Завдання XXIX Всеукраїнської учнівської олімпіади з інформатики

2015/16 н. р.

Віталій Бондаренко, Данило Мисак, Шаміль Ягіяєв

Зміст

Завдання першого туру

1.1. Клуб брутальних людей (Андрій Мостовий)

Умова

Степан є учасником відомого клубу брутальних людей. В цьому клубі ведуть рейтинг його учасників, який змінюється лише під час щорічних змагань.

Змагання складаються з M послідовних раундів. В кожному раунді K найкращих учасників (згідно з рейтингом) сідають навколо стола з цибулею. Після того як її починають різати, перший, хто проронить сльозу, переміщується на останню позицію рейтингу та раунд закінчується. Зверніть увагу, що рейтинг у кожного учасника різний. Під час змагань ведеться протокол, в якому після кожного раунду записується позиція вибулого учасника, що була в нього на початку раунду (у кінці раунду він переміщується на останню позицію).

Степан знайшов протокол змагань цього року, але він забув свою позицію в рейтингу до початку змагань. Тим не менш він ще пам'ятає свою теперішню позицію в рейтингу. Допоможіть Степану згадати його позицію до початку змагань.

Завдання. Напишіть програму `club`, яка допоможе Степану визначити його позицію до початку змагань.

Вхідні дані. В першому рядку вхідного файла `club.dat` записано три цілих числа N, K, M — кількість учасників клубу, кількість учасників в одному раунді та кількість раундів відповідно. В другому рядку міститься M цілих чисел: i-те число A_i ($1 \leqslant A_i \leqslant K$) задає позицію вибулого учасника у відповідному раунді. В третьому рядку записано єдине ціле число P ($1 \leqslant P \leqslant N$) — позицію Степана після змагань.

Вихідні дані. В єдиному рядку файла `club.sol` виведіть єдине число — позицію Степана до початку змагань.

Оцінювання. Набір тестів складається з 4 блоків, для яких додатково виконуються такі умови:

- 30 % балів: $1 \leqslant K < N \leqslant 1000$, $1 \leqslant M \leqslant 10^4$.
- 20 % балів: $1 \leqslant K \leqslant 100$, $K < N \leqslant 10^6$, $1 \leqslant M \leqslant 10^4$.
- 20 % балів: $1 \leqslant K \leqslant 100$, $K < N \leqslant 10^9$, $1 \leqslant M \leqslant 10^5$.
- 30 % балів: $1 \leqslant K < N \leqslant 10^9$, $1 \leqslant M \leqslant 10^5$.

Приклад вхідних та вихідних даних.

club.dat	club.sol
6 2 3 1 2 1 5	3

Пояснення. На початку змагань Степан займав третю позицію. Після першого раунду учасник з першої сходинки опустився на останню, через це Степан піднявся на одну позицію вгору, а саме на другу позицію. Після другого раунду Степан опустився на останню позицію. Після третього раунду він піднявся на одну позицію вгору, а саме на п'яту позицію.

Розв’язання

Будемо розглядати перебіг раундів задом наперед та щоразу, знаючи місце Степана в рейтингу після відповідного раунду, визначатимемо його місце в рейтингу перед даним раундом. Це можна зробити однозначно:

- Якщо переміщення в рейтингу стосувалося когось, хто мав рейтинг, нижчий від Степана, то старе місце в рейтингу Степана не відрізняється від нового.
- Якщо переміщення в рейтингу стосувалося когось, хто мав рейтинг, вищий від Степана, то старе місце в рейтингу Степана на 1 більше (тобто нижче) від нового.
- Якщо переміщення в кінець рейтингу стосувалося самого Степана, його старе місце в рейтингу — позиція переміщеного учасника.

Через M ітерацій отримаємо початкове місце Степана в рейтингу.

1.2. Watcha (Данило Мисак)

Умова

У віддаленому майбутньому, коли на Олімпії кіборги повністю замінять людей, центральна проблема, що постане перед суспільством, буде такою: тим кіборгам, які з естетичних причин надаватимуть перевагу механічному годиннику, буде важко, подивившись на годинник, розуміти, котра зараз година.

Завдання. Вирішіть цю екзистенціальну проблему вже зараз! Напишіть програму `watcha`, яка за виглядом механічного годинника визначатиме, котра зараз година.

Вхідні дані. Вхідний файл `watcha.dat` задає одне або кілька чорно-білих зображень годинника. Файл складається з одного або кількох блоків (щонайбільше 20 блоків у файлі). Один блок — 128 рядків по 128 символів у кожному. Кожен символ блоку — це або `0` (білий піксель: фон), або `1` (чорний піксель: елемент годинника). Між символами пропусків немає. Між собою блоки розділені порожніми рядками (по одному порожньому рядку між кожними двома сусідніми блоками). Вхідні дані завжди задають годинник одного типу, зображений нижче. Крім того, годинник завжди має однаковий розмір: перший та останній рядок і стовпець задають рамку з білих пікселів, але вже другий та передостанній рядок і стовпець містять принаймні по одному чорному пікселю годинника.

Вихідні дані. Вихідний файл `watcha.sol` повинен містити стільки рядків, скільки блоків є у вхідному файлі. Кожен рядок задає час, відображений на годиннику у відповідному блоці (година — від 1 до 12, без нулів попереду; символ, що розділяє години та хвилини, — двокрапка).

Оцінювання. Набір складається із 50 тестів однакової вартості.

Приклад вхідних та вихідних даних.

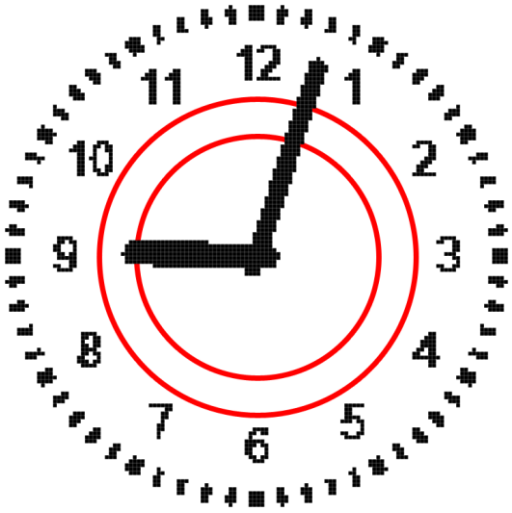
`watcha.dat`	`watcha.sol`

	`9:03`
	`3:50`
	`6:36`

Розв’язання

Задачу, безумовно, можна розв’язувати по-різному (можливо навіть, реалізацією штучного інтелекту, ідентичного людському). Однак один із найпростіших шляхів є, ймовірно, таким:

1. На такому радіусі від центра годинника, де ще є хвилинна стрілка, але де вже немає годинної та ще не почалися числа на циферблаті (приблизно 40 пікселів), скануємо коло на предмет хвилинної стрілки. Це можна робити, наприклад, з точністю до 1 градуса. Хвилину, на яку вказує хвилинна стрілка, можна оцінити як середнє арифметичне кутів, на яких піксель був чорним (при цьому необхідно окремо врахувати циклічність у 360 градусів).



2. На радіусі, трохи меншому за довжину годинної стрілки (приблизно 30 пікселів), скануємо коло на предмет годинної стрілки. При цьому ми вже знаємо, де розташована хвилинна стрілка, і пікселі, що належать їй (або принаймні лежать достатньо близько до неї), ігноруємо. Якщо при скануванні знайшлися чорні пікселі, які точно не належать хвилинній стрілці, розташування годинної можемо оцінити (приблизно) як середнє арифметичне відповідних кутів. Інакше вважаємо, що розташування годинної стрілки збігається (приблизно) з розташуванням хвилинної.
3. Якщо переведення кута хвилинної стрілки в кількість хвилин є лінійною, то для годинної спершу маємо відняти кут, що відповідає кількості хвилин, яка пройшла з останньої цілої години. Після цієї трансформації ми без проблем встановлюємо кількість годин, адже неточність, з якою встановлено кут у пункті 2, не перевищує 15 градусів: а саме такої точності вистачить для однозначного визначення відповіді.

1.3. Підчисло (Андрій Селіванов)

Умова

Петрик та Василько — справжні друзі, тому вони постійно задають один одному всілякі цікаві задачі. Проте Василько завжди з легкістю розв'язує задачі свого друга, тож Петрик вирішив придумати по-справжньому складну задачу. І ось що в нього вийшло.

Будемо називати число B *підчислом* числа A, якщо з числа A можна викреслити деякі цифри так, що цифри, які залишилися, утворюють число B.

Задано N-цифрове число X. Позначимо як X_K найбільше K-цифрове підчисло числа X. Необхідно відповісти на M запитів. Кожен запит складається з двох чисел — K і L. Відповіддю на запит є L-та цифра числа X_K.

Цього разу задача справді змусила Василька задуматись. А чи зможете ви розв'язати її швидше за нього?

Завдання. Напишіть програму `subnumber`, яка за заданим числом та послідовністю запитів знайде необхідні цифри.

Вхідні дані. У першому рядку вхідного файла `subnumber.dat` міститься ціле число X довжини N ($1 \leqslant N \leqslant 10^5$). У другому рядку міститься число M ($1 \leqslant M \leqslant 10^5$). У наступних M рядках міститься по два числа K_i, L_i ($1 \leqslant K_i \leqslant N$, $1 \leqslant L_i \leqslant K_i$) — параметри i-го запиту.

Вихідні дані. Вихідний файл `subnumber.sol` повинен містити один рядок довжини M, в якому i-й символ є відповіддю на i-й запит.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 15 % балів: $N = 20$, $M = 10^4$.
- 25 % балів: $NM \leqslant 5 \times 10^5$.
- 60 % балів: $N \leqslant 10^5$, $M \leqslant 5 \times 10^4$.

Приклад вхідних та вихідних даних.

subnumber.dat	subnumber.sol
31415926 7 2 2 3 1 1 1 4 3 5 2 8 2 7 3	6992511

Розв'язання

З'ясуємо, як шукати відповідь на один запит. Задано K та L; необхідно знайти L-ту цифру X_K. Позначимо через $F(left, right, K)$ найбільше K-цифрове підчисло відрізка $[left, right]$ числа X (при цьому одна чи кілька початкових цифр числа можуть бути нулями). Позначимо також через $G(left, right, K, L)$ L-ту цифру числа $F(left, right, K)$. Тоді відповідь на запит — це $G(1, N, K, L)$.

Спочатку побудуємо для кожної цифри масив, що містить позиції в X, де ця цифра трапляється.

Будемо знаходити $G(left, right, K, L)$ таким чином. На відрізку $[left, right]$ знайдемо *найбільшу* цифру D: оскільки цифр усього 10, можна за $O(\log N)$ часу для кожної з них знайти бінарним пошуком по масиву її позицій, скільки разів вона трапляється на відрізку $[left, right]$.

Позначимо кількість разів, яку цифра D трапляється на цьому відрізку, через c, а позиції, на яких стоїть D, через $p_1, p_2, \dots, p_c$. Якщо $c \geqslant K$, то $F(left, right, K)$ складається лише з цифр D, і відповіддю на запит є D. Якщо $c < K$, то всі c цифр D входять у $F(left, right, K)$ (це можна довести від супротивного). Тоді розглянемо такі числа $t_1, t_2, \dots, t_c$, що $G(left, right, K, t_j) = D$. Можливі чотири випадки:

- Якщо $t_1 > L$, то $G(left, right, K, L) = G(left, p_1 - 1, t_1 - 1, L)$.
- Якщо $t_c < L$, то $G(left, right, K, L) = G(p_c + 1, right, K - t_c, L - t_c)$.
- Якщо $t_j = L$ для деякого j, то $G(left, right, K, L) = D$.
- Якщо $t_{j-1} < L$ та $t_j > L$ для деякого j, то
$$G(left, right, K, L) = G(p_{j-1} + 1, p_j - 1, t_j - t_{j-1} - 1, L - t_{j-1}).$$

Зауважимо, що після переходу до меншої задачі на новому відрізку нема жодного числа D. Таким чином, пошук $G(left, right, K, L)$ можна звести до обчислення $G(left', right', K', L')$, причому на відрізку $[left', right']$ найбільша цифра є *меншою* за найбільшу цифру на відрізку $[left, right]$. Оскільки найбільша цифра лежить у межах від 0 до 9, для обчислення $G(left, right, K, L)$ потрібно зробити не більше ніж 9 зведень до меншої підзадачі.

З'ясуємо, як знаходити позиції $t_1, t_2, \dots, t_c$. Оскільки це позиції, на яких стоять максимальні цифри відрізка $[left, right]$, їх необхідно розташувати якомога лівіше в $F(left, right, K)$. Тому t_j визначається зі співвідношень $t_j \geqslant j$ і $right - p_j \geqslant K - t_j$ (щоб залишилось достатньо цифр на залишок числа $F(left, right, K)$), причому нас цікавить якомога менше значення. Таким чином, $t_j = max(j, K + p_j - right)$. Отже, визначати, який з чотирьох випадків, розглянутих вище, має місце, можна за $O(\log N)$ часу за допомогою бінарного пошуку.

Маємо, що обчислення, необхідні для опрацювання одного запиту, вимагають $O(9 \log N) = O(\log N)$ часу. На M запитів можна відповісти за $O(MlogN)$.

1.4. Пограбування (Роман Рубаненко)

Умова

В країні Олімпії дуже розвинена банківська система, тому жителі залюбки кладуть свої заощадження в банк. Усього є N банків, які стоять в ряд. У банку номер i зберігається a_i тугриків. Початково в банках немає системи безпеки, котра могла б завадити пограбуванню. Однак відомо, що якщо ввечері дня номер d пограбували банк номер b, то на ранок наступного дня необхідну систему безпеки буде встановлено у сусідніх банках ($b-1$ та $b+1$), після чого їх вже неможливо буде пограбувати. На ранок дня номер $d+i$ ($i>0$) систему безпеки буде встановлено у банках $b-i$ та $b+i$. Це буде продовжуватись, поки всі банки не буде захищено від пограбування. Пограбований банк вже ніколи не можна пограбувати.

На жаль, в Олімпії навіть злочинці займаються розв'язуванням олімпіадних задач, тому уряд не сумнівається в тому, що якщо хтось замислить провести серію пограбувань, то він зробить це оптимальним чином, інакше кажучи — максимізує сумарну кількість тугриків у тих банках, які йому вдасться пограбувати до того, як в них буде встановлено систему безпеки. Крім того, відомо, що злочинці грабують не більше ніж один банк за день, а також те, що вони орудують лише ввечері.

Банківська система, аналізуючи можливі збитки від пограбувань, розглядає послідовно $M+1$ варіант розміщення коштів у банках. Кожен варіант відрізняється від попереднього кількістю грошей в одному із банків.

Завдання. Напишіть програму `robbery`, що для кожного з варіантів розміщення коштів у банках підрахує максимальні можливі втрати від пограбувань.

Вхідні дані. У першому рядку вхідного файла `robbery.dat` містяться числа N і M — кількість банків та операцій зміни кількості тугриків. У наступному рядку записано N чисел a_i, що задають початкову кількість тугриків у банках. Далі йдуть M рядків, у кожному з яких записано пару чисел B та T, що означає, що після чергової операції в банку номер B стане T тугриків.

Вихідні дані. У вихідний файл `robbery.sol` для кожного i ($0 \leqslant i \leqslant M$) в окремий рядок виведіть максимальну кількість тугриків, яку зможуть вкрасти злочинці після виконання i операцій.

Оцінювання. Набір тестів складається з 3 блоків, для яких додатково виконуються такі умови:

- 10 % балів: $1 \leqslant N \leqslant 8$, $1 \leqslant M \leqslant 8$.
- 20 % балів: $1 \leqslant N \leqslant 1000$, $1 \leqslant M \leqslant 1000$.
- 70 % балів: $1 \leqslant N \leqslant 10^5$, $1 \leqslant M \leqslant 10^5$.

Приклад вхідних та вихідних даних.

robbery.dat	robbery.sol
7 4 6 7 5 6 2 2 4 6 5 7 2 7 6 4 6	17 18 18 19 19

Пояснення. У схемі, поданій далі, `0` позначає пограбований банк, а `-1` — банк, в якому вже встановили систему безпеки.

Початковому варіанту розміщення відповідатимуть такі дії грабіжників:

`6 7 5 6 2 2 4`	— початковий стан;
`6 7 5 0 2 2 4`	— пограбували четвертий банк (6 тугриків);
`6 7 -1 0 -1 2 4`	— ранком наступного дня встановили систему безпеки в сусідні банки;
`6 0 -1 0 -1 2 4`	— пограбували другий банк (7 тугриків);
`-1 0 -1 0 -1 -1 4`	— систему безпеки встановлено в перший банк (через пограбування другого) та в шостий (через пограбування четвертого 2 дні тому);
`-1 0 -1 0 -1 -1 0`	— грабують останній банк (4 тугрики).

В сумі грабіжники мають 17 тугриків.

Останній варіант розміщення буде таким: `6 7 5 6 2 5 6`. Всього грабіжники можуть отримати $6 + 7 + 6 = 19$ тугриків, якщо будуть грабувати четвертий, другий, а потім сьомий банки.

Розв'язання

Спочатку розв'яжемо задачу для одного запиту. Розглянемо множину індексів банків, які потрібно грабувати в оптимальному випадку: $ind[1] < ind[2] < ind[3] < \ldots < ind[r]$. Повинна справджуватись умова $ind[i+1] \neq ind[i] + 1$, оскільки в цьому випадку, який би з двох банків $ind[i]$ та $ind[i+1]$ не пограбували першим, на наступний день інший уже буде під охороною. Виявляється, що наведена умова є не лише необхідною, але й достатньою, оскільки при послідовному грабуванні банків зліва направо сигнал про небезпеку, умовно кажучи, не буде встигати за грабіжниками. Тепер маємо доволі стандартну задачу, де треба набрати найбільшу можливу суму з елементів масиву, не використовуючи жодних двох елементів, що йдуть поспіль.

Для розв'язання цієї задачі можна скористатися динамічним програмуванням. Нехай $f[k]$ — відповідь для префікса масиву $a[]$ довжиною k елементів (тобто найбільша можлива сума за додаткової умови, що доданки можна брати лише серед перших k елементів масиву). Тоді, якщо домовитись, що $f[0] = f[-1] = 0$, справедливою є рівність (пояснення див. нижче):

$$f[k] = max\{f[k-1]; f[k-2] + a[k]\}, k \geqslant 1.$$

Маючи відповідь для всіх менших за k чисел, щоб знайти відповідь для k, ми розглядаємо два випадки:

- $a[k]$ входить у відповідь: тоді $a[k-1]$ брати не можна, але можна брати будь-яку комбінацію з перших $k-2$ елементів. Нас цікавить оптимальна, яку вже пораховано: вона дорівнює $f[k-2]$.
- $\mathrm{a}[k]$ не входить у відповідь: тоді достатньо вибрати оптимальну комбінацію з перших $k-1$ елементів — тобто число $f[k-1]$.

Усі підрахунки можна здійснити в простому циклі й отримати значення $f[N]$ (для одного запиту) за $O(N)$ операцій.

На жаль, зміна навіть одного елемента в $a[]$ може змінити $O(N)$ значень $f[]$, тому наведений алгоритм не дає повного розв'язання задачі. Тим не менше ідею можна розширити.

Нехай маємо два масиви A та B, для кожного з яких пораховано чотири значення: оптимальну відповідь, якщо або брати, або не брати або перший, або останній елемент. Виявляється, з цих даних ми можемо дізнатися такі самі чотири значення для масиву $C = A + B$, де «+» позначає конкатенацію (об'єднання) масивів. Щоб обрахувати ці значення, треба зробити перебір, комбінуючи, чи беремо ми останній елемент масиву A, чи беремо перший елемент масиву B та, якщо цей вибір не суперечить умові про сусідні індекси, перебрати варіанти на інших кінцях цих двох масивів, які є кінцями результуючого масиву C. Отже, за $O(1)$ операцій можна підрахувати відповідь для об'єднання двох масивів. Задачу ж можна розв'язати, побудувавши на вхідному масиві дерево відрізків, яке в кожній вершині зберігає відповідні чотири значення, а в корені, таким чином, містить інформацію, що відповідає цілому масиву. Коли в одному з банків змінюється кількість тугриків, нам досить оновити лише $O(\log N)$ вершин дерева відрізків, які покривають цей банк. Розв'язок має складність $O(M \log N)$.

Завдання другого туру

2.1. Робот (Сергій Оришич)

Умова

В країні Олімпії сьогодні свято!

Єдине поштове відділення на вулиці Олімпійській переповнене подарунками, які необхідно якомога швидше доставити їхнім адресатам. Всього цього дня надійшло N подарунків, i-й з яких необхідно доставити в будинок з номером h_i. Декілька різних подарунків можуть бути адресовані в один будинок.

Поштове відділення розташоване на початку вулиці Олімпійської. Праворуч від пошти вздовж прямої дороги знаходяться будинки, нумерація яких починається з 1; i-й будинок розташований на відстані i кілометрів від пошти. Для доставки предметів в експериментальному режимі використовується єдиний спеціальний поштовий робот, який одночасно вміщує не більше ніж K предметів. Завантаження та відвантаження одного подарунка триває *одну* хвилину, два подарунки не можуть завантажуватися чи відвантажуватися одночасно. Один кілометр робот долає теж за *одну* хвилину.

Завдання. Напишіть програму `robot`, яка за інформацією про кожен з подарунків та характеристикою робота визначатиме найменший час у хвилинах, за який робот зможе доставити всі подарунки їхнім адресатам та повернутися у поштове відділення. Спочатку всі подарунки та робот знаходяться у відділенні.

Вхідні дані. Перший рядок вхідного файла `robot.dat` містить два цілих числа N і K ($1 \leqslant N \leqslant 10^5$, $1 \leqslant K \leqslant 1000$) — кількість подарунків, що надійшли до поштового відділення, та найбільшу кількість предметів, що може одночасно вмістити робот. Другий рядок містить N цілих чисел, записаних через пропуск: i-те число рівне h_i ($1 \leqslant h_i \leqslant 10^6$) — номер будинку, в який необхідно доставити i-й подарунок. Гарантується, що будинки із вказаними номерами справді розташовані на вулиці Олімпійській.

Вихідні дані. Єдиний рядок вихідного файла `robot.sol` повинен містити одне ціле число — найменший час у хвилинах, за який робот зможе доставити подарунки їхнім адресатам та повернутися у поштове відділення.

Оцінювання. Набір тестів складається з 4 блоків, для кожного з яких виконуються такі умови:

- 25 % балів: $1 \leqslant N \leqslant 10$, $1 \leqslant K \leqslant 5$, $1 \leqslant h_i \leqslant 100$.
- 25 % балів: $1 \leqslant N \leqslant 10^3$, $1 \leqslant K \leqslant 100$, $1 \leqslant h_i \leqslant 10^4$.
- 25 % балів: $1 \leqslant N \leqslant 10^5$, $1 \leqslant K \leqslant 1000$, $1 \leqslant h_i \leqslant 10^4$.
- 25 % балів: $1 \leqslant N \leqslant 10^5$, $1 \leqslant K \leqslant 1000$, $1 \leqslant h_i \leqslant 10^6$.

Приклад вхідних та вихідних даних.

robot.dat	robot.sol
7 3 3 9 3 2 1 1 3	40

Пояснення. Пронумеруємо подарунки від 1 до 7. Тоді один з оптимальних планів доставки має такий вигляд:

- 2 хвилини: завантаження 5 і 6 подарунків (обидва потрібно доставити у будинок 1),
- 1 хвилина: переміщення робота від відділення до будинку 1,
- 2 хвилини: відвантаження 5 і 6 подарунків,
- 1 хвилина: переміщення робота від будинку 1 до відділення,
- 2 хвилини: завантаження 2 і 7 подарунків (2 подарунок треба доставити в будинок 9, 7 подарунок — в будинок 3),
- 9 хвилин: переміщення робота від відділення до будинку 9,
- 1 хвилина: відвантаження 2 подарунку,
- 6 хвилин: переміщення робота від будинку 9 до будинку 3,
- 1 хвилина: відвантаження 7 подарунку,
- 3 хвилини: переміщення робота від будинку 3 до відділення,
- 3 хвилини: завантаження 1, 3 та 4 подарунків (1 і 3 подарунки треба доставити до будинку 3, 4 подарунок — до будинку 2),
- 2 хвилини: переміщення робота до будинку 2,
- 1 хвилина: відвантаження 4 подарунку,
- 1 хвилина: переміщення робота від будинку 2 до будинку 3,
- 2 хвилини: відвантаження 1 і 3 подарунків,
- 3 хвилини: переміщення робота від будинку 3 до відділення.

Розв'язання

Відсортуємо числа h_i за спаданням (незростанням) і вважатимемо, що ці числа розташовано саме в такому порядку, причому при $i > N$ покладемо $h_i = 0$. Тоді однією з оптимальних схем розвезення подарунків буде така: завантажити й розвезти подарунки в будинки $h_1, h_2, \dots, h_K$, повернутися, завантажити й розвести подарунки в будинки $h_{K+1}, h_{K+2}, \dots, h_{2K}$, знов повернутися й розвести $h_{2K+1}, h_{2K+2}, \dots, h_{3K}$ і т. д. (в останній групі може бути менше ніж K подарунків). На увесь процес нам знадобиться $2(h_1 + h_{K+1} + h_{2K+1}+\dots)$ хвилин на розвезення (враховуючи час, необхідний, щоб повернутися), а також $2N$ хвилин на завантаження та вивантаження подарунків. Отже, витративши $O(N \log N)$ часу, відповідь ми знайшли. Залишається довести, що така схема дійсно буде оптимальною.

Через $\lceil x \rceil$ позначимо найменше ціле число, не менше за x. Для довільного $i < N$ розглянемо ділянку шляху між будинками h_{i+1} та h_i. Зауважимо, що у випадку $h_{i+1} = h_i$ така ділянка буде нульової довжини, але це не завадить загальності міркувань. Оскільки в початковий момент часу всі подарунки, що необхідно доставити в будинки $h_1, h_2, \dots, h_i$, перебували у поштовому відділенні — зліва від ділянки $[h_{i+1}, h_i]$, а в кінцевий момент часу мають бути доставлені своїм адресатам праворуч від цієї ділянки (або на її правій межі), робот має всі такі i подарунків перемістити через цю ділянку в напрямку зліва направо. А оскільки за один раз робот може переміщувати щонайбільше K подарунків, йому доведеться здійснити не менше ніж $\lceil i/K \rceil$ переміщень зліва направо через цю ділянку — і стільки ж назад. Тоді обмеження знизу на сумарний час переміщення якраз і складе

$$2 \times \lceil 1/K \rceil \times (h_1 - h_2) + 2 \times \lceil 2/K \rceil \times (h_2 - h_3) + 2 \times \lceil 3/K \rceil \times (h_3 - h_4)+\dots =$$

$$= 2\big(1 \times \big((h_1 - h_2)+\dots+(h_K - h_{K+1})\big) + 2 \times \big((h_{K+1} - h_{K+2})+\dots+(h_{2K} - h_{2K+1})\big) + 3 \times \dots\big) =$$

$$= 2(1 \times (h_1 - h_{K+1}) + 2 \times (h_{K+1} - h_{2K+1}) + 3 \times (h_{2K+1} - h_{3K+1})+\dots) =$$

$$= 2(h_1 + h_{K+1} + h_{2K+1}+\dots).$$

Додамо: щоб зробити це доведення зовсім строгим, необхідно йти від супротивного і, зафіксувавши деякий маршрут з меншим сумарним часом, розбити ділянки $[h_{i+1}, h_i]$ на підділянки такі, що посередині цих підділянок робот ніколи не зупиняється. Тоді з подібних міркувань одержимо ту саму оцінку знизу, а отже — суперечність.

2.2. Метро (Віталій Герасимів)

Умова

Мер столиці Котоляндії врешті погодився побудувати в місті мережу метро. Тепер його очікує надзвичайно важливе завдання — зекономити якомога більше грошей на побудові цього метро.

Мережа метро буде складатись із певної кількості станцій, сполучених двосторонніми тунелями. Між будь-якою парою станцій повинен бути шлях по тунелях (можливо, через інші станції), при цьому кількість побудованих тунелів має бути мінімальною можливою. Зауважте, що за таких умов існує єдиний шлях між будь-якою парою станцій.

Пасажири повинні будуть заплатити певну кількість грошей для того, щоб увійти в метро на певній станції. При цьому для різних станцій ця сума може бути різною. А саме: для деякої станції X вартість входу буде рівна (в гривнях) максимальній кількості тунелів, які пасажир має проїхати для того, щоб доїхати від станції X до будь-якої іншої станції Y.

Крім того, мер міста дуже прискіпливий до чисел. Відомо, що мер любить числа $A_1, \dots, A_N$, а також ненавидить числа $B_1, \dots, B_M$. Мер не погодиться на побудову метро, якщо серед вартостей входу на станції будуть не всі його улюблені числа або буде хоча б одне, яке він ненавидить.

Завдання. Напишіть програму `metro`, яка за заданими числовими вподобаннями мера знаходитиме мінімальну кількість станцій, яку може містити метро, або визначатиме, що мережу із заданими вимогами побудувати неможливо.

Вхідні дані. Перший рядок вхідного файла `metro.dat` містить два цілих числа N та M — кількість чисел, які мер любить та ненавидить відповідно. Другий рядок містить список із N цілих чисел, розділених пропусками, — список чисел, які мер любить. Третій рядок містить в аналогічному форматі список із M чисел, які мер ненавидить.

Вихідні дані. У єдиному рядку вихідного файла `metro.sol` виведіть мінімальну кількість станцій, яку може містити метро. Якщо неможливо побудувати мережу зі заданими вимогами, виведіть одне число `-1`.

Оцінювання. Набір тестів складається з 2 блоків, для яких виконуються такі умови:

- 50 % балів: $1 \leqslant N \leqslant 2000$, $1 \leqslant M \leqslant 2000$, $1 \leqslant A_i \leqslant 2000$, $1 \leqslant B_i \leqslant 2000$.
- 50 % балів: $1 \leqslant N \leqslant 10^5$, $1 \leqslant M \leqslant 10^5$, $1 \leqslant A_i \leqslant 10^5$, $1 \leqslant B_i \leqslant 10^5$.

Приклади вхідних та вихідних даних.

metro.dat	metro.sol
2 3 4 7 13 1 3	8
2 1 4 7 6	-1

Розв'язання

Розв'язання на половину балів. З вимог умови випливає, що структурно відповідь повинна мати вигляд дерева — зв'язного неорієнтованого ациклічного графа.

Помітимо, що значення вартостей станцій залежить лише від діаметра дерева. А саме: для довільного дерева з діаметром, що містить D вершин, знайдуться всі вартості від $[D/2]$ до $D-1$ і лише вони. Тому є сенс розглядати, наприклад, лише ланцюг із D вершин, а серед чисел від $[D/2]$ до $D-1$ повинні бути всі числа $A_1, \ldots, A_N$ та не повинно бути жодного з чисел $B_1, \ldots, B_M$.

Переберемо всі можливі варіанти D і перевіримо, чи всі умови задоволено. Складність такого розв'язку: $O(W(N+M))$, де $W = 2\max\{\max\{A_i\}, \max\{B_i\}\}$.

Розв'язок на повний бал. Замість перебирання D розглянемо допустимі числа $A_1, \ldots, A_N$. Кожне з цих чисел дає певне нижнє і верхнє обмеження на значення для D. Аналогічно й кожне з недопустимих чисел $B_1, \ldots, B_M$ дає певне верхнє або нижнє обмеження на D. Перетнувши всі обмеження, знайдемо проміжок усіх можливих значень для D. Якщо проміжок пустий, то відповіді не існує, інакше виберемо мінімальну. Складність розв'язку: $O(N+M)$.

2.3. Лаврушка і розбиття масиву (Владислав Глембоцький)

Умова

Лаврушка — сумлінний учень, який мріє стати програмістом. На останньому уроку інформатики його улюблена вчителька запропонувала йому розв'язати таке завдання. Нехай $a_1, \dots, a_N$ — певна послідовність натуральних чисел. Лаврушці потрібно розділити послідовність чисел $1, 2, 3, \dots, N$ на дві послідовності $b_1, \dots, b_M$ і $c_1, \dots, c_K$ таким чином, щоб:

- Кожне натуральне число $1 \leqslant r \leqslant N$ було елементом рівно однієї з послідовностей b або c (тому $M + K = N$).
- Для кожної пари індексів $1 \leqslant i \leqslant M$, $1 \leqslant j \leqslant M$, $i \neq j$, числа a_{b_i} і a_{b_j} були взаємно простими.
- Для кожної пари індексів $1 \leqslant i \leqslant K$, $1 \leqslant j \leqslant K$, $i \neq j$, числа a_{c_i} і a_{c_j} були взаємно простими.

Взаємно простими називаються числа, найбільший спільний дільник яких дорівнює одиниці. Назвемо відповідний поділ послідовності $1, 2, 3, \dots, N$ *розбиттям* послідовності a_i.

Розбиття послідовності може бути неоднозначне. Тому вчителька попросила Лаврушу знайти таке розбиття, що максимізує кількість елементів у послідовності b_i. Якщо існує кілька розбиттів, що максимізують кількість елементів у послідовності b_i, то потрібна знайти серед них таке, щоб послідовність b_i була лексикографічно найменшою.

Будемо казати, що послідовність $q_1, q_2, \dots, q_W$ є лексикографічно меншою за послідовність $p_1, p_2, \dots, p_W$, якщо існує таке i, що $q_i < p_i$, а для довільного $j < i$ виконується $p_j = q_j$.

Завдання. Напишіть програму `split`, яка для заданої послідовності чисел a знайде оптимальне розбиття послідовності чисел $1, 2, 3, \dots, N$ на дві послідовності $b_1, \dots, b_M$ і $c_1, \dots, c_K$.

Вхідні дані. У першому рядку вхідного файла `split.dat` записано число Z ($1 \leqslant Z \leqslant 3$) — кількість тестових вхідних даних (учителька кілька разів пропонувала Лаврушці розв'язати це завдання для різних вхідних даних). Далі подаються Z тестових даних у такому форматі. У першому рядку даних міститься число N ($1 \leqslant N \leqslant 10^5$) — кількість елементів у послідовності a. У наступному рядку записано N чисел, розділених одиничними пропусками, — елементи послідовності a ($1 \leqslant a_i \leqslant 2 \times 10^6$).

Вихідні дані. Для кожних тестових даних виведіть у вихідний файл `split.sol` в одному рядку число M — кількість елементів у послідовності b, а у другому — M натуральних чисел — у порядку зростання номери елементів послідовності a, що увійшли до послідовності b. Якщо так трапиться, що вчителька помилилась і не існує жодного розбиття послідовності a, то виведіть в одному рядку `-1`.

Оцінювання. Набір тестів складається з 4 блоків, для кожного з яких виконуються такі умови:

- 24 % балів: $N \leqslant 15$, $1 \leqslant a_i \leqslant 2 \times 10^6$.
- 24 % балів: $N \leqslant 1000$, $1 \leqslant a_i \leqslant 2 \times 10^6$.
- 30 % балів: $N \leqslant 2 \times 10^4$, $1 \leqslant a_i \leqslant 2 \times 10^6$.
- 22 % балів: $N \leqslant 10^5$, $1 \leqslant a_i \leqslant 2 \times 10^6$.

Приклад вхідних та вихідних даних.

split.dat	split.sol
2 5 1 2 3 4 5 5 2 3 4 5 6	4 1 2 3 5 -1

Пояснення. У першому наборі тестових даних не можна отримати розбиття, де в послідовності b містилися б усі елементи $1, 2, 3, \ldots, N$, тому що числа 2 і 4 не взаємно прості. А в наступному наборі тестових даних взагалі не існує жодного розбиття послідовності a.

Розв'язання

Перебір. Найпростішим розв'язанням цієї задачі буде розв'язок перебором. Можемо рекурсивно або за допомогою бітових масок перебрати всі елементи першої послідовності. Тоді елементи другої послідовності визначаються автоматично. Далі необхідно перевірити, чи те, що ми отримали в результаті перебору, є розбиттям. Звісно, таке розв'язання занадто повільне й отримує тільки 24 % балів.

Графи. Задачу можна розв'язати за допомогою теорії графів. Можемо сконструювати граф таким чином: нехай вершини графа мають номери $1, 2, \ldots, N$. Тоді між двома різними вершинами v, u графа проведемо ребро тоді й лише тоді, коли числа a_v і a_u мають спільний простий дільник.

Зауважимо: ми можемо «розбити» послідовність a тоді й лише тоді, коли кожна компонента зв'язності даного графа є дводольною (тобто такою, що вершини цієї компоненти зв'язності можна «пофарбувати» в два кольори так, щоб вершини, які сполучено ребром, мали різний колір). Якщо якась компонента зв'язності не є дводольною, то ми не можемо «пофарбувати» вершини цього графа у два кольори, а відповідно, не можемо розділити нашу послідовність.

Розбиттям послідовності будуть долі побудованого графа. Якщо компонента зв'язності є дводольною, то до послідовності b потрібно включити номери вершин з тієї долі, у якій вершин більше. Якщо кількість вершин однакова, потрібно додати вершини з тієї долі, найменший номер вершини в якій менший.

Перевірка на дводольність. Як перевірити, чи компонента зв'язності графа дводольна? Доволі нескладно зробити це за допомогою пошуку в глибину. Якщо дана вершина «пофарбована» в якийсь колір (нехай білий), то всі її сусіди (такі вершини, що сполучені ребром з даною) мають бути пофарбовані в інший колір (нехай чорний). Рекурсивно для сусідів повторимо цю операцію. Якщо цього зробити не вдасться, то граф дводольним не є.

Зауважимо, що для побудови графа можна використати алгоритм Евкліда, який шукає спільний дільник за логарифмічну складність.

Авторське розв'язання. Зауважимо такий факт: довільне число від 1 до 2×10^6 має не більше ніж 7 різних простих дільників (добуток 8 найменших простих чисел більший за 2×10^6). З цього випливає, що в графі не може бути багато ребер.

Крім того, за допомогою сита Ератосфена можна ефективно визначити найменший простий дільник для кожного числа $1 \leqslant x \leqslant 2 \times 10^6$, що дозволить швидко факторизувати (розкласти на прості множники) числа, які подаються у вхідних даних.

За допомогою факторизації ми можемо сконструювати граф швидше, ніж раніше; це дозволяє отримати максимальний бал за задачу. Для кожного простого числа слід перевірити, які числа мають цей дільник. Якщо таких чисел більше ніж два, то розв'язку не існує (у графі утворяться три вершини, попарно сполучені ребрами). Інакше для кожної пари чисел, що мають даний дільник, додаємо відповідне ребро до графа.

2.4. Камінчиковий автомат (Данило Мисак)

Умова

Камінчиковий автомат «Мардж-2016» працює на квадратній таблиці, розбитій на комірки, у кожній з яких у довільний момент часу лежить певна (можливо, нульова) кількість камінців. На вхід автомат приймає два натуральних числа A та B у вигляді кількості камінців, розташованих у початковий момент часу в лівій верхній та у правій верхній комірках таблиці відповідно. У решті комірок у початковий момент часу камінців немає. Кожна комірка має прикріплену до неї функцію: ця функція приймає на вхід кількість камінців, розташованих на даний момент у комірці, а повертає кількості камінців, які на наступній ітерації необхідно перекласти з цієї комірки у сусідні з нею верхню, праву, нижню та ліву клітинки таблиці. Таким чином, на кожній ітерації камінчиковий автомат просто перекладає відповідні кількості камінців з усіх комірок у сусідні відповідно до інструкцій, які повернули прикріплені до комірок функції. Усі перекладання здійснюються водночас. Якщо під час чергової ітерації жодна комірка не переклала жодного камінця у жодну сусідню комірку, після цієї ітерації виконання «програми» завершується. На виході автомат повертає два числа: кількість камінців у лівій нижній та правій нижній комірках. Кількості камінців, які на момент завершення програми містили всі інші комірки, можуть бути довільними та ігноруються.

Завдання. Дана задача має 5 підзадач:

1. У першій підзадачі на виході потрібно отримати $(0, A + B)$, тобто у лівій нижній комірці камінців бути не має, а в правій нижній кількість камінців має дорівнювати сумі чисел на вході. Ця підзадача має вартість **10 балів**.
2. У другій підзадачі на виході потрібно отримати $(0, |A - B|)$, тобто у правій нижній комірці кількість камінців має дорівнювати модулю різниці чисел на вході (а в лівій нижній комірці камінців бути не має). Ця підзадача має вартість **20 балів**.
3. У третій підзадачі на виході необхідно отримати $(0, \min\{A, B\})$, де $\min\{A, B\}$ позначає менше з двох чисел A та B (якщо $A = B$, то $\min\{A, B\} = A = B$). Ця підзадача має вартість **15 балів**.
4. У четвертій підзадачі на виході необхідно отримати $(0, \max\{A, B\})$, де $\max\{A, B\}$ позначає більше з двох чисел A та B (якщо $A = B$, то $\max\{A, B\} = A = B$). Ця підзадача має вартість **15 балів**.
5. У п’ятій підзадачі на виході потрібно отримати $(1, 0)$ за умови $A > B$; $(0, 1)$ за умови $A < B$; $(0, 0)$ за умови $A = B$. Інакше кажучи, навпроти того місця, де на вході розташовувалося строго більше з двох чисел, на виході має розташовуватися один камінець. Ця підзадача має вартість **40 балів**.

Напишіть функцію/процедуру `automaton`, що за кількістю камінців у комірці таблиці та її координатами визначить, скільки камінців треба перекласти у сусідні з нею клітинки відповідно до вимог підзадачі. Інакше кажучи, вам необхідно реалізувати алгоритм роботи камінчикового автомата для кожної з п’яти підзадач. Усі підзадачі складаються з окремих блоків тестів; кожен блок має вартість 5 балів і може містити один або кілька тестів. При цьому, щоб заробити відповідні 5 балів, ваш алгоритм має пройти всі тести з блоку.

Деталі реалізації. Ви маєте надіслати файл, що містить реалізацію функції `automaton` та, за потреби, інший код, необхідний для коректної роботи цієї функції, але не містить самого тіла програми (тобто функції `main` у C++ чи блоку `begin`/`end.` у Pascal'і). При тестуванні ваш файл буде доповнено спеціальним тілом програми, написаним журі. Тіло відповідним чином викликатиме написану вами функцію та перевірятиме коректність алгоритму, який вона втілює. Реалізована вами функція повинна мати таку сигнатуру: C++, Pascal.

Параметри функції.

1. **subproblem** — номер підзадачі (див. вище). Номер підзадачі не змінюється під час запусків функції на одній і тій самій програмі.
2. **size** — розмір квадратної таблиці (кількість клітинок в одній горизонталі/вертикалі). Розмір не змінюється під час запусків функції на одній і тій самій програмі.
3. **row** — рядок, у якому розташована комірка; нумерація від 1 до `size` зверху вниз: 1 — верхній рядок, `size` — нижній рядок.
4. **column** — стовпець, у якому розташована комірка; нумерація від 1 до `size` зліва направо: 1 — лівий стовпець, `size` — правий стовпець.
5. **pebbles** — кількість камінців у комірці, додатне ціле число (якщо ця кількість нульова, жодних перекладань здійснити не можна, і виклику функції не відбувається). Ви можете змінити значення цієї змінної усередині функції, але така зміна не матиме жодного ефекту: кількість камінців після перекладань автоматично стане рівною різниці між початковою кількістю камінців у клітинці та кількістю перекладених у сусідні комірки камінців (див. далі).
6. **top** — початкове значення цього параметра (аргумент, який передає тіло програми), дорівнює 0. Якщо у даної клітинки є сусідня зверху і туди необхідно перекласти камінці з даної, це значення слід переписати, зробивши рівним відповідній кількості камінців.
7. **right** — початкове значення цього параметра дорівнює 0. Якщо у даної клітинки є сусідня справа і туди необхідно перекласти камінці з даної, це значення слід переписати, зробивши рівним відповідній кількості камінців.
8. **bottom** — початкове значення цього параметра дорівнює 0. Якщо у даної клітинки є сусідня знизу і туди необхідно перекласти камінці з даної, це значення слід переписати, зробивши рівним відповідній кількості камінців.
9. **left** — початкове значення цього параметра дорівнює 0. Якщо у даної клітинки є сусідня зліва і туди необхідно перекласти камінці з даної, це значення слід переписати, зробивши рівним відповідній кількості камінців.

Кінцеві значення параметрів `top`, `right`, `bottom`, `left` повинні бути невід'ємними та в сумі не мають перевищувати значення `pebbles`.

На кожному кроці функція повинна визначати перекладання, виходячи **виключно з даних, переданих їй на цьому кроці**. На результат роботи функції жодним чином не повинна впливати попередня взаємодія з тілом програми. Крім того, функція повинна бути детермінованою, тобто для сталих вхідних даних завжди повертати одні й ті самі значення.

Обмеження.

- Розмір таблиці може бути довільним числом від 5 до 10 включно.
- Числа на вході можуть мати довільні значення від 1 до 100 включно.

- Кількість ітерацій, після яких автомат зупиниться, не має перевищувати $10\,000$.

Власноручне тестування. Ви можете завантажити приклад основного тіла програми automaton_tester.cpp (C++) або automaton_tester.pas (Pascal), що запускає функцію `automaton` на вхідних даних, заданих у створеному вами вхідному файлі, і виводить кінцевий стан таблиці (або стан таблиці після $10\,000$ ітерацій, якщо автомат досі не зупинився). Щоб використати цю програму, розташуйте її в одному каталозі з файлом `automaton.cpp`/`automaton.pas`, який містить вашу реалізацію функції, та створіть у цьому ж каталозі файл `automaton.dat` зі структурою, описаною в наступному абзаці. Зауважте, що основне тіло програми, яке буде використано для оцінювання вашої функції, відрізнятиметься від наданого вам у `automaton_tester.cpp`/`automaton_tester.pas` прикладу. Зокрема, під час перевірки функцію може бути викликано для довільної клітинки таблиці та довільної кількості каменців від 1 до 200.

automaton_tester: вхідні дані. Єдиний рядок містить чотири натуральних числа: відповідно номер підзадачі, розмір таблиці, число A та число B.

automaton_tester: вихідні дані. Перший рядок вихідного файла міститиме пару чисел на виході автомата або словесну діагностику помилки в разі, якщо на якомусь кроці функція порушила технічні вимоги чи обмеження на кількість ітерацій. Якщо технічних порушень не було, наступні рядки міститимуть повний кінцевий стан таблиці.

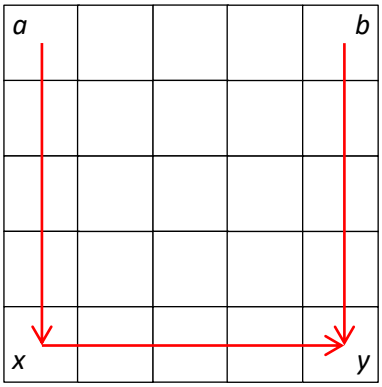
automaton_tester: приклад вхідних та вихідних даних.

automaton.dat	automaton.sol
1 5 3 4	0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7

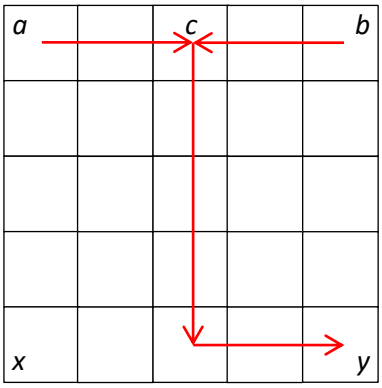
Розв'язання

Щоб краще сприймати наведений матеріал, пропонуємо скористатися візуалізатором авторського розв'язку.

Підзадача 1: $(\mathbf{0},\ \boldsymbol{A}+\boldsymbol{B})$**.** Ця підзадача є тривіальною: достатньо пересувати всі камінці у сусідню клітинку вниз або праворуч (якщо тільки це не права нижня клітинка).

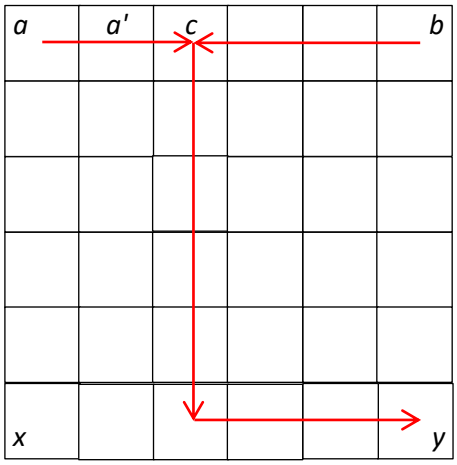


Підзадача 2: $(\mathbf{0},\ |\boldsymbol{A}-\boldsymbol{B}|)$**.** У випадку **непарного** розміру таблиці маємо таку схему (пояснення див. нижче):



Із клітинки a перекладаємо у клітинку праворуч щоразу по одному камінцю. З усіх клітинок між a та c (центральною клітинкою першого рядка) перекладаємо ці камінці далі праворуч. Таким чином утворюється «потік» одиничних камінців у c. Симетричний потік організовуємо і з клітинки b. Тоді у клітинці c будуть «зливатися» щоразу по два камінці, поки потік іде з обох боків. Відповідно, поки кількість камінців у c парна, ми залишаємо їх на місці і нічого не рухаємо. Щойно потік з одного боку (з того, де число було меншим) вичерпається, у c надійде лише один камінець з протилежного боку, і кількість камінців у c стане непарною. У такому випадку ми починаємо перекладати по одному камінцю уздовж маршруту до y.

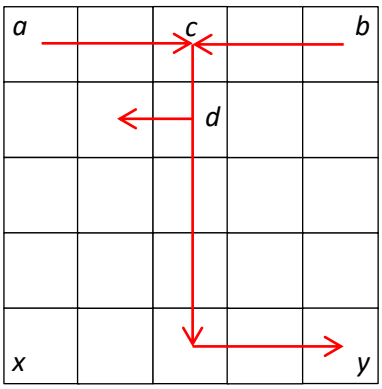
У випадку **парного** розміру принципово схема залишається тією ж, але, щоб мати змогу «зливати» камінці по два, ми штучно затримуємо потік з лівого боку на одну ітерацію:



Клітинка a, як і раніше, передає праворуч по одному камінцю. Натомість клітинка a' (правий сусід a) передає один камінець далі лише в тому випадку, якщо зараз у комірці їх два. Таким чином створюється затримка в одну ітерацію. Щоправда, при цьому, коли потік закінчиться, у клітинці a' залишиться один камінець — «жертва», яку ми мусимо зробити за організацію штучної затримки. Щоб цей один неврахований камінець не вплинув на відповідь, для клітинки b введемо додаткове правило: якщо в ній залишається один камінець, ліворуч ми його не передаємо. Таким чином ми досягаємо того, що обидва потоки зменшилися на один камінець, а отже, різниця чисел виходить якраз правильною.

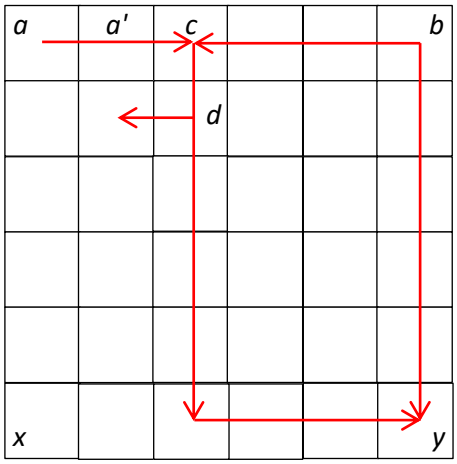
Підзадачі 3 і 4: $(\mathbf{0}, \mathbf{min/max}\{\boldsymbol{A}, \boldsymbol{B}\})$**.** Наступні дві підзадачі можна розв'язати за допомогою додаткових модифікацій до схеми порівняно з попередньою підзадачею.

У випадку **непарного** розміру маємо таку схему:



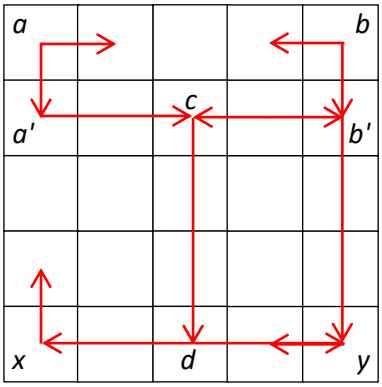
На відміну від другої підзадачі, комірка c завжди передає увесь свій вміст униз, а «розподілом» камінців займається комірка d: якщо кількість камінців дорівнює 2, один вона відкладає ліворуч, а один — униз по маршруту до y; якщо кількість камінців дорівнює 1, вона відкладає його або завжди ліворуч — якщо ми шукаємо мінімум — або завжди вниз — якщо шукаємо максимум.

У випадку **парного** розміру маємо таку схему:



Як і у парному випадку другої підзадачі, ми затримуємо потік у клітинці a'. Клітинка c, як у непарному випадку вище, спускає увесь свій вміст комірці d, яка у той самий спосіб, що й раніше, виконує розподіл камінців. Єдина проблема виникає із тим, що для подолання проблеми парності ми пожертвували одним камінцем із кожного з двох потоків у клітинку c. Тому й відповідь у нас вийде на одиницю меншою, ніж потрібно. Виправити це можна, відправивши додатково затриманий камінець із клітинки b вниз прямо в y.

Підзадача 5: $(\mathbf{0},\ \mathbf{1})/(\mathbf{1},\ \mathbf{0})/(\mathbf{0},\ \mathbf{0})$**.** У випадку **непарного** розміру схема може бути такою:



Загальна ідея така: ми відкладаємо по одному камінцю праворуч від a та ліворуч від b, поки там не стане по два камінці. Щойно в a стало два камінці, ми відправляємо їх за маршрутом $a \to a' \to c$ (c — середина другого рядка). Щойно два камінці стане в b, ми аналогічно скористаємось маршрутом $b \to b' \to c$, але надішлемо уздовж нього лише один камінець із двох, а інший — за допомогою розподільчої клітинки b' униз до y.

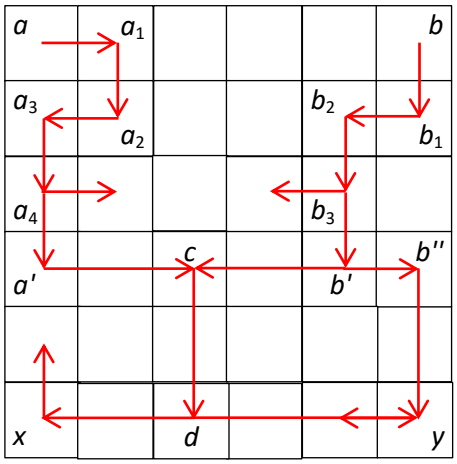
Із c камінці йдуть униз у d. Своєю чергою d ті камінці, які прийшли, розподіляє так: якщо в комірці один камінець, він нікуди не рухається; якщо в комірці два камінці, один із них іде ліворуч в x, а інший не рухається. Цим ми досягаємо того, що можемо розрізнити, що у d прийшло раніше, а що пізніше: якщо спершу прийшли два камінці з a, а потім один камінець із b (це значить, що число B було більшим за A), то ми двічі відправимо по одному камінцю з d у x. При цьому, щоб уникнути наявності цих двох зайвих камінців в x у випадку, якщо $B > A$, нам достатньо ввести правило, що два камінці з x піднімаються на клітинку вгору. Якщо натомість спершу в d прийшов один камінець з b, а потім — два камінці з a (тобто $A > B$), то в d опинилося три камінці, один із яких слід відправити в x, а решту два — в y, звідки, за допомогою правила «якщо в y більше ніж один камінець, перекласти всі камінці вліво», ми зможемо дістати зайвий камінець, який ми переслали туди з клітинки b раніше.

Залишається розглянути випадок, коли $A = B$. У такому випадку в певний момент часу в клітинці c опиниться відразу три камінці, які ми групою з усіх трьох домовимося надсилати за маршрутом $c \to b' \to y$, із тим, щоб вивести з y зайвий надісланий туди камінець.

Алгоритм побудовано таким чином, що він працює, навіть коли одне з чисел A та B менше за 2 (тобто дорівнює 1), і коректного «порівняння» клітинкою d не відбувається. Достатньо лише домовитися, що клітинки a та b нічого не роблять, якщо в них розташовується рівно один камінець:

- Якщо $A = B = 1$, ніякого руху в таблиці не відбувається, і в обох вихідних клітинках маємо нулі.
- Якщо $A = 1$, а $B > 1$, то в y опиняється один надісланий з b камінець (насправді це і є його призначенням), а в d «застрягає» інший.
- Якщо $A > 1$, а $B = 1$, то в y камінця не опиняється, зате в x опиняється один камінець з A (а інший «застрягає» в d).

У випадку **парного** розміру доведеться ще трохи ускладнити схему:



Спершу нам необхідно вирішити проблему парності, але цього разу це потрібно зробити, не залишаючи за собою камінця, як ми це робили раніше. Затримуватимемо рух камінців з комірки b: досягнемо того, щоб B камінців опинилося в комірці b_3 не за три ітерації, а за чотири. Насправді одним камінцем нам доведеться пожертвувати, але тільки в тому випадку, якщо $B = 1$. Але тоді цей єдиний камінець нам і не знадобиться.

- З комірки b перекладемо вниз 1 камінець, якщо кількість камінців у комірці парна; в іншому разі перекладемо вниз усі камінці.
- З комірки b_1 перекладемо ліворуч 1 камінець, якщо кількість камінців у цій клітинці непарна і більша за 1; перекладемо ліворуч усі камінці, якщо кількість камінців парна; не перекладатимемо нічого, якщо кількість камінців дорівнює 1.
- З комірки b_2 перекладемо униз усі камінці, якщо їх більше ніж 1; інакше не перекладатимемо.

За цей же час (4 ітерації) перекладемо тривіальним чином вміст комірки a у клітинку a_4. Далі наші дії ідентичні тим, що ми проводили у випадку непарного розміру, якщо за початкові клітинки взяти a_4 та b_3. Єдиною відмінністю буде необхідність розділити шлях із b' у комірку y на два фрагменти: один крок праворуч із b' у b'', після чого вниз у y. Утім, функціонуватиме цей маршрут точно так само, як і раніше.