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

Гуржій А.М., Бондаренко В.В.

ЗАВДАННЯ ПЕРШОГО ТУРУ

1. ЛУЧНИКИ

Умова. Змагання лучників проходить за такими правилами. Є N мішеней, розташованих у лінію і занумерованих від 1 до N включно відповідно до їх місця на лінії (сама ліва мішень має номер 1, а сама права мішень має номер N). Також є $2 \cdot N$ лучників. У довільний момент під час турніру у кожну з мішеней стріляють два лучники. Кожен тур змагань проходить за такою процедурою. Два лучники, що стріляють в кожну мішень, змагаються один з одним та визначають переможця та програвшого. Після цього всі лучники пересуваються так:

- переможці на мішенях з номерами від 2 до N включно пересуваються на одну мішень лівіше (тобто, на мішені від 1 до $N-1$ відповідно);
- програвші на мішенях з номерами від 2 до N включно, також як переможець мішені з номером 1, залишаються на тих самих мішенях;
- програвший на мішені з номером 1 пересувається на мішень з номером N.

Турнір складається з R турів, кількість турів не менше, ніж кількість лучників (тобто $R \geq 2N$).

Ви — єдиний лучник, котрий прибув на турнір вчасно. Решта $2N-1$ лучників прибули раніше і вже розташувались у лінію. Вам тепер треба «вставитись» у лінію десь серед них. Ви знаєте, що після того як ви займете свою позицію, два самих лівих лучники почнуть турнір на мішені з номером 1, два наступних на мішені з номером 2, і так далі. Два самих правих лучники почнуть турнір на мішені з номером N.

Усі $2N$ лучників на турнірі (включаючи вас) мають ранг, встановлений відповідно до їх навичок, при цьому менший ранг відповідає більш високим навичкам. Немає двох лучників з одинаковим рангом. Також, коли б не змагались два лучники, завджи переможе лучник з меншим рангом.

Знаючи якими сильними у стрільбі є ваші суперники, ви хочете розташуватись так, щоб закінчити турнір на мішені з найменшим можливим номером. Якщо є декілька способів це зробити, оберіть той, який починається на мішені з найбільшим номером.

Завдання. Напишіть програму, яка за заданим рангом усіх лучників, включаючи вас, а також за розташуванням ваших суперників на лінії, визначає, на якій мішені ви маєте почати турнір, щоб досягти описаної вище мети.

Обмеження. $1 \leq N \leq 200000$ — кількість мішеней (також дорівнює половині кількості лучників), $2N \leq R \leq 1000000000$ — кількість турів турніру, $1 \leq S_k \leq 2N$ — ранг лучника з номером k.

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

- Перший рядок містить цілі числа N і R, розділені пропуском.
- Наступні $2N$ рядків містять ранги лучників. Перший із цих рядків містить ваш ранг. Решта рядків містять ранги решти лучників по одному в рядку в тому порядку, у якому вони розташувались у лінії (зліва направо). Кожен із цих $2N$ рядків містить одне ціле число з діапазону від 1 до $2N$ включно. Ранг 1 — найкращий, ранг $2N$ — найгірший. Ніякі два лучники не мають одинакових рангів.

Вихідні дані. Ваша програма повинна записати у стандартний потік виведення один рядок, у якому буде міститись одне ціле число між 1 і N включно, що є номером мішені, на якій ви почнете турнір.

Оцінювання. Для набору тестів загальною вартістю 60 балів N не буде перевищувати 5000. Також, для деяких із цих тестів, сумарною вартістю 20 балів, N не буде перевищувати 200.

Приклад вхідних та вихідних даних

Приклад введення	Приклад виведення
4 8 7 4 2 6 5 8 1 3	3

Ви другий найгірший лучник. Якщо ви починаєте на мішені з номером 1, то ви потрапляєте на мішень з номером 4 і залишаєтесь там до кінця. Якщо ви починаєте на мішені з номером 2 або 4, ви просто залишаєтесь там до кінця турніру. Якщо ви почнете на мішені з номером 3, то переможете найгіршого лучника, пересунетесь на мішень з номером 2 і там залишитесь.

Приклад введення	Приклад виведення
4 9 2 1 5 8 3 4 7 6	2

Ви другий найкращий лучник. Найкращий лучник уже розташувався на мішені з номером 1, і залишиться там до закінчення турніру. Таким чином, незалежно від того, де ви почнете турнір, ви будете завжди пересуватись зі своєї мішені, проходячи через усі мішені з номерами від 4 до 1 знову і знову. Для того щоб закінчити турнір на мішені з номером 1 після дев'яти пересувань, ви маєте почати на мішені з номером 2.

Рекомендації щодо розв'язання

Доведення далі є дуже докладним та довгим, але ідеї не є такими складними. Кроки можна описати так.

1. Тривіальний розв'язок (пробуємо всі можливості та емулюємо турнір для кожної) дає алгоритм складності $O(N^2R)$.

2. Можна помітити, що не більше ніж через $2N$ раундів турнір стає циклічним, позиції повторюються кожні N раундів. Це дозволяє прискорити тривіальний алгоритм до швидкості $O(N^3)$.

3. Можна зоптимізувати емуляцію турніру (коли ми вже обрали початкову позицію) з $O(N^2)$ до $O(N)$. Це найбільш складна частина розв'язку. Основною ідеєю є те, що нас цікавить тільки наша позиція в кінці, і не цікавить позиція всіх інших.

4. Коли в нас уже є процедура, що знаходить кінцеву позицію за початковою, можна використати модифікований бінарний пошук, покращивши складність алгориту з $O(N^2)$ до $O(N\log N)$, або покращавши алгоритм складності $O(N^3)$ до $O(N^2\log N)$.

5. Останні два алгоритми мають також повільніші версії ($O(N^2\log N)$ та $O(N\log N\log N)$), якщо слідкувати за позиціями інших лучників, а не тільки своєю.

Оптимізація до $O(N^3)$. Тривіальний розв'язок перебирає всі можливі початкові позиції та емулює всі раунди турніру, даючи складність $O(N^2R)$. Покажемо, як можна зменшити оцінку до $O(N^3)$. Розглянемо лучників із номерами від $N+2$ до $2N$. Назвемо їх слабкими лучниками.

Твердження (Теорема 1). *Після достатньої кількості раундів* (*не більше* $2N$) *слабкі лучники будуть займати мішені з номерами від* 2 *до* N, *кожен лучник свою мішень, та будуть знаходитись там до кінця змангань.*

Доведення. Після $N-1$ раундів, лучник із номером 1 буде займати мішень 1 та залишиться там до кінця турніру. З цього моменту, якщо ми розглянемо множину N лучників, що складається з лучника 1 та $N-1$-го слабого лучника (назвемо його множиною *слабкі+1*), якщо уявити мішені розставленими по колу (від 1 до N та знову 1), отримаємо такий сценарій.

- Коли лучник із множини *слабкі+1* змагається з лучником, що не належить множині, лучник з множини залишиться на мішені, а інший лучник пересунеться.
- Коли змагаються лучники з множини *слабкі+1*, один із них залишиться, а інший пересунеться на мішень лівіше.

Твердження (Лема 1). *Протягом N раундів після того як лучник номер 1 прибув на мішень номер 1, на кожній з мішеней буде принаймні один лучник із множини слабкі+1.*

Доведення. Припустимо, що це не правильно. Ми знаємо, що як тільки мішень зайнято лучником із множини *слабкі+1*, її завжди буде займати принаймні один такий лучник (оскільки лучник із *слабкі+1* ніколи не залишає свою мішень, крім випадку, коли є інший лучник із *слабкі+1*, що його замінить). Тому, якщо Лема не справджується, має існувати мішень, яку ніколи не займав лучник із множини *слабкі+1* протягом N раундів). Назвемо цю мішень A. Якщо A не зайнята лучником із *слабкі+1*, тоді на мішені зліва від A (назвемо її B) має знаходитись не більше одного такого лучника протягом одного раунду, та так буде завжди. Тоді протягом двох раундів на мішені зліва від B буде не більше одного лучника із *слабкі+1*, протягом трьох раундів на наступній мішені зліва також буде не більше одного лучника із *слабкі+1*. Продовжуючи по колу, протягом $N-1$ раундів на мішені праворуч від A буде не більше одного лучника із *слабкі+1*. Таким чином, протягом $N-1$ раундів на всіх мішенях, крім A, буде не більше одного лучника із *слабкі+1*. Але оскільки є N таких лучників та N мішеней, це означає, що на А повинен бути принаймні один лучник із *слабкі+1*. Це протирічить нашому припущенню, що A залишається вільною від лучників із *слабкі+1* протягом N раундів, що доводить Лему 1.

Тепер ми знаємо, що на кожній із мішеней після щонайбільше $2N$ раундів знаходиться лучник із множини *слабкі*+1, та оскільки ми знаємо, що як тільки на мішень потрапляє такий лучник, там завжди буде знаходитись принаймні один такий лучник, ми довели Теорему 1.

Тепер розглянемо всіх лучників, що не належать множині *слабкі*+1. Якщо на кожній із мішеней розташований один лучник із *слабкі*+1, це також означає, що на кожній мішені розташований один лучник не із *слабкі*+1. Оскільки за цим сценарієм лучники із *слабкі*+1 завжди залишаються на своєму місці, це означає, що лучники з номерами від 2 до $N+1$ будуть циклічно пересуватись по N мішенях, періодично повторюючи свої позиції після кожних N раундів.

Це означає, що якщо ми замінимо R на $R'=2N+(R \bmod N)$, ми отримаємо ідентичну відповідь, оскільки результат після R раундів буде ідентичним результату після R' раундів (згадаємо, що $R \geq 2N$).

Зі сказаного випливає, що ми можемо покращити алгоритм складності $O(N^2R)$ до $O(N^3)$.

Оптимізація до $O(N^2)$. Коли ми обрали початкову позицію та емулюємо, що трапиться після R' раундів, ми виконуємо $O(N^2)$ обчислень для кожної початкової позиції. Можна зменшити складність цієї частини алгоритму до $O(N)$ так.

Є три типи лучників: ті, що кращі нас, назвемо їх *чорними* лучниками; ті, що гірші нас, назвемо їх *білими* лучниками; та ми (один лучник), назвемось *сірим* лучником. Щоб розв'язати нашу задачу, нам не потрібно розрізняти лучників одного кольору, оскільки це не впливає на фінальний результат. Якщо змагаються лучники одного кольору, не має значення, який сильніший (тобто, це не впливає на фінальну позицію сірого лучника). Також ми знаємо, що якщо змагаються лучники різних кольорів, перемагає темніший.

Існує три випадки, які розглянемо окремо.

Випадок 1. Немає чорних лучників. Це означає, що ми є найкращим лучником, очевидно що найкращою мішенню для старту буде мішень з номером N.

Випадок 2. Є принаймні один чорний лучник, але не більше ніж N, це означає, що наш ранг знаходиться між 2 та $N+1$, що означає, що ми належимо до групи лучників, що циклічно рухаються по мішенях та зупиняються після заданої кількості раундів. У цьому випадку ми маємо стежити не за всіма мішенями протягом турніру, а тільки за мішенню з номером 1. Якщо ми знаємо, хто змагається на мішені 1 на кожному раунді, тоді з'ясувавши, коли між раундами $2N$ та $3N$ сірий лучник буде змагатись проти лучника номер 1, ми зможемо підрахувати позицію сірого лучника після завершення турніру (що нас, власне, і цікавить). Будемо слідкувати за мішенню 1, використовуючи такий алгоритм.

Присвоємо кожному лучнику i номер P_i, котрий, кажучи неформально, показує найближчий можливий раунд, коли лучник i міг би потенційно змагатись на мішені 1. Спочатку, номер P кожного лучника дорівнює номеру його початкової мішені. Після цього емулюємо кожен раунд турніру за такою процедурою:

1. Визначимо, хто змагається на мішені 1. Першим лучником, зрозуміло, буде переможець на мішені 1 з попереднього раунду (або самий лівий лучник на початку). Визначимо його опонента так. Візьмемо всіх лучників із номером P, меншим або рівним номеру поточного раунду. Кращий із них буде змагатись на мішені 1 (доведення чому це так — нижче).

2. Порівняємо цих двох лучників та присвоїмо гіршому значення P, що дорівнює номеру поточного (тільки що завершеного) раунду плюс N. Це найближчий раунд, коли ми зможемо побачити цього лучника знову на мішені 1.

Тепер доведемо, що алгоритм є коректним. Позначимо A_j лучника, що змагається на мішені 1 у раунді j, але змагався на мішені 2 у раунді $j-1$. Кожен лучник i має значення P_i, таке, що якщо він виграє кожне змагання з того моменту, як він отримав це значення P_i, він закінчить турнір, будучи A_{P_i}. Тепер поглянемо на лучника, який за нашим алгоритмом став A_j (для деякого раунду j). Позначимо його W. Нехай $S=j-P_W$. Якщо S є нулем, це означає, що W не має можливості стати A_{j-1} навіть якщо б він виграв усі свої змагання. Отже, у цьому циклі W ніколи не зустрінеться з A_{j-1} (або з довільним з попередніх A). Оскільки W ніколи не зустрічав цих лучників і оскільки він кращий ніж будь-хто з тих, хто змагається за позначку A_j, це означає, що він ніколи не програє в цьому циклі (принаймні поки не досягне мішені 1), що означає що він дійсно є A_j (тобто, наш алгоритм є коректним у цьому випадку).

Якщо S є більшим за нуль, це означає, що W мав можливість стати A_{j-1}, але втратив її. А це означає, що він змагався безпосередньо з A_{j-1}, оскільки останній був єдиним кандидатом на A_{j-1}, який був кращим W (це правильно, оскільки за нашим алгоритмом кожен кандидат на A_{j-1} автоматично стає кандидатом на A_j, за винятком власне A_{j-1} — тобто, якщо W був кандидатом на A_{j-1}, але програв і тепер є кращим серед кандидатів на A_j, він мав бути другим серед кандидатів на A_{j-1}). Тепер, якщо W змагається з A_{j-1} та якщо він кращий ніж інші кандидати на A_j, то після зустрічі W буде «по п'ятах» переслідувати A_{j-1}: або на тій самій мішені, або безпосередньо праворуч. Це означає, що коли A_{j-1} досягає мішені 1 (у раунді $j-1$), W знаходиться на мішені 2. Оскільки за визначенням він кращий іншого лучника на мішені 2, він дійсно потрапить на мішень 1 у раунді j.

Довівши правильність алгоритму, що спостерігає за мішенню 1, ми можемо проаналізувати його складність. Оскільки ми не розрізняємо лучників одного кольору, ми можемо описувати довільну множину лучників тільки трьома числами: кількість білих, сірих та чорних лучників у множині. Це дозволить нам виконувати кожен крок алгоритму (тобто, кожен раунд емуляції) за константний час, оскільки все, що нам треба, це визначити колір кращого лучника з множини кандидатів та додати до цієї множини тільки одного або двох нових лучників (тих, номери P яких дорівнюють номеру наступного раунду). Оскільки нам потрібно емулювати не більше $3N$ раундів, то нам не цікаві значення P більші $3N$, ми зможемо реалізувати наш алгоритм емуляції за час та $O(N)$.

Випадок 3. Є більше ніж N чорних лучників. Це означає, що наш номер більше $N+1$, отже, ми є одним з лучників, що в якийсь момент зупиняються назавжди на одній із мішеней. Залишається з'ясувати, що це за мішень.

Ми вже показали, що як тільки лучник 1 досягає мішені 1, усе, що роблять лучники з множини *слабкі*$+1$, це штовхають один одного по мішенях доки кожен не займе свою мішень. Оскільки наш номер більше $N+1$, усі білі та сірий лучники належать до множини слабких лучників. Нам треба тільки земулювати, як білі та сірий лучники будуть штовхати один одного. Почнемо з мішені 1, де, як ми знаємо, ніякий із білих або сірий лучники не зупиняться. Потім спробуємо підрахувати скільки білих/сірих лучників закінчують турнір, будучи «проштовханими» по колу після кожної мішені. На початку білі/сірі лучники, проштовхані з 1 до N, будуть ті, що на початку були на мішені 1 (зауважте, що ми підраховуємо оцінку знизу; пізніше ми можемо з'ясувати, що існують ще білі/сірі лучники, яких проштовхали через мішень 1). Потім ми переходимо до мішені N. Додамо всіх білих/сірих лучників, які тут стартують до тих, що пересунулись з мішені 1, та залишимо там одного з нашої множини (ми завжди залишаємо білого, якщо він є; якщо немає, ми залишаємо сірого; якщо множина порожня, то ми, зрозуміло, нікого не залишаємо та віддіємо цю мішень чорним лучникам). Продовжуємо рухатись від N до $N-1$, до $N-2$, і так далі. На кожній мішені ми «підбираємо» всіх білих/сірих лучників та залишаємо одного, підібраного раніше або тільки що. У якийсь момент ми потрапляємо до мішені 1 і, якщо в нас є білі/сірі лучники, ми просто переносимо їх до мішені N та продовжуємо процедуру. Другий раз, коли ми потрапимо до мішені 1, ми напевне не будемо мати білих/сірих лучників для розпихування, оскільки за Теоремою 1 ми знаємо, що після $2N$ раундів усі білі/сірі лучники будуть займати свої мішені. Цей алгоритм очевидно потребує лінійного часу та пам'яті з тієї ж

причини, що і у Випадку 2. Він є коректним, оскільки ми рухаємо білих/сірих лучників тільки коли потрібно (тобто, коли вони зустрічаються на тій самій мішені з іншим білим/сірим лучником, або на мішені 1), та ми також пересвічуємось, що в кінці кожен білий/сірий лучник зупиниться в позиції, де він буде знаходитись до кінця турніру. Оптимізація емуляції турніру з $O(N^2)$ до $O(N)$, що описана вище, покращує розв'язок для всієї задачі з $O(N^3)$ до $O(N^2)$.

Оптимізація до $O(N\log N)$. Остання оптимізація, що буде використана для зменшення складності алгоритму до $O(N\log N)$, базується на добре відомій техніці бінарного пошуку. Ефективний алгоритм емуляції турніру, що описано вище, можна просто модифікувати для того щоб він підраховував, скільки разів сірий лучник пересувається з мішені 1 до мішені N (позначимо T). Комбінуючи цю інформацію з фінальною позицією сірого лучника (позначену X), ми зможемо побачити фінальні позиції в лінійному (а не круговому) представленні. Якщо ми подамо результат емуляції як число $X-N\cdot T$, то результат кожного переміщення сірого лучника з однієї мішені на іншу можна розглядати як зменшення результату на 1. Тоді якщо ми поглянемо на алгоритм симуляції вище, можна помітити, що якщо початкова позиція знаходиться вище, то фінальній результат буде меншим. Наприклад, якщо ви обираєте старт із більшого значення P, це не приведе вас далі вперед (у лінійному представленні, не круговому), ніж якби ви обрали менше початкове значення P.

Маючи монотонність відношення між початковою позицією та фінальним результатом турніру, ми можемо шукати оптимальну початкову позицію так.

1. Визначити результат при старті з мішені 1.
2. Визначити результат при старті з мішені N.
3. Для кожного числа з цього діапазону, що ділиться на N, використати стандартний бінарний пошук, щоб знайти найменшу можливу кінцеву позицію, що точно більша ніж це число (і відповідно ближча до мішені 1 при певній кількості обертань).
4. Зі знайдених початкових позицій обрати найкращу.

Оскільки розглядається тільки $O(N)$ раундів, діапазон пошуку є $O(N)$ і відповідно буде потрібно тільки $O(1)$ бінарних пошуків. Кожен такий пошук потребує аналізу $O(\log N)$ стартових позицій, даючи загальну складність $O(N\log N)$.

Додаткові зауваження. Нарешті, потрібно зауважити, що ефективний алгоритм емуляції, що описаний вище (який ігнорує різницю між однокольоровими лучниками та працює за $O(N)$ на симуляцію) може бути модифікований так, щоб розрізняти різних білих та чорних лучників, використовуючи бінарні кучі або інші схожі структури. Це дасть фінальний алгоритм складності $O(N^2\log N)$ або $O(N\log N\log N)$, залежно від того, чи використовується двійковий пошук. Можна також отримати оцінку $O(N^2\log N)$ або $O(RN\log N)$, використовуючи бінарний пошук без оптимізації емуляції.

Задачу також можна розв'язати за лінійний час, але розв'язок є достатньо складним, залишимо цю вправу читачеві. Розв'язку $O(N\log N)$ достатньо, щоб отримати повний бал.

2. ПРИЙОМ НА РОБОТУ

Умова. Вам необхідно найняти робітників для будівельного проекту. Заяви про прийом на роботу подало N кандидатів, занумерованих від 1 до N включно. Кожен кандидат з номером k вимагає, щоб у випадку прийому його на роботу йому платили не менше ніж S_k доларів. Також для кожного кандидата з номером k відомо його рівень кваліфікації Q_k. Положення про будівельну діяльність вимагає, щоб ви платили робітникам пропорційно до їх рівня кваліфікації один відносно іншого. Наприклад, якщо ви наймаєте двох робітників A і B, таких що $Q_A=3\cdot Q_B$, то ви забов'язані платити робітнику A рівно в три рази більше, ніж ви платите робітнику B. Вам дозволяється платити робітникам не цілу кількість грошей. Більш того, дозволяється навіть платити кількість грошей, яку не можна записати за допомогою скінченого числа десяткових цифр, таку як третину або шосту частину долара.

У вашому розпорядженні є W доларів, і ви хочете найняти якомога більше робітників. Ви вирішуєте, кого найняти і скільки їм платити, але ви маєте задовільнити як вимоги робітників про мінімальну зарплатню, так і вимоги положення про будівельну діяльність. Природньо, що вам потрібно залишитись у рамках бюджету, що дорівнює W доларам.

Для цього будівельного проекту рівень кваліфікації робітників не має значення. Ви зацікавлені тільки в тому, щоб найняти якомога більше робітників незалежно від їх рівня кваліфікації. Однак, якщо є декілька способів досягнути мети, то ви хочете обрати такий, щоб загальна сума грошей, яку ви сплатите робітникам, була якомога меншою. Якщо і це можна досягнути кількома способами, то немає ніякої різниці між цими способами, і вас задовольнить довільний з них.

Завдання. Напишіть програму, яка за заданими вимогами до зарплатні та рівнів кваліфікації кандидатів, а також кількості грошей, що є у вас, визначає, яких кандидатів вам треба найняти. Ви маєте найняти якомога більше з них і при цьому витратити якомога менше грошей, виконуючи вимоги положення про будівельну діяльність, що описано вище.

Обмеження. $1\le N\le 500000$ — число кандидатів, $1\le S_k\le 20000$ — мінімальні вимоги до зарплатні кандидата з номером k, $1\le Q_k\le 20000$ — рівень кваліфікації кандидата з номером k, $1\le W\le 10000000000$ — сума грошей, доступна вам.

Вхідні дані. Ваша програма повинна читати із стандартного потоку введення такі дані.

- Перший рядок вхідного файлу містить два цілих числа N і W, розділених пропуском.
- Наступні N рядків описують кандидатів, по одному кандидату на кожен рядок, k-й з них описує кандидата з номером k і містить два цілих числа S_k і Q_k, розділених пропуском.

Вихідні дані. Ваша програма повинна вивести у стандартний потік виведення такі дані.

- Перший рядок має містити одне ціле число H — кількість робітників, яких ви приймаєте на роботу.
- Наступні H рядків мають містити список номерів кандидатів у довільному порядку, яких ви об-

рали для найму на роботу (різні цілі числа від 1 до N), по одному у кожному рядку.

Оцінювання. Для кожного з тестів, що використовується для перевірки розв'язку цієї задачі, ви отримаєте повний бал, якщо ваш вибір кандидатів дозволяє досягнути вашої мети, задовольняючи всі задані обмеження. Якщо ви виведете коректно перший рядок (тобто, коректне значення H), але при цьому решта файлу не буде відповідати вищеописаним умовам, то ви отримаєте 50% балів за цей тест. Це правило також діє навіть у тому випадку, коли решта файлу відформатована неправильно, але перший рядок виведено правильно.

Для набору тестів загальною вартістю 50 балів значення N не буде перевищувати 5000.

Приклад вхідних та вихідних даних

Приклад введення	Приклад виведення
4 100 5 1000 10 100 8 10 20 1	2 2 3

Єдина комбінація, при якій ви можете дозволити собі найняти двох працівників і задовольнити всі вимоги — це обрати працівників із номерами 2 і 3. Ви можете заплатити їм 80 і 8 доларів, відповідно, таким чином, не вийшовши за бюджет 100 доларів.

Приклад введення	Приклад виведення
3 4 1 2 1 3 1 3	3 1 2 3

У цьому прикладі ви можете дозволити собі найняти всіх трьох робітників. Ви платите 1 долар робітнику з номером 1 і по 1,50 долара робітникам з номерами 2 і 3, і таким чином, не виходите за рамки бюджету, що дорівнює 4 доларам.

Приклад введення	Приклад виведення
3 40 10 1 10 2 10 3	2 2 3

У цьому прикладі ви не можете дозволити собі найняти всіх трьох працівників, так як це вартувало б вам 60 доларів, але ви можете дозволити собі найняти довільних двох з них. Ви обираєте працівників із номерами 2 і 3, тому що в цьому випадку виходить найменша сума грошей порівняно з іншими комбінаціями двох робітників. Ви можете заплатити 10 доларів робітнику з номером 2 і 15 доларів робітнику з номером 3, загальна сума буде дорівнювати 25 доларам. Якби ви найняли робітників з номерами 1 і 2, то вам добилось би заплатити їм 10 і 20 доларів відповідно. Якби ж би ви обрали робітників з номерами 1 і 3, то вам довелось би заплатити їм 10 і 30 доларів відповідно.

Рекомендації щодо розв'язання

Кожен працівник k описується двома числами: його зарплатнею S_k та його кваліфікацією Q_k.

Уявимо, що ми вже обрали множину K працівників, що буде прийнято на роботу. Як ми підрахуємо загальну кількість грошей, що ми маємо їм заплатити?

За умовою задачі, розміри зарплат мають бути пропорційними рівням кваліфікації. Отже, має бути деяка одинична зарплатня u, така що кожному працівнику $k \in K$ буде сплачено $u \cdot Q_k$ доларів. Проте, зарплатня кожного з працівників має бути не менше його мінімальної зарплатні. Таким чином, u має бути достатньо великим, щоб гарантувати для кожного $k \in K$ виконується $u \cdot Q_k \geq S_k$.

Для кращого розуміння, ми можемо переписати останню умову так. Для кожного $k \in K$ має виконуватись $u \geq S_k/Q_k$. Позначимо S_k/Q_k як U_k — мінімальний розмір одиничної зарплатні, з якою можна прийняти на роботу працівника k. Ми також хочемо заплатити якомога менше, отже ми хочемо обрати найменше u, що задовільняє всі умови. Отже отримаємо:

$$u = \max_{k \in K} U_k .$$

Зауважте, що одинична зарплатня визначається одним працівником з K — тим, що має максимальне значення U_k.

Тільки що ми показали, що для довільної множини працівників K (а, отже, і для оптимальної множини) одинична зарплатня u дорівнює значенню U_k одного з працівників з K. Це означає, що є всього тільки $O(N)$ можливих значень u.

Тепер уявімо, що ми починаємо конструювати оптимальну множину працівників K, вибираючи одиничну зарплатню u. Як тільки ми обираємо u, ми знаємо, що ми можемо прийняти на роботу тільки тих працівників, k для яких $U_k \leq u$. Але як визначити кого з них приймати?

Це просто: якщо ми приймаємо на роботу працівника з кваліфікацією Q_k, ми маємо платити йому $u \cdot Q_k$ доларів. Для того щоб максимізувати кількість працівників, що ми можемо найняти (там мінімізувати вартість), ми, очевидно, маємо наймати найменш кваліфікованих працівників.

Отже, ми можемо обчислити кращий розв'язок для заданої базової зарплатні u, знайшовши всіх працівників, що ми можемо наймати, впорядкувавши їх за кваліфікацією, та жадібно вибираючи одного за одним (починаючи з найменш кваліфікованого), поки ми можемо їм платити.

Цей підхід дасть розв'язок складності $O(N^2 \log N)$. Розв'язок можна просто покращити до $O(N^2)$, оскільки ми можемо відсортувати працівників за кваліфікацією один раз на початку, та потім кожну одиничну зарплатню u можна перевірити за час $O(N)$.

Нарешті, покажемо, як покращити наведений алгоритм до $O(N \log N)$. Почнемо з впорядкування всіх працівників відповідно до значення U_k у порядку зростання, та позначимо працівників у цьому порядку $k_1, k_2, \ldots, k_N$. Для того щоб знайти оптимальну множину працівників, будемо робити те ж саме, що і в попередньому алгоритмі, але більш ефективно.

Нехай $Z(m)$ буде таке питання: Якою є оптимальна підмножина $k_1, \ldots, k_m$ для заданої одиничної зарплатні $U_{k_m} = S_{k_m}/Q_{k_m}$? З написаного вище випливає, що оптимальний розв'язок буде відповіддю на запитання $Z(m)$ для деякого m. Отже, все, що потрібно зробити — це знайти відповідь на ці N запитань.

Неефективною частиною попереднього розв'язку є те, що для кожного m ми повторюємо всі обчислен-

ня спочатку. Можна помітити, що нам не потрібно це робити — ми можемо обчислювати відповідь на $Z(m+1)$, використовуючи відповідь на $Z(m)$ для кожного m. Припустимо, що ми вже знаємо оптимальну відповідь на $Z(m)$ для деякого m. Будемо зберігати обраних працівників у черзі з пріорітетами Q, впорядкованими за кваліфікацією, більш кваліфіковані працівники будуть мати вищий пріоритет.

Нам потрібно додати працівника k_{m+1}. Його рівень кваліфікації може зробити його кращим кандидатом, ніж деякі вже оброблені працівники. Додамо його в чергу Q. Тепер Q містить всіх працівників, яких ми маємо розглянути, коли шукаємо поточний оптимальний розв'язок, оскільки якщо працівник має кваліфікацію занадто велику, щоб бути в оптимальному розв'язку для m, ми ніколи не захочемо використовувати його знову. Це виконується тому, що одинична зарплатня не зменшується, а набір працівників тільки росте, таким чином вартість прийому на роботу працівника разом з усіма доступними менш кваліфікованими працівниками тільки росте.

Однак, Q все ще може відрізнятись від оптимальної відповіді на $Z(k+1)$, оскільки вартість оплати всіх прцівників з Q може перевищіти бюджет W. Для цього є дві підстави. По-перше, при додаванні працівника k_{m+1} могла збільшитись одинична зарплатня u. По-друге, навіть якщо одинична зарплатня залишилась такою ж, ми додали ще одного працівника, а це могло зробити загальну зарплатню обраних працівників більше ніж W.

Отже, тепер нам потрібно скорегувати множину обраних працівників, послідовно викидаючи найбільш кваліфікованих, до тих пір, поки ми не зможемо всім їм платити. І це буде оптимальною відповіддю на запитання.

Підсумовуючи, повний розв'язок має такий вигляд. По-перше, впорядкуємо працівників відповідно до одиничної зарплатні, що вони спричинюють. Потім, будемо обробляти працівників у цьому порядку. Будемо зберігати оптимальний набір працівників у черзі з пріорітетами Q, та у додатковій змінній T будемо зберігати суму кваліфікацій усіх працівників з Q. Для кожного працівника k, ми, по-перше, додаємо його до Q (та відповідно змінюємо T), а потім викидаємо найбільші елементи з Q поки не можемо всім їм платити, тобто, поки $T \cdot S_{k_m}/Q_{k_m}$ перевищує кількість грошей, що ми маємо. Після завершення, ми знаємо числові параметри оптимального розв'язку — оптимальну кількість працівників, мінімальну вартість найму такої кількості працівників та номер f працівника, для якого було знайдено ці значення. Щоб побудувати розв'язок, простіше за все почати процес із початку, та зупинитись після обробки f працівників.

Перший крок можна виконати за час $O(N\log N)$.

На другому кроці (пошук оптимальної кількості працівників та вартості найму), ми для кожного працівника вставляємо його кваліфікацію в Q один раз, та видаляємо з Q не більше разу. Отже, є не більше $2N$ операцій з чергою з пріорітетами, та кожна така операція може бути виконаною за $O(\log N)$ (наприклад, якщо чергу з пріорітетами реалізовано як бінарне дерево).

Третій крок (побудова оптимального набору робітників) займе не більше ніж другий крок.

Отже, загальна складність розв'язку є $O(N\log N)$.

Інший розв'язок. Замість перегляду m вгору, також можливо переглядати вниз. Припустимо, що P є оптимальною підмножиною $\{k_1,\ldots, k_m\}$ з $u=U_{k_m}$, та ми хочемо модифікувати P, щоб знайти оптимальну підмножину $\{k_1,\ldots, k_{m-1}\}$ з $u=U_{k_{m-1}}$. По-перше, ми маємо видалити k_m з Q, якщо він там знаходиться. Потенційно, зменшивши u та/або видаливши працівника, ми могли звільнити більше грошей для найму працівників. Але яких працівників ми маємо найняти?

Очевидно, ми не можемо наймати працівників, що вже найняті. Також, єдиною причиною, за якою ми видалили працівника k з P є те, що u впало нижче U_k, а оскільки u тільки зменшується, цей працівник ніколи не може бути найнятим знову. Отже, ми можемо підтримувати просту чергу працівників, впорядковану за кваліфікацаєю, та просто наймати наступного працівника, доки є гроші. Також необхідно видаляти працівників з черги, коли зменшується u, але це можна робити, просто пропускаючи працівників, яких ми не можемо найняти.

Кожен працівник може бути доданим до оптимальної множини не більше одного разу та видаленим також не більше одного разу. Кожен із цих кроків потребує константного часу, тобто основний алгоритм потребує часу $O(N)$. Однак, початкове сортування потребує $O(N\log N)$ часу.

3. РОІ

Умова. Пловдивська Олімпіада з Інформатики (РОІ) проходила згідно з такими незвичними правилами. Було N учасників і T задач. Кожна задача оцінювалась з використанням тільки одного тесту. Таким чином, для кожного учасника і кожної задачі було тільки дві можливості: або учасник розв'язав задачу, або не розв'язав. Не було часткових оцінювань розв'язків задач.

Кількість балів, що відповідало кожній задачі, визначалось після змагань, і дорівнювала кількості учасників, котрі не розв'язали задачу. Бали кожного учасника підраховувались як сума балів, що відповідають задачам, що розв'язав цей учасник.

Філіп брав участь у змаганнях, але він заплутався у складних правилах оцінювання, і зараз він, дивлячись на результати, невзмозі визначити своє місце у фінальному протоколі. Допоможіть Філіпу написати програму, яка підрахує його бали і його місце у фінальному протоколі.

Перед змаганнями участникам присвоїли унікальні номери від 1 до N включно. Номер Філіпа позначимо P. У фінальному протоколі учасники перелічені в порядку спадання набраних ними балів. У випадку рівності балів, першими будуть перелічені учасники, які розв'язали більше задач. У випадку рівності кількості розв'язаних задач, учасники з однаковими результатами будуть перелічені у порядку зростання їхніх номерів.

Завдання. Напишіть програму, яка за заданою інформацією про те, які задачі були розв'язані якими учасниками, визначить кількість балів у Філіпа та його місце у фінальному протоколі.

Обмеження. $1 \le N \le 2000$ — кількість учасників, $1 \le T \le 2000$ — кількість задач, $1 \le P \le N$ — номер Філіпа.

Вхідні дані. Ваша програма має прочитати зі стандартного потока вводу такі дані.

- Перший рядок містить цілі числа N, T та P, розділені пропусками.
- Наступні N рядків описують, які задачі були розв'язані якими учасниками. k-й з них описує, які задачі були розв'язані учасником з номером k. Кожен такий рядок містить T цілих чисел, розділених пропуском. Перше з цих чисел означає, чи розв'язав першу задачу учасник з номером k. Друге число означає те ж для другої задачі, і так далі. Ці T чисел можуть бути тільки 0 або 1, де 1 означає, що учасник з номером k розв'язав відповідну задачу, і 0 означає, що він її не розв'язав.

Вихідні дані. Ваша програма має записати у стандартний потік виведення один рядок із двома цілими числами, розділеними одним пропуском. Перше число — кількість балів, котрі Філіп отримав на змаганнях РОІ. Друге число — місце Філіпа у фінальному протоколі. Місце — це ціле число від 1 до N включно, де 1 означає, що учасник розташований зверху фінального протоколу (тобто має найбільшу кількість балів), а N означає, що він розташований звнизу фінального протоколу (тобто має найменшу кількість балів).

Оцінювання. Для набору тестів загальною вартістю 35 балів, ніхто з учасників не набере стільки ж балів, як Філіп.

Приклад вхідних та вихідних даних

Приклад введення
5 3 2 0 0 1 1 1 0 1 0 0 1 1 0 1 1 0

Приклад виведення
3 2

Перша задача не була розв'язана одним учасником, тобто вона оцінюється в 1 бал. Друга задача не була розв'язана двома учасниками, вона оцінюється в 2 бали. Третя задача не була розв'язана чотирма учасниками, вона оцінюється в 4 бали. Таким чином, перший учасник набере 4 бали. Другий учасник (Філіп), а також четвертий і п'ятий учасники наберуть по 3 бали кожен. Третій учасник набере 1 бал. Учасники з номерами 2, 4 і 5 мають однакову кількість балів і розв'язали одну і ту ж кількість задач, тому відповідно до другого правила (у цьому випадку учасники розташовуються у порядку зростання їхніх номерів) Філіп опиниться перед учасниками з номерами 4 і 5. Таким чином, у фінальному протоколі Філіп буде на другому місці, після учасника з номером 1.

Рекомендації щодо розв'язання

Ця задача є простою задачею на реалізацію. Після зчитування даних із файлу, перший прохід може використовуватись, щоб підрахувати кількість людей, що не розв'язали кожну із задач (і, отже, кількість балів, назначених кожній із задач). Другого проходу тоді достатньо, щоб визначити для кожного учасника кількість розвя'заних задач та кількість балів.

Немає необхідності повністтю визначати фінальний рейтинг: місце Філіпа є просто кількістю учасників, що стоять перед ним у фінальному протоколі плюс один. Це можна визначити, порівнюючи кожного з учасників з Філіпом. Учасник C стоїть над Філіпом тоді і тільки тоді, якщо

- C має більше балів ніж Філіп, або
- C має стільки ж балів як Філіп, але розв'язав більше задач, або
- C має стільки ж балів як Філіп, та розв'язав стільки ж задач, але має менший номер.

4. РОДЗИНКИ

Умова. Відома пловдивська шоколадниця Боні хоче розрізати плитку шоколаду з родзинками. Плитка має прямокутну форму і складається з одиничних квадратних шматочків. Шматочки вирівняні паралельно краям плитки так, що вони формують N рядків та M стовпців, усього виходить $N \times M$ шматочків. На кожному зі шматочків є одна або більше родзинок, і ніяка родзинка не лежить між шматочками і не перетинає розріз між ними.

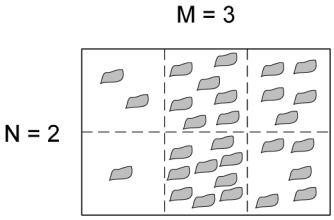

Рис. 1

Спочатку плитка шоколаду є одне ціле. Боні хоче розрізати її на все менші і менші частки, пока вона, нарешті, не розріже всю плитку шоколаду на $N \times M$ одиночних шматочків. Оскільки Боні дуже зайнята, вона попросила свого асистента Петра допомогти розрізати плитку шоколаду. Петро робить тільки прямі розріз від краю до краю частки. Він хоче, щоб йому платили за кажен розріз, який він зробить. У Боні зовсім немає грошей, але у неї залишилась нескінчена кількість родзинок, і вона збирається розраховуватись із Петром родзинками. Петра це влаштовує, але за такої умови: кожен раз, коли він розрізає частину шоколаду на дві менші частини, він отримує стільки ж родзинок, скільки було на вихідній частині.

Боні хоче заплатити Петру якомога менше. Вона знає, скільки родзинок знаходиться на кожному з $N \times M$ шматочків. Вона може обрати порядок, у якому дає Петру решту частин, і вона також може казати Петру, які саме розрізи робити (горизонтальні чи вертикальні) і в якому саме місці. Допоможіть Боні вирішити, як розрізати плитку шоколаду на одиночні шматки так, щоб розплатитися з Петром якомога меншою кількісттю родзинок.

Завдання. Напишіть програму, яка за заданою кількістю родзинок на кожному з одиночних шматочків визначить мінімальну кількість родзинок, котрими Боні має розрахуватись з Петром.

Обмеження. $1 \le N, M \le 50$ — кількість шматочків вздовж кожної зі сторін плитки шоколаду, $1 \le R_{k,p} \le 1000$ — кількість родзинок на шматочку у k-му рядку і p-му стовпці.

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

- Перший рядок містить два цілих числа N і M, розділені пропуском.
- Наступні N рядків описують, скільки родзинок знаходиться на кожному шматочку шоколаду. k-й з цих N рядків описує k-й рядок плитки. Кожен такий рядок містить M цілих чисел, розділених одиночними пропусками. Ці цілі числа описують шматочки у відповідному рядку плитки зліва направо, p-е з чисел у k-му рядку (серед цих N рядків) повідомляє, скільки родзинок знаходиться на шматочку, що розміщений у k-му рядку і p-му стовпчику.

Вихідні дані. Ваша програма має записати у стандартний потік виведення один рядок, що містить одне ціле число: мінімальну кількість родзинок, котрими Боні мала розрахуватися з Петром.

Оцінювання. Для набору тестів загальною вартістю 25 балів, N і M не будуть перевищувати 7.

Приклад вхідних та вихідних даних

Приклад введення	Приклад виведення
2 3 2 7 5 1 9 5	77

Один із багатьох можливих способів досягнути вартості, що становить 77 родзинок, такий:

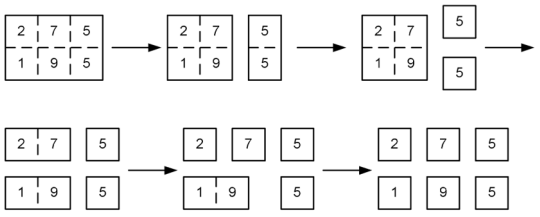

Рис. 2

Перший розріз, який Боні просить зробити Петра, відділяє третій стовпчик від решти плитки шоколаду. Боні повинна заплатити за це Петру 29 родзинок.

Потім Боні дає Петру меншу з двох частин — ту, яка складається з двох шматочків по 5 родзинок у кожному, і просить Петра розрізати цю частину на дві в обмін на 10 родзинок.

Після цього Боні дає Петру більшу частку, у якій залишились шматочки з 2, 7, 1 і 9 родзинками відповідно. Боні просить Петра розрізати її по горизонталі, відділяючи перший рядок від другого, і платить 19 родзинок.

Після цього Боні просить Петра розрізати ліву верхню частину за 9 родзинок. Нарешті, Боні просить Петра розрізати ліву нижню частину, сплачуючи 10 родзинок.

Загальна вартість складає 29+10+19+9+10=77 родзинок. Не існує способу розрізати цю плитку шоколаду на складові 6 шматочків за меншу віртість.

Рекомендації щодо розв'язання

У довільний момент роблячи розрізи, ми маємо множину незалежних підзадач — шматків шоколаду. Якщо ми знайдемо оптимальний розв'язок для кожного зі шматків, разом ми отримаємо оптимальний розв'язок для всієї плитки. Це прямо вказує на можливість використання динамічного програмування.

Кожна підзадача, яку ми зустрічаємо, відповідає прямокутній частині плитки, та може бути описана чотирма координатами, а саме двома x та двома y координатами — координатами верхнього лівого та правого нижнього кутів. Отже, ми маємо розв'язати $O(N^4)$ підзадач.

Щоб розв'язати задану підзадачу, потрібно перебрати всі можливі розрізи. Усього є $O(N)$ можливих розрізів — не більше $N-1$ горизонтальних та $N-1$ вертикальних. Кожен можливий розріз дає дві нові, менші підзадачі, які ми розв'яжемо рекурсивно. Очевидно, рекурсія зупиняється, коли ми досягаємо шматка розміром 1×1.

Припустимо, що в нас є функція $S(x_1,y_1,x_2,y_2)$, що повертає кількість родзинок у прямокутнику з координатами (x_1,y_1) та (x_2,y_2) за константний час.

Використовуючи цю функцію, ми зможемо розв'язати всю задачу за час $O(N^5)$. Будемо використовувати рекурсію із запам'ятовуванням. Для довільної з $O(N^4)$ підзадач, по-перше, перевіремо, чи не обчислювали ми вже її розв'язок. Якщо запам'ятали — просто повернемо це обчислене значення. Інакше, робимо таке. Вартість першого розрізу є $S(x_1,y_1,x_2,y_2)$, яку за припущенням можна обчислити за $O(1)$. Для кожного можливого розташування першого розрізу рекурсивно визначимо вартість решти розрізів для кожної з підзадач, та виберемо оптимальний розріз. Результат запам'ятаємо.

Не вистачає тільки однієї частини — S. Усі можливі значення можуть бути попередньо обчислені за $O(N^4)$ та збережені в масиві. Інший підхід — використовувати двомірні префіксні суми. Нехай A буде вхідним масивом та нехай

$$B_{x,y} = \sum_{i<x}\sum_{j<y} A_{i,j}\ .$$

Значення B називаються двовимірними префіксними сумами. Їх можна обчислити за формулою

$\forall\ x,y > 0$: $B_{x,y}=B_{x-1,y} + B_{x,y-1} - B_{x-1,y-1} + A_{x-1,y-1}$.

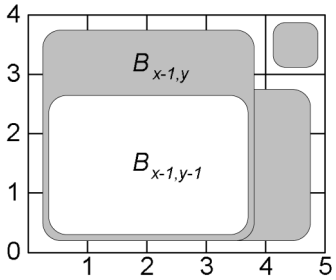

Рис. 3

Маючи двомірні префіксні суми, можна обчислити суму в довільному прямокутнику, використовуючи схожу формулу. Сума в прямокутнику з кутами (x_1,y_1) та (x_2,y_2) є

$S(x_1,y_1,x_2,y_2)=B_{x_2,y_2}-B_{x_1,y_2}-B_{x_2,y_1}+B_{x_1,y_1}$.

(Далі буде)