Данило Петрович Мисак,

керівник гуртка СШ № 52 м. Києва

Олександр Владиславович Рибак,

аспірант Інституту математики НАН України

Олександр Борисович Рудик,

доцент Київського університету імені Бориса Грінченка

Олімпіада з інформатики у місті Києві у 2014/2015 навчальному році

Передмова

Стаття містить умови завдань II (районного) і завдань III (міського) етапу олімпіади з основ інформатики й обчислювальної техніки у місті Києві у 2014/2015навчальному році та авторські розв'язання цих завдань. Публікацію адресовано учням класів з поглибленим вивченням математики, учасникам олімпіад з інформатики, студентам математичних спеціальностей, учителям і викладачам вищих навчальних закладів.

Цього навчального року, як і попереднього, упорядником завдань II етапу був Данило Мисак.

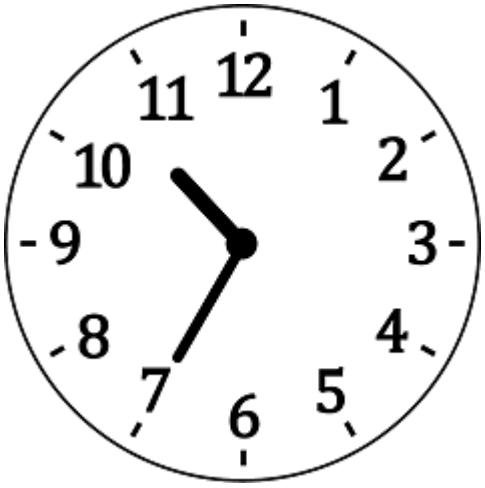
1. Умови завдань II етапу

Максимальна оцінка за кожну з чотирьох задач — 100 балів.Для всіх задачобмеження на час — 1 секунда / тест;обмеження на пам'ять — 256 МБ.

1. Стрілки

Назва програми: hands.pas / hands.cpp

В Орисі та Марисі є по одному улюбленому числу в межах від 1 до 12 включно, причому ці числа не обов'язково різні. З'ясуйте, скільки протягом доби є таких моментів часу, що хоча б одна зі стрілок годинника (годинна або хвилинна) точно вказує на улюблене число принаймні однієї з дівчат.



Вхідні дані

У вхідному файлі вказано два натуральних числа: улюблене число Орисі та улюблене число Марисі.

Вихідні дані

У вихідний файл виведіть кількість відповідних моментів часу протягом однієї доби.

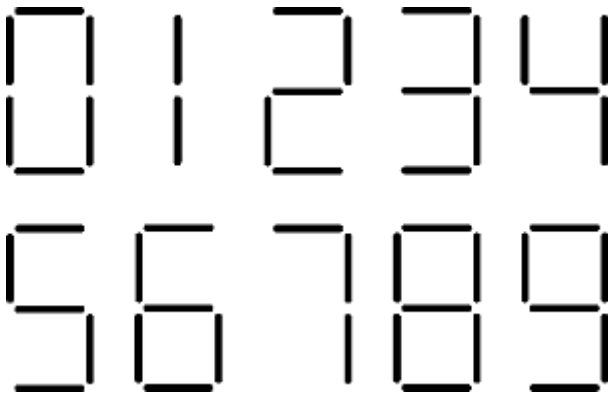
Приклад

hands.in	hands.out
2 7	52

2. Сірники

Назва програми: matches.pas / matches.cpp

Орися та Марися люблять гратися з сірниками. Кожна з дівчат має по *n* сірників. Орися хоче викласти із своїх сірників якомога більше число, а Марися — якомога менше число, причому дівчата хочуть використати всі свої сірники. Допоможіть Орисі та Марисі. Те, як з сірників викладаються цифри, ви можете подивитися на рисунку нижче.



Дівчата знають, що ставити на початок чисел нулі не можна.

Вхідні дані

У вхідному файлі записано натуральне число *n*. Воно є не меншим за 2 і не перевищує 2000.

Вихідні дані

У вихідний файл через пробіл виведіть два числа: найбільше і найменше натуральні числа, які можуть викласти дівчата саме з *n* сірників.

Приклад

matches.in	matches.out
5	71 2

3. Цукерки

Назва програми: candy.pas / candy.cpp

Якось Орисі наснилися *n* гномів, у кожного з яких була певна додатна кількість цукерок, що ділиться на $n - 1$. На Новий рік кожен гном розділив свої цукерки на $n - 1$ рівну частину та подарував усім іншим гномам по одній такій частині. Коли Орися розповіла про цей сон Марисі, тій стало цікаво, скільки ж цукерок було спочатку в кожного гнома. На жаль, Орися запам’ятала тільки те, скільки цукерок опинилося в кожного гнома після святкування Нового року. Допоможіть дівчатам за цією інформацією відновити початкові кількості цукерок.

Вхідні дані

У першому рядку вхідного файлу вказано натуральне число *n*, не менше за 2 і не більше за 200 000. У другому рядку записано *n* натуральних чисел, менших за мільйон, — кінцеві кількості цукерок у гномів.

Вихідні дані

У вихідний файл виведіть початкові кількості цукерок у кожного з гномів у тому самому порядку, в якому гноми задані у вхідному файлі. Якщо можливих відповідей декілька, виведіть будь-яку з них. Якщо жодного варіанта відповіді, що задовольняє умову задачі, не існує, виведіть лише одне число `0`.

Приклади

candy.in	candy.out
4 6 5 4 6	3 6 9 3
3 4 4 5	0

Пояснення до прикладів

При початковому розташуванні цукерок 3, 6, 9, 3 перший гном отримає від другого 6/(4 − 1) = 2 цукерки, від третього — 9/(4 − 1) = 3 цукерки, а від четвертого — 3/(4 − 1) = 1 цукерку (разом він матиме 2 + 3 + 1 = 6 цукерок, бо свої три він віддасть); другий гном отримає 3/3 + 9/3 + 3/3 = 5 цукерок; третій матиме 3/3 + 6/3 +3/3 = 4 цукерки; четвертий — 3/3 + 6/3 + 9/3 = 6 цукерок. Можна переконатися, що в другому прикладі жодного можливого початкового розташування не існує.

4. Поїзди

Назва програми: trains.pas / trains.cpp

У місті, де живуть Орися та Марися, є *n* ліній метрополітену і *m* станцій (деякі з них можуть бути станціями пересадки і належати відразу кільком лініям). Усі станції занумеровано натуральними числами від 1 до *m*. Орися живе біля станції з номером 1, а Марися — біля станції з номером *m*. Знаючи, скільки хвилин забирає проїзд між кожними двома сусідніми станціями на кожній лінії, визначте, за який найменший час Орися зможе доїхати до Марисі. Відомо, що між станціями, де живуть дівчата, існує сполучення (можливо, з пересадками). Пересадка між лініями, а також вхід у метро (на будь-яку лінію) та вихід (з будь-якої лінії) здійснюють миттєво. Поїзди ходять щохвилини, тож Орися не чекатиме поїздів на станціях. Часом зупинки поїзда на станції також можна знехтувати. На всіх лініях поїзди рухаються в обидва боки.

Вхідні дані

У першому рядку вхідного файлу записано два натуральних числа *n* та *m*, які не перевищують 500. Кожен з наступних *n* рядків задає лінію метро: перше

число в рядку — кількість станцій на лінії (не менша за 2). Далі вказано такі натуральні числа: номер першої станції на лінії; кількість хвилин, які забирає поїздка між першою та другою станціями лінії; номер другої станції лінії; кількість хвилин, які забирає поїздка між другою і третьою станціями лінії; номер третьої станції лінії і т. д. Жодна станція не може повторюватися на одній і тій самій лінії двічі, тобто поїзди не ходять по колу і лінії метро не перетинають самі себе. Крім того, жодні дві станції не можуть бути сусідніми відразу на двох різних лініях. Час, необхідний на переїзд між будь-якими двома сусідніми станціями, не перевищує 8 хвилин. У 50% тестів цей час дорівнює 1 хвилині для всіх пар сусідніх станцій.

Вихідні дані

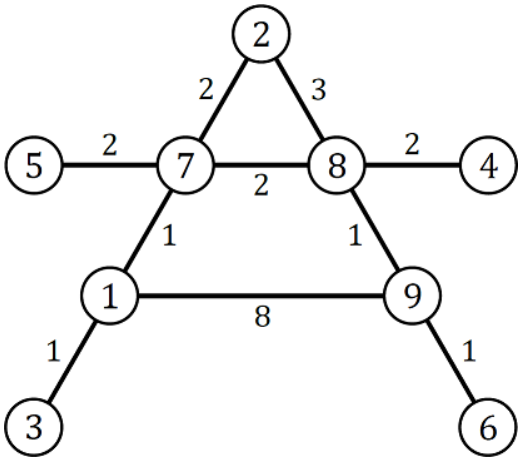
У вихідний файл виведіть єдине число — найменшу кількість хвилин, за яку Орися зможе доїхати до Марисі.

Приклад

trains.in	**trains.out**
3 9 4 5 2 7 2 8 2 4 7 6 1 9 1 8 3 2 2 7 1 1 1 3 2 1 8 9	4

Пояснення до прикладу

Маємо три лінії метро, що проходять через дев’ять станцій. Схему ліній та час переїзду між кожними двома сусідніми станціями показано на наступному рисунку.



Щоб дістатися з першої до дев’ятої станції якнайшвидше, зробимо так: сядемо на другу лінію (з сімома станціями), доїдемо до сьомої станції, пересядемо на першу лінію (з чотирма станціями), доїдемо до восьмої станції та знову пересядемо на другу лінію, вийдемо на дев’ятій станції. Це забере 1 + 2 + 1 = 4 хвилини. Альтернативні маршрути без пересадок тривають 1 + 2 + 3 + 1 = 7 хвилин по другій лінії та 8 хвилин по третій.

2. Ідеї розв’язання завдань II етапу

1. Стрілки

Якщо улюблені числа дівчат різні, то є рівно 2 × 2 = 4 моменти протягом доби, коли на одне з них вказує годинна стрілка та 2 × 24 = 48 моментів, коли на одне з них вказує хвилинна стрілка; загалом 4 + 48 = 52 моменти. При цьому деякі з них ми могли порахувати двічі — ті, під час яких і годинна, і хвилинна стрілка показують на улюблені числа дівчат. Але якщо годинна стрілка показує на ціле число, то хвилинна має показувати на 12. Якщо одне з чисел справді дорівнює 12, то двічі ми порахували чотири моменти: два, коли обидві стрілки показують на 12, і два, коли хвилинна показує на 12, а годинна — на улюблене число іншої дівчини. Відповідь у такому випадку дорівнює 52 – 4 = 48. Якщо ж серед чисел немає числа 12, то відповідь — 52.

Якщо улюблені числа дівчат однакові, то є рівно 2 моменти, коли на них указує годинна стрілка, та 24 моменти, коли на них вказує хвилинна стрілка. Разом 24 + 2 = 26 моментів. Одночасно стрілки можуть вказувати на числа лише тоді, коли ті дорівнюють 12. Тоді моментів, які ми порахували двічі, — два, а відповіддю є 26 – 2=24.

2. Сірники

Випишемо, скільки сірників містить кожна цифра:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
6	2	5	5	4	5	6	3	7	6

Найменшу кількість сірників — два — містить цифра 1, а найбільшу — сім — цифра 8. Як відомо, число є тим більшим, що більшою є в ньому кількість цифр, а за однакової кількості цифр число є більшим, якщо має більші старші розряди. Спершу розглянемо те, яке *максимальне* число можна утворити з n сірників:

- Якщо n парне, тобто $n = 2k$, то число може містити щонайбільше k цифр, причому k цифр воно міститиме тоді й лише тоді, коли всі вони — одиниці. Отже, Орися складе число 11…1, де кількість одиниць дорівнює $n/2$.
- Якщо n непарне, тобто $n = 2k + 1$, то число також може містити щонайбільше k цифр, причому k цифр воно міститиме тоді й лише тоді, коли всі цифри, крім однієї, одиниці, а цифра, відмінна від одиниці, містить три сірники. Такою є лише цифра 7; при цьому сімку, очевидно, є сенс поставити на перше місце. Отже, Орися складе число 711…1, де кількість одиниць дорівнює $(n - 3)/2$.

Тепер розглянемо варіанти для *мінімального* числа, яке можна утворити рівно з n сірників:

- Якщо n ділиться на 7, тобто $n = 7k$, то число може містити щонайменше k цифр, причому k цифр воно міститиме тоді й лише тоді, коли всі вони — вісімки. Отже, Марися складе число 88…8, де кількість вісімок дорівнює $n/7$.
- Якщо $n = 7k + 1$, то число може містити щонайменше $k + 1$ цифру, при цьому першою цифрою може бути одиниця, а другою — нуль. Тоді решта $(n - 8)/7$ цифр — вісімки.
- Якщо $n = 7k + 2$ ($k \geqslant 0$), то число може містити щонайменше $k + 1$ цифру, при цьому першою цифрою може бути одиниця. Тоді решта $(n - 2)/7$ цифр — вісімки.
- Якщо $n = 7k + 3$ ($k \geqslant 0$), то число може містити щонайменше $k + 1$ цифру, при цьому одиниця першою бути не може (бо інакше на решту k цифр залишилося б $7k + 1 > 7k$ сірників). Тоді у випадку $k = 0$ перша і єдина цифра дорівнює 7, а інакше першою цифрою можна зробити 2. Далі, якщо $k = 1$, наступна цифра остання і повинна містити 5 сірників. Найменшою

такою цифрою знову є 2. Отже, відповідь — 22. Якщо ж $k > 1$, то дві наступні цифри можна зробити нулями, а решту $(n – 17)/7$ цифр доведеться зробити вісімками.

- Якщо $n = 7k + 4$ $(k \geqslant 0)$, то число може містити щонайменше $k + 1$ цифру, при цьому одиниця першою бути знову не може. Тоді у випадку $k = 0$ перша і єдина цифра дорівнює 4, а інакше першою цифрою можна зробити 2. Наступна цифра може бути нулем, а решта $(n – 11)/7$ цифр — вісімки.
- Якщо $n = 7k + 5$ $(k \geqslant 0)$, то число може містити щонайменше $k + 1$ цифру, при цьому одиниця першою бути не може. Але першою цифрою можна зробити 2; решта $(n – 5)/7$ цифр — вісімки.
- Якщо $n = 7k + 6$ $(k \geqslant 0)$, то число може містити щонайменше $k + 1$ цифру, при цьому на першому місці повинна стояти ненульова цифра, що містить принаймні 6 сірників, найменша з таких — цифра 6. На решті $(n − 6)/7$ місць стоятимуть вісімки.

Наостанок зауважимо, що відповіді можуть виявитися досить великими і не вміщатися у межі стандартних типів, тому їх слід виводити не як числа, а просто як послідовність цифр.

3. Цукерки

Позначимо початкові кількості цукерок у гномів через a_k, а кінцеві через b_k, $1 \leqslant k \leqslant n$. Загальна кількість цукерок у гномів не змінилась, позначимо її через S:

$$S = b_1 + b_2 + \ldots + b_n = a_1 + a_2 + \ldots + a_n.$$

Крім того, згідно з умовою задачі

$$b_k = \frac{a_1}{n-1} + \frac{a_2}{n-1} + \ldots + \frac{a_{k-1}}{n-1} + \frac{a_{k+1}}{n-1} + \ldots + \frac{a_n}{n-1} = \frac{a_1 + a_2 + \ldots + a_n}{n-1} - \frac{a_k}{n-1} = \frac{S}{n-1} - \frac{a_k}{n-1},$$

звідки $a_k = S - (n-1)b_k$. Якщо всі такі значення діляться на $n-1$ (а це буде тоді й лише тоді, коли на $n-1$ ділиться число S) і є додатними, то вони становлять відповідь: у цьому нескладно переконатися, підставивши їх у вираз

$\frac{a_1 + a_2 + \ldots + a_n}{n-1} - \frac{a_k}{n-1}$ і пересвідчившись, що його значення справді дорівнює b_k. Інакше потрібно вивести нуль.

Додамо, що перераховувати суму чисел кожного разу не слід, адже тоді складність виконання алгоритму стане квадратичною від кількості гномів і для великих значень n програма буде працювати дуже повільно. Достатньо порахувати суму чисел один раз. Вона може виявитися досить великою, тому у програмі потрібно оперувати 64-бітовими змінними.

4. Поїзди

Заданий у вхідному файлі план ліній метрополітену слід подати у програмі як звичайний неорієнтований граф зі зваженими ребрами (вага ребра — кількість хвилин, які поїзд їде від однієї станції до іншої).

Для пошуку найкоротшого маршруту між вершиною 1 та вершиною m можна використати алгоритм Дейкстри, причому на повний бал достатньо найпростішої його реалізації, що працює за квадратичний від кількості вершин час.

Інший підхід — штучно розбити кожне ребро ваги k графа додатковими проміжними вершинами на k одиничних ребер. Далі застосувати пошук у ширину. Кількість ребер графа при цьому зростає не більш ніж у 8 разів, а нових вершин додається стільки ж, скільки й нових ребер. Отже, час виконання пошуку в ширину залишається фактично квадратичним від початкової кількості вершин.

Оскільки у 50 % тестів час на проїзд між будь-якими двома сусідніми станціями складає 1 хвилину, простий пошук у ширину на графі з ігноруванням ваг ребер заробить 50 балів. Якщо при цьому виводити не довжину маршруту, а суму ваг ребер на ньому, програма може пройти додатково ще один чи два тести.

3. Авторські розв’язання завдань II етапу

1. Стрілки

```
/* GCC */

#include<stdio.h>

intmain()
```

```
{
freopen("hands.in", "r", stdin);
freopen("hands.out", "w", stdout);

int a, b, ans;

// Зчитування даних:
scanf("%d %d", &a, &b);

// Обчислення відповіді:
if(a == 12 || b == 12)
ans = 24;
else
ans = 26;
if(a != b)
ans *= 2;

// Виведення відповіді:
printf("%d\n", ans);

return 0;
}
```

2. Сірники

```
/* GCC */

#include<stdio.h>

int main()
{
freopen("matches.in", "r", stdin);
freopen("matches.out", "w", stdout);

int n;

// Зчитування кількості сірників:
```

```
scanf("%d", &n);

// Виведення максимального числа, яке
    // можна скласти з n сірників:
printf(n % 2 == 0 ? "1" : "7");
for(inti = 0; i < n/2 - 1; i++)
printf("1");

printf(" ");

// Виведення мінімального числа, яке
    // можна скласти з n сірників:
inteights = 0; // Кількість вісімок у
                    // числі (буде оновлю-
                    // ватись далі в коді)
switch(n % 7)
    {   // Залежно від остачі при діленні
        // n на 7:
case0:
eights = n/7;
break;
case1:
printf("10");
eights = (n - 8)/7;
break;
case2:
printf("1");
eights = (n - 2)/7;
break;
case3:
if(n/7 == 0)
printf("7");
elseif(n/7 == 1)
printf("22");
else
{
printf("200");
```

```
eights = (n - 17)/7;
            }
break;
case4:
if(n/7 == 0)
printf("4");
else
{
printf("20");
eights = (n - 11)/7;
            }
break;
case5:
printf("2");
eights = (n - 5)/7;
break;
case6:
printf("6");
eights = (n - 6)/7;
break;
    }
// Виводимо вісімки:
for(inti = 0; i <eights; i++)
printf("8");
printf("\n");
return0;
}
```

3. Цукерки

```
{ FreePascal }

constmaxN = 200000; // Максимальна можлива
                    // кількість гномів

vara, b: array[1..maxN] ofint64; // По-
                // чаткові та кінцеві кіль-
```

```
                // кості цукерок у гномів
S: int64; // Загальна кількість цукерок
i, n: longint;
ans: boolean; // Чи існує відповідь

begin
assign(input, 'candy.in');
reset(input);
assign(output, 'candy.out');
rewrite(output);

    S := 0;

// Зчитування даних та підрахунок
    // загальної кількості цукерок:
read(n);
fori := 1 ton dobegin
read(b[i]);
        S := S + b[i];
end;

ans := (S mod (n - 1) = 0);
// Підрахунок відповіді:
if(ans) then
fori := 1 ton dobegin
a[i] := S - (n - 1) * b[i];
ifa[i] <= 0 thenbegin// Якщо
ans := false; // вийшло не-
break;    // додатне число,
end;      // відповіді не існує
end;

// Виведення відповіді:
if(ans) thenbegin
fori := 1 ton - 1 do
write(a[i], ' ');
writeln(a[n]);
```

```
endelse
writeln(0);
end.
```

4. Поїзди

```
/* GCC */

#definemaxM500    // Найбільша можлива
                    // кількість станцій
#include<stdio.h>
#include<vector>

usingnamespacestd;

vector<pair<int, int>>graph[maxM];
// graph[i][j].first — номер вершини,
// суміжної з вершиною i (нумерація з нуля)
// graph[i][j].second — вага відповідного
// ребра

intdistances[maxM];
// Поточні відстані від початкової вершини
// (-1, якщо дана вершина поки недосяжна)

boolvisited[maxM];
// Індикатор того, чи додано на даний
// момент відповідні вершини до тієї
// частини графа, де відстані пораховані
// остаточно

intmain()
{
freopen("trains.in", "r", stdin);
freopen("trains.out", "w", stdout);

intn, m;
```

```
// Зчитування даних та побудова графа:
scanf("%d %d", &n, &m);
for(inti = 0; i < n; i++)
    {
intstations, current, previous,
time;
scanf("%d %d", &stations,
&current); // Кіль-
                // кість станцій на лінії
                // та номер першої станції
for(intj = 0; j <stations - 1;
             j++)
        {
previous = current;
scanf("%d %d", &time,
&current); // Час
                    // на переїзд та номер
                    // наступної станції
            // Додаємо ребро ваги time між
            // вершинами previous та
            // current, зважаючи, що
            // нумерація вершин у нашій
            // програмі повинна починатися
            // не з одиниці, а з нуля:
graph[previous - 1].push_back(
make_pair(current - 1, time));
graph[current - 1].push_back(
make_pair(previous - 1, time));
        }
    }

// Шукаємо найкоротший шлях від вершини
    // 0 до всіх інших вершин графа:
for(inti = 0; i < m; i++) // Ініціа-
{          // лізація масивів відстаней
               // і доданості
```

```
distances[i] = -1;
visited[i] = false;
    }
intaddedVertex = 0; // Починаємо
                          // з вершини 0
distances[addedVertex] = 0;
while(addedVertex != -1) // Поки є
{      // нова вершина, яку слід додати
visited[addedVertex] = true; // По-
              // значаємо вершину як додану
for(inti = 0;
             i <graph[addedVertex].size();
             i++) // Для кожного ребра,
{         // що з неї виходить
intvertex = // Номер суміжної
                          // вершини
graph[addedVertex][i].first;
intweight = // Вага відповід-
                          // ного ребра
graph[addedVertex][i].second;
intdistance =
distances[addedVertex]
              + weight; // Час, за який
          // можна потрапити у vertex, якщо
          // йти через addedVertex
if(distances[vertex] == -1
|| distances[vertex]
>distance) // Якщо зна-
                 // йдений час менший за
                 // наявний, оновлюємо його
distances[vertex] =
distance;
        }
// Яку вершину слід додати
        // наступною:
addedVertex = -1;
for(inti = 0; i < m; i++) // Для
```

```
                        // кожної вершини
if(! visited[i]
&&distances[i] != -1 // Якщо
       // її ще не додано, але вона досяжна
&& (addedVertex == -1
|| distances[i]
<distances[addedVertex]))
// І якщо відстань до неї менша
       // за поточний мінімум
addedVertex = i;
    }

// Виводимо відповідь: найкоротшу від-
    // стань від вершини 0 до вершини m-1
printf("%d\n", distances[m - 1]);

return0;
}
```

4. Умови завдань III етапу

1. Дивовижне число (автор — Данило Мисак)

Максимальна оцінка: 200 балів

Обмеження на час: 2 сек.

Обмеження на пам’ять: 256 МБ

Вхідний файл: awesome.in

Вихідний файл: awesome.out

Програма: awesome.*

Якось Орися з Марисею натрапили на деяке натуральне число*n*. Орися порахувала суму його цифр, а Марися — добуток. На подив дівчатам пораховані сума та добуток виявилися рівними одному й тому самому числу *d*.

Завдання

Знаючи *d*,відновіть найбільше можливе значення початкового числа*n*.

Вхідні дані

У вхідному файлі вказано натуральне число d.

- У 50 % тестів $d \leq 20$.
- У 50 % тестів $20 < d \leq 1\,000\,000$.

Усі тести мають однакову вартість.

Вихідні дані

У вихідний файл виведіть найбільше можливе значення n або число `0`, якщо такого значення не існує.

Приклади

awesome.in	awesome.out
6	321
7	7
11	0

Пояснення до першого прикладу

І сума, і добуток цифр числа 321 дорівнюють 6: $3 + 2 + 1 = 3 \cdot 2 \cdot 1 = 6$. Таку саму властивість мають і числа 312, 231, 213, 132, 123, а також одноцифрове число 6, але 321 з них найбільше, тому саме його і треба вивести.

2. Черевички на підборах (автор — Данило Мисак)

Максимальна оцінка: 200 балів

Обмеження на час: 1,5 сек.

Обмеження на пам’ять: 256 МБ

Вхідний файл: heels.in

Вихідний файл: heels.out

Програма: heels.*

Орися та Марися збирають черевички на підборах: Орися ліві, а Марися — праві. Між собою черевички відрізняються лише висотою підбора. В обох дівчат назбиралося рівно по *n*черевиків. Деякі з черевиків, навіть якщо вони на одну й ту саму ногу, можуть мати однакову висоту підбора.

Завдання

Знаючи висоту підборів усіх лівих та всіх правих черевиків, з'ясуйте, скільки повноцінних пар взуття з них можна скласти. З лівого та правого черевика можна скласти пару, якщо висота підборів у них однакова.

Вхідні дані

У першому рядку вхідного файлу вказано натуральне число *n*. У другому рядку записано *n*натуральних чисел, що задають висоти підборів черевиків на ліву ногу. У третьому рядку записано *n*натуральних чисел, що задають висоти підборів черевиків на праву ногу. Висота жодного підбора не перевищує 10^9.

- У 25 % тестів $n \leq 1000$ і в жодної дівчини немає двох черевиків з однаковою висотою підбора.
- У 25 % тестів $n \leq 1000$ і принаймні в однієї з дівчат є хоча б два черевики з однаковою висотою підбора.
- У 25 % тестів $1000 < n \leq 200\,000$ і висоти підборів не перевищують 1000.
- У 25 % тестів $1000 < n \leq 200\,000$ і висота підбора принаймні на одному черевикові більша за 1000.

Усі тести мають однакову вартість.

Вихідні дані

Єдиний рядок вихідного файлуповинен містити кількість пар, які можна скласти з Орисиних лівих та Марисиних правих черевичків.

Приклад

heels.in	heels.out
7	4

heels.in	heels.out
6 533 1 58 5 2 83555	

Пояснення до прикладу

З черевиків дівчат можна скласти чотири пари: пару з висотою підборів 3, пару з висотою підборів 8 і дві пари з висотою підборів 5.

3. Два квадрати (автор — Данило Мисак)

Максимальна оцінка: 200 балів

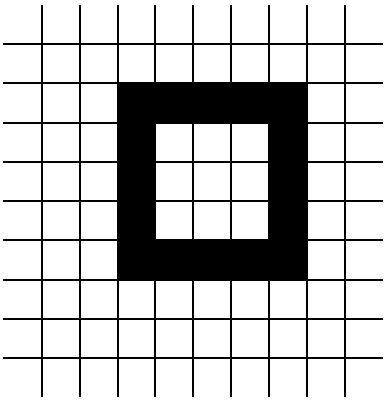
Обмеження на час: 2,5 сек.

Обмеження на пам’ять: 256 МБ

Вхідний файл: squares.in

Вихідний файл: squares.out

Програма: squares.*



В Орисі й Марисі є великий клітчастий аркуш паперу $n \times n$. Орися дуже товстим чорним маркером накреслила на ньому межі квадрата таким чином, що вони проходять не по межах клітинок, а по самих клітинках. З того ж боку аркуша свій квадрат тим же чорним маркером накреслила й Марися, причому її квадрат міг дотикатися, перетинатися або навіть збігатися з Орисиним. Розміри квадратів дівчат могли бути однаковими, але могли й відрізнятися. Відомо, однак, що розміри зовнішнього контуру обох квадратів не менші за 3×3 (тобто квадрат не вироджується в чотири чорних клітинки).

Завдання

За інформацією, як виглядає аркуш, визначте розміри й розташування на ньому обох квадратів.

Вхідні дані

У першому рядку вхідного файлу вказано натуральне число n — лінійний розмір аркуша. Далі йде n рядків, у кожному з яких *через пробіл* задано n символів, кожен з яких є або крапкою, що позначає незафарбовану клітинку, або символом #, що позначає зафарбовану клітинку. Після останнього символу кожного рядка (крапки чи #) пробілу *немає*.

- У 25 % тестів $3 \le n \le 10$.
- У 75 % тестів $10 < n \le 1000$.

Усі тести мають однакову вартість.

Остаточну оцінку за цю задачу буде отримано таким чином: сумарний бал за пройдені тести округлять униз до найближчого числа, що ділиться на 40. Наприклад, якщо ваша програма пройде 100 % тестів, вона набере 200 балів; якщо ваша програма пройде 95 % тестів, вона набере 160 балів; якщо програма пройде 45 % тестів, вона набере 80 балів; якщо програма пройде 10 % тестів, вона набере 0 балів.

Вихідні дані

Нумерація клітинок починається з лівого верхнього кута. Ліва верхня клітинка аркуша має координати (1, 1), клітинка праворуч від неї — (2, 1), клітинка знизу — (1, 2) і т. д.

У вихідному файлі мають міститися два рядки по чотири числа, записаних через пробіл: кожен рядок задає відповідний квадрат. Перші два числа рядка задають координати лівої верхньої клітинки квадрата, а наступні два числа — координати правої нижньої клітинки. Самі рядки повинні бути розташовані в лексикографічному порядку, тобто першим іде рядок з меншим першим числом; якщо перші числа однакові, то з меншим другим числом; якщо другі числа теж однакові, то з меншим третім числом; якщо й треті числа

однакові, то з меншим четвертим числом; якщо всі числа однакові, порядок ролі не грає.

У випадку, коли можливих розташувань квадратів декілька, виведіть інформацію про довільне одне з них. Вхідні дані гарантують наявність принаймні одного розташування, що задовольняє умову задачі.

Приклад

squares.in	**squares.out**
10 . . . . # # # # # # . . . . # . . . . # . . . . # . . . . # . . # # # # # . . # . . # . # . # . . # . . # . # # # # # # . . # . . . # . . . . . # # # # #	2 6 6 10 4 3 9 8

4. Схил (автор — Олександр Рибак)

Максимальна оцінка: 200 балів

Обмеження на час: 0,5 сек.

Обмеження на пам’ять: 32 МБ

Вхідний файл: slope.in

Вихідний файл: slope.out

Програма: slope.*

У стіну забито багато цвяхів. Орися і Марися намагаються прив’язати натягнуту нитку до двох цвяхів так, щоб отримати якомога менший, *але відмінний від нуля* кут нахилу її до горизонталі.

Завдання

Серед даних точок координатної площини знайти дві точки, які є кінцями відрізка з найменшим *додатним* кутом нахилу відносно горизонталі.

Вхідні дані

Перший рядок містить натуральне число n ($2 \le n \le 100\,000$) — кількість точок, де розташовано цвяхи. Далі ідуть n рядків, кожен з яких містить два цілих числа x_j, y_j ($0 \le x_j, y_j \le 30\,000$), що є прямокутними координатами *різних* точок. Число x_j задає розташування по горизонталі, а y_j — по вертикалі.

Вихідні дані

В єдиному рядку через пробіли вивести числа x_j, y_j, x_k, y_k. Тут (x_j, y_j) та (x_k, y_k) — координати двох точок з даних, які задають *найпологіший схил*. Інакше кажучи, при яких відношення $|x_j - x_k| / |y_j - y_k|$ *визначене* і є *найбільшим* серед відношень такого вигляду.

Точки вказати в такому порядку, щоб справджувалася нерівність $y_j < y_k$ і:

- якщо можливих розв'язків декілька, обрати той, де y_j є найменшим;
- якщо й за цієї умови є кілька варіантів, обрати розв'язок з найменшим x_j;
- якщо й за цих умов (попарного збігу перших двох чисел) є кілька варіантів, обрати розв'язок з найменшим y_k;
- якщо й за цих умов є кілька варіантів, обрати розв'язок з найменшим x_k.

Якщо задача не має розв'язку, вивести `0`.

Приклади

slope.in	slope.out
4 7 4 9 2 5 1	5 1 1 2

1 2	
2 123 45 256 45	0

5. Каса (автор — Олександр Рудик)

Максимальна оцінка: 200 балів

Обмеження на час: 10 сек.

Обмеження на пам'ять: 32 МБ

Вхідний файл: cash.in

Вихідний файл: cash.out

Програма: cash.*

Марися й Орися пішли до кінотеатру. Біля каси кінотеатру $m + n$ дітей (всього разом з Марисею та Орисею) вишикувалося у чергу за квитками, причому m з них мають лише по банкноті вартістю 20 гривень, а решта n мають лише по банкноті вартістю 10 гривень. Вартість квитка на дитячий сеанс складає 10 гривень. На початку роботи в касі є p банкнот вартістю по 10 гривень.

Завдання

Створити програму cash.*, яка визначить кількість способів надходження банкнот у касу таким чином, щоб жодна дитина не чекалана здачу, або лишок від ділення цієї кількості на певне натуральне число r.

Вхідні дані

Вхідний файл містить *натуральні* числа m, n і *невід'ємні цілі* числа p,r: $m \le n + p$; $m + n \le 32\,767$; $p \le 32\,767$; $r < 10^9$.

Вихідні дані

Вихідний файл має містити лише одне число:

- при $r = 0$ — кількість *різних* послідовностей банкнот по 10 і 20 гривень, при яких жодній дитині не доведеться чекати здачі (вхідні дані гарантують, що у цьому випадку вказана кількість менша від 10^{18});
- при $r > 0$ — лишок від ділення на r такої кількості.

Приклади

cash.in	cash.out
2 3 0 0	5
2 3 0 3	2
2 3 2 0	10
2 3 2 7	3

Пояснення до перших двох прикладів

(10, 10, 10, 20, 20), (10, 10, 20, 10, 20), (10, 10, 20, 20, 10), (10, 20, 10, 20, 10), (10, 20, 10, 10, 20) — вичерпний перелік прийнятних послідовностей надходження банкнот у касу, у якій на початку роботи немає нічого, а біля каси зібралося 5 дітей з трьома банкнотами по 10 гривень і двома банкнотами по 20 гривень.

6. Гра (автор — Олександр Рудик)

Максимальна оцінка: 200 балів

Обмеження на час: 0,1 сек.

Обмеження на пам’ять: 32 МБ

Вхідний файл: game.in

Вихідний файл: game.out

Програма: game.*

Марися й Орисяграють у таку гру. Скінченну кількість фішок розташовано в ряд і занумеровано послідовними натуральними числами, починаючи з 1. Два гравці по черзі забирають довільнунатуральну кількість фішок від 1 до певного натурального *g*, *розташованих поруч*.Інакше кажучи, номери забраних за один хід фішок — послідовні натуральні числа. Переможцем вважають того, хто зробитьостанній хід.

Завдання

Створіть програму game.*, яка для довільної позиції гри визначає всі виграшні ходи — такі ходи, що гарантують виграш (за умови правильного продовження гри зі свого боку) незалежно від ходів суперника.

Вхідні дані

Вхідний файл містить 2 рядки. У першому рядку —*натуральне* число *g* у межах від 2до 9. У другому рядку записано у порядку зростання *натуральні*номери наявних фішок, які менші за 256 (щонайменше 7 чисел).

Вихідні дані

Вихідний файл має містити *g* рядків, кожний з яких закінчується ознакою кінця рядка. *J*-ий рядок вихідного файлу має містити у довільному порядку номери фішок, забравши якіразом з *наступними сусідніми*(*j* – 1)фішкоюгравець робить виграшний хід з позиції, заданої вхідними даними. Якщо таких ходів немає, то відповідний рядок вихідного файлу порожній. Зайвими пропусками при перевірці буде знехтувано.

Приклади

game.in	game.out
2 1 2 3 4 6 7 8	1 2 3 4 6 7
6	

2 3 4 5 6 7 8 9	5 4 3

5. Ідеї розв’язання завдань III етапу

1. Дивовижне число

Якщо число d має простий дільник, більший за 10, воно не може виявитися добутком цифр жодного натурального числа. В іншому разі d можна подати як добуток степенів простих чисел 2, 3, 5 і 7: $d = 2^w \times 3^x \times 5^y \times 7^z$, де степені w, x, y, z невід’ємні цілі. Тоді шукане число міститиме w двійок, x трійок, y п’ятірок, z сімок і $d - 2w - 3x - 5y - 7z$ одиниць та матиме вигляд 77...755...533...322...211...1 (якихось із цифр 7, 5, 3, 2 чи 1 може й не бути, якщо відповідна кількість дорівнює нулю). Справді: число повинно містити рівно z сімок і рівно y п’ятірок, адже інші ненульові цифри не дають множників 7 і 5. Якщо число містить цифру 4, то її можна замінити двома цифрами 22, збільшивши число, але при цьому не змінивши ні суми, ні добутку його цифр. Аналогічно цифру 6 можна замінити цифрами 321, цифру 8 — цифрами 22211, а 9 — 33111. Відповідно, в найбільшому числі, що задовольняє умову задачі, є сенс використовувати лише двійки і трійки, причому рівно w та x таких цифр відповідно. Вже маємо добуток цифр d, але сума на даний момент становить $2w + 3x + 5y + 7z$. Єдиний спосіб отримати з неї суму d, не змінивши при цьому добутку, — додати рівно $d - 2w - 3x - 5y - 7z$ одиниць. Це число невід’ємне, адже сума довільного набору натуральних чисел, більших за 1, не може перевищувати добутку цих же чисел (пропонуємо довести це самостійно).

2. Черевички на підборах

Спершу числа в кожній з двох послідовностей потрібно впорядкувати за неспаданням, скориставшись одним з ефективних методів сортування. Далі,

починаючи з перших елементів, потрібно рухатись зліва направо одночасно по двох послідовностях. На кожному кроці:

- Якщо поточний елемент першої послідовності менший за поточний елемент другої послідовності, перейти до наступного елемента першої послідовності.
- Якщо поточний елемент другої послідовності менший за поточний елемент першої послідовності, перейти до наступного елемента другої послідовності.
- Якщо поточні елементи послідовностей однакові, додати до лічильника-відповіді 1 і перейти до наступних елементів в обох послідовностях.
- Якщо хоча б в одній послідовності закінчилися елементи, вийти.

Цей процес подібний до етапу злиття відсортованих фрагментів масиву в алгоритмі сортування злиттям, який, до речі, і можна використати в цій задачі як ефективний метод сортування.

Додамо, що, коли значення самих чисел не перевищують 1000, можна скористатися іншим підходом: у k-ту комірку масиву a, $1 \le k \le 1000$, записати кількість чисел kу першій послідовності (це можна зробити за єдиний прохід) і аналогічно в k-ту комірку масиву bзаписати кількість чисел k у другій послідовності. Таким чином, $a_1 + a_2 + \ldots + a_{1000} = b_1 + b_2 + \ldots + b_{1000} = n$.

Відповіддю у такому разі буде сума

$$\min\{a_1, b_1\} + \min\{a_2, b_2\} + \ldots + \min\{a_{1000}, b_{1000}\},$$

де $\min\{x, y\}$ позначає менше з двох чисел x, y.

3. Два квадрати

Розв’язати задачу можна за допомогою кількох підходів. Один із них такий.

Спершу визначимо розташування одного квадрата:

1. Повним перебором знайдемо чорну клітинку A, і зліва, і зверху від якої клітинки або відсутні, або білі. Така клітинка обов’язково знайдеться, бо інакше ми могли б переходити по чорних клітинках ліворуч або вгору нескінченно довго. Якщо клітинок з такою властивістю більше ніж одна,

нам потрібна найлівіша з них. Знайдена клітинка*A* обов'язково є лівою верхньою клітинкою принаймні одного з квадратів.

2. Порахуємо кількість чорних клітинок у тому ж рядку праворуч від знайденої і кількість чорних клітинок у тому ж стовпці під знайденою. Ці кількості можуть бути однаковими або різними, але в будь-якому разі менша з них задає довжину сторони квадрата з лівою верхньою вершиною в клітинці *A*.
3. За знайденою інформацією про ліву верхню клітинку і сторону квадрата повністю визначимо розташування квадрата.

Тепер розглянемо такі варіанти:

- Якщо крім клітинок першого квадрата немає жодних інших зафарбованих клітин, другий квадрат, очевидно, має те саме розташування, що й перший.
- Якщо є хоча б одна чорна клітинка за межами першого квадрата, то за його межами є й принаймні одна вершина другого квадрата, яку можна визначити як клітинку, з двох непротилежних боків від якої клітинки або білі, або відсутні (тобто зліва і зверху, справа і зверху, справа і знизу або зліва і знизу). З цієї вершини виходить принаймні одна сторона, про яку напевне можна сказати, що вона не перетинає перший квадрат. За цими вершиною і стороною можна повністю відновити розташування другого квадрата.
- Інакше, тобто якщо всі чорні клітинки лежать усередині першого квадрата, причому хоча б одна лежить строго всередині, зробимо таке. Розглянемо множину горизонталей та вертикалей, на яких лежить принаймні одна чорна клітинка, яка точно не належить першому квадрату. Включимо до цієї множини горизонталь, на якій лежить верхня сторона першого квадрата тоді й лише тоді, коли на наступній за нею горизонталі є хоча б одна чорна клітинка, що лежить строго всередині першого квадрата, але немає двох сусідніх таких клітинок. Аналогічно зробимо з лініями, на яких лежать нижня, ліва та права сторони першого

квадрата. Тоді шуканий другий квадрат утворюватиметься крайніми горизонталями та крайніми вертикалями побудованої множини ліній.

Зауважимо, що з поданого розв'язання випливає, що можливе розташування квадратів завжди єдине.

4. Схил

Позначимо точки з цвяхами через $P_1, P_2, ..., P_n$. Згідно з умовою, при $i = 1, 2, ..., n$ координатами точки P_i є x_i та y_i. Для вказаних точок запровадимо поняття порядку таким чином: будемо вважати, що точка P_i менша за точку P_j, якщо $y_i < y_j$, або якщо одночасно $y_i = y_j$ та $x_i < x_j$. Останнє будемо записувати як $P_i < P_j$. Легко пересвідчитися, що з висловлювань $P_i < P_j$ і $P_j < P_k$ випливає $P_i < P_k$.

Перенумеруємо дані точки в порядку зростання: $P_1 < P_2 < ... < P_n$. Це можна зробити за допомогою ефективного алгоритму впорядкування. Автор обрав метод злиття, як і в розв'язанні задачі «Істотні інверсії» 2013 року. Хоча для більшості учасників знайомішим буде швидкий метод Хоара.

Згідно з поданим принципом сортування, у впорядкованій послідовності $P_1, P_2, ..., P_n$ між двома точками з однаковою ординатою y можуть бути лише точки з цією самою ординатою. Тому впорядкована послідовність, $P_1, P_2, ..., P_n$ складається з блоків точок $B_1, B_2, ..., B_k$, де кожен блок містить точки з однаковою ординатою y, а в різних блоках точки мають різні ординати. У довільному блоці B_j точки розташовано за зростанням абсциси x.

Зауважимо, що у випадку $y_i < y_j < y_k$ пара точок P_i та P_k не може бути відповіддю:

- якщо точка P_j не лежить на прямій P_iP_k, то пара (P_i, P_j) або пара (P_j, P_k) дає менший, але також не нульовий кут нахилу до горизонталі;
- якщо точка (x_j, y_j) лежить на вказаній прямій, то пара (P_i, P_j) прийнятніша за пару (P_i, P_k) згідно з додатковими умовами вибору відповіді.

Тому достатньо переглянути лише *пари точок з сусідніх блоків*. На роль відповіді претендують такі пари P_i та P_j, де кожна точка є першою або

останньою в своєму блоці. Тому всього потрібно порівняти не більше $4n$ пар. Для достатньо великих n це потребує меншої кількості операцій, ніж процес впорядкування точок.

Ефективність всього розв'язання збігається з ефективністю методу впорядкування і становить $O(n \log_2 n)$.

5. Каса

Довільному розташуванню дітей у черзі поставимо у відповідність послідовність чисел a_j:

- $a_j = -1$, якщо j-та дитина має банкноту 10 гривень;
- $a_j = +1$, якщо j-та дитина має банкноту 20 гривень.

Розглянемо суму $S_j = p + a_1 + a_2 + ... + a_j$, що є різницею між кількостями банкнот по 20 і 10 гривень після купівлі квитків першими j дітьми. *Для наочності* у прямокутній системі координат xOy розглянемо ламану з *послідовними* вершинами $A_j = (j, S_j)$ при $j = 0, 1, 2, ..., m + n$. Ця ламана сполучає точку $A_0 = (0, p)$з точкою $A_{n+m} = (m + n, p + n - m)$ і проходить через точки $A_1, A_2, ..., A_{m+n-1}$. Називатимемо таку ламану траєкторією, яка відповідає даному способу розташування покупців у черзі (точніше, даному способу надходження банкнот у касу).

Кожна траєкторія складається з $m + n$ похилих відрізків, n з яких піднімаються вгору — їхній кут нахилу до осі абсцис Ox складає + 45°, а інші m відрізків опускаються донизу — їхній кут нахилу до осі абсцис Ox складає –45°. Якщо вказати номери тих відрізків, які піднімаються вгору, то тим самим траєкторію буде визначено повністю. Тому загальне число траєкторій — число послідовностей банкнот — дорівнює $C_{m+n}^{n} = \binom{m+n}{n} = \frac{(m+n)}{m!n!}$.

Траєкторії, що відповідають тим способам розташування дітей, при яких жодна дитина не чекає здачі, не перетинають пряму $y = -1$ і не дотикаються до неї. Для того щоб підрахувати число таких траєкторій, поставимо у відповідність кожній *непридатній*траєкторії T, яка *перетинає або дотикається* до прямої $y = -1$, нову траєкторію T' за таким правилом: до *першої* точки

дотику з прямою $y = -1$ траєкторія T' збігається з T, а далі T' є образом траєкторії T при симетрії відносно прямої $y = -1$. Інакше кажучи, абсциси вершин ламаної залишаються тими самими, а ординати зазнають таких змін:

$$y' = 2\cdot(-1) - y = -2 - y.$$

Всі траєкторії T' закінчуються в точці $A'_{m+n} = (m + n, m - n - p - 2)$, яка є образом точки A_{m+n} при симетрії відносно прямої $y = -1$. Встановлена відповідність є взаємно однозначною. Кількість непридатних траєкторій дорівнює загальній кількості ламаних, які сполучають $A_0 = (0, p)$ з A'_{m+n}. Якщо така ламана складається з x відрізків, що йдуть донизу, й y відрізків, що йдуть угору, то

$x + y = m + n$;

$p + y - x = m - n - p - 2$,

звідки $y = m - p - 1$. При невід'ємному y кількість траєкторій з A_0 в A'_{m+n} дорівнює

$$C_{x+y}^{y} = C_{m+n}^{m-p-1} = \binom{n+m}{m-p-1}.$$

Інакше, тобто при $m - p - 1 < 0$, що еквівалентно $m \le p$, ця кількість дорівнює 0. Остаточно маємо:

- при $m \le p$ шукана кількість розташувань дорівнює $C_{n+m}^{n} = \binom{n+m}{n}$;
- при $m > p$ шукана кількість розташувань дорівнює

$$C_{n+m}^{n} - C_{n+m}^{m-p-1} = \binom{n+m}{m} - \binom{n+m}{m-p-1}.$$

Таким чином, задачу зведено до обчислення біномних коефіцієнтів:

$$C_{n+m}^{n} = \frac{(m+n)!}{m!n!} = \frac{(m+n)(m+n-1)...(n+1)}{m!} = \frac{(m+n)(m+n-1)...(m+1)}{n!},$$

де з останніх двох виразів бажано вибрати той, що має менший знаменник. Потім, використавши алгоритм Евкліда, скоротити на найбільші спільні дільники множники чисельника та знаменника, щоб *подати знаменник добутком одиниць* і знайти чисельник. При $r > 0$ після кожного множення потрібно замінювати добуток на лишок від ділення на r, щоб не вийти не межі

базового типу (int64 для Pascal).

6. Гра

Свого часу цю задачу для випадку $g = 2$ і максимального номеру фішки 17 було запропоновано на III (міському) етапі олімпіади у 2004 році. Задачу з такими обмеженнями можна було розв'язати, використавши аналіз графа гри «з кінця». Але для запропонованих у даній умові обмежень щодо кількості фішок та часу виконання такий підхід не годиться для більшості тестів.

Будь-яка позиція розглядуваної гри є об'єднанням рядів послідовних фішок (без розриву). Інакше кажучи, гра з довільної позиції є сумою ігор з ряду послідовних фішок. Це явна вказівка на те, що повне розв'язання передбачає використання чисел Шпраґе — Гранді (оцінок позицій, що дорівнюють нулю для програшних позицій). В авторському розв'язанні їх знайдено для рядів послідовних фішок при зростанні довжини ряду:

- якщо довжина ряду не перевищує g, то вона і є числом Шпраґе — Гранді для позиції, що містить лише цей ряд;
- для решти позицій їхню оцінку потрібно знайти, керуючись тим, що:
    - *оцінка позиції — це найменше невід'ємне ціле число, відмінне від оцінок позицій, у які є хід з даної позиції;*
    - *оцінка суми позицій є побітною сумою* (без перенесень у старші розряди, див. операцію xor для мови Pascal) *оцінок позицій-доданків*.

Позиція є програшною тоді й лише тоді, коли її оцінка дорівнює нулю. Детальний і замкнений виклад теорії опубліковано, наприклад, на сайті «Київські учнівські олімпіади з інформатики».

6. Авторські розв'язання завдань III етапу

1. Дивовижне число

```
/* GCC */

#include<stdio.h>
```

```
intmain()
{
freopen("awesome.in", "r", stdin);
freopen("awesome.out", "w", stdout);

intd, s;
scanf("%d", &d);
    s = d; // s позначатиме, скільки зали-
    // шилося добрати, щоб вийшла правильна
    // сума цифр, а d — добуток

intdigits[4] = {7, 5, 3, 2}; // Пере-
    // бираємо цифри саме в такому порядку
intcounts[4] = {0, 0, 0, 0}; // Кіль-
    // кості відповідно цифр 7, 5, 3, 2,
    // які рахуємо далі

for(inti = 0; i <4; i++)
while(d % digits[i] == 0) // Поки
        // d ділиться на поточну цифру:
{
            d /= digits[i]; // Ділимо d на
                            // поточну цифру
s -= digits[i]; // Корегуємо
                  // значення суми цифр,
                  // яку залишилося добрати
counts[i]++; // Збільшуємо лі-
                     // чильник відповідної
                     // цифри на 1
}

if(d >1) // Якщо виявилося, що в чис-
      // ла є простий дільник, більший за 7
printf("0");
else
{
for(inti = 0; i <4; i++) // Ви-
```

```
        // водимо цифри 7, 5, 3, 2
        // відповідну кількість разів
for(intj = 0; j <counts[i];
                            j++)
printf("%d", digits[i]);
for(intj = 0; j < s; j++) // До-
printf("1"); // повнюємо число
            // одиницями так, щоб сума цифр
            // вийшла правильною
}

printf("\n");
return0;
}
```

2. Черевички на підборах

```
/* GCC */

#definemaxN200000 // Найбільше можливе
                  // значення n
#include<stdio.h>
#include<algorithm>// Ця бібліотека
              // містить функцію сортування

inta[maxN], b[maxN]; // Перша та друга
// послідовності. Масиви оголошено поза
// тілом програми, щоб уникнути
// переповнення стека

intmain()
{
freopen("heels.in", "r", stdin);
freopen("heels.out", "w", stdout);

intn;
```

```
// Зчитуємо дані:
scanf("%d", &n);
for(inti = 0; i < n; i++)
scanf("%d", &a[i]);
for(inti = 0; i < n; i++)
scanf("%d", &b[i]);

std::sort(a, a + n); // Сортуємо всі
                     // члени першої послідовності
std::sort(b, b + n); // Сортуємо всі
                     // члени другої послідовності

intanswer = 0; // Лічильник-відповідь
intac = 0, bc = 0; // Вказівники на
       // поточні елементи першої та другої
       // послідовностей відповідно
while(ac< n &&bc< n) // Поки ми не
    // вийшли за межі жодної з двох
    // послідовностей:
{
// Якщо поточні члени різні, пере-
        // ходимо до наступного в тій пос-
        // лідовності, де член менший, а
        // інакше додаємо до лічильника-
        // відповіді одиницю і переходимо
        // до наступних членів в обох
        // послідовностях:
if(a[ac] < b[bc])
ac++;
elseif(a[ac] > b[bc])
bc++;
else
{
answer++;
ac++;
bc++;
        }
```

```
    }

// Виводимо відповідь:
printf("%d\n", answer);

return0;
}
```

3. Два квадрати

```
/* GCC */

#definemaxN1000 // Найбільше можливе
                // значення n
#include<stdio.h>

boola[maxN + 2][maxN + 2];
// Беремо аркуш з «полями» по одній клітин-
// ці в кожен бік (так зручніше буде реалі-
// зовувати різноманітні перевірки).
// a[i][j] == true означатиме,
// що клітинка зафарбована

intmain()
{
freopen("squares.in", "r", stdin);
freopen("squares.out", "w", stdout);

intn;
scanf("%d\n", &n);

// Заповнюємо «поля» навколо аркуша
    // значенням «незафарбовано»:
for(inti = 0; i < n + 2; i++)
        a[0][i] = a[i][0] = a[n + 1][i] =
                a[i][n + 1] = false;
```

```
// Зчитуємо сам аркуш:
for(inty = 1; y <= n; y++) // Рядки —
         // це координата y, а стовпці — x
for(intx = 1; x <= n; x++)
        {
charc;
scanf(x < n ? "%c "
: "%c\n", &c);
// Не забуваємо зчитувати
            // пробіл або перенесення
            // рядка після кожного символу

a[x][y] = (c == '#'); // Кліт-
            // инка зафарбована,
            // якщо відповідний символ — #
}

// Шукаємо найлівішу з клітинок, що є
    // лівими верхніми вершинами квадратів:
ints1_left = 0, s1_top = 0; // У май-
    // бутньому — ліва та верхня межі пер-
    // шого квадрата (координати лівої вер-
    // хньої вершини). Нулі — індикатор то-
    // го, що клітинку поки що не знайдено
for(intx = 1; x <= n; x++) // Ідемо
{                     // зліва направо:
for(inty = 1; y <= n; y++)
// Ідемо зверху вниз:
if(a[x][y] && ! a[x - 1][y]
&& ! a[x][y - 1])
            { // Якщо клітинку з відповід-
              // ними властивостями знайде-
              // но, запам'ятовуємо її
              // та виходимо з циклу:
s1_left = x;
                s1_top = y;
break;
```

```
            }
if(s1_left >0) // Якщо клітинку
            // знайдено, виходимо також
            // і з зовнішнього циклу
break;
    }

// Визначаємо ліву або верхню сторону
    // квадрата з вершиною у знайденій
    // клітинці:
ints1_right = s1_left,
        s1_bottom = s1_top; // У майбутньо-
        // му — права та нижня межі першого
        // квадрата (координати правої ниж-
        // ньої вершини)
while(a[s1_right + 1][s1_top] &&
           a[s1_left][s1_bottom + 1])
    { // Поки і праворуч, і вниз клітинки
      // чорні, продовжуємо «розширювати»
      // квадрат:
s1_right++;
        s1_bottom++;
    }

// Визначаємо, чи є чорні клітинки
    // строго всередині та строго зовні
    // квадрата:
boolinternal = false; // Чи є клітинки
                          // всередині
boolexternal = false; // Чи є клітинки
                          // зовні
for(intx = 1; x <= n; x++) // Прохо-
          // димо по всіх клітинках аркуша:
for(inty = 1; y <= n; y++)
if(a[x][y]) // Якщо клітинка
{           // чорна:
if(x > s1_left &&
```

```
                    x < s1_right &&
                    y > s1_top &&
                    y < s1_bottom)
internal = true;
// Клітинка лежить стро-
                  // го всередині квадрата
elseif(x < s1_left ||
                         x > s1_right ||
                         y < s1_top ||
                         y > s1_bottom)
external = true;
// Клітинка лежить стро-
                  // го зовні квадрата
}

ints2_left, s2_top,
        s2_right, s2_bottom; // У майбут-
        // ньому — ліва, верхня, права та
        // нижня межі другого квадрата
        // (координати відповідних двох
        // вершин цього квадрата)
if(external) // Якщо є зафарбовані
         // клітинки поза першим квадратом:
{
for(intx = 1; x <= n; x++)
        {
boolfound = false; // Чи знай-
            // дено вже клітинку, що є вер-
            // шиною другого квадрата, яка
            // лежить поза межами першого,
            // а отже, визначено розташу-
            // вання другого квадрата
for(inty = 1; y <= n; y++)
if(a[x][y] && (x < s1_left
                         || x > s1_right
                         || y < s1_top
                         || y > s1_bottom))
```

```
                { // Якщо ми натрапили на
                 // чорну клітинку поза ме-
                 // жами першого квадрата:
boolisTopLeft =
                        ! a[x - 1][y] &&
                        ! a[x][y - 1];
// Чи є вона лівою
                       // верхньою вершиною
boolisBottomRight =
                        ! a[x + 1][y] &&
                        ! a[x][y + 1];
// Чи є вона правою
                       // нижньою вершиною
                    // Решту випадків можна
                    // не розглядати, так
                    // само як не слід хви-
                    // люватися через те,
                    // що зліва чи зверху
                    // від лівої верхньої
                    // вершини або справа
                    // чи знизу від правої
                    // нижньої вершини буде
                    // зафарбована клітинка
                    // першого квадрата,
                    // яка завадить розпіз-
                    // нати вершину

if(isTopLeft ||
isBottomRight)
                    {
// Ініціалізуємо
                        // межі другого
                        // квадрата:
s2_left =
                          s2_right = x;
                        s2_top =
                          s2_bottom = y;
```

```
found = true;
                    }

// Ідемо вздовж сторони
                    // другого квадрата, що
                    // точно не перетинає-
                    // ться (і не дотикає-
                    // ться) до першого
                    // квадрата, поки не
                    // дійдемо до незафар-
                    // бованої клітинки.
                    // Паралельно розширює-
                    // мо і другий вимір
                    // квадрата:
if(isTopLeft)
while(x < s1_left
              ? a[s2_left][s2_bottom + 1]
              : a[s2_right + 1][s2_top])
                        {
                            s2_right++;
                            s2_bottom++;
                        }
elseif(isBottomRight)
while(x > s1_right
              ? a[s2_right][s2_top - 1]
              : a[s2_left - 1][s2_bottom])
                        {
                            s2_left--;
                            s2_top--;
                        }

if(found) // Якщо роз-
                     // ташування другого
                     // квадрата визначено,
                     // виходимо з циклу
break;
```

```
                }
if(found) // Якщо розташування
                 // вже визначено, виходимо
                 // і з зовнішнього циклу
break;
        }
    }
elseif(internal) // Якщо нема зафар-
{ // бованих клітинок поза межами пер-
      // шого квадрата, але є такі клітини
      // строго всередині нього:
      // Визначаємо ліву, верхню, праву та
      // нижню межу зафарбованих клітинок,
      // що містяться строго всередині
      // першого квадрата:
s2_left = s1_right; // Ініціаліза-
s2_top = s1_bottom; // ція значень
s2_right = s1_left;
        s2_bottom = s1_top;
// Проходимо по всіх внутрішніх
        // клітинках квадрата:
for(intx = s1_left + 1;
                 x <= s1_right - 1; x++)
for(inty = s1_top + 1;
                 y <= s1_bottom - 1; y++)
if(a[x][y])
                { // Якщо клітинка чорна,
                  // оновлюємо межі:
if(x < s2_left)
s2_left = x;
if(y < s2_top)
s2_top = y;
if(x > s2_right)
s2_right = x;
if(y > s2_bottom)
s2_bottom = y;
                }
```

```
// Включаємо верхню межу першого
        // квадрата до другого квадрата,
        // якщо на наступній горизонталі є
        // хоча б одна чорна клітинка, але
        // немає двох сусідніх:
if(s2_top == s1_top + 1) // Якщо
{ // на наступній горизонталі
          // є чорна клітинка:
s2_top = s1_top; // Відразу
            // включаємо межу, однак далі,
            // можливо, виключимо:
for(intx = s1_left + 1;
                    x < s1_right - 1; x++)
if(a[x][s1_top + 1] &&
                   a[x + 1][s1_top + 1])
                { // Якщо знайшлися дві су-
                  // сідні чорні клітинки,
                  // виключаємо межу:
s2_top = s1_top + 1;
break;
                }
        }

// Аналогічно для нижньої межі:
if(s2_bottom == s1_bottom - 1)
        {
            s2_bottom = s1_bottom;
for(intx = s1_left + 1;
                    x < s1_right - 1; x++)
if(a[x][s1_bottom - 1] &&
                  a[x + 1][s1_bottom - 1])
                {
                    s2_bottom =
                      s1_bottom - 1;
break;
                }
```

```
        }

// Аналогічно для лівої межі:
if(s2_left == s1_left + 1)
        {
            s2_left = s1_left;
for(inty = s1_top + 1;
                    y < s1_bottom - 1; y++)
if(a[s1_left + 1][y] &&
                    a[s1_left + 1][y + 1])
                {
                    s2_left = s1_left + 1;
break;
                }
        }

// Аналогічно для правої межі:
if(s2_right == s1_right - 1)
        {
            s2_right = s1_right;
for(inty = s1_top + 1;
                    y < s1_bottom - 1; y++)
if(a[s1_right - 1][y] &&
                    a[s1_right - 1][y + 1])
                {
                    s2_right =
                      s1_right - 1;
break;
                }
        }
    } else// Якщо крім клітинок першого
{ // квадрата немає ніяких зафарбованих
      // клітин:
        // Просто копіюємо перший квадрат
        // у другий:
s2_left = s1_left;
        s2_top = s1_top;
```

```
        s2_right = s1_right;
        s2_bottom = s1_bottom;
    }

intо[2][4] = { // Рядки чисел, які
    // потрібно вивести в лексикографічному
    // порядку
{s1_left, s1_top,
         s1_right, s1_bottom},
        {s2_left, s2_top,
         s2_right, s2_bottom}
    };

booldirectOrder = true; // Виводити
          // рядки у прямому порядку (true)
          // чи в оберненому (false)
    // Визначаємо, в якому ж порядку
    // потрібно вивести рядки:
for(inti = 0; i <4; i++)
if(o[0][i] > o[1][i]) // Рядки
        // треба вивести в оберненому
        // порядку
{
directOrder = false;
break;
        } elseif(o[0][i] < o[1][i])
// Рядки треба вивести
        // у прямому порядку
break;

for(introw = 0; row<2; row++)
// Номер рядка
for(inti = 0; i <4; i++)
// Номер числа в рядку
            // Виводимо число з відповідно-
            // го рядка залежно від потріб-
            // ного порядку рядків:
```

```
printf(i <3 ? "%d " : "%d\n",
                  o[directOrder ? row
                           : 1 - row][i]);

return0;
}
```

4. Схил

```
{ Free Pascal }
var x,y: array[1..100000] of integer;
     {Координати точок з цвяхами.}
    xc,yc:array[1..65536] of integer;
     {Запасні масиви, які використовуються
      при сортуванні методом злиття.}
    x1,y1,x2,y2:integer; {Поточне значення
      пари точок з найменшим схилом.}
    i,j,i2,j2:longint; {Індекси для
      циклів.}
    n:longint; {Кількість точок з цвяхами.}

{Впорядковування точок за зростанням
 y-координати, а при рівних y-координатах -
 за зростанням x-координати. Сортування
 виконується методом злиття. Координати
 точок, що сортуються, розташовані в
 масивах x та y.}
PROCEDURE Sort;
  var i:longint; {Номер елемента, після
       якого починається перший з двох
       відрізків, які зливаються.}
      j,k:longint; {Індекси елементів, що
       порівнюються у процесі злиття
       відрізків.}
      kb:longint; {Верхня межа для k.}
      m:longint; {Довжина відрізків, які
       зливаються.}
```

```
Begin
  m:=1; {Початкова довжина відрізків, що
    зливається, встановлюється рівною 1.}
  {Цикл повторюватиметься, поки довжина
   відрізків, що зливаються, менша за
   довжини масивів.}
  while (m<n) do begin
    i:=0; {Встановлюємо, що перший
      відрізок розташований одразу після
      0-оїкомірки. Тобто цей відрізок
      починається з елемента, який має
      індекс 1.}
    while (i+m<n) do begin
      {Копіюємо перший з відрізків, що
       зливаються, до масивів xc та yc.}
      for j:=1 to m do begin
        xc[j]:=x[i+j];
        yc[j]:=y[i+j];
      end;
      j:=1; {Встановлюємо значення
        біжучого індекса у першому
        відрізку.}
      k:=i+m+1; {Встановлюємо значення
        біжучого індекса у другому
        відрізку.}
      {Визначаємо верхню межу для k,
       рівну min(i+m*2,n).}
      kb:=i+m*2;
      if (kb>n) then kb:=n;
      {Повторюємо порівняння, поки не
       закінчиться один з відрізків, що
       зливаються. Після кожного
       порівняння менший елемент
       переносимо у основний масив, а
       відповідний біжучий індекс
       збільшуємо на 1.}
      while ((j<=m)and(k<=kb)) do begin
```

```
          if ((yc[j]<y[k])or((yc[j]=y[k])
              and(xc[j]<=x[k]))) then begin
             x[j+k-m-1]:=xc[j];
             y[j+k-m-1]:=yc[j];
             Inc(j);
          end;
          if (j<=m) then
             if ((yc[j]>y[k])or
                 ((yc[j]=y[k])and
                 (xc[j]>x[k]))) then begin
                x[j+k-m-1]:=x[k];
                y[j+k-m-1]:=y[k];
                Inc(k);
             end;
        end;
        {Якщо у першому відрізку залишилися
         елементи, переносимо їх до
         основного масиву. Якщо елементи
         залишилися у другому відрізку,
         нічого робити не потрібно. :) У
         згаданому випадку елементи другого
         відрізка вже знаходяться на
         потрібних місцях після закінчення
         циклу.}
        for j:=j to m do begin
          x[j+k-m-1]:=xc[j];
          y[j+k-m-1]:=yc[j];
        end;
        i:=i+m*2; {Переходимо до наступних
          двох відрізків.}
      end;
      m:=m*2; {Подвоюємо довжину відрізків,
        які зливаються.}
    end;
  End;

{Порівняння схилів, заданих парою точок
```

```
 (x1,y1), (x2,y2) та парою (x[i],y[i]),
 (x[j],y[j]). Якщо друга пара дає більш
 пологий схил, то точкам першої пари
 присвоюється значення точок другої пари.}
PROCEDURE Compare(var x1,y1,x2,y2:integer;
                  i,j:longint);
  var p,q:longint; {Змінні, потрібні для
        того, щоб добутки різниць координат
        мали тип longint, а не integer. У
        протилежному випадку можлива
        помилка через перевищення
        максимально допустимого значення,
        бо вказаний добуток може сягати
        900000000.}
  Begin
    p:=abs(x2-x1);
    q:=abs(y2-y1);
    if (p*abs(y[i]-y[j])<q*abs(x[i]-x[j]))
    then begin
       x1:=x[i];
       y1:=y[i];
       x2:=x[j];
       y2:=y[j];
    end;
  End;

BEGIN
  {Відкриття файлів для вводу та виводу.}
  assign(input,'slope.in');
  assign(output,'slope.out');
  reset(input);
  rewrite(output);
  {Ввід вхідних даних.}
  read(n);
  for i:=1 to n do read(x[i],y[i]);
  Sort; {Сортування точок.}
  i:=1;
```

```
j:=1;
{Пошук найправішої точки, яка має таку ж
 y-координату, що й (x[i],y[i]).}
while ((j<n)and(y[j+1]=y[i])) do Inc(j);
if (j=n) then writeln(0); {Якщо всі точки
  лежать на одній горизонталі, на вивід
  подається число 0.}
if (j<n) then begin
   {Присвоєння точкам (x1,y1) та (x2,y2)
    значень, які претендують на те, щоб
    задавати найпологіший схил.}
   x1:=x[i];
   y1:=y[i];
   x2:=x[j+1];
   y2:=y[j+1];
   while (j<n) do begin
     i2:=j+1;
     j2:=i2;
     {Пошук найправішої точки, яка має
      таку ж y-координату, що й
      (x[i2],y[i2]).}
     while ((j2<n)and(y[j2+1]=y[i2])) do
       Inc(j2);
     {Порівняння пари точок (x1,y1) та
      (x2,y2) з іншими претендентами.}
     Compare(x1,y1,x2,y2,i,i2);
     Compare(x1,y1,x2,y2,i,j2);
     Compare(x1,y1,x2,y2,j,i2);
     Compare(x1,y1,x2,y2,j,j2);
     i:=i2;
     j:=j2;
   end;
   write(x1,' ',y1,' ',x2,' ',y2); {Вивід
     найкращої пари точок.}
end;
{Закриття файлів вводу та виводу.}
close(input);
```

```
  close(output);
END.
```

5. Каса

```
{ Free Pascal }
vari,m,n,p: longint;
o: text;      d,r: int64;
a,b: array[1..32665] of word;
functiongcd(a,b: word): word;
varx,y,z: word;     BEGIN
if (a=0) or (b=0) then gcd:=abs(a+b)
else       begin
x:=abs(a);
y:=abs(b);
repeat z:=x mod y;  x:=y;  y:=z
until z=0;
gcd:=x                    end END;

function c(k,m: integer):   int64;
varl,j: integer; d: int64;BEGIN
if (k<0) or (m<k) then c:=0 else
if (k=0) or (m=k) then c:=1 else  begin
if k<m-k then l:=k
else l:=m-k;
for j:=1 to l do     begin
a[j]:=m-j+1;
b[j]:=j               end;
for j:=2 to l do     begin
i:=0;
repeatinc(i);
d:=gcd(a[i],b[j]);
if d>1 then      begin
a[i]:=a[i] div d;
b[j]:=b[j] div d end
until   1=b[j]       end;
d:=a[1];
if r=0
```

```
then for j:=2 to l do d:= d*a[j]
else for j:=2 to l do d:=(d*a[j]) mod r;
c:=d                              end END;
                                    BEGIN
assign(o,'cash.in');
reset(o);
read (o,m,n,p,r);
close(o);
assign(o,'cash.out');
rewrite(o);
d:=c(m,m+n)-c(m-1-p,n+m);
while d<0 do d:=d+r;
writeln(o,d);
close(o)                           END.
```

6. Гра

```
{ Free Pascal }
const n_=255; l_=255;
type ab = array[0..n_] of byte;
     aw = array[0..n_] of word;
var a,b: aw;
   {кінці послідовностей фішок без розривів}

c,{оцінки послідовностей фішок без розривів}
d:ab;    {лишки від ділення на 2 кількостей
          послідовностей фішок без розривів}
yet: boolean;      {чи був запис у рядку?}
    o: text;
    g,           {найбільша кількість фішок,
                  яку можна взяти за 1 хід}
sg: byte;               {оцінка позиції}
     l,      {кількість послідовостей фішок}
     m,    {найбільша довжина послідовності}
    i,j,k,n: word;
    s: set of byte; {множина оцінок позицій,
у які є хід з позиції - послідовності фішок}
```

```
                                        BEGIN
assign(o,'game.in');
reset(o);
read (o,g);
k:=0;
l:=1;
read(o,a[1]);
b[1]:=a[1];
while not seekeof(o) do begin
inc(k);
read(o,j);
if a[l]+k=j then b[l]:=j
else begin
             k:=0;
inc(l);
             a[l]:=j;
             b[l]:=j end end;
close(o);
assign(o,'game.out');
rewrite(o);
m:=b[1]-a[1]+1;
for j:=2 to l do begin
  k:=b[j]-a[j]+1;
if m<k then m:=k end;
for j:=1 to m do d[j]:=0;
for j:=1 to l do   begin
i:=b[j]-a[j]+1;
d[i]:=(d[i]+1) mod 2 end;

{Оцінки послідовностей фішок без пропусків}
for j:=0   to g do c[j]:=j;
for j:=g+1 to m do begin
s:=[];
for k:=1 to g do           begin
include(s,c[j-k]);
for i:=1 to ((j-k) div 2) do
include(s,c[i] xor c[j-i-k]) end;
```

```
k:=0;  while k in s do inc(k);
c[j]:=k                              end;

                    {Оцінювання позиції}
sg:=0;
for j:=1 to m do
if d[j]=1 then sg:=sg xor c[j];

                        {Запис відповіді}
if sg=0 then for j:=1 to g do writeln(o)
else
for j:=1 to g do begin
yet:=false;
for k:=1 to l do if b[k]-a[k]+1>=j then
for n:=a[k] to b[k]-j+1 do begin
if (sg xor c[b[k]-a[k]+1]
xor c[n-a[k]]
xor c[b[k]-(n+j-1)])=0 then begin
if yet then write(o,' ');
write(o,n);
yet:=true end; end;
writeln(o)                               end;
close(o)                                 END.
```