Informatyczny kącik olimpijski (104): Obcy

W tym odcinku prezentujemy najtrudniejsze zadanie z zeszłorocznej Międzynarodowej Olimpiady Informatycznej.

Nad obcą planetą ma przelecieć statek badawczy, którego załoga chce sfotografować interesujące obszary planety. Na powierzchnię planety naniesiono kwadratową siatkę o rozmiarach $m \times m$ podzieloną na *pola* rozmiaru 1×1. Statek badawczy będzie poruszać się nad *główną przekątną* tej siatki, to znaczy od lewego górnego do prawego dolnego rogu. Każde wykonane zdjęcie obejmie kwadratowy obszar, który składa się z pewnej liczby całych pól i którego przekątna leży na głównej przekątnej całej siatki. Na potrzeby opisu każdy taki kwadratowy obszar nazwiemy *poprawnym kwadratem*.

Danych jest n *interesujących* pól. Naszym celem jest tak zaplanować wykonanie k zdjęć, by każde interesujące pole zostało sfotografowane. Dodatkowo naszym celem jest zminimalizowanie kosztu rozwiązania, za który przyjmujemy łączną liczbę sfotografowanych pól, przy czym wielokrotnie sfotografowane pola liczymy tylko raz. Wartości n, m i k są rzędu 100 000.

Zacznijmy od prostego spostrzeżenia. Rozważmy pewne interesujące pole r i najmniejszy poprawny kwadrat t, który je zawiera. Wówczas poprawne kwadraty zawierające pole r to dokładnie te, których główna przekątna zawiera główną przekątną t.

Dzięki takiemu spojrzeniu nasz problem staje się poniekąd jednowymiarowy. Każde interesujące pole wyznacza pewien *przedział* na prostej zawierającej główną przekątną siatki. Łącznie danych jest n takich przedziałów. Naszym celem jest zaznaczenie k odcinków na tej prostej (każdy odcinek wyznacza jeden sfotografowany obszar) w taki sposób, by każdy przedział był w pełni zawarty w jednym z odcinków. Technicznie rzecz biorąc, przedział i odcinek na prostej to to samo, jednak dla jednoznaczności dane odcinki będziemy nazywać przedziałami. Nieco trudniej wyrazić w tym języku wartość, którą powinniśmy zminimalizować – do tego problemu wrócimy za chwilę.

Zauważmy teraz, że jeśli jeden z danych przedziałów x zawiera się w innym przedziale y, przedział x możemy usunąć z danych wejściowych. Usuwamy więc zbędne przedziały, a następnie sortujemy pozostałe przedziały rosnąco względem ich lewych końców. Dzięki wykonaniu pierwszego kroku przedziały będą posortowane również względem swoich prawych końców.

Teraz czas na pierwsze rozwiązanie zadania. Oznaczmy lewe końce przedziałów przez $l_1, \ldots, l_n$ a prawe przez $p_1, \ldots, p_n$. Skorzystamy z metody programowania dynamicznego. Oznaczmy przez $d[i][j]$ minimalny koszt pokrycia pierwszych i przedziałów za pomocą j odcinków.

Zastanówmy się, jak skonstruowane jest rozwiązanie odpowiadające wartości $d[i][j]$. Pokrywamy q pierwszych przedziałów (dla pewnego q) za pomocą $j-1$ odcinków, a przedziały od $q+1$ do i musimy pokryć osobnym odcinkiem. Wartość q jest, oczywiście, dobrana tak, by zminimalizować koszt. Stąd, dla $j \geqslant 1$ otrzymujemy

$$d[i][j] = \min_{0 \leqslant q < i} d[s][j-1] + (p_i - l_{q+1})^2 - \max(0, (p_q - l_{q+1})^2),$$

gdzie ostatni składnik odpowiada za odjęcie pól sfotografowanych wielokrotnie. Poza tym przyjmujemy $d[0][0] = 0$, $d[i][0] = \infty$ dla $i > 0$ oraz $p_0 = l_1$. Za pomocą tego wzoru możemy wyznaczyć wartość $d[n][k]$ i gotowe. Problem w tym, że to rozwiązanie działa zdecydowanie zbyt wolno. W szczególności, jeśli minimum obliczać będziemy w czasie liniowym od i, to otrzymamy rozwiązanie działające w czasie $\Theta(n^2 k)$.

Obliczanie minimum w podanym wzorze można jednak zrealizować znacznie szybciej. Rozważmy następujący problem. Dany jest zbiór funkcji liniowych, w którym q-ta funkcja opisana jest wzorem $y = a_q \cdot x + b_q$ (wartości a_q oraz b_q są dane). Naszym zadaniem jest zbudowanie struktury danych, która pozwoli obliczać minimum tych funkcji w każdym punkcie. Innymi słowy, zadaniem struktury danych jest obliczenie wartości funkcji $h(x) = \min_q a_q \cdot x + b_q$ dla podanej wartości x. Ponadto struktura powinna pozwalać na dodawanie nowych funkcji do zbioru.

Zastanówmy się najpierw, jakie dane powinna przechowywać struktura danych. Spójrzmy na rysunek 1, gdzie przedstawiono kilka przykładowych funkcji i wyznaczoną przez nie funkcję h. Wykres funkcji h składa się z pewnej liczby *fragmentów*, gdzie każdy fragment to odcinek, półprosta lub prosta (w przypadku, gdy mamy tylko jeden fragment). Każdy fragment leży na jednej z prostych ze zbioru: kolejne (od lewej) fragmenty leżą na prostych o coraz mniejszych wartościach współczynnika a_i.

Aby reprezentować wykres funkcji h, struktura danych pamięta listę L kolejnych fragmentów (od lewej do prawej). Naturalny sposób reprezentacji polega na opisaniu każdego fragmentu przez punkty na jego końcach, jednak nie polecamy tego podejścia. Wymaga ono specjalnego traktowania fragmentów będących prostymi i półprostymi oraz odpowiedniej reprezentacji liczb wymiernych. Znacznie łatwiej pamiętać listę prostych $L = P_1, \ldots, P_t$, które wyznaczają kolejne fragmenty tworzące wykres h. Każdą prostą reprezentujemy tutaj przez parę liczb (a_i, b_i). Mówiąc konkretniej, lista $(a_1, b_1), (a_2, b_2), (a_3, b_3)$ oznacza, że wykres funkcji h przebiega najpierw po prostej $y = a_1 \cdot x + b_1$, następnie po $y = a_2 \cdot x + b_2$ i wreszcie po $y = a_3 \cdot x + b_3$. Co ciekawe, nie musimy tu wcale pamiętać, na jakim przedziale wykres h pokrywa się z każdą z prostych – to możemy obliczać w miarę potrzeby. Korzystamy tu z faktu, że prosta P_i pokrywa się z wykresem h od punktu, w którym P_{i-1} przecina się z P_i, aż do punktu, w którym P_i przecina się z P_{i+1}.

Aby obliczyć wartość $h(x)$, skorzystamy z wyszukiwania binarnego elementów listy L. Chcemy na tej liście znaleźć prostą P_j, która w punkcie x pokrywa się z wykresem funkcji h. Aby zrealizować wyszukiwanie binarne, musimy umieć dla każdej prostej P_i szybko stwierdzić, czy jej indeks i jest mniejszy, równy, czy też większy od j. To potrafimy rozstrzygnąć bez trudu. Jeśli, na przykład, $i > 1$ oraz pierwsza współrzędna punktu przecięcia prostych P_{i-1} i P_i jest większa od x, wnioskujemy, że $j < i$. Analogicznie możemy rozpoznać pozostałe przypadki. Co ważne, przy naszej reprezentacji potrzebne warunki możemy sprawdzić, wykonując jedynie operacje na liczbach całkowitych, o ile tylko odpowiednio przekształcimy niezbędne wzory. Tym samym obliczenie wartości $h(x)$ zrealizować możemy w czasie $O(\log n)$, gdzie n to liczba funkcji liniowych w zbiorze.

Pozostaje powiedzieć, jak dodawać nowe proste do zbioru. Rozważymy tu nieco uproszczony przypadek, gdy

rozpoczynamy od pustego zbioru prostych i następnie dodajemy proste o coraz mniejszych współczynnikach kierunkowych, tj. kolejno dodawane proste są „coraz bardziej pochylone w prawo". Jak się później okaże, taką własność będą miały proste, które będziemy dodawać do zbioru, gdy wykorzystamy strukturę danych do rozwiązania naszego zadania.

Spójrzmy na rysunek 2, który pokazuje, jak zmienia się wykres funkcji h po dodaniu nowej prostej do zbioru. Aby zaktualizować listę L, musimy najpierw usunąć pewną liczbę prostych z końca listy L, a następnie dopisać na jej końcu właśnie dodaną prostą.

Usuwanie prostych łatwo zrealizować w pętli, powtarzając następujący krok. Jeśli punkt przecięcia dodawanej prostej z przedostatnią prostą na liście L leży na lewo od punktu przecięcia przedostatniej i ostatniej prostej z listy L, usuwamy ostatnią prostą z listy L (patrz rysunek 3). W przeciwnym razie kończymy pętlę i dopisujemy dodawaną prostą na końcu listy L. Dodanie jednej prostej do zbioru może spowodować usunięcie wielu prostych z listy L. Niemniej jednak każda prosta zostaje dopisana do listy L raz i może być z niej usunięta co najwyżej jednokrotnie. Wobec tego łączny czas potrzebny na aktualizacje listy L w trakcie dodawania n prostych do początkowo pustego zbioru wynosi $O(n)$. Na tym kończymy opis pomocniczej struktury danych.

Pokażemy teraz, jak wykorzystać powyższą strukturę danych do rozwiązania oryginalnego zadania. Problematycznym elementem wzoru na $d[i][j]$ może się wydawać funkcja kwadratowa, jednak to tylko pozorna trudność. Zmieńmy nieco wzór na $d[i][j]$:

$$\begin{aligned} d[i][j] &= \min_{0\leqslant q<i} d[q][j-1] + p_i^2 - 2p_i\ l_{q+1} + l_{q+1}^2 - \\ &\quad - \left(\max(0,(p_q - l_{q+1}))\right)^2 \\ &= p_i^2 + \min_{0\leqslant q<i} p_i(-2l_{q+1}) + d[q][j-1] + l_{q+1}^2 - \\ &\quad - \left(\max(0,(p_q - l_{q+1}))\right)^2 \end{aligned}$$

Oznaczmy $a_q = -2l_{q+1}$ oraz

$$b_q = d[q][j-1] + l_{q+1}^2 - \left(\max(0,(p_q - l_{q+1}))\right)^2.$$

Przy takich oznaczeniach mamy

$$d[i][j] = p_i^2 + \min_{0\leqslant q<i} a_q \cdot p_i + b_q.$$

co pozwala nam skorzystać z opisanej powyżej struktury danych. W ten sposób otrzymujemy rozwiązanie działające w czasie $O(nk \log n)$. To już lepiej, jednak brakuje nam jeszcze jednego kroku.

Główną przeszkodę na drodze do efektywnego rozwiązania stanowi sztywne ograniczenie na liczbę zdjęć, które można wykonać. Przyjmijmy na chwilę, że zadanie sformułowane jest nieco inaczej, a mianowicie wykonanie każdego zdjęcia wiąże się z kosztem c. W tej sytuacji nie potrzebujemy już tablicy dwuwymiarowej. Oznaczmy przez $d[i]$ minimalny koszt rozwiązania pokrywającego pierwsze i przedziałów. Modyfikując poprzedni wzór, otrzymujemy:

$$d[i] = \min_{0\leqslant s<i} d[s] + (p_i - l_{s+1})^2 - \left(\max(0,(p_s - l_{s+1}))\right)^2 + c.$$

Teraz wystarczy użyć algorytmu analogicznego do tego powyżej. Do obliczenia mamy jednak jedynie n wartości, dlatego poszukiwaną wartość $d[n]$ obliczyć możemy w czasie $O(n \log n)$ i, jak się za chwilę okaże, przyda się to nam do zaprezentowania ostatecznego rozwiązania.

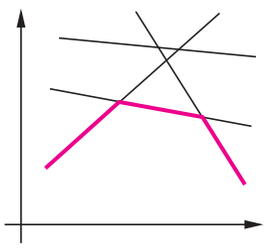
Wróćmy do pierwotnego problemu i oznaczmy przez $f(x)$ minimalny koszt rozwiązania, w którym wykonujemy dokładnie x zdjęć. Naszym zadaniem jest wyznaczenie wartości $f(k)$. Klucz do rozwiązania stanowi *wypukłość* funkcji f. Oznacza to tyle, że przez każdy punkt wykresu f można poprowadzić taką prostą l, by cały wykres znalazł się ponad prostą l lub na niej. Możemy też spojrzeć na tę własność od innej strony: zwiększanie limitu zdjęć k zmniejsza koszt rozwiązania, jednak zysk z każdego kolejnego dodatkowego zdjęcia jest coraz mniejszy. Dowód tej własności jest dosyć skomplikowany i techniczny (a przynajmniej dowód znany autorowi tego tekstu), dlatego nie będziemy go tu podawać. Czytelnikom, którzy chcieliby zdobyć choćby minimalne przekonanie o prawdziwości tej własności, polecamy wykonanie kilku eksperymentów na kartce.

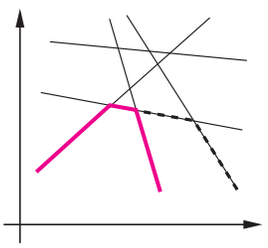
Algorytm opiszemy nietypowo, zaczynając od graficznej interpretacji jego działania (patrz rysunek 4). Naszym celem jest wyznaczenie wartości $f(k)$. Na początku rysujemy wykres funkcji f. Następnie rysujemy prostą l poniżej wykresu f i przesuwamy l do góry (zachowując jej nachylenie) do momentu, gdy dotknie ona wykresu f, tj. napotka pewien punkt $(k', f(k'))$. Naszym celem jest tak dobrać nachylenie prostej l, by trafić w punkt $(k, f(k))$. Wypukłość funkcji f pozwala nam zastosować wyszukiwanie binarne. Jeśli prosta l trafia w punkt $(k', f(k'))$ i $k' > k$, z wypukłości funkcji f wiemy, że jej współczynnik kierunkowy (wyznaczający nachylenie) jest zbyt duży. W przeciwnym razie ($k' < k$) współczynnik jest za mały. Dzięki temu w $O(\log m)$ iteracjach takiego algorytmu możemy znaleźć współczynnik kierunkowy, dla którego prosta l trafi w $(k, f(k))$.

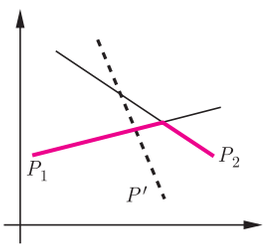
Pozostaje powiedzieć, jak zrealizować jedną iterację powyższego algorytmu. Rozważmy prostą o równaniu $y = a \cdot x + b$ i ustalmy nachylenie tej prostej, czyli wartość współczynnika kierunkowego a. Wówczas szukamy najmniejszej wartości b, dla której prosta przechodzi przez pewien punkt $(k', f(k'))$, czyli $x = k'$ a $y = f(k')$. Innymi słowy, szukamy najmniejszej wartości b, dla której $f(k') = a \cdot k' + b$ dla pewnego k'. To z kolei jest równoważne ze znalezieniem minimum funkcji $g(x) = f(x) - a \cdot x$.

Przyjrzyjmy się bliżej funkcji g. Wartość $g(x)$ oznacza koszt rozwiązania używającego x kwadratów pomniejszony o $a \cdot x$. Zatem minimum g to koszt optymalnego rozwiązania, jeśli wykonanie każdego zdjęcia kosztuje nas $-a$. Koszt ten, jak przed chwilą pokazaliśmy, potrafimy obliczyć w czasie $O(n \log n)$, co daje nam rozwiązanie zadania w czasie $O(n \log n \log m)$.

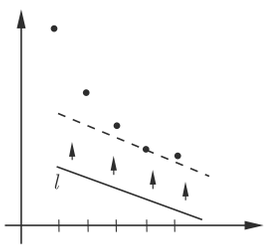
Jakub ŁĄCKI


Rys. 1


Rys. 2



Rys. 3



Rys. 4