АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
RotatingMaze

Задачата RotatingMaze не е особено трудна откъм решение, но все пак не очаквам голям брой хора да я решат за 100 точки (максимум 2-3?), тъй като съчетава вероятности (които са относително непознат материал за повечето ученици) и един не особено известен (по Български състезания) трик.

Първо, опитните състезатели в А група веднага трябва да са надушили, че трябва да приложат някакъв вид динамично оптимиране. Самият стейт беше почти очевиден – с ограничения до 1000 по 1000 и с наблюдението, че очакваното време от дадена клетка не зависи от това как сме дошли до нея – веднага можем да преценим, че динамичната таблица ще е двумерна и ще задава просто в коя клетка сме. Имаше един проблем, обаче – стейтът можеше да бъде цикличен (възможно е от клетка да отидем в същата клетка).

Преди да видим възможните решения, ще кажем какви наблюдения трябваше да е направят преди да се атакува задачата. Този път то беше само едно, при това относително очевидно: реално няма значение дали дадена клетка е '-' или '|', тъй като всеки ход може да се завърти, а може и да не се с равен шанс. Тоест независимо дали клетката, в която се намираме първоначално е била '-' или '|', то в момента, в който искаме да се придвижим на или от нея, тя може да е с равна вероятност което и да е от двете. Това ни разрешава да разглеждаме дъската като "бинарна" – всяка клетка е или кръстовище ('+') или не ('-' или '|').

За тази "бинарна" дъска трябва да намерим каква е вероятността да се преместим от дадена клетка на тази надясно или надолу от нея, в зависимост коя от трите каква е. Тъй като всяка може да е или кръстовище или не, то имаме 8 варианта: текущата да е '+' или не, тази отдясно да е '+' или не и тази отдолу дали е '+' или не. Една възможност беше да се дефинира масив с тези шансове (double числа), като се попълни с кратка функция. Тъй като шансовте бяха лесни за изчисление дори на ръка (а даже и наум!), друга алтернатива беше просто да си ги сметнат и да ги запишат като числа в кода си (както направих аз). Забележете, че трябва да се справим отделно със случаите, в които сме в краен ред или колона отделно – при тях няма възможност и двата варианта да са възможни, затова и шансовете са различни.

Сега си представете, че нямаше възможност Ели да остане на едно място (тоест винаги се движи надолу или надясно). Така с динамична таблица dyn[1000][1000] бихме могли да намерим очакваното време с проста рекурсия, на която подаваме само на кой ред и коя колона сме. Това, обаче, би било по-скоро задача за В група (а даже за С, ако нямаше вероятности).

Тъй като има тази възможност, обаче, това няма да работи в този вид, тъй като бихме се забили в безкрайна рекурсия, която от една клетка отива в себе си. Демек графът на стейтовете е цикличен.

Забележете, обаче, че шансът от дадена клетка да отидем в себе си никога не е 100%. В най-най-лошия случай сме в последния ред или последната колона и текущата и единствената възможна клетка не са кръстовища. Тогава шансът да останем в същата клетка е 75% (трябва и двете клетки да станат вертикални, ако сме в последната колона, или и двете клетки да станат хоризонтални, ако сме в последния ред).

Сега си представете, че си седим в същата клетка безкрайно дълго. Какъв е шансът за това? Ами – астрономически малък. Реално шансът 10 поредни пъти мостовете да се завъртят "по

кофти начин" и Ели да си стои на все същата клетка е 0.75^10 = 5.6%. Шансът да остане 100 поредни пъти на същата клетка е 0.00000000003%.

Защо, тогава, не вкараме още един параметър в динамичното – колко минути сме стояли в текущата клетка. Когато този брой стане голям, вече шансът е толкова малък, че не влияе на глобалния отговор. Вместо да продължим да се лутаме до безкрайност, можем да върнем някакво приближение (например оставащия брой редове и колони, които трябва да изминем). Тъй като то ще се умножи по някакво много малко число, реално "неточността" от приближението изобщо няма да се забележи.

Хубав въпрос е кога този шанс става толкова малък, че да не влияе на отговора (тоест колко минути трябва да седим в текущата клетка преди да се откажем)? Всъщност състезателите лесно можеха да експериментират – да си направят максимален тест и да видят какво се случва с 50, 100, 150 и т.н. Лесно може да се види кога отговорът спира да се променя. (Не е хубаво да изчисляваме тази стойност на базата на най-лошия случай, тъй като в огромен процент от клетките той няма как да се случи. Например ако сме в клетка, която не е в краен ред или колона, най-лошият случай е да имаме три некръстовища, в който случай шансът да останем на мястото си е 0.5, вместо 0.75. А тези клетки са мнозинството от дъската.)

Оказва се, че около 100 е хубава стойност, която балансира ползваната памет и точност (разликите с истинския отговор са чак на 9-тата цифра след десетичната точка). Така динамичната таблица трябва да бъде [1000][1000][100]… което е твърде много за дадения Memory Limit (32MB). С тази хитрина може да е хванат тестове с N и M до около 200 – което бяха около 70 точки.

Сега да видим какво е истинското решение. То, всъщност, е по-лесно от горното! Само че ви трябва един трик, който, макар и не особено сложен, не е очевиден.

Очакваното време за достигане до грая на лабиринта от дадена клетка се формира от три неща:

1. Шансът да отидем надясно * (отговора за надясно + 1)
2. Шансът да отидем надолу * (отговора за надолу + 1)
3. Шансът да останем на същото място * (отговора за текущата клетка + 1)

Навсякъде имаме ...+1, тъй като ни отнема една минута да се преместим от текущата клетка в някоя от съседните (или да установим, че не можем, в който случай оставаме в същата). Тъй като участва във всичките три варианта реално можем просто да го добавим отделно.

Забележете, че за 1 и 2 можем да викнем рекурсията директно, тъй като редът или колоната се променят (не оставаме в същата клетка). Така имаме проблем само с 3, където отговорът за текущата клетка зависи от себе си.

Това, обаче, е уравнение от първа степен с едно неизвестно. 5-ти клас, anyone? Нека отговорът за текущата клетка е X. Тогава имаме:

- X = 1 + chanceDown * recurse(row + 1, col) + chanceRight * recurse(row, col + 1) + (1.0 – chanceDown – chanceRight) * X
- X = 1 + cD * recD + cR * recR + X – cD * X – cR * X
- X * (cD + cR) = 1 + cD * recD + cR * recR
- X = (1 + cD * recD + cR * recR) / (cD + cR)

Ии готово - това е! Трябва ни единствено стандартното динамично, в което винаги се движим надясно или надолу, да сметнем отговора за тези два случая и да заместим в горното уравнение!

Така решението ни ползва едва около 9 мегабайта памет и работи почти моментално – неговата сложност е O(N * M) както по време, така и по памет.

Току-що се запознахте с така наречените "циклични динамични". Това е най-простият вариант, в който от стейт можем да се върнем в себе си с най-много един преход. В някои по-завъртяни динамични от този тип може да се наложи да трябва да гледате до няколко прехода напред, преди да се върнете в себе си. Те се решават по сходен, макар и малко по-сложен за имплементация начин.

П.С.: Малките тестове бяха дадени за Монте-Карло симулация, която би хванала около 20 точки, но като че ли беше по-сложна от варианта с "разширеното" динамично, затова няма да разглеждаме (и не вярвам някой да е написал).

Автор: Александър Георгиев