Анализ на задача 1 – НАМЕРИ МЕДА

Задачата както забелязаха повечето от състезателите, може да се раздели на две абсолютно независими части.

Първата част е да се намери времето за претърсване на всяко гърне. Т.е. за дадена редица a_1, a_2 , a_3… a_m да се намери най-дългата нарастваща подредица от не непременно съседни елементи. Това е известен проблеми за неговото ефективно решаване се изисква използването на динамично оптимиране. Ще изложа накратко два алгоритъма с различна алгоритмична сложност.

Алгоритъм 1: Нека f[k] е най-дългата нарастваща редица с последен елемент a_k. Тогава $f[k] = max(f[i], 0) + 1$, където $1 \leq i < k$ и $a_i < a_k$. Т.е. най-доброто решение с последен елемент a_k ние е равен на най-дългата редица, която можем да продължим увеличена 1. Пресмятаме последователно всички стойности на f[] отговорът на задачата ни е равен на max(f[i]), където $1 \leq i \leq m$. Този алгоритъм има алгоритмична сложност $O(m^2)$ и излиза от времевото ограничение, за по-големите тестове.

Алгоритъм 2: Обхождаме елементите на редицата един по един. Нека сме стигнали до елемент a_i, g[к] ни е равна на най-малкия елемент, на който може да завършва нарастваща редица с дължина k до момента, а len ни е дължината на най-дългата нарастваща редица до текущия елемент. В началото len е равно на 0. На всяка стъпка намираме минималното j, така че $g[j] \geq a_i$ и приравняваме $g[j] = a_i$. В случай, че не съществува такова j (т.е. $a_i > g[j]$: за всяко j = 1…len), то увеличаваме len с едно и присвояваме $g[len] = a_i$. След като сме обходили всички елементи отговорът на задачата ни е равен на len.

Забележете, че така получената редица ни е строго растяща. Затова на всяка стъпка можем да намерим индекс j чрез двоично търсене с алгоритмична сложност $O(\log_2 m)$ и да получим алгоритъм с обща сложност $O(m.\log_2 m)$. Доказателството за верността на алгоритъма оставям на читателя като леко упражнение.

Като приложим алгоритъм за намиране на най-дълга нарастваща редица за всяко гърне получаваме алгоритмична сложност $O(n.m.\log_2 m)$, където n е броят на гърнетата. Вече сме пресметнали необходимите ни стойности за да преминем към втората част.

Втората част по дадени цени на гърнета наредени в редица b_1, b_2 b_3 ... b_n, да се намери оптимален ред на взимане на гърнетата. Един очевиден алгоритъм за решаване на задачата е пробването на всички възможни наредби. Този алгоритъм има алгоритмична сложност (n!) и очевидно е твърде бавен. С този подход може да се реши само един от десетте тестови примера.

Другия подход отново се основава на динамичното оптимиране. А именно нека d[i][j] ни е равно на времето, което ни е нужно да вземем гърнетата с номера от i до j по оптимален начин. При i > j, d[i][j] = 0, а при i = j => $d[i][j] = b_i = b_j$. Забележете, че взимането на някое гърне от редицата ни разбива задачата на две абсолютно независими части – да намерим оптималния ред в лявата и в дясната част. Затова $d[i][j] = b_i + b_{i+1} + …+ b_{j-1} + b_j + min(d[i][k-1] + d[k+1][j])$, където k е в интервала [i,j]. Отговорът на задачата ни е d[1][n]. Тъй като за всяка двойка (i,j), обхождаме числата от i до j. То този алгоритъм има алгоритмична сложност $O(n^3)$. Този алгоритъм винаги намира правилното решение, но е достатъчно бърз за пет от десетте тестови примера. Много от състезателите, които имат 50 точки са реализирали именно него.

Най-интересния момент в задачата е оптимизирането на този алгоритъм. Една разпространена техника при динамичното оптимиране е съкращаване на вътрешния цикъл. Тази техника е подходяща и при тази задача. Нека p[i][j] ни е равно на k, при което се получава минимална сума d[i][k-1] + d[k+1][j]. Очевидно p[i][i] = i. Тогава след задълбочено наблюдение забелязваме, че p[i][j-1] ≤ p[i][j] ≤ p[i+1][j]. Този факт се доказва чрез пълна математическа индукция по дължината на интервала и чрез допускане на противното.

Това неравенство ни позволява, когато търсим оптималното к за интервала [i,j] да не обхождаме всички числа между i и j, а само частта от тях, които го удовлетворяват. Така за интервала [1,x] ще обходим индексите от p[1][x-1] до p[2][x]. За интервала [2,x+1], ще обходим индексите от p[2][x] до p[3][x+1]. За интервала [3,x+2] от p[3][x+1] до [4][x+2] и така нататък. Неформално казано - всеки път ще продължаваме от там откъде се приключили предишния път. Т.е. ще пресметнем всички интервали с дължина x чрез едно обхождане на числата от 1 до n или с O(n) операции. В крайна сметка за всички възможни n-1 дължини ще направим същото нещо и ще получим алгоритъм с алгоритмична сложност $O(n^2)$. Който комбиниран с втория изложен алгоритъм за намиране на най-дълга нарастваща редица има сложност (n^2 + $n.m.\log_2 m$) хваща десет от десетте тестови примера и получава пълен брой точки.

Този алгоритъм или близък до този е реализиран от всички състезатели с пълен брой точки на задачата. Като изключително ефективната реализация на Свилен Марчев го изведе една крачка пред останалите и му донесе първото място.