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

Гуржій А.М., Бондаренко В.В.

Закінчення, початок у №6 за 2009 рік

ЗАВДАННЯ ДРУГОГО ТУРУ

1. ПАРКОВКА

Умова. Парковка має N місць, пронумерованих від 1 до N включно. Парковка відкривається пустою кожен ранок і працює протягом дня так. Коли автомобіль приїжджає на парковку, паркувальник перевіряє, чи є вільні місця. Якщо таких немає, автомобіль чекає біля в'їзду до тих пір, поки звільниться якесь місце. Якщо є вільне місце, як тільки воно звільняється, автомобіль займає вільне паркувальне місце. Якщо є декілька вільних паркувальних місць, автомобіль займає місце з найменшим номером. Коли приїздять паркуватись інші автомобілі, але вже є автомобіль, що очікує, вони шикуються в чергу на в'їзді у тому порядку, в якому приїхали. Після того, як звільняється паркувальне місце, його займає перший автомобіль із черги (тобто той, який прибув паркуватись перший).

Вартість паркування одного автомобіля у доларах визначається як добуток ваги цього автомобіля у кілограмах на тариф його паркувального місця. Вартість паркування автомобіля не залежить від того, скільки часу цей автомобіль знаходиться на парковці.

Паркувальник знає, що сьогодні на парковку приїде M автомобілів, і він знає порядок їх приїзду і від'їзду. Допоможіть йому підрахувати, скільки доларів він сьогодні заробить.

Завдання. Напишіть програму, яка за заданими тарифами паркувальних місць, вагою автомобілів і порядком, у якому автомобілі приїздять та від'їжджають, визначає дохід парковки в доларах.

Обмеження. Кількість паркувальних місць — $1 \le N \le 100$, кількість автомобілів — $1 \le M \le 2000$, тариф паркувального місця з номером s в доларах за кілограм — $1 \le R_s \le 100$, вага автомобіля з номером k в кілограмах — $1 \le W_k \le 10000$.

Вхідні дані. Ваша програма має читати зі стандартного потоку введення такі дані:

- Перший рядок містить цілі числа N і M, розділені пропуском.
- Наступні N рядків описують тарифи паркувальних місць, s-й з цих рядків містить одне ціле число R_s — тариф паркувального місця з номером s в доларах за кілограм.
- Наступні M рядків описують вагу автомобілів. Автомобілі пронумеровані у довільному порядку від 1 до M включно, k-й з цих M рядків містить одне ціле число W_k — вагу автомобіля з номером k в кілограмах.
- Наступні $2 \cdot M$ рядків описують приїзд і від'їзд усіх автомобілів у хронологічному порядку. Додатне ціле число i показує, що автомобіль з номером i приїжджає на парковку. Від'ємне ціле число — i показує, що автомобіль з номером i виїздить з парковки. Ніякий автомобіль не виїздить з парковки до свого приїзду, і всі автомобілі від 1 до M включно з'являться у цій послідовності рядків рівно 2 рази, один раз як автомобіль, що приїжджає, і другий — як той, що виїздить. До того ж, ніякий з автомобілів не виїде з парковки, доки не займе місце на парковці (тобто жоден автомобіль не поїде, поки стоїть у черзі).

Вихідні дані. Ваша програма повинна записати у стандартний потік виведення один рядок, у якому буде міститись одне ціле число: загальна кількість доларів, яку заробить сьогодні паркувальник.

Оцінювання. Для набору тестів загальною вартістю 40 балів завжди буде хоча б одне вільне паркувальне місце для кожного автомобіля, що приїжджає. У цих тестах жоден автомобіль не має очікувати звільнення паркувального місця.

Приклад вхідних та вихідних даних

Приклад введення	Приклад виведення
3 4 2 3 5 200 100 300 800 3 2 −3 1 4 −4 −2 −1	5300

Автомобіль з номером 3 займає місце з номером 1 і платить $300 \times 2 = 600$ доларів. Автомобіль з номером 2 займає місце з номером 2 і платить $100 \times 3 = 300$ доларів. Автомобіль з номером 1 займає місце з номером 1 (яке звільнив автомобіль з номером 3) і платить $200 \times 2 = 400$ доларів. Автомобіль з номером 4 займає місце з номером 3 (останнє, що залишилось) і платить $800 \times 5 = 4000$ доларів.

Приклад введення	Приклад виведення
2 4 5 2 100 500 1000 2000 3 1 2 4 −1 −3 −2 −4	16200

Автомобіль з номером 3 займає місце з номером 1 і платить $1000\times5=5000$ доларів. Автомобіль з номером 1 займає місце з номером 2 і платить $100\times2=200$ доларів. Автомобіль з номером 2 приїжджає і має чекати на в'їзді. Автомобіль з номером 4 приїжджає і має чекати на в'їзді за автомобілем з номером 2. Коли автомобіль з номером 1 звільняє своє паркувальне місце, автомобіль з номером 2 займає його місце і платить $500\times2=1000$ доларів. Коли автомобіль з номером 3 звільняє своє паркувальне місце, автомобіль з номером 4 займає його місце і платить $2000\times5=10000$ доларів.

Рекомендації щодо розв'язання

Задача розв'язується простою симуляцією, але структури даних, що треба використати, не є такими простими. Проста симуляція, що не потребує ніяких структур даних, крім масиву сталого розміру, буде слідкувати за:

- кожним автомобілем, його станом (відсутній, у черзі, припаркований), його місцем паркування (якщо припаркований) та часом прибуття (якщо в черзі);
- кожним місцем паркування, чи є там припаркований автомобіль.

Тепер можна обробляти вхідні події по одній за раз. Коли автомобіль прибуває, пройдемо по всіх паркувальних місцях, щоб знайти перше вільне. Якщо таке є, припаркуємо там автомобіль. Інакше, автомобіль стає в чергу, запишемо його час прибуття.

Коли автомобіль виїздить, він буде замінений автомобілем на початку черги (якщо такий є). Пройдемо по всіх автомобілях та знайдемо той, що прибув раніше всіх та все ще знаходиться в черзі. Якщо такий є, припаркуємо його у паркувальному місці, яке тільки що звільнилося, та позначимо, що він більше не в черзі.

Цей розв'язок виконується за час $O(M^2+MN)$. Його можна покращити: зберігання черги в окремому масиві зменшить час до $O(MN)$, а зберігання вільних паркувальних місць у бінарній кучі зменшить час до $O(M\log N)$. Однак, ці оптимізації не потрібні для отримання повного балу.

2. ВЕДМІДЬ МИШКО

Умова. Ведмідь Мишко знайшов маленький скарб — схований горщик меду! Він з насолодою їв цей мед, але раптом одна бджола помітила його та забила тривогу. Мишко знає, що після цього бджоли вилетять зі своїх вуликів і будуть розлітатися навкруги, щоб наздогнати його. Він знає, що треба кидати горщик і швидко йти додому, але мед такий солодкий, що Мишко не хоче кидати його їсти занадто рано. Допоможіть Мишку визначити останній можливий момент, коли він може перестати їсти мед. Ліс подається мапою у вигляді квадратної сітки, яка складається з $N\times N$ одиничних клітин, сторони яких паралельні напрямкам «північ-південь»та «захід-схід». Кожна клітина лісу зайнята або деревом, або травою, або вуликом, або домом Мишка. Дві клітини називаються суміжними, якщо одна з них знаходиться безпосередньо на півночі, півдні, сході чи заході від іншої, але не по діагоналі. Мишко неповоротний, тому він може пересуватись тільки у суміжну клітину. Мишко може пересуватись тільки по клітинах з травою і не може пересуватись по клітинах з деревами або вуликами. Також він не може пересуватись більше, ніж на S клітин за хвилину.

У момент, коли пролунала тривога, Мишко знаходиться у клітині з травою, де він знайшов горщик меду, а усі бджоли — у клітинах, де розташовано вулики (у лісі може бути більше одного вулика). З цього моменту, протягом кожної наступної хвилини відбуваються такі події у такому порядку:

- Якщо Мишко все ще їсть мед, він вирішує, чи буде він продовжувати їсти, чи буде йти. Якщо він продовжує їсти, то він не пересувається всю хвилину. Інакше, він одразу йде і пересувається по лісу не більше, як на S клітин, як описано вище. Мишко не може брати з собою мед, і як тільки він пішов, він вже не може його їсти.
- Як тільки Мишко закінчує їсти або пересуватися протягом хвилини, бджоли розлітаються на одну клітину далі, займаючи тільки клітини з травою. Точніше, бджоли розлітаються у всі клітини з травою, що є суміжними з довільною клітиною, де вже є бджоли. Як тільки у клітині з'являються бджоли, вони там залишаються назавжди (бджоли не пересуваються, а розповсюджуються).

Іншими словами, бджоли розлітаються так: як тільки звучить тривога, бджоли знаходяться у клітинах, де розташовано вулики. У кінці першої хвилини вони займають усі клітини з травою, що суміжні з вуликами, та залишаються у тих клітинах, де розташовано вулики. У кінці другої хвилини бджоли займають усі клітини з травою, що є суміжними із суміжними з клітинами з вуликами і так далі. Маючи достатньо часу, бджоли займуть усі клітини з травою, яких вони можуть досягнути.

Ні Мишко, ні бджоли не можуть лишати межі лісу. Також зверніть увагу, що згідно з описаними правилами, Мишко їсть мед цілу кількість хвилин.

Бджоли наздоганяють Мишка, якщо в якийсь момент часу Мишко залишається у клітині, що зайнята бджолами.

Завдання. Напишіть програму, яка за мапою лісу визначає найбільшу кількість хвилин, протягом яких Мишко може продовжувати їсти мед у своєму початковому розташуванні, весь час маючи можливість потрапити додому до того, як бджоли його наздоженуть.

Обмеження. $1\le N\le800$ — розмір (довжина сторони) мапи лісу, $1\le S\le1000$ — максимальна кількість пересувань, які може робити Мишко за кожну хвилину.

Вхідні дані. Ваша програма має читати зі стандартного потоку введення такі дані:

- Перший рядок містить цілі числа N і S, розділені пропуском.
- Наступні N рядків задають мапу лісу. Кожен із цих рядків містить N символів, кожен символ задає одну клітину на сітці. Можливі символи та їх значення описані нижче:

  T означає клітину з деревом;

  G означає клітину з травою;

M означає початкове розташування Мишка і горщика меду у клітині з травою;

D означає клітину, де розташовано дім Мишка, в який Мишко може потрапити, а бджоли не мажуть;

H означає клітину з вуликом.

Примітка. Гарантується, що мапа лісу містить рівно одну букву M, рівно одну букву D і, принаймні, одну букву H. Також гарантується, що існує послідовність суміжних клітин G, які з'єднують клітину з початковим розташуванням Мишка і клітину, де розташовано дім Мишка, також як і послідовність суміжних клітин G, які з'єднують хоча б одну з клітин з вуликом з клітиною з горщиком (тобто з клітиною з початковим розташуванням Мишка). Послідовності можуть бути і з довжиною, що дорівнює нулю, у випадку, коли клітина з домом Мишка або клітина з вуликом є суміжними з клітиною початкового розташування Мишка. Також зауважте, що бджоли не можуть розповсюджуватись через клітину з домом Мишка. Для бджіл вона як клітина з деревом.

Вихідні дані. Ваша програма повинна записати у стандартний потік виведення один рядок, у якому буде міститись одне ціле число: максимально можлива кількість хвилин, протягом яких Мишко може продовжувати їсти мед у початковому розташуванні, маючи можливість безпечно повернутися додому.

Якщо Мишко не має можливості повернутися додому до того, як бджоли його наздоженуть, ваша програма має виводити від'ємне число -1.

Оцінювання. Для набору тестів загальною вартістю 40 балів завжди N не буде перевищувати 60.

Приклад вхідних та вихідних даних

Приклад введення	Приклад виведення
7 3 TTTTTTT TGGGGGT TGGGGGT MGGGGGD TGGGGGT TGGGGGT THHHHHT	1

Мишко може їсти мед одну хвилину, після чого може йти наступні 2 хвилини направо ($\rightarrow \rightarrow \rightarrow$) до свого дому найкоротшим шляхом, безпечним від бджіл.

Приклад введення	Приклад виведення
7 3 TTTTTTT TGGGGGT TGGGGGT MGGGGGD TGGGGGT TGGGGGT TGHHGGT	2

Мишко може їсти мед дві хвилини, після чого може зробити кроки $\rightarrow \uparrow \rightarrow$ протягом третьої хвилини, після чого кроки $\rightarrow \rightarrow \rightarrow$ протягом четвертої хвилини і кроки $\downarrow \rightarrow$ протягом п'ятої хвилини.

Рекомендації щодо розв'язання

Розв'язок 1. По-перше, працювати з дробовими числами незручно, будемо вимірювати час в одиницях, що дорівнюють $1/S$ секунди, назвем їх *тіки*. Бджолам потрібно S тіків, щоб пересунутись від однієї клітини до іншої, в той час, коли Мишку потрібен один тік.

Спробуємо спочатку розв'язати простішу задачу. Нехай ми знаємо, коли Мишко кидає мед, чи може він безпечно дістатись дому? Якщо ми зможемо розв'язати цю задачу, то ми зможемо використати її всередині бінарного пошуку, щоб знайти останній момент, коли він може кидати мед.

Рухи Мишка залежать від бджіл, але рухи бджіл є фіксованими, почнемо з них. Стандартний пошук у ширину відповість на питання, коли бджоли досягнуть кожної клітини з травою (це просто означає симуляцію розповсюдження бджіл з часом).

Далі ми можемо виконати схожий пошук в ширину для Мишка та відповісти на питання «Як скоро (якщо взагалі) може Мишко досягнути кожної клітини?» Це може бути реалізовано дуже схоже на випадок з бджолами, окрім того, що треба виключати ті кроки, що приведуть Мишка у клітини, де його одразу спіймають.

Кожен з цих пошуків в ширину можна реалізувати за час $O(N^2)$. Умова задачі гарантує, що Мишко буде схоплений бджолами, якщо він залишиться з медом. Бджолам потрібно $O(N^2)$ секунд, щоб захопити всі клітини, куди вони можуть потрапити. Нам цікаво лише ціле число секунд як результат бінарного пошуку. Отже, діапазон пошуку буде $O(N^2)$ та складність всього розв'язку буде $O(N^2 \log N)$.

Розв'язок 2. Замість використання бінарного пошуку ми можемо використати більш складний метод для прямого визначення оптимального часу, коли треба покидати клітину. Бджоли обробляються, як у першому розв'язку. Однак, замість проходження від меду до дому, почнемо з дому. Оскільки дома безпечно, немає обмеження, як пізно Мишко може туди прибути.

Тепер припустимо, що для деякої клітини Y Мишко має її покинути не пізніше ніж t тіків (з моменту, як пролунала тривога), щоб безпечно дістатись дому. Якщо X є сусідньою до Y клітиною, яким буде останній момент часу, що Мишко може покинути клітину X та безпечно дістатись дому через Y? Зрозуміло, що $t-1$ є верхньою границею, інакше він потрапить до Y надто пізно. Однак, він має також покинути Y до того, як туди дістануться бджоли. Останнім моментом буде просто мінімум з двох обмежень. Тепер можна виконувати пріоритетний пошук: симулювати у напрямку зворотнього часу, слідкуючи за останнім моментом, коли можна покинути кожну клітину (пам'ятаючи, що X має інших сусідів, та можливо краще виходити через них, ніж через Y).

Складність цього алгоритму залежить від черги з пріоритетами, що використовується для впорядкування клітин за часом, коли їх треба покидати. Бінарна куча дає реалізацію $O(N^2 \log N)$ і цього достатньо, щоб набрати повний бал. Однак можна показати, що кількість різних приорітетів, що знаходяться у черзі у довільний момент часу є $O(1)$, що робить можливим розв'язок за $O(N^2)$.

3. РЕГІОНИ

Умова. Агенція з Розвитку Регіонів Організації Об'єднаних Націй (АРРООН) має чітко визначену організаційну структуру. В АРРООН працює N чоловік, кожен з яких приїхав із одного з R різних географічних регіонів світу. Регіони пронумеровані від 1 до R включно у довільному порядку. Працівники мають номери від 1 до N включно у порядку зменшення їх віку. При цьому робітник з номером 1 (Голова) є найстаршим з них. Кожний робітник, крім Голови, має одного безпосереднього керівника. Керівник завжди старший за робітника, яким він керує.

Будемо казати, що робітник A є менеджером робітника B, якщо робітник A є або безпосереднім керівником робітника B, або менеджером безпосереднього начальника робітника B. Наприклад, Голова є менеджером кожного іншого робітника. Крім того, очевидно, що ніякі два робітники не можуть бути менеджерами один одного.

Нажаль, Бюро Досліджень Організації Об'єднаних Націй (БДООН) нещодавно отримало декілька скарг, що АРРООН має не збалансовану структуру, що більш вигідна для одних регіонів, і менш вигідна для інших. Для дослідження цих випадків БДООН збирається створити комп'ютерну систему, що буде отримувати опис організаційної структури АРРООН і буде відповідати на запити такого вигляду: за двома регіонами r_1 і r_2 знайти кількість таких пар робітників (e_1, e_2), що робітник e_1 приїхав з регіону r_1, робітник e_2 з регіону r_2, і, при цьому, e_1 є менеджером e_2. Результат виконання запиту — одне ціле число, кількість різних пар (e_1, e_2), що відповідають описаним умовам.

Завдання. Напишіть програму, яка за заданими регіонами, з яких приїхали робітники агентства, а також за інформацією про те, хто є чиїм безпосереднім керівником, інтерактивно відповідає на запити, що описані вище.

Обмеження. $1 \le N \le 200\,000$ — кількість робітників, $1 \le R \le 25\,000$ — кількість регіонів, $1 \le Q \le 200\,000$ — кількість запитів, які повинна обробити ваша програма, $1 \le H_k \le R$ — регіон, з якого приїхав робітник з номером k (де $1 \le k \le N$), $1 \le S_k \le k$ — безпосередній керівник робітника з номером k (де $2 \le k \le N$), $1 \le r_1, r_2 \le R$ — регіони, за якими робиться запит.

Вхідні дані. Ваша програма має прочитати зі стандартного потоку такі дані:

- Перший рядок містить в указаному порядку цілі числа N, R та Q, розділені одиночними пропусками.
- Наступні N рядків містять опис N робітників агентства в порядку від старших до молодших, k-ий з цих рядків містить опис робітника з номером k. Перший з цих рядків (тобто той, що містить опис Голови) складається з одного цілого числа H_1 — номера регіону, з якого приїхав Голова. Кожен з наступних $(N-1)$ рядків містить два цілих числа, розділених одним пропуском: ціле число S_k — номер безпосереднього керівника робітника з номером k і ціле число H_k — номер регіону, з якого приїхав робітник з номером k.

Інтерактивність Після того, як ваша програма прочитала вхідні дані, вона повинна розпочати в інтерактивному режимі читати запити із стандартного потоку введення і виводити результати обробки запитів до стандартного вихідного потоку. На кожен з Q запитів необхідно відповідати окремо, тобто ваша програма повинна вивести результат обробки прийнятого запиту до того, як вона отримає наступний запит.

Кожен запит представлений в одному рядку стандартного потоку введення і складається з двох цілих чисел — номерів регіонів r_1 і r_2. Ці числа розділені пропуском.

Відповіддю на кожний запит є один рядок, що виведений до стандартного потоку виведення. Цей рядок повинен містити одне ціле число — кількість пар робітників АРРООН e_1, e_2 таких, що робітник з номером e_1 приїхав із регіону з номером r_1, робітник з номером e_2 приїхав із регіону з номером r_2 і робітник з номером e_1 є менеджером робітника з номером r_2.

Зауваження. Тести будуть такими, що правильна відповідь на будь-який запит, заданий в стандартному потоці введення, буде завжди меншою 1 000 000.

Ще одне важливе зауваження. Щоб правильно взаємодіяти із системою оцінювання, вам необхідно очистити буфер стандартного потоку виведення після відповіді на кожен запит. Крім того, необхідно уникати операцій зі вхідним потоком, які можуть його заблокувати, таких, наприклад, як **scanf ("%d\n")**. Будь-ласка, подивіться сторінки з технічною документацією, на яких міститься опис того, як це правильно робити.

Оцінювання

1. Для набору тестів загальною вартістю 30 балів R не буде перевищувати 500.

2. Для набору тестів загальною вартість 55 балів немає регіонів, з яких би приїхало більше, ніж 500 робітників.

3. Тести, у яких виконуються обидві умови 1 і 2, мають вартість 15 балів.

4. Тести, у яких виконується хоча б одна з умов 1 і 2, мають вартість 70 балів.

Приклад вхідних та вихідних даних

Приклад введення	Приклад виведення
6 3 4 1 1 2 1 3 2 3 2 3 5 1 1 2 1 3 2 3 3 1	1 [очищення буферу виведення] 3 [очищення буферу виведення] 2 [очищення буферу виведення] 1 [очищення буферу виведення]

Рекомендації щодо розв'язання

Хоча працівникам уже призначено номери на вході, їх можна змінити так, щоб було зручніше. Відношення підпорядкованості зрозуміло організує працівників у дерево. Присвоїмо працівникам нові номе-

ри у порядку передвпорядкованого обходу дерева (передвпорядкований обхід дерева спершу обробляє корінь дерева, потім рекурсивно обробляє по черзі всі піддерева). Рисунок ілюструє таку нумерацію.

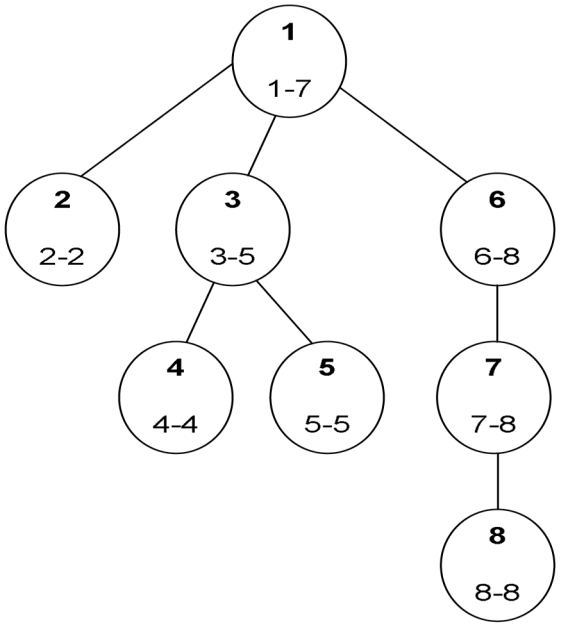

Рис. 1. Приклад нумерації у порядку передвпорядкованого обходу. Числа внизу показують діапазон номерів співробітників у кожному піддереві

Зручною властивістю цієї нумерації є те, що всі працівники у піддереві мають послідовні номери. Для заданого працівника e нехай $[e]$ буде діапазоном номерів працівників, якими керує e. Зауважимо, що для регіону ми можемо побудувати впорядкований масив всіх кінців інтервалів з цього регіону та список всіх працівників з цього регіону. Це можна зробити за лінійний час, коли присвоюються номери.

Тепер розглянемо, як можна відповідати на запити (r_1, r_2). Нехай розміри регіонів будуть відповідно S_1 і S_2. Маючи цю структуру даних, природнім розв'язком є розглянути кожну пару працівників (e_1, e_2) з цих регіонів та перевірити, чи лежить e_2 в інтервалі $[e_1]$. Однак, це потребує часу (S_1S_2) за кожен запит, що може бути покращено.

Кінці інтервалів для регіону r_1 ділять цілі числа на неперервні блоки. Всі працівники у блоці мають того самого менеджера r_1, ми можемо попередньо обчислити кількість таких менеджерів для кожного такого блоку. Це надасть нам можливість швидше відповідати на запити. Замість того, щоб порівнювати всіх працівників з r_2 з кожним блоком для r_1, ми можемо помітити, що обидва вони впорядковані за ідентифікаторами. Отже, можна підтримувати індекс для кожного списку і на кожному кроці просуватися по тому індексу, який відстає від іншого. Оскільки кожен індекс проходить по списку один раз, це займає час $O(S_1+S_2)$.

Використання тільки цього механізму запитів може також потребувати часу $O(NQ)$, оскільки всі запити можуть включати великі регіони. Однак, цього достатньо, щоб заробити бали за тести, де немає регіонів з більш ніж 500 працівників.

Передобчислені запити. У алгоритмі запитів, що наведено вище, також можна замінити список працівників з r_2 усім списком працівників, тобто обчислити відповіді на всі запити для окремого r_1. Це також вимагає тільки один прохід по блоках для r_1, отже займає $O(N)$ часу, щоб знайти всі відповіді для окремого r_1. Аналогічно, можна пройти по всіх кінцях інтервалів, зафіксувавши r_2, що дасть відповіді для окремого r_2.

Це дозволить обчислити попередньо усі можливі запити, використавши час $O(RN)$ та пам'ять $O(R^2)$. Цього достатньо, щоб отримати бали за тести, де $R \le 500$.

Наведений алгоритм є занадто повільним та використовує дуже багато пам'яті, щоб розв'язати усі тести. Однак, немає необхідності попередньо обчислювати всі відповіді, достатньо обчислити тільки найбільш дорогі. Будемо обчислювати відповіді тільки для регіонів розміром принаймні c. Очевидно є не більше N/c таких регіонів, це займе $O(N^2/c)$ часу та $O(RN/c)$ пам'яті. Решта запитів стосується тільки малих регіонів, їх можна обробити за час $O(Qc)$. Вибір $c = \sqrt{N}$ дає час $O(N\sqrt{N} + Q\sqrt{N})$ та пам'ять $O(R\sqrt{N})$, чого достатньо для повного розв'язку.

Кешування запитів. Альтернативою передобчисленню є кешування результатів усіх запитів та видача відповіді з кешу, якщо такий запит уже був. Нехай Q' буде кількість унікальних запитів. Вартість підтримки кешу запитів залежить від використаної структури даних; балансоване бінарне дерево додає до складності $O(Q\log N)$.

Комбінація кешування за алгоритмом $O(S_1+S_2)$ отримає бали за тести, в яких або не більше 500 працівників на регіон (навіть без кешування), також як і за тести, де не більше 500 регіонів (оскільки загальна вартість всіх різних запитів є разом $O(RN)$).

Щоб отримати вищий результат з кешуванням і без передобчислення, потрібно скористатись кращим методом обчислення запитів. Припустимо, що ми маємо блок в r_1 та хочемо знайти всіх відповідних працівників з r_2. Оскільки до цього ми використовували лінійний прохід по працівниках з r_2, тепер ми можемо використати бінарний пошук, щоб знайти початок і кінець діапазону за час $O(\log S_2)$ time. Це дозволить відповісти на весь запит за час $O(S_1\log S_2)$. Аналогічна зміна (бінарний пошук по блоках для кожного працівника в r_2) дає час $O(S_2\log S_1)$ для кожного запиту.

Тепер, відповідаючи на запити, оберемо кращий з $O(S_1\log S_2)$, $O(S_2\log S_1)$ та $O(S_1+S_2)$ механізмів запитів. Щоб встановити верхню границю на час виконання, ми зробимо припущення відносно того, який тип запитів буде використано для яких окремих типів запитів.

Знову, розділимо задачу на задачу для великих регіонів з принаймні c працівниками та інших. Для запитів, що стосуються одного з великих регіонів, використаємо алгоритм з оцінкою $O(A\log B)$, де A та B є, відповідно, менше й більше з S_1 та S_2. Кешування запитів забезпечує те, що ця частина дасть не більше,

ніж $O(N\log N/c)$ часу. Для решти запитів використаємо алгоритм $O(S_1+S_2)$. Менші регіони мають не більше c працівники, це дає час $O(Qc)$.

Оптимальне значення c виходить, коли обидві частини відповідають за однаковий час. Пошук оптимального c дає границю

$$O(N\sqrt{Q'\log N})$$

для відповідей на запити, що не повторюються; комбінуючи з витратами на підтримку кешу запитів, це дасть нам алгоритм зі складністю

$$O(N\sqrt{Q'\log N}+Q\log N)$$

за часом та $O(N+Q')$ за пам'яттю.

Часові оцінки для цього розв'язку є гіршими, ніж для першого розв'язку, але на практиці цей розв'язок працює з приблизно тією ж швидкістю та використовує значно менше пам'яті.

4. КОМІВОЯЖЕР

Умова. Комівояжер вирішив, що задача визначення оптимального розкладу його руху по дорогах належить до нерозв'язних обчислювальних проблем. Тому він зайнявся бізнесом, використовуючи лише рух по річці Дунай, що має пряме русло. У нього є моторний човен, який може довезти його з будь-якого міста на річці до будь-якого іншого, не витративши на це абсолютно ніякого часу. Нажаль, човен споживає дуже багато пального. На кожен метр проїзду проти течії (до джерела) споживається пальне вартістю U доларів, а на кожен метр проїзду за течією річки (від джерела) — D доларів.

Комівояжер хоче відвідати N ярмарок, які проводитимуться в різних місцях, розташованих уздовж річки. Кожна ярмарка триває рівно один день. Для кожної ярмарки з номером X комівояжер знає її дату проведення T_x, що вимірюється в днях з моменту покупки комівояжером човна. Йому також відомі місце розташування ярмарка L_x, що вимірюється в метрах від джерела річки в напрямку вниз за течією, і кількість доларів M_X, які він виручить, якщо відвідає цю ярмарку. Він починає і закінчує подорож у своєму будинку, розташованому в S метрах від джерела річки в напрямку вниз за течією.

Допоможіть комівояжеру вибрати, які ярмарки йому варто відвідати (якщо потрібно) і в якому порядку, щоб отримати максимальний загальний прибуток після закінчення подорожі. Загальний прибуток комівояжера обчислюється як різниця між сумою доларів, яку він виручив від відвідування ярмарок, та сумою доларів, витраченої ним на пальне при пересуванні вгору і вниз річкою.

Майте на увазі, що якщо ярмарка A проводиться раніше ярмарки B, то комівояжер може відвідати їх тільки в порядку їх дат проведення (тобто він не може відвідати ярмарку B раніше, ніж ярмарку A). Якщо дві ярмарки проводяться в один день, то комівояжер може відвідати ці дві ярмарки в будь-якому порядку в той же день. Немає обмежень на кількість ярмарок, відвідуваних в один день, але, природно, комівояжер не може відвідати одну ярмарку двічі, отримавши подвійний прибуток. При цьому він може проїжджати через місця вже відвіданих ярмарок, не отримуючи ніякого прибутку.

Завдання. Напишіть програму, яка за заданими датами проведення ярмарок, їх місцем розташування і виручки, яку можна отримати від їх відвідування, а також місця розташування будинку комівояжера і вартості переїзду, визначить максимальний можливий прибуток, який комівояжер може одержати після закінчення своєї подорожі.

Обмеження $1\le N\le 500\,000$ — кількість ярмарок, $1\le D\le U\le 10$ — вартість переміщення на один метр вгору за течією річки (U) і вниз за течією річки (D), $1\le S\le 500\,001$ — місце розташування будинку комівояжера, $1\le T_k\le 500\,000$ — день, в який проводиться ярмарка з номером k, $1\le L_k\le 500\,001$ — місце розташування ярмарки з номером k, $1\le M_k\le 4\,000$ — кількість доларів, які комівояжер отримає в результаті відвідування ярмарки з номером k.

Вхідні дані. Ваша програма має читати зі стандартного потоку введення такі дані:

- Перший рядок містить у зазначеному порядку цілі числа N, U, D і S, розділені одиночними пропусками.
- Наступні N рядків описують N ярмарок у довільному порядку, k-тий з цих N рядків описує k-у ярмарку і містить три цілих числа, розділених одиночними пропусками: день проведення ярмарки T_k, його місце розташування L_k і прибуток, який може отримати комівояжер від відвідування цієї ярмарки M_k.

Зауваження. Усі місця розташування, зазначені у вхідних даних, будуть різні. Зокрема, ніякі дві ярмарки не будуть проводитися в одному місці і ніяка ярмарка не буде проводитися в тому ж місці, де знаходиться будинок комівояжера.

Вихідні дані. Ваша програма повинна записати у стандартний потік виведення один рядок, що містить одне ціле число: максимальний прибуток, який комівояжер може одержати після закінчення своєї подорожі.

Оцінювання

1. Для набору тестів загальною вартістю 60 балів ніякі дві ярмарки не будуть проводитися в один і той же день.

2. Для набору тестів загальною вартістю 40 балів жодне з чисел у вхідних даних не буде перевищувати 5 000.

3. Тести, у яких виконуються обидві умови 1 і 2, мають вартість 15 балів.

4. Тести, у яких виконується хоча б одна з умов 1 і 2, мають вартість 85 балів.

Приклад вхідних та вихідних даних

Приклад введення	Приклад виведення
4 5 3 100 2 80 100 20 125 130 10 75 150 5 120 110	50

Оптимальний розклад полягає у відвідуванні ярмарок з номерами 1 та 3 (місця розташування яких — 80 і 75 метрів від джерела річки вниз за течією). Послідовність подій і прибуток, який буде отримано в результаті цих подій, такі:

1. Комівояжер пропливає 20 метрів вгору за течією і витрачає на це 100 доларів. Прибуток до цього моменту є від'ємним і становить −100 доларів.

2. Він відвідує ярмарку з номером 1 і виручає 100 доларів. Прибуток до цього моменту становить 0 доларів.

3. Він пропливає 5 метрів вгору за 25 доларів. Прибуток до цього моменту є від'ємним і становить −25 доларів.

4. Він відвідує ярмарку з номером 3, де він виручає 150 доларів. Прибуток до цього моменту становить 125 доларів.

5. Він пропливає 25 метрів вниз, щоб повернутися додому, на що витрачає 75 доларів. Прибуток після закінчення подорожі складає 50 доларів.

Рекомендації щодо розв'язання

Почнемо з випадку, коли ніякі дві ярмарки не проходять у один день. Пізніше покажемо, як модифікувати алгоритм, щоб він обробляв ярмарки, що проходять у один день.

Перший поліноміальний розв'язок. Спочатку опишемо достатньо стандартний алгоритм динамічного програмування. Впорядкуємо ярмарки за днем, коли вони відбуваються. Для кожної ярмарки i підрахуємо прибуток P_i, що ми можемо отримати безпосередньо після відвідання ярмарки.

Щоб уникнути спеціальних випадків, додамо пусті ярмарки 0 та $N+1$, що обидві відбуваються вдома у комівояжера, ярмарка 0 є першою, а ярмарка $N+1$ є останньою ярмаркою. Ми можемо одразу сказати, що $P_0=0$ та P_{N+1} — це відповідь, що нам треба обчислити.

Значення від P_1 до P_{N+1} можна обчислити по черзі, використавши таке спостереження: ми маємо прибути з якої ярмарки, нам треба обрати з якої.

Нехай $cost(x, y)$ буде вартість подорожі з точки x до точки y по річці. Якщо $x \le y$, маємо $cost(x,y)=(y-x)D$, інакше $cost(x,y)=(x-y)U$.

Можемо записати:

$$\forall i \in \{1,\dots,N+1\} : P_x = \max_{0 \le j < i} (P_j - cost(L_j, L_i)) + M_i .$$

Щоб обчислити P_i, оберемо номер j ярмарки, що ми відвідали безпосередньо перед i. Безпосередньо після ярмарки j кращий прибуток, що ми могли отримати є P_j. Після цього нам треба рухатись до місця поточної ярмарки, що буде нам коштувати $cost(L_j,L_i)$ та, нарешті, ми відвідаємо ярмарку i з прибутком M_i. Щоб отримати найбільше можливе значення P_i, знайдемо максимум серед усіх можливих j.

Складність алгоритму буде $O(N^2)$, чого достатньо для розв'язання випадків, де всі вхідні значення не перевищують 5000.

Покращений розв'язок. Тепер покращимо попередній алгоритм. Зауважимо, що прибуток M_i відвідання ярмарки i є однаковим для усіх виборів j. Отже, оптимальний вибір j залежить від прибутків $P_0, \dots, P_{i-1}$, місць $L_0, \dots, L_{ji}$ та місця L_i поточної ярмарки.

Ми можемо розділити ярмарки 0 до $i-1$ на дві групи: ті, що знаходяться уверх за течією від L_i, та ті, що знаходяться вниз за течією. Тепер можна розділити нашу задачу «знайти оптимальне j» на дві частини: «знайти оптимальний вибір для попередньої ярмарки вище» та «знайти оптимальний вибір для попередньої ярмарки нижче».

Розглянемо пошук оптимальної попередньої ярмарки вище від L_i. Якщо ми змінимо значення L_i (таким чином, що це не вплине на те, які ярмарки знаходяться вище від ярмарки i), чи вплине це на наш вибір? Ні, не вплине. Якщо ми, наприклад, збільшимо L_i на Δ, це означає, що для кожної ярмарки вище вартість подорожі до ярмарки i збільшується на стільки ж, $D\Delta$. Отже, оптимальний вибір буде тим же.

Тепер опишемо достатньо просту структуру даних, що дозволить нам знайти оптимальну попередню ярмарку вище i за час $O(\log N)$.

Ця структура відома як дерево інтервалів. Ми можемо призначити ярмаркам нові мітки відповідно до їх унікальних позицій на річці. Більш точно, нехай l_f буде кількість ярмарок, що знаходяться вище ярмарки f (включаючи ті, що відбуваються після ярмарки f).

Наше дерево інтервалів буде повним бінарним деревом з k рівнями, де k є найменшим цілим, таким що $2^{k-1} \ge N+2$. Зауважте, що $k=O(\log N)$.

Листки у цьому бінарному дереві відповідають ярмаркам та порядок, у якому ярмарки призначені листкам, задається значеннями l_i. Тобто, найлівіший листок є ярмаркою, що розташована найближче до джерела річки, другий листок є наступним від джерела і так далі.

Кожен вузол у нашому дереві відповідає інтервалу ярмарок, звідси назва «дерево інтервалів». У кожному вузлі дерева інтервалів ми можемо зберігати відповідь на наступне питання: «Нехай S буде множиною ярмарок, що відповідають листкам у цьому піддереві та які вже оброблено. Нехай ми знаходимся нижче за течією від кожної з них, який буде оптимальний вибір?».

Маючи цю інформацію, ми можемо легко визначити оптимальний вибір для наступної ярмарки i за час $O(\log N)$. Також легко обновлювати інформацію у дереві після обробки ярмарки i, це можна зробити за час $O(\log N)$.

У нашому розв'язку буде, зрозуміло, два дерева інтервалів: одне для напрямку вгору та одне для напрямку вниз. Для кожної ярмарки i ми спочатку зробимо два запити, щоб визначити кращого попередника вверх та вниз, після цього оберемо кращий з двох варіантів, обчислимо P_i та нарешті обновимо обидва інтервальних дерева.

Отже, ми обробляєм кожну ярмарку за $O(\log N)$, що приводить до загальної складності $O(N \log N)$.

Інший однаково гарний розв'язок. У цьому розділі ми покажемо інший розв'язок з тією ж складністю,

який використовує структуру даних «впорядкована множина» та може бути просто реалізований на C++ з використанням класу **set**.

Як раніше, ми будемо обробляти ярмарки одну за одною, у порядку днів, у які вони проходять. Уявімо ситуацію після обробки деяких ярмарок. Нехай a і b будуть дві ярмарки, які ми вже обробили. Будемо казати, що *a покривається b*, якщо $P_a < P_b - cost(L_b, L_a)$.

Людською мовою, a покривається b, якщо стратегія «відвідати ярмарку b останньою та потім рухатись до місця ярмарки a» є не гіршою, ніж стратегія «відвідати ярмарку a останньою».

Як тільки ярмарка a покривається якоюсь іншою ярмаркою b, то ця ярмарка ніколи не буде оптимальним попередником ніякої пізнішої ярмарки. Ярмарка b буде завжди (незалежно від розташування пізнішої ярмарки) не гірше вибору a.

З іншого боку, якщо ярмарка не покрита іншою ярмаркою, існують місця на річці для яких b буде оптимальним попередником, принаймні місце L_b та його безпосередні околиці. Будемо називати такі ярмарки активними.

У нашому розв'язку ми будемо підтримувати множину поточних активних ярмарок, що впорядковано за їх позицією на річці. Будемо використовувати структуру «впорядкована множина», що найчастіше реалізується як збалансоване бінарне дерево.

Можна просто показати, що для кожної активної ярмарки f існує інтервал на річці, де f є оптимальним вибором. Ці інтервали, очевидно, не перетинаються (за винятком, можливо, кінців) та разом покривають всю річку. Та оскільки інтервал для f містить f, інтервали розташовані у тому ж порядку, що і відповідні ярмарки.

Отже, коли ми обробляємо нову ярмарку i нам достатньо знайти найближчі активні ярмарки вище та нижче за течією від i, один з двох буде оптимальним вибором.

Після обробки ярмарки i та обчислення P_i нам потрібно обновити множину активних ярмарок. Очевидно, i є активним, оскільки ми обчислили P_i, обравши кращий спосіб дістатися L_i, та потім додали позитивний прибуток M_i. Ми додамо її до множини активних ярмарок. Ми ще не завершили, i може покривати деякі попередньо активні ярмарки. Але їх легко знайти: якщо ніякий з безпосередніх сусідів i (у множині активних ярмарок) не покривається i, ми очевидно завершили обробку. Якщо є ярмарки покриті i, заберемо їх з множини та повторимо перевірку знову.

У цьому розв'язку кожна ярмарка вставляється у множину активних ярмарок один раз та вилучається не більше одного разу. Додатково, коли обробляється кожна ярмарка, ми робимо один запит, щоб знайти дві найближчі активні ярмарки. Кожна з цих операцій потребує час $O(\log N)$, отже, загальна складність буде $O(N \log N)$.

Кілька ярмарок на день. По-перше, ми не можемо обробляти ярмарки, що проходять у один день у якомусь порядку, оскільки маємо дати комівояжеру можливість відвідати їх і в іншому порядку.

Може бути багато способів відвідати ярмарки у заданий день. Однак нам не треба розглядати їх всі, достатньо розглянути підмножину, що містить оптимальний розв'язок.

Припустимо, що ми вже обрали деякий порядок, у якому ми відвідаємо ярмарки у заданий день. Нехай u та d будуть найвіддаленішими ярмарками вверх та вниз за течією, що ми відвідаємо. Тоді ми, очевидно, також відвідаємо всі ярмарки між u та v, оскільки ми все одно мандруємо через їх місця. Та очевидно, щоб відвідати всі ці ярмарки, достатньо рухатись до u, потім з u до v, або навпаки. Розглядатимемо тільки такі шляхи.

Будемо обробляти кожен день у два етапи. На першому етапі обробимо кожну ярмарку i окремо, начебто у цей день проходить тільки одна ярмарка, та визначимо попередні значення P_i, кращий прибуток, що ми можемо отримати, прибувши на ярмарку i з якоїсь ярмарки у попередній день.

На другому етапі розглянемо рух вверх або вниз за течією, окремо кожен напрямок. Під час обробки напрямку ми будемо розглядати ярмарки по черзі та з'ясовувати для кожного, чи є найбільш прибутковим починати саме на цій ярмарці (тобто використавши значення обчислене на попередньому кроці) або починати раніше (тобто використавши оптимальне значення, обчислене для попередньої ярмарки на цьому кроці, мінус вартість подорожі з тої ярмарки на цей, плюс прибуток на цій ярмарці).

Для кожної ярмарки i фактичне значення P_i буде дорівнювати більшому із значень для подорожей вверх та вниз.

Нарешті, нам треба обновити множину активних ярмарок. Якщо використовується дерево інтервалів, це досягається просто додаванням кожної ярмарки до дерев для напрямку вниз та вгору. Коли використовується впорядкована множина, просто треба бути більш уважними, оскільки не всі ярмарки, що ми тільки-но обробили, стануть активними. Перед вставкою нової активної ярмарки ми маємо перевірити, що ярмарка насправді є активною, перевіривши його потенційних сусідів у структурі даних. Якщо якась із сусідніх ярмарок покриває ту, що додається, тоді вона не є активною і її не треба додавати до множини активних. З цією модифікацією ми можемо оновлювати всю структуру даних по черзі, намагаючись додати кожну ярмарку (у довільному порядку).

Зрозуміло, що додатковий час на обробку другого етапу у довільний день є лінійним відносно кількості ярмарок у цей день, маючи на увазі, що ми вже відсортували ярмарки за їх місцем (що легко досягається додаванням цієї умови, як додаткової при сортуванні ярмарок на початку). Кроки з оновлення дерева інтервалів та впорядкованої множини в обох випадках займуть час $O(\log N)$. Отже, цей додатковий крок не змінює загальну складність алгоритму, вона становить $O(N \log N)$.