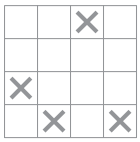
Informatyczny kącik olimpijski (87): Zliczamy puste prostokąty



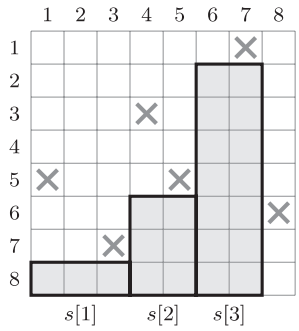
Rys. 1. Przykładowy kwadrat rozmiaru $n = 4$ z czterema polami zabronionymi (zaznaczone krzyżykami). Największy pusty prostokąt ma rozmiar 2×3. Poniższa tabelka przedstawia liczności pustych prostokątów dla wszystkich rozmiarów $h \times w$:

$h \times w$	Liczba	$h \times w$	Liczba
1×1	12	2×1	7
1×2	6	2×2	3
1×3	3	2×3	1
1×4	1	3×1	3

W tym miesiącu zajmiemy się dość klasycznym zadaniem. Dany jest kwadrat rozmiaru $n \times n$ podzielony na n^2 pól, przy czym niektóre pola są zabronione. Dowolny zawarty w tym kwadracie prostokąt, który nie zawiera żadnego pola zabronionego, nazwiemy *prostokątem pustym* (rys. 1). Należy znaleźć pusty prostokąt o jak największym polu.

To zadanie nie powinno być nowością ani dla miłośników zadań olimpijskich (pojawiło się m.in. na IX Olimpiadzie Informatycznej jako zadanie pt. *Działka*), ani dla stałych czytelników *Delty* (omawialiśmy jego rozwiązanie w artykule *Prostokąt arytmetyczny* w numerze 3/2012). Co więcej, różne modyfikacje tego zadania pojawiały się też (i nadal pojawiają) na innych konkursach. Przykładowe modyfikacje mogą polegać na tym, że jesteśmy proszeni o znalezienie pustego prostokąta o największym obwodzie, liczby pustych prostokątów czy też sumarycznego pola powierzchni wszystkich pustych prostokątów. I choć główna idea rozwiązania wszystkich tych wersji zadania jest taka sama, to różnią się one w szczegółach, które każdorazowo trzeba dopracować. Tutaj przedstawimy dość ogólną metodę, która pozwoli nam bardzo łatwo rozwiązywać podobne zadania. A mianowicie pokażemy, jak wyznaczyć w sumarycznym czasie $O(n^2)$ liczbę pustych prostokątów rozmiaru $h \times w$ dla wszystkich wartości $1 \leqslant h, w \leqslant n$.

Pole leżące na przecięciu i-tego wiersza i j-tej kolumny kwadratu będzie miało współrzędne (i, j). Przede wszystkim dla każdego pola (i, j) będziemy chcieli znać liczbę niezabronionych pól leżących powyżej niego (włącznie z tym polem); oznaczymy tę wartość przez $d[i, j]$. Całą tablicę d łatwo obliczyć w czasie $O(n^2)$, gdyż $d[i, j] = 0$, jeśli pole (i, j) jest zabronione oraz $d[i, j] = d[i - 1, j] + 1$ w przeciwnym przypadku. Dla uproszczenia zakładamy również, że wszystkie pola poza kwadratem są zabronione i $d[i, 0] = d[i, n + 1] = 0$.



Rys. 2. Zacieniono trzy prostokąty, które reprezentują obszar dla pola $(8, 7)$. Poniższa tabelka przedstawia zawartość stosu dla pól $(8, j)$, gdy $0 \leqslant j \leqslant 9$.

j	$d[8, j]$	Stos (elementy $h \times w$)
0	0	pusty
1	3	3×1
2	8	3×1, 8×1
3	1	1×3
4	5	1×3, 5×1
5	3	1×3, 3×2
6	8	1×3, 3×2, 8×1
7	7	1×3, 3×2, 7×2
8	2	1×3, 2×5
9	0	pusty

Rozwiązanie podzielimy na n faz; w i-tej fazie będziemy rozpatrywać prostokąty, których dolny bok zawiera pola z i-tego wiersza. Ustalmy pewne pole (i, j) i rozważmy wszystkie puste prostokąty, których prawy dolny róg znajduje się na tym polu. Zbiór wszystkich pól, w których może znajdować się lewy górny róg takiego pustego prostokąta, jest *obszarem*, którego górny obrys składa się z odcinków idących w prawo i do góry. Taki obszar będziemy reprezentowali jako sumę minimalnej liczby rozłącznych prostokątów o coraz większych wysokościach, których dolne boki dotykają i-tego wiersza (rys. 2). Rozmiary $h \times w$ kolejnych prostokątów będziemy trzymać na stosie, co umożliwi nam łatwe uaktualnianie tej reprezentacji, jeśli będziemy przeglądać kolumny od lewej do prawej.

Stos będziemy trzymać w tablicy s (wysokości i szerokości i-tego prostokąta na stosie to odpowiednio $s[i].h$ oraz $s[i].w$), a liczbę jego elementów w zmiennej m. Poniższy pseudokod pokazuje, jak uaktualnić zawartość stosu dla pola $(i, j - 1)$, aby odpowiadała ona polu (i, j). Dopóki wysokość $s[m].h$ najwyższego prostokąta na stosie jest co najmniej tak duża jak $d[i, j]$, to usuwamy go ze stosu. Następnie wrzucamy na stos prostokąt rozmiaru $d[i, j] \times w$, gdzie w jest sumaryczną długością wszystkich usuniętych prostokątów plus 1.

```
w := 0;
while m > 0 and s[m].h ⩾ d[i, j] do
  inc(w, s[m].w);
  dec(m);
inc(m);
s[m].w := w + 1;
s[m].h := d[i, j];
```

Zauważmy, że każdy prostokąt usuwany ze stosu jest kandydatem na największy pusty prostokąt. Co więcej, wystarczy, że rozpatrzymy tylko tych kandydatów. Ponieważ dla ustalonego i liczba operacji wykonanych na stosie jest $O(n)$, gdyż wrzucamy na niego n elementów, zatem cały algorytm działa w czasie $O(n^2)$.

Aby wyznaczyć liczbę pustych prostokątów wszystkich rozmiarów, postąpimy nieco sprytniej. Zachowamy następujący niezmiennik: jeśli znajdujemy się na polu (i, j), to znaczy, że policzyliśmy już te puste prostokąty, których prawy

dolny róg jest na polu (i, j') dla $j' < j$ i jednocześnie *nie* są one zawarte w obszarze dla pola (i, j).

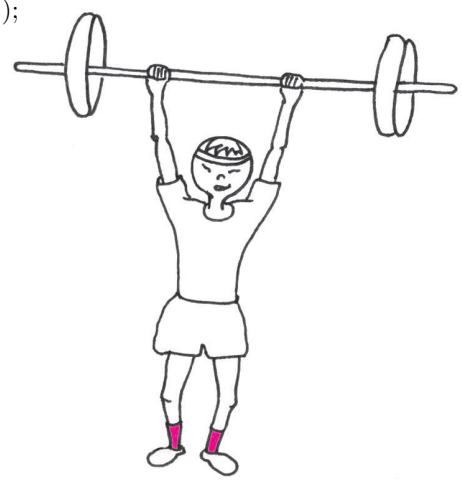
Zatem zwiększanie obszaru (poprzez wrzucanie nowych elementów na stos) nigdy nie będzie zmieniać liczności rozważonych pustych prostokątów, natomiast usuwanie elementów ze stosu może powodować konieczność aktualizacji. Wszystkie pola usuniętego prostokąta $s[m]$, które znajdują się w odległości co najmniej $h := \max(s[m-1].h, d[i, j])$ od wiersza i na pewno zostaną usunięte z obszaru (zakładamy przy tym, że $s[0].h = 0$). Musimy zatem policzyć puste prostokąty zawarte w prostokącie $s[m]$, które przestaną być w tym obszarze zawarte, czyli ich lewy górny róg jest jednym z pól, które zostaną usunięte z obszaru. Te prostokąty mają wysokości od $h+1$ do $s[m].h$ oraz szerokości od 1 do $s[m].w$, przy czym prostokątów o rozmiarze $y \times x$ jest $s[m].w + 1 - x$ (rys. 3).

Należy też uważnie rozważyć przypadek, gdy dla danego pola (i, j) usuwamy więcej niż jeden element ze stosu. Wtedy szerokość aktualnie usuwanego elementu należy zwiększyć o szerokości usuniętych poprzednio (wygodnie jest tu korzystać z wprowadzonej już zmiennej w).

Jeśli zatem liczności pustych prostokątów będziemy zapisywać w tablicy c (gdzie $c[y, x]$ oznacza liczbę pustych prostokątów rozmiaru $y \times x$), to mamy do rozwiązania następujący problem: dla danych liczb h_1, h_2 i w chcemy szybko wykonywać operację zwiększania komórek $c[y, x]$ o wartość $w + 1 - x$ dla $h_1 \leqslant y \leqslant h_2$ oraz $1 \leqslant x \leqslant w$. Nie możemy, oczywiście, pozwolić sobie na zrobienie tego zupełnie naiwnie. Zakładając, że początkowo w tablicy c są same zera, zastosujemy następujący trik: do komórki $c[h_1, w]$ wpiszemy 1, a do komórki $c[h_2 + 1, w]$ wpiszemy -1. Następnie niezależnie w każdej kolumnie tablicy c obliczymy sumy prefiksowe. To spowoduje, że jedynki znajdą się w komórkach $c[y, w]$ dla $h_1 \leqslant y \leqslant h_2$ (a -1 zniknie). Następnie w każdym wierszu obliczymy „trochę lepsze" sumy sufiksowe, które każdą jedynkę w komórce $c[y, x]$ zamienią na ciąg liczb $x, x-1, \ldots, 1$ w komórkach $c[y, 1], c[y, 2], \ldots, c[y, x]$. Po takich manipulacjach tablica c ma wartości zgodne z pojedynczą operacją zwiększania.

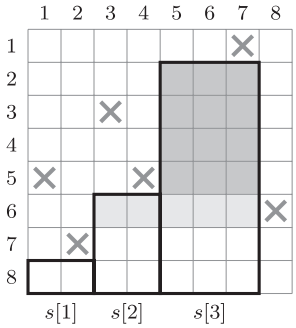
Nietrudno się przekonać, że jeśli chcemy wykonać więcej niż jedną operację zwiększania, to możemy najpierw zwiększyć wszystkie komórki $c[h_1, w]$ o jeden i zmniejszyć odpowiadające im komórki $c[h_2 + 1, w]$ o jeden, a dopiero na sam koniec jeden raz obliczyć powyższe sumy. Łącznie zajmie to czas $O(n^2)$, a pseudokod algorytmu może wyglądać następująco:

```
for i := 1 to n do
  for j := 0 to n + 1 do
    w := 0;
    while m > 0 and s[m].h ⩾ d[i, j] do
      h := max(s[m − 1].h, d[i, j]);
      inc(w, s[m].w);
      inc(c[h + 1, w]);
      dec(c[s[m].h + 1, w]);
      dec(m);
    inc(m);
    s[m].w := w + 1;
    s[m].h := d[i, j];
for i := 1 to n do
  for j := 1 to n do
    inc(c[i + 1, j], c[i, j]);
  c1 := c2 := 0;
  for j := n downto 1 do
    inc(c1, c[i, j]);
    inc(c2, c1);
    c[i, j] := c2;
```



Jako ćwiczenie dla Czytelnika proponujemy modyfikację tego algorytmu, aby dla każdego rozmiaru wyznaczał liczbę *maksymalnych* pustych prostokątów, czyli takich, które nie są zawarte w żadnym większym pustym prostokącie.

Tomasz IDZIASZEK



Rys. 3. Rozważając pole $(8, 8)$, usuwamy element $s[3]$ ze stosu. Musimy wtedy policzyć $4 \cdot 6$ prostokątów mających prawy dolny róg na polu $(8, 7)$, a lewy górny w ciemno zacienionym obszarze. Jest tam $4 - x$ prostokątów rozmiaru $y \times x$ dla $4 \leqslant y \leqslant 7$ i $1 \leqslant x \leqslant 3$.

Następnie usuwamy element $s[2]$ ze stosu, co powoduje zliczenie 15 prostokątów z jasno zacienionego obszaru ($6 - x$ prostokątów rozmiaru $3 \times x$ dla $1 \leqslant x \leqslant 5$).

Rozwiązanie zadania F 892.
Rozpatrzmy warunki równowagi k-tego naczynia. Z prawa Archimedesa wynika, że jeżeli wyjmiemy (w myśli) wszystkie naczynia pływające w k-tym naczyniu i dolejemy do niego tyle cieczy, aby zachować jej poziom wyjściowy, to siły działające na dno i ścianki naczynia ze strony cieczy w której ono pływa nie ulegną zmianie. Na k-te naczynie działa siła ciężkości kmg (km – masa tego naczynia), ciężar nalanej do niego wody $\rho kShg$ (kS – pole powierzchni dna, h – wysokość poziomu wody w tym naczyniu) i siła wyporu $\rho g V_z = \rho g k S h_z$, gdzie V_z – objętość zanurzonej części naczynia, h_z – głębokość zanurzenia.

Z warunku równowagi dla tego naczynia mamy $km = \rho kS\Delta h$, gdzie $\Delta h = h_z - h$ – różnica poziomów wody w rozpatrywanym naczyniu i w naczyniu w którym rozpatrywane naczynie pływa. Stąd $\Delta h = m/(\rho S)$. Wynika z tego, że różnica poziomów wody w dwóch dowolnych, sąsiednich naczyniach jest dla wszystkich naczyń taka sama.
Przyjmijmy, że poziom wody w naczyniu stojącym na stole wynosi H.
W następnym naczyniu wynosi on $H - \Delta h$, w kolejnym $H - 2\Delta h$ itd.
Całkowita objętość wody w naczyniach wynosi

$$NSH - (N-1)S\Delta h - \\ - (N-2)S\Delta h - \ldots - S\Delta h = \\ = NSH - S\Delta h(1 + 2 + \ldots + (N-1)) = \\ = NSH - S\Delta h((N-1)N/2) = \\ = NSH - (m(N-1)N/(2\rho)),$$

a stąd całkowita masa wody

$$M = NSH\rho - (m(N-1)N/2)$$

i ostatecznie

$$H = M/(N\rho S) + m(N-1)/(2\rho S).$$