Задача 1. Калкулатор

Автор: Тодор Петров Петров

Решението на тази задача е динамично оптимиране върху, което са приложени няколко неща, които се забелязват при внимателен анализ на резултатите при по-малки входни данни.

Каква е идеята?

Дадено е, че имаме операциите събиране, умножение и изваждане. Нека да поразсъждаваме върху това какви могат да са последователните операции.

Ако имаме няколко последователни събирания и след това изваждане, очевидно това не е оптимално тъй като сме можели да направим само събиране с едно по-малко и да постигнем същия резултат с по-малко стъпки. Аналогично и за изваждане – няма смисъл да имаме няколко последователни изваждания и след това събиране. Следователно логичните операции, които могат да бъдат направени са: направо умножение, няколко събирания и след това умножение или няколко изваждания и след това умножение.
Нещата тук са допълнително усложнени и от факта, че имаме различен брой клетки в стека.

Тук трябва да спомена, че е добре задачата да се решава в обратен ред т.е. като имаме числото, което трябва да получим да се опитваме да го докараме до 0. Като тогава изваждане ще отговаря на събиране, събиране на изваждане и деление на умножение.

Нека видим какви параметри е удачно да има нашето динамично оптимиране. В случая ние търсим някакво число как може да бъде получено, разполагайки с определен брой клетки и то го търсим за минимален брой операции. Т.е. динамичното ни е двумерно (N, K).

По-горе разгледахме възможните операции => Ако използваме матрицата A[N][K], за да пазим минималните стъпки за получаване на число N, разполагайки с К клетки в стека, то:

A[N][K] = MAX(A[X][K – 1] * A[Y][K] + Z), където X, Y и Z са такива числа, че X * Y + Z = N. (разглеждат се всички такива тройки; X, Y > 0, докато Z може да е и отрицателно). Малко разяснение тук защо в единия случай разполагаме с К клетки, а в другия с К – 1: работата е в това, че за да получим числото У ние разполагаме с всичките К клетки, но след това то остава в една от клетките и можем да използваме само останалите клетки (К – 1 на брой), за да получим X, така че после да го умножим с Y.

Граничните случеи са:

1) Ако клетките са 2 т.е. K = 2, то лесно се вижда, че можем да правим само събиране и минималният брой операции е N * 2 – 1.
2) Ако N < 6, тогава се вижда, че A[N][K] = N * 2 – 1 също

И сега останаха два важни въпроса за разрешаване. Как определяме числата X, Y и Z. Анализ върху малките входни данни показва, че е достатъчно да се провери само за Z в интеревала [-3;3] (По любознателните могат да се опитат да го докажат и теоретично тъй като това, което сме описали е емпирично). След като се намери числото Z, то е ясно, че се търсят всички X и Y, за които X * Y = N – Z. Тези двойки (X, Y) не са никак много.

В допълнение може да се забележи още нещо, което е необходимо за да може задачата да работи достатъчно бързо. Също емпирично върху малките входни данни се вижда, че A[N][K] = A[N][5], за K >= 5, което допълнително намалява сложността на задачата. (Отново любознателните могат да опитат да докажат теоретично това).

И последния въпрос за решаване е как да си решим въпроса с паметта тъй като логично не можем да направим масив [1000000000][5]. Именно заради този проблем а и заради скорост тъй като ако попълним цялата матрица A би отнело доста време, използваме рекурсия с помнене т.е. всеки път като сметнем нещо за [N][K] и го запомняме за да не го смятаме пак ако ни потрябва втори път. В каква структура да го помним обаче след като не можем да създадем масив.

На помощ ни идва map от STL, с който симулираме масива. Но за да постигнем по-голямо бързодействие използваме и map и масив т.е. за по-малките тестове например за N < 1000000 – масив, след това ползваме map.

И така рекурсията с помнене започва от въведените N и K и в себе си отчита всички детайли, които споменахме по-горе. Написано по този начин влиза във времевия лимит.