Laundry

Тагове	На пълното решение	На подзадачите
	Двоично търсене Сметки	Пълно изчерпване Динамично програмиране Лаком алгоритъм

Анализ

Подзадача №1

Както винаги, оставих подзадача с тестовия пример, за да има по-добра обратна връзка от системата.

Решение с Пълно изчерпване

Подзадача №2

Както повечето задачи в информатиката, и в тази може да се приложи пълно изчерпване. Може да забележим, че никога не е оптимално да има празна купчина в редицата. Възползвайки се от това, ние може да разгледаме всички начини на разбиване на N панера на M купчини и да намерим най-оптималния от тях. Забележете, броят на разбиванията е равен на 2^{N-1}, защото всяко разбиване може да бъде описано по единствен начин от число в двоична бройна система с N бита, завършващо на 1, където всяка единица отделя купчина (например за $N = 10$, $\{3,4,2,1\}$ отговаря на $0010001011_{(2)}$), или иначе казано има биекция между двете. Така, ние може да разгледаме всяко едно разбиване с рекурсия от вида `brute(pos, left)`, като pos е позицията, която сме достигнали в разбиването, а $left$ е броя на останалите панери за разпределяне. С вътрешен цикъл може да фиксираме големината на текущата купчина.

Постигната сложност: $O(T \times 2^{N-1} \times N)$.

Имплементация: `laundry_10p.cpp`

Решения с Динамично програмиране

Подзадача №3

За тази подзадача единствено трябва да забележим, че стейта на текущото разбиване се характеризира само от pos и $left$. Така може да добавим мемоизация към горното пълно изчерпване, да преименуваме `brute` на `f` и да му залепим стикер, че вече е ДП. Така сложността на динамичното ще е $O(N^3)$: $O(N^2)$ за стейта и $O(N)$ за вътрешния цикъл.

Постигната сложност: $O(TN^3)$.

Имплементация: `laundry_18p.cpp`

Подзадача №4

Нека разширим идеята с ДПто. То трябва да изглежда по подобен начин на този:

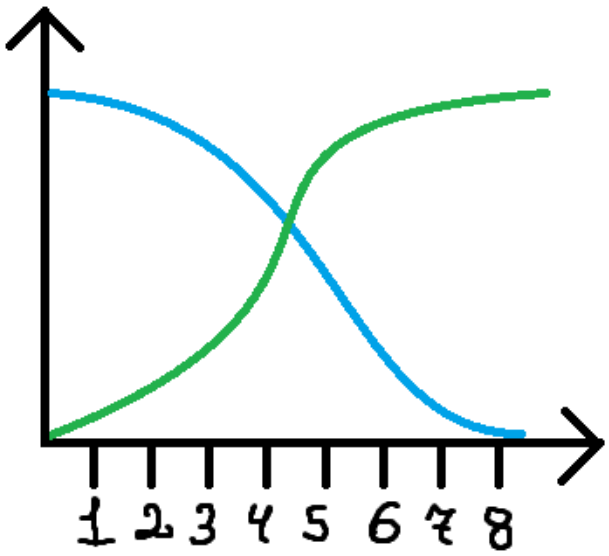
```
dp[pos][left] = INF;
for (int i = 1 ; i <= left ; ++i)
{
    dp[pos][left] = std::min(dp[pos][left],
    std::max(f(pos + 1, left - i), cost(pos, i));
}

return dp[pos][left];
```

Където неудобността на текущата купчина, която е с i панера, е равна на $cost(pos, i) = pos \times a + (i - 1) \times b$, защото най-трудно достижим е най-долният панер, като ще трябва да вдигнем всички $i - 1$ над него.

Очевидно, $dp[pos][left + 1] \geq dp[pos][left]$, защото ще ни отнеме повече време да достигнем $left + 1$ панера от $left$ на брой, ако ги поставяме от позиция pos нататък. Така с нарастване на i, $f(pos + 1, left - i)$ намалява.

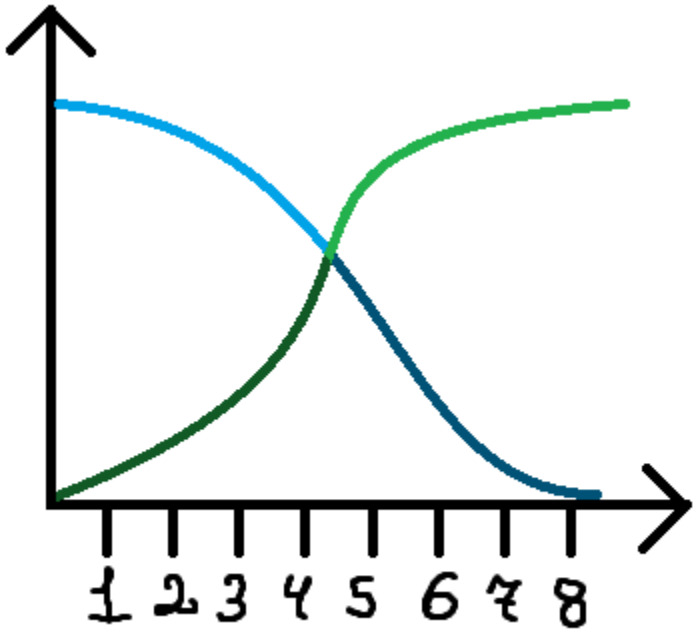
Очевидно и $cost(pos, i) \leq cost(pos, i + 1)$, защото $pos \times a + (i - 1) \times b \leq pos \times a + i \times b$. Така с нарастване на i, $cost(pos, i)$ нараства.

С тази информация може да си зададем въпроса – колко смислени i-та разглеждаме? Разгледайте графиката отдолу.



Laundry

Представете си, че абсцисата е стойността на i, синята крива – стойността на $f(pos + 1, left - i)$, а зелената крива – стойността на $cost(pos, i)$. Така погледнато, стойността на резултата ни спрямо i ще бъде по-светлия цвят на долната графика:



Ние целим да разгледаме отговорите за последната цяла точка, в която синята крива е над зелената, както и първата цяла точка, в която зелената е над синята. Изобразено на графиката,тези точки ще са с $i = 4{,}5$, като няма смисъл да разглеждаме отговорите за $i = 1{,}2{,}3$, защото $i = 4$ е по-оптимално, както и няма смисъл да разглеждаме отговорите за $i = 6{,}7{,}8$, защото $i = 5$ е по-оптимално. Така ние ще трябва единствено да проверим за 2 стойности на i. Единствено трябва да намерим къде са тези две точки, което може да стане с двоично търсене.

Постигната сложност: $O(TN^2 \log_2 N)$.

Имплементация: `laundry_38p.cpp`

Подзадача №5

Може лесно да разширим идеята от горната подзадача, като забележим, че търсените две точки се характеризират от пресечната точка на функиите на $f(pos + 1, left - i)$ и $cost(pos, i)$. Също, тази точка нараства при фиксирано pos и нарастващо $left$, тоест може да заменим двоичното търсене с показалки.

Постигната сложност: $O(TN^2)$.

Имплементация: `laundry_47p.cpp`

Решения с Лаком алгоритъм

Подзадача №5

Едно много лесно, същевременно важно наблюдение, е, че с учеливането на броя панери в една купчина, нараства неудобността за достигане на най-долния панер. Така може да се достигне следната идея – да се добавят панерите един по един в купчините, като всеки път да слагаме текущия панер в тази купчина, чиято неудобността след слагането му е най-малка. Иначе казано, слагаме панера там, където в момента е най-оптимално. Така за всеки панер трябва да проверим N купчини и да намерим най-оптималната от тях, като постигаме сложност от $O(N^2)$. Лакомият алгоритъм винаги достига оптимално решение, като това може да бъде доказано с допускане на противното.

Постигната сложност: $O(TN^2)$.

Имплементация: `laundryGreedy_47p.cpp`

Подзадача №6

Може да приложим алгоритъма, обяснен отгоре, като го оптимизираме. Той се свежда до това да поддържаме минимум на множество от елементи, от което N пъти трием най-малкия и добавяме нова стойност. Това може да бъде реализирано лесно чрез двоична пирамида, или иначе казано `priority_queue`.

Постигната сложност: $O(TN \log_2 N)$.

Имплементация: `laundry_60p.cpp`

Решения с Двоично търсене по отговора

Подзадача №7

Може много лесно да въведем двоично търсене по отговора в задачата – по дадено $\max$, нека $g(\max)$ е равно на максималния брой панери, които може да сложим в редица, така че най-отдалечения да е на разстояние $\leq \max$. Тъй като $g(1) \leq g(2) \leq \cdots \leq g(\infty)$, ние може с двоично търсене да намерим първото $\max$, за което $g(\max) \geq N$, като тогава отговора ни ще е равен на $\max$.

И сега остава въпроса – как да намерим $g(\max)$ по дадено $\max$. Може да забележим доста удобно свойство: $p_i \geq p_{i+1}$. Освен това, $\sum_{i=1}^{N} p_i = N$, следователно измежду $p_1, p_2, p_3, \ldots, p_N$ има до $\sqrt{2N}$ различни положителни стойности. Това е вярно, защото при наличието на x различни положително стойности, сборът им е поне $\sum_{i=1}^{x} i = \frac{x(x+1)}{2}$, следователно $\frac{x(x+1)}{2} \leq N$, тоест $x^2 + x \leq 2N$, $x \leq \sqrt{2N}$. Така винаги ще имаме до $\sqrt{2N}$ групи от последователни различни числа, където $\sqrt{2N}$ е относително малко – до $\approx 6 \times 10^4$. Следователно ако успеем да изчислим $g(\max)$ за сложност, от порядъка на броя на различни числа в редицата, ние ще си решим задачата.

Нека почнем отзад-напред. Коя ще е последната позиция, на която може да сложим панер? Ние ще трябва да я достигнем, следователно $maxPos \times a \leq \max$, тоест $maxPos = \lfloor \max/a \rfloor$. Ние може с двоично търсене да открием колко панера може да сложим на тази последна позиция. Нека този брой са x. След това може да открием с ново двоично търсене коя е най-дясната позиция, на която може да сложим поне $x + 1$ панера. Нека тази позиция е pos. Така на всяка позиция от $pos + 1$ до $maxPos$ ще може да сложим по точно x панера, след което може да продължим със същия алгоритъм за следващата група, която почва от pos-та позиция. Така за всяка група ще правим две двоични търсения, изчисляването на отговора ни ще е със сложност $O(\sqrt{N} \log_2 N)$ и крайното решение ще е със сложност $O\left(T\sqrt{N} \log_2^2 N\right)$.

Постигната сложност: $O(T\sqrt{N} \log_2^2 N)$.

Имплементация: `laundry_88p.cpp`

Подзадача №8

Остана много малко до 100-те точки – единствено да заменим двете двоични търсения със сметки. Те бяха:

1) По дадено $\max$ и позиция pos, да намерим колко най-много панера може да сложим.
   - Нека този брой е x. Тогава:
   - $pos \times a + (x-1) \times b \leq max$ и $pos \times a + x \times b > max$
   - $(x-1) \times b \leq max - pos \times a$ и $x \times b > max - pos \times a$
   - $(x-1) \leq \frac{max - pos \times a}{b}$ и $x > \frac{(max - pos \times a)}{b}$
   - $x \leq \frac{max - pos \times a}{b} + 1$ и $x > \frac{(max - pos \times a)}{b}$
   - $=> x = \left\lfloor \frac{max - pos \times a}{b} \right\rfloor + 1$
2) По дадено max и брой панери x, да намерим най-дясната позиция, на която може да сложим поне $x + 1$ панера.
   - Нека тази позиция е pos. Тогава
   - $pos \times a + x \times b \leq max$
   - $pos \times a \leq max - x \times b$.
   - $pos \leq \frac{max - x \times b}{a}$
   - $=> pos = \left\lfloor \frac{max - x \times b}{a} \right\rfloor$

С тези две пресмятания, задачата ни е решена.

Постигната сложност: $O(T\sqrt{N} \log_2 N)$

Имплементация: `laundry_100p.cpp`

Автори: Борис Михов, Даниел Койнов

Laundry

П.П: Докато правих задачата се оказа, че решение, с, на пръв поглед, сложност $O(T\sqrt{N}\log_2^2 N)$ се държи по-добре от авторовото. Не съм смятал по-подробно каква е истинската сложност на решението, но е учудващо, защото, някак си, двоично търсене се държи по-добре от деление. Тъй като постановката на задачата е ограничаваща откъм тестове (входа е 3 числа), не вярвам, че тестовете са ми твърде специфични, а по-скоро че самата задача е такава, че двоичното търсене прави относително малко стъпки, които са предимно умножение, събиране и шифтване (деление на 2). Поуката, която може да се извлече от това, е че при възможност да се пропусне $/$ или $\%$, го правете, особено когато работите с модули (примерно при динамични) ☺

П.П №2: Задачата е по действителен случай. Преди време видях, че майка ми е наредила трите панера за пране един върху друг и се замислих как да ѝ обясня, че не е оптимално (трябваше да оставя прането си в най-долния панер). В последствие разказах случката на Дани, който успя да формулира задачата в текущия ѝ вид.