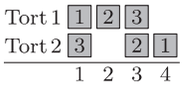
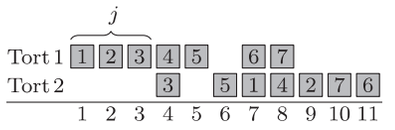


Rys. 1. Dla $n = 3$ i tortów o warstwach $1, 2, 3$ oraz $3, 2, 1$ potrzebujemy czterech minut.

Rozwiązanie zadania F 839.
Podczas jazdy z dużymi prędkościami niemal cała moc silnika zużywana jest na przezwyciężanie oporu powietrza. Siła oporu aerodynamicznego jest proporcjonalna do kwadratu prędkości pojazdu. Moc zużywana do utrzymania stałej prędkości jest więc proporcjonalna do trzeciej potęgi tejże prędkości. Po wymianie silnika maksymalna prędkość wzrośnie $2^{1/3} = 1{,}26$ raza.



Rys. 2. Przykład dla $j = 3$. Potrzebujemy $n = 7$ minut dla pierwszego tortu i dodatkowo $j + p(j) = 4$ minut dla pozostałych warstw drugiego tortu.

Rozwiązanie zadania M 1397.
Z nierówności między średnią arytmetyczną i geometryczną mamy

$$\left(\frac{1+x}{n+1}\right)^{n+1} = \left(\frac{n\cdot\frac{1}{n}+x}{n+1}\right)^{n+1} \geqslant \left(\frac{1}{n}\right)^{n}\cdot x.$$

Informatyczny kącik olimpijski (64): Dwa torty

W tym kąciku omówimy *Dwa torty* – kolejne zadanie z finałowej rundy *Potyczek Algorytmicznych* 2012. Torty oferowane przez cukiernię składają się z n różnych rodzajów warstw ułożonych w pewnej kolejności. Cukiernia ta zatrudnia n cukierników. Każdy z nich potrafi wykonać warstwę jednego rodzaju, co zajmuje mu dokładnie jedną minutę (i w tym czasie może zajmować się tylko jednym tortem). Warstwy na każdym torcie należy układać jedna po drugiej. Chcemy wyprodukować dwa torty, dla każdego z nich znamy wymaganą kolejność warstw. Należy obliczyć, jak najszybciej da się to zrobić (rys. 1).

Dość łatwo podać rozwiązanie tego zadania oparte na programowaniu dynamicznym i działające w czasie $O(n^2)$. Niech $a[i], b[i]$ oznaczają rodzaj i-tej warstwy od dołu (dla $1 \leqslant i \leqslant n$) odpowiednio dla pierwszego i drugiego tortu. Niech $d[i, j]$ oznacza minimalny czas ułożenia dolnych i warstw na pierwszym torcie i dolnych j warstw na drugim torcie. Wtedy

$$d[i,j] = \begin{cases} i + j & \text{dla } i = 0 \text{ lub } j = 0, \\ 1 + d[i-1, j-1] & \text{dla } i, j \geqslant 1 \text{ i } a[i] \neq b[j], \\ 1 + \min(d[i-1, j] + d[i, j-1]) & \text{w przeciwnym przypadku.} \end{cases}$$

Zależność rekurencyjna wynika z tego, że warstwy $a[i]$ i $b[j]$ mogą być wykonane w tym samym czasie tylko wtedy, gdy są różnego rodzaju. W przeciwnym przypadku musimy się zdecydować, którą z nich wykonujemy najpierw. Wynikiem jest $d[n, n]$.

Oczywiste jest to, że wynik będzie w granicach od n do $2n$. Jednak Czytelnicy, którzy zechcą przeprowadzić kilka eksperymentów praktycznych, przekonają się, że znalezienie przykładu z wynikiem bliskim górnej granicy nie jest wcale łatwe. W istocie bowiem wynik nigdy nie jest większy niż $n + \sqrt{n}$. To pozwala nam przyspieszyć rozwiązanie kwadratowe do $O(n\sqrt{n})$, gdyż musimy jedynie wypełniać przekątne tablicy d leżące nie dalej niż $\sqrt{n}$ od głównej przekątnej (tzn. możemy założyć, że $d[i, j] = \infty$ dla $|i - j| > \sqrt{n}$).

Musimy jeszcze wykazać, że zawsze istnieje strategia produkcji tortów, w której wystarczy $n + \sqrt{n}$ minut. Dla uproszczenia załóżmy, że n jest kwadratem liczby naturalnej. Dla ustalonego $j \geqslant 0$ możemy zastosować następującą strategię produkcji tortów: najpierw wykonujemy pierwsze j warstw pierwszego tortu, następnie $n - j$ par warstw (warstwę $a[i + j]$ równocześnie z warstwą $b[i]$, chyba że $a[i + j] = b[i]$, to wtedy po kolei), i w końcu ostatnie j warstw drugiego tortu (rys. 2). Będziemy potrzebowali n minut na wszystkie warstwy pierwszego tortu, j minut na ostatnie warstwy drugiego tortu oraz $p(j)$ minut na te z pozostałych warstw drugiego tortu, które nie zostały „sparowane” z warstwami pierwszego tortu. Dla $j < 0$ definiujemy analogiczną strategię, tylko zaczynamy od pierwszych $-j$ warstw drugiego tortu.

Kluczową obserwacją jest to, że dla ustalonego rodzaju warstwy k warunek $a[i + j] = k = b[i]$ jest spełniony dla dokładnie jednego j. Z tego wynika, że suma $\sum_j p(j)$ jest równa n. Sumując czas potrzebny dla wszystkich strategii dla $-\sqrt{n} \leqslant j \leqslant \sqrt{n}$, dostajemy:

$$\sum_{-\sqrt{n}\leqslant j\leqslant\sqrt{n}} n + |j| + p(j) \leqslant n(2\sqrt{n}+1) + 2(1 + 2 + \ldots + \sqrt{n}) + n = (n + \sqrt{n})(2\sqrt{n} + 1),$$

czyli średnio na strategię potrzebujemy nie więcej niż $n + \sqrt{n}$ minut, zatem istnieje takie j, dla którego tyle wystarczy.

Dla pełności dodajmy, że wynik $n + \sqrt{n}$ uzyskujemy dla tortu $1, 2, \ldots, n$ oraz tortu, w którym warstwy dzielimy na grupy kolejnych warstw o rozmiarach $1, 2, \ldots, \sqrt{n} - 1, \sqrt{n}, \sqrt{n} - 1, \sqrt{n} - 2, \ldots, 1$, a następnie odwracamy kolejność warstw w każdej grupie. Przykładowo, dla $n = 16$ drugim tortem jest

$$1,\ 3, 2,\ 6, 5, 4,\ 10, 9, 8, 7,\ 13, 12, 11,\ 15, 14,\ 16.$$

Powyższe rozważania stają się nieistotne, jeśli przyjrzymy się dokładniej rekurencji definiującej tablicę d. Zauważmy, że obliczenie $d[i, j]$ w przypadku $a[i] \neq b[j]$ zależy tylko od jednej komórki tabeli. Zatem możemy rozwinąć wzór do $d[i, j] = k + d[i - k, j - k]$, gdzie k jest taką najmniejszą liczbą, że $a[i - k] = b[j - k]$ i $0 < k \leqslant \min(i, j)$, lub $d[i, j] = i + j - \min(i, j)$, gdy takie k nie istnieje.

Zatem obliczenie dowolnego $d[i, j]$ sprowadza się do wyznaczenia wartości k (co można zrobić w czasie $O(\log n)$, wyszukując binarnie wśród par (i', j'), dla których $a[i'] = b[j']$ oraz $i - j = i' - j'$) i obliczenia $d[i', j']$ dla przypadku $a[i'] = b[j']$. Zauważmy jednak, że mamy dokładnie n par (i', j'), dla których $a[i'] = b[j']$. Wyznaczając więc $d[n, n]$ metodą rekurencji ze spamiętywaniem, dostajemy rozwiązanie działające w czasie $O(n \log n)$. Czytelnik Gorliwy zechce uzupełnić techniczne szczegóły, które przyspieszą powyższe rozwiązanie do optymalnego $O(n)$.

Tomasz IDZIASZEK