Юрій Володимирович Знов'як,

інженер з програмного забезпечення Google New York Center

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 2012–2013 навчальному році

Передмова

Стаття містить умови орієнтовних завдань II (районного) і завдань III (міського) етапу олімпіади з основ інформатики й обчислювальної техніки у місті Києві у 2012–2013 навчальному 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 орієнтовні завдання II етапу (упорядник — Данило Мисак), на відміну від попередніх років, не містили завдань відбірково-тренувальних зборів і були краще узгоджені з рівнем підготовки переважної кількості учасників олімпіади. Розглянемо їх детально.

1. Завдання II етапу

1. Петрик П’яточкін рахує складú

Назва програми: syllable.*

Умова

Допитливий київський школяр Петрик П’яточкін якось зацікавився мовознавством — наукою про мову. Для важливого дослідження Петрику потрібно з’ясувати, скільки на́звви різних натуральних чисел мають складів, тобто скільки містять голосних літер. Допоможіть хлопцю, написавши програму, яка дає відповідь на це питання.

Вхідні дані

У вхідному файлі вказано натуральне число *n*, кількість складів у якому необхідно порахувати. Число *n* не перевищує 100.

Вихідні дані

У вихідний файл виведіть кількість складів, які у своїй назві українською мовою має число *n*.

Приклади

№	Вхідний файл syllable.in	Вихідний файл syllable.out
1	1	2
2	25	3

Пояснення до прикладів

Слово «один» має два склади (містить дві голосні літери). Назва числа 25 — «двадцять п’ять» — має три склади (містить три голосні літери).

Ідея розв’язання

Можна вручну підрахувати кількість складів у кожному з чисел від 1 до 100 і створити програму з розгалуженнями типу IF THEN ELSE або CASE /

SWITCH, які залежно від числа на вході повертають потрібний результат. Але цей підхід досить трудомісткий і скоріше за все учасник, який пішов таким шляхом, зробить принаймні декілька помилок у підрахунку або під час занесення результатів у програму.

Натомість можна помітити, що назви всіх двоцифрових чисел, починаючи з числа 20, утворюються шляхом сполучення слова, що позначає кількість десятків («двадцять», «сорок», «дев'яносто») і слова, що позначає кількість одиниць («один», «вісім»). При цьому у назвах чисел, кратних 10, компонент одиниць відсутній. Тому слід окремо внести у програму (наприклад, як масив) кількості складів у назвах чисел 1, 2, 3, …, 18, 19, 20, 30, 40, …, 90, 100, а решту відповідей давати як суму двох значень, занесених для числа, що позначає кількість десятків, і числа, що позначає кількість одиниць заданого у вхідному файлі значення *n*.

2. Петрик П'яточкін купує книги

Назва програми: books.*

Умова

Щоб краще підготуватися до різноманітних олімпіад, у яких бере участь Петрик П'яточкін, хлопець замовляє книги у зарубіжному інтернет-магазині. На жаль, Петрику доводиться платити не лише за самі книги, але також і за їх доставку в Україну: незалежно від кількості книг, доставлених хлопцю за один раз, Петрик платить за одну доставку *d* гривень. Крім того, українська митниця за один раз дозволяє безплатно провозити через кордон щонайбільше *m* книг, а якщо кількість книг перевищує *m*, митниця бере за кожну понаднормово завезену книгу *t* гривень податку. Перед олімпіадою з інформатики Петрик хоче купити в інтернет-магазині *n* книг. Він може замовити доставку всіх книг разом або як завгодно розподілити книги на довільну кількість окремих доставок. Допоможіть хлопцю визначити, яку найменшу загальну суму грошей він повинен витратити на доставки та на мито, щоб перевезти усі *n* книг в Україну.

Вхідні дані

У єдиному рядку вхідного файла записані чотири натуральні числа *n, d, m, t* — відповідно кількість книг, які треба перевезти; вартість однієї доставки; найбільша кількість книг, яку можна перевезти за одну доставку без сплати мита; розмір мита за кожну понаднормово завезену книгу. Відомо, що $n(d + t) < 2 \cdot 10^9$ і $m < 2 \cdot 10^9$.

Вихідні дані

У вихідний файл виведіть єдине натуральне число — найменшу сумарну кількість грошей, яку має сплатити Петрик за доставку і як мито, щоб отримати замовлені в інтернет-магазині книги.

Приклади

№	Вхідний файл books.in	Вихідний файл books.out
1	5 4 3 6	8
2	3 2 2 1	3

Пояснення до прикладів

У першому прикладі потрібно перевезти 5 книг. Якщо замовити їх однією доставкою, доведеться сплатити 4 грн за доставку і за 5 − 3 = 2 понаднормово завезені книги по 6 грн, усього 16 грн. Якщо ж розподілити книги на дві доставки, наприклад 3 книги на першу і 2 книги на другу, треба буде сплатити 2 · 4 = 8 грн за дві доставки, зате під час жодної з доставок платити за понаднормово завезені книги не доведеться. Три чи більше доставок будуть коштувати явно більше за 8 грн, тому 8 грн — найменша сума, яку доведеться витратити.

У другому прикладі потрібно перевезти 3 книги. Якщо замовити їх однією доставкою, треба буде сплатити 2 + (3 − 2) · 1 = 3 грн. А за дві чи більше доставок доведеться заплатити вже не менше ніж 2 · 2 = 4 грн.

Ідея розв'язання

Якщо Петрик замовить k окремих доставок, то за умови оптимального розподілу книг між доставками хлопцю доведеться сплатити $kd + \max\{0, n-km\}\cdot t$ грн: kd грн за k доставок, а також $(n-km)t$ грн за $n-km$ понаднормово завезених книг, якщо $n-km>0.$ Запровадьмо таке позначення:

$$f(k) = kd + \max\{0, n-km\}\cdot t,\ k \in \square,$$

і розгляньмо різницю $f(k+1)-f(k)$:

$$f(k+1)-f(k) = \begin{cases} d, & n-km<0, \\ d-(n-km)t, & 0 \,,, n-km<m, \\ d-mt, & n-km \ldots m. \end{cases}$$

Запишімо це трохи інакше, позначивши через $n \bmod m$ остачу від ділення n на m:

$$f(k+1)-f(k) = \begin{cases} d, & k>\frac{n}{m}, \\ d-(n \bmod m)t, & \frac{n}{m}-1<k \,,, \frac{n}{m}, \\ d-mt, & k \,,, \frac{n}{m}-1. \end{cases}$$

Зрозуміло, що якщо $\frac{n}{m}<1$, тобто $n<m$, то замовляти більше ніж одну доставку Петрику сенсу немає. Тому відповіддю в цьому випадку є значення $f(1).$ Далі вважатимемо, що $\frac{n}{m} \ldots 1.$

Якщо $d-mt \ldots 0$, то $f(k+1) \ldots f(k)$ незалежно від величини k. Тому мінімальне значення функція f набуває за найменшого k, тобто при $k=1$, а шукана сума грошей — це $f(1).$

Якщо $d-mt<0$, але $d-(n \bmod m)t>0$, то $f(k+1)<f(k)$ при $k \,,, \left[\frac{n}{m}\right]-1$ та $f(k+1)>f(k)$ при $k \ldots \left[\frac{n}{m}\right]$. Тому мінімальну величину функція f набуває при $k=\left[\frac{n}{m}\right]$, а шукана сума грошей — це $f\left(\left[\frac{n}{m}\right]\right).$

А якщо $d-(n \bmod m)t \geq 0$, то $f(k+1) \geq f(k)$ при $k \geq \left[\frac{n}{m}\right]$ та $f(k+1) > f(k)$ при $k < \left[\frac{n}{m}\right]+1$. Тому найменшу величину функція f набуває при $k = \left[\frac{n}{m}\right]+1$, а шукана сума грошей — це $f\left(\left[\frac{n}{m}\right]+1\right)$.

На практиці досить написати програму, яка порівнює три числа — значення $f(1)$, $f\left(\left[\frac{n}{m}\right]+1\right)$ і, якщо $\left[\frac{n}{m}\right] \neq 0$, $f\left(\left[\frac{n}{m}\right]\right)$ — та виводить найменше з них. Усі обчислення слід проводити обережно, щоб не вийти за межі стандартного типу даних. Зокрема, число $f\left(\left[\frac{n}{m}\right]+1\right)$ може перевищувати $2 \cdot 10^9$, але в цьому випадку воно точно не буде найменшим із трьох чисел (бо, замовивши, приміром, одну доставку, Петрик заплатить менше ніж $d+nt$, $n(d+t) < 2 \cdot 10^9$ грн).

Алгоритм, який не враховує монотонності функції f при значеннях аргументу від 1 до $\left[\frac{n}{m}\right]$ та шукає найменше серед усіх чисел $f(1)$, $f(2)$, …, $f\left(\left[\frac{n}{m}\right]\right)$, $f\left(\left[\frac{n}{m}\right]+1\right)$, набирає лише частковий бал: значення $\left[\frac{n}{m}\right]$ попри обмеження, накладені в умові задачі, може досягати мільярда.

3. Петрик П’яточкін пише олімпіаду

Назва програми: olympiad.*

Умова

Прочитавши чимало книжок про алгоритми, Петрик П’яточкін прийшов на районну олімпіаду з інформатики. Перед початком олімпіади з’ясувалося, що багато хто з учасників знає одне одного. Тож організатори вирішили убезпечитися й розсадити всіх учасників по двох кабінетах, де проходить олімпіада, у такий спосіб, щоб жодні два знайомі між собою учасники не сиділи

в одному кабінеті. Напишіть програму, яка допоможе організаторам зробити це або скаже, що розсадити в такий спосіб учасників неможливо.

Вхідні дані

У першому рядку вхідного файла вказано два натуральних числа n та m — кількість учасників районної олімпіади та кількість пар знайомих учасників; $2 \le n < 2000$, $1 \le m < 450\,000$. У кожному з наступних m рядків задано по два числа a_i, b_i — номери знайомих між собою учасників, $1 \le a_i < b_i \le n$, $1 \le i \le m$. Жодна пара номерів a_i, b_i у вхідному файлі не повторюється. Крім того, вхідні дані гарантують, що є не більше ніж один спосіб розсадити учасників по двох кабінетах, щоб жодні два знайомі учасники не сиділи в одному кабінеті.

Вихідні дані

У першому рядку вихідного файлу виведіть у порядку зростання номери учасників, які мають сидіти в кабінеті разом із учасником під номером 1 (Петриком П’яточкіним) включно із самим числом 1 (Петриком).

У другому рядку виведіть у порядку зростання номери учасників, які повинні сидіти в іншому кабінеті.

Якщо жодне розташування учасників не задовольняє умову задачі, в обох рядках виведіть по нулю.

Приклади

Вхідний файл olympiad.in	**Вихідний файл olympiad.out**
5 6 2 5 2 3 1 5 4 5 3 4 1 3	1 2 4 3 5
3 3 1 2 2 3 1 3	0 0

Ідея розв'язання

Задачу можна переформулювати в термінах теорії графів. Нехай учасники олімпіади — це вершини графа, а ребро між двома вершинами проведене тоді й лише тоді, коли два відповідні учасники знайомі між собою. Нам необхідно розділити всі вершини графа на дві частини так, щоб кожне ребро графа сполучало вершини, що належать до різних частин. Граф, для якого це вдається зробити, називається дводольним, а відповідні його частини — долями.

Задачу можна розв'язати з допомогою пошуку у глибину або в ширину. Надавши вершині 1 (Петрику) кабінет № 1 і запустивши будь-який із типів пошуку з цієї вершини, будемо кожній новій пройденій вершині графа надавати кабінет № 2, якщо ми прийшли у неї з вершини, якій було присвоєно кабінет № 1, і навпаки. При цьому, якщо дана вершина сполучена ребром хоча б з однією іншою вершиною, якій раніше надали той самий кабінет, що й даній вершині, то розбити граф на дві частини неможливо.

Після завершення пошуку у глибину або в ширину, якщо вдалося надати кабінети всім вершинам, маємо потрібний розподіл учасників у компоненті зв'язності графа, яка містить вершину 1, а інакше виводимо нулі. Якщо граф складається з єдиної компоненти зв'язності, виводимо знайдений розподіл. Якщо ж у графі є дві або більше компонент зв'язності, тобто якщо після завершення пошуку, запущеного з вершини 1, одна чи кілька вершин залишились невідвіданими, то слід також вивести нулі. Справді: якщо потрібний розподіл вершин усіх компонент зв'язності існує, то він не єдиний, адже розподіли різних компонент можна як завгодно комбінувати. А згідно з умовою задачі, якби розподіл існував, то він мав би бути єдиним. Отже, у випадку двох і більше компонент зв'язності умова задачі гарантує відсутність потрібного розподілу учасників.

Насамкінець зауважимо, що обмеження $m < 450\,000$ має суто технічний характер і пов'язане з необхідністю обмежити розмір вхідного файлу.

Примітка. Решту матеріалів II етапу олімпіади можна знайти на сайтах «Київські учнівські олімпіади з інформатики» http://kievoi.narod.ru і на сайті олімпіади Солом'янського району http://soi.org.ua. У тексті подано варіант

третьої задачі, запропонований у Солом'янському й Печерському районах, він незначно відрізняється від розміщеного на сайті київських олімпіад.

2. Умови завдань III етапу

1. Поліклініка

Максимальна оцінка: 100 балів

Обмеження на час: 0,25 сек.

Обмеження на пам'ять: 250 МБ

Вхідний файл: clinic.in

Вихідний файл: clinic.out

Програма: clinic.*

На прийом до лікаря щодня приходить чимало людей. Кожен пацієнт перебуває на прийомі цілу кількість хвилин, але різних пацієнтів лікар може приймати різну кількість часу. Лікар починає прийом у момент часу t_1 хвилин і закінчує прийом у момент часу t_2 хвилин. Це означає, що будь-який пацієнт незалежно від того, скільки часу його прийматиме лікар, може зайти на прийом у моменти t_1, $t_1 + 1$, …, $t_2 - 1$. Заходити на прийом до лікаря в інший час або тоді, коли лікар приймає іншого пацієнта, заборонено. Якщо пацієнт приходить у поліклініку в момент t, він чекає на перший момент часу $s \geq t$ такий, що на цей момент лікар веде прийом, причому вже встиг оглянути всіх пацієнтів, які прийшли у поліклініку раніше, тобто до моменту t. Якщо лікар не встигає оглянути всіх до кінця прийому, решта пацієнтів має прийти наступного дня.

Завдання

Знаючи, в який момент лікар починає та закінчує прийом, те, хто й коли прийде на прийом у конкретний день, а також скільки часу оглядатиме кожного пацієнта лікар, визначте момент часу, в який потрібно прийти на прийом Петрику П'яточкіну, щоб гарантовано потрапити *в цей день* до лікаря, але при тому чекати на прийом якомога менше. У випадку, коли є кілька

альтернативних варіантів такого моменту часу, вам потрібно визначити найменший (найбільш ранній) із них.

Вхідні дані

У першому рядку вхідного файлу вказано три числа: кількість охочих потрапити на прийом n, час початку прийому t_1 і час завершення прийому t_2, що більший за t_1.

У другому рядку перераховані n чисел a_1, a_2, …, a_n — час, коли у поліклініку зайшли відповідно перший, другий, …, n-й охочий потрапити до лікаря. Числа a_1, a_2, …, a_n попарно різні й розташовані у порядку зростання.

У третьому рядку вхідного файлу перераховані n чисел b_1, b_2, …, b_n — час, необхідний лікарю на огляд відповідно першого, другого, …, n-го пацієнта.

Усі числа у вхідному файлі натуральні. Кількість пацієнтів n не більша за 10^5, решта чисел не перевищують 10^9.

Доба на планеті, де мешкає Петрик П’яточкін, триває значно довше, ніж на Землі, тому час початку прийому t_1, час завершення прийому t_2, а також числа a_1, a_2, …, a_n та b_1, b_2, …, b_n можуть бути більшими за 1440 — кількість хвилин у земній добі.

Вихідні дані

Вихідний файл повинен містити єдине натуральне число — найменший момент часу, коли Петрик П’яточкін має прийти в поліклініку, щоб гарантовано потрапити до лікаря, але прочекати на прийом якомога менше часу. Якщо Петрик прийде водночас з іншою людиною, його як молодшого пропустять уперед.

Приклади

№	clinic.in	clinic.out
1	3 10 20 7 14 18 5 2 1	17
2	5 10 20 4 9 12 16 22 4 10 10 9 2	9

3	1 10 20 5 15	5
4	1 10 20 15 15	10

Пояснення

В усіх чотирьох прикладах лікар запускає на прийом із 10-ї до 19-ї хвилини включно.

У прикладі 1 пацієнт, що прийшов у момент часу 7 хв, зайде до лікаря першим, тобто о 10-й хвилині, а вийде через 5 хвилин. Відразу після цього — о 15-й хвилині — в кабінет зайде пацієнт, що прийшов у момент часу 14 хв. Він вийде з кабінету через 2 хвилини, відразу після чого ні в кабінеті, ні в черзі нікого не буде. Отже, якщо Петрик П'яточкін прийде в поліклініку о 17-й хвилині, він не чекатиме ні хвилини. Зауважимо: якби Петрик прийшов о 18-й чи о 19-й хвилині, він би також не чекав. Але вивести потрібно 17, бо це перший з усіх моментів часу, коли Петрик може прийти, щоб не чекати на прийом.

У прикладі 2 Петрик повинен чекати не менше ніж 5 хвилин незалежно від того, коли він прийде в поліклініку. Найвигідніший варіант — прийти о 9-й хвилині, щоб зайти в кабінет другим о 14-й хвилині. У цьому випадку одночасно з хлопцем прийде ще один пацієнт. Але згідно з умовою задачі цей пацієнт пропустить Петрика вперед.

У прикладі 3 єдиний спосіб для Петрика потрапити до лікаря у перший день прийому — прийти не пізніше за іншого пацієнта.

У прикладі 4 хлопцю достатньо прийти у момент початку прийому — він одразу потрапить до лікаря.

2. Фарбування

Максимальна оцінка: 100 балів

Обмеження на час: 1 сек.

Обмеження на пам'ять: 32 МБ

Вхідний файл: farben.in

Вихідний файл: farben.out

Програма: farben.*

На заняттях шкільного гуртка Євгенію навчили виготовляти з паперу моделі правильних многогранників (платонових тіл) і напівправильних многогранників (архімедових тіл). Вона запам'ятала такі означення:

1. *Платоновим тілом називають (опуклий) многогранник, у якому всі грані — правильні однакові многокутники, а многогранні кути при всіх вершинах однакові.*
2. *Архімедовим тілом називають (опуклий) многогранник, у якому всі грані — правильні многокутники (не обов'язково з однаковою кількістю сторін), а многогранні кути при всіх вершинах однакові. При цьому є щонайменше дві різні грані.*

Таким чином, в архімедовому тілі грані кожного виду зустрічаються в тій самій кількості й у тій самій послідовності при обході навколо кожної вершини (чи для однакових, чи для протилежних напрямів обходу, якщо «дивитися ззовні»).

Кілька моделей многогранників Євгенія виготовила для оформлення кабінету математики і прихопила додому, щоб вразити своїх рідних. Враження, звичайно, вона справила. Але, вечеряючи, недогледіла, як молодша сестра Марічка взялася розфарбовувати грані: кожну грань лише однією фарбою. При цьому кілька граней могли бути й одного кольору, навіть якщо вони були суміжними, тобто мали спільне ребро. Євгенія задумалась… Її, як майбутнього математика, зацікавило питання: скількома способами можна розфарбувати грані тіла Архімеда, маючи певну кількість фарб, з точністю до *симетрій — рухів простору, що залишають многогранник на тому самому місці*. Інакше кажучи, коли не розрізняють розфарбування, отримані одне з іншого взаємно однозначним відображенням граней при збереженні відношення суміжності.

Завдання

Знаючи все про суміжність чи несуміжність граней тіла Платона чи Архімеда, визначте кількість розфарбувань цього многогранника для даної кількості кольорів.

Вхідні дані

У першому рядку вхідного файлу вказано *натуральну* кількість фарб m, $m < 6$.

Кількість рядків вхідного файлу на 1 перевищує кількість граней многогранника і менша від 24. Усі грані многогранника занумеровано послідовними натуральними числами, починаючи з 1. При $j > 1$ j-й рядок містить (невпорядкований) перелік номерів граней, суміжних з гранню $(j – 1)$.

Вхідні дані гарантують, що кількість симетрій многогранника не перевищує 120.

Вихідні дані

Вихідний файл має містити кількість розфарбувань граней тіла, виконаних за допомогою m фарб і різних з точністю до симетрій многогранника.

Приклад

farben.in	**farben.out**
2 2 3 4 1 3 4 1 2 4 1 2 3	5

Пояснення

Різні розфарбування двома фарбами правильного 4-гранника (який є трикутною пірамідою) розрізняють лише за кількістю граней одного кольору.

3. Істотні інверсії

Максимальна оцінка: 100 балів

Обмеження на час: 0,1 сек.

Обмеження на пам’ять: 32 МБ

Вхідний файл: inverses.in

Вихідний файл: inverses.out

Програма: inverses.*

Розглянемо довільну послідовність чисел x_1, x_2, ..., x_n. Пару індексів (j, k) називають *інверсією (порушенням порядку)*, якщо $j < k$ та $x_j > x_k$. Для довільного

невід'ємного числа t пару індексів (j, k) назвемо *t-істотною інверсією*, якщо $j < k$ та $x_j > x_k + t$.

Завдання

Підрахуйте кількість t-істотних інверсій у послідовності $x_1, x_2, ..., x_n$.

Вхідні дані

Перший рядок містить записи двох цілих чисел n та t при $1 \le n \le 50000$, $0 \le t \le 10^9$. У другому рядку записано цілі числа $x_1, x_2, ..., x_n$, які за модулем не перевищують 10^9.

Вихідні дані

Єдиний рядок вихідного файлу має містити кількість t-істотних інверсій у послідовності $x_1, x_2, ..., x_n$.

Приклади

№	inverses.in	inverses.out
1	6 0 1 2 2 9 5 4	3
2	5 2 1 2 7 3 6	1

Пояснення до прикладів

У прикладі 1 інверсії такі: (4, 5), (4, 6) і (5, 6). Усі ці інверсії — 0-істотні.

У прикладі 2 інверсії такі: (3, 4) і (3, 5). Але $x_3 \le x_5 + 2$, тобто інверсія (3, 5) не є 2-істотною. Такою є лише пара (3, 4).

4. Фортеця

Максимальна оцінка: 100 балів

Обмеження на час: 0,1 сек.

Обмеження на пам'ять: 32 МБ

Вхідний файл: fortress.in

Вихідний файл: fortress.out

Програма: fortress.*

На полі потрібно збудувати фортецю. План її вигляду згори повинен мати форму *невиродженого опуклого* многокутника, сторони якого зображують вали, а вершини — вежі. Також вежі можна розташовувати на валах. Місцевість, де треба збудувати фортецю, є дуже різноманітною з гео- та гідрологічної точки зору. Тому будувати вежі можна лише у певних точках. На відміну від веж, *прямолінійні* вали можна насипати довільно. Чим більше веж розташовано вздовж огорожі фортеці, тим краще.

Завдання

Визначте, яку найбільшу кількість веж можна розташувати у перетинах (стиках) валів і вздовж валів фортеці, яка при виді згори має форму *невиродженого опуклого* многокутника, за умови, що в усіх стиках валів (вершинах многокутника) міститимуться вежі.

Вхідні дані

Перший рядок містить запис цілого числа n ($1 \le n \le 100$) — кількості точок, де можна будувати вежі. У кожному з наступних n рядків записано по два цілих числа x_j та y_j — координати точки, де можна будувати вежу ($|x_j| \le 10000$, $|y_j| \le 10000$). Усі точки (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) є різними.

Вихідні дані

Єдиний рядок вихідного файлу має містити запис найбільшої кількості веж фортеці. Якщо побудувати фортецю неможливо, то потрібно записати 0.

Приклади

№	fortress.in	fortress.out
1	3 0 0 1 1 2 2	0
2	4 1 1 10 1 1 10 3 3	3
3	4 -1 0 0 0 1 0 0 1	4

5. Вкладені множини

Максимальна оцінка: 100 балів

Обмеження на час: 5 сек.

Обмеження на пам'ять: 32 МБ

Вхідний файл: nested.in

Вихідний файл: nested.out

Програма: nested.*

Петрик нещодавно дізнався про поняття множини і вкладення множин. Він зацікавився послідовностями $S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots \supseteq S_N$, у яких кожний наступний елемент — підмножина попереднього, S_0 — певна скінченна множина $\{a_1, a_2, \ldots, a_M\}$.

Петрик зауважив: кожній з вкладених множин можна поставити у відповідність ціле число $H(S) = \sum_{a_k \in S} 2^{k-1}$. Таким чином, $H(S_0) = 2^M - 1$, $H(\varnothing) = 0$.

Петрик випадковим чином обрав N чисел: $h_1, h_2, \ldots, h_N$. Йому стало цікаво: скільки існує різних наборів вкладених множин $S_1, S_2, \ldots, S_N$ при $H(S_1) = h_1 \pmod{41}$, $H(S_2) = h_2 \pmod{41}$, …, $H(S_N) = h_N \pmod{41}$? На жаль, Петрик збився з рахунку, тому просить вас про допомогу.

Вхідний файл

Перший рядок вхідного файлу містить два цілих числа N і M: $1 \le N \le 5$, $1 \le M \le 20$.

Другий рядок містить N цілих чисел h_1, h_2, …, h_N, кожне з яких задовольняє умову: $0 \le h_k \le 40$.

Вихідний файл

Єдиний рядок вихідного файлу повинен містити одне ціле число — шукану кількість вкладених множин. Відомо, що це число не перевищує 10^{18}.

Приклади

№	nested.in	nested.out
1	3 5 7 3 3	1
2	3 5 7 4 5	0
3	1 7 3	4
4	3 10 31 29 17	28

Пояснення до тестів 1–3:

1. Існує лише один варіант: $S_1 = \{a_1, a_2, a_3\}$, $S_2 = S_3 = \{a_1, a_2\}$.

$H(S_1) = 7$, $H(S_2) = H(S_3) = 3$.

2. Такої комбінації вкладених множин не існує.

3. $H(\{a_1, a_2\}) = 2^{1-1} + 2^{2-1} = 3$,

$H(\{a_3, a_4, a_6\}) = 2^2 + 2^3 + 2^5 = 4 + 8 + 32 = 44 = 41 + 3$,

$H(\{a_1, a_3, a_5, a_7\}) = 2^0 + 2^2 + 2^4 + 2^6 = 1 + 4 + 16 + 64 = 85 = 2 \cdot 41 + 3$,

$H(\{a_2, a_3, a_4, a_5, a_6, a_7\}) = 2^1+2^2+2^3+2^4+2^5+2^6 = 2+4+8+16+32+64 = 126 = 3 \cdot 41 + 3$.

6. Портали

Максимальна оцінка: 100 балів

Обмеження на час: 1,2 сек.

Обмеження на пам’ять: 250 МБ

Вхідний файл: portals.in

Вихідний файл: portals.out

Програма: portals.*

Деякі планети галактики, куди нещодавно переїхав Петрик П’яточкін, сполучено порталами. Якщо планети А і Б сполучено, це означає, що на кожній із планет стоїть спеціальний пристрій — портал — для телепортації між ними. Істота, що входить у портал на планеті А, миттєво опиняється на планеті Б і навпаки.

Один і той самий портал не можна використовувати для телепортації на різні планети. Якщо між парою планет ще немає сполучення, їх можна сполучити, але лише побудувавши на кожній із них по новому порталу. Будівництво ж порталів вимагає чималих витрат і може коштувати по-різному на різних планетах.

Будемо казати, що між двома планетами існує шлях телепортації, якщо з однієї планети можна потрапити на іншу, телепортувавшись один або кілька разів (використовуючи проміжні планети). На жаль, поки що не між кожними двома планетами існує шлях телепортації.

Завдання

Маючи схему наявного сполучення планет і знаючи вартість будівництва порталу на кожній планеті, визначте, яку найменшу суму грошей потрібно витратити, щоб забезпечити існування шляху телепортації між кожною парою планет галактики.

Вхідні дані

У першому рядку вхідного файлу вказано два натуральних числа:

n — кількість планет галактики, $n \le 1000$;

m — кількість пар планет, безпосередньо сполучених між собою порталами.

У другому рядку перераховані n натуральних чисел p_1, p_2, …, p_n — вартості будівництва нового порталу відповідно на першій, другій, …, n-й планеті. Жодна з вартостей не перевищує 10^6.

У кожному з наступних m рядків записано по два натуральних числа a_i та b_i, $1 \le a_i < b_i \le n$, $1 \le i \le m$, які задають пару сполучених між собою планет. Кожну пару записано у вхідному файлі лише один раз.

Вихідні дані

Вихідний файл повинен містити єдине натуральне число — *найменшу* суму грошей, яку потрібно витратити на будівництво на деяких планетах порталів, щоб між кожними двома планетами, між якими не було шляху телепортації, такий шлях було створено.

Приклад

portals.in	portals.out
4 2 7 4 7 3 1 3 2 4	10

Пояснення

Можна сполучити четверту планету або з першою, або з третьою, задовольнивши умову задачі і витративши 3 + 7 = 10 грошових одиниць. Легко зрозуміти, що меншою кількістю витрачених грошей обійтися не вдасться.

3. Ідеї розв'язання завдань III етапу

1. Поліклініка (автор — Данило Мисак)

Якщо $a_1 \ge t_1$, то Петрик може прийти у першу ж хвилину, коли прийматиме лікар, і відразу потрапити на прийом. У цьому випадку ми просто виводимо t_1. Тому далі вважатимемо, що $a_1 < t_1$.

Зрозуміло, що Петрику немає сенсу приходити до моменту часу a_1, бо інакше він потрапить на прийом тоді ж, коли потрапив би, якби прийшов у момент a_1 (а саме в момент часу t_1), але прочекає при цьому довше. Якщо хлопець прийде у момент t_2 або пізніше, він узагалі не потрапить на прийом. Таким чином, маємо $t_2 - a_1$ варіантів (a_1, $a_1 + 1$, …, $t_2 - 1$), коли може прийти

хлопець. Перше, що спадає на думку, — перебрати всі такі варіанти та для кожного моменту t визначити, чи потрапить Петрик на прийом, якщо прийде у цей момент, а якщо потрапить, то скільки йому доведеться чекати. Потім вивести оптимальний варіант.

Позначимо через $s(t)$ момент часу, коли хлопець потрапить на прийом за умови приходу у поліклініку у момент t. Почнемо перебір із варіанту $t = a_1$ у порядку зростання t до величини $t = t_2$.

- При $t = a_1$ маємо: $s(t) = t_1$.
- При переході від моменту приходу t до моменту $t + 1$ маємо:
  - Якщо жоден пацієнт не прийшов у момент t, то Петрик, прийшовши на хвилину пізніше у момент $t + 1$, ні одного нового пацієнта перед собою не пропустить:
    - при $s(t) = t$ маємо: $s(t + 1) = t + 1$ — вхід вільний;
    - при $s(t) > t$ маємо: $s(t + 1) = s(t)$ — час потрапляння на прийом залишиться незмінним.

    Таким чином,

    $$\boxed{s(t + 1) = \max(s(t), t + 1).}$$

  - Якщо якийсь пацієнт прийшов у момент t, тобто $t = a_k$ для деякого k, то, прийшовши на хвилину пізніше, Петрик пропустить його вперед. За умовою цей пацієнт перебував на прийомі b_k хвилин. Маємо:

    $$\boxed{s(t + 1) = s(t) + b_k.}$$

    При справдженні нерівності $s(t + 1) \geq t_2$ перебір можна припинити (тепер хлопець уже не зможе потрапити на прийом). Насправді перебір навіть треба перервати, щоб не спричинити переповнення типу даних.

Після закінчення перебору найменша з різниць $s(t) - t$ для всіх розглянутих величин t є найменшим часом очікування, а найменше відповідне значення t — шуканим моментом часу.

Зауважимо: замість того щоб шукати на кожному кроці пацієнта, який прийшов у момент часу t, достатньо одночасно зі збільшенням величини t рухатися масивом моментів приходу пацієнтів і зберігати у пам’яті величину k,

для якої $a_k \le t < a_{k+1}$, та збільшувати її час від часу на 1. Час роботи алгоритму складе в такому випадку $O(t_2 - a_1)$.

Щоб досягнути ефективності $O(n)$, можна помітити, що проміжки між двома сусідніми значеннями a_k і a_{k+1} можна «перескакувати». Інакше кажучи, ми будемо розглядати не моменти $t = a_1, a_1 + 1, a_1 + 2, \ldots$, а моменти $t = a_1, a_2, a_3, \ldots$, використовуючи рівність:

$$\boxed{s(a_{k+1}) = s(a_k) + b_k.}$$

Розгляд величин t потрібно припинити, якщо справджується одне з тверджень:

а) $s(a_k) + b_k \ge t_2$;

б) $s(a_k) + b_k \le a_{k+1}$ при $k < n$;

в) $s(a_k) + b_k < t_2$ при $k = n$.

Якщо справджується одна з умов б) або в), відповіддю буде число $s(a_k) + b_k$. Час очікування хлопця у цьому випадку дорівнюватиме нулю.

2. Фарбування (автор — Олександр Рудик)

Рівень 1 (10 балів). При $m = 1$ кількість розфарувань дорівнює 1.

Рівень 2 (84 бали). Алгоритм розв'язання, що не опирається на теорему Редфілда — Пойя, такий:

1. Зчитавши вхідні дані, встановити кількість граней і відношення суміжності.
2. За допомогою оптимізованого перебору перестановок номерів граней встановити групу симетрій — взаємно однозначних відображень множини граней многогранника у себе, що зберігають відношення суміжності (наявності спільного ребра).

   Примітка. *Кроки 1 і 2 реалізовано у розв'язанні задачі 16.2 відбірково-тренувальних зборів (Многогранник–2, III етап 2005 року), авторське розв'язання є в архіві тестових файлів..*
3. За допомогою структури даних, яку опрацьовують побітово, створити перелік цілих чисел від 0 до $m^n - 1$, де n — кількість граней.

   Примітка. *Для мови Pascal таку струкутуру можна подати масивом множин. Кожному такому цілому числу взаємно однозначно відповідає його*

запис довжини n у системі числення з основою m з дозволом писати нулі на початку запису. А запис у свою чергу взаємно однозначно відповідає набору кольорів для всіх граней: цифри з множини $\{0, 1, 2, \ldots, m-1\}$ *можна тлумачити як номери кольорів, а кожне місце у записі зв'язати з певною гранню.*

4. Надати змінній n_{out} величину 0.
5. Розглядаючи усі числа від 0 до $m^n - 1$ включно у порядку зростання, при виявленні невикресленого числа:
    - збільшити величину n_{out} на 1;
    - викреслити виявлене число — найменше число, що відповідає певному способу розфарбування;
    - для кожного елемента групи симетрій викреслити натуральне число, запис якого довжини n у системі числення з основою m утворено перестановкою цифр запису викресленого на попередньому кроці числа згідно із правилом перестановки граней — місць запису.
6. Вивести величину n_{out}.

При зростанні m такий алгоритм вимагатиме занадто багато оперативної памяті. Спроба зменшити використаний обсяг пам'яті за рахунок збільшення часу обчислень призводить до такої модифікації алгоритму.

1. Зчитавши вхідні дані, встановити кількість граней і відношення суміжності.
2. За допомогою оптимізованого перебору встановити групу симетрій — взаємно однозначних відображень множини граней многогранника у себе.
3. Надати змінній n_{out} величину 0.
4. Розглядаючи усі цілі числа k від 0 до $m^n - 1$ включно у порядку зростання, робити таке:
    - для кожного елемента групи симетрій знайти натуральне число, запис якого довжини n у системі числення з основою m утворено перестановкою цифр запису числа k згідно із правилом перестановки номерів граней, що є номерами місць запису, і перевірити, чи знайдене число не менше, ніж k;
    - якщо нерівності справджуються для всіх елементів групи симетрій

(інакше кажучи, якщо число k є найменшим серед тих, що відповідають певному способу розфарбування), збільшити величину n_{out} на 1.

5. Вивести величину n_{out}.

Але при накладених обмеженнях на час (1 секунда) таке розв'язання набере менше балів — 74 бали.

Рівень 3 (100 балів). Найефективніше *в умовах олімпіади* розв'язання ґрунтується на наслідку теореми Редфілда — Пойя. Теоретичний матеріал українською мовою вичерпно подано за такою адресою: http://kievoi.narod.ru/lectures/polya.html. Алгоритм, на відміну від теорії, легкий для сприйняття і програмного втілення:

1. Зчитавши вхідні дані, встановити кількість граней і відношення суміжності.
2. За допомогою оптимізованого перебору встановити групу симетрій S — взаємно однозначних відображень множини граней многогранника у себе — і кількість елементів цієї групи $|S|$.
3. При всіх k в межах від 1 до $|S|$ змінній p_k надати величину 0.
4. Для кожного елемента групи симетрій:
   - знайти j — кількість циклів, на які він розкладається (фрагмент задачі 8.1 відбірково-тренувальних зборів);
   - збільшити на 1 величину p_j .
5. Обчислити $n_{\text{out}} = (p_1 \cdot m + p_2 \cdot m^2 + p_3 \cdot m^3 + \ldots + p_{|S|} \cdot m^{|S|}) : |S|$.
6. Вивести величину n_{out}.

Рівень 4 — ідеальне розв'язання, *прийнятне лише за умови істотного збільшення часу виконання*, наприклад для заочного туру. Платонові тіла вивчають у школі. Архімедові тіла давно відомі — див. «Математическая энциклопедия», главный редактор И. М. Виноградов, М.: Советская Энциклопедия, том 3, 1982, стаття *Многогранник,* том 4, 1984, стаття *Полуправильные многогранники.* Або див. http://en.wikipedia.org/wiki/Archimedean_solid чи http://mathworld.wolfram.com/ArchimedeanSolid.html.

Маючи достатній запас часу, можна:

1) обчислити коефіцієнти $\{p_k\}$ тіл, що задовольняють умови завдання;
2) використати у програмі масиви сталих, у яких закладено коефіцієнти $\{p_k\}$ для тіл, що задовольняють умови завдання;
3) передбачити, що програма:
   - згідно із вхідними даними визначає вид многогранника за кількістю граней певного вигляду;
   - для зчитаної величини m обчислює величину $(p_1 \cdot m + p_2 \cdot m^2 + p_3 \cdot m^3 + \ldots + p_{|S|} \cdot m^{|S|}) : |S|$, яку і записує у вихідний файл.

Для *тестування* запропоновано 50 тестів:

- $m = 1$ для тестів 1–10;
- $m = 2$ для тестів 11–20;
- $m = 3$ для тестів 21–30;
- $m = 4$ для тестів 31–40;
- $m = 5$ для тестів 41–50.

Залежно від останньої цифри номера тесту розглянуто такі многогранники:

- 1 — тетраедр (4 грані 3-кутники);
- 2 — куб (6 граней 4-кутники);
- 3 — октаедр (8 граней 3-кутники);
- 4 — додекаедр (12 граней 5-кутники);
- 5 — ікосаедр (20 граней 3-кутники);
- 6 — зрізаний тетраедр (4 грані 6-кутники + 4 грані 3-кутники);
- 7 — зрізаний куб (6 граней 8-кутники + 8 граней 3-кутники);
- 8 — зрізаний октаедр (8 граней 6-кутники + 6 граней 4-кутники);
- 9 — кубооктаедр (8 граней 3-кутники + 6 граней 4-кутники);
- 0 — призма (20 граней 4-кутники + 2 грані 20-кутники).

3. Істотні інверсії (автор — Олександр Рибак)

Підрахунок t-істотних інверсій ґрунтується на впорядкуванні послідовності *злиттям*. Процес такого впорядкування складається з кількох кроків.

Спочатку всю послідовність розбивають на послідовності, що містять лише

по 1 елементу. Такі частини вже є впорядкованими.

На кожному наступному кроці вже впорядковані частини послідовності групують у пари у порядку зустрічі елементів у початковій послідовності. Якщо якась послідовність не стала елементом пари на цьому кроці, то вона на цьому кроці не зазнає жодних змін. Кожну утворену пару послідовностей перетворюють на одну впорядковану послідовність, переміщуючи до неї (крок за кроком) менший з перших елементів послідовностей пари (приклад такого перетворення послідовностей подано далі таблицею).

Утворення й перетворення пар послідовностей здійснюють, поки не отримають лише одну впорядковану послідовність.

Покажемо, як у процесі впорядкування злиттям знайти кількість *t*-істотних інверсій. Для шуканої кількості запровадимо змінну *I*, початкова величина якої — 0. У процесі впорядкування пари впорядкованих послідовностей (a_1, a_2, ..., a_r) і (b_1, b_2, ..., b_s) запровадимо змінну q, яку будемо змінювати таким чином. Якщо в деякий момент у другій послідовності залишаться елементи (b_k, b_{k+1}, ..., b_s), то q визначають з умови: a_q — (на даний момент виконання алгоритму) це найперше число першої послідовності, для якого справджується така нерівність: $a_q > b_k + t$. Якщо такого числа не існує, $q = r + 1$. Після переміщення b_k до новостворюваної послідовності такі пари: (a_q, b_k), (a_{q+1}, b_k), ..., (a_r, b_k) перестають утворювати *t*-істотні інверсії (у новостворюваній послідовності відповідні елементи буде розташовано у порядку зростання). Кількість таких пар дорівнює $r + 1 - q$. Саме на цю кількість одразу після переміщення b_k лічильник кількості інверсій *I* збільшимо на $r + 1 - q$, *після* чого поновимо величину q. Завдяки впорядкованості розглядуваних послідовностей, це можна зробити бінарним пошуком,шляхом послідовного збільшення q до справдження сукупності співвідношень: $a_q > b_{k+1} + t$ або $q = r$

Подамо таблицею послідовні кроки перетворення пари послідовностей 1, 3, 4, 7 та 1, 2, 9 при $t = 2$ і початковій величині $I = 3$ методом злиття.

Перша послідовність пари	**Друга послідовність пари**	**Створювана послідовність**	***I***	***q***

1, 3, 4, 7	1, 2, 9		3	3
3, 4, 7	1, 2, 9	1	3	3
3, 4, 7	2, 9	1, 1	5 = 3 + (4+1 – 3)	4
3, 4, 7	9	1, 1, 2	6 = 5 + (4+1 – 4)	5
4, 7	9	1, 1, 2, 3	6	5
7	9	1, 1, 2, 3, 4	6	5
	9	1, 1, 2, 3, 4, 7	6	5
		1, 1, 2, 3, 4, 7, 9		

Зауважимо: I та q змінюють лише тоді, коли переміщують елемент другої послідовності. Коли одна з послідовностей стане порожньою, величини цих змінних залишатимуться сталими.

Оцінимо ефективність алгоритму. Скільки операцій припадає на опрацювання послідовностей з r і s елементів?. Щоб вибрати й перемістити елемент до створюваної послідовності, потрібно лише порівняти деякі два елемента й перенести менший з них. Маємо $O(r + s)$ операцій загалом. При оновленні величини q достатньо проглянути першу послідовність лише в напрямку зростання індексу або використати бінарний пошук. Тому загальну кількість змін q оцінюємо відповідно як $O(r)$. Кількість операцій при перетворенні пари послідовностей має порядок $O(r + s)$. Тому для всіх пар на одному кроці потрібно виконати $O(n)$ операцій. Кількість кроків не перевищує $\log_2 n + 1$. Отже, загальна кількість операцій становить $O(n \log_2 n)$.

4. Фортеця (автор — Олександр Рибак)

Точки, в яких можна розташовувати вежі, будемо називати *відміченими*. Впорядкуємо ці точки за зростанням першої координати, а при однакових перших координатах — за зростанням другої. Згідно з нашим впорядкуванням занумеруємо відмічені точки натуральними числами від 1 до n, де n – кількість цих точок. Розглянемо довільний невироджений опуклий многокутник — реалізацію розв’язання задачі: всі вершини цього многокутника розташовані у відмічених точках, а його межа містить найбільшу можливу кількість таких точок. Нехай A – вершина многокутника, яка має найменший номер серед усіх його вершин, а B – вершина, яка має найбільший номер.

Введемо деякі означення. Ламану лінію $V_0V_1...V_k$ вважатимемо *опуклою вгору*, якщо при всіх $j = 1, ..., k – 1$ промінь, проведений з вершини V_j вертикально

вниз, перетинає відрізок $V_{j-1}V_{j+1}$. Аналогічно, ламану$V_0V_1...V_k$ вважатимемо *опуклою вниз*, якщо при всіх $j = 1, ..., k - 1$ промінь, проведений з вершини V_j вертикально вгору, перетинає відрізок $V_{j-1}V_{j+1}$. У тому випадку, коли ламана є відрізком, вважатимемо її опуклою одночасно вгору і вниз.

Вершини A та B розділяють межу многокутника на дві ламані лінії. За вибором A та B, одна з ламаних опукла вниз, а інша – вгору. Позначимо їх як l_1 та l_2 відповідно. Очевидно, що при русі від A до B вздовж l_1 або l_2 відмічені точки на цих лініях лежать у порядку зростання номерів. Многокутник невироджений, тому можливі лише такі три випадки:

1) l_1 складається з декількох відрізків, а l_2 є відрізком;

2) l_1 є відрізком, а l_2 складається з декількох відрізків;

3) обидві ламані l_1 та l_2 складаються з декількох відрізків.

У кожному з випадків ламані лінії, що складаються з декількох відрізків, мають містити максимальну кількість відмічених точок серед ліній свого типу. Наприклад, у першому випадку l_1 містить максимальну кількість відмічених точок серед усіх ламаних, які сполучають A та B, складаються з декількох відрізків і опуклі вгору. У другому випадку l_2 містить максимальну кількість таких точок серед усіх ламаних, які сполучають A та B, складаються з декількох відрізків і опуклі вниз. Те ж саме для третього випадку. Якщо одна зі згаданих умов не справджується, відповідну ламану можна замінити на лінію того самого типу з більшою кількістю точок. Тоді новий многокутник також буде невиродженим і опуклим, але міститиме більшу кількість відмічених точок. А це суперечить припущенню про максимальність відмічених точок для нашого многокутника. Отже, для кожної пари A, B достатньо знайти, яку найбільшу кількість відмічених точок може містити ламана, що сполучає A та B, складається з декількох відрізків і опукла вгору. Те ж саме для ламаних, опуклих вниз.

Назвемо ламану лінію *правильною*, якщо вона задовольняє такі умови:

1) всі її вершини є відміченими точками,
2) при русі вздовж ламаної від кінця з меншим номером до кінця з більшим номером номери відмічених точок на цій ламаній зростають;

3) ламана опукла вгору;

4) ламана не є відрізком.

Трійку відмічених точок (A, C, B) назвемо *впорядкованою*, якщо A має менший номер, ніж C, а C — менший номер, ніж B. Нехай $F(A, C, B)$ — найбільша можлива кількість відмічених точок, розташованих на *правильній* ламаній, де A є першою, C — передостанньою, а B – останньою відміченою точкою. Якщо відповідної ламаної не існує, покладемо $F(A, C, B) = 0$.

Перед тим, як шукати $F(A, C, B)$ для всіх упорядкованих трійок, зробимо декілька загальних спостережень. Впорядковану трійку (A, C, B) називатимемо *правою*, якщо ламана ACB опукла вгору. Якщо при цьому C не лежить на відрізку AB, трійку називатимемо *строго правою*. Нехай $S(C, B)$ – кількість відмічених точок на відрізку CB. Згідно з даними означеннями, на правильній ламаній трійка (A, C, B) має бути строго правою. Тому для інших трійок маємо $F(A, C, B) = 0$. Також потрібної ламаної не існує, якщо $S(C, B) > 2$ — у цьому випадку C не може бути передостанньою точкою. З іншого боку, якщо (A, C, B) строго права і $S(C, B) = 2$, то $F(A, C, B) > 0$. Справді, тоді на роль потрібної ламаної підходить об’єднання відрізків AC та CB. У зв’язку з цим строго праві трійки (A, C, B), для яких $S(C, B) = 2$, називатимемо *перспективними*.

Перейдемо до підрахунку $F(A, C, B)$. Кожне значення A розглянемо окремо. Для всіх C та B, для яких (A, C, B) є впорядкованою трійкою, знайдемо $F(A, C, B)$ методом динамічного програмування. А саме, за певного значення B нам буде потрібно розрахувати $F(A, C, B)$ для всіх таких C, що (A, C, B) — впорядкована трійка. Ми зробимо це на основі значень $F(A, P, Q)$, розрахованих для всіх таких впорядкованих трійок (A, P, Q), де Q має менший номер, ніж B. Як уже зазначалося, якщо (A, C, B) не перспективна, то $F(A, C, B) = 0$. У випадку перспективної трійки (A, C, B) визначимо $F(A, C, B)$ як максимум з двох значень:

$\max\{F(A, D, C) + 1$, де (D, C, B) — права трійка$\}$ та $S(A, C) + 1$. Якщо відповідних правих трійок (D, C, B) немає, покладемо: $F(A, C, B) = S(A, C) + 1$. Дійсно, потрібну ламану для трійки (A, C, B) можна отримати, якщо приєднати відрізок CB до відповідної ламаної для (A, D, C) або до відрізка AC. Залишилось

обрати той варіант, який дасть лінію з найбільшою кількістю відмічених точок.

Нехай у парі (A, B) точка A має менший номер, ніж B. Для кожної такої пари нехай $U(A, B) = \max\{F(A, C, B)$, де (A, C, B) — впорядкована трійка$\}$. А якщо для даних A та B впорядкованих трійок (A, C, B) немає, покладемо: $U(A, B) = 0$. Тоді $U(A, B)$ дорівнює максимальній кількості відмічених точок, що лежать на деякій ламаній з кінцями в A та B, яка опукла вгору і не є відрізком.

Точно так само знайдемо $V(A, B)$ — максимальну кількість відмічених точок, що лежать на деякій ламаній з кінцями в A та B, яка опукла вниз і не є відрізком. Цю задачу можна звести до попередньої, якщо замінити всі координати точок на протилежні за знаком.

Для кожної пари (A, B), де A має менший номер, ніж B, обчислимо:

$$M(A, B) = \max\{U(A, B) + S(A, B), S(A, B) + V(A, B), U(A, B) + V(A, B)\} - 2.$$

Величина $M(A, B)$ дорівнює найбільшій кількості відмічених точок, що знаходяться на межі невиродженого опуклого многокутника, в якому вершиною з найменшим номером є A, а з найбільшим — B. Взявши максимум $M(A, B)$ по всіх парах (A, B), отримаємо відповідь на питання задачі.

5. Вкладені множини (автор — Юрій Знов'як)

Динамічне програмування (неповне розв'язання)

Позначимо через $f(n,Q)$ кількість послідовностей вкладених множин $S_1 \supseteq S_2 \supseteq \ldots \supseteq S_n$, при яких:

- $S_0 \supseteq S_1$;
- $H(S_1) \bmod 41 = h_1$, $H(S_2) \bmod 41 = h_2$, …, $H(S_n) \bmod 41 = h_n$;
- $S_n = Q$.

Інакше кажучи, $f(n,Q)$ — відповідь на задачу з умови при $n = N$ і заданої кінцевої підмножини $(S_n = Q)$. При такому означенні $f(n, Q)$ шукану кількість можна подати сумою:

$$\sum_{Q \subseteq S_0} f(N, Q). \qquad (1)$$

Тут додавання здійснено за всіма Q, що є підмножинами S_0.

Розглянемо приклад № 1 з умови: $N = 3$, $M = 5$, $h_1 = 7$, $h_2 = 3$, $h_3 = 3$.

Згідно з умовою $S_0 = \{a_1, a_2, \ldots, a_5\}$. Існує лише один набір вкладених підмножин, що задовольняє умові. А саме: $S_1 = \{a_1, a_2, a_3\}$, $S_2 = S_3 = \{a_1, a_2\}$.

Маємо:

$$f(1, \{a_1, a_2, a_3\}) = 1,\ f(1, Q) = 0 \text{ при } Q \neq \{a_1, a_2, a_3\},$$
$$f(2, \{a_1, a_2\}) = 1, \qquad f(2, Q) = 0 \text{ при } Q \neq \{a_1, a_2\},$$
$$f(3, \{a_1, a_2\}) = 1, \qquad f(3, Q) = 0 \text{ при } Q \neq \{a_1, a_2\}.$$

Зауважимо: $f(1, Q)$ — це кількість послідовностей з одного елемента із заданим останнім елементом, що відповідає властивості $H(Q) \bmod 41 = h_1$. Маємо:

$$f(1, Q) = 1 \text{ при } H(Q) \bmod 41 = h_1, \qquad (2)$$
$$f(1, Q) = 0 \text{ при } H(Q) \bmod 41 \neq h_1. \qquad (3)$$

Покажемо, як обчислити $f(n, Q)$ при $n > 1$, якщо відомі $f(n - 1, R)$ при всіх $R \supseteq S_0$. Зауважимо: $f(n, Q)$ — це кількість різних послідовностей вкладених множин $S_1 \supseteq S_2 \supseteq \ldots \supseteq S_{n-1} \supseteq S_n$. Маємо рекурентне співвідношення:

$$f(n, S_n) = \sum_{S_n \subseteq S_{n-1} \subseteq S_0} f(n-1, S_{n-1}). \qquad (4)$$

Тут додавання здійснено за всіма S_{n-1}, що є підмножиною S_0 і містять S_n як підмножину.

Використавши початкові значення (2–3) і рекурентну формулу (4), можна знайти всі доданки відповіді — суми (1).

Розглянемо питання реалізації. Як подати $f(n, Q)$? Доволі очевидним рішенням буде подання $f(n, Q)$ масивом:

```
f: array[1..N, 0..2^M–1] of Int64;      // FreePascal
int64 f[N][ 2^M];                       // C++
```

Перший індекс — n, другий індекс — $H(Q)$: $f(n, Q) =$ `f[n][H(Q)]`.

На жаль, таке розв'язання використовує занадто багато пам'яті.

Динамічне програмування 2 (покращене неповне розв'язання)

Зауважимо: $f(n, Q) = 0$ при всіх Q таких, що $H(Q) \bmod 41 \neq h_n$. Можна у 41 раз зменшити потребу щодо пам'яті:

`f: array[1..N, 0.. 2^M div 41] of Int64;` // FreePascal

`int64 f[N][ 2^M/41];` // C/C++

У цьому випадку величину $f(n, Q)$ зберігають не в `f[`n`][`$H(Q)$`]`, а в `f[`n`][`$H(Q)$ `div 41]`.

Динамічне програмування 3 (повне розв'язання)

Найтривалішим у поданому розв'язанні є обчислення згідно з рекурентною формулою (4) — необхідно швидко перелічити всі підмножини S_{n-1} заданої множини S_n. На щастя, це зробити дуже легко!

Спочатку вирішимо, як подаватимемо множини. Найзручнішим буде працювати з множинами як зі звичайними цілими числами (LongInt у FreePascal або int в C++). Біт 0 відповідатиме елементу a_1, біт 1 — елементу a_2 і т.д. Наприклад, множині $\{a_1, a_3\}$ відповідає число $H(\{a_1, a_3\}) = 101_2 = 5_{10}$. Тут H — функція з умови завдання.

Існує зв'язок між діями з множинами — аргументами функції H — і побітовими діями з величинами функції H (використано позначення C/C++):

$$H(A \supseteq B) = H(A) \mid H(B);$$

$$H(A \cap B) = H(A) \,\&\, H(B),$$

де:

| — операція побітового «або» (у FreePascal пишуть `or`);

& — операція побітового «і» (у FreePascal пишуть `and`);

Доведемо, що результатом X & (X – 1) є відкидання наймолодшого одиничного біту числа. Подамо число X у двійковій системі числення:

$$X = (x_k x_{k-1} \ldots x_1 x_0)_2 = x_0 \cdot 2^0 + x_1 \cdot 2^1 + \ldots + x_k \cdot 2^k.$$

Без обмеження загальності міркувань будемо вважати, що саме m наймолодших бітів X — нулі. Інакше кажучи, X ділиться без лишку на 2^m, але не ділиться на 2^{m+1}: $x_0 = x_1 = \ldots = x_{m-1} = 0$, $x_m = 1$. Маємо:

$$X = (x_k x_{k-1} \ldots x_{m+1} 10 \ldots 0)_2$$

$$X - 1 = (x_k x_{k-1} \ldots x_{m+1} 01 \ldots 1)_2$$

Наймолодші $(m + 1)$ біт чисел X і X–1 відрізняються, а решта бітів — однакові. Тому

$$X \,\&\, (X-1) = (x_k x_{k-1} \ldots x_{m+1} 00\ldots0)_2,$$

що й потрібно було довести. Саме ця рівність лежить в основі переліку підмножин даної множини.

Нехай $Q = \{a_r, a_s, a_t, \ldots, a_u\}$, і R — непорожня підмножина Q. Маємо:

$H(Q) = 2^r + 2^s + 2^t + \ldots + 2^u$;

$H(\{a_r\}) = (\mathbf{1}0\ldots0\,0\,0\ldots0\,0\,0\ldots0\,0\,0\ldots0)_2$;

$H(\{a_s\}) = (00\ldots0\,\mathbf{1}\,0\ldots0\,0\,0\ldots0\,0\,0\ldots0)_2$;

$H(\{a_t\}) = (00\ldots0\,0\,0\ldots0\,\mathbf{1}0\ldots0\,0\,0\ldots0)_2$;

...

$H(\{a_u\}) = (00\ldots0\,0\,0\ldots0\,0\,0\ldots0\,\mathbf{1}\,0\ldots0)_2$;

$H(Q) = (\mathbf{1}0\ldots0\,\mathbf{1}\,0\ldots0\,\mathbf{1}\,0\ldots0\,\mathbf{1}0\ldots0)_2$;

$H(R) = (\boldsymbol{x_1}0\ldots0\boldsymbol{x_2}0\ldots0\boldsymbol{x_3}0\ldots0\boldsymbol{x_m}0\ldots0)_2$.

Як доведено, $H(R) \,\&\, (H(R) - 1)$ відповідає тій множині R без її елемента з найменшим номером. А якій множині відповідає $H(Q) \,\&\, (H(R) - 1)$? Це число відповідає тій підмножині Q, яка лексикографічно йде безпосередньо перед множиною R!

Розглянемо приклад. $Q = \{a_2, a_3, a_5, a_6\}$, $H(Q) = 110110_2 = 54_{10}$.

$H(R)$	$H(R) - 1$	$H(R-1) \,\&\, H(Q)$
000010	000001	000000
000100	000011	000010
000110	000101	000100
010000	001111	000110
010010	010001	010000
010100	010011	010010
010110	010101	010100
100000	011111	010110
100010	100001	100000
100100	100011	100010
100110	100101	100100
110000	101111	100110
110010	110001	110000
110100	110011	110010
110110	110101	110100

При m — множині, чиї підмножини ми перелічуємо, k — підмножині m, код

програми мовою Pascal виглядатиме так:

```
k, m: LongInt;
…
k := m;
while true do
begin
     …
     if k = 0 then break;
     k := (k - 1) AND m;
end;
```

або мовою C/C++:

```
for (int k = m; ; k = (k - 1) & m) {
    …
    if (k == 0) break;
}
```

Використання такого підходу до перелічування підмножин виявляється достатньо для того, щоб розв'язати задачу в відведений час.

6. Портали (автор — Данило Мисак)

Формулювання задачі в термінах теорії графів. *Задано незв'язний неорієнтований граф та вартість кожної його вершини. Необхідно зробити граф зв'язним, провівши ребра найменшої сумарної вартості за умови, що вартість нового ребра є сумою вартостей вершин, які це ребро сполучає.*

Позначимо через *c* кількість компонент зв'язності графа. Зауважимо таке:

- немає сенсу сполучати ребром вершини з однієї компоненти зв'язності;
- немає сенсу проводити більше ніж одне ребро між вершинами, що належать певним двом різним компонентам зв'язності;
- якщо *a* — найменша вага вершини однієї компоненти зв'язності, а *b* — найменша вага вершини іншої компоненти зв'язності, то ребро найменшої ваги, що сполучить вершини цих компонент зв'язності, має вагу *a* + *b*.

Перетворимо наш граф: замість кожної компоненти зв'язності залишимо вершину найменшої ваги у цій компоненті. Задачу зведено до побудови дерева із найменшою сумою ваг ребер на залишених *c* вершинах.

Розглянемо довільне таке дерево і зафіксуємо його вершину M найменшої

ваги (якщо таких вершин кілька, то довільну з них). Якщо дерево має ребро, що не виходить із цієї вершини, розглянемо вершини А та Б, які воно сполучає. Приберемо ребро. Дерево розпадеться на дві компоненти зв'язності, причому вершини А та Б належатимуть різним компонентам. Сполучимо тоді вершину М з тією з двох вершин А та Б, яка лежить в іншій, ніж М, компоненті зв'язності. Унаслідок цього, згідно з вибором вершини М, вага проведеного між компонентами зв'язності ребра не збільшилась, а з вершини М стало виходити на одне ребро більше. Отже, не збільшуючи сумарну вагу дерева, ми можемо змінити його так, щоб усі ребра дерева виходили з вершини М. Після цього як саме дерево, так, відповідно, і його вага відновлюються однозначно. Вага дерева дорівнює

$$(c-1)\cdot a_m + (a_1 + a_2 + \ldots + a_{m-1} + a_{m+1} + \ldots + a_c) = (c-2)\cdot a_m + (a_1 + \ldots + a_c),$$

де a_1, a_2, …, a_c — ваги всіх c вершин дерева, $a_m = \min\{a_1, a_2, \ldots, a_c\}$ — вага вершини М.

Залишилося реалізувати пошук компонент зв'язності початкового графа та найменшої ваги вершини у кожній компоненті. Найпростіше зробити це з допомогою пошуку в глибину, якщо рекурсивна функція повертає найменшу знайдену на даний момент вагу в поточній компоненті зв'язності. Складність алгоритму складає $O(n + m)$.

Для розв'язання задачі можна застосувати й загальний підхід до побудови т. зв. мінімального кістякового дерева, причому як уже після заміни компонент зв'язності окремими вершинами, так і без цього. В останньому випадку треба присвоїти усім уже наявним у графі ребрам нульову вартість.

Найшвидші реалізації традиційних алгоритмів побудови мінімального кістякового дерева (наприклад, Крускала або Прима) також дають повний бал.

4. Авторські розв'язання завдань III етапу

1. Поліклініка

```
/* GCC */

#include <stdio.h>
#define maxn 100000
```

```
int n, t1, t2, a[maxn], b[maxn], i, end,
                  minMoment, minTime, ans;

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

    scanf("%d %d %d", &n, &t1, &t2);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = 0; i < n; i++)
        scanf("%d", &b[i]);

    if (a[0] >= t1) // Якщо перший пацієнт
// прийшов у момент початку прийому
// або пізніше, Петрику достатньо прийти
// у момент початку прийому, щоб одразу
// потрапити до лікаря
        ans = t1;
    else
    {
        end = t1;   // Час, коли лікар
// закінчить огляд останнього розглянутого
// пацієнта (початкове значення — момент
// початку прийому t1)
        minTime = -1;   // Найменший час
// очікування, який Петрик може забезпечити
// на даний момент (початкове значення -1
// умовно позначає, що Петрик ще нічого
// не встиг собі забезпечити)
        for (i = 0; i < n; i++)
// Розглядаємо одне за одним усіх пацієнтів
        {
            if (minTime == -1 ||
                      end - a[i] < minTime)
// Якщо, прийшовши водночас
// із і-м пацієнтом, Петрик забезпечить
// менший час очікування, ніж міг
// забезпечити раніше
            {
                minTime = end - a[i];
// Запам'ятовуємо цей час очікування
                minMoment = a[i];
// А також запам'ятовуємо відповідний
// момент, коли має прийти Петрик
            }
            end += b[i];    // Час, коли
// лікар закінчить огляд пацієнта,
// якого зараз розглядаємо
            if (end >= t2)  // Якщо після
// цього лікар уже не вестиме прийом,
// виходимо з циклу
                break;
            if (i == n - 1 ||
```

```
                            a[i + 1] >= end)
// Якщо ми щойно розглянули останнього
// пацієнта з черги або якщо наступний
// пацієнт прийде не раніше, ніж лікар
// закінчить огляд розглянутого, Петрик
// може прийти в момент закінчення огляду
// розглянутого пацієнта — тоді він одразу
// потрапить до лікаря
            {
                minMoment = end;
                break;
            }
        }
        ans = minMoment;    // Як відповідь
// слід вивести відповідний момент часу
    }

    printf("%d\n", ans);
    return 0;
}
```

2. Фарбування

```
{ Free Pascal }
{$n+}             {Верхні межі кількостей:}
const ng_=22;                        {граней}
      ns_=120;     {симетрій многогранника}
      nb_=61000000;
      full: set of byte=[0..255];
      simple=true;   {Без т. Редфілда-Пойа}
label NEXTi,NEXTh,RES; {Мітки для перебору}
                                {Кількості:}
var ng,                              {граней}
    ns,            {симетрій многогранника}
     m,                             {кольорів}
     i,j,k,l: byte;  {Допоміжні лічильники}
           o: text;
                               {Суміжні грані}
sg: array[0..ng_] of set of 1..ng_;
 b: array[0..nb_] of set of byte;
  {Група підстановок-симетрій многогранника}
s: array[1..ns_,1..ng_] of 1..ng_;
c,                   {Номери кольорів граней
  або многочлен - ряд переліку конфігурацій}
h,                             {Номери гіпотез}
v: array[0..ng_] of byte;     {Перестановки}
fault: boolean;           {Хибність гіпотези}
ib,nb,imng,kmng,mng: longint;
nout: extended;
     {Перестановка елементів v при переборі}
procedure SWAP;                          BEGIN
          v[0]:=v[h[i]];
                v[h[i]]:=v[ng-i+1];
                         v[ng-i+1]:=v[0]END;
```

```
   {Перехід  до наступного числа  у системі
   числення з основою m, нумерація розрядів
                          - від наймолодшого}
procedure next;  var i: byte;          BEGIN
          inc(c[1]);  i:=1;
          while c[i]>=m do begin
                   c[i]:=c[i]-m;
                   inc(i);
                   inc(c[i]) end         END;
        {Число, записане у системі числення
 з основою m номерами кольорів граней після
 застосування симетрії  j  до розфарбування
               з номерами кольорів граней c}
function number(k: byte): longint;
            var v: longint;  l: byte; BEGIN
                        v:=    c[s[k,ng]];
for l:=ng-1 downto 1 do v:=v*m+c[s[k,l]];
                   number:=v            END;
                                       BEGIN
            {Зчитування і опрацювання даних}
for j:=1 to ng_ do sg[j]:=[];
assign(o,'farben.in');
reset (o);
readln(o,m);
ng:=0;    while not seekeof (o) do    begin
ng:=ng+1; while not seekeoln(o) do    begin
          read(o,j);  include(sg[ng],j) end;
          readln(o)                     end;
close  (o);
assign (o,'farben.out');
rewrite(o);
  {Знаходження групи симетрій многогранника}
ns:=0;
 i:=0;
for j:=1 to ng do v[j]:=j;
NEXTi: inc(i); h[i]:=0;
NEXTh:     inc(h[i]);
       if h[i]>ng-i+1 then            begin
            dec(i);
            if i=0 then goto RES;
            SWAP;
            goto NEXTh                  end;
       if i>1 then                    begin
          j:=0;
          repeat inc(j);
            fault:= (j in sg[i]) and
            not (v[ng-j+1] in sg[v[h[i]]]) or
                not (j in sg[i]) and
                (v[ng-j+1] in sg[v[h[i]]]);
          until (j=i-1) or fault;
          if fault then goto NEXTh      end;

       SWAP;  if i<ng then goto NEXTi;
       inc(ns);
       for j:=1 to ng do s[ns,j]:=v[ng-j+1];
```

```
       SWAP;  goto NEXTh;

      {Перебір усіх способів розфарбування}
RES: if simple then                    begin
  {без використання теореми Редфілда - Пойя}
if (ng*ln(m)>ln(256.*(nb_+1))) and
   (ng*ln(m)>ln(2147483647))  then halt;
mng:=m;  for j:=2 to ng do mng:=mng*m;
 nb:=mng div 256;
for ib:=0 to nb do b[ib]:=full;
for i :=1 to ng do c[i]:=0;
exclude(b[0],0);
nout:=1; imng:=0;
while imng < mng-1 do                  begin
  imng:=imng+1;
  next;
  if (imng mod 256) in b[imng div 256]
                                  then begin
  nout:=nout+1;
  for k:=1 to ns do                    begin
    kmng:=number(k);
    exclude(b[kmng div 256],
              kmng mod 256) end end end end
else                                   begin
   {з використанням теореми Редфілда - Пойя}
for j:=1 to ng do c[j]:=0;
for k:=1 to ns do                      begin
b[0]:=[1..ng];
i:=0;
for j:=1 to ng do if j in b[0] then    begin
l:=j;
repeat exclude(b[0],l);  l:=s[k,l];
 until not (l in b[0]);
inc(i)                                   end;
inc(c[i])                                end;
nout:=m*c[ng];
for i:=ng-1 downto 1 do nout:=(nout+c[i])*m;
                        nout:=nout/ns    end;
writeln(o,nout:0:0);
close(o);
END.
```

3. Істотні інверсії

```
{ Free Pascal }
PROGRAM Inverses;
var n:longint; {Кількість елементів у
      послідовності з вводу.}
    x:array[1..50000] of longint;
     {Послідовність, для якої потрібно
      знайти кількість істотних інверсій.}
    x2:array[1..32769] of longint;
      {Запасний масив для сортування.}
    t:longint; {Величина, яку має
      перевищувати різниця x[j]-x[k], щоб
```

```
      пара (j,k) вважалася істотною
      інверсією за умови j<k.}
    d,i,j,k,p,q:longint; {Індекси для
      циклів.}
    inv:longint; {Кількість t-істотних
      інверсій.}
BEGIN
  {Відкриття файлів для вводу.}
  assign(input,'inverses.in');
  reset(input);
  assign(output,'inverses.out');
  rewrite(output);
  {Читання вхідних даних.}
  read(n,t);
  for i:=1 to n do read(x[i]);
  inv:=0; {Спочатку кількість знайдених
    інверсій дорівнює 0.}
  {Тепер відбуватиметься сортування
   послідовності методом злиття з
   підрахунком істотних інверсій.}
  d:=1; {Змінна d означає довжину
    відрізків, що зливаються. Спочатку ця
    довжина дорівнює 1.}
  while (d<n) do begin
    {Цикл виконується, поки довжина
     відрізків, які мають об'єднуватися,
     менша за кількість елементів.}
    i:=0;    while (i+d<n) do begin
```

```
      {Перший з двох об'єднуваних відрізків
       копіюється до запасного масиву.}
      for j:=1 to d do x2[j]:=x[i+j];
      p:=i+d*2;
      if (p>n) then p:=n;
      q:=1; {У першому відрізку індексом q
        позначатиметься найлівіший елемент,
        який перевищує x[k] більше, ніж на
        t. До початку пошуку згаданого
        елемента значення q дорівнює 1.
        Далі воно лише зростатиме.}
      j:=1;
      k:=i+d+1;
      {Об'єднуємо два відсортованих
       відрізка, поки один з них не
       закінчиться.}
      while ((j<=d)and(k<=p)) do begin
        if (x2[j]<=x[k]) then begin
           x[k-d-1+j]:=x2[j];
           Inc(j);
        end;
        if (j<=d) then
           if (x2[j]>x[k]) then begin
              {У першому відрізку
               шукатиметься номер
               найлівішого елемента, який
               перевищує x[k] більше, ніж
               на t. Якщо такого елемента
               немає, q стане рівним d+1.}
              while ((q<=d)and
                     (x2[q]<=x[k]+t)) do
                Inc(q);
              inv:=inv+d-q+1; {Кількість
                знайдених інверсій
                збільшується на кількість
                елементів першого відрізка,
                що перевищують x[k] більше,
                ніж на t.}
              x[k-d-1+j]:=x[k];
              Inc(k);
           end;
      end;
      if (j<=d) then
         for j:=j to d do x[p-d+j]:=x2[j];
      i:=i+d*2; {Перехід до наступних двох
        відрізків.}
    end;
    d:=d*2; {Подвоєння довжини відрізків.}
  end;
  writeln(inv); {Вивід відповіді.}
  {Закриття файлів вводу та виводу.}
  close(input);
  close(output);
END.
```

4. Фортеця

```
{ Free Pascal }
PROGRAM Fortress;
var n:word; {Кількість відмічених точок.}
    x,y:array[1..100] of integer;
     {Координати відмічених точок. На
      початку роботи програми ці точки
      сортуються лексикографічно, тобто за
      зростанням x-координати, а при рівних
      x-координатах - за зростанням
      y-координати.}
    sort:array[3..100,1..99] of byte;
     {Для кожного i від 3 до n числа
      sort[i,1], ..., sort[i,i-1]
      складають послідовність, яка є
      перестановкою індексів 1, ..., i-1.
      Вказані індекси впорядковуються
      наступним чином. Через точку
      (x[i],y[i]) проводиться промінь,
      направлений вертикально вниз. Потім
      промінь починає обертатися за
      годинниковою стрілкою, а індекси
      1, ..., i-1 впорядковуються згідно з
      часом попадання точок
      (x[1],y[1]), ..., (x[i-1],y[i-1]) на
      цей промінь. Порядок точок, які
      попадають на промінь одночасно, для
      нас не важливий.}
    f:array[3..100,1..100] of byte;
     {f[j,k] - це найбільша можлива
      кількість відмічених точок на опуклій
      ламаній, останньою точкою якої є
      (x[j],y[j]), а передостанньою з
      відмічених точок - (x[k],y[k]).
      Ламана має починатися в точці
      (x[i],y[i]), де i - індекс, який
      пробігає значення від 1 до n-2. Для
      кожного i всі f[j,k] обчислюються
      окремо. При підрахунку f[j,k] точка
      (x[j],y[j]) не враховується. Ми
      обчислюємо значення f[j,k] лише для
      j>k, а елементи f[j,j] - це
      максимуми з чисел f[j,1], ...,
      f[j,j-1]. Спочатку розраховуються
      значення f[j,k] для ламаних, опуклих
      вгору, а потім - для опуклих вниз.
      При обчисленні масиву f розглядаються
      ламані, відмінні від відрізка.}
    seg:array[2..100] of byte;
     {seg[j] - це кількість відмічених
      точок на відрізку з кінцями
      (x[i],y[i]) та (x[j],y[j]), не
      враховуючи точки (x[j],y[j]). Тут i -
```

```
      індекс, який пробігає значення від 1
      до n-2. Для кожного i масив seg
      обчислюється окремо.}
    maxpoly:word; {Максимально можлива
     кількість веж на межі невиродженого
     опуклого многокутника. Вежі можна
     будувати лише у відмічених точках.}
    i,j,k,m,p,q:word; {Індекси для циклів.}
    vx1,vy1,vx2,vy2:longint;
     {Координати векторів з початками та
      кінцями у відмічених точках. За
      допомогою псевдоскалярного добутку
      векторів (vx1,vy1) та (vx2,vy2)
      з'ясовується, чи йдуть два промені за
      годинниковою стрілкою або чи лежать
      три точки на одній прямій.}
BEGIN
  {Відкриття файлів для вводу та виводу.}
  assign(input,'fortress.in');
  reset(input);
  assign(output,'fortress.out');
  rewrite(output);
  {Читання вхідних даних.}
  read(n);
  for i:=1 to n do read(x[i],y[i]);
  maxpoly:=0; {Для початкового варіанту
    відповіді встановлюється значення 0.
    Якщо існує многокутник, що задовольняє
    умову задачі, maxpoly набуде значення,
    рівне кількості відмічених точок на
    межі цього многокутника. Потім зміна
    значення maxpoly відбуватиметься
    кожного разу, коли з'являтиметься
    варіант відповіді з більшою кількістю
    відмічених точок на межі. Якщо
    потрібного многокутника не існує,
    maxpoly залишиться рівним 0, як і
    вимагає постановка задачі.}
  if (n>=3) then begin
     {Для n<3 відповідь залишається рівною
      0, бо потрібного многокутника точно
      немає. Для випадку n>=3
      відбуватиметься перевірка, чи існує
      многокутник, що задовольняє умову
      задачі. Якщо він існує, буде
      визначено, яка найбільша кількість
      відмічених точок може бути на межі
      такого многокутника.}
     for i:=1 to n-1 do begin
       {Відмічені точки сортуються в
        лексикографічному порядку.}
       m:=i;
       for j:=i+1 to n do
         if ((x[j]<x[m])or((x[j]=x[m])and
             (y[j]<y[m]))) then m:=j;
```

```
  vx1:=x[i];
  x[i]:=x[m];
  x[m]:=vx1;
  vy1:=y[i];
  y[i]:=y[m];
  y[m]:=vy1;
end;
for i:=3 to n do begin
  {Для кожного i від 3 до n
   будуються послідовності
   sort[i,1], ..., sort[i,i-1].
   Зміст масиву sort описаний у
   розділі var.}
  for j:=1 to i-1 do sort[i,j]:=j;
  for j:=1 to i-2 do begin
    m:=j;
    for k:=j+1 to i-1 do begin
      vx1:=x[sort[i,m]]-x[i];
      vy1:=y[sort[i,m]]-y[i];
      vx2:=x[sort[i,k]]-x[i];
      vy2:=y[sort[i,k]]-y[i];
      if (vx1*vy2-vy1*vx2>0) then
         m:=k;
    end;
    p:=sort[i,j];
    sort[i,j]:=sort[i,m];
    sort[i,m]:=p;
  end;
end;
{Для кожного i від 1 до n-2 шукається
 максимальний розв'язок, в якому
 найлівішою вершиною є (x[i],y[i]).}
for i:=1 to n-2 do begin
  {Для даного i шукається найбільша
   можлива кількість відмічених точок,
   які лежать на опуклій угору ламаній
   з початком в (x[i],y[i]).}
  if (i>1) then
     {Якщо цикл виконується не вперше,
      з масиву sort видаляються
      елементи, рівні i-1. Точки з
      номерами менше i при виконанні
      циклу вже не розглядатимуться.}
     for j:=i+2 to n do begin
       k:=1;
       while (sort[j,k]<>i-1) do
         Inc(k);
       for k:=k to j-i do
         sort[j,k]:=sort[j,k+1];
     end;
  {Побудова масиву seg для даного i.}
  for j:=i+1 to n do begin
    seg[j]:=1;
    vx1:=x[j]-x[i];
    vy1:=y[j]-y[i];
```

```
  for k:=i+1 to j-1 do begin
    vx2:=x[k]-x[i];
    vy2:=y[k]-y[i];
    if (vx1*vy2=vy1*vx2) then
       Inc(seg[j]);
  end;
end;
{Основний крок динамічного
 програмування, на якому для певного
 j обчислюються значення
 f[j,i+1], ..., f[j,j-1].}
for j:=i+2 to n do begin
  {Для всіх m, які в масиві sort[j]
   стоять перед i, то встановлюється
   f[j,m]=0, бо відповідної опуклої
   ламаної немає або вона є
   відрізком.}
  k:=1;
  while (sort[j,k]<>i) do begin
    f[j,sort[j,k]]:=0;
    Inc(k);
  end;
  vx1:=x[i]-x[j];
  vy1:=y[i]-y[j];
  vx2:=vx1;
  vy2:=vy1;
  {Якщо деякі точки (x[m],y[m])
   лежать на відрізку з кінцями
   (x[i],y[i]) та (x[j],y[j]), то
   для таких m встановлюється
   f[j,m]=0, бо відповідна опукла
   ламана є відрізком.}
  while ((k<=j-i)and
         (vx1*vy2-vy1*vx2=0)) do
  begin
    f[j,sort[j,k]]:=0;
    Inc(k);
    if (k<=j-i) then begin
       vx2:=x[sort[j,k]]-x[j];
       vy2:=y[sort[j,k]]-y[j];
    end;
  end;
  for k:=k to j-i do begin
    {Для тих m, для яких (x[m],y[m])
     лежить вище відрізка з кінцями
     (x[i],y[i]) та (x[j],y[j]),
     значення f[j,m] визначається на
     основі значень f[p,q] для p<j.}
    m:=sort[j,k]; {Індекс sort[j,k]
      для зручності позначається
      через m.}
    f[j,m]:=seg[m]+1; {Змінна f[j,m]
      отримує значення, яке
      відповідає ламаній з вершинами
      (x[i],y[i]), (x[m],y[m]) та
```

```
  (x[j],y[j]). У цьому виразі не
  враховані проміжні точки
  відрізка з кінцями (x[m],y[m])
  та (x[j],y[j]). Але якщо такі
  точки дають максимальну
  відповідь, вони будуть
  враховані в інших елементах
  масиву f.}
if (m>i+1) then begin
   vx1:=x[j]-x[m];
   vy1:=y[j]-y[m];
   p:=1;
   q:=1;
   while (q shl 1<m-i) do
     q:=q shl 1;
   {За допомогою двійкового
    пошуку знаходимо таке s, що
    точка (x[s],y[s]) утворює з
    точками (x[m],y[m]) та
    (x[j],y[j]) найбільший кут,
    який не більше 180 градусів.
    Вершиною кута має бути
    (x[m],y[m]), а сам кут
    відраховується за
    годинниковою стрілкою від
    променя, що починається в
    (x[m],y[m]) та проходить
    через (x[j],y[j]). Індекс s
    має задовольняти умову
    i<=s<m. У якості s
    виступатиме sort[m,p].}
   while (q<>0) do begin
     if (p+q<=m-i) then begin
        vx2:=
          x[sort[m,p+q]]-x[m];
        vy2:=
          y[sort[m,p+q]]-y[m];
        if (vx1*vy2-vy1*vx2<=0)
           then p:=p+q;
     end;
     q:=q shr 1;
   end;
   if (f[m,sort[m,p]]+1>f[j,m])
      then f[j,m]:=
             f[m,sort[m,p]]+1;
   {Якщо двійковий пошук дає
    точку, яка дозволяє
    побудувати ламану з більшою
    кількістю точок, значення
    f[j,m] поновлюється.}
end;
{Тепер елементу f[j,m] замість
 розв'язку для точок з номерами
 j та m присвоїться значення
 максимального серед розв'язків
```

```
      для точок з номерами j та s, де
      i<=s<j. Максимум обирається
      лише з тих f[j,s], для яких s у
      послідовності sort[j,1], ...,
      sort[j,j-1] знаходиться не далі
      за m. Саме це переприсвоєння
      дає можливість застосовувати
      двійковий пошук.}
     if (f[j,sort[j,k-1]]>f[j,m])
        then f[j,m]:=
               f[j,sort[j,k-1]];
  end;
  f[j,j]:=f[j,sort[j,j-i]]; {У
    елементі f[j,j] зберігатиметься
    максимальна кількість відмічених
    точок, які можуть знаходитися на
    опуклій вгору ламаній з кінцями
    (x[i],y[i]) та (x[j],y[j]).}
end;
{Аналогічний максимум шукається для
 ламаних, відмінних від відрізка та
 опуклих вниз. Розв'язок для ламаної
 з кінцями (x[i],y[i]) та
 (x[j],y[j]) знаходитиметься в
 елементі f[j,sort[j,1]].}
for j:=i+2 to n do begin
  k:=j-i;
  while (sort[j,k]<>i) do begin
    f[j,sort[j,k]]:=0;
    Dec(k);
  end;
  vx1:=x[i]-x[j];
  vy1:=y[i]-y[j];
  vx2:=vx1;
  vy2:=vy1;
  while ((k>0)and
         (vx1*vy2-vy1*vx2=0)) do
  begin
    f[j,sort[j,k]]:=0;
    Dec(k);
    if (k>0) then begin
       vx2:=x[sort[j,k]]-x[j];
       vy2:=y[sort[j,k]]-y[j];
    end;
  end;
  for k:=k downto 1 do begin
    m:=sort[j,k];
    f[j,m]:=seg[m]+1;
    if (m>i+1) then begin
       vx1:=x[j]-x[m];
       vy1:=y[j]-y[m];
       p:=m-i;
       q:=1;
       while (q shl 1<m-i) do
         q:=q shl 1;
```

```
              while (q<>0) do begin
                if (p>q) then begin
                   vx2:=
                     x[sort[m,p-q]]-x[m];
                   vy2:=
                     y[sort[m,p-q]]-y[m];
                   if (vx1*vy2-vy1*vx2>=0)
                      then p:=p-q;
                end;
                q:=q shr 1;
              end;
              if (f[m,sort[m,p]]+1>f[j,m])
                 then f[j,m]:=
                        f[m,sort[m,p]]+1;
            end;
            if (f[j,sort[j,k+1]]>f[j,m])
               then f[j,m]:=
                      f[j,sort[j,k+1]];
          end;
          {Перевірка, чи буде розв'язок для
           ламаних з кінцями в точках
           (x[i],y[i]) та (x[j],y[j])
           більшим за знайдений раніше.
           Розглядаються три варіанти:
           відрізок замість нижньої ламаної,
           відрізок замість верхньої ламаної
           та коли обидві ламані не є
           відрізками.}
          k:=sort[j,1];
          if ((f[j,j]+seg[j]>maxpoly)and
              (f[j,j]<>0)) then
             maxpoly:=f[j,j]+seg[j];
          if ((seg[j]+f[j,k]>maxpoly)and
              (f[j,k]<>0)) then
             maxpoly:=seg[j]+f[j,k];
          if ((f[j,j]+f[j,k]>maxpoly)and
              (f[j,j]<>0)and(f[j,k]<>0))
             then maxpoly:=f[j,j]+f[j,k];
        end;
      end;
    end;
    writeln(maxpoly); {Вивід відповіді.}
    {Закриття файлів вводу та виводу.}
    close(input);
    close(output);
END.
```

5. Вкладені множини

```
/* GCC */
```

```
#include <cstdio>

// Константи з умови задачі
const int MOD = 41;
const int MAX_N = 5;
const int MAX_M = 20;

// Звичніший синонім для 64-х бітових чисел
typedef long long int64;

// Вхідні данні
int N, M;
int h[MAX_N];

// f(n, Q) з розбору задачі
int64 f[MAX_N][1 + (1 << MAX_M) / MOD];

// Так як в C/C++ нумерація масивів
// починається з 0, то f(n, Q) з розбору
// задачі відповідатиме f[n-1][Q] в
// програмі, а h_n з умови задачі
// відповідатиме h[n-1] в програмі.

int main() {
  // Зчитуємо вхідні данні
  freopen("nested.in", "r", stdin);
  scanf("%d %d", &N, &M);
  for (int i = 0; i < N; i++)
    scanf("%d", &h[i]);

  // За означенням f: f(1, Q) = 1
  // (для всіх можливих Q).
  //
  // Тут mask -- двійкове представлення
  // усіх можливих множин S_1.
  for (int mask = h[0];
     mask < (1 << M);
     mask += MOD)
    f[0][mask / MOD] = 1;

  // Використовуючи рекурентну формулу
  // з розбору задачі обчислимо f(j, Q)
  // маючи f(j-1, Q').
  for (int j = 1; j < N; j++) {
    // Тут mask_prev -- двійкове
    // представлення Q'.
    for (int mask_prev = h[j-1];
       mask_prev < (1 << M);
       mask_prev += MOD) {

      // f_prev = f(j-1, Q').
      int64 f_prev =
        f[j-1][mask_prev / MOD];
```

```
      // Тут mask -- двійкове
      // представлення Q, підмножини Q'.
      // Використовується трюк для перебору
      // всіх підномжин заданої множини,
      // що описаний в розборі.
      for (int mask = mask_prev;
           ;
           mask = (mask - 1) & mask_prev) {
        if (mask % MOD == h[j])
          f[j][mask / MOD] += f_prev;
        if (mask == 0)
          break;
      }
    }
  }

  // Щоб отримати правильну відповідь
  // просумуємо f(N-1, Q) для  всіх
  // можливих Q.  Реалізація формули
  // [1] з розбору задачі.
  //
  // Тут total -- відповідь на задачу
  // mask -- двійкове представлення усіх
  // можливих множин S_n.
  int64 total = 0;
  for (int mask = h[N-1];
     mask < (1 << M);
     mask += MOD)
    total += f[N-1][mask / MOD];

  // Виводимо результат
  freopen("nested.out", "w", stdout);
  printf("%lld\n", total);

  return 0;
}
```

6. Портали

```
/* GCC */

#include <stdio.h>
#include <vector>
#define maxn 1000

using namespace std;

int n, m, p[maxn], i, a, b, cur, sum,
                            minimum, count;
vector<int> g[maxn];    // Граф:
// g[i] міститиме динамічний масив номерів
// вершин, суміжних із вершиною i
// (індексація з нуля)
bool visited[maxn]; // visited[i] матиме
// значення true тоді й тільки тоді, коли
```

```
// ми вже відвідали вершину i під час
// пошуку в глибину

// Рекурсивна функція для пошуку в глибину
// vert — номер вершини
// Повертає мінімальне з чисел на піддереві
// пошуку, починаючи з вершини vert
int dfs(int vert)
{
    visited[vert] = true;   // Позначаємо
// вершину як уже відвідану, щоб уникнути
// повторного її проходження
    int result = p[vert];   // Початкове
// значення результату, який поверне
// функція, — число у вершині, з якої
// починається пошук
    for (int i = 0; i < g[vert].size();
                                        i++)
// Для всіх вершин, суміжних із даною
        if (! visited[g[vert][i]])  // Якщо
// суміжну вершину ще не було відвідано
            result = min(result,
                         dfs(g[vert][i]));
// Відвідуємо суміжну вершину і оновлюємо
// найменше значення на піддереві
    return result;
}

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

    scanf("%d %d", &n, &m);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &p[i]);
        visited[i] = false; // Позначаємо
// кожну вершину як іще не відвідану
    }
    for (i = 0; i < m; i++)
    {
        scanf("%d %d", &a, &b);
        g[a - 1].push_back(b - 1);
// Додаємо ребра до графа, враховуючи,
// що індексація у задачі з одиниці,
// а у програмі — з нуля
        g[b - 1].push_back(a - 1);
    }
    sum = count = 0;    // Ініціалізуємо
// суму найменших чисел на всіх компонентах
// зв'язності і кількість компонент
// зв'язності графа
    for (i = 0; i < n; i++)
// Для кожної вершини
```

```
        if (! visited[i])   // Якщо вершина
// не належить уже розглянутим компонентам
// зв'язності
        {
            cur = dfs(i);   // Здійснюємо
// пошук у глибину на компоненті
// зв'язності, до якої належить вершина i,
// та заносимо у змінну cur найменше число
// з цієї компоненти
            if (count == 0 ||
                                cur < minimum)
// Якщо це перша компонента зв'язності,
// яку ми розглядаємо, або якщо мінімальне
// число на ній менше за знайдені раніше
                minimum = cur;
// Оновлюємо значення змінної minimum,
// яка буде містити найменше з усіх чисел
            sum += cur; // Оновлюємо суму
// найменших чисел кожної компоненти
// зв'язності
            count++;    // Оновлюємо
// кількість компонент зв'язності
        }

    printf("%d\n", sum +
                    minimum * (count - 2));
// Виводимо відповідь, підраховану
// за формулою
    return 0;
}
```