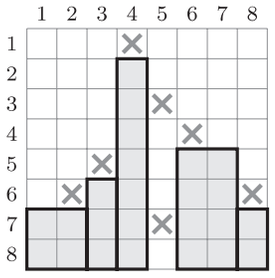
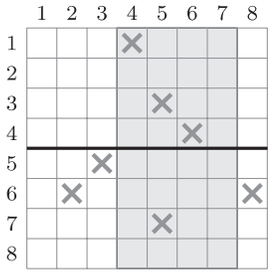
Informatyczny kącik olimpijski (88): Wykładzina



Rys. 1. Prawie pusty prostokąt dla pola $(8,5)$ zawiera zabronione pole $(7,5)$ ale nie zawiera zabronionego pola $(3,5)$, więc ma wysokość od $h_1 = 2$ do $h_2 = 5$. Kandydaci znajdowani przez algorytm mają rozmiary: 5×2, 4×4, 3×5 oraz 2×8.



Rys. 2. Wysokości prostokątów dla $K = 2$ i przedziału kolumn $i = 4$, $j = 7$ są następujące:

k	0	1	2
$r_1[k,i,j]$	0	1	3
$r_2[k,i,j]$	2	4	4

W zeszłym miesiącu zajmowaliśmy się uogólnieniem następującego zadania: dla danego kwadratu rozmiaru $n \times n$ podzielonego na n^2 pól, z których niektóre były zabronione, należało znaleźć prostokąt o największym polu, który nie zawierał żadnego zabronionego pola. W tym numerze rozważymy jeszcze inną wariację tego zadania, a mianowicie będziemy szukać największych prostokątów, które zawierają co najwyżej K zabronionych pól (nazwiemy je prostokątami *prawie pustymi*).

Na początek rozważymy przypadek $K = 1$, który był treścią zadania *Wykładzina* z finału Potyczek Algorytmicznych 2014. Zakładamy, że Czytelnicy są zaznajomieni z algorytmem szukania największego pustego prostokąta opisanym w poprzednim numerze (a pomocna może być również znajomość technik opisanych w kąciku z numeru 12/2013). Przypomnijmy, że w tamtym algorytmie ustalaliśmy wiersz i zawierający dolny bok szukanego pustego prostokąta i rozważaliśmy kolejne kolumny j, trzymając na stosie zbiór elementów opisujących obszar, w którym może znajdować się lewy górny róg pustego prostokąta o prawym dolnym rogu w polu (i,j).

Ustalmy teraz, że szukamy prawie pustego prostokąta o dolnym boku w wierszu i i zawierającego jedno zabronione pole w kolumnie j. Zatem prostokąt taki będzie miał wysokość między $h_1 = d[i,j] + 1$ a $h_2 = h_1 + d[i - h_1, j]$, gdzie $d[i,j]$ to liczba niezabronionych pól leżących na i powyżej pola (i,j) do pierwszego zabronionego pola (rys. 1). Załóżmy też, że obliczyliśmy stos dla obszaru obejmującego kolumny $1, 2, \ldots, j-1$ oraz symetryczny stos dla kolumn $n, n-1, \ldots, j+1$, a także usunęliśmy z tych stosów elementy wyższe niż h_2 (zostawiając co najwyżej po jednym elemencie wyższym niż h_2). Wszystkich kandydatów na prawie pusty prostokąt możemy znaleźć podczas usuwania z tych stosów elementów o wysokościach pomiędzy h_2 a h_1.

Aby uzyskać optymalną złożoność, wykonujemy jedynie dwa przebiegi oryginalnego algorytmu (jeden od lewej do prawej, a drugi od prawej do lewej), spamiętując dla każdej kolumny j listę prostokątów usuwanych ze stosu podczas przetwarzania tej kolumny. Następnie kandydatów na prawie puste prostokąty dla kolumny j uzyskujemy, przeglądając te listy. Dla ustalonego wiersza i praca algorytmu jest ograniczona przez $O(n)$ (tylko tyle elementów będzie na stosie), zatem cały algorytm działa w czasie $O(n^2)$.

Zadanie to ma również ciekawe rozwiązanie randomizowane, w którym algorytm szukania największego pustego prostokąta wykorzystujemy bardziej bezpośrednio. Wykonujemy f iteracji; w każdej z nich każde zabronione pole zamieniamy, niezależnie z prawdopodobieństwem p, na pole niezabronione. Następnie szukamy największego pustego prostokąta. Zauważmy, że ustalony maksymalny prawie pusty prostokąt, zawiera jedno zabronione pole oraz opiera się na trzech zabronionych polach (nie na czterech, bo ustalamy również dolny bok tego prostokąta). Zatem jeśli zamienimy pole w środku oraz nie zamienimy pól na bokach (co wydarzy się z prawdopodobieństwem $p(1-p)^3$), to ten prawie pusty prostokąt zostanie znaleziony przez algorytm szukania największego pustego prostokąta. Prawdopodobieństwo znalezienia jest największe dla $p = \frac{1}{4}$ i wynosi około $\frac{1}{10}$. Zatem po $f = 40$ iteracjach prawdopodobieństwo znalezienia ustalonego (a zatem też tego o największym polu) prawie pustego prostokąta wynosi $1 - \left(\frac{9}{10}\right)^f \approx 0{,}98$.

Zauważmy, że nasz algorytm może znajdować również prostokąty, które zawierają więcej niż jedno zabronione pole. Łatwo sobie z tym poradzimy, sprawdzając ile zabronionych pól zawiera każdy znaleziony kandydat (jak to zrobić w czasie stałym zostawimy jako nietrudne zadanie dla Czytelników, wymagające wykorzystania tzw. dwuwymiarowych sum prefiksowych). Ostateczna złożoność czasowa to $O(fn^2)$. Dla pełności dodajmy, że rozwiązanie randomizowane łatwo uogólnić na przypadek $K > 1$, ale nie będzie to zbyt praktyczne, gdyż f rośnie wtedy wykładniczo względem K.

Aby rozwiązać ogólny przypadek $K \geqslant 1$, użyjemy metody „dziel i zwyciężaj”. Podzielimy kwadrat poziomą prostą na dwie równe części, w których rekurencyjnie znajdziemy największe prawie puste prostokąty w całości zawarte w tych częściach. Pozostaną do rozważenia prawie puste prostokąty przecinające prostą (rys. 2). Dla każdego $k \leqslant K$ oraz dla każdej pary kolumn $1 \leqslant i \leqslant j \leqslant n$ znajdźmy wysokość $r_1[k,i,j]$ maksymalnego prostokąta znajdującego się w górnej części, ograniczonego prostą oraz kolumnami i, j i zawierającego k zabronionych pól – nie jest trudno zrobić to w czasie $O(kn^2)$. Następnie zróbmy to samo dla dolnej części, dostając wartości $r_2[k,i,j]$. Maksymalny prawie pusty prostokąt przecinający prostą ma rozmiar $\max_{i,j}(j+1-i) \cdot (\max_{k_1+k_2 \leqslant K} r_1[k_1,i,j] + r_2[k_2,i,j])$.

Jeśli oznaczymy przez $T(n)$ czas działania algorytmu dla kwadratu o boku n, to dostajemy następującą rekurencję: $T(n) = 4T(n/2) + 3O(kn^2)$ (najpierw dzielimy kwadrat na dwie prostokątne części, a potem każdą z nich pionową prostą na dwa kwadraty o bokach $n/2$). Rozwiązanie tej rekurencji (a zatem złożoność czasowa całego algorytmu) to $T(n) = O(kn^2 \log n)$.

Tomasz IDZIASZEK