ОЛІМПІАДА З ІНФОРМАТИКИ У МІСТІ КИЄВІ У 2015–2016 НАВЧАЛЬНОМУ РОЦІ

Мисак Данило Петрович,
керівник гуртка СШ №52 м. Києва.
Рудик Олександр Борисович,
доцент Київського університету імені Бориса Грінченка.

Продовження, початок у №2 за 2016 рік

IV. УМОВИ ЗАВДАНЬ III ЕТАПУ

1. Числа

Максимальна оцінка: 200 балів
Обмеження на час: 0,15 сек.
Обмеження на пам'ять: 256 МБ
Вхідний файл: numbers.in
Вихідний файл: numbers.out
Програма: numbers.*

Три натуральних числа назвемо *дружніми*, якщо вони попарно різні і добуток будь-яких двох із них націло ділиться на третє.

Завдання. За двома заданими натуральними числами встановіть кількість чисел, що утворюють з ними дружню трійку.

Вхідні дані

У єдиному рядку вхідного файлу вказано два різних натуральних числа, жодне з яких не перевищує 40 000.

Вихідні дані

Вихідний файл повинен містити єдине число: кількість натуральних чисел таких, що в сукупності з двома заданими вони утворюють трійку дружніх чисел.

Приклади

№	numbers.in	numbers.out
1	5 15	2
2	18 12	7

Пояснення до першого прикладу

Є рівно два числа, що в сукупності з числами 5 і 15 утворюють дружню трійку: це число 3 (бо 3×5 ділиться на 15, 3×15 ділиться на 5, а 5×15 ділиться на 3), а також число 75 (бо 5×15 ділиться на 75, 5×75 ділиться на 15, а 15×75 ділиться на 5).

2. Трикутник

Максимальна оцінка: 200 балів
Обмеження на час: 2,5 сек.
Обмеження на пам'ять: 256 МБ
Вхідний файл: triangle.in
Вихідний файл: triangle.out
Програма: triangle.*

Задано деякий набір відрізків цілих довжин.

Завдання. З відрізків заданого набору побудуйте трикутник із найбільшим або найменшим можливим периметром.

Вхідні дані

У першому рядку вхідного файлу вказано число n — кількість відрізків; ця кількість не менша за 4 і не перевищує 10^6. У другому рядку міститься n натуральних чисел, менших за 10^9, — довжини відрізків. У третьому рядку записано три латинських літери, що утворюють або слово **max**, або слово **min**: вони відповідають задачам пошуку найбільшого і найменшого периметра трикутника відповідно.

Вихідні дані

Вихідний файл повинен містити єдине число: залежно від поставленої у вхідних даних задачі або найбільший, або найменший можливий периметр трикутника, побудованого на деяких трьох відрізках із заданого набору. Якщо з жодних трьох відрізків побудувати трикутник неможливо, виведіть слово **none**.

Приклади

№	triangle.in	triangle.out
1	5 2 3 94 4 max	11
2	5 2 3 94 4 min	9
3	4 1 2 3 6 max	none

3. Граф

Максимальна оцінка: 200 балів
Обмеження на час: 1,5 сек.
Обмеження на пам'ять: 256 МБ
Вхідний файл: graph.in
Вихідний файл: graph.out
Програма: graph.*

У цій задачі ми говоримо про неорієнтовані графи.

Неорієнтований граф — це множина, що містить елементи двох типів:

- *вершини* графа — елементи довільної природи;
- *ребра* графа — неупорядковані пари вершин графа.

Традиційно вершини графа зображають точками на площині, а ребра — відрізками або кривими, що сполучають зображення відповідних вершин графа.

Інформацію про графи можна зберігати в пам'яті комп'ютера багатьма різними способами. Наприклад — простим переліком ребер, тобто пар номерів сполучених між собою вершин. Або з допомогою т. зв. списків суміжності: для кожної вершини зберігаємо в окремому масиві номери вершин, сполучених із даною вершиною ребром (тобто *суміжних* вершин).

Одержати з першого варіанта подання графа (переліку ребер) другий (списки суміжності) нескладно:

1. Запровадимо для кожної вершини окремий масив суміжних вершин, який спершу є порожнім.

2. Перебираючи ребра від першого до останнього, робимо таке: якщо на даному кроці ми розглядаємо ребро між вершинами A та B, то до списку суміжності ве-

ршини A додаємо вершину B, а до списку суміжності вершини B додаємо вершину A.

3. Закінчивши перебір ребер, матимемо подання графа списками суміжності.

Завдання. За списками суміжності, утвореними за допомогою вищенаведеного алгоритму, відновіть порядок, у якому було перераховано ребра графа в початковому переліку.

Вхідні дані

У першому рядку вхідного файлу вказано натуральне число n, що не перевищує 1000, — кількість вершин графа. Вершини занумеровано натуральними числами від 1 до n. Далі йдуть n рядків: у k-му з них перераховано без повторів номери вершин, суміжних з k-ю. Усі такі номери відмінні від самого числа k. Відомо також, що кожна вершина сполучена принаймні з однією іншою і якщо в списку суміжності вершини k є вершина l, то і в списку суміжності вершини l є вершина k.

Вихідні дані

У вихідний файл виведіть ребра графа в тому порядку, у якому вони йшли у початковому наборі. Одному ребру має відповідати один рядок: першим у рядку слід зазначити менший з двох номерів сполучених між собою вершин, другим — більший. Якщо є декілька варіантів порядку, у якому могло бути перераховано ребра, виведіть будь-який із них. Вхідні дані гарантують, що принаймні один такий порядок існує.

Приклад

graph.in	graph.out
4 2 4 4 1 4 2 3 1	2 4 1 2 3 4 1 4

4. Табори

Максимальна оцінка: 200 балів
Обмеження на час: 1 сек.
Обмеження на пам'ять: 32 МБ
Вхідний файл: camps.in
Вихідний файл: camps.out
Програма: camps.*

Група альпіністів планує піднятися на вершину гори стежкою, з якої неможливо звернути. Розташування на маршруті однозначно визначено відстанню від старту. Телевізійна компанія, яка профінансувала підйом, обумовила, що протягом підйому група розіб'є n *різних* таборів для ночівлі й відпочинку, де пройдуть відеозйомки з життя альпіністів.

Завдання. Визначити, скількома способами можна встановити n таборів для ночівлі й відпочинку за умови, що і s — довжина стежки, і $s_1, s_2, \ldots, s_n$ — відстані від старту до точок розташування таборів (у певних одиницях вимірювання довжини) можна подати різними натуральними числами, $0<s_1<s_2< \ldots <s_n<s$.

Вхідні дані

Вхідний файл містить у вказаному порядку два *натуральні* числа:

s — довжина стежки;

n — кількість таборів, $n<s<200\,000$.

Вихідні дані

Вихідний файл має містити одне натуральне число — шукану кількість способів розташування таборів. Відомо, що кількість цифр десяткового запису цієї кількості не перевищує 45 000.

Приклади

camps.in	camps.out
32	1
3 1	2
99 49	254776122589808569027304286 00

5. Дороги

Максимальна оцінка: 200 балів
Обмеження на час: 9 сек.
Обмеження на пам'ять: 256 МБ
Вхідний файл: roads.in
Вихідний файл: roads.out
Програма: roads.*

Керівництво будь-якої держави має постійно піклуватися про дороги, які сполучають усі населені пункти держави у єдине ціле і слугують як військовій, так і економічній могутності держави.

Завдання. Визначити набір доріг, за допомогою яких можна потрапити з будь-якого населеного пункту у будь-який інший населений пункт і утримання яких обходиться якомога дешевше для державної скарбниці. Рух кожною дорогою — двосторонній.

Вхідні дані

Перший рядок вхідного файлу містить у вказаному порядку два *натуральні* числа:

n — кількість населених пунктів, занумерованих натуральними числами від 1 до n включно, $3\le n\le 4096$;

m — кількість усіх доріг, $3\le m\le 8\,386\,560$.

Кожний з наступних m рядків містить опис певного шляху:

j, k — номери населених пунктів, сполучених дорогою, $1\le j, k\le n, j\ne k$;

v — натуральне число — вартість утримання дороги в солідусах (римських золотих монетах), $1\le v\le 654321$.

Вихідні дані

Єдиний рядок вихідного файлу містить номери доріг *у порядку зчитування*, що утворюють потрібний набір. Порядок цих номерів — довільний. Вхідні дані гарантують існування розв'язку. Якщо розв'язків кілька, потрібно записати будь-який, але лише один.

Приклад

roads.in	roads.out
3 3 1 2 3 2 3 1 3 1 2	2 3

6. Парасолька

Максимальна оцінка: 200 балів
Обмеження на час: 0,15 сек.
Обмеження на пам'ять: 32 МБ
Вхідний файл: umbrella.in
Вихідний файл: umbrella.out
Програма: umbrella.*

Фірма-виробник парасольок з рекламною метою обіцяє покупцям унікальність кожної виробленої нею парасольки. Цього планують досягнути за рахунок різного розфарбування секторів купола, які неможливо сумістити обертанням навколо стержня парасолі. Усі сектори купола однакової форми.

Завдання

Визначити кількість різних видів розфарбувань купола парасольки.

Вхідні дані

Вхідний файл містить у вказаному порядку *натуральні* числа:

n — кількість секторів купола парасолі, $2 \le n \le 19$;

m — кількість можливих кольорів, $2 \le m \le 19$.

У 55% тестів $n \le 6$. У 70% тестів $m \le 6$.

Вихідні дані

Вихідний файл має містити одне натуральне число — шукану кількість різних способів розфарбування куполу парасолі. Вхідні дані гарантують, що добуток n і цієї шуканої кількості не перевищує 10^{19}.

Приклади

umbrella.in	umbrella.out
3 2	4
3 4	24

V. ІДЕЇ РОЗВ'ЯЗАННЯ ЗАВДАНЬ III ЕТАПУ

1. Числа

Позначимо задані у вхідному файлі два числа через a та b.

Ідейно найпростіший шлях розв'язати задачу — скориставшись тим, що жодне з чисел у дружній трійці не може перевищувати добутку двох інших, перебрати всі числа від 1 до ab і безпосередньо порахувати, скільки з них дають разом із числами a та b дружню трійку. Утім, такий розв'язок набере неповний бал, оскільки не витримає обмеження на час.

Щоб заробити повний бал, необхідно розкласти обидва числа на прості множники. Якщо деяке просте число p входить у степені p_a у розклад числа a і в степені p_b у розклад числа b, то в розклад третього числа дружньої трійки число p не може входити у степені, більшому за p_a+p_b (інакше добуток ab не поділиться на третє число). Водночас цей степінь не може бути меншим за $|p_a-p_b|$ (інакше добуток одного з чисел a та b і третього числа не поділиться на інше з чисел a та b). При цьому одне з чисел p_a та p_b може бути й нулем (таке станеться, коли одне з чисел a та b ділиться на просте число p, а інше — ні). Якщо взяти всі прості числа, на які ділиться бодай одне з чисел a та b, то ці обмеження на степінь їх входження будуть водночас і достатніми для того, щоб третє число дійсно становило дружню трійку з двома заданими. На жодне ж інше просте число воно ділитися не може.

Отже, відповідь — добуток чисел вигляду

$$(p_a+p_b)-|p_a-p_b|+1$$

по всіх простих числах p, що входять у розклад хоча б одного з чисел a та b. Щоб порахувати цей добуток, достатньо, наприклад, перебрати всі числа від 2 до одного з чисел a та b у порядку збільшення: якщо хоча б одне з чисел a та b поділилося на чергове число, яке ми розглядаємо, то ділимо їх, поки діляться, обчислюючи таким чином p_a та p_b, після чого домножуємо відповідь на відповідну величину.

Лишається лише врахувати те, що в отриману відповідь ми могли включити одне чи обидва числа a та b, якщо вони задовольняють умови дружньої трійки, окрім обмеження на те, що числа мають бути різними. Тому після відповідної перевірки кожного з цих чисел, можливо, повинні будемо відняти від результату одну чи дві одиниці.

2. Трикутник

Простий перебір усіх трійок заданих довжин, звичайно, не пройде великі тести за обмеженням на час. Тому використаємо інший підхід. Спершу ефективно відсортуємо заданий набір чисел. А тоді залежно від підзадачі, яку розв'язуємо, зауважимо таке.

- Якщо ми шукаємо трикутник найбільшого периметра, то його сторони будуть трьома послідовними елементами у відсортованому масиві. Якби це було не так, ми могли б узяти дві менших сторони і збільшити їх до тих двох, які передують у відсортованому масиві більшій стороні. При цьому трикутник не перестав би існувати (оскільки сума двох менших сторін лише збільшилась, а більша сторона залишилась тією самою), а от периметр тільки збільшився б. Отже, залишається за лінійний час знайти у відсортованому масиві найбільшу (найближчу до кінця масиву) трійку послідовних елементів, які задовольняють нерівність трикутника.
- Якщо ми шукаємо трикутник найменшого периметра, то дві його більші сторони будуть послідовними елементами у відсортованому масиві. Якби це було не так, ми могли б узяти більшу сторону і зменшити її до тієї довжини, що йде в масиві одразу після середньої сторони трикутника. При цьому трикутник не перестав би існувати (оскільки сума двох менших сторін не змінилася, а більша сторона стала коротшою), а от периметр тільки зменшився б. Отже, залишається перебрати всі пари послідовних елементів масиву і для кожної за допомогою двійкового пошуку у лівій частині масиву знайти найменше число, яке б у сумі із середньою стороною перевищувало довжину найбільшої. Далі порівняти периметри всіх знайдених трикутників і вивести найменший.

Враховуючи, що для обох підзадач потрібно здійснити сортування, складність алгоритму завжди становить $O(n \log n)$.

Наостанок додамо, що периметр трикутника може вийти за межі стандартної знакової 4-байтової змінної, тож для зберігання відповіді потрібно використовувати, скажімо, або змінну беззнакового типу, або 8-байтову.

3. Граф

Опишемо спочатку *структуру даних* для списків суміжності вершин.

Черга — структура даних, що працює за принципом «хто раніше прийшов, той раніше пішов». У черги є голова і хвіст. Основні операції з чергою такі:

- «Поставити в чергу» — додати елемент у «хвіст» черги. Довжину черги при цьому буде збільшено на одиницю;
- «Отримати з черги» — повернути значення елемента з голови і видалити його з черги, встановлюючи голову черги на наступний за видаленим елемент. Довжину черги при цьому буде зменшено на одиницю.

Можливо реалізувати чергу за допомогою масиву $a[1\ldots n]$, у якому зберігають дані, і двох додаткових змінних *head* і *tail*, у яких зберігають індекси відпо-

відно «голови» і «хвоста» черги. Основні операції можна записати кількома рядками:

«поставити в чергу»	«отримати з черги»

if tail=n then tail:=1
else tail:=tail+1;
a[tail]:=x;
	x:=a[head];
	if head=n then head:=1
	elsehead:=head+1;

Але у випадку багатьох черг різної довжини чергу доречно реалізувати з використанням динамічного розподілу пам'яті (або скористатися вже готовою структурою даних queue у C++). Крім черг для списків суміжності вершин, одну додаткову чергу **edges** потрібно використати для незаблокованих ребер (означення див. далі).

Опишемо *жадібний алгоритм* розв'язання задачі: послідовно виписувати ті ребра, які в таблиці суміжності не заблоковано жодним іншим (ще не виписаним) ребром, і видаляти їх із таблиці.

Незаблокованими вважати ребра (A, B) з такими властивостями:

- першою у списку суміжності вершини A іде на даний момент вершина B;
- першою у списку суміжності вершини B іде на даний момент вершина A.

Початковим набором незаблокованих ребер чергу edges можна заповнити просто під час зчитування вхідних даних або за допомогою додаткового лінійного проходу. Після цього чергу потрібно пройти «від голови до хвоста» і на кожному кроці:

- вивести й видалити з черги поточне ребро (A, B);
- видалити це саме ребро з таблиці суміжності — видалити перші вершини з черг вершин A і B;
- проаналізувати нові перші вершини в чергах A та B і в разі необхідності додати одне або два нових ребра до черги незаблокованих ребер.

Процес завершується тоді, коли в черзі більше немає ребер.

Як можна зрозуміти, час виконання алгоритму є пропорційним до кількості чисел у вхідному файлі.

4. Табори

Шукана кількість дорівнює кількості способів вибору n різних натуральних чисел з множини $\{1, 2, ..., s-1\}$ — кількості n-елементних підмножин $(s-1)$-елементної множини:

$$C_{s-1}^{n} = \binom{s-1}{n} = \frac{(s-1)!}{n!(s-1-n)!}.$$

Можна вибрати один з таких алгоритмів.

1. Скоротити на більший з факторіалів-добутків у знаменнику і, використавши алгоритм Евкліда, скоротити на найбільші спільні дільники множники чисельника і знаменника, щоб подати знаменник добутком одиниць. Це було прийнятно для розв'язання задачі «Каса» III (міського) етапу Всеукраїнської учнівської олімпіади з інформатики у місті Києві у 2015 році, але у даному випадку це не гарантує дотримання обмежень на час.

2. Попередньо обчисливши всі прості числа від 1 до s (решето Ератосфена), знайти степінь простого числа p у канонічному розкладі $m!$ на прості множники: $[m/p]+[m/p^2]+[m/p^3]+...$ і використати отриману інформацію для знаходження добутку. Саме цей підхід реалізовано в авторському розв'язанні цієї задачі.

На останньому етапі — множенні чисел базового типу — для повного розв'язання потрібно використати подання числа масивом його цифр — «довгу арифметику». Інакше останні 3 тести пройти неможливо.

5. Дороги

У термінах теорії графів дану задачу називають пошуком мінімального кістякового дерева — зв'язного графа без циклів, що містить усі вершини початкового графа і має найменшу суму ваг ребер серед усіх таких графів. Одним із способів розв'язання цієї задачі є жадібний *алгоритм Прима* (Prim):

1. Утворити дерево T_1, що містить:
   - одне ребро, що має *найменшу вагу* серед ребер початкового графа;
   - дві вершини — кінці цього ребра найменшої ваги та надати значення $k=1$.
2. Якщо існують вершини початкового графа G зовні останнього побудованого дерева T_k з ребрами $e_1, e_2, e_3, ..., e_k$, то зробити таке:
   - вибрати ребро e_{k+1} з найменшою вагою серед тих, у яких одна вершина належить до T_k, а інша вершина не належить;
   - утворити дерево T_{k+1} долученням до T_k вибраного ребра e_{k+1} і його вершини, яка не належить до T_k;
   - збільшити значення k на 1;
   - перейти на початок виконання пункту 2.
3. Якщо всі вершини початкового графа G належать до дерева T_k, то припинити побудову мінімального кістякового дерева.

Теорема. *Алгоритм Прима породжує мінімальне кістякове дерево.*

Доведення (від супротивного). Позначимо через $e_1, e_2, e_3, ..., e_n$ ребра дерева T_n у тому порядку, як їх вибрано згідно з алгоритмом Прима. Нехай k — найбільше таке ціле невід'ємне число, при якому $e_1, e_2, e_3, ..., e_k$ — ребра деякого мінімального кістякового дерева T'. Припустимо, що $k<n$. Приєднаємо ребро e_{k+1} до мінімального кістякового дерева T'. В так утвореному графі є цикл. Цикл містить ребро e, відмінне від e_{k+1}, що також має лише одну вершину в дереві T_k. Утворимо дерево T'', вилучивши з T' ребро e і долучивши ребро e_{k+1}. Згідно з алгоритмом Прима, вага e_{k+1} не перевищує вагу e, тому T'' є мінімальним каркасним деревом, що суперечить означенню k. Отже, $k=n$, T_n — мінімальне каркасне дерево.

Алгоритм Прима використано в авторському розв'язанні для калібрування системи тестування. На думку автора, завдання, найімовірніше очікувати використання саме цього алгоритму учасниками олімпіади. Приклад вихідного файлу може наштовхнути на думку використати саме цей жадібний алгоритм.

Не менше відомий *алгоритм Крускала* (Kruskal):

1. Надати множині E значення величини $\varnothing$ — порожньої множини.

2. Визначити, які з ребер початкового графа, що не належать до E, при долученні до E не утворюють циклів.

3. Якщо такі ребра є, то:

- серед цих ребер визначити «найлегше» — з найменшою вагою;
- долучити це ребро до множини E;
- перейти до виконання пункту 2.

4. Якщо таких ребер немає, то припинити побудову мінімального кістякового дерева E.

Учасникам відбірково-тренувальних зборів радимо звернути увагу на *алгоритм Борувки*, що він гарантує максимальну кількість балів. Він полягає у послідовному долученні ребер до лісу, що містить усі вершини, а ребра утворюють окремі дерева, доки ліс не перетвориться на одне дерево:

1. Нехай спочатку T — порожня множина ребер. Інакше кажучи, починаємо з лісу, до якого кожна вершина входить як окреме дерево (без ребер).

2. Поки T не є деревом (кількість ребер у T менше ніж $V-1$, де V — кількість вершин у графі), робити таке:

- для кожного компонента зв'язності, що є деревом поточного лісу, знайти одне найлегше ребро, що пов'язує цю складову з будь-якою іншою складовою зв'язності. Якщо таких ребер кілька, вибрати довільне, але лише одне;
- долучити всі знайдені ребра до множини T.

Отримана у результаті ребер T є мінімальним кістяковим деревом початкового графа. Уперше цей алгоритм опубліковано 1926 року Отакаром Борувкою як метод пошуку оптимальної електричної мережі у Моравії.

У програмі найголовніше організувати облік належності вершин до складових (дерев). Для цього в авторському розв'язанні використано лінійні масиви:

- c — утворений записами з такими полями:
  - p — вказівник на попередній елемент опису складової;
  - f — вказівник на перший елемент опису складової;
  - k — номер ребра, що належить до складової або яке потрібно долучити;
  - m — вага ребра з номером k;
- p — вказує на елемент c, що завершує опис складової, що містить дану вершину;
- v — вказує на елемент c, що завершує опис даної складової.

При цьому для опису динамічної структури не потрібно використовувати динамічний розподіл пам'яті.

6. Парасолька

Можливі шляхи розв'язання такі:

1. *Перебір* можливих розфарбувань, які неможливо сумістити поворотами навколо стержня парасолі.

2. *Подання шуканої кількості многочленом змінної* m:

a) на основі аналізу *учасником* різних способів розфарбування у термінах базових понять комбінаторики;

b) використовуючи *обчислення за допомогою ПК* на основі наслідків теореми Редфілда-Пойа.

Спосіб 1 найпрозоріший, але найменш результативний.

Спосіб 2a проілюструємо на прикладі n=4, використавши такі позначення:

$A_m^k = \dfrac{m!}{(m-k)!}$ — кількість розташувань без повторення по k елементів з m;

$C_m^k = \dfrac{m!}{k!(m-k)!}$ — кількість k-елементних підмножин m-елементної множини.

У наступній таблиці у стовпчику ліворуч подано різні схеми розфарбування секторів (цифрами 1, 2, 3, 4 позначено різні кольори розфарбування), у стовпчику праворуч — кількості відповідних розфарбувань.

11 11	$C_m^1 = m$
12 21	$C_m^2 = m(m-1)/2$
11 22	$C_m^2 = m(m-1)/2$
12 22	$A_m^2 = m(m-1)$
11 23	$A_m^3 = m(m-1)(m-2)$
12 31	$C_m^1 \; C_{m-1}^2 = m(m-1)(m-2)/2$
12 43	$A_m^4 / 4 = m(m-1)(m-2)(m-3)/4$

Використання результатів цього аналізу і схожого аналізу для n=2, 3, 5, 6 гарантує нарахування більше половини балів за задачу.

Теоретичні основи способу 2b детально з доведенням викладено за адресою http://www.kievoi.ippo.kubg.edu.ua/kievoi/lectures/polya.html. У розділі 6 цієї публікації згадано про *еквівалентну* популярну у навчальній літературі задачу про знаходження кількості різних намист, що складаються з намистинок x кольорів і містять по n намистинок. Два намиста вважають різними, якщо їх не можна сумістити рухами *у площині* (не плутати з рухами площини!). Подано і формулу для обчислення кількості намист:

$$\frac{1}{n}\sum_{d|n}\varphi(d)x^{n/d}.$$

У цій формулі:

- додавання Σ здійснюють за всіма d — натуральними дільниками числа n;
- $\varphi(d)$ — функція Ейлера — кількість натуральних чисел, які менші від числа d і взаємно прості з ним, — дорівнює добутку d та різниць вигляду $(1-1/p)$, де p — простий дільник d.

Саме цей спосіб (m підставити у формулу вище замість x) гарантує успішне проходження усіх тестів. Його реалізовано в авторському розв'язанні. Він прийнятний для істотно більших значень n та m, якщо використати подання числа масивом його цифр («довга арифметика») або знаходити не саму кількість, а остача від ділення на число базового типу мови програмування.

Для 60 тестів використано такі множини значень:

n — {2, 3, 4, 5, 6, 11, 12, 18, 19};

m — {2, 3, 4, 5, 6, 11, 19}.

Не використано лише пари (18, 19), (19, 11) і (19, 19), при яких сума Σ — поза діапазоном значень базового типу qword мови Pascal.

VI. АВТОРСЬКІ РОЗВ'ЯЗАННЯ ЗАВДАНЬ III ЕТАПУ

1. Числа

```
/* GCC */
#include <stdio.h>
#include <algorithm>
using namespace std;
// Повертає показник найбільшого степеня
// числа divisor, на який ділиться число n.
// Паралельно ділить число n на цей
// степінь.
inline int getPowerAndDivide(int &n,
                  int divisor)
{
  int power = 0;
  while (n % divisor == 0)
  {
    power++;
    n /= divisor;
  }
  return power;
}
// Перевіряє, чи три заданих числа утворю-
// ють дружню трійку, ігноруючи те, чи вони
// різні.
inline bool check(int a, int b, int c)
{
  return a * b % c == 0 && a * c % b == 0
&& b * c % a == 0;
}
int main()
{
freopen(«numbers.in», «r», stdin);
freopen(«numbers.out», «w», stdout);
  int a, b;
scanf(«%d %d», &a, &b);
  int ans = 1;
  int m = min(a, b), curA = a, curB = b;
  for (int divisor = 2; divisor <= m;
                divisor++)
  {
    int powerA = getPowerAndDivide(curA, divisor);
    int powerB = getPowerAndDivide(curB, divisor);
    // Якщо принаймні одне з чисел
    // powerA та powerB виявилося нену-
// льовим, то поточний дільник
    // (divisor) є за побудовою алгоритму
    //простим числом.
    ans *= (powerA + powerB)–
          abs(powerA–powerB) + 1;
}
  // Поправка на те, що жодні два числа
// не повинні збігатися:
  if (check(a, a, b))
    ans––;
  if (check(a, b, b))
    ans––;
printf(«%d\n», ans);
  return 0;
}
```

2. Трикутник

```
/* GCC */
#define maxN 1000000 // Найбільша можлива
            // кількість відрізків
#include <stdio.h>
#include <algorithm>
using namespace std;
unsigned int a[maxN]; // Довжини відрізків
int n;
int main()
{
freopen(«triangle.in», «r», stdin);
freopen(«triangle.out», «w», stdout);
scanf(«%d\n», &n);
  for (int i = 0; i < n; i++)
scanf(«%u», &a[i]);
scanf(«\n»);
  char c[4];
scanf(«%3s», c);
  bool findMax = c[1] == 'a';
sort(a, a + n);
  if (findMax) // Шукаємо найбільший
  {       // периметр
    for (int i = n–3; i >= 0; i––)
      if (a[i] + a[i + 1] > a[i + 2])
      {
printf(«%u\n», a[i] +
a[i + 1] + a[i + 2]);
        return 0;
      }
  } else // Шукаємо найменший периметр
  {
    unsigned int curMin = 0;
    for (int i = 2; i < n; i++)
{
      // Шукаємо найменшу сторону
      // трикутника, яка б у сумі з
      // (i–1)-ю перевищила б і-ту:
int index = lower_bound(
  a, a + i–1, a[i]–a[i–1] + 1)–a;
      if (index < i–1) // Якщо це
      // вдалося, порівнюємо з
      // поточним мінімумом:
      {
        unsigned int curValue =
a[index] + a[i–1] + a[i];
        if (curMin == 0 ||
            curValue < curMin)
          curMin = curValue;
      }
    }
    if (curMin > 0)
    {
printf(«%u\n», curMin);
      return 0;
    }
  }
printf(«none\n»);
  return 0;
}
```

3. Граф

```
/* GCC */
#include <stdio.h>
#include <vector>
#include <queue>
using namespace std;
```

```
int n; // Кількість вершин графа
vector<queue<int>> all; // all[v–1] ––
// список суміжності вершини v
queue<pair<int, int>> edges; // Черга з
// ребер, які виводимо у вихідний файл
// Видаляє передній елемент зі списку сумі-
// жності вершини v та оновлює чергу ребер,
// які ми виводимо у вихідний файл, додавши
// туди нове доступне ребро, якщо таке
// з'явилося.
inline void pop(int v)
{
all[v–1].pop();
  if (!all[v–1].empty())
  {
    int adjacent = all[v–1].front();
    if (!all[adjacent–1].empty() &&
all[adjacent–1].front() == v)
edges.push(make_pair(
min(v, adjacent), max(v, adjacent)));
  }
}

int main()
{
freopen(«graph.in», «r», stdin);
freopen(«graph.out», «w», stdout);
scanf(«%d\n», &n);
  // Зчитуємо інформацію та заносимо
// початкові ребра в чергу edges:
for (int v = 1; v <= n; v++)
  {
    queue<int> adj; // Список вершин,
          // суміжних з поточною
    int num = 0; // Поточне зчитуване
          // число
    while (true)
    {
      char c;
scanf(«%c», &c); // Оскільки
       // кількість чисел у рядку невідома,
       // зчитуємо посимвольно
if (c>= '0' &&c<= '9')
       // Якщо черговий зчитаний
       // символ — цифра
         num = num * 10 + (c–'0');
else // Якщо черговий зчитаний
          // символ — пробіл або
          // перенесення рядка
       {
adj.push(num);
         num = 0;
       }
       if (c == '\n') // Якщо ми дійшли
        // до кінця рядка, виходимо з циклу
break;
    }
all.push_back(adj);
    // Якщо передня вершина списку су-
    // міжності уже опрацьована, і в її
    // списку суміжності поточна роз-
    // глядувана вершина теж є перед-
    // ньою, додаємо це ребро до черги:
if (adj.front() < v &&
all[adj.front()–1].front() == v)
edges.push(
         make_pair(adj.front(), v));
  }
  while (!edges.empty())
  {
    pair<int, int> edge = edges.front();
edges.pop();
printf(«%d %d\n», edge.first, edge.second);
pop(edge.first); pop(edge.second);
  }
  return 0;
}
```

4. Табори

```
{ Free Pascal }
const
  m=200000;          {Верхня межа n}
 mc=5000;
base=1000000000; {Основа системи числення}
var j,k,l,  {Лічильники}
h,n,   {Вхідні дані}
hn,    { =h-n-1}
nc, {Кількість цифр відповіді}
np, {Кількість знайдених простих чисел}
z:  longint;
 y,r: qword;
  o: text;
 s: array[1..m]  of boolean; {Число просте?}
 c: array[1..mc] of qword;   {Цифри відповіді}
 p: array[1..m]  of longint; {Прості числа}
pp: array[1..m]  of longint; {Степені простих
чисел у розкладі відповіді на прості множники}
BEGIN
assign(o,'camps.in');    reset(o);
read(o,h,n);       close(o);
assign(o,'camps.out');  rewrite(o);
dec(h);
hn:=h-n;
          {Пошук простих чисел}
for j:=1 to h do s[j]:=true;
for l:=2 to h div 2 do if s[l] then
for k:=2 to h div l do s[k*l]:=false;
np:=0;
for j:=2 to h do
if s[j] then begin
inc(np);
p[np]:=j;
pp[np]:=0 end;
    {Пошук степенів простих чисел}
for j:=1 to np do begin
 y:=1;
 repeat y:=y*p[j];
      z:=h div y;
inc(pp[j],z);
  until z=0;
 y:=1;
 repeat y:=y*p[j];
      z:=n div y;
dec(pp[j],z);
  until z=0;
```

```
 y:=1;
 repeat y:=y*p[j];
      z:=hn div y;
dec(pp[j],z);
  until z=0     end;
    {Пошук цифр відповіді}
nc:=1; c[1]:=1;
for j:=1 to np   do
for k:=1 to pp[j] do  begin
r:=0;
for l:=1 to nc do begin
   r:=r+p[j]*c[l];
 c[l]:=r mod base;
   r:=r div base  end;
while r>0 do     begin
inc(nc);
c[nc]:=r mod base;
    r:=r div base end end;
    {Запис відповіді}
write(o,c[nc]);
for j:=nc–1 downto 1 do
if c[j]<10 then write(o,'00000000',c[j])else
if c[j]<100 then write(o,'0000000',c[j])else
if c[j]<1000 then write(o,'000000',c[j])else
if c[j]<10000 then write(o,'00000',c[j])else
if c[j]<100000 then write(o,'0000',c[j])else
if c[j]<1000000 then write(o,'000',c[j])else
if c[j]<10000000 then write(o,'00',c[j])else
if c[j]<100000000 then write(o,'0',c[j])
           else write(o,   c[j]);
writeln(o);
close(o)
END.
```

5. Дороги

Алгоритм Прима

```
{ FreePascal }
{$I+} {Верхні межі кількості:}
constnv_max=4096;   {вершин}
ne_max=8386560;  {ребер}
type edge=record     {Ребро:}
 w: longint;        {вага}
a,b: word;         {вершини}
 n: longint         {номер}
        end;
var e: array[0..ne_max] of edge;
   u: array[1..nv_max] of boolean;
          {Кількості:}
  nv,       {вершин}
  nt: word;  {вершин дерева}
  ne,       {ребер}
   k: longint;
   o: text;
          {Обмін значень}
procedure swap(j,k: longint); BEGIN
e[0]:=e[j]; e[j]:=e[k];
           e[k]:=e[0]   END;
           {Упорядкування}
procedure QSort(l,r: longint);
 var x: edge;  i,j: longint;
              BEGIN
x:=e[l+random(r-l+1)];
i:=l;  j:=r;
while i<=j do  begin
 while e[i].w<x.w do inc(i);
 while e[j].w>x.w do dec(j);
 if i<=j then begin
swap(i,j); inc(i);
dec(j)  end end;
if l<j then qsort(l,j);
if i<r then qsort(i,r)  END;
              BEGIN
assign(o,'roads.in');
reset(o);
readln(o,nv,ne);
for k:=1 to ne do        begin
readln(o,e[k].a,e[k].b,e[k].w);
 e[k].n:=k             end;
close(o);
randomize;
qsort(1,ne);
for k:=1 to nv do u[k]:=false;
assign(o,'roads.out');
rewrite(o);
write(o,e[1].n);
u[e[1].a]:=true; u[e[1].b]:=true; nt:=2;
repeat  k:=1;
 repeat inc(k)
  until (u[e[k].a] and (not u[e[k].b]))
    or (u[e[k].b] and (not u[e[k].a]));
inc(nt);
write(o,' ',e[k].n);
u[e[k].a]:=true; u[e[k].b]:=true;
until nt=nv;
writeln(o);
close(o)  END.
```

Алгоритм Борувки

```
{ Free Pascal }
   {Верхні межі кількості:}
const nv_max=4096;   {вершин}
   ne_max=8386560;  {ребер}
max=654322;
type edge=record     {Ребро:}
      k,w: longint; {вага}
      a,b: word; {вершини}
                end;
var a,b,
          {Кількості:}
  nv,      {вершин}
np,      {складових}
 nte: word;  {використаних ребер}
ne,      {ребер}
i,j,k,l,m: longint;
     o: text;
e: array[0..ne_max] of edge;  {Ребра}
           {Частини складової}
c: array[1..nv_max] of record
        p,        {попередник}
        f: word;      {початок}
k,        {нове ребро}
        m:    {вага нового ребра}
              longint; end;
{Вказівники на початок опису складової}
p,              {для вершин}
```

```
v:                {для складових}
array[1..nv_max] of word;
                  BEGIN
assign(o,'roads.in');
reset(o);
readln(o,nv,ne);
for k:=1 to ne do
readln(o,e[k].a,e[k].b,e[k].w);
close (o);
assign(o,'roads.out');
rewrite(o);
for k:=1 to nv do begin
 p[k]:=k;
 v[k]:=k;
 c[k].p:=0;
 c[k].f:=k;
 c[k].m:=max      end;
nte:=0; np:=nv;
i:=0;  {Кількість складових з
відомими наступними ребрами}
REPEAT
{Пошук наступних ребер найменшої ваги}
 k:=0;  {Лічильник розглянутих ребер}
 repeat inc(k);
if  p[e[k].a]<>p[e[k].b]  then begin
   if c[p[e[k].a]].m >e[k].w then begin
     if c[p[e[k].a]].m=max then inc(i);
c[p[e[k].a]].m:=e[k].w;
c[p[e[k].a]].k:=k          end;
   if c[p[e[k].b]].m >e[k].w then begin
     if c[p[e[k].b]].m=max then inc(i);
c[p[e[k].b]].m:=e[k].w;
c[p[e[k].b]].k:=k       end end;
 until {i=np} k=ne;
l:=0; {Кількість складових з
невідомими наступними ребрами}
repeat
inc(l);
  if c[v[l]].m<>max then BEGIN
dec(i);
inc(nte);
    j:=c[v[l]].k;
    if v[l]=p[e[j].b]
    then begin a:=e[j].a; b:=e[j].b end
    else begin a:=e[j].b; b:=e[j].a end;
if((p[e[c[p[a]].k].a]=p[e[c[p[b]].k].a])
and(p[e[c[p[a]].k].b]=p[e[c[p[b]].k].b]))
or((p[e[c[p[a]].k].a]=p[e[c[p[b]].k].b])
and(p[e[c[p[a]].k].b]=p[e[c[p[b]].k].a]))
    then      begin
c[p[a]].m:=max;
dec(i)    end;
    c[c[p[a]].f].p:=p[b];
c[p[a]].f :=c[p[b]].f;
    j:=p[b];
    repeat p[e[c[j].k].a]:=p[a];
p[e[c[j].k].b]:=p[a];
        j:= c[j].p
until  j=0;
    v[l]:=v[np];
dec(l);
dec(np);          END;
 until (nte=nv-1) or (l=np);
until nte=nv-1;
k:=c[p[1]].p;
write(o,c[k].k);
repeat k:=c[k].p;
write(o,' ',c[k].k);
 until c[k].p=0;
writeln(o);
close(o)  END.
```

6. Парасолька

```
{ Free Pascal }
const
  m=100000;          {Верхня межа n}
var j,k,l,  {Лічильники}
   n,c,   {Вхідні дані}
   np: word; {Кількість простих дільників n}
answer: qword;
   o: text;
 s: array[1..m]  of boolean; {Число просте?}
 p: array[1..m,0..64]  of longint; {Прості числа}
pn,pd: array[1..m] of byte;    {Степені
простих чисел у розкладі n і відповіді
на прості множники}
procedure step(i: word);
var j,k,l,d: word;  f: qword;
                  BEGIN
for j:=0 to pn[i] do   begin
 pd[i]:=j;
 if i<np then step(i+1) else begin
 d:=1;
 for k:=1 to np do d:=d*p[k,pd[k]];
 f:=d;
 for k:=1 to np do if pd[k]>0 then
f:=(f div p[k,1])*(p[k,1]-1);
 for k:=1 to n div d do f:=f*c;
 answer:=answer+f     end end END;
                  BEGIN
assign(o,'umbrella.in');    reset(o);
read(o,n,c);            close(o);
assign(o,'umbrella.out');  rewrite(o);
          {Пошук простих чисел}
for j:=1 to n do s[j]:=true;
for l:=2 to n div 2 do if s[l] then
for k:=2 to n div l do s[k*l]:=false;
{Розкладання n на прості множники}
np:=0;
for j:=2 to n do
if s[j] and (n mod j = 0) then begin
inc(np); p[np,0]:=1; pn[np]:=0;
   l:=n;
   while l mod j = 0 do begin
    l:=l div j;
inc(pn[np]);
p[np,pn[np]]:=p[np,pn[np]-1]*j
end  end;
answer:=0;
step(1);
writeln(o,answer div n);
close(o)
END.
```