АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ОГРАДА

Тъй като всяко дърво е предсдтавено с точка в координатната система, в обясненията по долу говорим не за дървета, а за съответнвите им точки. Каквото и решение на такава задача да предложим, добре е първо да се редуцират координатите така, че да се елиминират x- и y-координатите които отсъстват. За такава обработка ще ни трябват две сортирания със сложност $O(n.\log n)$ и по едно преглеждане на всеки от двата сортирани масива със сложност $O(n)$. Нека след тази редукция сме намерили p различни x-координати x_1, x_2, ..., x_p и q различни y-координати y_1, y_1, ..., y_q, на точки, $p = O(n)$ и $q = O(n)$. Да добавим към тях и по една фиктивна „начална“ координата $x_0 = x_1 - 1$ и $y_0 = y_1 - 1$.

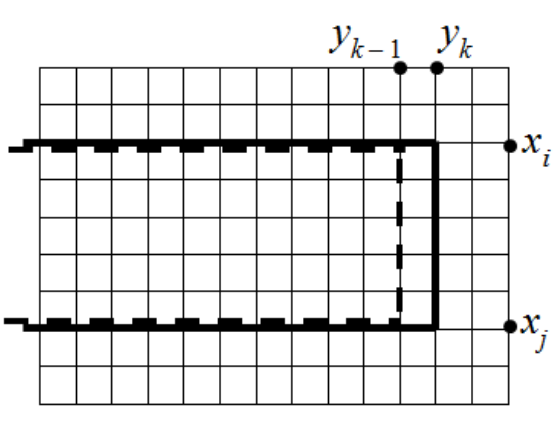
Задачата има тривиално решение, при което се генерират всички възможни p^2q^2 правоъгълника и за всеки от тях се проверява кои от точките лежат на контура на този правоъгълник. Сложността на този алгоритъм, в най-лошия случай, е $O(\binom{p}{2}\binom{q}{2}n) = O(n^5)$ и решение с такава сложност не може да разчита на повече от 10% от точките.

Малко по-добро решение е, след като изберем правоъгълник по един от $\binom{p}{2}\binom{q}{2}$ начина да намерим броя на точките по контура му за константно време, което ще означава сложност $O(n^4)$. Това може да стане като предварително създадем за всяка координата x_i вектор с префиксни суми $X_i[0] = 0$, $X_i[1]$, $X_i[2]$, ...,$X_i[q]$ и за всяка координата y_j вектор с префиксни суми $Y_j[0] = 0$, $Y_j[1]$, $Y_j[2]$, ..., $Y_j[p]$, където $X_i[s]$, $s \neq 0$, е броят на точките с x-координата x_i и y-координати $\leq y_s$, а $Y_j[r]$, $r \neq 0$, е броят на точките с y-координата y_j и x-координати $\leq x_r$. Тогава, броят на точките върху контура на правоъгълник с горен ляв ъгъл (по стара програмистка традиция оста x в нашата координатна система е насочена надолу, а оста y – надясно) в точката (x_a, y_b) и долен десен ъгъл в точката (x_c,y_d) ще бъде

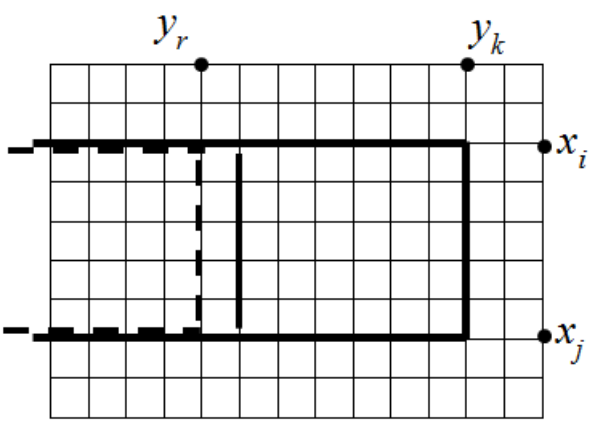
$$X_a[d] - X_a[b-1] + X_c[d] - X_c[b-1] + Y_b[c-1] - Y_b[a] + Y_d[c-1] - Y_d[a] .$$

Векторите с префиксни суми ще са полезни и за решение със сложност $O(n^3)$. За да получим такова решение, трябва при фиксирани две координати в една от посоките, например при фиксирани x_i и x_j, $i<j$, да можем да обработим всички правоъгълници в получената ивица (виж Фиг. 1) с $O(n)$ стъпки. За целта да означим с $\Pi_{i,j}[k]$ броя на точките върху контура на правоъгълника с горен ляв ъгъл в точката (x_i, y_0) и долен десен ъгъл в точката (x_j,y_k). Очевидно

$$\Pi_{i,j}[0]=0;\ \Pi_{i,j}[k] = \Pi_{i,j}[k-1] + (Y_k[j] - Y_k[i-1]) - (Y_{k-1}[j] - Y_{k-1}[i-1]) .$$



Фиг. 1



Фиг. 2

Броят на точките върху контура на правоъгълник от ивицата, започващ в y_r и завършващ в y_k е $\Pi_{i,j}[k] - \Pi_{i,j}[r] + (Y_{r+1}[j-1] - Y_{r+1}[i])$ (виж Фиг. 2). Максимален брой точки при фиксирано k ще получим, ако $\Pi_{i,j}[r] - (Y_{r+1}[j-1] - Y_{r+1}[i])$ е минимално, $r = 0, 1, 2, ..., k-1$. Затова преди да обработваме ивицата, зареждаме една променлива `lmin=MAXINT` в която да помним във всеки момент текущата стойност на минимума. Тогава, след пресмятането на $\Pi_{i,j}[k]$, намираме максималния брой на точките лежащи на контура на правоъгълник с дясна страна в y_k по формулата $\Pi_{i,j}[k]$ – `lmin`. За глобалния максимум `gmax` пресмятаме `gmax = max(gmax,` $\Pi_{i,j}[k]$ – `lmin)`, а `lmin` обновяваме по формулата `lmin = min(lmin,`$\Pi_{i,j}[k] - (Y_{k+1}[j-1] - Y_{k+1}[i])$`).`

Автори: Красимир Манев, Владислав Харалампиев

Коментар по решението на задачата от Йордан Чапъров

По-опитните състезатели със сигурност ще измислят решение O(N^3) сравнително бързо. Нито на мен, нито на Сашо ни отне доста време да го измислим, защото има задачи, които се решават с подобни идеи. Не съм чел целия анализ, но мисля, че идеята на моето O(N^3) и тяхното е една и съща. На мен ми отне доста време да подобря решението, защото мислех за коренно различни идеи, а не толкова да подобря решението си O(N^3).

Самата идея: пак ще търсим максимални леви и десни П-образни фигури, но ще ги търсим по-умно. Добре е да компресираме координатите, така че максималния ни x, y са <= N. Първо, фиксираме горния ред на правоъгълника (така че да има точки на този горен ред). Ще опиша процедура за намиране на ляво П.

Инициализираме масив A[MAX_COORDINATE] -> int, в който отначало пазим следното: A[x] = - (брой точки в горния ред, които са наляво от x). Обработваме точките под горния ред в следния ред: нарастване на реда, при равен ред, нарастване на колоната. За всяка точка p, за която има j точки преди нея в реда:

1. задаваме leftP[p] = max(leftP[p-1], max(A[1], A[2], ..., A[column(p)]) - j), където p-1 е точката преди p на същия ред като p.

2. A[column(p)]++;

В най-доброто решение или ще имаме точка на долната страна на правоъгълника, или може да считаме, че най-долната страна е на ред 0 (след компресирането).

Ако имаме точка p на най-долната страна, тогава решението е numPoints(горен_ред) + leftP[p] + rightP[p] + numPoints(row(p)). Ако не, тогава взимаме двете най-големи стойности на A[] след обработка на всички точки под фиксирания ред.

Идеята е подобна на решението O(N^3). Всъщност числата leftP[] ни казват най-големия брой точки, които можем да вземем от колона като минимизираме загубите от горния и долния ред.

Предполагам, че решението ще върви доста бързо, защото имаме RMQ заявки от координата 1 (може да се ползва Binary Indexed Tree). Все още не съм го написал, защото имам надежда, че ще може да се оптимизира до O(N^2).

Обосновка на надеждата:

Да допуснем, че имаме най-добрия правоъгълник. Тогава няма вертикална отсечка, която свързва горната и долната страна на правоъгълника с повече точки от сегашната лява страна. Тоест сегашната лява страна се явява максимална от всички страни наляво. Ако за фиксирани горна и долна страна направим списък List[] от кандидатите за лява страна, този списък ще бъде сортиран. От този списък, за да получим стойностите на

leftP[] трябва да извадим две числа: Up[] и Down[] - броят точки, които прескачаме отгоре и отдолу. И двата списъка са сортирани.

leftP[column] = max(LeftSide[x] - Up[x] - Down[x] | x in List[], x < column).

LeftSide, Up, Down - растат.