Мумия – Анализ

Автор: Емил Инджев

1 Наивно $N!$ решение

Най-базовото решение на задачата е да пробваме всички пермутации, докато не познаем точно. Това изкарва 9 точки.

2 Hill climbing решение

Една друга наивна идея е просто да пробваме произволни размени и да ги запазваме, ако увеличават броя мачове, а иначе да ги връщаме. Такова решение изкарва 24 точки.

3 $O(N^2)$ решение

$O(N^2)$ решенията имат до някъде подобна идея както с hill climbing, но по-структурирано и нерандомизирано. Интуитивно идеята е да се движим по позиции от ляво надясно и за всяка да откриваме кой елемент принадлежи там. За текущата позиция, пробваме да я разменим с всеки по-десен елемент, докато броят мачове не се смени. При всяка смяна дедуцираме кой елемент сега вече е на правилната си позиция или кой е преди е бил, а сега не е. За целта при смяна само с 1, тестваме да разменим един от двата елемента с вече открит правилен елемент, а ако няма такъв, пробваме да разменим и двата с други произволни елементи, докато не видим резултат, който потвърждава кой от двата елемента е правилен. Забележете, че с такъв вид процедура често откриваме елементите за позиции не в префикса, който градим, така че е важно да запазваме и тях. Това решение изкарва 42 точки.

Разнообразни "грешки", които биха довели до повече операции и по-малко точки са неща като: да не брейкваме след като открием елемента за текущата позиция, да не запазваме вярно открити по-десни елементи, да правим по две куерита на стъпка, а не едно (важно е да сравняваме с броя мачове от миналата итерация, а не да питаме и преди, и след размяната), и други.

Код: `mummy_emil_n2.cpp`.

4 $N \log N$ решения

Има няколко разнообразни идеи, които водят до решения с $O(N \log N)$ заявки. Едно общо наблюдение обаче е следното: ако рандомизираме пермутацията, шансът да получим 0 мача е приблизително $1/e$ или около 37%. Това е доста интуитивно, защото средният брой мачове е 1, а рядко имаме повече от 2-3 мача. Полезното е, че при 0 мача имаме доста ясни заключения, които може да се ползват по-разнообразни начини.

4.1 Елиминиране на опции

Едно наивно решение е просто да питаме произволни пермутации и всеки път, когато получим 0, да елиминираме от всяка позиция текущия елемент там. Спираме, когато за всяка позиция са елиминирани всички елементи без един.

Можем да видим, че това е $O(N \log N)$, използвайки следния резултат: ако имаме X опции и на всяка стъпка задраскваме една произволна (потенциално вече задраскана) опция, ще ни отнеме $X \log X$ стъпки да задраскаме всички опции. Тук сумарно трябва да се случат $N(N-1)$ елиминации, като всеки път елиминираме

(приблизително) произволна опция. Това отнема средно $N(N-1)\log(N(N-1)) \approx 2N^2 \log N$ елиминации. Но на всяка пермутация с 0 мача елиминираме N опции (ще ги третираме като независими произволни като апроксимация), т.е. отнема около $2N\log N$ такива пермутации или около $2eN\log N \approx 5.4N\log N$ заявки.

Възможни са доста оптимизации. Можем, когато заключим за даден елемент на коя позиция е, да го елиминираме от всички други позиции. Също така, веднъж като открием нещо, е добре да го държим фиксирано във всички бъдещи заявки (и да търсим не 0 мача, а 0 мача на неизвестни елементи). Допълнително, можем да следим освен възможни елементи за дадена позиция, също и възможни позиции за даден елемент (и отново да елиминираме и тук при откриване на елемент-позиция). Също така, всеки път като открием нещо, някои стари заявки, които тогава са били безполезни, сега може да имат 0 мача на все още неизвестни елементи, така че е добре да се преразглеждат. Допълнително, вместо произволни пермутации, може да питаме пермутации, които са циклични шифтове една на друга (и след всички такива да рандомизираме), защото по този начин елиминираме по-разнообразни опции (по рядко елиминираме една опция много пъти).

Може да намерите две решения от този тип, като нито едното не имплементира всички споменати оптимизации, а различни сетове от тях: `mummy_emil_nlog-elim.cpp` (73 точки) и `mummy_encho_nlog-good.cpp` (86 точки).

4.2 Двоично търсене на мач

Друга идея е отново да започнем с откриване на пермутация с 0 мача. След това обаче искаме да открием друга "свързана" пермутация с 1 или повече мачове. Накрая искаме да направим двоично търсене, за да открием къде е новият мач. За да правим такова търсене, ни трябва новата пермутация да се формира от старата с някаква поредица размени. Различни опции са възможни, но най-добрата сякаш е да групираме всички N елементи на $N/2$ двойки по произволен начин, и да направим тези $N/2$ размени наведнъж. Проверяваме дали новата пермутация има мачове и ако да, продължаваме. Ако не, пробваме отново. (Възможно е и да започнем с пермутация с положителен брой мачове и да търсим такива размени, че да стигнем до 0, а по-оптимално е просто да започнем с коя да е пермутация и да прилагаме такива размени по двойки, докато не стигнем до другия вид пермутация).

След като вече имаме две пермутации, една с 0 мача и една с 1 или повече, правим байнъри сърч по списъка двойки (на дадена стъпка прилагаме префикс от размените), докато не открием първата размяна, която води до мач. След това просто използваме същите техники както в $O(N^2)$ решението, за да открием кой от двата елемента е на правиланата си позиция. Като оптимизация, можем ако сме имали над 1 мач, да продължим с байнъри сърч в останалия суфикс от двойки.

Важно е, след като открием даден елемент къде трябва да е, да го държим там във всички бъдещи пермутации, които пробваме (освен за целите за размяна при открит мач).

Можем да видим, че при всички такива решения правим $N\log N + O(N)$ заявки, т.е. има място за оптимизиране само на линейния член. Една оптимизация е да забележим, че след двоичното търсене (или няколко такива) и фиксиране на откритите елементи оставаме с пермутация с 0 мача на неизвестни елементи, та можем да продължим с нея, вместо наново да търсим пермутация с 0.

Такова решение с описаните оптимизации изкарва 96 точки. Код: `mummy_emil_nlog.cpp`.

4.3 Комбинация на двете техники

Възможно е да се комбинират разнообразни идеи от двата описани подхода. Най-наивна комбинация би било, докато правим второто решение, просто да следим и елиминации, ако случайно дедуцираме нещо така. Това не е особено мощно, защото се правят само $O(N)$ независими пермутации (голяма част от заявките са подобни една на друга), та рядко се случва цяла дедцукция. По-полезно би било, ако размените, които генерираме се възползват от списъка възможни елементи за позиция.

По-наивна версия на такава комбинация може да намерите в `mummy_emil_nlog-better.cpp`, което изкарва 97 точки.

По-добър вариант е да се поддържа за всяка позиция списък от останали-кандидати възможни стойности след предишните елиминации. По този начин самите размени е възможно да се генерират за позиции, така че размяната на стойностите им винаги да ни даде нова информация (и така винаги да откриваме мач, или да елиминираме за дадена позиция). Този подход може да откриете в `mummy_iliyan_nlog.cpp`, което е и решението за 100 точки. Също една важна оптимизация там е, когато се правят двоичните търсения за откриване на мачове на дадена итерация, да се запазват намерените отговори при заявките в двоичните, така

че да не се повтарят заявки в текущата итерация. Могат да се направят и подобрения, така че размените да се генерират с по-добри стратегии (например да започваме да правим размените от позициите с най-малко останали стойности, да рандомизираме реда на позициите при правенето на размените), защото при реализирания подход не е задължително да се включат всички останали непознати позиции в размените. Но дори и при текущата реализация това се случва много рядко в началото и повече към края като останат малко непознати позиции, така че това не е съществено подобрение.

4.4 Алтернативни подходи

Съществуват и разнообразни други алтернативни подходи, които да правят $N \log N + O(N)$ заявки. Всички с константа 1 пред $N \log N$ се базират на някакъв вид двоично тъвсене. Например `mummy_bobi_nlog.cpp` (което изкарва 92 точки) прави двоични търсения на база частични циклични шифтове (с идеята първо да се открие един цикличен шифт с 0 мача и друг с повече от 0). Всякакви такива (разумни) решения трябва да са 90+ точки.