

Збірник задач та їхніх розв'язків обласних,
районних (міських) олімпіад з інформатики
Черкаської області (2012–20... р.р.)

Сама сторінка (оформлення) буде замінена на титульну сторінку,
набрану якимись іншими засобами.

Цю сторінку замінити на сторінку з ББК, грифом,
даними про авторів та рецензентів, набраною якимись
іншими засобами

Зміст

1	Передмова	4
2	Огляд особливостей «олімпіадного» та «алгоритмічного» програмування	5
	Які бувають формати задач? (5) Звідки беруться оцінки, проміжні між 0 і максимумом? (6) Що таке потестове та поблокове (блочне) оцінювання? (6) Що таке «чекери» (checkers) та «інтерактори» (interactors)? (6) Чи можна здавати багато розв'язків однієї задачі? (7) Як працювати у ejudge? (7) Які бувають вердикти (статуси, результати) перевірки ejudge-ем? (8) Які допустимі мови програмування? (10) Чому нема Delphi? (11) Які ще є рекомендації щодо мов програмування? (11) Щодо особливостей та переваг сайту ideone.com. (11) Чи потрібно доводити правильність алгоритмів? (12) Що означають записи, подібні до « $O(n^2)$ »? (12) Що таке діапазон (цілочисельного) типу? Переповнення? (13) Формат з рухомою комою (плаваючою крапкою, floating point) та похибка. (14) Щодо швидкості введення/виведення. (15) Щодо статичної, динамічної та стекової пам'яті, а також переповнення стеку. (15)	
3	Задачі та їхні розв'язки	17
3.1	II (районний/міський) етап 2012/13 н. р.	17
	A «Новий рік» (17) B «Матриця» (18) C «Двійкова система числення» (19) D «Лабіринт» (20)	
3.2	III (обласний) етап 2012/13 н. р.	22
	A «Квартири — flats» (22) B «Послідовність — series» (23) C «Декадні числа — dnumbers» (24) D «Кава — coffee» (25)	
3.3	Обласна інтернет-олімпіада 2013/14 н. р.	27
	A «Тор» (27) B «Паркет–1» (28) C «Сума квадратів» (29) D «Дільники» (30)	
3.4	II (районний/міський) етап 2013/14 н. р.	31
	A «Електричка» (31) B «Цифрові ріки» (32) C «Логічний куб» (32) D «Всюдисущі числа» (33)	
3.5	Дистанційний тур III (обласного) етапу 2013/14 н. р.	35
	A «ISBN» (35) B «Точні квадрати» (36) C «Ложбан» (37) D «Графічний пароль» (38)	
3.6	Обласна інтернет-олімпіада 2014/15 н. р.	38
	A «Три кола» (39) B «Перевезення вантажу» (39) C «Коло і точки» (40)	
3.7	II (районний/міський) етап 2014/15 н. р.	42
	A «Цифра» (42) B «Піраміда» (43) C «Ко-анagramічно-прості» (44) D «Хмарочоси» (46)	
3.8	III (обласний) етап 2014/15 н. р.	47
	A «Гірлянда — garland» (47) B «Прямокутники — rectangles» (48) C «Генератор паролів — password» (49) D «Зámok — castle» (51)	
3.9	Обласна інтернет-олімпіада 2015/16 н. р.	53
	A «Високосні роки» (53) B «Фасування олії» (54) C «Нестандартне рівняння» (55) D «П'ять неділь у місяці» (55)	
3.10	II (районний/міський) етап 2015/16 н. р.	57
	A «Купе» (57) B «Гра» (58) C «Сучасне мистецтво» (59) D «Кількість дільників на проміжку» (61)	
3.11	Обласна інтернет-олімпіада 2016/17 н. р.	62
	A «Квартира» (62) B «Максимальний добуток» (63) C «Гра “Вгадай число”» (64) D «Кількість прямокутників» (66)	
3.12	II (районний/міський) етап 2016/17 н. р.	67
	A «Квартали» (67) B «Квадрат і круг» (68) C «Сповзання кубиків» (69) D «Кратний куб» (71)	
3.13	Обласна інтернет-олімпіада 2017/18 н. р.	72
	A «Патрульні групи–1» (72) B «Патрульні групи–2» (73) C «День програміста» (74) D «Відстань між мінімумом та максимумом–1» (74) E «Відстань між мінімумом та максимумом–2» (75)	
3.14	II (районний/міський) етап 2017/18 н. р.	76
	A «Фарбування паралелепіпеда» (76) B «Точки та круг» (77) C «Цікаві числа» (79) D «Дірка і Пробка» (80)	
3.15	Обласна інтернет-олімпіада 2018/19 н. р.	82
	A «Кількість секунд» (82) B «Файлова система» (83) C «Палички — інтерактивна гра» (84) D «Квантифікація предиката» (86)	
3.16	II (районний/міський) етап 2018/19 н. р.	88
	A «Кількість шляхів» (88) B «Відтинання квадратів» (89) C «Остання ненульова цифра» (90) D «Палички, хто перемаже?» (93) E «Палички, інтерактивна гра» (96)	
3.17	Обласна інтернет-олімпіада 2018/19 н. р.	98
	A «Змії» (98) B «Ракета» (99) C «Checker для “Ракети”» (100) D «Preview» (101)	

Збірник задач та їхніх розв'язків обласних, районних (міських)
олімпіад з інформатики Черкаської області (2012–20... р.р.)

3.18 II (районний/міський) етап 2019/20 н. р.	104
А «ККД» (104) В «Спільні дотичні» (105) С «Прямокутні суми» (106) D «Прямокутні максимуми» (109)	
3.19 Дорішування деяких задач II етапу (14.12.2019) в режимі задачі частини коду	114
С «Прямокутні суми» (114) D «Прямокутні максимуми» (120)	
3.20 Обласна інтернет-олімпіада 2020/21 н. р.	121
А «Василько та циркуль-1» (121) В «Дуже важливі числа» (122) С «Василько та циркуль-2» (122)	
3.21 II (районний/міський) етап 2021/22 н. р. (лише м. Черкаси)	127
А «Знакозмінна сума» (127) В «Гра "WALLCRSH"» (128) С «Круглі дужки» (129) D «Паралелепіеди» (130)	

4 Список використаних джерел	136
-------------------------------------	------------

1 Передмова

Цей збірник містить задачі олімпіад з інформатики (програмування), що відбувалися у Черкаській області з 2012/13 по 2021/22 навчальні роки. Точніше, він охоплює тури обласних інтернет-олімпіад (крім 2012/13 н. р., котра була, але не потрапила в цей збірник, та 2021/22 н. р., котрої не було), другі (районні/міські) етапи Всеукраїнської олімпіади з інформатики (крім 2020/21 н. р., коли його не було, а у 2020/21 н. р. II етап м. Черкаси відрізнявся від II етапу решти області, й тут наведено той, що був у м. Черкаси), та ті з третіх (обласних) етапів, у яких завдання формувалися черкаськими авторами, але не охоплює ті з третіх (обласних) етапів, у яких завдання розроблялися іншими колективами авторів і надсилалися централізовано. Для розглянутих турів, до кожної задачі наведені умова та вказівки щодо її розв'язування.

Умови задач, що входять до збірника, формувалися авторським колективом у складі: Порубльов І. М. (ЧНУ), Богатирьов О. О. (ЧНУ), Шемшур В. М. (ЧОІПОПП), Фурник І. В. (ЧОІПОПП), Черненко Р. В. (випускник ЧНУ 2011 р.), Поліщук Д. І. (випускник ФіМЛі 2004 р.), Безпоясний Б. С. (ЧОІПОПП), Безпальчук В. М. (ЧНУ), Борзяк А. В. (випускник ФіМЛі 2016 р.; лише починаючи з осені 2016 р.). Умови задач наведені у вигляді, максимально близькому до того, який вони мали на самих етапах олімпіади (лише змінено форматування та виправлено окремі технічні помилки).

Більшість тексту розборів (пояснень) до задач написана Порубльовим І. М., з урахуванням порад вищезгаданих авторів (з яких варто особливо відзначити поради Поліщука Д. І.). Низку слушних пропозицій висловив Полосухін В. А. (випускник ФіМЛі 2014 р., який брав участь лише у підготовці розборів після відповідних турів). Є й кілька випадків, коли на пояснення помітно вплинули розв'язки, здані на перевірку в `ejudge.skipo.edu.ua`, чи задані засобами цього `ejudge` питання; тут окремої подяки заслуговує Граб Н. В. (вчителька Золотоніської СШІТ № 2). Тож, хто (чи то з учителів, чи то з учнів) бажає висловити зауваження та/або пропозиції — це, в розумній мірі, вітається, і, можливо, буде враховано при підготовці наступних версій цього збірника.

Збірник не призначений замінити підручник з програмування. Використовуючи конкретні задачі, нереально побудувати збалансований курс, що розглядає продуманий перелік тем; та й співвідношення обсягу задач та обсягу збірника унеможливорює детальний розгляд усіх потрібних у цих задачах алгоритмів. Тому основна увага у збірнику приділена поясненням нестандартних рішень у цих задачах. Коли задача зводиться до реалізації відомого алгоритму, зазвичай наводиться посилання на джерела в Інтернеті або в літературі. Рекомендується використовувати цей збірник у поєднанні із підручниками з програмування, монографіями та сайтами, де розглянуті ефективні алгоритми.

Тип паче збірник не є посібником з конкретної мови програмування. Значна частина пояснень сформульована без прив'язки до мови (словесно, математичними формулами, рисунками, тощо). Де цитати коду необхідні — найчастіше використані Pascal та C++, рідше Python та Java.

На олімпіаді з інформатики (програмування) розв'язком задачі учасником, як правило, є програма. Цей збірник містить трохи таких програм безпосередньо у своєму тексті, але значно більше таких програм-розв'язків доступні як посилання на сайт `ideone.com` (детальніше про його переваги та особливості див. стор. 11).

Збірник задач та їхніх розв'язків обласних, районних (міських)
олімпіад з інформатики Черкаської області (2012–20... р.р.)

Усі наведені у збірнику задачі (й не лише вони) доступні для автоматичної перевірки на сайті `ejudge.skipo.edu.ua` після простої безкоштовної реєстрації. Ця перевірка в цілому відображає і те, що було на відповідних турах, і написане у цьому збірнику. Разом з тим, за 9 років (з 2012 р., коли було перше змагання на сервері `ejudge.skipo.edu.ua`, по 2021 р., коли внесені наразі останні правки цей збірник) відбулося дві повні заміни апаратного забезпечення цього сервера. Через це не всі налаштування, які були актуальними на момент проведення відповідних олімпіад, актуальні зараз. З'явилися і ситуації, коли ті самі розв'язки, які раніше проходили менше тестів, почали проходити більше (бо на новішому «залізі» при використанні новішого компілятора той самий код почав працювати швидше, або тому, що внаслідок переходу з 32-бітової на 64-бітову архітектуру зменшилися чи зникли переповнення типів), і (рідше) ситуації, коли ті самі розв'язки почали проходити менше тестів (наприклад, тому, що внаслідок того ж переходу розв'язок почав перевищувати ліміт пам'яті). Зрідка трапляються навіть випадки, коли старий розв'язок, що нормально працював зі старою версією компілятора, тепер не компілюється, або зворотні. Тому, фактична робота `ejudge.skipo.edu.ua` може мати окремі незначні відхилення як від того, що було на відповідних турах, так і від написаного у збірнику, й на те нема ради.

2 Огляд особливостей «олімпіадного» та «алгоритмічного» програмування

Які бувають формати задач? На олімпіаді з інформатики (не інформаційних технологій, а інформатики; пишуть також «інформатики (програмування)») учасник отримує текстові задачі, розв'язком кожної з яких, як правило, повинна бути програма мовою програмування, що зчитує вхідні дані (з клавіатури чи файлу), обробляє їх згідно з умовою задачі та виводить результат (на екран чи у файл). Більшість розглянутих у збірнику задач саме такі.

У 2016–2019 р.р., на етапах, розглянутих у цьому збірнику, найпростіші задачі передбачають написання не програми, а виразу. Такий вид задач був запроваджений на основі досвіду деяких закордонних олімпіад, а не згідно традицій Всеукраїнської олімпіади з інформатики. Втім, серед простих задач і раніше траплялися такі, де програма зводилася до прочитання вхідних даних, обчислення простої формули й виведення її результату, так що відхилення від традицій і правил не значне. І при цьому робить ці задачі доступними ширшій частині учасників. Не секрет, що поєднання задач на написання програм з автоматичною перевіркою часом призводить до того, що не дурні, але не підготовлені учасники не можуть виконати у своїх програмах одночасно всі технічні вимоги, й у результаті отримують 0 балів за увесь тур. Що і сприймається негативно самими учасниками, і сумнівно з педагогічної точки зору. А задачі на написання виразу (а не програми) дещо пом'якшують цей момент.

Ще один формат — *інтерактивні* задачі, в яких програма учасника отримує вхідні дані не всі відразу, а частинами, у процесі спілкування з програмою-суперницею (т. зв. інтерактором), і подальші введення, отримувані програмою учасника, можуть залежати від попередніх її виведень. Це в деякому смислі робить олімпіадні задачі цікавішими й більш схожими на «не олімпіадне» програмування. Але, оскільки перевірка таких задач теж відбувається автоматично, пояснення формату спілкування з програмою-суперницею часто ще складніші для сприйняття, ніж пояснення форматів вхідних даних та результатів. Станом на 2021 р., збірник містить три такі задачі (стор. 64, 84, 96).

Починаючи з весни 2019 р., на III (обласному) та IV (фінальному) етапах набув поширення також формат, де учасник має писати і здавати не всю програму, а вказані її частини. Тобто, є чітко заданий (в умові задачі) перелік підпрограм із чітко заданими заголовками; треба реалізувати їх усі. Як правило, можна (якщо учасник вважає доцільним) реалізовувати також свої допоміжні підпрограми та/або оголошувати й використовувати свої глобальні змінні. Остаточна перевірка відбувається шляхом компіляції і запуску програми, в якій зібрані разом фрагменти, написані учасником, і фрагменти, написані журі — причому, журі може завчасно надавати учасникам самі ці фрагменти, а може й лише обмежену інформацію про них і приклад деякого *іншого* фрагменту. В цьому збірнику є лише

Збірник задач та їхніх розв'язків обласних, районних (міських)
олімпіад з інформатики Черкаської області (2012–20... р.р.)

дві (майже однакові) задачі такого вигляду, на стор. 114–121, причому навіть вони не грали ні на якому офіційному етапі в точності у такому вигляді.

Відомі ще деякі інші формати, але вони ні разу не використані в цьому збірнику, і на інших етапах олімпіади трапляються дуже рідко.

«Міркування на тему», виражені учасниками словесно, не приймаються і не перевіряються ще з кінця 1990-х рр.

Звідки беруться оцінки, проміжні між 0 і максимумом? Як правило, задачі перевіряють за допомогою *тестів*. Це розроблена (автором задачі або журі) сукупність прикладів, кожен з яких містить вхідні дані й відповідну їм правильну відповідь. Кожен розв'язок кожного учасника запускається на усіх цих прикладах, і ті з них, де програма вивела правильний результат, уклавшись у обмеження за часом та обсягом пам'яті, вважаються успішно пройденими.

Бувають програми, які виводять інколи правильні відповіді, інколи неправильні. Бувають правильні, але неефективні програми, які частину тестів проходять, а на решті отримують вердикт «Перевищено час роботи». І так далі. Звідси різні сукупності успішно пройдених тестів та різні бали.

Що таке потестове та поблокове (блочне) оцінювання? *Потестове* оцінювання означає, що бали нараховуються за кожен тест окремо: програма, що успішно пройшла тест, отримує заплановані за нього бали. *Поблокове* (воно ж *блочне*) — що тести групуються у блоки, і програма учасника отримує заплановані за блок бали, лише якщо успішно пройдено *всі* тести блока. Можуть бути (а може й не бути) також залежності «такий-то блок перевіряється й оцінюється лише за умови успішного проходження таких-то попередніх блоків». Приклади описів такого оцінювання можна бачити в багатьох задачах 2018/19 н. р.; найтипівішим, мабуть, є розділ «Оцінювання» зі стор. 93–94.

Якщо говорити про етапи, розглянуті в цьому збірнику, то поблокове оцінювання з'явилося восени 2018 р., а до того було лише потестове. Але якщо розглянути ті треті (обласні) етапи Всеукраїнської олімпіади з інформатики, які проводилися централізовано, то поблокове оцінювання поступово запроваджували від початку 2017 р. А на Міжнародних олімпіадах воно взагалі з 2011 р.

Поблокове оцінювання у середньому призводить до нижчих балів учасників, ніж потестове, але викликає менше нарікань вигляду «ну і за що отому поганому розв'язку аж так багато балів?» та/або «і оце така дрібна оптимізація змінює кількість балів?». Це різні наслідки однієї властивості: для потестового оцінювання значно більша, ніж для поблокового, ймовірність, що програма пройде якісь тести випадково — чи то зовсім випадково, чи то тому, що алгоритм розв'язує задачу лише при додаткових обмеженнях (наприклад, гарантовано правильний для квадратних масивів, але не для всіх прямокутних не квадратних), чи то тому, що деякі тести зовсім на грані то вкладаються, то не вкладаються у ліміт часу. Або, *якби* задача «Палички, хто переможе?» (стор. 93) оцінювалася потестово, то можна було б отримати десь із половину балів, взагалі не розв'язуючи задачу по суті, а просто здавши одну програму, яка завжди виводить 1, та іншу програму, яка завжди виводить 2. А поблокове оцінювання робить цей прийом практично неможливим — що, власне, й чесніше-то, і є однією з причин переходу від потестового оцінювання до поблокового. Ще два наведені в цьому збірнику приклади сумнівних наслідків потестового підходу описані в поясненнях, як набрати частину балів за задачі «Логічний куб» (стор. 33) та «Дірка і Пробка» (стор. 81).

Так що на рівень складності задачі (особливо у смислі отримання *частини* балів) істотно впливають не лише її умова та об'єктивна складність повного розв'язку, а й спосіб оцінювання.

Що таке «чекери» (checkers) та «інтерактори» (interactors)? У деяких задачах можливі різні правильні відповіді. Скажімо, у щойно згаданому «Логічному кубі» слід виводити мінімальний шлях, і різні шляхи (послідовності вершин) можуть мати однакову мінімальну довжину. На якісно підготовленій олімпіаді для таких задач пишуть т. зв. *чекери*, які перевіряють відповідь програми учасника за смислом. Чекери пишуть автори задач або члени журі, а не учасники, тож кому цікаво — шукайте деталі самостійно (зокрема, на codeforces.com), кому ні — можна обмежитися правилом, що у таких ситуаціях треба виводити будь-яку одну правильну відповідь.

Збірник задач та їхніх розв'язків обласних, районних (міських)
олімпіад з інформатики Черкаської області (2012–20... р.р.)

Для інтерактивних задач (див. стор. 5) усе ще трохи складніше, бо там є окремо чекери й окремо *інтерактори*, які проводять взаємодію («спілкування») з програмою учасника. Аналогічно, охочі можуть прочитати про це детальніше на `codeforces.com`, а кому це не цікаво, може обмежитися тим, що коли в інтерактивних задачах можуть бути різні правильні проміжні відповіді (наприклад, «ході»), слід виводити будь-який один з правильних варіантів.

Чи можна здавати багато розв'язків однієї задачі? Наразі на Всеукраїнській олімпіаді з інформатики учасники можуть багатократно здавати свої розв'язки у систему перевірки, визнавати результати (тривалість перевірки — від кількох секунд до кількох хвилин), і з усіх спроб автоматично вибирається найкращий результат. Обмеження на кількість спроб задачі існують, але, зазвичай, лояльні, як-то «до 60 на учасника сумарно по всіх задачах» (що, втім, при дуже марнотратному ставленні може бути й повністю розтринькано).

Інша справа, що правила не завжди були такими (на IV (фінальному) етапі вони такі з 2012 р., на етапах, розглянутих у цьому збірнику — з 2013 р.), й невідомо, як довго вони будуть такими. Існує точка зору, що ці правила «спонукають клацати замість думати», і на деяких інших змаганнях з програмування перевірка на *повному* наборі тестів відбувається *після* туру, а під час туру учасник визнає лише, чи проходить його програма всього кілька тестів. Зокрема, саме такими є і Всеукраїнська Інтернет-олімпіада з інформатики NetOI (`netoi.org.ua`), і менш офіційні, але більш популярні змагання, що проводяться сайтами `codeforces.com` та `topcoder.com`.

Є й системи оцінювання, коли зайві спроби задачі можливі, але якимось «штрафуються». Втім, на момент редагування цього збірника (2019 р.) у шкільних олімпіадах таких тенденцій не помітно.

Не типові, але технічно можливі й обмеження на кількість спроб задачі окремої задачі. Наприклад, на II етапі 2018/19 навч. року використовувалися одночасно і обмеження 60 спроб сумарно на всі задачі, і обмеження 5 спроб конкретно на задачу «Палички, інтерактивна гра».

Як працювати у ejudge? В Інтернеті багато описів роботи з ejudge. Наприклад, на офіційному сайті розробника, починаючи з `ejudge.ru/wiki/index.php/Веб-интерфейс_пользователя`. Ще один — `goo.gl/AxLPii`. На думку авторів збірника, для початку важливі такі питання.

1. За адресою `ejudge.skipo.edu.ua` має з'явитися сторінка з вибором доступних «змагань» (в т. ч. тренувань). При кліку на конкретне «змагання» з'являється пропозиція зайти (залогінитися).
 - (а) Якщо у Вас вже є акаунт на `ejudge.skipo.edu.ua`, спробуйте скористатися ним (акаунт може бути один на різні тренування). Для тренувань рекомендується використовувати акаунти, створені за Вашим запитом і підтверджені через електронну пошту. (А акаунти, роздані учасникам офіційних змагань організаторами, використовувати для тренувань *не* рекомендується.) Якщо акаунт, створений за Вашим запитом і прив'язаний до пошти, вже є, але Ви забули пароль — спробуйте автоматичне відновлення паролю. В окремих особливих випадках можна також звертатися (див. п. 4b) з проханням про ручне скидання пароля; але якщо акаунт не дуже цінний, а автоматичне відновлення не працює, можна просто створювати новий.
 - (б) Щоб зареєструватися, натисніть «Создать учётную запись», наберіть бажаний login та свій email (реально доступний). Знайдіть у пошті лист (він може потрапити у спам), перейдіть по вказаному лінку, наберіть вибраний Вами login, та пароль, що прийшов у листі. Якщо буде написано «Не зарегистрирован» — клікніть на «Подтвердить регистрацию».
 - (с) Натиснувши «Участвовать», Ви потрапите всередину тренування.
2. У більшості тренувань умови задач знаходяться в окремих документах за межами еджаджа. Щоб побачити їх, натисніть лінк «Условия». Коли умови задач відкриються в окремій вкладці, *не* закривайте вкладку з еджаджем.
3. Щоб здати задачу, потрібно:
 - (а) Добитися (наприклад, клікнувши на «Инфо»), щоб на сторінці були присутні маленькі прямокутнички з кодами задач (зазвичай — A, B, C, D).
 - (б) Клікнути на потрібний (A, B, ...) прямокутничок.
 - (с) Вибрати зі списку коло мітки «Язык» одну з доступних мов програмування.

- (d) Натиснувши кнопку вибору файлу навпроти мітки «Файл», знайдіть збережений на своєму комп'ютері файл з розв'язком. Це мусить бути вихідний (исходный, source, сирцевий) файл, з розширенням .pas, .crr, .c, тощо відповідно до мови програмування.
 - (e) Через деякий час (від кількох секунд до кількох хвилин) з'явиться вердикт з сумою набраних балів. При натисканні на лінк «Просмотр» стовпчика «Просмотреть протокол», можна побачити трохи детальніший протокол. Для задач, в яких треба здавати програму, для не зарахованих тестів буде вказано також вердикт (див. далі).
4. Можна задавати питання щодо незрозумілих моментів (типовий час надходження відповіді для тренувань — від кількох годин до кількох днів).
- (a) Рекомендований спосіб: якщо Ви благополучно увійшли всередину потрібного тренування, задавайте питання через лінк «Отправьте вопрос».
 - (b) Якщо попередній спосіб недоступний (наприклад, не можете залогінитися в еджадж) — можна писати на email oioick@ukr.net, а також у вже згаданий розділ «Олімпіади» форуму сайту cit.cskipo.edu.ua.

Які бувають вердикти (статуси, результати) перевірки ejudge-ем? Перелік можна бачити, зокрема, за адресою ejudge.ru/wiki/index.php/Вердикты_тестирования. Прокоментуємо більшість з них (але не всі).

OK — усе гаразд; є два різні вердикти з такою назвою: вердикт «OK» окремого тесту означає, що успішно пройдено цей окремий тест; вердикт «OK» усього розв'язку означає, що він успішно пройшов усі тести й повністю зарахований. Як правило, вердикт «OK» усього розв'язку означає також повний бал за задачу, але тут можуть бути й тонкощі конкретної системи оцінювання, наприклад, пов'язані зі штрафами за попередні невдалі спроби.

Compilation Error (Ошибка компиляции) — вердикт розв'язку в цілому (а не окремих тестів), який означає, що програму неможливо скомпілювати. Варто перевірити, чи правильно вибрано мову програмування і чи компілюється програма на локальному комп'ютері. Через відмінності у версіях компілятора на сервері та на локальному комп'ютері буває, що з цим все гаразд, а програма на ejudge все-таки не компілюється. В такому разі слід читати (через вищезгаданий лінк «Просмотр» стовпчика «Просмотреть протокол») повідомлення про помилку компіляції на сервері, яке *повинно* бути доступне в т. ч. й під час офіційних турів і в якому повинні бути вказані номер рядка, назва помилки та інша конкретна інформація. (Якщо раптом недоступне — змагання неправильно налаштоване, і слід звертатися з цим до адміністрації ejudge та/або журі.)

Wrong Answer (Неправильный ответ) — програма учасника завершилася і вивела якийсь результат, але він неправильний. Під час офіційних турів ніякої додаткової інформації не надається. Під час тренувань та в деяких навчальних наборах задач вміст вхідних даних, правильної відповіді та відповіді програми учасника можуть бути частково доступні через той самий стовпчик «Просмотреть протокол». Але не варто аж надто звикати до цього.

Presentation Error (Ошибка неправильного формата результата) — програма учасника завершилася і щось вивела, але воно зовсім не схоже на правильний результат. Ця помилка схожа на попередню, і ejudge може плутати їх (особливо, якщо в задачі є чекер та/або інтерактор), але, по ідеї, попередня помилка повинна видаватися, коли правильний формат, але неправильне конкретне значення, а ця — коли неправильний або незрозумілий формат. Наприклад, якщо відповідь повинна містити одне ціле число, і виведене одне ціле число, але не таке, як треба — це попередня помилка «Wrong Answer». А якщо відповідь повинна містити одне ціле число, а програма учасника вивела рядок "Masha!jila!kashu!!!" — це ця помилка «Presentation Error».

Run-Time Error (Ошибка при работе программы) — програма аварійно завершилася внаслідок виходу за межі масиву, чи ділення на нуль, чи ще якоїсь аналогічної помилки, чи генерації (і не обробки) виключної ситуації (exception) самою програмою. Взяти подальші деталі на офіційному турі, як правило, неможливо. Нерідко буває й так, що ejudge детектує лише цю нечітку помилку «Run-Time Error» там, де слід було б детектувати «Memory Limit Exceeded» чи «Security Violation».

Security Violation (Ошибка нарушения ограничений безопасности) — різновид помилки часу виконання, коли виконання програми заблоковано через те, що якась дія визнана потенційно небезпечною. Дуже сильно залежить від налаштувань конкретного змагання: може не з'являтися взагалі ніколи; може з'являтися на будь-яку спробу викликати `system` (включно з `system("pause")`); може з'являтися при спробі запустити додатковий процес/потік; може з'являтися при спробі відкрити будь-який файл, крім явно дозволених; тощо.

Time-Limit Exceeded (Ошибка превышения лимита времени) — програма не завершилася вчасно (перевищила відведений ліміт на час роботи). `Ejudge` досить добре (але не ідеально) обробляє ситуацію, коли програма учасника довго працює тому, що сервер зайнятий чимось іншим, і вміє розрізняти процесорний час (час справжнього використання процесора сервера саме програмою учасника) від астрономічного часу (що включає в себе як процесорний час, так і час очікування, поки сервер зайнятий чимось іншим). Як правило, це повідомлення з'являється тоді, коли програма учасника справді зайняла забагато процесорного часу. До речі, все це означає також, що хоча розпаралелювання між кількома процесорами/ядрами може бути корисним засобом пришвидшення на практиці, на «стандартних» олімпіадах (а не спеціальних олімпіадах з розпаралелювання) воно без толку і навіть шкідливе, бо, навіть якщо налаштування `ejudge` і дозволять програмі учасника запускати паралельні процеси/потоки (що не факт), з лімітом порівнюватиметься сумарний процесорний час.

Іноді (дуже рідко) бувають ситуації, коли цей вердикт ставиться безпідставно, бо вимірювання часу (особливо, в умовах, коли сервер справді зайнятий одночасно чимсь іншим) все-таки не ідеальне. Також бувають ситуації, коли `ejudge` ставить цей вердикт «*Time-Limit Exceeded*» замість наступного «*Wall Time Limit Exceeded*», причому це може відбуватися суто помилково, а може й тому, що поведінка «не розрізняти ці два вердикти» налаштована спеціально.

Wall Time Limit Exceeded (Ошибка превышения лимита реального времени) — перевищення ліміту астрономічного, а не процесорного часу. Особливо часто буває при перевірці інтерактивних задач, коли програма учасника намагається читати вхідні дані тоді, коли інтерактор (програма-суперниця) нічого не виводить. У свою чергу, це може бути спричинено тим, що програма учасника не помічає, що їй взагалі-то вже пора було завершитися, або грубим відхиленням від вказаного в умові задачі формату взаємодії (намагається прочитати те, чого їй ніхто і не обіцяв вводити), або недотриманням рекомендацій з умови задачі щодо `flush`. Це одна з проблем з інтерактивними задачами: якщо учасник ігнорує наполегливі рекомендації робити `flush`, або робить `flush` невчасно, то цілком імовірно, що виведення учасника «застряє по дорозі», не доходячи до програми-суперниці, й настає момент, коли програма учасника чекає, що їй щось введуть, а програма-суперниця не може це зробити, бо ще не прочитала попереднє виведення програми учасника, бо воно «застрягло десь по дорозі», бо учасник не робить у своїй програмі `flush`. І так ці дві програми можуть чекати одна на одну дуже довго, не споживаючи процесорний час, але й не пускаючи на перевірку інші розв'язки інших учасників. Не робити `flush` у розв'язку інтерактивної задачі — все одно, що при запуску звичайної програми зі звичайним введенням вхідних даних з клавіатури не натиснути `Enter`, а потім дивуватися «а чому оце я всі вхідні дані набрав, а воно їх не обробляє».

Ще цей вердикт може з'являтися, коли програма учасника прямо чи опосередковано викликає функцію `sleep` або аналогічну. Документація до `ejudge` стверджує (правда, це чомусь не підтверджується експериментально), що ще цей вердикт може з'являтися, коли (не інтерактивна) програма учасника намагається читати вхідні дані зі стандартного входу (клавіатури) там, де задача налаштована так, що їх треба читати лише з файлу.

Якщо цей вердикт «*Wall Time Limit Exceeded*» з'являється у ще якихось ситуаціях, крім описаних — це може свідчити про проблеми з `ejudge` (неправильний інтерактор, та/або неправильні ліміти процесорного та/або астрономічного часу, та/або неадекватно висока завантаженість сервера незрозуміло чим, та/або намагання `ejudge` розпаралелювати перевірку в більшій мірі, ніж це може надійно забезпечити «залізо» сервера); про таке варто повідомляти адміністрацію `ejudge` та/або журі. Але варто спочатку перевірити, чи точно нема якоїсь із згаданих ситуацій, коли цей вердикт справедливий.

Memory Limit Exceeded (Ошибка превышения лимита памяти) — програма учасника намагається використати більше пам'яті, ніж відведений їй ліміт. На жаль, детектування цієї помилки ненадійне, і замість неї учаснику нерідко повідомляється значно розмитіша помилка «Run-Time Error».

Skipped (Пропущено) — можливий лише при поблоковому оцінюванні, якщо на саме цьому тесті розв'язок можна взагалі не перевіряти, бо цей тест належить блоку, який вже точно не пройшов (бо розв'язок вже не пройшов один з раніших тестів цього блоку, або вже не пройшов хоча б один з блоків, успішне проходження яких обов'язкове для перевірки й оцінювання цього блоку).

Pending (Ожидает проверки), Compiling (Компилируется), Running (Выполняется) — тимчасові статуси розв'язку в цілому (а не окремих тестів), які відображають, що саме цей розв'язок ще не перевірений. Якщо все гаразд, то через якийсь невеликий час статус «Compiling» чи «Running» повинен автоматично змінитися на якийсь інший. Щодо статусу «Pending» це менш очевидно, бо він може означати те саме, але може також означати, що перевірка саме цієї задачі та/або саме цією мовою програмування саме зараз взагалі неможлива.

Check Failed (Внутренняя ошибка проверки) — вердикт, якого при нормальному перебігу перевірки учасник не повинен бачити, що б не робив; це надзвичайна ситуація, про яку варто негайно повідомляти адміністрацію ejudge та/або журі олімпіади. Виникає, коли розбаловка за окремі тести чи групи протирічить розбаловці за задачу в цілому, або взагалі не задано спосіб порівняння відповіді учасника з еталонною відповіддю (байт-у-байт, чи з деякою точністю, чи окремим чекером, тощо), або вміст хоча б одного файлу з тестом не відповідає встановленим описам його вмісту, або вказано назву чекера, а такого чекера нема, або чекер завершується аварійно, або аналогічна ситуація з інтерактором, тощо. Само собою, з усім цим повинні розбиратися адміністратор ejudge та автор задачі, а не учасники. Само собою, знання, що щось не так з перевіркою, не гарантує ні помилковості, ні правильності розв'язку учасника. Але розуміння смислу цього форс-мажорного вердикту все-таки може бути корисним учаснику — принаймні, можна осмислено відкласти цю задачу й зосередитися на інших, або зосередитися на аналізі власного розуміння цієї задачі, тимчасово ігноруючи вердикт ejudge-а, тощо. Якщо відомо, що іншим учасникам за цю задачу були успішно виставлені хоч якісь ненульові бали (це можна побачити, якщо під час туру доступна таблиця результатів усіх учасників, і в ній так написано, або якщо так відповість адміністрація ejudge на Ваше повідомлення про цю помилку) — можна бути більш-менш (але не абсолютно) впевненим, що проблема в тому, що чекер чи інтерактор коректно обробляє ті відповіді програми учасника, які розробник чекера чи інтерактора сприйняв за вірогідні, а Ваша програма виводить якийсь несподіваний варіант, не продуманий тим розробником. В принципі можна (не факт, чи треба, але можна) пробувати підібрати, які інші варіанти міг мати на увазі розробник чекера/інтерактора.

Які допустимі мови програмування? Згідно Правил Всеукраїнської олімпіади з інформатики — C++, C, Pascal, Java, Python. Сервер Черкаського ОІПОПП ejudge.skipo.edu.ua їх підтримує (Pascal — як Free Pascal та Pascal ABC.NET (mono), C++ — як g++, C — gcc, Java — версії 8, Python наявний і 2-й, і 3-й). Фактично на ejudge.skipo.edu.ua наявні також Perl, mono C#, Free Basic, і на більшості олімпіад, що відбуваються на цьому сервері, якщо учасник хоче — може здавати програми цими мовами, отримані бали враховуються. Але адміністрація серверу та журі туру не несуть відповідальності та не приймають претензій, якщо виявиться, що деякі задачі деякими з не рекомендованих мов не можуть бути розв'язані на повні бали, бо, наприклад, навіть найкращий алгоритм не на всіх тестах вкладається у обмеження за часом чи обсягом пам'яті. Більш того, за офіційними правилами Всеукраїнської олімпіади Java та Python теж дозволені з тією ж приміткою, що журі та адміністрація не несуть відповідальності (Для C++, C така відповідальність є; донедавна така відповідальність була також щодо Pascal, але з приблизно 2019 р. щодо нього теж з'явилася аналогічна примітка.)

Коли у тексті цього збірника згадується, що якусь частину задачі мовою Pascal треба писати самому, а мовою C++ можна використати готову бібліотечну функцію — часто (але не завжди) подібна функція є також і в Python та Java.

Чому нема Delphi? Зі «справжнім» Delphi є і проблема ліцензійної чистоти, і складнощі взаємодії ejudge, що працює під OS Linux, з Windows-програмами. Як компроміс, у переліку мов ejudge.ckipro.edu.ua є варіант «fpc-delphi — Free Pascal in Delphi mode». Він підключає деякі (не всі) «дельфійські» особливості (integer 32-бітовий; string дозволяє рядки, довші 255; функції містять спеціальну змінну Result; ...). До речі, цей режим можна увімкнути і зсередини своєї програми: написати в двох окремих рядках `{ $mode delphi }` та `{ $H+ }` (замість рядка `{ $APPTYPE CONSOLE }`) і здати під Free Pascal. (Конкретні рядки `"{ $mode delphi }"` та `"{ $H+ }"` варто запам'ятати, бо "fpc-delphi" не є невіддільною складовою будь-якої копії ejudge, а доданий на ejudge.ckipro.edu.ua вручну; відповідно, в інших системах автоматичної перевірки його за простою може не бути.)

Які ще є рекомендації щодо мов програмування? *PascalABC*: Як правило, варто здавати під fpc-delphi. Але іноді варто під pasabc-linux. Справа в тім, що pasabc-linux відрізняється від PascalABC під Windows, і при цьому працює значно повільніше, ніж fpc-delphi чи fpc. Тому, pasabc-linux далеко не завжди є найкращим вибором. Але іноді все ж є, бо pasabc-linux все-таки більш схожий на PascalABC під Windows, ніж fpc-delphi.

TurboPascal/BorlandPascal: Як правило — здавати під fpc-delphi. Але іноді виявляється доцільним написати на початку програми окремий рядок `{ $mode tp }` (який робить fpc більш схожим на Turbo Pascal) і здати під fpc.

Під усіма паскалями, украй небажано підключати uses crt. Результати взаємодії crt з ejudge украй дивні — програми працюють іноді правильно, іноді неправильно, іноді правильно, але довго, іноді по-своєму розриваючи рядки результату... Причому, ejudge не каже, що з програмою щось не так (краще б казав). Все це проявляється сильніше, коли програма читає з клавіатури і/або виводить на екран, і менше, коли введення/виведення суто файлове.

Visual C++: Здавати під g++, розуміючи, що g++ — C++, але інший. Зокрема: (1) у заголовку main не можна писати ні `_tmain` ні `_TCHAR`. Можна, наприклад, `int main(int arc, char* argv[])` або `int main()`. (2) Не можна підключати `#include "stdafx.h"`.

Під C/C++, не слід викликати функцію system (навіть `system("pause")`); втім, адміністрація ejudge може налаштувати це в різних змаганнях по-різному.

C#: Здавати під mcs (monoC#), розуміючи, що це не справжній C# (якого під Linux нема), а порт деякої дуже старої версії C#. Як наслідок, у mcs нема багатьох бібліотек, навіть деяких з тих, які сучасні версії Visual Studio зазвичай підключають автоматично. Зокрема, через це слід видаляти рядок «`using System.Threading.Tasks;`», навіть якщо його створено автоматично.

Java: Під java, не можна задавати package (а якщо середовище створює його автоматично, то видаляти перед відправкою в ejudge). Якщо є потреба, щоб програма містила кілька класів — слід забезпечити, щоб у єдиному *.java-файлі першим по порядку йшов заголовок того класу, котрий публічний і містить точку входу (метод main). Інші класи можна або робити вкладеними у той клас, або робити не публічними і розміщувати у тому ж файлі *нижче*. Для еджаджа це важливо, навіть якщо неважливо для компіляції у якихось інших ситуаціях.

Щодо особливостей та переваг сайту ideone.com. Наведені у тексті збірника посилання на сайт ideone.com дозволяють бачити тексти програм з підсвіткою синтаксиса, скачувати їх, а також створювати власну копію (ця дія називається «fork») і працювати з нею безпосередньо на сайті (це особливо зручно, якщо потрібно швидко спробувати якусь дрібну зміну з телефона або з чужого комп'ютера, де нема середовища програмування).

Сайт ideone.com передбачає, що вхідні дані програми слід ввести у текстове поле (input у смислі html) на тій же сторінці сайту під текстом цієї програми, до її запуску. Сайт ideone.com передбачає, що програма має читати вхідні дані (ті, що вводяться до запуску) через стандартний вхід, він же stdin («клавіатура», яка фактично перенаправляється, але з точки зору програми користувача сайту є клавіатурою) й виводити результати через стандартний вихід, він же stdout, він же, в аналогічному смислі, «екран».

Оскільки значна частина розглянутих у збірнику задач передбачає саме такі введення і виведення, відповідні тексти програм можуть бути використані і на ideone.com, і на ejudge.ckipro.

edu.ua без яких би не було змін; разом з тим, деяка частина старіших задач ejudge.skiro.edu.ua налаштована лише на файлове введення/виведення, через що потрібно або переписувати конкретно ті місця коду, які стосуються введення/виведення, або вживати неочевидну конструкцію з `ifdef` та `assign/freopen/...` (залежно від мови програмування), як у `ideone.com/XbSTUE`, де фрагмент між `{ifdef ONLINE_JUDGE}` та `{else}` виконується на `ideone.com` і не виконується ні на `ejudge.skiro.edu.ua`, ні на локальному комп'ютері, а фрагмент між `{else}` та `{endif}` — навпаки; `assign` забезпечує перенаправлення стандартних входу («клавіатури») і виходу («екрану») вже не засобами сайту, а засобами самої програми. Взагалі кажучи, такі засоби бувають дуже корисними, й рекомендуємо з ними ознайомитися детальніше, але за іншими джерелами, бо обсяг збірника не дозволяє розглянути це детально для всіх ймовірних мов програмування.

Чи потрібно доводити правильність алгоритмів? З одного боку, доведення не вимагаються та не оцінюються. З іншого — не вміючи доводити, важко оцінювати правильність ідей. Якщо програма видає неправильну відповідь, треба якось приймати рішення, чи шукати й виправляти технічну помилку, чи повністю відкинути цю ідею й шукати іншу. У такому сенсі доведення бувають корисні, навіть якщо робити їх виключно для себе. Приклади доведень можна бачити на стор. 28, 40, 42.

Що означають записи, подібні до « $O(n^2)$ »? Деталі можна знайти в Інтернеті чи літературі за назвою *асимптотичні позначення* O , o , Ω , ω , Θ ; « O » та « o » нерідко читають «о» (одним звуком), але правильніше «омікрон»; « Ω » та « ω » — «омега» (велика й маленька); « Θ » — «тета». Формальні означення складнуваті, наприклад один з варіантів означення O такий: числова функція $f(n)$ є функцією порядку $O(g(n))$, коли існує деяка числова функція $f_1(n)$, така, що виконуються обидва твердження «для всіх n , $f(n) \leq f_1(n)$ » та « $\lim_{n \rightarrow \infty} \frac{f_1(n)}{g(n)} = c > 0$, де c — скінченне число» (яке не зростає при $n \rightarrow \infty$).

Простіше (не зовсім точно) пояснення: нехай нас цікавить, наскільки стрімко зростає функція $2n^2 + 17n + 12\sqrt{n} + 500$ при $n \rightarrow \infty$. При дуже великих n , доданок $2n^2$ значно перевищує всі інші разом узяті, тож можна приблизно оцінювати всю суму самим лише $2n^2$. Підемо ще далі й скажемо, що нас не цікавить, чи $2n^2$, чи $7n^2$, чи $\frac{1}{6}n^2$ — головне, що коефіцієнт при n^2 є скінченним строго додатним числом. Оце й показує, що як $2n^2 + 17n + 12\sqrt{n} + 500$, так і $7n^2$, так і $\frac{1}{6}n^2 + 100n\sqrt{n} + 12345$ являють собою $O(n^2)$ та $\Theta(n^2)$. Відмінність між O і Θ у тому, що $O(g(n))$ дозволяє, щоб досліджувана функція була хоч «дуже приблизно рівною» $g(n)$ (у щойно описаному сенсі), хоч значно меншою, а $\Theta(g(n))$ — лише «дуже приблизно рівна». Наприклад, $2n\sqrt{n} + 17n$ можна вважати хоч $O(n\sqrt{n})$, хоч $O(n^2)$, хоч $\Theta(n\sqrt{n})$, але не $\Theta(n^2)$.

Навіщо все це програмісту? Бо саме так зручно оцінювати залежність часу роботи програми від розміру вхідних даних. Кількість тактів CPU, потрібних для обчислення деякого виразу, залежить від моделі CPU та від того, чи поміщаються дані у кеші, й усе це аналізувати дуже складно. Але можна ігнорувати ці складнощі, заявляючи, що будь-який фрагмент програми, який не містить циклів чи

```
for i:=2 to n do begin
  curr := A[i];
  j := i-1;
  while (j>0) and (A[j]>curr) do begin
    A[j+1] := A[j]; }  $\Theta(1)$ 
    dec(j);
  end;
  A[j+1] := curr;
end;
```

} $O(i)$, а також $O(n)$

Сумарно буде $\Theta(n)$ разів по $O(n)$, тобто $O(n^2)$. До речі, це $O(n^2)$ неможливо перетворити у Θ від чого б не було: при деяких вхідних даних (наприклад, уже відсортованих) маємо $\Theta(n)$ разів по $\Theta(1)$, тобто $\Theta(n)$; при деяких інших — $\Theta(n)$ разів по $\Theta(i)$, які, враховуючи $1 + 2 + \dots + n = \frac{n(n+1)}{2}$, перетворюються у $\Theta(n^2)$.

Рис. 1. Приклад проведення асимптотичного аналізу фрагменту програми

викликів підпрограм, працює за $\Theta(1)$. Приклад такого аналізу (для алгоритму сортування вставками) наведено на рис. 1.

Як потім використати отримані асимптотичні оцінки? Порівнювати ефективність алгоритмів (для чого не завжди треба реалізовувати їх, часто досить уявити структуру циклів). А також оцінювати шанси алгоритму поміститись у такі-то обмеження часу при обробці вхідних даних таких-то розмірів.

Тактова частота сучасних комп'ютерів — кілька гігагерц (мільярдів тактів за секунду). Враховуючи неточності у питанні «скільки тактів займають які дії» та ігнорування констант у самих означеннях O чи Θ , ці «кілька мільярдів» треба ще поділити (причому не ясно, на скільки), й у висновку виходить щось дуже приблизне «за секунду встигається десь 10^7 – 10^9 дій». Але навіть настільки приблизні оцінки можуть бути корисні. Наприклад: $n=10^8$, складність алгоритму $\Theta(n^2)$. Тоді $(10^8)^2 = 10^{16}$, час роботи від $10^{16}/10^9 = 10^7$ сек (≈ 4 місяці) до $10^{16}/10^7 = 10^9$ сек (≈ 30 років) — безнадійно далеко від того, щоб укластись у пару секунд. Або: при $n=10^5$ алгоритм складністю $O(n\sqrt{n})$ ($10^5 \cdot \sqrt{10^5} \approx 3,2 \cdot 10^7$) має шанси вкластись у секунду, але без «запасу», можуть бути важливі всілякі оптимізації у дрібницях.

Звісно, можна вимірювати час і по-простому в мілісекундах. Автоматичні системи перевірки (включаючи `ejudge`) так і вимірюють. Ці способи не заважають один одному, а доповнюють.

Нарешті, у збірнику масово використовуються записи $O(n \log n)$, $\Theta(\log n)$, тощо, в яких не вказана основа. Це не помилка, а традиція. По-перше, завдяки тотожності $\log_a n = \log_a b \cdot \log_b n$, вирази $O(\log_a n)$ та $O(\log_b n)$ задають один і той самий клас функцій при довільних константних основах a та b , строго більших 1. По-друге, у комп'ютерних науках «стандартною» основою логарифма вважається 2, і $\log n$ можна трактувати як $\log_2 n$.

Що таке діапазон (цілочисельного) типу? Переповнення?

Уявіть лічильник електроенергії (чи води, чи кілометрів — байдуже).

0	2	3	9	↑	9	9	9	9	↑
		4	0		0	0	0	0	

Кожен десятковий розряд відображається окремим барабаном, при завершенні значень одного відбувається збільшення наступного на 1... і у деякий момент не вистачає розрядів, і після великого числа одержується 0. Це і є переповнення (*overflow*).

У комп'ютері не барабани (і не буває зображених на рисунку проміжних станів), але теж скінченна кількість розрядів для подання чисел. Тільки розряди двійкові, а не десяткові, тому переповнення настає не після 9999 чи 999999, а після чисел виду $2^k - 1$ (де k — кількість бітів), тобто, у двійковому поданні, $\underbrace{11 \dots 1}_{k \text{ штук}}$. Звідси, діапазон беззнакових типів — від 0 до $2^k - 1$ (тут і далі оби-

дві межі включно). Знакові типи мають діапазон від -2^{k-1} до $2^{k-1} - 1$ (кому цікаво, звідки така несиметричність, див. uk.wikipedia.org/wiki/Доповняльний_код).

Наприклад, при спробі обчислити у 16 бітах $1000_{Dec} \times 100_{Dec} = 100000_{Dec} = 11000011010100000_{Bin}$ («*Dec*» означає десяткову систему, «*Bin*» — двійкову) фактично буде отримано лише 16 останніх бітів 1000011010100000_{Bin} , котрі у беззнаковому типі задають число 34464_{Dec} , а у знаковому — (-31072_{Dec}) .

Окремо відзначимо вирази, подібні до $c := a * b$, де a та b мають вужчий тип, c — ширший, і результат поміщається у ширший тип, але не у вужчий. Чи результат буде обчислений правильно у ширшому типі, чи з переповненням у вужчому — залежить від багатьох обставин. Можна вивчати ті обставини, й такі знання бувають корисними. Можна в усіх сумнівних ситуаціях робити *приведення типів* (*typecasting*), наприклад $c := \text{int64}(a) * \text{int64}(b)$.

Можна знайти таблички з точними діапазонами (напр., www.freepascal.org/docs-html/ref/refsu5.html). Але до них слід ставитися обережно, бо ці діапазони можуть залежати від архітектури «заліза», операційної системи та компілятора. Скажімо, у en.cppreference.com/w/cpp/language/types часто говориться «at least», тобто може бути й більше.

Найсумніше, коли типи з однаковими назвами мають різну розрядність на локальному комп'ютері, де пише учасник, і на сервері, де відбувається перевірка. Зокрема (але не тільки), така проблема виникає, коли учасник локально пише під Delphi чи PascalABC, а здає під Free Pascal, який

наразі при відсутності додаткових вказівок вважає `integer` 16-бітовим (директива `{ $mode delphi }` робить `integer` 32-бітовим; див. також стор. 11).

Насамкінець, при оновленні «заліза» та програмного забезпечення сервера `ejudge.skiro.edu.ua` 2019 р. не повністю дотримано сумісність із попередньою версією цього ж серверу, й одна з таких відмінностей полягає у зміні архітектури з 32-бітової на 64-бітову. Як наслідок, частина мов програмування стали використовувати ширшу розрядну сітку, а через це з'явилися і ситуації, коли ті самі розв'язки, які раніше проходили менше тестів, почали проходити більше (бо зменшилися або зникли переповнення типів), і (рідше) ситуації, коли розв'язки, які раніше проходили більше тестів, почали проходити менше (бо розв'язок почав перевищувати ліміт пам'яті).

Формат з рухомою комою (плаваючою крапкою, floating point) та похибка. Існує експоненційний формат виведення дробових чисел: наприклад, $1.234e2$ означає $1,234 \cdot 10^2 = 123,4$; частина після «e» називається *порядок*; до «e» — *мантиса*. У пам'яті зберігається теж експоненційний формат, але двійковий. Розрядні сітки як у порядку, так і в мантиї обмежені. Англійською мовою цей формат називається «*floating point*»; українською, на жаль, поширені зразу кілька варіантів. У цьому збірнику вживатиметься або англomовне «*floating point*», або «формат з рухомою комою»; при цьому варіанти, де замість «рухомою» вжито «плаваючою» та/або замість «комою» вжито «крапкою», теж поширені, й все це одне й те саме.

Обмеженість порядку критична для дуже великих чи дуже близьких до 0 чисел. Наприклад, тип `double`, реалізований згідно з IEEE 754, не може містити значення, модуль яких більший 10^{307} або менший 10^{-324} (нуль можливий; неможливі значення між 0 і 10^{-324}). Обмеженість же мантиї призводить до *втрати точності* або *похибки* — наприклад, неможливості розрізнити у типі `double` $1,00000000000000001_{Dec}$ від 1 (або $1,00000000000000001 \cdot 10^{-9}$ від 10^{-9} , або $1,00000000000000001 \cdot 10^{98}$ від 10^{98} — важливий не порядок, а те, що цифри, якими чісла відрізняються, не поміщаються у мантию).

В інших типах конкретні діапазони можуть бути іншими; але суть двох обмежень однакова в усіх типах з рухомою комою.

Проблема похибок загострюється тим, що у тексті програми та вхідних даних чісла пишуть у десятковій системі, а фактичне внутрішнє подання двійкове. Наприклад, програма `ideone.com/rnGPE1` показує, що 10-кратне додавання 0.3 до (-3) не дає рівно 0. Адже $0,3_{Dec}$ стає нескінченним періодичним двійковим дробом $0,0(1001)_{Bin}$, обмеженість мантиї призводить до його заокруглення, і похибка з'являється у самій константі.

Дії над числами теж можуть призводити до утворення чи зростання похибки. Сильно збільшують похибку тригонометричні та інші складні функції (але `sqrt` якщо й збільшує, то не сильно; щодо цього виклик `sqrt(x)` набагато кращий за `power(x, 0.5)`). А ще можуть сильно збільшити похибку віднімання $a-b$ при $a \approx b$ та додавання $a+b$ при $a \approx -b$. Наприклад, для вагів, якими можна зважити людину, похибка ± 10 г дрібна; але якщо спробувати взнати масу аркуша паперу шляхом того, що хтось зважиться, тримаючи цей аркуш у руках, потім ще раз без цього аркуша, й обчислить різницю вимірювань — результат безнадійно втоне у похибці. Приклад програми, де демонструється аналогічна втрата результату — `ideone.com/bKXT0z`. Тому буває важливо провести аналітичні перетворення, щоб отримати математично еквівалентну формулу з меншим впливом похибок.

Інший поширений прийом, який намагається нівелювати похибки — не порівнювати чісла з рухомою комою «по-простому», а лише з «допуском на похибку», як у таблиці праворуч.

$a = b$	$\text{abs}(a-b) < \text{EPS}$
$a < b$	$a + \text{EPS} < b$
$a \leq b$	$a \leq b + \text{EPS}$

«EPS» означає «якесь маленьке додатне число», його треба задати самому (наприклад, `const EPS=1e-6`). Як правильно вибирати EPS — питання складне. Буває легше йти не від того, якої величини можуть сягати похибки, а від того, наскільки близькими можуть бути значення, які треба розрізнити, й брати EPS у кілька разів меншим... Іноді варто переписувати умови так, щоб враховувати не абсолютну, а відносну похибку...

Детальніше про `floating point` та похибки можна прочитати у багатьох місцях, зокрема `habr.com/ru/post/112953` та/або `gamedev.ru/code/articles/FloatingPoint` (авторства Д. І. Поліщука).

Щодо швидкості введення/виведення. Є такий сумний факт, що деякі зручні стандартні засоби введення/виведення працюють повільно. Наприклад, мовою C++ `istream/ostream` (зокрема, `cin/cout`) *сильно* програють у швидкодії `scanf/printf`. Наприклад, на II етапі 2013/14 н. р. у найбільшому тесті задачі «Всюдисущі чісла» (стор. 33–35) саме лише читання `cin`-ом вхідних даних, взагалі без обробки по суті, вже перевищувало обмеження часу.

Радикальний вихід — читати та виводити функціями `scanf` та `printf` відповідно, які з незвички незручні (особливо, якщо змінні, що читаються/виводяться, мають якийсь хитрий тип), зате швидко працюють.

Інший спосіб зменшення проблеми (лише часткового зменшення, його не завжди достатньо) — сукупність викликів `cin.sync_with_stdio(false); cout.sync_with_stdio(false); cin.tie(0); cout.tie(0);`. Перша згадана пара викликів пришвидшує читання `cin`-ом та виведення `cout`-ом за рахунок вимикання синхронізації з `scanf/printf`, тобто після цього не можна читати поперемінно то засобами `cin`, то засобами `scanf` чи аналогічно виводити поперемінно то засобами `cout`, то засобами `printf`. Друга згадана пара викликів вимикає синхронізацію між `cin` та `cout`, тобто гарантію, що перед кожним введенням `cin`-ом точно відбудуться всі раніше зроблені виведення `cout`-ом.

Засоби з минулого абзацу орієнтовані суто на випадок великого об'єму введення/виведення «класичної» ситуації «прочитав вхідні дані, обробив, вивів результат»; у випадку ж інтерактивних програм вони, навпаки, шкідливі (порівняйте кінець попереднього абзацу з поясненнями щодо вердикту «Wall Time Limit Exceeded» зі стор. 9).

Ще гострішою проблема швидкості читання є у мові Java, причому там дати короткі корисні поради значно важче. Коротко можна сказати лише, що зручний клас `Scanner` має принципово нездоланні проблеми зі швидкістю, які не вимикаються якимись простими викликами, подібними до позаминулого абзацу. Часто (але не завжди) більш-менш непогане поєднання зручності та швидкодії дає `new StreamTokenizer(new BufferedReader(new FileReader(new File("input.txt"))))`. Детальніше шукайте самостійно.

Щодо статичної, динамічної та стекової пам'яті, а також переповнення стеку. Взагалі це питання варто вивчити за іншими джерелами, з урахуванням особливостей конкретної мови програмування. Але короткий опис (орієнтований у першу чергу на C++) наведемо.

1. Статична пам'ять (static memory, data segment) — глобальні змінні, зарані заданих розмірів, які існують протягом всього часу виконання програми.
2. Стекова пам'ять (program stack, програмний стек) — локальні змінні підпрограм чи інших блоків видимості, з'являються при входженні всередину підпрограми чи блоку і зникають при виході.
3. Динамічна пам'ять (heap, купа) — місце для зберігання даних може з'являтися (ставати доступним) та зникати (ставати недоступним) під час виконання програми, за рахунок отримання пам'яті від менеджера пам'яті операційної системи та навпаки. При цьому обсяги окремих ділянок пам'яті можуть визначатися «на льоту» (в т. ч. обчислюватися через вхідні дані); пам'ять стає доступною або за допомогою явних викликів `new/alloc/...` програмістом, або прихованими аналогічними викликами зсередини сучасних структур даних, як-то динамічні масиви, множини, тощо.

Наприклад, якщо мовою C++ виникає потреба в цілочисельному двовимірному масиві 4321×12345 , його можна оголосити будь-яким з цих способів:

1. статично: `int data[4321][12345]`, якщо це оголошення глобальне;
2. на стеку: `int data[4321][12345]`, якщо це оголошення всередині функції (причому, навіть якщо є одна лише функція `main`, то на те, яка пам'ять використовується, впливає факт оголошення всередині чи ззовні `main`);
3. динамічно: `vector<vector<int>> data(4321, vector<int>(12345))`, або `int** data = new int*[4321]; for(int i=0; i<4321; i++) data[i]=new int[12345];`.

Динамічна пам'ять у багатьох смислах зручна (використовуємо скільки треба й коли треба); як правило, великі дані (зокрема, масиви) цілком доцільно розміщувати в основному в ній (потребуючи також невеликих витрат статичної та/або стекової пам'яті, зокрема, на вказівники).

STL-контейнери (`vector`, `set`, `map`, `string`, тощо), роблять це автоматично: коли такий контейнер оголошений статично чи локально, у статичній чи стековій пам'яті розміщується лише допоміжна інформація, а самі дані (елементи) потрапляють у динамічну пам'ять.

Якщо є потреба мати най-най-найефективнішу технічну реалізацію, рахуючи кожен відсоток швидкодії та/або обсягу пам'яті, в принципі може мати сенс відмова від динамічної пам'яті. (По-перше, динамічна пам'ять працює повільніше за інші, особливо, на етапах звернень до менеджера пам'яті, тобто виділення (отримання, `new/alloc/...`) та звільнення (повернення, `delete/free/...`). По-друге, для динамічної пам'яті характерні додаткові витрати — зокрема, на вказівники; крім того, конкретно для `vector`-ів пам'ять виділяється «із запасом».) Але така потреба виникає досить рідко. Для більшості олімпіадних задач, вважається правильним вимагати досить добру асимптотику (див. стор. 12–13), але не чіплятися до таких дрібниць, як вплив різновиду пам'яті на швидкодію.

Для динамічної пам'яті зазвичай справедливе твердження «краще кілька великих масивів, ніж багато маленьких» (наприклад, масив 123456×12 , поданий як `vector<vector<int>> data(123456, vector<int>(12))`, потребує значно більше накладних витрат, ніж транспонований масив 12×123456 , поданий як `vector<vector<int>> data(12, vector<int>(123456))`); втім, деталі залежать від «заліза», OS, версії та налаштувань компілятора, а також порядку, в якому слід переглядати елементи цього масива.

Статична пам'ять у деяких смислах найшвидша і разом з тим найпростіша, але, як вже відзначено, абсолютно не гнучка.

Стекова пам'ять добре узгоджена з областю видимості локальних змінних — вони з'являються тоді й там, де потрібні, зникають при виході з відповідної підпрограми чи іншого блоку видимості, й це відбувається швидко і зручно. Саме стековою пам'яттю (або грамотним поєднанням стекової та динамічної) легко добитися, щоб різні рівні рекурсивної вкладеності мали справу кожен зі своїм контекстом локальних змінних. Але, як правило, сумарний обсяг стекової пам'яті фіксований і обмежений жорсткіше, ніж інші види пам'яті. (Коли в умові сказано, наприклад, «Обмеження пам'яті 256 Мб», мається на увазі сума обсягів всіх видів пам'яті; при цьому стек за простою може бути обмеженим, наприклад, розміром 2 Мб.) Через це, *переповнення стеку* (*stack overflow*), тобто вичерпання всієї доступної стекової пам'яті, є важливим частковим випадком всіх проблем роботи з пам'яттю та всіх помилок часу виконання (*run-time errors*). Причини переповнення стеку, як правило, можна віднести до одного з трьох сценаріїв.

(А) Деяка функція містить дуже великий локальний масив. Проблема переповнення стеку може зникнути, якщо зробити цей масив глобальним чи динамічним (зокрема, `vector`-ом). Якщо зробити масив глобальним, він стає загальнодоступним усій програмі, що не завжди добре. Деякими мовами, включаючи C++, є «компромісна» можливість оголосити його *статичним локальним*, просто написавши перед локальним оголошенням `static`, як-то `static int data[12345][4321]`. Такий масив (чи інша змінна, байдуже; просто, для масивів це особливо цінно) стає одночасно локальним з точки зору області видимості, але статичним з точки зору виділення пам'яті. Звісно, це означає, що різні рівні рекурсії матимуть справу з одним примірником. Крім того, якщо різними функціями треба різні великі масиви, то, незалежно від того, чи можуть вони викликати одна одну, чи ні, потреба у статичній пам'яті визначатиметься *сумою* розмірів цих масивів, а можливість використовувати одну й ту саму пам'ять у різні моменти часу для різних масивів втрачається.

(Б) Програма містить рекурсію, і ця рекурсія виходить з-під контролю, стаючи нескінченною. Це явна помилка в алгоритмі, треба уточнювати умови заглиблення в рекурсію та умови виходу з неї, а спроби збільшувати обсяг стеку призводили б лише до марнотратства, не вирішуючи суті проблеми.

(В) Програма містить рекурсію, з цілком розумними умовами заглиблення/виходу та цілком розумним набором аргументів та локальних змінних (які не можна робити статичними, бо треба, щоб вони були в кожного рівня рекурсії свої), але сумарний обсяг локальних змінних всіх рівнів рекурсії занадто великий. Це найсумніша ситуація, бо, з точки розу здорового глузду і загальних практик програмування тут цілком може бути найправильнішим збільшення розміру стеку... але в умовах олімпіади це зазвичай неможливо, бо цей розмір визначається налаштуваннями OS Linux та `ejudge`, а учасник з середини своєї програми вплинути на це не може. Тут може виникати потреба в таких

некрасивих діях, як: (а) ретельний аналіз, які з локальних змінних все-таки можна зробити статичними чи динамічними, в тому числі за рахунок перенесення інформації з програмного стеку як області автоматично підтримуваної пам'яті у стек як структуру даних, що зберігається у глобальному чи статичному локальному масиві; (б) відмова від рекурсії, як-то заміна пошуку вглиб на пошук ушир.

З іншого боку, ejudge цілком може бути налаштований, навпаки, так, що обсяг відведеної під програмний стек пам'яті досить великий, тож толку переносити дані зі стекової пам'яті у статичну чи динамічну може й не бути. Прикро, що щодо цього і нема єдиних загальноприйнятих правил чи хоча б традицій, і учасник не може ні вплинути на ці налаштування, ні навіть гарантовано точно взнати всі деталі цих налаштувань. Так що лишається тільки або намагатися писати програми, не надто сподіваючись на якісь конкретні налаштування (в т. ч. уникаючи глибоких рекурсій), або намагатися добитися відповіді щодо цих налаштувань, задаючи питання журі засобами ejudge, або витратити деяку частину дозволених спроб здачі на те, щоб здати глибокі рекурсії й узнати, при якому обсязі стеку починаються його переповнення.

Зазначимо також, що і в Microsoft Visual C++, і у FreePascal для Windows є можливість задати розмір стеку зсередини програми учасника, але ні одне, ні інше не працює під Linux-аналогами.

3 Задачі та їхні розв'язки

3.1 II (районний/міський) етап 2012/13 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання №3).

Задача А. «Новий рік»

Вхідні дані: newyear.in Обмеження часу: 1 сек
Результати: newyear.out Обмеження пам'яті: 64 Мб

В країні Олімпія наближається святкування Нового Року. Для підготовки святкування необхідно в кожную школу доставити новорічну ялинку. Якби кількість шкіл в країні була відомою, то доставка ялинок не була б складним завданням. Але в країні на протязі декількох останніх років проводилася освітня реформа і нажалі ніхто не може визначити точну кількість шкіл. Відомо лише кількість шкіл до початку проведення реформи, та статистичні відомості для кожного року, на скільки змінилась кількість шкіл відносно попереднього року.

Завдання. Напишіть програму newyear, яка за статистичними відомостями визначить кількість шкіл в країні Олімпія.

Вхідні дані. Перший рядок файлу newyear.in містить два натуральних числа: N — кількість шкіл до проведення реформи, та K — кількість років проведення реформ ($1 \leq N \leq 1000000$, $1 \leq K \leq 100$). Далі ідуть K рядків, кожен з яких містить одне ціле число (модуль якого не більше за 1000000) — на скільки змінилась кількість шкіл у порівнянні з попереднім роком.

Результати. Ваша програма повинна створити текстовий файл newyear.out і вивести туди єдине число — кількість шкіл у країні Олімпія.

Приклад:	newyear.in	newyear.out
	100 3	109
	1	
	-2	
	10	

Розбір задачі. Задача на реалізацію, тобто не треба нічого придумувати, лише прочитати й реалізувати. Приблизно як стандартний підрахунок суми, тільки врахувати ще початкову кількість шкiл. Наприклад, див. ideone.com/XbSTUE. Про смисл `ifdef` та `assign` у цьому коді див. стор. 12.

Цей код реалізований так, щоб рахувати суму «на ходу», не зберігаючи всі вхідні дані у масиві. Але, при обмеженні $K \leq 100$ та автоматичній перевірці, учасник має повне право вибрати простіший для себе варіант (з масивом чи без). На олімпіадах часто вимагають ефективний код, але це мають бути ситуації, де ефективний код легко розрізняється від неефективного. Ця ситуація такою не є, бо нереально помітити зайву сотню 4-байтових чисел на фоні виконуваного файлу, який займає значно більше місця у пам'яті. Якби хотіли розрізнити засобами автоматичної перевірки, чи зумів учасник реалізувати таку програму без масива, давали б обмеження не $K \leq 100$, а десь так $K \leq 10^6$ (чи навіть $K \leq 10^7$), і при цьому жорстке обмеження пам'яті.

А от забезпечити, щоб усі чiсла поміщалися (без переповнень) у тип, треба. Для більшості сучасних мов програмування «стандартне» ціле число вміщає в себе максимальне для цієї задачі $101 \cdot 10^6$, для деяких мов тут може бути проблема (зокрема, для Паскаля; див. також стор. 13 та 11).

Задача В. «Матриця»

Вхідні дані: `matrix.in` Обмеження часу: 1 сек
Результати: `matrix.out` Обмеження пам'яті: 64 Мб

Вам дана таблиця розміром N рядків на M стовпців. У кожній клітинці таблиці записано число 0 або 1. За один хід можна вибрати один із рядків таблиці і циклічно зсунути значення в ній на одну клітинку або вліво, або вправо.

Циклічно зсунути рядок таблиці на одну клітинку вправо означає перемістити значення кожної комірки цього рядка, крім останньої, в сусідню комірку праворуч, а значення останньої комірки перемістити в першу комірку. Аналогічним чином, але у зворотний бік виконується циклічний зсув рядка таблиці вліво. Наприклад, якщо циклічно зсунути рядок «00101» на одну клітинку вправо — вийде рядок «10010», якщо ж зрушити рядок «00101» на одну комірку вліво — вийде рядок «01010».

Завдання. Напишіть програму `matrix`, яка читає таблицю чисел (матрицю) та визначає найменшу кількість ходів при яких в матриці утвориться стовпчик, що містить лише одиниці.

Вхідні дані. Перший рядок файлу `matrix.in` містить два цілих числа, розділених пробілом: N ($1 \leq N \leq 100$) — кількість рядків в таблиці і M ($1 \leq M \leq 100$) — кількість стовпців в таблиці. Далі слідує N рядків, кожна з яких містить по M символів «0» або «1»: j -ий символ i -ого рядка описує вміст комірки в i -тому рядку і j -ому стовпці таблиці. Гарантується, що в описі таблиці не зустрічається ніяких символів крім «0» і «1».

Результати. Ваша програма повинна створити текстовий файл `matrix.out` і вивести туди єдине число: найменшу кількість ходів, за які можна в якому-небудь із стовпців таблиці отримати лише одиниці. Якщо цього зробити неможливо, виведіть число -1 .

Приклади:

<code>matrix.in</code>	<code>matrix.out</code>
3 6 101010 000100 100000	3

<code>matrix.in</code>	<code>matrix.out</code>
2 3 111 000	-1

Розбір задачі. Само собою, треба врахувати, що в якомусь рядку (чи кількох рядках) одиничок може взагалі не бути; очевидно, саме тоді слід видавати відповідь -1 (коли одиничок нема, як не крути рядок, одиничка не з'явиться; коли є, будь-яку з одиничок можна «докрутити» до будь-якого стовпчика, питання лише, як далеко крутити). Отже, властивість «у кожному рядку є хоча б одна одиничка» варто перевірити зразу: це і розрізнить, чи виводити -1 , і дасть можливість написати такий `if`, щоб справді шукати мінімальну кількість ходів лише у ситуаціях, коли вказана властивість виконується. Тому, в подальших міркуваннях вважаємо, що це вже перевірено й підтверджено.

Якщо знати, в якому стовпчику слід зібрати одинички, легко бачити, що для кожного рядка слід розглянути лише найближчу зліва та найближчу справа одинички й вибрати з них ближчу, а решта одиничок, якщо вони й існують, не вигідні для шуканого загального мінімуму. При цьому може бути, що «найближча ліворуч» і «найближча праворуч» — одна й та сама одиничка; тим паче,

найближча ліворуч/праворуч одиничка може бути «за розривом» (як-то, для стовпчика 2 найближча ліворуч одиничка у стовпчику $M - 1$). Як це врахувати? Є щонайменше два варіанти: (А) працювати з діапазоном індексів масиву від 0 до $M - 1$ (навіть у Паскалі, де прийнято нумерувати з 1), і зсув на k елементів праворуч виражати як $(s+k) \bmod M$, на k ліворуч — як $(s+M-k) \bmod M$; (Б) розглянути втричі ширшу матрицю, де (при тій самій кількості рядків) тричі повторено вміст заданої, й шукати найближчі одинички лише для стовпчиків від $M + 1$ до $2 \cdot M$; тоді «зарозривні ліві» одинички потрапляють у проміжок від 1 до M , «зарозривні праві» — від $2 \cdot M + 1$ до $3 \cdot M$.

Але ж ми не знаємо, в якому стовпчику вигідно збирати одинички! Що ж, не знаємо. Але ніщо не заважає перебрати всі можливі варіанти:

1. зовнішній цикл перебирає, в якому стовпчику намагаємося зібрати одинички;
2. вкладений у нього цикл перебирає всі рядки;
3. вкладені в нього два (послідовні один відносно одного) цикли шукають, як далеко праворуч та як далеко ліворуч одиничка (у поточному, визначеному циклом № 2, рядку, від поточного, визначеного циклом № 1, стовпчика).

Само собою, з результатів двох циклів № 3а та № 3б вибирається мінімум (звідки, справа чи зліва, ближче привести одиничку), цикл № 2 рахує суму цих мінімумів (треба отримати одинички в усіх рядках), цикл № 1 вибирає мінімум з цих сум. Легко бачити, що цикл № 1 має $\Theta(M)$ ітерацій, цикл № 2 $\Theta(N)$, цикли № 3 $O(M)$ обидва сумарно, що разом дає $O(N \cdot M^2)$, що при $M, N \leq 100$ цілком прийнятно. (Що таке “ O ” та “ Θ ” та як вони доводять це твердження, див., зокрема, стор. 12–13.)

Приклади реалізації можна бачити за посиланнями ideone.com/inBG9A (C++, ідея (А) 2-го абзацу) та ideone.com/gI8i5e (Pascal, ідея (Б)).

Якби обмеження були дещо більшими, можна було б замінити найбільш вкладені цикли 3а та 3б бінарними пошуками (див. посилання на стор. 34) й тим зменшити оцінку з $O(N \cdot M^2)$ до $\Theta(N \cdot M \cdot \log M)$. Але тут так робити не варто.

Задача С. «Двійкова система числення»

Вхідні дані: binary.in Обмеження часу: 1 сек
Результати: binary.out Обмеження пам'яті: 64 Мб

Василько нещодавно вивчив двійкову систему числення на уроці інформатики. Розв'язуючи своє завдання, він помітив цікаву особливість двійкових чисел, якщо додати одиницю до якогось числа, то в результаті отримаємо число, яке відрізняється від попереднього тим, що декілька його молодших розрядів містять обернене значення. Наприклад, додавши до числа 010_2 одиницю, отримаємо число 011_2 , яке відрізняється лише одним молодшим розрядом. А додавши до числа 0011_2 одиницю, отримаємо число 0100_2 , яке вже відрізняється трьома молодшими розрядами.

Василько вирішив написати програму, яка починаючи з якогось числа A , буде додавати до нього по одиниці, поки не утвориться число B . При цьому програма повинна підрахувати суму кількостей відмінних розрядів після кожної операції додавання.

Але тут пролунав дзвінок і Василько так і не встиг написати програму.

Завдання. Напишіть програму `binary`, яку Василько так і не встиг написати.

Вхідні дані. Перший рядок файлу `binary.in` містить два натуральних числа A та B ($1 \leq A < B \leq 10^{10}$). Кожне число записане в десятковій системі числення.

Результати. Ваша програма повинна створити текстовий файл `binary.out` і вивести туди єдине число, яке є відповіддю на задачу, теж в десятковій системі числення.

Приклад:

binary.in	binary.out
2 4	4

Розбір задачі. Щодо «лобового» підходу. Найочевидніший підхід (справді перебрати всі числа від A до B , перетворити у двійкове подання і для кожної пари сусідніх справді порівняти й порахувати біти) і не такий простий (як, власне, порівнювати біти?), і не має шансів узяти повні бали, бо, навіть якби кожна пара чисел порівнювалася за один процесорний такт (насправді явно більше), все одно при $A \approx 1$, $B \approx 10^{10}$ це було б кілька секунд.

Оскільки використання того, що числа в комп'ютері вже у двійковій системі, буває корисним у багатьох інших ситуаціях, пропонуємо охочим подивитися в ideone.com/d5w2Hx, як варто було б технічно реалізовувати цей підхід. Змінна k послідовно набуває значень $1_2, 10_2, 100_2, \dots$; "&" є побітовим and-ом; отже, $i \& k$ являє собою або 0 (коли відповідний біт числа i рівний 0), або k (якщо рівний 1); отже, умова $(i \& k) \neq ((i+1) \& k)$ виражає, що черговий біт чисел i та $i+1$ все ще різний (не однаковий). Крім того, з $(i+1) - i = 1$ випливає, що різними бітами в цих числах можуть бути лише кінець $011\dots 1$ в i та відповідний йому кінець $100\dots 0$ в $i+1$ (кількість одиниць, що перетворюються у нулі, може бути різною, в т. ч. 0; потім рівно один нуль перетворюється в одиницю); як тільки деякий біт однаковий — всі лівіші теж, цикл можна обривати.

І навіть така, досить вилизана у деталях, реалізація цього підходу набирає лише 50% балів.

Повний розв'язок. Розглянемо таблицю, в якій записані числа від 1 до $18_{Dec} = 10010_2$ та кількості розрядів, які змінюються при переході до цього числа від попереднього.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	<u>10</u>	<u>11</u>	<u>100</u>	<u>101</u>	<u>110</u>	<u>111</u>	<u>1000</u>	<u>1001</u>	<u>1010</u>	<u>1011</u>	<u>1100</u>	<u>1101</u>	<u>1110</u>	<u>1111</u>	<u>10000</u>	<u>10001</u>	<u>10010</u>
1	2	1	3	1	2	1	4	1	2	1	3	1	2	1	5	1	2

З неї легко бачити, що для непарних чисел змінюється 1 розряд; парних, але не кратних 4, змінюються 2 розряди; кратних 4, але не кратних 8, змінюються 3 розряди; і т. д. (Це неважко й довести: коли число кратне 2^k , але не кратне 2^{k+1} , його двійковий запис закінчується на рівно k нулів; при переході до цього числа від попереднього, змінюються вони всі, та ще один біт.)

А звідси вже можна порахувати, що на всьому проміжку від 1 до деякого N шукана сума всіх кількостей змін розряду може бути порохована як

$$K(N) = N + (N \operatorname{div} 2) + (N \operatorname{div} 2^2) + \dots + (N \operatorname{div} 2^k) + \dots, \quad (1)$$

де div — цілочисельне ділення; сума, хоч і завершується трикрапкою, фактично має лише приблизно $\log_2 N$ доданків (далі нулі). Чому така формула відповідає тому, що було наведено у табличці та подальших міркуваннях? Бо N — кількість взагалі всіх чисел у діапазоні, й для непарних вже пороховано правильно, а для всіх парних з потрібної кількості взято 1; далі, $(N \operatorname{div} 2)$ — кількість всіх парних чисел у діапазоні, й для парних не кратних 4 вже пороховано правильно (один раніше й один зараз), а для всіх кратних 4 з потрібної кількості взято 2; і т. д.

Ця формула працює лише «від 1 до деякого N », а треба від A до B . Тому суму від A до B варто виразити як $K(B) - K(A)$ (у більшості схожих ситуацій, як на стор. 61–62, $K(B) - K(A - 1)$, з міркувань « A у проміжку, його виключати не треба», а тут все-таки $K(B) - K(A)$, бо перехід від $A - 1$ до A не входить у потрібний проміжок).

Реалізуйте описаний алгоритм самостійно. $K(N)$ варто оформити функцією (підпрограмою), щоб могли викликати з різними значеннями аргумента, не дублюючи сам код обчислень згідно (1). Така реалізація повинна працювати практично миттєво, бо її складність всього лиш $O(\log B + \log A) = O(\log B)$. Так що обмеження $B \leq 10^{10}$ в цьому ракурсі сприймається як легкий тролінг, бо і вимагає того ж 64-бітового типу даних, і схиляє до пошуку дещо повільніших алгоритмів (як-то $O(\sqrt{B})$). Що ж, так теж буває...

Задача D. «Лабіринт»

Вхідні дані: maze.in Обмеження часу: 1 сек
Результати: maze.out Обмеження пам'яті: 64 Мб

Тезею з лабіринту Мінотавра допоміг вийти клубок ниток Аріадни. Ви можете замість клубка використовувати персональний комп'ютер.

Потрібно написати програму, яка вводить маршрут Тезея в лабіринті і знаходить найкоротший зворотний шлях, по якому Тезей зможе вийти з лабіринту, не заходячи в тупики і не роблячи кругів.

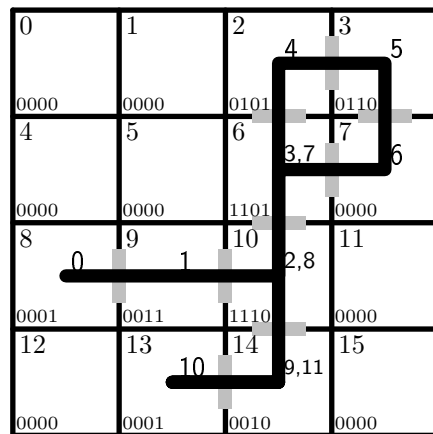
Завдання. Напишіть програму maze, яка читає один рядок тексту (шлях Тезея в лабіринті) та знаходить найкоротший зворотний шлях.

Вхідні дані. Файл maze.in містить маршрут Тезея, який представлений одним рядком, що складається з букв: N, S, W, E. Довжина рядка не більше 200 символів. Букви позначають:

```

*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****
*.*.*.*
*****

```



На рисунку зображено приклад з умови; ребра виділено сірими розривами у сторонах клітинок. Числа трохи правіше й вище центрів клітинок позначають порядок, в якому їх відвідував Тезей. Числа у лівих верхніх кутах клітинок позначають глобальні номери, а між рисунками зображено відповідні цим номерам і способу (А) списки суміжності. Послідовності з чотирьох нуликів та/або одиниць у лівих нижніх кутах клітинок позначають, як мають описуватися переходи з кожної клітинки способом (Б). Нарешті, окреме поле з самих лише зірочок («стін») і крапочок («проходів»), зображене ліворуч унизу, позначає поле, зображене праворуч, способом (В); для порівняння, ліворуч угорі наведено поле з таких само клітинок, яке не містить ніяких переходів Тезея.

N — один крок на північ,
S — один крок на південь,

W — один крок на захід,
E — один крок на схід.

Результати. Ваша програма повинна створити текстовий файл `maze.out` і вивести туди обернений шлях мінімальної довжини. Якщо відповідей декілька, виведіть будь-яку. Зверніть увагу, що можна іти лише по слідам Тезея і не можна скорочувати шлях, ідучи навпростець.

Приклад:

maze.in	maze.out
EENNESWSSWE	NWW

Розбір задачі. Головним є пошук найкоротшого шляху, тож природним є алгоритм пошуку вшир (він же пошук у ширину, він же BFS), причому у варіанті з відновленням шляху. Його описи знайдіть у літературі чи Інтернеті. А особливості графу, до якого його слід застосувати, розглянемо.

В умові нема понять «клітинка» та її «сторона», але їх варто ввести, бо без них важко формалізувати «крок» і «по слідам». Будемо вважати, що Тезей рухається клітинками деякого прямокутника, і кожен з його «кроків» являє собою перехід на клітинку вище (N), нижче (S), лівіше (W) чи правіше (E). Розміри прямокутника, на жаль, невідомі; але, на щастя, при співвідношенні «до 200 кроків, ліміт пам'яті до 64 Мб», це не є серйозною проблемою. Як один з варіантів (гроздкий, зате ідейно простий), можна вважати, що і рядки, і стовпчики занумеровані від 0 до 400 (обидві межі включно), і старт знаходиться у клітинці [200][200]. Можна придумати й інші, економніші за обсягом пам'яті, способи, але детальнішу дискусію про це залишимо за межами цього збірника.

Очевидно, що вершинами графа є клітинки; ребрами — не просто сусідні клітинки, а такі сусідні, що між ними Тезей проходив; граф неорієнтований, тобто те, що Тезей проходив хоча б в одному з напрямків, дозволяє перехід між цими клітинками у будь-якому напрямку.

Є щонайменше три природні способи подавати такий граф: (А) занумерувати клітинки (наприклад, але не обов'язково, рядок за рядком), після чого використати звичайні списки суміжності; (Б) зберігати для кожної клітинки чотири булеві значення: чи є проходи нагору (N), униз (S), ліворуч (W), праворуч (E) (можна в іншому порядку, саме цей взятий лише тому, що так в умові); (В) перетворити поле з $n \times m$ клітинок, де стінки між клітинками можуть містити чи не містити проходи, у поле з $(2n+1) \times (2m+1)$ клітинок, де проходи (ребра) є між усіма сусідніми крапочками, а стіни (відсутності проходів) позначені зірочками. Приклади всіх цих підходів зображені на рис. нагорі стор. 21. Само собою, потреби робити зразу все нема; слід вибрати один найзнайоміший з цих способів, або придумати який-небудь свій.

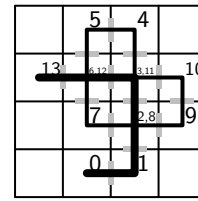
Насамкінець, пояснимо неправильність деяких «ідей», які часто виникають при недостатньо ретельному аналізі цієї задачі. По-перше, не можна «просто рахувати зсуви по вертикалі й по горизонталі», бо сказано «можна іти лише по слідам Тезея», і, наприклад, для вхідних даних EENNESWSSW

III (обласний) етап 2012/13 навч. року
Черкаська обл., 23.02.2013

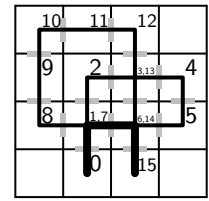
(як в умові, лише без останнього переходу) не можна відповідати “NW”, бо це проходило б крізь межу між клітинками, яку Тезей не перетинав; правильною була б відповідь ENWW.

По-друге, ідея «шукати та вирізати цикли» правильна часто, але теж не завжди. Зокрема, як видно з рисунка нагорі стор. 21, для прикладу з умови, 2-а та 8-а клітинки збігаються; 9-а та 11-а теж; значить, можна викинути все, що між 2-ю та 8-ю, об'єднавши їх; аналогічно між 9-ю та 11-ю.

Але це погано працює, наприклад, для вхідних даних ENNNWSSEENWWW (де важливо ще правильно здогадатися, який із кількох циклів вирізати, щоб отримати правильну відповідь EESSW, а не, наприклад, EEESWSW), і геть не працює, наприклад, для вхідних даних NNEESWWWNNEESSS (де правильна відповідь NWS взагалі не отримується таким способом).



ENNNWSSEENWWW



NNEESWWWNNEESSS

3.2 III (обласний) етап 2012/13 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання №8).

Задача А. «Квартири — flats»

Вхідні дані: клавіатура (стандартний потік введення) Обмеження часу: 1 сек
Результати: екран (стандартний потік виведення) Обмеження пам'яті: 64 Мб

Багатоквартирний житловий будинок складається з кількох під'їздів. Кількість поверхів у всіх під'їздах однакова й дорівнює s . На кожній сходовій клітці розміщено по k квартир. Як відомо, нумерація квартир розпочинається з першого під'їзду (від 1-го по останній поверх), потім продовжується у 2-му, і так далі.

Завдання. Напишіть програму flats, яка визначає в якому під'їзді і на якому поверсі розміщено квартиру номер N .

Вхідні дані. Програма flats повинна прочитати зі стандартного входу (клавіатури) три цілі числа s ($3 \leq s \leq 25$), k ($2 \leq k \leq 6$) та N ($1 \leq N \leq 999$), кожне в окремому рядку.

Результати. Програма flats повинна вивести на стандартний вихід (екран) два числа, кожне в окремому рядку: у 1-му рядку — номер під'їзду, у 2-му — номер поверху.

Приклади:

Вхідні дані	Результати
9	1
5	4
17	

Вхідні дані	Результати
3	2
4	3
24	

Вхідні дані	Результати
3	3
4	1
25	

Примітка. Не менш ніж у половині тестів $s = 9$, $k = 4$.

Розбір задачі. Задачу в принципі можна розв'язувати по-різному, в тому числі й з використанням циклів, щоб перебирати можливі номер під'їзду й номер поверху (при $N \leq 999$ це вкладається в обмеження часу). Але пряма формула природніша, і, мабуть, простіша, тож зосередимось на ній.

Для мов програмування, де є заокруглення вгору (ceil, Ceiling, тощо), можна використати його (разом з діленням у типі з рухомою комою). Якби був лише один під'їзд, то можна було б казати, що на 1-му поверсі розміщено квартири з №1 по № k , на 2-му — з № $(k+1)$ по № $(2 \cdot k)$, тощо; отже, для квартир з 1-го поверху $0 < n/s \leq 1$; з 2-го, $1 < n/s \leq 2$; і т. д. Звідси, номер поверху можна було б виражати як $(\text{int})(\text{ceil}(((\text{double})n)/k))$ (C/C++; частину дужок можна і прибрати; забезпечити, що ділення робиться в типі double, можна й іншими засобами, але якось це зробити треба), чи $\text{ceil}(N/k)$ (Python3), тощо. Оскільки під'їзд не завжди один, в точності ця формула малокорисна, але потрібні можна побудувати аналогічно. Скажімо, з того, що 1-й під'їзд містить квартири від 1 до $s \cdot k$ (включно), 2-й — від $s \cdot k + 1$ до $2 \cdot s \cdot k$, і т. д., слідує, що визначити номер під'їзду можна виразом $(\text{int})(\text{ceil}(((\text{double})N)/(s \cdot k)))$ (C/C++;), чи $\text{ceil}(N/(s \cdot k))$ (Python3), тощо. Якщо номер під'їзду вже пораховано і присвоєно у змінну (наприклад, p), можна обчислити «номер квартири у межах під'їзду» (в тих самих межах від 1 до $s \cdot k$ включно) як $n - s \cdot k \cdot p$; позначимо

цю величину як `n_p`, після чого номер поверху можна обчислити згідно раніше розглянутих міркувань, як `(int)(ceil(((double)n_p)/k))` (C/C++), чи `ceil(n_p/k)` (Python3), тощо.

А що робити, якщо писати на Паскалі, де нема заокруглення вгору? По-перше, якщо доступний PascalABC, там функція `Ceil` є; головний недолік PascalABC (що він помітно повільніший за Free Pascal) у цій задачі абсолютно не важливий. По-друге, ніщо не заважає переписати те саме, реалізуючи заокруглення вгору самому. Є щонайменше два способи виразити `ceil(a/b)`: (A) `res:=a div b; if a mod b <> 0 then inc(res);` (Б) `(a+b-1) div b`.

Варто відзначити: якби нумерація квартир, поверхів і під'їздів була не з 1, а з 0, всі формули були б простіші. А саме: номер під'їзду виражався б як `N/(s*k)` (C-подібними мовами) чи `N//(s*k)` (Python3) чи `N div (s*k)` (Pascal); номер поверху виражався б як `(N/k)%s` (C-подібними мовами) чи `(N//k)%s` (Python3) чи `(N div k) mod s` (Pascal). Власне, ці формули настільки простіші, що може мати сенс навіть перетворити прочитане `N` (зменшивши на 1), провести обчислення за цими формулами, й перетворити кожен з отриманих результатів, збільшивши на 1. (Само собою, перетворенням підлягають лише номери квартири, поверху й під'їзду, ні в якому разі не кількості `s` та `k`.) А ще це пояснює, чому, хоча людям нумерувати з 0 незручно, більшість сучасних мов програмування мають безальтернативну нумерацію масивів з 0: таке спрощення перерахунків з'являється не лише в цій задачі, а й у багатьох схожих (зокрема, при перетворенні індексів багатовимірного масиву в адресу в пам'яті та зворотньо).

Задача В. «Послідовність — series»

Вхідні дані: `series.dat` Обмеження часу: 1 сек
Результати: `series.sol` Обмеження пам'яті: 64 МБ

Дана послідовність ненульових цілих чисел. Визначити, скільки разів в цій послідовності змінюється знак.

Завдання. Напишіть програму `series`, яка б визначала, скільки разів в цій послідовності змінюється знак.

Вхідні дані. Перший рядок вхідного файлу `series.dat` містить ціле число N — кількість членів послідовності ($1 \leq N \leq 1000$). Наступний рядок містить N цілих чисел, розділених пробілами, кожне число не менше за -32768 і не більше за 32767 .

Приклад:

series.dat	series.sol
5 10 -4 12 56 -4	3

Розбір задачі. Завдяки гарантії, що числа ненульові, «змінюється знак» можна виражати досить простим виразом `((a[i-1]>0) and (a[i]<0)) or ((a[i-1]<0) and (a[i]>0))`, або ще простішим `a[i-1]*a[i] < 0`. Лишається тільки перевірити цю умову для всіх пар сусідніх елементів і підрахувати кількість разів, коли ця умова дала результат `true`. Мовою Pascal, при нумерації масиву з 1, цикл матиме вигляд `for i:=2 to n do ...`; C-подібними мовами, при нумерації з 0, `for(int i=1; i<n; i++) ...`; мовою Python, теж при нумерації з 0, `for i in range(1,n): ...`.

Якби гарантії відсутності нулів не було, задача була би складнішою. По-перше, треба було б чи то знайти в умові задачі, чи то взяти в журі чітке означення зміни знаку: `“-2, 0, +1”` — це одна зміна знаку чи дві? А `“-2, 0, 0, 0, +1”`? `“+2, 0, +3”` містить зміну знаку чи ні? А `“-2, 0, -3”`? А як бути з нулями на початку чи наприкінці? По-друге, треба було б переглядати послідовність, постійно перебуваючи в одному зі станів «послідовність лише починається, чисел ще не було», «від самого початку послідовності були лише нулі», «останнім був нуль чи кілька нулів, перед ними додатне число», «останнім був нуль чи кілька нулів, перед ними від'ємне число», «останнім було додатне число», «останнім було від'ємне число» (перелік приблизний, точний може залежати від уточнень, згаданих у «по-перше»), та приймаючи рішення щодо додавання чи не додавання нової зміни знаку згідно стану та значення наступного числа.

Задача С. «Декадні числа — dnumbers»

Вхідні дані: dnumbers.dat Обмеження часу: 3 сек

Результати: dnumbers.sol Обмеження пам'яті: 64 Мб

Декадні числа — це цілі додатні числа в яких сума i -тої цифри зліва та i -тої цифри справа завжди дорівнює 10. Наприклад число 13579 є декадним, так як $1 + 9 = 10$, $3 + 7 = 10$, $5 + 5 = 10$ (в даному випадку цифра 5 є третьою зліва і одночасно третьою справа).

Перші кілька декадних чисел в порядку зростання: 5, 19, 28, 37, 46, 55, 64, 73, 82, 91, 159, ...

Завдання. Напишіть програму dnumbers, яка б знаходила n -е в порядку зростання декадне число.

Вхідні дані. В єдиному рядку файлу dnumbers.dat міститься число n ($1 \leq n \leq 2^{31}$).

Результати. Ваша програма має створити текстовий файл dnumbers.sol і вивести туди єдине число, яке є шуканим n -им декадним числом.

Приклади:

dnumbers.dat	dnumbers.sol	dnumbers.dat	dnumbers.sol
2	19	11	159

Розбір задачі. «Лобовий» підхід (перебирати всі підряд числа, починаючи з 1, перевіряти кожне, чи декадне) перестане поміщатись у 3 сек десь при $n \approx 1000$. Такий розв'язок набирає деякі бали (навіть чималі, 11 з 25; правда, взяти це можна, лише здавши такий розв'язок). Якщо хотіти розв'язати задачу повністю, спинятися на цьому не можна. Але цей розв'язок може бути корисним, щоб подивитися на довшу, ніж в умові, послідовність декадних чисел: 5, 19, 28, 37, 46, 55, 64, 73, 82, 91, 159, 258, 357, 456, 555, 654, 753, 852, 951, 1199, 1289, 1379, 1469, 1559, 1649, 1739, 1829, 1919, 2198, 2288, ... Ще подивимося на останні 4-цифрові й перші 5-цифрові: ..., 9641, 9731, 9821, 9911, 11599, 12589, 13579, 14569, 15559, 16549, 17539, 18529, 19519, 21598, 22588, ...

Звідси вже неважко побачити такі властивості декадних чисел:

1. Декадні числа з непарною кількістю цифр, крім найпершого числа 5 (інакше кажучи, $(2k+1)$ -цифрові при $k \geq 1$) майже однакові з відповідними $2k$ -цифровими, лише містять середню цифру 5: 19 і 159, 28 і 258, тощо.
2. Хоч для $2k$ -цифрових, хоч для $(2k+1)$ -цифрових (при $k \geq 1$) послідовних декадних чисел, старшими k цифрами є послідовні k -значні числа, *пропускаючи ті, що містять цифру 0 (одну чи кілька)*.
3. Молодші k цифр можуть бути отримані зі старших k цифр: наймолодша — як 10 мінус найстарша, і т. д., симетрично назустріч одна одній.

(Хто бачить усе це й без написання «лобової» програми — молодець. Але біда в тому, що не всі так можуть. Тому наголошено, що «лобова» програма може бути корисною як для того, щоб здати її й отримати частину балів, так і для того, щоб побачити властивості.) Кількість k -цифрових чисел, які не містять жодної цифри 0, очевидно, 9^k (є k розрядів, кожен може набувати будь-яке значення від 1 до 9). А звідси вже легко отримати алгоритм:

1.
 - Якщо $n = 1$, то це 1-цифрове декадне число 5;
 - інакше, зменшимо n на 1 (щоб компенсувати, що 1 штука 1-цифрових декадних чисел розглянута й пропущена);
 - якщо (після зменшення) $n \leq 9$, то це 2-цифрове декадне число;
 - інакше, зменшимо n ще на 9 (щоб компенсувати, що 9 штук 2-цифрових декадних чисел розглянуті й пропущені);
 - якщо (після обох зменшень) $n \leq 9$, то це 3-цифрове декадне число;
 - інакше, зменшимо n ще на 9 (щоб компенсувати, що 9 штук 3-цифрових декадних чисел розглянуті й пропущені);
 - якщо (після всіх зменшень) $n \leq 9^2 = 81$, то це 3-цифрове декадне число;
 і так далі.
2. Коли у попередньому пункті отримали кількість цифр та номер серед чисел з такою кількістю цифр, його можна перетворити у старші k розрядів, а саме:
 - (а) відняти 1;

- (b) перетворити у k -цифрове число в 9-вій системі (можливо, з 0-ми спереду);
 (c) збільшити кожну 9-ву цифру 0–8 на 1, отримавши цим цифри 1–9.
3. Лишається тільки дописати молодші k цифр на основі старших (попередньо вставивши “5”, якщо кількість цифр непарна).

Наприклад, знайдемо вручну 2013-е декадне число. $2013 > 1$, тому число не 1-цифрове, і далі номером вважаємо $2013 - 1 = 2012$; $2012 > 9$, тому число не 2-цифрове, і далі номером вважаємо $2012 - 9 = 2003$; $2003 > 9$, тому число не 3-цифрове, і далі номером вважаємо $2003 - 9 = 1994$; $1994 > 9^2 = 81$, тому число не 4-цифрове, і далі номером вважаємо $1994 - 81 = 1913$; $1913 > 9^2 = 81$, тому число не 5-цифрове, і далі номером вважаємо $1913 - 81 = 1832$; $1832 > 9^3 = 729$, тому число не 6-цифрове, і далі номером вважаємо $1832 - 729 = 1103$; $1103 > 9^3 = 729$, тому число не 7-цифрове, і далі номером вважаємо $1103 - 729 = 374$; $374 \leq 9^4 = 6561$, тому число 8-цифрове, причому номер серед 8-цифрових становить 374 (при нумерації з 1), або ж 373 (при нумерації з 0). Це число 373 треба перетворити у 4-цифрове у 9-вій системі числення, тобто 0454_9 . Що означає, що старшими $k = 4$ цифрами відповіді є 1565. Оскільки шукане число 8-цифрове (число 8 парне), вставляти цифру 5 не треба. Так що лишається утворити другу половину числа-відповіді: $5 \xrightarrow{10-5} 5$, $6 \xrightarrow{10-6} 4$, $5 \xrightarrow{10-5} 5$, $1 \xrightarrow{10-5} 9$, що остаточно дає відповідь 15655459.

Складність цього розв'язку $O(\log N)$, бо кількість цифр у «половині» відповіді становить $k \approx \log_9 \frac{N}{2}$, а кожен з великих етапів “1”, “2”, “3”, очевидно, має складність $\Theta(k)$. При максимально можливому значенні вхідного $N = 2^{31}$, відповідь виявляється 21-цифровою, тобто не поміщається у 64-бітовий тип. Тому варто або виводити результат у файл-відповідь поцифрово, або формувати результат як рядок, але не варто формувати його як число.

Задача D. «Кава — coffee»

Вхідні дані: coffee.dat Обмеження часу: 1 сек

Результати: coffee.sol Обмеження пам'яті: 64 Мб

Програміст Василь дуже любить пити каву. Про нього ще говорять, що він перетворює каву в код. Василь знає, якщо він вип'є чашку кави перед виконанням певного завдання, то він витратить на нього на 20% менше часу, ніж без кави. Але на заварювання кави теж необхідно витратити певний час.

Вам необхідно визначити, за яку мінімальну кількість робочих днів Василь зможе справитися з усіма своїми завданнями, якщо у нього є запас кави на K чашок. Василь вже наперед визначив необхідну кількість часу для кожного завдання. Завдання необхідно виконувати послідовно. Якщо залишок робочого часу не дозволяє виконати наступне завдання, то Василь почне його виконувати наступного дня.

Зверніть увагу, що магічна дія чашки кави впливає лише на одне завдання, і що не можна випивати перед виконанням завдання більше однієї чашки кави.

Завдання. Напишіть програму coffee, яка б знаходила мінімальну кількість днів, які необхідно потратити на виконання всіх завдань.

Вхідні дані. Перший рядок вхідного файлу coffee.dat містить три цілих числа N , K , L — кількість завдань, кількість чашок кави та тривалість заварювання однієї чашки кави ($1 \leq N \leq \leq 1000$, $0 \leq K \leq 1000$, $1 \leq L \leq 100$). Наступний рядок містить N цілих чисел, розділених пробілами — необхідний обсяг часу для виконання кожного завдання (час задано в хвилинах, кожне число не менше за 1 і не більше за 480).

Результати. Ваша програма має створити текстовий файл coffee.sol і вивести туди єдине ціле число — мінімальну кількість днів, які необхідно потратити на виконання всіх завдань.

Приклади:	coffee.dat	coffee.sol	coffee.dat	coffee.sol
	5 1 10 10 10 10 100 360	1	5 2 5 200 281 240 240 480	3

Розбір задачі. Перш за все, покажемо неправильність «природного» (для тих, хто погано знає задачі такого роду) міркування «Каву слід застосовувати до K найдовших завдань». Візьмемо останній приклад з умови, але замінивши K на 1. Без застосування кави розподіл за днями виходить такий: $(200) + (281) + (240 + 240) + (480)$, тобто 4 дні, бо можливо об'єднати два завдання по 240,

а решта завдань потребують кожне окремого дня. Найдовше завдання має тривалість 480, але від того, що воно скоротиться до $480 \times 0,8 + 5 = 389$, сумарна кількість днів не зменшиться, бо $(200) + (281) + (240 + 240) + (389)$ все одно дає 4 дні. А якщо застосувати цю каву до будь-якого з перших двох завдань ($200 \times 0,8 + 5 = 165$ чи $281 \times 0,8 + 5 = 229,8$), з'явиться можливість об'єднати ці завдання в один день, як $(165 + 281) + (240 + 240) + (480)$ чи $(200 + 229,8) + (240 + 240) + (480)$.

А як правильно? Правильним є розв'язок, водночас і багато в чому очевидний, і досить нестандартний в одному моменті. Розв'яжемо задачу динамічним програмуванням (воно ж динпрог, воно ж ДП), поставивши серію підзадач «За який мінімальний час $T(i, j)$ можливо виконати завдання з 1-го по i -е, використавши не більш, ніж j чашок кави?», причому так, щоб ця сумарна для завдань з 1-го по i -е тривалість вимірювалася парою «кількість використаних днів; кількість хвилин, використаних останнього дня». Нехай ці кількості зібрані у структуру `tot_time`, наведену праворуч.

Динамічне програмування передбачає, що треба переходити від під-
задачі до підзадачі (отже, додавати одне завдання до сукупності завдань)
та вибирати мінімальне значення з кількох варіантів (отже, вибирати, що
менше). Тому треба задати правило, як додавати, і правило, як порівнювати.

Якщо говорити у термінах C++, то це `operator +` та `operator <`, наведені у фрагментах коду праворуч. Звісно, іншими мовами програмування це теж можна реалізувати, але трохи менш зручно.

Наприклад: якщо до пари (1; 200) додати 100 хв, має вийти пара (1; 300), бо ці 100 хв можна додати до того самого дня; а якщо такі самі 100 хв додавати до пари (1; 400), має вийти пара (2; 100), бо раз помістити в той самий день не можна ($400 + 100 = 500 > 480 = 8 \times 60$), то треба починати новий день, і з того дня буде зайнято стільки часу, скільки триває нове завдання, яке не помістилося в попередній.

З порівнянням ще простіше: при різних кількостях днів, треба враховувати лише ці різні кількості днів, а при однакових — врахувати кількість хвилин.

Так от, якщо операціям “+” та “<” надати саме такий смисл, то можна сформулювати й використати нижче наведені тривіальні підзадачі та рівняння ДП.

$$T(0, j) = (0; 480) \quad \text{для всіх } 0 \leq j \leq K \quad (2)$$

(щоб виконати 0 завдань, досить 0 днів; 480 хвилин (а не 0), щоб не можна було додати нові завдання в той самий неіснуючий 0-й день; для всіх j , а не лише $j = 0$, бо у серії підзадач вказано «не більш, ніж j чашок кави»).

$$T(i, 0) = T(i - 1, 0) + a_i \quad \text{для всіх } 1 \leq i \leq N \quad (3)$$

(де a_i — тривалість i -го завдання згідно вхідних даних; “+” має смисл, заданий вище у `operator +`; це правильно, бо $j = 0$ означає «взагалі не вживати кави», тож нема іншого вибору, крім як додати це завдання зі стандартною тривалістю).

$$T(i, j) = \min \left\{ \begin{array}{l} T(i - 1, j) + a_i, \\ T(i - 1, j - 1) + 0,8 \cdot a_i + L \end{array} \right\} \quad \begin{array}{l} \text{для всіх} \\ 1 \leq i \leq N, \\ 1 \leq j \leq K \end{array} \quad (4)$$

(смисли a_i та “+” такі самі, `min` вибирається згідно вищезгаданого `operator <`; тут розглядаються варіанти «не використовувати каву» (верхній аргумент `min`) та «використати» (нижній) і вибирається кращий з них; якщо каву не використовувати, то сумарно на всі попередні $i - 1$ завдань доступно так само j чашок, а якщо використовувати, то $j - 1$, бо без цієї одної, що зараз).

Якщо згадати, що 80% від цілого числа не завжди є цілим числом, виявляється, що поєднати рівно таке рівняння (4) з рівно такою структурою `tot_time` неможливо. Теоретично можна

```
struct tot_time {
    int days, minutes;
};

tot_time operator +
    (const tot_time &tt_old,
     int minutes_to_add) {
    tot_time res;
    int sum_minutes = tt_old.minutes
        + minutes_to_add;
    if(sum_minutes <= 8*60) {
        res.days = tt_old.days;
        res.minutes = sum_minutes;
    } else {
        res.days = tt_old.days + 1;
        res.minutes = minutes_to_add;
    }
    return res;
};

bool operator <
    (const tot_time &t1,
     const tot_time &t2) {
    if(t1.days != t2.days)
        return t1.days < t2.days;
    else // t1.days == t2.days
        return t1.minutes < t2.minutes;
}
```

подати кількість хвилин як `double`; але, враховуючи стор. 14–14 та факт, що сама константа 0.8 вже не є скінченним двійковим дробом, це погана ідея. Краще, наприклад, замінити поле `minutes` на поле `seconds`, правильно змінивши всі константи (тоді, якщо не використовувати каву, то тривалість завдання у хвилинах перетворюється у секунди як $60 * a[i]$, а якщо використовувати, то як $48 * a[i] + 60 * L$), або ще якимсь чином лишитися у точній цілочисловій арифметиці.

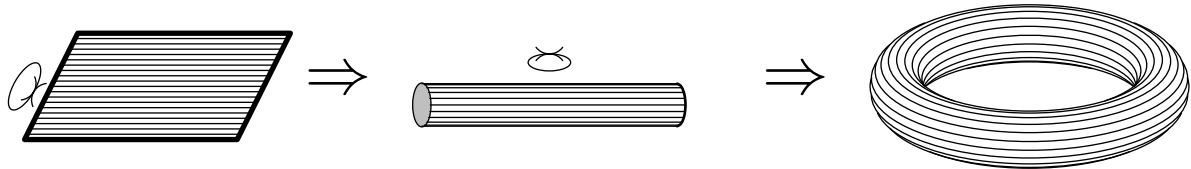
3.3 Обласна інтернет-олімпіада 2013/14 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №10).

Задача А. «Тор»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Як відомо, *тор* — це поверхня бублика, яку можна отримати таким чином: узяти прямокутник розміром n клітинок по вертикалі на m клітинок по горизонталі, склеїти верхню сторону з нижньою (отримається циліндр), потім закрутити циліндр у бублик і склеїти ліву сторону з правою.



Назвемо дві клітинки *сусідніми*, якщо вони мають спільну сторону. Нехай за одну секунду можна перейти з клітинки до будь-якої сусідньої з нею. За який мінімальний час можна потрапити з клітинки $(r_1; c_1)$ у клітинку $(r_2; c_2)$? Перше число у позначенні клітинки — номер рядка початкового прямокутника, друге — номер стовпчика.

Вхідні дані. Програма повинна прочитати зі стандартного входу (клавiатури) шість натуральних чисел, у порядку n, m, r_1, c_1, r_2, c_2 . Виконуються обмеження: $2 \leq n, m \leq 1\,000\,000\,000$, $1 \leq r_1, r_2 \leq n$, $1 \leq c_1, c_2 \leq m$.

Результати. Програма має вивести на стандартний вихід (екран) єдине ціле число — мінімальний час.

Приклади:	Вхідні дані	Результати	Вхідні дані	Результати
	10 10 5 5 1 1	8	10 10 9 9 1 1	4

Розбір задачі. Необхідно дістатися з точки (r_1, c_1) тору у точку (r_2, c_2) тору по «зацикленому» простору тора. Скориставшись відомим принципом незалежності переміщень, можна побачити, що переміщення в горизонтальному і вертикальному напрямках не залежать одне від одного. Розглянемо переміщення як перетин вертикальних і горизонтальних сторін клітинки. Незалежно від траєкторії, необхідно перетнути лише певну конкретну кількість вертикальних ліній (позначимо цю кількість Δr) і горизонтальних ліній (відповідно Δc). (Звісно, якщо не розглядати відверто не мінімальні шляхи, де одна й та ж лінія перетинається багатократно.) Тоді відповідь на задачу є $\Delta r + \Delta c$, оскільки для зміни будь-якої з координат на 1 необхідно затратити 1 секунду.

Є 2 способи переміститися з рядка r_1 у r_2 — перетинаючи край початкового (до згинів і склеювань) прямокутника і не перетинаючи. Вважаємо спочатку, що $r_1 < r_2$. Тоді не перетинаючи край витратимо $(r_2 - r_1)$ сек, а перетинаючи спочатку доберемося до рядка 1 $((r_1 - 1)$ сек), потім до рядка n (1 сек), і з нього у r_2 — $((n - r_2)$ сек). Сумарно $(r_1 - r_2 + n)$ сек. Тобто, треба вибрати мінімум зі значень $(r_2 - r_1)$ (не перетинаючи край) і $(r_1 - r_2 + n)$ (через край). Але це лише для випадку $r_1 < r_2$, а при $r_1 > r_2$ аналогічними міркуваннями отримуються схожі, але інші формули $(r_1 - r_2)$ і $(r_2 - r_1 + n)$. Щоб не задумуватися, яка з координат більша, можна використати формулу

$$\Delta r = \min((r_1 - r_2 + n) \bmod n, (r_2 - r_1 + n) \bmod n).$$

З Δc слід вчинити аналогічно, потім додати $\Delta r + \Delta c$.

Задача В. «Паркет–1»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Щоб зобразити за допомогою паркету Супер-Креативний Візерунок, треба N_1 дощечок розмірами 1×1 , N_2 дощечок розмірами 2×1 , N_3 розмірами 3×1 , N_4 розмірами 4×1 та N_5 дощечок розмірами 5×1 . Купити можна лише дощечки розмірами 5×1 . Дощечки можна різати, але не можна склеювати. Наприклад, коли потрібні п'ять дощечок 2×1 , їх не можна зробити з двох дощечок 5×1 , але можна з трьох. Для цього дві з них розріжемо на три частини 2×1 , 2×1 та 1×1 кожна, а третю — на дві частини 2×1 та 3×1 . Отримаємо потрібні п'ять дощечок 2×1 , а дві дощечки 1×1 та одна 3×1 підуть у відходи.

Напишіть програму, яка, прочитавши кількості дощечок N_1 , N_2 , N_3 , N_4 та N_5 , знайде, яку мінімальну кількість дощечок 5×1 необхідно купити.

Вхідні дані. Вхідні дані слід прочитати зі стандартного входу (клавiатури). Це будуть п'ять чисел N_1 , N_2 , N_3 , N_4 та N_5 (саме в такому порядку), розділені пропусками (пробілами).

Результати. Єдине число (скільки дощечок треба купити) виведіть на стандартний вихід (екран).

Приклади:

Вхідні дані	Результати
0 5 0 0 0	3

Вхідні дані	Результати
1 1 1 1 1	3

Оцінювання. Усі кількості невід'ємні; 90 балів (з 250) припадатиме на тести, в яких сумарна кількість $N_1 + N_2 + N_3 + N_4 + N_5$ перебуває в межах від 0 до 20, ще 80 балів — від 100 до 10 000, решта 80 балів — від 500 000 000 до 2 000 000 000.

Здати потрібно одну програму, а не для кожного випадку окремо; різні обмеження вводяться виключно для того, щоб дати приблизне уявлення, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі. Задача складна необхідністю дуже акуратно розглядати випадки, але проста тим, що потрібні *лише* розгалуження та присвоєння.

Алгоритм наведено внизу стор. 29. Мова коду — Python3; “=” (одинарне) — присвоєння; “==” (подвійне) — перевірити, чи дорівнює; “%” — залишок від ділення (mod); “//” (подвійне) — цілочисельне ділення (div); “a+=b” — те само, що a=a+b, тобто до a додати b і покласти результат у ту саму змінну a; аналогічно “a-=b”, “a%=b”. Спочатку, змінні n1, n2, n3, n4 та n5 містять потрібні кількості дощечок розмірами 1×1 , 2×1 , 3×1 , 4×1 та 5×1 відповідно. Протягом роботи алгоритму значення деяких зі змінних n1, n2, n3, n4 або n5 можуть зменшуватися — по мірі того, як враховуємо відповідні кількості у змінній res, яка наприкінці міститиме остаточну відповідь.

Аргументація у коментарях зовсім не зайва (див. також стор. 12). Задача є частковим випадком задачі упаковки корзин (*bin packing problem*), для загального вигляду якої не відомо правильного швидкого алгоритма (про простий не йдеться; наука не знає навіть складного, щоб був правильний та швидкий одночасно). Наприклад, *якби* розміри початкових дощечок були не 5×1 , а 7×1 , і треба було сформувати 2 шт. 3×1 і 4 шт. 2×1 , то треба було б узяти дві стандартні 7×1 і розрізати кожен на $(3 \times 1) + (2 \times 1) + (2 \times 1)$, діючи *всупереч* ідеї «перш за все вибирати якнайбільші розміри».

Посилити аргументацію (провести більш строге доведення цих коментарів) можна приблизно так. На усіх кроках (3), (5)–(8) справедливо «немає смислу викидати у відходи те, що можна використати; навіть якщо виявиться, що його можна узяти й пізніше, ми нічого не втрачаємо, узявши раніше». На кроках (5), (6), (8) — «де можна, варто віддавати перевагу 2×1 над 1×1 , бо замість кожної 2×1 завжди можна зробити хоч дві 1×1 , хоч одну, а зробити 2×1 замість двох 1×1 можна далеко не завжди». На кроках (9), (10), формується залишок тих дощечок 1×1 , які ніяк не могли бути сформовані раніше, бо, станом на початок кроку (9), $n1 > 0$ *лише* якщо на усіх попередніх кроках усі дощечки використовувалися геть без відходів.

А для більших розмірів стандартної дощечкидесь колись настає момент, коли узагальнення таких міркувань перестають бути правильними...

Задача С. «Сума квадратів»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Для заданого натурального числа N , визначити, скількома різними способами можна розкласти його в суму двох точних додатних квадратів.

Іншими словами: для заданого N , з'ясувати, скільки є різних способів подати його як $N = x^2 + y^2$, причому x та y являють собою цілі строго додатні числа, а розкладення, в яких значення x та y лише обміняні місцями, вважаються однаковими.

Вхідні дані. Вхідні дані — натуральне число N — слід прочитати зі стандартного входу (клавiатури).

Результати. Результат — знайдену кількість способів — слід вивести на стандартний вихід (екран).

Приклади:

Вхідні дані	Результати	Примітка
16	0	Розкласти у суму <i>додатних</i> точних квадратів неможливо
10	1	Єдине розкладення $10 = 1^2 + 3^2$
4225	4	Чотири різні розкладення: $4225 = 16^2 + 63^2 = 25^2 + 60^2 = 33^2 + 56^2 = 39^2 + 52^2$

100 балів (з 250) припадатиме на тести, в яких $1 \leq N \leq 1234$.

Ще 50 балів — на тести, в яких $12345 \leq N \leq 123456$.

Решта 100 балів — на тести, в яких $12345678 \leq N \leq 123456789$.

Фрагмент коду	Коментар
<code>res = n5</code>	(1) На кожную дощечку 5×1 неминуче потрібна окрема ціла дощечка.
<code>res += n4</code>	(2) На кожную 4×1 теж потрібна окрема...
<code>n1 -= min(n1, n4)</code>	(3) ...і кожен обрізок можна взяти як дощечку 1×1 . Тому <i>подальша</i> потреба в дощечках 1×1 складає вже не $n1$, а або $n1 - n4$, або 0.
<code>res += n3</code>	(4) На кожную дощечку розмірами 3×1 теж неминуче потрібна окрема дощечка...
<code>if n2 > n3: n2 -= n3</code>	(5) ...причому, при $N_2 > N_3$ використовуємо кожен обрізок як дощечку 2×1 ...
<code>else: n3 -= n2 n2 = 0 n1 -= min(n3*2, n1)</code>	(6) ...а при $N_2 \leq N_3$ формуємо з обрізків <i>усі</i> дощечки 2×1 , а з решти ще й $\min(n3*2, n1)$ дощечок 1×1 (доки не закінчатся дощечки 3×1 або потреба у дощечках 1×1).
<code>res += n2//2 n1 -= min(n1, n2//2) n2 %= 2</code>	(7) Оскільки всі дощечки 3×1 вже сформовані, тепер на кожную пару дощечок 2×1 потрібна окрема дощечка, причому обрізок можна використати як дощечку 1×1 .
<code>if n2==1: res += 1 n1 -= min(n1, 3) n2 = 0</code>	(8) Якщо на попередньому кроці кількість дощечок 2×1 була непарна, то зараз треба сформувати останню дощечку 2×1 , причому з обрізку можна зробити до 3 дощечок 1×1 .
<code>res += n1//5 n1 %= 5</code>	(9) Якщо після усіх попередніх кроків усе ще є потреба в дощечках 1×1 , формуємо їх, розрізаючи кожную дощечку на 5 частин.
<code>if n1 > 0: res += 1</code>	(10) Якщо все ще є потреба у 1×1 , вона ≤ 4 штук, достатньо ще <i>однієї</i> дощечки.

Рис. 2. Алгоритм розв'язання задачі «Паркет»

Здати потрібно одну програму, а не окремі для трьох випадків; різні обмеження вводяться виключно для того, щоб дати приблизне уявлення, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі. Задача передбачає *перебір* — проби різних значень, з перевіркою, чи виконується рівність. Треба лише організувати це правильно.

Один з класичних способів уникати подвійних урахувань розкладень, у яких значення x та y лише обміняні місцями — враховувати лише ті, де $x \leq y$.

Деякі учасники здавали реалізацію, наведену праворуч. Вона і працює надто довго, і може знаходити зайві розкладення. На стор. 13 роз'яснено, чому такий ($O(N^2)$) алгоритм очевидно не вкладається у 1 сек. А на стор. 13 — що таке переповнення, внаслідок яких $x*x+y*y=N$ може виявитися true при, наприклад, $x=1$, $y=65537$, $N=131074$.

```
res:=0;
for x:=1 to N do
  for y:=1 to N do
    if x<=y then
      if x*x+y*y=N then
        res:=res+1;
```

Через помилку автора, задача вийшла простішою, ніж планувалося. Повний бал набирає в т. ч. й «вилізана» у деталях програма, де принципова оптимізація лише одна: верхні межі циклів змінені з N на \sqrt{N} .

Все ж розглянемо можливу подальшу оптимізацію, яка зменшує складність з $O(\sqrt{N} \times \sqrt{N}) = O(N)$ до $O(\sqrt{N})$. Коли x вибраний, рівність $x^2 + y^2 = N$ не може виконатися для різних натуральних y . Тож вкладеного циклу по y можна позбутися, замінивши на перевірку, чи $\sqrt{N-x^2}$ ціле та $\geq x$.

```
res:=0;
sqrtN:=round(sqrt(N));
for x:=1 to sqrtN do
  for y:=x to sqrtN do
    if x*x+y*y=N then
      res:=res+1;
```

Перевірку, чи \sqrt{K} цілий, можна робити як $\text{frac}(\text{sqrt}(K))=0$. Або як $\text{sqr}(\text{round}(\text{sqrt}(K)))=K$. Теоретично, у 1-му виразі можливий підвох, якщо sqrt поверне значення з похибкою і frac верне не 0, де насправді 0, а у другому — якщо round матиме вужчий тип і виникне переповнення. Практично вони обидва працюють правильно.

Задача D. «Дільники»

Вхідні дані: Клавіатура (stdin) Обмеження часу: 2 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Для натурального числа N , виведіть у порядку зростання всі його різні натуральні дільники.

Вхідні дані. Вхідні дані слід прочитати зі стандартного входу (клавіатури). Це буде єдине натуральне число N . $1 \leq N \leq 1234567891011$.

Результати. Результат — послідовність усіх різних натуральних дільників, у порядку зростання — слід вивести на стандартний вихід (екран). Виводити обов'язково в один рядок, розділяючи пробілами.

Приклади:

Вхідні дані	Результати
9	1 3 9
120	1 2 3 4 5 6 8 10 12 15 20 24 30 40 60 120

120 балів (з 250) припадатиме на тести, в яких $1 \leq N \leq 4321$.

Решта 130 балів — на тести, в яких $12345678 \leq N \leq 1234567891011$.

Здати потрібно одну програму, а не окремі для двох випадків; різні обмеження вводяться виключно для того, щоб дати приблизне уявлення, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі. Перш за все, слід зрозуміти, що основна проблема алгоритма `for i:=1 to n do if n mod i = 0 then write(i, ' ')` — він не має ніяких шансів устигнути за потрібний час (див. також стор. 13).

Є сенс перебирати дільники лише до $\text{sqrt}(N)$. Дільники, більші за \sqrt{N} , можна обчислити діленням N на якийсь із дільників, менших \sqrt{N} . Адже: (1) якщо a — дільник N , то N/a — ціле, й також дільник N ; (2) числа a та N/a не можуть одночасно бути більшими \sqrt{N} .

Можна один раз прокрутити цикл від 1 до \sqrt{N} , і повиводити всі знайдені дільники, а потім наступний (не вкладений, а наступний) цикл від \sqrt{N} downto 1 і повиводити $N \text{ div } i$. Такий алгоритм матиме складність $O(\sqrt{N})$, що очевидно поміститься у обмеження часу ($\sqrt{1234567891011} \approx 1,1 \cdot 10^6$). Можна трохи оптимізувати програму, якщо у першому циклі не лише виводити дільники, а ще й

запам'ятовувати їх у масив, щоб другий цикл перебирав лише дільники, а не всі числа проміжку. Але така оптимізація *не* принципова.

Незалежно від організації другого циклу, слід перевірити, чи правильно програма розбирається з такими випадками: (а) \sqrt{N} цілий і є одним (а не двома) з дільників, як 6 для 36; (б) \sqrt{N} не цілий, але є два дільники близько до \sqrt{N} , як 6 і 7 для 42; (в) дільників, близьких до \sqrt{N} , нема.

Приклад розв'язку — ideone.com/wY3IY0 Варто відзначити такі його моменти: N мусить бути 64-бітовим, але решту величин зручніше лишити 32-бітовими; для масиву `dividers`, розмір мільйон узятий з величезним запасом, насправді досить 7000; але правильно оцінити цю кількість дуже складно, тож якщо не знати, то краще взяти з запасом (але враховуючи обмеження пам'яті).

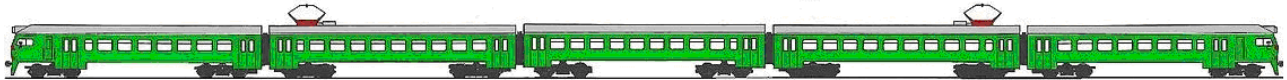
3.4 II (районний/міський) етап 2013/14 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання № 14).

Задача А. «Електричка»

Вхідні дані: Клавіатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

На кожному вагоні електрички є табличка, на якій фарбою написано його номер. Вагони занумеровані натуральними числами 1, 2, ..., N (крайній вагон має номер 1, сусідній з ним — номер 2, і т. д., до крайнього з протилежного боку вагону, який має номер N). Електричка має кабіни з обох боків, і може поїхати хоч 1-им вагоном уперед, хоч N -им.



Під час прибуття електрички на платформу, Вітя помітив, що $(i - 1)$ штук вагонів електрички проїхали мимо нього, а i -й по порядку зупинився якраз навпроти. Ще він помітив, що на табличці цього вагона написаний номер j . Ще він точно знає (і ці знання відповідають дійсності), що електрички ніколи не бувають ні коротшими 4 вагонів, ні довшими 12 вагонів. Вітя хоче визначити, скільки всього вагонів у електричці. Напишіть програму, яка або знаходитиме цю кількість, або повідомлятиме, що без додаткової інформації це зробити неможливо.

Вхідні дані. Програма має прочитати зі стандартного входу (клавіатури) два цілі числа i та j , розділені пропуском. $2 \leq i \leq 12$, $2 \leq j \leq 12$, числа гарантовано задовольняють всі вищезгадані обмеження.

Результати. Виведіть на стандартний вихід (екран) одне число — кількість вагонів у електричці. Якщо однозначно визначити кількість вагонів неможливо, виведіть замість кількості число 0.

Приклад:	Вхідні дані	Результати
	4 2	5

Розбір задачі. Розглянемо два випадки:

- електричка їде 1-м вагоном уперед: тоді 1-й у порядку слідування вагон має напис «№ 1», 2-й у порядку слідування — напис «№ 2», тощо. Тобто, $i = j$ незалежно від кількості вагонів електрички.
- електричка їде N -м вагоном уперед: тоді 1-й у порядку слідування вагон має напис «№ N », 2-й у порядку слідування — напис «№ $(N - 1)$ », тощо. Тобто, $i + j = N + 1$.

Звідси ясно, що при $i \neq j$ точно має місце 2-й випадок, для якого $N = i + j - 1$. Може здатися, ніби при $i = j$ слід виводити 0 («без додаткової інформації визначити неможливо»); але є виключення: при $i = j = 12$, відповідь 12 (електрички не бувають довшими 12 вагонів). А при $(i = j)$ and $(j < 12)$, таки виводити 0, бо така ситуація можлива при будь-якому N у межах $j \leq N \leq 12$. Приклад реалізації: ideone.com/vIUkUt. Програма містить всього кілька дій (складність $\Theta(1)$) і виконується миттєво.

Розв'язки, які виводили правильну відповідь при $i \neq j$, а при $i = j$ — завжди 0, оцінювалися на 180 балів з 200.

Задача В. «Цифрові ріки»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Цифрова ріка — це послідовність чисел, де число, що слідує за числом n , це n плюс сума його цифр. Наприклад, якщо число $n=12345$, то за ним буде йти $12345 + (1+2+3+4+5) = 12360$ і т. д. Якщо перше число цифрової річки N , ми будемо називати її «**річка N** ».

Для прикладу, **річка 480** — це послідовність чисел, яка починається з чисел 480, 492, 507, 519, ..., а **річка 483** — послідовність, що починається з 483, 498, 519, ...

Напишіть програму, яка приймає на вхід два цілих значення k ($1 \leq k \leq 16384$) та N ($1 \leq N \leq 10000$), і виводить k -те число річки N .

Приклад:

Вхідні дані	Результати
4 480	519

Розбір задачі. Задача робиться «в лоб», тобто без усяких придумок $k-1$ раз («мінус один», бо треба отримати k -те число з 1-го, а не 0-го) застосовується дія «порахувати й додати суму цифр поточного числа».

Стандартний спосіб рахувати суму цифр числа — у циклі розглядати останню цифру числа (Pascal: $n \bmod 10$; C/C++: $n \% 10$), а потім «відтинати» її (Pascal: $n := n \div 10$; C/C++: $n /= 10$). Ці операції треба робити з «копією» (а не самим поточним числом, щоб воно не втрачалося від «відтинання»); це може бути забезпечено або присвоєнням у додаткову змінну, або передачею у функцію як значення, а не за посиланням. Приклад реалізації — ideone.com/YWcv5B.

Іншим способом є перетворення числа у рядкову величину й подальші звернення до окремих символів-цифр. Деталі сильно залежать від конкретної мови програмування, навіть конкретних бібліотек, тому їх важко пояснити теоретично. Дивіться конкретні розв'язки: ideone.com/5poAvj (мовою C++) та ideone.com/d2F0rb (мовою Pascal із функціями `IntToStr` та `StrToInt` — найбільш лаконічний з усіх наведених).

Протягом перетворень значення числа може перевищити 32767, тому треба забезпечити, щоб тип мав хоча б 32 біти (див. також стор. 13 та стор. 11).

Задача С. «Логічний куб»

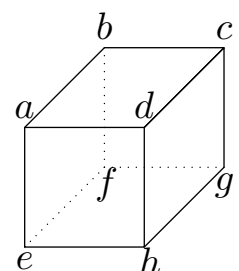
Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Логічний куб — це куб, у вершинах якого знаходяться значення 0 (false) або 1 (true). Потрібно знайти шлях від однієї заданої вершини до іншої; якщо такого шляху не існує, то вивести відповідне повідомлення. В кубі можна проходити через усі ребра, а також через вершини, значення яких рівне 1.

Напишіть програму, яка знаходить шлях між двома вершинами (якщо він існує), та виводить його у вигляді послідовності вершин куба. Гарантовано, що ці дві задані вершини мають значення 1.

Вхідні дані. В першому рядку задаються через пробіл дві вершини куба, це можуть бути дві з наступних маленьких латинських літер: a, b, c, d, e, f, g, h . В наступному рядку, послідовно записуються значення кожної з вершин куба (0 або 1). Значення у вершинах перелічені в алфавітному порядку.

Результати. Якщо шлях існує, то вивести (без пробілів) послідовність маленьких латинських літер мінімальної довжини, які визначають шуканий шлях. Якщо такого шляху не існує, то вивести рядок "NO".



Приклади:

Вхідні дані	Результати
e d 10011011	ead

Вхідні дані	Результати
e d 00011110	NO

Розбір задачі. Задачу можна розв'язати ручним аналізом випадків, але це сумнівний спосіб, доречний, лише коли нема кращих ідей. За посиланням ideone.com/jjq2Pq можна побачити трохи модифікований варіант програми, яку здав один з учасників. Він набирає 270 балів з 300, що з одного боку немало, але з іншого — на аналіз випадків явно пішло багато часу, бали все одно не повні, а шукати помилку в такому нагромадженні — невдячна справа.

Щоб уникнути такої ситуації, краще реалізувати осмислений алгоритм, який обробляє різні вхідні дані більш-менш однотипно.

Раз йдеться про мінімальні (за кількістю переходів) шляхи, природним є *пошук у шир* (він же *пошук у ширину*, англ. *breadth first search, BFS*). Застосувати BFS треба до неорієнтованого графа, вершини якого — вершини куба зі значенням 1 (по яким можна проходити), ребра — ті ребра куба, що поєднують такі вершини. Причому, раз питають не відстань (число), а шлях (послідовність вершин), потрібен варіант BFS, у якому запам'ятовують вершини-попередники та роблять відновлення шляху *зворотнім ходом*. Деталі знайдіть у літературі або Інтернеті.

Інший можливий спосіб — перебір (наприклад, рекурсивний) усіх можливих шляхів, що не містять повторень вершин. Приклад такого розв'язку — ideone.com/pihZzm. Детальніше про це можна прочитати в Інтернеті або літературі за назвами *пошук з поверненнями, бектрекінг* (рос. *поиск с возвратом*, англ. *back-tracking*). Взагалі кажучи, цим не дуже складним способом теоретично можна розв'язати дуже багато задач, але він часто працює надто довго, не вкладаючись в обмеження часу. А тут вкладається, бо вершин всього 8.

На жаль, ідея «писати осмислений алгоритм, щоб обробляти різні вхідні дані однотипно» мало придатна до заданої рисунком відповідності ребер куба його вершинам. Важко сформулювати, між якими саме вершинами є ребра. (Найкраще, що вдалося — «*Позначки вершин відрізняються або на 4, або на 1, але крім $d \leftrightarrow e$, а крім названих, є ще $a \leftrightarrow d$ і $e \leftrightarrow h$* ».) Мабуть, легше задати явний перелік (чи як у розв'язку з попереднього абзацу, чи вписати у текст програми константний масив — матрицю суміжності графа, чи ще якось). Але це треба ретельно звіряти з умовою, бо успішна побудова шляхів у одній частині графа ніяк не перевіряє правильність задання ребер у іншій частині.

Прості способи набрати частину балів. Тести цієї задачі такі, що програма, яка, не вирішуючи задачу по суті, завжди виводить “NO”, набирає 90 балів з 300. Можливо, це й несправедливо багато. Але з умови (та з того, що у 2013 р. використовувалося лише потестове оцінювання) очевидно, що така програма мусила хоч щось та набрати; хто не міг розв'язати правильно — мав би пошукати якісь такі варіанти. Трохи чесніший спосіб — перевіряти, чи є вершини кінцями одного ребра, і якщо так, то виводити ці вершини, а якщо ні, то “NO”. Він набрав 150 балів (рівно половину).

Задача D. «Всюдисущі чісла»

Вхідні дані: Клавіатура (stdin) Обмеження часу: 3 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Дано прямокутну таблицю $N \times M$ чисел. Гарантовано, що у кожному окремо взятому рядку всі чісла різні й монотонно зростають.

Напишіть програму, яка шукатиме перелік (також у порядку зростання) всіх тих чисел, які зустрічаються в усіх N рядках.

Вхідні дані. Вхідні дані слід прочитати зі стандартного входу (клавіатури). У першому рядку задано два чісла N та M . Далі йдуть N рядків, кожен з яких містить рівно M розділених пропусками чисел (гарантовано у порядку зростання).

Результати. Результати виведіть на стандартний вихід (екран). Програма має вивести в один рядок через пробіли у порядку зростання всі ті чісла, які зустрілися абсолютно в усіх рядках. Кількість чисел виводити не треба. Після виведення всіх чисел потрібно зробити одне переведення рядка. Якщо нема жодного числа, що зустрілося в усіх рядках, виведення повинно не містити жодного видимого символу, але містити переведення рядка.

Приклад:

Вхідні дані	Результати
4 5	8 13
6 8 10 13 19	
8 9 13 16 19	
6 8 12 13 15	
3 8 13 17 19	

20% балів припадатиме на тести, в яких $3 \leq N, M \leq 20$, значення чисел від 0 до 100. Ще 20% — на тести, в яких $3 \leq N, M \leq 20$, значення чисел від -10^9 до $+10^9$. Ще 20% — на тести, в яких $1000 \leq N, M \leq 1234$, значення від 0 до 12345. Решта 40% — на тести, в яких $1000 \leq N, M \leq 1234$, значення від -10^9 до $+10^9$. Здавати потрібно одну програму, а не чотири; різні обмеження вказані, щоб пояснити, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі. Очевидний підхід — брати кожне число 1-го рядка, і шукати його в усіх інших рядках. Якщо знайдене в усіх — поточне число слід включити у відповідь (а якщо ні, то ні). І цей підхід може бути правильним. Але залежно від того, чи реалізувати його наївно, чи ефективно.

Наївно — для пошуку кожного числа в кожному рядку запускати свій цикл, переглядаючи увесь рядок. Такий розв'язок потребує $O(N \cdot M^2)$ часу, що може не вкладатися в обмеження. Власне, $1234^3 \approx 1,88 \cdot 10^9$ простих порівнянь на потужному сервері могло б і поміститися у 3 сек. Але в тому й смисл задачі, щоб реалізувати щось ефективніше; на потужнішому сервері або ставили б жорсткіший time limit, або давали вхідні дані ще більших розмірів. На тому сервері, що був у 2013 році, такий розв'язок благополучно отримував свої 120 балів з 300 (якщо вчасно робити break — трохи більше).

Правильний розв'язок № 1. Той самий підхід можна реалізувати ефективніше завдяки тому, що кожен рядок гарантовано впорядкований. Адже для впорядкованих масивів можливий *бінарний пошук* (скорочено *бінпошук*, він же *двійковий пошук*, він же *дихотомія*). Суть бінпошуку: щоб знайти значення, починають з середнього (за індексом) елемента; якщо раптом він якраз рівний шуканому значенню, пошук успішно завершений; якщо шукане значення більше за середній елемент, то можна відкинути усю ліву половину масиву, а якщо менше — усю праву половину. На наступному кроці (якщо він взагалі потрібен) робиться те саме, але з половиною, що залишилась. І так далі. Тобто, за 1–2 порівняння можна зменшити діапазон пошуку щонайменше вдвічі, і пошук у масиві розміром 1234 потребує до ≈ 20 порівнянь (асимптотично — $O(\log M)$). А $N \cdot M$ штук таких пошуків поміщаються в обмеження.

Рекомендується знайти в Інтернеті або літературі додаткові деталі щодо бінпошуку. Бо це такий алгоритм, у якому, навіть добре знаючи загальну ідею, легко помилитися й отримати код, який часто працює правильно, але іноді зациклюється або видає неправильні результати (див., зокрема, стор. 65). До речі, на цьому ж сайті ejudge.skipo.edu.ua є змагання 53 «Дорішування теми “Бінарний та тернарний пошуки” Школи Бобра (23.10.2016)», де є і теоретичні матеріали, і комплект задач.

Тим, хто пише мовою C++, рекомендується вивчити, як у деяких ситуаціях (включно з цією задачею) можна не писати бінпошук самому, якщо навчитися правильно користуватися функцією `lower_bound` бібліотеки `algorithm`.

Правильний розв'язок № 2. Ще один правильний розв'язок (із *кращою* асимптотичною оцінкою $O(N \cdot M)$ проти $O(N \cdot M \cdot \log M)$ у попереднього; але чітко розрізнати цю відмінність за часом роботи програми не дуже реально, тому попередній розв'язок теж вважається ефективним) — багаторазово застосовувати модифікацію *злиття* (рос. *слияние*, англ. *merge*).

Суть стандартного злиття така. Нехай є дві (*обов'язково впорядковані!*) послідовності (звичай масиви або фрагменти одного масиву, але можуть бути й інші, як-то файли чи зв'язні списки). З них можна легко й швидко сформувати впорядковану послідовність, куди входять усі елементи обох заданих послідовностей. Призначаємо кожній зі вхідних послідовностей поточну позицію як початок цієї послідовності; повторюємо у циклі такі дії: (1) беремо (пишемо у відповідь) менший з поточних елементів (якщо елементи рівні, то беремо, наприклад, поточний елемент першої послідовності); (2) зсуваємо поточну позицію тієї вхідної послідовності (*лише однієї з двох!*), звідки взятий цей елемент. Коли одна з послідовностей закінчується, дописуємо у послідовність-відповідь увесь ще не використаний «хвіст» іншої. Це — *стандартне* злиття, що формує послідовність, яка містить усі елементи обох послідовностей. Очевидно, воно працює за час $O(l_1 + l_2)$, де l_1 та l_2 — довжини вхідних послідовностей.

У задачі треба інше (спільні елементи). Але злиття легко модифікувати під це. Треба при порівнянні поточних елементів двох послідовностей розрізнати три випадки: якщо поточний елемент

1-ої послідовності менший за поточний елемент 2-ої, то зсунути поточну позицію 1-ої (нічого не пишучи у відповідь); якщо більший, то зсунути позицію 2-ої (теж не пишучи); якщо поточні елементи рівні, то записати це однакове (спільне) значення у результат та зсунути поточні позиції обох послідовностей. При завершенні однієї з послідовностей, не дописувати «хвіст» іншої. Час роботи цього модифікату злиття теж $O(l_1 + l_2)$.

Зрештою, це — злиття двох послідовностей, а треба N . Пропонується першого разу злити 1-ий рядок з 2-им, а потім зливати з кожним черговим (3-ім, 4-им, ...) результат попереднього злиття. Завдяки тому, що завжди вибираються лише спільні елементи, розмір кожної з послідовностей $\leq M$, тому $N-1$ застосувань злиття займуть сумарний час $O(N \cdot M)$. (Якби робилося стандартне злиття і розміри зростали, оцінка була б значно більшою.)

Знайдіть додаткову інформацію про злиття. Її багато, але про злиття часто пишуть як про складову сортування, а тут потрібне саме злиття, без рекурсивної надбудови сортування. Крім того, не всі алгоритми, які правильно виконують стандартне злиття, легко модифікуються на вибір лише спільних елементів. До речі, на цьому ж сайті ejudge.skipo.edu.ua є змагання 61 «День Іллі Порубльова “Школи Бобра” (15.10.2017, злиття та два вказівники)», в якому є і теоретичні матеріали, і комплект задач.

Користувачам C++ можна також ознайомитися з готовою потрібною модифікацією злиття — функцією `set_intersection` бібліотеки `algorithm`.

Таким чином, грамотні користувачі C++ в обох наведених способах розв'язання у виграші, бо можуть значну частину потрібного алгоритму не писати самостійно, а використати бібліотечні засоби. Водночас, малодосвідчені користувачі C++ у програші, бо можуть і не знати, як вирішити проблему, що читання `cin`-ом не вкладається у обмеження часу. (Як? Див. стор. 15.)

Навіщо в умові згадані групи тестів зі значеннями до 12345? Щоб дати можливість набрати ще $\approx 20\%$ балів тим, хто не подумався ні до одного з правильних способів, але знає наступний, складність $\Theta(N \cdot M + V)$, де V — розмір діапазону від мінімуму до максимуму можливих значень.

Мається на увазі, що програму в цілому можна організувати як «if (розміри малі) then (вирішити способом згаданим на самому початку розбору) else (вирішити описаним далі способом)». Вхідні дані, коли великі одночасно і кількості, і значення, такий розв'язок не пройде, але, можливо, набере більше балів.

Так от, для малих значень ефективний такий підхід. Зведемо масив, *індексами* якого будуть значення зі вхідних даних, щоб щоразу, прочитавши деяке v , збільшувати `push[v]` на 1 (спочатку всі `push[·]` ініціалізуються нулями). Таким чином, після обробки усіх вхідних даних кожне значення `push[v]` означатиме, скільки разів зустрілося число v ; оскільки у кожному окремо взятому рядку всі числа різні, то `push[v] = N` рівносильно « v зустрілося в усіх рядках».

3.5 Дистанційний тур III (обласного) етапу 2013/14 н. р.

У 2013/14 навч. році III (обласний) етап у Черкаській області складався з двох турів, де один був частково дистанційним (учасники приїздили не до м. Черкаси, а до своїх райцентрів) і проводився на задачах черкаських авторів. Саме він і наведений у цьому збірнику. Цей тур позиціонувався одночасно і як змагальний, і як кваліфікаційний; тому рівень складності комплекта задач дещо нижчий, ніж зазвичай на III етапі.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання № 15).

2-й тур відбувався у Черкасах, але на задачах інших авторів, іншій системі ([ejudge](http://ejudge.skipo.edu.ua), але не ejudge.skipo.edu.ua), та й про його дорішування авторам цього збірника нічого не відомо. Тому, він до збірника не включений.

Задача А. «ISBN»

Вхідні дані: Клавіатура (`stdin`)

Обмеження часу: 1 сек

Результати: Екран (`stdout`)

Обмеження пам'яті: 64 мегабайти

ISBN (з англ. International Standard Book Number — міжнародний стандартний номер книги) універсальний ідентифікаційний номер, що присвоюється книзі або брошурі з метою їх класифікації.

ISBN призначений для ідентифікації окремих книг або різних видань та є унікальним для кожного видання книги. Даний номер містить десять цифр, перші дев'ять з яких ідентифікують книгу, а остання цифра використовується для перевірки коректності всього номеру ISBN. Для перевірки ISBN обчислюється сума добутків цифр на їхній номер, нумерація при цьому починається з крайньої правої цифри. В результаті має бути отримано число, що без остачі ділиться на 11.

Наприклад: **0201103311** — коректний номер, тому що $0 \times 10 + 2 \times 9 + 0 \times 8 + 1 \times 7 + 1 \times 6 + 0 \times 5 + 3 \times 4 + 3 \times 3 + 1 \times 2 + 1 \times 1 = 55$, що націло ділиться на 11.

Кожна з перших дев'яти цифр може приймати значення від 0 до 9.

Напишіть програму, що читає ISBN код з однією пропущеною цифрою (вона буде позначатись як символ « » (пробіл)) і виводить значення пропущеної цифри.

Приклад:

Вхідні дані	Результати
020110 311	3

Примітка. У справжніх ISBN-номерах в якості останньої цифри може бути також велика латинська X, що позначає 10. Але в цій задачі таких номерів гарантовано не буде.

Розбір задачі. В принципі, можна вивести аналітичну формулу, але це потребує знань з теорії чисел (охочі можуть знайти, що таке *кільце залишків за модулем* та *мала теорема Ферма*, і застосувати до цієї задачі). Все це було би доцільним, *якби* довжина ISBN-коду становила, наприклад, сотні тисяч, так що розглянутий далі простіший і значно більш «програмістський» (а не «математичний») підхід працював би надто довго.

А для 10 цифр підходить і значно простіший перебір, тобто перепробувати усі варіанти від 0 до 9 і для кожного подивитися, чи виконується описана в умові правильність ISBN. Приклад реалізації див. ideone.com/g3SbMb. Правда, при різних правильних відповідях ця програма вивела б усі, хоча на олімпіадах треба виводити будь-яку одну. Насправді це неактуально, бо різні правильні відповіді неможливі (хто вивчав питання, згадані у попередньому абзаці, можуть це довести; решта можуть або повірити, або дописати у розв'язок break для обривання циклу після виведення першої відповіді.)

Говорити про асимптотичну складність некоректно, бо нема того розміру вхідних даних, який міг би прямувати до ∞ . *Якби* довжина ISBN-коду становила довільне N (але таке, щоб $N+1$ було простим, і щоб аналогічна сума добутків ділилася без остачі на $N+1$), можна було би сказати, що перебір має складність $\Theta(N^2)$, а теоретико-числовий розв'язок — $\Theta(N + \log N) = \Theta(N)$.

Ще може бути проблемою формат вхідних даних, особливо для молодосвідчених користувачів C++: `cin>>a>>b` ніби читає в `a` те, що до пропуску, й у `b` те, що після; але в обох випадках «пробіл замінює найпершу цифру» та «... найостаннішу ...» всі 9 осмислених цифр ідуть в `a`. Може спасти на думку читати у циклі окремі `char`-и; але при стандартних налаштуваннях `cin>>c` (`c` — типу `char`) взагалі пропускає пропуски, і взнати його позицію неможливо. Один зі способів вирішення цієї проблеми — перед читанням `char`-ів викликати `cin.unsetf(ios::skipws)`, щоб пропуски та переведення рядка таки читались у ті `char`-и. Інший — використати функцію `getline(cin,s)` (`s` — типу `string`); вона (як і `readln` мови Pascal) читає все, включаючи пробіли, до кінця рядка.

Задача В. «Точні квадрати»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Напишіть програму, яка знаходитиме кількість натуральних чисел із проміжку $[a; b]$, які задовольняють одночасно двом таким вимогам:

- число є точним квадратом, тобто корінь з нього цілий (наприклад, точними квадратами є $1=1^2$, $9=3^2$, $1024=32^2$; а 8, 17, 1000 не є точними квадратами).
- сума цифр цього числа кратна K (наприклад, сума цифр числа 16 рівна $1+6=7$).

Програма повинна прочитати три числа в одному рядку a b K і вивести одне число — кількість чисел, які задовольняють умовам.

Оцінювання. В усіх тестах виконується $1 \leq a \leq b \leq 2 \cdot 10^9$, $2 \leq K \leq 42$.

40% балів припадає на тести, в яких виконується $1 \leq a \leq b \leq 30\,000$, $K=9$.

Приклад:	Вхідні дані	Результати
	7 222 9	4

Примітка. Цими чотирма числами є 9, 36, 81, 144.

Розбір задачі. Ніби нескладна задача — перебрати числа і перевірити кожне на відповідність умовам. Причому, «чи є точним квадратом» вже розглядали на стор. 30, суму цифр — на стор. 32... Але програма `ideone.com/gLqsYd` набирає лише 50 балів зі 100.

Перевіряти аж 2 млрд чисел — забагато. Навіть якщо (абсолютно слушно) рахувати суму цифр лише для тих, які пройшли перевірку «чи є точним квадратом». Пришвидшити розв'язок досить просто — *генерувати лише точні квадрати*, а не перебирати й перевіряти геть усі числа проміжку. Достатньо запускати цикл не від a до b , а від $\approx\sqrt{a}$ до $\approx\sqrt{b}$ й працювати з i^2 . Правда, тут можна заплутатися, як перетворити « $\approx\sqrt{a}$ » та « $\approx\sqrt{b}$ » у точні цілі значення. Не дуже красивий, зате точно правильний спосіб — узяти межі з невеличким «запасом», а потім отримане i^2 все-таки перевірити на належність проміжку $[a; b]$. Такий розв'язок (наприклад, `ideone.com/Ep1T1L`), навіть із цією зайвою перевіркою, вклаватиметься у 1 сек з великим запасом, бо тепер кількість ітерацій $\leq \sqrt{2 \cdot 10^9} \approx 45$ тис.

З асимптотичною оцінкою цього розв'язку є неоднозначність. Попередній абзац нашоує на $\Theta(\sqrt{b} - \sqrt{a})$ або $O(\sqrt{b})$; але якщо враховувати ще цикл у `sumOfDigits`, то вийде $O(\sqrt{b} \cdot \log b)$.

Задача С. «Ложбан»

Вхідні дані: Клавіатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Ложбан (англ. *Lojban*) — це штучна мова, яка була створена у 1987 році Групою Логічних Мов та базується на логлані (логічна мова). При створенні основною ціллю була повніша, вільно доступна, зручна для використання мова. Вона є експериментальною і сконструйована для перевірки гіпотези Сепіра-Ворфа. Ця гіпотеза припускає, що люди, які говорять різними мовами, по-різному сприймають світ і по-різному мислять.

Дана мова є однією з найпростіших штучних мов. Наприклад цифри від 0 до 9 записуються наступним чином:

1	pa	4	vo	7	ze	
2	re	5	mu	8	bi	0 no
3	ci	6	xa	9	so	

Великі числа утворюються склеюванням цифр разом. Наприклад, число 123 це *pareci*.

Завдання. Напишіть програму, яка зчитує рядок на Ложбані (що представляє собою число $\leq 1\,000\,000$) та виводить цей рядок у цифровому вигляді.

Приклад:	Вхідні дані	Результати
	reporavo	2014

Розбір задачі. По-перше, слід знайти в умові справді потрібну частину: «Є текст, який гарантовано отриманий таким чином: взяли число $\leq 1\,000\,000$, й замінили кожну цифру на дві букви згідно з наведеною табличкою. Провести зворотнє перетворення цього тексту у число.».

По-друге, все могло би бути вельми складним, якби траплялися ситуації, коли одне зі слів — початок іншого (як-то «7 позначається як *mis*, 8 — як *misiv*»). Тому варто відмовитися від ідеї писати універсальну програму, яка могла би працювати з різними позначеннями цифр, і ретельно дослідити, якими конкретними, заданими в умові, словами кодуються цифри. І побачити, що тут не лише нема такої ситуації, а ще й усі ці слова дволітерні.

Задача насправді досить проста. Треба лише зуміти прочитати це у громіздкій умові. Наприклад, див. `ideone.com/BNuDL`. Або, використавши, що вхідні дані *гарантовано* являють собою якесь закодоване число, можна написати щось іще простіше — наприклад, див. `ideone.com/0dw21F`.

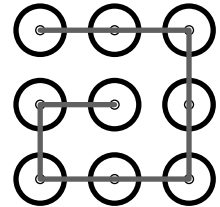
Така умова в принципі могла б містити підвох: ніде не сказано прямо, чи числа натуральні. А раптом від'ємні? А раптом взагалі десяткові дробі? А якщо від'ємні, то скільки може бути цифр,

адже $(-10^{100500}) < 10^6$? За правилами переважної більшості олімпіад з інформатики, учасник має право задати питання журі щодо таких моментів умови. Або, враховуючи можливість багатократної здачі, пробувати варіанти... Конкретно в цій задачі підвоху не було (числа цілі невід'ємні).

Задача D. «Графічний пароль»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 4 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Програміст Василь недавно придбав собі новий смартфон і встановив на ньому графічний пароль. Графічний пароль представляє собою ламану лінію, яка проходить через вершини сітки розміром 3×3 (не обов'язково через всі).



Але згодом Василь зрозумів, що пароль на сітці 3×3 не є досить безпечним, і вирішив виправити цю проблему. Для цього він збільшив розмір сітки до 5000×5000 , але при цьому наклав обмеження на відрізки ламаної — тепер вони можуть бути лише горизонтальними та вертикальними. Василь ще хотів додати функцію, яка б виводила кількість самоперетинів у графічному паролі, але так склалося, що він не досить знайомий з ефективними алгоритмами, тому він просить вашої допомоги.

Вхідні дані. Перший рядок містить ціле число N — кількість вершин в ламаній лінії ($2 \leq N \leq 1\,000\,000$). Кожен з наступних N рядків містить два цілих числа x та y — координати відповідної вершини ламаної ($0 \leq x, y < 5000$). Гарантується, що ламана у вхідних даних містить лише горизонтальні та вертикальні відрізки, які можуть перетинатися, але не можуть накладатися один на одного. Ніякі дві вершини ламаної (в т. ч. старт та фініш) не знаходяться в одній і тій самій точці.

Результати. Необхідно вивести єдине ціле число — кількість самоперетинів у ламаній.

Приклади:

Вхідні дані	Результати
6 1 1 0 1 0 0 2 0 2 2 0 2	0

Вхідні дані	Результати
5 0 1 3 1 3 2 2 2 2 0	1

Розбір задачі. Головне при розв'язуванні цієї задачі — не перестаратися й не почати писати «чесні» перевірки перетинів. Такі перевірки могли б бути доречними при більших розмірах сітки та меншій кількості відрізків ламаної. А при заданій кількості, відомі автору цього розбору оптимізації перебору всіх можливих перетинів не вкладаються у обмеження часу.

У цій задачі краще помітити такі факти:

- Хоча окремо взятий відрізок (ланка ламаної) може мати довжину аж 5000, а кількість відрізків може сягати мільйона, сумарна довжина відрізків обмежується не $5000 \times 10^6 = 5 \cdot 10^9$, а тим, що раз відрізки лише горизонтальні й вертикальні, а накладання заборонені, то кожна з 5000×5000 вершин сітки може бути задіяна щонайбільше двічі, і сума довжин усіх відрізків < 50 млн.
- $5000 \times 5000 = 25$ млн — дуже багато для людини, але для комп'ютера — не так і багато. Не лише у розрізі «виконати 25 млн дій», а також і у розрізі «тримати в пам'яті 25 млн елементів». (Але, щоб укластися в обмеження пам'яті, вони мусять бути 1- чи 2-байтовими!)

Завдяки цьому, найдоречнішим (досить ефективним для саме цих обмежень і при цьому значно простішим за альтернативи) виявляється такий простий підхід: «завести масив 5000×5000 , ініціалізувавши всі елементи нулями; ходити уздовж ліній, збільшуючи на 1 значення у комірках, відповідних пройденим вершинам сітки; відповіддю буде кількість комірок зі значенням 2». Великий time limit 4 сек потрібен не стільки для роботи алгоритму, скільки для читання величезних вхідних даних. Приклад реалізації цього алгоритма — ideone.com/S1R8oV.

3.6 Обласна інтернет-олімпіада 2014/15 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання № 18).

Задача А. «Три круги»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Є три круги радіусами R_1 , R_2 та R_3 .

Чи можна перекласти їх так, щоб відразу два менші круги лежали один поруч з іншим на найбільшому, не накладаючись один на одного і не звисаючи за його межі? Торкатися один одного менші круги можуть.

Вхідні дані. Програма повинна прочитати три цілі числа R_1 , R_2 та R_3 , в один рядок, через пропуски (пробіли). Усі три значення R_1 , R_2 та R_3 є цілими числами в межах від 1 до 1000.

Результати. Якщо перекласти круги вказаним чином неможливо, програма повинна вивести єдине слово "NO" (без лапок).

Якщо можливо, то в єдиному рядку повинно бути записано "YES, the ... disk is the maximal" (без лапок), де замість "..." повинно бути одне з трьох значень:

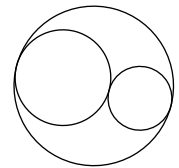
- "1st" (без лапок), якщо 2-й і 3-й круги можна покласти поверх 1-го;
- "2nd" (без лапок), якщо 1-й і 3-й круги можна покласти поверх 2-го;
- "3rd" (без лапок), якщо 1-й і 2-й круги можна покласти поверх 3-го.

Зверніть увагу: фраза повинна бути однаковою з правильною байт-у-байт, тобто всі великі чи маленькі літери, всі пропуски (пробіли) та інші подібні дрібниці важливі.

Приклади:

Вхідні дані	Результати
1 2 3	YES, the 3rd disk is the maximal
2 3 4	NO
9 3 1	YES, the 1st disk is the maximal

Розбір задачі. Два менші диски, яким заборонено накладатися, займають найменше місця, коли торкаються. Тому гранична ситуація, коли при хоч трохи меншому найбільшому диску вони вже звисають, а при рівно такому або більшому все гаразд, зображена на рисунку. Наприклад, 2-й і 3-й круги можна покласти поверх 1-го тоді й тільки тоді, коли $R_1 \geq R_2 + R_3$. Решта випадків аналогічні. Щоб правильно виводити "NO", легше не формулювати умову цього випадку, а зробити розгалуження вкладеними, щоб виводити "NO", коли не виконалася жодна з трьох інших умов. Реалізацію див. ideone.com/rx20dn.



Можливий інший підхід — спочатку знайти, який з дисків максимальний, а вже потім провести одне порівняння. Для даної задачі це погана ідея: і тому, що треба виводити номер максимального диску (ще й у вигляді "1st"/"2nd"/"3rd"), і тому, що з'ясування, який диск максимальний, займе чи не більше дій, ніж розгляд випадків у попередньому розв'язку. Але для багатьох інших задач деякий аналог другого підходу буває набагато кращим за аналіз випадків, аналогічний першому підходу.

Задача В. «Перевезення вантажу»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Перевізник транспортує вантаж залізницею, а потім річковим транспортом.

Товар транспортується в потязі, який складається з N ($1 \leq N \leq 100$) вагонів, в кожному з яких k_1, \dots, k_N ($1 \leq k_i \leq 1000$) коробок вантажу.

У річковому порті вантаж з потягу перевантажують на корабель, який може взяти не більше ніж P ($1 \leq P \leq 10000$) коробок. Якщо якісь коробки не помістяться на цей корабель, вони дуже довго чекатимуть наступного. Причому чекатимуть обов'язково у тих вагонах, в яких прибули у порт.

Для ефективного транспортування диспетчеру необхідно розвантажити якомога більше вагонів, завантаживши корабель, та відправити звільнені вагони на наступне завантаження.

Завдання. Напишіть програму `transport`, яка визначала б максимальну кількість вагонів, які можна розвантажити, не перевищуючи вантажопідйомність корабля.

Вхідні дані. 1-й рядок: єдине число N — кількість вагонів в потязі.

2-й рядок: $k_1 \dots k_N$ через пропуски (пробіли) — кількості коробок у вагонах.

3-й рядок: P — кількість коробок, яку може взяти на борт корабель.

Результати. Максимальна кількість вагонів потягу, які вдасться розвантажити.

	Вхідні дані	Результати
Приклад:	3	2
	5 7 3	
	9	

Розбір задачі. Ця задача, хоч і нескладна, потребує і з'ясувати правило, за яким слід вибирати вагони, і написати не зовсім елементарну програму.

Тож правило — *«Щоразу вибирати (ще не вибраний) вагон з мінімальною кількістю коробок, доки не вичерпається вантажопідйомність корабля або не будуть задіяні всі вагони»*.

Доведемо (див. також стор. 12), що це дасть максимальну кількість вагонів. *«Припустимо, ніби замість вагона з мінімальною кількістю коробок взяли вагон з деякою більшою кількістю. Це збільшило кількість розвантажених вагонів так само на 1, якби взяли вагон з мінімальною, а залишок вантажопідйомності корабля зменшило сильніше. Отже, ніякого виграшу від того, щоб брати вагон з немінімальною кількістю коробок, немає»*. Ця схема доведень типова, тобто її аналоги придатні у багатьох ситуаціях.

Сформульовані правила — ще не зовсім алгоритм. Є мінімум два способи подальшого їх уточнення (як саме *«вибирати (ще не вибраний) вагон з мінімальною кількістю коробок»*?).

Можна застосувати сортування (воно ж упорядкування), після нього це будуть просто елементи по порядку. Особливо зручно, якщо писати мовою, де є готове бібліотечне сортування (наприклад, у `ideone.com/sElRrj` використано функцію `sort` бібліотеки `algorithm` мови C++). Та й якщо писати сортування самому, все ж є деяка зручність у тому, щоб окремо написати та ретельно вивірити правильність сортування, окремо решту дій.

Або, можна багатократно проходити по усьому масиву, вибираючи мінімальний елемент, і після кожного такого проходу враховувати знайдений елемент та замінити його на якесь велике значення, щоб не знаходити повторно. Таким чином, можна обійтися без зсувів чи обмінів елементів, що може бути простішим для початківців. Втім, інших переваг цей спосіб не має. Зокрема, заявка *«він швидший, бо не сортує всі елементи, а вибирає лише стільки, скільки треба»* не витримує критики, бо його складність $\Theta(N \cdot P)$, тобто, враховуючи $P \leq N$, по суті $O(N^2)$. А складність *ефективних* сортувань $O(N \log N)$, що набагато менше.

Задача С. «Коло і точки»

Вхідні дані: Клавіатура (stdin) Обмеження часу: 1 сек

Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Є одне коло та N точок.

Завдання. Напишіть програму, яка знаходитиме, скільки з цих N точок потрапили всередину цього кола, скільки на саме коло і скільки ззовні кола.

Вхідні дані. Перші два рядки вхідних даних задають коло. Можуть бути два різні формати його задання:

1. Якщо 1-ий рядок містить єдине число 1, то 2-ий рядок містить рівно три цілі числа x_C, y_C, R_C — координати центра кола та його радіус.
2. Якщо 1-ий рядок містить єдине число 2, то 2-ий рядок містить рівно шість цілих чисел $x_A, y_A, x_B, y_B, x_C, y_C$. Їх слід трактувати як координати трьох точок A, B, C , через які проведене коло, котре треба дослідити. Ці три точки гарантовано всі різні і гарантовано не лежать на одній прямій. Гарантовано також, що ніяка з цих трьох точок не збігається ні з одною з точок подальшого переліку з N точок.

Третій рядок вхідних даних завжди містить єдине ціле число N — кількість точок. Подальші N рядків містять по два цілі числа кожен — x та y координати самих точок.

Абсолютно всі задані у вхідних даних числа є цілими і не перевищують за абсолютною величиною (модулем) 1000. При цьому кількість точок і радіус гарантовано додатні, а координати можуть бути довільного знаку.

Скрізь, де в одному й тому ж рядку записано по кілька чисел, вони відділені одне від одного пропусками (пробілами).

Результати. Програма повинна вивести три цілі числа — спочатку кількість точок всередині кола, потім кількість точок на самому колі, потім кількість точок ззовні кола.

Сума цих трьох чисел повинна дорівнювати N . Зокрема, якщо коло задане 2-им способом, то ті три точки, якими воно задане, треба не вважати точками, належними колу.

Приклади:

Вхідні дані	Результати	Вхідні дані	Результати
1 -1 2 5 4 0 0 -3 -3 -6 2 6 -2	1 1 2	2 -1 -3 2 6 3 5 4 0 0 -3 -3 -6 2 6 -2	1 1 2

Примітки. В обох наведених прикладах задане (різними способами) одне й те само коло, тому що коло, проведене через точки $(-1; -3)$, $(2; 6)$ та $(3; 5)$ якраз і має радіус 5 та центр у точці $(-1; 2)$.

Перша одиничка відповіді виражає, що лише одна точка з переліку потрапила всередину кола (і це точка $(0; 0)$, але цього не питають). Друга одиничка відповіді виражає, що лише одна точка з переліку потрапила на саме коло (і це точка $(-6; 2)$, але цього не питають). Число 2 у відповіді позначає, що решта дві точки переліку (це точки $(-3; -3)$ та $(6; -2)$, але цього не питають) потрапили ззовні кола.

Можна здавати програму, яка враховує лише подання кола через координати центра і радіус. На тести, в яких коло подається 1-им способом, припадатиме майже половина балів, і така програма може їх отримати. Але навіть така програма повинна враховувати, що в цих тестах все одно буде 1-ий рядок з єдиним числом 1. Разом з тим, якщо враховувати обидва випадки, це все одно повинна бути одна програма.

Розбір задачі. Коло — множина точок, рівновіддалених від центру, тож досліджувана точка потрапляє всередину кола, коли відстань між нею і центром кола менша за радіус, на саме коло — коли рівна, і назовні, коли більша. Ця відстань рівна $\sqrt{(x_i - x_C)^2 + (y_i - y_C)^2}$ (де $(x_i; y_i)$ — координати досліджуваної точки, $(x_C; y_C)$ — центру кола). Тож (для програми, що працює лише для 1-го способу подання кола) лишається тільки написати цикл зі вкладеними розгалуженнями на три випадки. При бажанні, можна позбутися похибок (див. стор. 14–14), прибравши корені, тобто порівнювати $(x_i - x_C)^2 + (y_i - y_C)^2$ з R^2 . Остаточна реалізація — ideone.com/k9iLan.

Розібратися, що робити з 2-м способом подання, *значно* складніше (можливо навіть складніше усієї решти $2^{1/2}$ задач цього туру). Можна звести 2-й випадок до 1-го, тобто перейти від трьох точок до центру й радіусу проведеного через ці три точки кола, та використати вже розглянутий розв'язок.

Один зі способів — будувати перетин серединних перпендикулярів двох сторін $\triangle ABC$, тобто ті ж дії, що й на уроці геометрії, але виконані замість циркуля і лінійки засобами *обчислювальної геометрії* (англ. *computational geometry*). Вступ до обч. геометрії можна знайти у багатьох місцях, зокрема goo.gl/6urrju. Програма, що розв'язує задачу цим способом — ideone.com/k1aFcF (але звідти свідомо прибрані допоміжні функції, пояснені за попереднім посиланням).

Інший спосіб — вивести на папері прямі формули, розв'язавши систему

$$\begin{cases} (x-x_A)^2 + (y-y_A)^2 = (x-x_B)^2 + (y-y_B)^2, \\ (x-x_A)^2 + (y-y_A)^2 = (x-x_C)^2 + (y-y_C)^2, \end{cases}$$

де $x_A, y_A, x_B, y_B, x_C, y_C$ — координати заданих точок (до них треба ставитися як до відомих значень), а x та y — координати центра кола, їх-то й шукаємо. Розв'язати цю систему легше, ніж

може здатися, бо після перетворень, аналогічних $(x-x_A)^2 = x^2 - 2x_A \cdot x + x_A^2$, можна позводити x^2 та y^2 , і рівняння стають лінійними відносно x та y (що логічно, бо це серединні перпендикуляри).

Можна й не виражати 2-ий випадок через 1-ий. Виявляється, відповідь на задачу можна знаходити за знаком детермінанта, наведеного праворуч. (Детермінант, він же *визначник* — стандартний термін теорії матриць. Як обчислювати детермінант та чому цей детермінант має описану далі властивість, охочі можуть знайти самостійно.)

$$\begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_i & y_i & x_i^2 + y_i^2 & 1 \end{vmatrix}$$

Отже: рівність цього детермінанта 0 означає, що $(x_i; y_i)$ лежить на колі, проведеному через $(x_A; y_A)$, $(x_B; y_B)$, $(x_C; y_C)$; додатне значення — всередині; від'ємне — ззовні. Тільки це якщо обхід $\triangle ABC$ (у порядку A, B, C) додатний (проти годинникової стрілки), а при зміні напрямку обходу $\triangle ABC$ слід обміняти місцями смисл додатного і від'ємного знаків детермінанту. Зовсім не очевидно, але ж це *лише один* зі способів розв'язання задачі...

3.7 II (районний/міський) етап 2014/15 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання № 46).

Так склалося, що на ejudge.skiro.edu.ua починаючи саме з цього змагання запроваджене *комбіноване введення-виведення*: більшість задач налаштовані так, що програма учасника може читати вхідні дані хоч з клавіатури, хоч зі вхідного текстового файлу input.txt (але лише з чогось одного, а не поперемінно). Аналогічно, програма учасника може виводити результати хоч на екран, хоч у вихідний текстовий файл output.txt (теж лише на/у щось одне).

Задача А. «Цифра»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

В заданому додатному числі потрібно закреслити одну цифру так, щоб число, яке залишиться в результаті, було найбільшим.

Напишіть програму, яка читає одне ціле значення n ($10 \leq n \leq 99999$), і виводить число без однієї цифри (це число має бути найбільшим серед усіх можливих варіантів закреслень цифри).

Приклади:

Вхідні дані	Результати	Вхідні дані	Результати
321	32	129	29

Розбір задачі. Діяти за принципом «*Викреслювати мінімальну цифру*» *неправильно* (всупереч підступним прикладам з умови, які провокують таку хибну думку.) Наприклад, із числа 9891 треба викреслити 8 і отримати 991, а викреслення мінімальної цифри 1 дасть не максимальне 989.

Оскільки розміри малі (кількість цифр ≤ 5 , видаляється одна), можна просто перебрати всі варіанти викреслення однієї цифри (усе число без 1-ої, усе без 2-ої, тощо), і вибрати з них максимальний. Зручно (хоча й не обов'язково) перевести число у рядок (string), і вилучати символи у рядковому поданні. Реалізацію див. ideone.com/jFBIM6. При бажанні, її можна спростити, роблячи все виключно рядками. (Взагалі буває проблема «у числах $7 < 10$, а у рядках "7" > "10"», але тут вона не проявиться, бо кількості цифр усіх потрібних чисел однакові.)

Якщо мати лише мету розв'язати цю задачу при цих обмеженнях — краще обмежитися першим правильним способом і не читати далі. А якщо мати бажання ще раз розглянути, як можна доводити правильність алгоритма (див. також стор. 12) — тоді подальший текст важливий.

Ще є правильний розв'язок «*Знайти найлівіше місце, де зразу після меншої цифри йде більша, і викреслити меншу саме з цих двох; якщо жодного такого місця нема (наприклад, у числі 97752) — викреслити останню цифру*».

Занумеруємо цифри початкового числа зліва направо $\overline{a_1 a_2 \dots a_n}$. Розглянемо спочатку випадок «нема жодного місця, щоб після меншої цифри йшла більша», тобто $a_1 \geq a_2 \geq \dots \geq a_n$. Згідно

алгоритму, треба викреслити a_n , лишивши $\overline{a_1 a_2 \dots a_{n-1}}$. Якщо всупереч алгоритму викреслити деяку a_j ($1 \leq j < n$), вийде $\overline{a_1 a_2 \dots a_{j-1} a_{j+1} \dots a_n}$. Початок $\overline{a_1 a_2 \dots a_{j-1}}$ спільний, тож результат порівняння визначається частиною, де a_{j+1} замість a_j , a_{j+2} замість a_{j+1} , ..., a_n замість a_{n-1} . Оскільки розглядаємо ситуацію $a_1 \geq a_2 \geq \dots \geq a_n$, то або $a_j > a_{j+1}$, або $a_j = a_{j+1}$. При $a_j > a_{j+1}$, число, отримане всупереч алгоритму, менше (гірше) отриманого згідно з алгоритмом, бо після спільного початку йде $a_{j+1} < a_j$. Якщо ж $a_j = a_{j+1}$, то можна приєднати цю цифру до спільного початку і повторити всі міркування для « a_{j+2} замість a_{j+1} ». І так далі. Кінець кінцем, або десь отримаємо, що число всупереч алгоритму менше (гірше) числа згідно з алгоритмом, або дійдемо до $a_j = a_{j+1} = \dots = a_n$, тобто викреслення a_j призводить до того ж результату, що алгоритм.

Лишилося розглянути випадок, коли місце, де $a_i < a_{i+1}$, існує. Нехай i^* — найлівіша з таких позицій, тобто $a_1 \geq a_2 \geq \dots \geq a_{i^*}$ і $(a_{i^*} < a_{i^*+1})$. Невигідність видаляти замість a_{i^*} деяку a_j при $1 \leq j < i^*$ доводиться аналогічно попередньому абзацу. Лишилося довести невивідність видаляти a_j при $i^* < j \leq n$, а це зовсім легко: згідно з алгоритмом отримуємо $\overline{a_1 a_2 \dots a_{i^*-1} a_{i^*+1} \dots a_n}$, всупереч — $\overline{a_1 a_2 \dots a_{i^*-1} a_{i^*} \dots a_{j-1} a_{j+1} \dots a_n}$, тобто початок $\overline{a_1 a_2 \dots a_{i^*-1}}$ спільний, потім $a_{i^*+1} > a_{i^*}$. Розглянуті випадки покрили всі можливі ситуації, доведення успішно завершено.

Чи має другий алгоритм переваги над першим? При $n \leq 99999$ — ні. Якби числа були значно більшими (наприклад, до мільйона цифр; не « $n \leq 10^6$ », а якби кількість цифр могла сягати мільйона) — тоді виявилось б, що перший алгоритм правильний, але повільний, а другий правильний і ефективний. Асимптотичні оцінки: $\Theta(L^2)$ для першого, $\Theta(L)$ для другого, де L — кількість цифр.

Задача В. «Піраміда»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Гілла Бейтсу захотілося увічнити пам'ять про свою корпорацію MegaHard. Він обрав перевірений часом метод — побудувати піраміду.

Піраміда має висоту n Стандартних Будівельних Блоків, і кожен її рівень — квадрат $k \times k$ блоків, де k — номер рівня, рахуючи згори. З'ясувалося, що фірма, що виготовляє Стандартні Будівельні Блоки, продає їх лише партіями по m штук.

Потрібно написати програму, що визначатиме, скільки блоків залишаться не використаними після побудови піраміди (Гілл Бейтс забезпечить закупівлю мінімально необхідної для побудови піраміди кількості партій).

Напишіть програму, яка читає в один рядок через пропуск (пробіл) спочатку кількість бажаних рівнів піраміди n ($1 \leq n \leq 10^9$, тобто мільярд), потім розмір партії Стандартних Будівельних Блоків m ($1 \leq m \leq 10^6$, тобто мільйон), і виводить єдине ціле число — кількість Блоків, що залишаться не використаними, якщо купити найменшу можливу кількість цілих партій.

Приклад:

Вхідні дані	Результати
7 16	4

Примітка. Піраміда з 7 рівнів міститиме $1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 = 140$ блоків, і якщо купити 8 партій по 16 блоків, то цих 128 блоків не вистачить; тому треба купити 9 партій по 16 блоків, тоді з цих 144 блоків 4 залишаться зайвими.

Розбір задачі. Ця задача проста, щоб набрати частину балів, але набрати повний бал чи значну частину балів не так просто. Розв'язок ideone.com/w8Yu4L набирає половину балів. І тут є де помилитися й отримати ще менше (важливо і правильно врахувати смисл m , і взяти тип `int64`).

При $n \geq 3 \cdot 10^6$ (приблизно), сума $1^2 + 2^2 + \dots + n^2$ виходить навіть за межі 64-бітового типу. З цим можна боротися, застосовуючи найпростіший прийом *модульної арифметики*: робити додавання i^2 не як `s=s+sqr(int64(i))`, а як `s=(s+sqr(int64(i))) mod m`. Реалізація з вчасними «... mod m», набирає 150 балів (з 250). Тепер критичним стає те, що $\approx 10^9$ ітерацій циклу (ще й з громіздкою операцією `mod`) не поміщаються в обмеження часу (1 сек).

100%-ий спосіб № 1. Якщо знати (або вивести під час туру) формулу $1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$, можна поєднати її з засобами модульної арифметики й отримати зовсім інший розв'язок ideone.com/g6nSKa. Він взагалі не містить циклів (складність $\Theta(1)$), тож працює миттєво.

Щоб написати цей розв'язок, треба знати деякі математичні факти та властивості, і це може не всім подобатися. Але, по-перше, задача має альтернативний розв'язок; по-друге, навіть якби цей розв'язок був єдиним, це не суперечило б традиціям Всеукраїнської олімпіади з інформатики.

У цьому розв'язку треба писати саме " $\dots \bmod (6 \cdot m)$ " (а не " $\dots \bmod m$ "), бо такі властивості модульної арифметики: хоча $(a + b) \bmod p = ((a \bmod p) + (b \bmod p)) \bmod p$ — правильна для всіх a, b, p тотожність, і так само для множення, але для ділення не так (наприклад, $\frac{12}{2} \bmod 10 = 6 \bmod 10 = 6 \neq 1 = \frac{2}{2} = \frac{12 \bmod 10}{2 \bmod 10}$). Детальніші відомості про модульну арифметику просимо знайти самостійно.

Щодо того, як вивести формулу $1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$ під час туру — див., наприклад, dxdy.ru/topic22151.html. Звісно, доцільність витрачання часу туру на виведення формул істотно залежить від знань та умінь конкретного учасника.

100%-ий спосіб № 2. Навіть не знаючи ні формули зі способу № 1, ні модульної арифметики, можна побудувати інший повнобальний розв'язок, користуючись лише базовими, в межах загальнообов'язкового мінімуму, знаннями математики, а також спостережливістю та кмітливістю.

Якщо $n < 2 \cdot 10^6$, можна порахувати відповідь «в лоб», як на початку розбору. Інакше (враховуючи, що $m \leq 10^6$, $n \geq 2 \cdot 10^6$) вийде, що проміжок від 1 до n містить кілька (як мінімум, два, як максимум — сотні мільйонів) проміжків від 1 до m , від $m + 1$ до $2m$, від $2m + 1$ до $3m$, і т. д.

Розглянемо очевидні тотожності на проміжках від 1 до m та від $m + 1$ до $2m$. Помітимо, що сума $m^2 + 2m$ кратна m і тому не впливає на остаточною відповідь задачі. Аналогічно не впливають $m^2 + 4m$, $m^2 + 6m$, і т. д. Тобто, $(m + 1)^2 \bmod m = 1^2 \bmod m$, $(m + 2)^2 \bmod m = 2^2 \bmod m$, $(m + 3)^2 \bmod m = 3^2 \bmod m$, \dots , а звідси — сума усього другого проміжку $(m + 1)^2 + (m + 2)^2 + \dots + (m + m)^2$ має той самий залишок від ділення на m , що й сума усього першого $1^2 + 2^2 + \dots + m^2$.

З аналогічних причин, такий самий залишок мають і сума усього третього проміжку $(2m + 1)^2 + (2m + 2)^2 + \dots + (2m + m)^2$, і сума будь-якого подальшого. Тому досить цей одинковий для всіх проміжків залишок домножити на $n \operatorname{div} m$, а потім, якщо n не кратне m , окремо порахувати й додати шматочок від $(n \operatorname{div} m) \cdot m + 1$ до n (або від 1 до $n \bmod m$).

Отже, сумарна кількість усіх ітерацій не перевищить $m + (n \bmod m) < 2m \leq 2 \cdot 10^6$. Це довше, ніж $\Theta(1)$ зі «способу № 1», але теж вкладається у 1 сек. Щоправда, якби обмеження було не $m \leq 10^6$, а $m \leq 10^9$, це стало б не так (а «способу № 1» байдуже, тому він в деякому сенсі кращий).

Задача С. «Ко-анаграмічно-прості»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Число називається *простим*, якщо воно має рівно два різні дільники — себе і одиницю. Наприклад: 23 — просте число, а 35 не просте, бо $5 \times 7 = 35$. Число 1 теж не просте (лише один дільник).

Якщо у числі змінити порядок цифр, властивість простоти може змінитися: наприклад, 35 — не просте число, а 53 — просте.

Будемо називати число *ко-анаграмічно-простим*, коли при хоча б одному можливому порядку цифр утворюється просте число. Наприклад, 35 є ко-анаграмічно-простим (бо 53 просте), а 225 — не є, бо жодне з чисел 225, 252 та 522 не є простим.

Напишіть програму, яка читає одне ціле додатне значення n ($10 \leq n \leq 9999$) і виводить у першому рядку мінімальне можливе ко-анаграмічно-просте число, більше-рівне за n , а у другому рядку — ту перестановку цифр, яка робить його простим. Якщо при перестановці з'являються нулі спереду числа — слід вважати, що вони не впливають на значення числа (05 дорівнює 5), але виводити

слід обов'язково із цими нулями (якщо правильно 05, то вивести 5 неправильно). Разом з тим, перше число відповіді починати з нуля не можна.

Якщо можливі різні правильні відповіді — виводьте будь-яку одну з них.

Приклади:

Вхідні дані	Результати	Вхідні дані	Результати	Вхідні дані	Результати
35	35 53	49	50 05	225	227 227

Розбір задачі. Тут треба акуратно писати досить велику, як для II етапу, програму, поєднуючи різні стандартні алгоритми. Але, не зважаючи на страшну назву «ко-анаграмічно» та інші громіздкості, тут не дуже-то й треба самому придумувати щось математичне. Якщо, звісно:

- знати стандартний алгоритм перевірки простоти числа;
- вміти *або перевіряти, чи числа складаються з одних і тих самих цифр, або генерувати перестановки послідовності цифр.*

Стандартний алгоритм перевірки простоти числа n (при $n \geq 2$) — пробувати ділити його на 2, 3, ..., $\text{round}(\sqrt{n})$, і якщо хоч раз поділилося без остачі — число складене, якщо ні разу — просте. Важливо перевіряти до кореня (а не до n чи $n/2$), бо це значно менше (див. також стор. 30).

Перевіряти, чи числа складаються з одних і тих самих цифр у різному порядку, можна по-різному. Один з простих і зручних підходів — відсортувати (наприклад, за неспаданням) цифри окремо одного з них, окремо іншого, й порівняти отримані відсортовані послідовності (вони рівні тоді й тільки тоді, коли числа складаються з одних і тих самих цифр). Оскільки кількість цифр дуже маленька, нема смислу використовувати quickSort чи подібні ефективні алгоритми сортування, доречнішим буде сортування вставками чи навіть бульбашкове.

Таким чином, схема алгоритму може бути приблизно такою. Функція (підпрограма) «перевірити, чи є число ко-анаграмічно-простим» має вигляд:

1. Створити копію цього числа-аргумента у рядковому (string-овому) вигляді;
2. Відсортувати цифри (символи рядка) за неспаданням;
3. Перебрати усі числа з відповідною кількістю цифр, від 0...01 до 9...99, і для кожного з них:
 - (а) перетворити у рядок (знову як копію, щоб не псувати оригінал);
 - (б) теж відсортувати цифри числа за неспаданням;
 - (с) якщо відсортовані послідовності виявилися різними — значить, поточне число не є перестановкою цифр досліджуваного числа і його слід пропустити, а якщо однаковими — запустити перевірку поточного числа на простоту.

Якщо перевірка у п. 3с хоча б один раз виявила, що число просте — функція в цілому має повернути результат «число є ко-анаграмічно-простим». Якщо жодного разу не виявила — результат «не є».

Тепер лишається тільки перевіряти, чи є ко-анаграмічно-простим саме введене число n , потім $n+1$, $n+2$, ... Приклад реалізації — ideone.com/mGI9hm.

Для цієї реалізації дуже складно і вивести асимптотичну оцінку часу роботи, і використати цю асимптотику для оцінювання конкретного часу роботи в мілісекундах. Наприклад, зовсім не ясно, як оцінити кількість ітерацій самого зовнішнього циклу (скільки чисел n , $n+1$, ... треба перебрати, щоб гарантовано знайти ко-анаграмічно-просте). У перевірці простоти, хоч і видно верхню межу циклу $\text{round}(\sqrt{n})$, але ж розумна реалізація обриває цикл після першого знайденого дільника, і середня кількість ітерацій виявляється меншою (а наскільки — не ясно); і так далі.

Програма може працювати швидше, якщо, замість перебору всіх чисел з відповідною кількістю цифр (від 0...01 до 9...99), відразу генерувати лише ті, що складаються з потрібного набору цифр. При обмеженні $n \leq 9999$ це неважливо, тож не будемо описувати, як робити це вручну (охочі можуть знайти в літературі чи Інтернеті за назвою *генерація перестановок*, англ. *generate permutations*). У мові C++ така генерація є готова (функція `next_permutation` бібліотеки `algorithm`).

Задача D. «Хмарочоси»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 2 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Як відомо, місто Прямий Ріг являє собою одну пряму вулицю, уздовж якої прокладена координатна вісь Ox . Останнім часом у місті розгорнувся будівельний бум, внаслідок якого ПрямМіськБуд звів N хмарочосів. Кожен хмарочос можна охарактеризувати висотою h_i та координатою x_i (усі x_i різні, усі h_i строго додатні).

За задумами мерії Прямого Рігу, настав час обладнати на дахах деяких із хмарочосів оглядові майданчики. Прибутковість такого майданчику залежить від того, скільки з даху даного хмарочосу видно інших хмарочосів. Тому Вас попросили написати програму, яка підготує відповідну статистику. Дані про хмарочоси зберігаються у мерії в порядку зростання x_i , тож саме в такому порядку вони і вводяться у Вашу програму. Якщо дахи трьох або більше хмарочосів виявляються розміщеними на одній прямій, ближчі затуляють дальші, і дальших не видно.

Вхідні дані. 1-й рядок містить кількість хмарочосів N , наступні N рядків містять по два цілі числа кожен — координату x_i та висоту h_i . Гарантовано, що $1 \leq N \leq 4321$, $0 \leq x_1 < x_2 < \dots < x_N \leq 10^6$ (мільйон), $1 \leq h_i \leq 10^5$ (сто тисяч).

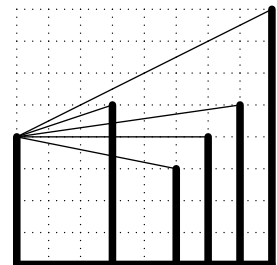
Результати. Програма має вивести N рядків, i -ий рядок має містити одне ціле число — кількість інших хмарочосів, які видно з даху хмарочосу № i .

Приклад:

Вхідні дані	Результати	
11	2	
0 4	5	
3 5	3	
5 3	4	
6 4	3	
7 5	10	
8 8	3	
10 5	4	
12 4	4	
14 3	3	
17 1	5	
19 7		

Розбір задачі. Намагання розв'язати задачу через три вкладені цикли (два перебирають поточну пару хмарочосів, всередині них третій перебирає усі проміжні, дивлячись, чи не затуляє він видимість поточної пари) — так собі ідея. Такий розв'язок набирає 150–190 балів з 250. Конкретне значення у проміжку 150–190 залежить, головним чином, від того, чи обривається третій цикл негайно, як тільки встановлений результат. Асимптотична складність: якщо 3-ій цикл обривається, то $O(N^3)$, якщо ні — $\Theta(N^3)$.

Для ефективнішого розв'язку, введемо поняття *нахил* з j -го хмарочосу на k -й (на рис. зображені нахили з 1-го хмарочоса на інші). Нахили можна подавати у програмі або кутовими коефіцієнтами $\frac{y_k - y_j}{x_k - x_j}$, або векторами $(x_k - x_j; y_k - y_j)$. (Про переваги таких векторів у багатьох інших задачах див. goo.gl/6урррју зокрема і за назвою «обчислювальна геометрія» взагалі; але у цій задачі достатньо й кутових коефіцієнтів, якщо подавати їх у якнайточнішому типі extended (long double).)



Розглянемо (припускаючи, що $N > 1$), як порахувати кількість хмарочосів, видимих з 1-го. 2-й хмарочос гарантовано видимий з 1-го. Запам'ятаємо нахил з 1-го хмарочоса на 2-й і оголошимо його *поточним нахилом затулення горизонту*. Потім бігтимемо по решті хмарочосів зліва направо, і нахил зі все того ж 1-го на кожен наступний виявлятиметься або нижчим-або-рівним, ніж нахил поточного затулення горизонту (за годинниковою стрілкою), або вищим (проти годинникової стрілки). У першому випадку, поточного хмарочосу не видно й нічого робити не треба. У другому — треба додати одиничку на позначення того, що поточний хмарочос видно з 1-го, і оновити нахил поточного затулення горизонту.

Для інших хмарочосів, є ще хмарочоси зліва, які теж треба врахувати. Ускладнення не принципове, але *як краще* це врахувати? Писати ще один цикл, який біжить від поточної позиції наліво — спосіб можливий, але *не* найкращий. Зокрема, тому, що саме в таких дублюваннях схожих фрагментів часто з'являються помилки (особливо, якщо написати спочатку неправильно і вносити правки).

Досить зручний спосіб, який і дозволяє не писати зайвого, і працює дещо швидше за інші — врахувати симетричність видимості (k -й видно з j -го тоді й тільки тоді, коли j -й видно з k -го). Завдяки цьому, коли при розгляді сусідів j -го хмарочоса з'ясовано, що з нього видно k -й ($k > j$), можна додати по одиниці і до кількості видимих з j -го, і до кількості видимих з k -го.

Сумарна асимптотична оцінка такого розв'язку становить $\Theta(N^2)$.

3.8 III (обласний) етап 2014/15 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання № 48).

Задача А. «Гірлянда — garland»

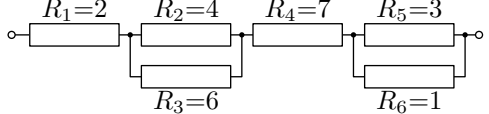
Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

На Новорічні свята Василь разом з батьком вирішили зробити електричну гірлянду із лампочок. При створенні гірлянди лампочки включаються в коло послідовно та деякі паралельно так, що в паралельному з'єднанні може знаходитись лише дві лампочки. Для з'ясування можливості підключення гірлянди до джерела живлення, Василю треба визначити загальний опір створеної гірлянди. З фізики йому відомо, що при послідовному з'єднанні загальний опір кола розраховується за формулою: $R_1 + R_2 + \dots + R_n$, а при паралельному: $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$.

Напишіть програму `garland`, яка б обчислювала загальний опір гірлянди.

Вхідні дані. У першому рядку задано загальну кількість лампочок N , в другому — через пробіл опір кожної з лампочок (значення опору є цілим числом і знаходиться в межах від 1 до 1000). В третьому рядку знаходиться ціле число K — кількість пар лампочок ($0 \leq K \leq N/2$), які включені в коло паралельно. Кожен наступний рядок містить по два порядкових номери лампочок, що включені паралельно.

Результати. Загальний опір гірлянди (як дійсне число без заокруглень).

Приклад:	Вхідні дані	Результати	Схема прикладу
	6 2 4 6 7 3 1 2 2 3 5 6	12.15	

Примітка. Відповідь можна виводити також і у форматі 1.21500000000000E+01.

Розбір задачі. Задача в основному на реалізацію, тобто головне — уважно прочитати й акуратно реалізувати. Єдиний неочевидний момент — як рахувати суму всіх тих *опорів*, які *не* були задіяні у паралельних підключеннях.

Один з можливих способів такий: спочатку порахувати суму абсолютно всіх опорів; читаючи чергову пару номерів опорів, з'єднаних паралельно, не лише обчислювати опір їхнього паралельного з'єднання $1/(1/R[i] + 1/R[j])$ і додавати його до результату, але також і віднімати з того ж результату $R[i] + R[j]$, бо ці опори раніше додали, а виявилось, що даремно.

Ще один можливий спосіб — почати з розгляду паралельних пар, і щоразу, додавши до загальної суми $1/(1/R[i] + 1/R[j])$, *замінити* $R[i]$ та $R[j]$ на нулі. Завершивши розгляд усіх паралельних пар, пододавати всі $R[i]$ (уже використані будуть нулями й не вплинуть на результат).

Приклади програм-розв'язків: ideone.com/QwkKWN (першим способом), ideone.com/QNgTua (другим). Асимптотика однакова — $\Theta(N)$.

Задача В. «Прямокутники — rectangles»

Вхідні дані: Або клавіатура, або input.txt

Обмеження часу: 1 сек

Результати: Або екран, або output.txt

Обмеження пам'яті: 64 мегабайти

Дані два прямокутники, сторони яких паралельні вісям координат.

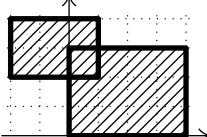
Завдання. Напишіть програму rectangles, яка б визначала площу їхнього об'єднання, тобто усієї частини площини, що покрита хоча б одним з прямокутників. Площу спільної частини обох прямокутників (якщо така є) слід враховувати один раз.

Вхідні дані. Вхідні дані містять два рядки, кожен з яких описує один з прямокутників, у форматі $x_{\min} \ x_{\max} \ y_{\min} \ y_{\max}$. Усі координати є цілими числами, що не перевищують по модулю 1000.

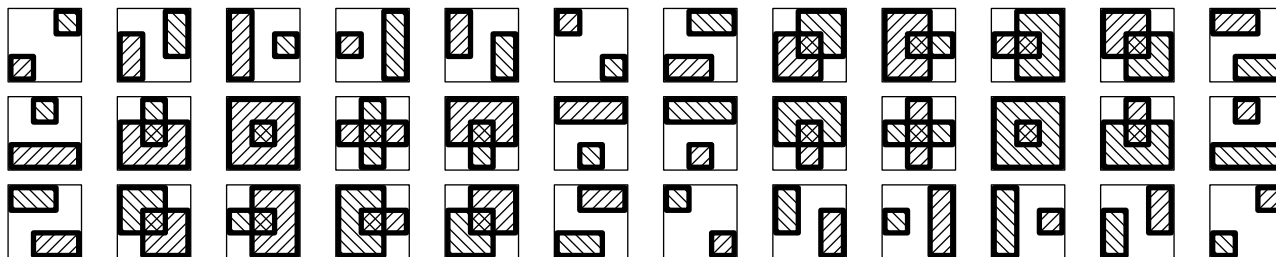
Результати. Ваша програма має вивести єдине ціле число — знайдену площу об'єднання.

Приклади:

Вхідні дані	Результати
0 10 0 10 20 30 20 30	200
0 20 0 30 10 12 17 23	600

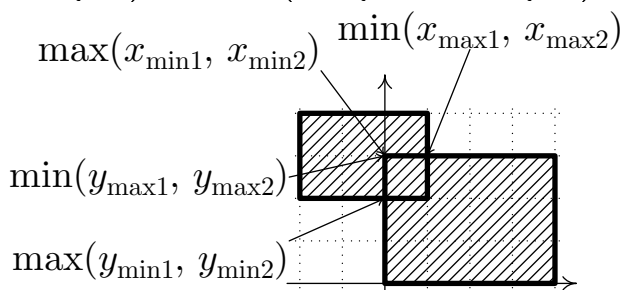
Вхідні дані	Результати	Зображення
0 4 0 3 -2 1 2 4	17	

Розбір задачі. Цю задачу важко вирішити на повні бали шляхом аналізу випадків. Просто тому, що їх більше, ніж може здатися. Навіть якщо не розглядати випадки рівності деяких із координат (як-то «не розглядати окремо ситуацію $x_{\min 1} = x_{\max 2}$, а об'єднати з ситуацією $x_{\min 1} > x_{\max 2}$, написавши $x_{\min 1} \geq x_{\max 2}$ »), все одно лишаються такі потенційно різні випадки:



Ситуації, коли прямокутники обміняні місцями, наведено як різні, бо якщо виводити для кожного випадку свою формулу залежності результату від вхідних даних, вони можуть бути різними. Звісно, деякі з цих випадків все одно можна об'єднати. Але не так просто зробити це правильно і ніде не помилитися. Тому пропонується розв'язати задачу інакше.

1-й спосіб (придатний для довільних координат). Площа об'єднання дорівнює сумі площ окремо взятих прямокутників мінус площа перетину (спільної частини); якщо спільної частини нема, площею перетину вважається 0. (Це міркування є частковим випадком *принципу включень та виключень*). Щоб знайти перетин, можна (окремо для x , окремо для y) узяти максимум (максимум із мінімумів) і мінімакс (мінімум із максимумів). Наприклад:



$$\begin{aligned} \text{максимін } x &= \max(\underbrace{x_{\min 1}}_{=0}, \underbrace{x_{\min 2}}_{=-2}) = 0; \\ \text{мінімакс } x &= \min(\underbrace{x_{\max 1}}_{=4}, \underbrace{x_{\max 2}}_{=1}) = 1; \\ \text{максимін } y &= \max(\underbrace{y_{\min 1}}_{=0}, \underbrace{y_{\min 2}}_{=2}) = 2; \\ \text{мінімакс } y &= \min(\underbrace{y_{\max 1}}_{=3}, \underbrace{y_{\max 2}}_{=4}) = 3. \end{aligned}$$

Якщо за кожною з координат максимум строго менший мінімакса, то площа перетину ненульова і її справді треба відняти.

Асимптотична оцінка: $\Theta(1)$. Реалізація: ideone.com/naDNCA.

2-й спосіб (менш універсальний, але саме тут теж правильний). В умові задано невеликі (як для комп'ютера) обмеження на значення координат. Це дає можливість перебрати «клітинки»

і для кожної перевірити, чи належить вона 1-му прямокутнику та чи належить 2-му. Завдяки використанню операції or (C-подібними мовами $||$) можна не розбиратися з випадками, і не важливо, чи мають прямокутники непорожній перетин, чи не мають. Ідея частково повторює згадану на стор. 38 (задача «Графічний пароль»), але тут не треба нічого (крім вхідних даних) запам'ятовувати.

Реалізація цього способу: `ideone.com/BN0trE`. Код навіть коротший і простіший, ніж першим способом. Але: якби дозволялися дробові значення координат, цей спосіб був би принципово неможливим; при цілих координатах, його асимптотична оцінка $O(X \cdot Y)$ (де X та Y — розміри діапазонів можливих значень координат), що немало.

Задача С. «Генератор паролів — password»

Вхідні дані: Або клавіатура, або `input.txt` Обмеження часу: 1 сек
Результати: Або екран, або `output.txt` Обмеження пам'яті: 64 мегабайти

У генераторі паролів закладена схема, яка за певними правилами утворює паролі із комбінацій цифр та великих літер англійського алфавіту. Цифри та літери в паролі можуть розташовуватись лише за зростанням (для літер зростання визначається алфавітом). Паролі містять, як мінімум, один символ (літеру або цифру). Якщо пароль містить і цифри, і літери, то цифри завжди йдуть після літер. Повторення символів не допускається.

Наприклад: `CH15` — коректний пароль, а `OLYMPIAD` — некоректний пароль (оскільки літери розташовані не в алфавітному порядку).

Усі паролі генеруються послідовно у вигляді впорядкованого списку. Якщо два паролі містять різну кількість символів, то першим іде пароль з меншою кількістю. Якщо декілька паролів мають однакову кількість символів, то вони генеруються в алфавітному порядку (при цьому літери вважаються меншими за цифри).

Початок впорядкованого списку паролів має наступний вигляд: `A, B, ..., Z, 0, 1, 2, ..., 9, AB, AC, ..., A9, BC, ...`

Завдання. Напишіть програму `password`, яка визначатиме n -ий пароль у списку паролів, який утворить генератор.

Вхідні дані. Число n ($1 \leq n \leq 10^{10}$).

Результати. Ваша програма має вивести n -ий пароль.

Вхідні дані	Результати
1	A
37	AB

Приклади:

Оцінювання. Приблизно 50% балів припадає на тести, в яких $n \leq 50000$. Ще приблизно 25% — на тести, в яких $n \leq 10^7$. Решта (приблизно 25%) — на тести, в яких $10^9 \leq n \leq 10^{10}$.

Примітка. Англійський алфавіт має вигляд `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`. Літери алфавіту у таблиці ASCII йдуть поспіль, під номерами від 65 ("A") до 90 ("Z"). Зростаюча послідовність цифр `0 1 2 3 4 5 6 7 8 9` також неперервна у таблиці ASCII (але не поспіль з літерами). Цифри йдуть під номерами від 48 ("0") до 57 ("9").

Розбір задачі. Легенда задачі відірвана від життя, ці правила не мають нічого спільного з реальною генерацією паролів. Тим не менш, будемо користуватися термінами «пароль» і «номер паролю», раз вони введені в умові.

Позбудемось окремих статусів літер і цифр, перейшовши до єдиного алфавіту. Правила про порядок можна спростити до таких:

- будемо вважати алфавітом `A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9` (саме в такому порядку);
- всередині кожного пароля (крім 1-символьних), кожен наступний символ мусить бути строго більшим за попередній (згідно алфавіту п. 1).

Правила порівняння різних паролів не міняються (але примітка «літери менші за цифри» тепер слідує з «алфавітного порядку»).

Частковий розв'язок на 76 балів (зі 100). Оскільки в умові сказано, що досить багато балів припадає на не дуже великі значення n , можна писати перебір, тобто дійсно генерувати послідовно 1-й, 2-й, ... паролі аж до n -го. Виявляється, при цьому не дуже важко добитися, щоб генерувалися відразу лише допустимі (згідно з умовою задачі) паролі.

Розглянемо простіший випадок. Нехай потрібно вивести послідовно пари (1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5), (4,5), тобто пари з чисел від 1 до 5, щоб 2-й елемент пари завжди був строго більшим за 1-й. Це можна зробити за допомогою циклів та if-а, як у лівому стовпчику рис. 3. А можна, як у правому, замінити перевірку $i < j$ на задання лише потрібного діапазону.

<pre>for i:=1 to 5 do for j:=1 to 5 do if i < j then writeln(i, ', ', j);</pre>	<pre>for i:=1 to 4 do for j:=i+1 to 5 do writeln(i, ', ', j);</pre>
------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------

Рис. 3. Два способи генерації однієї послідовності пар

Для 2-х вкладених циклів це дрібна оптимізація (\approx удвічі). Але якщо застосувати ту саму ідею до циклів більшої вкладеності, пришвидшення істотне: для п'яти вкладених циклів \approx сотні разів, для десяти — \approx мільйони. (Доведення потребує знань комбінаторики, але не дуже глибоких, тож пропонується як вправа. До того ж, писати програму можна й без доведення...)

Приклад такого розв'язку — ideone.com/Xg6sKT. Це зовсім не вірець краси й лаконічності. У ньому легко допустити і важко шукати технічні помилки. Але він все ж набирає чимало балів.

Повний (100%) розв'язок. Перш за все, загальна кількість k -символьних паролів рівна $C(36, k)$, де $C(n, k)$, воно ж C_n^k — кількість *сполучень* (рос. *сочетания*, англ. *combinations*) з n по k . Це так, бо з 36 символів вибираються k різних, і одні й ті самі вибрані символи не можна переставляти місцями, бо дозволений лише порядок за зростанням (у алфавіті A, B, ..., Z, 0, 1, ..., 9).

Так що почнемо розв'язок з того, що визнаємо довжину (кількість символів) шуканого пароля і його номер *серед паролів цієї довжини*.

Наприклад, прочитали у вхідних даних 2015.

- Кіль-ть 1-символьних паролів $C(36, 1)=36$, $36 < 2015$ — отже, у паролі більше одного символу, й номер серед (більш-ніж-1)-символьних $2015 - 36 = 1979$.
- Кіль-ть 2-символьних паролів $C(36, 2)=630$, $630 < 1979$ — отже, у паролі більше двох символів, і номер серед (більш-ніж-2)-символьних $1979 - 630 = 1349$.
- Кіль-ть 3-символьних паролів $C(36, 3)=7140$, $7140 \geq 1349$ — отже, пароль 3-символьний, і його номер серед 3-символьних рівний 1349.

Узнавши довжину пароля та його номер серед паролів відповідної довжини, починаємо визнавати цей пароль символ за символом, зліва направо — на тій підставі, що для кожного можливого початку можна визнавати (засобами комбінаторики) кількість паролів з таким початком і знову приймати рішення, чи цей початок треба пропустити (всі паролі з таким початком мають менші номери), чи використати (потрібний номер якраз потрапляє у діапазон).

Продовжимо аналіз того самого прикладу (вхідний номер 2015, раніше з'ясовано, що його номер серед 3-символьних рівний 1349).

- Кіль-ть 3-символьних паролів, що починаються з А, рівна $C(35, 2) = 595$, бо продовження 2-символьне з 35 символів (від В до 9). $1349 > 595$ — отже, початок паролю не А, а якийсь подальший символ, і номер серед тих подальших $1349 - 595 = 754$.
- Кіль-ть 3-символьних паролів, що починаються з В, рівна $C(34, 2) = 561$, бо продовження 2-символьне з 34 символів (від С до 9). $754 > 561$ — отже, початок паролю не В, а якийсь подальший символ, і номер серед тих подальших $754 - 561 = 193$.
- Кіль-ть 3-символьних паролів, що починаються з С, рівна $C(33, 2) = 528$, бо продовження 2-символьне з 33 символів (від D до 9). $193 \leq 528$ — отже, початок паролю якраз-таки С, і його номер серед 3-символьних, що починаються з С — теж 193.

Далі відбувається аналогічний підбір наступного символу:

- Кіль-ть 3-символьних паролів, що починаються з “CD”, рівна $C(32, 1) = 32$, бо лишається дописати один символ, від E до 9. $193 > 32$ — отже, 2-га літера не D, а якийсь подальший символ, і номер серед тих подальших $193 - 32 = 161$.

: І так далі.

- Продовживши аналогічні міркування, отримаємо, що шуканий пароль 16-й серед тих, що починаються з “CJ”, а оскільки після J можуть іти лише символи, починаючи з K, то цим 16-м буде Z.

! Остаточоно, 2015-й пароль має вигляд “CJZ”.

Ці пояснення займають багато місця, але лише тому, що наведено приклад. Правильна реалізація цього комбінаторного алгоритму працює дуже швидко, вкладаючись у секунду з величезним запасом. Адже всього-то треба:

1. Познаходити $C(n, k)$, наприклад, усі зразу із проміжку $0 \leq k \leq n \leq 36$ за допомогою трикутника Паскаля;
2. Знайти кількість символів у паролі — віднімати циклом $C(36, 1)$, $C(36, 2)$, ...; якби дійшли до $C(36, 36)$, а номер після усіх віднімань все ще лишався надто великим — це означало б, що пароля вказаних вигляду і номера взагалі не існує. Але такого не буде, бо паролів $(2^{36} - 1)$ все-таки більше, чим $n \leq 10^{10}$. Значить — тут не більш як 36 порівнянь та віднімань;
3. Для кожної позиції (1-й символ, 2-й, ...) запустити цикл, щоб знайти конкретне значення відповідного символу — теж не багато, бо і позицій, і значень символів не більше 36.

Від вираження асимптотичної оцінки складності алгоритму типовим чином (через n зі вхідних даних) утримаємось, бо надто багато залежить від розміру алфавіту, а залежність часу роботи від значення n заплутана. Якби розмір алфавіту був змінним (і при цьому не з'являлася «довга» арифметика), можна було б говорити про час роботи $O(A^2)$, де A — розмір алфавіту.

Задача D. «Зámok — castle»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: D1 — 1 с, D2 — 3 с

Результати: Або екран, або output.txt Обмеження пам'яті: 128 мегабайтів

Стародавній záмок має прямокутну форму. Záмок містить щонайменше дві кімнати. Підлогу замка можна умовно поділити на $M \times N$ клітин. Кожна така клітинка містить «0» або «1», які задають порожні ділянки та стіни замку відповідно.

Завдання. Напишіть програму castle, яка б знаходила кількість кімнат у замку, площу найбільшої кімнати (яка вимірюється кількістю клітинок) та площу найбільшої кімнати, яку можна утворити шляхом видалення стіни або її частини, тобто, замінивши лише одну «1» на «0». Видаляти зовнішні стіни заборонено.

Вхідні дані. План замку задається у вигляді послідовності чисел, по одному числу, яке характеризує кожну клітинку. Перший рядок містить два цілих числа M та N — кількість рядків та кількість стовпчиків ($3 \leq M \leq 1000$, $3 \leq N \leq 1000$). M наступних рядків містить по N нулів або одиниць, що йдуть поспіль (без пробілів). Перший та останній рядок, а також перший та останній стовпчик формують зовнішні стіни замку і складаються лише з одиниць.

Результати. Дана задача розділена в системі ejudge на дві підзадачі. У підзадачі D1 треба здати програму, що знаходить кількість кімнат та площу найбільшої кімнати замку (по одному числу в рядку), у підзадачі D2 — площу найбільшої кімнати, яка утвориться в разі видалення внутрішньої стіни.

Приклади:

Вхідні дані	Результати (D1)	Результати (D2)
6 8 11111111 10011001 10011001 11111001 10101001 11111111	4 8	10

III (обласний) етап 2014/15 навч. року
Черкаська обл., 20.01.2015

Вхідні дані	Результати (D1)	Результати (D2)
9 12 111111111111 101001000001 111001011111 100101000001 100011111101 100001000101 111111010101 100000010001 111111111111	4 28	38

Оцінювання. Значна частина тестів буде містити план замку з кімнатами лише прямокутної форми. Не менше половини тестів такі, що $3 \leq M \leq 20$, $3 \leq N \leq 50$.

Розбір задачі. Як легко набрати частину балів. Для самої лише підзадачі D1 і часткового випадку «всі кімнати мають прямокутну форму» в якості розв'язку можна запропонувати програму ideone.com/HHr9a3. Вона спирається на те, що у випадку прямокутності кімнат кожна кімната однозначно задається лівим верхнім кутом, а перевіряти, чи справді клітинка є таким кутом, можна умовою ($\text{data}[i][j]='0'$) and ($\text{data}[i-1][j]='1'$) and ($\text{data}[i][j-1]='1'$), тобто сама клітинка вільна, а ліворуч і згори стіни. Очевидно (в т. ч. з 2-го тесту з умови), що для «закручених» кімнат це може й не бути правдою. Але в умові обіцяно значну частину тестів з кімнатами прямокутної форми, тож при відсутності кращих ідей можна написати хоча б такий розв'язок. Він набирає 26 балів (з 50 за усю D1, зі 100 за усю D).

Повний розв'язок підзадачі D1 (лише ідеї). Для відстеження (як завгодно «закручених») кімнат можна реалізувати будь-який з алгоритмів:

1. Пошук ушир (у ширину), англ. *breadth first search (BFS)*;
2. Пошук углиб (у глибину), англ. *depth first search (DFS)*;
3. Різноманітні алгоритми графічної заливки (*flood fill*).

Усі ці алгоритми легко знайти в літературі чи Інтернеті, й усі вони надто громіздкі, щоб пояснювати їх тут. Кожним із них можна реалізувати задачу зі складністю $\Theta(N \cdot M)$. Для цього треба:

1. Просто перебирати усі клітинки замку, і щоразу, знайшовши 0, запускати BFS/DFS/заливку, щоб повністю виділити відповідну кімнату, позамінявши її нулі на інші значення, та обчислити її розмір.
2. Забезпечити, щоб кожен такий виклик BFS/DFS/заливки, якому вказується, починаючи звідки дослідити кімнату, працював за час, пропорційний розміру цієї кімнати, а не усього замку.

Що з DFS/BFS/заливки тут простіше й доречніше — важко сказати, сильно залежить від умінь конкретного учасника. Часто стверджують, ніби найпростіше реалізувати DFS, але ця думка сумнівна, бо сформована під впливом мови Паскаль, у якій є рекурсія й нема стандартної бібліотечної черги. Згадуючи DFS, варто згадати й про проблему переповнення стеку (див., зокрема, стор. 15–17). Конкретно на цій обласній олімпіаді забезпечено великий стек, тож можна без проблем писати рекурсивний DFS; але, враховуючи великий розмір замку 1000×1000 , при інших налаштуваннях ejudge проблеми з переповненням стеку рекурсивним DFS цілком можливі.

Підзадача D2. Частину балів (орієнтовно до 20 з 50) можна отримати, розв'язуючи підзадачу D1 багатократно (для абсолютно кожної «1» у внутрішній стіні, замінимо її на «0» і заново розв'яжемо D1; серед усіх таких відповідей виберемо максимальну). Але такий алгоритм має складність $O(M^2 \cdot N^2)$, тож ніяк не може бути ефективним розв'язком для $M \approx N \approx 1000$.

Перепишемо програму-розв'язок підзадачі D1 так, щоб при підрахунку кількостей та розмірів кімнати не просто виділялися, а *різні кімнати* виділялися *різними значеннями* (а клітинки однієї кімнати — однаковими). Якщо це робити прямо у масиві з позначками «0 — прохід, 1 — стіна», виділяючи кімнати позначками 2, 3, 4, ..., може вийти, наприклад, так, як праворуч.

Якщо зберігати розміри кімнат у масиві так, щоб індексами масиву були ті самі числа, якими позначено кімнати, то для визначення, яка вийде площа кімнати після руйнування стіни, можна просто додати до одинички (площі самої зруйнованої стіни) площі кімнат-сусідів.

Обласна інтернет-олімпіада 2015/16 навч. року
Черкаська обл., 08.10.2015

Тому перебір усіх можливих «1» (у внутрішніх стінах) можна залишити, бо тепер для кожної такої «1» треба робити значно менше дій (не виділяти заново відповідні кімнати, а брати готові, знайдені один раз, дані про них).

Здається «логічним» (і приклади з умови це «підтверджують»), ніби максимальна кімната буде утворена за рахунок об'єднання двох кімнат. Але це лише поширений випадок, а не обов'язкова властивість: замість 2-х може бути будь-яке число від 1 до 4.

Вхідні дані	Виділені кімнати	Розміри кімнат				
9 12		індекс	...	2	3	4 5
111111111111	1 1 1 1 1 1 1 1 1 1 1	значення	...	1	5	28 9
101001000001	1 2 1 3 3 1 4 4 4 4 1					
111001011111	1 1 1 3 3 1 4 1 1 1 1					
100101000001	1 5 5 1 3 1 4 4 4 4 1					
100011111101	1 5 5 5 1 1 1 1 1 1 4					
100001000101	1 5 5 5 5 1 4 4 4 1 4					
111111010101	1 1 1 1 1 1 4 1 4 1 4					
10000010001	1 4 4 4 4 4 4 1 4 4 4					
111111111111	1 3 2 5 1 1 1 1 1 1 1 1					
111111111111	1 1 1 1 1 1 1 1 1 1 1 1	9 11				
100000110000110000110001		11111111111	11111111111			
100000110000110000110001		10011011001	10011011001			
100000110000110000110001		11001010011	11000010011			
100000110000110000110001		11101010111	11101010111			
100000110000111111110001		10100100101	10100100101			
111111110000110000110001		11101010111	11101010111			
111111110000110000110001		11001010011	11001010011			
100000110000111111110001		10011011001	10011011001			
100000110000111111110001		11111111111	11111111111			

Тест № 9 (найлівіший з прикладів) призводить до з'єднання кімнат. А в тому єдиному місці, де призводить (2-й знизу рядок) — утворюється кімната площею всього-навсього $2 + 1 + 1 = 4$. Набагато більшу площу $44 + 1 = 45$ можна отримати, зруйнувавши будь-яку (не зовнішню) стіну кімнати площі 44, утворивши «нішу» замість «проходу».

Тест № 16 (середній, він же лівіший з майже однакових прикладів). Руйнування однієї «1» по центру призводить до з'єднання *відразу* 4-х кімнат. Переконавшись у можливості такої ситуації, легко уявити і вхідні дані найправішого прикладу, де різні сусіди центральної одинички до того ж ще й десь далеко з'єднані в одну кімнату.

Отже — акуратно переглянути для кожної не зовнішньої «1» усі кімнати-сусіди, додаючи площі *усіх різних*. Складність цього алгоритму теж (як і для підзадачі D1) $\Theta(N \cdot M)$, але множник, яким нехтують у асимптотичних позначеннях, тепер значно більший.

3.9 Обласна інтернет-олімпіада 2015/16 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання № 12).

Задача А. «Високосні роки»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Сучасний григоріанський календар діє, починаючи з п'ятниці 15 жовтня (жовтня) 1582 року. Він відрізняється від попереднього юліанського трохи складнішим, але зате й більш точним правилом визначення, які роки високосні (містять дату 29 лютого (февреля)), а які ні. У юліанському правило було дуже просте: якщо номер року кратний 4 (іншими словами, ділиться на 4 без остачі), то лютий містить 29 днів, а якщо не кратний, то 28 днів. Але Земля обертається навколо Сонця за приблизно 365,242 доби, а не 365,250 рівно, що й призвело до накопичення похибки (10 днів за 1500 років). Тому правило визначення, чи високосний рік, у сучасному григоріанському календарі таке: **“Високосним є кожен четвертий рік, за винятком років, номер яких ділиться без залишку на 100, але не ділиться без залишку на 400.”** Зокрема, 1700, 1800 і 1900 роки були не високосними за григоріанським календарем (хоч і високосними за юліанським). Але 1600 і 2000 роки були високосними за обома календарями. Так і вийшло, що різниця між цими календарями, яка у XVI ст. складала 10 днів, наразі становить 13 днів.

Напишіть програму, яка читатиме номер року і визначатиме, чи високосний цей рік за кожним із цих календарів.

Вхідні дані. Єдине ціле число, у проміжку $1583 \leq y \leq 4321$ — номер року.

Результати. Виведіть у першому рядку або єдине слово “YES”, якщо рік високосний за григоріанським календарем, або єдине слово “NO”, якщо не високосний. Другий рядок теж має містити єдине слово “YES”, або “NO”, з аналогічним смислом, але для юліанського календаря. Важливо, щоб слова “YES” та/або “NO” були написані великими латинськими літерами, без лапок.

Приклади:

Вхідні дані	Результати	Вхідні дані	Результати
2015	NO NO	2000	YES YES
2012	YES YES	1900	NO YES

Розбір задачі. З юліанським (старим) календарем усе дуже просто: `if year mod 4 = 0 then writeln('YES') else writeln('NO')`. З григоріанським (новим) трохи складніше, але потребує лише акуратності, без придумок. Буквально слідуючи умові задачі, можна отримати таке: `if (year mod 4 = 0) and ((year mod 100 <> 0) or (year mod 400 = 0)) then writeln('YES') else writeln('NO')`. Інший більш-менш зручний спосіб — спочатку, якщо рік взагалі не кратний 4, то не високосний; інакше, якщо кратний 400, то високосний; інакше, якщо кратний 100, то не високосний (випадок кратності 400 вже розглянутий в іншій гілці); інакше, високосний (раніше вже розглянуто і взагалі не кратні 4, і особливі випадки, лишаються високосні).

Можна і не писати перевірку високосності за григоріанським календарем самому, а використати бібліотечну. Наприклад, мовою FreePascal є функція `IsLeapYear`; мовою Java — метод `isLeapYear` класу `GregorianCalendar`.

Чи завжди можна використовувати на олімпіаді бібліотечні засоби замість писати сам(ому/ій)? Якщо бібліотека не стандартна — мабуть, ні (додавати бібліотеки на сервер на прохання учасника якось не прийнято). Крім того, на локальному комп'ютері, де пише учасник, і на сервері, де відбувається перевірка, можливі відмінності у переліку бібліотек. Тут лишається тільки або знати конкретні факти (що в яких версіях є), або питати в журі, або пробувати (якщо дозволена багато-кратна відправка розв'язків, можна витратити 1–2 спроби, щоб узнати, чи працює на сервері бажана функція бажаної бібліотеки: ніщо не заважає, навіть не пишучи розв'язок повністю, швиденько написати і здати щось, що при наявності бажаної бібліотеки/функції пройде хоча б пару тестів, а при відсутності дасть вердикт «Помилка компіляції», і глянути, що виходить).

Задача В. «Фасування олії»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек

Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

На виробництві по переробці насіння в день виробляють N літрів олії. Для її фасування використовують тару об'ємом 1, 2, 3, 4, 5, 6 літрів.

Напишіть програму `oil`, яка б давала можливість з'ясувати, яку мінімальну кількість тари необхідно для фасування.

Вхідні дані. Єдине ціле число N — кількість літрів олії.

Результати. Програма повинна вивести єдине ціле число — мінімальну кількість тари.

Приклади:

Вхідні дані	Результати	Вхідні дані	Результати	Вхідні дані	Результати
7	2	12	2	19	4

Розбір задачі. Завдяки тому, що в переліку об'ємів тари є *всі* підряд об'єми 1, 2, 3, 4, 5 та 6 л, завжди можна використати $N \div 6$ штук якнайбільших 6-літрових пляшок, а решту (якщо така є, тобто якщо $N \bmod 6 > 0$) розмістити у ще одну, останню, пляшку. Ще меншої кількості пляшок не досить, бо сумарний об'єм меншої кількості пляшок точно строго менший потрібного.

Тобто, відповідь можна виразити як `res:=N div 6; if N mod 6>0 then inc(res)`, або навіть одним виразом $\lceil N/6 \rceil$ (тобто $N/6$, заокруглене догори) — це може бути записано як $(N+5) \div 6$ (Pascal) чи `ceil(N/6.0)` (C/C++).

Насамкінець, *якби* у переліку об'ємів були не всі підряд числа від 1 до 6, а лише деякі, жадібний алгоритм «взяти якнайбільшу кількість якнайбільших пляшок» міг би виявитися і неправильним (наприклад: якби існували лише пляшки 1 л, 5 л та 6 л, а пофасувати треба було 16 л, то оптимальний спосіб $6+5+5$ отримувався б *усупереч* згаданій жадібній ідеї). Тоді слід було б використовувати деякий набагато складніший алгоритм; наприклад, оснований на псевдополіноміальному динамічному програмуванні (з використанням масиву, індекси якого відповідають кількості олії, яку треба пофасувати, а значення — відповідним мінімальним кількостям пляшок).

Задача С. «Нестандартне рівняння»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Напишіть програму, яка шукатиме (шляхом перебору) кількість розв'язків на проміжку $[a; b]$ рівняння

$$k \times f(n) = n,$$

де $f(n)$ — сума квадратів цифр у десятичному записі числа n .

Вхідні дані. Єдиний рядок містить три розділених пробілами цілі числа, у порядку k а b . Гарантовано $a \leq b$, а також, що усі ці числа не менші 1 й не більші 10^{18} .

Результати. Програма повинна вивести єдине число — кількість таких n з проміжку $a \leq n \leq b$, що задовольняють рівняння.

Приклад:

Вхідні дані	Результати
51 5000 10000	3

Примітки. Цими трьома розв'язками є 7293, 7854 та 7905. Наприклад, 7293 є розв'язком, бо $51 \times (7^2 + 2^2 + 9^2 + 3^2) = 51 \times (49 + 4 + 81 + 9) = 51 \times 143 = 7293$.

Але нічого цього виводити не треба, треба лише кількість розв'язків.

Розбір задачі. Таке «рівняння» не має стандартних аналітичних засобів розв'язування, тож справді треба якось перебирати різні потенційно можливі варіанти й перевіряти, які з них дають розв'язок. Але перебір «в лоб» (для кожного n від a до b рахувати суму квадратів цифр і перевіряти виконання чи не виконання «рівняння») надто довгий, аж до 10^{18} ітерацій (час роботи — багато років). Тому, перебір треба суттєво оптимізувати. Помітимо такі факти:

1. Щоб виконалася рівність $k \times f(n) = n$, число n мусить бути кратним k , тож можна зразу перебирати лише ті числа від a до b , які кратні k (у цієї ідеї є дещо частково спільне з ідеями зі стор. 37 та 50).
2. На всьому проміжку до 10^{18} найбільшу суму квадратів цифр має $\underbrace{99 \dots 9}_{18 \text{ шгук}}$, тому для всіх n виконується $f(n) \leq 18 \times 9^2 = 1458$.

Тобто, перебір можна почати з найменшого n , одночасно кратного k і більшого або рівного a , і продовжувати, збільшуючи щоразу на k , доки не виявиться більшим $\min(b, 1548 \times k)$ (у програмі, мабуть, зручніше писати `while (n<=b) and (n<=18*81*a)`). Очевидно, такий цикл повторюватиметься ніяк не більше 1548 разів, що для комп'ютера зовсім мало. Лишається тільки реалізувати це акуратно, щоб не було проблем ні з переповненнями типу `int64` (`long long`), ні з похибками (якщо використовувати для проміжних обчислень типи з рухомою комою). Наприклад, ideone.com/PCj7qR.

Задача D. «П'ять неділь у місяці»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Для сучасного григоріанського календаря (деякі маловідомі його деталі наведені у задачі А), порахуйте, які з місяців уведеного року містять аж 5 неділь (як днів тижня; рос. воскресений, англ. sundays).

Вхідні дані. Єдине ціле число, у проміжку $1583 \leq y \leq 4321$ — номер року.

Результати. Розділені пробілами номери місяців, що містять по 5 неділей, обов'язково у порядку зростання. “Номери місяців” слід розуміти так: січень (январь) — 1, лютий (февраль) — 2, і так далі, грудень (декабрь) — 12. Вказувати кількість таких місяців не треба.

Приклади:	Вхідні дані	Результати	Вхідні дані	Результати
	2015	3 5 8 11	2000	1 4 7 10 12
	2012	1 4 7 9 12	1900	4 7 9 12

Примітки. Зокрема, у першому з прикладів стверджується, що у 2015 році чотири таких місяці: березень (март), травень (май), серпень (август), листопад (ноябрь). Для підтвердження, що це справді так, а також для загальної довідкової інформації, наведемо повний календар на 2015 рік (внизу сторінки).

Розбір задачі. Тут ще істотніше, ніж у задачі А, що можна або реалізовувати все самостійно, або використати бібліотечні засоби.

Самостійна реалізація. Можна використати (згаданий у задачі А) факт, що 15.10.1582 було п'ятницею, отже 01.01.1583 — субота (наприклад, за наданим календарем можна підрахувати, що 1 січня наступного року припадає на наступний після 15 жовтня день тижня). Ще можна згадати (чи побачити у наданому календарі), що не високосний рік містить цілу кількість тижнів + 1 день. Відповідно, високосний — цілу кількість тижнів + 2 дні. Отже:

01.01.1584	неділя	(01.01.1583 субота+1)
01.01.1585	вівторок	(01.01.1584 неділя+2, бо 1584 рік високосний)
01.01.1586	середа	(01.01.1585 вівторок+1)
01.01.1587	четвер	(01.01.1586 середа+1)
01.01.1588	п'ятниця	(01.01.1587 четвер+1)
01.01.1589	неділя	(01.01.1588 п'ятниця+2, бо 1588 рік високосний)
:	:	:

Значить, можна перебрати усі роки від 1583 по $y-1$ (де y — рік зі вхідних даних; кілька тисяч ітерацій — для комп'ютера зовсім не багато) і щоразу додавати 1 чи 2, користуючись правильною перевіркою високосності з задачі А. Так узнáємо, яким днем тижня починається потрібний рік y .

Тепер можна, наприклад, послідовно перебрати усі дні цього року, як у ideone.com/m71JWc. Можна і трохи оптимізувати, перебираючи лише місяці (а не дні), як у ideone.com/gnuUXY. Але раз раніше перебирали роки (можливо, кілька тисяч), виграш не буде справді помітним. Ще, можна замість функції `daysInMonth` тримати масив кількостей днів (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31) (у випадку високосності року 28 міняється на 29). І так далі — у цієї задачі взагалі багато правильних розв'язків...

Розв'язки, що істотно спираються на бібліотечні засоби. Вони істотно залежать від конкретних бібліотек та конкретної мови програмування, тож нема сенсу займати тут багато місця описами цих бібліотек. Наведемо лише приклади: ideone.com/2jLmH1 (Delphi чи FreePascal),

Січень	Лютий	Березень	Квітень	Травень	Червень
5 12 19 26	2 9 16 23	2 9 16 23 30	6 13 20 27	4 11 18 25	1 8 15 22 29
6 13 20 27	3 10 17 24	3 10 17 24 31	7 14 21 28	5 12 19 26	2 9 16 23 30
7 14 21 28	4 11 18 25	4 11 18 25	1 8 15 22 29	6 13 20 27	3 10 17 24
1 8 15 22 29	5 12 19 26	5 12 19 26	2 9 16 23 30	7 14 21 28	4 11 18 25
2 9 16 23 30	6 13 20 27	6 13 20 27	3 10 17 24	1 8 15 22 29	5 12 19 26
3 10 17 24 31	7 14 21 28	7 14 21 28	4 11 18 25	2 9 16 23 30	6 13 20 27
4 11 18 25	1 8 15 22	1 8 15 22 29	5 12 19 26	3 10 17 24 31	7 14 21 28
Липень	Серпень	Вересень	Жовтень	Листопад	Грудень
6 13 20 27	3 10 17 24 31	7 14 21 28	5 12 19 26	2 9 16 23 30	7 14 21 28
7 14 21 28	4 11 18 25	1 8 15 22 29	6 13 20 27	3 10 17 24	1 8 15 22 29
1 8 15 22 29	5 12 19 26	2 9 16 23 30	7 14 21 28	4 11 18 25	2 9 16 23 30
2 9 16 23 30	6 13 20 27	3 10 17 24	1 8 15 22 29	5 12 19 26	3 10 17 24 31
3 10 17 24 31	7 14 21 28	4 11 18 25	2 9 16 23 30	6 13 20 27	4 11 18 25
4 11 18 25	1 8 15 22 29	5 12 19 26	3 10 17 24 31	7 14 21 28	5 12 19 26
5 12 19 26	2 9 16 23 30	6 13 20 27	4 11 18 25	1 8 15 22 29	6 13 20 27

ideone.com/X9xo5s (Java), а описи відповідних бібліотек пропонуємо знайти в Інтернеті самостійно. Див. також зауваження наприкінці пояснень до задачі А цього змагання.

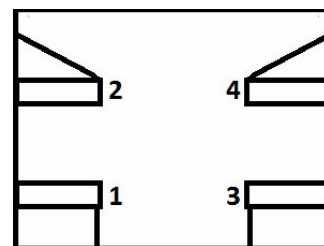
3.10 II (районний/міський) етап 2015/16 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №51).

Задача А. «Купе»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Купейний вагон містить 36 пасажирських місць — 9 купе по 4 місця у кожному. Половина серед цих місць нижні, решта — верхні. На рисунку наведено схему розміщення місць 1, 2, 3, 4 у купе №1. Купе №2 містить аналогічно розміщені місця 5, 6, 7, 8, і так далі, до купе №9, яке містить аналогічно розміщені місця 33, 34, 35, 36.



Напишіть програму, яка, прочитавши номери двох різних місць одного купейного вагону, визначатиме:

- чи в одному й тому ж купе розміщені ці місця;
- верхнє чи нижнє перше з цих місць;
- верхнє чи нижнє друге з цих місць.

Вхідні дані. Два числа p_1 p_2 у одному рядку, розділені одним пропуском (пробілом). Гарантовано виконуються обмеження $1 \leq p_1 \leq 36$, $1 \leq p_2 \leq 36$, $p_1 \neq p_2$.

Результати. Перший рядок повинен містити або єдине слово NO (якщо місця у різних купе), або слово YES і після нього через пробіл номер того купе, в якому розміщені обидва ці місця. Другий рядок повинен містити або єдине слово LOW (якщо перше з уведених місць нижнє), або єдине слово HIGH (якщо верхнє). Третій рядок теж повинен містити або слово LOW, або слово HIGH, але стосовно другого з уведених місць.

Приклади:

Вхідні дані	Результати
2 3	YES 1 HIGH LOW

Вхідні дані	Результати
23 17	NO LOW LOW

Примітка. Перевірте правильність написання у Вашій програмі слів YES, NO, LOW, HIGH (ВЕЛИКИМИ латинськими літерами). Автоматична перевіряюча система зараховує відповідь, лише коли вона правильна буква-в-букву.

Розбір задачі. Для пошуку першого рядка відповіді слід реалізувати код, що знаходить номер купе за номером місця, й порівняти результати для двох введених місць. Можна, наприклад, поділити (дробово) на 4, потім заокруглити вгору (мовами C/C++, $(\text{int})(\text{ceil}(p/4.0))$). Мовами, де стандартного аналога `ceil` нема, це можна виразити, наприклад, формулою $(p+3) \div 4$, де \div — цілочисельне ділення. Див. також стор. 23.

Очевидно, місця з непарними номерами нижні, а з парними верхні, тобто при $p\%2==0$ (на Pascal, $p \bmod 2=0$) HIGH, інакше LOW. Є й інші способи перевірки парності, як-то функція `odd(p)` (Pascal) чи порівняння $(p\&1)==1$ (C/C++/Java/C#/Python), воно ж $(p \text{ and } 1)=1$ (Pascal), що дають true при непарному p ; це працює, спираючись на т. зв. *бітовий and*, деталі знайдіть самостійно).

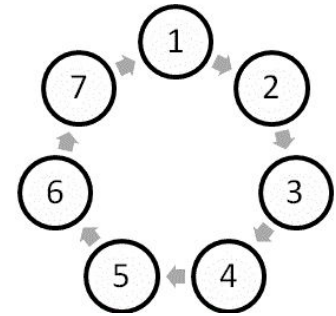
Складність усього разом узятого алгоритму очевидно $\Theta(1)$.

У ideone.com/0iknfP визначення номера купе і виведення LOW/HIGH оформлені як підпрограми. Це не обов'язково, але дозволяє уникнути деяких прикрих помилок, як-то «помилівся у формулі визначення номера купе, скопіював неправильну, потім для одного з місць виправив, а для іншого забув». Тому, винесення таких дій у підпрограми вважається правильним стилем і прикладом т. зв. «повторного використання коду» (code reusing).

Задача В. «Гра»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

В дитячій грі по колу розташовані кульки, кожна з них має номер від 1 до N ($1 \leq N \leq 100$). По черзі з кола забирають кожну K -ту кульку. Це відбувається до тих пір, поки в колі не залишиться остання кулька. Напишіть програму game, яка б визначала номер останньої кульки, що залишилась. При вхідних даних 7 3 кульки забираються у послідовності № 3, № 6, № 2, № 7, № 5, № 1.



Вхідні дані. N, K .

Результати. Номер кульки, що залишилась.

Приклад:

Вхідні дані	Результати
7 3	4

Розбір задачі. Наголосимо, що приклад з умови «№ 3, № 6, № 2, № 7, № 5, № 1» якраз відповідає описаним словами правилам:

- Маємо («закручену») послідовність 1 2 3 4 5 6 7. Рахуємо, починаючи з № 1, доки не дійдемо до 3, тобто кажемо «1» на № 1, «2» на № 2, «3» на № 3. Значить, прибрати з кола слід кульку № 3.
- Маємо («закручену») послідовність 1 2 4 5 6 7, і обрахунки починаються з кульки № 4 (наступної після щойно прибраної кульки № 3). Тобто, кажемо «1» на № 4, «2» на № 5, «3» на № 6, і прибираємо з кола кульку № 6.
- «Закручена» послідовність 1 2 4 5 7, починаємо з кульки № 7 — кажемо «1» на № 7, «2» на № 1, «3» на № 2, і прибираємо кульку № 2.
- «Закручена» послідовність 1 4 5 7, починаємо з кульки № 4 — кажемо «1» на № 4, «2» на № 5, «3» на № 7, і прибираємо кульку № 7.
- «Закручена» послідовність 1 4 5, починаємо з кульки № 1 — кажемо «1» на № 1, «2» на № 4, «3» на № 5, і прибираємо кульку № 5.
- «Закручена» послідовність 1 4, починаємо з кульки № 1 — кажемо «1» на № 1, «2» на № 4, «3» на № 1 і прибираємо кульку № 1.

Лишається єдина кулька № 4, вона й буде відповіддю.

На жаль, деякі учасники це не зрозуміли, стверджуючи, ніби кульки слід забирати у порядку «№ 3, № 6, № 2, № 5, № 1, № 4, № 7». Хоча це протирічить тому, що кульки забирають з кола. І взагалі, приклади для того й наводять, щоб учасники могли звірити з ними своє розуміння задачі; невідповідність прикладу словесному опису — ненормальна ситуація, що трапляється дуже рідко, і тут її не було.

Щодо власне розв'язку — при $N \leq 100$ годиться який-небудь спосіб «у лоб». Наприклад, підтримувати масив із номерами ще не забраних кульок, якимось так:

1. Заповнити масив як `for i:=1 to N do who[i]:=i;`
2. встановити поточну позицію на початковий індекс 1;
3. $N - 1$ раз повторити такі дії:
 - (а) обчислити, яка позиція знаходиться на $K - 1$ правіше поточної (з урахуванням «закільцованості» послідовності), і перейти туди;
 - (б) видалити елемент у поточній позиції (зі зсувом «хвоста»).

Повторювати треба $N - 1$ раз, бо один останній елемент лишається. Змінювати позицію від поточної треба на $K - 1$, бо сама поточна — вже або перша, а не нульова (на першій ітерації циклу), або перша *після* щойно видаленої (на подальших ітераціях), тож лишається зсув на $K - 1$.

Приклади реалізації щойно наведеного алгоритму: ideone.com/sRGzzS (Pascal), ideone.com/Otr18C (C++ з активним використанням STL). Зверніть увагу, що в обох реалізаціях при обчисленні нової поточної позиції береться залишок від ділення на поточну (не початкову!) кількість кульок (наприклад, `p:=(p+(k-1)) mod n`), і такий підхід значно кращий, ніж щось у стилі “`p:=p+(k-1); if p>N then p:=p-N;`”, бо кількість кульок зменшується аж до однієї (ну, двох, якщо рахувати перед

останнім вилученням), так що K запросто може багатократно перевищувати зменшене N . (Власне, в умові взагалі нема обмеження на K ; раптом з самого початку $K > N$? Це, в принципі, недолік умови; але, оскільки все одно є причини брати $\bmod N$, несуттєвий.) Складність цього алгоритму становить $O(N^2)$ (кожен зсув потребує $O(N)$ дій), що для $1 \leq N \leq 100$ цілком прийнятно (навіть «із запасом»).

Є й інші прості способи, прийнятні при $N \leq 100$. Наприклад, при вилученні кульки можна не зсовувати хвіст масиву, а ставити якесь спеціальне число (як-то -1), яке позначало б «видалено». Тоді, щоб знайти число, яке видалятимуть наступним, треба послідовно перебирати елементи масива, відраховуючи K (чи $K \bmod N_{curr}$) ще не видалених і пропускаючи вже видалені.

Ця задача має назву («Йосипа Флавія», рос. «Иосифа Флавия», англ. «Josephus problem»), і для неї відомі ефективніші алгоритми. Охочі можуть знайти їх, але вони складніші й були б актуальними лише для значно більших N .

Задача С. «Сучасне мистецтво»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Галерея готує показ творів сучасного мистецтва кількох митців і розглядає різні шляхи оформлення виставки. Так як це сучасне мистецтво, всі твори одного художника мало чим відрізняються один від одного.

Наприклад, припустимо, що є один твір митця А, два твори митця В і один твір митця С. Тоді є 12 різних шляхів оформити виставку в галереї:

- | | | | | | |
|---------|---------|---------|---------|----------|----------|
| 1. ABBC | 3. ACBB | 5. BACB | 7. BBCA | 9. BCBA | 11. CBAB |
| 2. ABCB | 4. BABC | 6. BBAC | 8. BCAB | 10. CABB | 12. CBBA |

Список наведений в алфавітному порядку.

Напишіть програму, що визначає n -ий спосіб оформлення виставки, враховуючи що всі способи слідує в алфавітному порядку.

Ваша програма приймає на вхід 5 цілих чисел: a , b , c і d (кожне від 0 до 5 включно), що визначають кількість робіт митців А, В, С і D відповідно, і останнє число n ($1 \leq n \leq 2^{34}$) — n -ий спосіб оформлення виставки. Гарантується, що хоча б один митець виставляє свою роботу в галереї. Також гарантується, що n не перевищує загальної кількості способів.

Як результат ваша програма має вивести рядок (із літер А, В, С, D), що визначає n -ий спосіб оформлення виставки.

Приклад:	Вхідні дані	Результати
	1 2 1 0 8	BCAB

Розбір задачі. Задача має багато спільного з задачею «Генератор паролів» (стор. 49–51). Тут теж варто знайти, скільки всього можливих послідовностей починається з 'А' і або побачити, що задане у вхідних даних $n \leq$ цієї кількості (тоді послідовність-відповідь починається з 'А' і слід продовжити підбирати літери на подальших позиціях), або, у протилежному випадку, слід відняти з заданого у вхідних даних n кількість цих послідовностей (що починаються з 'А') і зробити те саме щодо послідовностей, що починаються з 'В', 'С', 'D'.

Тож треба знаходити кількість послідовностей, що починаються з такої-то послідовності літер (і тому відомо, з яких літер у яких кількостях складаються можливі продовження). Виборки, де всі елементи розставляють усіма можливими порядками, не змінюючи набір цих елементів, називають *перестановками* (*permutations*). Якщо у наборі елементів, які переставляють, є однакові (наприклад, "АВСВ", причому, якщо обміняти місцями одне 'В' з іншим 'В', це вважається тим самим способом), то виборки називають *перестановками з повтореннями*, і їх кількість може бути виражена як

$$\overline{P}(k_1, k_2, \dots, k_n) = \frac{(k_1 + k_2 + \dots + k_n)!}{k_1! \cdot k_2! \cdot \dots \cdot k_n!},$$

де "!" означає факторіал, k_1, k_2, \dots, k_n — кількості елементів кожного типу. (Звідки береться ця формула? Пропонуємо знайти у літературі чи Інтернеті; при розв'язуванні задачі це не потрібне,

досить знати саму формулу; але виведення і допомагає її запам'ятати, і може бути корисним у інших ситуаціях.) В цій задачі, формула набуває вигляду $\overline{P}(a, b, c, d) = \frac{(a+b+c+d)!}{a! \cdot b! \cdot c! \cdot d!}$.

Лишається тільки поєднати розглянуті міркування. Зробіть це самостійно. А для контролю правильності розглянемо приклад $a = 1, b = 2, c = 3, d = 4, n = 2015$.

При початку на 'А', можливих продовжень $\overline{P}(0, 2, 3, 4)$, бо якщо взяти на першу позицію 'А', на подальші позиції не лишається жодної 'А', $b = 2$ штук 'В', $c = 3$ штук 'С', $d = 4$ штук 'D'. $\frac{(0+2+3+4)!}{0! \cdot 2! \cdot 3! \cdot 4!} = 1260 < 2015$, тобто відповідь починається не на 'А', а на одну з подальших літер, і номер відповіді серед тих подальших дорівнює $2015 - 1260 = 755$.

При початку на 'В', можливих продовжень $\overline{P}(1, 1, 3, 4)$, бо якщо взяти на першу позицію 'В', на подальші позиції лишається $a = 1$ штука 'А', $b = 1$ штука 'В', $c = 3$ штук 'С', $d = 4$ штук 'D'. $\frac{(1+1+3+4)!}{1! \cdot 1! \cdot 3! \cdot 4!} = 2520 \geq 755$, тобто відповідь починається якраз на 'В', і треба шукати подальші літери.

При початку на "ВА", можливих продовжень $\overline{P}(0, 1, 3, 4)$, бо якщо взяти на перші дві позиції "ВА", на подальші позиції лишається $a = 0$ штук 'А', $b = 1$ штука 'В', $c = 3$ штук 'С', $d = 4$ штук 'D'. $\frac{(0+1+3+4)!}{0! \cdot 1! \cdot 3! \cdot 4!} = 280 < 755$, тобто відповідь починається не на "ВА", а має на другій позиції одну з подальших літер, і номер відповіді серед тих подальших дорівнює $755 - 280 = 475$.

При початку на "ВВ", можливих продовжень $\overline{P}(1, 0, 3, 4) = 280 < 475$, тож відповідь починається не на "ВВ", а має на другій позиції одну з подальших літер, і номер серед тих подальших рівний $475 - 280 = 195$.

При початку "ВС", можливих продовжень $\overline{P}(1, 1, 2, 4) = \frac{(1+1+2+4)!}{1! \cdot 1! \cdot 2! \cdot 4!} = 840 \geq 195$, тож відповідь починається якраз на 'BC'.

При початку "BCA", можливих продовжень $\overline{P}(0, 1, 2, 4) = \frac{(0+1+2+4)!}{0! \cdot 1! \cdot 2! \cdot 4!} = 105 < 195$, тож відповідь має на третій позиції одну з подальших літер, номер серед подальших $195 - 105 = 90$.

При початку "BCB", можливих продовжень $\overline{P}(1, 0, 2, 4) = 105 \geq 90$, тож відповідь починається на 'BCB'.

При початку "BCBA", можливих продовжень $\overline{P}(0, 0, 2, 4) = \frac{(0+0+2+4)!}{0! \cdot 0! \cdot 2! \cdot 4!} = 15 < 90$, тож відповідь має на четвертій позиції одну з подальших літер, номер серед подальших $90 - 15 = 75$.

Тепер треба правильно врахувати (наприклад, передбачити if для таких ситуацій), що літер 'В' спочатку було лише $b = 2$ й тому початок "BCBB" взагалі неможливий (кількість = 0) і тому треба продовжити, перейшовши до "BCBC".

І так далі. Остаточна відповідь — "BCBDDCDCAD".

У тексті це займає багато місця, але виконується такий алгоритм дуже швидко — сумарна кількість обчислень формули $\overline{P}(\dots)$ не перевищує $4 \times (a + b + c + d)$ (бо кількість розрядів $a + b + c + d$, у кожному пробуються, щонайбільше, 'А', 'В', 'С', 'D'), тож загальна кількість дій може бути виражена як $O((a + b + c + d)^2)$ навіть при не найкращій реалізації самої формули, яка щоразу заново переобчислює факторіали; якщо ж зберігати готові факторіали, можна оптимізувати усю разом узятую програму до $\Theta(a + b + c + d)$ (втім, при настільки малих a, b, c, d це несуттєво).

Як легко й просто набрати 60% балів (лише C++). У бібліотеці `algorithm` мови C++ є функція `next_permutation`, яка дозволяє послідовно перебирати перестановки у порядку зростання (що й треба у цій задачі), і яка правильно працює з будь-якими перестановками (з повтореннями чи без). Так що можна сформувану string `s` зі значенням $\underbrace{\text{AA} \dots \text{A}}_{a \text{ штук}} \underbrace{\text{BB} \dots \text{B}}_{b \text{ штук}} \underbrace{\text{CC} \dots \text{C}}_{c \text{ штук}} \underbrace{\text{DD} \dots \text{D}}_{d \text{ штук}}$ і $n-1$ разів застосувати `next_permutation(s.begin(), s.end())`;.

Складність такого алгоритму становить $\Theta(n)$ (причому, `next_permutation`, хоч і має амортизовану складність $\Theta(1)$, не дуже швидко), тож він ніяк не може бути досить ефективним при $n \approx \overline{P}(5, 5, 5, 5) = \frac{20!}{5! \cdot 5! \cdot 5! \cdot 5!} \approx 1,173 \cdot 10^{10}$. В умові задачі нема розбаловки, тож не ясно, що це саме 60%; але очевидно, що якісь бали мусять бути.

А інші? Автори збірника не впевнені, чи достатньо дослідили бібліотеки інших мов програмування, але хоч трохи схожі засоби вдалося знайти лише у Python (клас `permutations` модуля `itertools`). Причому, цей клас не так, як треба, працює з повтореннями елементів, тож невідомо, як використати його краще, ніж `a = list(set(permutations(s)))`, `a.sort()`, тобто спочатку згенерувати геть усі перестановки (звичайні), потім використати `set`, щоб позбутися повторень, потім перетворити у масив (`list`) і відсортувати. Все це, звісно, працює *набагато* довше, головним чином, тому, що на проміжному етапі генеруються перестановки без повторень, а їх може бути значно більше: наприклад, при $a = b = 4, c = d = 3$ маємо $14! \approx 8,7 \cdot 10^{10}$ (причому їх треба ще й одночасно тримати у пам'яті) проти $\frac{14!}{4!4!3!3!} \approx 4,2 \cdot 10^6$ (що для комп'ютера небагато, причому вони перебираються одна за одною, не займаючи багато пам'яті). Цей спосіб набирає 40% балів.

А ще інші? На жаль, для ще інших мов програмування важко запропонувати простий спосіб набрати навіть таку частину балів. Це сумно, але: (1) 100%-й алгоритм можна написати будь-якою з доступних мов програмування; (2) нікому з учасників не заборонялося вчити й використовувати C++.

Писати власний аналог `next_permutation` не доцільно, бо потребує майже стільки ж зусиль, як ефективний розв'язок початкової задачі, а принести може, щонайбільше, ті ж 60% балів. Тим не менш, кому цікаво, може знайти у літературі чи Інтернеті *алгоритми генерації перестановок*. Є кілька різних класичних алгоритмів такої генерації. І *одні* з них (які визначають, що поміняти у поточній перестановці, щоб перейти до наступної) оптимізовані під відносно швидшу організацію перебору усіх підряд перестановок (у `next_permutation` саме такий). А *інші* (зокрема, рекурсивні) дозволяють вписування додаткових умов, щоб пропускати непотрібні. Говорячи про цю задачу, такі модифікації можна кінець кінцем перетворити до рекурсивної версії рекомендованого способу.

Задача D. «Кількість дільників на проміжку»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Напишіть програму, яка знайде суму кількостей дільників усіх чисел у проміжку від A до B (обидві межі включно, $A \leq B$).

Вхідні дані. У єдиному рядку через пробіл задані два натуральні числа A та B , які являють собою межі проміжку.

Результати. Виведіть у одному рядку єдине число — суму кількостей дільників усіх чисел проміжку.

Приклад:	<table><tr><th>Вхідні дані</th><th>Результати</th></tr><tr><td>119 122</td><td>27</td></tr></table>	Вхідні дані	Результати	119 122	27
Вхідні дані	Результати				
119 122	27				

Примітка. Число 119 має 4 дільники (1, 7, 17, 119); число 120 має 16 дільників (1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120); число 121 має 3 дільника (1, 11, 121); число 122 має 4 дільники (1, 2, 61, 122). Звідси відповідь $4 + 16 + 3 + 4 = 27$.

Оцінювання. 40% балів припадає на тести, в яких $1 \leq A \leq B \leq 1000$. Ще 20% балів припадає на тести, в яких $1 \leq A \leq B \leq 10^7$. Ще 10% балів припадає на тести, в яких $10^8 \leq B \leq 10^9$, але $B - 100 \leq A \leq B$. Решта 30% балів припадає на тести, де виконуються обмеження $1 \leq A \leq B \leq 10^{12}$ і не виконуються обмеження попередніх блоків. Писати треба одну програму, а не різні програми для різних випадків; єдина мета цього переліку різних блоків обмежень — дати уявлення про те, скільки балів можна отримати, якщо розв'язати задачу правильно, але не ефективно.

Розбір задачі. **Розв'язок на 40–50%.** Для отримання 40% балів досить реалізувати те, що написано в умові. Наприклад, див. праворуч. Але складність такого алгоритма $\Theta((B - A + 1) \times B)$, тож він отримує вердикт «Перевищено час роботи» на тестах усіх блоків, крім 1-го ($1 \leq A \leq B \leq 1000$).

```
res:=0;
for c:=a to b do
  for p:=1 to c do
    if c mod p = 0
    then
      res:=res+1;
```

Враховуючи міркування зі стор. 30 про те, чому для знаходження дільників N достатньо перебирати «претендентів» до \sqrt{N} (а не до N), цей алгоритм можна модифікувати, лишивши незмінним зовнішній цикл, але зменшивши діапазон внутрішнього до \sqrt{c} . Це дає складність $\Theta((B - A + 1) \times \sqrt{B})$, що допустимо як для 1-го блоку, так і 3-го.

Повний розв'язок. Не варто дотримуватися (нав'язуваної приміткою з умови) тактики «знайти кількості дільників кожного числа й додати». Кількості дільників окремих чисел не питають, тож можна рахувати відповідь якимось інакше. Наприклад, у іншому порядку. В усіх подальших алгоритмах, кількості дільників легше не шукати на проміжку від A до B , а знайти один раз на проміжку від 1 до B , інший — від 1 до $(A - 1)$, й відняти другий результат з першого (див. також стор. 20).

Число 1 є дільником усіх чисел, тож дільник 1 приносить у шукану суму стільки, скільки чисел у проміжку. Число 2 є дільником усіх парних чисел, тож приносить у шукану суму стільки, скільки на проміжку парних чисел. І т. д. А серед усіх чисел від 1 до N є рівно $N \operatorname{div} k$ чисел, кратних k .

На перший погляд, так треба перебирати всі дільники аж до N (яке один раз дорівнює $A - 1$, інший раз B), і алгоритм матиме складність $\Theta(A + B) = \Theta(B)$, що не досить ефективно ні для 4-го блоку тестів, ні навіть для 3-го.

Але це можна оптимізувати, використавши властивість «якщо p є дільником N , то (N/p) теж є дільником N , а з чисел p та N/p хоча б одне $\leq \sqrt{N}$ ». Приблизно так: перебравши дільники лише до \sqrt{N} , подвоїти результат, щоб урахувати дільники, більші \sqrt{N} . Це не зовсім правда, бо враховує деякі дільники двічі; але цю похибку можна компенсувати, й отримати правильний алгоритм підзадачі складності $\Theta(\sqrt{N})$ (отже, складність усієї задачі буде $\Theta(\sqrt{A} + \sqrt{B}) = \Theta(\sqrt{B})$).

Розглянемо всі дільники всіх чисел до 32. Кругечки позначають, що число «№ рядка» є дільником числа «№ стовпчика». Чорний (•) кругечок — дільник менший кореня відповідного числа, білий (○) — більший, напівзаповнений — рівний. Вертикальні лінії виражають попарний зв'язок між p та N/p . Штрихова горизонтальна лінія між 5 та 6 виражає перебір дільників до $\lfloor \sqrt{N} \rfloor$ (воно ж $\text{trunc}(\text{sqrt}(N))$ чи $\text{floor}(\text{sqrt}(1.0*N))$). Очевидно, сумарну кількість дільників можна виразити як кількість чорних кругечків, помножену на 2, плюс кількість напівзаповнених. А для пошуку цих кількостей достатньо перебрати рядки з 1-го по $\lfloor \sqrt{N} \rfloor$ -й, відзначаючи, що кількість усіх кругечків у рядку № i (при $1 \leq i \leq \lfloor \sqrt{N} \rfloor$) становить $N \text{ div } i$, і серед них є рівно 1 напівзаповнений та рівно $(i - 1)$ білий (що дає кількість чорних $(N \text{ div } i) - i$). Завершіть ці міркування самостійно та запрограмуйте їх.



Насамкінець, можлива ситуація, коли учасник олімпіади придумав і реалізував як алгоритм, що проходить 1-й та 2-й блоки тестів, так і алгоритм, що проходить 1-й та 3-й блоки, але не вміє розв'язати задачу повністю. Чи може такий учасник гарантувати собі 70% балів, узявши однією програмою бали і за 2-й, і за 3-й блоки? Запросто, якщо зробить так: прочитавши A та B , знайде, яке зі значень $(A + B)$ чи $((B - A) \times \sqrt{B})$ менше, і залежно від цього викличе одну чи іншу підпрограму, що реалізує відповідний алгоритм.

III (обласного) етапу 2015/16 н. р. нема

У 2015/16 навч. році III (обласний) етап у Черкаській області відбувався у Черкасах, але на задачах інших авторів, іншій системі (ejudge, але не ejudge.skipo.edu.ua), та й про його дорішування авторам цього збірника нічого не відомо. Тому, він до збірника не включений.

3.11 Обласна інтернет-олімпіада 2016/17 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №19).

Задача А. «Квартира»

Розглянемо 9-поверховий житловий будинок, в якому кілька під'їздів, і кожен поверх кожного під'їзду містить рівно 4 квартири. Таким чином, 1-й поверх 1-го під'їзду містить квартири №№ 1, 2, 3, 4; 2-й поверх 1-го під'їзду — квартири №№ 5, 6, 7, 8; ...; 9-й поверх 1-го під'їзду — квартири №№ 33, 34, 35, 36; 1-й поверх 2-го під'їзду — квартири №№ 37, 38, 39, 40, і так далі.

Напишіть *вираз*, котрий за номером під'їзду, номером поверху та «локальним» номером квартири у межах сходової клітки обчислюватиме «глобальний» номер квартири у будинку. Номер під'їзду повинен бути позначений обов'язково змінною `pid`, його значення гарантовано будуть від 1 до 20. Поверх повинен бути позначений обов'язково змінною `rov`, його значення гарантовано будуть від 1

до 9. Номер квартири у межах сходової клітки повинен бути позначений обов'язково змінною k_v , його значення гарантовано будуть від 1 до 4.

Ще раз: у цій задачі (єдиній з переліку) треба здати не програму, а вираз: вписати його (сам вираз, не назву файлу) у відповідне поле перевіряючої системи і відправити на перевірку. Правила запису виразу — спільні для більшості мов програмування: можна використовувати десяткові числа, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), круглі дужки “(” та “)” для групування та зміни порядку дій. Дозволяються пропуски (пробіли), але, звісно, не всередині чисел і не всередині імен змінних.

Наприклад: можна здати вираз $pid * pov * kv$ і отримати 26 балів з 200, бо він неправильний, але все ж іноді відповідь збігається з правильною. А «цілком аналогічний» вираз $a*b*c$ буде оцінений на 0 балів, бо змінні повинні називатися pid , pov , kv , а не a , b , c .

Розбір задачі. Розглянемо квартири 1-го під'їзду, розміщені одна над одною (з однаковим «локальним» номером). Кожен поверх збільшує шуканий номер квартири на 4: над 1-ю, на 2-у поверсі розміщена 5-а, на 3-у 9-а, і т. д. Це саме справедливо і для інших квартир тих самих сходових клітин: над 2-ю, на 2-у поверсі розміщена 6-а, на 3-у 10-а, і т. д. Тобто, в номері квартири присутній доданок $4*(pov-1)$ («-1», бо рахується кількість поверхів *під* поверхом pov).

Аналогічно, кожен з менших за номером під'їздів додає у номер квартири $4 \times 9 = 36$: квартира, що розміщена так само й на такому само поверсі, як 1-а, але у 2-му під'їзді, має номер 37; у 3-му — номер 73, тощо.

Відповіддю може бути, наприклад, $36*(pid-1) + 4*(pov-1) + kv$ (що вельми перегукується з міркуваннями зі стор. 23 про те, що формули були б зручніші, якби нумерація була з 0, а не з 1). Формулу-відповідь можна (це теж оцінюється на повні бали) й замінити на будь-який інший еквівалентний вигляд, як-то $kv + (pid*9+pov)*4 - 40$ чи ще якийсь.

Задача В. «Максимальний добуток»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Дано чотири цілі числа. Виведіть два з них, добуток яких максимальний.

Вхідні дані. Рівно чотири числа, що задаються кожне в окремому рядку. Серед цих чисел можуть (але не зобов'язані) бути рівні.

Результати. Рівно два (кожне в окремому рядку) з 4-х уведених чисел, які дають максимальний добуток. Це повинні бути різні введені елементи. Інакше кажучи, повторити обидва рази одакове значення можна *лише* якщо це саме значення зустрічалося двічі (або ще більше разів) у вхідних даних.

Якщо можливі різні правильні відповіді — виводьте будь-яку одну з них.

Приклади:	Вхідні дані	Результати	Вхідні дані	Результати
	17 7 42 23	42 23	42 17 42 23	42 42

Оцінювання. 25% балів припадає на тести, де значення кожного з чисел у межах від 1 до 100. Ще 25% балів припадає на тести, де значення кожного з чисел не перевищує 100 за модулем, але числа можуть бути довільних знаків. Ще 25% балів припадає на тести, де значення кожного з чисел у межах від 1 до 10^9 . Решта 25% балів припадає на тести, де значення кожного з чисел не перевищує 10^9 за модулем, але числа можуть бути довільних знаків.

Здавати потрібно одну програму, а не чотири; різні обмеження вказані, щоб пояснити, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі. **Навіщо там від'ємні числа?** Добуток двох від'ємних чисел додатний, так що відповідь може (але не зобов'язана) являти собою два від'ємних числа. наприклад, для четвірки чисел 2, 3, -50, -60 найбільшим можливим добутком є $(-50) \times (-60) = 3000$.

Навіщо вказані діапазони значень? При значеннях до 100, добуток поміщається у майже будь-який числовий тип (крім однобайтових, як-то `byte`), так що можна й не приділяти особливої уваги типу. При значеннях до 10^9 , добуток гарантовано поміщається лише у 64-бітових типах (`Int64` у `Pascal`, `long long` у `C/C++`, тощо). Див. також стор. 13. Втім, після переходу навесні 2019 р. з 32-бітової на 64-бітову архітектуру сервера значна частина розв'язків, які раніше набирали неповний бал саме з цієї причини, почали набирати повний бал. Що, втім, ніяк не заважає тому, щоб аналогічна проблема виникала на якійсь іншій системі автоматичної перевірки...

«Лобовий» повний розв'язок. Можна просто перебрати всі шість варіантів пар і вибрати з них максимальний за звичайними правилами пошуку максимум. Якщо скласти чісла у масив, перебір буде дещо приємнішим та «більш алгоритмічним», див. ideone.com/yQ1Aq0. Але, раз кількість чисел рівно 4, то можна обійтися й без масиву і навіть без циклів, див. ideone.com/VYnuIW.

Альтернативний розв'язок. При виборі пари з конкретно 4-х чисел, цей спосіб не кращий, а просто інший. Можна довести (пропонуємо читачам зробити це доведення самостійно), що максимальний добуток завжди дають або два максимальні числа, або два мінімальні (якщо вони від'ємні й великі за модулем). Іншими словами: якщо відсортувати масив, то відповідь утворять або останній та передостанній елементи, або перший та другий. Якщо мова програмування має готове (бібліотечне) сортування, це досить зручно (див. ideone.com/exmAiY), а за умови ефективного сортування ще й швидше за попередій спосіб (ця відмінність стає помітною при кількостях чисел від кількох десятків тисяч). Можна пробувати й додатково прискорити цей підхід, бо навіть ефективне сортування відбувається повільніше, ніж вибір самих лише максимального, наступного максимального, мінімального та наступного мінімального елементів. Але це громіздко, в ньому легко помилитися, тож писати таке було б варто *лише якби* чисел було дуже багато (сотні тисяч чи ще більше).

Задача С. «Гра “Вгадай число”»

Вхідні дані: Клавiатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 256 мегабайтів

Напишіть програму, яка гратиме у гру «Вгадай число»: визначатиме загадане суперником ціле число із заданого діапазону на основі запитів до суперника.

У кожному запиті Ваша програма повинна виводити слово `try` та одне ціле число, що трактується як запитання «Загадане число ...?». Суперник на це відповідає (гарантовано чесно) одне з трьох: або, що саме це число й загадане, або, що загадане число більше, або, що загадане число менше. Ваша програма повинна продовжувати (або завершити) гру, враховуючи отримані відповіді. Кількість спроб вгадування (незалежно від величини проміжку) — до 50 (п'ятдесяти), включно. При перевищенні цього ліміту, Вашій програмі присуджується технічна поразка (вона не отримує балів за відповідний тест).

Тобто, ця задача є інтерактивною: Ваша програма не отримає всіх вхідних даних на початку, а отримуватиме по мірі виконання доуточнення, котрі залежатимуть від попередніх дій Вашої програми. Тим не менш, її *перевірка теж відбувається автоматично*. Тому, у цій Вашій програмі, як і в програмах-розв'язках інших задач, теж слід не «організовувати діалог інтуїтивно зрозумілим чином», а чітко дотримуватися формату. Тільки це не формат вхідного та вихідного файлів, а формат спілкування з програмою, котра грає роль суперника.

Вхідні дані. На початку, один раз, на вхід Вашій програмі подаються записані в один рядок через пропуск (пробіл) два цілі числа a, b ($-2 \cdot 10^9 \leq a \leq b \leq 2 \cdot 10^9$) — межі проміжку. Це означає, що загадане ціле число гарантовано перебуває в межах $a \leq x \leq b$. На кожному наступному кроці, на вхід Вашій програмі подається рядок, що містить рівно один з трьох символів:

- `=` означає, що останнє виведене Вашою програмою число і є загаданим;
- `+` означає, що загадане число більше, ніж виведене Вашою програмою;
- `-` означає, що загадане число менше, ніж виведене Вашою програмою.

Результати. Поки Ваша програма «не впевнена», яке число загадане, вона повинна повторювати виведення в один рядок через пропуск (пробіл або табуляцію) слова “try” (маленькими латинськими буквами, без лапок) та одного цілого числа — чергової спроби вгадування. Кожне число-спроба повинно бути в межах $a \leq x \leq b$ (див. вище). Настійливо рекомендується після кожного такого виведення робити дію `flush(output)` (Pascal), вона ж `cout.flush()` (C++), вона ж `fflush(stdout)` (C), вона ж `sys.stdout.flush()` (Python), вона ж `System.out.flush()` (Java). Це істотно зменшує ризик, що проміжна відповідь «застряне» десь по дорозі, не дійшовши до програми-суперника.

Коли Вашій програмі «стає абсолютно очевидно», яке число загадане, вона повинна припинити основну частину гри і вивести в один рядок через пропуск (пробіл або табуляцію) слово “answer” (маленькими латинськими буквами, без лапок) та вгадане число. Після цього слід остаточно завершити роботу, не виводячи жодного іншого символу.

Приклад:

Вхідні дані	Результати	Примітки
1 100 + - - - = 	try 10 try 20 try 19 try 18 try 17 answer 17	1) Програма-суперник з самого початку загадала 17. 2) Щоб було видніше, в якому порядку відбуваються введення та виведення, у прикладі використано розділення порожніми рядками. При фактичній перевірці програма-суперник не виводитиме ніяких порожніх рядків і не чекатиме їх від Вашої програми. 3) Ваша програма не зобов'язана грати за стратегією, що наведена у цьому прикладі. Це лише приклад можливої роботи програми, що успішно завершує конкретно цю гру.

Розбір задачі. Перш за все, рекомендуємо перечитати все, що сказано на стор. 9 про те, як неправильні розв'язки інтерактивних задач провокують вердикт “Wall Time Limit Exceeded”. Переходячи ж до суті конкретно цієї задачі, наведена у прикладі «стратегія» фактично такою не є. Хоч за її допомогою і зручно вгадати 17, але як щодо 91? А щодо 987987981?

Насправді це широковідома класична задача, яку слід розв'язувати так званим *бінарним пошуком* (він же *бінпошук*, він же *двійковий пошук*, він же *дихотомія*). Коротко суть описана на стор. 34, детальніше рекомендуємо знайти в Інтернеті або літературі. До речі, на цьому ж сайті ejudge.skipo.edu.ua є змагання 53 «Дорішування теми “Бінарний та тернарний пошуки” Школи Бобра (23.10.2016)», в якому є і теоретичні матеріали, і комплект задач.

Так що наведемо, без детальних пояснень, посилання на готовий розв'язок ideone.com/C5i8Wc та зауваження щодо деяких тонкостей.

Нехай межі проміжку називаються `left` та `right`. Є два поширені способи вибирати середину проміжку `mid`: (1) `mid := (left+right) div 2`; (2) `mid := left + (right-left) div 2`. Для знакових `left` та `right`, *кожен* з них може призвести до переповнень (див. також стор. 13). Тільки $(left + right) \div 2$ переповнюється тоді, коли обидва значення `left` та `right` дуже великі за модулем і одного знаку, а `left + (right - left) div 2` — коли `left` дуже велике за модулем від'ємне, а `right` дуже велике додатне. Так що треба або рахувати у ширшому типі, або робити розгалуження, щоб вибрати ту з формул, яка краща для поточних значень `left` та `right`.

Ще один тонкий момент — уникнути зациклювань. Вони можливі, зокрема (але не тільки) якщо при отриманні від суперника вердикту “+” робиться присвоєння `left:=mid`. Такий алгоритм успішно звужує проміжок від величезного до невеликого, але при, наприклад, `left=3`, `right=4` і загаданому числі 4 одні й ті самі присвоєння `mid := (left+right) div 2 = (3+4) div 2 = 3`, `left:=mid=3` повторюватимуться вічно (ну, або доки програму не завершать за те, що вона за 50 запитів так і не добила вердикту “=”). Конкретно для цієї задачі зручно гарантовано позбутися зациклювань за рахунок того, що при вердикті “+” робити `left:=mid+1`, а при “-” робити `right:=mid-1`. Тобто, зменшувати проміжок одночасно і за рахунок присвоєння межі значення середини, і за рахунок виключення самої цієї середини, яка теж не є шуканою відповіддю (бо не отримала вердикту “=”). Але, на жаль, при багатьох інших застосуваннях бінпошуку така ідея просто неправильна.

Ще одна тонкість — якщо сума `left+right` одночасно і непарна, і від'ємна, вищезгадані формули `mid := (left+right) div 2` та `mid := left + (right-left) div 2` не взаємозамінні навіть при відсутності переповнень. Перша з них заокруглює до нуля, друга — у менший бік. Зокрема, якщо

при вердикті “+” робити `left:=mid+1`, а при “-” робити `right:=mid` (без «-1»), то перша з них може зациклити алгоритм, а друга не містить такого ризику.

Задача D. «Кількість прямокутників»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

На перерві Андрійку нема чого робити, друзі пішли за шаурмою, а дівчинка, яка йому подобається, пішла гуляти з іншими хлопчиками. Однак щоб не сумувати він вирішив сумувати, тобто рахувати. Він хоче знайти кількість різних прямокутників на своєму листочку в клітинку.

Вхідні дані. В єдиному рядку задано 2 числа — кількість клітинок на листочку по ширині та висоті. Кожен з розмірів завжди не менший 1, обмеження на максимальний розмір наведені далі.

Результати. Виведіть кількість різних прямокутників.

Приклади:

Вхідні дані	Результати
2 2	9
3 2	18

Повний перелік усіх прямокутників тесту №1



Оцінювання. 20% балів припадатиме на тести, в яких обидва розміри не перевищують 4. Ще 30% балів припадатиме на тести, в яких більший з розмірів у межах від 17 до 42. Ще 20% балів припадатиме на тести, в яких більший з розмірів у межах від 12345 до 54321. Решта 30% балів припадатиме на тести, в яких більший з розмірів у межах від 10^7 (десяти мільйонів) до 10^9 (мільярда).

Здавати потрібно одну програму, а не чотири; різні обмеження вказані, щоб пояснити, скільки балів можна отримати, розв'язавши задачу не повністю.

Розбір задачі. Смысл окремого блоку 1 (розміри до 4) хіба що в тому, що для нього можна порахувати все вручну і понаписувати у програму готові відповіді (вибираючи потрібну if-ами). Якщо ж робити реальний алгоритмічний розв'язок, блок 2 (розміри до 42) нічим не складніший за блок 1.

Аналізуючи наведений у прикладі повний перелік усіх прямокутників, можна побачити, що у перших трьох використовується діапазон y -координат від 0 до 1, у наступних трьох від 0 до 2 і в останніх трьох від 1 до 2. Аналогічно, всередині кожної з цих трійок перший прямокутник має діапазон x -координат від 0 до 1, другий (кожної трійки) — від 0 до 2, третій — від 1 до 2.

Розв'язок для блоків 1–2. Можна продовжити аналіз того ж переліку, і помітити, що «від 0 до 1» та «від 0 до 2» являють собою початок 0 і всі можливі (обидва) кінці, «від 1 до 2» — початок 1 і єдиний можливий кінець. Тобто, всі можливі діапазони окремо взятої координати можна перебрати двома вкладеними циклами, де зовнішній перебирає можливі початки, внутрішній — можливі кінці. Неважко переконатися, що це підтверджується також і при більших розмірах листочка.

Що можна перетворити у перебір усіх можливих прямокутників чотирма вкладеними циклами, наведеними праворуч: перші два цикли перебирають початки та кінці діапазонів y -координати, наступні (вкладені) два — аналогічно для x -координати.

```
read(xsz, ysz);
res:=0;
for ymin:=0 to ysz-1 do
  for ymax:=ymin+1 to ysz do
    for xmin:=0 to xsz-1 do
      for xmax:=xmin+1 to xsz do
        res:=res+1;
```

Тільки цей спосіб, хоч і правильний, не ефективний. Такі цикли працюють надто довго не лише для блоку 4 (розміри до 10^9), але і для блоку 3 (розміри до 54321). Причому, питають-то лише кількість, а не самі прямокутники! Отже, насправді не варто будувати (перебирати) кожен прямокутник окремо.

Аналіз засобами комбінаторики. Важливе спостереження №1: раз кожен можливий діапазон y -координат можна поєднувати з кожним можливим діапазоном x -координат — достатньо порахувати кількості можливих діапазонів для кожної з координат окремо, й перемножити ці кількості. Це просто-таки еталонний приклад так званого *правила добутку* комбінаторики.

Важливе спостереження №2: діапазони однієї окремо взятої координати є парами чисел від 0 до максимального значення (розміру за цією координатою). Причому, це пари різних елементів (при однакових, прямокутник вироджується у відрізок чи точку, а з прикладу видно, що це не рахується);

причому, ці пари не впорядковані (обмін місцями, як-то $(0;2) \leftrightarrow (2;0)$, не утворює нового діапазону). Отже, ці пари являють собою сполучення (вони ж сполуки, вони ж комбінації) без повторень по 2, і їхню кількість можна знайти за формулою $C_{s+1}^2 = \frac{(s+1)(s+1-1)}{1 \cdot 2} = \frac{(s+1)s}{2}$ (де s — розмір за відповідною координатою; $s+1$, а не s , бо до можливих значень координати включаються і 0, і s).

Отже, відповіддю задачі є $\frac{x(x+1)y(y+1)}{4}$ (де x та y — вхідні дані).

А в чому тоді відмінність між блоками тестів 3 та 4? Частково — для блоку 3 ще проходять деякі розв'язки, де частина з цих спостережень зроблена, а частина ні (наприклад, позбулися вкладених циклів, але, не зумівши перетворити у пряму аналітичну формулу, один цикл лишили).

Але це не все. Підставивши у формулу максимальні значення $x = y = 10^9$, легко бачити, що при них відповідь перевищуватиме $\frac{1}{4} \cdot 10^{36}$, що за межами стандартних типів даних. Так що для отримання повних балів треба використати т. зв. *довгу арифметику* — подавати чісла, наприклад, масивами цифр і виконувати арифметичні дії не стандартними процесорними командами, а виконанням алгоритмів, подібних до множення у стовпчик. Правда, тут не обов'язково писати повноцінну довгу арифметку: окремо взяті $\frac{x(x+1)}{2}$ та $\frac{y(y+1)}{2}$ цілком можна обчислити у стандартному 64-бітовому типі, тож лишається тільки перетворити один з цих добутків у довге число й помножити його на 64-бітове (єдина дія довгої арифметики). Приклад саме такої програми (мовою Pascal) див. ideone.com/ug49QD.

При бажанні можна обійтися без власноручного написання навіть і окремих операцій довгої арифметики. У переліку мов програмування є Python, в якому довга арифметика вбудована і викликається автоматично, а також Java, де довга арифметика не настільки зручна, але є (клас BigInteger). (C# теж має клас BigInteger, але там усе сумніше, бо засоби, якими C#-BigInteger підключається на локальному Windows-комп'ютері, не працюють на Linux-сервері ejudge, і авторам збірника невідомо ні як їх підключити там, ні, навіть, чи можливо це взагалі.)

Розв'язок мовою Python наведено праворуч. Це повний текст повнобального розв'язку всієї задачі. Разів так у 15 коротший, ніж мовою Pascal.

```
x, y = map(int, input().strip().split())
print((x*(x+1)*y*(y+1))/4)
```

Можна заявити, ніби це порушує права учасників, що пишуть іншими мовами. Але: (1) Ніхто нікому не забороняв вивчити і використати Python. Хто більше знає — має більше шансів на перемогу, що тут несправедливого? (2) Довгу арифметику *можна* написати *будь-якою* з дозволених мов програмування (зокрема, див. посилання вище). Питання лише в тому, чи це зручно й легко, чи дещо довше й важче. Ситуації, коли мовою C++ можна використати бібліотечну функцію, а мовою Pascal доводиться реалізовувати алгоритм самому, вже згадувалися у цьому збірнику дуже багато разів; ну, а тут виграв має інша мова. (3) Довга арифметика потрібна *лише* у останньому 4-му блоці тестів.

Мова Python *не* є універсально-найкращою для олімпіад. Просто розглянута така задача, що для неї Python має велику перевагу. У багатьох інших ситуаціях Python програє (зокрема, тому, що програми мовою Python виконуються помітно повільніше, ніж мовами C/C++ чи Pascal). Цікавою є, зокрема, ситуація з задачею «Остання ненульова цифра» (стор. 90–93), в якій водночас і суперзручна довга арифметика мови Python дозволяє легко отримати 60% балів за те, що іншими мовами набирає 20–34% балів, і набрати мовою Python більше, ніж 80% балів, значно важче, ніж іншими мовами.

3.12 II (районний/міський) етап 2016/17 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання №56).

Задача А. «Квартали»

Місто з квадратними кварталами має вигляд прямокутника, розмір якого зі сходу на захід n кварталів, а з півночі на південь — m кварталів. Як між кварталами, так і по краям міста прокладені вулиці ширини a . Самі квартали (без вулиць) мають розмір $b \times b$.

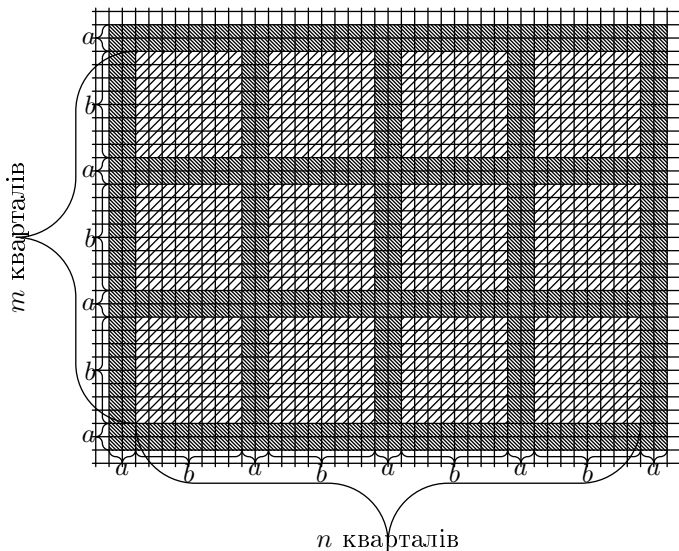
Напишіть *вираз*, залежний від a , b , n , m , за яким можна обчислити сумарну площу всіх доріг. Правила запису виразу спільні для більшості мов програмування: дозволені десяткові цілі числа, операції $+$ (плюс), $-$ (мінус), $*$ (помножити), круглі дужки для задання порядку дій; множення треба писати явно (тобто, не можна писати добуток як mn , лише як $m*n$).

Здати треба вираз, а не програму. В *ejudge* треба вписати у відповідне поле сам вираз, не створюючи ніякого файлу-розв'язку. Змінні a , b , n , m з умови задачі повинні називатися саме a , b , n , m , перейменовувати не можна.

Приклад: на рисунку зображено місто, де $a=2$, $b=8$, $n=4$, $m=3$; для такого міста, шукана загальна площа доріг дорівнює 576.

Розбір задачі. Найлегше, мабуть, відняти з площі всього міста (доріг та кварталів) площу кварталів. Розмір міста «по горизонталі» (зі сходу на захід) становить $(n+1)*a + n*b$, бо кварталів n , вулиць $(n+1)$ (між усіма сусідніми кварталами та по обидва боки). Аналогічно, розмір «по вертикалі» становить $(m+1)*a + m*b$. Кварталів же $m*n$ штук, кожен площею $b*b$. Що дає формулу " $((n+1)*a + n*b) * ((m+1)*a + m*b) - n*m*b*b$ ".

Є й інші відповіді, як-то " $n*m*a*b*2 + (n+m)*a*b + (n+1)*(m+1)*a*a$ " чи " $(n*(a+b)+a) * (m*(a+b)+a) - n*m*b*b$ ". Зараховується будь-яка правильна.



Задача В. «Квадрат і круг»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 128 мегабайтів

Є круг радіуса R та квадрат зі стороною a . Напишіть програму, яка визначатиме, яка з трьох ситуацій має місце:

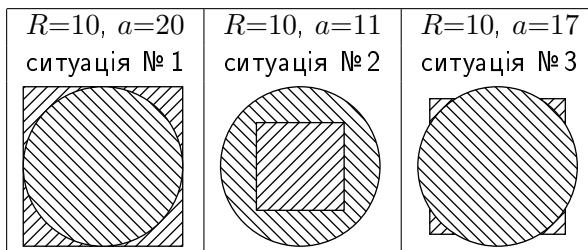
1. круг можна покласти на квадрат, і круг не звисатиме за межі квадрату;
2. квадрат можна покласти на круг, і квадрат не звисатиме за межі круга;
3. як не клади, верхня фігура звисатиме за межі нижньої.

Вхідні дані. Перший рядок містить єдине число R — радіус круга. Другий рядок містить єдине число a — сторону квадрата. Кожен з розмірів є цілим числом у межах від 1 до 12345.

Результати. Виведіть єдине число 1, 2 або 3 на позначення того, яка з ситуацій (згідно з вищезгаданим переліком) має місце.

Приклади:

Вхідні дані	Результати
10 20	1
10 11	2
10 17	3



Розбір задачі. Враховуючи приклад № 1 (з умови), очевидно, що ситуація № 1 має місце тоді й тільки тоді, коли сторона квадрата більша-або-рівна подвоєного радіуса (діаметра) круга.

Лишилося проаналізувати аналогічний «переломний момент», що розділяє ситуації № 2 та № 3; він зображений на рис. праворуч. Бачимо прямокутний трикутник, катети якого є одночасно радіусами і половинками діагоналей квадрата, гіпотенуза — стороною квадрата. Отже, сам «переломний момент» має місце при $R^2 + R^2 = a^2$, або $a = R \cdot \sqrt{2}$, а ситуація № 2 — при $a \leq R \cdot \sqrt{2}$.

Що може бути перетворено у, наприклад, фрагмент коду, наведений праворуч. Звісно, можуть бути й інші правильні перевірки. Можна замінити $a \leq r \cdot \sqrt{2}$ на $a * a \leq r * r * 2$, але тоді стає важливим не допустити переповнень (які можуть виникнути, а можуть і не виникнути, для типу integer при перевірці під fpc; див. також стор. 13 та 11). Писати замість виразу $\sqrt{2}$ значення, як-то 1.41421356 — не найкраща ідея, бо написати багато правильних знаків важче, ніж $\sqrt{2}$, а пишучи мало знаків (наприклад, 1.414) можна отримати неправильну відповідь саме через цю неточність.

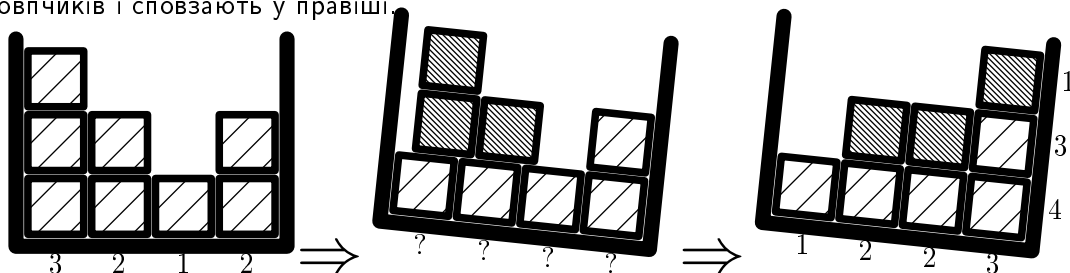
```
read(r, a);  
if a >= r*2 then  
  writeln(1)  
else if a <= r*sqrt(2) then  
  writeln(2)  
else  
  writeln(3)
```

Задача С. «Сповзання кубиків»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 128 мегабайтів

Є N стовпчиків, утворених кубиками: 1-й стовпчик являє собою a_1 поставлених один на іншого кубиків, 2-й — a_2 кубиків, тощо. Усі ці кубики однакові.

Все це разом узятє акуратно нахилиють праворуч — так, що деякі з кубиків зісковзують зі своїх стовпчиків і сповзають у правіші.



Вважаємо (хоч це й не зовсім відповідає реальним законам фізики), що при сповзанні кубики ніколи не летять, перекидаючись, а лише зміщуються на правіші позиції.

Напишіть програму, яка за початковою конфігурацією (а саме, кількостями кубиків у кожному стовпчику) знаходитиме кінцеву (а саме, кількості кубиків у кожному стовпчику та кількості кубиків у кожному рядку).

Вхідні дані. 1-й рядок містить єдине число N ($2 \leq N \leq 123456$) — кількість стовпчиків. 2-й рядок містить (розділені пробілами) N натуральних чисел a_1, a_2, \dots, a_N — кількості кубиків 1-му, 2-му, ..., N -му стовпчиках (зліва направо). Кожна з цих кількостей перебуває у межах $1 \leq a_j \leq 123456$.

Результати. Виведіть у першому рядку розділені пробілами кінцеві кількості кубиків по стовпчикам (зліва направо), а у другому рядку — кінцеві кількості кубиків по рядках (усім непорожнім, знизу догори). Ні кількість стовпчиків, ні кількість рядків виводити не треба.

Приклади:	Вхідні дані	Результати	Вхідні дані	Результати
	4 3 2 1 2	1 2 2 3 4 3 1	7 4 1 1 1 2 1 1	1 1 1 1 1 2 4 7 2 1 1

Примітка. 1-й приклад входу/виходу відповідає наведеним рисункам.

Оцінювання. 20% балів припадає на тести, в яких кількість стовпчиків $N = 4$, максимальне серед усіх a_j дорівнює 3. Ще 20% балів — тести, де $2 \leq N \leq 123$, $1 \leq \max(a_j) \leq 123$; ще 20% балів — тести, де $2 \leq N \leq 123$, $1 \leq \max(a_j) \leq 123456$; ще 20% балів — тести, де $2 \leq N \leq 123456$, $1 \leq \max(a_j) \leq 123$; решта 20% балів — тести, де $2 \leq N \leq 123456$, $1 \leq \max(a_j) \leq 123456$.

Писати треба одну програму, а не різні програми для різних випадків; єдина мета цього переліку різних блоків обмежень — дати уявлення про те, скільки балів можна отримати, якщо розв'язати задачу правильно, але не ефективно.

Якщо якийсь один з рядків-відповідей (або увесь перший рядок відповіді, тобто кількості кубиків по стовпчикам, або увесь другий рядок відповіді, тобто кількості кубиків по рядкам) повністю правильний, а інший ні, програма отримуватиме за відповідний тест 40% балів.

Розбір задачі. Знаходження 1-ї відповіді. З «при сповзанні кубики ніколи не летять, перекидаючись» *впливає*, що вони лише зміщуються на правіші позиції, тобто ніякий кубик не міняє своєї висоти. Кубикам, що лежать на дні, сповзати нема куди (з $1 \leq a_j$ слідує, що увесь нижній ряд зайнятий). Будь-який інший кубик на початку лежав на нижчому, а не висів у повітрі; от і при нахилі він або сповзає разом із нижчим, на якому лежить, або, якщо той нижчий впирається у правого сусіда, а сам поточний ні, то поточний зсувається праворуч, на нового нижчого, теж не змінюючи своєї висоти. (Ситуація, що поточний впирається у правого сусіда, а нижчий ні, неможлива, бо для цього треба, щоб кубик праворуч до того вже висів у повітрі, що неможливо.)

Звідси, для простого часткового випадку, коли стовпчиків лише два і $a_1 > a_2$ (на початку 1-й стовпчик був вищим за 2-й), верхні $a_1 - a_2$ кубиків сповзають з 1-го на 2-й, після чого вже 1-й стовпчик виявляється нижчим (тієї ж висоти a_2 , якої раніше був 2-й), а 2-й вищим (висоти a_1). Якщо ж на початку $a_1 \leq a_2$, взагалі нічого не змінюється. В обох випадках, *перший рядок відповіді являє собою відсортовану послідовність висот зі вхідних даних*.

Якщо стовпчиків не два, а більше, то все одно кубики або не рухаються відносно дна, або зсуваються послідовно на наразі сусідній стовпчик (можливо, багатократно, але кожен з цих багатьох разів — на сусідній), і завжди призводить до того, що в кожній парі сусідніх стовпчиків висоти чи то лишаються які були, чи то обмінюються місцями (більша йде праворуч, менша ліворуч). Так що всі сповзання разом узяті, хоч і можуть відбуватися в іншому порядку, ніж сортування, в результаті дають відсортовану послідовність висот.

Отже, 1-й рядок відповіді зручно отримати, не моделюючи весь процес сповзань, а застосувавши ефективний алгоритм сортування (див. далі).

Знаходження 2-ї відповіді. Щоб не плутати вже відсортовану (для 1-ї відповіді) послідовність висот із початковою, будемо в цьому тексті називати відсортовану послідовність h_1, h_2, \dots, h_N (хоча технічно це той самий масив у різні моменти часу). Усі рядки з № 1 по № h_1 заповнені повністю (бо h_1 — *мінімальна* з усіх висот), тобто можна вивести у 2-й рядок відповіді h_1 штук значень N (де N — загальна кількість стовпчиків). Потім $h_2 - h_1$ рядків містять $N - 1$ кубиків, причому це так і при $h_2 > h_1$ (тоді у кожному з рядків від № $(h_1 + 1)$ до № h_2 включно після зсувів рівно один крайній лівий стовпчик вільний, а решта зайняті, бо рівно один крайній лівий стовпчик має строго меншу висоту, а решта більшу-або-рівну), і при $h_2 = h_1$ (тоді у відповідь не потрапляє жодне значення $N - 1$, і це теж правильно, бо тоді на всіх висотах по h_1 включно по N кубиків, а починаючи з висоти $h_1 + 1$ строго менше ніж $N - 1$ кубик). Аналогічне міркування можна повторити багатократно й отримати, що далі є рівно $h_3 - h_2$ значень $N - 2$, рівно $h_4 - h_3$ значень $N - 3$, \dots , рівно $h_N - h_{N-1}$ значень 1.

Асимптотична складність визначається, в основному, використаним сортуванням. Бульбашка, вибір та інші прості методи сортування мають складність $O(N^2)$ і не мають шансів укластися в обмеження часу на всіх тестах. Сортування злиттям, сортування Хоара (QuickSort) та пірамідальне сортування мають складність $\Theta(N \log N)$ що при $N \leq 123456$ цілком прийнятно. (Хто не знає цих алгоритмів — знайдіть у літературі або в Інтернеті; сортування Хоара, хоч і називається QuickSort, насправді має складність $\Theta(N \log N)$ не завжди, а тільки у більшості випадків, іноді погіршуючись аж до N^2 ; ця задача не містить спеціальних анти-quicksort-івських тестів, тому саме тут це не важливо, але для розуміння загальної картини варто це знати.)

Можна й скористатися бібліотечним сортуванням, якщо таке є (функція `sort` бібліотеки `algorithm` мови C++, або метод `Arrays.sort` мови Java, або метод `sort` пітонівського `list`-а, тощо), не розбираючись, як це влаштовано всередині. Так робити не заборонено. Інша справа, що знання про те, як насправді влаштовані ці підпрограми, можуть знадобитися в якихось інших ситуаціях.

Альтернативний розв'язок: асимптотика $\Theta(N + \max a_j)$ та отримання 2-ї відповіді навіть без 1-ї. В цій задачі можна застосувати *сортування підрахунком* (*counting sort*). Його суть така: заводиться масив *push*, *індекси* якого відповідають *висотам*, а значення — *кількості* стовпчиків

відповідної висоти (технічно — усі елементи ініціалізуються нулями, а при читанні вхідних даних, замість `read(a[i])`, робляться дії `read(a); num[a] := num[a] + 1`).

1-а відповідь (саме сортування підрахунком) формується як «`num[1]` штук одиниць, потім `num[2]` штук двійок, тощо» (до речі, це дещо схоже на формування 2-го рядка відповіді раніше розглянутим способом, так що знання сортування підрахунком може бути корисним як для того, щоб використати його тут, так і для придумування вищезгаданого знаходження 2-ї відповіді 1-м способом).

А 2-у відповідь можна отримати так: сформувати `num_gr_eq[max_a_j] := num[max_a_j]`; на основі масиву `num` масив `num_gr_eq`, тобто «кількість `for j := max_a_j - 1 downto 1 do` більших-або-рівних» (наприклад, способом, показаним у `num_gr_eq[j] := num_gr_eq[j+1] + num[j]`; кодї праворуч), і просто вивести усі елементи `num_gr_eq` чи то до кінця, чи то доки вони ненульові.

Легко бачити, що асимптотика такого підходу $\Theta(N + \max a_j)$, що формально краще за $\Theta(N \log N)$. Втім, на практиці різниця між ними невелика й у значній мірі нівелюється часом читання вхідних даних. Так що питання більше в тому, що зручніше писати. Якщо є зручне готове бібліотечне сортування — мабуть, краще використати його і знаходження 2-ї відповіді через 1-у.

Задача D. «Кратний куб»

Вхідні дані: Або клавіатура, або `input.txt` Обмеження часу: 1 сек
Результати: Або екран, або `output.txt` Обмеження пам'яті: 128 мегабайтів

Напишіть програму, яка для заданого натурального числа k знаходитиме, куб якого найменшого натурального числа кратний цьому k .

Вхідні дані. Єдине число k .

Результати. Виведіть єдине число — мінімальне натуральне (ціле строго додатне), куб (третя степінь) якого ділиться націло (без остачі) на k .

Приклад:

Вхідні дані	Результати
12	6

Примітка. Ні $1^3=1$, ні $2^3=8$, ні $3^3=27$, ні $4^3=64$, ні $5^3=125$ не діляться на 12 націло, а $6^3=216$ ділиться. Тобто, 6^3 є найменшим натуральним кубом, кратним 12.

Оцінювання. 20% балів припадає на тести, в яких $2 \leq k \leq 100$; ще 20% балів — на $10^5 < k < 10^6$; ще 20% балів — на $10^7 < k < 10^8$; ще 20% балів — на $10^{10} < k < 10^{12}$; решта 20% балів — на $10^{15} < k < 10^{18}$.

Писати треба одну програму, а не різні програми для різних випадків; єдина мета цього перебіку різних блоків обмежень — дати уявлення про те, скільки балів можна отримати, якщо розв'язати задачу правильно, але не ефективно.

Розбір задачі. «Лобовий» розв'язок (перебирати послідовно $1*1*1$, $2*2*2$, $3*3*3$, тощо, аж доки не поділиться на введене k) благополучно і гарантовано проходить перші два блоки тестів, але у наступних блоках — лише деякі окремі тести, бо для більших k цей підхід має відразу дві проблеми:

1. Значення куба може не поміщатися у стандартні типи даних, наприклад при використанні 64-бітового беззнакового цілого (QWord (Pascal), `unsigned long long` (C/C++)) — починаючи з $2642246^3 = 18446745128696702936 \geq 18446744073709551616 = 2^{64}$. (Якщо взяти менший тип, як-то 32-бітовий, межа буде переходитися раніше, й проблеми виникнуть вже у 2-му блоці тестів.)
2. Навіть якби перша проблема була вирішена (чи то використанням мови Java або Python, що мають вбудовану довгу арифметику, чи то написанням довгої арифметики вручну), такий перебір не помістився б у обмеження часу, причому для деяких з тестів — у мільярди чи трильйони разів.

Повний розв'язок. Якби провести факторизацію (розкладення числа на прості множники), можна було б по кожному степеню простого множника $p_j^{k_j}$ сказати, що відповідь повинна містити цей простий множник $\left\lceil \frac{k_j}{3} \right\rceil$ разів (де $\lceil x \rceil$ — заокруглення вгору, воно ж стеля; наприклад, $\left\lceil \frac{1}{3} \right\rceil = \left\lceil \frac{2}{3} \right\rceil = \left\lceil \frac{3}{3} \right\rceil = 1$, $\left\lceil \frac{4}{3} \right\rceil = \left\lceil \frac{5}{3} \right\rceil = \left\lceil \frac{6}{3} \right\rceil = 2$, тощо). Наприклад, $2016 = 2^5 \times 3^2 \times 7^1$, значить 2 треба брати у степені $\left\lceil \frac{5}{3} \right\rceil = 2$, 3 у степені $\left\lceil \frac{2}{3} \right\rceil = 1$ і 7 у $\left\lceil \frac{1}{3} \right\rceil = 1$, тож відповіддю для 2016 є $2^2 \cdot 3^1 \cdot 7^1 = 84$. Технічно

для заокруглення вгору у багатьох мовах програмування є бібліотечна функція `ceil` (чи `Ceiling`); інший спосіб — виразити $\left\lceil \frac{k_j}{3} \right\rceil$ як $(k_j+2) \operatorname{div} 3$ (де div — цілочисельне ділення, як-то $17 \operatorname{div} 3 = 5$).

Як робити факторизацію, написано у багатьох джерелах, зокрема — теорії до змагання №49 «День Іллі Порубльова “Школи Бобра” (2015, деякі теми теорії чисел)» цього ж сайту `ejudge.skipo.edu.ua` (зайти у змагання №49 і у повідомленнях журі знайти посилання на розбір). Але проблема в тім, що «чесна» факторизація числа N працює за час $O(\sqrt{N})$, що при $N \sim 10^{18}$ забагато.

Тут треба помітити, що увесь проміжок від $\sqrt[3]{N}$ до \sqrt{N} перебирається з єдиною метою: розрізнити ситуацію «число N є простим» від ситуації «число N є добутком двох простих, не дуже далеких від \sqrt{N} ». А з точки зору саме цієї задачі їх можна і не розрізняти: хоч $z = p^1$, хоч $z = p_1^1 \times p_2^1$ однаково означають, що у відповідь входить саме число z , яке досліджуємо. Так що верхню межу перебору можна знизити з \sqrt{N} до $\sqrt[3]{N}$, а $\sqrt[3]{10^{18}} = 10^6$ для комп'ютера небагато. Звісно, треба окремо перевірити, чи не є поточне число точним квадратом, бо спостереження щодо «хоч $z = p^1$, хоч $z = p_1^1 \times p_2^1$ » справедливе лише при $p_1 \neq p_2$.

III (обласного) етапу 2016/17 н. р. нема

У 2016/17 навч. році III (обласний) етап у Черкаській області відбувався у Черкасах, але на задачах інших авторів, іншій системі (`ejudge`, але не `ejudge.skipo.edu.ua`), та й про його дорішування авторам цього збірника нічого не відомо. Тому, він до збірника не включений.

3.13 Обласна інтернет-олімпіада 2017/18 н. р.

Задачі доступні для дорішування (`ejudge.skipo.edu.ua`, змагання №21).

Задача А. «Патрульні групи–1»

Патрульна група повинна складатися з одного командира та рівно двох рядових. Загальна кількість офіцерів, які можуть виконувати обов'язки командира патрульної групи, становить s . Загальна кількість рядових, з яких можуть вибиратися підлеглі патрульні, становить r . Офіцер не може виконувати обов'язки рядового, а рядовий не може виконувати обов'язки командира групи.

Напишіть *вираз*, котрий знаходитиме, яку максимальну кількість *патрульних груп* можна сформувати *одночасно*.

Кількість офіцерів, які можуть виконувати обов'язки командира патрульної групи, повинна позначатися обов'язково змінною k . Загальна кількість рядових, з яких можна вибирати підлеглих патрульних, повинна позначатися обов'язково змінною r . Обидві кількості гарантовано натуральні (цілі додатні).

У цій задачі треба здати не програму, а вираз: вписати його (сам вираз, не назву файлу) у відповідне поле перевіряючої системи і відправити на перевірку. Правила запису виразу: можна використовувати десяткові числа, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), “/” (ділення дробове, наприклад, $17/5=3,4$), “//” (ділення цілочисельне, наприклад, $17//5=3$), круглі дужки “(” та “)” для групування та зміни порядку дій, а також функції `min` та `max`. Формат запису функцій: після `min` або `max` відкривна кругла дужка, потім перелік (через кому) виразів, від яких береться мінімум або максимум, потім закривна кругла дужка. Дозволяються пропуски (пробіли), але, звісно, не всередині чисел і не всередині імен `min` та `max`.

Наприклад: можна здати вираз r/k і отримати 4 бали зі 100, бо він неправильний, але все ж іноді відповідь випадково збігається з правильною. А «цілком аналогічний» вираз b/a буде оцінений на 0 балів, бо змінні повинні називатися k та r , а не a та b .

Для кращого розуміння умови задачі та суті її відмінності від наступної, наведемо також приклад: при $k=3$, $r=3$ числова відповідь дорівнює 1, бо хоч і є аж три можливих командири, але трьох рядових не вистачає на формування навіть двох груп по двоє рядових у кожній.

Розбір задачі. На жаль, в умові була допущена помилка: кількість командирів (офіцерів) у одному місці називається s , в інших k . На щастя, цю помилку було досить швидко помічено, причому її вдалося швидко нівелювати шляхом того, щоб вважати правильними і розв'язки, де ця змінна називається k , і розв'язки, де вона називається s . Це саме стосується й наступної задачі.

З останнього абзацу умови та здорового глузду легко побачити, що треба подивитися, скільки груп можна сформувати, беручи до уваги лише обмеження за кількістю командирів, скільки — лише за кількістю рядових, і взяти мінімум. Тобто, виходить $\min(k, r/2)$, бо всі k офіцерів цілком можуть одночасно взяти по групі, а з r рядових гарантовано можна сформувати $r/2$ груп, і ніяк не можна сформувати (одночасно!) ще більше.

Задача В. «Патрульні групи–2»

Патрульна група повинна складатися з одного командира та рівно двох рядових. Загальна кількість офіцерів, які можуть виконувати обов'язки командира патрульної групи, становить s . Загальна кількість рядових, з яких можуть вибиратися підлеглі патрульні, становить r . Офіцер не може виконувати обов'язки рядового, а рядовий не може виконувати обов'язки командира групи.

Напишіть *вираз*, котрий знаходитиме *кількість способів* сформувати *одну* патрульну групу.

Кількість офіцерів, які можуть виконувати обов'язки командира патрульної групи, повинна позначатися обов'язково змінною k . Загальна кількість рядових, з яких можна вибирати підлеглих патрульних, повинна позначатися обов'язково змінною r . Обидві кількості гарантовано натуральні (цілі додатні).

У цій задачі треба здати не програму, а вираз: вписати його (сам вираз, не назву файлу) у відповідне поле перевіряючої системи і відправити на перевірку. Правила запису виразу: можна використовувати десяткові чісла, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), “/” (ділення дробове, наприклад, $17/5=3,4$), “//” (ділення цілочисельне, наприклад, $17//5=3$), круглі дужки “(” та “)” для групування та зміни порядку дій, а також функції \min та \max . Формат запису функцій: після \min або \max відкривна кругла дужка, потім перелік (через кому) виразів, від яких береться мінімум або максимум, потім закривна кругла дужка. Дозволяються пропуски (пробіли), але, звісно, не всередині чисел і не всередині імен \min та \max .

Наприклад: можна здати вираз r/k і отримати 2 бали зі 100, бо він неправильний, але все ж іноді відповідь випадково збігається з правильною. А «цілком аналогічний» вираз b/a буде оцінений на 0 балів, бо змінні повинні називатися k та r , а не a та b .

Для кращого розуміння умови задачі та суті її відмінності від попередньої, наведемо також приклад: при $k=3$, $r=3$ числова відповідь дорівнює 9, і цими дев'ятьма способами сформувати одну групу є: $(K1, P1, P2)$, $(K1, P1, P3)$, $(K1, P2, P3)$, $(K2, P1, P2)$, $(K2, P1, P3)$, $(K2, P2, P3)$, $(K3, P1, P2)$, $(K3, P1, P3)$, $(K3, P2, P3)$ (де $K1$ — 1-й командир, $P1$ — 1-й рядовий, $P2$ — 2-й рядовий, тощо).

Розбір задачі. Щодо позначення кількості офіцерів, див. 1-й абзац розбору задачі А.

Знаючи комбінаторику, легко побачити, що відповіддю задачі є $k \cdot C_r^2$, що можна перетворити до, наприклад, вигляду $k \cdot r \cdot (r-1) / 2$.

Пояснимо, звідки це береться. З прикладу видно, що будь-хто з офіцерів $K1, K2, K3$ може командувати будь-якою групою рядових $((P1, P2), (P1, P3), (P2, P3))$. Навіть не знаючи, що це і є комбінаторне правило добутку, очевидно, що раз будь-якого офіцера можна призначити на будь-яку групу рядових, то треба помножити кількість офіцерів k на кількість можливих груп рядових. З цією кількістю трохи складніше, але не набагато. Кожного з r рядових можна поєднувати в групу з будь-яким з *решти* $r-1$ рядових (ну, не поєднувати ж з самим собою...). Звідси $r \cdot (r-1)$. Але цей добуток рахує пари, де рядові переставлені місцями (як-то $(P1, P2)$ та $(P2, P1)$) як різні, й це суперечить прикладу. Значить, треба компенсувати те, що кожна пара врахована двічі, поділивши на 2.

Задача С. «День програміста»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

День програміста припадає на 256-й день року, у невисокосний рік це 13 вересня, а у високосний — 12.

Аналогічно пропонується розпізнати число та номер місяця, що припадає на будь-який день за номером n у не високосному 2017 році.

Вхідні дані. Єдине число від 1 до 365 — номер дня у році.

Результати. Два числа в один рядок через пропуск — відповідні день та місяць.

Приклад:

Вхідні дані	Результати
256	13 9

Розбір задачі. Тут зручно завести масив з кількостями днів у місяцях {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31} і повторювати цикл «поки глобальний номер дня більший за кількість днів у поточному місяці, відняти з номера дня кількість днів у місяці й перейти до наступного місяця». Наприклад, для 100-го дня року буде так: $100 > 31$, тож віднімаємо $100 - 31 = 69$ і переходимо до наступного місяця (лютого); $69 > 28$, тож $69 - 28 = 41$ і переходимо до березня; $41 > 31$, тож $41 - 31 = 10$ і переходимо до квітня; $10 \leq 30$, отже цей день — 10 квітня. Приклад реалізації цієї ідеї — ideone.com/OKOghK.

Можна також написати програму, яка значно активніше використовує деяку бібліотеку роботи з календарем; див. ideone.com/b370Jg та міркування щодо такого підходу в розборах задач «Високосні роки» (стор. 53–54) та «П'ять неділь у місяці» (стор. 55–57). Втім, саме в цій задачі користь від бібліотечних засобів сумнівна: неясно, що легше, чи написати самому, чи розбиратися з неочевидними деталями бібліотечних засобів.

Задача D. «Відстань між мінімумом та максимумом–1»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 2 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Напишіть програму, яка за заданим масивом цілих чисел знайде місце мінімального її елемента, максимального її елемента, та відстань (кількість проміжних елементів) між ними. У випадку, якщо масив містить однакові мінімальні та/або однакові максимальні елементи на різних позиціях, слід вибрати перший з мінімумів та останній з максимумів.

Вхідні дані. Перший рядок містить єдине число N ($2 \leq N \leq 100$) — кількість елементів масиву. Другий рядок містить (розділені пробілами) числа–елементи масиву; значення цих чисел–елементів цілі, від 1 до 100.

Результати. Виведіть єдине ціле число — відстань (кількість проміжних елементів) між першим з мінімальних і останнім з максимальних елементів.

Приклади:

Вхідні дані	Результати	Примітки
8 3 9 7 5 2 8 6 4	2	Між елементами 2 та 9 два проміжні елементи 5 та 7
8 4 3 2 1 8 7 6 5	0	Між елементами 1 та 8 нема проміжних елементів
8 3 1 4 1 5 9 2 6	3	Між першим з елементів 1 та єдиним (і тому останнім) елементом 9 три проміжні елементи 4, 1, 5
8 1 1 1 1 1 1 1 1	6	Оскільки всі елементи однакові, кожен з них мінімальний і максимальний; отже, першим мінімальним є перший з усіх елементів масиву, останнім максимальним -- останній з усіх

Розбір задачі. Тут досить акуратно внести дрібні зміни у стандартні («шкільні») алгоритми пошуку мінімуму й максимуму.

Щоб знаходити *перший* мінімум та *останній* максимум, можна або запускати цикли у різних напрямках (один зліва направо, тобто від менших індексів до більших, інший навпаки), або для кожного поточного елемента діяти, як у кодї праворуч (“<” для мінімуму та “>” для максимуму).

```
if curr < min_val then begin
    min_val := curr;
    min_idx := i;
end;
if curr >= max_val then begin
    max_val := curr;
    max_idx := i;
end;
```

Остаточну відповідь можна виразити як $\text{abs}(\text{max_idx} - \text{min_idx}) - 1$: модуль, бо перший мінімум може бути як лівіше за останній максимум, так і правіше; мінус 1, бо, наприклад, для сусідніх елементів ця різниця дорівнює 1, а проміжних елементів між сусідніми нема (0 штук). Приклади реалізації див. ideone.com/1wPo1q та ideone.com/b0AWG2 (обидві мовою Pascal, перша без масиву, друга з масивом).

Можна використати й бібліотечні засоби пошуку мінімуму та максимуму, але навряд чи це буде простіше; наприклад, у ideone.com/5mEcZH задача розв’язана через `min_element` та `max_element` (бібліотека `algorithm` з C++ STL), і потреба знайти *перший* мінімум та *останній* максимум призводить до не дуже очевидних різних ітераторів та різних перетворень їх у індекси.

Задача Е. «Відстань між мінімумом та максимумом–2»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 2 сек
Результати: Або екран, або output.txt Обмеження пам’яті: 64 мегабайти

Напишіть програму, яка за заданим масивом цілих чисел знайде місце мінімального її елемента, максимального її елемента, та відстань (кількість проміжних елементів) між ними. У випадку, якщо масив містить однакові мінімальні та/або однакові максимальні елементи на різних позиціях, слід вибрати такий з мінімальних та такий з максимальних елементів, щоб шукана відстань виявилася мінімальною (див. також приклади 3 та 4).

Вхідні дані. Перший рядок містить єдине число N ($2 \leq N \leq 10^6$) — кількість елементів масиву. Другий рядок містить (розділені пробілами) числа–елементи масиву; значення цих чисел–елементів цілі, від 1 до 10^6 .

Результати. Виведіть єдине ціле число — відстань (кількість проміжних елементів) між найближчими мінімальним і максимальним елементами.

Приклади:

Вхідні дані	Результати	Примітки
8 3 9 7 5 2 8 6 4	2	Між елементами 2 та 9 два проміжні елементи 5 та 7
8 4 3 2 1 8 7 6 5	0	Між елементами 1 та 8 нема проміжних елементів
8 3 1 4 1 5 9 2 6	1	Максимальний елемент (дев’ятка) єдиний; найближчою до єдиної дев’ятки є остання з мінімальних елементів (одиниць); між ними один проміжний елемент 5
8 1 1 1 1 1 1 1 1	0	Оскільки всі елементи однакові, кожен з них мінімальний і максимальний; між елементом і ним же самим нема проміжних

Оцінювання. 20% балів припадає на тести, в яких масив містить єдине мінімальне та єдине максимальне значення; 80% — на тести, де і мінімальне, і максимальне повторюються. Розподіл балів за розмірами тестів такий: 20% балів припадає на тести, де $2 \leq N \leq 100$; 30% — на тести, де $4000 < N \leq 10^4$; 20% — на тести, де $5 \cdot 10^4 < N \leq 10^5$; 30% — на тести, де $7 \cdot 10^5 \leq N \leq 10^6$. Писати різні програми для різних випадків не треба; розподіли вказані, щоб показати, скільки приблизно балів можна отримати, розв’язавши задачу не повністю.

Розбір задачі. Сформуємо два масиви: в одному всі індекси, де зустрічається мінімальне значення, в іншому — всі, де максимальне. Наприклад,

початковий масив	(індекси)	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)
(вхідні дані)	(значення)	3	1	4	1	5	9	2	6	1	8	2	8	1	8	2	8	4	5	9

призводить до формування таких масивів:

мінімуми	1	3	8	12
максимуми	5	18		

. Потім пройдемо по цим масивам, використовуючи злиття, але не формуючи послідовність-результат, а шукаючи мінімум серед усіх значень $\text{abs}(a[i] - b[j])$; потім ще 1 відняти наприкінці, якщо тільки це не буде ситуація, коли відповідь і так 0; на відміну від попередньої задачі, це можливо (коли всі елементи рівні).

На основі цього можна отримати код ideone.com/PBDdKu (C++).

На стор. 34 є короткий опис злиття та рекомендації щодо інших джерел.

3.14 II (районний/міський) етап 2017/18 н. р.

Задачі доступні для дорішування (ejudge.skiro.edu.ua, змагання №63).

Задача А. «Фарбування паралелепіпеда»

Є прямокутний паралелепіпед розмірами $x \text{ см} \times y \text{ см} \times z \text{ см}$. Є три фарби, які мають різну ціну: найдешевша — ціну $a \text{ коп/см}^2$, середня — ціну $b \text{ коп/см}^2$, найдорожча — ціну $c \text{ коп/см}^2$. Зверніть увагу, що $a < b < c$, але ніякого аналогічного співвідношення між x, y, z не гарантовано, вони можуть бути будь-якими.

Напишіть *вираз*, залежний від a, b, c, x, y, z , за яким можна обчислити *мінімальну* можливу вартість (у копійках) фарбування паралелепіпеда, щоб протилежні грані були пофарбовані в однакові кольори, а не протилежні — у різні. (Гранями паралелепіпеда є прямокутники, їх шість, і їх можна розбити на три пари протилежних: наприклад, якщо покласти паралелепіпед на рівну поверхню, можна говорити про протилежну пару «верхня та нижня грані», протилежну пару «ліва та права грані», і протилежну пару «передня та задня грані».)

Здати треба вираз, а не програму. В ejudge треба вписати у відповідне поле сам вираз, не створюючи ніякого файлу-розв'язку. Змінні a, b, c, x, y, z з умови задачі повинні називатися саме a, b, c, x, y, z , перейменовувати не можна. У виразі можна використовувати десяткові числа, дії “+” (плюс), “-” (мінус), “*” (множення), круглі дужки “(” та “)” для групування та зміни порядку дій, а також функції \min та \max . Формат запису функцій: після \min або \max відкривна кругла дужка, потім перелік (через кому) виразів, від яких береться мінімум чи максимум, потім закривна кругла дужка. Пропуски (пробіли) дозволяються, але не всередині чисел і не всередині імен \min та \max .

Наприклад, можна здати вираз $(a+b+c)*(\min(x,y)+\max(x,y,z))$; він неправильний за словом, але змінні названі правильно, функції використані дозволеним способом, і він іноді випадково дає правильний результат; цей розв'язок оцінюється на 11 балів з 200 можливих.

Наприклад, при $a=2, b=3, c=4, x=y=z=1$ числовий результат має дорівнювати 18 (на дві протилежні грані розмірами 1×1 йде фарба ціною 2, на ще дві протилежні, теж 1×1 , фарба ціною 3, на ще дві протилежні, теж 1×1 , ціною 4; всього $2 \times 2 \times 1 \times 1 + 2 \times 3 \times 1 \times 1 + 2 \times 4 \times 1 \times 1 = 4 + 6 + 8 = 18$). А, наприклад, при $a=2, b=3, c=4, x=20, y=30, z=40$ числовий результат має дорівнювати 14400.

Розбір задачі. Очевидно, що сумарна вартість буде мінімальною, якщо використати найдорожчу фарбу (ціни c) для пари граней мінімальної площі, середньої ціни b для граней середньої площі, найдешевшу (a) — для граней максимальної площі. При бажанні, це твердження можна *довести*, але ми доведення пропустимо; приклади аналогічних доведень (звісно, для інших задач та розв'язків) можна бачити, зокрема, на стор. 28, 40, 42.

Так що головна складність — знайти, яка з площ максимальна, яка середня і яка мінімальна. Для максимуму це, очевидно, $\max(x*y, x*z, y*z)$; аналогічно, для мінімуму $\min(x*y, x*z, y*z)$. Середню ж можна виразити, наприклад, як $\max(x,y,z)*\min(x,y,z)$ (чому? бо, якщо довжини всі різні, то є рівно три довжини: мінімальна, середня й максимальна, і з них можна утворити три пари (мінімальна, середня), (мінімальна, максимальна), (середня, максимальна), і мінімальну площу утворює перша пара, максимальну — остання, так що для середньої площі лишається друга пара). Ще один спосіб знайти середню площу $(x*y + x*z + y*z) - \max(x*y, x*z, y*z) - \min(x*y, x*z, y*z)$. (Якщо додати всі три й відняти з суми мінімум та максимум, залишиться середня.)

Насамкінець, досі скрізь шукали площу однієї грані, а сумарна площа пари протилежних граней удвічі більша. Це для першого способу дає вираз " $a^2 \cdot \max(x \cdot y, x \cdot z, y \cdot z) + c^2 \cdot \min(x \cdot y, x \cdot z, y \cdot z) + b^2 \cdot \max(x, y, z) \cdot \min(x, y, z)$ ", а для другого " $a^2 \cdot \max(x \cdot y, x \cdot z, y \cdot z) + c^2 \cdot \min(x \cdot y, x \cdot z, y \cdot z) + b^2 \cdot ((x \cdot y + x \cdot z + y \cdot z) - \max(x \cdot y, x \cdot z, y \cdot z) - \min(x \cdot y, x \cdot z, y \cdot z))$ ".

Ще один спосіб розв'язання задачі — ігноруючи міркування першого абзацу, не шукати, яка з площ мінімальна, середня та максимальна, а просто перебрати всі $3! = 6$ варіантів, які грані якою фарбою фарбувати, й вибрати мінімум: " $2 \cdot \min(a \cdot x \cdot y + b \cdot x \cdot z + c \cdot y \cdot z, a \cdot x \cdot y + c \cdot x \cdot z + b \cdot y \cdot z, b \cdot x \cdot y + a \cdot x \cdot z + c \cdot y \cdot z, b \cdot x \cdot y + c \cdot x \cdot z + a \cdot y \cdot z, c \cdot x \cdot y + a \cdot x \cdot z + b \cdot y \cdot z, c \cdot x \cdot y + b \cdot x \cdot z + a \cdot y \cdot z)$ ".

(Само собою, зараховуються не лише ці вирази, а будь-які правильні.)

Задача В. «Точки та круг»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 128 мегабайтів

На площині задано круг з центром у початку координат та набір точок. Напишіть програму, що знаходитиме, як(a/i) з точок леж(и/а)ть ззовні круга, але найближче до нього; якщо є різні такі точки, програма має знайти їх усі.

Вхідні дані. Перший рядок вхідних даних містить єдине ціле число N ($1 \leq N \leq 12345$) — кількість точок. Другий рядок містить єдине ціле число R ($1 \leq R \leq 12345$) — радіус круга. Кожен з подальших N рядків містить рівно два цілі числа — x - та y -координати відповідної точки. Всі координати є цілими числами, які не перевищують за абсолютною величиною (модулем) 12345.

Результати. Якщо шуканих точок не існує (жодна з уведених точок не ззовні круга), програма має вивести єдиний рядок з єдиним числом 0.

Інакше, програма має вивести перелік номерів усіх тих точок, які лежать ззовні круга і мають однакову мінімальну відстань від нього. Кількість точок у переліку виводити не треба, сам перелік виводити у порядку зростання номерів.

Приклади:

Вхідні дані	Результати	Примітки
2 50 30 40 17 23	0	Одна з точок на межі круга, інша всередині нього, жодної точки ззовні

Вхідні дані	Результати	Примітки
5 10 30 40 40 30 -30 -40 777 555 0 -50	1 2 3 5	Зразу чотири з п'яти точок мають однакову мінімальну відстань (50 до центру круга, або $50 - 10 = 40$ до найближчої його частини)

Розбір задачі. Формула відстані від початку координат до точки з координатами $(x; y)$ має вигляд $\sqrt{x^2 + y^2}$ і легко слідує з теореми Піфагора (відстань рахується уздовж гіпотенузи, де довжини горизонтального та вертикального катетів рівні $|x|$ та $|y|$ відповідно; $|x|^2 = x^2$). Точка всередині круга або на межі, коли $\sqrt{x^2 + y^2} \leq R$, і поза ним, коли $\sqrt{x^2 + y^2} > R$. Для деяких поєднань мови програмування та налаштувань компілятора надійніше переписати ці формули як $x^2 + y^2 \leq R^2$ та $x^2 + y^2 > R^2$, щоб уникнути похибок (див. стор. 14), але для багатьох і так, і так правильно. Важливо також уникнути переповнень (стор. 13), для чого варто подати координати та радіус у типі, який поміщає значення не лише самих координат та радіусів, а також і їхніх квадратів.

«Стандартний шкільний» алгоритм пошуку мінімуму велить спочатку взяти за мінімум перший елемент послідовності, далі порівнювати кожен елемент з поточним мінімумом, і щоразу, коли черговий елемент менший мінімуму, виправляти це, змінюючи значення мінімуму на цей елемент. У цій задачі ситуація ускладнюється тим, що треба шукати мінімум не серед усіх, а лише серед тих, які строго більші R (чи R^2 у варіанті, де не добувають корінь). Це суттєво ускладнює етап «спочатку взяти за мінімум перший елемент», бо 1-а (а також 2-а, 3-я, ...) точка може бути всередині кола. Це можна вирішити будь-якою однією з двох відомих модифікацій алгоритму пошуку мінімуму.

(А) Зберігати, крім мінімуму `minSqrDist`, ще білеву змінну `anyFound`, що означає «чи знайдено хоч одну точку за межами круга». Тоді можна написати так: `if (currSqrDist>R*R) and (not anyFound or (currSqrDist<minSqrDist)) then minSqrDist:=currSqrDist`. Тобто, оновлювати мінімум, коли поточна точка за межами круга і при цьому або перша така, або її відстань менша за раніше знайдену мінімальну.

(Б) Ініціалізувати мінімум не першим значенням, а «плюс нескінченістю», яка конкретно у цій задачі може бути, наприклад, вигляду 2100100100: це число більше будь-якого реально можливого, а отже при знаходженні першої точки поза кругом обов'язково буде замінене на відстань до цієї точки. «Головне» присвоєння набуває більш очевидного вигляду `if (currSqrDist>R*R) and (currSqrDist<minSqrDist) then minSqrDist:=currSqrDist`.

В обох розглянутих підходах, першу точку слід обробляти так само, як і всі подальші.

Ще питання: які тут потрібні масиви? Щоб зберігати координати, можна використати або два окремі масиви `x` та `y`, або масив записів (структур) чи об'єктів, де кожен елемент містить обидва поля `x` та `y`. Але координати взагалі не дуже-то й треба складати у масив, їх цілком можна читати й тут же обробляти і забувати. А відстані (чи квадрати відстаней), мабуть, легше все-таки зберегти у масиві. Адже потрібно вивести *номери всіх* потрібних точок, а поки не завершено обробку всіх точок, не можна бути впевненим, що поточна точка на мінімальній відстані (бо пізніше може знайтися на ще меншій). Так що легше першим проходом знайти мінімальну відстань (чи її квадрат), а другим повиводити номери всіх точок на такій відстані. Приклади таких реалізацій мовою Pascal: ideone.com/wvA2kg (підхід А), ideone.com/YgNUld (підхід Б).

В принципі можна й обійтися зовсім без масиву. Наприклад, так: щоразу, коли знаходиться нова точка на такій самій наразі мінімальній відстані, її номер дописується у вихідний файл, а щоразу, коли мінімум оновлюється, бо нова точка має строго меншу відстань, файл результату очищується й номер цієї точки пишеться у щойно очищений файл. Зокрема, саме так організований код ideone.com/jSPTGR, що використовує `assign(output, 'output.txt')` та `rewrite(output)`, при задачі в ejudge.skipo.edu.ua мовою `fpc` працює правильно. І ця реалізація працювала б навіть при кількості точок 10^7 та обмеженні пам'яті 4 Мб (втім, зараз це не треба). Тут важливо, що програма працює з файлом, а не екраном; через це, конкретно цей код (один з дуже небагатьох) не працює ні безпосередньо на ideone.com, ні якщо здати його на ejudge.skipo.edu.ua, прибравши оператори `assign`. Міркування «сказано «або екран, або `output.txt`», значить нема ніякої відмінності» правильне часто, але якщо з файлами намагаються працювати якось хитро, то не завжди. З цієї ж причини автори збірника не гарантують, чи всіма мовами програмування доступний цей прийом.

Ще можна діяти подібно до попереднього абзацу, але номери точок, відстані яких рівні поточному мінімуму, не писати у файл, а складати в деяку проміжну структуру. Мовою C++ найзручнішим здається `vector` (з використанням `push_back` та `clear`; див. ideone.com/D21hEr). Мовами Java та C# — `StringBuilder`, де формувати зразу рядкове подання відповіді, теж з очищенням при знаходженні ще меншого мінімуму; див. ideone.com/3VmbDa (Java; якщо замінити `Scanner` на `StreamTokenizer`, читання відбуватиметься швидше, див. також стор. 15). Але навряд чи якийсь з цих способів легший і простіший, ніж згадане першим використання масиву відстаней (чи квадратів відстаней) і двох проходів. Казати ж «формувати відповідь у `StringBuilder` краще тим, що потребує менше пам'яті, бо зберігаються номери не всіх точок, а лише мінімальних (за відстанню)» — правда лише частково, бо з тими обмеженнями, які є, ця різниця неважлива, а якби задачу ставили у жорсткому варіанті « 10^7 точок, 4 Мб пам'яті», то, мабуть, дали б частину тестів, де $9,99 \cdot 10^6$ з усіх 10^7 точок на однаковій мінімальній відстані, й пам'яті на `StringBuilder` не вистачило б.

Само собою, в будь-якому розв'язку треба не забувати правильно виводити відповідь 0.

Задача С. «Цікаві числа»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 128 мегабайтів

Напишіть програму, яка знаходитиме кількість «цікавих» чисел, менших за задане число N . Число називається «цікавим», якщо в ньому використовуються лише цифри 1, 2, 4 та 8 (не обов'язково усі; обов'язково, щоб не було інших). Наприклад числа 12, 18, 8, 284 є цікавими, а числа 246, 143, 3, 487 — ні.

Вхідні дані. В єдиному рядку задано єдине число N (обмеження у розділі «Оцінювання»).

Результати. Виведіть єдине число в єдиному рядку — кількість «цікавих» чисел, менших N .

	Вхідні дані	Результати	Примітки
Приклад:	18	7	Цими сімома числами є: 1, 2, 4, 8, 11, 12, 14.

Оцінювання. 20% балів припадає на тести, де $N = 10^T$, $1 \leq T \leq 18$. Ще $\geq 20\%$ на тести, де $1 \leq N \leq 10^5$. Ще $\geq 40\%$ на тести, де $10^6 < N \leq 10^{12}$. Ще $\geq 20\%$ на тести, де $10^{12} < N \leq 10^{18}$. Писати треба одну програму, а не різні програми для різних випадків; єдина мета цього переліку різних блоків обмежень — дати уявлення про те, скільки балів можна отримати, якщо розв'язати задачу частково.

Розбір задачі. Скільки повинен набирати очевидний розв'язок «перебрати всі числа від 1 до $N-1$, розібрати кожне на цифри, й подивитися, чи всі ці цифри з набору 1, 2, 4, 8?» Приблизно 100 балів з 300. Точна величина залежить від мови програмування (це впливає на швидкість) та деталей реалізації цього підходу.

А як набрати більше? Ця задача має багато спільного з задачами «Генератор паролів» (стор. 49–51) та «Сучасне мистецтво» (стор. 59–61). Деталі подумайте самостійно, враховуючи це та наведений далі приклад ручного аналізу. (Можливі й інші розв'язки цієї задачі; деякі з них теж достатньо ефективні, щоб пройти всі тести, вклавшись у обмеження часу, але ж далеко не всі.)

Навіщо виділено випадок $N = 10^T$? У цьому частковому випадку задача має простий комбінаторний розв'язок: $4 + 4^2 + \dots + 4^T$, або $4 \cdot \frac{4^T - 1}{4 - 1}$. Наприклад, при $N = 10000$ можна сказати, що відповіддю є сумарна кількість 1-значних, 2-значних, 3-значних та 4-значних «цікавих» чисел. Однозначних «цікавих» чисел чотири, бо це і є цифри 1, 2, 4, 8. Двозначних «цікавих» чисел шістнадцять; це можна підтвердити їх повним переліком (11, 12, 14, 18, 21, 22, 24, 28, 41, 42, 44, 48, 81, 82, 84, 88), але корисніше отримати цей результат так: є дві позиції, на кожную можна поставити будь-яку з чотирьох цифр 1, або 2, або 4, або 8; кожна з цифр на першій позиції може поєднуватися з будь-якою з цифр на другій позиції — отже, кількість пар може бути обчислена як $4 \times 4 = 16$. Аналогічно, 3-значних «цікавих» чисел є $4^3 = 64$, 4-значних $4^4 = 256$, а менших за 10^4 є $4 + 16 + 64 + 256 = 340$.

Приклад ручного аналізу цієї задачі. Знайдемо кількість «цікавих» чисел, менших 8432. Це, зокрема, всі 1-, 2- та 3-значні «цікаві», їх $4 + 4^2 + 4^3 = 84$. Крім того, раз старша цифра дорівнює 8, треба врахувати ще всі 4-значні «цікаві», що починаються з 1, 2 або 4 (їх у кожній з цих трьох груп по $4^3 = 64$, тобто всього 192; разом з раніше згаданими це $84 + 192 = 276$), і додати до них ті 4-значні «цікаві», які починаються з 8 і при цьому менші 8432. Оскільки початкова «8» тепер зафіксована, вона не впливає на кількість, можна рахувати все одно як кількість усіх 3-значних «цікавих», менших 432. Цілком аналогічно виділяємо окремо всі 3-значні «цікаві», що починаються з 1 або 2 (їх у кожній з цих двох груп по $4^2 = 16$, тобто всього 32; разом з раніше згаданими це $276 + 32 = 308$). Лишається порахувати кількість 3-значних «цікавих», які починаються з 4 і при цьому менші 432. Знову, початкова «4» зафіксована й не впливає на кількість, так що рахуємо кількість 2-значних «цікавих», менших 32. Цілком аналогічно виділяємо окремо всі 2-значні «цікаві», що починаються з 1 або 2 (їх у кожній з цих двох груп по $4^1 = 4$, тобто всього 8; разом з раніше згаданими це $308 + 8 = 316$). Тільки тепер, хоча цифри ще не скінчилися, процес обривається, бо «цікаві» числа все одно не можуть містити цифру 3. Так що відповіддю є 316.

А чим відрізняється останній блок від передостаннього? З точки зору пропонованого алгоритму — майже нічим. Але відмінність є з точки зору іншого алгоритму: генерувати всі підряд лише «цікаві» числа, доки не дійде до вказаної межі. Такий алгоритм може пройти тести передостаннього блоку, але тести останнього повинні не вкладатися в ліміт часу.

Задача D. «Дірка і Пробка»

Вхідні дані: Або клавіатура, або input.txt

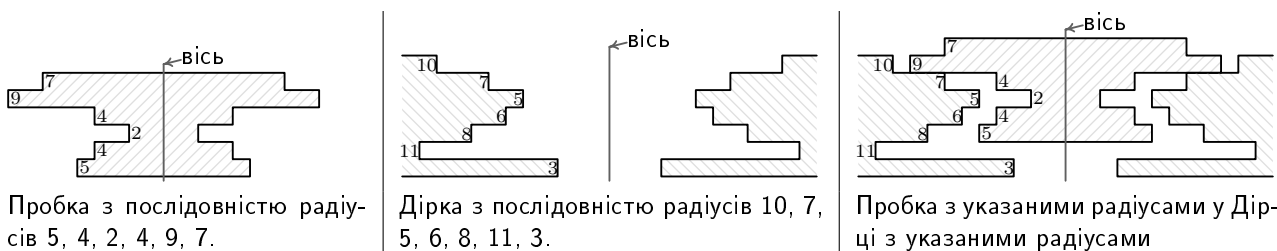
Обмеження часу: 1 сек

Результати: Або екран, або output.txt

Обмеження пам'яті: 128 мегабайтів

Пан Дивак не пожалів зусиль, щоб виготовити Дуже Дивну Дірку та Дуже Дивну Пробку. Тепер його цікавить, чи можна цією Пробкою затикати цю Дірку, і якщо можна, то наскільки глибоко вона заходитиме всередину Дірки.

Дуже Дивна Пробка являє собою послідовність нанизаних на спільну вісь циліндрів однакової висоти 1, але, як правило, різних радіусів. Аналогічно, Дуже Дивна Дірка являє собою послідовність циліндричних порожнин зі спільною віссю, однаковими висотами 1, але, як правило, різними радіусами. Циліндри, які утворюють Пробку, пан Дивак рахує знизу догори, а циліндри, які утворюють Пробку — згори донизу. Затикати Дірку Пробкою пан Дивак планує лише вставляючи Пробку в Дірку згори донизу, так, щоб спільна вісь усіх циліндрів Пробки збігалася зі спільною віссю усіх циліндрів Дірки.



Наприклад, на рисунку зображено окремо Пробку, окремо Дірку, та ситуацію, коли ця Пробка заткнула цю Дірку; рух Пробки донизу скінчився тоді, коли частина Пробки радіусом 9 лягла поверх частини Дірки радіусом 7. Зверніть увагу, що частина Пробки радіусом 5 пройшла крізь частину Дірки радіусом 5, бо Пан Дивак натискає на Пробку зі значною силою (але не настільки великою, щоб пропхнути строго більше крізь строго менше).

Напишіть програму, яка за описами Пробки та Дірки з'ясуватиме, як глибоко ця Пробка може зайти углиб цієї Дірки.

Вхідні дані. 1-й рядок містить єдине число N — кількість циліндрів у Пробці. 2-й рядок містить (розділені пробілами) N радіусів циліндрів Пробки a_1, a_2, \dots, a_N , знизу догори. 3-й рядок містить єдине число M — кількість циліндрів у Дірці. 4-й рядок містить (розділені пробілами) M радіусів циліндрів Дірки b_1, b_2, \dots, b_M , згори донизу. Обмеження на N, M див. у розділі «Оцінювання»; усі значення a_i та b_j є цілими з проміжку від 1 до 10^9 .

Результати. Виведіть єдине число — глибина, на яку Пробка (рахуючи від її низу) увиходить всередину Дірки (рахуючи від її верху). У випадку, коли Пробка не заходить всередину Дірки, бо найнижчий циліндр Пробки ширший за найвищий отвір Дірки, слід виводити число 0. У випадку, коли Пробка може пройти Дірку наскрізь, слід виводити число 777555777.

Приклади:

Вхідні дані	Результати	Примітки
6 5 4 2 4 9 7 7 10 7 5 6 8 11 3	5	Приклад з рисунків; нижче за верхній край Дірки зайшли 5 з 6 циліндрів Пробки
2 17 23 2 4 7	0	Самий низ Пробки лежить згори на самому верху Дірки
2 9 23 2 14 7	1	Самий низ (9) Пробки зайшов всередину самого верху (14) Дірки, а потім Пробка обома циліндрами вперлася в Дірку (9 у 7, 23 у 14)
2 4 7 2 17 23	777555777	Пробка може пройти крізь Дірку наскрізь
5 2 4 7 17 23 4 36 30 24 18	7	Пробка майже пройшла крізь Дірку, але самий верх Пробки (23) все-таки вперся у самий низ Дірки (18)

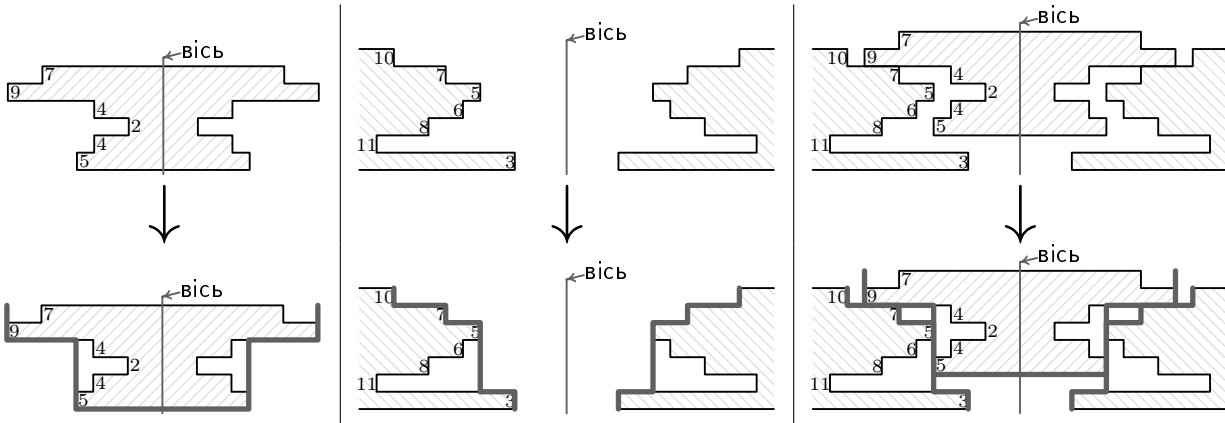
Оцінювання. 20% балів припадає на тести, в яких кількість циліндрів Пробки $N = 1$, Дірки $2 \leq M \leq 12345$; ще 20% балів — на тести, в яких $2 \leq N \leq 12345$, $M = 1$; ще 20% балів — на тести, в яких $12 \leq N \leq 123$, $12 \leq M \leq 123$; решта 40% балів — на тести, в яких $12 \leq N \leq 123456$, $12 \leq M \leq 123456$. Писати треба одну програму, а не різні програми для різних випадків; єдина мета цього переліку різних блоків обмежень — дати уявлення про те, скільки балів можна отримати, якщо розв'язати задачу частково.

Розбір задачі. Для 1-го ($N = 1$) та 2-го ($M = 1$) блоків досить моделювати рух послідовності Пробки крізь єдиний циліндр Дірки чи рух єдиного циліндру Пробки крізь послідовність Дірки відповідно, й на кожному кроці перевіряти, чи впирається черговий циліндр Пробки в єдиний циліндр Дірки чи єдиний циліндр Пробки в черговий циліндр Дірки. За посиланням ideone.com/u3mBLC наведено код, який по суті лише це й перевіряє. Він набирає 144 бали з 300, серед яких 120 тому, що згідно описаного оцінювання і повинен їх набрати за 1-й та 2-й блоки, а ще 24 практично випадково (що буває при потестовому оцінюванні, яке у 2017 р. все ще було єдиним можливим на цьому етапі; див. також стор. 6). Це частково свідчить про недостатню продуманість тестів, але й частково завдячує непоганій продуманості коду на рівні таких прийомів: хоча код правильний лише при $N=1$ або $M=1$, будемо застосовувати один з цих підходів також і в решті випадків; розміри масивів візьмемо максимальними відносно всіх тестів, а не лише блоків 1–2. Адже якщо все це зробити, то шанси взяти якісь випадкові бали є, якщо не робити — нема.

Тепер розглянемо, як написати код, який гарантовано розв'язує блоки 1–3 плюс випадково деякі тести блоку 4. Спочатку розглянемо, чи впирається найнижчий циліндр Пробки у найвищий циліндр Дірки (якщо так, то відповідь = 0 й не треба більш нічого перевіряти). Інакше, розглянемо дві пари циліндрів, які можуть дати відповідь 1 (найнижчий циліндр Пробки впирається у 2-й згори циліндр Дірки, або 2-й знизу циліндр Пробки впирається у найвищий циліндр Дірки), і якщо хоча б одна з двох пар впирається, то відповідь = 1 (вона не більша, бо знайдено пару циліндрів, які впираються при заглибленні 1, і не менша, бо це вже було б визначено раніше). І так далі: три можливі пари циліндрів, що можуть дати відповідь 2, потім чотири можливі пари циліндрів, що можуть дати відповідь 3, ... Додаткового клопоту завдають ситуації, коли низ Пробки вже вийшов за межі Дірки та/або верх Пробки вже занурився нижче верху Дірки, а інші пари циліндрів все ще треба порівнювати, бо вони все ще можуть впертися; при неакуратному програмуванні тут можуть бути виходи за межі масиву. Можна пробувати вирішити цю проблему, умовно «доточивши» Пробку циліндрами дуже малого радіусу, а Дірку — циліндрами дуже великого радіусу, і це може бути компромісом між ефективністю та простотою коду. Але все ж код з «доточуваннями» ideone.com/CE0f36 наби-

рає менше балів, ніж код ideone.com/r8j9FS, де не витрачається час на неіснуючі пари циліндрів. Ще з іншого боку, такий код не дуже простіший, ніж наведені далі способи взяти повні бали.

Заміна Дірки та Пробки на монотонні. Оскільки Пробку вставляють у Дірку строго згори донизу, зберігаючи вісі спільними, очевидно, що і Пробку, і Дірку можна замінити на Пробку та Дірку, в яких послідовності радіусів циліндрів монотонні: у Пробці, чим вищу частину дивимось, тим вона ширша, а у Дірці, чим глибшу частину дивимось, тим вона вужча. Якимось так:



Після цього, є щонайменше два способи використати монотонність отриманих послідовностей, щоб отримати ефективніші, ніж $O(M \cdot N)$, алгоритми.

Багатократні бінарні пошуки. Для кожного з циліндрів Дірки можна шукати бінпошуком, який саме циліндр Пробки упреться у нього, а результати всіх цих пошуків поєднати в аналозі стандартного пошуку мінімуму. Враховуючи, що тут можна не писати бінпошук вручну (а, наприклад, використати функцію `lower_bound` з бібліотеки `algorithm` C++ STL), цей спосіб може виявитися досить простим для реалізації, як у ideone.com/CgmbBs. Асимптотична складність становить $\Theta(M \cdot \log N)$, що при $\max(M, N) \leq 123456$ цілком прийнятно. Можна й «симетрично» для кожного циліндру Пробки шукати, в який циліндр Дірки він упреться, й отримати теж прийнятну складність $\Theta(N \cdot \log M)$.

А за $\Theta(N + M)$ можна? Так, можна. Щоправда, це мало впливає на фактичний час виконання (спосіб читання вхідних даних, див. також стор. 15, впливає сильніше), але теоретично краща оцінка $\Theta(N + M)$ тут можлива. На жаль, ідею дуже важко пояснити словами, тож пропонується читати код ideone.com/cr1W66 (в якому наведено коментарі), і намагатися співвіднести сам код, коментарі та самостійно виконані приклади застосування цього коду до різних вхідних даних. Цей алгоритм навіть не є якимсь класичним; він має дещо спільне зі злиттям (див. стор. 34), але небагато.

3.15 Обласна інтернет-олімпіада 2018/19 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №23).

Задача А. «Кількість секунд»

Є два моменти часу: a годин b хвилин c секунд та d годин e хвилин f секунд. Гарантовано, що вони належать одній добі; гарантовано, що перший з них ($a : b : c$) відбувся раніше, ніж другий ($d : e : f$); гарантовано, що $0 \leq a, d \leq 23$, $0 \leq b, c, e, f \leq 59$.

Напишіть вираз, який знаходитиме, на скільки секунд другий момент часу пізніше, ніж перший.

Змінні, від яких залежить вираз, обов'язково повинні мати саме той смисл, який описаний раніше, і називатися вони повинні саме a, b, c, d, e, f (маленькими латинськими літерами).

У цій задачі треба здати не програму, а вираз: вписати його (сам вираз, не назву файлу) у відповідне поле перевіряючої системи і відправити на перевірку. Правила запису виразу: можна використовувати цілі десяткові числа, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), “/” (ділення дробове, наприклад, $17/5=3,4$), “//” (ділення цілочисельне, наприклад, $17//5=3$), круглі дужки “(” та “)” для групування та зміни порядку дій. Дозволяються пропуски (пробіли), але не всередині чисел.

Наприклад: можна здати вираз $(d+e+f)-(a+b+c)$ і отримати 6 балів з 200, бо він неправильний, але все ж іноді відповідь випадково збігається з правильною. А «цілком аналогічний» вираз $(h+m+s)-(x+y+z)$ буде оцінений на 0 балів, бо змінні повинні називатися так, як вказано.

Розбір задачі. Дуже проста задача. Без проблем працюють обидва очевидні підходи «перетворити кожне окремо в секунди, потім відняти» (вираз $“(3600*d+60*e+f) - (3600*a+60*b+c)”$) та «відняти окремо години, окремо хвилини, окремо секунди, звести до купи з відповідними коефіцієнтами» (вираз $“(3600*(d-a) + 60*(e-b) + (f-c))”$). В цьому виразі значення деяких його частин можуть виявлятися від’ємними, але це ніяк не заважає виразу в цілому. Само собою, зараховуються не лише ці дві відповіді, а будь-яка правильна.

Задача В. «Файлова система»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам’яті: 64 мегабайти

У багатьох файлових системах дисковий об’єм, зайнятий файлом, не завжди дорівнює розміру файла. Це пов’язане з тим, що диск поділяється на кластери однакового розміру, і кожен кластер може бути або вільним, або використаним одним файлом (але не поділений між кількома файлами). Для прикладу, на диску з розміром кластера 512 байтів зберігається файл розміром 600 байтів. В такому випадку файл зберігатиметься у двох кластерах, займаючи 1024 байти.

Напишіть програму, яка, прочитавши розмір кластера, кількість файлів і їхні розміри, обчислюватиме сумарний об’єм, який займають ці файли на диску.

Вхідні дані. Ваша програма має прочитати спочатку розмір кластера, потім (у наступному рядку) кількість файлів N , потім (у наступному рядку) N чисел, розділених пропусками (пробілами) — розмір кожного з цих файлів. Розмір кластера є натуральним числом від 1 до 1048576; кількість файлів є цілим числом від 0 до 10; розмір кожного файлу є цілим числом від 0 до 1048576.

Результати. Ваша програма має вивести єдине ціле число в єдиному рядку — знайдений сумарний об’єм, зайнятий файлами на диску.

Приклади:	Вхідні дані	Результати	Вхідні дані	Результати
	512	1024	92651	555906
	1		6	
	600		6767 8170 6899 4827 1 9019	
	1024	114688	32768	131072
	3		5	
	4096 33792 76800		16 32 128 128 0	

Примітка. Під час перевірки в едžadжі, саме ці чотири приклади, саме в такому порядку, й будуть тестами 1–4.

Розбір задачі. З’ясуємо, як обчислювати дисковий розмір окремо взятого файлу. Надалі вважаємо, що розмір файла у байтах лежить у змінній `f_len`, розмір кластера у змінній `cluster`.

У багатьох мовах (C/C++, Java, Python, C#, pascalABC — саме ABC, а не будь-який Паскаль) є функція `ceil` (у C#, `Ceiling`), яка виконує округлення вгору: якщо число-аргумент ціле, то результат дорівнює самому цьому числу, а якщо дробове, то за результат береться найменше ціле число, більше за аргумент. Отже, кількість кластерів дорівнює `ceil(f_len/cluster)`. Багатьма мовами програмування символ-у-символ так, але не всіма. Слід забезпечити, щоб це було ділення за дробовими, а не цілочисельними, правилами (наприклад, $17/5$ повинно давати 3.4, а не 3). У Pascal та Python3 це вийде саме; в інших це можна зробити записом у стилі `“ceil((1.0*len)/cluster)”`, або записом у стилі `“ceil(((double)len)/cluster)”` (це називають *приведенням типів*, англ. *typecast*), або оголосивши самі змінні `f_len` та `cluster` у типі з рухомою комою. Детальніше варто розбиратися за підручником з конкретної мови програмування.

Можна й обійтися без функції `ceil`. Наприклад, спочатку цілочисельним діленням (`“f_len div cluster”` у Pascal, `“f_len//cluster”` у Python3, `“f_len/cluster”` у C-подібних мовах та Python2;

все це за умови, що обидві змінні `f_len` та `cluster` цілочисельні) порахувати кількість повних кластерів, зайнятих файлом; потім подивитися, чи розмістився файл повністю в цих кластерах, чи потрібен ще один. Ще один кластер потрібен тоді й тільки тоді, коли він неповний, тобто при діленні є залишок, тобто "`f_len mod cluster <> 0`" (Pascal), "`f_len%cluster != 0`" (C-подібні та Python).

Коли пораховані дискові розміри окремих файлів, лишається тільки подавати їх.

Слід переконатися, що програма правильно працює у «крайніх», але дозволених умовою задачі, випадках: (1) файл розміром 0 байтів займає 0 байтів; (2) якщо кількість файлів 0, «вони усі сумарно» займають 0 байтів; (3) розмір кластера може дорівнювати 1 байту; це суперечить реаліям і самому смислу кластерів, але не заборонено умовою задачі.

Неприємною несподіванкою може бути також те, що у Free Pascal тип `integer` 16-бітовий, а отже не може вмістити в собі потрібні значення (див. також стор. 13).

Задача С. «Палички — інтерактивна гра»

Вхідні дані: Стандартний вхід (клавіатура) Обмеження часу: 1 сек
Результати: Стандартний вихід (екран) Обмеження пам'яті: 64 МБ

Є одна купка, яка спочатку містить N паличок. Двоє грають у таку гру. Кожен з гравців на кожному своєму ході може забрати з купки або 1, або 2, або 3 палички. Ніяких інших варіантів ходу нема. Ходять гравці по черзі, пропускати хід не можна. Виграє той, хто забирає останню паличку (можливо, разом із ще однією або ще двома).

Напишіть програму, яка інтерактивно гратиме за першого гравця.

На початку, один раз, Ваша програма повинна прочитати одне ціле число в окремому рядку — початкову кількість паличок N ($1 \leq N \leq 12345$). Потім вона повинна повторювати такий цикл:

1. Вивести єдине число в окремому рядку — свій хід, тобто кількість паличок, які вона зараз забирає з купки. Це повинно бути ціле число від 1 до 3, не більше за поточну кількість паличок у купці.
2. Якщо після цього купка стає порожньою, вивести окремим рядком фразу "I won!" (без лапок, символ-у-символ згідно зразку) і завершити роботу.
3. Інакше, прочитати хід програми-суперниці, тобто кількість паличок, які вона зараз забирає з купки (єдине ціле число, в окремому рядку). Гарантовано, що хід допустимий (є цілим числом від 1 до 3 і не перевищує поточного залишку паличок у купці). Само собою, ця гарантія дійсна лише за умови, що Ваша програма правильно визначила, що гра ще не закінчилася.
4. Якщо після цього купка стає порожньою, вивести фразу "You won..." (без лапок, символ-у-символ згідно зразку) і завершити роботу.

Все вищезгадане повинно повторюватися, доки не будуть забрані всі палички (тобто, доки якась із програм-гравців не виграє). Програма-суперниця не виводить фраз "I won!" / "You won..." чи якихось їх аналогів.

Ця задача є інтерактивною: Ваша програма не отримає всіх вхідних даних на початку, а отримуватиме по мірі виконання доуточнення, що залежатимуть від попередніх дій Вашої програми. Тим не менш, її перевірка буде *автоматичною*. Тому, в цій програмі, як і в програмах-розв'язках інших задач, теж слід не «організовувати діалог інтуїтивно зрозумілим чином», а чітко дотримуватися формату. Тільки це не формат вхідного та вихідного файлів, а формат спілкування з програмою, яка грає роль суперника.

Настійливо рекомендується, щоб Ваша програма після кожного свого виведення робила дію `flush(output)` (Pascal), вона ж `cout.flush()` (C++), вона ж `fflush(stdout)` (C), вона ж `sys.stdout.flush()` (Python), вона ж `System.out.flush()` (Java). Це істотно зменшує ризик, що проміжна відповідь «застрягне» десь по дорозі, не дійшовши до програми-суперниці.

Приклад:

Вхідні дані	Результати	Примітки
5		У купці з самого початку 5 паличок.
2	1	Ваша програма забирає одну, лишається чотири;
	2	програма-суперниця забирає дві, лишається дві;
	I won!	Ваша програма забирає обидві, повідомляє про свій виграш і завершує роботу.

Оцінювання.

У приблизно половині тестів Ваша програма матиме справу з ідеальною програмою-суперницею, яка не робить помилок. У іншій приблизно половині — з різними програмами-суперницями, які грати не вміють — тобто, роблять лише ходи, які дотримуються формальних вимог «забирати лише від 1 до 3 паличок» та «забирати не більше паличок, ніж реально є у купці», але можуть вибирати не найкращий з допустимих ходів, дотримуючись кожна своїх власних уявлень про те, як краще грати в цю гру. Буде оцінюватися як уміння Вашої програми виграти там, де це легко, так і вміння Вашої програми достойно, згідно правил гри, програти, так і вміння Вашої програми скористатися (теж згідно правил) помилками чи іншими неадекватностями програми-суперниці, якщо такі будуть.

За будь-яке порушення правил гри з боку Вашої програми, відповідний тест оцінюватиметься як не пройдений.

Розбір задачі. Нехай у купці одна, дві або три палички. Тоді за один хід можна забрати їх усі й цим виграти. Від суперника тут нічого не залежить.

Тепер нехай у купці чотири палички. Тоді після будь-якого допустимого правилами гри ходу супернику залишиться або $4 - 1 = 3$ палички, або $4 - 2 = 2$, або $4 - 3 = 1$. Після чого, суперник зможе взяти усі палички одним ходом і тим виграти. Ми (1-й гравець) не можемо цьому перешкодити.

Таким чином, позиції «1 паличка», «2 палички» та «3 палички» *виграшні*: починаючи з них, можна виграти, і суперник з цим нічого не вдіє. А позиція «4 палички» *програшна*: якщо суперник вміє грати, він точно виграє, і той, кому дісталася 4 палички, з цим нічого не вдіє.

Якщо у купці 5, 6 або 7 паличок, можна зробити такий хід, щоб після нього супернику дісталася купка з 4-х паличок (що, як ми вже знаємо, означає, що він нічого не зможе протиставити правильній грі свого суперника, тобто нашій правильній грі). Таким чином, позиції «5 паличок», «6 паличок» та «7 паличок» теж виграшні. Аналогічно показується, що позиція «8 паличок» програшна. І так далі: позиції, де кількість паличок кратна 4, програшні (починаючи з них, перемогти грамотного суперника неможливо), а позиції, де кількість паличок не кратна 4, виграшні (треба забирати $n \bmod 4$ паличок, і цим або виграємо негайно (при $1 \leq n \leq 3$), або супернику дістанеться програшна позиція).

Аналіз гри часто цим і закінчують, а питання «що робити, коли нам дісталася програшна позиція?» або ігнорують, або відповідають на нього абияк. Зокрема, часто заявляють «раз при ході з програшної гарантовано виграти неможливо, будемо відтягувати кінець, беручи щоразу лише по одній паличці: чим довше триватиме гра, тим більше шансів, що неідеальний суперник помилиться».

Але в умові не дарма сказано, що у приблизно половині тестів треба грати проти безграмотних програм-суперниць, *причому в різних тестах різних*. Серед цих безграмотних програм-суперниць є така, яка завжди намагається забирати якнайбільше паличок, тобто або 3, або, якщо їх всього лишилося менш, ніж 3, то всі. Правда, у *такої* суперниці можна виграти, маючи на початку програшну позицію? Але для цього треба, перебуваючи у програшній позиції, взяти не одну паличку, а 2 або 3... (Це не допомагає, якщо початкова кількість паличок = 4: програма-суперниця «не встигає збитися»; але при значно більших початкових кількостях паличок «перехоплення ініціативи» цілком реальне.)

Так само, як не можна у програшних позиціях завжди виводити 1, не можна й виводити завжди 2 чи завжди 3, бо серед інших програм-суперниць є та, що завжди намагається забирати дві палички (й тільки якщо лишилася всього одна, то забере її одну), а також та, що абсолютно завжди забирає одну паличку. Тому треба або робити хід із програшної позиції якимсь хитро залежним від того, з якої саме програшної позиції ходимо...

...або застосувати всемогутні випадкові числа: з виграшних позицій ходити, згідно з основним аналізом, ходом $n \bmod 4$, а з програшних — суто випадково, від 1 до 3. Це, звісно, не допоможе проти грамотного суперника, але ситуації, описані у двох попередніх абзацах, стануть неможливими, й у випадках, де це можливо, ініціатива перехоплюватиметься. (Ще раз: тут *не* пропонується *завжди* ходити випадково; така програма майже завжди програвала б «ідеальному» супернику, котрий не помиляється. Пропонується у першу чергу дивитися на виграшність/програшність, а вже у випадку програшності, коли гарантій все одно нема, ходити випадково.)

У розв'язках цієї задачі якось трапилася кумедна помилка. Одна особа, бажаючи зробити числа якнайвипадковішими, писала перед кожним викликом $(\text{rand}() \% 3) + 1$ (згенерувати чергове псев-

довипадкове число від 1 до 3) новий виклик `srand(time(NULL))` (переналаштувати генератор псевдовипадкових чисел). Така реалізація (мовою C++) програє згаданим суперникам вигляду «намагатися завжди брати ...», причому при різних запусках — різним суперникам (на різних тестах): то супернику, що намагається завжди брати 3, то тому, що намагається завжди брати 2, то тому, що завжди бере 1. Причина в тому, що `time(NULL)` залежить від поточного системного часу, але не наносекунд, які щоразу різні, а цілих секунд, які, як правило, однакові протягом усього виконання програми. Так і виходило, що генератор, щоразу однаково переініціалізований, видавав одне й те ж.

Розглянута в цій задачі гра має назву: *гра Баше*. Втім, цю задачу потроху розв'язували різні люди, внесок К.Г. Баше не є вирішальним, і назва «гра Баше» поширена, але не загальноприйнята.

Задача D. «Квантифікація предиката»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Задано прямокутну таблицю, що складається лише з нулів та/або одиниць. До цієї таблиці треба застосувати одну з таких дій:

- $A_i A_j$ або $A_j A_i$ (дія може бути позначена будь-яким з цих двох способів, при цьому смисл цих записів однаковий) — якщо геть усі комірки таблиці містять одиниці, то результат дії одиниця, в усіх інших випадках результат нуль;
- $E_i E_j$ або $E_j E_i$ (дія може бути позначена будь-яким з цих двох способів, при цьому смисл цих записів однаковий) — якщо хоча б десь у таблиці є хоча б одна одиниця, то результат дії одиниця, інакше результат нуль;
- $E_i A_j$ — якщо у таблиці є хоча б один рядок, що складається з самих лише одиниць, то результат дії одиниця, інакше результат нуль;
- $E_j A_i$ — якщо у таблиці є хоча б один стовпчик, що складається з самих лише одиниць, то результат дії одиниця, інакше результат нуль;
- $A_i E_j$ — якщо у кожному рядку таблиці є хоча б одна одиниця, то результат дії одиниця, інакше результат нуль;
- $A_j E_i$ — якщо у кожному стовпчику таблиці є хоча б одна одиниця, то результат дії одиниця, інакше результат нуль.

«Хоча б один/одна» скрізь означає, що можна 1, а можна й більше.

Напишіть програму, яка застосуватиме вказані дії до таблиці.

Вхідні дані. Перший рядок містить два цілі числа N та M (кожне від 1 до 12) — кількість рядків та кількість стовпчиків відповідно. Кожен з подальших N рядків містить по рівно M нулів та/або одиниць, розділених одинарними пробілами. Останній $(N+2)$ -й рядок містить потрібну дію, тобто один із записів $A_i A_j$, або $A_j A_i$, або $E_i E_j$, або $E_j E_i$, або $E_i A_j$, або $E_j A_i$, або $A_i E_j$, або $A_j E_i$.

Результати. Виведіть єдину цифру 0 або 1 — результат дії.

Приклади:

Вхід	Рез-ти
2 3 1 0 1 1 1 0 $A_i A_j$	0
2 3 1 0 1 1 1 0 $A_j A_i$	0

Вхід	Рез-ти
2 3 1 0 1 1 1 0 $E_i E_j$	1
2 3 1 0 1 1 1 0 $E_j E_i$	1

Вхід	Рез-ти
2 3 1 0 1 1 1 0 $E_i A_j$	0
2 3 1 0 1 1 1 0 $E_j A_i$	1

Вхід	Рез-ти
2 3 1 0 1 1 1 0 $A_i E_j$	1
2 3 1 0 1 1 1 0 $A_j E_i$	1

Оцінювання. Тести 1–8 є тестами з умови; вони перевіряються, але самі по собі не приносять балів. Решта тестів утворюють такі 14 блоків:

дія	$M = N = 2$	$1 \leq N \leq 12, \quad 1 \leq M \leq 12$
AiAj або AjAi	5 балів; тести 9–40; перевіряється й оцінюється завжди	20 балів; тести 145–148; перевіряється й оцінюється лише в разі успішного проходження тестів 9–40
EiEj або EjEi	5 балів; тести 41–72; перевіряється й оцінюється завжди	20 балів; тести 149–152; перевіряється й оцінюється лише в разі успішного проходження тестів 41–72
EiAj	5 балів; тести 73–88; перевіряється й оцінюється завжди	20 балів; тести 153–168; перевіряється й оцінюється лише в разі успішного проходження тестів 73–88
EjAi	5 балів; тести 89–104; перевіряється й оцінюється завжди	20 балів; тести 169–184; перевіряється й оцінюється лише в разі успішного проходження тестів 89–104
AiEj	5 балів; тести 105–120; перевіряється й оцінюється завжди	20 балів; тести 185–200; перевіряється й оцінюється лише в разі успішного проходження тестів 105–120
AjEi	5 балів; тести 121–136; перевіряється й оцінюється завжди	20 балів; тести 201–216; перевіряється й оцінюється лише в разі успішного проходження тестів 121–136
будь-яка	20 балів; тести 137–144; перевіряється й оцінюється лише в разі успішного проходження тестів 9–136	80 балів; тести 217–232; перевіряється й оцінюється лише в разі успішного проходження тестів 1–216

Скрізь мається на увазі, що для нарахування балів за блок Ваша програма повинна успішно пройти *всі* тести цього блоку.

Розбір задачі. Один з підходів — акуратно буквально реалізувати кожну окремо з описаних в умові задачі дій. Так розв'язувати можна, приклади саме таких розв'язків — ideone.com/7Vnypm (Pascal), ideone.com/z13Mo1 (Python3).

Праворуч наведено одну з шести гілок такого розв'язку. Внутрішній цикл перевіряє, чи складається поточний рядок з самих лише одиниць, тому зручно ініціалізувати `res_inner` одиницею і при знаходженні хоча б одного нуля в цьому рядку скинути `res_inner` в нуль. Зовнішній цикл перевіряє, чи є хоча б один рядок, для якого `res_inner` виходить одиницею, тому зручно ініціалізувати `res_all` нулем і при знаходженні такого рядка скинути `res_all` в одиницю.

```

if command = 'EiAj' then
begin
  res_all := 0;
  for i:=1 to N do begin
    res_inner := 1;
    for j:=1 to M do
      if A[i,j] = 0 then
        res_inner := 0;
    if res_inner = 1 then
      res_all := 1;
  end
end

```

Але така буквально реалізація виходить громіздкою і дещо схильною до технічних помилок; див. також міркування на цю тему в розборі задачі «Логічний куб» (стор. 33) та міркування в літературі та Інтернеті щодо т. зв. *code reusing*. Тому наведемо ще один спосіб, який далеко не в усіх смислах простіший, зате і потребує менше коду, і робить різні «дії» більш взаємопов'язаними.

1. Позбудемося того, що в частині «дій» треба аналізувати рядки, а в частині стовпчики. Для цього, якщо «дія» містить спочатку «j», потім «i», транспонуємо матрицю (віддзеркалимо відносно діагоналі, що виходить з верхнього лівого кута, так, щоб рядки стали стовпчиками, а стовпчики рядками); якщо ж «дія» містить спочатку «i», потім «j», нічого не робимо. Все це можна робити лише для EjAi та AjEi (де це важливо), а можна завжди, тобто також і для AjAi та EjEi (де це ні на що не впливає); і так, і так правильно.
2. Співставимо кожному окремо взятому рядку одну цифру 0 або 1:
 - Якщо передостанній символ «дії» «A», то одиниця вийде тільки якщо рядок складається з самих лише одиниць, в усіх інших випадках вийде нуль;
 - Якщо передостанній символ «дії» «E», то одиниця вийде, якщо рядок містить хоча б одну одиницю, й тільки якщо нема жодної, то вийде нуль.

Так на основі двовимірного масиву будується одновимірний, кожен елемент якого є результатом застосування «частини дії» до відповідного рядка.

3. До одновимірного масиву, утвореного внаслідок виконання попереднього пункту, застосуємо знов таке саме перетворення (настільки таке саме, що можна навіть викликати ту саму підпрограму з іншими аргументами). Тільки тепер дивимось не на передостанню в «дії» букву «A» чи «E», а на першу. Отриманий нолик чи одиничка і є остаточною відповіддю.

Приклад реалізації такого підходу — ideone.com/OTBJHA.

Чому все це правда? І чому задача так називається? Сами за правилами, описаними в умові задачі, можна знаходити значення квантифікації (застосування кванторів, тобто значків “ \forall ” («для всіх») та “ \exists ” («існує хоча б один»)) предиката, залежного від двох змінних: $AiAj$ та $AjAi$ описують (однаковий) смисл $\forall i\forall jP(i,j)$ та $\forall j\forall iP(i,j)$; $EiEj$ та $EjEi$ — (однаковий) смисл $\exists i\exists jP(i,j)$ та $\exists j\exists iP(i,j)$; $EiAj$ — смисл $\exists i\forall jP(i,j)$; $EjAi$ — смисл $\exists j\forall iP(i,j)$; $AiEj$ — смисл $\forall i\exists jP(i,j)$; $AjEi$ — смисл $\forall j\exists iP(i,j)$. А щойно описані трьома пунктами дії — інший спосіб знаходити ті самі квантифікації. Й це відомо тим, хто ретельно вивчив математичну логіку в обсязі першого курсу університету. Само собою, про це в принципі можна (але важче) і здогадатися, не знаючи наперед.

Як уже відзначалося, задачу можна розв’язати й без цих знань математичної логіки — вони дозволяють написати менш громіздкий варіант коду, але не необхідні. Програмістам часом доводиться реалізовувати не дуже зрозумілі правила, сформульовані спеціалістами з інших галузей, так що задача може розглядатись і як вправа на саме цю сторону програмістської діяльності.

3.16 II (районний/міський) етап 2018/19 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №67).

Задача А. «Кількість шляхів»

На рисунку зображено, через які проміжні пункти можливо дістатися з пункту A до пункту D . Підписи ребер означають «між цими пунктами є стільки-то різних рейсів». Скільки всього є способів дістатися з A до D ?

Рахувати треба всі способи, не намагаючись вибирати найкоротші чи ще якісь; переходи дозволені лише у відповідності з наведеними напрямками ребер; відповідь виразити як формулу від $k1, k2, k3, k4$.

Напишіть *вираз*, котрий знаходитиме цю кількість способів.

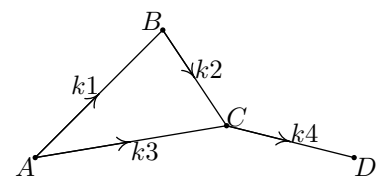
Змінні, від яких залежить вираз, обов’язково повинні мати саме описаний смисл, і називатися вони повинні саме $k1, k2, k3, k4$ (“ k ” — маленькі латинські).

У цій задачі треба здати не програму, а вираз: вписати його (сам вираз, не назву файлу) у відповідне поле відповідної сторінки ejudge і відправити на перевірку. Правила запису виразу: можна використовувати цілі десяткові числа, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), “/” (ділення дробове, наприклад, $17/5=3,4$), “//” (ділення цілочисельне, наприклад, $17//5=3$), круглі дужки “(” та “)” для групування та зміни порядку дій. Дозволяються пропуски (пробіли), але не всередині чисел. Множення треба писати явною зірочкою.

Наприклад: можна здати вираз $k1+k2+k3*k4$ і отримати 20 балів з 200, бо він неправильний, але все ж іноді відповідь випадково збігається з правильною. А «цілком аналогічні» вирази $a+b+c*d$ та $k1+k2+k3k4$ будуть оцінені на 0 балів кожен, бо змінні повинні називатися так, як вказано, а множення треба писати зірочкою, не пропускаючи.

Вкажемо також, що при $k1 = k2 = k3 = 2, k4 = 1$ кількість способів дорівнює 6, і цими шістьма способами є:

- 1-й зі способів $A \rightarrow C$, єдиний спосіб $C \rightarrow D$;
- 2-й зі способів $A \rightarrow C$, єдиний спосіб $C \rightarrow D$;
- 1-й зі способів $A \rightarrow B$, 1-й зі способів $B \rightarrow C$, єдиний спосіб $C \rightarrow D$;
- 1-й зі способів $A \rightarrow B$, 2-й зі способів $B \rightarrow C$, єдиний спосіб $C \rightarrow D$;
- 2-й зі способів $A \rightarrow B$, 1-й зі способів $B \rightarrow C$, єдиний спосіб $C \rightarrow D$;
- 2-й зі способів $A \rightarrow B$, 2-й зі способів $B \rightarrow C$, єдиний спосіб $C \rightarrow D$.



Розбір задачі. Спочатку розглянемо способи дістатися з A до C . З наведеного прикладу-переліку видно, що кожен із рейсів з A у B можна поєднувати з кожним із рейсів з B у C , тому треба *множити* $k_1 \cdot k_2$; до цих способів слід *додати* k_3 безпосередніх рейсів з A у C , бо з A до C можна добиратися або через B , або прямим рейсом. Так отримуємо $k_1 \cdot k_2 + k_3$. Далі, кожен з цих способів можна *поєднувати* з кожним з рейсів із C у D . Отже, остаточна відповідь: $(k_1 \cdot k_2 + k_3) \cdot k_4$.

Задача В. «Відтинання квадратів»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 0,5 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

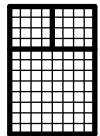
Від заданого прямокутника розміром $A \times B$ (натуральні числа) щоразу відрізається квадрат найбільшого розміру. Знайти число таких квадратів.

Вхідні дані. Два рядки, кожен з яких містить єдине натуральне число: у першому рядку A , тобто розмір по висоті; у другому рядку B , тобто розмір по ширині. Обмеження на значення A, B див. далі.

Результати. Єдине число — кількість квадратів.

Вхід	Рез-ти	Вхід	Рез-ти	Вхід	Рез-ти
12	3	2018	1	17	12
8		2018		42	

Примітки. У першому тесті маємо ситуацію з рисунку: спочатку відрізають квадрат 8×8 і лишається 4×8 , потім відрізають квадрат 4×4 і лишається 4×4 , що і є останнім третім квадратом. У другому тесті відразу маємо квадрат, відрізати взагалі не доводиться, але один квадрат (початковий) все-таки є.



Оцінювання. В цій задачі тести перевіряються й оцінюються незалежно. Половина балів припадає на тести, в яких обидва розміри від 1 до 100. Інша половина — на тести, в яких більший-або-рівний із розмірів від 1000 до 10^9 (мільярда), а менший-або-рівний від 1 до більшого-або-рівного.

Здавати треба одну програму, а не дві; Вас просто інформують, у скількох тестах які обмеження на розміри.

Розбір задачі. Для отримання 75% балів досить реалізувати відтинання квадратів один за одним «в лоб», наприклад, так, як праворуч: ініціалізуємо змінну, де накопичуватиметься результат, одиницею, бо колись буде останній квадрат, який вже не розрізатиметься, але рахуватиметься; поки прямокутник не став квадратом, відрізаємо квадрат (шляхом зменшення більшої зі змінних на значення меншої) і додаємо одиничку до кількості квадратів.

```
res:=1;
while a<>b do begin
  if a>b then a:=a-b
  else b:=b-a;
  res:=res+1
end
```

Само собою, в наведеному коді маєтися на увазі, що всі змінні цілочисельні 32-бітові (або 64-бітові), див. також стор. 13.

Але при деяких вхідних даних це надто довго працює. Скажімо, при $A = 1$, $B = 10^9$ буде аж мільярд ітерацій. А при, наприклад, $A = 999\,999\,998$, $B = 5$ ітерацій трохи менше, але теж дуже багато, і при цьому після дуже багатьох відтинань по 5×5 будуть ще кілька відтинань менших квадратів.

Легко бачити, що настільки велика кількість відтинань виникає, коли одне зі значень A, B дуже велике, а інше досить мале. Тоді кількість усіх квадратів однакового розміру (що відтинаються підряд) може бути обчислена цілочисельним діленням ($a \div b$ при $a \geq b$, або $b \div a$ при $a < b$); від сторони, що до всіх цих віднімань була більшою, залишиться $a \bmod b$ (якщо було $a \geq b$) або $b \bmod a$ (якщо було $a < b$). («div» та «mod» — позначення Pascal; div у Python3 позначається «/», більшістю решти мов — «/» за умови цілочисельності аргументів; mod більшістю мов позначається «%».)

Так що попередній код слід модифікувати, замінивши віднімання на mod, а додавання одинички — на додавання результату div. Через це, доводиться також змінити умову продовження циклу з «ще не квадрат» на «ще лишилось щось ненульове», а ініціалізацію з $res:=1$ на $res:=0$, бо останній квадрат тепер теж враховується при обчисленні div.

Насамкінець, усі ці відрізання квадратів відбуваються так само, як працює алгоритм Евкліда. Так що, якщо знати цей алгоритм (особливо, якщо обидві його версії — «класичну» та «сучасну»),

легше і здогадатися про заміну віднімання на mod, і згадати, що кількість ітерацій не перевищує $2 + \log_{\varphi} \min(A, B)$, де $\varphi = \frac{\sqrt{5}+1}{2} \approx 1,618$ є основою золотого перерізу (отже, при A, B до 10^9 , кількість ітерацій менша 50).
plus 2in

```
res:=0;
while (a>0)and(b>0) do
  if a>=b then begin
    res:=res + a div b;
    a:=a mod b;
  end else begin
    res:=res + b div a;
    b:=b mod a;
  end
```

Задача С. «Остання ненульова цифра»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 128 мегабайтів

Для натуральних (цілих строго додатних) чисел, факторіал можна отримати як добуток $N! = 1 \times 2 \times \dots \times N$. Напишіть програму, що знаходитиме останню ненульову цифру факторіала введенного натурального числа.

Вхідні дані. Єдине натуральне число N ($1 \leq N \leq 2018019$).

Результати. Остання ненульова цифра числа $N!$.

Приклади:

Вхід	Рез-ти
4	4

Вхід	Рез-ти
6	2

Вхід	Рез-ти
7	4

Примітки. $4! = 1 \times 2 \times 3 \times 4 = 24$, останньою ненульовою цифрою є 4.

$6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$, останньою ненульовою цифрою є 2.

$7! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 = 5040$, останньою ненульовою цифрою є 4.

Оцінювання. Тести з умови не приносять балів. Тести з рештою 20 значеннями з проміжку $1 \leq N \leq 23$ приносять по 2% балів кожен (сумарно 40%). Решта 60% балів поділені порівну (по 10%) між такими шістьма блоками: два блоки з обмеженнями $25 \leq N \leq 555$, два з обмеженнями $1234 \leq N \leq 54321$ та два з обмеженнями $123456 \leq N \leq 2018019$. Для кожного з блоків, бали нараховуються, лише якщо успішно пройдено *всі* тести цього блоку. Блоки оцінюються незалежно один від одного.

Здавати треба одну програму, а не різні програми для різних випадків; головна мета цього опису блоків — не даючи всіх деталей використаних тестів, дати приблизне уявлення про них.

Розбір задачі. Чому очевидний підхід набирає так

мало балів? Справді обчислювати сам факторіал і шукати його останню ненульову цифру — формально правильно, але набирає дуже мало балів. Факторіал дуже швидко зростає: $8! = 40320$ вже не поміщається у 16-бітовий знаковий цілий тип, $13! = 6227020800$ — у 32-бітовий, $21! = 51090942171709440000$ — у 64-бітовий. Так що для мов програмування, в яких найбільшим цілочисельним типом є 64-бітовий, цей підхід правильний лише для $N \leq 20$.

Межу $N \leq 20$ можна збільшити до $N \leq 23$, якщо щоразу, коли додається нуль наприкінці, тут же відкидати його: $1 \xrightarrow{\times 2} 2 \xrightarrow{\times 3} 6 \xrightarrow{\times 4} 24 \xrightarrow[\text{/10}]{\times 5} 12 \xrightarrow{\times 6} 72 \xrightarrow{\times 7} 504 \xrightarrow{\times 8} 4032 \xrightarrow{\times 9} 36288 \xrightarrow[\text{/10}]{\times 10} 36288 \xrightarrow{\times 11} \dots$

Деякі з дозволених на олімпіаді мов програмування мають вбудовану довгу арифметику, тобто вміють працювати зі значно більшими цілими числами: Python — просто; Java — через бібліотечний тип BigInteger. (C# теж має тип BigInteger, але там усе сумніше, бо засоби, якими BigInteger підключається на локальному Windows-комп'ютері, не працюють на Linux-сервері ejudge, і авторам збірника невідомо ні як їх підключити там, ні, навіть, чи можливо це взагалі.) Використання довгої арифметики (байдуже, готової чи написаної власноруч) дозволяє дещо розширити діапазон N , але недостатньо: з нею повинні проходити не лише всі тести з $N \leq 23$, а й усі блоки тестів з $N \leq 555$. А подальші блоки повинні не проходити: наприклад, $50000!$ має понад 200 тис. десяткових цифр, так що обчислення не мають шансів поміститися в обмеження за часом.

То що ж робити? Раз питають *лише останню* ненульову цифру, *нема потреби* обчислювати та зберігати *всі* цифри факторіалів. Але це й не класична «модульна арифметика» (в якій досить зберігати лише останню цифру), бо питають *останню ненульову*, кількість нулів наприкінці $N!$ зі зростанням N час від часу збільшується, й потрібний розряд час від часу зсувається: при $1 \leq N \leq 4$ це розряд одиниць (остання цифра), при $5 \leq N \leq 9$ — розряд десятків (передостання цифра), тощо.

Спробуємо з'ясувати, коли скільки нулів додається наприкінці факторіалу. Важливо, що розглядаємо саме десяткову систему числення ($10 = 2^1 \cdot 5^1$) і саме $N! = 1 \times 2 \times \dots \times N$. Яке б не було

число вигляду $5^a \times b$, число $2^a \times b$ (при тих самих додатних a, b) існує і менше $5^a \times b$. Так що, при послідовному домноженні $(1 \times 2 \times \dots \times (N-1)) \times N$, на кожну п'ятірку, скільки б їх не було в розкладенні N , «вже чекає» відповідна кількість двійок, наявних у розкладенні $(N-1)!$ і ще не використаних іншими п'ятірками. Так що кожне число, яке можна подати як $5^a \times b$ (де b не кратне 5) додає наприкінці факторіала рівно a нулів.

Спосіб №1 отримати повні бали (крім Python). Оскільки $5^9 = 1\,953\,125 < 2\,018\,019 < 9\,765\,625 = 5^{10}$, за одне домноження з'являтиметься щонайбільше дев'ять нулів. І ніщо не заважає поєднати такі ідеї: (1) відкидати нулі наприкінці, як тільки вони з'являються; (2) зберігати (у 64-бітовій змінній) останні 10–12 десяткових цифр. Якось так: $1 \xrightarrow{\times 2} 2 \xrightarrow{\times 3} 6 \xrightarrow{\times 4} 24 \xrightarrow{\times 5} 12 \xrightarrow{\times 6} 72 \xrightarrow{\times 7} 504 \xrightarrow{\times 8} 4032 \xrightarrow{\times 9} 36288 \xrightarrow{\times 10} 36288 \xrightarrow{\times 11} 399168 \xrightarrow{\times 12} 4790016 \xrightarrow{\times 13} 62270208 \xrightarrow{\times 14} 871782912 \xrightarrow{\times 15} 1307674368 \xrightarrow{\times 16} 922789888 \xrightarrow{\times 17} 5687428096 \xrightarrow{\times 18} 2373705728 \xrightarrow{\times 19} 5100408832 \xrightarrow{\times 20} 200817664 \xrightarrow{\times 21} 4217170944 \xrightarrow{\times 22} 2777760768 \xrightarrow{\times 23} 3888497664 \xrightarrow{\times 24} 3323943936 \xrightarrow{\times 25} 830985984 \xrightarrow{\times 26} 1605635584 \xrightarrow{\times 27} \dots$

Для $N < 5^{10}$, 10–12 цифр водночас і досить багато, щоб не втратити зразу всі ненульові цифри, і досить мало, щоб при черговому домноженні ніколи не виникало переповнення 64-бітового типу.

Спосіб №2 отримати повні бали (крім Python). Вже відзначено, що класична модульна арифметика (зберігати лише останню цифру) незастосовна, бо питають останню *ненульову*, й потрібний розряд час від часу зсувається. Але можна помітити, що раз кількість нулів наприкінці визначається кількістю п'ятірок, то, прибравши з $N!$ відповідну кількість п'ятірок і таку саму кількість двійок, отримаємо якраз факторіал крім нулів наприкінці; для цього факторіала крім нулів наприкінці можна зберігати й обчислювати саму лише останню цифру.

Конкретніше, будемо при обчисленні $1 \times 2 \times \dots \times i \times \dots$ «виймати» з кожного числа всі множники 2 та 5, і рахувати окремо останню цифру добутку *всього*, крім «вийнятих» двійок та п'ятірок, окремо різницю кількостей «вийнятих» (на скільки двійок більше, ніж п'ятірок). Якось так:

Було	Множимо на	Коментар	Стає	
1	0	$2 = 2^1 \cdot 5^0 \cdot 1$	двійок стає на одну більше	1
1	1	$3 = 2^0 \cdot 5^0 \cdot 3$	кількості двійок та п'ятірок незмінні, добуток решти домножується на 3	3
3	1	$4 = 2^2 \cdot 5^0 \cdot 1$	двійок стає на дві більше	3
3	3	$5 = 2^0 \cdot 5^1 \cdot 1$	«зайвих» двійок стає на одну менше, бо одна двійка «зв'язується» п'ятіркою	2
3	2	$6 = 2^1 \cdot 5^0 \cdot 3$	і двійок стає на одну більше, і добуток решти домножується на 3	9
9	3	$7 = 2^1 \cdot 5^0 \cdot 7$	кількості двійок та п'ятірок незмінні, добуток решти домножується на 7, зберігаємо з 63 лише останню цифру 3	3
3	3	$8 = 2^3 \cdot 5^0 \cdot 1$	двійок стає на три більше	3
3	6	$9 = 2^0 \cdot 5^0 \cdot 9$		7
7	6	$10 = 2^1 \cdot 5^1 \cdot 1$	додаються одна двійка й одна п'ятірка, різниця кількостей незмінна	7
7	6	$11 = 2^0 \cdot 5^0 \cdot 11$		7
7	6	$12 = 2^2 \cdot 5^0 \cdot 3$		1
1	8	$13 = 2^0 \cdot 5^0 \cdot 13$		3
3	8	$14 = 2^1 \cdot 5^0 \cdot 7$		1
1	9	$15 = 2^0 \cdot 5^1 \cdot 3$	«зайвих» двійок стає на одну менше, бо одна двійка «зв'язується» п'ятіркою; крім того, добуток решти домножується на 3	3

І так далі. Дорахувавши до множника N (включно), знаємо останню цифру « $N!$ без двійок та п'ятірок» (позначимо як d) та кількість «вільних» (не «зв'язаних» п'ятірками) двійок (позначимо як k). Остаточна відповідь може бути виражена як $(d \cdot 2^k) \bmod 10 = (d \cdot (2^k \bmod 10)) \bmod 10$. Враховуючи, що $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$, $2^8 = 256$, \dots , тобто при $k > 0$ останні цифри циклічно повторюються, велике k можна замінити на $(k \bmod 4) + 4$, що дає можливість обчислювати $2^k \bmod 10$ за $\Theta(1)$. Втім, це мало на що впливає, бо основний процес домножень (із «вийманням» з кожного числа двійок та п'ятірок) значно довший.

Який з цих способів кращий? Важко сказати. Вони обидва мають складність $\Theta(N)$. (Може здатися, ніби для другого є ще внутрішні цикли «викидання» двійок і п'ятірок, які дають додатковий множник $\log N$; але це не так. Наприклад, сумарну кількість ітерацій, які шукають двійки, можна виразити як $N \operatorname{div} 2 + N \operatorname{div} 2^2 + \dots + N \operatorname{div} 2^k$, де $k = \lfloor \log_2 N \rfloor$; див. аргументацію формули (1) на стор. 20. Формула суми геометричної прогресії каже, що ця сума менша N .) При вказаних обмеженнях, обидва ці способи гарантовано проходять усі тести (всіма доступними на олімпіаді мовами програмування, крім *Python*; див. наступний абзац). Теоретичний недолік першого способу — якби дозволялися значення $N \geq 5^{11} \approx 48,8$ млн, то 64-бітового типу вже було б недостатньо, щоб задовольнити обидві вимоги «не втратити зразу всі ненульові цифри» та «не утворювати переповнень»; другому ж способу цілком достатньо як 64-бітового, так і навіть 32-бітового типу аж до $N \approx \frac{2^{31}}{9} \approx 238$ млн (але при $N \approx 10^8$ обчислення вже тривали б десятки секунд).

Навіщо скривдили Python? Він сам себе скривдив. Програмам усіма мовами програмування виділявся однаковий ліміт часу. Але для розрахунків, вжитих в обох цих способах, Python виявився значно повільнішим не лише за традиційно швидкі мови *g++* та *frs*, а також і за відносно повільні *java*, *rasabc-linux* та *mcs*. Враховуючи цю об'єктивну і значну різницю швидкодії, перед туром було прийняте свідоме рішення змиритися з тим, що реалізації одного алгоритму різними мовами набирають різні бали. З точки зору правил та традицій Всеукраїнської олімпіади з інформатики, це погано, але в такого роду виключних випадках допускається. До того ж, у цій самій задачі Python дозволяв отримати бали за тести вигляду $25 \leq N \leq 555$, пишучи «лобовий» розв'язок і взагалі не замислюючись над тим, що така програма використовує довгу арифметику. Учасники, що вміють писати лише на *Pascal* чи *C++*, такої можливості не мали, і їм це ніяк не компенсувалося. Тож Python виграє в одному, програє в іншому, й навряд чи було б справедливим надавати йому додаткові переваги, збільшуючи ліміт часу. Тим паче, що існує ще наступний спосіб.

Спосіб №3 отримати повні бали (включно з Python). Обмеження $N \leq 2018019$ підбиралося під те, щоб переважною більшістю мов працювали й набирали повний бал способи 1–2. Але задачу можна розв'язати ще значно ефективніше, якщо правильно модифікувати початий у способі №2 підхід «двійки компенсують п'ятірки, для решти рахуємо останню цифру». Вважаємо гарантованим $N \geq 10$, бо для $N < 10$ можна написати окремий *if*, щоб рахувати «в лоб». Виділимо окремо множники, кратні 5, окремо решту, причому решту розіб'ємо на групи згідно десятків. Наприклад, для $37!$ буде так: $(1 \times 2 \times 3 \times 4 \times 6 \times 7 \times 8 \times 9) \times (5 \times 10) \times (11 \times 12 \times 13 \times 14 \times 16 \times 17 \times 18 \times 19) \times (15 \times 20) \times (21 \times 22 \times 23 \times 24 \times 26 \times 27 \times 28 \times 29) \times (25 \times 30) \times (31 \times 32 \times 33 \times 34 \times 36 \times 37) \times (35)$. Добуток $(5 \times 10) \times (15 \times 20) \times (25 \times 30) \times (35)$ можна подати як $((5 \cdot 1) \times (5 \cdot 2)) \times ((5 \cdot 3) \times (5 \cdot 4)) \times ((5 \cdot 5) \times (5 \cdot 6)) \times (5 \cdot 7) = 5^7 \times 7!$. (Важливо, що з кожного числа, кратного 5, виділяється рівно одна п'ятірка, навіть якщо, як з 25, їх можна було б виділити більше.) Так можна робити й при інших N , отримуючи $5^{N \operatorname{div} 5} \times (N \operatorname{div} 5)!$. Тобто, з початкової задачі для $N!$ можна виділити таку саму задачу для $(N \operatorname{div} 5)!$, і при цьому решта множників поділені на більш-менш зручні групи по 8 з кожного десятку (в останній групі, й лише у ній, може бути менше). Тож будемо окремо заново розв'язувати всю задачу для $(N \operatorname{div} 5)!$, і окремо компенсувати п'ятірки, зібрані в $5^{N \operatorname{div} 5}$, двійками з решти добутків.

Тобто, $(N \operatorname{div} 5)$ двійок треба «вийняти» з тих решти добутків. І міркувати «щоб вийняти множник з добутку, поділимо цей добуток на цей множник» тут не можна, бо в модульній арифметиці це часто не так. Наприклад, $26 \bmod 10 = 6 = 56 \bmod 10$, тобто з точки зору остач за модулем 10 числа 26 та 56 однаковісенькі, але $\frac{26}{2} \bmod 10 = 13 \bmod 10 = 3 \neq 8 = \frac{56}{2} \bmod 10$.

Спробуємо компенсувати дві п'ятірки з (5×10) двома двійками, «вийняти» з $(1 \times 2 \times 3 \times 4 \times 6 \times 7 \times 8 \times 9)$, дві п'ятірки з (15×20) двома двійками з $(11 \times 12 \times 13 \times 14 \times 16 \times 17 \times 18 \times 19)$, тощо. До того, що остання група може бути неповна, повернемося пізніше. Тим, що в кожних таких 8-множникових дужках можна знайти більше, ніж дві, двійки, знехтуємо: їх більше, але «вийматимемо» дві.

Візьмемо число, що закінчується цифрою 2, та число, що закінчується цифрою 4, й «виймемо» з кожного по одній двійці, поділивши кожне окремо на 2 (ділимо самі числа, не остачі). Можуть бути рівно два випадки: або ці числа являють собою $20k + 2$ та $20k + 4$, або $20k + 12$ та $20k + 14$; у першому випадку після ділень виходить $10k + 1$ та $10k + 2$, у другому $10k + 6$ та $10k + 7$. Оскільки в межах цих 8-множникових дужок більше не треба ділень, можна вертатися до модульної арифме-

тики, відкидати $10k$ і казати, що $1 \times 2 = 2$ та $6 \times 7 = 42$ закінчуються на однакову цифру 2. Лишається домножити це 2 на останні цифри решти шести множників у дужках (1, 3, 6, 7, 8, 9), і отримати, що остання цифра добутку (після «виймання» двох двійок) дорівнює $(2 \cdot 1 \cdot 3 \cdot 6 \cdot 7 \cdot 8 \cdot 9) \bmod 10 = 4$. Остання цифра загального добутку всіх $N \div 10$ штук таких дужок рівна $4^{N \div 10} \bmod 10$. Оскільки $4^1 = 4$, $4^2 = 16$, $4^3 = 64$, $4^4 = 256$, ..., а розглядаємо випадок $N \geq 10$, тобто $(N \div 10) \geq 1$, значення $4^{N \div 10} \bmod 10$ можна знайти якимось у стилі `if ((N div 10) mod 2) = 1 then a:=4 else a:=6`.

Повернемося до врахування того, що при $N \bmod 10 \neq 0$ існує ще остання дужка, яка рахується не за тими правилами, що решта дужок. Якщо $1 \leq N \bmod 10 \leq 4$, то цій дужці не треба компенсувати ніяку п'ятірку, можна просто перемножити в модульній арифметиці останні цифри; інакше кажучи, $(N \bmod 10)! \bmod 10$. Якщо ж $5 \leq N \bmod 10 \leq 9$, то треба компенсувати рівно одну п'ятірку. (Навіть якщо число останнього неповного десятку, що закінчується на 5, кратне деякому 5^k при більшому k , то всі п'ятірки, крім однієї, перейшли у $(N \div 5)!$.) Поділимо на 2 число, що закінчується цифрою 2; можливі рівно два випадки: або воно вигляду $20k + 2$, або $20k + 12$; у першому отримуємо $10k + 1$, у другому $10k + 6$; більше ділень не треба, можна вертатися до модульної арифметики. Точно треба домножити на 3 та 4 (бо розглядаємо випадок $5 \leq N \bmod 10 \leq 9$); отримуємо або $(1 \times 3 \times 4) \bmod 10 = 12 \bmod 10 = 2$, або $(6 \times 3 \times 4) \bmod 10 = 72 \bmod 10 = 2$, тобто в обох випадках 2. Множити на 5 не треба й не можна, ця п'ятірка вже врахована іншими засобами. Лишається тільки домножити в модульній арифметиці на 6, 7, ..., $N \bmod 10$ (зокрема, при $N \bmod 10 = 5$, не домножувати ні на що).

Дії попереднього абзацу насправді рівносильні таким: «порахувати “в лоб” $(N \bmod 10)!$; якщо кратний 10, поділити на 10; взяти останню цифру». Це навіть простіше писати. Але неясно, як строго доводити правильність цього, не спираючись на попередній абзац.

Само собою, треба помножити (в модульній арифметиці) результат (поза)минулого абзацу на раніше отриманий (рівний або 4, або 6) результат $4^{N \div 10} \bmod 10$, а також домножити (в модульній арифметиці) цей добуток на $(N \div 5)! \bmod 10$, розв'язавши всю задачу заново з аргументом $N \div 5$ замість N . Це повторюватиметься приблизно $\log_5 N$ разів. Що й задає асимптотичну оцінку всього розв'язку $\Theta(\log N)$, бо все інше виконується за $\Theta(1)$ (щось if-ом, щось циклом, але з дуже малою кількістю ітерацій). Так що задачу в принципі можна було давати і для значно більших N . Втім, вона і з обмеженням $N \leq 2018019$ виявилася дещо заскладною для переважної більшості учасників.

Задача D. «Палички, хто переможе?»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 128 мегабайтів

Є одна купка, яка спочатку містить N паличок. Двоє грають у таку гру. Спочатку перший може забрати з купки або 1, або 2 палички. На кожному подальшому ході кожен з гравців може забрати будь-яку кількість паличок від 1 до подвоєної кількості, щойно забраної суперником (обидві межі включно). Інших варіантів ходу нема. Ходять гравці по черзі, пропускати хід не можна. Виграє той, хто забирає останню паличку (можливо, разом з деякими іншими).

Напишіть програму, яка визначатиме, хто виграє при правильній грі обох гравців. Іншими словами, хто з гравців може забезпечити собі виграш, хоч би як не грав інший.

Вхідні дані. Єдине ціле число N — початкова кількість паличок у купці ($2 \leq N \leq 1234567$).

Результати. Єдине ціле число, або 1 (якщо перший гравець може забезпечити собі виграш, як би не грав другий), або 2 (якщо другий, як би не грав перший).

Примітка. Ці приклади частково пояснені також у прикладах до наступної задачі.

Оцінювання. 1-й блок (тести 1–5) містить тести з умови, перевіряється завжди, але безпосередньо не оцінюється. 2-й блок (тести 6–10) містить усі значення з діапазону $6 \leq N \leq 10$ і оцінюється у 30% балів; перевіряється й оцінюється завжди. 3-й блок (тести 11–20) має обмеження $11 \leq N \leq 100$

Приклади:

Вхідні дані	Результати
2	1
3	2
4	1
5	2
1024	2

й оцінюється у 20% балів; перевіряється й оцінюється завжди. 4-й блок (тести 21–30) має обмеження $101 \leq N \leq 1234$ й оцінюється у 20% балів; перевіряється й оцінюється завжди. 5-й блок (тести 31–40) має обмеження $12345 \leq N \leq 43210$ і оцінюється у 15% балів; перевіряється й оцінюється, лише якщо успішно пройдено всі попередні блоки. 6-й блок (тести 41–50) має обмеження $123456 \leq N \leq 1234567$ і оцінюється у 15% балів; перевіряється й оцінюється, лише якщо успішно пройдено всі попередні блоки. Для кожного окремо взятого блоку, бали нараховуються, лише якщо успішно пройдено *всі* тести блоку.

Розбір задачі. Хоч ця та наступна задачі схожі з задачею «Палички» зі стор. 84–86, вони *значно складніші*. Там аналіз виграшності зводився до дуже простої ознаки «чи кількість паличок кратна 4?». А тут простої ознаки не виходить, і треба думати, спочатку — як пристосувати до цієї задачі загальні правила перевірки виграшності, потім — як це суттєво оптимізувати.

Нагадаємо, що позиція виграшна, коли той, кому вона дісталася, або самим цим фактом вже виграв, або має хоча б один хід у програшну. Іншими словами: якщо гравець знає, який це хід (або які це ходи), і використає саме його (або один з них), то супернику дістанеться програшна позиція. Аналогічно, позиція програшна, коли той, кому вона дісталася, або самим цим фактом вже програв, або абсолютно всі його ходи ведуть у виграшні позиції. Іншими словами: хоч би як добре гравець не вмів грати, він не має ніяких інших ходів, крім як походити в таку позицію, що починаючи з неї суперник може виграти.

Кожна позиція або виграшна, або програшна; не може бути ні третього варіанту, ні поєднання обох цих варіантів для однієї позиції. (Охочі можуть знайти в літературі чи Інтернеті доведення цього твердження. Воно правильне не для абсолютно всіх ігор, але достатньо (одночасного) виконання таких умов: гравців двоє; гравці мають повну інформацію про правила гри; гравці мають повну інформацію про поточну позицію гри; гравці вибирають ходи з зарані відомих можливостей, і ці можливості не залежать ні від чого, крім поточної позиції; гра завершується перемогою одного з гравців і поразкою іншого, нема ні нічийх, ні числового вираження, «наскільки програв»; кількість позицій гри скінченна; гра не може зациклюватися, багатократно проходячи через ті самі позиції.)

З усього цього випливає, що позицією цієї гри не можна вважати кількість паличок. Хоча б тому, що в перших двох прикладах наступної задачі показано неможливість виграти, ходячи першим при $N = 3$, а в останньому прикладі наступної задачі суперник забирає всі три палички одним ходом і виграє. Це суперечить умові «гравці вибирають ходи з зарані відомих можливостей, і ці можливості не залежать ні від чого, крім поточної позиції».

Позицією цієї гри можна і варто вважати пару «кількість паличок, що лишилися у купці; кількість паличок, забраних на наразі останнього ходу». Причому, хоча на початку гри «наразі останнього ходу» не було, дозволені ходи (забирати або одну, або дві палички) такі самі, які були б, якби щойно забрали одну паличку й лишилося N . Тож початковою є позиція $(N; 1)$; якщо 1-й гравець забирає одну паличку, то 2-му дістається позиція $(N-1; 1)$, а якщо дві, то позиція $(N-2; 2)$. І так далі.

На перший погляд, висновок «позицією слід вважати пару ...» украй погано поєднується з обмеженнями задачі: якщо $N \approx 1234567$, а позицією є пара, то кількість позицій перевищує $10^{12}/4$, що не вкладається ні в обмеження пам'яті, ні в обмеження часу. Але дослідити, що вийде, якщо запрограмувати стандартний аналіз виграшності/програшності для позицій-пар, варто. Хоча б тому, що така сама гра розглядається й у наступній задачі, а там обмеження менші. Але варто було б, навіть якби наступної задачі взагалі не було. Наяк, навіть неефективну правильну програму можна здати, щоб отримати бали хоча б за деякі з блоків. Але є й важливіша причина: буває, що ручний аналіз результатів, згенерованих менш ефективним розв'язком, дозволяє помітити деякі особливості, на основі яких можна придумати більш ефективний розв'язок. Так що дивимося на сукупності виграшних/програшних позицій, зображені внизу стор. 95.

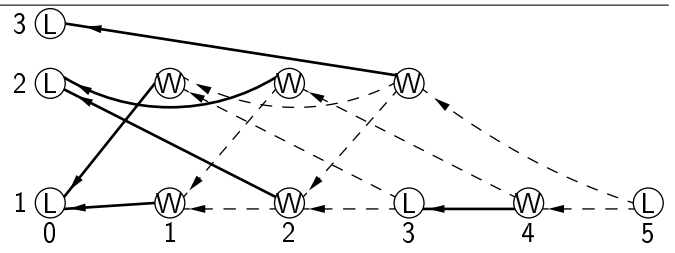
Легко помітити, що позиції $(0; s)$ програшні при всіх s , бо $n = 0$ означає, що позиція дістається після того, як суперник вже забрав останню паличку і виграв. Також легко помітити, що у випадку $n > 0$ всі позиції $(n; s)$ при $s \geq \frac{n}{2}$ виграшні (бо при $s \geq \frac{n}{2}$ можна забрати всі палички за один хід)

А ще можна помітити таке: *якщо деяка позиція $(n^*; s^*)$ виграшна, то виграшні й усі позиції $(n^*; s)$ при тому самому n^* та $s \geq s^*$* (на діаграмах і в таблиці — кожен стовпчик (крім $n = 0$) або складається з самих лише “W”, або має внизу скінченну кількість “L”, а вище самі лише “W”).

Доведемо, що це не збіг обставин для саме цих позицій, а так і є для всіх $n > 0$. Куди можна піти з позиції $(n; 1)$? Лише в $(n - 1; 1)$ (при всіх $n > 0$) та в $(n - 2; 2)$ (при $n \geq 2$). А куди з $(n; 2)$? При $n \leq 2$, в ті самі позиції; при $n \geq 3$ до них додається $(n - 3; 3)$, а при $n \geq 4$ також $(n - 4; 4)$. Порівнюючи ходи з $(n; 2)$ та з $(n; 3)$, знов бачимо, що всі ходи з $(n; 2)$ лишаються допустимими, і до них додаються (якщо n досить велике) нові $(n - 5; 5)$ та $(n - 6; 6)$. І так для всіх збільшень s при сталому n : всі старі ходи лишаються, і до них, можливо, додаються нові. В цій грі, позиція може бути виграною лише за рахунок того, що з неї існує хід у деяку програшну позицію. Отже, раз цей самий хід можливий і при всіх більших s , всі позиції з більшим s при тому самому n теж виграні.

Це дає ключ до такого розв'язку. Реалізуємо стандартний аналіз «позиція виграшна, коли є хоча б один хід у програшну», але не зберігаючи кожен пар $(n; s)$, а подаючи інформацію в одно-

Зобразимо всі позиції для початкових кількостей паличок 5 (верхня діаграма) та 12 (нижня). Позначимо позиції кружечками й розмістимо так, щоб стовпчики відповідали кількості паличок, що лишилися, рядки — кількості паличок, забра-
них на останньому



ході. Літери у кружечках позначають, виграшна (Win) позиція чи програшна (Lose). Жирні стрілки позначають переходи, які ведуть до програшних позицій і тим забезпечують виграшність; пунктирні — інші переходи. Верхня діаграма є частиною (лівою нижньою) нижньої; це природньо, бо раз подальші ходи з позиції не залежать від того, як потрапили в цю позицію, то не залежить і виграшність / програшність.

Далі зображено самі позначки W/L (без стрілок) для ще більшої кількості позицій, та вказано мінімальні s , починаючи з яких позиції стають виграшними.

[illegible]

вимірному масиві `minWinS` з діапазоном індексів $[0..N]$ (обидві межі включно), де `minWinS[i]=k` означає, що всі позиції $(i; 1), (i; 2), \dots, (i; k-1)$ (якщо вони взагалі існують, тобто $k > 1$) програшні, а всі позиції $(i; k), (i; k+1), \dots$ виграшні.

По суті, останній рядок стор. 95 і є таким масивом.

Остаточну відповідь можна визначити так: якщо `minWinS[N] = 1`, то позиція $(N; 1)$ виграшна (виграє 1-й гравець), інакше програшна (2-й).

Визначити асимптотичну складність цього розв'язку важко. Дивитися на верхню межу внутрішнього циклу `for s:=1 to (i+1) div 2 do...` і робити висновок про «завелику для $N \approx 10^6$ складність $O(N^2)$ » нема толку, бо хоч цю межу й підібрано з осмисленого міркування «при $2 \cdot s \geq i$ вже можна забрати всі і паличок одним ходом, тож ще більші s не потрібні», фактично цей цикл завжди обривається `break`-ом, і справжня кількість його ітерацій незрозуміла. Експериментально, складність всього алгоритму схожа на $\Theta(N \log N)$ з малим константним множником. І цього досить, щоб пройти всі тести всіх блоків.

При бажанні можна побудувати й ще значно ефективніший алгоритм. Продовживши аналіз таблиці з виграшними та програшними позиціями та/або масиву `minWinS`, можна помітити, що особливо великі значення у `minWinS` досягаються на індексах, рівних числам Фібоначчі. Тож можна спробувати, що буде, якщо записувати значення N у т. зв. «фібоначчієвій системі числення» (що це таке, зайдіть самостійно), й отримати, що 2-й гравець може забезпечити собі виграш при тих і тільки для тих початкових значеннях N , які у фібоначчієвій системі закінчуються на хоча б два нулі. Як це строго довести і чи можна це використати також і в наступній задачі (де значно більша потреба працювати з $s > 1$), нехай залишиться за межами цього збірника.

Насамкінець, якщо визначити, що позиціями гри є пари $(n; s)$, але не помітити властивість «якщо деяка позиція $(n^*; s^*)$ виграшна, то виграшні й усі позиції $(n^*; s)$ при тому ж n^* та $s \geq s^*$ », може мати смисл реалізувати стандартну перевірку позицій на виграшність/програшність рекурсією із запам'ятовуваннями, зберігаючи результати вже переглянутих позицій не у двовимірному масиві, а в шар-і (в деяких мовах програмування це називається `dictionary`, в літературі поширена також назва «асоціативний масив»), ключами якого є такі пари $(n; s)$. При грамотній реалізації (зокрема, оголошувати позицію виграшною, щойно знайшовши перший хід у програшну, а не крутити цикл завжди до кінця) значна частина з $\Theta(N^2)$ позицій, теоретично досяжних з позиції $(N; 1)$, практично не розглядаються, тож не потрапляють у шар, тож такий розв'язок *можна* написати так, що він пройде передостанній блок тестів ($12345 \leq N \leq 43210$). Але пройти так останній блок, начебто, неможливо.

Задача Е. «Палички, інтерактивна гра»

Вхідні дані: Клавіатура (stdin) Обмеження часу: 1 сек
Результати: Екран (stdout) Обмеження пам'яті: 64 мегабайти

Суть гри в точності та сама, що у попередній задачі.

Напишіть програму, яка інтерактивно гратиме за першого гравця.

На початку, один раз, Ваша програма повинна прочитати одне ціле число в окремому рядку — початкову кількість паличок N ($2 \leq N \leq 12345$). Потім вона повинна повторювати такий цикл:

1. Вивести єдине число в окремому рядку — свій хід, тобто кількість паличок, які вона зараз забирає з купки. На першому ході це повинно бути 1 або 2, на подальших — ціле число від 1 до подвоєної кількості, щойно забраної програмою-суперницею, причому не більше за поточну кількість паличок у купці.
2. Якщо після цього купка стає порожньою, вивести окремим рядком фразу "I won!" (без лапок, символ-у-символ згідно зразку) і завершити роботу.
3. Прочитати хід програми-суперниці, тобто кількість паличок, які вона зараз забирає з купки (єдине число, в окремому рядку). Якщо Ваша програма правильно визначила, що гра не закінчилася

і цей хід відбудеться, то гарантовано, що він допустимий (число є цілим від 1 до подвоєної кількості, щойно забраної Вашою програмою, і не перевищує поточну кількість паличок у купці).

4. Якщо після цього купка стає порожньою, вивести окремим рядком фразу "You won..." (без лапок, символ-у-символ згідно зразку) і завершити роботу.

Ці дії повинні повторюватися, доки якась із програм-гравців не виграє. Програма-суперниця не виводить фраз "I won!" / "You won..." чи якихось їх аналогів.

Ця задача є інтерактивною: Ваша програма не отримує всіх вхідних даних на початку, а отримуватиме по мірі виконання доуточнення, що залежатимуть від попередніх дій Вашої програми. Тим не менш, її перевірка буде автоматичною, тому слід чітко дотримуватися формату спілкування з програмою-суперницею.

Настійливо рекомендується, щоб Ваша програма після кожного свого виведення робила дію `flush(output)` (Pascal), вона ж `cout.flush()` (C++), вона ж `fflush(stdout)` (C), вона ж `sys.stdout.flush()` (Python), вона ж `System.out.flush()` (Java). Це істотно зменшує ризик, що проміжна відповідь «загубиться» десь по дорозі, не дійшовши до програми-суперниці.

Оцінювання. Тести оцінюються кожен окремо (без блоків). У 1-му тесті $N=3$, у 2-му $N=5$, і ці тести не приносять балів. Решта тестів приносять однакові бали. У 20% тестів $2 \leq N \leq 25$ ($N \neq 3$, $N \neq 5$), програма-суперниця ідеальна (не робить помилок). Ще у 20%, $100 < N \leq 1234$, суперниця ідеальна. Ще у 20%, $1234 < N \leq 12345$, суперниця ідеальна. Ще у 20%, $100 < N \leq 1234$, суперниці інші. Ще у 20%, $1234 < N \leq 12345$, суперниці інші. Ці інші програми-суперниці (їх кілька різних) роблять ходи, де гарантовано дотримані вимоги «забирати лише від 1 палички до подвоєної щойно забраної кількості» та «забирати не більше паличок, ніж є у купці», але дотримуються кожна власних уявлень, як треба грати, частенько вибираючи не найкращий з допустимих ходів.

Буде оцінюватися і вміння Вашої програми виграти там, де це точно можливо, і вміння Вашої програми достойно, згідно правил, програти, де вигравш неможливий, і вміння Вашої програми скористатися (теж згідно правил) помилками чи іншими неадекватностями програми-суперниці, якщо такі будуть. Якщо Ваша програма програє там, де могла виграти, зате зробить це з дотриманням усіх вимог, відповідний тест буде оцінено на 1 бал з 5. За будь-яке порушення правил гри з боку Вашої програми, відповідний тест буде оцінено на 0 балів.

Приклади:

Вхідні дані	Результати	Примітки
3 2	1 You won...	У купці спочатку 3 палички. Ваша програма забирає одну, лишається дві; програма-суперниця забирає обидві й виграє.
3 1	2 You won...	Спробуємо забрати не одну, а дві з трьох паличок, тобто лишити одну; програма-суперниця забирає її і теж виграє.
5 1 2	1 1 You won...	У купці спочатку п'ять паличок. Ваша програма забирає одну, лишається чотири; програма-суперниця забирає одну, Вашій програмі дістається три палички, й вона ніяк не може виграти з вищеописаних причин.
5 3	2 You won...	Спробуємо забрати дві з п'яти паличок (лишається три); програма-суперниця забирає всі три (має право, бо Ваша програма щойно взяла дві) й теж виграє.

Примітки. (1) Всі наведені послідовності ходів є прикладами правильної гри. Ваша програма не зобов'язана при різних запусках для однієї початкової кількості паличок робити різні ходи. Але вона має таке право. Якщо Ваша програма при різних запусках грає по-різному, система автоматичної перевірки не шукатиме ні найкращий, ні найгірший з результатів, а просто оцінюватиме перший. (2) Вводити/виводити порожні рядки не треба; додаткові вертикальні відступи у прикладах зроблені умовно, щоб краще було видно, хто коли ходить.

Розбір задачі. Оскільки гра та сама, що в попередній задачі, варто спиратися на ті самі виграшні та програшні позиції, що в попередній задачі; організація циклу «грати, доки хтось не виграє», елементарна й не потребує розгляду. Тож лишаються тільки дрібні зауваження.

Чи краще визначити, які позиції виграшні й які програшні, один раз на початку, чи переобчислювати на кожному ході? Якщо раптом виграшність/програшність перевіряє згадана в самому кінці розбору попередньої задачі рекурсія з запам'ятовуваннями, то варто щоразу запускати рекурсію (раптом суперник вибере хід, який не був прорахований при аналізі початкової ситуації), але не чистячи шар (те, що вже зберігається в ньому після аналізу попередніх ходів, лишається правильним і може допомогти скоротити повторні обчислення). У решті випадків, зазвичай краще обчислити один раз і надалі лише брати готові відповіді (це не відповідає тому, як грала б людина... але далеко не завжди варто моделювати поведінку людини; людині важко й рахувати точну кількість паличок, коли їх багато). Втім, вибір між єдиною і багатократним визначенням виграшності/програшності важливий, *лише* якщо користуватися не дуже ефективним алгоритмом. Якщо алгоритм досить ефективний (наприклад, заповнення масиву `minWinS` кодом зі стор. 96), навіть багатократним його використанням можна знехтувати у порівнянні з часом на комунікацію з програмою-суперницею.

У розборі схожої гри на стор. 85 наголошувалося, що для проходження всіх тестів треба грати проти різних безграмотних програм-суперниць, і, щоб одна й та сама наша програма вміла перехоплювати ініціативу в будь-якої з них, при ході з програшної позиції варто ходити випадковим чином. В цій задачі значно складніша залежність, які позиції виграшні й які програшні, й автору задачі не вдалося придумати, якою мала би бути програма-суперниця, щоб використання нашою програмою випадкових чисел значно полегшувало перехоплення ініціативи у неї. Так що в цій задачі, начебто, не важливо, чи користуватися випадковими числами при ході з програшної позиції. Незважаючи на те, що це було справді важливо, щоб пройти абсолютно всі тести тієї задачі.

Насамкінець, якщо не вміти правильно визначати, які позиції виграшні й які програшні, справді великих балів не набрати ні за цю задачу, ні за попередню; але, враховуючи особливості їх оцінювання, набирати *частину* балів на цій задачі значно легше, ніж на попередній. По-перше, нарахування балів за окремі тести (а не блоки) збільшує ймовірність отримати бали за те, що в деяких окремих тестах Ваша програма виграє випадково. (Враховуючи, що не в усіх тестах програма-суперниця грає ідеально, це цілком можливо.) По-друге, за рахунок того, що 20% балів нараховується навіть за те, що програма правильно згідно правил програє там, де могла б виграти: написати програму, яка лише дотримується правил гри, не переймаючись своїми шансами виграти, значно легше, ніж розв'язувати задачу по суті. А є ж ще тести, в яких і слід програти — саме за них така проста програма має взяти повні бали.

3.17 Обласна інтернет-олімпіада 2018/19 н. р.

Задачі доступні для дорішування (ejudge.ckipo.edu.ua, змагання №27).

Задача А. «Змії»

Із тераріуму втекли x гадюк, y кобр та z гюрз. Довжина кожної гадюки — 1 м, кожної кобри — 1 м 30 см, а гюрзи — 1 м 15 см.

Напишіть *вираз*, який знаходитиме, скільки повних метрів змій втекло з тераріуму. У цій задачі треба здати не програму, а вираз: вписати його (сам вираз, не назву файлу) у відповідне поле перевіряючої системи і відправити на перевірку. Правила запису виразу: можна використовувати цілі десяткові чісла, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), “/” (ділення дробове, наприклад, $18/5=3,6$), “//” (ділення цілочисельне, наприклад, $18//5=3$), круглі дужки “(” та “)” для групування та зміни порядку дій. Дозволяються пропуски (пробіли), але не всередині чисел.

Змінні, від яких залежить вираз, обов'язково повинні мати саме той смисл, який описаний раніше, і називатися вони повинні саме x , y , z (маленькими латинськими літерами). Наприклад: можна здати вираз $x+y+z$ і отримати 50 балів з 200, бо він неправильний, але все ж іноді відповідь випадково збігається з правильною. А «цілком аналогічний» вираз $ga+ko+gu$ буде оцінений на 0 балів, бо змінні повинні називатися так, як вказано.

Розбір задачі. Найлегше порахувати загальну сумарну довжину в сантиметрах, а потім перевести у повні метри цілочисельним діленням. Наприклад, так: $((100 \cdot x + 130 \cdot y + 115 \cdot z) // 100)$ (без лапок). Само собою, зараховується не лише цей вираз, а будь-який правильний.

Задача В. «Ракета»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

В далекому космосі, де можна нехтувати гравітаційними впливами та тертям, запускають ракету. Розмірами ракети теж можна знехтувати, вважаючи її матеріальною точкою. До запуску, ракета нерухома; ракета весь час летить прямолінійно, перші d секунд рівноприскорено з прискоренням $a \text{ М/с}^2$, далі рівномірно з тією швидкістю, яку встигла набрати за ці d с.

Слід спиратися на такі відомі з фізики факти: поки ракета летить рівноприскорено (час t , що вимірюється від її старту, в межах $0 \leq t \leq d$), її відстань від старту може бути виражена як $\frac{at^2}{2}$, а швидкість — як at ; коли закінчується рівноприскорений та починається рівномірний рух ракети, її швидкість незмінна й дорівнює ad , тож за кожен секунду до пройденої відстані додається ще ad . Застосовувати інші фізичні моделі (як-то теорію відносності) не треба й не можна.

Напишіть програму, яка рахуватиме, скільки часу знадобилося цій ракеті (яку вважаємо матеріальною точкою), щоб пройти фрагмент свого маршрута, починаючи з відстані b м від старту до відстані c м від старту.

Вхідні дані. Чотири цілі числа $a \ b \ c \ d$ в один рядок через одинарні пропуски (пробіли); a вимірюється у М/с^2 і перебуває в межах $1 \leq a \leq 100$; b, c вимірюються у метрах і перебувають у межах $0 \leq b < c \leq 10^9$; d вимірюється у секундах і перебуває в межах $1 \leq d \leq 1000$.

Результати. Виведіть єдине число — проміжок часу між моментами, коли ракета пролетіла через вказані точки.

Оцінювання. Кожен тест оцінюється окремо. Якщо відповідь не є числом (зокрема, містить у собі ще щось, крім чисел), цей тест оцінюється на 0. Не виводьте ніяку «допоміжну» інформацію, як-то "Raketa projde promiuk za" чи якоесь аналогічно. Така заборона стосується не лише цієї задачі, а й усіх задач змагання, і взагалі переважної більшості задач, що перевіряються автоматично. Крім того, заборонено (конкретно у цій задачі), щоб увесь виведений результат містив більше 30 символів.

Запис числа може бути хоч суто десятковим, хоч експоненційним; якщо відповідь виявляється цілою, її можна хоч виводити як цілу, хоч дописувати 0.000000 (з довільною, але з урахуванням вищезгаданого обмеження, кількістю нулів). Якщо відповідь на деякий тест правильна з відносною похибкою до 10^{-15} , за тест ставиться повний бал (10 з 10). Інакше, якщо відповідь правильна з відносною похибкою до 10^{-9} , за тест ставиться 60 % балів (6 з 10). Інакше, якщо відповідь правильна з відносною похибкою до 10^{-3} , за тест ставиться 30 % балів (3 з 10). Якщо похибка ще більша, тест оцінюється на 0.

Приклади:	Вхідні дані	Результати
	2 4 9 12	1
	100 999999999 1000000000 1000	1.0e-005
	9 2000 3000 20	5.555555555555556
	1 55 777 12	60.26191151829848

Примітки. (1) У першому тесті, через 2 с після старту ракета перебуває якраз за $\frac{2 \cdot 2^2}{2} = 4$ (м) від старту, а через 3 с — якраз за $\frac{2 \cdot 3^2}{2} = 9$ (м). Тому проміжок між цими моментами становить 1 с. У другому тесті маємо проміжок, де ракета рухається рівномірно зі швидкістю $100 \text{ М/с}^2 \times 1000 \text{ с} = 10^5 \text{ М/с}$, тож відстань $1000000000 - 999999999 = 1$ (м) долатиметься за 10^{-5} (с).

(2) Смысл понятия «відносна похибка» можна уточнити у наступній задачі. І взагалі, перевірка цієї задачі відбувається в точності за правилами, описаними у наступній задачі, за єдиним виключенням: там гарантовано, що відповідь учасника являє собою один рядок, а тут, якщо Ваша програма виводитиме не один рядок, за тест ставитиметься 0 балів.

Розбір задачі. Само собою, потрібно з'ясувати, яка з трьох ситуацій має місце: (А) потрібний проміжок повністю потрапляє на рівноприскорений рух ракети; (Б) потрібний проміжок повністю потрапляє на рівномірний рух ракети; (В) потрібний проміжок містить і кінець рівноприскореного руху, і початок рівномірного.

Найпростішим є випадок (Б); він має місце при $\frac{ad^2}{2} \leq b$ (початок потрібного проміжку далі від старту, ніж точка, до якої триває прискорений рух), а остаточна відповідь виражається простою формулою $\frac{c-b}{ad}$ (бо для рівномірного руху час = $\frac{\text{відстань}}{\text{швидкість}}$, а за d секунд рівноприскореного руху набувається швидкість ad).

Випадок (А) має місце при $c \leq \frac{ad^2}{2}$; для нього слід перетворити згадану в умові формулу $s = \frac{at^2}{2}$ до $t = \sqrt{\frac{2s}{a}}$; підставивши двічі для початку й кінця фрагменту, маємо $\sqrt{\frac{2c}{a}} - \sqrt{\frac{2b}{a}}$.

Лишається тільки випадок (В); розпізнавати його у програмі, мабуть, краще не шляхом виписування його умови, а за рахунок організації розгалужень з `else`-ами так, щоб легко виражати «все, що не (А) і не (Б)»; для нього слід поєднати формулу пункту (А) для проміжку від b до $\frac{ad^2}{2}$ та формулу пункту (Б) для проміжку від $\frac{ad^2}{2}$ до c .

Тільки рівно такий розв'язок, найімовірніше, набере значну частину балів, та не всі. Щось із 210 з 250 (точна кількість може залежати від використаної мови програмування, а в деяких з мов — від використаних типів). І причина тому — похибки (на що досить явно натякає й те, що при здачі такого розв'язку він отримує за деякі з тестів і не 0, і не максимум). При обчисленні $\sqrt{\frac{2c}{a}} - \sqrt{\frac{2b}{a}}$ доводиться віднімати числа, які можуть бути досить близькими. Коли самі b та c досить великі, але при цьому $c - b$ становить всього кілька метрів, то різниця між $\sqrt{\frac{2c}{a}}$ та $\sqrt{\frac{2b}{a}}$ стає взагалі малопомітною. Це стандартна проблема (див. також стор. 14), і один з можливих шляхів її зменшення (повністю ліквідувати її неможливо, бо так уже влаштовані комп'ютери, що нема стандартних способів подавати ірраціональні числа точно) — спробувати аналітично перетворити формулу так, щоб смисл був той самий, але вплив похибок менший. І тут це досить просто: $\sqrt{\frac{2c}{a}} - \sqrt{\frac{2b}{a}} = \sqrt{\frac{2}{a}} \cdot (\sqrt{c} - \sqrt{b}) = \sqrt{\frac{2}{a}} \cdot \frac{c-b}{\sqrt{c} + \sqrt{b}}$ (останнє перетворення спирається на тотожність $(x-y) \cdot (x+y) = x^2 - y^2$, де $x = \sqrt{c}$, $y = \sqrt{b}$). Само собою, для отримання повних балів слід зробити таке перетворення і для ситуації (Б), і для ситуації (В).

Задача С. «Checker для “Ракети”»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

У цій задачі Вам пропонується зробити те, що зазвичай роблять автори задач, а не учасники. В минулій задачі були обіцяні різні бали залежно від того, з якою точністю знайдено відповідь. У цій задачі, Вашій програмі будуть надані правильна відповідь попередньої задачі та відповідь учасника, а Ваша програма повинна буде видати, як слід це оцінити.

Вхідні дані. Перший рядок вхідних даних являє собою правильну відповідь на задачу «Ракета», з рівно 25 десятковими цифрами після крапки; гарантовано, що це буде значення відповіді, правильне (в рамках наведених знаків, отже, з абсолютною похибкою до 10^{-25}) для якихось вхідних даних попередньої задачі (яких саме вхідних даних, невідомо, але в рамках вказаних там обмежень). Надалі до значення з 1-го рядка слід ставитися як до точної відповіді, ігноруючи, що воно саме може бути заокругленням.

Другий рядок вхідних даних являє собою відповідь учасника. Він може містити абсолютно що завгодно, але гарантовано являє собою лише один рядок. Вхідні дані в цілому гарантовано містять рівно два рядки, не більше й не менше. Кожен з рядків завершується символом переведення рядка (в Unix-форматі).

Результати. Якщо довжина 2-го рядка перевищує 30 символів (31 і більше), виведіть “РЕ”. Якщо 2-й рядок не є записом числа (зокрема, якщо містить і запис числа, і ще щось), теж виведіть “РЕ”. Але якщо зайвими є лише пробіли (ASCII-код 32) перед числом та/або після числа, вони

повинні не впливати на результат — зокрема, тому, що деякі версії Паскаля ставлять їх самі, як-то “ 1.0000000000000000E-005” (без лапок, але з пробілом).

Якщо 2-й рядок є записом числа, відповідь Вашої програми повинна залежати від похибки. Позначимо значення числа з 1-го рядка (правильної відповіді) як a , числа з 2-го рядка (відповіді учасника) як b . При $\frac{|b-a|}{a} < 10^{-15}$, слід вивести “OK 10”. При $10^{-15} < \frac{|b-a|}{a} < 10^{-9}$, слід вивести “PT 6”. При $10^{-9} < \frac{|b-a|}{a} < 10^{-3}$, слід вивести “PT 3”. При $\frac{|b-a|}{a} > 10^{-3}$, слід вивести “WA”. (Тести не містять перевірок, що слід виводити, коли відносна похибка в точності рівна 10^{-15} (чи 10^{-9} , чи 10^{-3}). Головним чином тому, що при обчисленні значення похибки теж можливі похибки, й точної рівності все одно не виходить.)

Будь-яку з відповідей слід виводити в першому і єдиному рядку. Якщо відповідь містить і двобуквенний код, і число, вони повинні бути розділені одинарним пробілом. Самі двобуквенні коди повинні бути виведені великими латинськими буквами. “PE” є скороченням від “Presentation Error”, “PT” — від “Partial solution”, “WA” — від “Wrong Answer”.

Приклади:

Вхідні дані	Результати
1.00000000000000000000000000000000 +1	OK 10
1.00000000000000000000000000000000 1.00000000000000000000000000000000	OK 10
1.00000000000000000000000000000000 1.00000000000000000000000000000000	PE
1.00000000000000000000000000000000 1.00000000002718281828	PT 6
0.00010000000000000000000000000000 0.0001000002718281828	PT 3
0.0000120762762995368423496 +1.207627629953684235E-0005	OK 10
0.00010000000000000000000000000000 0.1000000000000000000082e-3	OK 10
0.00010000000000000000000000000000 Маша їла кашу.	PE
0.00010000000000000000000000000000 +++0.0001	PE
0.00010000000000000000000000000000 Відповідь дорівнює: 0.0001	PE

Оцінювання. Перші 10 тестів є тестами з умови й не оцінюються безпосередньо (але, для отримання повного балу за задачу, програма повинна пройти їх усі). 40 % (100) балів припадають на потестове оцінювання (кожен з 25 тестів приносить або 4 бали з 4-х, або 0, інші тести на це не впливають). Решта 60 % (150) балів припадають на поблокове оцінювання: 10 блоків, кожен блок дає або 15 балів з 15-ти (якщо пройдено всі його тести), або 0. У цій задачі, детальніша інформація про розподіл тестів між блоками, а також про відмінності та залежності між різними блоками не підлягає розголошенню до кінця туру.

Розбір задачі. Одна з основних складностей цієї задачі — потреба і прочитати вхідні дані, як рядок (щоб переконатися, що довжина ≤ 30 символів), і виділити з нього число, щоб провести розрахунок похибки за наданою формулою (примітивною, там нема чого пояснювати) та написати правильні розгалуження. Тут важко щось сказати загально, бо все це — більше про особливості мов програмування, ніж про задачу. Учасники, які зазвичай пишуть на python або C#, тут опинилися в трохи виграшному становищі, бо їм постійно доводиться робити перетворення з рядкового подання у чисельне, й вони повинні б це добре пам'ятати; але засоби такого перетворення є в усіх дозволених на цій олімпіаді мовах програмування. Пізніше сюди будуть дописані посилання на правильні розв'язки цієї задачі різними мовами програмування.

Задача D. «Preview»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 64 мегабайти

Preview (попередній перегляд) графічного зображення — це спеціально зменшена його версія, призначена, щоб могли показати його загальний вигляд швидко й не займаючи багато пам'яті. Таке зменшення розмірів зазвичай призводить до втрати якості. В цій задачі розглянемо аж надто простий (та досить поганий за якістю) спосіб створення preview-зображення.

Нехай початкове зображення має форму квадрата $2^k \times 2^k$ пікселів, де $2 \leq k \leq 9$. Інакше кажучи, сторони цього квадрата рівні або 4, або 8, або 16, або 32, або 64, або 128, або 256, або 512 піксел(ів). Будемо будувати preview-зображення розмірами $2^{k-1} \times 2^{k-1}$, $2^{k-2} \times 2^{k-2}$, ..., $2^1 \times 2^1$ (інакше кажучи, зі сторонами квадрата удвічі меншими, вчетверо меншими, і т. д., до розміру 2×2 включно). Таким чином, кожен піксель утвореного preview-зображення відповідає квадрату 2×2 , чи 4×4 , ..., чи $2^{k-1} \times 2^{k-1}$ початкового зображення. От і робитимемо колір відповідного пікселя preview-зображення рівним середньому арифметичному кольорів пікселів відповідного квадрата.

Як розуміти «середнє арифметичне кольорів»? Слід розглянути дві версії:

1. Зображення подане в сірих кольорах різної інтенсивності, й колір можна задати цілим числом від 0 до 9. Тоді середнє арифметичне кольорів зводиться просто до середнього арифметичного відповідних чисел.
2. Зображення подане у форматі RGB, тобто для кожного пікселя окремо вказано яскравість R-каналу (червоного), окремо G-каналу (зеленого), окремо B-каналу (синього). Кожна яскравість подається рівно двома шістнадцятковими цифрами, тобто в межах від 00 до FF (де $FF_{Hex} = 255_{Dec}$; обидві межі включно). Треба порахувати середнє за кожним каналом окремо, й зібрати три результати в колір пікселю. Значення трьох яскравостей кожного пікселю записані у вхідних даних та повинні бути записані у знайденій відповіді підряд (без пропусків) і з символом “#” (решітка, ASCII-код 35) спереду.

В обох випадках, вимагається рахувати середнє арифметичне з округленням до найближчого цілого (якщо відстані до цілих однакові, то до більшого з них). Наприклад: $ave(1, 1, 1, 2) = 1$; $ave(2, 2, 1, 2) = 2$; $ave(1, 2, 1, 2) = 2$.

Вхідні дані. Перший рядок містить одне з двох слів “gray” чи “rgb” (без лапок, маленькими латинськими буквами) на позначення того, чи в цьому тесті кольори сірі (як у пункті 1), чи RGB (як у пункті 2). Другий рядок містить розмір N (сторону квадрата), яка гарантовано є однією з вищеперелічених. Далі слідує N рядків, кожен з яких містить кольори N пікселів (якщо перший рядок тесту був “gray”, то одноцифрових чисел від 0 до 9, а якщо “rgb”, то описаних у пункті 2 послідовностей з “#” та шести 16-вих цифр; для цифр зі значеннями від 10 до 15 використані *маленькі* букви a–f). У кожному з випадків, кольори різних пікселів розділяються одинарними пробілами (ASCII-код 32).

Приклади:

Вхідні дані	Результати
gray	2 3 2 4
8	3 2 7 7
1 2 3 2 1 2 5 1	5 4 5 6
2 1 2 3 0 3 9 1	4 4 6 5
3 2 1 2 2 9 7 5	
2 3 2 1 8 7 7 8	2 5
1 5 3 0 3 5 9 5	4 6
9 6 9 5 4 6 6 5	
2 4 4 3 8 4 5 4	
1 8 4 5 9 4 8 4	
rgb	#9dcfa6 #59a279
4	#907148 #7fc37d
#a4e794 #b0c1fa #c4d243 #3acf90	
#b09726 #6efde5 #28c477 #3d2299	
#55b48f #b76573 #6883a1 #c2bf34	
#d3951d #601601 #2dcbaa #a3fe75	

Результати. Отримані preview-зображення, аналогічно формату вхідних даних, але без перших двох рядків (зразу кольори пікселів). Спочатку $N/2$ рядків, що містять preview розміру $(N/2) \times (N/2)$; якщо $N/4 \geq 2$, то один порожній рядок і $N/4$ рядків, що містять preview розміру $(N/4) \times (N/4)$; і так далі (до preview розміру 2×2 включно, розділяючи різні preview одинарними порожніми рядками).

Примітки. Зверніть увагу на перше число передостаннього рядка відповіді першого тесту та на передостанній символ відповіді другого тесту.

Оцінювання. У цій задачі оцінювання поблокове, тобто тести розбиті на блоки, і бали за блок нараховуються лише в разі успішного проходження *всіх* тестів блоку. Блоки 1–2 складаються кожен з єдиного тесту з умови, й результати їх перевірки показуються, але ніяк не впливають на бали. Далі є ще 10 блоків:

Обласна інтернет-олімпіада 2018/19 навч. року
Черкаська обл., 17.10.2019

	Перший рядок (gray/rgb)		
	лише gray	лише rgb	будь-який з двох
$N = 4$	блок 3 тести 3–6 5% балів перевіряється завжди	блок 4 тести 7–10 5% балів перевіряється завжди	блок 5 тести 11–14 5% балів перевіряється лише в разі зарахування блоків 3–4
$N = 8$	блок 6 тести 15–18 5% балів перевіряється завжди	блок 7 тести 19–22 5% балів перевіряється завжди	блок 8 тести 23–26 5% балів перевіряється лише в разі зарахування блоків 6–7
N або 4, або 8, або 16	блок 9 тести 27–30 10% балів перевіряється лише в разі зарахування блоків 3, 6	блок 10 тести 31–34 10% балів перевіряється лише в разі зарахування блоків 4, 7	такого блоку нема
N будь-яке з вищезгаданих	блок 11 тести 35–45 20% балів перевіряється лише в разі зарахування блоків 3, 6, 9	такого блоку нема	блок 12 тести 46–60 30% балів перевіряється лише в разі зарахування блоків 3–11

Розбір задачі. На основних можливих проблемах, які могли б виникнути, звернуто увагу (звісно, без пояснень) у примітці наприкінці умови.

Зокрема, не можна будувати кожен наступний квадрат-відповідь на основі попереднього квадрату-відповіді, бо просто заокруглене середнє і заокруглене середнє заокруглених середніх не завжди дорівнюють одне одному. Зокрема, це проявляється на згаданому «першому числі передостаннього рядка відповіді першого тесту»: $(1 + 2 + 3 + 2 + 2 + 1 + 2 + 3 + 3 + 2 + 1 + 2 + 2 + 3 + 2 + 1) / 16 = 32 / 16 = 2$, тоді як при обчисленнях окремих квадратів 2×2 виходить $(1 + 2 + 2 + 1) / 4 = \frac{6}{4} \approx 2$, $(3 + 2 + 2 + 3) / 4 = \frac{10}{4} \approx 3$, $(3 + 2 + 2 + 3) / 4 = \frac{10}{4} \approx 3$, $(1 + 2 + 2 + 1) / 4 = \frac{6}{4} \approx 2$, після чого спроба побудувати результат квадрату 4×4 на основі квадратів 2×2 дає $(2 + 3 + 3 + 2) / 4 = \frac{10}{4} \approx 3$.

Є щонайменше два способи вирішення цієї проблеми.

(А) Діяти згідно зі словесним формулюванням з умови, тобто справді рахувати суми квадратів розмірами 2×2 й ділити на 4, потім суми квадратів розмірами 4×4 й ділити на 16, і так далі.

(Б) Можна (і, мабуть, простіше) формувати з квадрата розмірами $N \times N$ квадрат розмірами $\frac{N}{2} \times \frac{N}{2}$, де кожен елемент меншого квадрату є сумою відповідних чотирьох елементів початкового квадрату, а ділення й округлення робити безпосередньо перед виведенням; далі рахувати $\frac{N}{4} \times \frac{N}{4}$ сум відповідних чотирьох елементів квадрату розмірами $\frac{N}{2} \times \frac{N}{2}$, і так далі. На відміну від знаходження заокруглених середніх, для додавання виконуються комутативність та асоціативність (вони ж переставний та сполучний закони), тож суми сум гарантовано є правильними сумами квадратів 4×4 , наступні суми — правильними сумами квадратів 8×8 , тощо.

Перевага цього підходу (Б) над попереднім підходом (А) — не треба циклів проходження квадрата $2^j \times 2^j$ при різних j , досить значно простішого коду `arrNew[i][j] = arrOld[2*i][2*j] + arrOld[2*i][2*j+1] + arrOld[2*i+1][2*j] + arrOld[2*i+1][2*j+1]` (при нумерації масивів з 0; при нумерації з 1 такий підрахунок набуває вигляду `arrNew[i][j] := arrOld[2*i-1][2*j-1] + arrOld[2*i-1][2*j] + arrOld[2*i][2*j-1] + arrOld[2*i][2*j]`), узятого у три цикли: два внутрішні перебирають індекси чергового, удвічі зменшеного відносно попереднього, квадрата, зовнішній перебирає розмір квадрата.

Тобто, досі описано вже два способи отримати ті 40 % балів (що не так мало), які призначені за блоки 3, 6, 9, 11, в яких зображення сіре (gray). Що стосується варіанту “rgb”, то ускладнення більш технічні, ніж ідейні (олімпіада, кінець кінцем, «з інформатики (програмування)», а не математики). Просто треба грамотно читати вхідні дані (чи то «розбираючи» рядки (string) на окремі двозначні шістнадцяткові числа, чи то «збираючи» ці двозначні шістнадцяткові числа з окремих символів (char)), виконати такі самі, як у сірому варіанті, дії для кожного з трьох каналів R, G та B, а наприкінці зробити зворотне перетворення (чи то сформувавши для кожного пікселя семисимвольний рядок, сформувавши його зі знайдених R-, G- та B-значень, чи то розділити знайдені R-, G- та

В-значення на символи, що є окремими шістнадцятковими цифрами, й вивести для кожного пікселя потрібну послідовність цих окремих символів).

Більшість мов програмування мають вбудовані засоби перетворення між рядковим та чисельним поданнями. Тут ситуація дещо ускладнюється тим, що потрібно вводити/виводити *шістнадцяткові* числа, причому завжди двозначні (для менших 16 — з додатковим нулем спереду) та обов'язково з малими буквами для цифр a–f. Більшість сучасних мов програмування дозволяє все це налаштувати у викликах стандартних функцій. Щоправда, далеко не всі пам'ятають такі деталі, тож важливим стає питання, як і де їх швидко подивитися; враховуючи, що на олімпіадах взагалі-то має бути обмежений доступ до Інтернету, корисно навчитися користуватися, крім пошуку в Інтернеті, також підказками середовищ програмування та вбудованою документацією (help-om).

Можна також самостійно написати функції, які виконуватимуть потрібний вузький частковий випадок перетворень (в обох напрямках) між внутрішнім чисельним поданням і рядковим шістнадцятковим записом. У професійному програмуванні такі реалізації давно реалізованого, як правило, не вітаються і вважаються «написанням велосипедів». Але навіть у професійному програмуванні бувають виключення з цього правила; якщо ж ситуація посилюється тим, що на олімпіаді сильно обмежені можливості дивитися документацію, то цілком можна й «написати велосипед».

Насамкінець, що робити з тим, що на вхід одній і тій самій програмі може бути подано як “gray”, так і “rgb”? Мабуть, прочитавши перший рядок, тут же робити розгалуження вигляду `if mode='gray' then DoAllGray else DoAllRGB`, де “DoAllGray” та “DoAllRGB” — самостійно написані підпрограми, які розв'язують задачу (включно з читанням решти, крім першого рядка, вхідних даних, та виведенням результатів) одна для сірого випадку, інша для RGB. Пізніше сюди будуть дописані також приклади розв'язків, у яких такого дублювання коду практично нема (суто алгоритмічна частина написана все-таки один раз, а відмінності в діях для “gray” та “rgb” зроблені за допомогою перевантаження операцій та/або поліморфізму); але в цій задачі це не дуже доцільно, а засоби доводиться використовувати досить складні.

3.18 II (районний/міський) етап 2019/20 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №70).

Задача А. «ККД»

Різні електролампи можуть мати різні коефіцієнти корисної дії (ККД). За означенням, ККД електролампи є відношенням потужності, яку вона випромінює у вигляді світла, до потужності, яку вона споживає з електромережі. Діапазон реальних ККД становить десь від 3 % у поганих ламп розжарювання до 50 % у якісних світлодіодних. У цій задачі вважаємо, що все, що споживається з електромережі й не випромінюється як світло, переходить лише у тепло.

Нехай є дві різні лампи, які мають однакову світлову потужність p Вт, але різні ККД k_1 % та k_2 %. Наприклад, нехай $p = 4$ (Вт), $k_1 = 4$ (%) та $k_2 = 50$ (%). Це можливо, якщо перша лампа споживає з електромережі 100 Вт (перетворюючи 96 Вт у тепло), а друга 8 Вт і 4 Вт відповідно.

Напишіть вираз, що знаходитиме, у скільки разів більше теплота виділяє перша лампа у порівнянні з другою. (Наприклад, для щойно згаданих $p=4$, $k_1=4$, $k_2=50$ правильна відповідь $= 24$, як $\frac{96 \text{ Вт}}{4 \text{ Вт}} = 24$; а для $p=17$, $k_1=42$, $k_2=47$, правильна відповідь ≈ 1.22461815 .)

У цій задачі треба здати не програму, а вираз: вписати його (сам вираз, не створюючи файл) у відповідну сторінку ejudge і відправити. У виразі можна використовувати цілі десяткові числа, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), “/” (ділення дробове), круглі дужки “(” та “)” для групування та зміни порядку дій. Дозволяються пропуски (пробіли), але не всередині чисел та не всередині назв змінних. Множення треба писати явно зірочкою (наприклад, не можна писати добуток $k_1 k_2$ як “ $k_1 k_2$ ”, лише як “ $k_1 * k_2$ ”).

Змінні, від яких залежить вираз, повинні називатися саме k_1 , k_2 , p (“ k ” та “ p ” — маленькі латинські букви, “1” та “2” — цифри). Вираз перевірятимуть при цілих значеннях усіх цих змінних, у межах $3 \leq k_1 < k_2 \leq 50$ (відсотків), $1 \leq p \leq 20$ (Вт).

Розбір задачі. Там взагалі потрібне p ? Насправді, ні. Тепер почнемо сам розбір.

Позначимо ті потужності, котрі споживаються лампами з електромережі, *математичними* змінними p_1 та p_2 відповідно. (Тобто, ці змінні з’являться у проміжних викладках на папері, але їх не треба й не можна включати в остаточну відповідь.) Тоді однакові, рівні p , корисні світлові потужності можна виразити як $\frac{k_1\%}{100\%} \cdot p_1$ та $\frac{k_2\%}{100\%} \cdot p_2$ відповідно. Звідси маємо $\frac{k_1\%}{100\%} \cdot p_1 = \frac{k_2\%}{100\%} \cdot p_2$.

Потрібно ж знайти відношення не корисних теплових потужностей, які, згідно примітки «все, що споживається з електромережі й не випромінюється як світло, переходить лише у тепло» рівні $\frac{100\% - k_1\%}{100\%} \cdot p_1$ та $\frac{100\% - k_2\%}{100\%} \cdot p_2$ відповідно. Отже, шуканою є величина $\frac{\frac{100-k_1}{100} \cdot p_1}{\frac{100-k_2}{100} \cdot p_2} = \frac{100 - k_1}{100 - k_2} \cdot \frac{p_1}{p_2}$. Дріб $\frac{p_1}{p_2}$ можна виразити з формули наприкінці попереднього абзацу: після скорочення «100%», ділення обох частин на p_2 та множення на k_1 виходить $\frac{p_1}{p_2} = \frac{k_2}{k_1}$. Отже, остаточна відповідь може мати, наприклад, вигляд “ $((100-k_1)*k_2)/((100-k_2)*k_1)$ ”, чи, наприклад, вигляд “ $(100-k_1)/(100-k_2)*k_2/k_1$ ”. Звісно, зараховувалися не лише конкретно ці дві відповіді, а будь-яка правильна, включно з тими, де p залишили (могли б скоротити, але не зробили цього).

Задача В. «Спільні дотичні»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам’яті: 256 мегабайтів

Як відомо, дотичною до кола є пряма, що має рівно одну спільну точку з цим колом. Можлива ситуація, коли одна й та сама пряма є дотичною відразу до двох кіл. Тоді вона називається спільною дотичною. Напишіть програму, яка знаходитиме кількість різних спільних дотичних для заданих двох кіл. При виведенні врахуйте стародавню традицію приписувати числу 7 значення «багато». Тобто, виводьте 7 для всіх кількостей спільних дотичних ≥ 7 .

Вхідні дані. Шість цілих чисел X_1 , Y_1 , R_1 , X_2 , Y_2 , R_2 — координати центра і радіуси 1-го і 2-го кола, записані саме в такому порядку, кожне число в окремому рядку. Абсолютні величини (модулі) координат не перевищують мільйон. Для радіусів значення у межах від 1 до мільйона.

Результати. Ваша програма повинна вивести єдине число — кількість спільних дотичних, з урахуванням згаданої стародавньої традиції.

Приклад:

Вхідні дані	Результати	Зображення цього тесту
20 0 4 50 0 10	4	

Примітка. Зображення наведене суто для кращого розуміння прикладу, Ваша програма малювати його не повинна.

Оцінювання. У цій задачі оцінювання потестове (кожен тест перевіряється й оцінюється незалежно від решти, отримані бали додаються).

Розбір задачі. Легко придумати ситуації, коли два кола мають 2 чи 4 спільні дотичні; ситуації, коли їх 0, 1 або 3, придумуються дещо важче; може здатися, ніби строго більше, ніж 4, взагалі не буває. Але насправді такий випадок (один) все-таки є: $(X_1 = X_2)$ and $(Y_1 = Y_2)$ and $(R_1 = R_2)$, тобто кола повністю однакові. Тоді абсолютно будь-яка дотична є спільною, і кількість виявляється нескінченною; це й треба позначати числом 7.

Тому для розв’язування задачі досить вибрати, який із розглянутих випадків має місце, а для цього потрібні *лише* радіуси R_1 , R_2 та відстань між центрами $d = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$. Для зручності подальших викладок забезпечимо $R_1 \geq R_2$ (якщо це не так, досить обміняти їх місцями).

∞ спільних дотичних:	0 спільних дотичних:	1 спільна дотична:	2 спільні дотичні:	3 спільні дотичні:	4 спільні дотичні:
					
кóла повністю однакові	одне коло повністю всередині іншого	кóла дотикаються внутрішньо	кóла перетинаються двома точками	кóла дотикаються зовнішньо	кóла не перетинаються
$(X1 = X2) \text{ and } (Y1 = Y2) \text{ and } (R1 = R2)$	$d < R1 - R2$	$d = R1 - R2$	$R1 - R2 < d < R1$	$d = R1$	$d > R1$

З суто технічних (кодерських) питань слід зауважити: коли координати можуть сягати за модулем мільйон, то квадрат різниці координат вже не поміщається у 32-бітовий тип, надійніше використати 64-бітовий чи «дійсний» (з рухомою комою, він же з плаваючою точкою). А для когось може виявитися несподіванкою, що у FreePascal (якщо говорити конкретно про варіанти, які пропонує ejudge.skipo.edu.ua, то лише fpc, не fpc-delphi та не pasabc-linux) тип integer взагалі досі 16-бітовий. Див. також стор. 13 та стор. 11.

Задача С. «Прямокутні суми»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 256 мегабайтів

Дано прямокутну таблицю розмірами N рядків на M стовпчиків, елементи якої є натуральними числами. Потрібно багатократно знаходити суми всіх чисел, що потрапляють у деякі прямокутники цієї таблиці, утворені рядками з i_1 -го по i_2 -й та стовпчиками з j_1 -го по j_2 -й (нумерація з 1, межі включно, $1 \leq i_1 \leq i_2 \leq N$, нумерація рядків згори донизу, $1 \leq j_1 \leq j_2 \leq M$, нумерація стовпчиків зліва направо).

Вхідні дані. У 1-му рядку через одинарний пробіл записано числа N та M , далі N рядків по M чисел у кожному містять значення комірок таблиці. Далі в окремому рядку записано число T — кількість подальших запитів, потім ще T рядків — самі запити, кожен у вигляді $i_1 \ i_2 \ j_1 \ j_2$ (через одинарні пробіли, $1 \leq i_1 \leq i_2 \leq N$, $1 \leq j_1 \leq j_2 \leq M$).

Результати. Рівно T рядків, кожен з яких містить єдине число — суму, що є відповіддю чергового запиту.

Приклад:

Вхідні дані	Результати	Коментар
4 5	90	«1 4 1 5» задає
1 2 3 4 5	16	всю таблицю;
2 3 4 5 6	12	«1 2 3 4» — правий- верхній з виділених прямокутників;
3 4 5 6 7		«2 4 2 2» — лівий.
4 5 6 7 8		
3		
1 4 1 5		
1 2 3 4		
2 4 2 2		

Оцінювання. Оцінювання блокове (бали за блок нараховуються лише в разі успішного проходження всіх тестів блоку); тест з умови є 1-м і входить у блок 1. Стовпчик «передумови» означає, що розв'язок запускатиметься і перевірятиметься на поточному блоці, лише якщо цей самий розв'язок успішно пройшов блоки, згадані в передумовах. Нерівності через кому (як-то « $100 \leq N, M \leq 400$ »)

означають, що всі перелічені через кому змінні перебувають в указаному діапазоні. Обмеження на " a_{ij} " вказують можливі значення елементів таблиці.

блок	тести	обмеження	перед-умови	бали
1	1–5	$2 \leq N, M, T, a_{ij} \leq 9$	нема	15%
2	6–10	$1 \leq N, M, T \leq 100; 1 \leq a_{ij} \leq 10^4$	блок 1	15%
3	11–15	$1 \leq N \leq 25; 100 \leq M \leq 2019; 10^3 \leq T, a_{ij} \leq 10^5$	блок 1	15%
4	16–20	$100 \leq N \leq 2019; 1 \leq M \leq 25; 10^3 \leq T, a_{ij} \leq 10^5$	блок 1	15%
5	21–27	$100 \leq N, M \leq 400; 123 \leq T, a_{ij} \leq 12345$	бл.1–2	15%
6	28–35	$400 \leq N, M \leq 700; 12345 \leq T \leq 222555;$ $1 \leq a_{ij} \leq 10^9$	блоки 1–2, 5	25%

Розбір задачі. Чому в одному турі дві майже однакові задачі С, D? Деякі способи їх розв'язувань, справді, майже однакові. Але «лобовий» спосіб набирає далеко не всі бали, а серед ефективніших способів далеко не всі однаково придатні до обох задач.

Це що, аж так важливо, чи знає учасник хитрі способи читання вхідних даних? І так, і ні. Що ж зробиш, як олімпіади з інформатики (програмування) вже більше 25 років передбачають розрізнення більш ефективних розв'язків від менш ефективних, а в цих задачах швидкість введення/виведення істотно впливає на швидкість програми в цілому?.. Це не створено автором задачі навмисно, це сумний об'єктивний факт, який час від часу проявляється на олімпіадах (див. також стор. 15, 33–35). І це точно *не* є істотною складовою оцінювання: якби хотіли це перевіряти, то інформацію про різні способи читання не доносили б різними способами безпосередньо на турі, а всіляко приховували. І якраз приховування було б несправедливим щодо тих учасників, які взагалі-то придумали й реалізували ефективний алгоритм, але програма не набирає гідний бал виключно тому, що повільно читає вхідні дані. «Просто зменшити вхідні дані та/або збільшити обмеження часу» — теж так собі пропозиція, бо тоді почали б надто легко проходити не досить ефективні алгоритми.

Як досить легко набрати частину балів? Чому так набирається лише частина балів?

Приклади нехитрих «лобових» розв'язків можна бачити за посиланнями **TODO: add IDEONE links here**. Вони повинні проходити блоки 1–2, але не проходити жодного іншого блоку. Адже складність $O(M \cdot N \cdot T)$ (див., зокрема, стор. 12–13) свідчить про крайню сумнівність вкладання виконання у секунду для одночасно максимальних N, M, T будь-якого з блоків 3–6. («Заперечення» у стилі «ну не будуть же там геть усі прямокутники-запити вигляду " $1\ N\ 1\ M$ ", бо тоді взагалі всі суми-відповіді будуть однакові» нічогоісінько не змінює: наприклад, у тесті з блоку 3 дуже навіть може бути, скажімо, 80 % запитів, де $1 \leq i_1 \leq 5, 20 \leq i_2 \leq 25, 1 \leq j_1 \leq 200, 1800 \leq j_2 \leq 2019$, і 20 % випадкових; це не сильно зменшить час, а однакових відповідей буде мало.)

То що, рахувати суми вкладеними циклами, як праворуч — погано? Якби рахувати треба було один раз (чи двічі, чи навіть 10 разів), це був би найдоцільніший (бо найприродніший) спосіб. Але ж просять поразувати T разів для (мабуть, різних) прямокутників-запитів. Найзагальніша ідея обох задач С, D — придумати, що б такого зробити з двовимірним масивом (одним! незмінним!), щоб, хай навіть витративши (єдинократно!) якісь додаткові зусилля/час/пам'ять (це, до речі, називають *передобробкою*), надалі могли швидше відповідати на самі запити. **Як саме робити передобробку і що саме вона повинна дати** — це вже складніше питання; різним варіантам відповіді на нього присвячена решта розбору цієї задачі та майже весь розбір наступної.

Префіксні суми для одновимірного випадку. Розглянемо аналогічну (але простішу) задачу, де дано одновимірний масив і треба відповідати на багато запитів «яка сума всіх підряд елементів з i -го по k -й?». (до речі, така задача вже давно є на ejudge.skipo.edu.ua, як задача А змагання 7).

Там варто завести масив, кожен елемент якого має смисл «Яку суму мають усі елементи від початку до поточного?». Чи починати, як у наведеному прикладі, нумерацію елементів самого масиву з 1, а масиву сум з 0 — питання дискусійне, такі деталі можна робити по-різному, в кожного способу свої переваги й вади; можна у програмі написати якось інакше, але у поясненнях буде саме так.

Маючи такий масив `sum`, можна рахувати суму будь-якого проміжку `data[j] + data[j+1] + ... + data[k]` за одне віднімання, як `sum[k] - sum[j-1]`, адже з «загальної» суми `data[1] + data[2] + ... + data[k]` якраз приберуть «непотрібну» суму `data[1] + data[2] + ... + data[j-1]`.

Що це дає для початкової задачі? Навіть якщо реалізувати лише цей прийом, тобто порахувати префіксні суми для кожного рядка двовимірного масиву окремо, і відповіді на запити рахувати як «перебрати рядки прямокутника, й додати в циклі суми, знайдені одним відніманням кожна» — навіть тоді виходить $O(N \cdot M + T \cdot N)$ дій, що повинно точно проходити блок 3, а при акуратній реалізації цілком може пройти також блок 5. (Звісно, для цього потрібно правильно рахувати масив префіксних сум, наприклад, якимось у стилі `sum[i][0] := 0; for j := 1 to M do sum[i][j] := sum[i][j-1] + a[i][j]`, ні в якому разі не запускаючи ще один цикл для кожного елементу префіксних сум.)

Блок 4 яскраво демонструє, що префіксні суми за рядками й перебір рядків — лише часткове покращення, і може підбити до того, щоб рахувати в одній програмі зразу два варіанти префіксних сум: за рядками і за стовпчиками. Що ж, якщо правильно вибирати, коли яка сума доцільніша, це може ще щось покращити: дозволити одній програмі пройти і блок 3, і блок 4; покращити «запас» часу при проходженні блоку 5 (але якщо він і так пройшов, то збільшення запасу ніяк не підвищує бали за задачу). Але само по собі це ще не повинно давати можливість пройти блок 6 і тим розв'язати задачу повністю. У поєднанні зі ще якимись оптимізуючими прийомами, це вже можливо; наприклад, можна придумати такий прийом, як «за розмірами прямокутника поточного запиту, вибрати, який з трьох підходів має бути найшвидшим саме для нього: порахувати за рядками, чи порахувати за стовпчиками, чи (якщо він займає майже увесь основний масив) порахувати як різницю суми всього основного масиву, мінус сума «рамки», що не потрапляє, а суму «рамки», що не потрапляє, порахувати через горизонтальні та вертикальні префіксні суми». Або такий прийом, як «порахувати усі ці горизонтальні та вертикальні префіксні суми і для кожного рядка/стовпчика окремо, і для шматків по 10 рядків/стовпчиків підряд; це дасть можливість рахувати за одне віднімання суму не одного рядка/стовпчика, а зразу десяти (щоправда, якщо межі прямокутника-запиту не потрапили на ці красиві межі шматків по 10 рядків/стовпчиків підряд, то по краям (чи з одного краю) треба буде пододавати також суми окремих рядків/стовпчиків)». Або ще якийсь оптимізуючий прийом — їх тут може бути чимало різних. Але спосіб, наведений у наступному пункті, і простіший у плані технічної роботи, і переконливіший у плані теоретичного доведення часу його роботи.

Рекомендований 100 %-й спосіб — двовимірне узагальнення префіксних сум. Замість незручно й неефективно працювати з багатьма окремими одновимірними префіксними сумами, краще здогадатися до такого їх двовимірного узагальнення:

будемо для кожного елементу двовимірного масиву тримати суму всіх елементів, розміщених лівіше&вище за нього (конкретно у тому варіанті, який наведено на рисунку праворуч — включаючи рядок&стовпчик самого поточного елемента; наприклад, `sum[2][3] = 17` виражає суму $(2 + 3 + 4) + (3 + 1 + 4)$; але, як і для одновимірного випадку, можуть бути варіанти, де індекси зсунуті якимось інакше).

		0	1	2	3	4
0		0	0	0	0	0
1		0	2	5	9	14
2		0	5	9	17	23
3		0	10	23	33	45

Коли такий масив уже побудований, то взнавати потрібну суму для прямокутника $i_1 \ i_2 \ j_1 \ j_2$ можна виразом `sum[i2][j2] - sum[i2][j1-1] - sum[i1-1][j2] + sum[i1-1][j1-1]`.

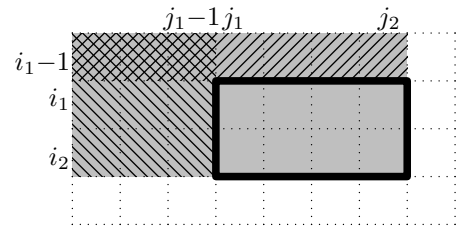
Аргументація цього частково зображена на рисунку: щоб знайти суму області, виділеної рамкою, з суми області, виділеної сірим, віднімемо суму області, виділеної однією штриховкою, та суму області, виділеної іншою штриховкою; область, де штриховки накладаються, один раз додана й двічі віднята, тож, щоб це компенсувати, треба додати її.

Будувати цей масив `sum` можна за правилом `sum[i][j] = sum[i][j-1] + sum[i-1][j] - sum[i-1][j-1] + a[i][j]`, яке обґрунтовується майже так само, тільки сіра незаштрихована область має складатися з єдиного елемента.

Тобто, заповнити масив `sum` можна за $\Theta(1)$ дій на кожен елемент, а відповідати на запити можна за $\Theta(1)$ дій на кожен запит. Читання вхідних даних та виведення результатів мають аналогічні оцінки часу. Таким чином, весь алгоритм має складність $\Theta(N \cdot M + T)$. (До речі, якраз поєднання «час роботи пропорційний розміру вхідних даних» та «вхідні дані величезні» пояснює, чому в цій задачі аж так важливі швидкі введення/виведення: вони не «губляться» за рештою операцій.)

То мені що, заводити масив розмірами 2020×2020 , незважаючи на те, що фактично таких тестів не буде?

Якщо писати мовою програмування, в якій розмір масиву задається зарані (під час написання програми, отже, до прочитання вхідних даних) — так, тоді це єдиний вихід. Але ліміту пам'яті на це вистачає, із запасом. А якщо задавати розміри масивів після прочитання вхідних даних, таке питання, зазвичай, взагалі не виникає: прочитали N , M , задали розміри...; і в цій задачі не виникає рідкісної ситуації, коли заміна динамічної пам'яті на статичну дає істотне прискорення.



А без префіксних сум чи їхніх прямокутних узагальнень набрати більше 30 % балів не можна? Не те, щоб геть не можна; частина способів, описаних у наступній задачі, цілком годяться, щоб пройти у цій задачі, наприклад, блоки 1–5. Є й інші способи, геть невеликі модифікації яких повністю вирішують зразу обидві задачі. Але вони складніші за згадані тут. Тому, автор більш схильється до того, що, мабуть, легше написати окремо спосіб для цієї задачі, якимось оснований на префіксних сумах, і окремо інший спосіб для наступної задачі; але це сильно залежить від того, які саме ідеї вдалося пригадати чи придумати під час туру.

А блоки 1 та 2 взагалі чимсь відрізняються з точки зору розв'язування? З точки зору переважної більшості алгоритмів, нічим: майже будь-яка розумно написана програма, що проходить 1-й блок і не містить явних обмежень проти 2-го блоку, має проходити також і 2-й. Але відмінність може проявитися для помилкових реалізацій, які: неправильно читають багатоцифрові числа; неправильно обробляють ситуацію, коли є лише один стовпчик чи лише один рядок; намагаються використати аж надто короткий тип даних (тут, зокрема, для когось може виявитися несподіванкою, що у FreePascal тип `integer` досі 16-бітовий, тому треба використовувати для самих елементів щонайменше `longint`, для сум `QWord` чи `int64`; див. також стор. 13 та стор. 11); якимось дуууже вже неефективно читають вхідні дані. Що ж, ситуації, коли програмі (нехай навіть мовою Python) не вистачає 2 секунд на тому тесті, на якому найкраща наявна програма мовою C++ вкладається у 0,001 с (різниця у тисячі разів), все-таки є явною ознакою погано написаної програми...

Задача D. «Прямокутні максимуми»

Вхідні дані: Або клавіатура, або `input.txt` Обмеження часу: 1 сек
Результати: Або екран, або `output.txt` Обмеження пам'яті: 256 мегабайтів

Єдина відмінність від попередньої задачі — потрібно знаходити максимальне число прямокутника, а не суму.

Вхідні дані. Формат повністю відповідає попередній задачі.

Результати. Формат відрізняється від попередньої задачі лише тим, що для кожного з T прямокутників треба вивести максимальне число, а не суму.

Оцінювання. Аналогічно попередній задачі, але блоки трохи

інші:

блок	тести	обмеження	передумови	бали
1	1–5	$2 \leq N, M, T, a_{ij} \leq 9$	нема	20%
2	6–10	$1 \leq N, M, T \leq 100; 1 \leq a_{ij} \leq 10^4$	блок 1	20%
3	11–19	$100 \leq N, M \leq 400; 123 \leq T, a_{ij} \leq 43210$	блоки 1–2	40%
4	20–25	$400 \leq N, M \leq 500; 54321 \leq T \leq 222555; 1 \leq a_{ij} \leq 10^9$	блоки 1–3	20%

Приклад:

Вхідні дані	Результати
4 5	8
1 2 3 4 5	5
2 3 4 5 6	5
3 4 5 6 7	
4 5 6 7 8	
3	
1 4 1 5	
1 2 3 4	
2 4 2 2	

Примітка. Для отримання значних балів за ці дві останні задачі потрібно вибрати й реалізувати не просто правильні, а ефективні правильні алгоритми. Крім того, якщо мова програмування має багато різних способів введення&виведення, вибір швидкого способу читання вхідних даних і швидкого способу виведення результатів теж може вплинути на результат. Див. також <https://ejudge.cskipo.edu.ua/io.pdf>

Розбір задачі. Що може бути перенесено в цю задачу з попередньої і що ні? «Лобовий» спосіб може бути перенесений з очевидними змінами, наприклад,

TODO: add IDEONE links here

. І він так само, з тих самих причин, повинен проходити лише блоки 1–2 (які, втім, тепер дають вже не 30 %, а 40 % балів). Прочитати короткі пункти на початку та наприкінці розбору попередньої задачі, якщо цього ще не зробили, теж варто.

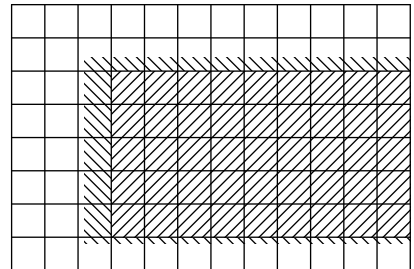
Але описані в попередньому розборі префіксні суми та їх двовимірне узагальнення *принципово неможливо* пристосувати до цієї задачі. Головним чином тому, що при їх використанні потрібне віднімання, яке є оберненою операцією до додавання: наддавали зайвого — можна відняти це зайве й лишиться те, що треба. А для мінімуму оберненої операції з такими властивостями не існує: ну знаємо, що на префіксі «5 9 2 6 3» максимумом є 9, на коротшому префіксі «5 9» максимумом теж є 9 — це щось дає для визначення максимуму на відрізьку «2 6 3», який входить у перший префікс і не входить у другий? Нічого і ніяк.

Основна ідея — запам'ятовувати результати деяких таких прямокутників, щоб вони потім були частинами потрібного. Саме частинами. Щоб не виникало потреби «відкинути зайве» (а отже, й потреби у оберненій операції). Є чимало різних варіантів уточнення цієї основної ідеї.

Варіант № 1. Можна, наприклад, запам'ятати максимум для прямокутника «1 10 1 10» (у термінах вхідних даних задачі), максимум для прямокутника «1 10 11 20», для «1 10 21 30», і так доки не закінчаться стовпчики; аналогічно, для «11 20 1 10», для «11 20 11 20», для «11 20 21 30», ...; для «21 30 1 10», для «21 30 11 20», для «21 30 21 30», ...; і так доки не закінчаться рядки.

Звісно, межі прямокутників-запитів не зобов'язані збігатися з межами цих штучно (неприродно) виділених квадратів, і це слід якось врахувати. Наприклад, так: де квадрат входить у прямокутник-запит повністю, там замість порівнянь поточного максимуму з усіма $10 \times 10 = 100$ числами цього квадрата робити одне-єдине порівняння, а де прямокутник-запит зачіпає квадрат лише частково — там, як і раніше, перебирати поелементно.

Наприклад, на рисунку зображено ситуацію для запиту «17 72 23 120» в масиві 90×120 : «сіткою» зображено квадрати, однією штриховкою виділено область, де вдалося використати відповіді цілих квадратів, іншою — область, де довелося порівнювати поелементно. Бачимо, що з $(120 - 23 + 1) \times (72 - 17 + 1) = 98 \times 56 = 5488$ елементів вдалося виділити $9 \times 5 = 45$ квадратів, тож поелементно лишилося перебрати тільки $5488 - 4500 = 988$.



Це явно менше... але *наскільки* (якщо говорити не про приклад, а в цілому)? Чи досить цього, щоб пройти усі блоки тестів? І чи точно за розмір квадрата варто брати саме 10, а не 7 чи 40?

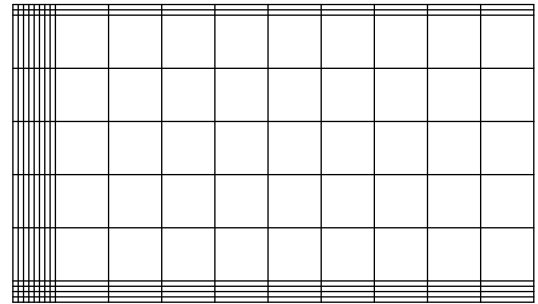
Спробуємо почати з останнього питання. Визначити оптимальний розмір точно неможливо, але, припускаючи, що $N \approx M$ (масив не надто витягнутий у ширину чи висоту), цей розмір повинен бути трохи меншим за \sqrt{N} (чи \sqrt{M} , що, за припущенням, приблизно те саме), бо при ще менших розмірах квадрата надто мало виграшу від того, що замінюємо одним порівнянням поелементний перебір квадрата, а при ще більших — забагато елементів лишається по краях прямокутника-запиту. (Охочі можуть отримати цей висновок як результат математично строгого аналізу засобами шкільного курсу алгебри; але тоді треба ввести якісь додаткові припущення, як-то « $M = N$ », «розмір прямокутника-запиту становить такий-то відсоток від розміру всього масиву», «при розмірі квадратів $a \times a$, ширина «рамки» з елементів, які доводиться обробляти поелементно, становить $a/2$ », а ці припущення можуть бути надто приблизними, тож точність результату все одно лишається сумнівною.)

В межах тих самих (розкритикованих за приблизність) припущень, можна сказати, що кожен з T запитів потребує $O(N\sqrt{N})$ (чи $O(N\sqrt{M})$, чи $O(M\sqrt{N})$) — все це те саме, бо вже припустили,

що $N \approx M$), тобто лише у \sqrt{N} разів менше, ніж «лобовий» варіант. Тим не менш, при акуратній та оптимальній у деталях реалізації це цілком може пройти блоки 1–3 й отримати 80 % балів.

Варіант № 2. Хоч би й на тому самому прикладі з попереднього рисунка, можна помітити, що «рамка», яку там пропонувалося переглядати поелементно, має досить тонкі й досить довгі верхню, нижню, ліву, праву сторони (конкретно на тому рисунку правої сторони нема, але то виключення, а не правило). Їх можна розглянути швидше, ніж поелементно, якщо ввести не один, а три допоміжні масиви: `maxInSquare` розмірами $N/a \times M/a$, де кожен елемент містить максимум відповідного квадрату $a \times a$ (як вже з'ясовано, a повинно бути трохи меншим \sqrt{N} ; зручно, щоб у подальших масивах a було те саме); `maxInHorStripe` розмірами $N/a \times M$, де кожен елемент містить максимум відповідної горизонтальної смужечки $1 \times a$; `maxInVertStripe` розмірами $N \times M/a$, де кожен елемент містить максимум відповідної вертикальної смужечки $a \times 1$.

Наприклад, для відповіді на такий самий, як на минулому рисунку, прямокутник-запит “17 72 23 120” за того самого розміру квадратиків та смужечок $a = 10$ були б використані квадрати, вертикальні й горизонтальні смужки та окремі елементи, зображені на рисунку.



Неважко перекоонатися, що (при згаданих обмеженнях $a \approx \sqrt{N}$, $N \approx M$) тепер формування відповіді на кожен запит займає $O(N)$: квадратів $a \times a$ сумарно не більш як $(N/a) \times (M/a) \approx (N/\sqrt{N}) \times (N/\sqrt{N}) \approx N$; горизонтальних смужок $1 \times a$ сумарно не більш як двічі (згори і знизу) по $a \times (M/a) \approx N$; вертикальних смужок $1 \times a$ сумарно не більш як двічі (ліворуч і праворуч) по $(N/a) \times a \approx N$; одинарних елементів не більш як чотири рази (лівий-верхній, правий-верхній, лівий-нижній та правий-нижній кути) по $a \times a \approx N$. Сумарно, $O(N)$, тобто ще у \sqrt{N} разів менше порівняно з «варіантом №1». Враховуючи, що підготувати всі ці масиви `maxInSquare`, `maxInHorStripe` та `maxInVertStripe` на етапі передобробки все ще можна за час $\Theta(N \cdot M)$, і що в цій задачі максимальні значення N , M дещо менші, ніж у попередній, а ліміт часу виконання більший, цього вже достатньо для проходження тестів усіх блоків і отримання повного балу. (Можна було б зробити й так, щоб цей підхід набрав повні бали також і в попередній задачі — тут же не використовуються ніякі особливості максимуму, які не виконуються для додавання. . . Але підхід із двовимірним узагальненням префіксних сум все-таки і значно ефективніший, і на суб'єктивну думку автора задач, красивіший, і досить легко пишеться; тому хотілося, щоб він усе-таки проявляв свої переваги.)

Варіант № 3. Цей варіант не кращий за попередні, але, в деяких смислах, простіший. Для кожного рядка окремо, спробуємо запам'ятати готові відповіді від кожного можливого j_1 до кожного можливого j_2 . Якби це вдалося, можна було б для потрібного діапазону $j_1 \ j_2$ щоразу просто брати готову відповідь (у межах поточного рядка, а рядки просто перебирати кожен окремо від i_1 до i_2). Щоправда, рівно в такому вигляді потрібен тривимірний масив розмірами $N \times M \times M$, і просто не вистачає пам'яті. Причому, не лише для останнього блоку тестів, а й для передостаннього: $400 \times 400 \times 400 \times 4 = 256000000 > 251658240 = 240 \times 1024 \times 1024$ (а пам'ять потрібна не лише на цей найбільший масив). Але, якщо провести деякі оптимізації, цей спосіб все-таки можна «впихнути» в обмеження. Одна з таких можливих оптимізацій — пам'ятати максимуми від *майже* кожного j_1 до *майже* кожного j_2 . Наприклад, через один: елемент `MX[0][0][0]` зберігає максимум серед пари елементів `data[1][1]`, `data[1][2]`; елемент `MX[0][0][1]` — максимум серед чотирьох елементів `data[1][1]`, `data[1][2]`, `data[1][3]`, `data[1][4]`; елемент `MX[0][0][2]` — максимум серед шести елементів від `data[1][1]` по `data[1][6]` включно, і так далі; потім, елемент `MX[0][1][0]` зберігає максимум серед пари елементів `data[1][3]`, `data[1][4]`; елемент `MX[0][1][1]` — максимум серед чотирьох елементів `data[1][3]`, `data[1][4]`, `data[1][5]`, `data[1][6]`, і так далі. При цьому об'єм пам'яті зменшується вчетверо, а відповідь (у межах рядка), хоч і не завжди готова для потрібних j_1 , j_2 , але завжди можна отримувати за $\Theta(1)$ (у межах рядка): вибрати той найкращий проміжок, де ліва межа або j_1 , або $j_1 + 1$, права — або j_2 , або $j_2 - 1$, тож лишається тільки взяти готовий результат проміжку, і, якщо треба, врахувати щонайбільше по одному елементу ліворуч та праворуч («якщо треба» включає у себе перевірки, чи вийшло так, що готовий проміжок недонакрив один

елемент, і якщо вийшло, то чи більший той один елемент за досі знайдений максимум). Але, повторимося, все це слід робити для кожного рядка від i_1 до i_2 окремо, тож кожен запит займає $O(N)$. І при цьому потрібен величезний масив, дуже витратний і за пам'яттю, і за часом його заповнення (само собою, потрібно визначати $MX[i][j1div2][j2div2]$,). Тим не менш, якщо реалізувати все це якнайефективніше у деталях, таким способом хоч і важко, але в принципі можна пройти навіть геть усі тести; пройти ж блоки 1–3 відносно неважко й менш «вилізаними» реалізаціями та менш ефективними мовами програмування.

А є щось, що має складність окремого запиту $o(N)$ (тобто, асимптотично строго меншу N)? Є, але ще складніше. Наприклад, можна почитати у літературі чи в Інтернеті про *дерево відрізків* та його двовимірне узагальнення. Двовимірне дерево відрізків має складність $\Theta(N \cdot M)$ на етапі передобробки і по $O(\log N \cdot \log M)$ на кожен з T запитів, і при цьому з мінімальними змінами годиться і для сум, і для максимумів. Тож цей підхід, хоч і складний, в принципі дозволяє «вбити обидві задачі C, D одним пострілом». Щоправда, для проходження задачі C (з більшими N, M при меншому обмеженні часу) треба ще й реалізувати його ефективно у деталях (асимптотика $O(\log N \cdot \log M)$, хоч не набагато, але більша, ніж $\Theta(1)$ в узагальнених префіксних сумах).

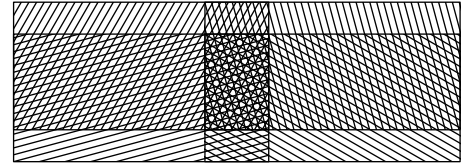
Розглянемо ще один підхід, придатний для задачі D, але не C. Він потребує чимало пам'яті: аж $\Theta(N \cdot M \cdot \log N \cdot \log M)$; це може бути двовимірний масив розмірами $\lceil \log_2 N \rceil \times \lceil \log_2 M \rceil$, елементами якого є двовимірні масиви розмірами до $N \times M$ кожен; або, що майже те саме, один 4-вимірний масив розмірами $\lceil \log_2 N \rceil \times \lceil \log_2 M \rceil \times N \times M$. Але зате цей алгоритм найшвидший з відомих автору задачі при великій кількості запитів (кожен з T запитів робиться за $\Theta(1)$, уся передобробка за $\Theta(N \cdot M \cdot \log N \cdot \log M)$), і, якщо його зрозуміти, не такий уже й складний для написання. А об'єм пам'яті, хоч і величезний, все ж менший, ніж у «варіанті №3».

Розмістимо чи то у двовимірному масиві двовимірних масивів, чи то у чотиривимірному масиві таку інформацію: *елемент $[a][b][i][j]$ дорівнює максимуму серед елементів прямокутника розмірами $2^a \times 2^b$, який займає рядки від i до $i + 2^a - 1$, стовпчики від j до $j + 2^b - 1$* . Тобто, елементи з індексами $[0][0][i][j]$ дорівнюють відповідним елементам масиву вхідних даних; елементи з індексами $[0][1][i][j]$ є максимумами з двох елементів (тих, що у масиві вхідних даних мають індекси $[i][j]$ та $[i][j+1]$); елементи з індексами $[0][2][i][j]$ — максимумами з 4-х елементів $[i][j]$, $[i][j+1]$, $[i][j+2]$ та $[i][j+3]$; і так далі. Наприклад, елементи з індексами $[5][3][i][j]$ дорівнюють максимумам з відповідних прямокутників розмірами $2^5 \times 2^3 = 32 \times 8$ масиву зі вхідних даних, з проміжком рядків від i до $i + 31$, стовпчики від j до $j + 7$ (і там, і там — включно). Подальші деталі (чи нумерувати i, j з 0, чи з 1; чи враховувати, що при підході «двовимірний масив двовимірних масивів» для більших a, b можна робити відповідний вкладений масив уже не розмірами $N \times M$, а меншим, як-то $N - 2^a + 1 \times M - 2^b + 1$; тощо) можна реалізовувати по-різному (тим паче, враховуючи особливості різних мов програмування), залишимо це на розсуд читачів. Головне, щоб зберігалися максимуми саме таких прямокутників.

Для дотримання вищезгаданої оцінки «передобробка за $\Theta(N \cdot M \cdot \log N \cdot \log M)$ » треба ще забезпечити, щоб такі максимуми формувалися швидко, за $\Theta(1)$ кожен. (Насправді, не обов'язково кожен, досить, щоб в середньому; але можна, щоб кожен.) Якось так: елементи $M[0][0][i][j]$ просто копіюються з масиву вхідних даних (чи навіть читаються з файлу вхідних даних прямо туди); усім $M[0][1][i][j]$ надається значення $\max(M[0][0][i][j], M[0][0][i][j+1])$, бо саме ці два елементи й утворюють прямокутник 1×2 ; усім $M[0][2][i][j]$ надається значення $\max(M[0][1][i][j], M[0][1][i][j+2])$, бо прямокутник 1×4 можна подати як два прямокутники 1×2 , один з яких «починається» (має лівий-верхній кут) у тій самій клітинці i, j , а інший на 2 клітинки правіше (саме там «починається» частина більшого, не покрита першим меншим); і так далі. Для заповнення $M[a][b][i][j]$ при $a > 0$, можна, наприклад, надавати всім $M[a][1][i][j]$ значення $\max(M[a-1][0][i][j], M[a-1][0][i+1][j])$, а далі заповнити всі $M[a][b][i][j]$ при $b > 0$ аналогічно тому, як вище детально пояснено для $a = 0$.

Коли увесь вміст описаного масиву M готовий, можна і зручно скористатися тим, що для будь-якого прямокутника-запиту можна підібрати такі a, b , щоб покрити цей прямокутник-запит рівно 4-ма прямокутниками однакових розмірів $2^a \times 2^b$. («Покрити» тут означає, що: кожен елемент

прямокутника-запиту належить хоча б одному з 4-х прямокутників $2^a \times 2^b$; жоден з 4-х прямокутників $2^a \times 2^b$ ніде ні на один елемент не виходить за межі прямокутника-запиту. Наприклад, на рисунку зображено, як 4 прямокутники 4×8 покривають прямокутник-запит 5×14 .) Як правило, ці 4 прямокутники $2^a \times 2^b$ накладаються частково, але якщо прямокутник-запит має розмір (один чи обидва), що є степенем/ями двійки, отримуємо чи то дотикання без накладань, чи то повну однаковість прямокутників (двох пар чи всіх 4-х); з цим розберіться самостійно, не забуваючи, що прямокутник-запит може мати в т. ч. й лише один рядок чи стовпчик.



Раніше було пояснено, що алгоритм, ефективний для задачі C, тут принципово незастосовний, бо додавання має обернену операцію віднімання, а максимум не має оберненої операції. Але максимум має іншу приємну властивість, якої не має сума: ідемпотентність. Це означає, що байдуже, чи враховувати кожен потрібний елемент рівно один раз, чи врахувати деякі один раз, деякі двічі, деякі тричі, тощо. Саме ідемпотентність максимуму й дає можливість отримувати максимум прямокутника-запиту як максимум з 4-х результатів для тих прямокутників, якими його покрили, і байдуже, якщо ці 4 прямокутники накладаються й тому деякі елементи враховані двічі й деякі по чотири рази.

Лишилося тільки навчитися швидко перетворювати опис прямокутника-запиту i_1, i_2, j_1, j_2 зі вхідних даних у те, які саме 4 прямокутники $2^a \times 2^b$ його покривають. Оскільки деякі технічні моменти вже віддано на розсуд читачів, тут мусимо теж віддати значну частину на розсуд читачів, лише відзначимо деякі факти. Для визначення тих a, b , при яких якраз вдається покрити прямокутник-запит, можна використати функцію логарифму з математичної бібліотеки, причому логарифм треба брати від вертикального розміру $i_2 - i_1 + 1$ та горизонтального розміру $j_2 - j_1 + 1$; щоправда, може бути трохи мороки з тим, щоб узяти саме двійковий логарифм (а не натуральний чи десятковий), та з тим, як правильно заокруглити дійсний результат до цілого. Можливий і альтернативний підхід: не вживаючи логарифми, починаючи з розміру 1 та індексу 0, пробувати щоразу збільшувати розмір удвічі, а індекс на 1, доки не виявиться, що розмір якраз від половини потрібного до потрібного. Це формально збільшує час виконання кожного запиту від $\Theta(1)$ до $O(\log N + \log M)$, але майже не впливає на реальний сумарний час виконання, виміряний у мілісекундах. Так буває (тим паче, що $O(\log N + \log M)$ — не багато, а логарифм — повільна функція). Ліві-верхні клітинки 4-х прямокутників повинні мати індекси, схожі на $(i_1; j_1)$, $(i_1; j_2 - 2^b + 1)$, $(i_2 - 2^a + 1; j_1)$, $(i_2 - 2^a + 1; j_2 - 2^b + 1)$ (можливо, якимось модифікованим згідно з тим, яка у Вас нумерація в масивах), тобто частина прив'язані до верхнього рядка / лівого стовпчика, частина — до нижнього рядка / правого стовпчика та розміру. Насамкінець, обчислення *степенів двійки* може бути реалізовано значно точніше&швидше, ніж для інших основ. Завдяки тому, що комп'ютери використовують двійкову систему числення, а 2^a являє собою $\underbrace{100\dots0}_{a \text{ штук}} (Bin)$, 2^a можна виразити як «узяти (двійкову) одиничку й дописати a нулів».

У Паскалі відповідний оператор називається `shl` (від *shift left*), у решті мов програмування, про які автору задачі відомо, що там таке є, як `<<` (два знаки «менше», записані підряд). Тож, наприклад, $i_2 - 2^a + 1$ можна виразити як `i2-(1<<a)+1` чи `i2-(1 shl a)+1` (Паскаль).

Абсолютно альтернативний підхід, який ледь не набрав 100 % балів. Цей підхід у деяких смислах трохи нечесний, але все ж заслуговує згадки. Повернімось до «лобового» підходу (перебирати для кожного прямокутника-запиту всі його елементи) і задумаємось, коли він працює особливо погано (довго). Очевидно, коли прямокутник-запит великий. А коли прямокутник-запит великий, то логічно припустити, що максимум серед нього буде близьким до максимуму серед усього масиву. Так що можна зробити так: прочитавши вхідні дані, і пам'ятати їх у звичайному двовимірному масиві, і сформувати масив трійок вигляду (номер рядка; номер стовпчика; значення) та відсортувати цей масив трійок за незростанням (від більших до менших) поля «значення»; при використанні ефективного сортування це займе $O(M \cdot N \cdot \log(M \cdot N))$ часу. Для кожного запиту, спочатку рахувати розмір прямокутника-запиту $(i_2 - i_1 + 1) \times (j_2 - j_1 + 1)$, і до деякої «планки» (якої? неясно, треба пробувати різні) запускати «лобовий» перебір всіх елементів прямокутника-запиту, а при її перевищенні запускати лінійний пошук за цим відсортованим масивом, тобто дивитися ці трійки від початку

Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду
Дистанційне тренування на ejudge.skipo.edu.ua, відкрите у січні 2020 р.

(найбільшого значення) послідовно, доки не знайдеться трійка, індекси якої потрапляють всередину прямокутника; поле «значення» цієї трійки і буде відповіддю на запит, бо щодо всіх більших значень усього початкового масиву було встановлено, що вони не належать прямокутнику-запиту.

Для цього алгоритму дуже важко вивести асимптотичну оцінку, бо важко оцінити, скільки треба пропустити трійок, щоб чергова потрапила в межі прямокутника-запиту (очевидно, в найгіршому випадку це можуть бути всі елементи початкового масиву, що не належать прямокутнику-запиту; але просто неможливо, щоб завжди так і було для різних прямокутників-запитів, які накладаються лише частково чи не перетинаються взагалі). Поки значення елементів масиву вхідних даних генерувалися випадково з рівномірним розподілом, однаковим для всіх індексів, такий розв'язок, хоч і програвав за часом найкращому згаданому, проходив абсолютно всі тести. Але він усе-таки не є *завжди* ефективним, тому було прийнято рішення додати контрприклад спеціально проти цього підходу. Суть цього контрприкладу — значення елементів масиву ніби й випадкові, але діапазон, з якого випадково вибирається значення, залежить від індексів (чим ближче до краю, тим більша верхня межа). Таким чином, за рахунок зсуву розподілів значно збільшується середня кількість трійок, які треба розглянути, перш ніж трапиться належна поточному прямокутнику-запиту; разом з тим, завдяки тому, що певна випадковість лишається, тест не вихолощується до того, щоб максимум завжди був лише в кутках (чи навіть лише на межах) прямокутника-запиту.

Оскільки такий алгоритм виявився досить добрим у середньому, щоб «ламатися» *лише* на тестах, спеціально сконструйованих проти нього, було прийняте рішення додати такий контрприклад лише в останній блок тестів, а в передостанній не додавати, тож 80 % балів він набирає. Але рішення «ладно, нехай він проходить передостанній блок» було досить волюнтаристським, його неможливо строго вивести з правил олімпіади. Відповідно, учасник, яким би розумним та досвідченим не був, не може адекватно оцінити, які блоки пройде такий розв'язок, доки не здасть його в ejudge. Тому, написання такого роду розв'язків незрозумілої асимптотичної складності малоприємне, бо є ризик, витративши на нього купу зусиль, отримати ту ж кількість балів, що й за «лобовий» підхід...

3.19 Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №72).

Задача С. «Прямокутні суми»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 256 мегабайтів

У цій задачі, алгоритмічна суть повністю повторює задачу С «Прямокутні суми» з II (районного/міського) етапу Всеукраїнської олімпіади з інформатики (програмування) по Черкаській області, що відбувся 14.12.2019 (коротка суть наведена в цитаті праворуч, детальніше можна подивитися у змаганні 70 сайту ejudge.skipo.edu.ua). Але значна частина технічних деталей організовані абсолютно інакше. Й мета цього комплекту з двох задач — ознайомитися з цим іншим технічним форматом, котрий з 2018/19 н. р. наявний на III (обласному) та IV (фінальному) етапах Всеукраїнської учнівської олімпіади з інформатики (програмування), але досі не згадувався на змаганнях, що проводяться в межах ejudge.skipo.edu.ua.

Дано прямокутну таблицю розмірами N рядків на M стовпчиків, елементи якої є натуральними числами. Потрібно **багатократно** знаходити суми всіх чисел, що потрапляють у деякі прямокутники цієї таблиці, утворені рядками з i_1 -го по i_2 -й та стовпчиками з j_1 -го по j_2 -й.

Розв'язки цієї задачі можна здавати *лише* мовами g++ (C++), fpc (Free Pascal, у режимі сумісності з Delphi), javac (Java). Додавати інші мови програмування поки що не планується; чи буде це змінено в майбутньому, невідомо. Скачайте за лінками, вказаними праворуч, файли згідно Вашої мови програмування. **Здавати в *ejudge* на перевірку треба лише виправлений фрагмент**

Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду
Дистанційне тренування на ejudge.ckipo.edu.ua, відкрите у січні 2020 р.

програми, відповідний вмісту «sample». Відповідно, редагувати слід у першу чергу його, бо при перевірці на ejudge будуть використані header та footer, підготовлені журі, а не відредаговані Вами. (Причому, header буде в точності такий, як наведено за посиланням, а footer буде дещо відрізнятися, й Вам не повідомляється, як саме.)

Протокол взаємодії (опис для C++).	Прямий лінк на файл	Його потім здавати?
Перелік доступних Вам функцій, підготовлених журі.	ejudge.ckipo.edu.ua/201920-2a-C-header-g++.cpp	Ні
	ejudge.ckipo.edu.ua/201920-2a-C-sample-g++.cpp	Так
	ejudge.ckipo.edu.ua/201920-2a-C-footer-g++.cpp	Ні
• <code>vector<vector<int>> > getArray(void)</code> — повертає масив, над елементами якого потрібно виконувати операції знаходження сум прямокутників-запитів. Нумерація рядків та стовпчиків починається з 0.	ejudge.ckipo.edu.ua/201920-2a-C-header-fpc.pas	Ні
Якщо Ви виконаєте у своєму коді дію <code>vector<vector<int>> arr = getArray();</code> , то надалі зможете взнавати кількість рядків як <code>arr.size()</code> , кількість стовпчиків як <code>arr[0].size()</code> , значення елемента з індексами <code>[i][j]</code> (при нумерації з 0) як <code>arr[i][j]</code> , тому всі три подальші функції цього переліку насправді непотрібні. Але, якщо зручно, можете ними користуватися (тоді не обов'язково користуватися цією функцією).	ejudge.ckipo.edu.ua/201920-2a-C-sample-fpc.pas	Так
	ejudge.ckipo.edu.ua/201920-2a-C-footer-fpc.pas	Ні
• <code>int getHeight(void)</code> — повертає кількість рядків у масиві.	ejudge.ckipo.edu.ua/201920-2a-C-header-javas.java	Ні
• <code>int getWidth(void)</code> — повертає кількість стовпчиків у масиві.	ejudge.ckipo.edu.ua/201920-2a-C-sample-javas.java	Так
• <code>int getElement(int i, int j)</code> — повертає значення елемента, розміщеного в i-му рядку, j-му стовпчику (вважаючи, що нумерація рядків та стовпчиків починається з 0). Якщо значення i або j виходять за межі діапазону $0 \leq i < \text{getHeight}()$, $0 \leq j < \text{getWidth}()$, виклик <code>getElement(i, j)</code> повертає 0.	ejudge.ckipo.edu.ua/201920-2a-C-footer-javas.java	Ні

Усі ці функції є просто функціями (а не методами якогось класу).

Перелік функцій, які Ви зобов'язані реалізувати.

- `void init(void)` — гарантовано буде викликатися один раз, на початку (раніше всіх запитів на підрахунок суми прямокутника). До результатів діяльності цієї функції ніяких вимог нема, на Ваш розсуд.
- `long long calcRectSum(int iTop, int iAfterBottom, int jLeft, int jAfterRight)` — повинна повернути суму всіх елементів, що потрапляють у прямокутник з індексами рядків $iTop \leq i < iAfterBottom$, стовпчиків $jLeft \leq j < jAfterRight$ (нумерація з 0, початок діапазону включно, кінець не включно).

Це не відповідає формату вхідних початкової задачі II (районного/міського) етапу (наприклад, прямокутник, який там описувався рядком вхідних даних "1 2 3 4", тут описується викликом `calcRectSum(0, 2, 2, 4)`); але, на думку автора задачі, для мови C++ так якраз природніше.

Обидві ці функції повинні бути просто функціями (а не методами якогось класу).

Протокол взаємодії (опис для Pascal). Фрагмент header містить рядок `"{$mode delphi}"`, тому вся компіляція відбуватиметься в режимі (максимально можливої для Free Pascal) сумісності з Delphi; зокрема, це забезпечує, що тип `integer` 32-бітовий.

Перелік доступних Вам типів та підпрограм, підготовлених журі.

- `type Arr2D = array[1..5000,1..5000] of integer` — Ви не зобов'язані скрізь користуватися лише такими масивами, але наступна процедура використовує саме цей тип.
- `procedure getArray(var arr : Arr2D; var N, M : integer)` — надає переданому масиву той вміст, для елементів якого слід шукати суми прямокутників-запитів, а ще двом змінним — розміри: спочатку (N) кількість рядків, потім (M) кількість стовпчиків. Нумерація рядків та стовпчиків починається з 1.

Оскільки і сам масив, і його розміри тут вже задані, всі три подальші функції цього переліку насправді непотрібні. Але, якщо зручно, можете ними користуватися (тоді не обов'язково користуватися цією процедурою).

- `function getHeight : integer` — повертає кількість рядків у масиві.

Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду
Дистанційне тренування на ejudge.skipo.edu.ua, відкрите у січні 2020 р.

- `function getWidth : integer` — повертає кількість стовпчиків у масиві.
- `function getElement(i, j : integer) : integer` — повертає значення елемента, розміщеного в *i*-му рядку, *j*-му стовпчику (вважаючи, що нумерація рядків та стовпчиків починається з 1). Якщо значення *i* або *j* виходять за межі діапазону $1 \leq i \leq \text{getHeight}$, $1 \leq j \leq \text{getWidth}$, виклик `getElement(i, j)` повертає 0.

Усі ці процедури/функції є просто процедурами/функціями (а не методами якогось класу).

Перелік підпрограм, які Ви зобов'язані реалізувати.

- `procedure init` — гарантовано буде викликатися один раз, на початку (раніше всіх запитів на підрахунок суми прямокутника). До результатів діяльності цієї процедури ніяких вимог нема, на Ваш розсуд.
- `function calcRectSum(iTop, iBottom, jLeft, jRight : integer) : int64` — повинна повернути суму всіх елементів, що потрапляють у прямокутник з індексами рядків $iTop \leq i \leq iBottom$, стовпчиків $jLeft \leq j \leq jRight$ (нумерація з 1, усі межі включно; це якраз відповідає формату вхідних початкової задачі II (районного/міського) етапу, і вважається природнім для мови Pascal).

Ці процедура та функція повинні бути просто процедурою та функцією (а не методами якогось класу).

Протокол взаємодії (опис для Java).

Перелік доступних Вам методів, підготовлених журі. Всі ці методи є статичними публічними класу `GetData`.

- `static public int[][] getArray()` — повертає двовимірний масив (масив масивів), над елементами якого потрібно виконувати операції знаходження сум прямокутників-запитів. Нумерація рядків та стовпчиків починається з 0.
Якщо Ви виконаєте у своєму коді дію `int[][] arr = GetData.getArray();`, то надалі зможете взнавати кількість рядків як `arr.length`, кількість стовпчиків як `arr[0].length`, значення елемента з індексами `[i][j]` (при нумерації з 0) як `arr[i][j]`, тому всі три подальші методи цього переліку насправді непотрібні. Але, якщо зручно, можете ними користуватися (тоді не обов'язково користуватися цим методом).
- `static public int getHeight()` — повертає кількість рядків у масиві.
- `static public int getWidth()` — повертає кількість стовпчиків у масиві.
- `static public int getElement(int i, int j)` — повертає значення елемента, розміщеного в *i*-му рядку, *j*-му стовпчику (вважаючи, що нумерація рядків та стовпчиків починається з 0). Якщо значення *i* або *j* виходять за межі діапазону $0 \leq i < \text{GetData.getHeight}()$, $0 \leq j < \text{GetData.getWidth}()$, виклик `GetData.getElement(i, j)` повертає 0.

Всі згадані тут методи є статичними публічними класу `GetData`.

Перелік методів, які Ви зобов'язані реалізувати. Всі ці методи повинні бути статичними публічними класу `UserSolver`. Весь клас повинен не бути публічним.

- `static public void init()` — гарантовано буде викликатися один раз, на початку (раніше всіх запитів на підрахунок суми прямокутника). До результатів діяльності цього методу ніяких вимог нема, на Ваш розсуд.
- `static public long calcRectSum(int iTop, int iAfterBottom, int jLeft, int jAfterRight)` — повинна повернути суму всіх елементів, що потрапляють у прямокутник з індексами рядків $iTop \leq i < iAfterBottom$, $jLeft \leq j < jAfterRight$ (нумерація з 0, початок діапазону включно, кінець не включно).
Це не відповідає формату вхідних початкової задачі II (районного/міського) етапу (наприклад, прямокутник, який там описувався рядком вхідних даних "1 2 3 4", тут описується викликом `UserSolver.calcRectSum(0, 2, 2, 4)`); але, на думку автора задачі, для мови Java так якраз природніше.

Всі згадані тут методи повинні бути статичними публічними класу `UserSolver`. Весь клас повинен не бути публічним. Ви можете, якщо бажаєте, додавати у клас `UserSolver` свої методи та/або поля,

Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду

Дистанційне тренування на ejudge.skipo.edu.ua, відкрите у січні 2020 р.

робити їх будь-якого рівня доступу (public/protected/private), але обов'язково *лише статичними*.
Примірники (instances) цього класу створюватися не будуть.

Оцінювання.	блок	тести	обмеження	передумови	бали
Оцінювання поблокове (бали за блок нараховуються лише в разі успішного проходження всіх тестів блоку). Стопчик «передумови» означає, що розв'язок запускатиметься і перевірятиметься на поточному блоці, лише якщо цей самий розв'язок успішно пройшов блоки, згадані в передумовах. Нерівності через кому (як-то « $100 \leq N, M \leq 400$ ») означають, що всі перелічені через кому змінні перебувають в указаному діапазоні. Обмеження на “ T ” вказують можливу кількість викликів calcRectSum для однієї й тієї ж таблиці. Обмеження на “ a_{ij} ” вказують можливі значення елементів таблиці.	1	1–5	$2 \leq N, M, T, a_{ij} \leq 9$	нема	10%
	2	6–10	$1 \leq N, M, T \leq 100; 1 \leq a_{ij} \leq 10^4$	блок 1	10%
	3	11–15	$1 \leq N \leq 25; 100 \leq M \leq 2019; 10^3 \leq T, a_{ij} \leq 10^5$	блок 1	10%
	4	16–20	$100 \leq N \leq 2019; 1 \leq M \leq 25; 10^3 \leq T, a_{ij} \leq 10^5$	блок 1	10%
	5	21–23	$100 \leq N, M \leq 400; 123 \leq T, a_{ij} \leq 12345$	блоки 1–2	10%
	6	24–27	$400 \leq N, M \leq 700; 12345 \leq T \leq 222555; 1 \leq a_{ij} \leq 10^9$	блоки 1–2, 5	15%
	7	28–31	$1000 \leq N, M \leq 2019; 12345 \leq T \leq 10^5; 1 \leq a_{ij} \leq 10^9$	блоки 1–6	15%
	8	32–35	$4000 \leq N, M \leq 5000; 10^5 \leq T \leq 10^6; 1 \leq a_{ij} \leq 10^9$	блоки 1–7	20%

У блоках 1 та 2 тести рівносильні відповідним тестам задачі С «Прямокутні суми» II (районного/міського) етапу Всеукраїнської олімпіади з інформатики по Черкаській області, що відбувся 14.12.2019; зокрема, 1-й тест 1-го блоку відповідає тесту, який там наведений в умові. В подальших блоках тести інші.

Розбір задачі. Тут мається на увазі розбір особливостей задачі з урахуванням того, що вона дана у варіанті «з header-ом та footer-ом»; розбір суто алгоритмічної її складової можна знайти через повідомлення у згаданому змаганні 70 сайту ejudge.skipo.edu.ua або за *цим прямим посиланням*.

Короткий огляд основних властивостей підходу з header-ами та footer-ами

1. Фрагмент sample, який Ви маєте дописати, не містить точки входу у програму (функції main (C++) / головної програми (Pascal) / метода main (Java)), і це слід так і зберегти. Точка входу (головна програма чи функція/метод main) все одно є у частині footer (мовою Java — частині header, але в будь-якому разі в частині, написаній автором задачі, а не Вами), тож якщо Ви напишете ще одну точку входу — найімовірніше, програма просто не скомпілюється. А якщо і скомпілюється (це можливо, зокрема, мовою Паскаль, де все після “END.” головної програми ігнорується), то Ви не знаєте всіх особливостей, які відрізняють «секретний» footer від вказаного на початку умови. Зокрема (але не тільки), в цій задачі «секретний» footer читає вхідні дані з деякого файлу, і Вам свідомо не повідомляють, як він називається. Тож замінити розроблену журі точку входу своєю Ви, найімовірніше, не зможете (а якщо і зможете, то неясно, яка в тому користь).
2. Фрагмент sample, який Ви маєте дописати, не читає вхідні дані з клавіатури/файлу й не виводить результат на екран чи у файл; це треба так і залишити; все це робить footer (написаний автором задачі, а не Вами). Тим паче, саме в цій задачі Ви не знаєте, звідки читати й куди виводити. А якби і знали, то *зайві* читання та/або виведення призводили б до порушення формату введення/виведення (яке, досить непередбачувано, може дати чи то вердикт “Помилка виконання”, чи то “Неправильний формат відповіді”, тощо).
3. Фрагмент sample, який Ви маєте дописати, зобов'язаний містити реалізацію кожної з підпрограм (функцій/процедур/методів), які Вам дано завдання реалізувати. Навіть якщо Ви вважаєте, що така дія непотрібна для досягнення остаточного результату (що саме в цій задачі, насправді, не так) — хоча б не прибирайте вже наявні оголошення й порожні реалізації.
4. Фрагмент sample, який Ви маєте дописати, може, якщо Вам так зручно, містити також додаткові Ваші підпрограми (функції/процедури/методи) та/або Ваші типи, включаючи класи.
5. Фрагмент sample, який Ви маєте дописати, може, якщо Вам так зручно, містити *організовану Вами додаткову інформацію, що зберігається між викликами тих підпрограм, які Вам слід реалізувати*. Це можуть бути змінні будь-яких типів, включно з масивами та/або об'єктами (примірниками класів):

Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду
Дистанційне тренування на ejudge.skipo.edu.ua, відкрите у січні 2020 р.

- (a) для C++ та Pascal це можуть бути глобальні змінні;
- (b) для Java це можуть бути статичні поля всередині того класу, в якому Вам слід реалізувати вказані методи.

(Підказка: варто, щоб підпрограма `init` заповнювала ті глобальні змінні чи статичні поля результатами передобробки, а підпрограма `calcRectSum` використовувала їх для свого пришвидшення.)

6. Коментарі з текстом “TODO”, наявні у фрагменті `sample`, рекомендують саме там і дописувати Ваш код, але ні до чого істотного не зобов'язують. Їх можна прибирати (для компілятора це просто коментарі); можна (і нерідко треба) писати свій код також у деяких інших місцях фрагмента `sample`; тощо. Але якщо не вписувати свій код ні в які з місць, позначених коментарями “TODO”, правильного розв'язку не вийде.

Як мені локально (на своєму комп'ютері) запускати код, поділений між трьома файлами? Можна просто «склеїти» (наприклад, копіюваннями та вставками тексту, або написавши у командному рядку `copy "201920-2a-C-header-g++.cpp"+"201920-2a-C-sample-g++.cpp"+"201920-2a-C-footer-g++.cpp" C-all.cpp` або ще якось) `header`-, `sample`- та `footer`-фрагменти (саме в цьому порядку) в один файл. Кому це зручно — може так і робити (тим паче, що тепер є вибір — чи традиційно здавати файл із розв'язком, чи потрібну частину коду виділяти й копіювати у середовищі програмування та вставляти безпосередньо у сторінку `ejudge`). Але в цього простого підходу є недолік: при кожній спробі надіслати розв'язок в `ejudge` треба вибирати й копіювати ту частину коду, яка є дописаною версією колишнього фрагменту `sample`. Це і займає час, і збільшує ризик помилитись і здати щось не те. (Зазвичай, це призводитиме до помилки компіляції; але за деяких обставин можливі й якісь інші викривлення процесу перевірки, й аналіз результатів таких викривлених перевірок може змарнувати багато часу й зусиль. Особливо, мовою Pascal, де весь вміст файлу після “END.” головної програми ігнорується.) Тому, непогано було б мати спосіб здавати в `ejudge` повний вміст усього файлу.

Один з не універсальних, але реальних способів такий: **якщо** Ви пишете мовою C++ і користуєтеся на своєму комп'ютері *Microsoft Visual Studio в Debug-режимі*, можна робити, як вказано праворуч, і виходитиме, що фрагменти від “`#ifdef _DEBUG`” до “`#endif`” будуть копіюватися й виконуватися на Вашому локальному комп'ютері, але пропускатимуться при компіляції (а отже, й виконанні) на `ejudge`. Якщо Ви хочете користуватися локально іншим середовищем програмування (або студією в іншому режимі), то для користування цим прийомом треба розібратися з так званою умовною компіляцією, і самостійно робити, щоб у Вашому середовищі програмування (не в цій самій програмі, яка здаватиметься в `ejudge`, а в середовищі (IDE), в якому Ви запускаєте цю програму на Вашому локальному комп'ютері) було означене (defined) деяке ім'я, не означене на сервері `ejudge`.

```
#ifdef _DEBUG
... (багато рядків - увесь вміст файлу
"201920-2a-C-header-g++.cpp") ...
#endif

... (багато рядків - увесь вміст
відредагованого Вами колишнього файлу
"201920-2a-C-sample-g++.cpp") ...

#ifdef _DEBUG
... (багато рядків - увесь вміст
(можливо, відредагованого Вами)
колишнього файлу
"201920-2a-C-footer-g++.cpp") ...
#endif
```

Для Pascal ніби й досить замінити “`#ifdef ...`” на “`{ifdef ...}`” та “`#endif`” на “`{endif}`”, але конкретно слово “`_DEBUG`”, найімовірніше, не спрацює. Треба знайти аналогічне слово, означене (defined) саме у Вашому середовищі, але не на `ejudge` (або зробити таке означення самостійно). Як (і чи можна) зробити щось аналогічне для Java, автору тексту невідомо (умовної компіляції в Java, начебто, не існує, а просто розкласти різні класи по різних файлам не можна, бо для «звичайної» компіляції багатофайлового проекту ті класи повинні бути `public`, а для тієї, що відбувається в `ejudge`, вони не можуть бути `public`).

Якщо хтось може запропонувати зручніші чи універсальніші способи — прохання так і зробити (питанням у ejudge.skipo.edu.ua, чи повідомленням користувачеві IlyaSk на сайті codeforces.com, чи ще якось).

Редагувати можна **тільки** фрагмент «`sample`»? Залежить від того, як і навіщо. Наприклад, підключити у `header` 100 бібліотек і використовувати їх у `sample` — дуже погана ідея, бо хоч воно і працюватиме на локальному комп'ютері, на `ejudge` не скомпілюється, бо тих додаткових бібліотек

Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду

Дистанційне тренування на `ejudge.skiro.edu.ua`, відкрите у січні 2020 р.

як не було, так і нема у `header-i`, підготовленому журі. Але, наприклад, якщо в наданому журі `footer-i` вхідні дані читаються з клавіатури, а Ви хочете з файлу (чи навпаки) — таке редагування `footer-a` може бути доброю ідеєю; причому, Вам не треба буде при здачі в `ejudge` перероблювати спосіб введення на початковий, бо Ваші правки `footer-a` лишаться на Вашому комп'ютері й не вплинуть на перевірку на `ejudge`.

Є й чимало інших випадків, коли доцільно редагувати `footer` (див. далі); потреба редагувати `header` з'являється значно рідше, але іноді все ж буває. Наприклад, Ви можете підключити у своєму `header-i` бібліотеки, що будуть використовуватися лише у Вашому `footer-i`. Тільки тоді треба слідкувати, щоб не використати ті додаткові бібліотеки в основній частині, що здаватиметься в `ejudge`.

Чи є ще якісь переваги для учасника в цього підходу з `header-ами` та `footer-ами`?

Їх, насправді, чимало (хоч вони й незвичні та неочевидні тим, хто досі з таким не стикався; само собою, незручності в такого підходу теж є, але зараз не про них). Одна з таких переваг — зникає дуже вже велика залежність часу (тривалості) роботи програми учасника від використаних засобів читання вхідних даних та виведення результатів (для C++, `scanf` та `printf` проти `cin` та `cout`; для java, поєднання `FileReader`, `BufferedReader` та `StreamTokenizer` проти `Scanner` — такі зміни, будучи технічними, а не алгоритмічними, кардинально впливають на оцінку розв'язку, бо загальний час роботи всієї програми (найефективнішого з алгоритмів, двовимірного узагальнення префіксних сум; див. *розбір алгоритмічної складової задачі*) відрізняється не на відсотки і навіть не на десятки відсотків, а у багато разів). Див. також наступний пункт.

Зручність підходу з `header-ами` та `footer-ами` для порівняння результатів різних алгоритмів. (Часто, але не завжди, це основна перевага цього підходу з точки зору учасника.) У цій задачі не проблема написати «лобовий» розв'язок, де `calcRectSum` додає всі значення відповідного прямокутника-запиту поелементно. Само собою, цей «лобовий» розв'язок не має шансів пройти щось більше, ніж блоки 1–2, тож для отримання більших балів слід писати щось ефективніше (див. *розбір алгоритмічної складової задачі*). Те ефективніше виявляється складнішим (особливо, коли не читати готові формули у розборі, а самостійно придумувати їх під час туру), в ньому легше допуститися помилок, і з цим треба якось працювати. Зокрема, шукати помилки в ефективнішій&складнішій версії `calcRectSum`, якщо вона працює правильно часто, але не завжди, й `ejudge` це ловить, а Ви не бачите, коли виникають помилки.

Тут часто корисно робити, як праворуч: ефективнішу&складнішу `calcRectSum` розмістити у `sample-фрагменті`, а підпрограму з «лобовим» підходом назвати `calcRectSumSlow` та розмістити у `footer-фрагменті`; зробити, щоб у `footer-фрагменті` послідовність прямокутників-запитів не читалася зі вхідних даних, а перебиралися всі можливі варіанти, і при знаходженні відмінності результатів двох алгоритмів, наприклад, тут же виводилося, де саме (при яких `i1`, `i2`, `j1`, `j2`) це сталося. Й усе це можна організувати так, щоб `sample-фрагмент` лишився у вигляді, цілком готовому до негайної здачі в `ejudge`.

```
for i1:=1 to N do
  for i2:=i1 to N do
    for j1:=1 to M do
      for j2:=j1 to M do begin
        res1:=calcRectSum(i1,i2,j1,j2);
        res2:=calcRectSumSlow(i1,i2,j1,j2);
        if res1<>res2 then begin
          ... ..
        end
      end
    end
  end
end
```

Звісно, такий запуск реалізацій і порівняння результатів — не панацея; він не знайде помилок, однакових в обох реалізаціях (наприклад, коли і там, і там є однакове переповнення типу), чи якусь проблему, що виникає лише на настільки великих розмірах масиву та прямокутників-запитів, що чекати «лобового» підходу задовго... Але це все ж спосіб шукати помилки, доступний під час туру (навіть без `ejudge`!), і є чимало ситуацій, коли ним варто скористатися. Й у варіанті з `header-ами` та `footer-ами` все це робити зручніше, ніж у традиційному.

У 72-му змаганні відкрита частина тестів, так вони в геть зовсім різному форматі; частина схожа на формат 70-го змагання (й то не зовсім), а частина геть не схожі. Так і є. Але учасникам просто не треба про це хвилюватися. Просто робіть ту частину, яка впливає з умови задачі та загальних правил і традицій «алгоритмічних» олімпіад (*«реалізувати підпрограму `calcRectSum` якнайефективніше, враховуючи, що суми треба рахувати багатократно для різних прямокутників одного масиву, і що Вам дається можливість один раз зробити передобробку в підпрограмі*

Дорішування деяких задач II етапу (14.12.2019) в режимі здачі частини коду

Дистанційне тренування на ejudge.ckipo.edu.ua, відкрите у січні 2020 р.

`init`»), і не надто звертайте увагу на те, що там читається з файлу/клавіатури та виводиться на екран чи у файл.

А те, що у 72-му змаганні найбільші тести не читаються з файлу повністю, а частково читаються й частково догенеровуються у footer-і секретним для учасників способом — насправді ще одна перевага підходу з header-ами та footer-ами. Адже це дозволяє різко зменшити (вже згадану і дуже істотну для «традиційної» версії цієї задачі з 70-го змагання) проблему, що читання вхідних даних і виведення результату займає дуже вже значну частину всього часу виконання програми. У ще більшій мірі стає правдою, що виграють ті, хто може запропонувати кращий спосіб вирішення основної задачі (реалізувати підпрограму `calcRectSum` якнайефективніше), а не ті, хто добре знає всякі прийоми швидкого читання, які в цілому корисні, але не мають ніякого стосунку до конкретної задачі.

Задача D. «Прямокутні максимуми»

Вхідні дані: Або клавіатура, або `input.txt`

Обмеження часу: 1 сек

Результати: Або екран, або `output.txt`

Обмеження пам'яті: 256 мегабайтів

Алгоритмічна суть задачі повністю повторює задачу D «Прямокутні максимуми» з II (районного/міського) етапу Всеукраїнської олімпіади з інформатики (програмування) по Черкаській області, що відбувся 14.12.2019 (див. змагання 70 сайту ejudge.ckipo.edu.ua). Єдина відмінність від попередньої задачі — потрібно знаходити максимальне число прямокутника, а не суму.

Всі технічні умови такі ж, як у попередній задачі цього комплекту, крім того, що: (1) одна з підпрограм (функцій для C++ та Pascal, методів для Java), які Ви повинні реалізувати, називається `calcRectMax` замість `calcRectSum` та повинна вертати 32-бітовий тип (`int/integer/int` залежно від мови програмування) замість 64-бітового; (2) мовою Pascal, означення типу масиву має вигляд `type Arr2D = array[1..1234, 1..1234] of integer`.

Оцінювання.

Аналогічно попередній задачі, але обмеження блоків та розподіл балів між ними трохи інші (див. праворуч). Аналогічно попередній задачі, у блоках 1 та 2

тести в точності збігаються з відповідними тестами задачі D «Прямокутні максимуми» з II (районного/міського) етапу Всеукраїнської олімпіади з інформатики (програмування) по Черкаській області, що відбувся 14.12.2019; зокрема, 1-й тест 1-го блоку відповідає тому тесту, котрий там наведений в умові. В подальших блоках тести інші.

Розбір задачі. Розбирати майже нема чого, бо треба лише врахувати і розбір алгоритмічної складової, який можна знайти через повідомлення у згаданому змаганні 70 сайту ejudge.ckipo.edu.ua або за [цим прямим посиланням](#), і розбір цього способу з header-ами та footer-ами з попередньої задачі. Варто тільки наголосити, що збільшення розмірів масиву та кількості запитів призводить до того, що, як правило, асимптотично ефективніші способи сильніше проявляють свої переваги (особливо, якщо рахувати у відсотках набраних балів, а не кількості набраних балів чи пройдених блоків).

	Прямий лінк на файл	Його потім здавати?
C++	ejudge.ckipo.edu.ua/201920-2a-D-header-g++.cpp	Ні
	ejudge.ckipo.edu.ua/201920-2a-D-sample-g++.cpp	Так
	ejudge.ckipo.edu.ua/201920-2a-D-footer-g++.cpp	Ні
Pascal	ejudge.ckipo.edu.ua/201920-2a-D-header-fpc.pas	Ні
	ejudge.ckipo.edu.ua/201920-2a-D-sample-fpc.pas	Так
	ejudge.ckipo.edu.ua/201920-2a-D-footer-fpc.pas	Ні
Java	ejudge.ckipo.edu.ua/201920-2a-D-header-javac.java	Ні
	ejudge.ckipo.edu.ua/201920-2a-D-sample-javac.java	Так
	ejudge.ckipo.edu.ua/201920-2a-D-footer-javac.java	Ні

блок	тести	обмеження	підумки	бали
1	1–5	$2 \leq N, M, T, a_{ij} \leq 9$	нема	10%
2	6–10	$1 \leq N, M, T \leq 100; 1 \leq a_{ij} \leq 10^4$	блок 1	10%
3	11–14	$100 \leq N, M \leq 400; 123 \leq T, a_{ij} \leq 43210$	блоки 1–2	15%
4	15–18	$400 \leq N, M \leq 500; 54321 \leq T \leq 222555; 1 \leq a_{ij} \leq 10^9$	блоки 1–3	15%
5	19–22	$555 \leq N, M \leq 1111; 12345 \leq T \leq 54321; 1 \leq a_{ij} \leq 10^9$	блоки 1–3	15%
6	23–26	$500 \leq N, M \leq 700; 5 \cdot 10^5 \leq T \leq 10^6; 1 \leq a_{ij} \leq 10^9$	блоки 1–4	15%
7	27–32	$12 \leq N, M \leq 1234; 8 \cdot 10^5 \leq T \leq 10^6; 1 \leq a_{ij} \leq 10^9$	блоки 1–6	20%

Втім, асимптотично добрий за часом (calcRectMax за $\Theta(1)$) спосіб з чи то чотиривимірним масивом розмірів $\lceil \log_2 N \rceil \times \lceil \log_2 M \rceil \times N \times M$, чи то двовимірним масивом двовимірних масивів тепер не проходить тести блоків 5 та 7, бо не вкладається у ліміт пам'яті. Чи можна покращити (зменшити) його витрати пам'яті, не надто погіршивши (збільшивши) його витрати часу? До певної (достатньої, щоб пройшли всі тести) міри так, якщо змінити розміри прямокутників з $2^a \times 2^b$ на, наприклад, $6^a \times 6^b$ чи $8^a \times 8^b$. Об'єм пам'яті зменшиться у $\approx 7\text{--}7,5$ разів ($\lceil \log_8 1234 \rceil = \lceil \log_6 1234 \rceil = 4$ проти $\lceil \log_2 1234 \rceil = 11$, внаслідок чого замість $11 \times 11 = 121$ масивів по 1234×1234 лишається тільки $4 \times 4 = 16$ таких масивів; $\frac{121}{16}$ навіть більше 7,5, але є ще інші витрати пам'яті). Щоправда, визнавати відповідь на запит вибором максимуму з 4-х (2×2) елементів масиву вже не вийде, але можна вибором максимуму з не більш, як 36-ти (6×6) чи не більш, як 64-х (8×8) елементів відповідно.

Або, можна все-таки знайти в Інтернеті чи літературі та реалізувати двовимірне узагальнення д'єрева відрізків — не таке воно вже й складне, а з точки зору потенційної корисності в інших задачах набагато краще тим, що дозволяє не лише виконувати багато запитів щодо незмінного масиву, а й змінювати його елементи.

3.20 Обласна інтернет-олімпіада 2020/21 н. р.

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №30).

Задача А. «Василько та циркуль–1»

Василько взяв великого циркуля та зайшов до порожньої кімнати, підлога якої являє собою прямокутник, вершини якого мають координати $(0; 0)$, $(A; 0)$, $(A; B)$, $(0; B)$. Поставивши циркуль на підлозі цієї кімнати в точці з координатами $(x; y)$ (ця точка розміщена всередині (внутрі, inside) цієї кімнати, тобто $0 < x < A$, $0 < y < B$), він почав будувати кола (окружності, circles), радіусами r , $2r$, $3r$, ... (радіус кожного наступного на r більший за радіус попереднього).

Скільки повних кіл може побудувати в цій кімнаті Василько?

Всі згадані величини A , B , x , y , r гарантовано натуральні (цілі додатні).

Якщо для побудови чергового кола потрібно чиркати циркулем об стіну — вважати, що побудувати відповідне коло вже неможливо.

У цій задачі треба здати не програму, а вираз: вписати його (сам вираз, не назву файлу) у відповідне поле перевіряючої системи і відправити на перевірку. Правила запису виразу: можна використовувати десяткові числа, арифметичні дії “+” (плюс), “-” (мінус), “*” (множення), “/” (ділення дробове, наприклад, $17/5=3,4$), “//” (ділення цілочисельне, наприклад, $17//5=3$), круглі дужки “(” та “)” для групування та зміни порядку дій, а також функції `min` та `max`. Формат запису функцій: після `min` або `max` відкривна кругла дужка, потім перелік (через кому) виразів, від яких береться мінімум або максимум, потім закривна кругла дужка; кількість виразів, з яких береться мінімум або максимум, може бути довільна (можна хоч з 2-х, хоч з 3-х, хоч з 4-х, хоч з 10-и, хоч з 1-го). Дозволяються пропуски (пробіли), але, звісно, не всередині чисел і не всередині імен `min` та `max`. Згадані в умові A , B , x , y , r повинні називатися саме A , B , x , y , r , а не якимось інакше. Всі ці букви повинні бути латинські (англійські), причому A та B — великі, x , y та r — маленькі. Функції `min` та/або `max` слід писати маленькими латинськими (англійськими) буквами.

Наприклад, на цьому рисунку $A=120$, $B=90$, $x=40$, $y=45$, $r=8$, а пунктирні лінії (яких нема ні в кімнаті, ні в решті умови задачі, лише на рисунку) проведені там, де координата кратна 10. Як бачимо, правильна числова відповідь (кількість кіл) для цих A , B , x , y , r дорівнює 4 (п'яте коло можна було б намалювати, подряпавши стіну, але Василько цього не робитиме).

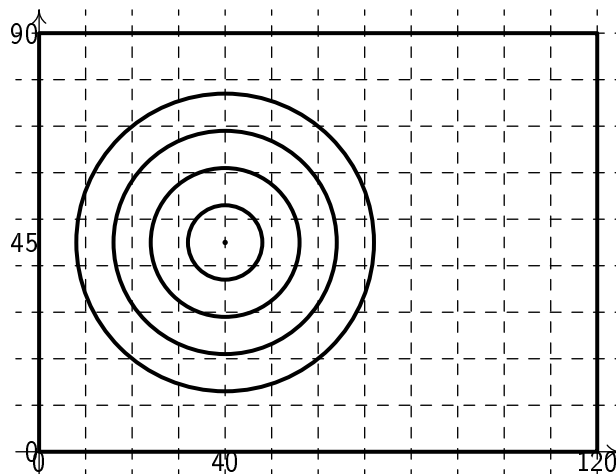
Розбір задачі.

Для початку, припустимо, ніби є лише «ліва стіна», тобто вісь Oy . Тоді кількість кіл, які поміщаються між x -координатою 0 та x -координатою x (тією, котра згідно з вимогами до формули називається x) становить *приблизно* $\frac{x}{r}$; але треба не просто поділити, і навіть не просто поділити

цілочисельно (що, згідно умови, позначається “//”), а ще й врахувати (й це видно з наведеного в умові прикладу), що коли відстань в точності кратна радіусу, то останнє коло вже не малюється. Тобто, треба рахувати кількість кіл, які поміщаються у трохи меншу відстань; враховуючи цілочисельність, «трохи меншу» фактично означає «на 1 меншу». Тобто, цю кількість можна виразити як $(x-1)//r$.

Оскільки практично ті самі міркування можна повторити також щодо відстані y від нижньої стіни, відстані $A-x$ від правої стіни та відстані $B-y$ від верхньої стіни, остаточною відповіддю може бути, наприклад, “ $\min((x-1)//r, (y-1)//r, (A-x-1)//r, (B-y-1)//r)$ ”.

Само собою, на повні бали зараховується не лише цей вираз, а будь-який правильний. Наприклад, його можна перетворити до рівносильного “ $(\min(x, y, A-x, B-y)-1)//r$ ” чи ще якимось.



Задача В. «Дуже важливі числа»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 256 мегабайтів

Марічка знає два Дуже Важливі Числа — A та B . Усі числа між A та B теж дуже важливі для Марічки. Сашко пообіцяв Марічці надрукувати всі дуже важливі для Марічки числа, а також порахувати їх кількість. Причому, якщо $A < B$, то Сашко надрукує ці числа в порядку зростання; інакше — в порядку спадання.

Вхідні дані. Програмі на вхід подаються два Дуже Важливі Числа A та B ($-100 \leq A \leq 100$, $-100 \leq B \leq 100$, кожне число у своєму рядку).

Результати. У перший рядок виведіть повідомлення про кількість дуже важливих для Марічки чисел (див. приклади; перший рядок завжди виводити символ-у-символ, як у прикладі, замінюючи лише число). Далі виведіть усі дуже важливі для Марічки числа так, як це пообіцяв надрукувати Сашко.

Приклади:

Вхідні дані	Результати
1 5	5 very important numbers: 1 2 3 4 5
5 2	4 very important numbers: 5 4 3 2

Розбір задачі.

Задача, насправді, досить проста й нудна, в ній нема чого пояснювати. Треба просто перевірити, чи $A \leq B$, чи $A > B$, й залежно від цього виразити кількість чисел як $B - A + 1$ чи $A - B + 1$ відповідно, та пустити цикл в будь-якому разі від A до B , але (залежно від все тієї ж умови) чи то у бік збільшення, чи то у бік зменшення відповідно. Головне, на чому можна було помилитися — випадок $A = B$. Його цілком можна, як тут і запропоновано, включити до випадку $A \leq B$, але треба не забути про це і перевірити, чи це зроблено правильно.

Задача С. «Василько та циркуль-2»

Вхідні дані: Або клавіатура, або input.txt Обмеження часу: 1 сек
Результати: Або екран, або output.txt Обмеження пам'яті: 256 мегабайтів

Василько знову взяв великого циркуля та зайшов до іншої порожньої кімнати, підлога якої теж являє собою прямокутник, вершини якого мають координати $(0; 0)$, $(A; 0)$, $(A; B)$, $(0; B)$, і знову

має намір поставити циркуль на підлозі цієї кімнати в точці з координатами $(x; y)$ (яка теж розміщена всередині (внутрі, inside) цієї кімнати, тобто $0 < x < A$, $0 < y < B$) та побудувати кола (окружности, circles), радіусами r , $2r$, $3r$, ... (радіус кожного наступного на r більший за радіус попереднього). Тільки тепер виявилось, що підлога цієї кімнати має N круглих дірок: з центром у точці $(x_1; y_1)$ та радіусом r_1 , з центром у точці $(x_2; y_2)$ та радіусом r_2 , ..., з центром у точці $(x_N; y_N)$ та радіусом r_N .

Василько дуже боїться зламати свій циркуль, тому пропускатиме ті кола, які хоча б частково потрапляють у хоча б одну з дірок, або торкаються до хоча б однієї дірки (у смислі, звичайному для «кола торкаються зовнішнім/внутрішнім чином»). За рахунок цього, кількість намальованих кіл може виявитися меншою, чим у задачі А. Можлива навіть така сумна ситуація, що точка з координатами $(x; y)$, куди Василько хотів поставити циркуль, сама потрапляє у деяку з дірок чи на межу дірки; тоді Василько не зможе намалювати жодного кола.

Скільки повних кіл може побудувати в цій кімнаті Василько?

Вхідні дані. Перший рядок містить два числа A та B . Другий рядок містить два числа x та y . Третій рядок містить єдине число r . Смысл чисел A , B , x , y , r точнісінько такий самий, як у задачі А; всі ці числа є натуральними (цілими додатними); обмеження на діапазони чисел див. у розділі «Оцінювання». Четвертий рядок містить єдине ціле невід'ємне число N — кількість круглих дірок у підлозі. Подальші N рядків, з 5-го по $(N + 4)$ -й, містять рівно по три числа кожен: x_i , y_i та r_i відповідної дірки в підлозі. Всі ці x_i , y_i , r_i теж є натуральними (цілими додатними); обмеження на діапазони теж див. у розділі «Оцінювання». Гарантовано, що всі дірки повністю розміщені всередині кімнати, та що різні дірки не можуть ні перетинатися, ні дотикатися, ні бути вкладеними одна в іншу.

Результати. Виведіть єдине ціле невід'ємне число — кількість кіл, які зможе намалювати Василько.

Приклад:

Вхідні дані	Результати	
120 90 40 45 8 2 60 30 5 90 20 15	3	

Оцінювання. Тест 1 є тестом з умови і сам по собі не приносить балів.

Тести 2–11 оцінюються кожен окремо і незалежно від того, чи пройшли інші тести; можуть принести по 10 балів (2%) кожен. У них виконуються такі додаткові обмеження: $1 < x < A < 200$, $1 < y < B < 200$, $0 \leq N \leq 20$.

Тести 12–15 оцінюються блоком (тут і далі це означає, що бали нараховуються, лише якщо всі тести цього діапазону пройшли успішно), завжди (незалежно від того, чи пройшли попередні тести). У них виконуються такі додаткові обмеження: $1 < x < A < 200$, $1 < y < B < 200$, $N = 0$ (дірок нема). Цей блок може принести 25 балів (5%).

Тести 16–20 оцінюються блоком, лише якщо успішно пройшли всі попередні тести 1–15. У них виконуються такі додаткові обмеження: $1 < x < A < 123$, $1 < y < B < 123$, $1 \leq N \leq 10$. Цей блок може принести 50 балів (10%).

Тести 21–30 оцінюються блоком, лише якщо успішно пройшли всі попередні тести 1–20. У них виконуються такі додаткові обмеження: $1 < x < A < 12345$, $1 < y < B < 12345$, $0 \leq N \leq 100$. Цей блок може принести 125 балів (25%).

Тести 31–40 оцінюються блоком, лише якщо успішно пройшли всі попередні тести 1–30. У них виконуються такі додаткові обмеження: $1 < x < A < 10^6$, $1 < y < B < 10^6$, $0 \leq N \leq 2020$. Цей блок може принести 100 балів (20%).

Тести 41–50 оцінюються блоком, лише якщо успішно пройшли всі попередні тести 1–40. У них виконуються такі додаткові обмеження: $1 < x < A < 10^7$, $1 < y < B < 10^7$, $0 \leq N \leq 98765$. Цей блок може принести 100 балів (20%).

В усіх тестах усіх блоків виконується також обмеження $1 \leq r \leq \min(A, B)$.

Примітки. Зображення наведене лише для кращого розуміння умови, Ваша програма малювати його не повинна.

Описи блоків наведені лише для пояснення системи оцінювання; Ваша програма не зобов'язана аналізувати, тест з якого блоку вона обробляє, і кожна Ваша програма-спроба буде оцінюватися заново на всіх тестах від початку й або до кінця, або доки не виявиться, що якийсь із блоків не пройдено.

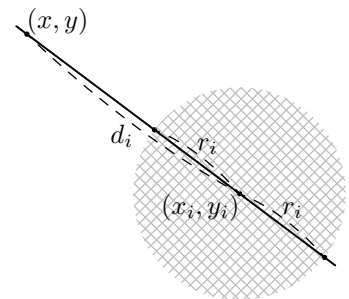
Розбір задачі.

Само собою, для розуміння цієї задачі варто спочатку зрозуміти задачу А (стор. 121), бо ця задача є подальшим розвитком тієї.

В будь-якому (від найпростішого до найефективнішого) розв'язку цієї задачі використовуються такі факти та спостереження аналітичної геометрії:

- відстань між точками з координатами (x, y) та (x_i, y_i) становить $\sqrt{(x - x_i)^2 + (y - y_i)^2}$ (оскільки система координат прямокутна, цю формулу можна отримати як наслідок з теореми Піфагора);
- коли *центр* круга має відстань $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$ від точки (x, y) , з якої як з центра Василько намагається малювати свої кола, то найближча точка цього круга розміщена на відстані $d_i - r_i$, а найдалша — на відстані $d_i + r_i$ (де r_i — радіус цього круга; нижня межа $d_i - r_i$ взята у припущенні, що точка (x, y) не потрапляє всередину цього кола; якщо потрапляє, величина $d_i - r_i$ стає від'ємною; взагалі-то відстані від'ємними не бувають, але в рамках цієї задачі це означає, що потрібно за спеціальним розгалуженням виводити відповідь 0 і не рахувати далі по суті, тому можна все-таки рахувати за формулою $d_i - r_i$, заодно перевіряючи знак).

(Аргументувати, що (при перебуванні (x, y) ззовні круга) проміжок відстаней між (x, y) та різними частинами круга справді становить «від $d_i - r_i$ до $d_i + r_i$, де $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$ », можна, наприклад, так: проведемо пряму, що проходить через (x, y) та (x_i, y_i) ; межі круга радіусом r_i завжди, в тому числі й на цій прямій, перебувають на відстані r_i від його центру; так і виходить, що найближча до (x, y) точка круга розміщена на відстані $d_i - r_i$ (на r_i ближче, чим центр), а найдалша на відстані $d_i + r_i$ (на r_i далі, чим центр).)



Очевидний підхід, 60% балів. З урахуванням усього досі описаного, досить очевидним є такий, наприклад, розв'язок:

1. ініціалізувати `ans = 0`;
 2. перебрати зовнішнім циклом усі можливі кола радіусів $r, 2r, 3r, \dots$ (можна обчислити кількість кіл за формулою, виведеною в задачі А, чи написати тут `while`, який збільшуватиме радіус на r , доки не відбудеться торкання до чи вихід за хоча б одну зі стінок (ліва, нижня, права, верхня)), і для кожного такого кола
 - (а) перебрати внутрішнім циклом всі можливі дірки-круги, і для кожного такого круга
 - і. перевірити, чи потрапляє радіус кола, яке міг малювати Василько (визначається зовнішнім циклом) у той проміжок відстаней «від $d_i - r_i$ до $d_i + r_i$ », що відповідає поточній дірці-кругу (визначається внутрішнім циклом).
 - Якщо коло (визначається зовнішнім циклом) не потрапило ні в одну дірку-круг (визначається внутрішнім циклом), збільшити `ans` на 1.
3. вивести `ans`

Правильність такого алгоритму досить зрозуміла: Василько міг би намалювати деяке коло — перевіряємо, чи справді ніщо не заважає; щоб перевірити, порівнюємо з дірками-кругами.

Але біда в тому, що якщо розглянути обмеження нехай навіть не останнього, а передостаннього блоку ($A, B < 10^6$, $N \leq 2020$), то зрозуміло, що пройти цей блок шансів нема: при $A \approx B \approx 10^6$, $x \approx y \approx \frac{10^6}{2}$, $r = 1$ виходить, що кількість кіл, які міг би намалювати Василько, $\approx 5 \cdot 10^5$, і якщо кожне треба порівнювати з кожною з $N \approx 2000$ дірок-кругів — це $\approx 10^9$ досить громіздких порівнянь. (Якби нарахування балів було потестовим, а не блоковим, можна було б сподіватися набрати бали

хоча б за частину тестів з такими обмеженнями, якщо грамотно використати break для обривання внутрішнього циклу перебору дірок-кругів, як тільки буде знайдено перший пертин поточного кола; але з блоковим нарахуванням балів шанси на успішність таких оптимізацій істотно зменшуються, і для використаного набору тестів break, начебто, просто не допомагає.)

А як набрати більше балів? Як завжди: придумавши спосіб розглянути задачу під іншим кутом зору. Запровадимо «вісь радіусів», де радіусами є радіуси тих кіл, які міг би провести Василько. Можна сказати, що це радіуси, котрі використовуються у полярній системі координат (якщо початок координат полярної системи — у спільному центрі тих кіл, які намагається проводити Василько)... втім, таку «вісь радіусів» можна запровадити й не пов'язуючи її з полярною системою координат. Головне, що на цій «вісі» відкладаються відстані від центру (x, y) кіл, які намагається провести Василько. Причому, важливі *лише* відстані (а напрям не важливий); якщо пов'язувати це із полярною системою координат, то будемо розглядати самі лише радіуси, без кутів.

Кожне коло, яке намагається провести Василько, відображається на «вісі радіусів» єдиною точкою, єдина координата якої дорівнює радіусу цього кола. Отже, всі кола, які міг би намалювати Василько, якби стіни були такі самі, але не було дірок-кругів, перетворюються у точки $r, 2r, 3r, \dots, K \cdot r$ на «вісі радіусів» (тут і далі, K — кількість кіл, знайдена, як у задачі А).

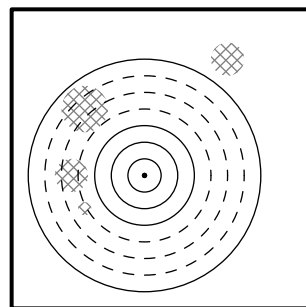
А наведені на самому початку розбору задачі міркування «дірка-круг № i забороняє Васильку проводити кола з радіусами у проміжку від $d_i - r_i$ до $d_i + r_i$, де $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$ » означають, що така дірка-круг перетворюється на тій самій «вісі радіусів» у відрізок від $d_i - r_i$ до $d_i + r_i$.

Скажімо, приклад з умови перетвориться так. Є дві дірки-круги. В однієї центр на відстані $d_i = \sqrt{20^2 + 15^2} = 25$ і радіус $r_i = 5$, тому вона створила на «вісі радіусів» відрізок від $25 - 5 = 20$ до $25 + 5 = 30$, у якому Василько не може проводити свої кола. В іншій центр на відстані $d_i = \sqrt{50^2 + 25^2} = 25\sqrt{5} \approx 55,902$ і радіус $r_i = 15$, тому вона створила на «вісі радіусів» відрізок від $25\sqrt{5} - 15 \approx 40,902$ до $25\sqrt{5} + 15 \approx 70,902$, у якому Василько не може проводити свої кола.

А відстані до стін та значення $r = 8$, на яке Василько планує щоразу змінювати радіус своїх кіл, задають точки 8, 16, 24, 32. Серед них, 8, 16, 32 враховуються в остаточну відповідь (кола з такими радіусами проводити можна, бо відповідні цим колам точки на «вісі радіусів» не потрапляють у жоден з відрізків на «вісі радіусів», відповідних діркам-кругам).

Звідси можна бачити, що задачу «скільки кіл може намалювати Василько?» можна переформулювати як «скільки з точок $r, 2r, 3r, \dots, K \cdot r$ не потрапляють у жоден з відрізків, відповідних діркам-кругам?» (щоправда, з поправкою: перевіряти, що точка (x, y) не потрапляє ні в яку дірку-круг, треба окремо); як вже сказано, K означає кількість кіл, знайдену, як у задачі А.

Хоч і гарантовано, що самі дірки-круги не можуть перетинатися, утворені з них відрізки на «вісі радіусів» цілком можуть: ми ж при переході від початкових прямокутних координат до «вісі радіусів» звертаємо увагу лише на відстані, ігноруючи напрям, а в різних напрямках на приблизно однакових відстанях цілком можуть бути одночасно багато дірок-кругів. На рисунку праворуч зображено ситуація, коли є відразу три дірки-круги, котрі не перетинаються як круги, але відповідні їм відрізки на «вісі радіусів» перетинаються. Очевидно, що їх може бути й ще більше. Тому, задачу «пошуку точок, що не потрапляють у жоден з відрізків, відповідних діркам-кругам» слід розв'язувати, враховуючи, що такі відрізки можуть хоч не мати спільних точок (вже на «вісі радіусів»), хоч перетинатися частково, хоч коротший може бути повністю вкладеним у довший — програма мусить правильно враховувати всі ці випадки. Тому зовсім простого алгоритму не виходить.



100%-й спосіб № 1. Спробуємо розв'язати задачу так. Перш за все, розберемося, чи справді центр усіх Василькових кіл (x, y) не потрапляє ні в одну з дірок; якщо потрапляє, виводимо відповідь "0"; якщо не потрапляє, перетворюємо кожен дірку-круг у відрізок на «вісі радіусів». (Що таке «вісь радіусів» та що таке K , див. вище.) Само собою на «вісі радіусів» є точки $r, 2r, 3r, \dots, K \cdot r$, відповідні місцям, де Василько міг би проводити кола. Оголосимо їх елементами масиву (від 1-го по

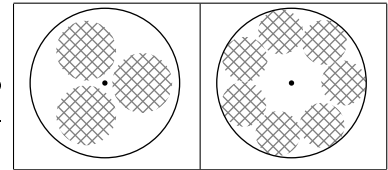
K -го, чи від 0-го по $(K-1)$ -й, це неважливо), і нехай ті елементи відповідають на питання «чи правда, що відповідна точка не потрапляє у жоден відрізок?», або, що те саме, «чи правда, що Василько може провести відповідне коло?». Спочатку, ініціалізуємо всі елементи цього масиву значеннями true. Потім будемо перебирати зовнішнім циклом відрізки на «вісі радіусів», відповідні діркам-кругам, і для кожного такого відрізка запускати внутрішній цикл, який перетворить з true на false усі елементи масиву, які потрапляють у відповідний відрізок. Наприкінці лишається тільки порахувати кількість елементів, які лишилися true. Або, якщо весь час використовувати цілий, а не логічний, тип (1 замість true і 0 замість false), то можна порахувати суму масиву.

Правильність цього алгоритму теж більш-менш очевидна (тим, хто зрозумів суть «вісі радіусів»). Але... все одно маємо вкладені цикли, один по кругам-діркам, інший по $r, 2r, 3r, \dots, K \cdot r$, тільки й того що переставлені місцями відносно очевидного підходу; чому раптом це повинно бути швидше? Хіба цей алгоритм має не ту саму оцінку $\Theta(N \cdot K)$, про яку раніше говорилося, що з нею можна набрати лише 60% балів?

Виявляється, не зовсім ту саму. По-перше, цей алгоритм треба правильно реалізувати у деталях, зокрема — не перебирати внутрішнім циклом усі $1, 2, 3, \dots, K$ чи $0, 1, 2, \dots, K-1$, відповідні $r, 2r, 3r, \dots, K \cdot r$; слід починати з $\approx \frac{d_i - r_i}{r}$ і закінчувати, коли досягається $\approx \min(K, \frac{d_i + r_i}{r})$, тобто переглядати лише ті елементи, яким справді треба надати false чи 0.

Може здатися, ніби це якась дивна оптимізація, бо «вона нічого не дасть, якщо для майже кожного з відрізків доведеться пробігти майже увесь діапазон масиву». Що ж, для таких відрізків така оптимізація справді нічого (чи майже нічого) не дала б. Але насправді **не може бути водночас багато довгих відрізків**. Бо відрізки в нас не любо-які, а утворені з дірок-кругів, причому дірки-круги не перетинаються.

Наприклад, праворуч зображено, що якби всі дірки-круги мали радіус $0,4 \cdot K \cdot r$ (відповідно, відрізки на «вісі радіусів» займали 80% проміжку від 0 до $K \cdot r$; смисл K див. вище) і мусіли повністю розміщуватися всередині останнього кола радіусом $K \cdot r$, то їх помістилось би щонайбільше 3 (три) штуки; аналогічно, для $0,3 \cdot K \cdot r$ щонайбільше 7 (сім) штук (додати ще одну посередині не можна, бо спільний центр всіх Василькових кіл потрапив би у дірку).



Само собою, за рахунок того, що дірки-круги можуть бути різних радіусів, а також можуть потрапляти у коло радіуса $K \cdot r$ лише частково (маючи помітну свою частину ззовні), замість наведених 3 чи 7 дірок-кругів їх може бути й більше. Але значно більше — лише за рахунок значного зменшення розміру більшості дірок-кругів, і загальний приблизний висновок «не може бути водночас багато довгих відрізків» від цього не порушується. А це означає, що асимптотична оцінка $\Theta(N \cdot K)$ сильно завищена, насправді програму можна реалізувати так, щоб вона працювала значно швидше. На жаль, виразити теоретично правильну асимптотичну оцінку досить складно. Але виявляється, що акуратна в деталях реалізація цього алгоритма проходить усі тести. (Хоча, якщо чесно, початковий намір автора задачі був, щоб такий алгоритм не проходив останній блок.)

А якщо координати будуть значно більші, як-то до 10^{18} ? (100%-й спосіб № 2.) Попередній алгоритм справді не дуже добрий тим, що час його роботи залежить від значень координат. Тому й пропонується розглянути також наступний алгоритм. Щоб це стало справді важливим, значення координат треба істотно збільшити... але тоді стане важливим такий чинник, як похибки обчислень (див. також стор. ??). Якщо при координатах до 10^7 майже всі математично-правильні обчислення, виконані у стандартному типі даних з рухомою комою (floating point), не матимуть особливих проблем через похибки, то при більших значеннях з цим усе значно гірше та складніше. Тому й вийшло, що задача дана з обмеженнями, при яких наступний алгоритм не має істотних переваг над попереднім, хоча, якби додатково збільшили значення координат, наступний алгоритм мав би значні переваги над попереднім і за швидкістю виконання, і за обсягом використаної пам'яті.

Розглянемо ту саму «вісь радіусів», але не будемо пов'язувати її з масивом. Натомість, розглянемо на ній більш-менш відому задачу «міра об'єднання відрізків на прямій», що розв'язується методом, який називають «вимітанням», «замітанням», «методом сканувальної (рос. «сканирующей») прямої»; це все різні назви одного й того ж методу. Він розглянутий у багатьох джерелах (одне

з таких джерел — ejudge.skipo.edu.ua, змагання №54), тому не пояснюватимемо його справді детально. Але короткий опис, поєднаний із конкретним прикладом, розглянемо. Нехай маємо вхідні дані, зображені на рис. зі стор. 125. Наведені там дірки-круги перетворюються у відрізки на «вісі радіусів» $\sqrt{22^2 + 0^2} \pm 5 = [17 \dots 27]$, $\sqrt{18^2 + 20^2} \pm 7 \approx [19,907 \dots 33,907]$, $\sqrt{18^2 + 10^2} \pm 2 \approx [18,591 \dots 22,591]$ та $\sqrt{25^2 + 35^2} \pm 5 \approx [38,012 \dots 48,012]$, зображені нижче. Якщо відсортувати всі координати початків та кінців таких відрізків (спільним масивом, але так, щоб після сортування було ясно, де початки й де кінці), можна йти зліва направо, перебираючи не значення окремих координат, а лише «точки подій», якими є: 0; відсортовані координати тих початків та кінців, які потрапили в проміжок від 0 до $K \cdot r$; координата $K \cdot r$. При цьому слід підтримувати певну змінну, яку часто називають «статус»; конкретно у цій задачі «статус» є цілим невід'ємним числом; він спочатку рівний 0, потім на кожному початку відрізка збільшується на 1, на кожному кінці відрізка зменшується на 1. Отже, точки не належать жодному відрізку там і лише там, де статус=0. Щоразу, коли статус змінюється з 0 на 1, потрібно порахувати (за формулою, без циклу), скільки було координат, кратних r , до цієї точки, після попередньої (причому, попередня обов'язково є або координатою 0 (початком «вісі радіусів»), або такою, де статус змінювався з 1 на 0).



3.21 II (районний/міський) етап 2021/22 н. р. (лише м. Черкаси)

Задачі доступні для дорішування (ejudge.skipo.edu.ua, змагання №74).

Задача А. «Знакозмінна сума»

Напишіть програму, що обчислюватиме знакозмінну суму $1 + 2 - 3 + 4 + 5 - 6 + 7 + \dots \pm n$ (доданки/від'ємники від 1 до n ; кратні 3 — від'ємні).

Вхідні дані. Єдине натуральне число n .

Результати. Виведіть єдине ціле число — значення виразу.

Оцінювання. Потестове (проходження кожного тесту оцінюється окремо, інші тести на це не впливають). Перші 10 тестів є тестами з умови й не оцінюються (але перевіряються, а детальний протокол показується).

60 балів припадає на тести, де n у межах $9 \leq n \leq 20$.

Ще 60 — на тести, де $100 \leq n \leq 3000$.

Ще 20 — на тести, де $10^5 \leq n \leq 10^6$.

Ще 60 — на тести, де $10^9 \leq n \leq 2 \cdot 10^9$.

Примітки. Перетворити вираз, щоб його обчислення стало швидшим, можна. Ви не повинні й не можете писати різні програми для різних обмежень (це стосується й усіх інших задач, де не сказано протилежне). Вам просто повідомляють, скільки за які тести передбачено балів.

Розбір задачі.

Якщо обчислити кілька перших трійок ($1 + 2 - 3 = 0$; $4 + 5 - 6 = 3$; $7 + 8 - 9 = 6$; $10 + 11 - 12 = 9$; ...), можна помітити, що результати таких трійок дорівнюють 0, 3, 6, 9, ..., утворюючи тим самим арифметичну прогресію.

Приклади:

Вхідні дані	Результати
1	1
2	3
3	0
4	4
5	9
6	3
7	10
8	18
2021	682425
20211212	68082198594168

Таким чином, якщо порахувати кількість трійок через вираз $k := n \text{ div } 3$ (Pascal), він же $k = n // 3$ (Python3), ... (в разі потреби допишіть своєю мовою програмування самостійно), то суму всіх трійок можна порахувати через формулу $\frac{0+3(k-1)}{2} \cdot k$.

Само собою, введене n може бути й не кратним 3, тобто можуть лишитися один чи два останні доданки, які не входять ні в яку трійку; їх можна додати до результату окремими if-ами.

А без цих прогресій можна? Якщо Вас влаштовує 70% балів, то можна просто порахувати в циклі суму всіх підряд натуральних чисел від 1 до n , але доданки, кратні 3, не додавати, а віднімати (включивши if всередину for-a).

Само собою, можна прийти до правильного результату і ще якимось. Можна, наприклад, не пам'ятаючи формулу суми арифметичної прогресії, вивести самостійно якийсь її аналог, враховуючи якісь особливості конкретно цієї задачі. Ніхто не перевіряє, чи знаєте Ви, що це називається арифметичною прогресією, та чи пам'ятаєте стандартну формулу. Зумієте замінити чимось іншим, що теж впадеться в обмеження часу — ну то й добре.

А згорнути зовсім в один вираз, щоб обчислювати і без циклів, і без if-ів, можна? В принципі, можна. Якби сума була не знакозмінною, а просто $1 + 2 + 3 + 4 + 5 + 6 + \dots + n$, її можна було б перетворити (наприклад, за все тією ж формулою суми арифметичної прогресії) до $\frac{n \cdot (n+1)}{2}$. Чим потрібна знакозмінна сума відрізняється від цієї? Тим, що доданки, кратні 3, віднімаються, а не додаються... але це саме можна сформулювати інакше: тим, що із $\frac{n \cdot (n+1)}{2}$ треба двічі відняти $3 + 6 + 9 + \dots + 3k$ (де k , як і кілька абзаців тому, має смисл $n \text{ div } 3$, а віднімати двічі треба тому, що якщо просто відняти, то вийшла б сума $1 + 2 + 4 + 5 + 7 + 8 + 10, + \dots$, тобто доданки, кратні 3, просто «зникли», а треба, щоб вони не «зникли», а стали від'ємними). Легко бачити, що $3 + 6 + 9 + \dots + 3k = 3 \cdot (1 + 2 + 3 + \dots + k) = 3 \cdot \frac{k \cdot (k+1)}{2}$, а раз цю суму треба відняти двічі, маємо

$$\text{остаточний результат} = \frac{n \cdot (n+1)}{2} - 3 \cdot k \cdot (k+1), \quad \text{де } k = n \text{ div } 3.$$

Також, для отримання повних балів важливо рахувати суму, щонайменше, в 64-бітовому цілочисельному типі, бо в менших (хоч 32-бітовому цілочисельному, хоч з рухомою комою (плаваючою точкою), як-то double) не вистачає розрядів (відбувається переповнення (див. також стор. 13) чи похибка (див. також стор. 14), що призводить до неправильної відповіді (наскільки сильно не вистачає і наскільки неповні бали, може залежати від конкретної мови програмування та конкретних типів). Рахувати у ще більших типах (наприклад, BigInteger мови Java) теж можна.

Задача В. «Гра "WALLCRSH"»

Гра "WALLCRSH" вимагає розбити усі броньовані плити, якими обкладена оборонна башта супротивника. Башта має вигляд правильного N -кутника, і для кожної з N сторін відома кількість броньованих плит, що її прикривають. Для руйнування плит можна використати лише спеціальну двоствольну гармату, котра за один постріл розбиває або дві плити на одній стороні башти, або по одній плиті на двох сусідніх сторонах. Стріляти по сторонах башти, де вже нема броньованих плит, дозволяється, але це вважається марнотратством.

Напишіть програму, котра буде знаходити, за яку найменшу кількість пострілів з такої гармати можна зруйнувати описану сукупність плит.

Вхідні дані. В першому рядку вводиться число N ($5 \leq N \leq 12345$), в другому, через пробіли — N цілих чисел (від 0 до 100 кожне) — кількості плит на відповідних сторонах, в порядку обходу многокутника; остання сторона сусідня з першою.

Приклад:

Вхідні дані	Результати
6 3 2 2 1 1 3	6

Результати. Слід вивести єдине число — мінімальну кількість пострілів.

Оцінювання. Потестове (проходження кожного тесту оцінюється окремо, інші тести на це не впливають). Перший тест є тестом з умови й не оцінюється (але перевіряється, а детальний протокол показується).

Розбір задачі.

На перший погляд, слід просто порахувати суму всіх елементів $a_1 + a_2 + \dots + a_N$, і поділити на 2 з заокругленням догори (наприклад, $\lceil \frac{1}{2} \rceil = \lceil \frac{2}{2} \rceil = 1$, $\lceil \frac{3}{2} \rceil = \lceil \frac{4}{2} \rceil = 2$, $\lceil \frac{5}{2} \rceil = \lceil \frac{6}{2} \rceil = 3$, ...). Це цілком «обґрунтовується» такими правдоподібними міркуваннями: почнемо стріляти з 1-ї сторони; поки на ній лишається ≥ 2 плит, стріляємо з обох стволів по цій стороні, доки не лишиться або 0, або 1; якщо лишиться 2, то перейдемо до пострілів з обох стволів по наступній стороні, а якщо лишиться 1, то зробимо один постріл, який збиває по одній плиті з поточної та наступної сторін, а потім продовжуємо залежно від кількості плит, що лишилися на цій наступній стороні, тощо. Й усе це начебто доводить, що цілком марнотратних пострілів (які не збивають жодної плити) не буде взагалі, а частково-марнотратним (який збиває лише одну плиту, а не дві) може виявитися хіба лише останній постріл, якщо так уже вийде, що на останній стіні лишилася тільки одна плита...

Однак, вищезгадане міркування не враховує, що за умовою кількість плит на стороні може бути й 0. Наприклад, при кількостях плит 0 1 0 1 0 1 0 0 доведеться витратити окремий постріл на кожну плиту. Втім, навіть якщо нулі у вхідних даних є, ті самі вищезгадані міркування все ж частково правильні, просто застосовувати їх треба не до всього масиву, а до кожного проміжку між нулями окремо. Наприклад, при кількостях плит 0 1 3 1 0 7 0 3 9 0 можна і варто порахувати окремо $\lceil \frac{1+3+1}{2} \rceil = 3$, окремо $\lceil \frac{7}{2} \rceil = 4$, окремо $\lceil \frac{3+9}{2} \rceil = 6$, і потім додати ці результати ($3 + 4 + 6 = 13$).

Крім того, «зацикленість» многокутника (остання сторона сусідня з першою) може давати (а може й не давати) можливість зекономити один постріл за рахунок об'єднання фрагментів на початку й наприкінці. Наприклад, для кількостей плит 3 0 5 1 0 3 нема потреби робити $\lceil \frac{3}{2} \rceil + \lceil \frac{5+1}{2} \rceil + \lceil \frac{3}{2} \rceil = 2 + 3 + 2 = 7$ пострілів, досить і $\lceil \frac{5+1}{2} \rceil + \lceil \frac{3+3}{2} \rceil = 3 + 3 = 6$.

Все, що слід *помітити*, вже перелічено. Лишається тільки реалізувати все це, правильно розділивши весь масив на фрагменти між нулями, причому враховуючи, що нулів може й не бути.

Насамкінець, у багатьох мовах програмування для «заокруглення догори» може бути корисною функція, що називається `ceil`, `Ceiling` чи ще якимось схоже. Можливий і спосіб, що не потребує такої функції: якщо вищезгадана сума вже поражена у змінній `s`, лишається обчислити $(s+1) \div 2$ (Pascal), $(s+1)//2$ (Python 3), $(s+1)/2$ (за умови цілочисельності `s`, C-подібні мови та Python 2).

Задача С. «Круглі дужки»

Напишіть програму, яка з'ясовуватиме, чи є дужковий вираз правильним. Тобто, чи правда, що для кожної дужки є парна, й у кожній парі відкривна дужка йде раніше, чим закривна.

Вхідні дані. 1-й рядок містить число N — кількість різних рядків, які треба перевірити на правильність. Кожен з подальших N рядків містить послідовність символів, яку треба перевірити на правильність. Послідовності гарантовано не містять ніяких інших символів, крім дужок '(' та/або ')'.

Результати. Виведіть рівно N рядків, у кожному єдиний символ 1 або 0 (1, якщо відповідний рядок утворює правильну дужкову послідовність; 0, якщо неправильну).

Примітки. Задача розділена в єджаджі на дві задачі C1 та C2, які мають однакові умови (зокрема, однакові формати вхідних даних та результатів), але різні обмеження. Для отримання повних балів, слід здати обидві. Ви самі вирішуєте, чи здавати в C1 та C2 однакові розв'язки, чи різні; єджадж вибиратиме окремо максимальні бали серед розв'язків C1, окремо максимальні серед C2, і додаватиме ці максимуми.

На задачу C1 припадають 180 балів, її оцінювання потестове (проходження кожного тесту оцінюється окремо, інші тести на це не впливають), 1-й тест є тестом з умови й не оцінюється

Приклад:

Вхідні дані	Результати
3	1
(())()	0
((())))	0
)) ((

(але перевіряється, а детальний протокол показується), в усіх тестах $2 \leq N \leq 7$. 100 балів припадають на тести, де довжини кожного окремо з рядків не перевищують 100, ще 40 балів — на тести, де ці довжини від 100 до 5000, ще 40 балів — на тести, де ці довжини від 10^5 до 10^6 .

На задачу C2 припадають 70 балів, для їх отримання необхідно, щоб Ваша програма пройшла всі тести (можна отримати лише або всі 70 балів, або 0). В задачі C2 гарантовано, що кожен з N рядків містить хоча б одну дужку, а розмір вхідних даних (кожного тесту окремо) не перевищує 20 мегабайтів.

Зверніть увагу, що в задачі C2, при більших, чим у C1, вхідних даних, жорсткіші обмеження на пам'ять (конкретні значення можна бачити в єджаджі).

Розбір задачі.

Основна рекомендована ідея на повні бали. Можна підтримувати лічильник, який в момент початку обробки рядка рівний 0, при кожній "(" збільшується на 1, а при кожній ")" зменшується на 1. Рядок правильний тоді й тільки тоді, коли виконуються обидві властивості: (1) наприкінці обробки всього рядка цей лічильник знову рівний 0; (2) за весь час обробки рядка лічильник ні разу не ставав строго від'ємним.

Щоб пройти частину C2, важливо не тримати в пам'яті одночасно весь рядок, а читати його символ за символом, і, обробивши поточний символ, тут же його забувати. Перевірено (ще до офіційного туру), що всіма мовами, оголошеними в пам'ятці як гарантовані (g++, fpc, python3, java) це написати можливо. Щоправда, для python3 для цього довелося додатково збільшити ліміт пам'яті та ліміт часу (й на турі перевірка відбувалася на вже збільшених; суть у тому, що не було спроб учасників читати по одному символу. . .); також, для python3 не вдалося написати код так, щоб один і той самий код проходив хоч C1, хоч C2 (це й було однією з причин розділення однієї задачі C на технічно різні частини C1 та C2 в єджаджі).

Альтернативний підхід, що дозволяє набрати значну частину балів. Досить природним є також також розв'язок «поки можливо, видаляти всі входження "("»; початковий рядок був правильним тоді й тільки тоді, коли такі видалення закінчуються порожнім рядком». Він повільніший за вищезгаданий, особливо на вхідних даних, де велика вкладеність дужок. Початковим наміром було, щоб цей розв'язок набирав десь 120–140 балів зі 180. Однак, через недогляд автора задачі, цей розв'язок міг набирати чи не набирати повний бал складової C1 (180 балів), залежно від того, наскільки ефективно реалізована заміна підрядка у відповідному бібліотечному методі, через що виникла ситуація, що коли мова програмування забезпечує більш ефективну заміну підрядків, то цей алгоритм проходить усі тести частини C1, а коли менш ефективно, то не всі. Оскільки це з'ясувалося лише після того, як дехто з учасників отримав повний бал за такий розв'язок, було прийнято рішення так і залишити, бо інакше це було б відбирання вже виставлених балів, що украй небажано.

А в яких ситуаціях може бути 170 балів зі 180? Якщо дуже уважно прочитати умову, можна побачити, що в C2 заборонено порожній рядок (жодного символа), а в C1 він не заборонений (отже, дозволений) і формально задовольняє вимогам правильності; тобто, треба вміти обробляти ситуацію, коли серед рядків є порожній (що є проблемою, зокрема, якщо писати мовою C++ і читати оператором ">>"), і виводити для порожніх рядків відповідь 1.

Задача D. «Паралелепіеди»

Напишіть програму, яка знаходитиме перелік прямокутних паралелепіедів, що мають об'єм V та площу поверхні S . Враховувати лише паралелепіеди, в яких всі три розміри виражаються натуральними числами.

Вхідні дані. Єдиний рядок містить розділені одинарним пробілом два натуральні числа V S .

Результати. Кожен рядок повинен містити розділені пробілами цілочисельні розміри a b c чергового паралелепіеда. Ці трійки обов'язково виводити у порядку зростання першого розміру a , при рівних a — у порядку зростання другого розміру b (а різних відповідей, в яких рівні і a , і b , не буває).

II (районний/міський) етап 2021/22 навч. року
м. Черкаси (лише), 12.12.2021

У випадку, якщо жодного паралелепіпеда з потрібними V , S не існує, виводьте єдиний рядок 0 0 0.

Примітки. Якщо прямокутний паралелепіпед має розміри a b c , то його об'єм дорівнює $a \cdot b \cdot c$, а площа поверхні $2 \cdot (a \cdot b + b \cdot c + c \cdot a)$, бо є дві грані (наприклад, передня й задня) площі $a \cdot b$, дві (наприклад, ліва та права) площі $b \cdot c$ та дві (наприклад, верхня й нижня) площі $c \cdot a$.

Оцінювання.

тести	обмеження	кіль-ть балів
1–4		0
5–14	$2 \leq V, S \leq 100$	60
15–24	$500 \leq V, S \leq 10^4$	60
25–29	$100 \leq V, S \leq 500$	30
30–34	$10^4 \leq V, S \leq 10^6$	30
35–39	$10^6 \leq V, S \leq 10^8$	30
40–44	$10^7 \leq S \leq 10^9 \leq V \leq 10^{11}$	30
45–53	$\begin{cases} 10^{10} \leq S \leq 10^{17}, \\ 10^{12} \leq V \leq 10^{18} \end{cases}$	60
54–65	такі ж, як у попередньому	50

спосіб нарахування балів

безпосередньо не оцінюються (з умови)

Потестове (можна отримувати бали за окремі тести, якщо інші не пройшли)

Поблокове (якщо не пройшов хоч один тест блоку, весь блок оцінюється на 0); блоки з тестів 25–29, 30–34, 35–39, 40–44 (усі, крім двох останніх) перевіряються й оцінюються незалежно від того, чи зараховані інші блоки та тести; два останні (тести 45–55 та 56–65) перевіряються й оцінюються лише якщо успішно пройдено всі попередні тести (1–44 чи 1–55) 4

Суть відмінності двох останніх блоків є секретною (до кінця туру).

Приклади:

Вхідні дані	Результати
7 7	0 0 0
6 22	1 2 3 1 3 2 2 1 3 2 3 1 3 1 2
1600 1120	4 20 20
40 8 5	40 8 5

Гарантовано, що у кожному з тестів, що використовуються для оцінювання цієї задачі, кількість трійок-відповідей строго менша 100.

Розбір задачі.

Найпростіший підхід (≈ 90 балів з 350). Можна просто перебрати всі можливі поєднання a , b , c (кожна змінна у проміжку від 1 до V) наведеними вкладеними циклами. У ньому мається на увазі, що V та S — введені у вхідних даних об'єм та площа поверхні, а V_2 та S_2 — обчислені при поточних значеннях a , b , c . Трійки перебиратимуться якраз у тому порядку, як їх треба виводити, тож такий розв'язок повинен набрати свої ≈ 90 балів. Але не більше (принаймні, не значно більше), бо три вкладені цикли, кожен від 1 до V , працюватимуть $\Theta(V^3)$ часу, і вже при $V \approx 1000$ виконання не вкладатиметься у ліміт часу.

```
for a:=1 to V do
  for b:=1 to V do
    for c:=1 to V do begin
      V2:=a*b*c;
      S2:=2*(a*b+a*c+b*c);
      if (V2=V)and(S2=S) then
        writeln(a,' ', b,' ', c)
    end
```

Навіть тут може бути важливим не «жмотитися» на типах даних. Значення добутку $a*b*c$ може значно перевищувати значення окремих змінних a , b , c . І намагання «зекономити», впихнувши їх у 16-бітовий тип, призведе до переповнень (див. також стор. 13) при обчисленні $a*b*c$. Наприклад, при $V=188$, $S=126$, якщо a , b , c перебирають всі значення від 1 до V , буде розглянуто зокрема і трійку $a=68$, $b=97$, $c=159$. Взагалі-то $68 \cdot 97 \cdot 159 = 1048764_{Dec} = 10000000000010111100_{Bin}$, але якщо рахувати це у 16-бітовому типі з переповненням, то від цього числа залишиться тільки $10111100_{Bin} = 188_{Dec}$. Аналогічно й $2 \cdot (68 \cdot 97 + 68 \cdot 159 + 97 \cdot 159) = 65662_{Dec} = 1000000000111110_{Bin}$ перетвориться у $1111110_{Bin} = 126_{Dec}$, і таким чином трійка $(68; 97; 159)$ та її перестановки будуть неправильно виведені як буцімто відповіді для $V=188$, $S=126$, хоча фактично ними не є. Вибирайте тип даних «із запасом», і цієї проблеми не буде. У більш ефективних алгоритмах, де значення змінних можуть бути ще більшими, важливість уникнення переповнень ще більша.

Переходимо від трьох вкладених циклів до двох (≈ 150 балів, тобто $\pm \approx 60$ балів відносно попереднього). Нам потрібно, щоб було $a \cdot b \cdot c = V$. (Крім того, ще й $2 \cdot (a \cdot b + b \cdot c + c \cdot a) = S$, але це на пару хвилин відкладемо, зосередившись спочатку на $a \cdot b \cdot c = V$.) Коли два (а не три) вкладені цикли вже вибрали поточні значення як a , так і b , враховуючи, що значення V ще раніше прочитане у вхідних даних — нема потреби перебирати можливі значення c . Якщо таке c існує, то воно дорівнює $\frac{V}{ab}$. Точніше кажучи, якби c могло бути дробовим, його завжди можна було б обчислювати як $\frac{V}{ab}$; але треба, щоб c було цілим. Тому варто міркувати так: перевірити, чи V кратне добутку $a \cdot b$; якщо ні, то для цих a , b підібрати c неможливо й треба далі перебирати наступні

a , b ; якщо ж кратне, то можна *знайти* c (не перебираючи!), як $c := V \operatorname{div}(a \cdot b)$ (Pascal), $c = V // (a \cdot b)$ (Python 3), ... (допишіть своєю мовою програмування самостійно). А тепер, для підібраних циклами a , b і обчисленого за формулою c , можна перевірити також умову $2 \cdot (a \cdot b + b \cdot c + c \cdot a) = S$ і прийняти остаточне рішення, чи виводити поточну трійку a , b , c . Виведення трійок все ще можна робити відразу ж, як відповідна трійка була знайдена. Складність такого алгоритму $\Theta(V^2)$, тож повинні проходити чи то всі, чи то хоча б більшість тестів, у яких $V \leq 10^4$.

Пропускаємо безперспективні ітерації зовнішнього циклу та уточнюємо межі вкладеного циклу (≈ 210 балів, тобто $+\approx 60$ балів відносно попереднього). Знаючи, що потім треба буде перевіряти, чи V кратне $a \cdot b$, варто звернути увагу, що для « V кратне $a \cdot b$ » необхідно, але не достатньо, щоб V було кратним a . Інакше кажучи, якщо V кратне a , то буде чи не буде V кратним добутку $a \cdot b$, залежить в тому числі й від b , треба перевіряти; але якщо V не кратне самому лише a , то тим паче не буде кратним добутку $a \cdot b$, і для таких a можна взагалі не запускати вкладений цикл перебору b . Це істотна оптимізація, бо у чисел насправді не буває дуже вже багато дільників; разом з тим, важко виразити конкретну асимптотичну оцінку того, скільки дільників може бути у числа.

Ще одна оптимізація (яку в принципі можна використати й незалежно від оптимізації минулого абзацу, але краще, якщо разом із нею) — коли вже є поточне значення a , то обчислити $V_{\operatorname{div} a} := V \operatorname{div} a$ (Pascal), воно ж $V_{\operatorname{div} a} = V // a$ (Python 3), ... (допишіть своєю мовою програмування самостійно), і вкладений цикл по b крутити не до V , а до $V_{\operatorname{div} a}$. Причому, крім того, що ця оптимізація пришвидшує виконання програми, вона має ще одну приємну рису: коли цикл по b крутиться лише до $V_{\operatorname{div} a}$, чи то зникає, чи то сильно зменшується (залежно від того, як написати решту коду) ризик переповнень. Якби була запроваджена лише ця оптимізація (без оптимізації попереднього абзацу), час роботи склав би $\Theta(V \log V)$. (Ідея доведення: $V + \frac{V}{2} + \frac{V}{3} + \dots + \frac{V}{V} = V \cdot (1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{V}) \approx V \cdot (\int_1^V \frac{1}{x} dx) = V \cdot (\ln x|_1^V) = V \cdot \ln V$. Автор розбору в курсі, що олімпіада не лише для 11-х класів, а молодші не зобов'язані знати інтеграли. Але це не важливо, бо інтеграл використаний лише в доведенні, а не в коді; здогадатися, що вищеописані зміни в коді повинні якось та скоротити перебір, цілком можна і не знаючи інтегралів.)

Разом узяті, оптимізації двох минулих абзаців покращують час роботи ще сильніше (але дуже важко виразити аналітично, наскільки саме). Якщо поєднувати ці оптимізації, то замість перевірки «чи V кратне $a \cdot b$ » варто використовувати перевірку «чи $V_{\operatorname{div} a}$ кратне b », вона повинна працювати дещо швидше; але це вже оптимізація у деталях, яка не така й важлива.

Наскільки відомо автору задачі, подальшим «вилізуванням» таких деталей можна добитися, щоб розв'язок набрав 240 балів з 350 (тільки не факт, чи будь-якою мовою програмування), але не більше.

100%-й розв'язок. Враховуючи симетричність формул $a \cdot b \cdot c$ та $2 \cdot (a \cdot b + b \cdot c + c \cdot a)$ відносно a , b , c , легко бачити, що якщо деяка трійка a , b , $c \in$ розв'язком, то й усі її перестановки теж є розв'язками. Тому, досить шукати лише трійки, в яких $a \leq b \leq c$, а решту формувати з них, переставляючи ці три числа. (Якщо значення a , b , c всі різні, таких перестановок 6, а саме: (a, b, c) , (a, c, b) , (b, a, c) , (b, c, a) , (c, a, b) , (c, b, a) ; якщо всі однакові, то лише 1 (як не переставляй однакові числа, виходить те саме); якщо два однакові, а третє інше, то 3 (зокрема, при $a \neq b = c$ це будуть (a, b, b) , (b, a, b) , (b, b, a) ; при $a = b \neq c$ — (a, a, c) , (a, c, a) , (c, a, a) ; при $a = c \neq b$ — (a, b, a) , (a, a, b) , (b, a, a)).

(Фраза «досить шукати лише» досить важлива, зокрема, тому, що грамотний аналіз такого роду спостережень часто є важливим для побудови ефективних алгоритмів. Бувають ситуації, коли варто сказати «воно-то досить цим обмежитися, але ж можна й не обмежуватися; навіщо нам брати на себе додаткові обмеження, якщо цього навіть не просять?». А бувають ситуації, коли варто сказати «і це добре, що можна обмежитися, бо це зменшує перелік варіантів, які слід розглянути; шукати за межами цих варіантів — воно-то можна, але навіщо, якщо на пошук за межами треба вирити купу часу, й це все одно не дасть нічого нового?». Важливо навчитися визначати, яка з цих реакцій доречніша для конкретної ситуації. Найчастіше, це залежить від таких факторів: (1) наскільки сильно

це обмеження дозволяє скоротити перебір? (2) наскільки складнішим чи простішим від урахування цього обмеження стає код?)

З разом узятих $a \leq b \leq c$ та $V = a \cdot b \cdot c$ випливає $a \leq \sqrt[3]{V}$; навіть найбільше обмеження на V становить $V \leq 10^{18}$, тож маємо $a \leq \sqrt[3]{V} \leq \sqrt[3]{10^{18}} = 10^6$; значить, навіть при найбільших обмеженнях, всіх можливих претендентів на значення a можна перебрати. Як вже згадано вище, можна зразу перевіряти кожне a , чи є воно дільником V , і якщо не є, то пропускати.

А при відомих V, S, a вже можна розв'язувати відносно b, c систему

$$\begin{cases} V = abc, \\ S = 2(ab + bc + ac). \end{cases}$$

Наприклад, виразимо з першого рівняння $c = \frac{V}{ab}$ і підставимо у друге; вийде $S = 2 \cdot (ab + b \frac{V}{ab} + a \frac{V}{ab}) = 2 \cdot (ab + \frac{V}{a} + \frac{V}{b})$. Нагадаємо, що V та S відомі, бо задані у вхідних даних, а a відоме, бо його перебирає цикл; таким чином, змінною лишається тільки b , і, враховуючи $b \neq 0$ (при 0 не виходило б ненульового $V = abc$), рівняння можна перетворити так, щоб воно стало квадратним відносно b :

взяли початок і кінець попереднього ланцюжка рівностей	$S = 2 \cdot (ab + \frac{V}{a} + \frac{V}{b})$
домножили на b й поділили на 2	$\frac{S}{2}b = ab^2 + \frac{V}{a} \cdot b + V$
перенісши все в один бік, привели до стандартного вигляду квадратного рівняння	$a \cdot b^2 + (\frac{V}{a} - \frac{S}{2}) \cdot b + V = 0$

Нагадаємо, що після того, як розв'яжемо це квадратне рівняння відносно b , можна виразити $c = \frac{V}{ab}$, а a вже відоме, бо його перебирає цикл, і можна сформувати всі можливі перестановки знайдених a, b, c та додати їх у відповідь. Приклад 4 з умови показує (більш того, це можна було б помітити, навіть якби такого прикладу в умові не було), що трійки, утворені перестановками одних і тих самих (a, b, c) , не завжди йдуть у відповіді підряд. Тому треба спочатку сформувати потрібні трійки у пам'яті, а потім, коли вони вже всі будуть сформовані, вивести їх у потрібному порядку. Тобто можна, наприклад, відсортувати масив (наприклад, двовимірний, переставляючи цілі рядки; або, наприклад, це може бути масив структур з полями (a, b, c)).

В умові сказано, що кількість трійок-відповідей гарантовано менша 100. У фактично використаних тестах навіть не перевищує 24, але автору задачі невідомо, звідки береться саме 24, і чи більше взагалі не може бути ні при яких V, S , чи такі приклади існують, але їх не вдалося знайти. Тому й вирішено повідомити учасникам неочевидний факт, який чи то важко, чи то неможливо самостійно отримати під час туру, і який в принципі може бути корисним: трійок-відповідей *досить мало*. Це, зокрема, означає, що не варто перейматися високою ефективністю сортування чи якогось його аналога. А ще це дозволяє використати не дуже ефективний, зате простий спосіб вирішення тієї проблеми, що є ситуації, коли трійка з усіма різними числами (a, b, c) має бути виведена як шість різних трійок $(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)$, є ситуації, коли $a = b = c$ і трійка має бути виведена лише раз, і є ситуації, коли два числа однакові й третє інше, і трійка має бути виведена як три різні трійки (причому, що саме треба переставляти, залежить від того, які два числа однакові). Замість аналізу кожного з цих випадків окремо (де легко заплутатися!) можна завжди класти у масив всі шість перестановок, а після сортування першу трійку (якщо вона існує, тобто відповідь усієї задачі не "0 0 0") виводити завжди, а кожну подальшу трійку — лише якщо вона не дорівнює попередній. І хоча можна заявляти, що впишування у масив (чи в якийсь його аналог) шести трійок там, де частина з них однакові, поганеньке у плані ефективності, але будь-яка адекватна реалізація цього (простішого для написання!) підходу вклаватиметься в ліміт часу з величезним запасом. Варто оптимізувати інші складові алгоритму, а не цю.

Хто знає множинні структури даних сучасних мов програмування (паскалівський `set` до них не належить; мова про контейнер `set` мови `C++`, колекцію `TreeSet` мови `Java` чи `SortedSet` мови `C#`,

тощо; тип `set` мови Python сюди підходить *частково*) — може спробувати використати замість масиву якусь із них. Деякі з них можуть виконати відразу і сортування, і виключення повторів. Залежно від мови програмування та деталей реалізації, це може бути доступним відразу (порядок «сортувати за першим елементом, а якщо вони однакові, то за другим» може вважатися природнім і бути готовим; зокрема, це так у C++ при використанні `set<vector<long long>>`, де кожен такий `vector` є 3-елементним і подає одну трійку), або може потребувати задання власного компаратора, або може виявитися настільки незручним, що легше використати вищезгаданий підхід із сортуванням масиву.

У квадратного рівняння зазвичай буває два корені; як зрозуміти, який з них брати до уваги? Після фіксації a перебором у циклі все ще лишилося справедливе твердження «якщо трійка (a, b, c) є розв'язком для потрібних V, S , то й трійка (a, c, b) теж». Тому, якби ми брали окремо один корінь b_1 , окремо інший b_2 , потім обчислювали на основі кожного з них $c_1 = \frac{V}{ab_1}$, $c_2 = \frac{V}{ab_2}$, то просто виходило б те саме в різних порядках ($b_1 = c_2$, $b_2 = c_1$). Тому здається природнім розглядати лише один з коренів квадратного рівняння. Будь-який. Інший все одно буде отриманий як $c = \frac{V}{ab}$. (Можна отримати підтвердження цього, співставивши смисл коефіцієнтів рівняння з теоремою Вієта.)

А в чому відмінність між двома останніми блоками? При розв'язуванні вищезгаданого квадратного рівняння, доводиться використати обчислення в типі `double` чи аналогічному; через це, можуть виникати похибки (див. також стор. 14), й ці похибки можуть бути досить великими, щоб, навіть коли теоретично-правильний-при-відсутності-похибок результат цілий, обчислення давало щось не те. Передостанній блок тестів максимально уникає таких ситуацій: тести підбиралися так, щоб прості більш-менш адекватні способи боротьби з цією проблемою (як-то заокруглення до *найближчого* цілого через `round` і подальша перевірка, чи справді знайдені при розв'язуванні квадратного рівняння b, c , разом з зафіксованим у циклі a , задовольняють умови $a \cdot b \cdot c = V$ та $2 \cdot (a \cdot b + b \cdot c + c \cdot a) = S$) працювали й давали правильний результат. Останній блок, навпаки, провокує такі ситуації; зокрема, спеціально підбиралися такі значення, де обчислення за очевидною формулою створює *великі* похибки (такі, що навіть заокруглення `round` дає не те ціле число, яке мало би бути, якби не було похибок), тощо.

Це може здатися жакливим (як взагалі щось зробити в ситуації, коли неправильно працюють стандартні засоби?!). Але все не так погано. Є чимало різних способів нівелювати проблему похибок.

Наприклад, так: оскільки все ще досить розглядати $b \leq c$ (звідки випливає $b^2 \leq bc = \frac{V}{a} \leq 10^{18}$, отже $b \leq 10^9$), можна бути більш-менш упевненим (якщо, звісно, мати уявлення про діапазон можливих похибок у стандартних типах даних), що навіть якщо в такому b є похибка, вона не перевищує кілька одиниць; тому, виконаємо перевірку, чи $\frac{V}{a}$ кратне b , і якщо так, то обчислення $c = \frac{V}{ab}$ та перевірку, чи $2 \cdot (a \cdot b + b \cdot c + c \cdot a) = S$, не лише для b' , знайденого за формулою меншого з коренів квадратного рівняння, а також і для недалеких (наприклад, для всіх b з проміжку $\max(a, b' - 10) \leq b \leq b' + 10$). Правда, в комплекті з таким прийомом доводиться також вжити ще один прийом: якщо обчислений дискримінант виходить від'ємним, але близьким до нуля — варто припустити, що то похибка й насправді він повинен бути рівно 0, і спробувати розв'язати рівняння з цією поправкою. Зайві розв'язки при цьому не потраплять у відповідь, бо все'дно треба робити перевірку, чи $V : (ab)$, обчислення $c = \frac{V}{ab}$ та перевірку, чи $2 \cdot (a \cdot b + b \cdot c + c \cdot a) = S$.

Або, наприклад, так (цей спосіб трохи складніший, зате в ньому все краще з математичним обґрунтуванням). Потрібно знайти розв'язок рівняння $a \cdot b^2 + \left(\frac{V}{a} - \frac{S}{2}\right) \cdot b + V = 0$. Розглянемо його як функцію $f(b) = a \cdot b^2 + \left(\frac{V}{a} - \frac{S}{2}\right) \cdot b + V$; це квадратний тричлен, парабола гілками догори, мінімум у точці $b^* = \frac{\frac{S}{2} - \frac{V}{a}}{2a}$. Корінь b_1 (такий, що $f(b_1) = 0$) існує й задовольняє нашому бажаному співвідношенню $a \leq b_1 \leq c_1 = \frac{V}{ab_1}$ тоді й тільки тоді $b^* \geq a$, $f(a) \geq 0$ та $f(b^*) \leq 0$; як відомо, на проміжку лівіше вершини парабола (гілками догори) монотонно спадає, тому можна застосувати бінарний пошук (почати з проміжку від a до b^* (`left:=a`; `right:=b^*`), потім багатократно брати середину проміжку $m := \frac{\text{left} + \text{right}}{2}$, і, залежно від того, чи $f(m) = 0$, чи $f(m) < 0$, чи $f(m) > 0$ приймати рішення, чи якраз вже знайшли корінь, чи можна відкинути всю праву половину (`right:=m`), чи можна відкинути всю ліву половину (`left:=m`); детальніше про бінпошук, зокрема й про застосування його до розв'язування рівнянь, можна знайти в багатьох різних джерелах у літературі чи Інтернеті). Звісно, сам по собі факт використання бінпошуку замість формули розв'язування квадратного рів-

няння ще не гарантує, що всі похибки щезнуть — бінпошук можна написати так, що похибки його перепаскудять і він даватиме неправильний результат. Але можна написати й так, що результат буде гарантовано правильним. Наприклад, постійно підтримувати `left`, `right` та `m` цілими (нам же все'дно потрібні цілі розв'язки), але при цьому спочатку обчислювати $f(m)$ у `double`, і якщо результат великий за модулем, то приймати рішення на основі цього, а якщо більш-менш близький до 0 (і тому нема ситуації, коли в цілочисельному типі стається переповнення й результат дуже далекий від правильного), то обчислити ще раз вже у 64-бітовому цілому типі й не мати похибок. Якщо мова програмування має вбудовану «довгу арифметику» (здатність працювати з точними, без похибок, цілими числами потенційно нескінченної довжини; в особливо виграшному становищі тут Python 3, бо там не просто є «довга арифметика», там перемикання на неї відбувається автоматично, без вказівок з боку програміста), це полегшує роботу, бо тоді досить працювати лише в «довгій арифметиці» замість двоступінчастого монстру з двох різних типів.

Є й кілька ще інших способів обійти проблему похибок; не всі з них мають математично строгі доведення, але ж учасники мають можливість спробувати здати кілька різних розв'язків, а доведень у них (вас) не питають. Тому, хоча б якось здати («пропхнути») на повні бали цілком можливо. А, щоб здати на 300 з 350, не треба взагалі нічого, крім циклів&розгалужень, сортувань (будь-якого рівня ефективності), ну і цілком стандартних знань шкільного курсу алгебри.

4 Список використаних джерел

1. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. — М.–СПБ–К., Вильямс, 2005. — 1296 с.
2. Левитин А. Алгоритмы: введение в разработку и анализ / А Левитин. — М.–СПБ–К., Вильямс, 2006. — 576 с.
3. Порублёв И. Н. Алгоритмы и программы. Решение олимпиадных задач / И. Н. Порублёв, А. Б. Ставровский — М.–СПБ–К., Диалектика, 2007. — 576 с.
4. Сайт ejudge.ckipo.edu.ua [Електронний ресурс]
5. Сайт uk.wikipedia.org [Електронний ресурс]
6. Сайт codeforces.com [Електронний ресурс]
7. Сайт habrahabr.ru [Електронний ресурс]
8. Сайт www.freepascal.org [Електронний ресурс]
9. Сайт en.cppreference.com [Електронний ресурс]
10. Сайт ideone.com [Електронний ресурс]