**А.Гриненко, Ю.Знов'як, Д.Кордубан,
Д.Мисак, О.Рудик, В.Ткачук**

III (міський) етап олімпіади з ОІОТ у місті Києві у 2006 році

Передмова голови журі Олександра Рудика

Стаття містить умови завдань III (міського) етапу олімпіади з основ інформатики й обчислювальної техніки у місті Києві у 2006 році та авторські розв'язання цих завдань.

Проведення III етапу у 2006 році у місті Києві мало ряд суттєвих відмінностей у порівнянні з минулими роками.

2001–2005 роки — це час становлення і власне олімпіади, і відбірково-тренувальних зборів команди міста Києва до IV (Всеукраїнського) етапу *у їх сучасному вигляді.* У цей час завдання упорядковував один автор — голова журі, свідомо поставивши перед собою такі цілі.

1. Учасники олімпіади мають можливість використати здобуті знання, вміння й навички для нестандартних *для них* умов завдань. Зміст завдань має створювати й зміцнювати зацікавленість у опануванні прикладною математикою.
2. Значну частину завдань розв'язують за допомогою алгоритмів, які неможливо уникнути при повноцінному вивченні математичних понять з шкільного курсу математики. Такі завдання у подальшому можна використати у навчанні, що перетворює нестандартні для учнів задачі у суто *технологічні*. На I (шкільний) і II (районний) етап олімпіади бажано пропонувати орієнтовні завдання, які знайомлять з математичними поняттями і прийомами розв'язання найскладніших завдань III (міського) етапу.
3. Деякі завдання мають бути настільки алгоритмічно складні й „багатоешелоновані", щоб їх можна було використати на відбірково-тренувальних зборах навіть з наперед відомими завданнями. Мета таких тренувань: виховати в учнів наполегливість у створенні *повного* розв'язання, що гарантує перемогу на олімпіаді.
4. Потрібно максимально розкрити процес перевірки робіт учасників і перевіряти роботи у присутності самих учасників. Мета: відучити учасників від сліпої довіри до журі та процесу перевірки, привчити учнів та їхніх наставників використовувати всі можливості під час проведення апеляції. Всі тести мають бути такими, щоб їхню коректність можливо було перевірити без програмних засобів. Це сприяє навчанню учасників олімпіади самостійно тестувати свої програми.

При висуненні багатьох критеріїв не завжди можливо по кожному з них досягнути найкращого результату. Наприклад, відчувалось незадоволення тим, що всі бали на ІІІ (міському) етапі не здобувалися навіть призерами Міжнародних олімпіад. Хоча на відбірково-тренувальних зборах учні показували можливість повного розв'язання по дві найскладніших задачі за 3 години. Всі тамували незадоволення, зважаючи на результати:

- у 2001–2003 роках у складі команди України один з чотирьох учнів був з навчальних закладів міста Києва;
- у 2004–2005 роках у складі команди України три з чотирьох учнів були з навчальних закладів міста Києва. Четвертий учасник — з Українського фізико-математичного ліцею КНУ ім. Тараса Шевченка, який брав участь у ІІІ етапі олімпіади у місті Києві й мав доступ до умов і системи тестування.

У ці роки *всі* (!) учасники-кияни успішно брали участь у відбірково-тренувальних зборах до Міжнародних олімпіад з інформатики. Це засвідчує *доброт-ність* їхньої підготовки до олімпіади. Принагідно зазначимо, голова журі при цьому дотримувався певної ідеології, яку *перед втіленням* виклав і відкрито поширював (Яка мета освіти? — Матеріали Всеукраїнської конференції "Актуальні проблеми вивчення природничо-математичних дисциплін у загальноосвітніх навчальних закладах України". – К.: КУ ім. Т.Шевченка. – 1999. – С.3–4., Інформатизація і мета освіти. — Всеукраїнська науково-практична конференція "Сучасний стан і перспективи шкільних курсів математики та інформатики у зв'язку з реформуванням у галузі освіти, 14–16 листопада, 2000 рік, м. Дрогобич, с. 144–148). Ідеологія мала своє змістовне доповнення — Програму спеціального курсу "Прикладна математика" для учнів 8–11 (9–12) класів з поглибленим вивченням дисциплін математичного профіля (Збірник МОН України "Програми факультативних курсів та курсів за вибором з математики для загальноосвітніх навчальних закладів. К., "Навчальна книга", 2002, с.55–69).

У 2001–2005 роках «*авторитаризм щодо репертуарної політики*» виконав всі поставлені завдання і вичерпав всі свої можливості. У 2006 році з ми повернулися до традиційної практики колегіального упорядкування олімпіадних завдань. Авторами завдань стали учасники Міжнародних олімпіад з інформатики, колишні випускники навчальних закладів міста Києва. Голова журі лише координував роботу журі та здійснював загальну редакцію матеріалів комісії зі створенню завдань.

Принагідно зазначимо, що учасники олімпіади після проведення кожного туру могли отримати електронні копії умов завдань, ідеї розв'язання та тестові файли. Учасники олімпіади, таким чином, мали можливість ґрунтовно підготуватися до апеляції результатів перевірки найкращим .

Журі вдалося упорядкувати завдання з максимальним урахуванням можливостей учасників олімпіади: найкращий результат — 595 балів з 600, але є учасники, які набрали всі бали за один з двох турів. Колективне редагування умов суттєво зменшило кількість уточнень умови під час проведення олімпіади. Підчас апеляції всі умови завдань і тести до них визнано коректними. Всі заяви

учасників стосувалися лише роботи системи тестування і невиконання учасниками умов Порядку проведення ІІІ етапу. Журі задовольнило всі такі прохання, враховуючи те, що учасникам про взаємодію зі системою тестування повідомили лише під час проведення олімпіади. Вирішено наступного року:

- надавати учасникам можливість перевірити коректність взаємодії їхніх розв'язань з системою тестування (замовлення компілятора та визначення авторства) під час виконання завдань;
- відмовляти у апеляції щодо коректність взаємодії їхніх розв'язань з системою тестування, про що явно попередити у Порядку проведення.

З метою максимального наближення умов проведення до умов проведення IV (Всеукраїнського) етапу та Міжнародних олімпіад з інформатики перелік компіляторів (Turbo Pascal 7.0 і Turbo C++ 3.0) було доповнено: (Free Pascal 2.0 та Dev C++ 4.9.9.2 разом з Mingw/GCC 3.4.2). Повне розв'язання поданих далі завдань можна отримати лише за допомогою нових програмних засобів.

1. Умови завдань

1. Хеш-функція (автор Дмитро Кордубан)

Вхідний файл:	hf.in	**Максимальна оцінка: 100 балів**
Вихідний файл:	hf.out	**Обмеження по часу: 2 сек.**
Програма:	hf.pas/.c/.cpp	**Обмеження по пам'яті: 64 МБ**

Нехай $X = x_1 x_2 \ldots x_n$ — послідовність символів довжини n, що складається з малих літер латиниці. Позначимо через $X_{ij} = x_i x_{i+1} \ldots x_j$ його підпослідовність, що містить усі символи з i-го по j-ий.

Візьмемо довільну таку послідовність символів X. Її можна ототожнити з числом у системі числення з основою 26:

$$N(X) = N(x_1)\cdot 26^{n-1} + N(x_2)\cdot 26^{n-2} + \ldots + N(x_{n-1})\cdot 26 + N(x_n).$$

Тут $N(a) = 0$, $N(b) = 1$, $N(c) = 2$, … , $N(z) = 25$ — порядковий номер літери в латиниці, починаючи з нуля. Для довільних X — не порожньої послідовності і h — натурального числа означимо таку функцію: $HF(X, h)$ — це остача від ділення $N(X)$ на h. Вам потрібно дослідити поведінку функції HF.

Завдання

Дано послідовність X довжини n, що містить лише малі літери латиниці, та натуральне число h. Розглянемо всі можливі (у сенсі різних індексів i та j) її непорожні підпослідовності X_{ij} і значення $HF(X_{ij}, h)$, що їх набуває функція на цих підпослідовностях. Вам потрібно визначити, скільки разів функція HF набула значення 0, скільки разів набула значення 1, ... , скільки разів набула значення $h - 1$.

Вхідні дані

Перший рядок вхідного файлу містить ціле число n ($0 \le n \le 5000$) і натуральне число h ($h \le 10000$), розділені пропуском. Другий рядок містить X

— послідовність n малих літер латиниці. Жодних інших символів у другому рядку немає (крім ознаки кінця рядка). У 30% вхідних файлів $n \le 100$.

Вихідні дані

Вам потрібно вивести до вихідного файлу h чисел. Першим виведіть кількість підпослідовностей X, на яких функція $HF(X, h)$ набуває значення 0. Другим — кількість підпослідовностей X, на яких функція $HF(X, h)$ набуває значення 1. Аналогічно, k-им — кількість підпослідовностей X, на яких функція $HF(X, h)$ набуває значення $k - 1$.

Приклад

hf.in	hf.out
3 5 bbc	0 2 2 1 1

Пояснення до прикладу

В прикладі підпослідовності X_{11} та X_{22} вважаються різними, бо починаються з різних індексів, і враховуються окремо, незважаючи на їх рівність як рядків.

$HF(X_{11}, 5) = 1$, $HF(X_{12}, 5) = 2$
$HF(X_{22}, 5) = 1$, $HF(X_{23}, 5) = 3$
$HF(X_{33}, 5) = 2$, $HF(X_{13}, 5) = 4$

2. Кур'єри (автор Андрій Гриненко)

Вхідний файл: courier.in **Максимальна оцінка: 100 балів**
Вихідний файл: courier.out **Обмеження по часу: 2 сек.**
Програма: courier.pas/.c/.cpp **Обмеження по пам'яті: 64 МБ**

У місті X є пряма вулиця Товстунів. Всі її мешканці, за дивним збігом обставин, дуже люблять добряче попоїсти. Компанія ННС («Нагодуємо навіть слона») вирішила заробити на цьому гроші. Вона розгорнула широку мережу доставки їжі замовникам. На вулиці одночасно і постійно перебувають N кур'єрів. Після отримання замовлення компанією продукти доставляє кур'єр, що перебував у момент отримання замовлення найближче до будинку замовника. Вважати, що такий кур'єр завжди єдиний і доставка ним замовлення відбувається миттєво. Надалі кур'єр залишається чекати наступного замовлення вже на цьому новому місці.

Завдання

Визначити сумарну відстань, яку пройдуть усі кур'єри при виконанні всіх замовлень.

Вхідні дані

Перший рядок містить два цілих числа: N ($2 \le N \le 100\ 000$) — кількість кур'єрів та M ($0 \le M \le 100\ 000$) — кількість замовлень.

Другий рядок містить N натуральних чисел X_1, X_2, ... , X_N. Тут X_i — координата початкового положення i-го кур'єра на вулиці Товстунів, $1 \le X_i \le 1\ 000\ 000\ 000$.

Третій рядок містить M натуральних чисел Y_1, Y_2, ... ,Y_M. Тут Y_j — координата будинку j-го замовника з вулиці Товстунів, $1 \le Y_j \le 1\ 000\ 000\ 000$.

Вихідні дані

Єдиний рядок вихідного файлу має містити ціле число — шукану сумарну відстань, яку подолають кур'єри.

Приклади

courier.in	courier.out
3 6 5 11 3 7 8 5 1 12 4	13
2 5 2 3 1 1 1 1 1	1

3. Робочий календар (автор Володимир Ткачук)

Вхідний файл: calendar.in
Вихідний файл: calendar.out
Програма: calendar.pas/.c/.cpp
Максимальна оцінка: 100 балів
Обмеження по часу: 2 сек.
Обмеження по пам'яті: 16 МБ

На планеті Олімпія місяць триває N тижнів, а в кожному тижні M днів. Пересічний мешканець Олімпії сам формує свій робочий графік, але у зв'язку з магнітними бурями одні дні сприятливі для його роботи, а інші ні. У мешканця є календар на місяць, в якому для кожного дня написано "число сприятливості". Це число вказує на те, наскільки відповідний день сприятливий для роботи.

Робочий графік складається з довільної кількості дводенних змін. Другий день кожної зміни завжди за календарем або наступний після першого дня, або саме через тиждень після першого дня. Можна розпочинати нову робочу зміну, не закінчивши попередні. Кожен день місяця може належати не більше, ніж одній зміні. Всі зміни потрібно завершити до кінця місяця.

Плануючи свій робочий місяць, житель планети Олімпія прагне збільшити суму чисел сприятливості тих днів, в які він працюватиме протягом місяця. Ваша задача розробити програму яка б визначала, на яке максимально можливе сумарне число сприятливості може розраховувати житель Олімпії протягом місяця.

Вхідні дані

В першому рядку вхідного файла записано два натуральних числа через прогалину: N і M ($1 \le N \le 100$, $1 \le M \le 10$). Далі у файлі записано календар: N рядків по M цілих чисел у кожному рядку. i-е число j-ого рядка — це число

сприятливості *i*-ого дня *j*-ого тижня календаря. Кожне таке число є цілим і за абсолютним значенням (модулем) не перевищує 100.

Вихідні дані

Єдиний рядок вихідного файла має містити одне ціле число — максимально можливе сумарне число сприятливості для робочих днів жителя планети Олімпія.

Приклад

calendar.in	calendar.out
3 4 **1 3 3** -7 **2** -2 -6 **5** **-3** -4 -5 -6	11

Пояснення до прикладу

Найкращий робочий графік виглядає так: перша робоча зміна — перший день першого тижня і перший день другого тижня, друга робоча зміна — другий і третій дні першого тижня, третя робоча зміна — четвертий день другого тижня і перший день третього тижня. Сумарне число сприятливості дорівнює 1 + 2 + 3 +3 +5 + (–3) = 11.

4. Симетричні послідовності (автор Юрій Знов'як)

Вхідний файл: symmetry.in
Вихідний файл: symmetry.out
Програма: symmetry.pas/.c/.cpp

Максимальна оцінка: 100 балів
Обмеження по часу: 2 сек.
Обмеження по пам'яті: 64 МБ

Останнім часом Петрик захопився послідовностями. Петрику дуже хочеться отримати з заданої послідовності симетричну послідовність. Послідовність (a_1, a_2, … , a_N) називають симетричною, якщо $a_i = a_{N+1-i}$ для всіх i = 1, 2, 3 , … , N. Наприклад, послідовність (10, 20, 30) не симетрична, а послідовності (10, 20, 10) і (10, 20, 20, 10) — симетричні. Послідовність з одного елементу завжди симетрична.

За один хід Петрик може замінити два суміжні числа на їхню суму. Інакше кажучи, за один хід з послідовності (a_1, a_2, ... , a_{i-1}, $\boldsymbol{a_i}$, $\boldsymbol{a_{i+1}}$, a_{i+2}, ..., a_N) можна отримати послідовність (a_1, a_2, ..., a_{i-1}, $\boldsymbol{a_i+a_{i+1}}$, a_{i+2},..., a_N). Петрика цікавить найкоротша послідовність ходів, якою він може з заданого набору отримати симетричний набір.

Завдання

Для заданої початкової послідовності знайдіть найменшу кількість ходів, якими можна дану послідовність перетворити на симетричну.

Вхідні дані

У першому рядку файлу symmetry.in задано натуральне число N — кількість чисел у початковій послідовності ($N \le 200\,000$). Другий рядок містить послідовність N натуральних чисел — члени цієї послідовності. Сума всіх членів послідовності не перевищує 2 000 000 000.

Вихідні дані

У єдиний рядок файлу symmetry.out виведіть довжину найкоротшої послідовності ходів, якою можна з заданого набору зробити симетричний.

Приклади

Symmetry.in	symmetry.out
5 1 2 2 4 5	2
5 1 2 3 2 1	0
5 1 2 5 7 1	1

Пояснення до прикладів

В першому тесті нам задано набір (1, 2, 2, 4, 5). Існує послідовність з 2-х ходів, якою можна цей набір перевести в симетричний набір:

(1, 2, 2, 4, 5) → (1, 4, 4, 5) → (5, 4, 5).

Але не існує коротшої послідовності, якою можна було б отримати симетричний набір.

В другому тесті нам задано уже симетричний набір, тому відповідь 0.

В третьому тесті за 1 хід можна об'єднати другий і третій елементи отримавши симетричну послідовність (1, 7, 7, 1).

5. Садок вишневий (автор Данило Мисак)

Вхідний файл: garden.in
Вихідний файл: garden.out
Програма: garden.pas/.c/.cpp

Максимальна оцінка: 100 балів
Обмеження по часу: 1 сек.
Обмеження по пам'яті: 32 МБ

Білому володарю всесвіту Генд-Альфу належить садок на нескінченній в усі сторони площині, в якому ростуть n вишневих дерев. Генд-Альф хоче залишити цей садок у спадок двом своїм синам. Для цього йому необхідно розділити всі дерева прямолінійним парканом таким чином, щоб по обидві сторони паркану знаходилась однакова кількість вишневих дерев. При цьому може виявитися, що Генд-Альфу необхідно буде прокласти паркан через деякі дерева. В цьому разі він спочатку спилює ці дерева, а вже потім будує паркан. Ваша задача – допомогти Генд-Альфу побудувати паркан так, щоб спиляти найменшу можливу кількість дерев, або сказати йому, що розділити садок на дві частини з однаковою кількістю дерев неможливо.

Вхідні дані

Перший рядок вхідного файлу містить натуральне число n ($1 \le n \le 10\,000$) — кількість дерев у садку Генд-Альфа. Другий рядок містить $2n$ чисел — n пар координат дерев відносно певної точки площини садка. Всі координати — цілі числа, що за модулем не перевищують 10 000. Кожне дерево у вхідному файлі описується рівно один раз, і два різних дерева не можуть рости в одному і тому ж місці.

Вихідні дані

Єдиний рядок вихідного файлу має містити чотири числа: перші два — координати точки *A*, останні два — координати точки *B*, відмінної від *A*, такі, що пряма *AB* (як паркан) задовольняє умові задачі. Всі координати мусять бути цілими числами, що за модулем не перевищують 2 000 000 000. Якщо неможливо знайти дві такі різні точки *A* та *B*, необхідно вивести рядок, що містить чотири нулі: „0 0 0 0”.

Приклади

garden.in	garden.out
2 0 0 2 0	1 0 1 1
5 0 1 1 0 1 1 1 2 2 1	-1 -1 2 2

Пояснення до другого прикладу: паркан, заданий, наприклад, точками (1,1) та (3,1), не задовольняє умові задачі, бо при його побудові довелось би спиляти три дерева, в той час, як ми можемо спиляти лише одне.

6. Острови (автор Юрій Знов’як)

Вхідний файл: islands.in
Вихідний файл: islands.out
Програма: islands.pas/.c/.cpp

Максимальна оцінка: 100 балів
Обмеження по часу: 2 сек.
Обмеження по пам’яті: 64 МБ

Далеко в океані знаходиться держава, розташована на архіпелагу з N островів, ($N \le 100000$). Між деякими з островів збудовано мости. Між довільними двома островами існує не більше одного маршруту, в якому кожен міст проходять лише один раз вздовж цього маршруту. Геологічні дослідження виявили, що майже усі острови здатні приносити чималий зиск від видобутку кольорових металів у копальнях. На жаль, інженерні обрахунки показали, що міст може зруйнуватись, якщо він сполучає острови, на яких одночасно працюють шахти. Отже, якщо по обидва боки мосту працюють копальні, то цей міст необхідно буде закрити. Закриття мостів створює багато незручностей, тому держава виділяє кошти, щоб упоратися з незадоволенням мешканців.

Завдання

Задано опис мостів, які сполучають острови, а також штраф за закриття кожного з мостів. Для кожної шахти відомо, який вона дає прибуток, коли працює. Потрібно вказати такий набір островів, на яких треба проводити видобувні роботи, щоб принести державі максимальний зиск. Зиск — це різниця суми прибутків з працюючих шахт та суми штрафів за закриті мостові з'єднання.

Вхідні дані

Перший рядок файлу islands.in містить два натуральних числа: N — кількість островів і M — кількість мостів. Другий рядок містить послідовність N натуральних чисел: j-те число — прибуток, який буде приносити працююча шахта на острові j. В наступних M рядках задано опис мостів. В кожному з цих рядків — три числа — номери островів, які з'єднуються цим мостом (острови нумерують натуральними числами від 1 до N), а також штраф за закриття даного мосту. Прибуток від шахти і штраф за закриття мосту — натуральні числа, що не перевищують 10 000.

Вихідні дані

Сформуйте відповідь до задачі у файл islands.out за таким принципом:

- в перший рядок файлу виведіть максимальний зиск;
- в другий рядок виведіть K — кількість островів в вашому наборі. Після чого виведіть K номерів островів без повторів, на яких Ви радите почати видобуток цінних металів.

Оцінювання

За правильну величину зиску вам буде зараховано 30% балів за відповідний тест. Якщо ваш набір островів виявиться одним з оптимальних, то вам буде зараховано 70% балів за відповідний тест. Бали за обидві частини задачі нараховують незалежно. Не менш, ніж у 40% тестів $N \leq 1\,000$.

Приклади

islands.in	islands.out
7 5 10 70 70 5 5 1 1 1 2 8 2 3 30 3 4 5 3 6 2 2 7 2	117 4 1 2 3 5
7 5 10 70 70 6 5 1 1 1 2 8 2 3 30	118 5 1 2 3 4 5

3 4 5 3 6 2 2 7 2	

3. Ідеї розв'язання

1. Хеш-функція. Основна ідея вирішення задачі така. Масив $b[0..h-1]$ заповнюємо нулями. Нехай ми розглядаємо непорожній підрядок X_{ij} рядку X та безпосередньо визначаємо для нього хеш-функцію. Якщо $HF(X_{ij}, h) = k$, збільшуємо значення $b[k]$ на одиницю. Розглянувши всі можливі непорожні підрядки X, у масиві b отримуємо значення, що залишається вивести у вихідний файл. Отже, задача звелась до такої: для будь-якого підрядку X_{ij} безпосередньо визначити хеш-функцію від нього.

Це можна зробити, наприклад, таким чином. Для всіх i в межах від 1 до n заповнюємо масив $a[1..n]$ значеннями хеш-функції: $a[i] = HF(X_{ii}, h)$, що можна зробити за $O(n)$ операцій. Далі для всіх i в межах від 1 до $n-1$ перебираємо j в межах від $i+1$ до n та за допомогою рекурентного співвідношення

$$HF(X_{ij}, h) = (HF(X_{i\,(j-1)}, h) \cdot 26 + HF(X_{jj}, h)) \bmod h$$

знаходимо значення хеш-функції для підрядка X_{ij}.

Загальний час розв'язку: $O(n^2)$

2. Кур'єри. Алгоритм подано коментарями авторського розв'язання (див. далі):

1. Зчитування значень N і M.
2. Зчитування початкових позицій кур'єрів.
3. Впорядкування позицій кур'єрів за зростанням координати.
4. Зчитування позиції всіх поточних замовлень, і для них:
   - перевірка крайніх кур'єрів (чи поточне замовлення лівіше найлівішого з кур'єрів або правіше найправішого з них);
   - визначення найближчого кур'єра двійковим (бінарним) пошуком;
   - збільшення пройденої відстані на відповідну величину й відповідна зміна координати одного кур'єра.
4. Виведення результату

Порядок розташування кур'єрів незмінний при виконанні всіх замовлень.

3. Робочий календар. Спочатку розглянемо більш просту задачу: в кожному тижні один день ($M = 1$). Тоді календар можна подати одновимірним масивом C довжиною N, а робоча зміна — це пара чисел, що стоять поряд. Така спрощена задача розв'язується методами динамічного програмування. Подамо один з методів розв'язання.

Запровадимо допоміжні масиви A і B**:**

- в елементі A**[**i**]** будемо зберігати найбільше сумарне число сприятливості яке можна отримати, якщо всі робочі зміни відпрацьовані за перші i тижнів (днів), а останній день останньої робочої зміни — це день з номером i**;**
- в елементі B**[**i**]** будемо зберігати найбільше сумарне число сприятливості за перші i днів, але за умови, що i-ий робочий день не належить ні до якої

робочої зміни.

Справджуються такі висловлювання:

1. $B[1] = 0$, $A[1] = 0$,
2. Для всіх $i > 1$ $A[i] = B[i-1] + C[i-1] + C[i]$
3. Для всіх $i > 1$ $B[i] = \max(B[i-1], A[i-1])$

За допомогою співвідношень 2 і 3 можна обчислити $A[i]$ та $B[i]$ для будь-якого i. Відповіддю задачі буде число $K = \max(A[N], B[N])$. Складність роботи алгоритму $O(N)$.

Тепер розглянемо задачу при $M > 1$ і $N > 1$. У цьому разі робоча зміна це пара чисел, що можуть стояти як поряд, так і через $M - 1$ число одне від одного.

Запровадимо такі поняття:

- Для тижня робочого календаря кожен робочий день або належить якійсь робочій зміні, або ні. Тобто кожному робочому тижню можна поставити у відповідальність масив довжиною M, елементами якого є 0 або 1. Причому на i-ому місці стоїть 1, якщо i-ий день належить робочій зміні, *що завершується до кінця цього тижня,* або 0 у іншому випадку. Такий масив з 0 та 1 будемо називати станом робочого тижня. Станом i-го робочого дня будемо називати i-ий елемент стану тижня, до якого цей день належить.
- Розглянемо стани двох послідовних робочих тижнів. Вартістю переходу між цими станами, назвемо максимально можливе сумарне число сприятливості тих робочих змін, що починаються на першому тижні у день зі станом 0, або на другому тижні у день зі станом 1. В обох випадках робоча зміна обов'язково має закінчуватись на другому тижні в день зі станом 1.

Запровадимо допоміжний двомірний масив $A[1..N, 0..2^M-1]$:
в елементі $A[i, j]$ будемо зберігати найбільше можливе число сприятливості для перших i тижнів, причому стан тижня i відповідає запису числа j у двійковій системі числення.

Справджуються такі висловлювання.

1. Для всіх станів j: $A[0, j] = 0$;
2. Для всіх $i > 0$, $0 \le j < 2^M$: $A[i, j] = \max\limits_{k \in \{0,1,\dots,2^M-1\}} (A[i-1, k] + F(i-1, k, i, j))$,

де F — вартість переходу між станом k тижня $i-1$ та станом j тижня i.

За допомогою цих правил 1–2 можна обчислити $A[i, j]$ послідовно для всіх i, j.

Відповіддю задачі буде число $\max\limits_{k \in \{0,1,\dots,2^M-1\}} A[N, k]$.

Складність алгоритму є $O(N \cdot 2^{2M})$.

Подане розв'язання задачі ілюструє підхід, відомий як „*динаміка по краю*".

4. Симетричні послідовності. Цю задачу можна розв'язати жадібним алгоритмом. Позначимо члени даної послідовності через $a[1], a[2], \dots, a[N]$.

Якщо $a[1] = a[N]$, то відповідь буде такою самою, як і відповідь для послідовності $a[2], a[3], \dots, a[N-1]$.

Якщо $a[1] < a[N]$, тоді довільна послідовність ходів, яка перетворює початкову послідовність на симетричну, одним з ходів об'єднує перший

елемент з другим. У цьому випадку відповідь на задачу на одиницю більша, ніж для послідовності $a[1]+a[2], a[3], \ldots, a[N]$.

Аналогічно розглядають випадок $a[N] > a[1]$. Тоді відповідь на 1 більша, ніж для послідовності $a[1], a[2], \ldots, a[N-1] + a[N]$.

5. Садок вишневий. Покажемо, що побудова паркану завжди можлива. Запровадимо для дерев відношення (порядку) „північно-східності". Нехай одне дерево (перше) північно-східніше іншого (другого) тоді і лише тоді, коли

$$\left[\begin{array}{l} x_1 > x_2 \\ \left\{\begin{array}{l} x_1 = x_2 \\ y_1 > y_2 \end{array}\right. \end{array}\right. ,$$

де (x_1, y_1) — координати першого дерева, а (x_2, y_2) — другого. Впорядкуємо всі дерева за цим відношенням так, щоб отримана послідовність дерев (x_1, y_1), (x_2, y_2), … , (x_n, y_n) задовольняла такій умові:

$$\forall\, i, j, 1 \le i < j \le n : (x_j, y_j) \text{ північно-східніше } (x_i, y_i).$$

Спочатку розглянемо випадок, коли n непарне. В такому разі неможливо побудувати паркан, що поділяє площину на дві рівні за кількістю на них дерев півплощини і не проходить ні через одне дерево (бо тоді сумарна кількість дерев, — сума кількостей дерев на обох півплощинах, була б парною). Отже, достатньо побудувати паркан, що проходитиме рівно через одне дерево і ділитиме площину на дві рівні за кількістю дерев півплощини. Це буде шуканий паркан.

Нехай точка M має координати $(x_{(n+1)/2}, y_{(n+1)/2})$, тобто збігається з деревом-медіаною („серединою") за відношенням північно-східності. Проведемо через точку M пряму, що „майже паралельна" осі ординат та відділяє перші за відношенням північно-східності $(n-1)/2$ дерев від останніх $(n-1)/2$. Наприклад, пряму MN, де N має координати $(x_{(n+1)/2} - 1, 2\,000\,000\,000)$. Координати всіх дерев за модулем не перевищують 10 000, тому така пряма проходитиме лише через дерево, що стоїть у точці M. Зауважимо, що координати точок M та N цілі та за модулем не перевищують 2 000 000 000. У вихідний файл запишемо координати точок M та N.

Тепер розглянемо випадок, коли n парне. Для цього випадку побудуємо паркан, що не проходить ні через одне дерево та поділяє площину на дві рівні за кількістю дерев півплощини. Очевидно, тоді це й буде паркан, що задовольнить умову.

Нехай точка M має координати $(x_{1+n/2}, y_{1+n/2} - \frac{1}{2})$. Проведемо через точку M пряму, що „майже паралельна" осі ординат та відділяє перші $n/2$ дерев від наступних $n/2$. Наприклад, пряму MN, де N має координати $(x_{1+n/2} - 1, 1\,000\,000\,000)$. Координати всіх дерев за модулем не перевищують 10 000, тому така пряма не проходитиме ні через одне дерево. Ордината точки

M неціла. Знайдемо ще одну точку на *MN*, таку, щоб обидві її координати були цілими. Такою точкою є, наприклад, точка $K(x_{1+n/2}+1, 2y_{1+n/2}-1-1\,000\,000\,000)$.

Зауважимо, що координати точок *N* та *K* за модулем не перевищують 2 000 000 000. У вихідний файл запишемо координати точок *N* і *K*.

Загальний час розв'язку: $O(n)$ (або $O(n \cdot \log n)$, якщо дерево-медіану шукати сортуванням масиву).

6. Острови. Архіпелаг з мостами є лісом (графом без циклів). Шукана відповідь — це максимальна сума прибутків на кожній компоненті зв'язності графу (дереві). В кожній компоненті зв'язності ми можемо зафіксувати довільну вершину і назвати її *коренем компоненти*. Це дасть нам змогу *орієнтувати усі ребра* так, щоб вони ішли від кореня. Інакше кажучи, припишемо кожній вершині відстань від кореня, що є довжиною маршруту (у кількості ланок), який сполучає корінь і дану вершину. Далі орієнтуємо кожне ребро таким чином, щоб воно йшло від вершини з меншою відстанню до вершини з більшою відстанню. Таке орієнтування дає нам змогу розбити нашу задачу на підзадачі. Дійсно, розглянемо піддерево, утворене якоюсь вершиною разом з усім досяжними з неї вершинами, якщо рухатися у сторону «від кореня». Запровадимо такі позначення:

- `adj[v]` — множина суміжних з островом `v` островів;
- `income[v]` — дохід від шахт з острову `v`;
- `closeCost[e]` — ціна закриття мосту `e`;
- `profitUse[v]` — відповідь для підзадачі для дерева з коренем в вершині `v`, якщо на острові `v` будуть відбуватись видобувні роботи;
- `profitLeave[v]` — те ж саме, що і `profitUse[v]` за умови, що на острові `v` *не* будуть проходити видобувні роботи.

Справджуються такі рекурентні співвідношення:

$$\texttt{profitUse}[v] = \texttt{income}[v] + \sum_{u \in \texttt{adj}[v]} \max\{\texttt{profitLeave}[u], \texttt{profitUse}[u] - \texttt{closeCost}[(v,u)]\}$$

$$\texttt{profitLeave}[v] = \sum_{u \in \texttt{adj}[v]} \max\{\texttt{profitUse}[u], \texttt{profitLeave}[u]\}$$

Авторське розв'язання використовує рекурсивну процедуру.

4. Авторські розв'язання.

1. Хеш-функція

```
{$I-,Q-,R-,S-}
const
  file_in  = 'hf.in';
  file_out = 'hf.out';
  dim1       = 5005;
  dim2       = 10005;
var
  a       : array[1..dim1] of longint;
```

```
  b       : array[0..dim2] of longint;
  n,i,j,k,x,h  : longint;
  ch      : char;
begin
  assign(input,file_in);
  reset(input);
  assign(output,file_out);
  rewrite(output);
  readln(n,h);
  fillchar(b,sizeof(b),0); { nullify b }
  for i:=1 to n do { for each letter }
  begin
    read(ch); { ch = Xi }
    a[i]:=ord(ch)-ord('a'); { a[i] = N(X_i) }
  end;
  for i:=1 to n do { for each suffix X_in }
  begin
    x:=0;
    for j:=i to n do { for each substring X_ij }
    begin
    x:=(26*x+a[j]) mod h; {HF(X_ij)=(26*HF(X_ij-1)+N(X_jj)) mod h}
      inc(b[x]); { count this remainder }
    end;
  end;

  { output answer }
  for i:=0 to h-2 do write(b[i],' ');
  writeln(b[h-1]);
  close(output);
end.
```

2. Кур'єри

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define INPUT "courier.in"
#define OUTPUT "courier.out"
int i, m, n, x;
vector <int> a;
long long r;
int main ()
{
  r = 0;
  freopen (INPUT, "r", stdin);
  freopen (OUTPUT, "w", stdout);

// зчитування значень N і M
  scanf ("%d%d", &n, &m);
  a.resize (n);
```

```
// зчитування початкових позицій кур'єрів у масив a
  for (i=0; i<n; i++)
  {
    scanf ("%d", &a[i]);
  }

// сортування позицій кур'єрів за зростанням координати

  sort (a.begin(), a.end());
  while (m--)
  {
// зчитування позиції поточного замовлення
    scanf ("%d", &x);

//перевірка крайніх кур'єрів (для випадку, якщо поточне замовлення
// лівіше найлівішого з кур'єрів або правіше найправішого з них)

    if (a[0]>=x)
    {
      r+= (long long)(a[0]-x);
      a[0] = x;
    }
    else if (a[n-1]<=x)
    {
      r+= (long long)(x-a[n-1]);
      a[n-1] = x;
    }
    else
    {

// визначення найближчого кур'єра двійковим пошуком
      i = lower_bound (a.begin(), a.end(), x) - a.begin ();
      if ((a[i]-x)<(x-a[i-1]))
      {
       r+= (long long)(a[i]-x);
       a[i] = x;
      }
      else
      {
       r+= (long long)(x-a[i-1]);
       a[i-1] = x;
      }
    }
  }

// виведення результату
  cout << r << endl;
  return 0;
}
```

Неповне розв'язання "в лоб" (перевіряє правильність тестів і набирає лише 40 балів через обмеження за часом)

```
#include <cstdio>
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;
#define INPUT "courier.in"
#define OUTPUT "courier.out"

int i, j, m, n, x;
vector <int> a;
long long r;
int main ()
{
  r = 0;
  freopen (INPUT, "r", stdin);
  freopen (OUTPUT, "w", stdout);
  scanf ("%d%d", &n, &m);
  a.resize (n);
  for (i=0; i<n; i++)
  {
    scanf ("%d", &a[i]);
  }
  sort (a.begin(), a.end());
  while (m--)
  {
    scanf ("%d", &x);

    i = 0;

    for (j=1; j<n; j++)
    {
      if (abs(a[i]-x)==abs(a[j]-x))
      {
       cout << "ERROR: EQUAL DISTANCE!!!!!!!!!!!!";
       return 0;
      }
      if (abs(a[i]-x)>abs(a[j]-x))
      {
       i = j;
      }
    }
    r+=(long long)abs(a[i]-x);
    a[i] = x;
  }
  cout << r << endl;
  return 0;
}
```

3. Робочий календар

{Блок сталих:

maxN – максимально можлива кількість тижнів,

maxM – максимально можлива кількість днів у тижні,

inputfile – назва вхідного файлу,
outputfile – назва вихідного файлу.}

```
const
      maxN = 100;
      maxM = 10;
      inputfile  = 'calendar.in';
      outputfile = 'calendar.out';
var
```
{C – масив в який зчитуємо числа сприятливості}
```
    C      : array[0..maxN,1..maxM] of longint;
```

{В масиві А зберігаємо максимальне число сприятливості для календаря, останній тиждень якого має певну конфігурацію. Конфігурація задає другий індекс масиву А і її двійковий запис відповідає розташуванню вільних і занятих якоюсь робочою зміною днів: якщо 1 – то день занятий, 0 – день вільний. Для опису всіх можливих конфігурацій потрібно 2^N чисел – від 0 до 2^N-1. Для розв'язання нам необхідно пам'ятати конфігурацію лише попереднього рядка, тому масив А має розмірність 2 на 2^N}
```
    A      : array[0..1,0..(1 shl maxM)-1] of longint;
```

{M,N – кількість днів у тижні і кількість тижнів у місяці відповідно}
```
    M,N    : integer;
```

{Зчитування даних з файлу}
```
procedure ReadData;
var i,j : integer;
begin
     assign(input,inputfile);
     reset(input);
     readln(N,M);
```
{спочатку заповнюємо весь масив С нулями}
```
     fillchar(C,sizeof(C),0);
     for i:=1 to N do
         for j:=1 to M do
             read(C[i,j]);
     close(input)
end;
```

{Запис відповіді у файл}
```
procedure WriteAnswer(answer : longint);
begin
     assign(output,outputfile);
     rewrite(output);
     writeln(answer);
     close(output)
end;
```

{Сама «делікатна» процедура програми, визначає максимальне число сприятливості для даної конфігурації. Рекурсивно перебирає всі конфігурації з яких ми можемо отримати дану. Параметри мають такий зміст:
i – кількість тижнів, або поточний тиждень в календарі (насправді остача від ділення номера тижня на 2;
j – конфігурація, максимальне число сприятливості ми шукаємо;

b – поточний день конфігурації cur, який ми зараз підбираємо, в процедурі маємо підібрати всі M днів;
cur – поточна конфігурація з якої ми хочемо «отримати» конфігурацію j;
sum – поточне число сприятливості переходу від конфігурації cur до конфігурації j.

В процедурі використано бітові операції:
Shl – зсув двійкового запису числа ліворуч, a shl b = a*2^b;
xor – побітове «чи», a xor b = c, де і-ий біт числа c – і-ий біт числа a xor і-ий біт числа b;
and – побітове «і»}

```
procedure maximize(i,j,b,cur : integer; sum : longint);
begin
     if (b = M) or ((cur and (1 shl b))=0) then
{Обмеження рекурсії, якщо ми вже підібрали всі дні конфігурації cur}
        begin
             if A[i mod 2,j]<A[(i-1) mod 2,cur]+sum then
{запам'ятовуємо краще число сприятливості}
                A[i mod 2,j]:=A[(i-1) mod 2,cur]+sum
        end
     else
{в цьому блоці перебираємо декілька варіантів для b-го дня конфігурації cur}
        begin

{Варіант, коли перший день поточного тижня в одній робочій зміні з останнім днем попереднього тижня}
    if (b=0) and (j and 1 > 0) and (C[i,1]+C[i-1,M]>0) then
    maximize(i,j,b+1,cur xor (1 shl (M-1)),sum+C[i,1]+C[i-1,M]);

{Варіант, коли b-ий день поточного тижня в одній робочій зміні з b-им днем попереднього тижня}
    if (j and (1 shl b)>0) and (C[i,b+1]+C[i-1,b+1]>0) then
    maximize(i,j,b+1,cur xor (1 shl b),sum+C[i,b+1]+C[i-1,b+1]);

{Варіант, коли b-ий день поточного тижня в робочій зміні з b+1-им днем цього ж тижня}
    if (b<M-1) and (j and (1 shl b)>0) and (j and (1 shl (b+1))>0)
                        and (C[i,b+1]+C[i,b+2]>0) then
                maximize(i,j,b+2,cur,sum+C[i,b+1]+C[i,b+2]);

{Варіант, коли b-ий день поточного тижня не належить жодній робочій зміні}
             maximize(i,j,b+1,cur,sum);
        end
end;

{Обчислення відповіді}
function GetAnswer : longint;
var i,j : integer;
begin
{Уявимо календар, що складається з 0 тижнів, в ньому всі конфігурації мають дуже маленьке число сприятливості (-1000), крім конфігурації, де всі дні зайняті. Такий трюк потрібен лише для загальності розв'язку}
     for i:=0 to (1 shl M) -2 do
         A[0,i]:=-1000;
```

{Конфігурація, де всі дні зайняті має число сприятливості 0}
```
     A[0,(1 shl M)-1]:=0;
```

{Послідовно збільшуємо кількість тижнів у календарі (від 1 до N)}
```
     for i:=1 to N do
         for j:=0 to (1 shl M)-1 do
```

{Розглядаємо всі можливі конфігурації останнього рядка}
```
             begin
                  A[i mod 2,j]:=0;
```

{Для кожної конфігурації знаходимо максимально можливе число сприятливості}
```
                  maximize(i,j,0,(1 shl M)-1,0)
             end;
    i:=0;
```

{Знаходимо таку конфігурацію останнього рядка календаря, при якій сумарне число сприятливості максимальне}
```
    for j:=1 to (1 shl M)-1 do
        if A[N mod 2,j]>A[N mod 2,i] then
           i:=j;
```

{Повертаємо максимальне число сприятливості}
```
    GetAnswer:=A[N mod 2,i]
end;
```

{Основна частина програми}
```
begin
     ReadData; {Зчитуємо дані}
     WriteAnswer(GetAnswer); {Обчислюємо результат і записуємо його у файл}
end.
```

4. Симетричні послідовності

```
#include <queue>
#include <cstdio>

using namespace std;

FILE *fIn, *fOut;
int N;
deque<int> Q;

int main()
{
     // Зчитуємо вхідні данні
     fIn = fopen("symmetry.in", "rt");
          fscanf(fIn, "%d", &N);
          Q.resize(N); // Виділяємо N елементів у черзі
          for (int i = 0; i < N; i++)
               fscanf(fIn, "%d", &Q[i]);
```

```
    fclose(fIn);

    // Розв'язуємо задачу
    int moves = 0;
    while (Q.size() > 1) { // Поки у нас не менше 2х елементів...
        if (Q.front() == Q.back()) {// На краях однакові числа
            Q.pop_front();        // вилучаємо спереду і ззаду по елементу
            Q.pop_back();
            continue;
        } else
            moves++; // Доведеться урівнювати елементи на кінцях

        if (Q.front() < Q.back()) {// Найлівіший елемент треба збільшити
            int x = Q.front();
            Q.pop_front();
            Q.front() += x; // Об'єднаємо його з наступним
елементом
        } else { // Треба збільшити найправіший елемент
            int x = Q.back();
            Q.pop_back();
            Q.back() += x; // Об'єднаємо його з попереднім елементом
        }
    }

    // Вивід відповіді
    fOut = fopen("symmetry.out", "w+");
        fprintf(fOut, "%d\n", moves);
    fclose(fOut);

    return 0;
}
```

5. Садок вишневий

```
// Ефективність n ln(n) завдяки сортуванню всього масиву злиттям
program garden;
const maxn=10000;
      maxxy=10000;
      max=2000000000;
var n,i,x,y: longint;
    a,t: array[0..maxn-1] of longint;
    // t - допоміжний масив у процедурі сортування
    f: text;

procedure sort(l,r: longint); // Процедура сортування злиттям
var m,cur1,cur2: longint;
begin
        if r-l>1 then
        begin
                m:=(l+r) div 2;
                sort(l,m);
                sort(m,r);
                cur1:=l;
```

```
                cur2:=m;
                while cur1+cur2<m+r do
                if (cur2=r) or ((cur1<>m) and (a[cur1]<a[cur2]))
then
                begin
                        t[cur1+cur2-m]:=a[cur1];
                        inc(cur1);
                end else
                begin
                        t[cur1+cur2-m]:=a[cur2];
                        inc(cur2);
                end;
                for m:=l to r-1 do a[m]:=t[m];
        end;
end;

begin
        assign(f,'garden.in');
        reset(f);
        readln(f,n);
        for i:=0 to n-1 do
        begin
                read(f,x,y);
                a[i]:=(x+maxxy)*(2*maxxy+1)+(y+maxxy);
// Значення масиву a задають відношення впорядкованості:
// a[i]<a[j] <=> (x[i]<x[j]) або (x[i]=x[j] та y[i]<y[j])
        end;
        close(f);
        sort(0,n);
// Після сортування a[n/2] - середній за значенням елемент a,
// тобто медіана масиву, якщо n непарне,  та права медіана масиву, якщо n парне

        x:=(a[n div 2] div (2*maxxy+1))-maxxy;
// Видобуваємо значення абсциси медіани масиву

        y:=(a[n div 2] mod (2*maxxy+1))-maxxy;
// Видобуваємо значення ординати медіани масиву
        assign(f,'garden.out');
        rewrite(f);
        if n mod 2=1 then writeln(f,x,' ',y,' ',x-1,' ',max)
// Якщо n непарне

        else writeln(f,x-1,' ',max div 2,' ',x+1,' ',
                        2*y-1-(max div 2));
// Якщо n парне
        close(f);
end.
```

Повне розв'язання

```
#include <stdio.h>
```

```
#include <algorithm>
#define maxn 10000
#define maxxy 10000
#define max 2000000000

using namespace std;

int n,i,x,y,m[maxn];

int main()
{
    freopen("garden.in","r",stdin);
    freopen("garden.out","w",stdout);

    scanf("%d\n",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d%d",&x,&y);
        m[i]=(x+maxxy)*(2*maxxy+1)+(y+maxxy);
// Значення масиву m задають відношення впорядкованості:
// m[i]<m[j] <=> (x[i]<x[j]) або (x[i]=x[j] та y[i]<y[j])
    }

    nth_element(m,m+n/2,m+n);
// використання процедури, що входить до складу STL C++
// Тепер m[n/2] - середній за значенням елемент m, тобто
// медіана масиву, якщо n непарне, та права медіана масиву, якщо n парне

    x=m[n/2]/(2*maxxy+1)-maxxy;
// Видобуваємо значення абсциси медіани масиву

    y=m[n/2]%(2*maxxy+1)-maxxy;
// Видобуваємо значення ординати медіани масиву

    if(n%2)printf("%d %d %d %d\n",x,y,x-1,max);
// Якщо n непарне

    else printf("%d %d %d %d\n",x-1,max/2,x+1,2*y-1-max/2);
// Якщо n парне
    return 0;
}
```

6. Острови

```
#include <cstdio>
#include <vector>
#include <algorithm>

using namespace std;

const int MaxV = 102400;

FILE *fIn, *fOut;
```

```
int V, E; // Кількість вершин (островів) і ребер (мостів)
pair<int, int> edgelist[MaxV]; // Список ребер (див 1)
int profit[MaxV]; // Прибуток
vector< vector<int> > next; // Список суміжних ребер

int best[MaxV][2]; // Оптимальний план
bool visited[MaxV];
vector<int> sequence;

// Процедура для розв'язування підзадачі з коренем в вершині v:
// розглядається та компонента, яка знаходиться по шляху ребра prev->v...
// Підрахунок за вказаною формулою
void dfs(int v, int prev = -1)
{
    visited[v] = true;
    best[v][0] = 0;
    best[v][1] = profit[v];
    for (int i = 0; i < next[v].size(); i++) {
        int k = next[v][i];
        int u = edgelist[k].first-v;
        if (u == prev)
            continue; else
            dfs(u, v);

        best[v][0] += max(best[u][0], best[u][1]);
        best[v][1] += max(best[u][0], best[u][1] -
edgelist[k].second);
    }
}

// Процедура для побудови оптимальної послідовності. v, prev - параметри
// компоненти зв'язності. use вказує на те, чи потрібно нам включати вершину v в план.
void route(int v, bool use, int prev = -1)
{
    if (use)
        sequence.push_back(v);

    for (int i = 0; i < next[v].size(); i++) {
        int k = next[v][i];
        int u = edgelist[k].first-v;
        if (u == prev)
            continue;

        if (use)
            route(u, best[u][1] - edgelist[k].second > best[u][0],
v); else
            route(u, best[u][1] > best[u][0], v);
    }
}

int main()
```

```
{
    // Зчитуємо вхідні данні
    fIn = fopen("islands.in", "rt");
        fscanf(fIn, "%d %d", &V, &E);
        next.resize(V);

        for (int i = 0; i < V; i++) // Зчитуємо прибутки з шахт
            fscanf(fIn, "%d", &profit[i]);

        for (int i = 0; i < E; i++) {// Зчитуємо опис мостів і будуємо граф
            int u, v, c;
            fscanf(fIn, "%d %d %d", &u, &v, &c);
            u--, v--;
            edgelist[i] = make_pair(u+v, c);
            next[u].push_back(i);
            next[v].push_back(i);
        }
    fclose(fIn);

    // Розв’язуємо для кожної компоненти зв'язності
    fill_n(visited, V, false); // Спочатку жодний острів не було розглянуто
    int answer = 0;
    for (int i = 0; i < V; i++)
        if (!visited[i]) { // Острів і належить іншій компоненті
            dfs(i);
            answer += max(best[i][1], best[i][0]);
            route(i, best[i][1] > best[i][0]);
        }

    // Виводимо відповідь і саму послідовність
    fOut = fopen("islands.out", "wt");
        fprintf(fOut, "%d\n", answer);

        fprintf(fOut, "%d", sequence.size());
        for (int i = 0; i < sequence.size(); i++)
            fprintf(fOut, " %d", sequence[i]+1);
        fprintf(fOut, "\n");
    fclose(fOut);
    return 0;
}

/*
Кожне ребро має 3 параметри: 2 кінці та ціну. В цій реалізації було використано збереження ребра за допомогою двох полів:
     edgelist[i].first — сума вершин кіців
     edgelist[i].second — ціна

     Таким чином ребро (u,v,c) зберігається як пара (u+v, c). Якщо ми знаємо, що ребро edgelist[i] суміжне вершині u, то протилежний кінець ребра, очевидно, буде: edgelist[i].first-u. Таке подання, насправді, зручніше, ніж трикомпонентне, бо дозволяє уникнути операторів if.
*/
```