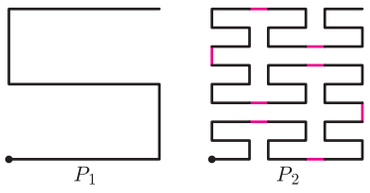
Informatyczny kącik olimpijski (43): Ploter



Rys. 1. Krzywa Peano rzędu i powstaje z połączenia dziewięciu krzywych rzędu $i-1$.

W celu zaznajomienia się ze smoczą krzywą, zachęcamy do przeczytania artykułu *Fraktalny świat papierowej tasiemki*.

Napisanie programu, który generuje rysunek fraktala, idealnie nadaje się na zadanie dla początkującego programisty. Proste reguły prowadzące do powstania skomplikowanych wzorów powodują, że przy stosunkowo niewielkim wysiłku programistycznym można osiągnąć całkiem ambitne efekty wizualne. Ponadto samopodobieństwo fraktali pozwala ćwiczyć jedną z podstawowych koncepcji programistycznych – rekurencję.

Nic więc dziwnego, że autorzy zadań na konkursach programistycznych chętnie sięgają po inspirację do świata fraktali. Jednym z popularnych typów zadań jest generowanie krzywych wypełniających płaszczyznę, np. krzywej Peano (zob. rysunek), Hilberta lub ich wariacji. Rozmiar takich krzywych rośnie wykładniczo względem rzędu krzywej, zatem zadanie zwykle jest formułowane następująco: dla krzywej rzędu n i liczby k, stwierdzić, jakie współrzędne ma punkt płaszczyzny, leżący w odległości k od początku krzywej. Oczekuje się, że program będzie działać w czasie $O(n)$.

Najczęściej takie zadania nie są koncepcyjnie trudne. Przykładowo, gdybyśmy chcieli rozwiązać taki problem dla smoczej krzywej S_n, należy rozpatrzyć dwa przypadki:

(a) $k < 2^{n-1}$ i wtedy rekurencyjnie znajdujemy rozwiązanie dla krzywej S_{n-1}, albo
(b) $k \geqslant 2^{n-1}$ i wtedy znajdujemy punkt na krzywej S_{n-1} w odległości $k - 2^{n-1}$ od początku krzywej i odpowiednio przesuwamy go na płaszczyźnie.

Dla krzywej Peano rozwiązanie jest analogiczne, z tym że trzeba będzie rozpatrzyć nie dwa, ale 9 przypadków.

Można jednak zadanie sformułować na odwrót: mając dany punkt płaszczyzny (x, y) leżący na krzywej, pytamy się, w jakiej odległości od początku krzywej się on znajduje. Takie zadanie dla krzywej Peano pojawiło się m.in. na światowych finałach konkursu ACM ICPC w roku 2003 (zadanie pt. *Riding the Bus*). Jego rozwiązanie także jest łatwe: krzywa Peano P_n rzędu n mieści się na kwadratowej siatce rozmiaru $3^n - 1 \times 3^n - 1$, a jej długość to $9^n - 1$. Patrząc na wartości $\lfloor x/3^{n-1} \rfloor$ i $\lfloor y/3^{n-1} \rfloor$, jesteśmy w stanie stwierdzić, do której z dziewięciu mniejszych krzywych P_{n-1} należy punkt (x, y). Jeśli należy do i-tej krzywej (w kolejności ich występowania w P_n), to wynikiem będzie suma $i \cdot 9^{n-1}$ oraz rozwiązania podzadania dla krzywej P_{n-1} i punktu $(x \bmod 3^{n-1}, y \bmod 3^{n-1})$, odpowiednio obróconego. Taki algorytm działa w czasie $O(n)$.

Odwrotny problem dla smoczej krzywej został postawiony przed uczestnikami tegorocznych Potyczek Algorytmicznych w zadaniu *Ploter*. Na pierwszy rzut oka wydaje się, że pomysł, którego użyliśmy dla krzywej Peano, tu się zupełnie nie sprawdzi. Brakuje nam bowiem istotnego składnika: jak dla punktu na krzywej S_n sprawdzić, do której z dwóch mniejszych krzywych S_{n-1} on należy. Spróbujmy więc trochę inaczej: wykonajmy procedurę rekurencyjnie dla *obu* mniejszych krzywych. Takie rozwiązanie działać będzie, oczywiście, w czasie $O(2^n)$. Ale zauważmy, że w wielu przypadkach, jeśli szukamy punktu na niewłaściwej krzywej mniejszego rzędu, to po kilku zejściach rekurencyjnych nasz punkt na tyle istotnie oddali się od aktualnie rozważanego kawałka krzywej, że będziemy mogli to łatwo wykryć i odciąć przeszukiwanie.

Zobaczmy, jak ta idea sprawdzi się w praktyce. Niech (x_n, y_n) oznacza ostatni punkt na krzywej rzędu n. Łatwo pokazać obliczającą go rekurencję ($n \geqslant 2$):

$$x_1 = y_1 = 1, \qquad x_n = x_{n-1} - y_{n-1}, \qquad y_n = x_{n-1} + y_{n-1}.$$

Niech $d(x, y, n)$ będzie funkcją, która zwraca zbiór odległości, w jakich punkt (x, y) leży od początku krzywej S_n (dany punkt może leżeć w co najwyżej dwóch odległościach od początku krzywej). Funkcja ta wykona dwa wywołania rekurencyjne. Jej wynikiem będzie suma dwóch zbiorów:

$$d(x, y, n) = d(x, y, n-1) \cup \\ \cup \{2^n - k \mid k \in d(y_n - y, x - x_n, n-1)\}.$$

Dla S_1 mamy $d(x, y, 1) = \{x + y\}$ dla $0 \leqslant y \leqslant x \leqslant 1$, w przeciwnym wypadku $d(x, y, 1) = \emptyset$.

A kiedy możemy odciąć przeszukiwanie? Nie jesteśmy w stanie łatwo sprawdzić, *czy* dany punkt występuje na krzywej S_n, ale dla wielu punktów możemy powiedzieć, że na pewno na niej nie występują. Rozważmy bowiem najmniejszy prostokąt R_n o bokach równoległych do osi układu współrzędnych, który zawiera wszystkie punkty krzywej S_n. Ilekroć funkcja $d(x, y, n)$ zostanie wywołana dla punktu (x, y), który leży poza prostokątem R_n, możemy natychmiast zwrócić $\emptyset$. Niech r_n, t_n, l_n, b_n oznaczają odległości punktu $(0, 0)$ od krawędzi prostokąta R_n (odpowiednio prawej, górnej, lewej i dolnej). Możemy je obliczyć rekurencyjnie ($n \geqslant 2$):

$$\begin{aligned} r_1 &= t_1 = 1, \\ l_1 &= b_1 = 0, \\ r_n &= \max(r_{n-1}, t_{n-1} + x_n), \\ t_n &= \max(t_{n-1}, l_{n-1} + y_n), \\ l_n &= \max(l_{n-1}, b_{n-1} - x_n), \\ b_n &= \max(b_{n-1}, r_{n-1} - y_n). \end{aligned}$$

Pozostało pytanie, na ile powyższa optymalizacja wpływa na czas działania algorytmu. Narysujmy krzywą S_n i zaznaczmy wszystkie prostokąty otaczające 2^{n-m} krzywych S_m, które składają się na krzywą S_n. Oszacujmy, ile maksymalnie narysowanych prostokątów może zawierać dany punkt (x, y); ta liczba będzie oznaczać, ile razy wywołanie $d(\cdot, \cdot, m)$ nie zostanie odcięte. Wynika z tego, że jeśli liczba prostokątów będzie rzędu $O(1)$, to na każdym poziomie rekurencji będziemy mieli stałą liczbę wywołań, a zatem cały algorytm będzie działał w czasie $O(n)$.

Można udowodnić przez indukcję, że x_m i y_m są podzielne przez $M = 2^{\lfloor m/2 \rfloor}$, zatem każda z krzywych S_m zaczyna się w jednym z punktów zbioru $Z_M = \{(Mi, Mj) \mid i, j \in \mathbb{Z}\}$. Można też pokazać, że $r_m + l_m, t_m + b_m < 2^{\lfloor m/2 \rfloor + 1}$, tak więc każdy z narysowanych prostokątów jest zawarty w kwadracie o boku $2M - 1$. Zatem prostokąty zawierające punkt (x, y) pokrywają w sumie co najwyżej 16 punktów ze zbioru Z_M. W każdym z tych punktów mogą zaczynać się co najwyżej cztery krzywe S_m, zatem (x, y) należy do nie więcej niż $16 \cdot 4$ prostokątów. Zatem nasz algorytm istotnie działa w czasie liniowym!

Tomasz IDZIASZEK