Анализ на задача Коридор

Първата стъпка към решението на задачата е наблюдението, че използвайки и двата вида движение (наляво и надясно) и един брояч, можем на всяка стъпка да знаем в коя клетка (релативно към началната) се намираме – при всяко движение надясно, увеличаваме брояча, а при всяко движение наляво го намаляме.

Ключовото наблюдение е: Нека знаем, че дължината на коридора не е по-малка от **K**, и направим следната поредица от стъпки – **K** стъпки надясно, промяна на лампата (в клетка **K**) и **K** стъпки наляво (връщайки се в оригиналната клетка). Ако в този момент лампата в клетката, от която сме започнали тази поредица стъпки, се е променила (спрямо момента на започване на поредицата стъпки), то дължината на коридора е точно **K**.

Използвайки това наблюдение, можем първо да проверим за дължина 1, после за дължина 2, дължина 3, и т.н. докато не стигнем до отговора. Това решение прави точно **(N+1)^2** стъпки, не хаби твърде много памет (достатъчни са му три променливи, които могат да се компресират в около 15-тина бита) и е достатъчно за първата подзадача. (50 точки)

Друго наблюдение, което е нужно за решаване на другите подзадачи е: Нека разполагаме с достатъчно памет за съхраняване на целия коридор. В началото правим **MAX_N** (очевидно за **MAX_N**=200, това няма да работи, защото имаме 160 бита памет) на брой стъпки надясно, по време на които записваме стойностите на всички лампи в паметта. След това променяме стойността на лампата в клектата, до която сме стигнали, и почваме да вървим наляво, докато не стигнем до лампа, чиято стойност се е променила спрямо това, което имаме записано в паметта. В този момент, броят стъпки които сме направили в движението си наляво е броят клетки на коридора. Всичко това се базира на факта, че правейки **MAX_N** на брой стъпки в една посока, със сигурност последната клетка, до която стигаме, ще бъде посетена поне 2 пъти.

Само използвайки горното наблюдение, можем да решим задачата за **MAX_N + N + 1** стъпки, при условие че имаме поне **MAX_N** бита памет. Това се случва във втората подзадача. (20 точки)

На този етап можем да комбинираме двете решения – проверяваме дали **N** е от 1 до 50, използвайки второто решение (с **MAX_N**=50), а ако не е – намираме го чрез първото (почвайки да търсим от 51 нагоре). (70 точки)

Освен паметта за коридора при второто решение, то има нужда от памет за единствена променлива по време на изпълнението си – брояч за това в коя клетка се намира. Поради техническата имплементация (многократно викане на функция), то има нужда да знае и в кое състояние на изпълнението си е (ходене наляво, ходене надясно...), тоест общо две променливи. Дори когато го комбинираме с първото решение, броят различни състояния е доста малък (6 са достатъчни), което се побира в 3 бита. Тоест можем да използваме и 29-те свободни бита от промелнивата за състояние за пазене на част от коридора – общо стават 3*32 + 29 = 125 бита. По този начин вече използваме линейното решение за проверка на първите 125 възможности, смъквайки броя стъпки, и се вместваме в 3-тата подзадача. (90 точки)

Променливата, в която пзаим коя е текущата клетка, също има сравнително тесни ограничения – максималната и стойност е 200, което се събира в 8 бита. Останалите 24 можем също да

използваме за пазене на част от коридора – общо стават 125 + 24 = 149 бита. Така смъкваме броя стъпки за решаване на задачата достатъчно, че да се вместим и в 4-тата подзадача. (100 точки)

Една от целите при формулиране на задачата беше състезателите да не могат да изкарат пълен брой точки, използвайки само втората идея (линейното решение). Затова трябваше от една страна **N** да е достатъчно малко, че квадратното решение да работи в разумно време, а от друга – по някакъв начин да се ограничи допустимата памет до нещо от порядъка на **N/8** бита (на практика невъзможно, използвайки външни инструменти). Това е причината за доста тежката техническа имплементация на задачата – в рамките на един тест се симулират няколко коридора, и то симулациите са преплетени една с друга – с цел всякаква глобална памет да бъде неизползваема. За да е възможно да се прави каквото и да е решение в такава среда, беше осигурено всяка симулация да има достъп до нейна локална памет (менажирана от grader-а, достъпвана чрез функциите get_memory и set_memory). Интересно е, че дори след всичко това, все още беше възможно (и то сравнително просто) решението да може да прецени в коя симулация се намира, и на база на това да използва глобална памет, заделена единствено за нея. Това е причината в рамките на един тест, всеки от коридорите да бъде симулиран по няколко пъти и след това да се проверява дали решението работи по еднакъв начин в ряазличните симулации на еднаквите коридори. Почти съм убеден, че този подход не може да бъде разбит по смислен начин в тази задача (ако някой успее, може да сподели), но дори и да може – съмнявам се да е достатъчно просто, че да е приложимо на състезание.

Автор: Иван Тончев

Детайли по реализацията: Илиян Йорданов