АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
МЕТРО

Тази на пръв поглед геометрична задача всъщност крие в себе си съвсем други неща. По-опитните състезатели трябва бързо да забележат графовата постановка (и какъв всъщност граф задава задачата), но дори справяйки се с това, тепърва трябва да измислят как да построят графа, а също така как бързо да намират върховете от куеритата.

Първото наблюдение, което трябва да направят състезателите е, че задачата е графова, а не геометрична, тъй като можем да ходим с мишката в произволни посоки (а не само по права линия).

Следва да създадем графа, който представя задачата. Най-очевидното решение би било да представим всеки пиксел като връх и да можем да преминаваме от пиксел само в съседен пиксел. Решението е относително лесно – даваме цена за преминаване през връх (пиксел), който е част от ръб на прозорец цена 1, а на останалите – цена 0. Така за всяко куери просто пускаме някой от известните ни алгоритми за намиране на най-къс път (в случая модифицирано BFS е достатъчно), като така бихме могли да намерим отговора. За съжаление, с това представяне графът бързо става много голям, а куеритата – неефективни. Наистина, при 100000 прозореца дори след компресия на координатите графът би имал около 10,000,000,000 върха. Имайки предвид, че трябва да правим това за всяко куери, общата сложност за цялата задача ще е към 1,000,000,000,000,000 операции. Нот куул.

До сега не сме ползвали никое от специфичните за задачата неща. Например, че прозорците са "вложени" един в друг – тоест не се пресичат, освен ако единият не се съдържа напълно в другия. Така можем да представим всеки от *прозорците* като връх в графа. Нещо повече, тъй като целият екран съдържа всички прозорци, можем да считаме и него за един огромен прозорец, съдържащ в себе си всички останали. Два върха (прозореца) в този граф имат ребро между себе си само ако:

1) Единият е директно вложен в другия (с цена на реброто 1)

или

2) Двата се съдържат директно от един и същ друг прозорец (с цена на реброто 2)

Под "директно" съдържане или влагане разбираме, че прозорец B е по-малък от прозорец A и се съдържа в него, като не съществува прозорец C такъв, че C да се съдържа в A и B да се съдържа в C.

Наистина, сега отговарянето на куеритата става по-бързо. За всяко от тях първо трябва да намерим най-малкия прозорец, който съдържа началния пиксел, после най-малкия прозорец, който съдържа крайния пиксел и накрая да пуснем алгоритъм за най-къс път (отново модифицирано BFS е достатъчно) за да намерим най-късия път между тях. Тук броят на върховете е до 100,000, което помага за забързване на решението ни (спрямо предишното). Все пак, за всяко куери трябва да направим по приблизително толкова операции, демек за цялата задача се нуждаем от 10,000,000,000 операции (в най-лошия случай). Куулър, ама все още бавно.

Тук трябва да направим едно не много трудно, и все пак не тривиално наблюдение. Ако в построения граф запазим само ребрата между директно вложени прозорци (тоест ребра от тип 1), полученият граф ще е дърво.
Наистина, тъй като всяки прозорец има точно един предшественик (освен екрана, който няма такъв), то броят ребра е N (при N+1 върха, тъй като считаме и екрана за връх).
Сега вече, за да намерим разстоянието между два пиксела (след като знаем кои са най-малките прозорци, които ги съдържат), трябва просто да намерим разстоянието в дървото (забележете, че премахнахме ребрата с цена 2, следователно минималното разстояние наистина е броя на ребрата в дървото). За това има сравнително стандартни алгоритми, които добрите състезатели в А група трябва да знаят. Макар и да има известни оптимални (линеен препроцесинг и константно куери) алгоритми за offline намиране на LCA, тук можем да ползваме и по-бавни такива – като например логаритмичния алгоритъм за намиране на най-близък общ предшественик (Lowest Common Ancestor, или LCA). Преизчислявайки "височината" на всеки връх в дървото, така можем за O(logN) време да отговаряме какво е най-късото разстояние между два върха в дърво, което е и достатъчно да се справим с всички Q въпроса със сложност O(Q*logN) за цялата задача.

Все още обаче не сме напълно готови. Трябва да отговорим на два доста важни въпроса:

1) Как строим дървото?
2) Как намираме най-малките прозорци (върховете в дървото), съдържащи пикселите от всяко куери?

И двата въпроса ще решим по един и същ начин – sweep line.
Първо сортираме точките (по X, а при равен X по Y) на правоъгълниците, както и на куеритата (като пазим всяка каква е – дали начало или край на правоъгълник, или точка от куери). След това почваме да ги обхождаме в този ред, като пазим STL-ски map с двойки (y-coordinate, info). Така при стигане до точка от куери ще можем да питаме в map-а коя е най-голямата координата, която е по-малка или равна на y-ка на точката чрез upper_bound (всъщност върха *преди* upper bound-а). Value-то на стойността в map-а може да ни задава именно индекса на най-малкия правоъгълник, който я съдържа. При добавяне на нов правоъгълник пък можем да видим кой го "съдържа" и да добавим ребро за дървото.

Горното нещо с имплементационни детайли и нагледен пример:
Преди да започнем да обработваме точките, за да избегнем частни случаи слагаме фиктивен прозорец, отговарящ на екрана, с някакви достатъчно големи координати да покрива цялото поле от валидни стойности. Така примерно в началото в мап-а имаме само едно ентри, {(-INF, 0)}, отбелязващо, че в y = -INF почва прозорец с индекс 0 (екрана). По някое време добавяме прозорец с y-span [13, 42]. В мап-а се случват следните неща: {(-INF, 0), (13, 1), (43, 0)} (разделихме досегашния прозорец на два нови, и добавихме между тях новия). Добавяйки друг прозорец с y-span [666, 1337]

получаваме {(-INF, 0), (13, 1), (43, 0), (666, 2), (1338, 0)}. Още един прозорец в [1000, 1234] би довел до {(-INF, 0), (13, 1), (43, 0), (666, 2), (1000, 3), (1235, 2), (1338, 0)}.

Така за всеки правоъгълник вкарваме най-много две точки в мап-а, всяка за лог време. Изкарването е същото. Накрая имаме:

- O(N*logN) за сортиране на точките.
- O(Q*logQ) за сортиране на куеритата.
- O(N*logN + Q*logN) за sweep line-а.
- O(Q*logN) за LCA-тата.

Или общо O(N*logN + Q*logQ + Q*logN).

Автор: Александър Георгиев