ЗАДАЧІ XXV ВСЕУКРАЇНСЬКОЇ ОЛІМПІАДИ З ІНФОРМАТИКИ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇХ РОЗВ'ЯЗУВАННЯ

Бондаренко В.В.

Закінчення, початок у №4 за 2012 рік

ЗАВДАННЯ ДРУГОГО ТУРУ

1. Розфарбування

На планеті Олімпія щорічно проводиться традиційна гра у «Фарбувашки». Місцем проведення цієї гри є велике олімпійське поле $N \times M$, кожна клітинка якого є або білою, або чорною. Двоє гравців ходять по черзі, і загалом здійснюють на двох рівно K ходів. Під час свого ходу гравцю дозволяється замалювати власним кольором усі клітинки будь-якого *одного* прямокутника, чия висота і ширина не перевищує D. Перший гравець замальовує клітинки у білий колір, а другий — у чорний. Після закінчення K-го ходу підраховується остаточний рахунок гри — кількість клітинок кожного кольору. Переможцем оголошується той гравець, у чий колір розмальовано більшу кількість клітинок, ніж у супротивника. Тому метою кожного гравця є максимізація кількості клітинок його власного кольору в остаточному розфарбуванні поля.

Завдання. Напишіть програму **paint**, що за заданим початковим полем, кількістю ходів й обмеженням на прямокутник, який може зафарбувати гравець, знайде остаточний рахунок гри за оптимальної стратегії обох гравців.

Вхідні дані. Перший рядок вхідного файлу paint.dat містить чотири натуральних числа: N, M, D і K ($N \le 400$, $M \le 400$, $D \le 400$, $K \le 10^9$) — висота і ширина поля, обмеження на розмір прямокутника, який може зафарбовувати гравець, кількість ходів, що належить виконати до підведення остаточного рахунку. У наступних N рядках розташовано по M символів: W, якщо відповідну клітинку розфарбовано у білий колір, і B у випадку, якщо клітинку поля розфарбовано у чорний колір.

Вихідні дані. Єдиний рядок вихідного файлу paint.sol має містити два цілих числа — кількість білих і кількість чорних клітинок в остаточному розфарбуванні за умови оптимальної гри обох гравців.

Оцінювання. Принаймні у 50% тестів N, M, D і K не перевищують 100.

Приклад вхідних і вихідних даних

paint.dat	paint.sol
3 3 2 1 BWB BBW WBB	6 3

Рекомендації щодо розв'язування

Нехай деяке початкове розфарбування P містить хоча б одну клітинку чорного кольору і гравцям у сумі належить зробити K ходів до завершення гри. Позначимо через $F(R)$ максимальну кількість білих клітинок в остаточному розфарбуванні, що може досягнути перший гравець за оптимальної гри обох, якщо початкове розфарбування — R. Оберемо клітинку чорного кольору у P і замалюємо її у білий колір, позначимо отримане розфарбування P'.

Твердження 1. Для будь-якої фіксованої послідовності ходів кількість клітинок білого кольору в остаточному рахунку при початковому розфарбуванні P' не менша, аніж при початковому розфарбуванні P.

Доведення. Справді, розглянемо будь-яку клітинку дошки. Якщо внаслідок послідовності ходів відбувається перефарбування цієї клітинки, то її остаточний колір не залежить від початкового розфарбування. Якщо ж унаслідок послідовності ходів не відбувається перефарбування клітинки, то її колір залежить лише від початкової позиції. Але за побудовою всі білі клітинки P є білими й у P', тому кількість клітинок білого кольору в остаточному рахунку не зменшиться при заміні початкового розфарбування P на P'.

Твердження 2. $F(P') \ge F(P)$.

Доведення. Розглянемо дві гри: гру A «фарбувашки» за початкового розфарбування P і гру B «фарбувашки» за початкового розфарбування P'. Нехай ми знаємо оптимальну стратегію для гри A і намагаємось побудувати стратегію, що дає не гірший результат для гри B. Зробимо хід у грі B, який є оптимальним першим ходом у грі A, позначимо його u. Нехай другий гравець, який слідує оптимальній стратегії в B, відповів нам деяким ходом v (якщо, звісно, $K \ne 1$). Тепер припустимо, що у грі A було здійснено два ходи u та v. Нехай тепер оптимальним ходом є x. Здійснимо x у грі B. Якщо $K \ne 3$, отримаємо у відповідь деякий хід z, і знову оберемо хід, виходячи з того, що у грі A здійснено вже u, v, x, z. Тобто для будь-якої послідовності ходів непарної довжини, здійсненої у грі B, ходом у відповідь ми обираємо оптимальний (за мірками гри A) хід у відповідь на цю ж саму послідовність ходів. Зауважимо, що оскільки перший гравець грав оптимально на кожному кроці (за мірками гри A), то отримана послідовність ходів призводить до не гіршого рахунку у грі A, аніж $F(P)$. У силу твердження 1 ця послідовність ходів у B призводить до не гіршого рахунку, аніж у грі A. Тобто, перший гравець може гарантувати собі рахунок не гірший, аніж $F(P)$, незалежно від ходів суперника, а, отже, $F(P') \ge F(P)$.

Твердження 3. Для обох гравців існує оптимальна стратегія, що включає лише перефарбування прямокутників розміром $\min(N, D) \times \min(M, D)$.

Доведення. Справді, нехай це не так, і на деякому кроці одному з гравців *необхідно* для досягнення оптимального результату перефарбувати прямокутник Q менших розмірів. Але з твердження 2 випливає, що використавши для перефарбування прямокутник з розміром $\min(N, D)\times\min(M, D)$, який накриває Q, гравець отримає результат не гірший, аніж при перефарбуванні прямокутника Q.

Твердження 4. Нехай у початковому розфарбуванні було W білих клітин і перший гравець зробив хід, яким перефарбував X чорних клітин у білий колір. Тоді кількість білих клітин в остаточному розфарбуванні при оптимальній грі обох буде рівною $W+X$ при непарному K і $W+X-\min(N, D)\cdot \min(M, D)$ при парному.

Доведення. Нехай баланс розфарбування — це різниця між кількістю білих і кількістю чорних клітинок. Тоді перший гравець намагається максимізувати баланс, а другий — мінімізувати. Нехай у початковому розфарбуванні баланс дорівнює T, тоді після першого ходу баланс дорівнюватиме $T+2\cdot X$. Унаслідок твердження 3 можна вважати, що обидва гравці на кожному своєму ході перефарбовують прямокутник розмірами $\min(N, D)\times\min(M, D)$. Тому, починаючи з другого ходу, незалежно від ходів суперника, у кожного з гравців є можливість змінити баланс на $2\cdot\min(N, D)\cdot\min(M, D)$ у вигідну для нього сторону.

Розглянемо кінцевий баланс: $T+2\cdot X-A_2+A_3-A_4+ \ldots +(-1)^{k+1}\cdot A_k$, де A_i — це зміна балансу на i-му кроці. Позначимо $\min(N, D)\cdot\min(M, D)$ за U. Зрозуміло, що перший гравець може собі гарантувати щонайменше $Q=T+2\cdot X-2\cdot U+2\cdot U-2\cdot U+ \ldots +2\cdot(-1)^{k+1}\cdot U$, змінюючи на кожному своєму кроці баланс на $+2\cdot U$. З іншого боку, другий гравець може гарантувати собі результат, не більший від Q, змінюючи на кожному своєму кроці баланс на $-2\cdot U$. Тому при оптимальній грі обох баланс у кінцевому розфарбуванні буде рівний Q. З цього маємо, що при парному K баланс дорівнює $T+2\cdot X-2\cdot U$, а при непарному — $T+2\cdot X$. Тобто при оптимальній грі при парному K маємо $W+X-U$, а при непарному — $W+X$ білих клітин.

Для розв'язання задачі залишається знайти прямокутник розмірами $\min(N, D)\times\min(M, D)$ з максимальною кількістю чорних клітин.

1. Наївний розв'язок $O(N\cdot M\cdot\min(N, D)\cdot\min(M, D))$ набирає 50 балів.

2. Розв'язок $O(N\cdot M\cdot\min(N, D))$ з одномірними частковими сумами набирає 100 балів. Для кожного рядка поля i підрахуємо величину $S[i][k]$, що дорівнює кількості клітин чорного кольору з початку рядка до його k-ї клітини включно. Нехай $S[i][0]=0$. Тоді, щоб за $O(N)$ знайти кількість клітин чорного кольору на прямокутнику $[x,y]\times[c,d]$, необхідно підрахувати суму $S[x][d]-S[x][c-1]+S[x+1][d]-S[x][c-1]+ \ldots +S[y][d]-S[y][c-1]$.

3. Розв'язок $O(N\cdot M)$ із двомірними частковими сумами набирає 100 балів.

Для кожної пари індексів поля (i, j) підрахуємо величину $S[x][y]$, рівну кількості клітин чорного кольору на прямокутнику $[1, x]\times[1, y]$. Нехай $S[0][y]=S[x][0]=0$. Тоді кількість клітин чорного кольору на прямокутнику $[x, y]\times[c, d]$ можна розрахувати так: $S[y][d]-S[x-1][d]-S[y][c-1]+S[x-1][c-1]$. Щоб підрахувати $S[x][y]$ за $O(N\cdot M)$ можна використати динамічне програмування: $S[x][y]=S[x-1][y]+S[x][y-1]-S[x-1][y-1]+c$, де $c=1$ у випадку, коли (x, y) чорного кольору, і $c=0$, якщо ні.

2. Буфер обміну

Барт Сімпсон, герой мультсеріалу «Сімпсони», як покарання за те, що прогулює уроки, має надіслати директору школи електронного листа. У листі Барт повинен N разів надрукувати фразу «Я більше не буду прогулювати уроків». На думку директора, написання такого листа справить на Барта вплив, і він більше ніколи не буде прогулювати школу. Наївний директор!

Замість того, щоб N разів друкувати потрібну фразу, Барт надрукував її один раз, після чого вирішив скористатися буфером обміну. За одну операцію Барт або копіює увесь поточний вміст листа до буфера обміну, або вставляє скопійований до буфера текст у лист.

Завдання. Напишіть програму **copypast**, яка знаючи, скільки фраз має бути в листі директору, визначить, за яку найменшу кількість операцій з буфером обміну Барт зможе скласти листа, кількість фраз у якому дорівнює потрібній.

Вхідні дані. Вхідний файл copypast.dat містить єдине число N — кількість фраз «Я більше не буду прогулювати уроків», які мають потрапити до електронного листа. Число N — натуральне і не перевищує $2\cdot10^9$.

Вихідні дані. Вихідний файл copypast.sol повинен містити єдине ціле число — найменшу можливу кількість операцій із буфером обміну, після яких у листі буде рівно N фраз.

Оцінювання. У 25% тестів N не перевищує 10. У 65% тестів N не перевищує 3000.

Приклад вхідних і вихідних даних

copypast.dat	copypast.sol
12	7

Пояснення. Після того, як Барт надрукував першу фразу, він може виконувати такі операції з буфером обміну:

1. Копіює поточний вміст листа (1 фразу).
2. Вставляє скопійований текст (фраз у листі стає 2).
3. Вставляє скопійований текст (фраз стає 3).
4. Вставляє скопійований текст (фраз стає 4).
5. Копіює поточний вміст листа (4 фрази).
6. Вставляє скопійований текст (фраз стає 8).
7. Вставляє скопійований текст (відтак лист містить необхідні 12 фраз).

Рекомендації щодо розв'язування

Барт діє в такий спосіб: копіює початкову фразу, вставляє її a_1 разів, копіює утворені a_1+1 фразу,

вставляє їх a_2 разів, копіює утворені $(a_1+1)(a_2+1)$ фраз, вставляє їх a_3 разів, …, копіює утворені $(a_1+1)(a_2+1)\ldots(a_{k-1}+1)$ фраз, вставляє їх a_k разів, унаслідок чого дістає $(a_1+1)(a_2+1)\ldots(a_k+1)=n$ фраз. Для зручності позначимо a_i+1 через p_i. Неважко підрахувати, що Барт зробив усього $p_1+p_2+\ldots+p_k$ операцій і отримав число $n=p_1p_2\ldots p_k$. Отже, початкова задача зводиться до такої: розкласти число n на множники $p_1, p_2, \ldots, p_k$ так, щоб сума $p_1+p_2+\ldots+p_k$ була мінімальною. Залишається помітити, що якщо хоча б одне із чисел $p_1, p_2, \ldots, p_k$ у відповідному розкладі не буде простим, то його можна розкласти хоча б на два множники, більші за 1, зменшивши при цьому суму: якщо, наприклад, $p_1=d_1d_2$ і $d_1>1$, $d_2>1$, то $d_1+d_2<p_1$, а тому $d_1+d_2+p_2+p_3+\ldots+p_k<p_1+p_2+\ldots+p_k$. Отже, сума $p_1+p_2+\ldots+p_k$ є найменшою тоді, коли $n=p_1p_2\ldots p_k$ — розклад числа n на прості множники (при цьому деякі з чисел $p_1, p_2, \ldots, p_k$ можуть збігатися). Значить, щоб отримати відповідь, число n треба розкласти на прості множники, а потім знайти їхню суму.

Щоб розкласти задане число n на прості множники, можна перебрати в порядку збільшення всі потенційні дільники цього числа — числа від 2 до $n-i$, щойно n націло поділиться на якесь із цих чисел k, зменшувати значення n у k разів (і продовжувати перебір із k, а не з $k+1$, на випадок, якщо n ділиться на k у деякому степені). Утім, щоб суттєво покращити час роботи в найгіршому випадку, можна здійснювати перебір не від 2 до n, а від 2 до $\sqrt{n}$. Якщо після закінчення перебору, після всіх ділень, n не дорівнюватиме 1, це означатиме, що поточне значення n — просте число: інакше хоча б один із його дільників не перевищував би $\sqrt{n}$. Під час реалізації не слід забувати також, що початкове значення n може дорівнювати 1, у цьому випадку правильна відповідь — 0.

Знехтувавши часом, потрібним на ділення, можна оцінити складність поданого алгоритму як $O(\sqrt{n})$.

До розв'язання задачі можна підійти з допомогою динамічного програмування. У даному випадку стандартне його застосування буде таким: послідовно для всіх пар чисел k, m, $1\le k\le n$, $1\le m\le k$, ми підраховуємо найменшу кількість операцій $f(k, m)$ із буфером обміну, які приводять до стану (k, m), де k — кількість фраз у листі, а m — кількість фраз, які на даний момент зберігаються в буфері обміну. При цьому $f(1,1)=1$, а коли $k>1$, то $f(k, m)$ дорівнює: $f(k-m, m)+1$, якщо $k-m\ge m$; $\min\{f(k, i)+1 \mid 1\le i\le k-1\}$, якщо $m=k$; ∞ в інших випадках («∞» позначає, що відповідного стану неможливо досягти в принципі; на практиці можна використовувати досить велике число, яке достеменно перевищує правильну відповідь). Коли ми підрахуємо всі значення, відповіддю буде число $\min\{f(n, i) \mid 1\le i\le n-1\}$.

Складність такого алгоритму можна оцінити як $O(n^2)$; кількість пам'яті, яку використовує алгоритм, теж дорівнює $O(n^2)$. Але якщо зауважити, що $f(k,m)\ne\infty$ лише в тому випадку, якщо m є дільником k, то для кожного фіксованого k можна знаходити всі дільники за $O(\sqrt{k})$ і рахувати $f(k, m)$ лише для відповідних значень m. Це дасть оцінку $O(n\sqrt{n})$ як на час виконання алгоритму, так і на використовувану пам'ять (якщо зберігати результати обчислень для кожного k не індексованим по m масивом, а списком).

Цікавим є той факт, що коли реалізовувати ті самі рекурентні співвідношення з допомогою простої рекурсії, можна зекономити порівняно з простішим підходом динамічного програмування не лише оперативну пам'ять, але й час виконання програми — принаймні, для n достатньо малих, щоб на рекурсію не забракло пам'яті у стеку.

За допомогою динамічного програмування можна створити і такий алгоритм, який працюватиме за $O(\sqrt{n})$. Для цього достатньо зауважити, що якщо виписати кількості фраз, які опинялися в буфері обміну на оптимальному «шляху» до n фраз, матимемо послідовність, у якій кожне наступне число ділиться на попереднє. Якщо ввести функцію $g(k)$, яка дорівнює мінімальній кількості операцій, потрібних для отримання k фраз, зможемо записати співвідношення: $g(1)=0$, $g(k)=\min\{g(i)+k/i \mid i$ ділить k, $1\le i<k\}$, $k>1$. Порахувавши $g(k)$ послідовно для всіх дільників n, матимемо відповідь — $g(n)$. Це займе $O(\sqrt{n}+d^2(n))=O(\sqrt{n})$ часу і стільки ж пам'яті (через $d(n)$ позначено кількість дільників числа n).

3. Перехрестя

Країна Квадроляндія є квадратом $N\times N$ клітинок. Її мешканці використовують систему координат, у якій лівий нижній кут квадрата має координати (0,0), правий верхній: (N,N). У Квадроляндії знаходяться K міст, кожне в точці з цілими координатами. Мешканці Квадроляндії пересуваються по країні паралельно осям координат. Для пришвидшення подорожей до сусідніх країн уряд вирішив побудувати дві швидкісні автомагістралі. Автомагістралі мають бути перпендикулярними між собою і паралельними осям координат. Кожна магістраль з'єднуватиме дві протилежні сторони квадрата.

Відстанню від міста до магістралей буде відстань від міста до найближчої з них. Уряд вирішив розмістити магістралі так, щоб максимальна відстань від міст до магістралей була мінімально можливою.

Завдання. Напишіть програму **cross**, що за довжиною сторони квадрата і розташуванням міст знайде оптимальну відстань і розташування магістралей.

Вхідні дані. У першому рядку вхідного файлу cross.dat містяться два цілих числа: N ($1\le N\le 1\,000\,000$) і K ($1\le K\le 40\,000$) — довжина сторони квадрата і кількість міст відповідно. Наступні K рядків містять по два цілих числа — координати міст (від 0 до N).

Вихідні дані. У єдиному рядку вихідного файлу cross.sol виведіть три цілих числа — шукану оптимальну відстань від найвіддаленішого міста до найближчої магістралі, і координати точки перетину магістралей (перехрестя). Якщо оптимальних відповідей декілька, виведіть будь-яку з них.

Оцінювання

1. Принаймні у 35% тестів $N \le 50$, $K \le 50$
2. Принаймні у 50% тестів $K \le 200$.
3. Принаймні у 65% тестів $K \le 2000$.

Приклади вхідних і вихідних даних

cross.dat	cross.sol
4 3 1 1 2 2 3 3	1 2 2

Рекомендації щодо розв'язування

Відстань d будемо вважати допустимою, якщо існує розташування перехрестя, при якому максимальна відстань від міст до нього не перевищує d. Допустима відстань має такі властивості.

1. Якщо d_1 — допустима відстань, і $d_2 > d_1$, то d_2 — також допустима відстань.

2. Якщо d_1 — не допустима відстань, і $d_2 < d_1$, то d_2 — також не допустима відстань.

Ці властивості дозволяють скористатися бінарним пошуком для знаходження оптимальної відстані як мінімальної допустимої відстані. Попередньо відсортуємо міста за x-координатою.

Наведемо алгоритм перевірки відстані d на допустимість.

Нехай x-координата вертикальної магістралі рівна x_0. Тоді міста з x-координатою з проміжку $[x_0-d, x_0+d]$ будуть задовольняти умови допустимості. Решта міст будуть знаходитися на відстані, більшій за d, від вертикальної магістралі. Отже, відстань від цих міст до горизонтальної магістралі не повинна перевищувати d. Це можливо тоді і лише тоді, коли різниця максимальної і мінімальної y-координат цих міст не перевищує $2d$.

Методом двох вказівників переберемо найлівіше і найправіше міста, які будуть віддалені від вертикальної смуги не більш ніж на d. На кожному кроці це буде певний проміжок $i..j$. Отже, відстань від міст $1..i-1$ та $j+l..k$ до горизонтальної магістралі не повинна перевищувати d. Для ефективної перевірки можливості відповідного розташування горизонтальної магістралі попередньо обчислимо мінімальну і максимальну y-координати міст на проміжках вигляду $1..i$ та $i..к$.

Складність алгоритму — $O(K \log N)$.

4. Опукла оболонка

Задано множину точок на площині, а також N операцій, що модифікують цю множину. Кожна операція може бути одного з двох типів.

1. Додати до множини точку з цілими координатами. При цьому абсциса (тобто X координата) нової точки є строго більшою за абсциси всіх інших точок, які вже знаходяться у множині.

2. Видалити з множини точку з найбільшою абсцисою.

Завдання. Напишіть програму **convex**, яка за заданою послідовністю з N операцій промоделює їх виконання і після кожної з них виведе подвоєну площу опуклої оболонки точок множини. Перед виконанням першої операції множина точок є порожньою. Вважатимемо, що площа опуклої оболонки порожньої множини точок рівна нулю.

Опуклою оболонкою множини точок на площині називається опуклий багатокутник найменшої площі, який містить всі точки з множини. Багатокутник називається опуклим, якщо відрізок, що сполучає будь-які дві його точки цілком належить цьому багатокутнику. Тут під багатокутником розуміється границя фігури разом з її внутрішністю.

Вхідні дані. Перший рядок вхідного файлу convex.dat містить одне ціле число: N ($1 \le N \le 100\,000$) — кількість операцій, які потрібно виконати. Наступні N рядків містять по три цілих числа — інформацію про операції у зашифрованому форматі. Для того, щоб отримати параметри операції з номером i, потрібно зчитати три цілих числа T^*, X^* та Y^* ($0 \le T^* \le 1$, $0 \le X^*, Y^* \le 2\,000\,000\,000$) з $(i+1)$-го рядка, після чого отримати число T за такою формулою: $T = (T^* + S) \bmod 2$, де S — це подвоєна площа опуклої оболонки точок множини до виконання операції з номером i. Можна переконатись, що S завжди є цілим числом.

1. Якщо $T=0$, то чергова операція першого типу і координати (X,Y) нової точки отримуються за такими формулами:

$X = L + ((S + X^*) \bmod 2\,000\,000\,001)$,

$Y = ((Y^* + S) \bmod 2\,000\,000\,001) - 1\,000\,000\,000$.

Тут L — це максимальна з абсцис усіх точок з множини. Якщо множина порожня, то $L = -1\,000\,000\,000$.

2. Якщо $T=1$, то потрібно K разів виконати операцію другого типу. Число K знаходиться за формулою: $K = ((X^* + Y^* + S) \bmod Q) + 1$. Тут Q — це кількість точок у множині.

Гарантується, що за правильного розшифрування всіх операцій виконуються такі обмеження:

1. Координати всіх точок, які додаються до множині, не перевищують 1 000 000 000 за модулем.

2. При додаванні нової точки її абсциса є строго більшою за абсциси всіх точок, які вже лежать в множині.

3. Операція видалення застосовується лише до непорожньої множини точок.

Вихідні дані. Вихідний файл convex.sol має містити N рядків. У рядок з номером i потрібно вивести подвоєну площу опуклої оболонки множини точок, яка утворилася після виконання перших i операцій. Якщо множина точок виявилася порожньою, то площу її опуклої оболонки вважати рівною 0.

Оцінювання. Набір тестів складається з 4 окремих блоків, з такими додатковими обмеженнями:

1. 15% тестів: $N \le 300$.

2. 15% тестів: $N \le 3000$.

3. 20% тестів $N \le 100\,000$, кількість рядків, у яких $T=1$ (після розшифрування) не перевищує 100.

4. 50% тестів: $N \le 100\,000$.

Приклад вхідних і вихідних даних

convex.dat	convex.sol
6	0
0 1000000000 1000000000	0
0 1000000 1001000000	3000000000000
0 1000000 999000000	6000000000000
0 1001500 1000001500	3000000000000
1 85072946 2	0
1 61619603 2	

Пояснення. Після розшифрування тест матиме такий вигляд:

$T=0, X=0, Y=0$

$T=0, X=1000000, Y=1000000$

$T=0, X=2000000, Y=-1000000$

$T=0, X=3000000, Y=0$

$T=1, K=1$

$T=1, K=2$

Рекомендації щодо розв'язування

По-перше, помітимо, що всі операції видалення декількох точок можна розбити на елементарні операції видалення однієї точки. Оскільки кожна точка додається до верхнього ланцюга не більше одного разу, то й у сумі операцій видалення буде не більше ніж N.

Нехай ми вже побудували опуклу оболонку множини точок у певний момент часу. Розглянемо відрізок, що сполучає найлівішу і найправішу точки з множини. Частину опуклої оболонки, яка лежить над цим відрізком разом з двома його кінцями, назвемо верхнім ланцюгом опуклої оболонки. Іншу частину оболонки і два кінці обраного відрізка назвемо нижнім ланцюгом опуклої оболонки. Отже, найлівіша і найправіша точки належать обом ланцюгам одночасно. Верхній і нижній ланцюги разом з обраним відрізком утворюють опуклі багатокутники. Площа всієї опуклої оболонки дорівнює сумі площ цих двох багатокутників. Отже, достатньо навчитись виконувати всі необхідні операції з верхнім ланцюгом, а для нижнього все буде аналогічно.

Розв'язок на 50 балів. Заведемо стек, у якому будуть зберігатись точки верхнього ланцюга. Додавати нову точку будемо так само, як і в алгоритмі Грехема. А саме, видалятимемо останню точку зі стека до тих пір, поки поворот між двома останніми точками стека (назвемо їх A і B) і новою точкою (D) не стане правим (тобто векторний добуток $AB \times AD < 0$). Після цвого додамо нову точку до стека. Для того щоб окрім самого ланцюга знаходити ще й його площу, потрібно разом з кожною точкою зберігати площу опуклого багатокутника, утвореного всіма точками, які в стеку знаходяться перед нею (включаючи її). Тоді при додаванні нової точки достатньо до вже збереженої площі ланцюга додати площу трикутника, утвореного першою й останньою точками зі стека, а також новою точкою.

Ми вже вміємо додавати точки до верхнвого ланцюга і рахувати після цього його площу. Але описаний вище алгоритм не дозволяє видаляти точки з множини. А саме, у разі додавання нової точки ми видаляємо декілька останніх точок зі стека. Тепер під час операції видалення ми маємо знову повернути ці точки до стека. Для цвого заведемо ще один стек, куди ми будемо перекладати точки у разі видалення їх з першого стека. Якщо потрібно повернути декілька видалених точок у перший стек, то ми можемо взяти їх із другого. Потрібно лише для кожного запиту додатково запам'ятати розмір другого стека для того, щоб ми знали, скільки точок потрібно повернути до верхнього ланцюга. Цей алгоритм може обробляти всі типи операцій і має часову складність $O(NM)$, де M — це кількість рядків вхідного файлу, у яких просять видалити декілька останніх точок. Набирає такий розв'язок 50 балів.

Розв'язок на 100 балів. Для отримання повного розв'язку потрібно зрозуміти, яке місце є найповільнішим у цьому алгоритмі. Розглянемо наступну ситуацію: нехай у нас є побудований верхній ланцюг якоїсь множини точок (зберігаєтвся у стеку). Після цього ми додаємо нову точку, яка лежить набагато вище всіх інших, і тому зі стека видаляються всі точки, крім першої. Тепер ми видаляємо цю точку, і всі ті точки знов перекладаються з одного стека в інший. Отже, за 2 операції алгоритм зробив $O(N)$ дій. Якщо їх повторювати багато разів, то складність буде $O(N^2)$.

Замінимо в нашому розв'язку стек звичайним масивом, для якого будемо додатково пам'ятати кількість точок, яка належить верхньому ланцюгу (розмір стека). Під час додавання нової точки знайдемо бінарним пошуком кількість точок, яку потрібно видалити зі стека перед тим, як покласти цю нову точку до нього. Замість перекладання всіх цих точок у другий стек ми покладемо туди лише розмір верхнього ланцюга, а також всю інформацію про останню з точок, що видаляються (тобто її номер у масиві, її координати, а також площа многокутника, пов'язаного з нею). Після цього ми замінимо цю точку на нову, перерахуємо для неї площу і змінимо розмір ланцюга. Водночас усі інші точки, які мали бути видалені, залишаються в масиві без змін (проте вони знаходяться за межами ланцюга). Після видалення останньої точки нам достатнво змінити лише розмір ланцюга і замінити одну точку на ту, що зберігається у другому стеку. Складність такого розв'язку — $O(N \log N)$.

* * *