Подзадача 1 $N \leq 20$ **– 10 т.**

За тази подзадача предвиденото решение е символът да няма значение (например стоим само на **S**) и само да итерираме през състоянията, отброявайки на всяка стъпка, като прекратяваме изпълнение, когато стигнем N.

Подзадача 2 $N \leq 60$ **– 15 т.**

Тук решението е подобно, но трябва да се сетим за следната формула $3 \times 20 = 60$. С други думи, трябва да итерираме (за максималния случай) през всички комбинации от символ и състояние. Един ред, в който можем да направим това е да броим до 20 върху **S**, като на последната стъпка презапишем **0** и броим още 20, аналогично след това и с **1**. Естествено отново трябва да терминираме на коректната итерация.

Подзадача 3 $N \leq 120$ **– 25 т.**

За тази подзадача вече е нужно да се възползваме от факта, че имаме лента, а не само една позиция, защото иначе очевидно максималната стойност е 60. Предвиденото решение се състои от това да използваме половината си състояния да отидем надясно, а другата половина за връщането до **S**, като това се случва като докато се връщаме между всяка стъпка броим до 10. За да реализираме това решение, в началото му трябва да изчислим колко надясно да отидем ($N/10$) и кога точно да спрем (което се случва върху **S**). Може да се стигне 120, а не просто 100, защото можем да броим и върху **S** и можем да отидем 11 надясно използвайки 10 състояния – т.е. общо 10×12.

Подзадача 4 $N \leq 399$ **– 30 т.**

Решението тук е подобно, но вместо да е порядък 10^2 е 20^2. Това се случва като вместо да използваме други състояния за връщане назад, използваме друг символ. Това го правим като докато се движим надясно пишем **1**. Съответно състоянията ни водят до движение надясно, когато са върху **0**, а до движение наляво върху **1**. Конкретния максимум идва от 19×21.

Подзадача 5 $N \leq 500$ **– 40 т.**

Макар и да не изглежда така, тук се очаква да започнат решенията, които са по прости версии на пълното решение. Идеята е да построим брояч в двоична бройна система, като пазим битовете от най-младши към най-старши бит. Първо ще използваме 9 състояния движейки се надясно, за да напишем числото в двоична на лентата, а след това ще итерираме операции за вадене на 1 от числото. Когато то стигне 0 (което откриваме като в операцията на вадене отброим 9 състояния и още не сме видели единица) терминираме. Решението може да брои до $2^9 - 1$.

Тук има и още едно алтернативно решение, което се постига чрез един подход приложен върху решението за четвърта подзадача, но той ще бъде дискутиран по-късно в анализа. Основната му идея е да се възползваме от всички изпълнени инструкции по-добре.

Подзадача 6 $N \leq 1000$ **– 45 т.**

За тази подзадача идеята е да използваме решението на предния, но всяко вадене да отброяваме по два пъти (и съответно накрая или в някой момент да се оправим с нечетните стойности, например дали да отброяваме при терминирането или не). Този вид оптимизация върху брояч се използва и в някои от следващите подзадачи.

Подзадача 7 $N \leq 5000$ – **50 т.**

Тук отново правим оптимизация върху брояча, която ще фигурира и във финалното решение, както и в други подзадачи. Основното наблюдение е, че почти никои от състоянията си не използваме върху символа **S**. Идеята е да се възползваме от това, като между всеки две вадения използваме всички от тях като броим върху **S**-а итерирайки през всички. Съответно накрая ще трябва да се оправим с терминирането, така че отговорът ни да е верен по модул $\sim N$ (реално не точно, защото не през всички итерираме тогава).

Подзадача 8 $N \leq 130\,000$ – **60 т.**

За тази и следващите подзадачи е основната структура на брояча ни да прави 2^{20} итерации вместо 2^{10}. Проблемът при предния ни брояч е че първо записва числото и после отброява надолу. Решението е да започнем от 0, защото това е началното състояние на лентата и да отброяваме нагоре. Това отброяване нагоре се прави използвайки няколко от състоянията ни. След това между всеки две отброявания правим проверка, дали сме стигнали до нужната стойност като минем през всички битове на числото и видим дали са правилните, ако всички минат проверката – терминираме.

Подзадача 9 $N \leq 260\,000$ – **65 т.**

За тази подзадача използваме същата оптимизация както в шестата подзадача, но с този брояч.

Алтернативно можем да имаме по ефикасен брояч, който може да стига до по-високи стойности – един такъв е брояч, който не прави проверки, ами запазва числото в обратен ред и терминира когато стигне **S**. Това решение с други оптимизации също се представя добре, но не подлежи на една от ключовите за пълното решение.

Подзадача 10 $N \leq 600\,000$ – **70 т.**

Тук се очаква оптимизация подобна на тази от подзадача седем. Проблемът е, че при някои броячи тя изисква допълнителни приготовления, които заемат част от състоянията и съответно се брои до по малко.

Подзадача 11 $N \leq 1\,000\,000$ – **75 т.**

Тази и следващите няколко подзадачи все още могат да се хванат от някой от по-добрите броячи с оптимизации като от предната подзадача, но очакваното решение приема не изцяло конструктивен характер.

Основното наблюдение е, че теоретичният максимум за до колко можем да броим е броят инструкции, които изпълняваме. Трябва да пуснем решението до максималната си стойност и после да запишем по колко пъти се изпълнява всяка инструкция. Чак след това избираме на кои инструкции да прибавим отброяване и на кои не, така че сумата да е равна на N. При подходящ брояч тези стойности изпълняват закономерността, че всяко N под максимума може да се представи като сума на съответните стойности, и тогава алчното решение (да вземаме максималната стойност под текущата и да я вадим) работи.

Продължение

Следващата част от анализа ще представи няколко различни оптимизации и решения, които могат да се комбинират. Подзадачите са направени, така че максимално да рефлектират кои оптимизации присъстват, тъй като са възможни различни комбинации.

Подобрен базов брояч

Една добра идея е, тъй като вече не броим до конкретно N, а до максималното такова, да подобрим брояча си за тази цел. Повечето от тях поддържат лесен начин това да се случи, но един добър брояч е такъв, който само прибавя едно като следи на коя позиция е и ако не види единица до последното си състояние терминира.

Той използва само едно допълнително състояние за връщане назад и 19 състояния за движение напред. Върху **1** те пишат **0** и се придвижват напред и преминават в следващото състояние, а върху **1** пишат **0** и преминават в състоянието за връщане назад до **S**, когато то достигне там преминава в първото състояние за движене напред. Ако последното такова състояние все още е върху **1**, това би означавало да направим 20-тия бит на **1**, но това не може да се случи и затова приключваме изпълнение на програмата. Този брояч сам по себе си отброява 2^{19} пъти и изпълнява 2 милиона инструкции ($\sim 2^{19} \times 4$).

Леки оптимизации за движенията

Добре е на микро ниво да правим леки оптимизации по посоките на движения, за да увеличим броя изпълнени инструкции. Например, ако следващото състояние, към което минаваме, се движи само наляво, когато минаваме към него да се придвижм надясно, за да се увеличи броя изпълнени инструкции. Тези оптимизации са най-лесни за правене, но не се очаква да водят до вземане на по-висока подзадача освен в ограничен брой случаи, най-основния от които е за последните 5 точки на задачата. Приложена върху описания по-горе брояч такъв вид оптимизация прибавя около милион изпълнени инструкции ($\sim 2^{19} \times 2$).

Броене до K върху S

Тази оптимизация е много подобна като идея на тази от подзадачи седем и десет, но няма нужда да следим отговора да е верен по модул, тъй като самото изпълнение не зависи от стойността на N. Отново се възползваме от факта, че не използваме състоянията си върху **S**. Когато стигнем до него, изцикляме през всички тях и след това продължаваме напред със следващото прибавяне/вадене. Приложена върху брояча описан по-горе тази оптимизация прибавя около 10 милиона изпълнени инструкции ($\sim 2^{19} \times 19$).

Броене до K при увеличаване/намаляване

Тази оптимизация се базира на следното наблюдение: когато увеличаваме/вадим, вървим надясно докато стигнем до **0** и тогава веднага я сменяме на **1** преминаваме към движене назад към **S**. Това значи, че доста често спираме още на първия или втория бит и изобщо не стигаме до по-късните състояния и съответните им инструкции. Можем да променим това обаче, ако направим така, че само последното състояние от 19-те за движение напред да има тази функционалност (върху **0** да пише **1** и да тръгва назад), а всички останали в този случай само да стоят на място и да преминават към следващото състояние. Приложена върху брояча описан по-горе тази оптимизация прибавя около 9 милиона изпълнени инструкции ($\sim 2^{19} \times 17$).

Заключение

Крайното авторово решение поддържа броене до малко над 22 милиона ($\sim 2^{19} \times 42$). В прикачените към анализа решения може да намерите имплементирани всички дискутирани решения. Може да видите различни комбинации от оптимизациите и как те се представят. За целта в решенията те са наречени (в ред на анализа): OPT, ALL и EXH. Също така двата основни вида броячи са наречени (в ред на анализа): TWOSTEP и ONESTEP.