ЗАВДАННЯ XXVI МІЖНАРОДНОЇ ОЛІМПІАДИ З ІНФОРМАТИКИ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ РОЗВ'ЯЗАННЯ

Гуржій Андрій Миколайович,
віце-президент НАПН України, доктор технічних наук, професор, академік НАПН України.
Бондаренко Віталій Вікторович,
асистент факультету кібернетики Київського Національного університету ім. Тараса Шевченка.

ЗАВДАННЯ ДРУГОГО ТУРУ

1. Гондола

Гондола Мао-Конг є відомим місцем в Тайпеї. Гондольна система складається з кругової колії, однієї станції та n гондол, пронумерованих послідовно від 1 до n, що рухаються по колії у фіксованому напрямку. На початку, після того, як гондола i минає станцію, наступною гондолою, що мине станцію, буде гондола $i+1$, якщо $i<n$, або гондола 1, якщо $i=n$.

Гондоли можуть ламатись. На щастя, ми можемо дістати нескінчену кількість запасних гондол, які пронумеровані $n+1$, $n+2$, і так далі. Коли гондола ламається, ми замінюємо її (у тій самій позиції на колії) першою доступною запасною гондолою, тобто гондолою з найменшим номером. Наприклад, якщо є 5 гондол і гондола 1 ламається, ми замінюємо її гондолою 6.

Вам подобається стояти на станції та дивитись, як проминають гондоли. *Послідовністю гондол* є послідовність n номерів гондол, що минають станцію. Можливо, що одна або більше гондол зламались (і були замінені) до того, як ви прибули, але ніякі гондоли не ламались, поки ви спостерігаєте за ними.

Зауважте, що деякі конфігурації гондол на колії дають кілька послідовностей гондол, залежно від того, яка гондола минає станцію першою після вашого прибуття. Наприклад, якщо ніякі гондоли ще не ламались, послідовності гондол (2, 3, 4, 5, 1) та (4, 5, 1, 2, 3) є можливими послідовностями гондол, але (4, 3, 2, 5, 1) не є такою (оскільки гондоли з'являються у неправильному порядку).

Якщо гондола 1 ламається (табл. 1), ми будемо спостерігати послідовність гондол (4, 5, 6, 2, 3). Якщо наступною ламається гондола 4, ми замінюємо її гондолою 7 та зможемо спостерігати послідовність гондол (6, 2, 3, 7, 5). Якщо після цього ламається гондола 7, ми замінюємо її гондолою 8 і можемо спостерігати послідовність гондол (3, 8, 5, 6, 2).

Таблиця 1

Зламана гондола	Нова гондола	Можлива послідовність гондол
1	6	(4, 5, 6, 2, 3)
4	7	(6, 2, 3, 7, 5)
7	8	(3, 8, 5, 6, 2)

Послідовність замін — це послідовність номерів гондол, що зламались, у порядку, у якому вони ламались. У попередньому прикладі послідовністю замін є (1, 4, 7). Послідовність замін r *призводить* до послідовності гондол g, якщо після того, як гондоли ламаються відповідно до послідовності замін r, можна спостерігати послідовність гондол g.

Перевірка послідовності гондол

У перших трьох підзадачах ви маєте перевірити, чи є вхідна послідовність послідовністю гондол. Дивіться таблицю нижче з прикладами послідовностей, які є або не є послідовностями гондол. Вам потрібно реалізувати функцію **valid**.

- **valid(n, inputSeq)**
  - — n: довжина вхідної послідовності.
  - — **inputSeq**: масив довжини n; inputSeq [i] є елементом i вхідної послідовності, для $0 \le i \le n-1$.
  - — Функція має повернути 1, якщо вхідна послідовність є послідовністю гондол, або 0 у іншому випадку.

Підзадачі 1, 2, 3

Таблиця 2

Підзадача	Бали	n	inputSeq
1	5	$n \le 100$	Має кожне число від 1 до n рівно один раз
2	5	$n \le 100\,000$	$1 \le$ inputSeq [i] $\le n$
3	10	$n \le 100000$	$1 \le$ inputSeq[i] ≤ 250000

У таблиці 3 наведено приклади.

Послідовність замін. У наступних трьох підзадачах ви маєте побудувати можливу послідовність замін, що призводить до заданої послідовності гондол. Дові-

Таблиця 3

Підзадача	inputSeq	Повернене значення	Примітка
1	(1, 2, 3, 4, 5, 6, 7)	1	
1	(3, 4, 5, 6, 1, 2)	1	
1	(1, 5, 3, 4, 2, 7, 6)	0	1 не може бути безпосередньо перед 5
1	(4, 3, 2, 1)	0	4 не може бути безпосередньо перед 3
2	(1, 2, 3, 4, 5, 6, 5)	0	Дві гондоли мають номер 5
3	(2, 3, 4, 9, 6, 7, 1)	1	Послідовність замін (5, 8)
3	(10, 4, 3, 11, 12)	0	4 не може бути безпосередньо перед 3

льна така послідовність замін буде розв'язком. Ви маєте реалізувати функцію **replacement**.

- **replacement(n, gondolaSeq, replacementSeq)**

— *n*: довжина послідовності гондол.

— gondolaSeq: масив довжини *n* гарантується, що gondolaSeq є послідовністю гондол, gondolaSeq[i] є елементом *i* послідовності, для $0 \le i \le n-1$.

— Функція має повертати *l*, довжину послідовності замін.

— replacementSeq: масив, достатньо великий, щоб зберігати послідовність замін; ви маєте повернути послідовність замін, помістивши елемент *i* послідовності замін до replacementSeq [i], для $0 \le i \le l-1$.

Підзадачі 4, 5, 6

Таблиця 4

Підзадача	Бали	*n*	gondolaSeq
4	5	$n \le 100$	1 ≤gondolaSeq [i] ≤*n +1*
5	10	$n \le 1000$	1≤gondolaSeq[i] ≤ 5000
6	20	$n \le 100\,000$	1≤gondolaSeq[i] ≤250 000

Приклади

Таблиця 5

Підзадача	gondolaSeq	Повернене значення	replacementSeq
4	(3, 1, 4)	1	(2)
4	(5, 1, 2, 3, 4)	0	( )
5	(2, 3, 4, 9, 6, 7, 1)	2	(5, 8)

Кількість послідовностей замін. У наступних чотирьох підзадачах ви маєте підрахувати кількість можливих послідовностей замін, що призводять до заданої послідовності (що може бути або не бути послідовністю гондол), за модулем 1 000 000 009. Ви маєте реалізувати функцію **countReplacement**.

- **countReplacement(n, inputSeq)**

— n: довжина вхідної послідовності.

— inputSeq: масив довжини *n* inputSeq [i] є елементом *i* вхідної послідовності, для $0 \le i \le n-1$.

— Якщо вхідна послідовність є послідовністю гондол, підрахуйте кількість послідовностей замін, що призводять до цієї послідовності гондол (яка може бути дуже великою), *та поверніть це значення за модулем* 1 000 000 009. Якщо вхідна послідовність не є послідовністю гондол, функція має повертати 0. Якщо вхідна послідовність є послідовністю гондол, але гондоли не ламались, функція має повертати 1.

Підзадачі 7, 8, 9, 10

Таблиця 6

Підзадача	бали	*n*	inputSeq
7	5	$4 \le n \le$ **50**	**1**≤ inputSeq[i]≤*n* +3
8	15	$4 \le n \le$ **50**	**1**≤ inputSeq[i]≤**100,** і принаймні *n — 3* початкових гондол *1,..., n* не ламались.
9	15	$n \le$ **100 000**	**1**≤inputSeq[i]≤ **250000**
10	10	$n \le$ **100 000**	**1**≤inputSeq[i]≤**1 000 000 000**

Приклади наведено у таблиці 7.

Деталі реалізації. Ви маєте відіслати тільки один файл, що має ім'я **gondola.c**, **gondola.cpp** або **gondola.pas**. Цей файл реалізує підпрограми, що описано вище.

Таблиця 7

Підзадача	inputSeq	Повернене значення	Послідовність замін
7	(1, 2, 7, 6)	2	(3, 4, 5) or (4, 5, 3)
8	(2, 3, 4, 12, 6, 7, 1)	1	(5, 8, 9, 10, 11)
9	(4, 7, 4, 7)	0	inputSeq не є послідовністю гондол
10	(3, 4)	2	(1, 2) or (2, 1)

Рекомендації щодо розв'язання

Це проста задача. Навіть якщо здається, що вона містить три підзадачі (як написано в умові), вони включаються одна в одну. Всі три частини (перевірити чи існує розв'язок — знайти якийсь розв'язок — підрахувати всі розв'язки) дуже пов'язані.

Підзадачі 1–3. В підзадачі 1, як тільки ми бачимо першу гондолу (тобто, inputSeq[0]), решта визначається однозначно. Нам тільки треба пройти по послідовності та перевірити, що все співпадає.

- **N=len(sequence)**
- **for n:=l..N: if sequence[n] % N!=(sequence[0]+n) % N: return False**
- **return True**

Той самий код розв'язує і підзадачу 2. Псевдокод для загальної підзадачі 3 буде таким.

- Якщо існує одна з початкових гондол: перевірити, чи знаходяться решта початкових гондол на очікуваних місцях, якщо ні, повернути **false**.
- Повернути **true** якщо всі значення є різними, повернути **false** у іншому випадку.

Підзадачі 4–6. Простий розв'язок підзадачі 4: якщо найбільший номер в послідовності це *n*, то повернути пусту послідовність, інакше повернути одне відсутнє число.

Розв'язок підзадач 5 та 6 базується на тій самій ідеї, різниця в тому, що підзадача 4 дозволяє неефективну реалізацію. Є багато можливих розв'язків, ось один з них.

- Зібрати всі нові гондоли. Для кожної з них визначити, яку початкову гондолу вона замінила.
- Відсортувати ці записи за номером нової гондоли.
- У відсортованому порядку, замінити початкові гондоли новими.

Зауважте, що відсутність нових гондол може вимагати додатково уваги не тільки в учасників, а ще і при перевірці цих підзадач.

Підзадачі 7–10. Прямо підрахувати послідовності замін дуже складно. Одним із шляхів є ставити питання «Скільки послідовностей замін починаються з заміни гондоли x?» для кожного x. Кожен вибір x веде нас у новий стан, що містить на одну менше замінюваних гондол.

Використовуючи цю ідею, можна розв'язати підзадачу 8 за допомогою динамічного програмування: для кожного допустимого стану гондольної системи ми підраховуємо кількість розв'язків для неї.

Підзадачі 9 та 10 вимагають одну додаткову здогадку (це, можливо, єдина каверзна частина задачі, але здогадка не є складною).

Замість аналізу гондоли, що видаляється, ми поглянемо на нову гондолу, що додається. Скільки різних варіантів ми маємо для її розміщення? Якщо нова гондола присутня у фінальній послідовності, її місце визначено однозначно. Інакше, кількість місць, де можна розташувати нову гондолу є просто кількістю місць де потім буде гондола з більшим номером. Додатково ми маємо помножити результат на n, якщо не залишається жодної початкової гондоли. (Можливі всі n циклічних перестановок вихідної послідовності та послідовності замін для різних перестановок вихідної послідовності відрізняються). Алгоритм матиме такий вигляд:

- запустити алгоритм для підзадачі 1–3 щоб перевірити допустимість вхідної послідовності;
- якщо допустима, для кожної заміненої гондоли ми знаходимо початкову гондолу, яка була замінена, сортуємо ці записи за номерами нових гондол;
- загальна кількість можливостей тепер може бути обчислена множенням кількості можливих розташувань кожної гондоли з номерами між $n+1$ та максимальним присутнім номером гондоли. Зауважте, що множення відбувається за модулем;
- у кінці, результат помножимо на n, якщо було замінено всі початкові гондоли.

2. Друг

Ми будуємо соціальну мережу з n людей, пронумерованих 0, … , $n-1$. Деякі пари людей у цій мережі будуть друзями. Якщо людина x стає другом людини y, то людина y також стає другом людини x.

Люди добавляються у мережу за n кроків, які також пронумеровані від 0 до $n-1$. Людина i додається на кроці i. На кроці 0, додається людина 0 як єдина людина у мережі. На кожному з наступних $n-1$ кроків наступна людина додається в мережу *господарем*, яким може бути довільна людина, яку вже додано до мережі. На кроці i ($0<i<n$), господар кроку може додати чергову людину i в мережу за одним з наступних протоколів:

- **IAmYourFriend** робить людину i другом тільки господаря;
- **MyFriendsAreYourFriends** робить людину i другом *кожної* людини, яка є другом господаря у цей момент. Зауважте, що цей протокол не робить людину i другом господаря;
- **WeAreYourFriends** робить людину i другом господаря, а також другом *кожної* людини, яка є другом господаря у цей момент.

Після того, як ми побудували мережу, ми маємо підібрати *вибірку* для опитування, тобто вибрати групу людей з мережі. Оскільки друзі зазвичай мають спільні інтереси, ця вибірка не повинна містити пари людей, що є друзями. Кожна людина має певний *рівень довіри* в опитуваннях, виражений додатнім цілим числом, і нам треба підібрати вибірку з максимальним загальним рівнем довіри.

Приклад (табл. 8)

Спочатку мережа містить тільки людину 0. Господар кроку 1 (людина 0) запрошує нову людину 1, використовуючи протокол IamYourFriend, та вони стають друзями. Господар кроку 2 (знову людина 0) запрошує людину 2, використовуючи MyFriendsAreYourFriends, який робить людину 1 (єдиний друг господаря) єдиним другом людини 2. Господар кроку 3 (людина 1) додає людину 3, використовуючи WeAreYourFriends, що робить людину 3 другом людини 1 (господаря) і людей 0 і 2 (друзі господаря). Кроки 4 і 5 також показано у таблиці вище. Кінцеву мережу показано на рис. 1, на якому числа в кружках показують номери людей, а числа поруч з кружками показують рівень довіри в опитуваннях для цих людей. Вибірка, що складається з людей 3 і 5, має загальний рівень довіри в опитуваннях, що дорівнює 20+15=35, що є максимальним можливим загальним рівнем довіри.

Таблиця 8

Крок	Господар	Протокол	Додані дружні зв'язки
1	0	IAmYourFriend	(1, 0)
2	0	MyFriendsAreYourFriends	(2, 1)
3	1	WeAreYourFriends	(3, 1), (3, 0), (3, 2)
4	2	MyFriendsAreYourFriends	(4, 1), (4, 3)
5	0	IAmYourFriend	(5, 0)

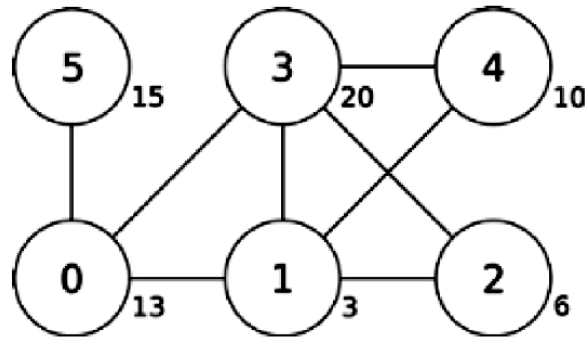

Рис. 1

Задача

Маючи опис кожного кроку та рівень довіри до кожної людини, знайдіть вибірку з максимальним загальним рівнем довіри. Ви маєте реалізувати функцію **findSample**.

- **findSample(n, confidence, host, protocol)**

— n: кількість людей.

— confidence: масив довжини n; confidence [i] задає рівень довіри до людини i.

— host: масив довжини n host [i] задає господаря кроку i.

— protocol: масив довжини n protocol [i] задає код протоколу, що використовується на кроці i ($0<i<n$): 0 для IAmYourFriend, 1 для MyFriendsAreYourFriends, та 2 для WeAreYourFriends.

— Оскільки на кроці 0 немає господаря, host[0] та protocol[0] невизначені і ваша програма не повинна звертатись до них.

— Функція має повертати максимальний можливий загальний рівень довіри для вибірки.

Підзадачі

Деякі підзадачі використовують не всі протоколи, як показано в таблиці 9.

Таблиця 9

Підзадача	Бали	*n*	Рівень довіри	Використаний протокол
1	11	$2 \le n \le 10$	1≤Рівень довіри≤1 000 000	Всі три протоколи
2	8	$2 \le n \le 1000$	1≤Рівень довіри≤1 000 000	Тільки MyFriendsAreYourFriends
3	8	$2 \le n \le 1000$	1≤Рівень довіри≤1 000 000	Тільки WeAreYourFriends
4	19	$2 \le n \le 1000$	1≤Рівень довіри≤1 000 000	Тільки IAmYourFriend
5	23	$2 \le n \le 1000$	Всі рівні довіри дорівнюють 1	Тільки MyFriendsAreYourFriends та IAmYourFriend
6	31	$2 \le n \le 100\,000$	1≤Рівень довіри≤10 000	Всі три протоколи

Деталі реалізації

Ви маєте відіслати тільки один файл, що має ім'я **friend.c**, **friend.cpp** або **friend.pas**. Цей файл має реалізовувати підпрограму, що описана вище.

Рекомендації щодо розв'язання

Опишемо 6 розв язків, 5 для перших підзадач і один загальний.

Розв'язок 1. У підзадачі 1 N не більше 10. Отже, застосуємо бектрекінг для кожної людини — обирати чи не обирати. Складність буде $O(N*2^N)$. Розмір N в інших підзадачах є заладто великим, щоб застосовувати цей алгоритм.

Розв'язок 2. У підзадачі 2 є тільки відносини 'MyFriends AreYourFriends', що формує граф без ребер. Отже, треба обрати всіх людей, складність буде $O(N)$. Для інших підзадач є не тільки ці відносини, отже, цей розв'язок працювати не буде.

Розв'язок 3. У підзадачі 3 є тільки відносини 'WeAreYourFriends', що формує повний граф. Оскільки кожна пара людей з'єднана ребром, треба знайти людину з максимальним рівнем довіри, що має складність $O(N)$. Для інших підзадач є не тільки ці відносини, отже, цей розв'язок працювати не буде.

Розв'язок 4. У підзадачі 4 є тільки відносини 'IamYourFriend', що формує дерево. Отже, застосуємо динамічне програмування на дереві.

Визначимо dp[i][j] як максимальну суму для вузла i зі статусом j, де j=0 означає, що вузол не обрано та j=1 означає, що вузол обрано. Тоді:

- якщо вузол i є листом, то
  - dp[i][0]=0;
  - dp[i][1]= confidence[i];
- інакше
  - dp[i][0]=max(dp[k][0], dp[k][1]), для всіх k, де k є сином i;
  - dp[i][1]=dp[k][0], для всіх k, де k є сином i.

Остаточною відповіддю буде max{dp[root][0]; dp[root][1]), де root є коренем дерева. Для інших підзадач є не тільки відносини 'IamYourFriend', отже, цей розв'язок працювати не буде.

Розв'язок 5. У підзадачі 5 є тільки відносини 'MyFriendsAreYourFriends' і 'IamYourFriend', що формує граф без непарних циклів. Отже, ми отримуємо дводольний граф. З усіма рівнями довіри, що дорівнюють 1, задача стає задачею пошуку максимальної незалежної множини у дводольному графі. Як відомо, множина є незалежною тоді і тільки тоді, коли її доповненням є вершинне покриття. Якщо доповнення незалежної множини не є вершинним покриттям, то існує принаймні одне ребро, що з'єднує вершини u і v, які включаються в незалежну множину, що суперечить визначенню незалежної множини. Очевидно, максимальна незалежна множина є доповненням мінімального вершинного покриття.

За теоремою Кьоніга, у довільному дводольному графі кількість ребер у максимальному паросполученні дорівнює кількості вершин у мінімальному вершинному покритті. Отже, ми можемо застосувати алгоритм пошуку доповнюючого шляху, що знайти паросполучення максимального розміру у дводольному графі, що має складність $O(NE)$ або $O(\sqrt{NE})$, залежно від реалізації, де E є кількістю ребер.

Нехай k буде розміром максимального знайденого паросполучення, відповіддю задачі буде $N-k$, оскільки максимальна незалежна множина є доповненням мінімального вершинного покриття.

Щоб розділити дводольний граф на долі, застосуємо пошук в глибину та помітимо усі парні та непарні вершини, парні утворять одну долю, непарні — іншу.

У підзадачах 2 та 4 відношеннями є 'MyFriendsAreYourFriends' та 'IamYourFriend', але рівні довіри не дорівнюють 1, отже, цей метод буде коректно розв'язувати тільки підзадачу 5.

Розв'язок 6. Будемо обробляти операції у зворотному порядку.

Для кожної операції будемо змінювати граф так, щоб у ньому не залишилося нової людини, але щоб оптимальний розв'язок мав ту саму величину. Спочатку заведемо два значення $p(x)$ та $q(x)$ для кожної людини x, де $p(x)$=confidence[x] та $q(x)$=0. Фактично, p означає *обрати* та q означає *не обирати*.

Щоб спростити позначення, використаємо p для $p(x)$, q для $q(x)$, p' для $p(y)$ та q' для $q(y)$.

- WeAreYourFriends. Щоб відкинути y, ми можемо обрати або x, або y, або жодаого з них.

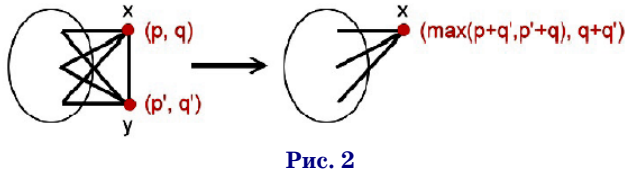

Рис. 2

Обираємо x: $p=p+q'$.
Обираємо y: $p=p'+q$.
Жодного: $q=q+q'$. Отже, $p=\max(p+q'; p'+q)$, $q=q+q'$.

• MyFriendsAreYourFriends. Щоб відкинути y, ми можемо обрати x, обрати y, обрати обох, або жодного.

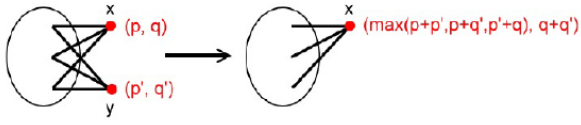

Рис. 3

— Обираємо x: $p=p+q'$.
— Обираємо y: $p=p'+q$.
— Обираємо обох: $p=p+p'$.
— Жодного: $q=q+q'$. Отже, $p=\max(p+q'; p'+q; p+p')$, $q=q+q'$.

• IamYourPriend. Щоб відкинути y, ми можемо обрати x, або взяти чи не взяти y.

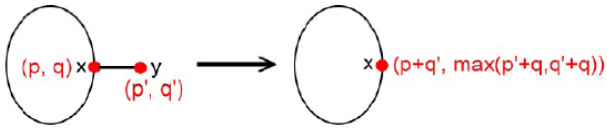

Рис. 4

— Обираємо x: $p=p+q'$
— Обираємо y: $q=p'+q$.
— Жодного: $q=q+q'$. Отже, $p=p+q'$, $q=\max(p'+q; q+q')$.

Відповіддю буде max(p; q) для останньої людини, що залишилась. Складність алгоритму O(N). Цей метод розв'язує всі підзадачі.

3. Відпустка

Жиан-Жиа планує свою наступну відпустку у Тайвані. Під час відпустки Жиан-Жиа їздить з міста до міста та відвідує пам'ятки у цих містах.

У Тайвані є n міст, всі знаходяться впродовж однієї автомагістралі. Міста пронумеровані послідовно від 0 до $n-1$. Для міста i, де $0<i<n-1$, сусідніми містами є $i-1$ та $i+1$. Єдиним містом, сусіднім до міста 0, є місто 1, та єдиним містом, сусіднім до міста $i-1$, є місто $i-2$.

У кожному місті є деяка кількість пам'яток. Жиан-Жиа має d днів відпустки та планує відвідати якомога більше пам'яток. Жиан-Жиа вже обрав місто, з якого він почне свою відпустку. У кожен день відпустки Жиан-Жиа може або переїхати до сусіднього міста, або відвідати всі пам'ятки у місті, де він знаходиться, але не те й інше разом. Жиан-Жиа *ніколи не відвідуватиме пам'ятки у тому самому місті двічі*, навіть якщо він буде знаходить у місті декілька разів. Будь ласка, допоможіть Жиан-Жиа спланувати його відпустку так, щоб він відвідав якомога більше пам'яток.

Приклад

Припустимо, Жиан-Жиа має 7 днів відпустки, є 5 міст (перелічених у таблицю 10), та він починає з міста 2. У перший день Жиан-Жиа відвідує 20 пам'яток у місті 2 (табл. 11). У другий день Жиан-Жиа рухається з міста 2 до міста 3, та у третій день він відвідує 30 пам'яток у місті 3. Після цього Жиан-Жиа проводить наступні три дні, рухаючись з міста 3 до міста 0, та відвідує 10 пам'яток у місті 0 у сьомий день. Загальна кількість пам'яток, відвіданих Жиан-Жиа, буде 20+30+10=60, що є максимальною кількістю пам'яток, що може відвідати Жиан-Жиа за 7 днів, починаючи з міста 2.

***Таблиця* 10**

Місто	Кількість пам'яток
0	10
1	2
2	20
3	30
4	1

***Таблиця* 11**

День	Дія
1	Відвідати пам'ятки у місті 2
2	Їхати з міста 2 до міста 3
3	Відвідати пам'ятки у місті 3
4	Їхати з міста 3 до міста 2
5	Їхати з міста 2 до міста 1
6	Їхати з міста 1 до міста 0
7	Відвідати пам'ятки у місті 0

Задача

Реалізуйте функцію findMaxAttraction, що обчислює максимальну кількість пам'яток, які може відвідати Жиан-Жиа.

• findMaxAttraction(n, start, d, attraction)
— n: кількість міст.
— start: індекс початкового міста.
— d: кількість днів.
— attraction: масив довжини n; attraction[i] є кількістю пам'яток у місті i, для $0\le i\le n-1$.

Функція має повертати максимальну кількість пам'яток, що може відвідати Жиан-Жиа.

Підзадачі

У всіх підзадачах $0\le d\le 2n+[n/2]$, та кількість пам'яток у кожному місті невід'ємна (табл. 12).

Деталі реалізації

Ви маєте відіслати тільки один файл, що має ім'я **holiday.c**, **holiday.cpp** або **holiday.pas**. Цей файл має реалізовувати підпрограму, що описано вище.

***Таблиця* 12**

Підзадача	Бали	n	Максимальна кількість пам'яток у місті	Початкове місто
1	7	$2 \le n \le 20$	1 000 000 000	Без обмежень
2	23	$2 \le n \le 100\,000$	100	Місто 0
3	17	$2 \le n \le 3000$	1 000 000 000	Без обмежень
4	53	$2 \le n \le 100\,000$	1 000 000 000	Без обмежень

Рекомендації щодо розв'язання

Для простоти опису, спочатку розглянемо простий розв'язок, що потребує час $O(n\log n)$, для спеціального випадку початкового міста з індексом 0 для довільного фіксованого d. Тобто start=0 і d є фіксованим числом. Потім розширимо розв'язок, щоб побудувати таблицю розв'язків для всіх можливих d за час $O(n\log^2 n)$. З рештою, опишемо, як розширити цей розв'язок на загальний випадок довільного початкового міста, зберігши асимптотичну складність.

Простий розв'язок для start=0 та довільного фіксованого *d*. Без втрати загальності, припустимо, що ми починаємо у найлівішому місті та рухаємось праворуч. Легко бачити, що нам потрібно рухатись тільки праворуч та немає потреби рухатись ліворуч ні в якому випадку. Припустимо, що в оптимальному розв'язку місто *right* є найправішим містом, до якого ми дісталися. Тоді ми можемо відвідати не більше $d-right$ міст серед міст з мітками 0; 1;...; *right*.

Щоб розв'язок був оптимальним, ми хочемо відвідати $d-right$ міст з найбільшою кількістю пам'яток. Тобто, якщо ми відсортуємо міста 0; 1;...; *right* за кількістю пам'яток, нам потрібно буде знайти суму $d-right$ найбільших скупчень пам'яток.

Використаємо структуру даних, що називається *дерево відрізків* для нашого випадку, хоча може здаватись, що така структура даних не є необхідною у цьому дуже спеціальному випадку. Однак ми зрозуміємо, навіщо використовувати цю структуру, коли перейдемо до наступного випадку. Деревом відрізків є повне кореневе бінарне дерево з листками, що містять помітку, чи є лист активним, та деяке значення. Для кожного внутрішнього вузла v, воно містить суму значень усіх активних листків у піддереві з коренем у v. Кожний внутрішній вузол також містить кількість активних листків у піддереві з коренем у ньому.

Нехай дерево відрізків містить n листків. Зауважте, що нам потрібно буде додати пусті листки, якщо n не є ступенем 2. Також припустимо, що значення у листках розташовані у порядку не збільшення зліва направо. Щоб підтримувати таку структуру даних, потрібно $O(\log n)$ часу, щоб активувати або деактивувати довільний лист. Також потрібно $O(\log n)$, часу щоб знайти суму значень у найбільших x активних листках для довільного заданого x.

Спочатку відсортуємо міста за кількістю пам'яток в них, у незростаючому порядку. Потім помістимо їх як листи у дерево відрізків, усі листки — не активні. Кількість пам'яток буде значенням листка. Ініціалізація потребує $O(n\log n)$ часу. Ми будемо активувати листок, коли ми до нього доходимо під час пошуку розв'язку. Перебираємо всі можливі найправіші міста, до яких ми можемо дістатись. Загальна складність алгоритму буде $O(n\log n)$.

Проміжний розв'язок для $start$=0 та довільних d Розглянемо розв'язок для проміжного випадку. Нехай $f(d)$ буде міткою міста, куди ми перемістимося за d днів, так щоб побачити найбільшу кількість пам'яток. Зауважте, що $f(d)$ не обов'язково буду унікальним. Якщо є декілька варіантів, оберем один з найменшою міткою. Тепер ми побудуємо таблицю для всіх можливих значень d. Використаємо рекурсивний підхід розділяй та володарюй. Нехай M буде максимальним значенням d. Для простоти, нехай M буде ступенем 2. Щоб обчислити розв'язки для $f(1)$; $f(2)$; ... ; $f(M)$, ми спочатку знайдемо $f(M/2)$, використовуючи наш попередній алгоритм, що проходить по всіх містах від 0 до n. Потім рекурсивно обчислимо $f(1)$; $f(2)$;...; $f(M/2-1)$ у одній гілці, розглядаючи тільки міста від 0 до $f(M/2)$, та $f(M/2+1)$; $f(M/2+2)$; ...; f(M) у іншій гілці, розглядаючи тільки міста від $f(M/2)$ до n. У гілці, де обчислюються $f(1)$; $f(2)$; ...; $f(M/2-1)$, ми спочатку підрахуємо $f(M/4)$ серед всіх від 0 до $f(M/2)$. Загалом обсяг часу, що буде використано на кожен з рівнів рекурсивних викликів буде $O(n\log n)$. Всього є $O(\log n)$ рівнів. Отже, загальна складність буде $O(n\log^2 n)$.

У цьому розв'язку зрозуміло, чому потрібне дерево відрізків. По-перше, його треба ініціалізувати всього один раз. По-друге, ми можемо активувати і деактивувати вузол, щоб врахувати які листи ми використовуємо на кожному рівні рекурсії.

Загальний розв'язок з довільним місцем початку. Тепер ми можемо перейти до загального випадку. Помітимо, що значення *start* є довільним, розв'язок може бути знайдений у одному з двох випадків. Або ми спочатку рухаємось праворуч від початкового міста, потім рухаємось ліворуч та зупиняємось у місті ліворуч від *start*. Або ми спочатку рухаємось ліворуч від початкового міста, потім рухаємось праворуч та зупиняємось у місті праворуч від *start*. Тобто напрямок руху ми змінюємо лише один раз. Не втрачаючи загальності, розглянемо перший сценарій. Спочатку використаємо проміжний розв'язок, щоб знайти всі $f(t)$ для всіх можливих значень t. Потім також використаємо проміжний розв'язок, щоб знайти $g(t)$ для всіх можливих значень t, де $g(t)$ є містом, до якого ми дійдемо у оптимальному розв'язку, якщо будемо рухатись ліворуч з початкового міста $start-1$. Спочатку пройдемось по значеннях d_0 — кількості днів, ми бажаємо провести, рухаючись і відвідуючи міста праворуч і включаючи *start*. В проміжному розв'язку ми маємо $f(d_0)$. Потім ми знаємо, що можемо провести $d-d_0-(f(d_0)-start)-1$ днів у містах ліворуч від *start*. Загальна складність буде $O(n\log^2 n)$.

★ ★ ★