АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ДОИЗРАВНЯВАНЕ

Първото важно наблюдение, което трябва да направим е, че, като вземем произволен интервал от L метра, за да постигнем изравняване на надморската височина с минимална цена в тези L метра, е нужно, чрез прилагане на копаещата и попълващата операция, да се направи височината на всеки от тези L метра равна на:

- ако L е нечетно: височината на средния по надморска височина метър от тези L (т.е. медианата на множеството от височините на тези L метра);
- ако L е четно: произволна височина, която е между височините на двата средни по височина метра (включително и самите тях).

Доказателство:

Да предположим, че L е нечетно, т.е. $L = 2m+1$ и средният по надморска височина метър е с височина M. Тогава имаме m на брой метра, които трябва да „досипем" до височина M и m на брой метра, които трябва да „изкопаем" до височина M. Нека общата цена, на която се постига изравняването на височина M, е S. Да допуснем, че минималната цена $Smin$ за изравняването на тези L последователни метра се постига на височина, различна от M. Можем да допуснем, че тази височина е $M+x$ ($x>0$) (разсъжденията за случая $M-x$ са идентични). За „досипването" на всички метри, които са с по-малка надморска височина от $M+x$ ще увеличим S поне с $m*x+x$ (увеличението може да е по-голямо, ако има метри с височина между M и $M+x$). От намаленото „изкопаване" на метрите с височина, по-голяма от $M+x$, ще намалим S най-много с $m*x$ (намалението може да е по-малко, ако има метри с височина между M и $M+x$). Като резултат $Smin>S+m*x+x-m*x$, т.е. $Smin>S$, което противоречи на допускането.

При четно L разсъжденията са аналогични.

Нека в нашата задача надморските височини на отсечките от един метър са A_1, A_2, …., A_n . По условие $A_1 \leq A_2 \leq \ldots \leq A_n$. Ако искаме да изравним пътеката само с операции +/-1, то височината, към която трябва да изравним е равна на A_m, където $m=(n+1)/2$, при нечетно n и $m=n/2$ или $n/2+1$, при четно n. Можем да приемем, че винаги $m=(n+1)/2$. За метрите с номера от 1 до m-1 ще досипваме, а за метрите от m+1 до n ще копаем. Тогава минималната сума, за която можем да изравним пътеката само с операции +/-1 е:

$$S=(A_m-A_1)+(A_m-A_2)+\ldots+(A_m-A_m)+(A_{m+1}-A_m)+(A_{m+2}-A_m)+\ldots(A_n-A_m)$$
$$=(m-1)*A_m - A_1 - A_2 - \ldots - A_{m-1} + A_{m+1} + A_{m+2} + \ldots + A_n - (n-m)*A_m$$

Нека с $ps[i]$ означим частичната сума $A_1 + A_2 + \ldots + A_i$.

При нечетно n, $m-1=n-m$ и можем да запишем:

$S= A_n + A_{n-1} + \ldots + A_{m+1} - A_{m-1} - A_{m-2} - \ldots - A_1 = ps[n]-ps[m]-ps[m-1]$.

При четно n, $m=n-m$ и можем да запишем

$S= A_n + A_{n-1} + \ldots + A_{m+1} - A_m - A_{m-1} - A_{m-2} - \ldots - A_1 = ps[n]-ps[m]-ps[m]$

Това съображение е достатъчно за решаване на **подзадача 2**.

За решаване на подзадачи 1 и 3.

Нека F[p] е минималната цена, за която можем да построим пътека с дължина p (пътеката може да включва асансьори и може да се извърви без слизане и качване). Имаме следната зависимост:

$F[p] = MIN\{ F[x] + K + cost(x+1, p) \} = K + MIN\{ F[x] + cost(x+1, p) \}$, където x приема стойностите от 0 до $p-1$, а $cost()$ е минималната цена за изравняване на пътеката от отсечка с номер $x+1$ до отсечка с номер p като се използват само операции +/- 1 (приемаме, че F[0] = -K). Очевидно, F расте, когато p расте.

Тази зависимост служи за динамично изчисляване на $F[n]$, т.е. за решаване на задачата. Ако $cost(x+1,p)$ се изчислява линейно всеки път, то ще се получи алгоритъм със сложност $O(n^3)$, който ще решава **подзадача 1**.

Ако използваме изведената по-горе зависимост, т.е. че при $m = (x+1+p)/2$

- $cost(x+1, p) = ps[p] - ps[m] - ps[m-1] + ps[x]$, ако $p-x$ е нечетно и
- $cost(x+1, p) = ps[p] - ps[m] - ps[m] + ps[x]$, за четно $p-x$.

получаваме, че $cost(x+1,p)$ може да се изчислява за константно време при попълнен предварително масив ps с частичните суми и това води до алгоритъм със сложност $O(n^2)$, който решава и **подзадача 3**.

За решаването на подзадача 4 е нужен алгоритъм, който да не изисква преминаването през всички възможни стойности на x от 0 до p-1 при търсенето на $MIN\{ F[x] + cost(x+1, p), 0 \le x < p$.

Нека $cost'(x+1,p)= ps[m] + ps[m-1]$, ако $p-x$ е нечетно и $cost'(x+1,p)= ps[m] + ps[m]$, ако $p-x$ е четно, където $m=(x+1+p)/2$.

Тогава можем да запишем $cost(x+1, p) = ps[p] + ps[x] - cost'(x+1, p)$

и съответно уравнението на динамичното оптимиране става

$F[p] = K + MIN\{ F[x] + ps[x]+ps[p] - cost'(x+1, p) \}$

Да означим с $value(x, p) = F[x]+ps[p]+ps[x] - cost'(x+1, p)$, т.е. това е функцията, чийто минимум по x търсим, при последователни стойности на p. Това е функция от две променливи x и p.

Нестандартният подход при решаването на задачата е в това, че изследваме поведението на тази функция при фиксирани стойности на x и променящо се p и получените резултати използваме след това за намиране на минимума по x при различните стойности на p.

За да разберете алгоритъма прочетете и осмислете твърденията, дадени по-долу с удебелени букви. Доказателствата можете да четете след това.

1) **При фиксирано x, $value(x,p)$ се явява ненамаляваща функция по p, т.е. $value(x,p+1) \ge value(x,p)$**

Доказателство:

$value(x,p+1)-value(x,p)=F[x]+ps[x]+ps[p+1]-cost'(x+1,p+1)-F[x]-ps[x]-ps[p]-cost'(x+1,p)=ps[p+1]-ps[p]-(cost'(x+1,p+1)-cost'(x+1,p))=A[p+1]-(cost'(x+1,p+1)-cost'(x+1,p))$

За да пресметнем $(cost'(x+1,p+1)-cost'(x+1,p)$, да разгледаме два случая:

а) $p-x$ нечетно => $(p+1)-x$ четно, $((x+1)+(p+1))/2 = ((x+1)+p)/2 = m$

$cost'(x+1, p) = ps[m] + ps[m-1]$; $cost'(x+1, p+1) = ps[m] + ps[m]$

$cost'(x+1,p+1)-cost'(x+1,p)=A[m] = A[(x+1+p)/2] = A[(x+p+2)/2]$

2) *p-x* четно => *(p+1)-x* нечетно, *((x+1)+(p+1))/2 = ((x+1)+p)/2 + 1 = m+1*

cost'(x+1, p) = ps[m] + ps[m] ; cost'(x+1, p+1) = ps[m+1] + ps[m]

cost'(x+1,p+1)-cost'(x+1,p)= A[m+1] = A[(x+1+p)/2+1] = A[(x+p+2)/2]

Тогава

value(x,p+1)-value(x,p)= A[p+1]-(cost'(x+1,p+1)-cost'(x+1,p))=A[p+1]-A[(x+p+2)/2]≥0, тъй като *p+1≥(x+p+2)/2* и масивът с височините е ненамаляващ.

2) Ако *x<y*, то *value(x,p)* расте не по-бавно от *value(y,p)* като функции на *p*, т.е. *value(x,p+1)-value(x,p)≥value(y,p+1)-value(y,p).*

Доказателство:

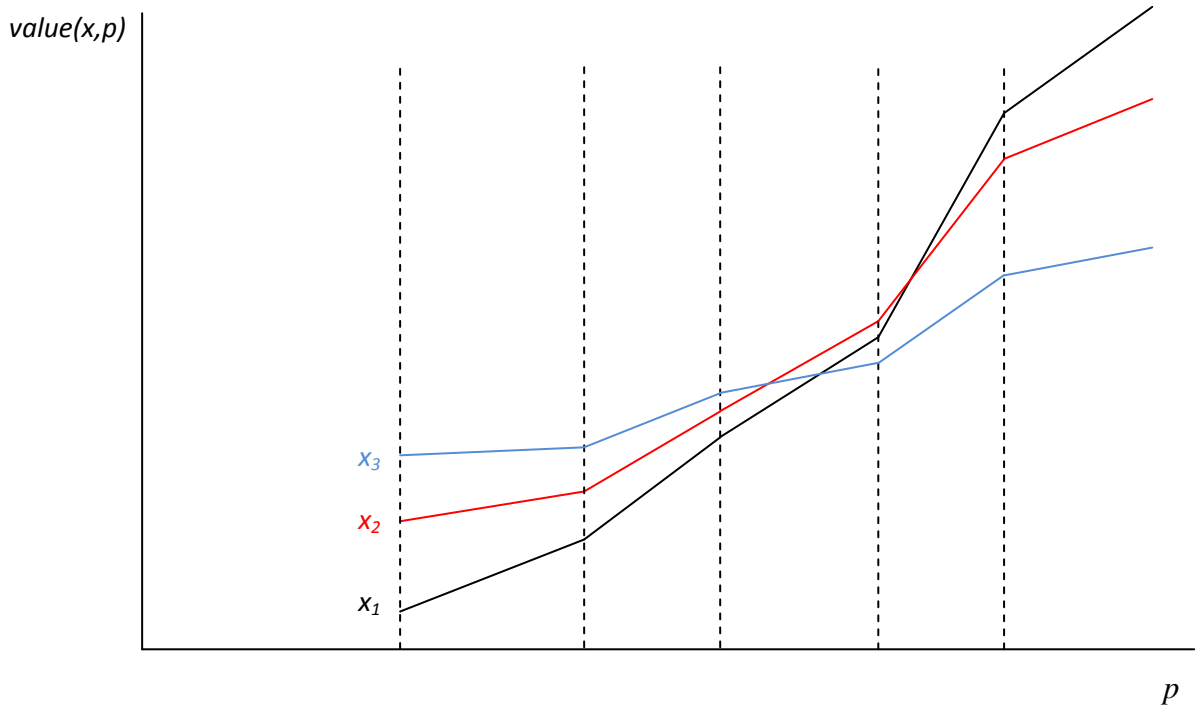
От доказателството на твърдението в т.1) имаме, че

value(x,p+1)-value(x,p)=A[p+1]-A[(x+p+2)/2], а

value(y,p+1)-value(y,p)=A[p+1]-A[(y+p+2)/2]

Тъй като x<y и масивът *A* е ненамаляващ, то *A[(x+p+2)/2]≤A[(y+p+2)/2],* откъдето следва и верността на твърдението в т. 2)

Да не забравяме, че нашата цел е за всяко *p* да смятаме бързо *MIN{value(x,p)}* при *0≤x<p*. И така, фиксирайки различни стойности на *x,* ние получаваме множество от ненамаляващи функции от *p*.



На фигурата са дадени графиките на три функции при $x_1<x_2<x_3$. Както доказахме, те са ненамаляващи и най-бързо расте *value(x_1,p)*, след това *value(x_2,p)* и най-бавно *value(x_3,p).* От доказаните свойства следва, че, ако *x<y* и за някоя стойност p_{xy} на *p* е вярно неравенството *value(x,p_{xy})≥value(y,p_{xy})*, то ще е вярно неравенството *value(x,p)≥value(y,p)* за всяко *p>p_{xy}*. Графично това означава, че ако в един момент графиката на *value(x,p)* премине над графиката на *value(y,p)*, то тя ще остане отгоре за всички следващи стойности на *p*.

Нека сега се върнем към задачата за търсене на *F[p]*, т.е. за търсене на *MIN{value(x,p)}* при $0 \le x < p$.

*Ако **x** е решение за F[p], решението за F[q], $q > p$ ще бъде някое $y \ge x$ (value(z,p) са били по-големи или равни на value(x,p) за всички $z < x$ и ще бъдат по-големи или равни и когато p бъде заменено от някое $q > p$).*

Нека дадена стойност на *x* наречем ***кандидат за решение***, ако съществува вероятност тази стойност да носи минимум на функцията *value(x,p)* за някое *p*. Кога дадена стойност *x* престава да бъде кандидат? – когато в процеса на динамичното пресмятане достигнем до такава стойност на *p*, че $value(x,p) \ge value(y,p)$ за някое $y > x$.

Нека дефинираме функцията *cross(x, y),* която ни дава (за $x < y$) стойността на *p*, такава че $value(x,p) \ge value(y,p)$.

Докато изчисляваме *F[p]* в нарастващ ред на *p*, можем да пазим списък на кандидатите

$0 \le x_1 < x_2 < x_3 < \ldots < x_h < p$, такива, че:

$value(x_1, p) < value(x_2, p) < \ldots < value(x_h, p)$

$cross(x_1, x_2) < cross(x_2, x_3) < \ldots < cross(x_{(h-1)}, x_h)$

Т.е., всеки един от елементите в списъка е кандидат за решение.

Нека сега изчисляваме *F[p]*. Ако $cross(x1, x2) \le p$, то *x2* е по-добър кандидат от *x1* за всички $q \ge p$. Тогава можем да премахнем *x1* от списъка и да проверим с *cross(x2, x3).* В момента, в който получим $cross(x_i, x_{i+1}) > p$, x_i дава решението за *F[p]*.

След като изчислим *F[p],* остава да добавим *p* в списъка на кандидати (ако става за кандидат).

Ако $cross(x_h, p) > n$, тогава *p* няма никога да стане кандидат.

Ако $cross(x_h, p) > cross(x_{(h-1)}, x_h)$, добавяме *p* в края на списъка с кандидати.

В противен случай, $cross(x_h, p) \le cross(x_{(h-1)}, x_h)$, т.е. x_h повече не е кандидат за решение. Следователно, можем да го премахнем от списъка с кандидати и да продължим да пробваме с $x_{(h-1)}$.

Отначало в списъка имаме само $x_1 = 0$.

Функцията *cross()* може да се реализира с двоично търсене.

Сложността на алгоритъма е $O(n*\log n)$ и той решава подзадача 4.

Автор: Йордан Чапъров