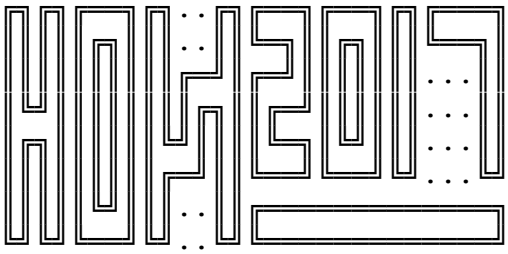
АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ВЛАКЧЕТА НА УЖАСИТЕ

Оригинално изходът на задачата ползваше символите '╫', '═', '╕', '╝', '╙', и '╒'. Така решението на третия примерен вход беше:



Честито, на тези от вас, които са го забелязали! А сега да пристъпим към решението...

Първото наблюдение, което трябваше да направите, беше, че всяка релса (независимо дали прав участък или завой) може да бъде поставена по два начина. Това автоматично трябва да ви бие камбанка в главата – за всяка непразна "клетка" от дъската трябва да определим едно от две състояния (да кажем true или false). Нещо повече, ако имаме на един ред 'S' и отдясно до него 'D', ако поставим правата релса вертикално, това автоматично означава, че завоят на другата ще е надясно. Ако пък я поставим хоризонтално, автоматично означава, че завоят на другата ще е наляво. Така можем за всяка двойка съседни сегменти релси да правим заключения от типа на "ако релса X е завъртяна в <дадена_посока>, то релса Y е завъртяна в <друга_дадена_посока>". Тъй като възможните завъртания са точно две, това е еквивалентно да кажем "Ако X е (true/false), то Y е (true/false)." Това напомня ли ви на нещо?

Ако не, това е знак, че трябва да погледнете какво е 2-SAT. Реално, тази задача може перфектно да бъде дефинирана чрез булев израз, където всяка двойка съседни сегменти от релса добавя по една дизюнкция на две променливи (или техните отрицания) в израза. Вземайки всички такива двойки, искаме да дадем такива стойности на променливите, че всички едновременно да са изпълнени. 2-SAT решава точно това.

При имплементацията има няколко детайла (например как можем по лесен начин да фиксираме стойността на някоя променлива, ако знаем, че може да е само true или само false? Друг изключително важен детайл е как се справяме с цялата логика "ако имаме еди какъв си знак отляво/отгоре/отдясно/отдолу на друг знак, то каква импликация добавя това в булевия израз? Ами ако е точка? Ако е на ръба на дъската? Имплементирайки тези неща пишейки if след if със сигурност ще доведе до грешки, които могат да ви отнемат много време.

Има относително елегантен начин това да стане, ползвайки два допълнителни масива – единият, указващ накъде "гледа" всеки от типовете релси, в зависимост дали е true или false, а другият казва "ако гледайки наляво/нагоре/надясно/надолу виждам символ 'X' (където X е някой от типовете релси), то X трябва да с еди-каква-си-стойност (true, false, която и да е от двете, или никоя от двете). Вижте авторското решение за относително чиста имплементация. Тъй като имаме N * M клетки, всяка от които е променлива в 2-SAT инстанцията, и понеже решението на 2-SAT проблема е линейно, то сложността на цялото решение на задачата става O(N * M).

Едно от интересните неща, които открих, докато се опитвах да направя грешно решение на задачата, е, че ако има решение, то то е само едно. Така изискването за най-дълъг цикъл или за "което и да е решение" е само за заблуда – реално решението е само едно! Това не е твърде

интуитивно, но не е и някакво "уау!" наблюдение. Нека видим как бихме могли да се досетим за това!

Ще ползваме твърдението, че винаги съществува поне един (нефиксиран) релсов сегмент, който може да бъде насочена в най-много една посока. Възможно е да има сегмент, който не може да бъде насочен наникъде (както е горната дясна клетка във втория примерер в условието) – в този случай веднага знаем, че няма решение.

Нека видим защо винаги има поне една (все още нефиксирана) клетка, която има максимум един вариант за насочване. Започвайки от най-горния ред и движейки се по колони, нека намерим първата непразна клетка. Тя ще има празна клетка или стена както отляво, така и отгоре. Това означава, че тя трябва да е ъгъл, при това да гледа надолу и надясно и можем директно да я фиксираме. Продължавайки нататък ред по ред, колона по колона, всяка непразна клетка, която срещаме, ще има за съседи отгоре и отляво някое от:

1) Ръб на дъската
2) Празна клетка
3) Вече насочена клетка, която не гледа към нея
4) Вече насочена клетка, която гледа към нея

Варианти 1), 2), и 3) ни задължават текущата клетка да не гледа в тази посока, докато вариант 4) ни задължава да гледа в тази посока. Независимо коя комбинация от горните варианти имаме "наляво" и "нагоре" от текущата клетка, обаче, те еднозначно я определят.

Така можем да ползваме значително по-просто решение – ще обхождаме дъската клетка по клетка и ще фиксираме позициите на всяка непразна такава, която срещнем. Тя потенциално ще позволи фиксирането на други клетки и т.н., докато фиксираме цялата дъска. Ако по някое време стигнем до клетка, която не можем да насочим в предопределената ѝ позиция можем директно да твърдим, че няма решение. Наистина, няма как да сме направили "грешна" стъпка, тъй като на всеки ход фиксираме релса, която няма по какъв друг начин да бъде насочена.

Тъй като всяка клетка насочваме само по веднъж, а разглеждането на съседите ѝ е константна операция, това решение също е със сложност O(N * M), но с по-малка константа и на практика работи по-бързо. Разбира се, и тук трябва да се обърне внимание на чистата имплементация (тъй като и това решение може доста да се забатачи), макар и тук да е малко по-лесен за решаване.

Автор: Александър Георгиев