Задачі та розв'язки XX Всеукраїнської олімпіади з інформатики

Бондаренко В.В., Галковський Т.О., Стасюк А.В, Ткачук В.В, Ягіяєв Ш.І.

1 Завдання першого туру

1.1 «І знову ланцюг» (Андрій Стасюк)

Є ланцюг, що складається з N кілець. Яку мінімальну кількість кілець потрібно розчепити, щоб зі шматків, що залишилися, можна було б зібрати ланцюг будь-якої довжини від 1 до N кілець? При утворенні нових ланцюгів можна використовувати кільця, які було розчеплено.

Наприклад, при $N = 21$, достатньо розчепити всього 2 кільця таким чином, щоб утворилися шматки довжиною 3, 5 та 11. Два кільця, які було розчеплено, вважаються за шматки одиничної довжини.

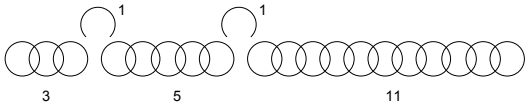


Рис. 1: Розірваний ланцюг

Завдання Напишіть програму `CHAIN`, що за натуральним N — довжиною вихідного ланцюга, знаходить мінімальну кількість кілець, які необхідно розчепити задля досягнення описаної цілі.

Вхідні дані Єдиний рядок вхідного файлу `CHAIN.DAT` містить одне ціле число N ($1 \leq N \leq 10^9$).

Вихідні дані Єдиний рядок вихідного файлу `CHAIN.SOL` має містити одне ціле число — знайдену мінімальну кількість кілець.

Приклад вхідних та вихідних даних

CHAIN.DAT
21

CHAIN.SOL
2

1.2 «Дерева у саду» (Володимир Ткачук)

Центральний сад країни Олімпія настільки великий, що один садівник не в змозі його обслуговувати. Було прийнято рішення розділити сад на дві частини. Певні дерева буде віднесено до першої частини, а решта — до другої. Одна з частин саду може залишитися порожньою.

Між кожною парою дерев у саду протоптані стежки. Коли садівники ідуть від дерева до дерева вони обов'язково ідуть стежкою, що з'єднує безпосередньо ці два дерева. Довжина стежки однакова при переміщенні в обидва боки.

Для спрощення роботи садівникам, розділення вирішили провести так, щоб довжина найдовшої стежки між парою дерев, що належать до однієї і тієї ж самої частини, була мінімальна.

Завдання Напишіть програму `TREES`, що за інформацією про довжини стежок між усіма парами дерев знаходить довжину найдовшої стежки між деревами з однієї частини саду, при оптимальному розділенні саду на частини.

Вхідні дані Перший рядок вхідного файлу `TREES.DAT` містить ціле число N ($2 \leq N \leq 1000$) — кількість дерев у саду. Кожен i-ий з наступних $N-1$-го рядків містить $N-i$ чисел, які послідовно представляють довжини стежок між i-им деревом і деревами з $i+1$-го до N-го. Всі числа цілі, невід'ємні, та не перевищують 10^6.

Вихідні дані Єдиний рядок вихідного файлу `TREES.SOL` має містити одне ціле число — мінімальну для всіх можливих розбиттів саду довжину найдовшої стежки між деревами в одній з частин саду.

Приклад вхідних та вихідних даних

TREES.DAT	TREES.SOL
3 1 5 1	1

1.3 «Доставка» (Андрій Гриненко)

Місто Прямий Ріг являє собою одну пряму вулицю. У місті працює компанія, що займається доставкою товарів. Для зручності, адреси доставки надані у вигляді чисел, що задають відстань від офісу компанії. Додатні числа в один бік, а від'ємні — в інший. Замовлення на доставку виконуються компанією послідовно, у тому порядку, в якому вони були задані.

В компанії працює два кур'єра. На початку робочого дня замовлення розподіляються між ними, і кожен вирушає за своїм маршрутом. Компанії необхідно так спланувати розподілення замовлень, щоб сумарна відстань, яка буде пройдена кур'єрами на момент виконання останнього замовлення, була мінімальною.

Завдання Напишіть програму `ORDER`, що за відстанями адресатів від офісу компанії знаходить найменшу сумарну відстань, що пройдуть її робітники.

Вхідні дані Перший рядок вхідного файлу `ORDER.DAT` містить ціле число N ($1 \leq N \leq 100\ 000$) — кількість замовлень. Далі слідує N рядків, кожен з яких містить єдине ціле число — відстань від офісу до адресата. Якщо відстань додатня, то адресат знаходиться у одній частині міста відносно офісу компанії, а якщо від'ємна — то у іншій. Відстані за модулем не перевищують 10^8.

Вихідні дані Єдиний рядок вихідного файлу `ORDER.SOL` має містити одне ціле число, яке є мінімальною можливою сумарною відстанню, що пройдуть обидва робітники компанії.

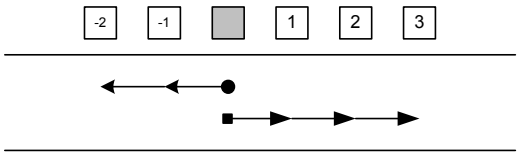


Рис. 2: План доставки

Приклад вхідних та вихідних даних

ORDER.DAT	ORDER.SOL
5 1 -1 2 -2 3	5

2 Завдання другого туру

2.1 «Розділення багатокутників» (Богдан Рубльов, Тарас Галковський)

На площині задано дві фігури, що обмежені опуклими багатокутниками. Фігури розташовані таким чином, що їх вершини не співпадають, а контури мають рівно 2 точки перетину.

Довільним чином розділимо площину прямою. Будемо вважати, що півплощина з одного боку прямої відповідає першій фігурі, а з іншого боку — другій фігурі. Визначимо поняття дефекту розділення — сума площі першої фігури, що розташована на півплощині другої фігури, та площі другої фігури, що розташована на півплощині першої фігури. З двох можливих відповідностей півплощин до фігур оберемо таку відповідність, де значення дефекту менше.

Наприклад, на рисунку зображена пряма, що задає певне розділення багатокутників. Оцінка дефекту цього розділення (два випадки відповідності): $(D) + (C + E)$ та $(A + D) + (B + C)$. З рисунку, $D + C + E$ менше, отже, загалом, оцінка розбиття дефекту розділення утвореного цією прямою є $D + C + E$.

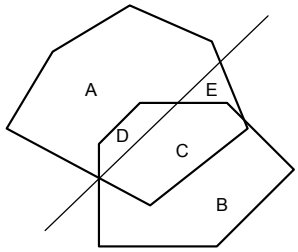


Рис. 3: Приклад розділення багатокутника

Завдання Напишіть програму `DIVIDE`, що за заданими двома багатокутниками знаходить пряму, що утворює розділення з найменшим дефектом.

Вхідні дані Перший рядок вхідного файлу `DIVIDE.DAT` містить одне ціле число $N (3 \leq N \leq 20)$ — кількість вершин першого багатокутника. Наступні N рядків містять пари цілих чисел — координати вершин першого багатокутника у порядку обходу. $(N+2)$-й рядок файлу містить число M $(3 \leq M \leq 20)$ — кількість вершин другого багатокутника. Наступні M рядків містять його координати задані таким же чином, як і для першого багатокутника. Координати точок додатні та не перевищують 1000.

Вихідні дані Єдиний рядок вихідного файлу `DIVIDE.SOL` має містити дві пари чисел — координат двох точок, що однозначно задають розділення (пряму) з найменшим дефектом. Числа потрібно виводити у порядку: x_1 y_1 x_2 y_2. Координати потрібно виводити з точністю 10^{-3}. Координати точок мають бути додатні та не більші за 1000.

Приклад вхідних та вихідних даних

DIVIDE.DAT
3 2 1 3 3 4 1 3 5 2 3 2 4 3

DIVIDE.SOL
2 5 4 1

2.2 «Мудреці» (Андрій Стасюк)

В королівстві Олімпія живуть N мудреців, що мають досконалі інтелектуальні здібності. Для підтримання творчого настрою серед інтелектуального осередку королівства король ввів таку традицію.

Кожен вечір M ковпаків розфарбовуються в один з K різних кольорів. Таким чином M_i ковпаків розфарбовано у i-й колір ($i = 1 \dots K$). Мудрецям ці дані відомі. Поки мудреці сплять, кожному з них на голову чіпляють один з ковпаків, а решту ховають. Коли мудреці прокидаються, вони сідають у круг, так щоб кожен з них бачив ковпаки усіх інших, і починають думати про колір свого ковпаку. Це відбувається таким чином. Кожен мудрець пише на папірці, чи знає він колір свого ковпака. Після того усі демонструють свої папірці. Якщо хтось написав, що знає колір — процедура закінчується, та усі йдуть на сніданок. Якщо ніхто не зміг визначити колір, то мудреці починають думати знову, операція з папірцями повторюється.

Завдання Напишіть програму `WISEMEN`, що за інформацією про колір усіх ковпаків та розподіленням ковпаків серед мудреців визначає які з них першими визначать колір свого ковпака та скільки разів для цього потрібно буде демонструвати папірці, або встановить, що ніхто не зможе цього зробити.

Вхідні дані Перший рядок вхідного файлу `WISEMEN.DAT` містить цілі числа N ($1 \le N \le 10^6$) — кількість мудреців, M ($N \le M \le 10^6$) — кількість ковпаків, та K ($1 \le K \le 10^6$) — кількість різних кольорів. Другий рядок містить K цілих чисел, кожне з яких визначає кількість ковпаків певного кольору. Третій рядок складається з N цілих чисел, що являють собою кольори ковпаків кожного з мудреців. Кольори пронумеровані від 1 до K.

Вихідні дані Перший рядок вихідного файлу `WISEMEN.SOL` має містити впорядковані за зростанням номери мудреців, що першими дізнаються кольору свого ковпака. Другий рядок має містити одне число, що визначає, скільки разів для цього будуть демонстровані папірці. Якщо дізнатися про колір неможливо, то єдиний рядок вихідного файлу має містити цифру 0 (нуль).

Приклади вхідних та вихідних даних

WISEMEN.DAT
3 5 2 3 2 1 1 2

WISEMEN.SOL
1 2 2

Пояснення. Нехай в королівстві всього три мудреця, два з яких мають білий ковпак (колір 1) та один - чорний (колір 2). При цьому ще один білий та один чорний ковпак сховані. На першому етапі ніхто не зможе визначити колір свого ковпаку. На другому етапі кожен з володарів білого ковпаку буде розмислювати так: «Якщо б я мав чорний ковпак, то колега, що має білий, здогадався б про колір свого ковпаку ще на першому кроці, але ж цього не відбулося, тому мій ковпак білий!». А володар чорного ковпака все ще не зможе визначитися. Тобто, першими здогадаються мудреці 1 та 2 з білими ковпаками і папірці для цього будуть продемонстровані два рази.

2.3 «Живопис» (Володимир Ткачук)

В країні Олімпія дуже розвинутий живопис. Картиною вважається будь-який прямокутник, що складається з чорних і білих одиничних квадратів. Художник Олімпус вирішив радикально покращити свої картини. Для цього він планує до білого та чорного кольорів додати ще й сірий відтінок. За його задумом, границя між кожними чорним і білим квадратом має містити сіру лінію, щоб утворився ефект плавного переходу.

Але, перед початком роботи, він визначив, що сіра фарба дуже дорого коштує. Щоб заощадити гроші художник вирішив оцінити, чи не вигідніше спочатку перефарбувати деякі білі квадрати в чорні, а чорні в білі для того, щоб мінімізувати витрати на фарбу.

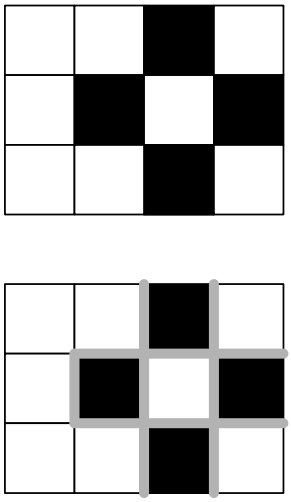


Рис. 4: Приклад розділення багатокутника

Завдання Напишіть програму `DRAWING`, що за інформацією про існуючу картину визначить мінімальну суму грошей, що знадобляться на її покращення.

Вхідні дані Перший рядок вхідного файлу `DRAWING.DAT` містить п'ять натуральних чисел N, M, w, b, g. $1 \le N, M \le 70$ — висота та ширина картини, $1 \le w, b, g \le 1000$ — ціна малювання одного білого одиничного квадрата, чорного одиничного квадрата та сірої лінії довжиною одиниця, відповідно. Далі слідує N рядків, кожен з яких складається з M літер. Літера `B` відповідає чорному квадрату, а `W` — білому.

Вихідні дані Єдиний рядок вихідного файлу `DRAWING.SOL` має містити одне ціле число, яке є мінімальною сумою витрат на покращення картини.

Приклади вхідних та вихідних даних

DRAWING.DAT	DRAWING.SOL
3 2 3 3 2 BB WW WB	7

Пояснення: необхідно перефарбувати нижню праву чорну клітку у білу.

3 Розв'язки першого туру

3.1 «І знову ланцюг»

Перший підхід Розглянемо зворотню задачу. За заданим M — кількістю кілець, які буде розчеплено, знайдемо таке максимальне N, що з отриманих шматків ланцюга можна буде зібрати ланцюг будь-якої довжини від 1 до N кілець.

Якщо розчеплено M кілець, то маємо $(M + 1)$ шматок ланцюга, та, власне, кільця, що було розчеплено — M шматків одиничної довжини. Очевидно, що з цих кілець можна зібрати ланцюги будь-якої довжини від 1 до M. Щоб зібрати ланцюг довжиною $(M + 1)$, потрібно використати ще один шматок довжиною не більше $(M + 1)$. Нехай використано шматок довжиною K $(1 \le K \le M+1)$. Якщо цей шматок буде присутній у наборі, з нього та розчеплених кілець можна зібрати ланцюг будь-якої довжини від 1 до $(M + K)$. Чим довший шматок буде використано, тим довший ланцюг можна зібрати. Тому зрозуміло, що оптимально використовувати шматок довжини $(M + 1)$ — максимально можливої.

З використанням шматка довжини $(M+1)$ та M розчеплених кілець можна зібрати ланцюг будь-якої довжини від 1 до $(2M+1)$. Щоб зібрати ланцюг довжиною $(2M+2)$ потрібно мати ще один шматок довжиною не більше $(2M+2)$. Аналогічно, оптимальніше використати шматок довжини саме $2(M+1)$.

Якщо є два шматки довжинами, відповідно, $(M+1)$ та $2(M+1)$ і M розчеплених кілець, з цього набору можна зібрати ланцюг будь-якої довжини від 1 до $4(M+1)-1$. Для утворення ланцюга довжини $4(M+1)$ найкраще взяти ще один шматок довжини саме $4(M+1)$ і так далі.

Загальна кількість шматків — $(M+1)$, а розчеплених кілець — M. Звідси отримаємо формулу для обчислення N:

$$N = (M+1) + 2(M+1) + 4(M+1) + \ldots + 2^M(M+1) + M =$$

$$= (1 + 2 + 4 + \ldots + 2^M)(M+1) + M = (2^{M+1} - 1)(M+1) + M = 2^{M+1}(M+1) - 1$$

Залишається підібрати найменший аргумент M наведеної формули, при якому її значення буде дорівнювати або перевищувати N, вказане в умові. Найпростіше це зробити перебором значень M у циклі, паралельно підраховуючи степені двійки. Оскільки M пропорційно $\log_2 N$, складність такого алгоритму буде $O(\log N)$. Якщо значення формули для різних M (не більших $\log_2 N$) підрахувати заздалегідь та зберегти у масиві констант, а для підбору шуканого значення M використовувати бінарний пошук, складність алгоритму зменшиться до $O(\log\log N)$, що й буде оптимальним розв'язком.

Другий підхід Розв'яжемо пряму задачу. За заданим N знайдемо кількість кілець, що треба розчепити. Спочатку визначимо, якої довжини буде найдовший шматок ланцюга. Для цього розчепимо K-те кільце, отримавши шматок α довжиною $(N-K)$, шматок β довжиною $(K-1)$, який, можливо, буде розчеплено далі, та власне розчеплене кільце. Тоді для збирання ланцюгів, коротших за α, будемо використовувати кільця та шматки, отримані подальшим розчепленням шматка β; а для збирання ланцюгів більшої довжини, додамо до них шматок α.

Нехай поточне M буде зберігати кількість кілець, що вже розчеплено, тобто на даному етапі дорівнюватиме одиниці. Якщо різниця між довжинами шматків буде більшою за $(M+1)$, то ми жодним чином не зможемо зібрати з отриманого набору ланцюги довжиною більшою за $(K+M-1)$ та меншою за $(N-K)$. Тобто $(N-K)-(K-1) \le M+1$. Звідси можемо знайти умову для K: $K \ge (N-M)/2$.

Але чим довшим буде шматок α, тим коротшим буде шматок β, що залишається для подальшого розчеплення, та, очевидно, що кількість кілець, які буде розчеплено, не збільшиться. Для цього мінімальним повинне бути K.

Розглянемо шматок, що залишився — β. Його довжина дорівнює $(K-1)$, а K, як слідує з наведених міркувань, дорівнює мінімальному значенню, що задовільняє умові, тобто $\lceil (N-M)/2 \rceil$. Якщо довжина шматка β, $(\lceil (N-M)/2 \rceil - 1)$, не більша за $(M+1)$, розчіпляти його далі не потрібно. Якщо ж ця умова не виконується, треба продовжувати процес, прийнявши за N довжину шматка β.

Тобто розв'язок зводиться до моделювання процесу — послідовного розчеплення ланцюга зі зберіганням кількості кілець, що розчеплено. Наприкінці роботи алгоритму отримане M і буде відповіддю, отже кожен раз шматок ланцюга зменшуватиметься принаймні вдвічі, складність алгоритму дорівнюватиме $O(\log N)$.

3.2 «Дерева»

Перший підхід Ідея підходу полягає в розв'язанні наступної підзадачі. Перевірити, чи можна розділити дерева на дві частини так, щоби відстань між будь якими двома деревами в одній частині була не більша деякого наперед заданого числа X. Тобто, спочатку вибираємо число, а потім для нього перевіряємо, чи можна так розділити сад.

Перед тим як детальніше розглянути підзадачу, покажемо, як вона пов'язана з розв'язком початкової задачі. Відповіддю на задачу буде мінімальне число для якого ще можна розділити сад.

Також, очевидно, що відповідь буде дорівнювати відстані між деякою парою дерев, а значить буде належати діапазону від найкоротшої до найдовшої з відстаней. Таким чином, її можна шукати за допомогою двійкового пошуку (binary search) по діапазону, або ж по впорядкованому масиву довжин всіх стежок.

Перевірка можливості розбиття Тепер повернемося до підзадачі. Нехай число X задано і треба перевірити, чи можна розбити сад на дві частини. Це рівносильне перевірці на дводольність графу, в якому вершинам відповідають дерева, а ребра проведені між тими вершинами, відстань між якими більша за X. Якщо такий граф є дводольним, то розбиття саду можливе, інакше — ні. Перевірку можна виконати за допомогою одного пошуку в глибину (DFS).

Оцінимо складність роботи: одна перевірка потребує $O(N^2)$ часу, де N — кількість дерев. Якщо в розв'язку використовувати двійковий пошук по діапазону, то загальна складність буде $O(N^2 \log\ D)$, де D — різниця між найкоротшою і найдовшою відстанями. Пошук по впорядкованому масиву ребер дає складність $O(N^2 \log\ N)$.

Другий підхід Підхід полягає в побудові такого розбиття, для якого відстань між деревами з однієї частини саду буде мінімальною.

Коротко алгоритм можна описати так: беремо найбільшу відстань і намагаємося зробити так, щоб дерева між якими вона пролягає потрапили в різні частини саду. Потім беремо другу за величиною відстань і т.д. Якщо для якоїсь відстані неможливо зробити так, щоб дерева між якими вона пролягала потрапили в різні частини саду, то ця відстань є шуканою відповіддю.

Такий розв'язок вимагає використання ефективних структур даних. Автор використовував систему множин, що не перетинаються.

Будемо вважати, що кожне дерево, належить якійсь множині (і тільки одній). Дерева, що належать одній множині, мають бути віднесені до однієї частини саду. Також для кожної множини ми знаємо «альтернативну» множину — множину тих дерев, що обов'язково мають належати іншій частині саду.

Спочатку кожне дерево належить своїй множині, а всі альтернативні множини пусті. Тобто є N множин по одному дереву в кожній і N пустих альтернативних множин. Далі алгоритм буде об'єднувати множини, слідкуючи щоби відстані всередині однієї множини були мінімальними.

Впорядкуємо всі відстані за спаданням і будемо послідовно перебирати їх всі від найбільшої до найменшої. На кожному кроці, в нас є два варіанти:

1. Якщо дерева між якими така відстань належать одній множині, то ці дерева попадуть в одну частину саду а значить ця відстань є шуканою величиною. Робота алгоритму на цьому завершується.

2. Якщо дерева належать різним множинам U_1 і U_2, а V_1 і V_2 — їх альтернативні множини, то можна гарантувати, що відповідь буде не більша за відстань, якщо ми помістимо ці дерева в різні (альтернативні) множини. Тобто введемо множину $C_1 = U_1 \cup\ V_2$ і $C_2 = U_2 \cup\ V_1$. Множини C_1 і C_2 — альтернативні одна до одної. Переходимо далі до наступної відстані.

Такий розв'язок вимагає ефективної реалізації системи множин, що не перетинаються. В книзі «Алгоритмы построение и анализ» описані структури данних, зо можуть забезпечити виконання проходу по всім відстаням за $O(N^2)$. Тому, основний час буде затрачений на впорядкування масиву відстаней.

В результаті, отримуємо загальну складність алгоритму $O(N^2 \log N)$.

3.3 «Доставка»

Динамічне програмування $O(N^2)$ Розглянемо оптимальний план відвідування робітниками пунктів доставки. За цим планом пункт N відвіданий одним з двох робітників. Інший же робітник, після виконання цього плану, залишився в деякому пункті j ($j < N$).

Будемо вважати, що пара чисел (i, j), де $i > j$ відповідає всім планам обслуговування пунктів з 1-го по i-ий такі, що після завершення виконання такого плану, один робітник знаходиться у пункті i (тільки що виконав доставку до i-го пункту), а інший у пункті j.

Введемо функцію $f(i, j)$ яка дорівнює вартості найоптимальнішого плану з множини планів (i, j). Для обчислення її значення з'ясуємо вигляд кроків, що приводить нас до класу (i, j). В загальному випадку, єдиним можливим кроком є перехід робітника з пункту $i-1$ до пункту i. Тоді попереднім є клас $(i-1, j)$. У цьому випадку значення функції пов'язані таким рекурентним співвідношенням:

$$f(i, j) = f(i-1, j) + \Delta(i-1, i), i > j+1$$

Окремо розглянемо випадок $i = j+1$, бо основна властивість пари $(i > j)$ порушується. Ми маємо можливість перевести деякого робітника з пункту k, де $k < j$ до пункту i, а в цей час робітник у пункті j «відпочиває». Формально це можна записати:

$$f(i, j) = \min_{k=1\ldots j-1} [f(j, k) + \Delta(k, i)], i = j+1$$

Запишемо загальну формулу для обчислення функції f:

$$f(i, j) = \{ \begin{array}{l} \max_{k=1\ldots j-1[f(j,k)+\Delta(k,i)]}, i = j+1 \\ f(i-1, j) + \Delta(i-1, i), i > j+1 \end{array}$$

Розв'язком поставленої задачі буде мінімум по всіх $j < N$, функції $f(N, j)$.

За цією формулою для обчислення всіх значень функції f необхідно використати час пропорційний $O(N^2)$.

Ефективний розв'язок $O(N \log N)$ Припустимо, що існує лише один робітник. Тобто всі пункти будуть відвідані ним у заданому порядку. Нехай функція $g(N)$ дорівнює сумарній відстані необхідній для виконання плану з N пунктів. За даним визначенням:

$$g(N) = g(N-1) + \Delta(i-1, i), g(0) = 0$$

Почнемо розв'язання задачі з цього плану, і будемо поступово його покращувати за допомогою другого робітника. Введемо функцію $benefit(i, x)$ — величину, на яку ми можемо покращити вихідний план для обслуговування пунктів $1 \ldots i$ при умові, що по закінченні виконання нового плану, один з робітників буде знаходитися в точці з координатою x (а інший в пункті з номером i).

Тоді за визначенням сумарна пройдена відстань у мінімальному плані може бути обчислена за формулою:

$$ans(N) = \min_{k=0\ldots N-1} [g(N) - benefit(N, X(k))] = g(N) - \max_{k=0\ldots N-1} benefit(N, X(k))$$

Обчислення значень функції $benefit(i, x)$ Визначимо початкові значення введеної функції: $benefit(0, 0) = 0$ та $benefit(i, x) = -\infty$.

Для обчислення значення функції, з'ясуємо множину положень, з яких ми можемо попасти у (i, x). Виберемо деякий пункт j $(j < i)$ з якого робітник прийде до пункту i. Цей крок додасть до загальної вартості плану доданок $\Delta(j, i)$; у той же час, у порівнянні з використанням лише одного робітника, який би пройшов усі пункти у заданому порядку, ми позбавляємося доданку $\Delta(i-1, i)$. Ці відносини можна відобразити формулою:

$$benefit(i, x) = \max\{ \begin{array}{l} benefit(j, x), j < i \\ \max_{j<i}[benefit(i-1, X(j)) + \Delta(i-1, i) - \Delta(j, i)] \end{array}$$

Розглянемо залежність між значеннями функції на поточному та попередньому кроці: між значеннями $benefit(i, *)$ та $benefit(i-1, *)$. З наведеного означення слідує, що ці функції мають різні значення, можливо, лише в точці $(i, X(i))(\ldots)$. Отже, на кожному кроці і ми будемо зберігати вектор значень $benefit(i, *)$, і оновлювати його значення відповідно до наведених формул.

Спростимо процес обчислення значень функції $benefit$: виносячи доданок, що не залежить від j за max, а модуль розкриваємо в залежності від знаку виразу під модулем:

$$\max_{j<i}[benefit(i-1,X(j))+\Delta(i-1,i)] = \Delta(i-1,i)+\max_{j<i}[benefit(i-1,X(j))-|X(i)-X(j)|] =$$

$$\Delta(i-1,i)+\begin{cases} \max_{j<i}[benefit(i-1,X(j))+X(j)]-X(j), X(i)\geq X(j) \\ \max_{j<i}[benefit(i-1,X(j))+X(j)]+X(j), X(i)\geq X(j) \end{cases}$$

Помітимо, що відповідати на запит $\max_{j<i}[benefit(i-1,X(j))+X(j)]$ необхідно максимально швидко (перебір всіх можливих j не підходить). З цією метою ми будемо зберігати дві структури: $benefit(i,X(j))+X(j)$ та $benefit(i,X(j))-X(j)$, замість одного масиву $benefit(i,*)$.

Нам необхідно швидко виконувати дві операції над цими масивами:

1. знайти значення максимуму на проміжку від i-го до j-го елементу масиву;
2. оновити структуру даних, записавши на i-ту позицію нове значення X.

Існують структури даних, що дозволяють виконувати ці операції за час, пропорційний $O(\log N)$. Розглянемо одну з таких структур – Максимі затор (задача RMQ: Range Minimum/Maximum Query).

Максимізатор *Максимізатором* будемо називати структуру даних, що виконує дві операції:

`MODIFY (pos, value)` — встановити значення `A[pos]` рівним `max(A[pos], value)`;

`FINDMAX (l, r)` — знайти максимальне значення в таблиці з індексами від l до r, тобто `max(A[l], A[l+1], ..., A[r-1], A[r])`.

Час роботи обох процедур не перевищує оцінки $O(\log N)$.

Деталі щодо конкретної реалізації такої структури містяться у багатьох підручниках з алгоритмів (наприклад, Кормен Т., Лейзерсон Ч, Рівест Р. Алгоритми: Побудова та Аналіз).

4 Розв'язки другого туру

4.1 «Розділення багатокутників»

Лема 1. *Перетин двох опуклих фігур є опуклою фігурою.*

Доведення. За означенням опуклості, $\forall x, y$, що належать фігурі відрізок (x, y) належить цій фігурі. Оскільки області перетину належать лише такі точки, що належать обом фігурам одночасно, то і відрізок, що з'єднує будь-яку пару точок з перетину також належать області перетину. Отже перетин задовольняє означенню опуклої фігури. □

Дефект розділення площини довільною прямою не може бути меншим за площу спільної частини багатокутників A та B: внезалежності від того, як будуть віднесені півплощини відносно довільної проведеної прямої, точки з перетину фігур завжди будуть враховані при обчисленні дефекту розділення (бо вони належать обом фігурам одночасно). А отже дефект розділення *не може бути меншим* за площу перетину заданих фігур.

Користуючись доведеними фактами, явно вкажемо метод побудови такої прямої, що буде задавати розділення з дефектом рівним площі області перетину двох багатокутників.

Оскільки контури фігур A та B перетинаються у двох точках, проведемо пряму через них. За означенням обчислимо дефект такого розділення, і переконаємося, що він строго дорівнює площі перетину.

Задача пошуку точок перетину двох багатокутників Фактично, нам необхідно знайти такі пари ребер даних багатокутників, що перетинаються (при чому одне ребро належить одному багатокутнику, а друге — іншому). Для цього будемо перебирати всі пари ребер, перевіряючи при цьому чи перетинаються вони. Якщо — так, виводимо координати точок перетину у вихідний файл. Зважаючи на розмір вхідних даних цей розв'язок, час роботи якого пропорційний $O(N^2)$, задовольнить обмеження по часу.

4.2 «Мудреці»

Використаємо для розв'язання математичну індукцію.

Базове твердження S(0) Нехай на деякому мудреці ковпак кольору A. Якщо він бачить на своїх колегах всі ковпаки всіх кольорів, крім кольору A — він одразу здогадується про колір свого ковпака, тому що всі ковпаки інших кольорів вже зайняті. Назвемо цей набір ковпаків тривіальним для мудреця та будемо казати, що йому для відгадування потрібен один показ папірців.

Перехід $S(0) \Rightarrow S(1)$ Нехай мудрець бачить на своїх колегах тривіальний набір ковпаків, в якому один з ковпаків не кольору A замінений на ковпак кольору A. Очевидно, що в такому разі під час першого показу папірців він не зможе здогадатися про колір свого ковпаку. Але під час наступного показу він буде міркувати наступним чином:

«Якщо б на мені був такий ковпак, якого не вистачає до тривіального набору, то той мудрець, на якому тепер ковпак кольору A, під час попереднього показу обов'язково здогадався б, який на ньому ковпак. Але цього не відбулося, тому на мені не може бути такий ковпак. Всі ковпаки інших кольорів вже зайняті, тому на мені ковпак кольору A! »

Тобто при заміні одного ковпака для відгадування потрібно на один показ більше.

Перехід $S(L-1) \Rightarrow S(L)$ Припустимо, що у будь-якій ситуації, коли мудрець бачить на своїх колегах набір ковпаків, отриманий із тривіального заміною $(L-1)$ ковпаків не кольору A на $(L-1)$ ковпаків кольору A, для відгадування кольору йому потрібно L показів папірців.

Тоді, якщо у наборі замінено L ковпаків, мудрець не зможе здогадатися про колір свого ковпака за таку ж кількість показів. Але якщо за L показів ніхто не здогадався про колір свого ковпака, при наступному показі мудрець буде міркувати так:

«Якщо б на мені був будь-якого кольору крім A — його не вистачає до набору з $(L-1)$ заміненими ковпаками — то той мудрець, на якому тепер ковпак кольору A, під час L-го показу обов'язково здогадався б, який на ньому ковпак. Але цього не відбулося, тому на мені не може бути ковпак не кольору A. Значить, на мені ковпак кольору A.»

Тобто потрібно на один показ більше, а саме $(L+1)$, якщо у наборі замінено L ковпаків.

Висновок Якщо задані кількості ковпаків кожного кольору M_i і кількість мудреців N, та хоч при одному наборі ковпаків на своїх колегах мудрець може здогадатися про колір свого ковпака під час першого показу (цей набір ковпаків на колегах буде для нього тривіальний), то він може здогадатися і при будь-якому іншому наборі та для цього буде потрібна кількість показів, що на одиницю більша за кількість ковпаків, які відрізняють цей набір від тривіального.

Алгоритм При заданих кількостях ковпаків існує таке їх розподілення, що під час першого показу мудрець із кольором ковпака A може здогадатися про свій колір, якщо $M - M_A \leq N - 1$. Тобто він зможе побачити перед собою всі ковпаки всіх кольорів, крім свого. Для всіх кольорів A, що задовольняють цієї умови, підрахуємо кількість ковпаків, які відрізняють заданий набір ковпаків на колегах мудреця від тривіального. Це число — кількість ковпаків кольорів крім A, що не знаходяться на мудрецях у даний час, та її можна підрахувати за формулою $S = M - N - (M_A - P_A)$, де P_A — кількість ковпаків кольору A, одягнених на мудреців. Ті мудреці, для яких це число мінімальне здогадаються про колір свого ковпака першими та зроблять це за $(S+1)$ показ папірців.

Якщо P_A підрахувати заздалегідь, наприклад, під час читання вхідних даних, то перебрати всіх мудреців та всі кольори можна буде за $O(N+K)$, тобто алгоритм матиме лінійну складність.

4.3 «Живопис»

Розв'язувати дану задачу можна декількома шляхами, різними за складністю реалізації і часом роботи. Не беручи до уваги можливі евристики, розглянемо найбільш типові, з точки зору автора, розв'язки.

Повний перебір Всього можливо 2^{NM} варіантів перефарбування картини, а отже можна перебрати їх усі і для кожного визначити кінцеву вартість. Час роботи такого розв'язку є $O(NM \cdot 2^{NM})$, і він є неефективним вже при $NM > 20$.

Динамічне програмування «по краю» У випадку, коли $N = 1$ (одновимірна картина), задача може бути просто вирішена методом динамічного програмування. Проте, динамічне програмування можна використовувати і в двовимірному випадку при невеликих N.

Розв'язок полягає в послідовному знаходженні мінімуму вартості для картини фіксованої ширини і з фіксованим розфарбуванням останнього стовпчика. Підхід потребує $O(2^N)$ пам'яті і $O(M \cdot 2^{2N})$ часу (у випадку $N < M$). Межі застосування такого розв'язку визначаються найменшою з сторін картини: $\min(N, M) \leq 12$

Зведення задачі до знаходження максимального потоку Тепер розглянемо розв'язок що передбачався автором, як оптимальний для цієї задачі. Ідея полягає в зведенні задачі до еквівалентної, що може бути розв'язана відомими алгоритмами.

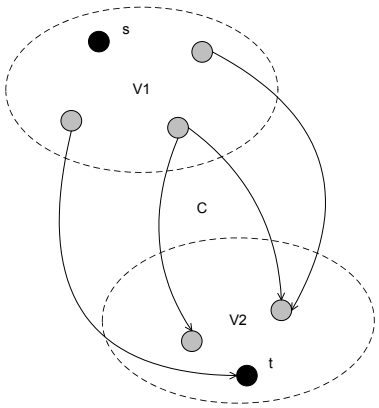
Отже, нам необхідно віднести кожен з одиничних квадратиків картини або до чорного або до білого кольору, і якщо спочатку квадратик був іншого кольору ми маємо врахувати штраф за перефарбування. Крім того, треба врахувати штраф за сусідні квадратики різного кольору. Задача полягає в мінімізації сумарного штрафу.

Серед відомих задач на пошук мінімуму звернемо увагу на задачу пошуку мінімального перерізу в мережі.

Нагадаємо, мережею називається орієнтований граф $G = (V, E)$ з двома виділеними вершинами: джерелом s (*source*) і стоком t (*target*) (V — множина вершин графу, E — множина ребер, $s, t \in V$). Крім того для кожного ребра $e \in E$ визначено число $c(e)$ — пропускна спроможність ребра (*capacity*). Перерізом мережі називається розбиття множини вершин на дві підмножини V_1 і V_2, $V = V_1 \cup V_2, V_1 \cap V_2 = \varnothing$. Причому, джерело завжди належить першій множині а стік другій: $s \in V_1, t \in V_2$. Пропускною спроможністю або вартістю перерізу C називається сума пропускних спроможностей тих ребер, що виходять з V_1 і входять у V_2 :

$$C = \sum_{i \in V_1, j \in V_2} c(i, j)$$

Приклад перерізу показано на рис.1.



Загальна кількість можливих перерізів — $2^{|V|-2}$. Отже, якщо побудувати граф в якому вершини відповідають одиничним квадратикам на картині і додати до нього дві спеціальні

вершини s і t, то отримаємо граф в якому кількість можливих перерізів співпадає з кількістю можливих кінцевих розфарбувань картини.

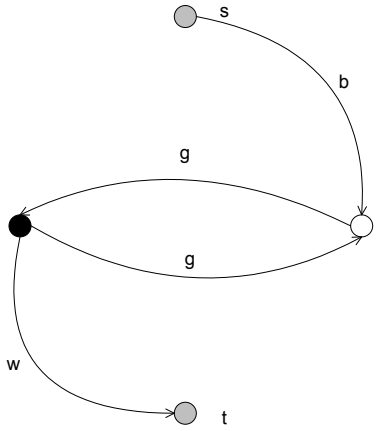
Встановимо взаємно однозначну відповідність між розфарбуваннями і перерізами — будемо вважати, що якщо квадратик в кінцевому розфарбуванні має білий колір, то відповідна йому вершина належить до V_1, якщо ж в чорний — до V_2.

Тепер залишилося провести ребра так, щоби штраф за кінцеве розфарбування картини дорівнював вартості відповідного перерізу. Зокрема, якщо якийсь квадратик був білий, а став чорний, то має сплачуватись штраф b. Для того, щоб це виконувалось, проведемо ребро з джерела до кожної з вершин, що відповідають білим квадратикам. Покладемо пропускну спроможність таких ребер рівною b — штраф за перефарбування в чорний колір. Тепер, якщо в кінцевому розфарбуванні білий квадратик став чорним, то відповідна йому вершина буде входити в V_2, а ребро з джерела до неї буде вносити свій вклад в вартість перерізу.

Аналогічно проводимо ребро з кожної вершини, що відповідає чорному квадратику, до стоку. Покладемо пропускною спроможністю цього ребра w — штраф за перефарбування в білий колір.

Нарешті врахуємо штраф за сусідні квадратики різного кольору. Для цього між кожними двома вершинами, що відповідають сусіднім квадратикам проведемо ребро пропускною спроможністю g — штраф за проведення сірої лінії.

На рис.2 показаний граф що відповідає картині 2x1 з одним чорним і одним білим квадратиком.



Отже, вище було показано, як побудувати мережу, вартість перерізу якої дорівнює штрафу за розфарбування картини. Очевидно, що мінімальний переріз відповідає мінімальному штрафу, а значить є шуканою відповіддю. За відомою теоремою Форда-Фалкерсона вартість мінімального перерізу в мережі з невід'ємними пропускними спроможностями дорівнює величині максимального потоку, що може бути пропущений через мережу. Для знаходження величини потоку можна скористатися відомими поліноміальними алгоритмами.

Отже, авторський розв'язок передбачає:

1. Побудову мережі з $NM + 2$ вершинами, що відповідає картині.
2. Знаходження максимального потоку в цій мережі.