# Анализ

**Тривиално решение за 0 т.**

Можем винаги да питаме за всяка една алея в парка и това ще отнеме $4N$ питания ($\pm$ константа), това решение е твърде тривиално и затова не са предвидени точки за него.

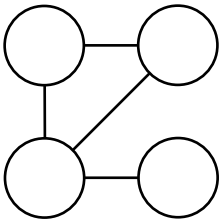
**Подзадача 1 за 15 т.**

Първата подзадача е предвидена в случай, че не успеем да направим всички наблюдения нужни за втората, но сме направили някое, което ни стига да сведем броя заявки надолу. Затова и няма предвидено конкретно решение за нея.

**Подзадача 2 за 45 т.**

Решението на втората задача се базира само на разписване на възможните посоки на алите и след това да се пита за всички неизвестни. Всъщност, се оказва, че страничните алеи (тези между места с разлика две) винаги имат положителна посока (доказателството за това ще бъде описано по-долу). След това питаме по една заявка за алея от тези, които не знаем – това са $2N$ питания.

Метода, който ще използваме за откриване на този факт, е следният. Ще разглеждаме алеите на по поредни четворки. Четири такива поредни алеи са всички алеи без една между четири поредни места. Следващата четворка е отново всички алеи без една за отново четири поредни места, но изместени с два



напред. Разглежданата структура е от типа показан на фигурата.

Очевидно, ако не спазваме никои ограничения, можем да имаме $2^4 = 16$ комбинации от посоките на тези четири алеи. След това можем да елимираме тези с цикли в тях и остават $_{14}$ комбинации. След това от там можем да разгледаме, кои са възможни за първата четворка, това са: 1111, 1110, 1101, но не и 1100 (отбелязани са посоките на алеите подред). 1100 не е разрешено, защото тогава няма да има алеи излизащи от място 2. Можем след това да разлгедаме за всяка от тези конфигурации кои са възможни след нея, и от всяка от тях и т.н. Ще видим, че не всички кофигурации са достижими. Можем да повторим същия процес започвайки от последната четворка, за която имаме други възможноти, и да видим кои могат да достигнат до някоя от тях. Така ще елиминираме и други. В крайна сметка остават само три конфигурации: 1111, 1101 и 0111. Това значи, че в нито една разрешена конфигурация няма външни алеи с отрицателна посока. Цялото разписване на този процес не е включено в анализа, защото то е тривиално след като знаем метода.

**Подзадача 3 за 70 т.**

В решението на подзадача 2, все още въобще не се въползваме от това че не всички възможности за вътрешните алеи са разрешени. Вече откритите ориентации на външните алеи изпълняват правилото за поне една влизаща и поне една излизаща алея от всяко място (освен входа и изхода), но правилото за

никакви цикли може да бъде нарушено. (От тук нататък ще разглеждаме само вътрешните алеи.) Всъщност, можем да забележим, че ако посоките на две поредни (вътрешни) алеи са отрицателни, винаги ще се получи цикъл. Това значи че всъщност сведохме задачата до следното:

Имаме низ от $2N$ нули и единици без поредни нули. Да се открие низа чрез заявки от вида, който ни интересува. Можем да видим, че има точно $F_{2N+2}$ такива низа, където $F_k$ е $k$-тото число на фибоначи. Това се дължи на факта, че броя низове с дължина $k$, които завършват на $1$ е равен на броя низове с дължина $k-1$, а на тези които завършват с $10$ е равен на броя низове с дължина $k-2$. Всяка заявка в най-добрия случай елиминира половината от всички възможности. Т.е. броя нужни заявки за голямо $N$ е приблизително $\log_2 F_{2N} \approx \log_2 \phi^{2N} = 2N \log_2 \phi \approx 1.388N$.

Пробелмът е, че няма как бързо да намерим най-добрите заявки. Затова можем да използваме съботимален метод и да разделим низа на някакви равни интервали, и всеки интервал да открием с минимален броя запитвания. Това не е оптимално, защото приема, че всички комбинации от поредни интервали са възможни, а това не е така, защото може да се срещат с нули в двата си края. Но всъщност решнията и на третатата и на четвъртата подзадача се базират точно на този метод.

Можем да си напишем програма която смята колко запитвания ни трябват за дадена дължина, като първо намира броя низове с тази дължина, взема $\log_2$ от това и закръгля този резултат нагоре. Можем да видим, че първите две дължини, стрикнто по-добри от по-малките от тях, са 4 и 11. За дължина 4 са ни нужни по 3 запитвания, а за 11 – по 8. Имаме $2N$ елемента в низа и така получаваме ограниченията за трети и четвъртия събтаск. За тези константи можем да се досетим и от оценяването на задачата.

За решаване на третия събтаск е достатъчно на ръка да разпишем 8те опции и да открием какви запитвания да питаме:

1111
1110
1101
1011
0111
1010
0101
0110

Ако първо питаме за XOR на четирите алеи и после за XOR на първата и последната, получаваме следните четири групи от по две конфигурации:

| Първи XOR<br>Втори XOR | 0 | 1 |
|---|---|---|
| 0 | 1111<br>0110 | 1101<br>1011 |
| 1 | 1010<br>0101 | 1110<br>0111 |

Лесно можем с третото запитване да разграничим между двете останли конфигурации. С това завършва и решението на третата подзадача.

**Подзадача 4 за 100 т.**

Четвъртата подзадача не изисква откриването на нови математични идеи. Фокуса при нея е откриване на нужното дърво от запитвания за низ с дължина 11. Докато при дължина 4, лесно можеем да направим това на ръка, при 11 това далеч не е така. Възможни са няколко подхода: precompute в началото на решението (time limit-ът е максималко свободен, за да позволява това), precompute на локална среда и след това hard code-ване и смесица между тези две (например hard code-ване на първите няколко запитвания, които са най-бавни за откриване). В случая когато нещо се прави на локална среда е възможно и да се прави не изцяло автоматизиран метод, а да пробваме да измислим първите въпроси и програмата ни да провери дали може да се попълни цялото дърво със запитвания.

Авторовото решние извършва precompute-а в началото на изпълнението на програмата. За целта е използван brute force, с разни оптимизации:

1. Когато сме в ситуация в която $2^Q > T$, където $Q$ е броя оставащи въпроси, а $T$ броя възможни конфигурации, прекъсваме изпълнение на този клон на brute force-а.
2. Когато остават по-малко на броя запитвания (по-ниско в дървото сме) се пробват по-прости заявки, например само с две или три алеи.
3. В по високите нива на дървото, когато нашите избори силно влияят колко трудни за откриване са подходящи въпроси в по ниските нива, използваме по силна версия на 1. Т.е. не гледаме просто дали е теоретично възможно да открием подходящи следващи запитвания, ами избираме такъв въпрос, че възможностите с XOR 0 и тези с XOR 1 да са максимално близо по брой. Константата, която вършеше най-добра работа беше $0.55$ по броя на текущите възможности. При по-малко от това и става твърде ограничаваща и при повече от това става твърде свободна.
4. Когато остава само една заявка и две възможности, използваме по-бърз метод (около двойно по-бърз) да открием нужния въпрос.

Не всички от тези оптимизации (особено не и последната) са нужни за влизане в time limit-а и най-вероятно са възможни и всякакви други. В приложеното авторово решение може да намерите имплементация на този brute force с тези оптимизации. Важно е да се отбележи, че можем непренебрежимо да оптимизираме авторовото решение и ако премахнем обособяваниятана парчета код в отделни функции (особено на критични места), тъй като викането им отнема време. Това обаче не е направено в авторовото решение към този анализ с цел то да е по лесно за преследяване.

*Автор: Емил Инджев*