Kites
(Решение)

Задачата *Kites* имаше няколко възможни решения, носещи различен брой точки. Всъщност, тъй като входният пример беше относително малък, при това лесно може да се изкара верният резултат дори с грешно решение, беше полезно състезателите да напишат някое от "бавните" решения. Чрез тях те можеха да си генерират тестове (и съответстващите им верни отговори) с които да тестват "бързото" си решение. Допълнително, дори с бавно решение можеха да се хванат нелоши точки.

Опитните състезатели веднага трябва да са надушили Динамичното Оптимиране. "По колко начина" и отговор по модул неминуемо са хинтове за това. Въпросът е как точно да направим динамичното – това не е очевидно, което прави задачата трудна.

Най-простото решение би било да сортираме децата по C_i, след което стейтът ни да бъде (индекс_на_дете)(наредени_използвани_хвърчила). В случая, едно грешно решение би било да пазим само индексите на използваните хвърчила, а не и наредбата им. Това, обаче, е грешно, тъй като *има* значение кое дете държи кое от използваните хвърчила! Тоест, в случая не бихме могли да ползваме битови маски, а ще се наложи да държим индексите на използваните хвърчила във вектор.

Тъй като възможните стейтове в тази динамична таблица са ужасно много, ще ги държим в STL-ски map. Забележете, че всъщност индексът на детето е еднозначно определен от наредената множество на изполваните хвърчила – ако вече сме ползвали 5 хвърчила, то в момента сме на дете с индекс 5 (индексирано от 0). В случая, тъй като ползваме мап, това не ни помага, тъй като броят елементи в мап-а е един и същ. Оказва се, че ако ги пазим в различни индекси, дори е малко по-бързо, тъй като имаме повече на брой, но по-малки мапове, а търсенето в по-малък мап е по-бързо.

Разбира се, както повечето динамични, ползването на мап значително забавя нещата и това решение в най-простия си вид, макар и вярно, би хванало едва около 20 точки (що-годе колкото би хванало и грешното решение с битови маски). Все пак, ползвайки това решение можем да си генерираме тестове, с които да тестваме по-сложни такива.

Преди да видим умното решение, нека видим как бихме могли да оптимизираме това.

Оптимизация 1
Тази оптимизация е по-очевидната от двете, които ще покажем, при това носи повече точки. Ако в даден момент знаем, че има неизползвано хвърчило, което никое от оставащите деца не може да държи (тъй като въжето би се оплело с някое от тези, които вече сме фиксирали), то няма нужда да продължаваме търсенето и можем директно да върнем 0. С тази оптимизация бихме забързали решението си достатъчно, че да хванем цели 60 точки!

Оптимизация 2
Ако някакви две вече ползвани хвърчила няма как да влияят на отговора нататък (тоест никое от необработените деца не ги "вижда", и няма как те да попречат на ползването на хвърчило по-нататък), то може да не пазим реда им. Така, например, ако имаме наредено множество от ползвани хвърчила (5, 1, 3, 7, 2, 4), но знаем, че последното дете, държехки 4, пречи на всички останали деца да "виждат" 1 и 7 (тоест

отсечката от тях до 1 и до 7 се пресича с отсечката на последното дете до 4), то нас реално не ни интересува дали сме имали (5, 1, 3, 7, 2, 4) или (5, 7, 3, 1, 2, 4). Вместо това можем да пазим стейта като (5, -1, 3, -1, 2, 4). Така, добавяйки нови хвърчила, значително намаляме броя стейтове, съответно забързваме решението си. С тази оптимизация (сама по себе си) бихме хванали около 40 точки.

Комбинирайки двете оптимизации, успяваме да хванем цели 80 точки – това са всички "рандом" тестове, дори тези с N = 50.

Все пак съществуват и тестове, на които дори с тези оптимизации се стига до твърде много стейтове. Включил съм 5 такива теста (от 25 общо) – тоест хората, стигнали до най-доброто решение, не печелят твърде много, но все пак трябва да се чувстват горди, че са направили задачата за 100.

Истинското решение отново е базирано на динамично оптимиране, но на тотално различна идея от тази, която ползвахме за решението с мап. Трябва да направим наблюдението, че ако определим кое дете държи най-високото хвърчило (това с най-голям Y_i), останалите деца и хвърчила биват разделени на две независими части, представляващи независими подзадачи. Ако изчислим броя валидни комбинации във всяка от тези части, и умножим двете получени числа, ще получим и отговора за цялата задача.

Това, всъщност, си е чисто приложение на Разделяй и Владей, в динамична задача. Единствената нужда изобщо да имаме динамично е, че най-високото хвърчило може да бъде свързано с потенциално повече от едно дете, и все пак да образува валидно решение (тоест нямаме еднозначен сплит, както е при Разделяй и Владей). В следствие на това е възможно да се получат много еднакви подзадачи, за които ни трябва динамичното.

Така стейтът ни е определен от две неща: въжето на хвърчилото, което лимитира подзадачата отляво, както и въжето на хвърчилото, което я лимитира отдясно. Тъй като всяко въже се определя от дете и хвърчило, реално динамичната ни таблица ще е четиримерна: (ляво_дете)(ляво_хвърчило)(дясно_дете)(дясно_хвърчило). Ако сме сортирали децата по X предварително, всички "валидни" деца без хвърчила ще са в интервала (ляво_дете, дясно_дете). За хвърчилата това не е изпълнено, но можем просто да обиколим всички и да видим кои попадат в този "интервал" – тъй като на всяка стъпка избираме най-високото хвърчило, то хвърчила, които не попадат в интервала, ще се пресичат или с лявото или с дясното "ограничително" въже.

Така сложността на това решение е $O(N^5)$, тъй като имаме $O(N^4)$ стейта и $O(N)$ операции вътре в динамичното.

Автор: Александър Георгиев