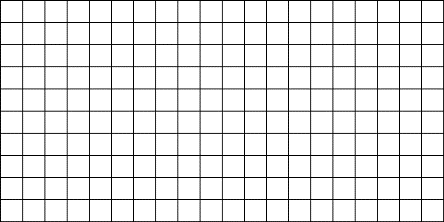
**АНАЛИЗ**

Да се представим как изглежда оставащата фигура след изпълнението на всяка заявка.



Отляво, отдясно и отдолу тя е ограничена от страните (или от части от тях) на първоначалния правоъгълник. Отгоре е ограничена от начупена линия, чийто долни върхове са някои от точките на заявките за рязане, а горните се получават като пресечни точки на прекарваните отсечки. Долните върхове винаги са с целочислени координати, а горните могат да бъдат както целочислени, така и от вида *а*.5, когато две отсечки се пресичат в средата на някое от квадратчетата. За да работим само с цели числа ще „раздробим“ мрежата на два пъти по-малки квадратчета (т.е. ще удвоим *N* и *M*) и ще си представяме, че всяка заявка за рязане се задава с точка с четни координати, т.е. при получаване на заявка (*x*,*y*) ще умножаваме по 2 координатите ѝ.

Най-простото решение е да поддържаме в масив (нека бъде *h*) височината на оставащата (неизрязана) фигура за всяко цяло *x*  между 0 и *N.* При получаване на нова заявка, определяна от точка с координати (*x*,*y*) можем да постъпваме по следния начин:

Тръгваме по оста *x* наляво от *x-*координатата на заявката и се движим през 1 по оста *x* до координатното начало или докато височината на неизрязаната част от фигурата не достигне *M*, като на всяка стъпка увеличаваме с 1 *y-*координатата на отсечката, която прекарваме под ъгъл 135° от точката, задаваща заявката за рязане. Ако тази координата е по-голяма или равна на стойността на масива *h* за тази стойност на *x* (т.е. ако отсечката на новата заявка минава над горната граница на текущата фигура), то за това *x* няма да има рязане на фигурата и височината ѝ остава непроменена. В противен случай височината на фигурата след изрязването ще стане равна на *y* координатата на отсечката в тази точка.

По същия начин се придвижваме и надясно (под ъгъл 45°) от точката, определяща заявката за изрязване до *x=N* или докато височината на неизрязаната част от фигурата не достигне *M.*

След като сме изчислили височините на новата начупена линия, която определя фигурата отгоре, изчисляваме лицето, като сумираме лицата на всички трапци между всеки две съседни точки от новата начупена линия, която определя фигурата отгоре (с разлика 1 в *x*-координатите им). Да не забравяме, че работим с с удвоени координати и поради това ще получим истинското лице, умножено по 4. Тук е моментът да пресметнем истинското лице в променлива от тип float или double, като разделим полученото при сумирането лице на 4.

Това решение е със сложност *O(N\*Q)*, реализирано е във файл **cutting\_stupid.cpp** и ще получи 20 точки.

Това решение може да бъде оптимизирано на две стъпки. Първата стъпка е да съобразим, че ако отсечката веднъж излезе над горната граница на текущата фигура, няма смисъл да продължаваме по-нататък наляво или надясно. Тази оптимизация няма да ни помогне особено, ако продължаваме след това да пресмятаме всеки път лицето, минавайки по всички стойности на *x* от 0 до *N.* Втората стъпка е да пресмятаме изменението на лицето само между левия и десния край, до които сме стигнали при движението наляво или надясно. За целта трябва да пазим в масив лицата на всички всички трапци с разлика 1 по *x*-координатите им и, движейки се наляво или надясно да обновяваме този масив, като заедно с това преизчисляваме лицето. Това решение отново е със сложност *O(N\*Q)*, но средно константта му е по-добра и то ще получи около 40 точки. Реализирано е във файл **cutting\_stupid\_opt.cpp.**

**Решения със сложност *O(Q\*logQ)***

Добрата идея за решаване на задачата е да се откажем от обхождането по всички *x*-координати, а да пазим актуалните заявки (само долните върхове на горната граница на текущата фигура) или, което е малко по-лошо, но също е достатъчно за 100 т., всички върхове на горната граница на текущата фигура в структура, която позволява бързо търсене, добавяне и премахване.

Нека си представим, че сме изпълнили вече *k* заявки и знаем, че текущото лице е *S*. Идва текуща заявка (*x,y*). Нека сега се движим по сортирани *x*-координатите на заявките, като започваме от тази, разположена непосредствено преди (или точно) на *x*, и се движим наляво. Нека текущата заявка *P*, до която сме стигнали, има координати (*Px,Py*). Ако (*x-Px+y*) е по-голямо или равно на *Py*, то можем да забележим, че всяка заявка с *x*-координата по-наляво от *Px* няма да изреже нищо и можем да спрем да ги обхождаме. В противен случай заявката *P* се оказва, че е изрязала нещо, което новата ни заявка, сега, така или иначе ще изреже. Затова можем да я премахнем и да забравим, че я е имало. След като сме премахнали излишните заявки, забелязваме, че частта, в която има новоизрязано се намира между текущата заявка и нейната нова съседна лява такава (нека я означим с *L*).

Има няколко възможности как да изглежда останалата площ в този участък:

m

m

Тези фигури се състоят от два завъртяни трапеца и един правоъгълник (някои с лице 0 понякога). Затова можем, когато махаме заявка, да вадим и лицето на горния вид фигури, с край нея заявка, и накрая остава да изчислим новото лице на фигурата между *L* и текущата заявка и да го добавим към S.

Същите разсъждения правим и за изрязаната част отдясно на текущата заявка. В кода е реализирано това решение, като поддържа 8\*лицето ( за по-лесни изчисления и за да бъде цяло число, въпреки че и x4 е достатъчно) и накрая се извежда резултатът разделен на 8. *X*-координатите се поддържат в структура от тип *map*.

Сложност: Всяка една заявка бива добавена точно веднъж и премахната максимум веднъж и сложността идва основно от операциите на map. Затова сложността е *O(QlogQ)*. Това решение е реализирано във файл **cutting.cpp** и получава 100 точки.

Подобно по идея решение е реализирано във файл **cutting\_fastS.cpp**, но в него в структура от тип *set* се поддържа множеството от върхове на начупената линия, ограничаваща фигурата отгоре. За целта между точките в равнината е въведено отношение „<“, като (*x1,y1*) < (*x2,y2*) ⬄ (*x1<x2*) || ((*x1==x2*) && (*y1<y2*)). Аналогично, при нова заявка, се прави обхождане на върховете на съществуващата начупена линия наляво и надясно, като се премахват ония върхове, които ще бъдат изрязани от новата заявка. Успоредно с това се пресмята и изменението на лицето. Такова решение също има сложност *O(QlogQ)* и получава 100 точки, въпреки че е малко по-бавно от първото изложено „бързо“ решение.

Алтернативно може да се използва и сегментно дърво. Когато ***N*** е по-малко става директно, като всеки връх пази всъщност какво е лицето на фигурата в него, стойността на ординатата си в най-лявата и си най-дясна точка. Така с една заявка можем да намерим къде се пресича новото рязане със старата начупена линия, а можем да използваме lazy propagation за ъпдейта, че даден интервал се покрива от нова отсечка. При големи ***N***, можем да правим същото, като компресираме координатите *х*, но технически малко се усложнява работата ни и се трупа допълнително време. Поради повечето операции този подход се оказва по-бавен, въпреки че не използва тежка структура като *set* и се очаква да вземе около 88 точки.

Ако пресмятането на лицето не се извършва успоредно с обхождането на върховете на начупената линия, а всеки път се пресмята изцяло отново, минавайки по всички върхове на начупената линия, то се получава решение със сложност *O(Q2logQ),* което ще получи 40 точки. Такова решение е реализирано във файл **cutting\_slowS.cpp**.

*Автори: Руско Шиков, Румен Михов*