Sortowanie biegunowe a dualizacja

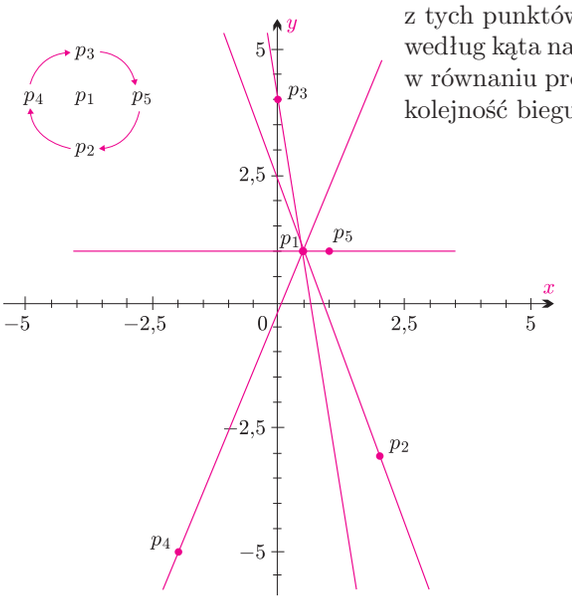
Grzegorz JAKACKI[*]

Artykuł powstał na podstawie tekstu autora w książce *W poszukiwaniu wyzwań. Wybór zadań z konkursów programistycznych Uniwersytetu Warszawskiego.*

Przykłady problemów, w których należy wykonać sortowanie względem każdego punktu, można znaleźć np. w *Delcie* 4/2013 (zliczanie czworokątów wypukłych w Informatycznym Kąciku Olimpijskim) i 5/2013 (problem 3PointsOnLine w artykule o problemach 3sum-trudnych).

Jedną z najczęściej wykonywanych operacji w geometrii obliczeniowej jest sortowanie biegunowe (zwane też kątowym) zbioru n punktów na płaszczyźnie względem wybranego punktu. Innymi słowy, chcemy uporządkować punkty $p_1, \ldots, p_n$ według współrzędnej kątowej w układzie biegunowym zaczepionym w wybranym punkcie p. Stosując jeden z efektywnych algorytmów sortowania, operację tę można zrealizować w czasie $O(n \log n)$. W wielu zastosowaniach musimy jednak pójść o krok dalej i wyznaczyć uporządkowanie biegunowe punktów $p_1, \ldots, p_n$ najpierw w układzie biegunowym zaczepionym w p_1, potem zaczepionym w p_2 itd. Mamy zatem do wykonania n sortowań, co łącznie możemy zrealizować w czasie $O(n^2 \log n)$. Okazuje się jednak, że istnieje algorytm pozwalający wyznaczyć wszystkie potrzebne porządki biegunowe w czasie $O(n^2)$. Jest on ciekawy, jednak dosyć trudny – w tym artykule przedstawimy szkic tego algorytmu.

Punkty i proste. Pomyślmy o zbiorze punktów $p_1, \ldots, p_n$ na płaszczyźnie. Wybierzmy jeden z nich, na przykład p_1. Poprowadźmy proste przez p_1 tak, by połączyć go ze wszystkimi pozostałymi punktami $p_2, \ldots, p_n$ (rys. 1). Każdemu z tych punktów odpowiada jedna prosta. Gdybyśmy umieli posortować te proste według kąta nachylenia lub, inaczej mówiąc, według współczynnika kierunkowego a w równaniu prostej $y = ax + b$, to takie uporządkowanie pozwoliłoby nam uzyskać kolejność biegunową punktów $p_2, \ldots, p_n$ w układzie o środku w p_1.



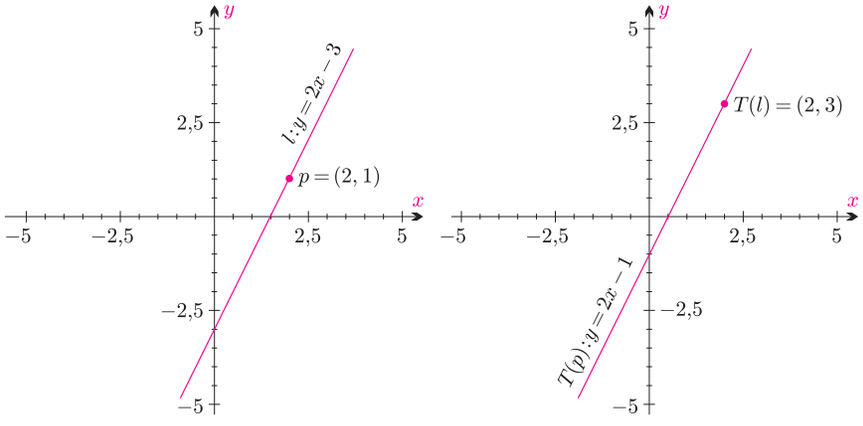
Rys. 1. Proste z p_1 oraz porządek biegunowy punktów względem p_1.

Spójrzmy teraz na wszystkie proste przechodzące przez pary punktów ze zbioru $p_1, \ldots, p_n$. Jest ich nie więcej niż $\frac{n(n-1)}{2}$. Jeśli uda nam się skonstruować algorytm, który dla każdego z punktów posortuje $n-1$ przechodzących przez niego prostych w sumarycznym czasie lepszym niż $O(n^2 \log n)$, to dostaniemy lepsze rozwiązanie całego zadania. Algorytm nie może jednak działać szybciej niż $O(n^2)$, bo taka jest liczba prostych, które ma posortować. Okazuje się, że algorytm działający w czasie kwadratowym istnieje i opiera się na bardzo ciekawej konstrukcji myślowej zwanej dualizacją.

Dualizacja. Wyobraźmy sobie przekształcenie geometryczne T, które punktowi p przyporządkowuje prostą $T(p)$, a prostej l punkt $T(l)$ w następujący sposób:

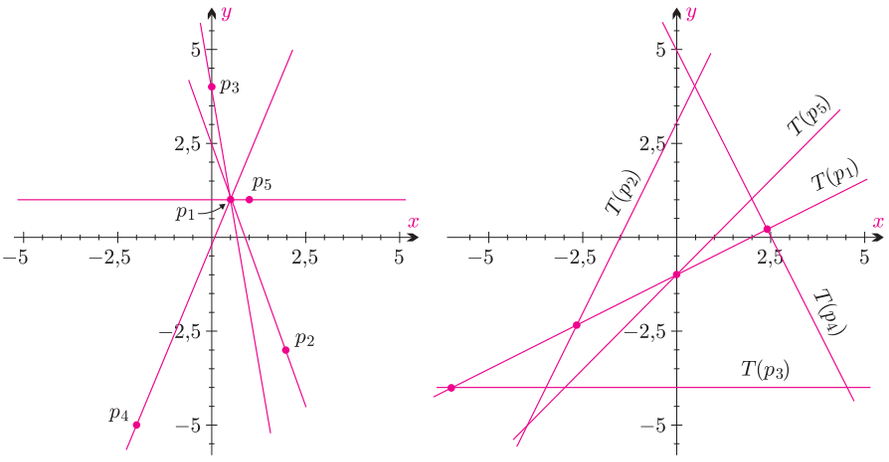
$$p = (a, b) \mapsto T(p) : y = ax - b, \qquad l : y = kx + d \mapsto T(l) = (k, -d),$$

czyli punktowi o współrzędnych (a, b) odpowiada prosta o równaniu $y = ax - b$, a prostej o równaniu $y = kx + d$ odpowiada punkt o współrzędnych $(k, -d)$. Punkty i proste oraz ich obrazy leżą w płaszczyźnie $\mathbb{R}^2$. Płaszczyznę, z której bierzemy argumenty przekształcenia T, nazywamy przestrzenią pierwotną, natomiast płaszczyznę, w której znajdują się jego wartości – przestrzenią dualną. Opisane przekształcenie ma bardzo ciekawą właściwość: punkt p leży na prostej l wtedy i tylko wtedy, gdy prosta $T(p)$ przechodzi przez punkt $T(l)$.



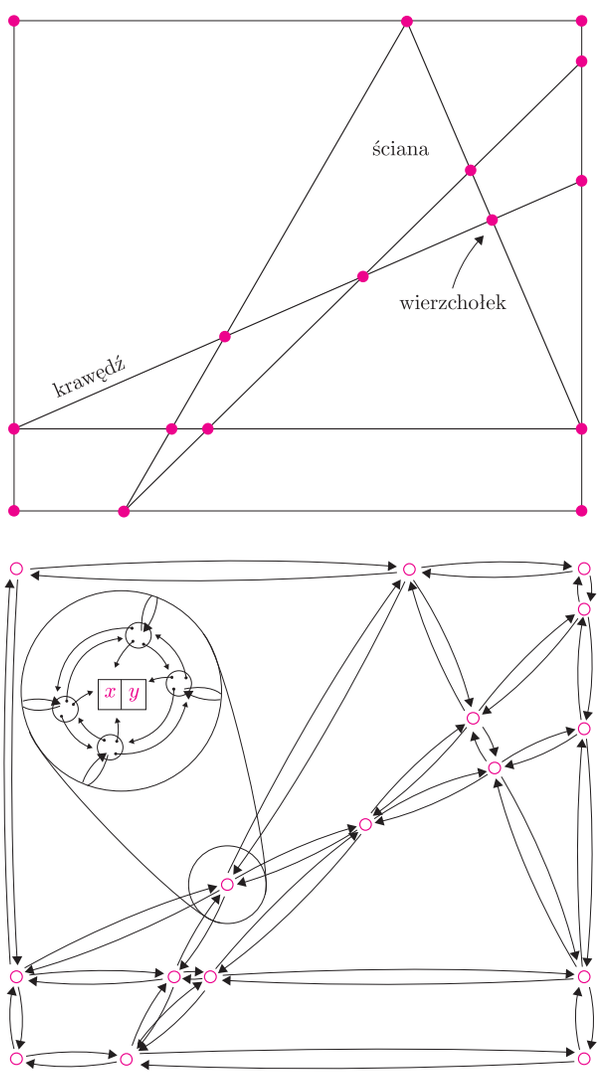
Rys. 2. Przestrzeń pierwotna (po lewej) oraz odpowiadająca jej przestrzeń dualna (po prawej).

[*]firma Codility



Rys. 3. Wszystkie proste z rysunku 1.

Zbiór punktów $p_1, \ldots, p_n$ oraz prostych łączących je w pary możemy odwzorować w zbiór prostych $T(p_1), \ldots, T(p_n)$. W przestrzeni dualnej proste, które łączą np. p_1 z pozostałymi punktami, odpowiadają punktom przecięć prostej $T(p_1)$ z prostymi $T(p_2), \ldots, T(p_n)$. Od tej własności jest jednak mały wyjątek. Gdy pewne punkty p_i i p_j mają tę samą pierwszą współrzędną, to w przestrzeni dualnej proste $T(p_j)$ oraz $T(p_i)$ będą równoległe. Taką sytuacją zajmiemy się jednak za chwilę.



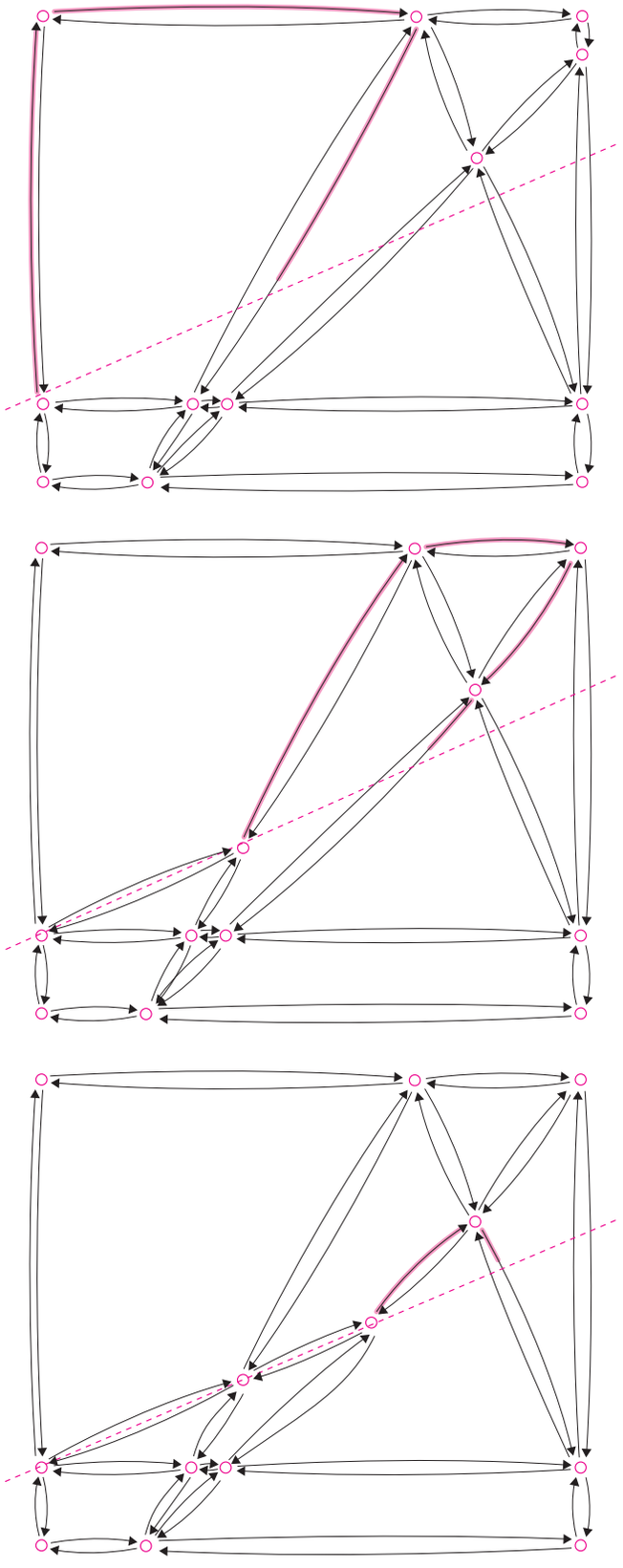
Rys. 4. Struktura danych u dołu reprezentuje układ odcinków przedstawiony u góry. Został on zbudowany przez otoczenie prostokątną ramką prostych z rysunku 3.

Dotarliśmy zatem do następującego problemu. Danych jest n prostych na płaszczyźnie. Dla każdej prostej należy obliczyć, w jakiej kolejności (względem pierwszej współrzędnej) będą się z nią przecinały pozostałe proste. Z konstrukcji przestrzeni dualnej wynika, że żadna prosta nie będzie pionowa. Znając porządek przecięć dla prostej $T(p)$ w przestrzeni dualnej, możemy odtworzyć porządek prostych przechodzących przez punkt p w przestrzeni pierwotnej, a z niego porządek biegunowy wszystkich punktów względem p.

Jak wykonać ostatni z tych kroków? Wystarczy dwa razy przejrzeć proste w niemalejącej kolejności współczynników kierunkowych i przy pierwszym przejściu wypisać tylko punkty leżące na lewo od p_1 (tj. mające mniejszą pierwszą współrzędną), a przy drugim przejściu – wypisać pozostałe. Musimy jeszcze osobno rozważyć wszystkie punkty mające tę samą pierwszą współrzędną co p_1, jednak bardzo łatwo możemy umieścić je w odpowiednim miejscu w porządku biegunowym wokół p_1. Ten proces powtarzamy dla każdego punktu wśród $p_1, \ldots, p_n$.

Wyznaczanie przecięć prostych. Spójrzmy na rysunek 4. Przedstawia on strukturę danych, która reprezentuje układ odcinków stykających się jedynie końcami. Co więcej, każdy koniec odcinka jest jednocześnie końcem jakiegoś innego odcinka. Każdy punkt styku reprezentujemy za pomocą listy cyklicznej. Element listy odpowiada końcowi odcinka i oprócz wskaźników do następnika i poprzednika zawiera także wskaźnik do elementu innej listy, który reprezentuje drugi koniec tego samego odcinka. Ponadto elementy listy reprezentującej punkt styku zawierają jego współrzędne. Kolejność elementów na liście jest istotna i musi odpowiadać kolejności występowania odcinków na płaszczyźnie, na przykład być zgodna z kierunkiem ruchu wskazówek zegara.

Opisana struktura jest wariantem tzw. *doubly-connected edge list*. Pozwala ona szybko dodawać nowe odcinki



Rys. 5. Fazy dodawania prostej do struktury danych.

do istniejącego układu. Zauważmy też, że dzięki wskaźnikom do sąsiedniego odcinka oraz dzięki odpowiedniemu uporządkowaniu elementów na listach cyklicznych, odpowiadających punktom styku, możemy w prosty sposób obchodzić dookoła obszary (ściany) wyznaczone przez układ odcinków. Rysunek 5 przedstawia kolejne fazy dodawania odcinków prostej do układu.

Operacje na strukturze są dosyć intuicyjne. Nas interesuje odpowiedź na pytanie, jaki jest ich czas działania względem n, bo od tego zależy złożoność algorytmu. Widać, że nowo powstająca ściana może mieć bardzo wiele krawędzi, nawet liniowo wiele względem n. Okazuje się jednak, że sumaryczna liczba krawędzi, które trzeba odwiedzić przy dodawaniu odcinków jednej prostej, jest również rzędu $O(n)$. Dowód tego faktu pominiemy, choć nie jest bardzo trudny. Opiera się on na obserwacji, że odwiedzane krawędzie można przyporządkować do punktów przecięć przetwarzanej prostej w ten sposób, by na jeden punkt przypadało co najwyżej 10 krawędzi. Zainteresowani dowodem powinni szukać hasła *twierdzenie strefowe* lub *zone theorem*.

Powyższy fakt jest kluczowy dla analizy złożoności czasowej algorytmu, bo wynika z niego, że strukturę daje się zbudować w czasie $O(n^2)$. Ponadto przejście po kolejnych odcinkach jednej prostej można zrealizować w czasie $O(n)$. Zauważmy, że jeżeli zbiór punktów $p_1, \ldots, p_n$ na płaszczyźnie przekształcimy za pomocą dualizacji na zbiór prostych $T(p_1), \ldots, T(p_n)$, a te proste otoczymy ramką, to otrzymamy układ odcinków podobny do tego z rysunku 4.

Prostokątną ramkę, obejmującą wszystkie punkty przecięcia układu n prostych, możemy obliczyć następująco. Sortujemy (w czasie $O(n \log n)$) wszystkie proste względem ich współczynników kierunkowych i bierzemy pod uwagę tylko punkty przecięć prostych sąsiadujących w tym porządku oraz punkt przecięcia pierwszej i ostatniej prostej. Wszystkie pozostałe punkty przecięć znajdują się wewnątrz otoczki wypukłej tych punktów.

Dalej, za pomocą opisanego powyżej algorytmu konstrukcji struktury *doubly-connected edge list* jesteśmy w stanie w czasie $O(n^2)$ wyznaczyć kolejność punktów przecięć dla wszystkich prostych. Jeśli teraz dla dowolnego punktu z $p_1, \ldots, p_n$ (np. p_1) zapragniemy wyznaczyć porządek biegunowy pozostałych punktów $p_2, \ldots, p_n$, to wystarczy odczytać porządek przecięć prostej $T(p_1)$ z pozostałymi prostymi. Numery prostych w tym porządku wyznaczą kolejność punktów $p_2, \ldots, p_n$ w porządku biegunowym dookoła p_1, o czym pisaliśmy powyżej.

Za pomocą dualizacji udało nam się przyspieszyć algorytm sortowania biegunowego zbioru n punktów z czasu $O(n^2 \log n)$ do czasu $O(n^2)$. Usprawnienie to nie ma dużego znaczenia w przypadku np. zadań opisywanych w Informatycznym Kąciku Olimpijskim, ponieważ jego implementacja jest bardzo skomplikowana, a przyspieszenia raczej nie dałoby się zauważyć na zbiorach danych o stosunkowo niewielkim rozmiarze. Przyspieszanie takich algorytmów ma jednak rację bytu w poważniejszych zastosowaniach geometrii obliczeniowej, takich jak generowanie grafiki lub projektowanie układów scalonych.