АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ПРАВИ

Оставям доказателствата на някои факти в отделна секция в края на анализа. Повечето факти, които съм доказал са очевидни. Четете доказателства ако не ви се струва така.

Подзадача 1: достатъчно е да проверим къде е x-координатата на пресечната точка на нашата права с всеки от лъчите и да вземем най-голямата.
Решаваме уравнението Ai*x + Bi = C*x + D
=> x = (D-Bi) / (C-Ai)
Сложност **O(N*Q)**

За да подобрим сложността ще трябва да приложим някои трикове. От израза за x виждаме, че най-вероятно нещо интересно се случва в зависимост кое от C и Ai е по-голямо.

(*) Да пробваме да правим двоично търсене по отговора: фиксираме стойност x0 и искаме да разберем дали правата C*x + D ще пресече някой лъч в точка с x-координата x1 : x1 > x0. Да разгледаме всички лъчи, за които Ai < C (случая, когато Ai > C е напълно аналогичен. От тук нататък ще разглеждаме само лъчи, за които Ai < C).
Ако Ai*x0 + Bi < C*x0 + D => лъча i няма да пресече правата в по-голяма x-координата.
Ако Ai*x0 + Bi > C*x0 + D => лъча i ще пресече правата в някоя точка с x-координата x1 > x0.
(Доказателства (1) и (2))

Ако можем бързо да намерим максималната стойност на израза Ai*x0 + Bi за някое i ще можем да правим двоичното търсене бързо! Структурата наречена Upper Envelope (някои го наричат Convex Hull Trick или по друг начин; за повече информация [1]) върши тази работа. Преди да продължите с четенето на анализа е важно да разбирате какви операции може да правим с тази структура и какво се случва като правим всяка операция. Да разгледаме алгоритъм за строене на Upper Envelope на множество от прави:

1. Дадено: множество от прави във вида (Ai*x + Bi).
2. Сортираме правите по нарастване на коефициента Ai (наклона).
3. Инициализираме празен стек от двойки (начален_момент, права) S, където начален_момент е минималната x-координата, за която правата е активна – тази права дава по-голяма стойност на Ai*x + Bi от всички прави с по-малък наклон за x-координати по-големи от начален_момент.
4. За всяка права k от сортирания списък:
    1. Докато S не е празен и има права t на върха си
и пресечната точка на k и t < начален_момент(t) :
премахнете t от върха на стека
    2. Ако S е празен: добави (-infinity, k) към стека.
    3. В противен случай: добави (пресечната точка на k и t, k) към стека.

След изпълнение на алгоритъма в стека ще имаме сортиран списък на прави по началния момент на активност. Можем да правим двоично търсене по този списък, за да намерим стойността на max(Ai*x0 + Bi) за фиксирано x0. Забележете, че веднъж

като премахнем права от стека, няма да я добавим никога повече. Тоест, тази права няма момент на активност и не е част от Upper Envelope.

Подзадача 2: Вече сме готови да решим подзадача 2. Тъй като знаем, че всички прави са с еднакъв наклон (C = 0), можем да изчислим Upper Envelope за всички лъчи с Ai < 0 и да правим заявки по структурата. (Не забравяйте за правите Ai > 0).
Сложност **O(N*log N + Q*log (?)*log N)**, където log (?) е сложността на двоичното търсене по реални числа до необходимата точност.

Подзадача 3: Подзадача 3 всъщност ни казва, че можем да правим заявките "офлайн", тоест може да прочетем всички заявки и да ги обработваме едновременно. Така можем да сортираме всички лъчи и прави по наклон (Ai или Cj) и да ги обработваме в ред на нарастване на наклона. Ако видим лъч, тогава го добавяме към енвелоуп-а. Ако видим права, правим двоично търсене.
Сложност **O(N*log N + Q*log (?)*log N)**

Какъв е проблема да добавим всички лъчи наведнъж в енвелоуп-а ? Може да се окаже, че оптималния лъч i за координатата x0 (от двоичното търсене за правата C*x+D) е бил премахнат от някой друг лъч с наклон по-голям от C. (Ако е лъчът i е бил премахнат от лъч с наклон по-малък от C, това означава, че лъчът i не е бил оптимален).

Ако строим Upper Envelope за всички лъчи със стандартния алгоритъм, стека (списъка с прави), който ще ни трябва, за да отговорим на всеки един въпрос ще бъде образуван в даден момент от изпълнението на алгоритъма.

Как да съхраняваме различните състояния на Upper Envelope, през които преминаваме по време на строене, така че да можем да възстановим бързо състоянието, което ни трябва да отговорим на заявка?

1. Можем да направим Binary Indexed Tree (или интервално дърво), във всеки връх на което, пазим интервал от прави с определен наклон. Крайната сложност на алгоритъм с тази идея може да се сведе до O(N*log N + Q*log N*log N) или O(N*log N*log N + Q*log N). Упражнение: измислете и напишете решение със сложност O(N*log N*log N + Q*log N).
2. *(Идея на Владислав Харалампиев)* Можем да пазим всички състояния на Upper Envelope в дърво. Ще разгледаме до какво води тази идея.

Със съвсем малки модификации на алгоритъма можем да превърнем стека в дърво. Всеки връх на дървото отговаря на една права:

1. Дадено: множество от прави във вида (Ai*x + Bi).
2. Сортираме правите по нарастване на коефициента Ai.
3. Създаваме дърво с корен 0.
4. Масив par[] от родители на всеки връх в дървото.
5. Масив startX[] от минимални x-координати, за които всяка права е активна.
6. Текущия връх V = 0.
7. За всяка права k от сортирания списък:
    1. Докато V != 0 и (пресечна точка на правите V и k) < startX[V] : V = par[V].
    2. Създаваме нов връх U съответстващ на правата k. par[U] = V.
    3. Ако V == 0 : startX[U] = -infinity
    4. В противен случай : startX[U] = (пресечна точка на правите V и k).

След изпълнение на този алгоритъм получаваме дърво. Всеки път от връх V до корена на дървото, съдържа същата информация като стека точно след добавяне на правата, съответстваща на V.

Как да намерим envelope-а, който ни интересува в дървото?
Envolope-а, който е сформиран от всички лъчи с наклон <= C представлява път в дървото.

Как да правим двоично търсене върху този път?
Да си припомним какво ни е необходимо, за да изчислим най-близкия общ предшественик на два върха в дърво ([2]). Изграждаме масив P[v][i], който ще ни каже в кой връх ще се озовем ако започнем във връх v и се придвижим с едно ниво нагоре (към родителя) 2^i пъти. Можем да заменим двоичното търсене със "скачане" със стъпки степени на двойката по пътя към корена.

Подзадача 4: Вече сме готови да решим подзадача 4:
1. Изграждаме envelope дърво.
2. За всяка заявка правим двоично търсене по отговора.
3. За всяко x0 от двоичното търсене "скачаме" по пътя към корена, за да намерим max(Ai*x0+Bi)

Сложност на алгоритъма: **O(N*log N + Q * log (?) * log N)**

Досега предложените решения използват двоично търсене по реална стойност. Това често води до проблеми с време за изпълнение и точност. Само добре написани решения биха минали тестовете.

Трябва ли ни наистина двоичното търсене?
Оказва се, че ако помислим още малко по това какво се случва като пресечем envelope-а с права, можем да премахнем двоичното търсене по реални числа!
Нека C*x + D е правата, за която трябва да отговорим в момента.
Да разгледаме функцията F(x) = max(Ai*x + Bi), където Ai < C. Тази функция е дефинирана и чрез Upper Envelope. Ако Upper Envelope е съставен от лъчите L1, L2, L3, …, Lk, с начални точки S_{L1}, S_{L2}, …, S_{Lk}, $S_{L(k+1)}$ = infinity, тогава F(x) = A_{Li}*x + B_{Li}, за S_{Li} <= x < $S_{L(i+1)}$. Търсим точката с максимално x, където правата C*x + D пресича някой от лъчите, за които Ai < C. Нека търсената x-координата е x1. На всяка стъпка от двоичното търсене (*) ние сравняваме F(x0) с C*x0 + D:
F(x0) > C*x0 + D <=> x0 < x1
F(x0) < C*x0 + D <=> x0 > x1
и F(x1) = C*x1 + D – тоест търсим пресечната точка на F(x) и C*x+D. (x1,F(x1)) = P.
За всяко x0 < x1 : F(x0) > C*x0 + D. В частност, това е вярно за S_{L1}, S_{L2}, …, S_{Lk}. Тоест, съществува j, такова че
F(S_{Lj}) >= C*(S_{Lj}) + D и F($S_{L(j+1)}$) < C*($S_{L(j+1)}$) + D
=> S_{Lj} <= x1 < $S_{L(j+1)}$
P е точка от F(x) => F(x1) = A_{Lj}*x1 + B_{Lj} = C*x1 + D. За да намерим x1 е достатъчно да намерим максималното j (1 <= j <= k) , за което F(S_{Lj}) >= C*(S_{Lj}) + D. Това може да се направи с двоично търсене (по път в дървото).

Подзадача 5:
Вече можем да решим и последната подзадача.

1. Изграждаме envelope дърво.
2. За всяка заявка правим двоично търсене по път в дървото, за да намерим j.
3. Смятаме x1.
4. Не забравяме да работим с правите, за които Ai > C !

Сложността на алгоритъма е **O(N*log N + Q*log N)**

Ако направим наблюдението, че не ни трябва двоично търсене по реални числа по-рано това ще оптимизира алгоритъма за всяка подзадача. Също си спестяваме проблеми с двоично търсене по double.

Доказателства:

(1) Ако Ai < C и Ai*x0 + Bi < C*x0 + D ; нека x1 = x0 + d, където d > 0
=> Ai*x1 + Bi = Ai*x0 + Bi + Ai*d < C*x0 + D + C*d = C*x1 + D
=> за всяко x1 > x0 : Ai*x1 + Bi < C*x1 + D

(2) Ако Ai < C и Ai*x0 + Bi > C*x0 + D ; нека x1 = x0 + d
Ai*x1 + Bi = C*x1 + D
<=> Ai*x0 + Bi + Ai*d = C*x0 + D + C*d
<=> d = (Ai*x0+Bi – C*x0-D) / (C-Ai) . Тъй като Ai*x0 + Bi – C*x0 – D > 0 и C-Ai > 0
=> уравнението има решение за d > 0 => правата ще пресеча лъча в някое x1 > x0.

Препратки:

[1] http://wcipeg.com/wiki/Convex_hull_trick
[2] http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=lowestCommonAncestor#Another%20easy%20solution%20in%20O%28N%20logN,%20O%28logN%29

Автор: Йордан Чапъров