ОЛІМПІАДА З ІНФОРМАТИКИ У МІСТІ КИЄВІ У 2013–2014 НАВЧАЛЬНОМУ РОЦІ

Кордубан Дмитро Олександрович,
аспірант Інституту програмних систем Національної академії наук України.
Мисак Данило Петрович,
керівник гуртка СШ №52 м. Києва.
Рудик Олександр Борисович,
доцент Київського університету імені Бориса Грінченка.

Продовження, початок у №1 за 2015 рік

IV. УМОВИ ЗАВДАНЬ ІІІ ЕТАПУ

1. Лижний спорт

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

Як відомо, нещодавно Україна подала заявку на проведення зимових Олімпійських ігор 2022 року. Щоб збільшити шанси перемоги цієї заявки над іншими, нам потрібно спроектувати якнайдосконаліші олімпійські об'єкти. Одним із таких об'єктів є лижна траса, яку прокладатимуть уздовж фрагмента вузького плоскогір'я. Плоскогір'я є низкою пологих підйомів і спусків. Якщо його розбити на ділянки завдовжки 1 км, кожну ділянку можна охарактеризувати як підйом або як спуск. На оптимальній для спортсменів трасі кількість ділянок-підйомів і кількість ділянок-спусків мають збігатися.

Завдання. Знаючи, у якій послідовності йдуть підйоми і спуски плоскогір'я, визначте довжину найдовшої потенційної траси, тобто такого фрагмента плоскогір'я, що містить однакову кількість підйомів і спусків.

Вхідні дані. У першому рядку вхідного файлу вказано натуральне число n — кількість кілометрових ділянок плоскогір'я. У другому рядку записано n чисел, що задають рельєф плоскогір'я. Кожне з цих чисел — або одиниця (підйом), або нуль (спуск). Плоскогір'я не є круглим, тобто перша та остання його ділянки не сполучені між собою.

У 40 % тестів $2 \le n \le 100$.

У 30 % тестів $100 < n \le 10\,000$.

У 30 % тестів $10\,000 < n \le 1\,000\,000$.

Усі тести мають однакову вартість.

Вихідні дані. Єдиний рядок вихідного файлу повинен містити довжину в кілометрах найдовшого фрагмента плоскогір'я, що містить однакову кількість підйомів і спусків. Якщо жодного такого фрагмента немає, виведіть число 0.

Приклади

skiing.in	skiing.out
6 1 1 1 0 0 1	4
5 0 1 1 1 1	2
4 0 0 0 0	0
6 1 0 1 0 1 0	6

Пояснення до прикладів

У першому прикладі фрагмент 1 1 0 0 довжини 4 містить по два підйоми і спуски. Таку саму властивість має і фрагмент 1 0 0 1. Довших фрагментів, що містять однакову кількість підйомів і спусків, задана послідовність немає.

У другому прикладі фрагмент 0 1 довжини 2 містить по одному підйому і спуску. Довших фрагментів із рівною кількістю підйомів і спусків немає.

У третьому прикладі, як видно, жоден фрагмент не містить однакової кількості підйомів і спусків.

У четвертому прикладі найдовшим фрагментом, що містить рівну кількість підйомів і спусків, є вся послідовність.

2. Голосування

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

Після завершення прийому заявок на проведення Олімпіади під час спеціального засідання Міжнародного олімпійського комітету проходять вибори заявки-переможця. Процедура голосування така. Спершу всі делегати, присутні на засіданні, голосують за будь-яку одну з n отриманих заявок-претендентів. Заявку, що набрала найменше голосів, виключають зі списку, після чого голосування повторюють — уже між $n-1$ претендентом. Далі знову виключають заявку, що набрала найменше голосів, і голосування проводять ще раз. Цю процедуру повторюють доти, доки не залишиться єдина заявка, яка й стає переможцем.

Назвімо *послідовним* такого делегата, що має список уподобань (перестановку чисел від 1 до n) і в кожному раунді голосує за ту заявку, що з-поміж тих, які ще залишилися, стоїть першою в його списку. Дехто з делегатів може й не бути послідовним і голосувати, наприклад, випадково.

Завдання. Знаючи результати кожного з раундів голосування, визначте, якою серед усіх присутніх на засіданні делегатів могла бути найбільша кількість послідовних. Подайте також приклад можливих уподобань цих послідовних делегатів.

Вхідні дані. У першому рядку вхідного файлу вказано натуральне число n — кількість поданих заявок. Далі йде n рядків, що містять невід'ємні цілі числа: перший з цих рядків відповідає заявці, що перемогла; наступний рядок — заявці, що вибула в останньому раунді; ...; передостанній рядок — заявці, що вибула другою; останній рядок — заявці, що вибула першою.

Число на позиції j в рядку, що відповідає деякій заявці, дорівнює кількості делегатів, які проголосували за дану заявку в j-му раунді. Таким чином, у k-му рядку вхідного файлу записано $n+2-k$ чисел, $3 \le k \le n+1$, а в другому рядку записано $n-1$ число. Відомо, що кількість голосів, відданих за заявку, яка вибула в деякому раунді, строго менша за кількість голосів, яку здобула в цьому раунді будь-яка інша заявка. Сума голосів у кожному з раундів однакова і дорівнює кількості делегатів, присутніх на засіданні. І кількість заявок, і кількість делегатів не менші за 2 і не перевищують 100.

Вихідні дані. У першому рядку вихідного файлу виведіть число m — найбільшу можливу кількість послідовних делегатів. Наступні m рядків повинні задавати приклад уподобань, які могли мати m послідовних делегатів. У кожному з цих рядків має міститися по n чисел — перестановка чисел від 1 до n, що відповідає порядку вподобань відповідного делегата. Число 1 відповідає при цьому заявці, що перемогла; число 2 — заявці, що вибула в останньому раунді; ...; число $n-1$ — заявці, що вибула другою; число n — заявці, що вибула першою. Лівіше у списку повинні стояти номери привабливіших заявок, а правіше — менш привабливих, тобто делегат щоразу голосуватиме за ту заявку, номер якої стоїть якомога ближче до початку його списку. Порядок m рядків, що задають уподобання послідовних делегатів, може бути довільним. Якщо можливих прикладів уподобань m делегатів є декілька, виведіть будь-який із них.

Якщо перший рядок вихідного файлу, створеного вашою програмою, міститиме правильну відповідь (найбільшу можливу кількість послідовних делегатів), але наступні рядки будуть порожніми, відсутніми або задаватимуть неправильний приклад уподобань, то за відповідний тест буде нараховано 40% балів.

Приклади

election.in	election.out
3 1 2 2 1 0	2 1 2 3 2 1 3
5 3 1 2 7 3 6 9 5 3 5 1 2 0 1	6 1 2 3 4 5 2 1 3 4 5 2 1 3 4 5 2 1 3 4 5 3 2 1 4 5 5 2 1 3 4
2 2 1	3 1 2 1 2 2 1

Пояснення до прикладів

У першому прикладі маємо трьох делегатів (1+2+0=2+1=3). Усі три не можуть бути послідовними, бо інакше проголосували б у другому раунді так само, як і в першому. Разом із тим два з трьох делегатів можуть бути послідовними, при цьому один з них серед трьох заявок надає перевагу першій, а інший — другій. Непослідовний же делегат у першому раунді голосує за другу заявку, а в другому — за першу.

У другому прикладі маємо 12 делегатів: 3+3+3+ +2+1=1+6+5+0=2+9+1=7+5=12. Щонайбільше шість із них можуть бути послідовними і голосувати відповідно до вподобань, наведених у прикладі вихідного файлу. Тоді результати голосування, наведені у прикладі вхідного файлу, досягаються, якщо інші шість непослідовних делегатів голосують так: у першому раунді по 2 за першу, третю і четверту заявки; у другому раунді 2 за другу і 4 за третю заявку; у третьому раунді 1 за першу і 5 за другу заявку; в останньому раунді всі 6 за першу заявку.

У третьому прикладі кожен із трьох делегатів, присутніх на голосуванні, може бути послідовним.

3. Олімпійське містечко

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

Якщо Україна хоче здобути право приймати Олімпійські ігри, нам треба подбати не лише про спортивні споруди, але й про туристичну інфраструктуру, а також про житловий комплекс, де мешкатимуть під час Ігор спортсмени, — про Олімпійське містечко. Одним з аспектів створення сучасного Олімпійського містечка повинні стати «розумні» програмні засоби, що забезпечуватимуть комфортне буття олімпійців. Одну з таких програм маєте написати саме ви.

Уявімо, що на Олімпіаду прибула команда, яка складається з парної кількості спортсменів, і її вирішили розселити у двомісні номери. Звісно, у кожного спортсмена є побажання стосовно того, з яким саме членом команди він або вона хотіли б поселитися в одному номері.

Завдання. Знаючи, кого хоче мати за співмешканця кожен із членів команди, визначте, яку найбільшу кількість спортсменів можна поселити відповідно до їхніх побажань: знайдіть найбільше ціле k, для якого існує таке розміщення спортсменів по кімнатах, що відповідає бажанню рівно k спортсменів.

Вхідні дані. У першому рядку вхідного файлу записано парне натуральне число n — кількість членів команди. Усіх спортсменів занумеровано натуральними числами від 1 до n. У другому рядку вказано n натуральних чисел у межах від 1 до n кожне: k-те з цих чисел задає номер члена команди, з яким би хотів поселитися k-й спортсмен (цей номер, природно, не дорівнює самому числу k).

У 40% тестів $4 \le n < 20$.

У 30% тестів $20 \le n \le 2000$.

У 30% тестів $2000 < n \le 200\,000$.

Усі тести мають однакову вартість.

Вихідні дані. У вихідний файл виведіть єдине число — найбільшу кількість спортсменів, побажання яких може задовольнити деяке розселення.

Приклади

village.in	village.out
4 2 1 4 1	3
8 2 3 4 5 6 7 8 1	4
6 3 3 5 3 3 3	2

Пояснення до прикладів

У першому прикладі можна задовольнити побажання трьох членів команди, якщо першого спортсмена поселити з другим, а третього — з четвертим. За будь-якого іншого розміщення або другий, або че-

твертий спортсмен не будуть мешкати разом із першим, тому задовольнити побажання водночас усіх чотирьох членів команди неможливо.

У другому прикладі можна задовольнити побажання чотирьох спортсменів, приміром, з непарними номерами: першого поселити з другим, третього з четвертим, п'ятого з шостим, а сьомого — з восьмим. Задовольнити ж відразу п'ятьох членів команди не вийде: жодні два спортсмени не хочуть мешкати разом, тому в кожній з чотирьох кімнат житиме щонайбільше один задоволений спортсмен.

У третьому прикладі задоволеними можуть виявитися щонайбільше двоє спортсменів: третій і лише один серед решти членів команди, хто хоче з ним жити. Щоб задовольнити двох спортсменів, третього потрібно, очевидно, поселити з п'ятим.

4. Перегони

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

Велосипедист готується до гірських велоперегонів з роздільним стартом (тобто на час). Він оцінив свою очікувану швидкість використання енергії (тобто *потужність*) залежно від часу після початку гонки й хоче розробити відповідний графік харчування протягом перегонів. Єдиним джерелом енергії велосипедиста будуть вуглеводи у поживних батончиках. Усі батончики однакові й містять два типи вуглеводів: швидко засвоювані (моно- і дисахариди) й повільно засвоювані (полісахариди). Ефект від споживання одного батончика можна наблизити такою моделлю: перші t_1 секунд після споживання батончик дозволяє велосипедисту розвинути потужність a міліват («швидкі» вуглеводи), наступні t_2 секунд — потужність b міліват («повільні» вуглеводи). Після цього ефект від споживання батончика повністю зникає. Для спрощення вважатимемо:

- модель травлення і використання вуглеводів лінійна, тобто потужності від споживання різних батончиків у будь-який момент часу додаються;
- надлишки енергії не накопичуються;
- велосипедист може споживати батончики миттєво будь-коли під час або до початку перегонів.

Завдання. Знайти найменшу кількість батончиків, необхідних для того, щоб задовольнити енергетичні потреби велосипедиста на весь час перегонів.

Вхідні дані. Перший рядок вхідного файлу містить п'ять цілих чисел n, t_1, t_2, a, b — очікуваний час перегонів у секундах та параметри батончика, $1 \le n, t_1, t_2 \le 10^5$, $1 \le a, b \le 10^9$.

Другий рядок містить n цілих чисел $p_0, p_1, ..., p_{n-1}$ — величини бажаної потужності велосипедиста для кожної секунди після початку перегонів, $1 \le p_j \le 10^9$.

Вихідні дані. Єдиний рядок вихідного файлу має містити одне число — мінімальну необхідну кількість батончиків.

Приклади

race.in	race.out
7 2 3 5 2 3 4 5 5 5 4 5	4
5 10 10 1 10 10 10 10 10 10	1

5. Вибирання каменів

Максимальна оцінка: 200 балів
Обмеження на час: 1 сек.
Обмеження на пам'ять: 64 МБ
Вхідний файл: stones.in
Вихідний файл: stones.out
Програма: stones.*

Середньовічні європейці вперше отримали опис Китаю від Марко Поло, який відвідав країну за правління першого імператора монгольської династії Юань — Хубілáя.

Подорожуючи Китаєм, одного дня Марко побачив двох китайців, які сиділи на узбіччі дороги під вербою і перекладали камінці, не помічаючи нічого навколо. Марко попросив їх пояснити, що вони роблять. Йому розповіли правила старовинної гри «цзяньшицзи» (вибирання каменів), у яку вони саме грали.

На початку гри є 2 купи камінців. Два гравці по черзі забирають камінці з цих куп: або лише з однієї групи довільну *додатну* кількість, або з обох куп *однакову додатну* кількість. Переможцем вважають того, хто зробить останній хід.

Марко помітив, що один із гравців крутив у руках табличку з таким чудернацьким зображенням (рис. 1). Цей гравець після ходу суперника кидав погляд на табличку і одразу робив хід у відповідь. І майже постійно вигравав.

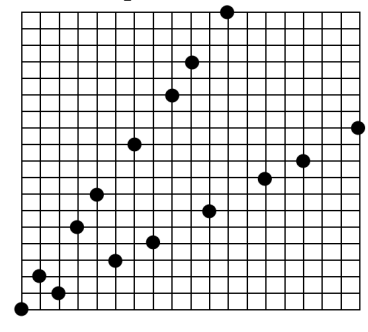


Рис. 1

Завдання. Допоможіть Марку, щоб він зміг вибирати камені так само успішно, як китаєць з його чудернацькою табличкою.

Вхідні дані. Єдиний рядок вхідного файлу містить 2 невід'ємних цілих числа — кількості камінців у обох купках. Ці кількості не перевищують 10^7+7.

Вихідні дані. Єдиний рядок вихідного файлу має містити 3 невід'ємних цілих числа. Перше /друге/ третє число — найбільша кількість камінців, яку потрібно забрати відповідно з першої /другої/ кожної купки, щоб здійснити виграшний хід. Якщо якогось *виграшного* ходу немає, то відповідне число дорівнює нулю.

Примітка. Виграшний хід — це хід, що призводить до поразки суперника *в кінці гри* незалежно від дій суперника після цього ходу за умови, що даний гравець і надалі буде дотримуватися виграшної стратегії, тобто робити виграшні ходи.

Приклади

stones.in	stones.out
3 4	0 0 2
5 3	0 0 0
3 6	0 1 0

Примітка: 36 балів буде присуджено за успішне проходження тестів, у яких числа у вхідному файлі менші за 19.

6. Перехрестя

Максимальна оцінка: 200 балів
Обмеження на час: 2 сек.
Обмеження на пам'ять: 64 МБ
Вхідний файл: crossing.in
Вихідний файл: crossing.out
Програма: crossing.*

Перший китайський імператор монгольської династії Юань — Хубілáй (онук Чингісхана), якого монголи прозивали кагáн Сецéн — проводив завойовницькі війни. Тому постійно потребував грошей, які намагався отримати від купців, стягуючи мито. Хитрі купці часто обминали перехрестя з митарями. А завойовників-монголів було замало, щоб перекрити *всі* перехрестя. Тому митарів намагалися ставити так, щоб їх неможливо було оминути.

Завдання. З'ясуйте, які перехрестя *повністю* розбивають систему шляхів на не зв'язані між собою складові.

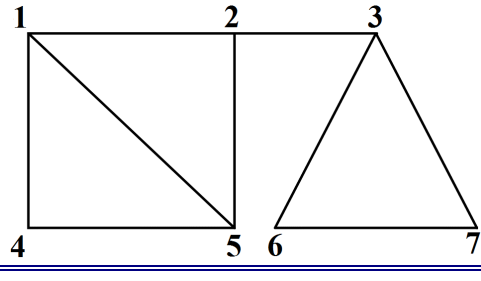
Вхідні дані. Усі пункти системи доріг — перехрестя й закінчення доріг — занумеровано послідовними натуральними числами у межах від 1 до деякого натурального числа n включно, $n \le 1234567$. Кожний рядок вхідного файлу містить пару натуральних чисел — номерів пунктів, які сполучено дорогою без пунктів між кінцями. Так перераховано всі такі дороги *по одному разу*. Цими дорогами *з будь-якого пункту можна потрапити у будь-який інший пункт*. Кількість доріг не перевищує 1234567.

Вихідні дані. Єдиний рядок вихідного файлу має містити *у порядку зростання* номери усіх пунктів, вилучення кожного з яких (лише одного) разом з дорогами, що з нього виходять (входять у нього), призводить до розбиття системи шляхів на не зв'язані між собою складові (щонайменше на дві). Інакше кажучи, після такого вилучення перестає справджуватися передостаннє речення опису вхідних даних. Купець не зможе оминути митаря на такому перехресті, якщо рухатиметься з однієї складової системи доріг до іншої. Вхідні дані гарантують існування щонайменше одного такого перехрестя.

Приклад

crossing.in	crossing.out
1 2 1 4 1 5 2 5 2 3 3 6 3 7 4 5 6 7	2 3

ілюстрація



V. ІДЕЇ РОЗВ'ЯЗАННЯ ЗАВДАНЬ ІІІ ЕТАПУ

1. Лижний спорт (автор — Данило Мисак)

Ідейно найпростішим способом розв'язати задачу є такий: перебрати всі фрагменти послідовності, для кожного фрагмента підрахувати кількість одиниць і нулів на ньому і серед тих фрагментів, де ці кількості збігаються, вибрати найдовший. Такий алгоритм має час виконання порядку $O(n^3)$: усього $O(n^2)$ фрагментів, а на перевірку одного фрагмента потрібно $O(n)$ операцій. У найпростішому вигляді алгоритм набирає 40 балів. Якщо його оптимізувати, наприклад перебирати фрагменти в порядку від найдовших до найкоротших і зупиняти виконання на першому фрагменті, що задовольняє умову, то вдасться добрати ще 10 балів.

Можна вдосконалити цей алгоритм і досягти оцінки $O(n^2)$. Для цього розглядаємо фрагменти обов'язково в такому порядку: спочатку всі, які починаються з першого члена послідовності (у порядку від найкоротшого до найдовшого), потім усі, які починаються з другого члена послідовності, потім усі, які починаються з третього члена послідовності і т. д. При цьому, розглядаючи наступний фрагмент із тим самим лівим кінцем, що попередній, ми вже знаємо кількість нулів та одиниць на ньому за винятком одного-єдиного нового числа в кінці цього фрагмента. Якщо ж наступний фрагмент має інший лівий кінець, ніж попередній, то він короткий (довжини 1). Тож на аналіз кожного з $O(n^2)$ фрагментів потрібно лише $O(1)$ операцій. Такий алгоритм набирає 70 балів. Існує й дещо інший підхід, який після попередньої підготовки дозволяє аналізувати фрагмент за фіксований час (тобто за $O(1)$), але при цьому не змушує прив'язуватися до певного порядку перебору. Якщо перебирати фрагменти від найдовшого до найкоротшого, цей підхід заробляє 80 балів.

Усі бали дозволяє здобути алгоритм, що дає відповідь за $O(n)$ часу. Уявімо, що ліва точка плоскогір'я має певну фіксовану висоту (приміром, 10^6 м), а кожен підйом і спуск відповідно збільшують або зменшують цю висоту на 1 м. Якщо деякий фрагмент плоскогір'я містить однакову кількість підйомів і спусків, то висота, на якій розташований лівий кінець цього фрагмента, дорівнює висоті, на якій розташований правий кінець фрагмента. І навпаки: якщо лівий і правий кінці фрагмента розташовані на однаковій висоті, то фрагмент містить однакову кількість підйомів і спусків. Отже, якщо замість масиву нулів і одиниць розглядати масив чисел-висот, задачу можна переформулювати так: знайти в масиві два однакових числа на якнайбільшій відстані одне від одного. Цю задачу можна розв'язати так. Проходячи масив висот зліва направо, для кожного значення висоти h будемо запам'ятовувати в окремому індексному масиві *index* першу позицію *index*[h], на якій у масиві висот це значення трапилося. Якщо ж значення h трапляється не вперше, то просто порівнюємо фрагмент, лівий кінець якого — *index*[h], а правий кінець — поточна позиція, з найдовшим знайденим на даний момент фрагментом з однаковими числами-висотами на кінцях.

2. Голосування (автор — Данило Мисак)

Нехай D — деяка підмножина присутніх на виборах делегатів. Залишимо у таблиці голосування голоси лише тих делегатів, що належать до підмножини D. Числа в комірках нової таблиці, очевидно, не перевищують відповідні числа у початковій таблиці (але останнє число в кожному стовпці вже не обов'язково є найменшим).

Лема. *Усі делегати з підмножини D можуть бути послідовними в тому й лише у тому випадку, якщо в кожному рядку нової таблиці числа йдуть в порядку неспадання.*

Доведення. Якщо послідовний делегат голосує в деякому раунді за певну заявку, то він продовжуватиме голосувати за цю ж заявку доти, доки її не буде виключено. Тож якщо всі делегати з множини D послідовні, то всі, хто голосував за певну заявку в деякому раунді, будуть голосувати за неї і в наступному. Отже, числа в рядках ідуть у порядку неспадання.

Тепер, навпаки, нехай числа в рядках таблиці йдуть у порядку неспадання. Тоді можна безпосередньо побудувати приклад уподобань, які міг би мати кожен делегат із множини D. Позначимо число, що стоїть на перетині i-го рядка та j-го стовпця, через a_{ij}, $1 \le i \le n$, $1 \le j \le n+1-i$ (тут ми покладаємо a_{1n} рівним загальній кількості делегатів у множині D, тобто проводимо фіктивний додатковий n-й раунд, у якому всі голосують за єдину заявку, що лишилася). У цих позначеннях рівно у a_{11} делегатів першою у списку стоїть заявка 1, у a_{21} делегатів першою у списку стоїть заявка 2, …, у a_{n1} делегатів першою у списку стоїть заявка n. Після першого раунду голоси тих, у кого першою в списку була заявка n, перерозподіляються на користь інших заявок. Інакше кажучи, другою в списку у кожного делегата, що голосував за n-ту заявку, маємо поставити одну з перших $n-1$ заявок, а саме: у $a_{12}-a_{11}$ із цих делегатів другою в списку буде стояти перша заявка, у $a_{22}-a_{21}$ із них другою в списку буде друга заявка, …, у $a_{(n-1)2}-a_{(n-1)1}$ із них другою в списку буде $(n-1)$-ша заявка. Аналогічно після другого раунду голоси тих, у кого останньою на даний момент у списку була заявка $n-1$, перерозподіляються на користь перших $n-2$ заявок. Тобто наступною в списку у кожного делегата, що голосував за $(n-1)$-шу заявку, маємо поставити одну з перших $n-2$ заявок, а саме: у $a_{13}-a_{12}$ із них наступною в списку буде стояти перша заявка, у $a_{23}-a_{22}$ із них наступною в списку буде друга заявка, …, у $a_{(n-2)3}-a_{(n-2)2}$ із них наступною в списку буде $(n-2)$-га заявка. Продовжуємо такі самі операції далі. На останньому кроці, після $(n-1)$-го раунду, $a_{2(n-1)}$ голосів «перерозподіляються» між єдиною заявкою, що залишилася, — тобто наступною у списку у кожного делегата, що голосував за другу заявку, маємо поставити першу заявку.

Звичайно, у такий спосіб ми не обов'язково повністю заповнимо списки кожного делегата. Наприклад, ті, хто голосував за першу заявку, починаючи вже з першого раунду, голосуватимуть за неї до кінця, і, що йде у їхніх списках після першої заявки, ми не визначимо. Утім, у кінець кожного незаповненого списку ми можемо дописати решту заявок (яких іще не було у списку) в довільному порядку, не змінивши результатів голосування в жодному з раундів. Лему доведено.

Отже, задачу можна звести до такої: знайти таблицю T, числа в комірках якої не перевищують відповідні числа початкової таблиці, у кожному рядку числа йдуть у порядку неспадання, а сума чисел у кожному стовпці однакова і якомога більша. Така сума (чисел одного стовпця) і є шуканим значенням m. За допомогою ж алгоритму, наведеного в доведенні леми, з таблиці T можна відновити і приклад уподобань усіх m послідовних делегатів. Залишається знайти відповідну таблицю T.

Спершу замінимо у початковій таблиці кожне число на найменше серед чисел, що стоять праворуч від нього, і його самого. Отриману таблицю позначимо через U. У таблиці U числа в рядках ідуть у порядку незменшення. За побудовою числа в комірках шуканої таблиці T не можуть перевищувати відповідні числа таблиці U, тому значення m не може бути більшим за суму чисел довільного стовпця таблиці U, зокрема й за найменшу з таких сум. Насправді найменша із цих сум і є шуканим значенням m: на базі таблиці U нам вдасться побудувати таблицю T так, щоб кожен стовпець таблиці T мав суму m. Для цього спершу, якщо потрібно, довільно зменшимо числа в першому стовпці таблиці, щоб сума чисел у ньому стала рівною m. Далі, якщо потрібно, зменшимо числа у другому стовпці так, щоб вони не стали меншими за своїх сусідів зліва, але їхня сума також виявилася рівною m (це можна зробити, адже якщо зменшити всі числа максимально, тобто так, щоб вони стали рівними своїм сусідам зліва, то сума чисел другого стовпця не буде перевищувати суму першого стовпця, тобто числа m). Далі зменшимо у такий же спосіб величини у третьому стовпці, не порушивши монотонності чисел у рядках і досягнувши того, що сума вже й у третьому стовпці стане рівною m. Продовжуватимемо цю операцію, поки сума не стане однаковою і рівною m в усіх стовпцях. Отримаємо таблицю T.

Додамо, що за умови оптимальної реалізації наведений розв'язок дає відповідь за $O(n(n+m))$ часу. Разом із тим у контексті олімпіади задача була розрахована більше на ідейне розв'язання і мала відносно невеликі обмеження, тож повний бал можна було набрати, не зосереджуючись занадто на деталях реалізації.

Насамкінець зауважимо, що в дійсності процедура голосування за місто, що прийматиме Олімпійські ігри, має незначні відмінності від описаної в задачі: по-перше, у раунді не можуть брати участь делегати, що представляють країни, заявки яких беруть у цьому раунді участь; по-друге, якщо якась із заявок набере понад 50 % голосів у довільному раунді, голосування припиняють достроково; по-третє, у разі рівності голосів у кількох заявок, що претендують на виліт, між ними проводять додатковий проміжний раунд. Пропонуємо читачеві самостійно подумати над тим, як саме ці зміни впливають на задачу визначення найбільшої підмножини послідовних делегатів.

3. Олімпійське містечко (автор — Данило Мисак)

Заробити близько 40 балів можна за допомогою перебору: перебираючи або варіанти розселення, або підмножини спортсменів, побажання яких потрібно задовольнити. Одним же з підходів, що дозволяє за-

робити повний бал, є так званий жадібний. Його ми розглядаємо нижче.

Передусім, *оптимальним* варіантом розселення назвемо такий, що забезпечує найбільшу кількість задоволених спортсменів. Оптимальних варіантів розселення може бути й декілька різних (тоді всі вони задовольняють однакову й максимально можливу кількість спортсменів). Пару спортсменів називатимемо *взаємною*, якщо ці спортсмени хочуть поселитися одне з одним.

Лема 1. *Серед оптимальних варіантів розселення є такий, за якого кожна взаємна пара спортсменів оселяється разом.*

Доведення. Розглянемо довільний оптимальний спосіб розселення спортсменів. Нехай деяка взаємна пара живе окремо. Тоді один зі спортсменів a з цієї пари живе з деяким спортсменом x, а інший спортсмен b з цієї пари живе разом із y. Очевидно, жодна з пар (a, x) та (b, y) не може бути взаємною, а разом у цих парах щонайбільше два задоволених спортсмени (x та y). Помінявши місцями спортсменів x та b, ми дістанемо пари (a, b) та (x, y), серед яких є вже принаймні одна взаємна і в яких, відповідно, кількість задоволених спортсменів не менша за два. У такий спосіб ми збільшили кількість взаємних пар, які живуть разом, не порушивши оптимальності розселення. Повторивши подібне достатню кількість разів, одержимо оптимальне розселення, у якому всі взаємні пари живуть разом. Лему доведено.

Отже, можемо поселити всі взаємні пари спортсменів одне з одним і відкинути їх. Тепер, звичайно, у деяких спортсменів може не залишитися бажаного співмешканця, але в іншому задача відповідає початковій. Тому коректним є говорити про оптимальні розселення для решти спортсменів, тобто для підмножини спортсменів, яких ми ще не відкинули. Спортсмена, з яким ніхто (серед тих, хто залишився) не хоче жити, будемо називати *непопулярним*.

Лема 2. *Якщо серед спортсменів, яких іще не відкинуто, є непопулярний і того, з ким він хоче жити, також не відкинуто, то існує оптимальне розселення, у якому побажання цього непопулярного спортсмена задоволено.*

Доведення. Як і в доведенні попередньої леми, розглядатимемо довільний оптимальний спосіб розселення спортсменів. Хай непопулярний спортсмен a живе з деяким спортсменом x, а спортсмен b, з яким хоче жити a, живе з y. У парі (a, x) немає жодного задоволеного спортсмена, а в парі (b, y) такий щонайбільше один: усі взаємні пари вже виключено. Помінявши x та b місцями, ми дістанемо пари (a, b) та (x, y), у яких задоволеним є принаймні один спортсмен a, і, отже, не порушимо оптимальності розселення. Лему доведено.

Отже, якщо серед спортсменів, які залишилися, є непопулярний, можемо поселити його з тим, з ким він хоче жити, і, відкинувши цю пару, повторити міркування. А якщо того, з ким хоче жити непопулярний спортсмен, уже відкинуто, то можемо відкинути лише самого цього непопулярного спортсмена: незалежно від того, з ким він зрештою житиме, ні він, ні його співмешканець задоволеними не будуть. Після певної кількості відкидань настане момент, коли серед тих спортсменів, що залишилися, не знайдеться непопулярних. Пропонуємо читачеві самостійно пересвідчитись, що це можливо лише за таких обставин: або просто не залишиться жодного спортсмена (тобто всіх уже буде розселено), або спортсмени, які залишаться, утворюватимуть один чи кілька циклів: цикл — це ситуація, коли a_1 хоче жити з a_2, a_2 хоче жити з a_3, …, a_{k-1} хоче жити з a_k, а a_k хоче жити з a_1. У циклі довжини k неможливо задовольнити більше ніж $[k/2]$ спортсменів; разом із тим задовольнити рівно $[k/2]$ можна тривіально. Вирахувавши для кожного циклу таку величину і склавши їх усі з кількістю раніше задоволених (і відкинутих) спортсменів, дістанемо остаточну відповідь.

Неоптимальна реалізація поданої схеми розв'язування (або інший алгоритм із часом виконання $O(n^2)$) набирає від 70 балів. Щоб здобути повний бал, потрібно для кожного спортсмена зберігати у спеціальному лічильнику кількість охочих із ним жити. Після відкидання спортсмена лічильник того, з ким він хотів жити, треба зменшувати на одиницю, а якщо внаслідок цієї операції лічильник став нульовим, тобто спортсмен став непопулярним, — додавати його у спеціальну чергу для подальшого розгляду й відкидання. Це дозволить зробити час виконання алгоритму лінійним.

Зауважимо, що існує й альтернативний спосіб розв'язати задачу за лінійний час.

4. Перегони (автор — Дмитро Кордубан)

Розглянемо спочатку інтуїтивно простіший випадок $b \le a$. Якщо оптимальний план харчування передбачає споживання батончиків до початку перегонів, то їх споживання можна пересунути вперед на момент початку перегонів. Вимоги щодо енергозабезпечення не буде порушено, бо $b \le a$. Так і зробимо. Для того щоб задовольнити початкову потребу у потужності p_0, потрібно спожити щонайменше (p_0/a) батончиків у момент 0 (якщо частка не є цілою, потрібно округлити до найближчого цілого, що не менше за цю частку). Споживати більше немає сенсу, бо зсунувши споживання додаткових батончиків на 1 секунду у майбутнє, ми можемо лише виграти, бо $b \le a$.

Після розгляду першої секунди перегонів отримуємо задачу для $(n-1)$ секунд з модифікованими (зменшеними) величинами p (бо частину потреб у перші t_1+t_2 секунд вже компенсовано). Отже, розв'язання зведено до використання жадібного алгоритму: рухаючись від початку до кінця перегонів, у кожен момент часу споживати найменшу кількість батончиків, що покривають поточні потреби з урахуванням того, що було спожито раніше.

У випадку $b > a$ розглянемо час у зворотному напрямку.

Викладену ідею можна реалізувати за лінійний час. Спочатку перейдемо до величин $q_j = p_j - p_{j-1}$, так що $p_j = q_0 + q_1 + \ldots + q_j$. Тоді споживання батончика у момент k призводить до зміни лише трьох таких чисел q_j при $j = k,\ k+t_1,\ k+t_1+t_2$. Розглядаючи масив q у порядку зростання індексу, ми завжди знатимемо поточну потребу у потужності.

Структура даних із швидкими запитами на відрізку (дерево відрізків, «коренева ідея» тощо) також дає можливість вибрати всі бали.

(Далі буде)