**Анализ**

Първата подзадача е за 10 точки. Тук се иска единствено наивно решение – генериране на всички пермутации на ***N***-те числа и проверяване кои от тях се делят на 11. Сложността е .

Втората подзадача също е за 10 точки. Състезателите би трябвало да се усетят, че за задача, при която се въвежда само едно число и имат наивно решение могат да направят частичен прекъмпют на някои отговори. В случая първото решение е доста бавно, но можем да използваме един трик при прекъмпюта. Ако генерираме всички пермутации на числата от 1 до ***N***, ще имаме генерирани в тях и пермутациите на числата от 1 до ***N***-1, от 1 до ***N***-2 и т.н. Така можем да генерираме пермутациите само на числата от 1 до 13 (ограничението на подзадачата) и с тях да намерим отговорите за ***N*** от 1 до 13. Авторовият прекъмпют ползва и още един трик – по-бързо е да се намери търсената бройка при генериране на пермутациите с рекурсия, отколкото с вградената функция, защото докато ги генерираме с рекурсията можем да смятаме колко е текущия остатък при деление на 11, а иначе за всяка пермутация ще трябва да отделяме време за смятане на остатъка. Така тази програма за около 10 минути успява да намери нужните отговори.

Третата подзадача е за 15 точки. Тук също се прилага нещо стандартно за оптимизиране на пълното изчерпване (при генериране на пермутациите с рекурсия се вижда по-лесно). Наблюдението е следното: когато сме стигнали до някаква позиция и имаме някакъв текущ остатък, не ни интересува в какъв ред са били числата досега, а само множеството от тях. С други думи ако следващ път стигнем до тази позиция и сме използвали същото множество от числата и остатъка е същия, ще имаме същия възможен брой начини за продължаване, така че цялата пермутация да се дели на 11. Това можем да използваме за мемоизация и да направим така нареченото динамично оптимиране по подмножества. Стейтът е вече описаният – двоична маска на използваните числа досега в пермутацията и текущ остатък. Така сложността на изчерпването се оптимизира от факториел на експонента! В крайна сметка – .

Четвъртата подзадача е също за 15 точки. Разбира се с по-доброто пълно изчерпване можем да направим и по-добър прекъмпют. Отново правим оптимизации, за да е възможно най-бързо – ще го смятаме итеративно вместо рекурсивно. Сега по-лесно се вижда, че смятайки отговорът за дадено ***N***, ще сме намерили и за всички по-малки. В интерес на истината сигурно за 30 минути решението би могло да намери отговори и за ***N*** ≤ 30. Ограниченията на подзадачата са по-малки, защото имаме друг проблем – динамичното използва прекалено много памет. В интерес на истината, може да се направи реализация, която използва малко по-малко памет, но тя е много тежка, а това дори не е част от същинското решение на задачата.

Последната подзадача е за 50 точки. Лесно би трябвало да се вижда, че решението ще е с динамично програмиране. Първо с едно пълно изчерпване намираме отговора за едноцифрените числа. За малко забързване можем да използваме трика при обсъдения прекъмпют – от пермутациите на числата от 1 до 9 да намерим броя пермутации за числата от 1 до 8, от 1 до 7 и т.н. След това идва динамичното. Искаме като знаем за всеки остатък броя възможни пермутациите на ***N***-1 числа, да можем да намерим за всеки остатък броя възможни пермутации на ***N*** числа. Затова гледаме какво се променя при добавяне на новото число ***N***, към пермутация с ***N***-1 числа. Използва се следното наблюдение: когато добавяме число с четен брой цифри, не променяме четността на позициите на останалите цифри. Това означава, че като добавим двуцифреното число ab, то остатъка на новото число при деление с 11 ще се промени с a-b. Въпросът е дали се увеличава с това число остатъкът или намалява. Това се определя от това, на каква позиция слагаме ***N*** – четна или нечетна. Един проблем е, че двуцифреното число в представянето на пермутацията като число е неделимо, т.е. няма формула да определим броя четни позиции за добавяне и броя нечетни за добавяне като знаем броя на числата в пермутация (четност на позиция имаме предвид от гледна точка на слепеното число). Затова добавяме още едно измерение към динамичното – колко позиции са четни на пермутациите (или колко са нечетни). Така стейтът е следният – брой числа в пермутация, брой четни позиции на сформираното число от пермутация и остатъкът на това число. Лесно се вижда, че това е достатъчна информация, за да изчисляваме динамичното и накрая да намерим отговорът на задачата. Нека искаме да сметнем , където ***N*** са броя числа на пермутацията. Когато добавяме ***N*** към четна позиция, то броя четни позиции се увеличава с една, защото***N***също е с четен брой цифри. Например ако имаме пермутацията 1 2 3 …, позициите са както следва: н. 1 ч. 2 н. 3 ч. … Ако добавим ***N****=ab* на първа позиция ще имаме: н. *ab* н. 1 ч. 2 н. 3 ч. …, а ако го добавим на втора позиция ще имаме: н. 1 ч. *ab* ч. 2 н. 3 ч. … Общо взето това, което трябва да забележим, че ако добавяме ***N*** на четна позиция, увеличаваме четните позиции с 1, а ако е на нечетна позиция – увеличаваме нечетните позиции с 1. Така че от пермутациите, които са в (*k* са броя четни позиции), добавяйки ***N*** на четна позиция получаваме точно пермутации в , а ако добавим *n* на нечетна позиция ще получим пермутации . По-точно към ще добавим *k* пъти (общо *k* четни позиции за слагане), а към ще добавим ***N***-*k* (общо имаме ***N*** позиции за слагане в пермутацията с ***N***-1 числа). Това представляваше запълването на динамичното. Окончателната сложност е .

Задачата е само за ***N*** ≤ 99, защото при трицифрените числа вече идеята на динамичното не работи, понеже при добавяне на трицифрено число от дадено място нататък се сменя кои цифри са на четна позиция и кои на нечетна и полученият остатък зависи от това как са разположени числата и точно на коя позиция добавяме новото число.  
*Забележка: Условието е препратка към задачата от НОИ-1 2019, В група, perm16, където се иска същото нещо само че пермутациите, които се делят на 16.*

*Автор: Илиян Йорданов*