Dawid Pawliczek
Lista 3, Zadanie 4

Treść

Dana jest kolekcja n prostych $l_i : \ y = a_i x + b_i$ $(i = 1, \ldots, n)$ na płaszczyźnie, przy czym żadne trzy nie przecinają się w jednym punkcie. Mówimy, że prosta l_i jest *widoczna* z punktu $(0, +\infty)$, jeśli istnieje punkt $q \in l_i$ taki, że odcinek $(0, +\infty)\, q$ nie przecina żadnej innej prostej (oprócz, ewentualnie, w punkcie q). Zaprojektować algorytm znajdujący wszystkie widoczne proste.

Geometria patrzenia „z góry” sprowadza problem do wyznaczenia *górnej obwiedni* układu prostych.

Rozwiązanie

1. Wstępne sortowanie

- Posortuj proste rosnąco według współczynnika kierunkowego a.
- Jeżeli dwie proste mają ten sam a, pozostaw tylko tę o większym b (druga leży *niżej*, więc nigdy nie będzie widoczna).

2. Budowa obwiedni stosem

Przechodzimy po posortowanej liście. Stos S przechowuje zawsze aktualny zestaw widocznych prostych w kolejności rosnących a.

Algorithm 1 VisibleLines

Require: lista L posortowana wg a (po kroku 1)
Ensure: zbiór linii widocznych z $(0, +\infty)$

$S \leftarrow$ pusty stos
for all l w L **do** ▷ $l : y = ax + b$
  while $|S| \geq 2$ **do**
    niech $l_1 = S$.top(), $l_2 =$ drugi od góry w S
    wyznacz x-owe współrzędne punktów przecięcia $l_1 \cap l_2$ oraz $l_2 \cap l$
    if $\mathrm{x}(l_1 \cap l_2) \ \geq \ \mathrm{x}(l_2 \cap l)$ **then**
      S.pop() ▷ l_1 zasłonięty — usuń
    elsebreak
    end if
  end while
  S.push(l)
end for
return S

Komentarz. Jeśli punkt przecięcia nowej prostej l z poprzednikiem l_2 leży *na lewo* od przecięcia l_2 z l_1, to l_1 znajduje się całkowicie pod obwiednią i znika z widoku; pętla `while` usuwa kolejne takie linie aż warunek przestanie być spełniony.

Poprawność

1. **(Usuwanie nadmiarowych linii o tym samym a).** Jeśli $a_i = a_j$ i $b_i < b_j$, linia l_i leży pod l_j dla każdego x; z punktu $(0, +\infty)$ nigdy nie może być zatem widoczna. Jej odrzucenie nie wpływa na wynik.

2. **(Monotoniczność przecięć).** Niech stos zawiera (od dołu) linie $l_0, l_1, \dots, l_t$ w kolejności rosnących współczynników a. Łatwo sprawdzić, że

$$x\big(l_{t-1} \cap l_t\big) \ < \ x\big(l_t \cap l\big)$$

oznacza, iż l_t i l przecinają się *na prawo* od przecięcia l_{t-1} z l_t. Odcinek $l_t l$ wypukłej obwiedni pozostaje więc odkryty; l_t musi zostać w odpowiedzi, a pętla `while` się kończy.

3. **(Warunek usuwania).** Jeśli natomiast $x(l_{t-1} \cap l_t) \geq x(l_t \cap l)$, cała prosta l_t leży pod odcinkiem obwiedni łączącym l_{t-1} z l. Z punktu $(0, +\infty)$ l_t jest całkowicie zasłonięta, więc jej usunięcie zachowuje zbiór linii widocznych. Utrwalenie tej właściwości dla kolejnych linii zapewnia, że stos zawiera wyłącznie krawędzie końcowej obwiedni.

4. **(Zużycie każdej linii najwyżej raz).** Każda linia trafia na stos raz i z niego schodzi najwyżej raz, nie może więc zostać „nadmiernie" odsiana.

Po przetworzeniu całego wejścia stos zawiera dokładnie te i tylko te proste, które tworzą górną obwiednię, a więc są widoczne z punktu $(0, +\infty)$. □

Złożoność

- Sortowanie po współczynniku a: $O(n \log n)$.
- Pętla główna: każde wstawienie lub usunięcie to koszt $O(1)$, a każda linia może być wypchnięta ze stosu najwyżej raz – łącznie $O(n)$.

$$\boxed{T(n) = O(n \log n)}, \qquad \boxed{\text{pamięć } O(n)}$$