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.

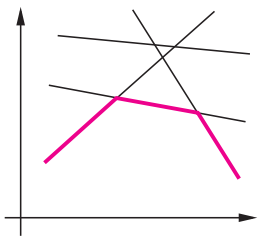
Wróćmy do pierwotnego problemu i oznaczmy przez $f(x)$ minimalny koszt rozwiązania, w którym wykonujemy dokładnie x zdjęć. Naszym zadaniem jest wyznaczenie wartości $f(k)$. Klucz do rozwiązania stanowi *wypukłość* funkcji f. Oznacza to tyle, że przez każdy punkt wykresu f można poprowadzić taką prostą 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 $(k, f(k))$.

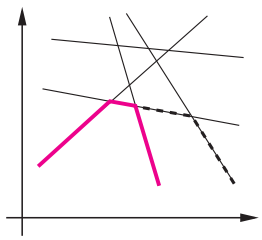
Pozostaje powiedzieć, jak zrealizować jedną iterację powyższego algorytmu. Rozważmy prostą o równaniu $y = a \cdot x + b$ i ustalmy nachylenie tej prostej, czyli wartość współczynnika kierunkowego a. Wówczas szukamy najmniejszej wartości b, dla której prosta przechodzi przez pewien punkt $(k', f(k'))$, 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 $O(n\log n\log m)$.

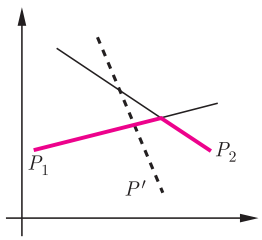
Jakub ŁĄCKI



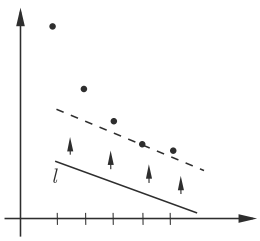
Rys. 1



Rys. 2



Rys. 3



Rys. 4