65535 (числу, що напевно перевищує mn), якщо j-ий символ (i+1)-го рядка вхідного файлу дорівнює ' ', '+' чи 'Ш' відповідно. Умова задачі не фіксує m та n, тому цю таблицю зберігаємо у пам'яті ЕОМ у вигляді лінійного масиву `a`, елемент якого `a[i*n+j]` відповідає елементу матриці в `(i+1)`-му рядку та `(j+1)`-му стовпчику.

Всі елементи таблиці, які є найближчими сусідами по вертикалі чи горизонталі числа 1 і дорівнюють 0, покладаємо рівними 2, потім всі елементи таблиці, які є найближчими сусідами по вертикалі чи горизонталі числа 2 і дорівнюють 0, покладаємо рівними 3, ... всі елементи таблиці, які є найближчими сусідами по вертикалі чи горизонталі числа k і дорівнюють 0, покладаємо рівними k+1, і т.д. Елемент заповненої таким чином таблиці — збільшена на 1 найменша

кількость кроків, яку потрібно зробити, щоб потрапити у відповідний квадрат лабіринту, стартуючи з початкового розташування. Для поданого в умові задачі прикладу така таблиця має наступний вигляд.

```
65535 65535 65535 65535 65535 65535 65535 65535 65535
65535    16    17    18    19    20    21    22    23
65535    15 65535 65535 65535 65535 65535 65535 65535
65535    14 65535     4     3     2     1     2 65535
65535    13 65535     5 65535     3 65535     3 65535
65535    12 65535     6 65535     4 65535     4 65535
65535    11 65535     7 65535     5 65535     5 65535
65535    10     9     8 65535     6     7     6 65535
65535 65535 65535 65535 65535 65535 65535 65535 65535
```

Зміну елементів таблиці таким чином припиняємо, якщо отримано елемент першого чи останнього рядка чи стовпчика, який відмінний від 0 та 65535. Таблицю неможливо змінити до такого вигляду, якщо сусіди останніх із змінених елементів відмінні від 0, а самі ці останні із змінених елементів розташовані поза першим і останнім рядком чи стовпчиком.

Пошук найкоротшого шляху здійснюємо "з кінця": починаючи з останнього елемента таблиці, якому поміняли величину, переходимо до найближчого по вертикалі чи горизонталі сусіднього елемента, величина якого на 1 менша від розглядуваного елемента.

Зробимо деякі пояснення щодо змісту змінних, які використано у поданій далі програмі.

`i+1` — номер рядка розглядуваного елемента таблиці.

`j+1` — номер стовпчика розглядуваного елемента таблиці.

`step` — натуральне число, яким замінюють величини деяких елементів прямокутної таблиці.

`n0` — кількість елементів прямокутної таблиці, змінених на попередньому кроці, тобто величина яких дорівнює `step-1`.

`k0[l]`, де `l` змінюється в межах від 1 до `n0` — номери елементів масиву `a`, які дорівнюють `step-1`. На етапі пошуку найкоротшого шляху `k0[l]` дорівнює 0, 1, 2 чи 3, якщо на `l`-му кроці потрібно рухатися на південь, північ, схід чи захід відповідно.

`n1` — кількість елементів прямокутної таблиці, змінених на розглядуваному кроці, тобто величина яких дорівнює `step`.

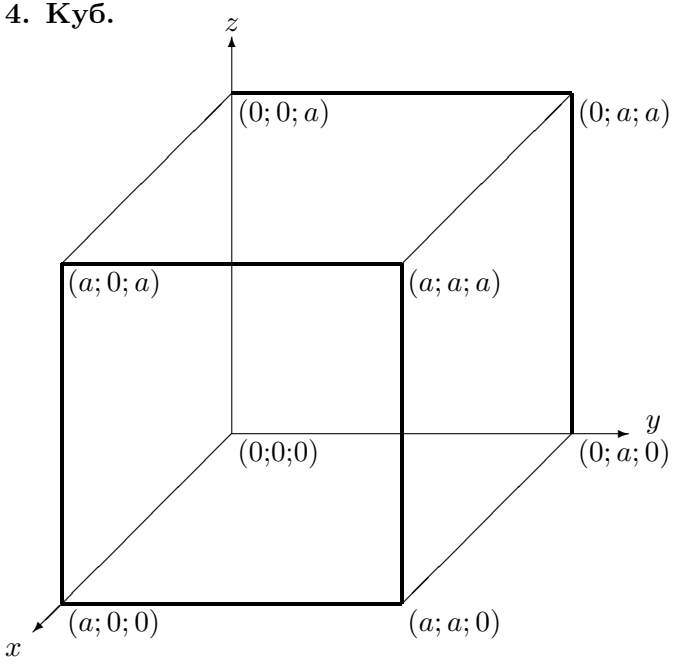
`k1[l]`, де `l` змінюється в межах від 1 до `n1` — номери елементів масиву `a`, які дорівнюють `step`.

```
const k_max=10000;  a_max=65535;          Label NEXT,FINISH;
var   o: text;                i,j,k,l,m,n,n0,n1,step:  word;
      s: char;              a,k0,k1: array[0..k_max] of word;
```

```
begin{labirint.pas}
     {Зчитування даних і запам‘ятовування малюнку масивом a}
assign(o,'LABIRINT.DAT');  reset(o);  readln(o,m,n);  k:=0;
for i:=0 to m-1 do                                    begin
for j:=0 to n-1 do                                    begin
read(o,s);  k:=i*n+j;
if s='Ш' then a[k]:=a_max else
if s=' ' then a[k]:=0     else
if s='+' then begin a[k]:=1; k0[1]:=k end;  k:=k+1      end;
readln(o)                                              end;
close(o);  assign(o,'LABIRINT.RES');  rewrite(o);  n0:=1;
step:=1;
NEXT:        {Перевірка, чи не досягнуто границі лабіринту}
                                  for i:=0 to m-1 do begin
k:=i*n;              if (a[k]>0) and (a[k]<a_max) then begin
             j:=0;  goto FINISH                     end end;
                                  for i:=0 to m-1 do begin
k:=(i+1)*n-1;        if (a[k]>0) and (a[k]<a_max) then begin
             j:=n-1; goto FINISH                    end end;
                                  for j:=0 to n-1 do begin
k:=j;                if (a[j]>0) and (a[j]<a_max) then begin
             i:=0;   goto FINISH                    end end;
                                  for j:=0 to n-1 do begin
k:=(m-1)*n+j;        if (a[k]>0) and (a[k]<a_max) then begin
             i:=m-1; goto FINISH                    end end;

          {Рекурентне визначення найменшої кількості кроків
                                 від початкового розташування}
       n1:=0;  step:=step+1;        for l:=1 to n0 do begin
i:=k0[l] div n;  j:=k0[l] mod n;
if i>0   then begin k:=(i-1)*n+j; if a[k]=0 then
               begin a[k]:=step; n1:=n1+1; k1[n1]:=k end end;
if i<m-1 then begin k:=(i+1)*n+j; if a[k]=0 then
               begin a[k]:=step; n1:=n1+1; k1[n1]:=k end end;
if j>0   then begin k:=i*n+j-1;   if a[k]=0 then
               begin a[k]:=step; n1:=n1+1; k1[n1]:=k end end;
if j<n-1 then begin k:=i*n+j+1;   if a[k]=0 then
               begin a[k]:=step; n1:=n1+1; k1[n1]:=k end end
                                                       end;
if n1=0 then begin writeln(o,'-1'); close(o); halt end;
n0:=n1;  for k:=1 to n0 do k0[k]:=k1[k];  goto NEXT;

                                          {Визначення шляху}
FINISH:  step:=step-1;  k:=step;  while k>0 do        begin
if i>0   then if a[(i-1)*n+j]=k then
               begin i:=i-1; k0[k]:=0; k:=k-1 end;
if i<m-1 then if a[(i+1)*n+j]=k then
```

```
              begin i:=i+1; k0[k]:=2; k:=k-1 end;
if j>0   then if a[i*n+j-1]  =k then
              begin j:=j-1; k0[k]:=1; k:=k-1 end;
if j<n-1 then if a[i*n+j+1]  =k then
              begin j:=j+1; k0[k]:=3; k:=k-1 end        end;

                                                {Запис відповіді}
writeln(o,step); for k:=1 to step do
if k0[k]=0 then write(o,'s')
           else if k0[k]=2 then write(o,'n')
                           else if k0[k]=1 then write(o,'e')
                                           else write(o,'w');
writeln(o);  close(o);  halt   end.
```

4. Куб.



Очевидно, що шукана величина — квадрат довжини найкоротшого відрізка, який з'єднує відповідні точки на розгортках куба і повністю належить квадратам розгортки. На поданих далі малюнках:

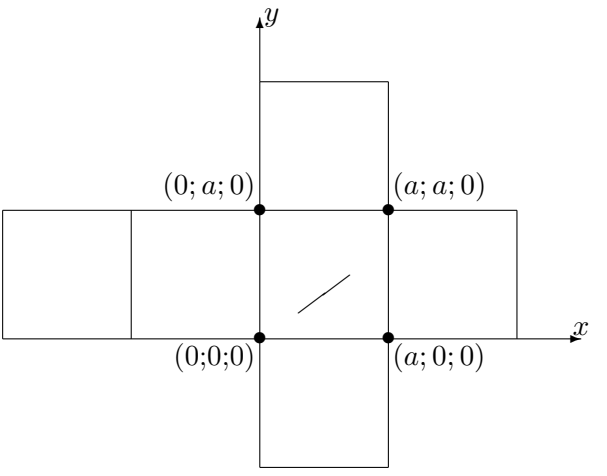
- відповідними розгортками куба на координатну площину $\{z{=}0\}$ проілюстровано можливі випадки розташування найкоротшої ламаної на поверхні куба, яка (ламана) на розгортці вироджується у відрізок, виділений на малюнках лінією більшої ширини;

- кругами • виділено вершини тих граней, які містять дві дані точки, і подано координати вершин цих граней.

∗ довжина проекцій найкоротшої ламаної на горизонтальні і вертикальні прямі дорівнюють відповідно абсолютним величинам виразів, сумою квадратів яких подано шукану величину.

Нехай перша точка, яка належить координатній площині xy, має координати $(x_0; y_0; 0)$. Можливі лише наступні варіанти значень координат другої точки на поверхні куба.

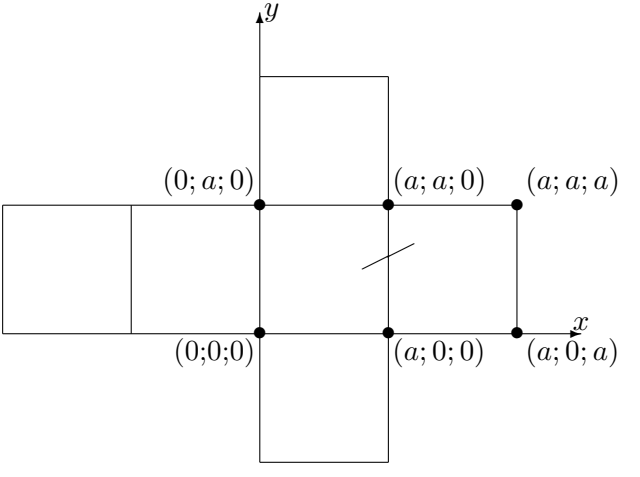
1. $(x; y; 0)$ — точка розташована у площині $\{z{=}0\}$.



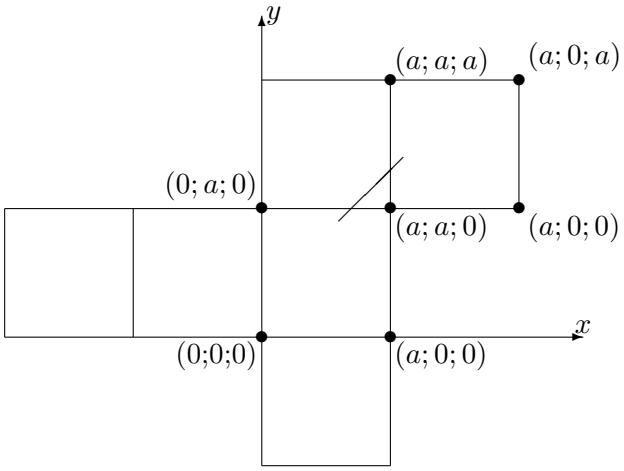
Тоді шукана величина дорівнює

$$(x - x_0)^2 + (y - y_0)^2.$$

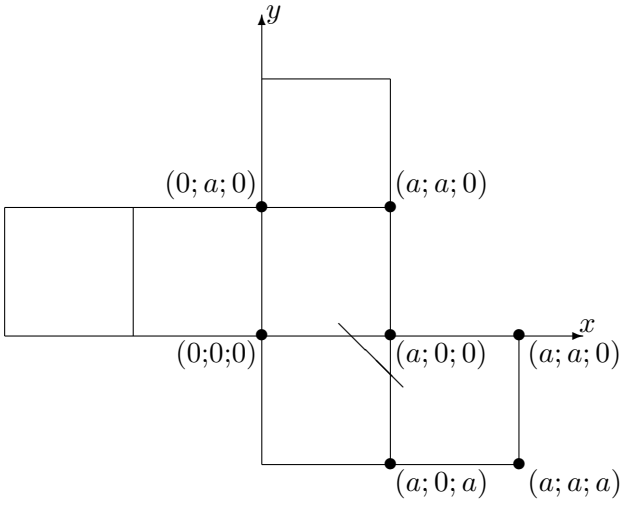
2. $(a; y; z)$ — точка розташована у площині $\{x{=}a\}$. Залежно від взаємного розташування двох даних точок $(x_0; y_0; 0)$ і $(a; y; z)$ шукана величина дорівнює:



або $(a+z-x_0)^2+(y-y_0)^2$;



або $(2a-y-x_0)^2+(a+z-y_0)^2$;



або $(a+y-x_0)^2+(z+y_0)^2$.

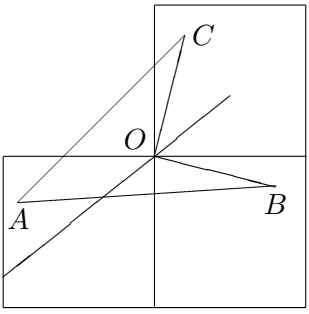
Для того, щоб бути впевненим у тому, що найменше з цих чисел є шуканою величиною, і відповідний відрізок повністю належить квадратам розгортки, потрібно довести наступне твердження[10].

Теорема. *Нехай на площині дано квадрат і всередині нього точку A. Два інших квадрати утворено обертанням*[11] *відповідно на* 90° *і* 180° *навколо*

[10] Зауважимо, що за теперішніх умов проведення олімпіади ніхто ні в якій формі не вимагає від учасників обгрунтування правильності алгоритму.

[11] Напрям обертання — за чи проти руху годинникової стрілки — впливає лише на те, яка з двох останніх розгорток розглядається. Поданий далі малюнок ілюструє випадок обертання

його вершини O. Всередині новоутворених квадратів вибрано точки B і C такими чином, що точка C — образ точки B після обертання на кут 90° навколо вершини O (див. наступний малюнок).



Якщо $AC<AB$, то відрізок AC не має спільних точок з квадратом, який містить точку B.

Доведення. Розглянемо пряму на площині, всі точки якої рівновіддалені від точок B і C. Оскільки $BO=CO$, то ця пряма містить точку O. Умова $AC<AB$ еквівалентна тому, що точки A і C лежать по один бік від цієї прямої. Саме у тій півплощині, яка не містить квадрат з точкою B.

3. $(x;a;z)$ — точка розташована у площині $\{y=a\}$. Обертанням навколо прямої $\{x=y=a/2\}$ на кут 90° *за напрямом руху годинникової стрілки, якщо дивитися на координатну площину xy з боку додатнього напряму осі аплікат z*, цей випадок зводиться до попереднього[12]. Координати даних точок при такому повороті зазнають таких змін:

$$(x_0;y_0;0)\to(y_0;a-x_0;0),\qquad (x;a;z)\to(a;a-x;z).$$

4. $(0;y;z)$ — точка розташована у площині $\{x=0\}$. Обертанням навколо прямої $\{x=y=a/2\}$ на кут $180^\circ=2\cdot90^\circ$ цей випадок зводиться до випадку 2. Можна також використати симетрію відносно площини $\{x=a/2\}$, яка перетворює координати двох даних точок наступним чином:

$$(x_0;y_0;0)\to(a-x_0;y_0;0),\qquad (0;y;z)\to(a;y;z).$$

У поданій далі програмі використано саме це перетворення.

проти напряму руху годинникової стрілки, що відповідає випадку, коли шукана відстань дорівнює $(2a-y-x_0)^2+(a+z-y_0)^2$.

[12]Звичайно, можна провести детальний детальний розгляд цього випадку за тим же сценарієм (з розглядом 3-х малюнків і виведенням 3-х відповідних їм формул), що й для попереднього випадку. Але це буде непродуктивною витратою часу. Особливо, якщо врахувати, як часто перетворення симетрії куба буде використано у подальшому розв'язанні цієї задачі, що дозволить суттюво спростити математичну модель і алгоритм розв'язання.

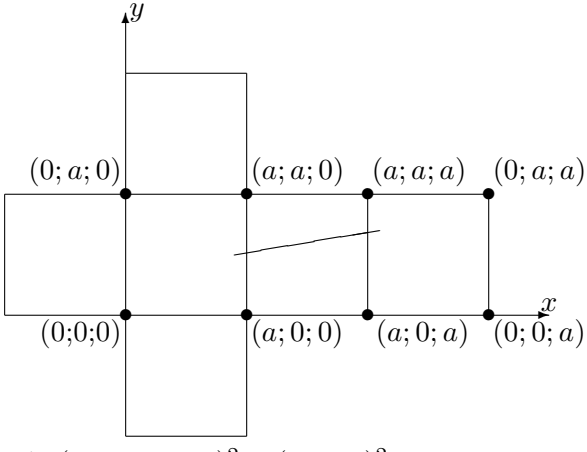
5. $(x;0;z)$ — точка розташована у площині $\{y{=}0\}$. Обертанням навколо прямої $\{x{=}y{=}a/2\}$ на кут $270°{=}3{\cdot}90°$ *за напрямом руху годинникової стрілки, якщо дивитися на координатну площину xy з боку додатнього напряму осі аплікат z*, цей випадок зводиться до випадку 2. Такий оберт еквівалентний обертанню навколо тієї самої прямої $\{x{=}y{=}a/2\}$ на кут $90°$ *проти руху годинникової стрілки, якщо дивитися на координатну площину xy з боку додатнього напряму осі аплікат z.* Координати двох даних точок при цьому зазнають наступних змін:

$$(x_0;y_0;0)\to(a-y_0;x_0;0),\qquad (x;0;z)\to(a;x;z).$$

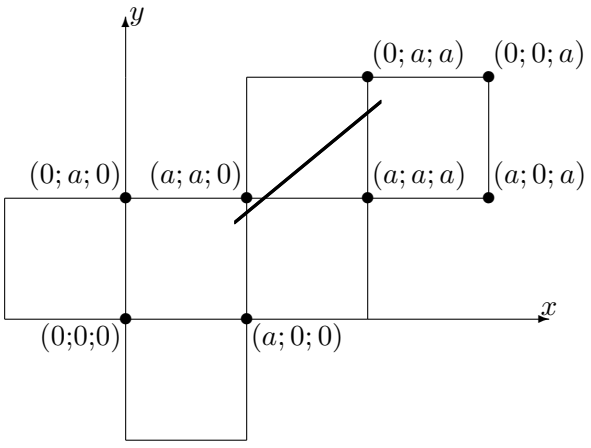
6. $(x;y;a)$ — точка розташована у площині $\{z{=}a\}$. Випадок, коли найкоротша ламана перетинає одне з ребер куба, що з'єднує вершини:

   - $(a;a;0)$ і $(0;a;0)$;
   - $(0;a;0)$ і $(0;0;0)$;
   - $(0;0;0)$ і $(a;0;0)$,

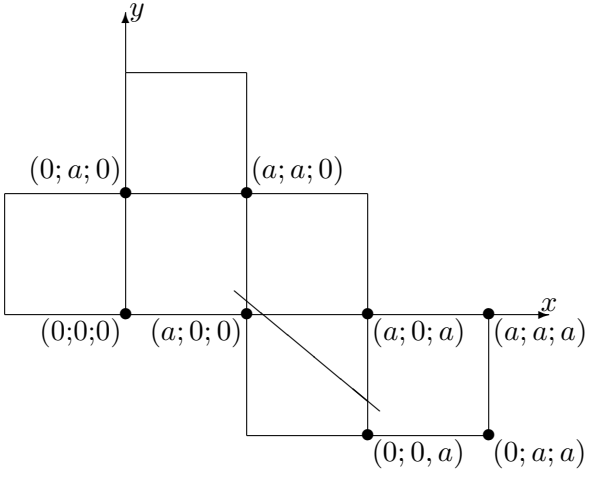
   перетворенням симетрії куба (див. пункти 3–5) зводиться до випадку перетину ламаною ребра куба, що сполучає вершини $(a;0;0)$ і $(a;a;0)$. Тому достатньо детально розглянути лише останній випадок, коли шукана величина може дорівнювати:



або $(3a-x-x_0)^2+(y-y_0)^2$;



або $(3a - y - x_0)^2 + (2a - x - y_0)^2$;



або $(2a + y - x_0)^2 + (a - x + y_0)^2$.

Умову задачі можна узагальнити: "... Яка довжина найкоротшої ламаної, що з'єднує дві точки на поверхні куба з ребром довжини a і які ребра вона перетинає?.."

Узагальнення на випадок інших многогранників (необов'язково правильних і, навіть, опуклих) вимагає детального аналізу моделі багатогранника. Шкільною програмою такий аналіз не передбачено проводити навіть для правильних многогранників. Тому дане узагальнення недоречне на олімпіаді, але можливе в учнівській дослідницькій роботі.

```
var a,x,y,z,x0,y0,z0,x1,y1,j,n,nd: integer;   o,i: text;
                d: array[1..12] of integer;

   {Процедура обчислення квадратів довжин ламаних, які:
 *  на відповідних розгортках вироджуються у відрізки;
```

```
 *  з‘їднують точки на нижній та бічній гранях
    (z=0  або  x=a  відповідно).}
procedure side;                                          begin
d[1]:=  (a+z-x0)*(a+z-x0)  +  (y-y0)*(y-y0);
d[2]:=(2*a-y-x0)*(2*a-y-x0)+(a+z-y0)*(a+z-y0);
d[3]:=  (a+y-x0)*(a+y-x0)  +  (z+y0)*(z+y0);   nd:=3    end;

   {Процедура обчислення квадратів довжин ламаних, які:
 *  на відповідних розгортках вироджуються у відрізки;
 *  з‘їднують точки на нижній та верхній гранях
    (z=0  або  z=a  відповідно);
 *  перетинають ребро куба [(a;0;0);(a;a;0)].}
procedure top;                                           begin
d[nd+1]:=(3*a-x-x0)*(3*a-x-x0)+    (y-y0)*(y-y0);
d[nd+2]:=(3*a-y-x0)*(3*a-y-x0)+(2*a-x-y0)*(2*a-x-y0);
d[nd+3]:=(2*a+y-x0)*(2*a+y-x0)+  (a-x+y0)*(a-x+y0);
nd:=nd+3                                                   end;

begin{cube.pas}
assign(i,'CUBE.DAT');    reset(i);   readln(i,a,n);
assign(o,'CUBE.RES');  rewrite(o);  writeln(o,a:3,n:3);
for j:=1 to n do                                         begin
readln(i,x0,y0,z0,x,y,z);  nd:=0;
write(o,x0:3,y0:3,z0:3,x:3,y:3,z:3);
                {Обчислення квадратів довжин ламаних,
які на відповідних розгортках вироджуються у відрізки,
серед яких (квадратів довжин)  треба вибрати найменшу}
     if z=0 then                                         begin
               d[1]:=(x-x0)*(x-x0)+(y-y0)*(y-y0); nd:=1 end
else if x=a then side
else if y=a then                                         begin
            x1:=x0; y1:=y0; x0:=y1; y0:=a-x1;
            x1:=x;  y1:=y;   x:=y1;  y:=a-x1;  side     end
else if x=0 then                                         begin
                          x0:=a-x0;  x:=a-x;   side     end
else if y=0 then                                         begin
            x1:=x0; y1:=y0; x0:=a-y1; y0:=x1;
            x1:=x;  y1:=y;   x:=a-y1;  y:=x1;  side     end
else                                                     begin
                                                    top;
            x1:=x0; y1:=y0; x0:=y1;   y0:=a-x1;
            x1:=x;  y1:=y;   x:=y1;    y:=a-x1; top;
            x1:=x0; y1:=y0; x0:=y1;   y0:=a-x1;
            x1:=x;  y1:=y;   x:=y1;    y:=a-x1; top;
            x1:=x0; y1:=y0; x0:=y1;   y0:=a-x1;
            x1:=x;  y1:=y;   x:=y1;    y:=a-x1; top     end;
```

```
                          {Вибір найменшої довжини ламаної}
for x:=2 to nd do if d[x]<d[1] then d[1]:=d[x];
writeln(o,d[1]:4)                                          end;
close(i);  close(o);   halt                                end.
```

5. Вуличні перегони. Для того, щоб зрозуміти ідею розв'язання задачі, бажано знати елементи теорії графів хоча б на рівні означень. Бажано, але необов'язково. Наочність математичної моделі даної задачі дозволяє ілюструвати виклад початків теорії графів розв'язанням даної задачі.

Зауваження 1. *Пункт i_{start} — старт, якщо число i_{start} не зустрічається у жодному рядку вхідного файлу RACE.DAT (для визначення i_{start} на початку поданої далі програми використано масив булевих змінних* `ll`*).*

Зауваження 2. *Пункт i_{finish} — фініш, якщо i_{finish}-ий рядок вхідного файлу RACE.DAT містить лише порожній символ.*

Сформулюємо твердження, яке лежить в основі рекурентного[13] визначення значення функцій, запроваджених далі для розв'язання задачі.

Зауваження 3. *Множина всіх пунктів, які можна досягнути з певного пункту i, подолавши $(m{+}1)$ ділянок вулиць між окремими пунктами — це множина всіх тих пунктів, які можна досягнути, подолавши 1 ділянку з тих пунктів, для досягнення яких з i-го пункту потрібно подолати m ділянок.*

Запровадимо наступні функції[14] x, y, z невід'ємних цілих чисел, всі значення яких — невід'ємні цілі числа.

$y(i,j)$ — кількість пунктів, відмінних від i-го, які можна досягнути, подолавши не більше, ніж j ділянок, розпочавши свій шлях з i-го пункту. Згідно означення $y(i,0){=}0$, і для довільного j в межах від 1 до n включно

$$y(j,0) \le y(j,1) \le y(j,2) \le \cdots \le y(j,n-1) = y(j,n) = \cdots \ .$$

Щодо нестрогих нерівностей ліворуч, то згідно зробленого зауваження 3, якщо нестрога нерівність перетворюється у рівність[15], то у рівність перетворюються і всі нерівності, які знаходяться праворуч від цієї рівності. Рівності праворуч — наслідок очевидного факту: якщо у ламаній кількість *різних* вершин менша

[13]Кажуть, що члени послідовності a_n задано рекурентно, якщо декілька перших членів цієї послідовності задано явно, а кожний наступний член визначається як значення (відомої) функції від попередніх членів.

[14]Подана програма передбачає використання значень запроваджених функцій як елементів масивів з тими самими назвами.

[15]Тобто, збільшення з m до $(m{+}1)$ кількості ланок шляху, який виходить з i-го пункту, не призводить до появи нових пунктів, які можна досягнути з i-го пункту.

від кількості ланок, то з цієї ламаної можна вилучити ланки, які утворюють замкнені ламані (цикли) таким чином, щоб:

- ті ланки, що залишилися, утворювали б ламану, перша й остання вершини якої ті самі, що й у початкової ламаної;
- кількість вершин утвореної ламаної на 1 більша від кількості її ланок.

У поданій далі програмі величина `b[i]` дорівнює чисел в i-му рядку файла RACE.DAT, тобто $y(i,1)$ — кількості стрілок, які виходять з i-го пункту на запропонованому плані перегонів.

$z(i,j)$ — найменша кількість ланок шляху, який сполучає i-ий та j-ий пункти. Очевидно, що для довільного i в межах від 1 до n включно:

- $z(i,i)$=0;
- i-ий рядок файла RACE.DAT перелік всіх тих j, для яких $z(i,j)$=1 для запропонованого плану перегонів.

$\{x(i,j)\}_{j=y(i,(k-1))+1}^{y(i,k)}$ — вичерпний перелік без повторення номерів тих пунктів, які можна досягнути з i-го пункту, подолавши k ділянок, і яких неможливо досягнути, здолавши меншу кількість ділянок шляху. Якщо функцію x побудовано для запропонованого плану перегонів (без будь-якого вилучення ділянок шляху чи пунктів), то для довільного i в межах від 1 до n включно i-ий рядок вхідного файлу RACE.DAT містить послідовність $\{x(i,j)\}$, де j змінюється у межах від $y(i,0)$+1=1 до $y(i,1)$ включно. У поданій далі програмі для зберігання саме цих даних використано масив `a`.

Наступне твердження — обгрунтування програмно втіленого алгоритму знаходження шуканих пунктів.

Зауваження 4. *Деякий пункт i_{no} поданого плану неможливо уникнути на шляху від старту до фінішу тоді й лише тоді, коли фініш стає недосяжним із старту за умови вилучення з плану всіх тих стрілок, які виходять з даного пункту i_{no} (і лише їх).*

Для того, щоб пункт i_{no} розбивав початковий план на два, додатково до вказаної умови необхідна ще й недосяжність всіх пунктів, що розташовані на шляху між пунктами i_{start} та i_{no}, з довільного пункту, розташованого на шляху між пунктами i_{no} та i_{finish}.

Зауваження 5. *Серед пунктів з однаковою найменшою кількістю ланок шляху до старту є не більше одного (пункту), який неможливо уникнути на шляху від старту до фінішу.*

Справджується й дуальне[16] твердження.

Зауваження 5′. *Серед пунктів з однаковою найменшою кількістю ланок шляху до фінішу є не більше одного (пункту), який неможливо уникнути на шляху від старту до фінішу.*

Використовуючи наступне твердження (необхідну умову), можна істотно зменшити кількість обчислень у процесі знаходження пунктів, які неможливо уникнути.

Зауваження 6. *Якщо пункт i_{no} поданого плану неможливо уникнути на шляху від пункту i_{start} до пункту i_{finish}, то*

$$z(i_{\text{start}}, i_{\text{no}}) + z(i_{\text{no}}, i_{\text{finish}}) = z(i_{\text{start}}, i_{\text{finish}}). \tag{1}$$

Таким чином, для розв'язання задачі достатньо, зменшуючи кількість ділянок шляху від старту до розглядуваного пункту i_{no} в межах від $z(i_{\text{start}}, i_{\text{finish}})-1$ до 1 включно, у разі виконання рівності (1), встановити істинність першої чи одночасно першої і другої умови[17] зауваження 4. У разі істиності переходимо до розгляду плану, всі пункти якого лежать на шляху від старту до тільки що знайденого пункту i_{no}. Саме таке "відкидання зайвої частини плану" із запровадженням нової нумерації пунктів[18] перевіряється останнім тестом.

Перевірка можливої оптимізації пошуку шуканих пунктів за допомогою "поділу навпіл кількості ланок" для планів з великою кількістю шуканих пунктів не передбачено.

```
const nmax=111;                    label NEXTj, NEXTk, EODATA;
var   x,y: array[1..nmax,0..nmax] of byte;
   a,z,z0: array[1..nmax,1..nmax] of byte;
b,new,old: array[1..nmax]          of byte;          o: text;
     l,ll: array[1..nmax]          of boolean;    lz:boolean;
i,j,k,ii,jj,kk,n,n0,start,finish,finish0,ino,nout,ndiv:byte;

{Процедура заповнення масивів x, y, z даними про вилучення з
розгляду всіх шляхів, які виходять з пункту ino  і лише їх}
procedure XYZ (ino:byte);                                  begin
for i:=1 to n do for j:=1 to n do z[i,j]:=0;

{Початок заповнення - заповнення даними масивів a та b}
for i:=1 to n do  if ((i<>ino) and (i<>finish)) then  begin
y[i,1]:=b[old[i]];           for j:=1 to b[old[i]] do begin
```

[16] Від латинського dualis — двоїстий. Тут у розумінні симетричне: поміняйте одночасно напрям всіх стрілок на плані.

[17] У поданій програмі відповідно булеві змінні `l[ino]` та `ll[ino]`.

[18] Див. використання у програмі масивів `new` та `old`

```
x[i,j]:=new[a[old[i],j]];  z[i,new[a[old[i],j]]]:=1 end end
                           else y[i,1]:=0;

{Визначення найменшої кількості ділянок  z[i,ii],  які
треба подолати,  щоб дістатися з пункту i до пункту ii.
y[i,j]  -  кількість пунктів,  у які можна потрапити з
i-го, якщо подолати не більше, ніж j ділянок. y[i,0]=1.}
for j:=2 to n do                                          begin
for i:=1 to n do                                          begin
y[i,j]:=y[i,j-1];      if y[i,j-2]   <  y[i,j-1] then
                    for k:=y[i,j-2]+1 to y[i,j-1]   do begin
jj:=x[i,k]; if y[jj,1]>0 then for kk:=1 to y[jj,1] do begin
ii:=x[jj,kk];       if ((z[i,ii]=0) and (ii<>i)) then begin
y[i,j]:=y[i,j]+1;  x[i,y[i,j]]:=ii;  z[i,ii]:=j end end end
                                                    end end end;
begin{race.pas}{$I-}
for i:=1 to nmax do l[i]:=true;
                                              {Зчитування даних}
        j:=0;                assign(o,'RACE.DAT');  reset(o);
NEXTj:  j:=j+1;  k:=0;
NEXTk:                  if SeekEoln(o) then               begin
readln(o);  b[j]:=k;  if  not Eof(o) then goto NEXTj
                                     else goto EODATA   end
                                     else k:=k+1;
        read(o,a[j,k]); l[a[j,k]]:=false; goto NEXTk;
EODATA: close(o); n:=j; n0:=n;

      {Визначення старту й фінішу, початок запису відповіді}
for i:=1 to n do if b[i]=0 then finish:=i
                           else if l[i] then start:=i;
assign(o,'RACE.RES');          rewrite(o);
writeln(o,start:3,finish:4);    close(o);   append(o);

{Визначення найменшої кількості ділянок, які треба подолати,
   щоб дістатися від довільного пункту до будь-якого іншого
                           пункту згідно початкового плану.}
for i:=1 to n do                                          begin
y[i,0]:=0;  new[i]:=i;   l[i]:=false;
x[i,0]:=i;  old[i]:=i;  ll[i]:=false                        end;
XYZ(n+1); for i:=1 to n do for j:=1 to n do z0[i,j]:=z[i,j];

     {Впорядкування пунктів за кількістю ділянок від старту}
k:=0;  for i:=0 to n-1 do
       for j:=1 to n   do   if  z[start,j]=i  then    begin
k:=k+1;                     new[j]:=k;   old[k]:=j      end;
finish0:=finish;
                {Визначення пунктів, які неможливо уникнути,
```

```
                        а серед них тих, які розбивають план}
nout:=0;  ndiv:=0;  ino:=new[finish];  finish:=new[finish];
while ino>1 do                                              begin
while z0[start,old[ino]]=z0[start,old[finish]] do ino:=ino-1;
if ino=1 then break;
if z0[start,old[ino]]+z0[old[ino],finish0]=z0[start,finish0]
                                      then                  begin
XYZ(ino);           if z[1,finish]=0 then                   begin
nout:=nout+1;  ll[old[ino]]:=true;  l[old[ino]]:=true;

                  {Запровадження нової нумерації пуктів}
finish:=ino;  k:=ino;  j:=n+1;  for i:=ino+1 to n do
if z[1,i]>0 then begin k:=k+1; x[k,1]:=old[i] end
            else begin j:=j-1; x[j,1]:=old[i] end;
for i:=ino+1 to n do
begin old[i]:=x[i,1]; new[old[i]]:=i end;               n:=k;

                  {Визначення можливості поділу плану}
i:=k;    while ll[old[ino]] and (i<n0) do                   begin
i:=i+1;  ll[old[ino]]:=z0[old[i],old[ino]]=0                  end;
i:=k;    while ll[old[ino]] and (i>1)  do                   begin
i:=i-1;  ll[old[ino]]:=z0[old[ino],old[i]]=0                  end;
if ll[old[ino]] then ndiv:=ndiv+1;                            end

                                      else ino:=ino-1     end
                                      else ino:=ino-1     end;
              {Дописування відповіді}
write(o,nout:3);
for i:=1 to n0 do if  l[i] then write(o,i:4);    writeln(o);
write(o,ndiv:3);
for i:=1 to n0 do if ll[i] then write(o,i:4);    writeln(o);
close (o);  halt                                         end.
```

6. Криптограма. Запропонована задача навіяна відомими задачами для розвитку алгоритмічного мислення — математичних ребусів (див., наприклад, Л.М. Лоповок, "Збірник математичних задач логічного характеру", Київ, "Радянська школа", 1972, 151 с., звідки й взято тести для перевірки). Для її успішного розв'язання необхідно звернути увагу на таке.

- *Опрацювання вхідного файлу:* кодування умови задачі елементами масиву `a` та змінною `na`.
  `na[i]` — кількість цифр `i`-го числа криптограми.
  `a[i,j]:=k`, якщо `j`-ій цифрі `i`-го числа криптограми відповідає `k`-ий символ з набору символів, зчитаних з 2-го рядка вхідного файлу, відмінних від знаків додавання чи рівності і пронумерованих (без повторення) у тому порядку, як вони (символи) зчитувалися.

`ncmax` — основа числення;

`nnum` — кількість всіх чисел криптограми;

`namax` — найбільша кількість цифр доданків чи суми;

`nc` — кількість різних символів другого рядка вхідного файлу, відмінних від знаків додавання чи рівності.

- *Запровадження нової нумерації* за допомогою масивів `new` та `old`: спочатку ті, які зустрічаються у 1-му розряді, потім — ті, що зустрічаються у 2-му розряді і т.д.

- Можливість розгляду всіх взаємно однозначних відповідностей цифр розглядуваної системи числення символам криптограми, які відмінні від знаків додавання чи рівності. Іншими словами, маємо втілити *перебір розташувань без повторення* з `ncmax` по `nc`. Використано такі масиви:

  `dd[i]` — цифра, яка відповідає i-му символу;
  `kk[1]:=0,`
  `kk[i]:=k[i-1]+ncmax-i+2` — допоміжний масив;
  множина значень `d[kk[i]+jj[i]]`, де `jj[i]` змінюється в межах від 1 до `ncmax-i+1` включно — набір цифр, з якого здійснюється вибір тієї, яка відповідає `i`-му символу.

- *Оптимізація перебору відкиданням продовження тих варіантів дешифрації, які неузгоджені з вхідними даними і умовою задачі.* Якщо визначення нового `i`-го символа не призводить до збільшення кількості розрядів, у яких можна зробити перевірку на узгодження (порівняти цифру суми з сумою цифр доданків і кількості одиниць, що переноситься з попереднього розряду суми), то `ii[i]:=0`. Інакше

  `ii[i]` — номер розряду, рахуючи справа, в якому можна перевірити відповідність умові, як тільки но визначено перші `i` символів (згідно з новою нумерацією).

  Використано також масиви:

  `b[i]` — кількість одиниць, які переносяться в наступний розряд в результаті додавання цифр суми `i`-го розряду;

  `l0[i]` — набуває значення `true`, якщо `i`-ий символ відповідає першій цифрі одного з доданків чи суми. Нагадаємо, що така цифра відмінна від 0.

На етапі запровадження нової нумерації символів `l0[i]=true` лише доти, доки не визначено новий номер `i`-го символа з набору символів 2-го рядка вхідного файлу, відмінних від знаків додавання чи рівності і пронумерованих у тому порядку, як їх (символи) зчитували.

```
const nmax=9; nnux=6;  label CLOSED,NEXT,NEXTi,NEXTj,BREAKj;
var         a: array[1..nmax,1..nnux+1] of byte;    o: text;
                   na: array[1..nnux+1] of byte;    s: char;
dd,ii,jj,kk,new,old,b: array[0..nmax+1] of byte;
                   l0: array[1..nmax+1] of boolean;
                    c: array[1..nmax+1] of char;
   d: array[1..(nmax+1)*(nmax+2) div 2] of byte;
          i,j,k,l,nc,namax,nnum,ncmax,sum: byte;
begin{cripto.pas}{$I-}
for i:=1 to nmax do c[i]:=' ';for i:=1 to nnux do na[i]:=0;
{Зчитування даних, кодування масивом  a  умови задачі}
assign(o,'CRIPTO.DAT');  reset(o);  readln(o,ncmax);
      read(o,c[1]); a[1,1]:=1; i:=1; j:=1; nc:=1; namax:=1;
NEXT: read(o,s);           if (s='+') or (s='=') then begin
na[i]:=j;  i:=i+1;  if j>namax then namax:=j;  j:=0;
goto NEXT                                                 end
                                                    else begin
j:=j+1;  k:=1; while (s<>c[k]) and (k<nc+1) do k:=k+1;
if s<>c[k] then begin nc:=nc+1;c[nc]:=s;a[i,j]:=nc end
           else a[i,j]:=k;
if SeekEoln(o) then                                     begin
na[i]:=j;nnum:=i; if j>namax then namax:=j; goto CLOSED end
              else goto NEXT                             end;
CLOSED: close(o);  assign(o,'CRIPTO.RES');  rewrite(o);

                                    {Нова нумерація символів:
спочатку ті, що вперше зустрічаються у 1-му розряді, потім
- ті, що вперше зустрічаються у 2-му, потім - у 3-му і т.д.}
for i:=1 to nc do begin ii[i]:=0; l0[i]:=true end;     k:=0;
for i:=1 to namax do                                    begin
for j:=1 to nnum  do if na[j]>=i then                   begin
                         if l0[a[j,na[j]-i+1]] then   begin
k:=k+1;                 old[k]:=a[j,na[j]-i+1];
l0[old[k]]:=false;  new[old[k]]:=k                  end end;
ii[k]:=i; if k=nc then break                            end;

       {Визначення символів, з яких починаїться запис чисел}
for i:=1 to nc   do l0[i]:=false;
for i:=1 to nnum do l0[new[a[i,1]]]:=true;

                           {Перебір всіх можливих варіантів}
for i:=1 to ncmax+1 do d[i]:=i-1;                  kk[1]:=0;
for i:=2 to nc      do kk[i]:=kk[i-1]+ncmax-i+2;
       i:=0;
NEXTi: i:=i+1; jj[i]:=0;
NEXTj:         jj[i]:=jj[i]+1;
if jj[1]=ncmax  +1 then halt else
```

```
if jj[i]=ncmax-i+2 then begin i:=i-1; goto NEXTj end;
dd[i]:=d[kk[i]+jj[i]];
if (10[i] and (dd[i]=0)) then goto NEXTj
                         else if ii[i]=0 then        begin
for j:=1       to jj[i] -1 do d[kk[i+1]+j]  :=d[kk[i]+j];
for j:=1+jj[i] to nmax-i+2 do d[kk[i+1]+j-1]:=d[kk[i]+j];
goto NEXTi                                             end
                                        else        begin
if ii[i]=1 then sum:=0 else sum:=b[ii[i]-1];
for j:=1 to nnum-1 do if na[j]>=ii[i] then
sum:=sum+dd[new[a[j,na[j]-ii[i]+1]]];
b[ii[i]]:=sum div ncmax;
if dd[new[a[nnum,na[nnum]-ii[i]+1]]] <> (sum mod ncmax)
   then goto NEXTj
   else if i<nc then                                    begin
for j:=1       to jj[i] -1 do d[kk[i+1]+j]  :=d[kk[i]+j];
for j:=1+jj[i] to nmax-i+2 do d[kk[i+1]+j-1]:=d[kk[i]+j];
goto NEXTi                                              end
                else                                    begin

{Якщо перед перевіркою відповідності умові задачі цифр
найстаршого розряду спосіб дешифрації вже вибрано...}
if ii[nc]<namax then                                    begin
for k:=ii[nc]+1 to namax do                             begin
sum:=b[k-1];
for j:=1 to nnum-1 do if na[j]>=k then
sum:=sum+dd[new[a[j,na[j]-k+1]]];
b[k]:=sum div ncmax;
if dd[new[a[nnum,na[nnum]-k+1]]] <> (sum mod ncmax)
then goto NEXTj                                       end end;

     {Перевірка узгодженності у найстаршому розряді,
                        дописування відповіді}
if b[namax]=0 then                                      begin
append(o);
for k:=1 to nnum do                                     begin
for l:=1 to na[k] do write(o,dd[new[a[k,l]]]);
if k=nnum then writeln(o)
          else if k=nnum-1 then write(o,'=')
                           else write(o,'+')              end;
close (o)                                               end;
goto NEXTj                                        end end end.
```

На завершення подамо приклад розв'язання задачі на дешифрацію конкретного запису дії додавання, для того, щоб проілюструвати різницю між використаним алгоритмом для цілого класу задач і традиційним для математиків способом розв'язання і запису розв'язання окремої задачі. В останньому випадку частину варіантів перебору можна відкинути, розв'язавши певні рівності чи нерівності. Як і які саме — істотно

залежить від умови задачі. При цьому деякі з нерівностей навіть не записують, користуючись очевидністю висновків. Зауважимо також, що дешифрацію запису інколи зручніше починати з цифр старших розрядів.

Задача. Дешифрувати запис у десятковій системі числення дії додавання

$$ten + ten + forty = sixty.$$

Розв'язання. Будь-яка цифра за величиною не перевищує 9, а її добуток на 2 не перевищує 18, що в свою чергу менше, ніж 20. З аналізу цифр розряду одиниць маємо

$$\left[\begin{array}{l} n+n+y=y \\ n+n+y=y+10 \end{array}\right. \Rightarrow \left[\begin{array}{l} n=0 \\ n=5 \end{array}\right. ,$$

причому, якщо $n = 5$, то з розряду одиниць один десяток перейде у розряд десятків. З аналізу цифр розряду десятків маємо

$$\left[\begin{array}{l} e+e+t=t \\ e+e+t=t+10 \\ e+e+t+1=t \\ e+e+t+1=t+10 \end{array}\right. \Rightarrow \left[\begin{array}{l} e=0 \\ e=5 \end{array}\right.$$

В записаній сукупності чотирьох рівностей не справджуються третя та четверта, які відповідають тому випадку, коли з розряду одиниць суми переходить один десяток, тобто коли $n = 5$. Отже, $n = 0$, $e = 5$, а запис набирає вигляду

$$\begin{array}{cccccc} & & & t & 5 & 0 \\ + & & & t & 5 & 0 \\ \hline & f & o & r & t & y \\ \hline & s & i & x & t & y \end{array}.$$

Очевидно, що $f \le 8$, $s = f+1$, $o \ge 8$. Якщо $o = 8$, то з розряду сотень має переноситись 2 тисячі, що суперечить умові $i \ne 0 = n$. Отже, $o = 9$, $i = 1$. Запис набирає вигляду

$$\begin{array}{cccccc} & & & t & 5 & 0 \\ + & & & t & 5 & 0 \\ \hline & f & 9 & r & t & y \\ \hline & s & 1 & x & t & y \end{array}.$$

Для літер t, r, f, y залишилися цифри 2, 3, 4, 6, 7, 8, причому $s = f+1$, а x однозначно визначається для відомих t і r.

Оскільки $n = 0$, $i = 1$, то $x \ge 2$. З аналізу цифр третього розряду, враховуючи, що в розряд тисяч переносяться 2 тисячі, маємо:

$$x + 20 = 2t + r + 1. \tag{2}$$

Очевидно, що

$$\begin{array}{rcl} 2 \le x & \Rightarrow & 22 \le x + 20 \\ r \le 8 & \Rightarrow & 2t + r + 1 \le 2t + 9 \end{array}$$

Враховуючи рівність (2), маємо:

$$22 \le 2t + 9 \Rightarrow 13 \le 2t \Rightarrow 6,5 \le t.$$

Оскільки t ціле число, то воно дорівнює 7 або 8.

$$22 \le x + 20 = 2t + r + 1 \le 15 + r,$$

звідки $r \geq 7$. Отже, справджується одна з двох систем рівнянь:

$$\left\{ \begin{array}{l} t = 7 \\ r = 8 \end{array} \right. \vee \left\{ \begin{array}{l} t = 8 \\ r = 7 \end{array} \right. .$$

У першому випадку запис набирає вигляду

$$\begin{array}{cccccc} & & & 7 & 5 & 0 \\ + & & & 7 & 5 & 0 \\ \hline & f & 9 & 8 & 7 & y \\ \hline & s & 1 & 3 & 7 & y \end{array} .$$

Для f, s, y залишаються цифри 2, 4, 6. Але з них не можна вибрати такі, щоб $s = f + 1$.

В другому випадку запис має вигляд

$$\begin{array}{cccccc} & & & 8 & 5 & 0 \\ + & & & 8 & 5 & 0 \\ \hline & f & 9 & 7 & 8 & y \\ \hline & s & 1 & 4 & 8 & y \end{array} .$$

Для f, s, y залишаються цифри 2, 3, 6, причому $s = f + 1$. Отже, $f = 2$, $s = 3$, $y = 6$.

Відповідь:

$$\begin{array}{cccccc} & & & 8 & 5 & 0 \\ + & & & 8 & 5 & 0 \\ \hline & 2 & 9 & 7 & 8 & 6 \\ \hline & 3 & 1 & 4 & 8 & 6 \end{array} .$$