1. ANALIZA ALGORYTMÓW — PRZYPOMNIENIE

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Wprowadzenie do algorytmów, rozdziały 1-4
Wydawnictwa naukowo-Techniczne (2004)

Jak mierzyć *efektywność* algorytmu?

- Przyjęcie modelu obliczeniowego (np. RAM).
- Ustalenie rozmiaru wejścia.
- Ustalenie liczby operacji (instrukcji) podstawowych;
  - ↳ zdefiniowanie operacji podstawowych.
- Ustalenie minimalnej potrzebnej liczby komórek pamięci;
  - ↳ odniesienie do zastosowanych struktur danych.

Rozmiar wejścia

W formalnym modelu (np. maszyna Turinga) ***rozmiar wejścia*** zdefiniowany jest jako długość napisu wejściowego nad skończonym alfabetem.

- ▶ Dla jednej zmiennej $n \in \mathbb{N}$ przyjmuje się liczbę bitów $\lfloor \log_2 n \rfloor + 1$.
- ▶ Dla ustalonej liczby k zmiennych $n_1, \ldots, n_k$ przyjmuje się ich sumę długości, czyli $\sum_1^k \lfloor (\log_2 n_i \rfloor + 1)$;
  - ↳ np. w algorytmie Euklidesa rozmiar wejścia to $\lfloor \log_2 m \rfloor + \lfloor \log_2 n \rfloor + 2$.
- ▶ Dla wektora n-elementowego przyjmujemy liczbę elementów, czyli n.
- ▶ Dla tablicy wielowymiarowej przyjmujemy największy z jej wymiarów;
  - ↳ np. dla tablicy TAB[$1\ldots100$][$1\ldots1000$] rozmiarem wejścia jest 1000.

Szacowanie tempa wzrostu

- Symbol $O(\cdot)$:
  - ↳ $h\colon \mathbb{N} \to \mathbb{R}^+$;
  - ↳ klasa $O(h) = \{g\colon \mathbb{N} \to \mathbb{R}^+$, gdzie $\exists_{c>0}\ \exists_{n_0 \in \mathbb{N}}\ \forall_{n \geq n_0}\ g(n) \leq c \cdot h(n)\}$;
  - ↳ $g \in O(h)$ oznacza, że *funkcja g rośnie nie szybciej niż funkcja h*.
  - ↳ $g \in O(h)$ czytamy: „*g jest klasy 'O' duże od h*”
    lub „*g jest rzędu 'O' duże od h*.”

Przykłady:

- ↳ $g(n) = n \in O(n^2)$;
- ↳ $g(n) = 1/n \in O(1)$;
- ↳ $g(n) = 100 \cdot n^2 + 100^{100} \cdot n + 100^{100^{100}} \in O(n^2)$;
- ↳ $g(n) = (\log n)^a \in O(n^b)$ dla każdych $a, b > 0$; np. $(\log n)^{1000} \in O(n^{0.001})$.

- Symbol $\Omega(\cdot)$:

  ↳ $f\colon \mathbb{N} \to \mathbb{R}^+$;

  ↳ klasa $\Omega(f) = \{g\colon \mathbb{N} \to \mathbb{R}^+$, gdzie $\exists_{c>0}\, \exists_{n_0\in\mathbb{N}}\, \forall_{n\geq n_0}\; g(n) \geq c\cdot f(n)\}$;

  ↳ $g \in \Omega(f)$ oznacza, że *funkcja f rośnie nie szybciej niż funkcja g*.

  ↳ $g \in \Omega(f)$ czytamy: „*g jest klasy 'Omega' duże od f*”
  lub „*g jest rzędu 'Omega' duże od f*.”

Przykłady:

↳ $g(n) = n^2 \in \Omega(n)$;

↳ $g(n) = \frac{1}{10} \in \Omega(1/n)$;

↳ $g(n) = \frac{n^2}{100} - 100^{100}\cdot n - 100^{100^{100}} \in \Omega(n^2)$;

↳ $g(n) = n^b \in \Omega((\log n)^a)$ dla każdych $a, b > 0$.

- Symbole $o(\cdot), \omega(\cdot)$ oraz $\Theta(\cdot)$.

Czasowa złożoność obliczeniowa

- Operacje podstawowe:
  - ↳ operacje arytmetyczne, logiczne oraz porównania;
  - ↳ odczytanie, zapisanie do komórki pamięci;
  - ↳ wywołanie funkcji, procedury.
- W modelu RAM koszty powyższych operacji podstawowych są jednostkowe.
- Dla ustalonych danych wejściowych I możemy (zazwyczaj) precyzyjnie wyznaczyć liczbę $\mathbf{lop}(I)$ operacji podstawowych wykonanych przez program.
- (Pesymistyczna) ***Czasowa złożoność obliczeniowa***

  $$T(n) := \max\{\mathrm{lop}(I) : I \in I(n)\},$$

  gdzie $I(n)$ jest to zbiór wszystkich danych wejściowych o rozmiarze n.
- Ze względów praktycznych szuka się jak najlepszego oszacowania tempa wzrostu funkcji T, czyli możliwie „najwolniej rosnącej” funkcji g takiej, że $T \in O(g)$.

Np. sortowanie „bąbelkowe” n liczb:

→ rozmiar danych wejściowych: n;

→ liczba operacji podstawowych $\leq \text{const} \cdot n^2$;

→ złożoność czasowa: $T(n) \in O(n^2)$ (ale i też $T(n) \in O(n^{100})$).

Twierdzenie. *Dowolny algorytm sortujący n elementów za pomocą porównań wymaga w przypadku pesymistycznym czasu rzędu $\Omega(n \log n)$.* /Cormen et al./

- ▶ Dla ustalonych danych wejściowych I możemy (zazwyczaj) precyzyjnie wyznaczyć liczbę $\mathbf{lop}(I)$ operacji podstawowych wykonanych przez program.

- ▶ (Pesymistyczna) ***Czasowa złożoność obliczeniowa***

$$T(n) := \max\{\text{lop}(I) : I \in I(n)\},$$

gdzie $I(n)$ jest to zbiór wszystkich danych wejściowych o rozmiarze n.

- ▶ Ze względów praktycznych szuka się jak najlepszego oszacowania tempa wzrostu funkcji T, czyli możliwie „najwolniej rosnącej” funkcji g takiej, że $T \in O(g)$.

Pamięciowa złożoność obliczeniowa

- W każdym momencie wykonywania programu jesteśmy w stanie sprawdzić, ile aktualnie wykorzystywanych jest komórek (jednostek) pamięci.
- Dla ustalonych danych wejściowych I możemy wyznaczyć liczbę $\mathbf{lkp}(I)$ komórek pamięci wykorzystywanych podczas działania programu.
- (Pesymistyczna) ***Pamięciowa złożoność obliczeniowa***

$$M(n) := \max\{\mathrm{lkp}(I) : I \in I(n)\}.$$

  - ↳ Nie zliczamy łącznej liczby użytych komórek pamięci, a pytamy się, jaki powinien być najmniejszy rozmiar pamięci gwarantującej wykonanie się programu.
  - ↳ W przypadku programów ze zmiennymi dynamicznymi należy ustalić maksymalny rozmiar stosu zmiennych dynamicznych.
- Szukamy jak najlepszego oszacowania tempa wzrostu funkcji M, czyli możliwie najwolniej rosnącej funkcji f takiej, że $M \in O(f)$.

Np. sortowanie „bąbelkowe” n liczb:
- → rozmiar danych wejściowych: n;
- → liczba operacji podstawowych $\leq \mathrm{const} \cdot n^2$;
- → złożoność czasowa: $T(n) \in O(n^2)$;
- → złożoność pamięciowa: $M(n) \in O(1)$.

- ► (Pesymistyczna) ***Pamięciowa złożoność obliczeniowa***

$$M(n) := \max\{\mathrm{lkp}(I) : I \in I(n)\}.$$

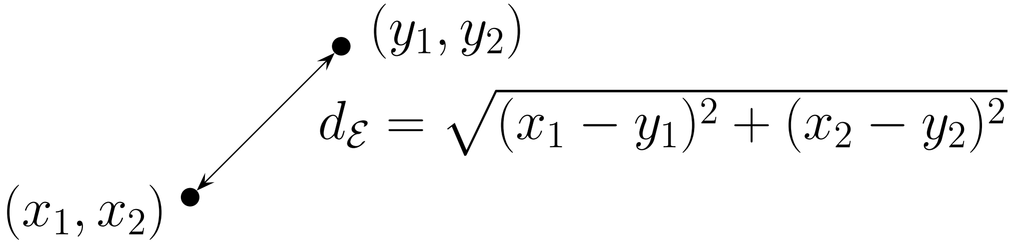
  - ↳ Nie zliczamy łącznej liczby użytych komórek pamięci, a pytamy się, jaki powinien być najmniejszy rozmiar pamięci gwarantującej wykonanie się programu.
  - ↳ W przypadku programów ze zmiennymi dynamicznymi należy ustalić maksymalny rozmiar stosu zmiennych dynamicznych.

- ► Szukamy jak najlepszego oszacowania tempa wzrostu funkcji M, czyli możliwie najwolniej rosnącej funkcji f takiej, że $M \in O(f)$.

2. ELEMENTY GEOMETRII OBLICZENIOWEJ

Dla danego zbioru X funkcję $d\colon X \times X \to \mathbb{R}^+ \cup \{0\}$ nazywamy **metryką** (odległością), jeżeli dla dowolnych $p, q, w \in X$ zachodzi:

1. $d(p, q) = 0 \iff p = q$ (w jednym miejscu jeden punkt);
2. $d(p, q) = d(q, p)$ (symetria);
3. $d(p, q) + d(q, w) \geq d(p, w)$ (nierówność trójkąta).



Przez $\mathcal{E}^d$ oznaczamy ***d*-wymiarową przestrzeń euklidesową**, czyli przestrzeń d-krotek $(x_1, \ldots, x_d)$ liczb rzeczywistych x_i, $i = 1, \ldots, d$ z metryką

$$d_{\mathcal{E}}(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

PROSTE OBIEKTY GEOMETRYCZNE

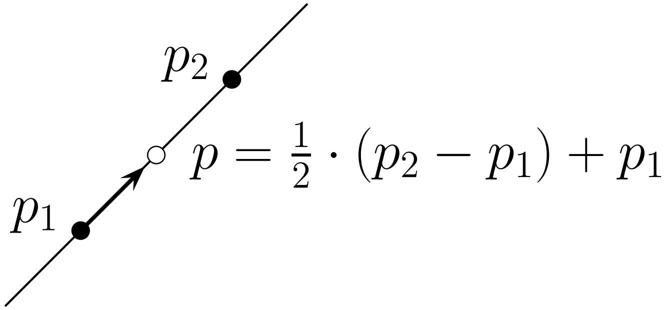
Punkt: umiejscowienie w przestrzeni.

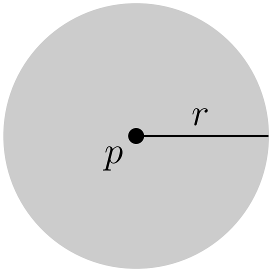
Wektor: kierunek w przestrzeni.

Współrzędne homogeniczne: punkt $p = (x_1, \dots, x_d) \in \mathcal{E}^d$ oraz wektor $\vec{v} = (v_1, \dots, v_d) \in \mathcal{E}^d$ reprezentowane są jako $(d+1)$-krotki z dołączoną 1 i – odpowiednio – 0: $(1, x_1, \dots, x_d)$, $(0, v_1, \dots, v_d)$.

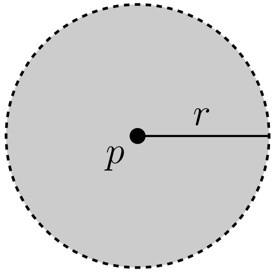
Prosta przechodząca przez punkty p_1 i p_2: zbiór wszystkich kombinacji liniowych punktów p_1 i p_2.

$$l(p_1, p_2) = \{(1-\alpha) \cdot p_1 + \alpha \cdot p_2 : \alpha \in \mathbb{R}\} = \{\alpha \cdot (p_2 - p_1) + p_1 : \alpha \in \mathbb{R}\}$$





kula $K(p,r)$

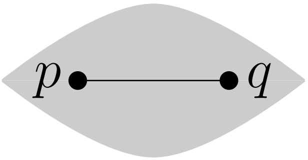


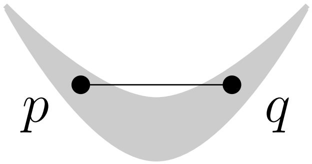
kula otwarta $K(p,r)$

Kulą $K(p,r)$ o środku w punkcie p i promieniu $r \geq 0$ nazywamy zbiór

$$K(p,r) := \{q \in X : d(p,q) \leq r\}.$$

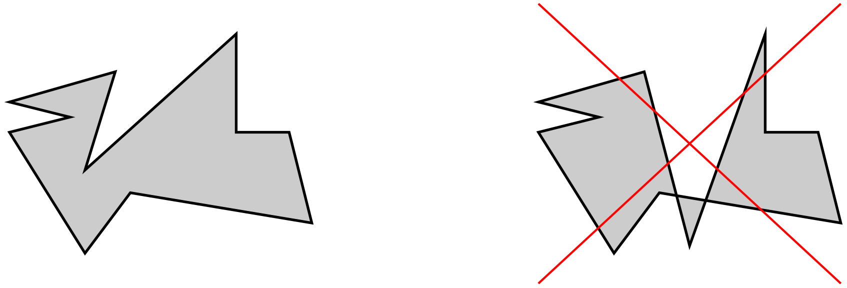
Kula otwarta: $K(p,r) := \{q \in X : d(p,q) < r\}$.





Podzbiór $S \subset \mathbb{R}^d$ nazywamy **wypukłym** wtedy i tylko wtedy, gdy dla dowolnej pary punktów $p, q \in S$ odcinek $\overline{pq}$ jest całkowicie zawarty w S.

Otoczka wypukła zbioru S jest to najmniejszy wypukły zbiór zawierający S.



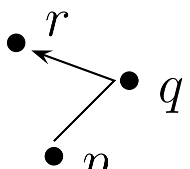
Wielokątem prostym nazywamy obszar ograniczony przez pojedynczy, domknięty, wielokątny łańcuch, który nie przecina się ze sobą.

Proste problemy geometryczne na płaszczyźnie

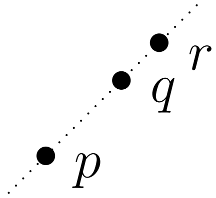
Problem orientacji punktów

Mówimy, że (uporządkowana) trójka punktów (p, q, r) ma **dodatnią orientację** wtedy i tylko wtedy, gdy punkty te definiują wierzchołki trójkąta przeciwnie do ruchu wskazówek zegara; **ujemną orientację** – zgodnie z ruchem wskazówek zegara; **zerową orientację** – są współliniowe.

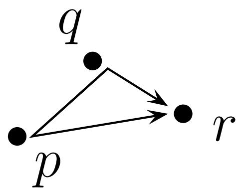
$$\operatorname{sgn}\left(\begin{vmatrix} \mathrm{x}(p) & \mathrm{y}(p) & 1 \\ \mathrm{x}(q) & \mathrm{y}(q) & 1 \\ \mathrm{x}(r) & \mathrm{y}(r) & 1 \end{vmatrix}\right) = \begin{cases} > 0 & \text{orientacja dodatnia} \\ \;\;0 & \text{współliniowe} \\ < 0 & \text{orientacja ujemna} \end{cases}$$



$o(p, q, r) > 0$



$o(p, q, r) = 0$

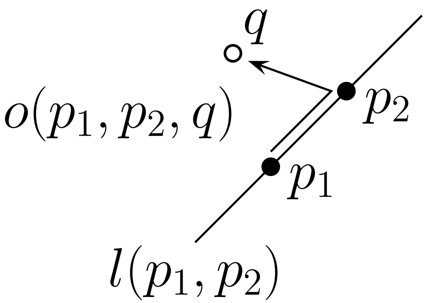


$o(p, q, r) < 0$

$o(q, p, r) > 0$

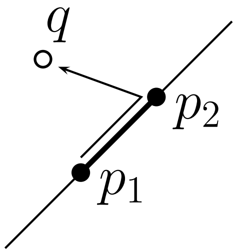
Położenie punktu q względem prostej $l(p_1, p_2)$

Położenie punktu względem $l(p_1, p_2)$ określone jest przez orientację $o(p_1, p_2, q)$.

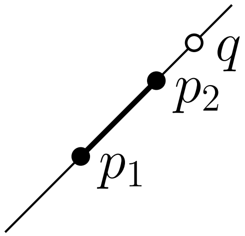


Położenie punktu q na odcinku $\overline{p_1 p_2}$

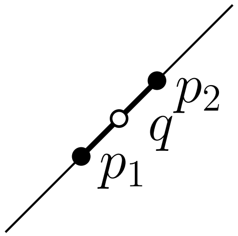
$o(p_1, p_2, q) = 0$ oraz $\mathrm{x}(q) \in [\mathrm{x}(p_1), \mathrm{x}(p_2)]$ i $\mathrm{y}(q) \in [\mathrm{y}(p_1), \mathrm{y}(p_2)]$.



$o(p_1, p_2, q) \neq 0$



$o(p_1, p_2, q) = 0$
$\mathrm{x}(q) \notin [\mathrm{x}(p_1), \mathrm{x}(p_2)]$

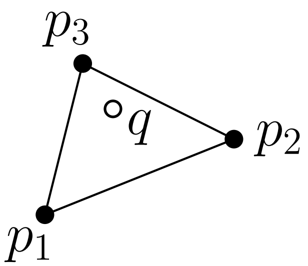


$o(p_1, p_2, q) = 0$
$\mathrm{x}(q) \in [\mathrm{x}(p_1), \mathrm{x}(p_2)]$
$\mathrm{y}(q) \in [\mathrm{y}(p_1), \mathrm{y}(p_2)]$

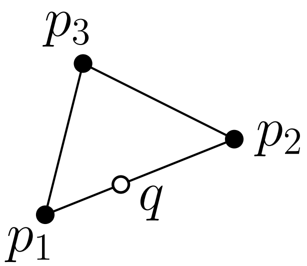
Położenie punktu względem trójkąta $\triangle(p_1, p_2, p_3)$

Wyznaczamy orientacje $o(p_1, p_2, q), o(p_1, p_3, q), o(p_2, p_3, q)$:

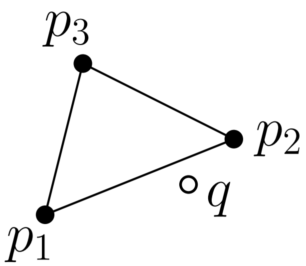
- jeżeli wszystkie orientacje są takie same i $\neq 0$, to punkt leży wewnątrz trójkąta;
- jeżeli dwie orientacje są takie same, a trzecia jest równa 0, to punkt leży na jednym z boków trójkąta;
- jeżeli znaki dwóch orientacji są różne, to punkt leży poza trójkątem.



$$o(p_1, p_2, q) = \\ = o(p_2, p_3, q) = \\ = o(p_3, p_1, q)$$



$$o(p_1, p_2, q) = 0 \\ o(p_2, p_3, q) = \\ = o(p_3, p_1, q)$$

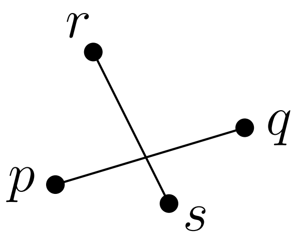


$$o(p_1, p_2, q) \neq o(p_3, p_1, q)$$

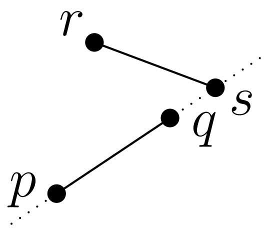
Problem przecięcia dwóch odcinków pq oraz rs

Wyznaczamy orientacje $o(p,q,r), o(p,q,s), o(r,s,p), o(r,s,q)$:

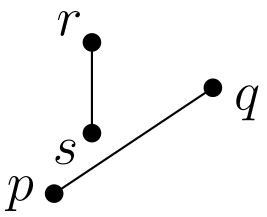
- jeżeli któraś z orientacji $o(a,b,c)$ jest równa zero, wtedy należy sprawdzić, czy punkt c leży na odcinku $\overline{ab}$;
- w przeciwnym wypadku wystarczy sprawdzić, czy $o(p,q,r) \neq o(p,q,s)$ oraz $o(r,s,p) \neq o(r,s,q)$.



$o(p,q,r) \neq o(p,q,s)$
$o(r,s,p) \neq o(r,s,q)$



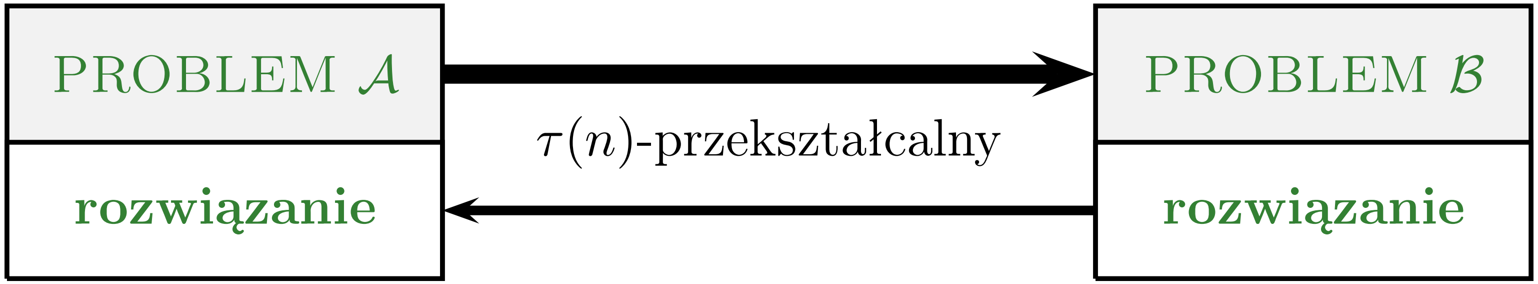
$o(p,q,s) = 0$
$s \notin pq$



$o(p,q,s) = o(p,q,r)$

3. Przekształcanie problemów

F.P. Preparata, M.I. Shamos
Geometria obliczeniowa – wprowadzenie
rozdziały 1.4 oraz 5.1, Helion (2003)



PROBLEM $\mathcal{A}$

rozwiązanie

$\tau(n)$-przekształcalny

PROBLEM $\mathcal{B}$

rozwiązanie

$$\mathcal{A} \propto_{\tau(n)} \mathcal{B}$$

PROBLEM SORTOWANIA

Posortuj ciąg liczb $(x_1, x_2, \ldots, x_n)$.

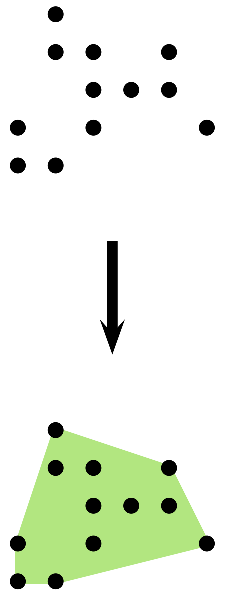
$$(2, 1, 3, 0)$$

$$\Big\downarrow$$

$$(0, 1, 2, 3)$$

PROBLEM OTOCZKI WYPUKŁEJ

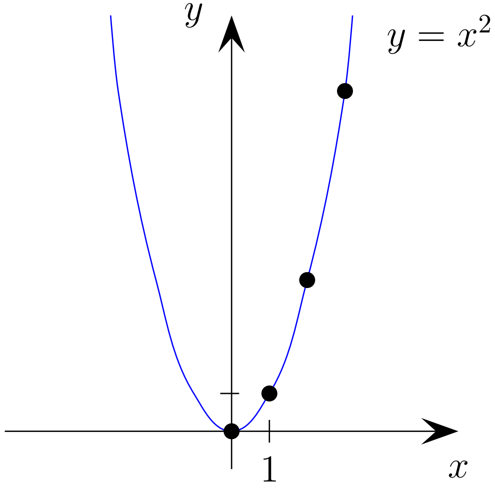
Dla zbioru punktów $S \subset \mathbb{R}^2$
wyznacz najmniejszy wielokąt
wypukły, który zawiera
wszystkie punkty z S.



$\tau(n)$-przekształcalny

ciąg liczb $(x_1, x_2, \ldots, x_n)$ $\rightleftarrows$ $S = \{(x_1, x_1^2), \ldots, (x_n, x_n^2)\}$

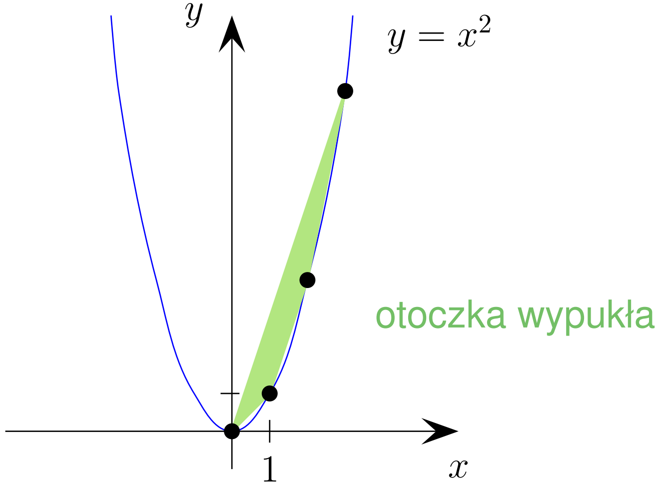
$(2, 1, 3, 0)$ $\longrightarrow$



$\tau(n)$-przekształcalny

ciąg liczb $(x_1, x_2, \ldots, x_n)$ $\rightleftarrows$ $S = \{(x_1, x_1^2), \ldots, (x_n, x_n^2)\}$

$(2, 1, 3, 0)$ $\longrightarrow$



$\big\downarrow$

$(0, 1, 2, 3)$ $\longleftarrow$ $((0, 0), (1, 1), (2, 4), (3, 9))$

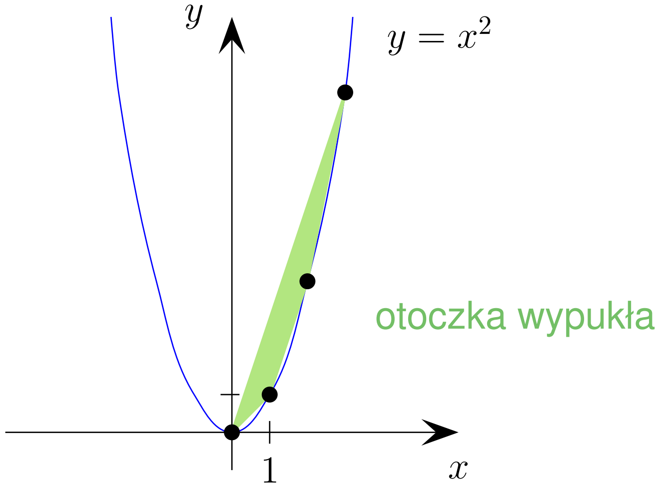
rozwiązanie

rozwiązanie

$\tau(n)$-przekształcalny

$$\begin{tikzcd} \text{ciąg liczb } (x_1, x_2, \ldots, x_n) \arrow[r, shift left] & S = \{(x_1, x_1^2), \ldots, (x_n, x_n^2)\} \arrow[l, shift left] \end{tikzcd}$$

$(2, 1, 3, 0)$ $\xrightarrow{\text{szybko}}$



$\Big\downarrow$ szybko

$(0, 1, 2, 3)$

rozwiązanie

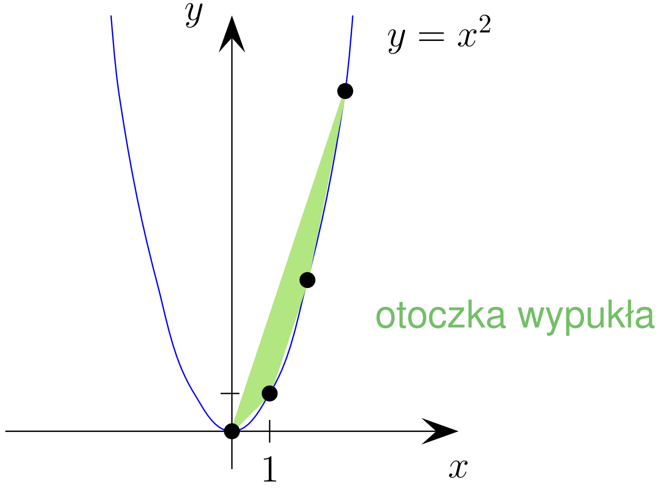
$\xleftarrow[\text{szybko}]{}$

$((0, 0), (1, 1), (2, 4), (3, 9))$

rozwiązanie

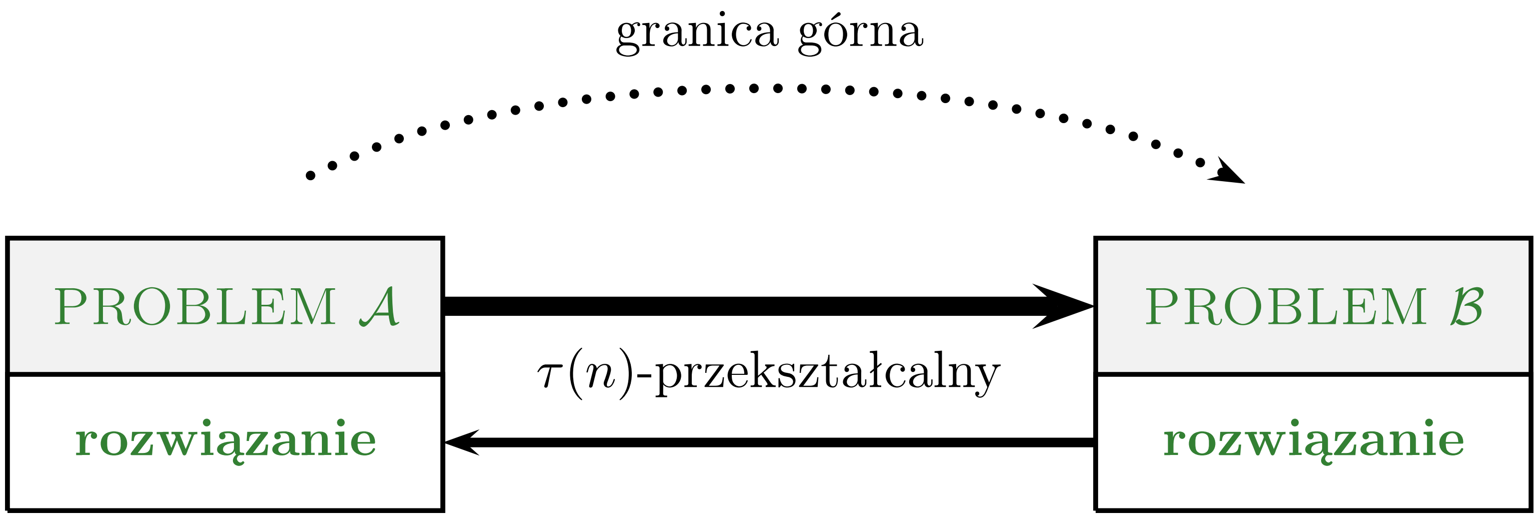
$\tau(n)$-przekształcalny

$$\begin{tikzcd}
\text{ciąg liczb } (x_1, x_2, \ldots, x_n) \arrow[r, shift left] & S = \{(x_1, x_1^2), \ldots, (x_n, x_n^2)\} \arrow[l, shift left]
\end{tikzcd}$$



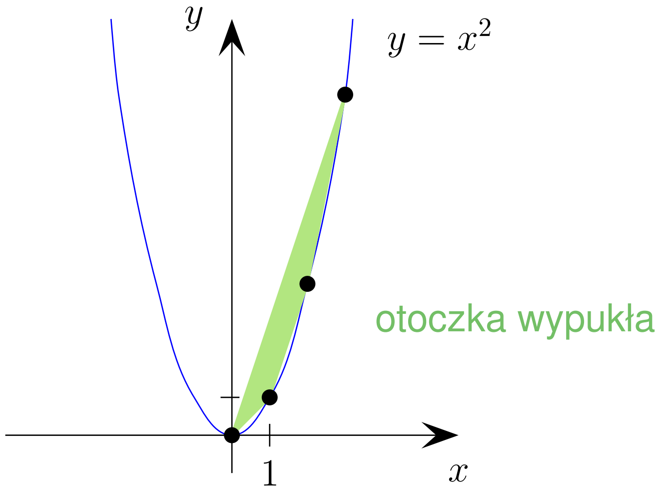
$$\begin{tikzcd}
(2, 1, 3, 0) \arrow[r, "\text{szybko}"] \arrow[d, "\text{szybko}"'] & {} \arrow[d, "\text{szybko}"] \\
\underset{\text{rozwiązanie}}{(0, 1, 2, 3)} & \underset{\text{rozwiązanie}}{((0, 0), (1, 1), (2, 4), (3, 9))} \arrow[l, "\text{szybko}"']
\end{tikzcd}$$

- *Jeśli problem $\mathcal{A}$ jest przekształcalny w problem $\mathcal{B}$ w „krótkim czasie” i rozwiązanie problemu $\mathcal{B}$ wymaga „mało czasu”, to rozwiązanie problemu $\mathcal{A}$ wymaga także „mało czasu”.*



$\tau(n)$-przekształcenie

$$\begin{tikzcd} \text{ciąg liczb } (x_1, x_2, \ldots, x_n) \arrow[r, shift left] & S = \{(x_1, x_1^2), \ldots, (x_n, x_n^2)\} \arrow[l, shift left] \end{tikzcd}$$



$$\begin{tikzcd} (2, 1, 3, 0) \arrow[r, "\text{szybko}"] \arrow[d, "\text{nie da się szybko}"'] & \, \\ (0, 1, 2, 3) & ((0, 0), (1, 1), (2, 4), (3, 9)) \arrow[l, "\text{szybko}"'] \end{tikzcd}$$

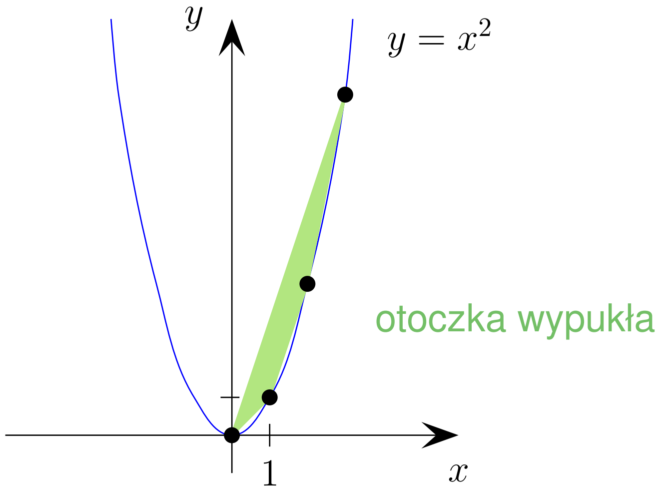
rozwiązanie

rozwiązanie

$\tau(n)$-przekształcenie

$$\begin{tikzcd} \text{ciąg liczb } (x_1, x_2, \ldots, x_n) \arrow[r, shift left] & S = \{(x_1, x_1^2), \ldots, (x_n, x_n^2)\} \arrow[l, shift left] \end{tikzcd}$$

$(2, 1, 3, 0)$ $\xrightarrow{\text{szybko}}$



$\big\downarrow$ nie da się szybko

$\big\downarrow$ ~~szybko~~ nie da się szybko

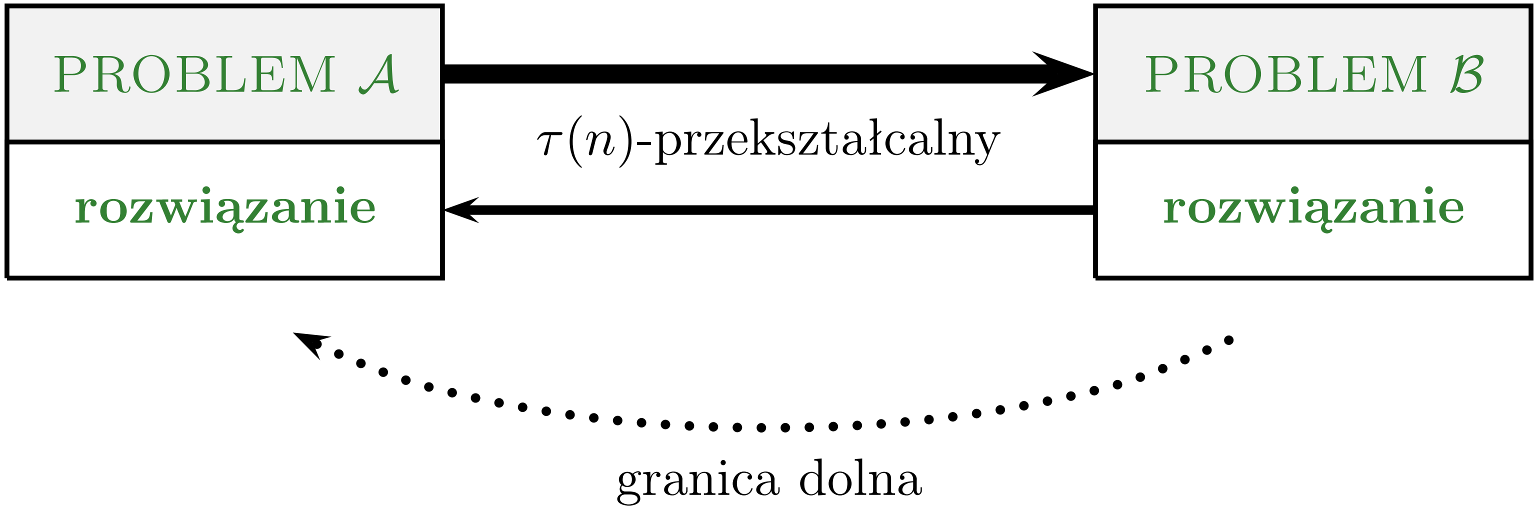
$(0, 1, 2, 3)$

rozwiązanie

$\xleftarrow[\text{szybko}]{}$

$((0, 0), (1, 1), (2, 4), (3, 9))$

rozwiązanie



PROBLEM $\mathcal{A}$

rozwiązanie

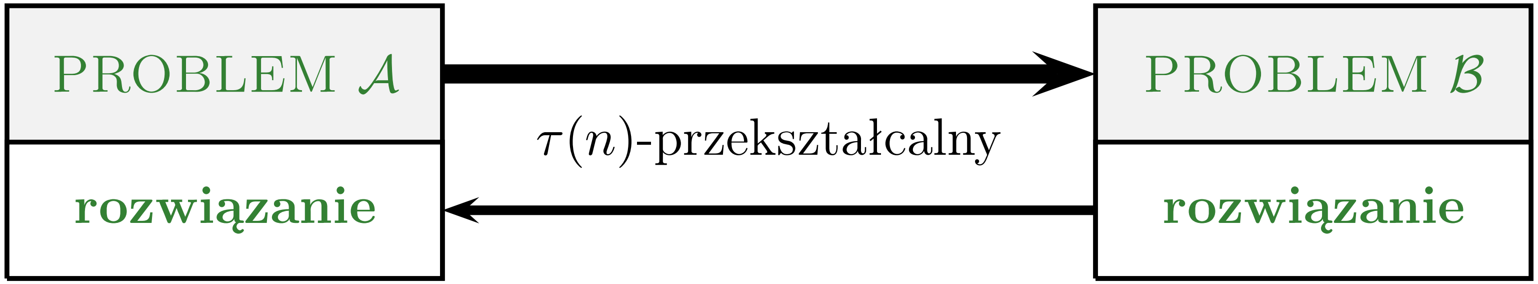
$\tau(n)$-przekształcalny

PROBLEM $\mathcal{B}$

rozwiązanie

granica dolna

- *Jeśli problem $\mathcal{A}$ jest przekształcalny w problem $\mathcal{B}$ w „krótkim czasie”, a rozwiązanie problemu $\mathcal{A}$ wymaga „dużo czasu”, to rozwiązanie problemu $\mathcal{B}$ wymaga także „dużo czasu”.*



PROBLEM $\mathcal{A}$

rozwiązanie

$\tau(n)$-przekształcalny

PROBLEM $\mathcal{B}$

rozwiązanie

Załóżmy, że mamy dane dwa problemy, $\mathcal{A}$ i $\mathcal{B}$, które są ze sobą powiązane tak, że problem $\mathcal{A}$ można rozwiązać następująco.

(1) Dane wejściowe problemu $\mathcal{A}$ są przekształcane w dane wejściowe problemu $\mathcal{B}$.

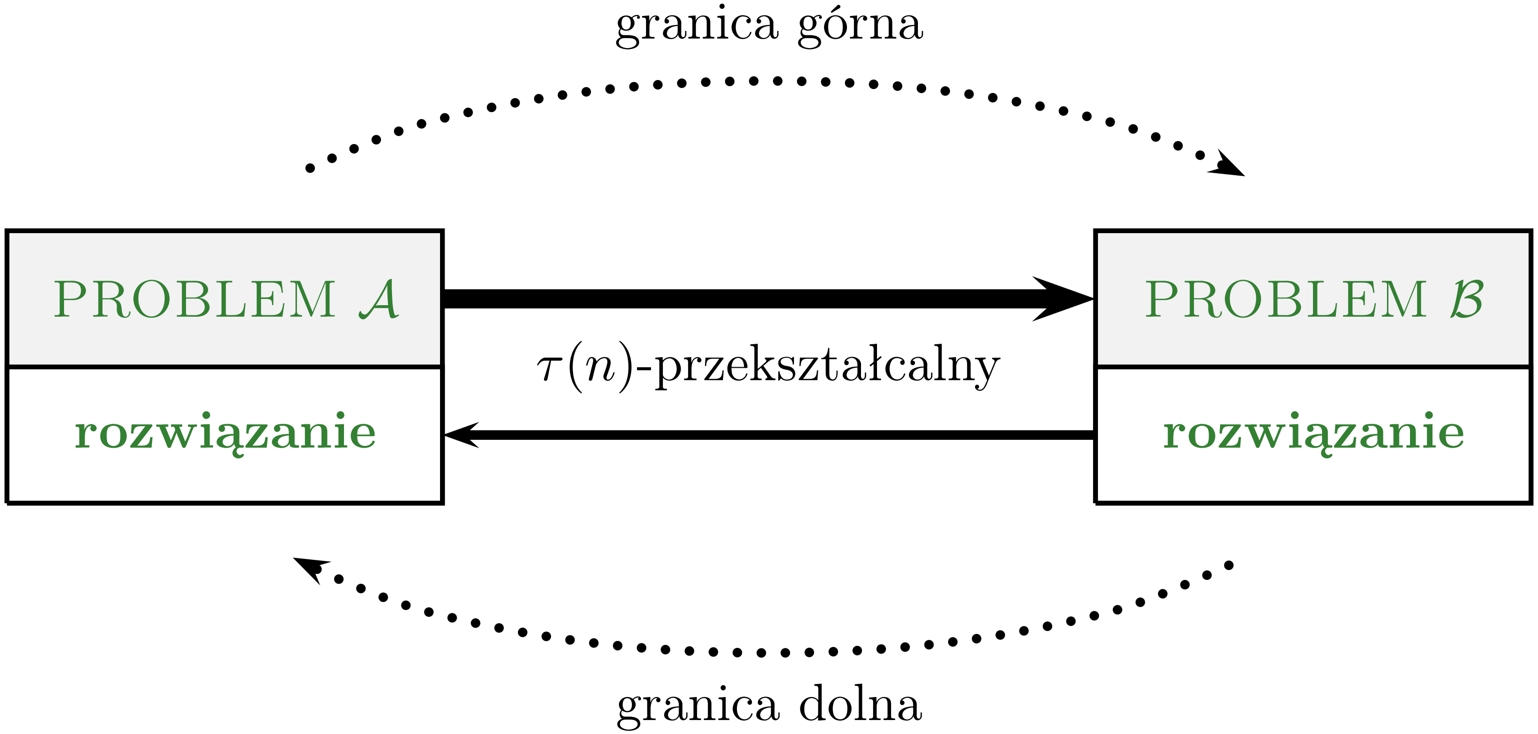
(2) Rozwiązywany jest problem $\mathcal{B}$.

(3) Wyniki problemu $\mathcal{B}$ przekształcane są w poprawne rozwiązanie problemu $\mathcal{A}$.

Mówimy wtedy, że problem $\mathcal{A}$ *został przekształcony* w problem $\mathcal{B}$ (albo *zredukowany* do problemu $\mathcal{B}$). Ponadto, jeśli powyższe przekształcenie (1)-(3) można wykonać w czasie $O(\tau(n))$, gdzie n jest rozmiarem (instancji) problemu $\mathcal{A}$, to mówimy, że $\mathcal{A}$ jest *$\tau(n)$-przekształcalne* w $\mathcal{B}$, co zapisujemy

$$\mathcal{A} \propto_{\tau(n)} \mathcal{B}.$$

Twierdzenie 3.1. (Górne ograniczenie przez możliwość przekształcania.) *Jeśli problem $\mathcal{B}$ wymaga czasu $T(n)$ i $\mathcal{A}$ jest $\tau(n)$-przekształcalny w problem $\mathcal{B}$, to $\mathcal{A}$ wymaga co najwyżej czasu $T(n) + O(\tau(n))$.*



Twierdzenie 3.2. (Dolne ograniczenie przez możliwość przekształcania.) *Jeśli problem $\mathcal{A}$ wymaga czasu $T(n)$ i $\mathcal{A}$ jest $\tau(n)$-przekształcalny w problem $\mathcal{B}$, to $\mathcal{B}$ wymaga co najmniej czasu $T(n) - O(\tau(n))$.*

Algebraiczne drzewo obliczeń jest uogólnieniem algebraicznego drzewa decyzyjnego. Posiada ono dwa rodzaje wierzchołków:

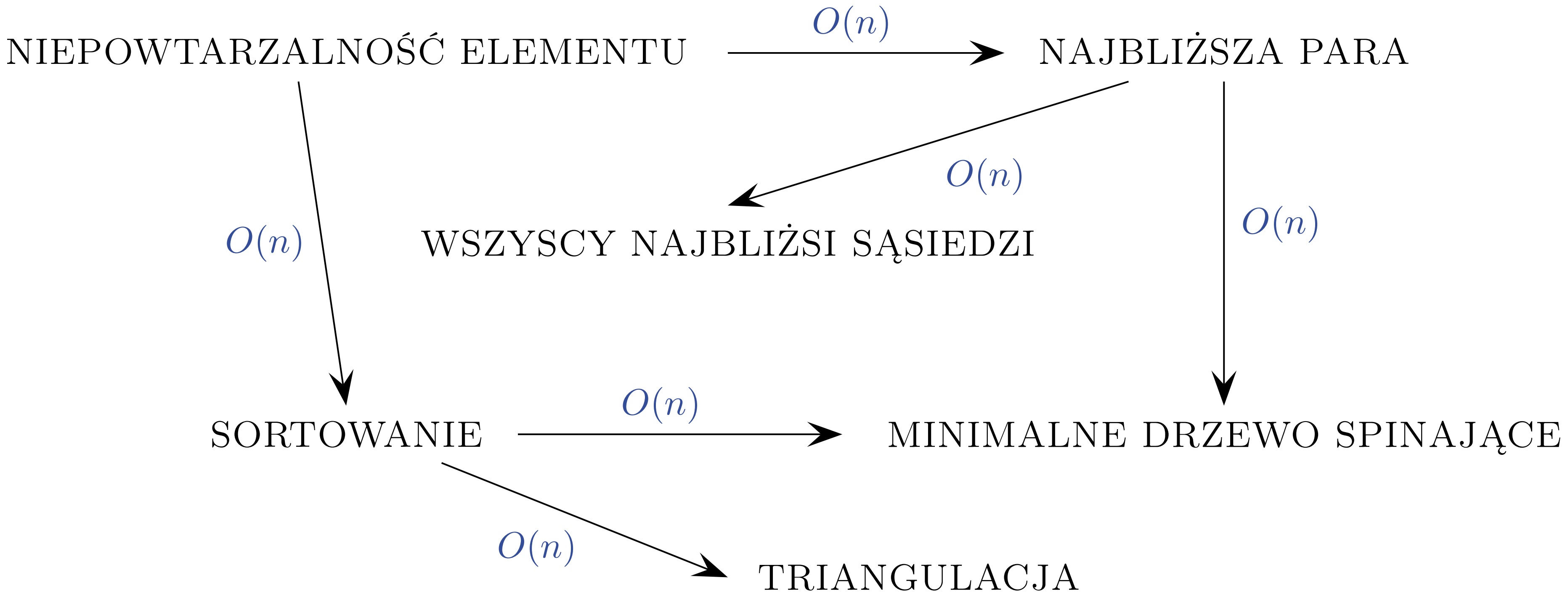
- Wierzchołki obliczeniowe: z każdym takim wierzchołkiem u związana jest wartość f_u, która jest określona jako wynik jednej z poniższych operacji:

  $$f_u := f_w + f_v; \quad f_u := f_w - f_v; \quad f_u := f_w \cdot f_v; \quad f_u := f_w / f_v; \quad f_u := \sqrt{f_v};$$

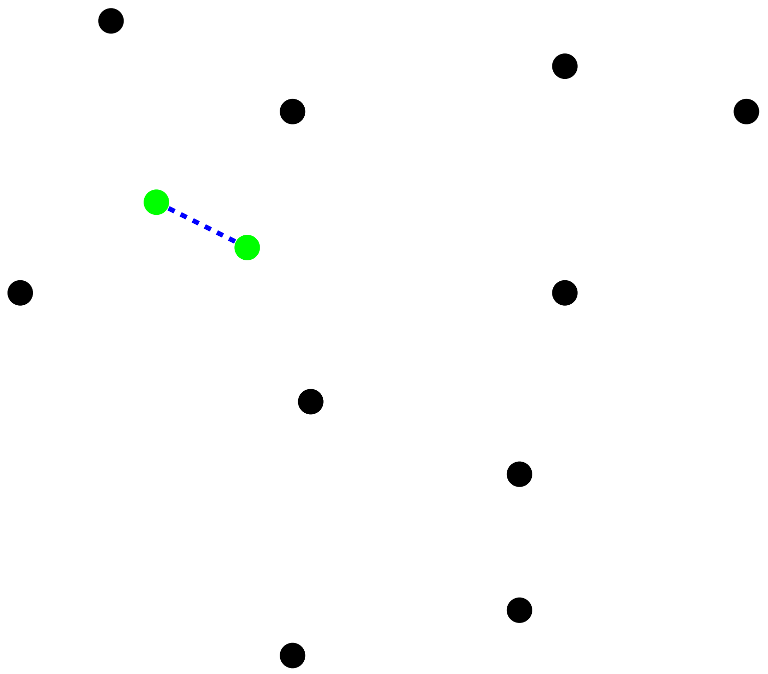
  gdzie f_w i f_v są wartościami skojarzonymi z pewnymi przodkami wierzchołka u lub są elementami ciągu wejściowego, lub stałymi z $\mathbb{R}$.
- Wierzchołki rozgałęziające: wierzchołek v wykonuje test $f_u < 0$ bądź $f_u \geq 0$, bądź $f_u = 0$, gdzie u jest przodkiem v.

Dla danego wejścia $(x_1, x_2, \ldots, x_n)$ podążamy ścieżką od korzenia do liścia, w każdym z wierzchołków wykonując obliczenie lub test (i wybór odpowiedniego potomka). Algorytm zwraca wartość (TAK/NIE) w osiągniętym liściu. *Złożoność czasowa algorytmu dla danej instancji* to długość ścieżki obliczeń, natomiast (pesymistyczna) *złożoność czasowa algorytmu* to głębokość tak zdefiniowanego drzewa.

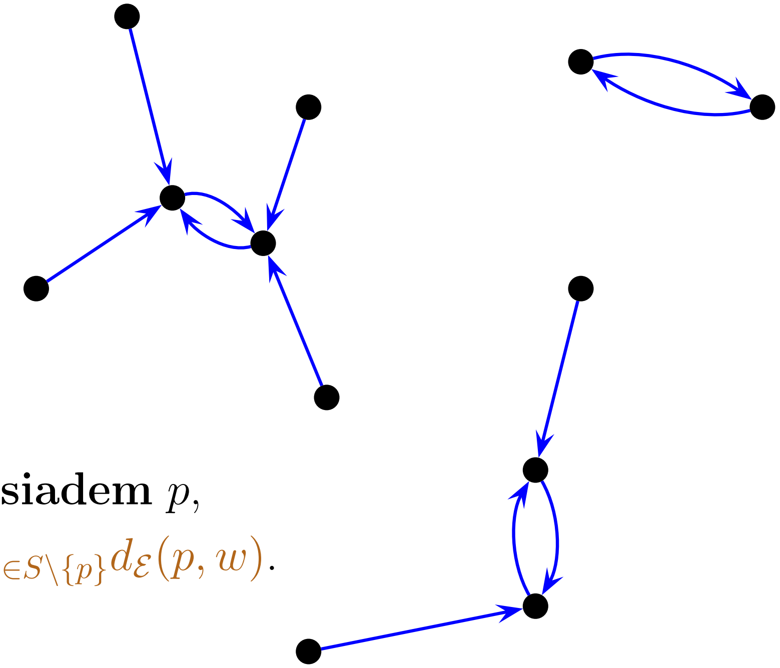
Twierdzenie 3.3. (Ben-Or 1983)
W modelu algebraicznego drzewa obliczeń dowolny algorytm określający, czy wszystkie spośród n danych liczb naturalnych są różne, wymaga czasu rzędu $\Omega(n \log n)$.


NIEPOWTARZALNOŚĆ ELEMENTU
$O(n)$
NAJBLIŻSZA PARA
$O(n)$
WSZYSCY NAJBLIŻSI SĄSIEDZI
$O(n)$
$O(n)$
SORTOWANIE
$O(n)$
MINIMALNE DRZEWO SPINAJĄCE
$O(n)$
TRIANGULACJA

Problem. (Para najbliższych punktów na płaszczyźnie)
Wyznacz najbliższą parę punktów spośród punktów z danego zbioru S $(\subset \mathcal{E}^2)$.

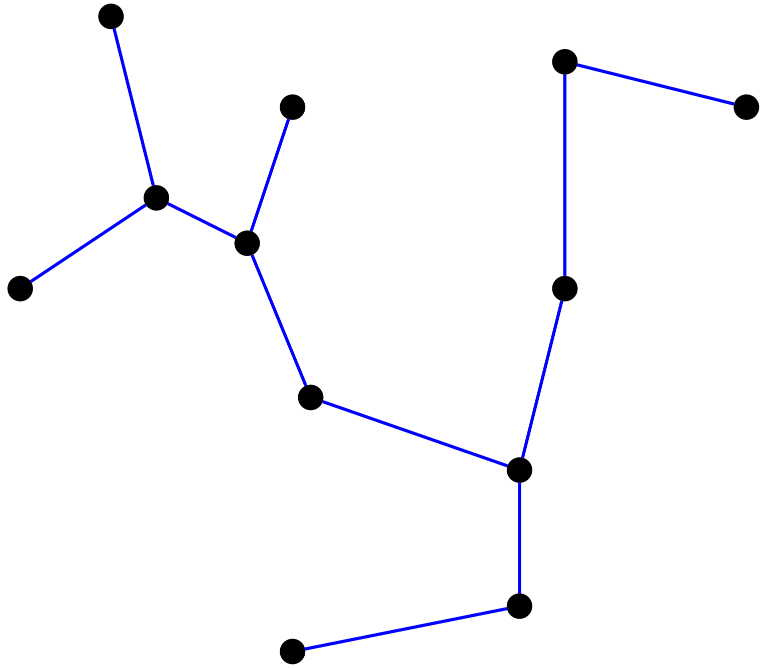


Problem. (Wszystkie najbliższe punkty na płaszczyźnie)
Dla każdego punktu ze zbioru S $(\subset \mathcal{E}^2)$ *znajdź „najbliższego mu sąsiada” z* S.

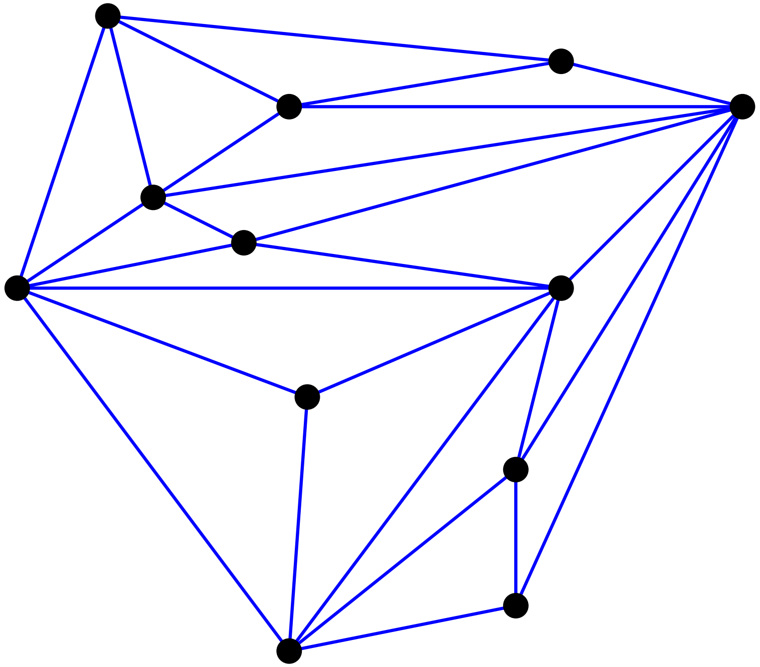


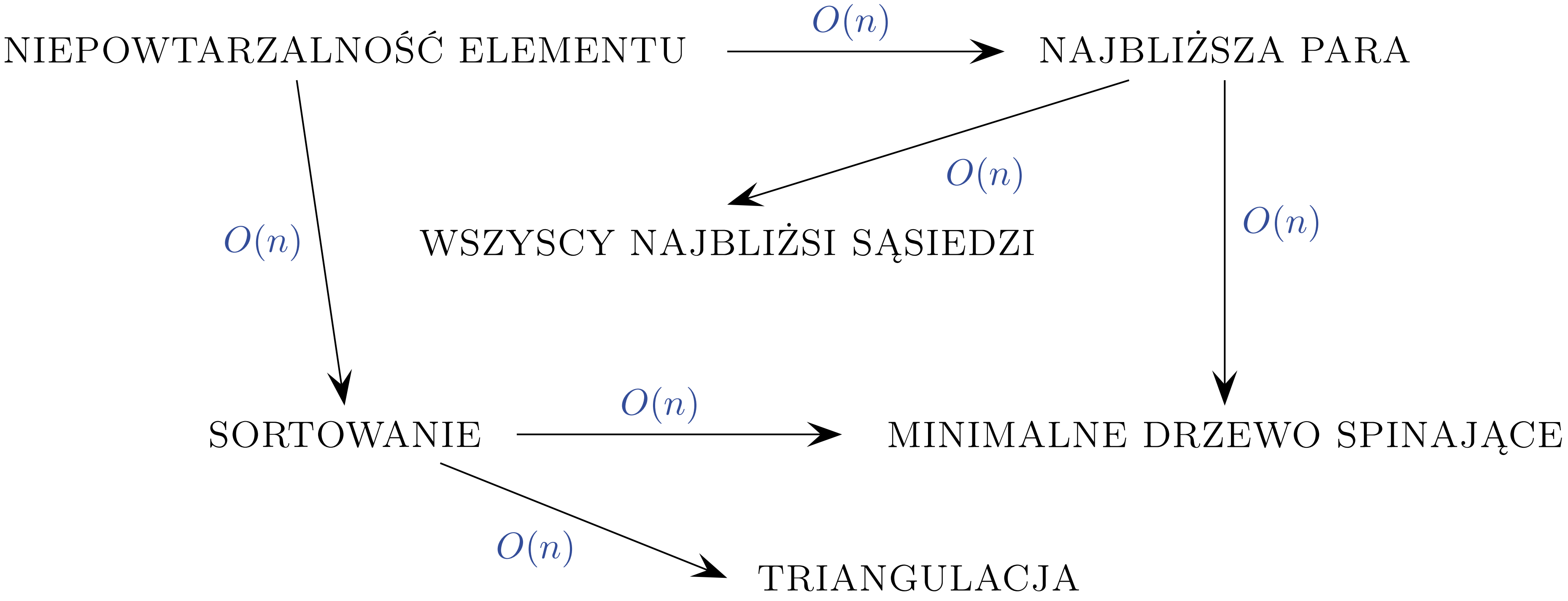
Punkt q jest **najbliższym sąsiadem** p,
jeśli zachodzi $d_{\mathcal{E}}(p,q) = \min_{w \in S \setminus \{p\}} d_{\mathcal{E}}(p,w)$.

Problem. (Minimalne drzewo spinające na płaszczyźnie)
Dla danego zbioru S punktów na płaszczyźnie skonstruować minimalne drzewo spinające, tj. drzewo o najmniejszej całkowitej sumie odległości/wag krawędzi, którego wierzchołki odpowiadają punktom z S.



Problem. (Triangulacja zbioru punktów na płaszczyźnie)
Niech S będzie zbiorem n punktów na płaszczyźnie. Połącz punkty z S nieprzecinającymi się odcinkami tak, aby każdy obszar wewnątrz otoczki wypukłej CH(S) *był (niezdegenerowanym) trójkątem.*




NIEPOWTARZALNOŚĆ ELEMENTU
$O(n)$
NAJBLIŻSZA PARA
$O(n)$
WSZYSCY NAJBLIŻSI SĄSIEDZI
$O(n)$
$O(n)$
SORTOWANIE
$O(n)$
MINIMALNE DRZEWO SPINAJĄCE
$O(n)$
TRIANGULACJA

4. TECHNIKA „DZIEL I ZWYCIĘŻAJ”

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Wprowadzenie do algorytmów, rozdział 2.3
Wydawnictwa naukowo-Techniczne (2004)

Dziel

Dzielimy problem na k podproblemów ($k > 1$) o rozmiarach $n_1, \ldots, n_k$.

Rekurencja

Rozwiązujemy podproblemy rekurencyjnie, chyba że są one „małego” rozmiaru – wtedy używamy bezpośrednich metod.

Połącz

Konstruujemy rozwiązanie problemu w oparciu o rozwiązania podproblemów.

Złożoność czasowa takiego podejścia wyraża się następującym wzorem:

$$T(n) = \begin{cases} O(1) & \text{jeśli } n \leq \text{const;} \\ \sum_{i=1}^{k} T(n_i) + f(n) & \text{w przeciwnym wypadku.} \end{cases}$$

Jeśli np. $k = 2$, $n_1 \approx n/2$, $n_2 \approx n/2$ oraz $f(n) = O(n)$, wówczas $T = O(n \log n)$.

Przykłady

- ▶ Sortowanie przez scalanie.
  ↳ $k = 2$, $n_1 \approx n/2$, $n_2 \approx n/2$ oraz $f(n) = \Theta(n)$: $T(n) = \Theta(n \log n)$.
- ▶ Wyznaczanie mediany (problem selekcji).
  ↳ $k = 2$, $n_1 \approx 2n/10$, $n_2 \approx 7n/10$ oraz $f(n) = \Theta(n)$: $T(n) = \Theta(n)$.

Twierdzenie 4.1. (Cormen *et al.* 1990) */Tw. o rekurencji uniwersalnej/*

Niech dana będzie funkcja $T : \mathbb{N}^+ \to \mathbb{N}^+$ określona zależnością reukrencyjną

$$T(n) = a \cdot T(\frac{n}{b}) + f(n),$$

gdzie $a \geq 1, b > 1$, natomiast $\frac{n}{b}$ oznacza $\lfloor \frac{n}{b} \rfloor$ lub $\lceil \frac{n}{b} \rceil$.

1. *Jeśli $f(n) = O(n^{\log_b a - \epsilon})$ dla pewnego $\epsilon > 0$, wówczas $T(n) = \Theta(n^{\log_b a})$.*
2. *Jeśli $f(n) = \Theta(n^{\log_b a})$, wówczas $T(n) = \Theta(n^{\log_b a} \log_2 n)$.*
3. *Jeśli $f(n) = \Omega(n^{\log_b a + \epsilon})$ dla pewnego $\epsilon > 0$ oraz jeśli $af(\frac{n}{b}) \leq cf(n)$ dla pewnej stałej $c < 1$ i wszystkich dostatecznie dużych n, to $T(n) = \Theta(f(n))$.*

Przykłady

- Sortowanie przez scalanie.
  ↳ $k = 2$, $n_1 \approx n/2$, $n_2 \approx n/2$ oraz $f(n) = \Theta(n)$: $T(n) = \Theta(n \log n)$.
- Wyznaczanie mediany (problem selekcji).
  ↳ $k = 2$, $n_1 \approx 2n/10$, $n_2 \approx 7n/10$ oraz $f(n) = \Theta(n)$: $T(n) = \Theta(n)$.

- …
- Wyznaczanie pary najbliższych punktów (na płaszczyźnie $\mathbb{E}^2$).
  ↳ $\mathbb{E}^2 = (\mathbb{R}^2, d_{\mathcal{E}})$, gdzie $d_{\mathcal{E}}$ jest metryką (odległością) euklidesową.

- Wyznaczanie punktów przecięć pionowych i poziomych odcinków (na $\mathbb{R}^2$).
- …
- Redukcja złożoności pamięciowej (m.in. problem najkrótszej ścieżki).
- Szacowanie parametrów (m.in. problem przeszukiwania wielokąta).

4.1 Sortowanie przez scalanie /J. von Neumann (1945)/

T.H. Cormen *et al.*, C.E. Leiserson, R.L. Rivest, C. Stein
Wprowadzenie do algorytmów, rozdział 2.3
Wydawnictwa naukowo-Techniczne (2004)

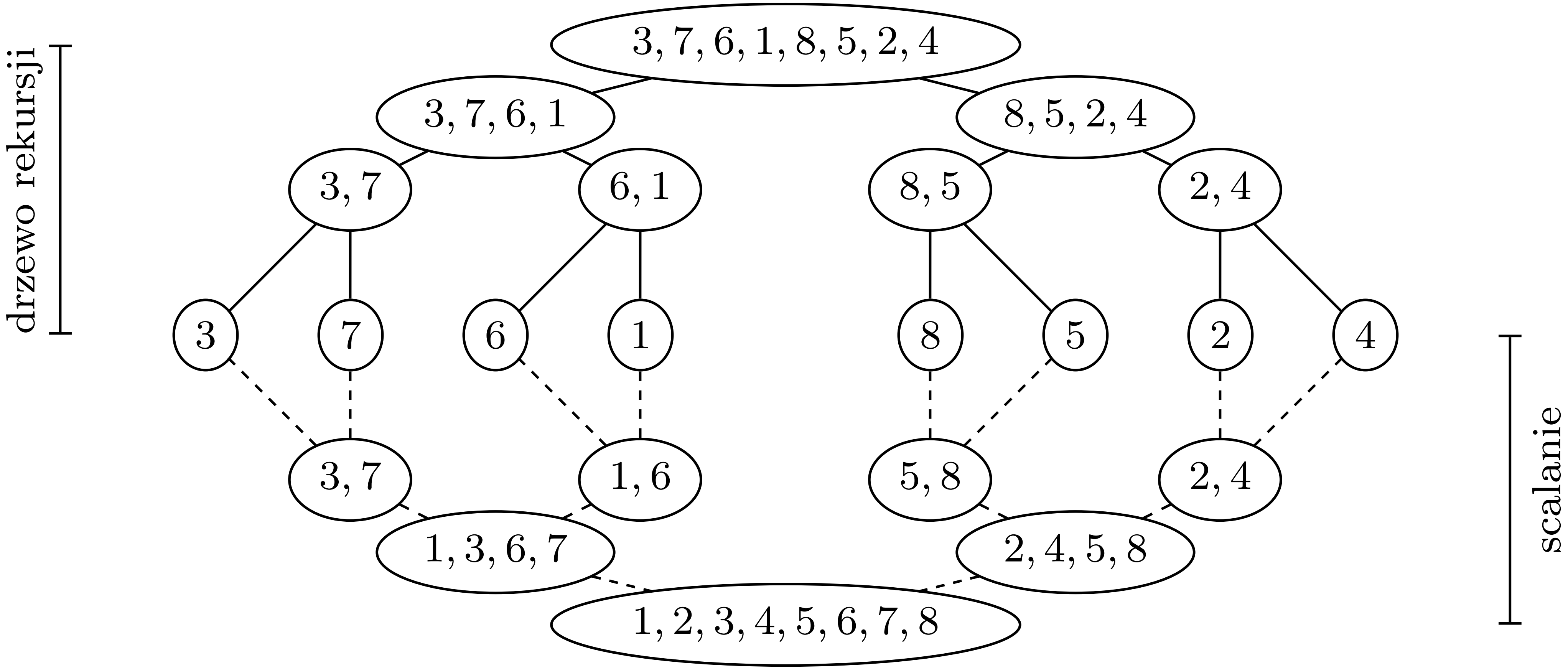
Problem. (Sortowanie)

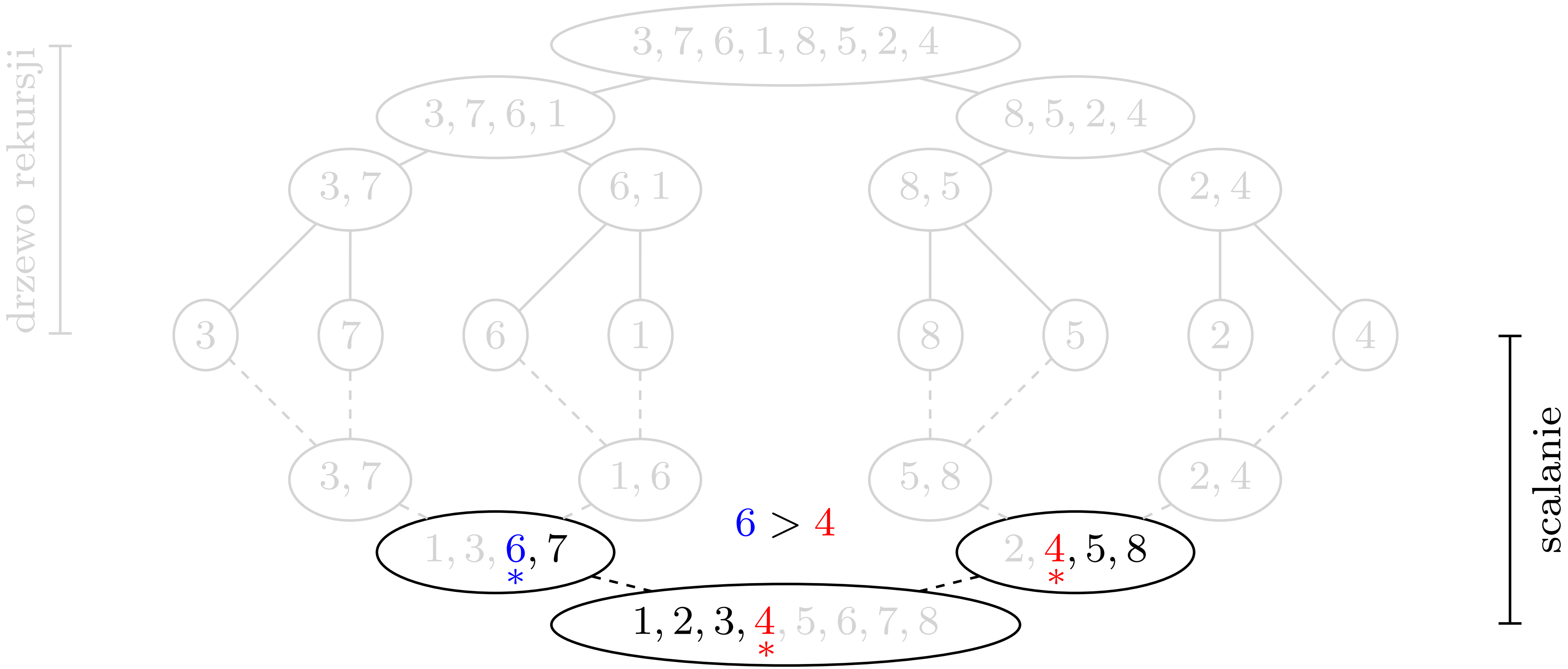
Wejście: *Zbiór S liczb (rzeczywistych/całkowitych).*

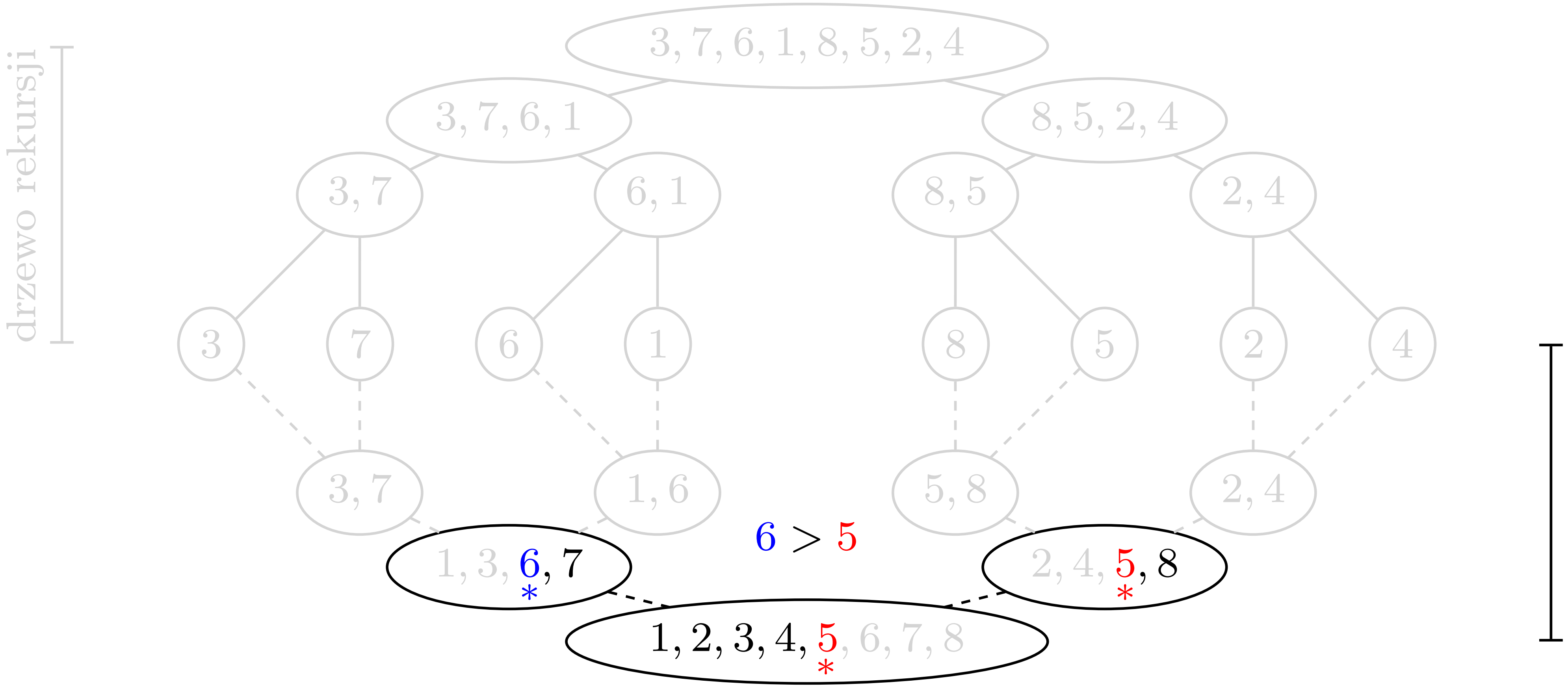
Wyjście: *Permutacja $(a_1, \ldots, a_{|S|})$ liczb wejściowych taka, że $a_i \leq a_{i+1}$, $i = 1, \ldots, |S| - 1$.*

Algorytm sortowania przez scalanie `merge-sort`**(S).**

```
Jeśli zbiór $S$ ma tylko jeden element, zwróć $S$.
W przeciwnym przypadku:
    Podziel $S$ na mniej więcej równe cześci $S_1$ i $S_2$.
    merge-sort($S_1$);
    merge-sort($S_2$);
    Scal $S_1$ i $S_2$ w jeden ciąg $S^*$ z zachowaniem kolejności i zwróć $S^*$.
```


drzewo rekursji
3, 7, 6, 1, 8, 5, 2, 4
3, 7, 6, 1
8, 5, 2, 4
3, 7
6, 1
8, 5
2, 4
3
7
6
1
8
5
2
4
3, 7
1, 6
5, 8
2, 4
1, 3, 6, 7
2, 4, 5, 8
1, 2, 3, 4, 5, 6, 7, 8
scalanie


drzewo rekursji
3, 7, 6, 1, 8, 5, 2, 4
3, 7, 6, 1
8, 5, 2, 4
3, 7
6, 1
8, 5
2, 4
3
7
6
1
8
5
2
4
3, 7
1, 6
5, 8
2, 4
6 > 4
1, 3, 6, 7
*
2, 4, 5, 8
*
1, 2, 3, 4, 5, 6, 7, 8
*
scalanie


drzewo rekursji
3, 7, 6, 1, 8, 5, 2, 4
3, 7, 6, 1
8, 5, 2, 4
3, 7
6, 1
8, 5
2, 4
3
7
6
1
8
5
2
4
3, 7
1, 6
5, 8
2, 4
6 > 5
1, 3, 6, 7
*
2, 4, 5, 8
*
1, 2, 3, 4, 5, 6, 7, 8
*
scalanie

Problem. (Sortowanie)

Wejście: *Zbiór S liczb (rzeczywistych/całkowitych).*

Wyjście: *Permutacja $(a_1, \ldots, a_{|S|})$ liczb wejściowych taka, że $a_i \leq a_{i+1}$, $i = 1, \ldots, |S| - 1$.*

Algorytm sortowania przez scalanie `merge-sort`**(S).**

```
(1) Jeśli zbiór $S$ ma tylko jeden element, zwróć $S$.
(2) W przeciwnym przypadku:
(3)     Podziel $S$ na mniej więcej równe cześci $S_1$ i $S_2$.
(4)     merge-sort($S_1$);
(5)     merge-sort($S_2$);
(6)     Scal $S_1$ i $S_2$ w jeden ciąg $S^*$ z zachowaniem kolejności i zwróć $S^*$.
```

Zachodzi $T(n) = 2 \cdot T(n/2) + \Theta(n)$, zatem $a = 2$, $b = 2$ i $f(n) = \Theta(n^{\log_b a})$.
Z twierdzenia o rekurencji uniwersalnej: $T(n) = \Theta(n^{\log_b a} \log_2 n) = \Theta(n \log n)$.

4.2 PROBLEM SELEKCJI

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Wprowadzenie do algorytmów, rozdział 9.3
Wydawnictwa Naukowo-Techniczne (2004)

Problem. (Selekcja)

Wejście: *Tablica A składająca się z $n \geq 1$ różnych liczb oraz liczba $k \geq 1$.*

Wyjście: *k-ty co do wielkości element tablicy A,
tzn. takie $a \in A$, że $|\{x \in A : x < a\}| = k - 1$.*

Idea algorytmu selekcji `SELECTION`$(A[1 \ldots n], k)$

(1) Jeśli $n \leq 140$, posortuj tablicę A i zwróć $A[k]$.

(2) W przeciwnym wypadku podziel tablicę A na $\lfloor \frac{n}{5} \rfloor$ grup $A_1, \ldots, A_{\lfloor \frac{n}{5} \rfloor}$ po 5 elementów oraz na co najwyżej jedną grupę $A_{\lceil \frac{n}{5} \rceil}$ o $(n \bmod 5)$ elementach.

(3) Posortuj elementy grup $A_1, \ldots, A_{\lceil \frac{n}{5} \rceil}$ i wyznacz medianę m_i każdej z nich.

(4) Wywołaj rekurencyjnie procedurę `SELECTION`, aby wyznaczyć (dolną) medianę m zbioru $\{m_1, \ldots, m_{\lceil \frac{n}{5} \rceil}\}$.

(5) ...

$k = 17$: **szukamy 17-tego elementu x w następującej tablicy** $(x = 62)$**:**

3 50 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

Idea algorytmu selekcji $\texttt{SELECTION}(A[1 \dots n], k)$

(1) Jeśli $n \leq 140$, posortuj tablicę A i zwróć $A[k]$.

(2) W przeciwnym wypadku podziel tablicę A na $\lfloor \frac{n}{5} \rfloor$ grup $A_1, \dots, A_{\lfloor \frac{n}{5} \rfloor}$ po 5 elementów oraz na co najwyżej jedną grupę $A_{\lceil \frac{n}{5} \rceil}$ o $(n \bmod 5)$ elementach.

(3) Posortuj elementy grup $A_1, \dots, A_{\lceil \frac{n}{5} \rceil}$ i wyznacz medianę m_i każdej z nich.

(4) Wywołaj rekurencyjnie procedurę $\texttt{SELECTION}$, aby wyznaczyć (dolną) medianę m zbioru $\{m_1, \dots, m_{\lceil \frac{n}{5} \rceil}\}$.

(5) ...

$k = 17$: szukamy 17-tego elementu x w następującej tablicy ($x = 62$):

3 50 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

‖ 3 50 60 63 11 ‖ 4 5 85 70 99 ‖ 61 101 62 19 22 ‖ 10 30 1 100 9 ‖ 82 21 40 71 20 ‖ 80 81 79 ‖

Idea algorytmu selekcji $\texttt{SELECTION}(A[1\ldots n], k)$

(1) Jeśli $n \leq 140$, posortuj tablicę A i zwróć $A[k]$.

(2) W przeciwnym wypadku podziel tablicę A na $\lfloor \frac{n}{5} \rfloor$ grup $A_1, \ldots, A_{\lfloor \frac{n}{5} \rfloor}$ po 5 elementów oraz na co najwyżej jedną grupę $A_{\lceil \frac{n}{5} \rceil}$ o (n mod 5) elementach.

(3) Posortuj elementy grup $A_1, \ldots, A_{\lceil \frac{n}{5} \rceil}$ i wyznacz medianę m_i każdej z nich.

(4) Wywołaj rekurencyjnie procedurę $\texttt{SELECTION}$, aby wyznaczyć (dolną) medianę m zbioru $\{m_1, \ldots, m_{\lceil \frac{n}{5} \rceil}\}$.

(5) ...

$k = 17$: **szukamy 17-tego elementu** x **w następującej tablicy** ($x = 62$):

3 50 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

‖ 3 50 60 63 11 ‖ 4 5 85 70 99 ‖ 61 101 62 19 22 ‖ 10 30 1 100 9 ‖ 82 21 40 71 20 ‖ 80 81 79 ‖

‖ 3 11 50 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

Idea algorytmu selekcji `SELECTION`$(A[1 \ldots n], k)$

(1) Jeśli $n \leq 140$, posortuj tablicę A i zwróć $A[k]$.

(2) W przeciwnym wypadku podziel tablicę A na $\lfloor \frac{n}{5} \rfloor$ grup $A_1, \ldots, A_{\lfloor \frac{n}{5} \rfloor}$ po 5 elementów oraz na co najwyżej jedną grupę $A_{\lceil \frac{n}{5} \rceil}$ o (n mod 5) elementach.

(3) Posortuj elementy grup $A_1, \ldots, A_{\lceil \frac{n}{5} \rceil}$ i wyznacz medianę m_i każdej z nich.

(4) Wywołaj rekurencyjnie procedurę `SELECTION`, aby wyznaczyć (dolną) medianę m zbioru $\{m_1, \ldots, m_{\lceil \frac{n}{5} \rceil}\}$.

(5) …

$k = 17$: **szukamy 17-tego elementu** x **w następującej tablicy** ($x = 62$):

3 50 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

‖ 3 50 60 63 11 ‖ 4 5 85 70 99 ‖ 61 101 62 19 22 ‖ 10 30 1 100 9 ‖ 82 21 40 71 20 ‖ 80 81 79 ‖

‖ 3 11 50 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

50 70 61 10 40 80

Idea algorytmu selekcji `SELECTION`$(A[1\dots n], k)$

(1) Jeśli $n \leq 140$, posortuj tablicę A i zwróć $A[k]$.

(2) W przeciwnym wypadku podziel tablicę A na $\lfloor \frac{n}{5} \rfloor$ grup $A_1, \dots, A_{\lfloor \frac{n}{5} \rfloor}$ po 5 elementów oraz na co najwyżej jedną grupę $A_{\lceil \frac{n}{5} \rceil}$ o $(n \bmod 5)$ elementach.

(3) Posortuj elementy grup $A_1, \dots, A_{\lceil \frac{n}{5} \rceil}$ i wyznacz medianę m_i każdej z nich.

(4) Wywołaj rekurencyjnie procedurę `SELECTION`, aby wyznaczyć (dolną) medianę m zbioru $\{m_1, \dots, m_{\lceil \frac{n}{5} \rceil}\}$.

(5) …

$k = 17$: **szukamy 17-tego elementu** x **w następującej tablicy** ($x = 62$):

3 50 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

‖ 3 50 60 63 11 ‖ 4 5 85 70 99 ‖ 61 101 62 19 22 ‖ 10 30 1 100 9 ‖ 82 21 40 71 20 ‖ 80 81 79 ‖

‖ 3 11 50 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

50 70 61 10 40 80

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

Idea algorytmu selekcji `SELECTION`$(A[1 \ldots n], k)$

(4) …

(5) Podziel tablicę wejściową względem mediany median m tak, że wszystkie elementy podtablicy $A[1 \ldots i-1]$ są mniejsze od m, $A[i] = m$, a wszystkie elementy podtablicy $A[i+1 \ldots n]$ są większe od m.

(6) Jeśli $i = k$, zwróć m. W przeciwnym razie wywołaj rekurencyjnie procedurę `SELECTION`, aby wyznaczyć:

- k-ty najmniejszy element w podtablicy $A[1 \ldots i-1]$, jeśli $i > k$;
- $(k-i)$-ty najmniejszy element w podtablicy $A[i+1 \ldots n]$, jeśli $i < k$.

$k = 17$: **szukamy 17-tego elementu** x **w następującej tablicy** ($x = 62$):

3 50 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

‖ 3 50 60 63 11 ‖ 4 5 85 70 99 ‖ 61 101 62 19 22 ‖ 10 30 1 100 9 ‖ 82 21 40 71 20 ‖ 80 81 79 ‖

‖ 3 11 50 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

50 70 61 10 40 80

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

$i = 14$, $k = 17 > 14$: **3-ci element w** ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

Idea algorytmu selekcji `SELECTION`$(A[1\ldots n], k)$

(4) …

(5) Podziel tablicę wejściową względem mediany median m tak, że wszystkie elementy podtablicy $A[1\ldots i-1]$ są mniejsze od m, $A[i] = m$, a wszystkie elementy podtablicy $A[i+1\ldots n]$ są większe od m.

(6) Jeśli $i = k$, zwróć m. W przeciwnym razie wywołaj rekurencyjnie procedurę `SELECTION`, aby wyznaczyć:

- k-ty najmniejszy element w podtablicy $A[1\ldots i-1]$, jeśli $i > k$;
- $(k-i)$-ty najmniejszy element w podtablicy $A[i+1\ldots n]$, jeśli $i < k$.

Złożoność czasowa algorytmu

- Krok 1. Sortowanie przez wstawianie tablicy A, gdy $|A| \leq 140$: $\Theta(1)$.
- Krok 2. Podział tablicy na grupy $A_1, \ldots, A_{\lceil \frac{n}{5} \rceil}$: $\Theta(n)$.
- Krok 3. Wyznaczenie mediany w każdej z grup (sortowanie): $\lceil \frac{n}{5} \rceil \cdot \Theta(1) = \Theta(n)$.
- Krok 4. Rekurencyjne wyznaczenie mediany m zbioru $\{m_1, \ldots, m_{\lceil \frac{n}{5} \rceil}\}$: $T(\lceil \frac{n}{5} \rceil)$.
- Krok 5. Podział tablicy A względem mediany median: $\Theta(n)$.

  ↳ Np. procedura `partition` używana przy sortowaniu szybkim: $\Theta(n)$.

$k = 17$: **szukamy 17-tego elementu x w następującej tablicy** ($x = 62$):

3 50 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

‖ 3 50 60 63 11 ‖ 4 5 85 70 99 ‖ 61 101 62 19 22 ‖ 10 30 1 100 9 ‖ 82 21 40 71 20 ‖ 80 81 79 ‖

‖ 3 11 50 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

50 70 61 10 40 80

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

$i = 14$, $k = 17 > 14$: **3-ci element w** ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

3 **50** 60 63 11 4 5 85 70 99 61 101 62 19 22 10 30 1 100 9 82 21 40 71 20 80 81 79

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

- Krok 6. Rekurencja dla podtablicy $A[1\ldots i-1]$ lub $A[i+1\ldots n]$: $T(\lfloor\frac{7n}{10}\rfloor+7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor - 6$. Zatem rozmiar podtablicy $A[1\ldots i-1]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor + 7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor - 6$. Zatem rozmiar podtablicy $A[i+1\ldots n]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor + 7$.

‖ 3 **50** 60 63 11 ‖ 4 5 85 70 99 ‖ 61 101 62 19 22 ‖ 10 30 1 100 9 ‖ 82 21 40 71 20 ‖ 80 81 79 ‖

podział na grupy 5-elementowe

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

- ▶ Krok 6. Rekurencja dla podtablicy $A[1\dots i-1]$ lub $A[i+1\dots n]$: $T(\lfloor\frac{7n}{10}\rfloor+7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor-6$. Zatem rozmiar podtablicy $A[1\dots i-1]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor+7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor-6$. Zatem rozmiar podtablicy $A[i+1\dots n]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor+7$.

‖ 3 11 **50** 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

posortowane grupy 5-elementowe i ich mediany

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

- ▶ Krok 6. Rekurencja dla podtablicy $A[1\ldots i-1]$ lub $A[i+1\ldots n]$: $T(\lfloor\frac{7n}{10}\rfloor+7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor - 6$. Zatem rozmiar podtablicy $A[1\ldots i-1]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor + 7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor - 6$. Zatem rozmiar podtablicy $A[i+1\ldots n]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor + 7$.

‖ 3 11 **50** 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

posortowane grupy 5-elementowe i ich mediany

3	11	**50**	60	63
4	5	70	85	99
19	22	61	62	101
1	9	10	30	100
20	21	40	71	82
	79	80	81	

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

- ▶ Krok 6. Rekurencja dla podtablicy $A[1 \ldots i-1]$ lub $A[i+1 \ldots n]$: $T(\lfloor \frac{7n}{10} \rfloor + 7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[1 \ldots i-1]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[i+1 \ldots n]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.

‖ 3 11 **50** 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

posortowane wiersze wzg. median

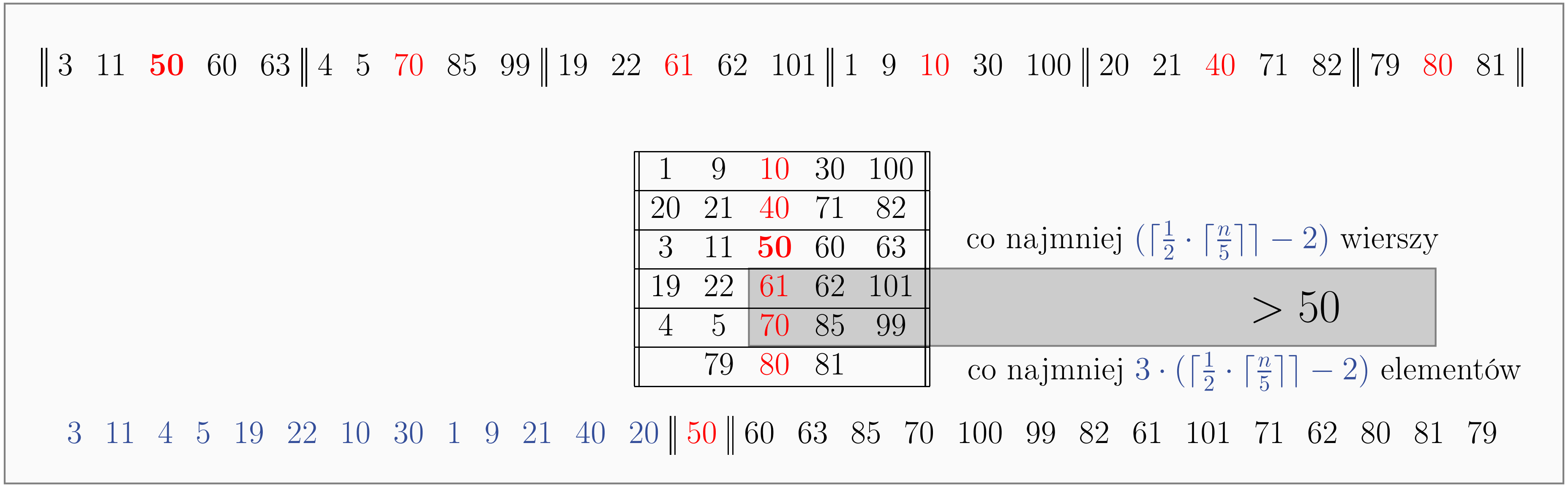
1	9	10	30	100
20	21	40	71	82
3	11	**50**	60	63
19	22	61	62	101
4	5	70	85	99
	79	80	81	

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

▶ Krok 6. Rekurencja dla podtablicy $A[1\ldots i-1]$ lub $A[i+1\ldots n]$: $T(\lfloor\frac{7n}{10}\rfloor+7)$.

↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor - 6$.
Zatem rozmiar podtablicy $A[1\ldots i-1]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor + 7$.

↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor - 6$.
Zatem rozmiar podtablicy $A[i+1\ldots n]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor + 7$.



‖ 3 11 **50** 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

1	9	10	30	100
20	21	40	71	82
3	11	**50**	60	63
19	22	61	62	101
4	5	70	85	99
	79	80	81	

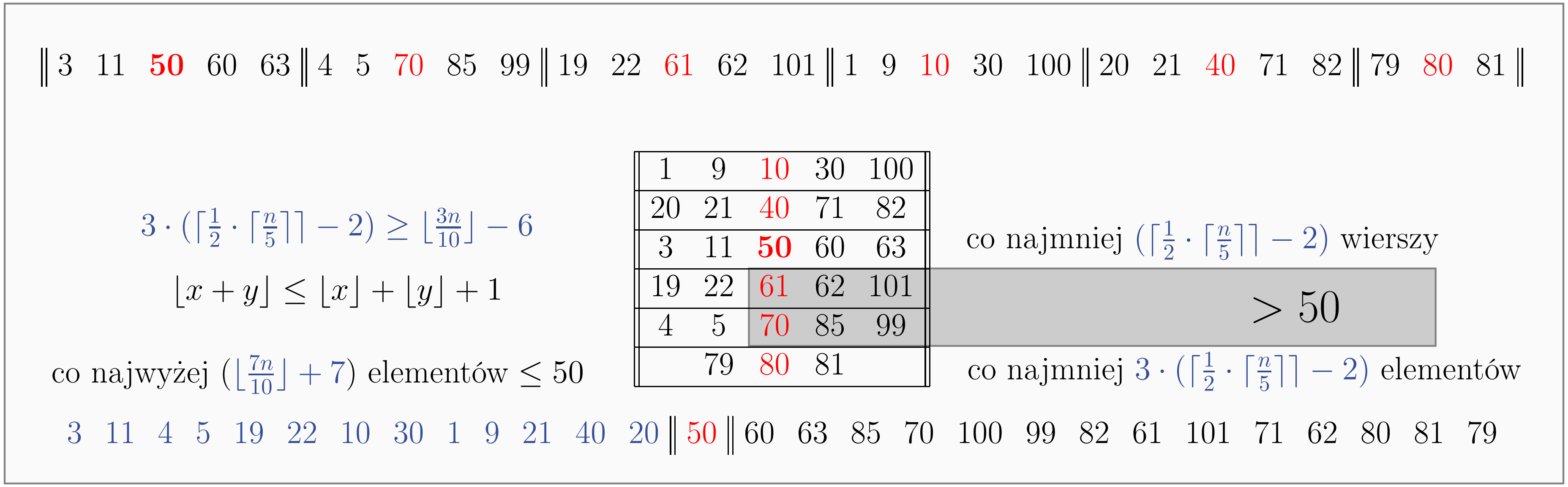
co najmniej $(\lceil\frac{1}{2}\cdot\lceil\frac{n}{5}\rceil\rceil-2)$ wierszy

> 50

co najmniej $3\cdot(\lceil\frac{1}{2}\cdot\lceil\frac{n}{5}\rceil\rceil-2)$ elementów

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

- Krok 6. Rekurencja dla podtablicy $A[1\ldots i-1]$ lub $A[i+1\ldots n]$: $T(\lfloor\frac{7n}{10}\rfloor+7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor-6$. Zatem rozmiar podtablicy $A[1\ldots i-1]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor+7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor\frac{3n}{10}\rfloor-6$. Zatem rozmiar podtablicy $A[i+1\ldots n]$ wynosi co najwyżej $\lfloor\frac{7n}{10}\rfloor+7$.



‖ 3 11 **50** 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

$3 \cdot (\lceil \frac{1}{2} \cdot \lceil \frac{n}{5} \rceil \rceil - 2) \geq \lfloor \frac{3n}{10} \rfloor - 6$

$\lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1$

co najwyżej $(\lfloor \frac{7n}{10} \rfloor + 7)$ elementów ≤ 50

1	9	10	30	100
20	21	40	71	82
3	11	**50**	60	63
19	22	61	62	101
4	5	70	85	99
	79	80	81	

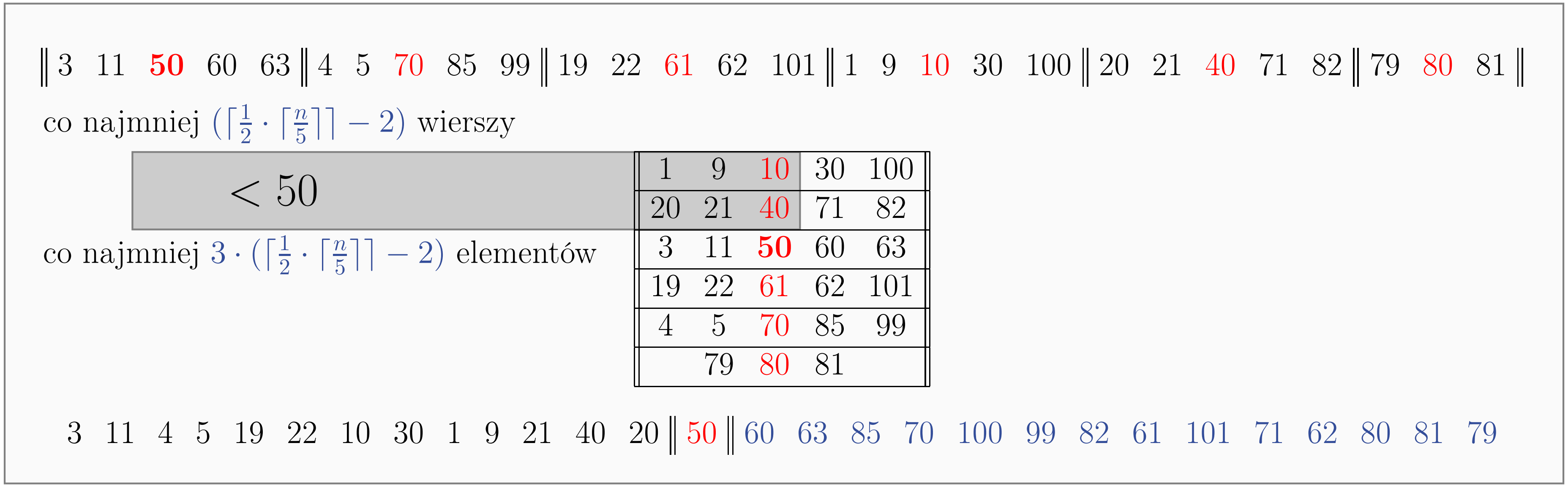
co najmniej $(\lceil \frac{1}{2} \cdot \lceil \frac{n}{5} \rceil \rceil - 2)$ wierszy

> 50

co najmniej $3 \cdot (\lceil \frac{1}{2} \cdot \lceil \frac{n}{5} \rceil \rceil - 2)$ elementów

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

- ▶ Krok 6. Rekurencja dla podtablicy $A[1 \ldots i-1]$ lub $A[i+1 \ldots n]$: $T(\lfloor \frac{7n}{10} \rfloor + 7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[1 \ldots i-1]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[i+1 \ldots n]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.



‖ 3 11 **50** 60 63 ‖ 4 5 70 85 99 ‖ 19 22 61 62 101 ‖ 1 9 10 30 100 ‖ 20 21 40 71 82 ‖ 79 80 81 ‖

co najmniej $(\lceil \frac{1}{2} \cdot \lceil \frac{n}{5} \rceil \rceil - 2)$ wierszy

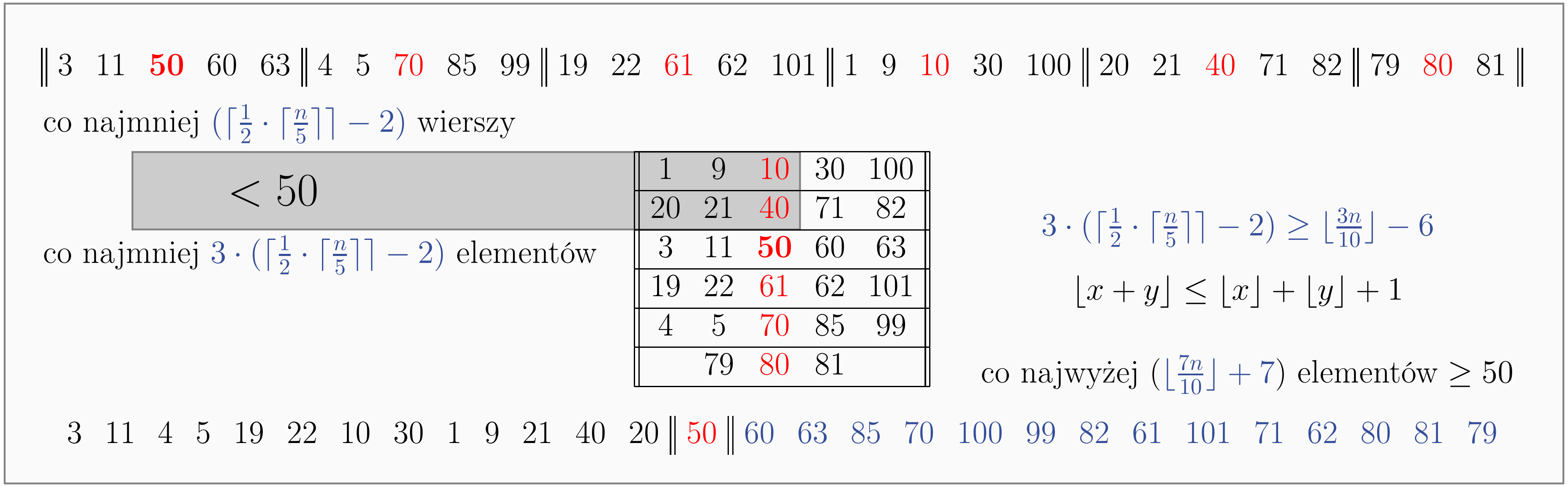
< 50

co najmniej $3 \cdot (\lceil \frac{1}{2} \cdot \lceil \frac{n}{5} \rceil \rceil - 2)$ elementów

1	9	10	30	100
20	21	40	71	82
3	11	**50**	60	63
19	22	61	62	101
4	5	70	85	99
	79	80	81	

3 11 4 5 19 22 10 30 1 9 21 40 20 ‖ 50 ‖ 60 63 85 70 100 99 82 61 101 71 62 80 81 79

- ▶ Krok 6. Rekurencja dla podtablicy $A[1 \ldots i-1]$ lub $A[i+1 \ldots n]$: $T(\lfloor \frac{7n}{10} \rfloor + 7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[1 \ldots i-1]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[i+1 \ldots n]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.



$\| 3 \quad 11 \quad \mathbf{50} \quad 60 \quad 63 \| 4 \quad 5 \quad 70 \quad 85 \quad 99 \| 19 \quad 22 \quad 61 \quad 62 \quad 101 \| 1 \quad 9 \quad 10 \quad 30 \quad 100 \| 20 \quad 21 \quad 40 \quad 71 \quad 82 \| 79 \quad 80 \quad 81 \|$

co najmniej $(\lceil \frac{1}{2} \cdot \lceil \frac{n}{5} \rceil \rceil - 2)$ wierszy

< 50

co najmniej $3 \cdot (\lceil \frac{1}{2} \cdot \lceil \frac{n}{5} \rceil \rceil - 2)$ elementów

1	9	10	30	100
20	21	40	71	82
3	11	**50**	60	63
19	22	61	62	101
4	5	70	85	99
	79	80	81	

$$3 \cdot (\lceil \tfrac{1}{2} \cdot \lceil \tfrac{n}{5} \rceil \rceil - 2) \geq \lfloor \tfrac{3n}{10} \rfloor - 6$$

$$\lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1$$

co najwyżej $(\lfloor \frac{7n}{10} \rfloor + 7)$ elementów ≥ 50

$3 \quad 11 \quad 4 \quad 5 \quad 19 \quad 22 \quad 10 \quad 30 \quad 1 \quad 9 \quad 21 \quad 40 \quad 20 \| 50 \| 60 \quad 63 \quad 85 \quad 70 \quad 100 \quad 99 \quad 82 \quad 61 \quad 101 \quad 71 \quad 62 \quad 80 \quad 81 \quad 79$

- Krok 6. Rekurencja dla podtablicy $A[1 \ldots i-1]$ lub $A[i+1 \ldots n]$: $T(\lfloor \frac{7n}{10} \rfloor + 7)$.
  - ↳ Liczba elementów w A większych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[1 \ldots i-1]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.
  - ↳ Liczba elementów w A mniejszych od m wynosi przynajmniej $\lfloor \frac{3n}{10} \rfloor - 6$. Zatem rozmiar podtablicy $A[i+1 \ldots n]$ wynosi co najwyżej $\lfloor \frac{7n}{10} \rfloor + 7$.

Złożoność czasowa algorytmu

- Krok 1. Sortowanie przez wstawianie tablicy A, gdy $|A| \leq 140$: $\Theta(1)$.
- Krok 2. Podział tablicy na grupy $A_1, \ldots, A_{\lceil \frac{n}{5} \rceil}$: $\Theta(n)$.
- Krok 3. Wyznaczenie mediany w każdej z grup (sortowanie): $\lceil \frac{n}{5} \rceil \cdot \Theta(1) = \Theta(n)$.
- Krok 4. Rekurencyjne wyznaczenie mediany m zbioru $\{m_1, \ldots, m_{\lceil \frac{n}{5} \rceil}\}$: $T(\lceil \frac{n}{5} \rceil)$.
- Krok 5. Podział tablicy A względem mediany median: $\Theta(n)$.
- Krok 6. Rekurencja dla podtablicy $A[1 \ldots i-1]$ lub $A[i+1 \ldots n]$: $T(\lfloor \frac{7n}{10} \rfloor + 7)$.

$$T(n) = \begin{cases} \Theta(1) & \text{dla } n \leq 140; \\ T(\lceil \frac{n}{5} \rceil) + T(\lfloor \frac{7n}{10} \rfloor + 7) + \Theta(n) & \text{w przeciwnym wypadku,} \end{cases}$$

Można wykazać, że rozwiązaniem powyższej zależności jest $T(n) = \Theta(n)$.

Twierdzenie 4.2. (Blum, Floyd, Pratt, Rivest, Tarjan 1973)
Problem selekcji można rozwiązać w czasie liniowym.

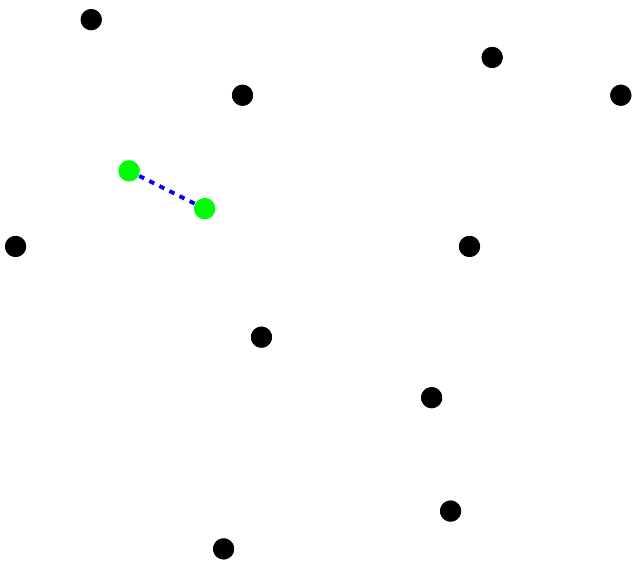
4.3 PARA NAJBLIŻSZYCH PUNKTÓW

Problem. (Para najbliższych punktów na płaszczyźnie)

Wejście: *Zbiór S punktów na płaszczyźnie $\mathcal{E}^2$.*

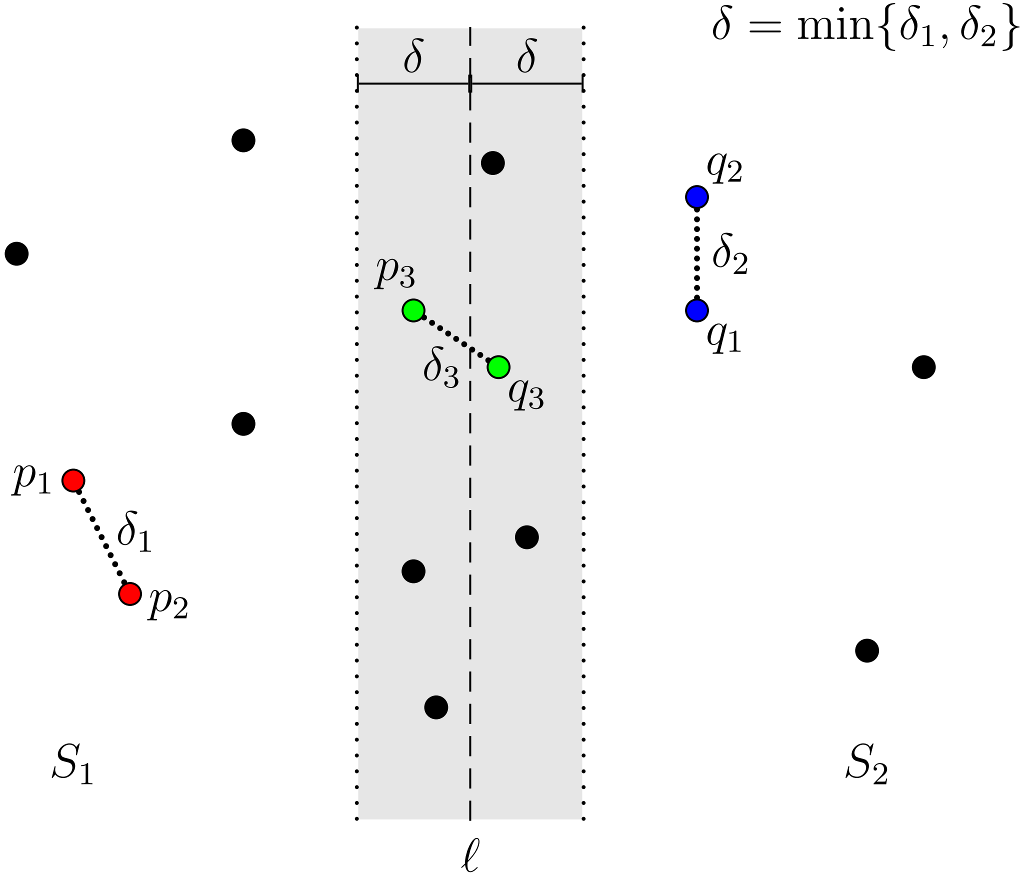
Wyjście: *Para najbliższych punktów z S.*

F.P. Preparata, M.I. Shamos
Geometria obliczeniowa – wprowadzenie
rozdział 5.4, Helion (2003)



Algorytm naiwny – sprawdzenie wszystkich par – czas rzędu $\Theta(n^2)$.

Para najbliższych punktów na płaszczyźnie $\mathbb{E}^2$



Rozwiązaniem problemu jest para, która należy do któregoś ze zbiorów:

- $S_1 \times S_1$;
- $S_2 \times S_2$;
- $(S_1 \times S_2) \cap$ (δ-otoczenie prostej ℓ).

Wstępne przetwarzanie. /Czas $O(n \log n)$; wykonywane tylko raz./

S_x – posortowany zbiór S punktów wejściowych względem współrzędnej x.
S_y – posortowany zbiór S punktów wejściowych względem współrzędnej y.

Zbiór wejściowy S reprezentowany jest przez parę (S_x, S_y).

Uwaga. Niech $p, q \in \mathbb{R}^2$ będą dwoma różnymi punktami.

- $\mathrm{x}(p)$ – odcięta punktu p;
- $\mathrm{y}(p)$ – rzędna punktu q.

W przypadku, gdy punkty nie mają różnych odciętych, sortowanie zbioru S_x należy wykonać w porządku leksykograficznym '$\prec$': *p jest mniejszy od q* (ozn. $p \prec q$), jeśli $\mathrm{x}(p) < \mathrm{x}(q)$ albo $\mathrm{x}(p) = \mathrm{x}(q)$ oraz $\mathrm{y}(p) < \mathrm{y}(q)$.

Wstępne przetwarzanie. /Czas $O(n \log n)$; wykonywane tylko raz./

S_x – posortowany zbiór S punktów wejściowych względem współrzędnej x.
S_y – posortowany zbiór S punktów wejściowych względem współrzędnej y.

Zbiór wejściowy S reprezentowany jest przez parę (S_x, S_y).

Idea algorytmu /opartego na metodzie „dziel i zwyciężaj”/

1. Podziel zbiór S na dwa (prawie) równoliczne podzbiory S_1 i S_2 (wraz z odpowiednią reprezentacją przez pary uporządkowanych zbiorów) mające tę własność, że dla wszystkich $p \in S_1$ i $q \in S_2$ zachodzi $p \prec q$.

Niech ℓ będzie pionową prostą przechodzącą przez największy punkt w S_1.

2. Rozwiąż rekurencyjnie problem w S_1 i S_2. Niech $\{p_1, p_2\}$ i $\{q_1, q_2\}$ będą rozwiązaniami problemu w S_1 i w S_2.

$$\delta := \min\{d_{\mathcal{E}}(p_1, p_2), (d_{\mathcal{E}}(q_1, q_2)\}.$$

3. Niech $S_1(\delta, \ell)$ będzie zbiorem tych punktów z S_1, które znajdują się nie dalej niż δ od ℓ; analogicznie, niech $S_2(\delta, \ell)$ będzie zbiorem tych punktów z S_2, które znajdują się nie dalej niż δ od prostej ℓ.

Wyznacz parę (p_3, q_3) najbliższych punktów w $S_3 = S_1(\delta, \ell) \times S_2(\delta, \ell)$.
Niech δ_3 będzie odległością punktów p_3 i q_3.

4. Zwróć parę punktów realizującą $\min\{\delta, \delta_3\}$.

Złożoność czasowa algorytmu

- ▶ Przetwarzanie wstępne: $O(n \log n)$ (uwzględnione tylko raz).
- ▶ Krok (1) /podział na zbiory S_1 i S_2/: $\Theta(n)$.
- ▶ Krok (2) /rekurencja/: $2 \cdot T(n/2)$.
- ▶ Krok (3) /scalanie/: $O(n)$.
  - ↳ Zbiory $S_1(\delta, \ell)$ i $S_2(\delta, \ell)$ są posortowane względem y.
  - ↳ Punkty z $S_1(\delta, \ell)$ (z $S_2(\delta, \ell)$) są w odległości przynajmniej δ od siebie.
- ▶ Kroki (4) /wyznaczenie ostatecznego rozwiązania/: $\Theta(1)$.

Otrzymujemy tym samym:

$$T(n) = \begin{cases} \Theta(1) & \text{jeśli } n \leq 2; \\ 2 \cdot T(n/2) + \Theta(n) & \text{w przeciwnym przypadku,} \end{cases}$$

a zatem $T(n) = \Theta(n \log n)$. /Twierdzenie o rekurencji uniwersalnej/

Twierdzenie 4.3. (Shamos, Hoey 1975) *Najbliższa para w zbiorze n punktów na płaszczyźnie może być wyznaczona w czasie $O(n \log n)$; jest to czas opytmalny.*

Złożoność czasowa algorytmu

- Przetwarzanie wstępne: $O(n \log n)$ (uwzględnione tylko raz).
- Krok (1) /podział na zbiory S_1 i S_2/: $\Theta(n)$.
- Krok (2) /rekurencja/: $2 \cdot T(n/2)$.
- **Krok (3) /scalanie/: $O(n)$.**
  - ↳ Zbiory $S_1(\delta, \ell)$ i $S_2(\delta, \ell)$ są posortowane względem y.
  - ↳ Punkty z $S_1(\delta, \ell)$ (z $S_2(\delta, \ell)$) są w odległości przynajmniej δ od siebie.
- Kroki (4) /wyznaczenie ostatecznego rozwiązania/: $\Theta(1)$.

Otrzymujemy tym samym:

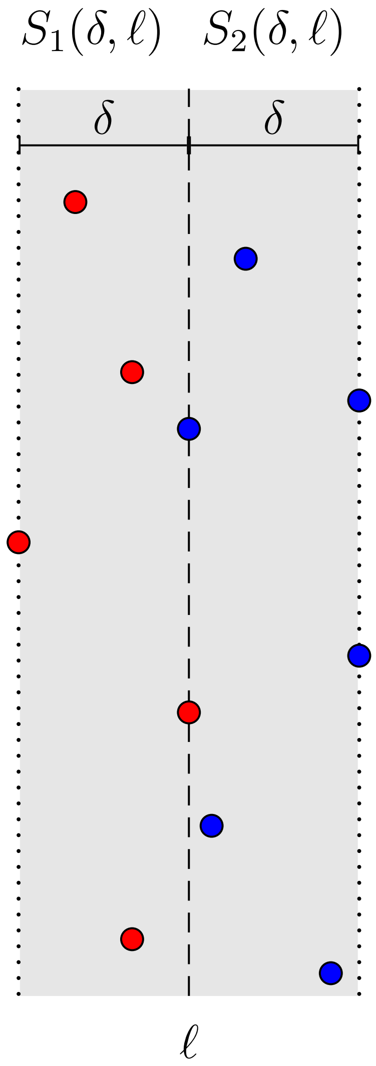
$$T(n) = \begin{cases} \Theta(1) & \text{jeśli } n \leq 2; \\ 2 \cdot T(n/2) + \Theta(n) & \text{w przeciwnym przypadku,} \end{cases}$$

a zatem $T(n) = \Theta(n \log n)$. /Twierdzenie o rekurencji uniwersalnej/

Twierdzenie 4.3. (Shamos, Hoey 1975) *Najbliższa para w zbiorze n punktów na płaszczyźnie może być wyznaczona w czasie $O(n \log n)$; jest to czas opytmalny.*

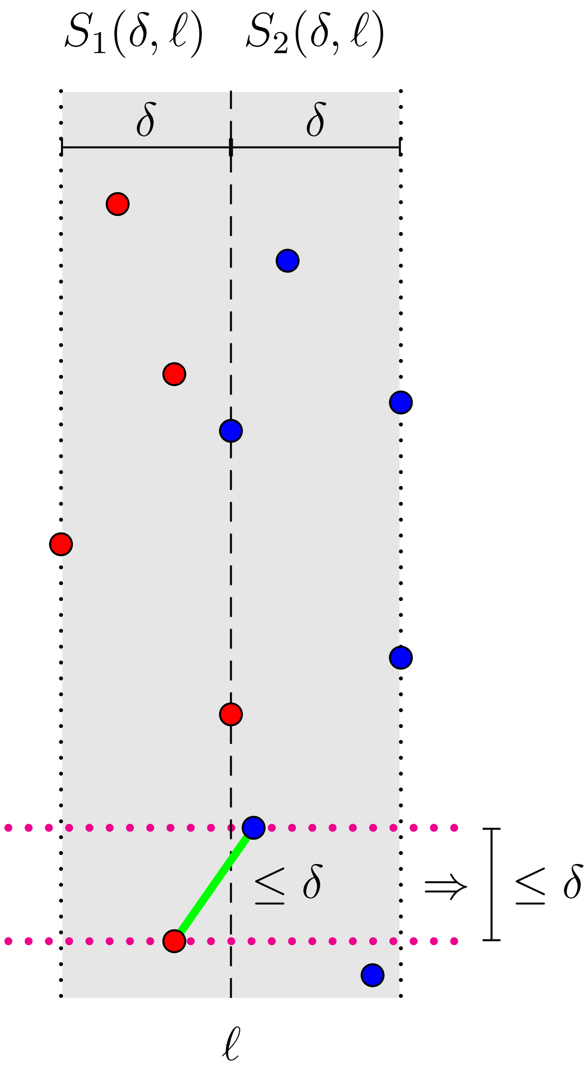
Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

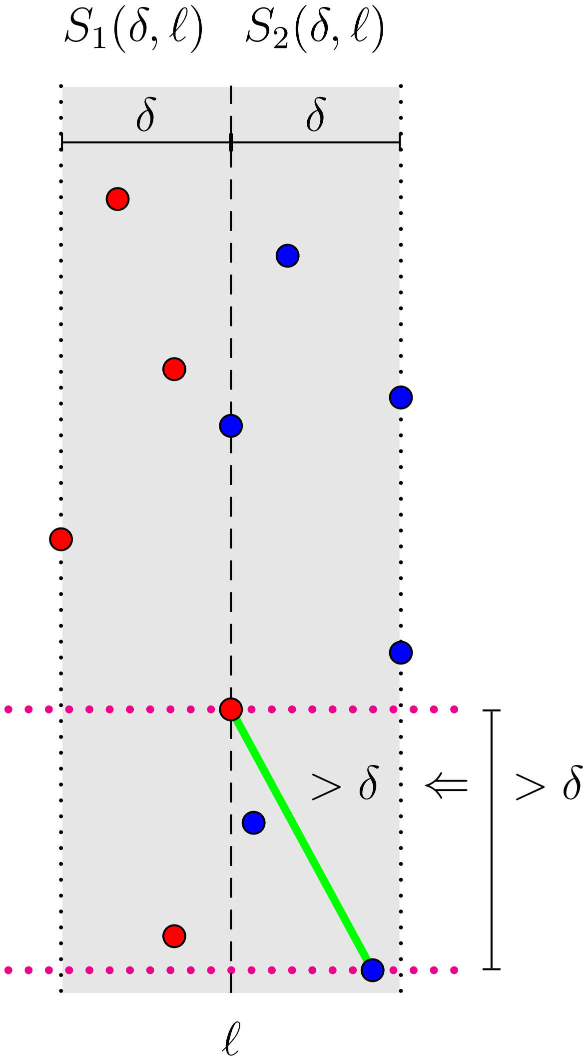
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

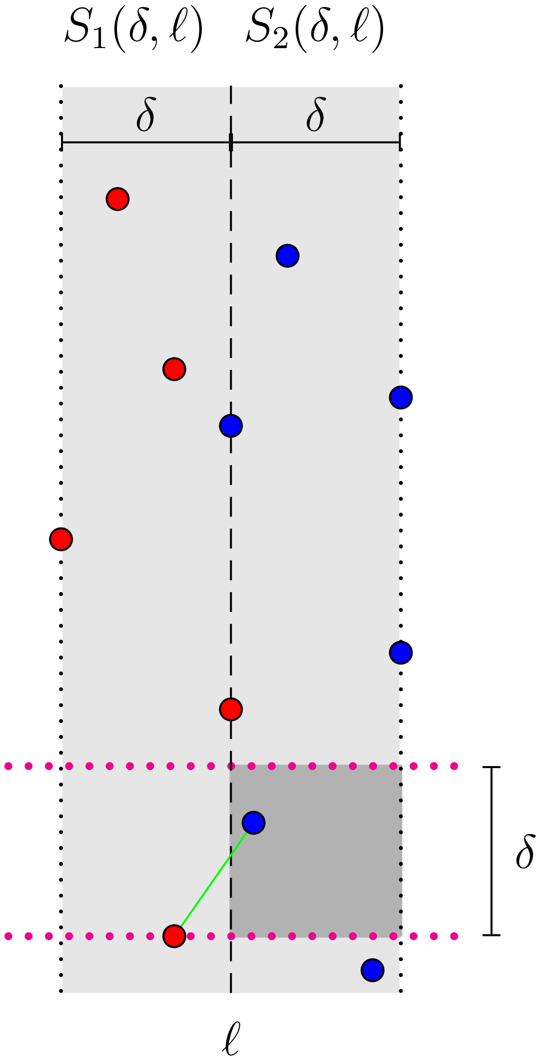
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

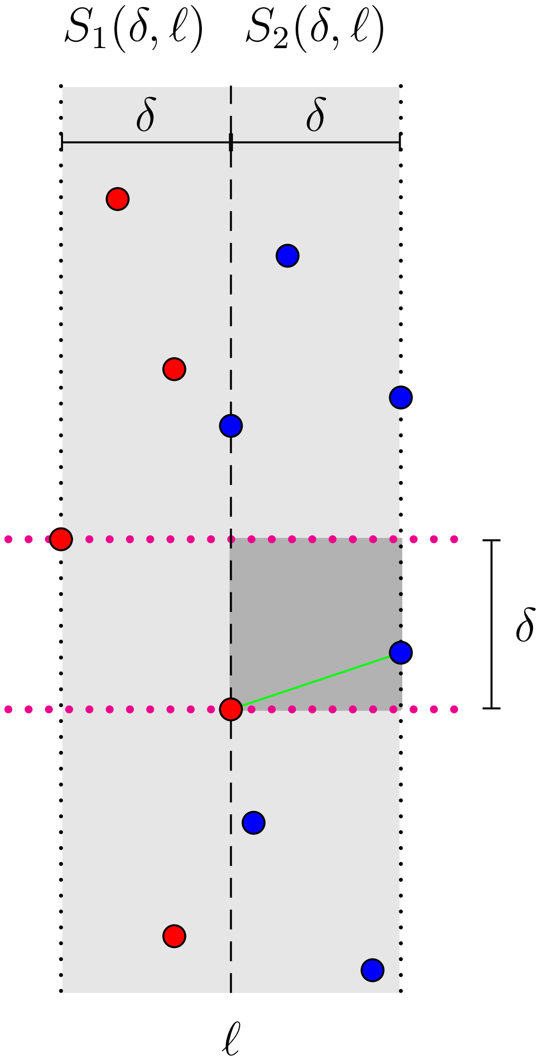
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

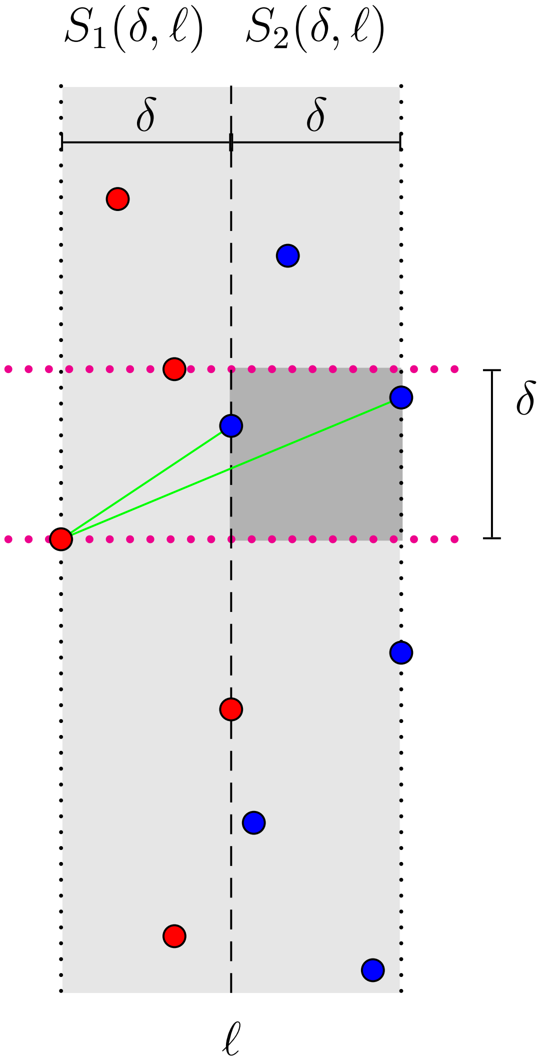
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

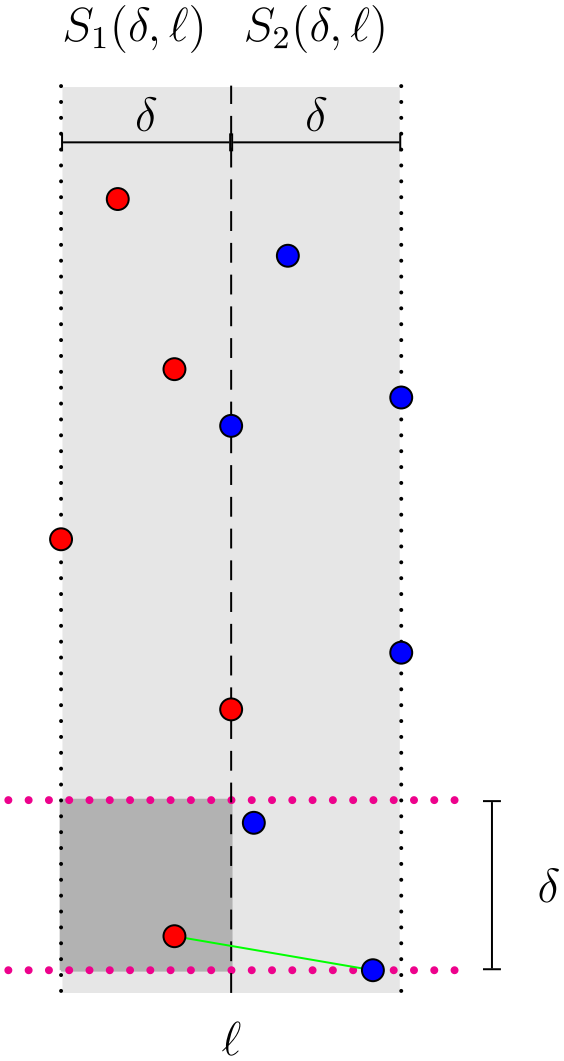
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

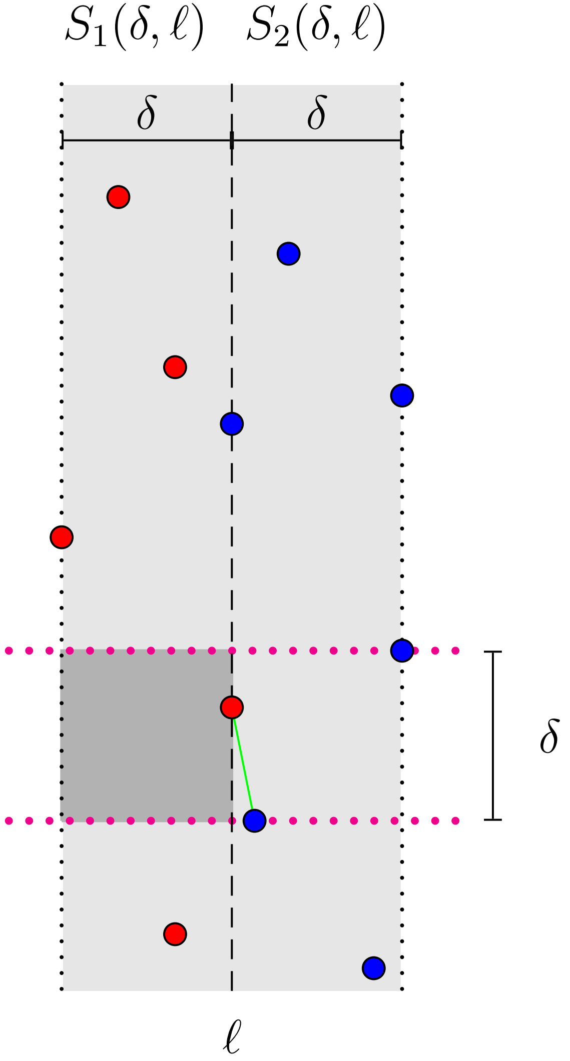
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

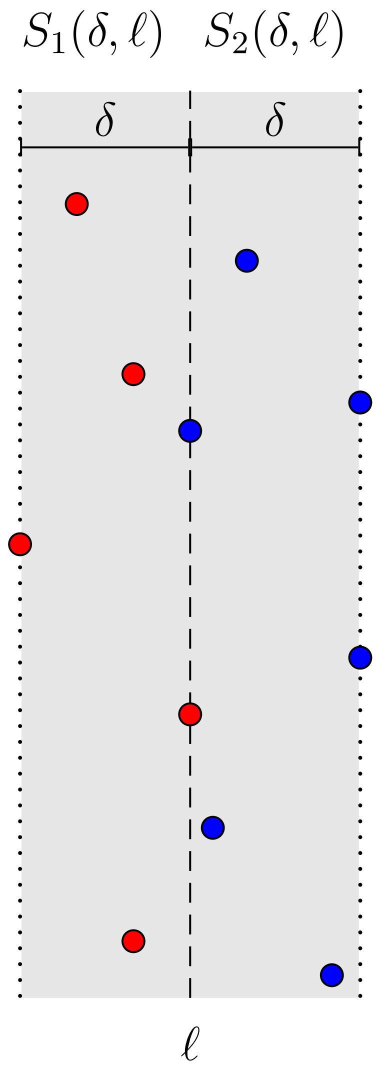
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

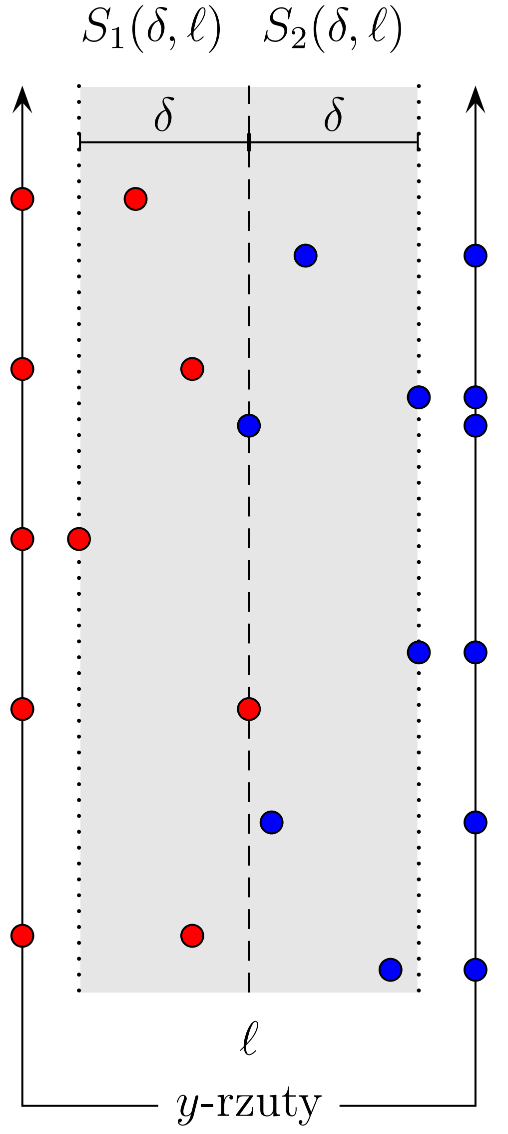
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- Niech $(p, q) \in R \times B$.
  ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.
- Szukane pary można wyznaczyć w czasie liniowym.
  ↳ $S_1(\delta, \ell)$ i $S_2(\delta, \ell)$ są posortowane względem y.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta,\ell) \times S_2(\delta,\ell)$.

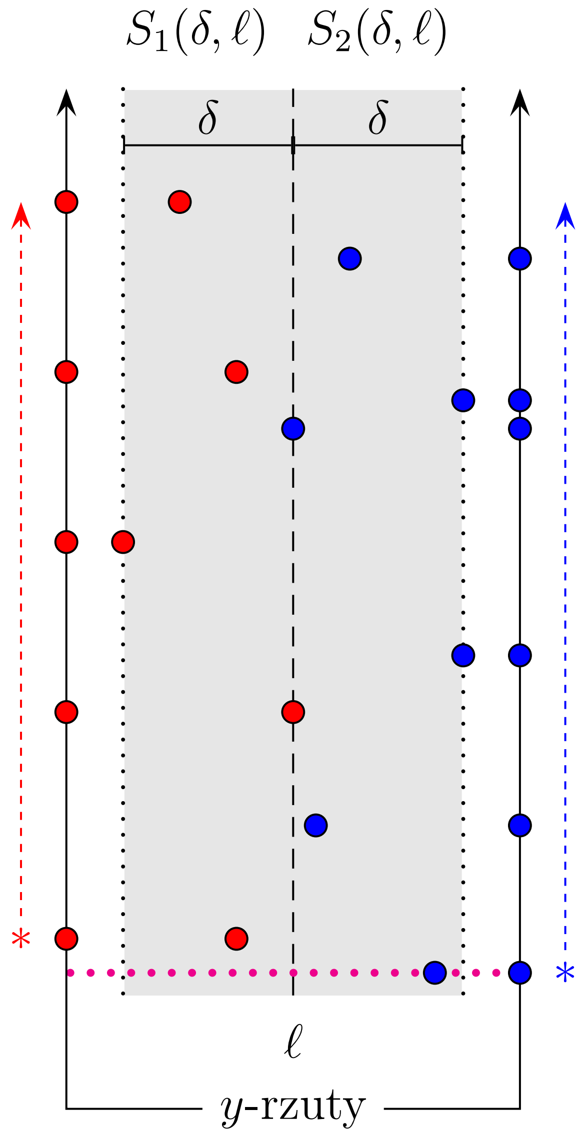
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p,q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p,q) > \delta$.
- Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p,q) \leq \delta$.
- Szukane pary można wyznaczyć w czasie liniowym.
  - ↳ $S_1(\delta,\ell)$ i $S_2(\delta,\ell)$ są posortowane względem y.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta,\ell) \times S_2(\delta,\ell)$.

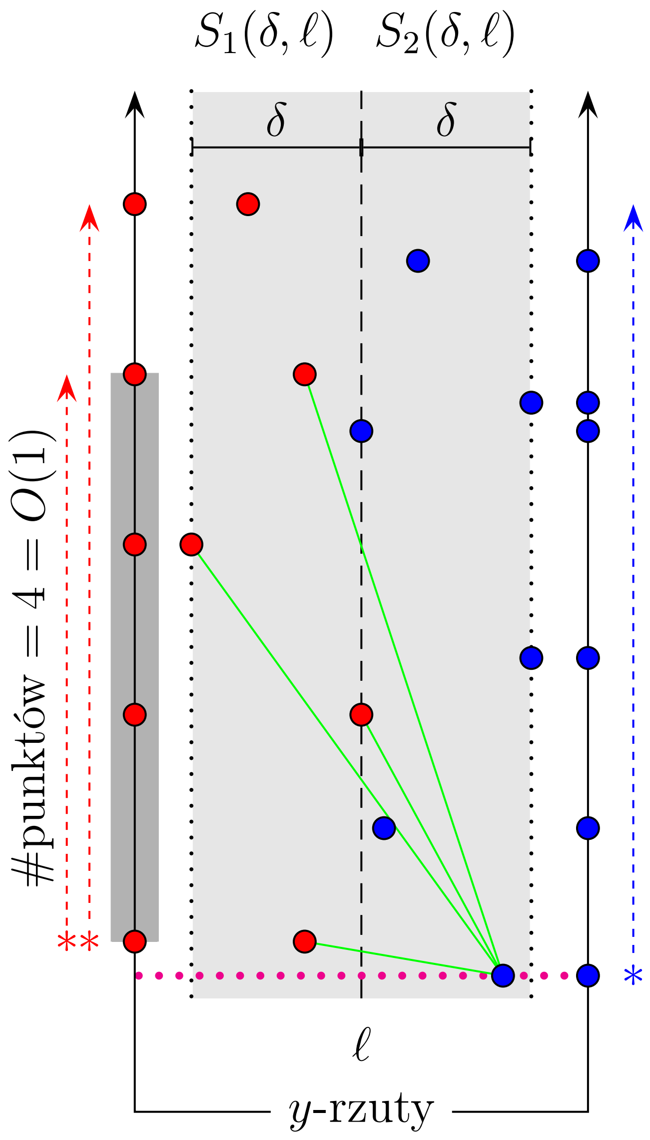
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- Niech $(p,q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p,q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p,q) > \delta$.
- Dla dowolnego punktu $p \in R$ $(q \in B)$ istnieją co najwyżej cztery punkty $q \in B$ $(p \in R)$ takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ $(\mathrm{y}(p) \geq \mathrm{y}(q))$ oraz $d_{\mathcal{E}}(p,q) \leq \delta$.
- Szukane pary można wyznaczyć w czasie liniowym.
  - ↳ $S_1(\delta,\ell)$ i $S_2(\delta,\ell)$ są posortowane względem y.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta, \ell) \times S_2(\delta, \ell)$.

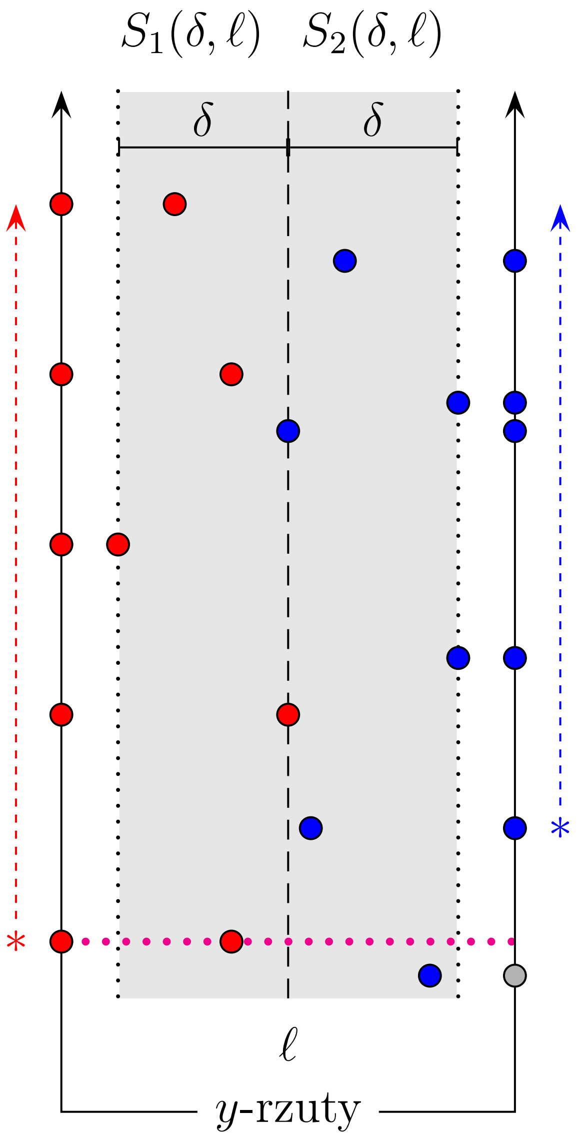
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.
- ▶ Szukane pary można wyznaczyć w czasie liniowym.
  - ↳ $S_1(\delta, \ell)$ i $S_2(\delta, \ell)$ są posortowane względem y.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta,\ell) \times S_2(\delta,\ell)$.

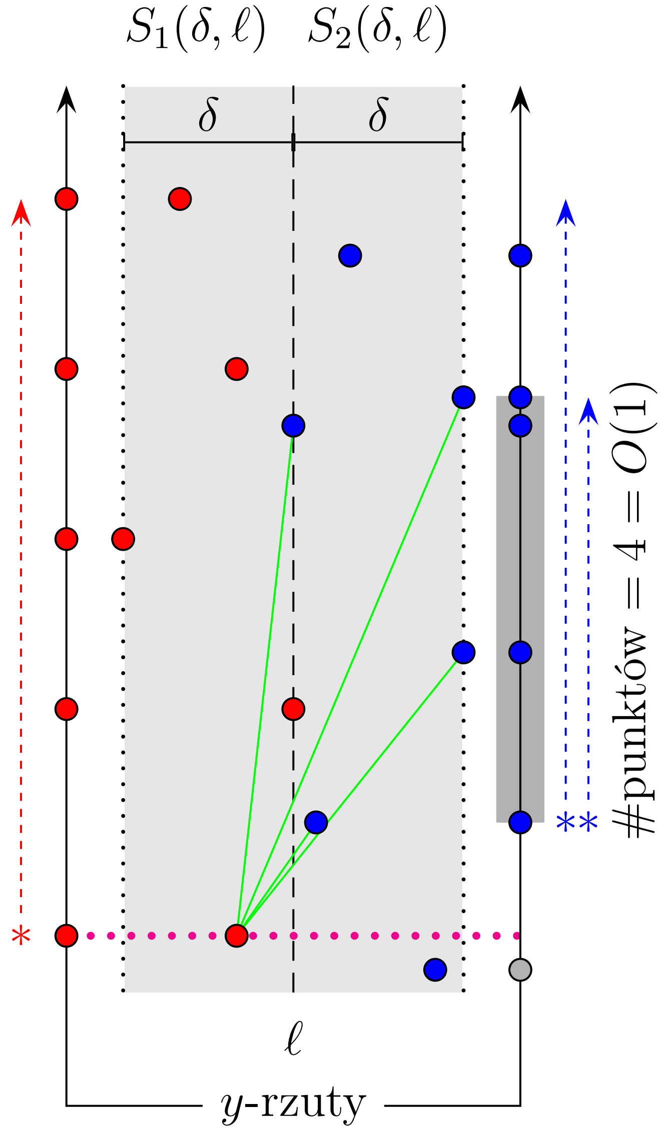
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p, q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p, q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p, q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p, q) \leq \delta$.
- ▶ Szukane pary można wyznaczyć w czasie liniowym.
  - ↳ $S_1(\delta,\ell)$ i $S_2(\delta,\ell)$ są posortowane względem y.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta,\ell) \times S_2(\delta,\ell)$.

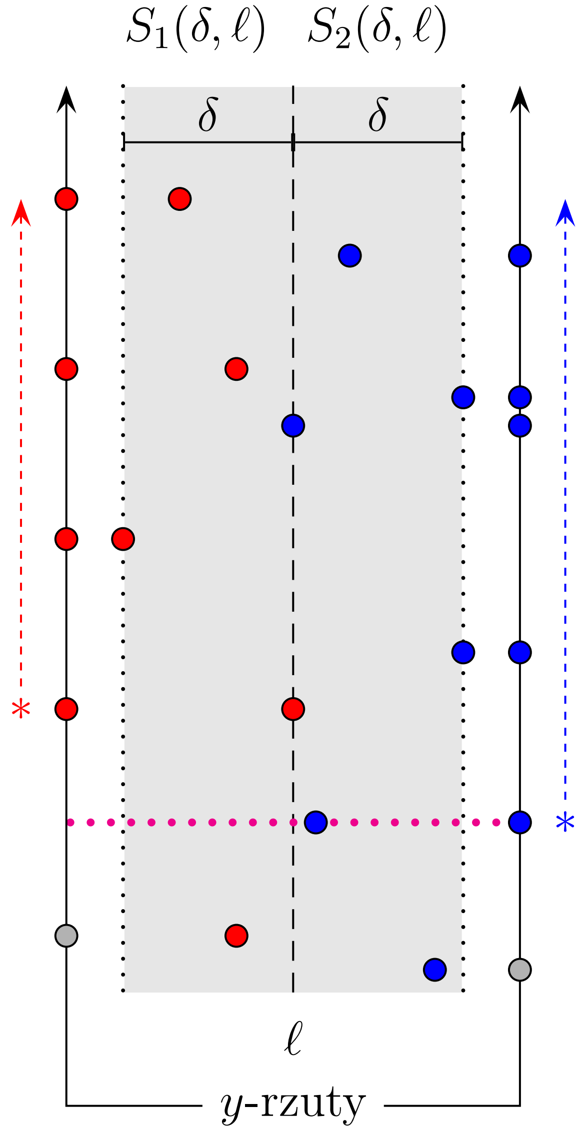
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r, b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p,q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p,q) \le \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \le \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p,q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \le \mathrm{y}(q)$ ($\mathrm{y}(p) \ge \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p,q) \le \delta$.
- ▶ Szukane pary można wyznaczyć w czasie liniowym.
  - ↳ $S_1(\delta,\ell)$ i $S_2(\delta,\ell)$ są posortowane względem y.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta,\ell) \times S_2(\delta,\ell)$.

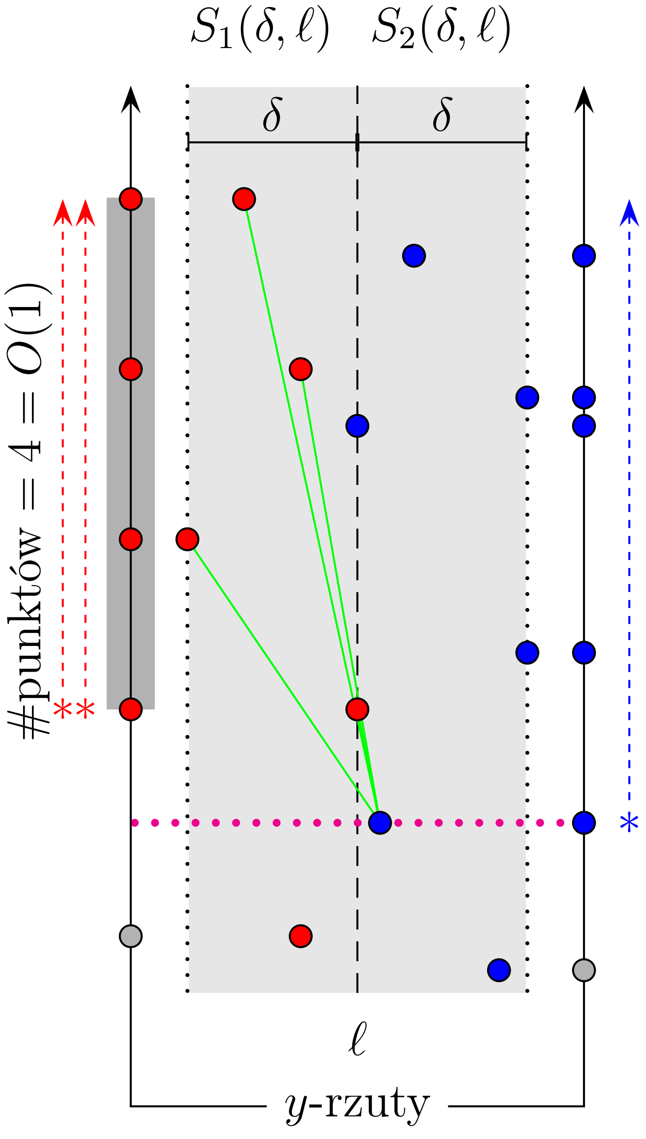
Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r,b), które znajdują się w odległości co najwyżej δ.*



- ▶ Niech $(p,q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p,q) \leq \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \leq \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p,q) > \delta$.
- ▶ Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \leq \mathrm{y}(q)$ ($\mathrm{y}(p) \geq \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p,q) \leq \delta$.
- ▶ Szukane pary można wyznaczyć w czasie liniowym.
  - ↳ $S_1(\delta,\ell)$ i $S_2(\delta,\ell)$ są posortowane względem y.

Krok (3). Wyznaczanie pary najbliższych punktów z $S_1(\delta,\ell) \times S_2(\delta,\ell)$.

Problem. *Mając dane dwa zbiory punktów R i B (czerwonych i niebieskich) z $\mathbb{E}^2$ w δ-otoczeniu pionowej prostej ℓ, $\delta > 0$, gdzie punkty z R (z B) znajdują się po lewej (po prawej) stronie lub na prostej ℓ oraz są w odległości przynajmniej δ od siebie, wyznacz wszystkie te dwukolorowe pary punktów (r,b), które znajdują się w odległości co najwyżej δ.*



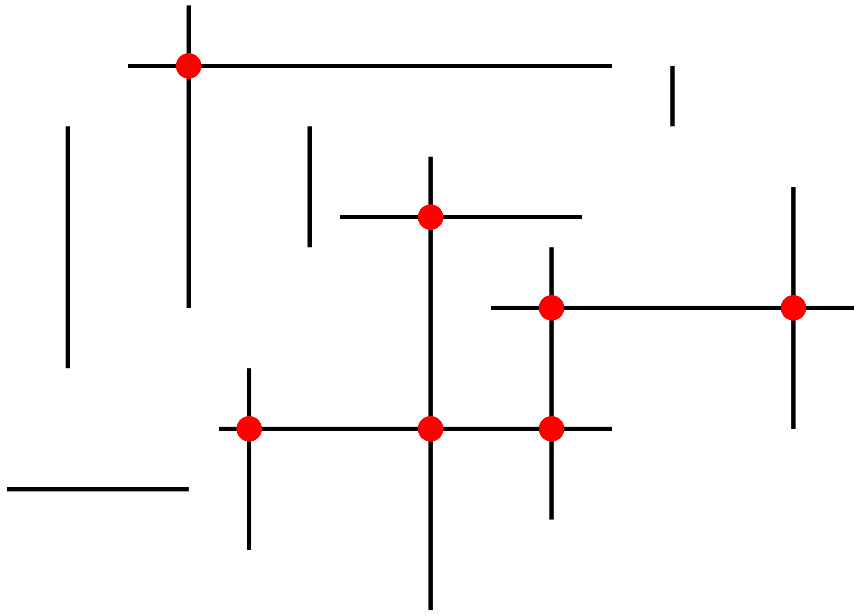
- Niech $(p,q) \in R \times B$.
  - ↳ Jeśli $d_{\mathcal{E}}(p,q) \le \delta$, to $|\mathrm{y}(p) - \mathrm{y}(q)| \le \delta$.
  - ↳ Jeśli $|\mathrm{y}(p) - \mathrm{y}(q)| > \delta$, to $d_{\mathcal{E}}(p,q) > \delta$.
- Dla dowolnego punktu $p \in R$ ($q \in B$) istnieją co najwyżej cztery punkty $q \in B$ ($p \in R$) takie, że $\mathrm{y}(p) \le \mathrm{y}(q)$ ($\mathrm{y}(p) \ge \mathrm{y}(q)$) oraz $d_{\mathcal{E}}(p,q) \le \delta$.
- Szukane pary można wyznaczyć w czasie liniowym.
  - ↳ $S_1(\delta,\ell)$ i $S_2(\delta,\ell)$ są posortowane względem y.

4.4 PRZECIĘCIA ODCINKÓW NA PŁASZCZYŹNIE

Problem. (Przecięcia odcinków)

Wejście: *Zbiór H rozłącznych odcinków poziomych oraz zbiór V rozłącznych odcinków pionowych na $\mathbb{R}^2$.*

Wyjście: *Punkty przecięć odcinków z $H \cup V$.*



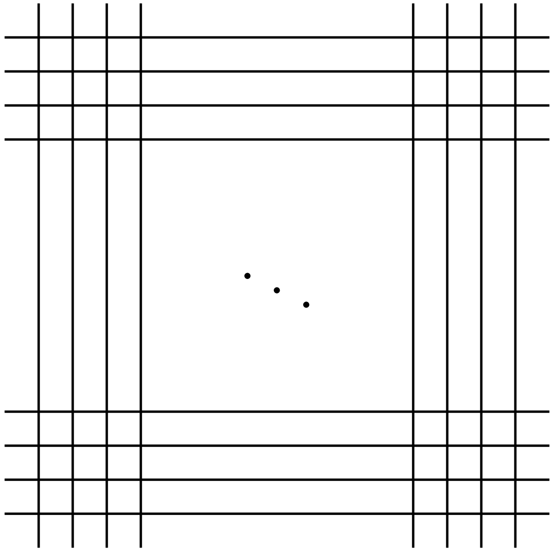
Algorytm naiwny – sprawdzenie wszystkich par odcinków – czas rzędu $\Theta(n^2)$.

4.4 PRZECIĘCIA ODCINKÓW NA PŁASZCZYŹNIE

Problem. (Przecięcia odcinków)

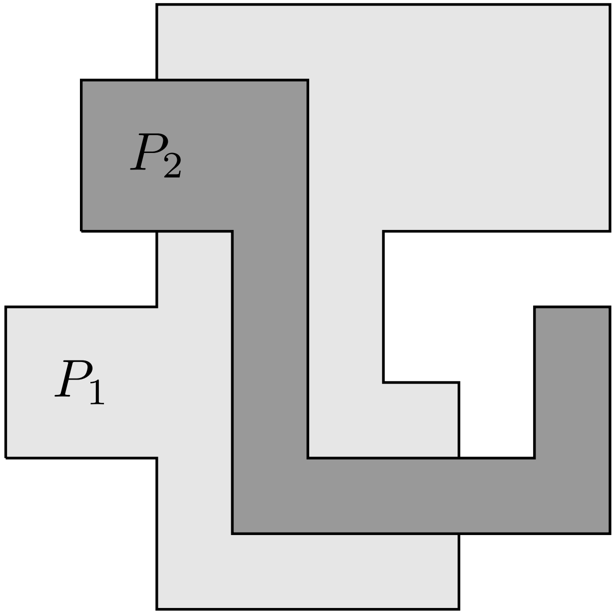
Wejście: *Zbiór H rozłącznych odcinków poziomych oraz zbiór V rozłącznych odcinków pionowych na $\mathbb{R}^2$.*

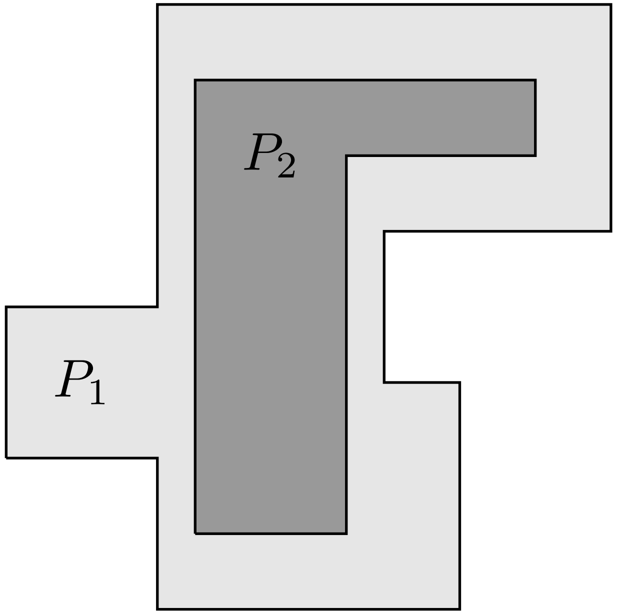
Wyjście: *Punkty przecięć odcinków z $H \cup V$.*



...czasami liczba przecięć jest rzędu $\Theta(n^2)$

Algorytm naiwny – sprawdzenie wszystkich par odcinków – czas rzędu $\Theta(n^2)$.





Zastosowanie. Przecinanie się ortogonalnych wielokątów prostych P_1 i P_2.

↳ Sprawdzenie, czy któraś z krawędzi P_1 przecina jakąś krawędź z P_2.

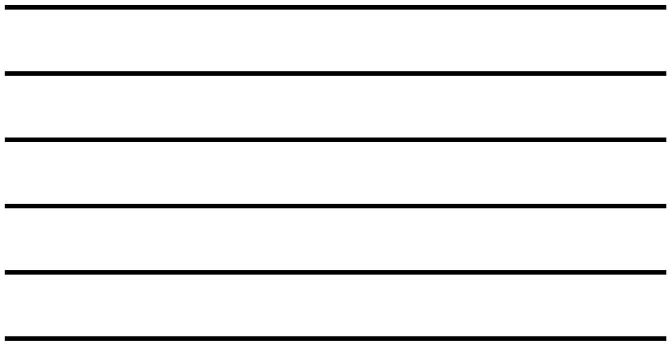
↳ Sprawdzenie, czy $P_1 \subset P_2$ lub $P_2 \subset P_1$.

Wystarczy sprawdzić, czy któryś z wierzchołków wielokąta P_1 należy do P_2 i na odwrót. /Problem *przynależności punktu do wielokąta*/

Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

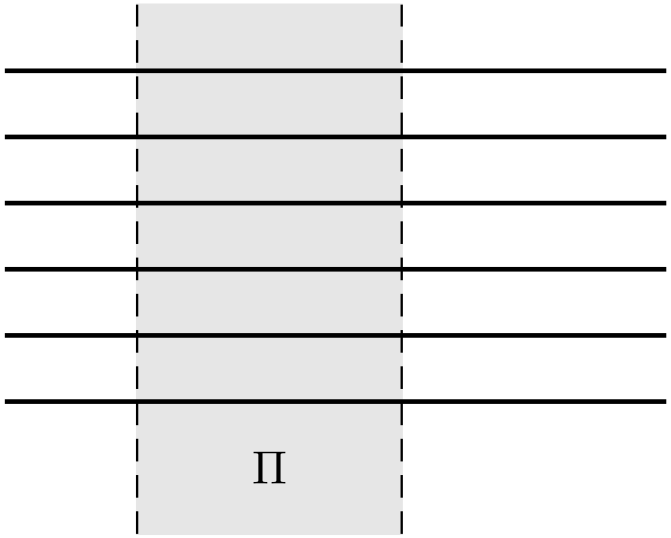
Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.



Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.

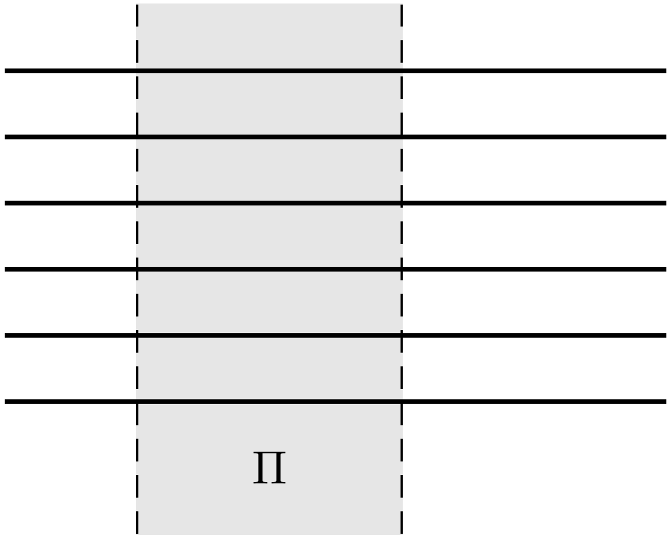


rozmiar instancji
w obszarze Π
jest ten sam

Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.



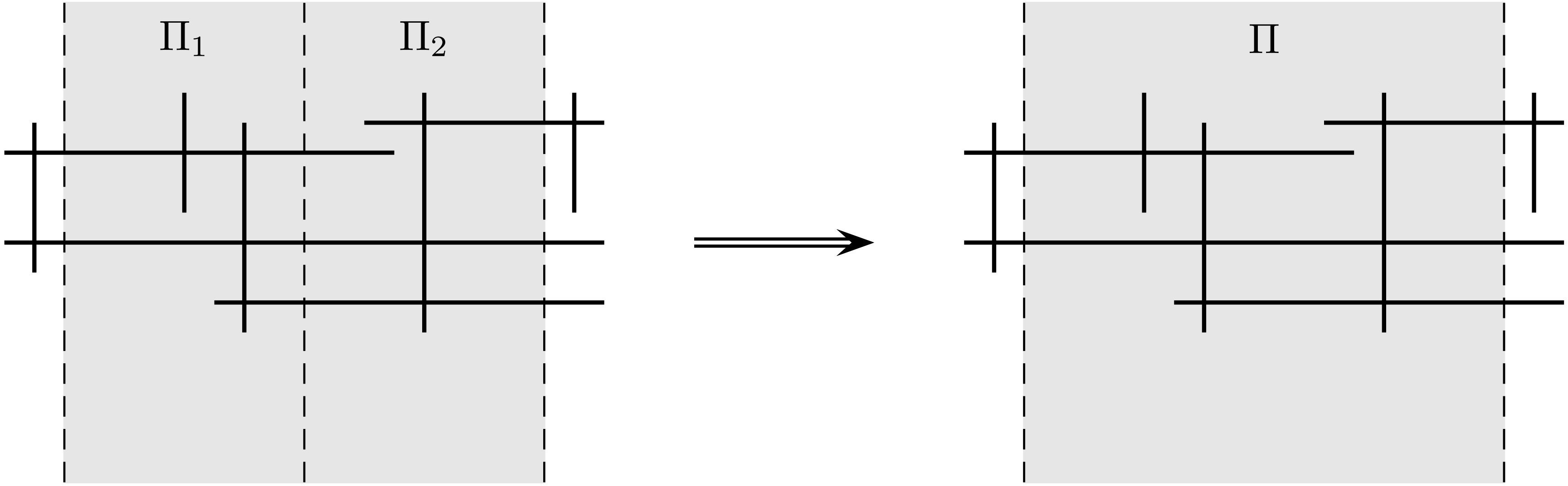
rozmiar instancji
w obszarze Π
jest ten sam

Niezmiennik.
Dla pionowego paska Π *algorytm zwraca wszystkie i tylko te przecięcia* $(\in \Pi)$ *par odcinków ze zbioru* $H \cup V$*, których przynajmniej jeden z końców leży w* Π.

Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.

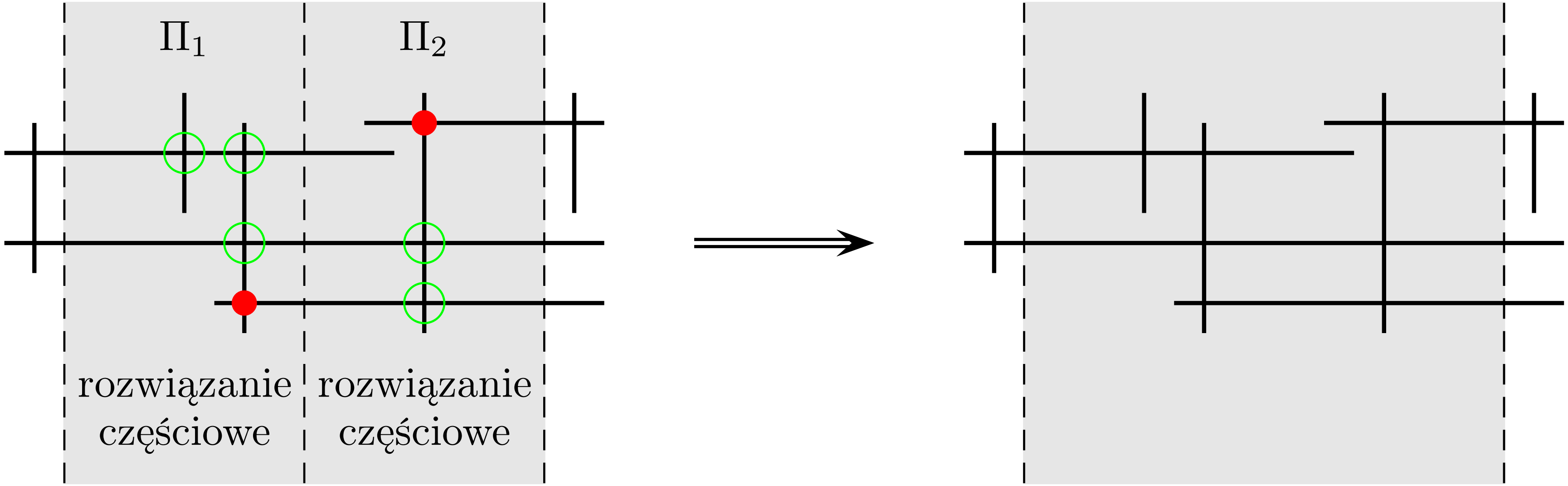


Niezmiennik.
Dla pionowego paska Π algorytm zwraca wszystkie i tylko te przecięcia ($\in \Pi$) par odcinków ze zbioru $H \cup V$, których przynajmniej jeden z końców leży w Π.

Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.

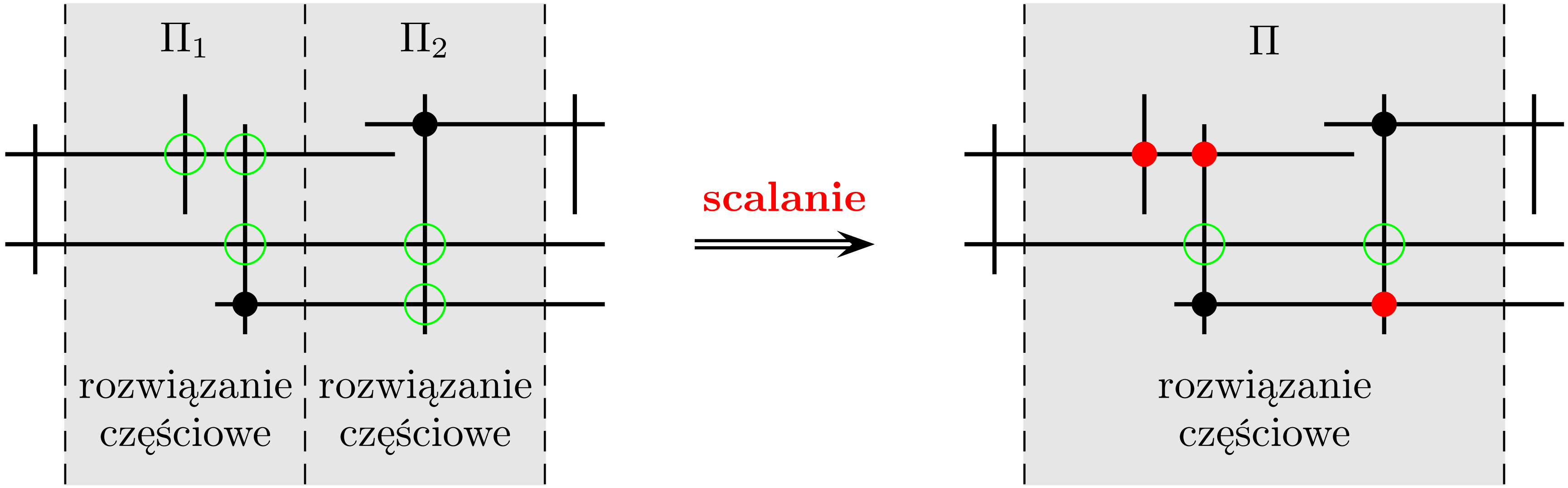


Niezmiennik.
Dla pionowego paska Π *algorytm zwraca wszystkie i tylko te przecięcia* $(\in \Pi)$ *par odcinków ze zbioru* $H \cup V$*, których przynajmniej jeden z końców leży w* Π*.*

Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.

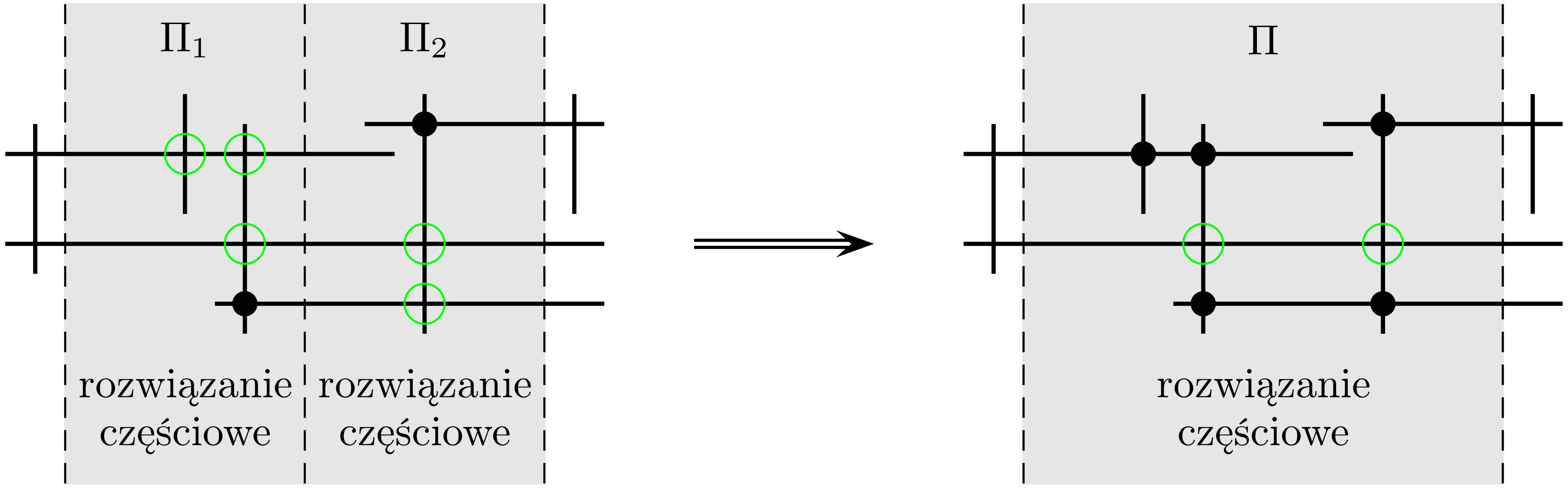


Niezmiennik.
Dla pionowego paska Π algorytm zwraca wszystkie i tylko te przecięcia ($\in \Pi$) par odcinków ze zbioru $H \cup V$, których przynajmniej jeden z końców leży w Π.

Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.



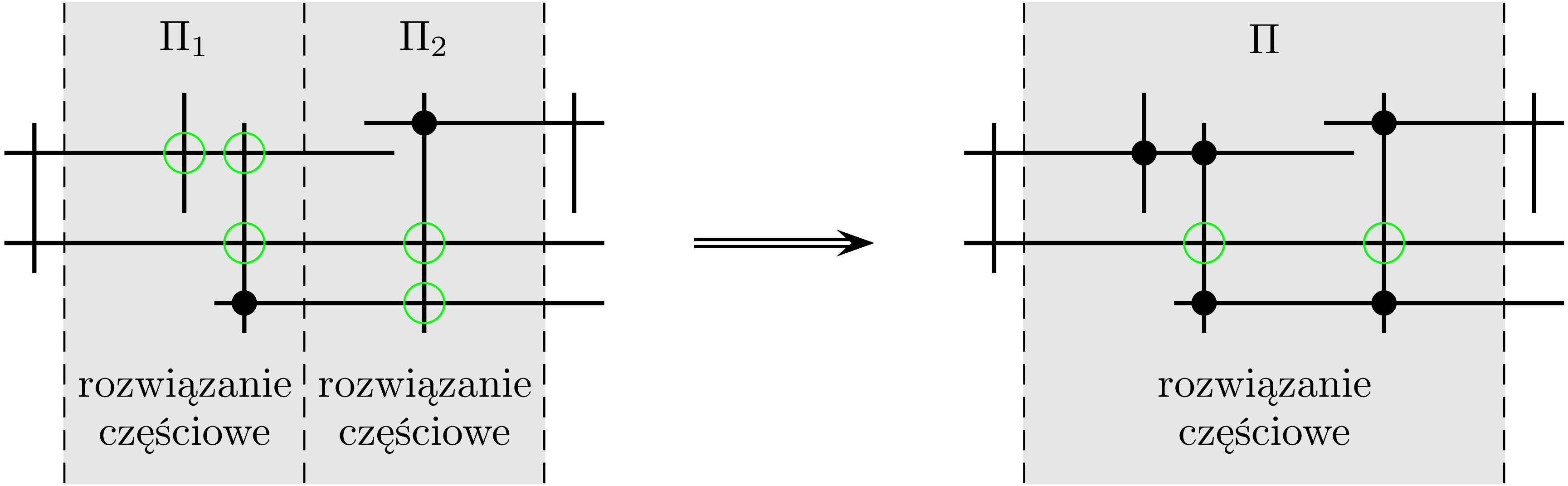
Niezmiennik.

Dla pionowego paska Π *algorytm zwraca wszystkie i tylko te przecięcia* ($\in \Pi$) *par odcinków ze zbioru* $H \cup V$, *których przynajmniej jeden z końców leży w* Π.

Algorytm „dziel i zwyciężaj”

R. H. Güting, D. Wood
Finding rectangle intersections by Divide-and-Conquer
IEEE Transactions on Computers C-33(7), 671-675 (1984)

Idea metody „dziel i zwyciężaj” polega na podzieleniu płaszczyzny na pionowe paski, w których rekurencyjnie wyznaczamy rozwiązania, tj. odpowiednie podzbiory przeciąć, a następnie „scaleniu” tych rozwiązań.

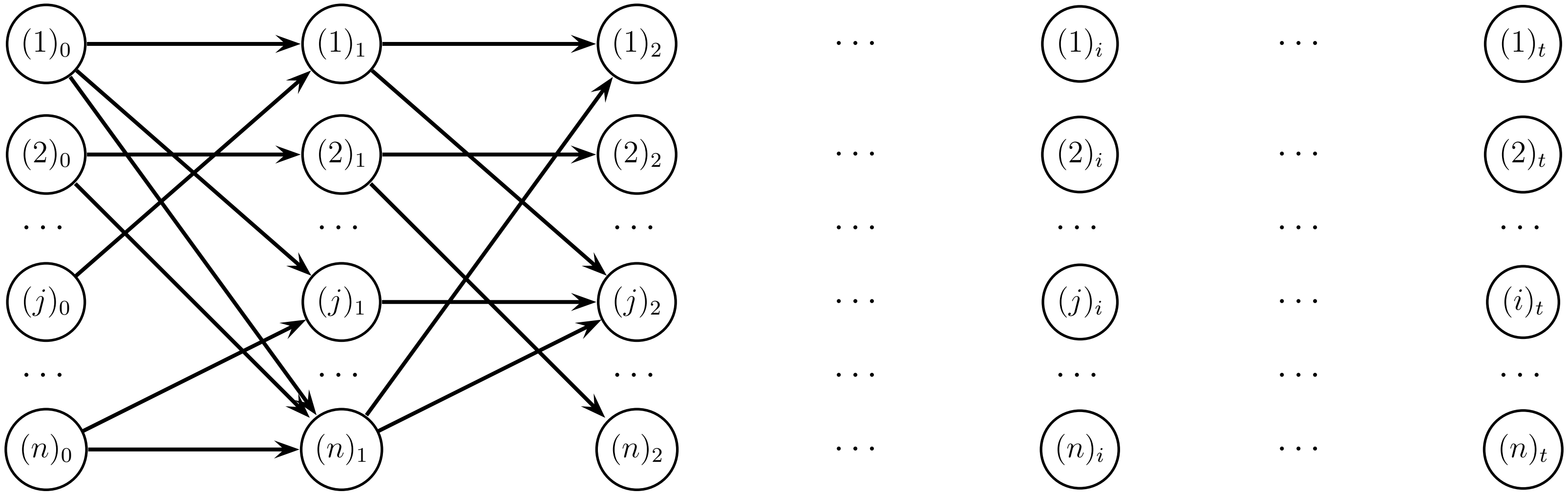


Twierdzenie 4.4. (Güting, Wood, 1984) *Problem wyznaczenia punktów przecięć w n-elementowym zbiorze pionowych i poziomych odcinków można rozwiązać w czasie $O(n \log n + k)$ i pamięci $O(n)$, gdzie k jest liczbą przecieć.*

4.5 REDUKCJA ZŁOŻONOŚCI PAMIĘCIOWEJ

J. Ian Munro, R. J. Ramirez
Reducing space requirements for shortest path problems
Operations Research 30, 1009-1113 (1982)

Graf warstwowy $D = (V, A, w)$ jest to skierowany graf ważony z funkcją wagową $w : A \to \mathbb{R}$, gdzie $V = V_0 \cup V_1 \cup \ldots \cup V_t$, $t \in \mathbb{N}$, a $V_i = \{(1)_i, (2)_i, \ldots, (n)_i\}$, $n \in \mathbb{N}$, dla każdego $0 \leq i \leq n$, oraz każdy łuk $a \in A$ jest postaci $((p)_i, (q)_{i+1})$ dla pewnego $0 \leq i \leq n - 1$.

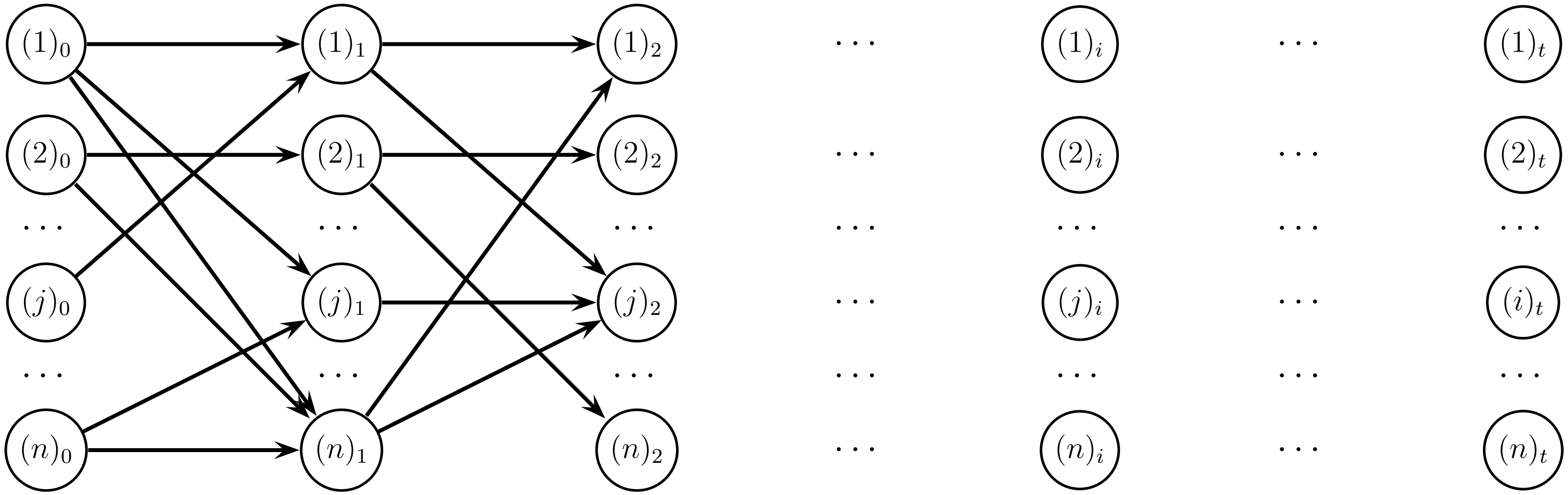


Problem. (Najkrótsza ścieżka w grafie warstwowym)

Wejście: *Graf warstwowy* $D = (V, E, w)$ *oraz dwa wierzchołki* $(p)_0, (q)_t \in V$.

Wyjście: *Najkrótsza* $((p)_0, (q)_t)$*-ścieżka w grafie* D *(o ile istnieje).*

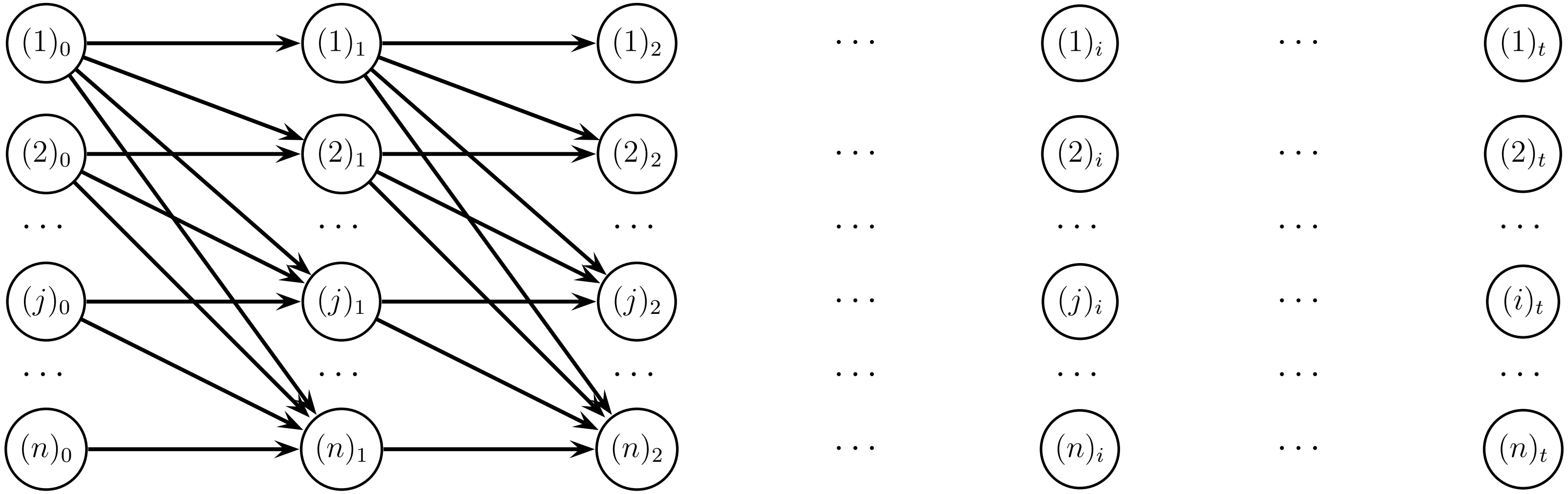
Rozmiar wejścia: $|V| = n \cdot t$, $|E| = O(n^2 \cdot t)$.



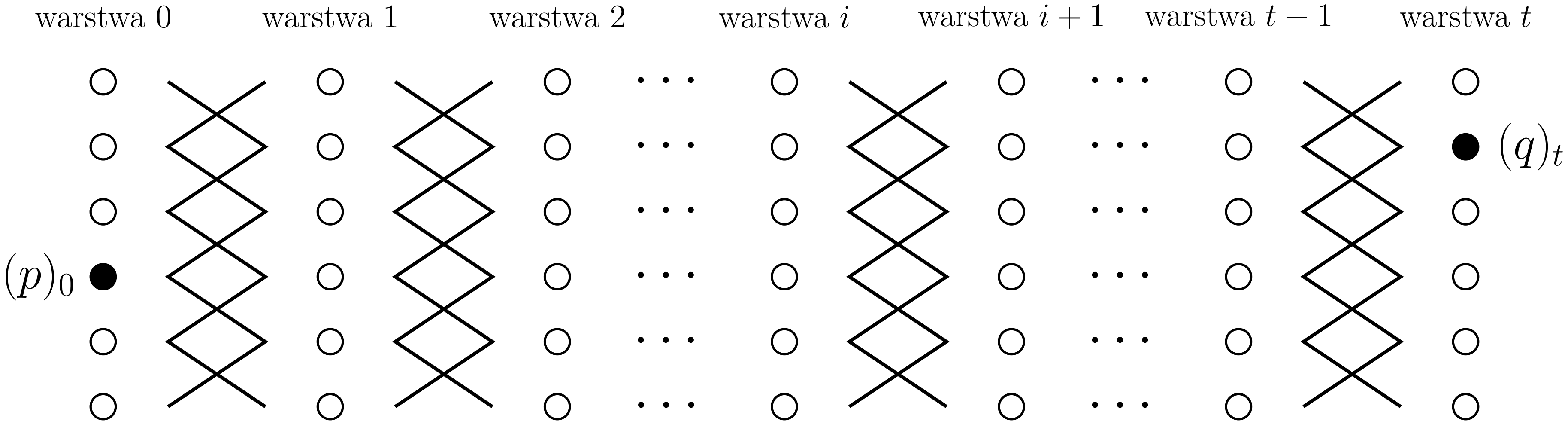
- Algorytm Bellmana-Forda (brak ujemnych cykli)
  - ↳ Złożoność czasowa: $O(|V| \cdot |E|) = O(n^3 \cdot t^2)$.
  - ↳ Złożoność pamięciowa: $O(|V| + |E|) = O(n^2 \cdot t)$.
- Algorytm Dijkstry (wagi nieujemne)
  - ↳ Złożoność czasowa: $O(|V| \cdot \log |V| + |E|) = O(n \cdot t \cdot \log(n \cdot t) + n^2 \cdot t)$.
  - ↳ Złożoność pamięciowa: $O(|V| + |E|) = O(n^2 \cdot t)$.

W szczególnych przypadkach zbiór krawędzi (a tym samym i cały graf) może nie być podany *explicite*, a wyrażony za pomocą reguły (wzoru) przejścia pomiędzy kolejnymi warstwami, np.:

łuk $((p)_i, (q)_{i+1}) \in A$ wtedy i tylko wtedy, gdy $p \leq q$.



Jako że rozmiar parametru t jest rzędu $\Theta(\log t)$, złożoność pamięciowa rzędu np. $\Theta(n^2 \cdot t)$ nie jest wielomianową względem parametru t.



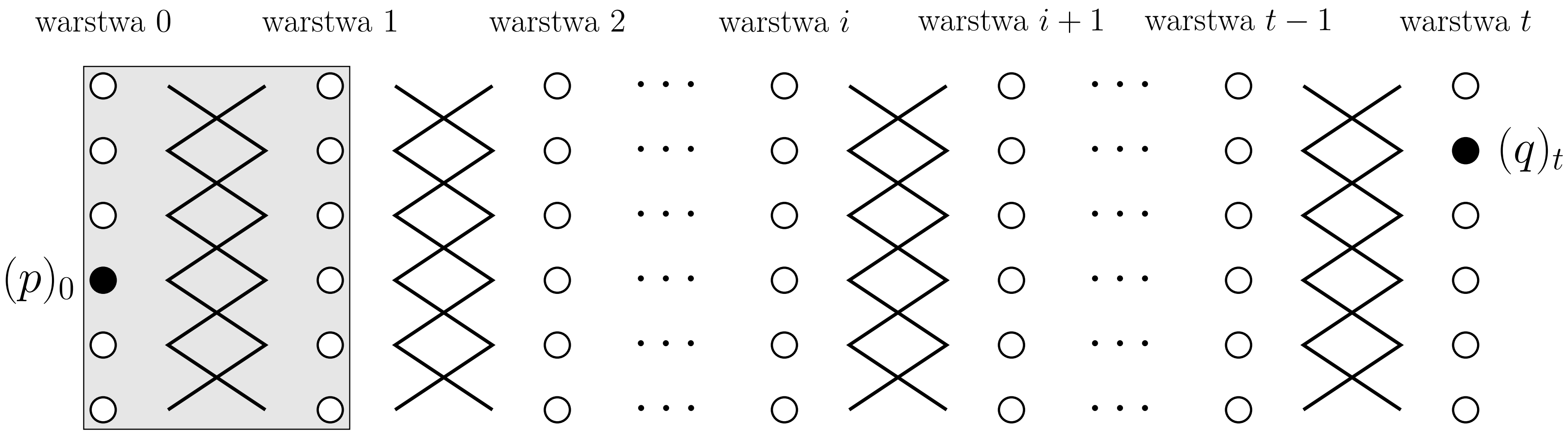
Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy” $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

(3) Wywołaj rekurencyjnie algorytm, aby wyznaczyć najkrótszą ścieżkę π_1 pomiędzy (p_0) i $(x)_{\lfloor \frac{t}{2} \rfloor}$ oraz najkrótszą ścieżkę π_2 pomiędzy $(x)_{\lfloor \frac{t}{2} \rfloor}$ i $(q)_t$.

(4) Rozwiązaniem jest $\pi_1 \cup \pi_2$.



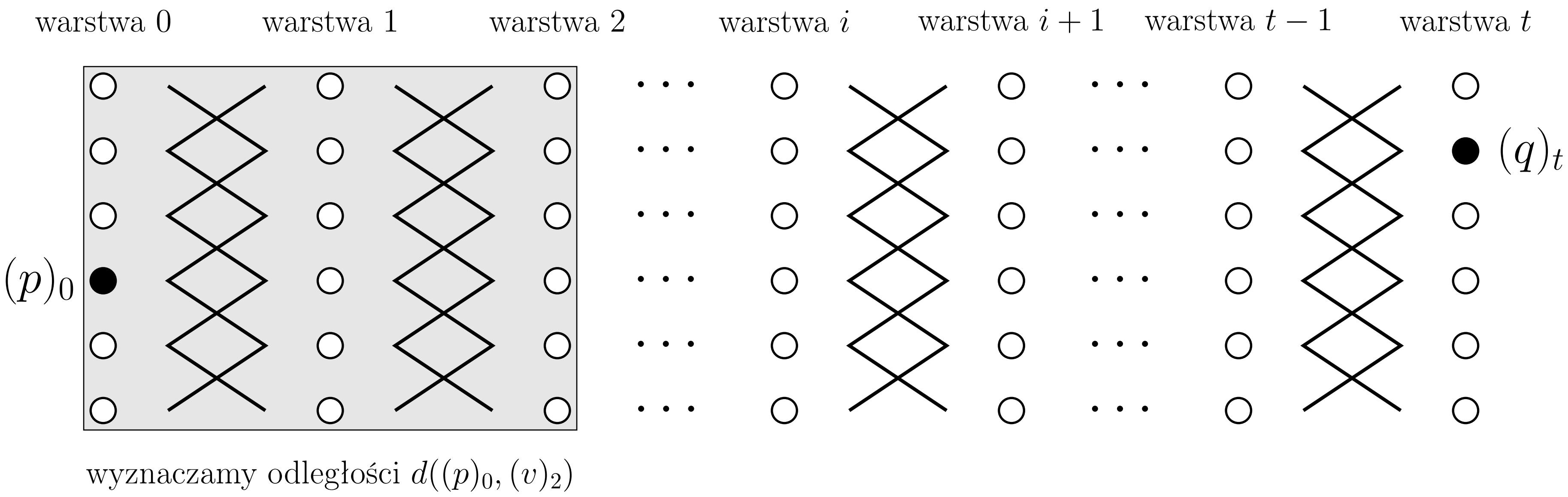
wyznaczamy odległości $d((p)_0, (v)_1)$
w oparciu o funkcję wagową w

Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

↳ Dla $i = 1, 2, \ldots, n$ wyznaczamy długości $d((p)_0, (v)_i)$ najkrótszych ścieżek z $(p)_0$ do $(v)_i$. A dokładnie, aby wyznaczyć odległości $d((p)_0, (v)_{i+1})$, korzystamy z (poprawnie wyznaczonych) odległości $d((p)_0, (u)_i)$:

$$d((p)_0, (v)_{i+1}) = \min\{d((p)_0, (u)_i) + w(((u)_i, (v)_{i+1})) : ((u)_i, (v)_{i+1}) \in A\}.$$

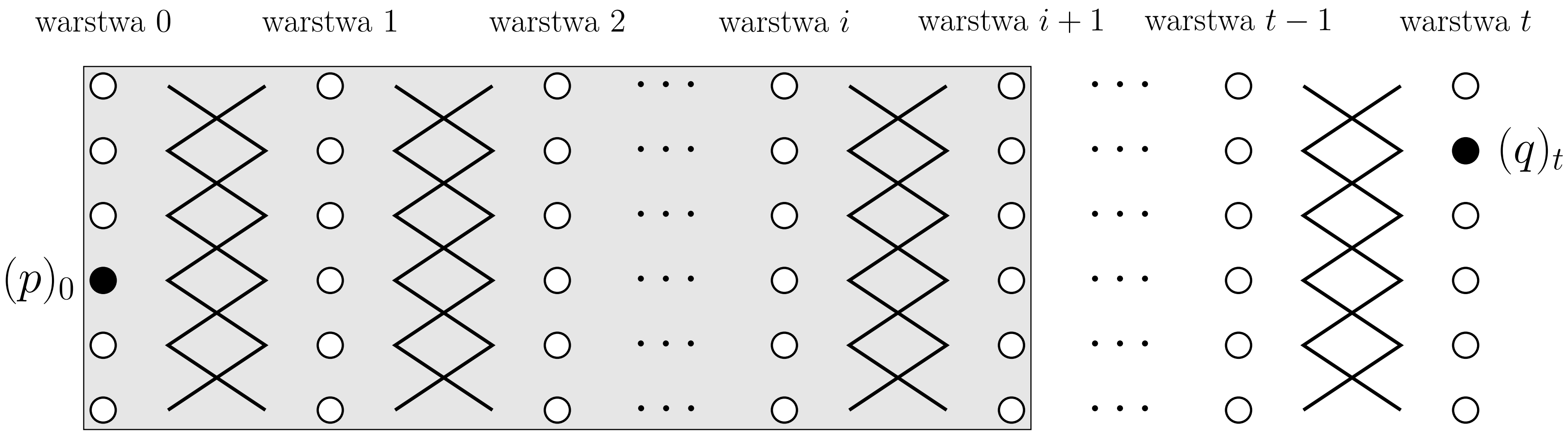


Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0,(q)_t)$-ścieżki π w grafie warstwowym.

↳ Dla $i = 1, 2, \ldots, n$ wyznaczamy długości $d((p)_0,(v)_i)$ najkrótszych ścieżek z $(p)_0$ do $(v)_i$. A dokładnie, aby wyznaczyć odległości $d((p)_0,(v)_{i+1})$, korzystamy z (poprawnie wyznaczonych) odległości $d((p)_0,(u)_i)$:

$$d((p)_0,(v)_{i+1}) = \min\{d((p)_0,(u)_i) + w(((u)_i,(v)_{i+1})) : ((u)_i,(v)_{i+1}) \in A\}.$$



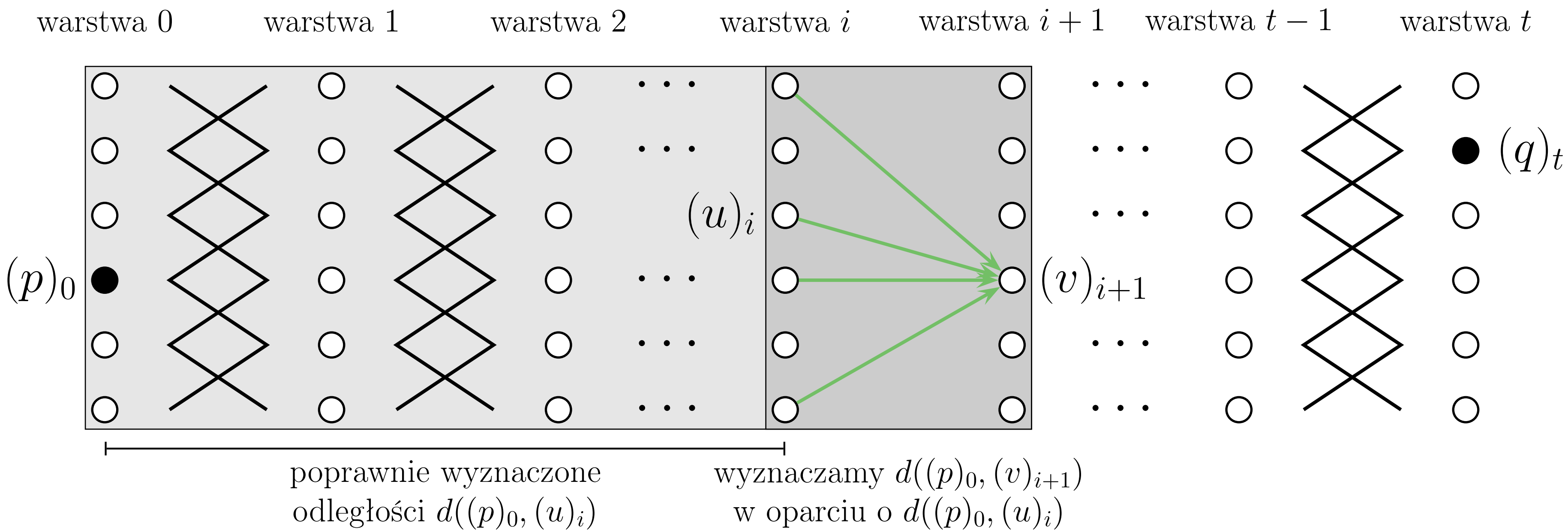
wyznaczamy odległości $d((p)_0,(v)_{i+1})$

Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0,(q)_t)$-ścieżki π w grafie warstwowym.

↳ Dla $i = 1, 2, \ldots, n$ wyznaczamy długości $d((p)_0,(v)_i)$ najkrótszych ścieżek z $(p)_0$ do $(v)_i$. A dokładnie, aby wyznaczyć odległości $d((p)_0,(v)_{i+1})$, korzystamy z (poprawnie wyznaczonych) odległości $d((p)_0,(u)_i)$:

$$d((p)_0,(v)_{i+1}) = \min\{d((p)_0,(u)_i) + w(((u)_i,(v)_{i+1})) : ((u)_i,(v)_{i+1}) \in A\}.$$

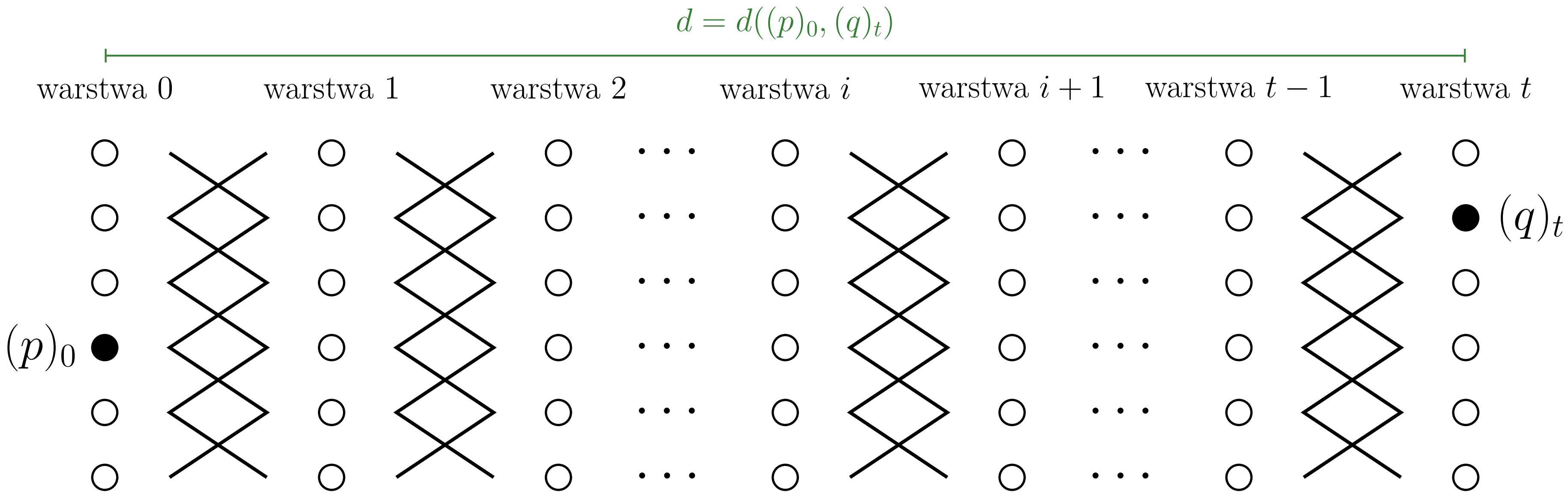


Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0,(q)_t)$-ścieżki π w grafie warstwowym.

↳ Dla $i = 1, 2, \ldots, n$ wyznaczamy długości $d((p)_0,(v)_i)$ najkrótszych ścieżek z $(p)_0$ do $(v)_i$. A dokładnie, aby wyznaczyć odległości $d((p)_0,(v)_{i+1})$, korzystamy z (poprawnie wyznaczonych) odległości $d((p)_0,(u)_i)$:

$$d((p)_0,(v)_{i+1}) = \min\{d((p)_0,(u)_i) + w(((u)_i,(v)_{i+1})) : ((u)_i,(v)_{i+1}) \in A\}.$$



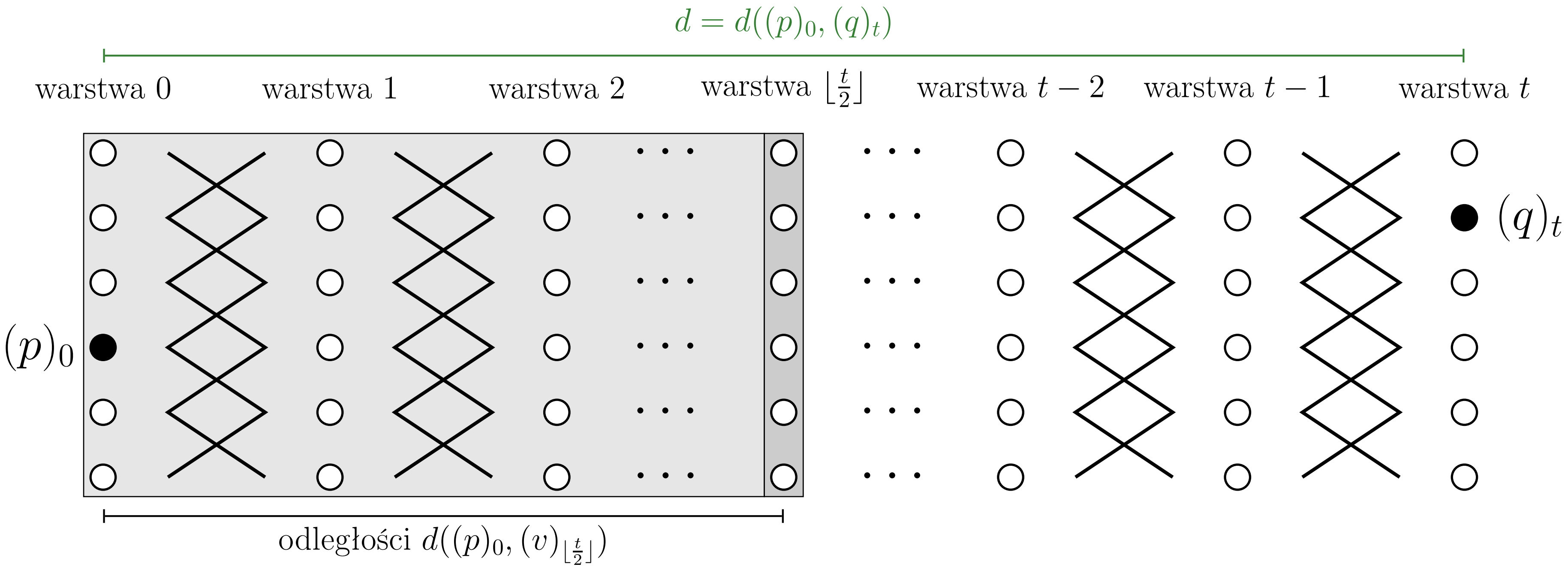
Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy" $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

(3) Wywołaj rekurencyjnie algorytm, aby wyznaczyć najkrótszą ścieżkę π_1 pomiędzy (p_0) i $(v)_{\lfloor \frac{t}{2} \rfloor}$ oraz najkrótszą ścieżkę π_2 pomiędzy $(v)_{\lfloor \frac{t}{2} \rfloor}$ i $(q)_t$.

(4) Rozwiązaniem jest $\pi_1 \cup \pi_2$.

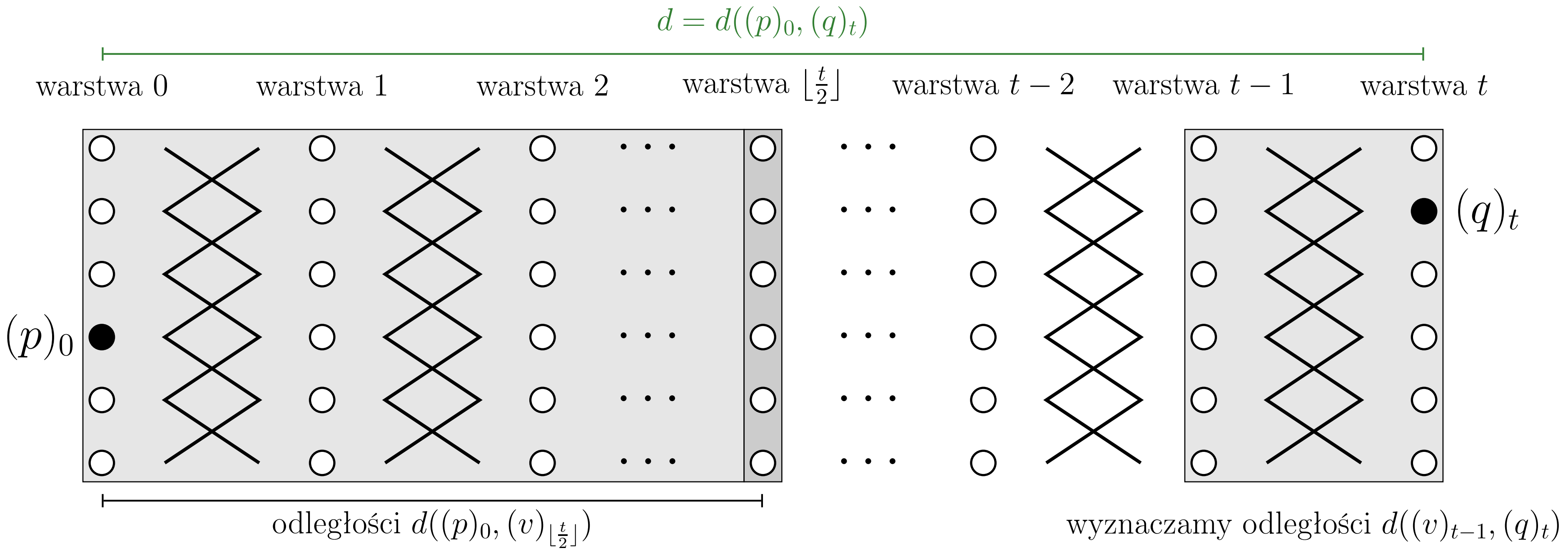


Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy” $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

↳ Tak jak w kroku (1), wyznaczamy odległości $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor})$. Analogicznie wyznaczamy odległości $d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t)$. Wierzchołek $(x)_{\lfloor \frac{t}{2} \rfloor}$ jest to taki $(v)_{\lfloor \frac{t}{2} \rfloor} \in V_{\lfloor \frac{t}{2} \rfloor}$, dla którego zachodzi $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor}) + d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t) = d$.

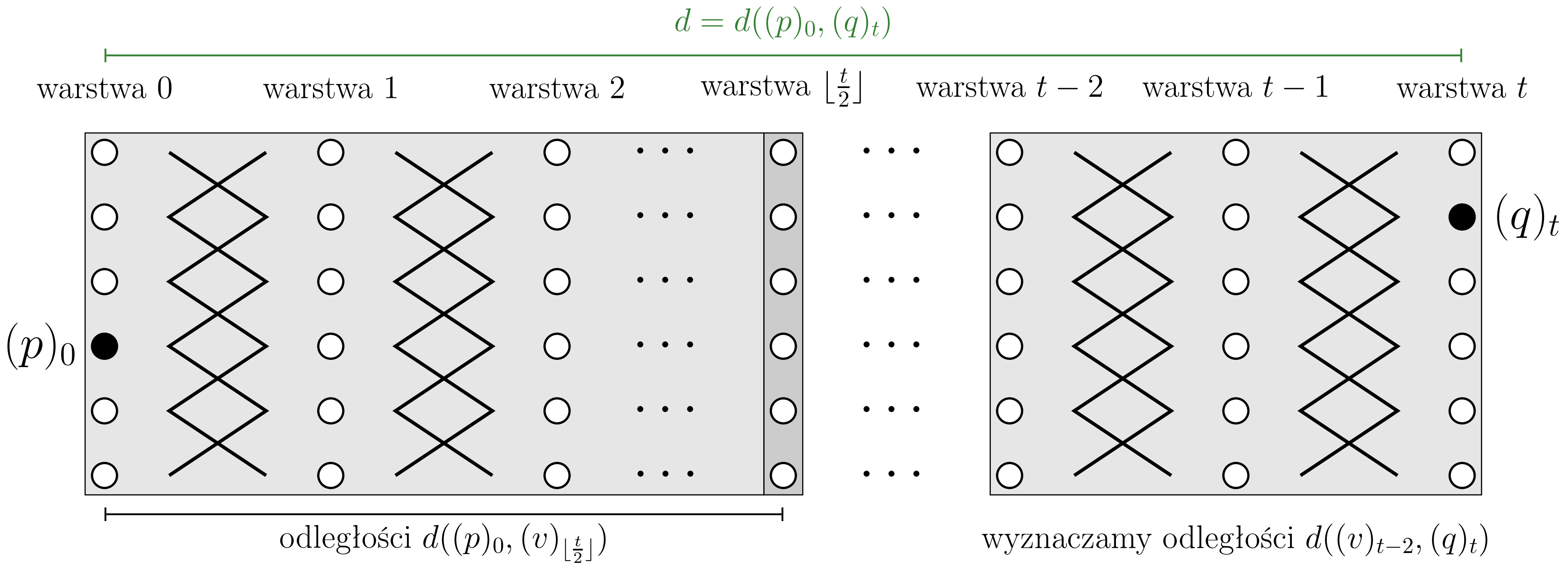


Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy” $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

↳ Tak jak w kroku (1), wyznaczamy odległości $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor})$. Analogicznie wyznaczamy odległości $d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t)$. Wierzchołek $(x)_{\lfloor \frac{t}{2} \rfloor}$ jest to taki $(v)_{\lfloor \frac{t}{2} \rfloor} \in V_{\lfloor \frac{t}{2} \rfloor}$, dla którego zachodzi $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor}) + d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t) = d$.

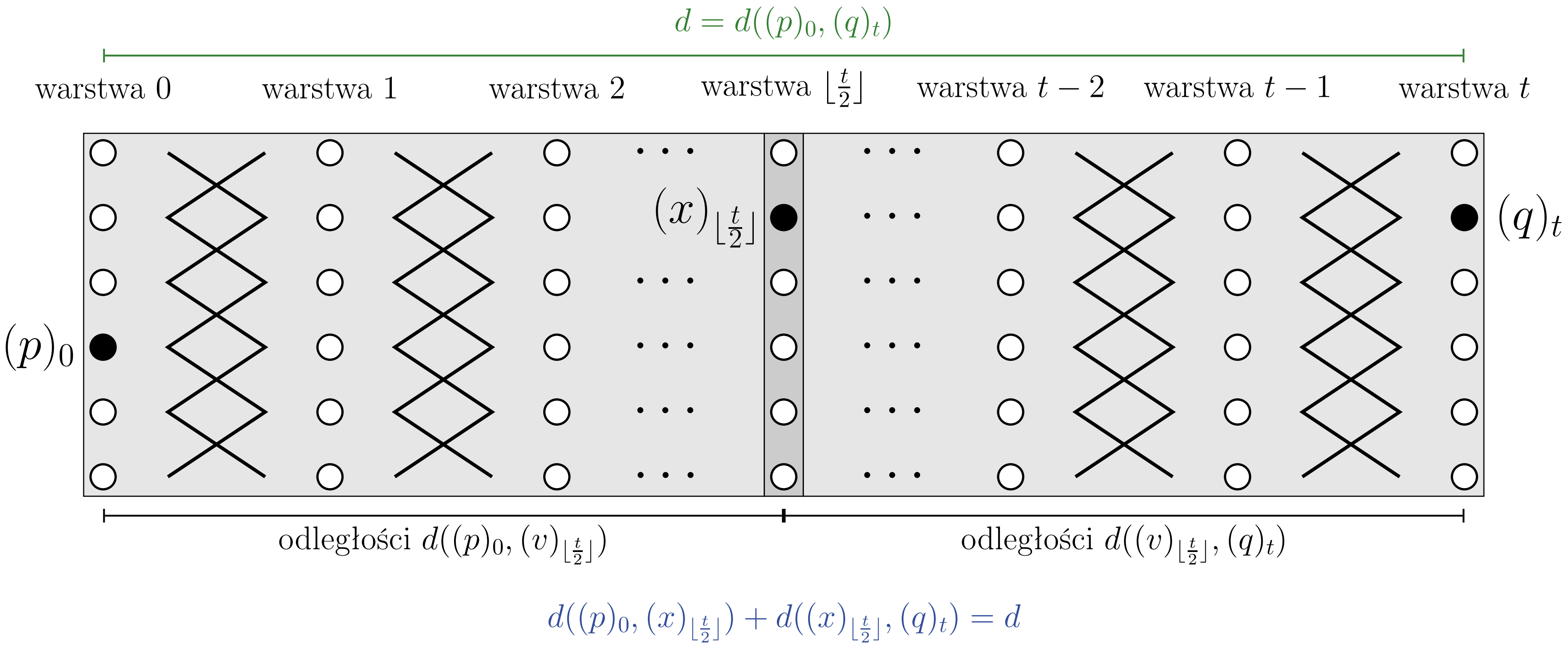


Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy" $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

↳ Tak jak w kroku (1), wyznaczamy odległości $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor})$. Analogicznie wyznaczamy odległości $d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t)$. Wierzchołek $(x)_{\lfloor \frac{t}{2} \rfloor}$ jest to taki $(v)_{\lfloor \frac{t}{2} \rfloor} \in V_{\lfloor \frac{t}{2} \rfloor}$, dla którego zachodzi $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor}) + d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t) = d$.

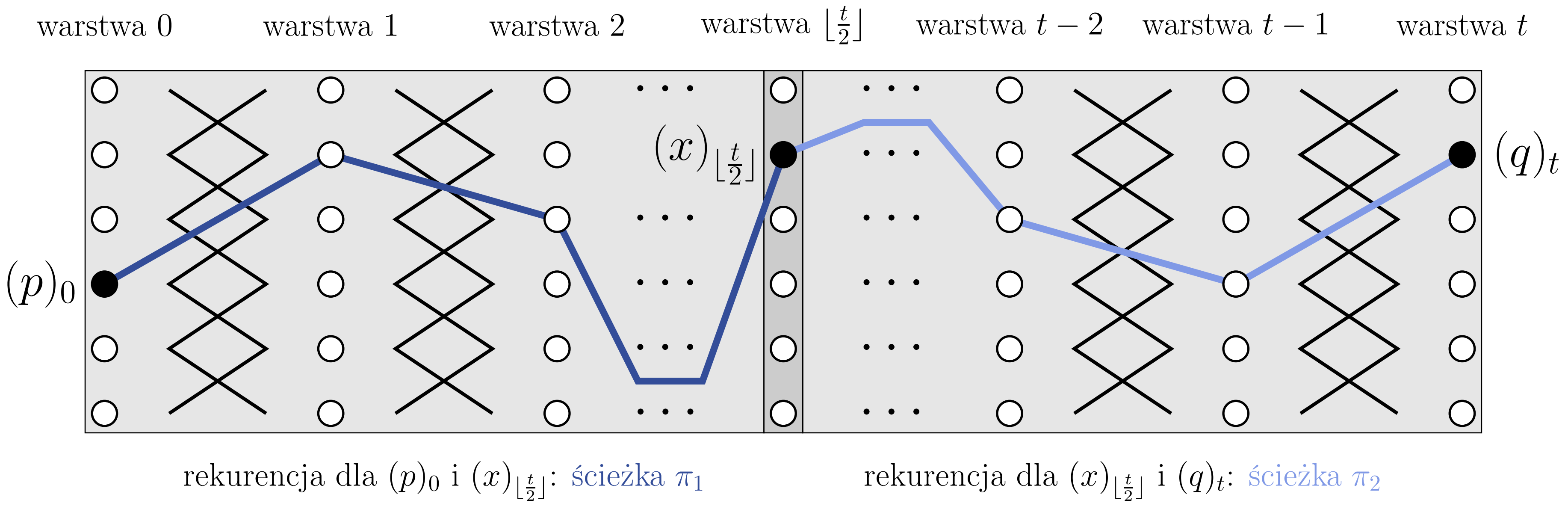


Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy” $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

↳ Tak jak w kroku (1), wyznaczamy odległości $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor})$. Analogicznie wyznaczamy odległości $d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t)$. Wierzchołek $(x)_{\lfloor \frac{t}{2} \rfloor}$ jest to taki $(v)_{\lfloor \frac{t}{2} \rfloor} \in V_{\lfloor \frac{t}{2} \rfloor}$, dla którego zachodzi $d((p)_0, (v)_{\lfloor \frac{t}{2} \rfloor}) + d((v)_{\lfloor \frac{t}{2} \rfloor}, (q)_t) = d$.



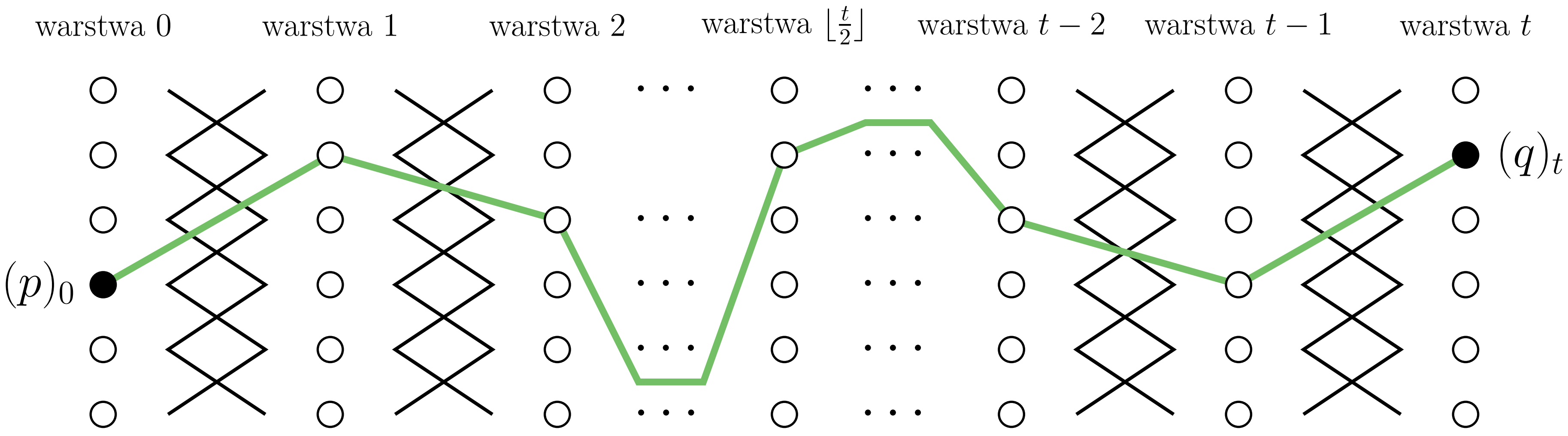
Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy” $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

(3) Wywołaj rekurencyjnie algorytm, aby wyznaczyć najkrótszą ścieżkę π_1 pomiędzy (p_0) i $(v)_{\lfloor \frac{t}{2} \rfloor}$ oraz najkrótszą ścieżkę π_2 pomiędzy $(v)_{\lfloor \frac{t}{2} \rfloor}$ i $(q)_t$.

(4) Rozwiązaniem jest $\pi_1 \cup \pi_2$.



Idea algorytmu

(1) Wyznacz długość d najkrótszej $((p)_0, (q)_t)$-ścieżki π w grafie warstwowym.

(2) Wyznacz wierzchołek „środkowy" $(x)_{\lfloor \frac{t}{2} \rfloor}$ na ścieżce π.

(3) Wywołaj rekurencyjnie algorytm, aby wyznaczyć najkrótszą ścieżkę π_1 pomiędzy (p_0) i $(v)_{\lfloor \frac{t}{2} \rfloor}$ oraz najkrótszą ścieżkę π_2 pomiędzy $(v)_{\lfloor \frac{t}{2} \rfloor}$ i $(q)_t$.

(4) Rozwiązaniem jest $\pi_1 \cup \pi_2$.

Analiza złożoności czasowej algorytmu

- Krok 1. Wyznaczenie najkrótszej odległości $d((p)_0, (q)_t)$: $O(n^2 \cdot t)$.
- Krok 2. Wyznaczenie wierzchołka $(x)_{\lfloor \frac{t}{2} \rfloor}$: $O(n^2 \cdot t)$.
- Krok 3. Rekurencyjne wyznaczenie ścieżek π_1 i π_2: $T(n, \lfloor \frac{t}{2} \rfloor) + T(n, \lfloor \frac{t+1}{2} \rfloor)$.

$$T(n,t) = \begin{cases} \Theta(n) & \text{dla } t = 0, 1; \\ T(n, \lfloor \frac{t}{2} \rfloor) + T(n, \lfloor \frac{t+1}{2} \rfloor) + O(n^2 \cdot t) & \text{w przeciwnym wypadku.} \end{cases}$$

Rozwiązaniem powyższej zależności jest $T(n,t) = O(n^2 \cdot t \cdot \log t)$.

Analiza złożoności pamięciowej algorytmu

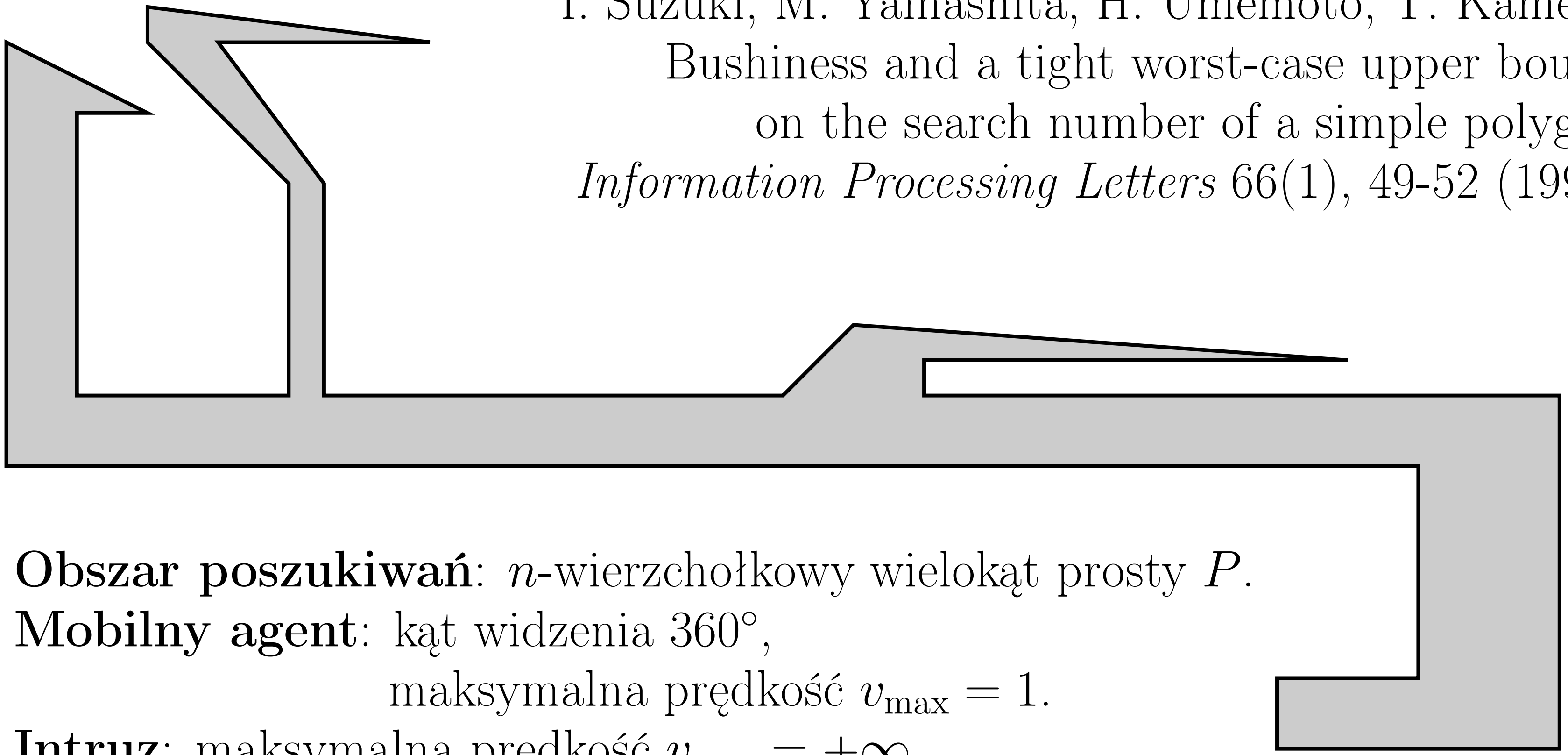
- Tablice najkrótszych odległości wierzchołków: $\Theta(n)$.
- Kolejne wierzchołki najkrótszej ścieżki z $(p)_0$ do $(q)_t$: $\Theta(t)$.

Twierdzenie 4.5. (Munro, Ramirez 1982)
Problem najkrótszej ścieżki w grafie warstowym $D = (V, A, w)$ z parametrami n i t można rozwiązać w czasie $O(n^2 \cdot t \cdot \log t)$ i (dodatkowej) pamięci rzędu $\Theta(n + t)$.

4.6 Przeszukiwanie wielokątów

L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, R. Motwani
Visibility-based pursuit-evasion in a polygonal environment
Int. J. of Computational Geometry and Applications 9(5), 471-494 (1999)

I. Suzuki, M. Yamashita, H. Umemoto, T. Kameda
Bushiness and a tight worst-case upper bound
on the search number of a simple polygon
Information Processing Letters 66(1), 49-52 (1998)

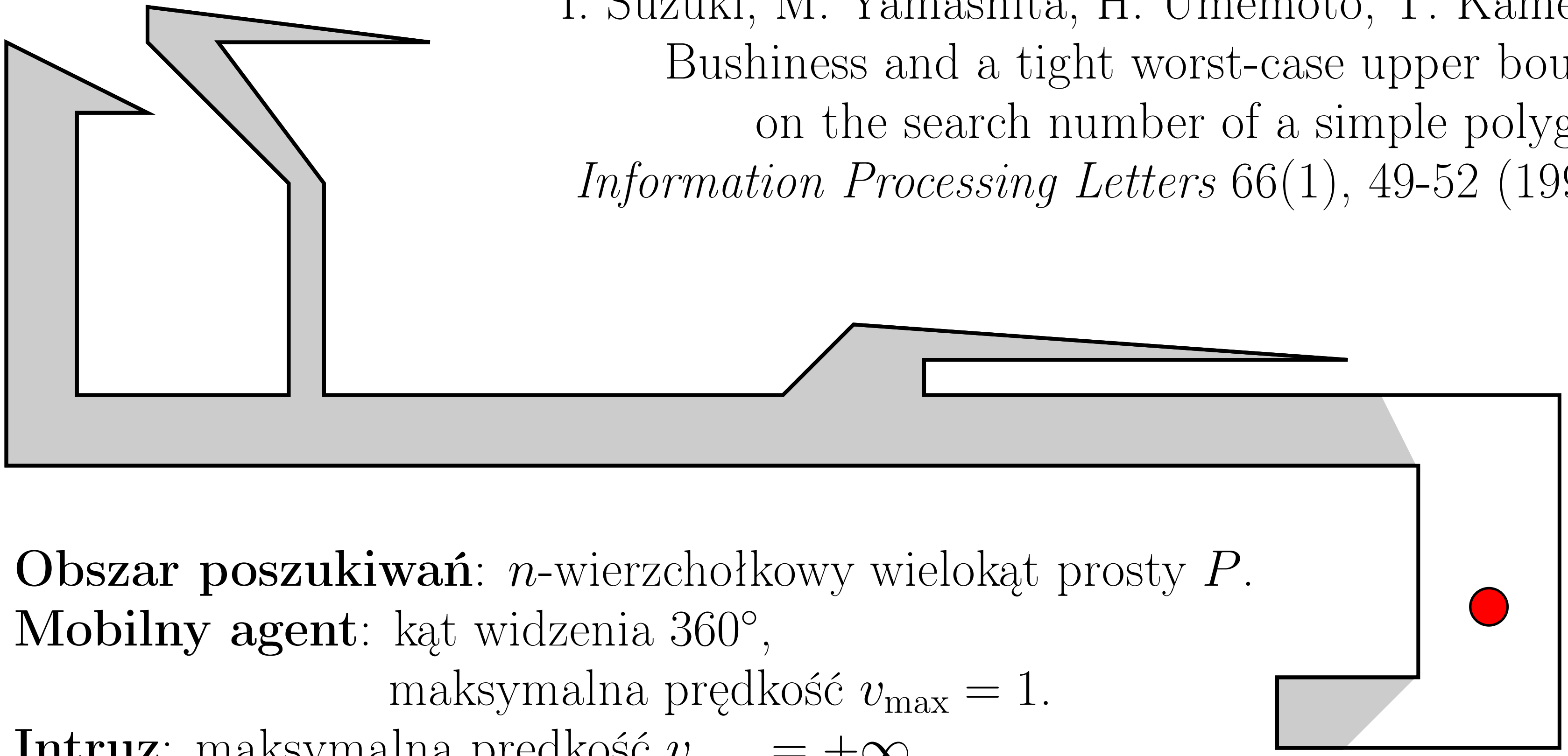


- **Obszar poszukiwań**: n-wierzchołkowy wielokąt prosty P.
- **Mobilny agent**: kąt widzenia 360°,
  maksymalna prędkość $v_{\max} = 1$.
- **Intruz**: maksymalna prędkość $v_{\max} = +\infty$,
  zna strategię mobilych agentów.

4.6 Przeszukiwanie wielokątów

L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, R. Motwani
Visibility-based pursuit-evasion in a polygonal environment
Int. J. of Computational Geometry and Applications 9(5), 471-494 (1999)

I. Suzuki, M. Yamashita, H. Umemoto, T. Kameda
Bushiness and a tight worst-case upper bound
on the search number of a simple polygon
Information Processing Letters 66(1), 49-52 (1998)



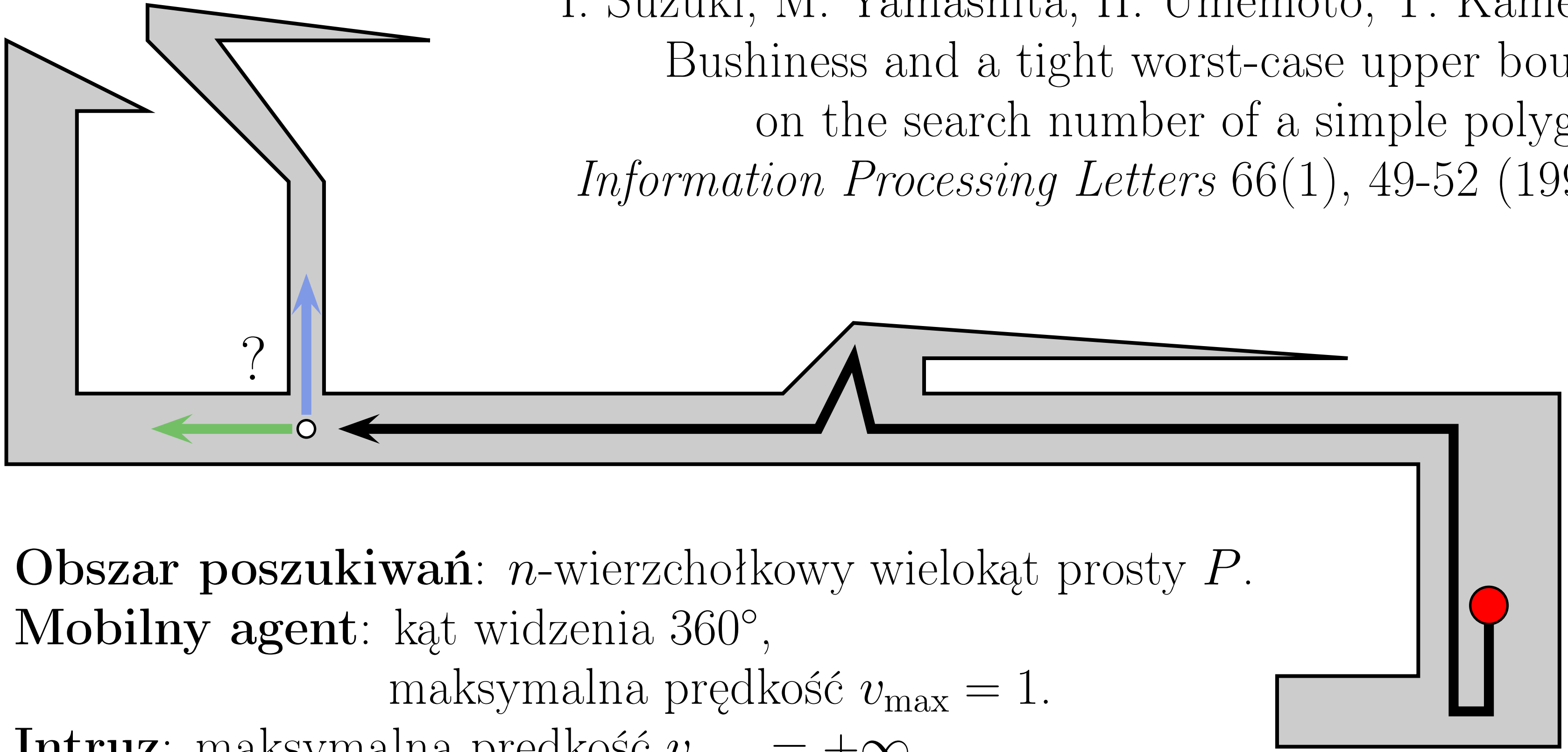
pole widoczności agenta

- **Obszar poszukiwań**: n-wierzchołkowy wielokąt prosty P.
- **Mobilny agent**: kąt widzenia 360°,
  maksymalna prędkość $v_{\max} = 1$.
- **Intruz**: maksymalna prędkość $v_{\max} = +\infty$,
  zna strategię mobilych agentów.

4.6 Przeszukiwanie wielokątów

L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, R. Motwani
Visibility-based pursuit-evasion in a polygonal environment
Int. J. of Computational Geometry and Applications 9(5), 471-494 (1999)

I. Suzuki, M. Yamashita, H. Umemoto, T. Kameda
Bushiness and a tight worst-case upper bound
on the search number of a simple polygon
Information Processing Letters 66(1), 49-52 (1998)

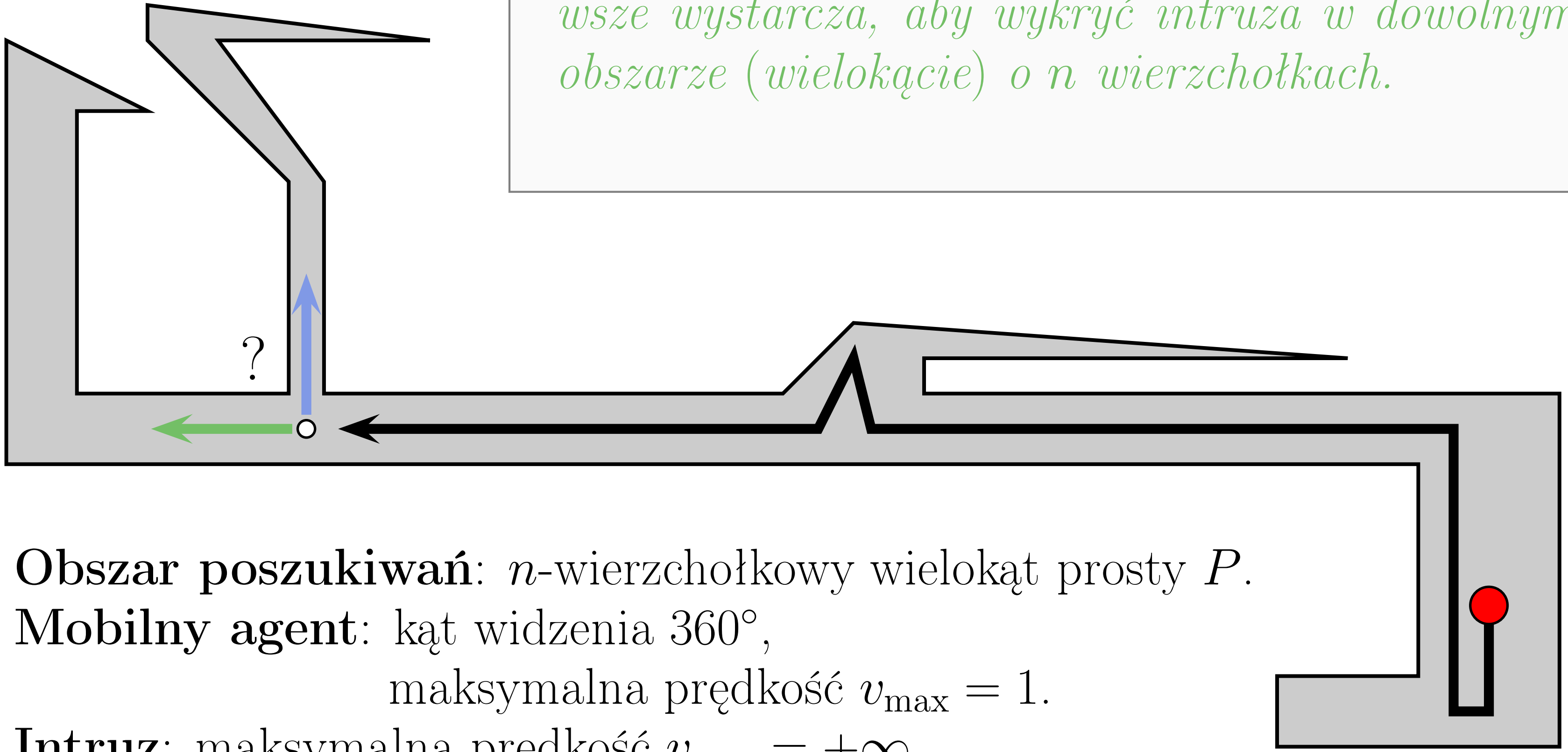


- **Obszar poszukiwań**: n-wierzchołkowy wielokąt prosty P.
- **Mobilny agent**: kąt widzenia 360°,
  maksymalna prędkość $v_{\max} = 1$.
- **Intruz**: maksymalna prędkość $v_{\max} = +\infty$,
  zna strategię mobilych agentów.

4.6 PRZESZUKIWANIE WIELOKĄTÓW

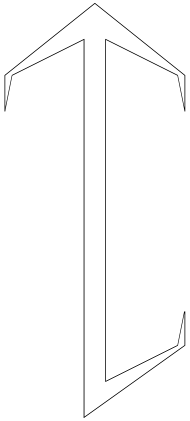
Problem. (Przeszukiwanie wielokąta)

Ilu mobilnych agentów czasami potrzeba, ale zawsze wystarcza, aby wykryć intruza w dowolnym obszarze (wielokącie) o n wierzchołkach.

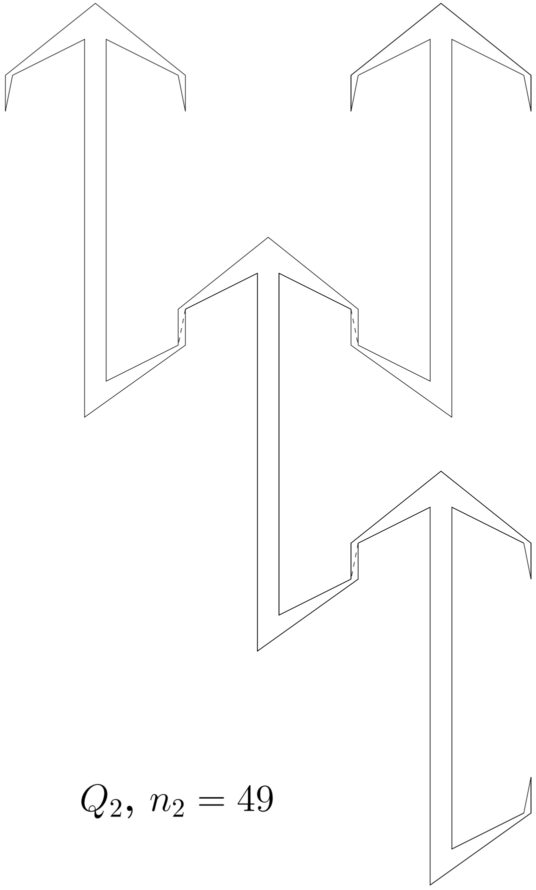


- ▶ **Obszar poszukiwań**: n-wierzchołkowy wielokąt prosty P.
- ▶ **Mobilny agent**: kąt widzenia 360°,
  maksymalna prędkość $v_{\max} = 1$.
- ▶ **Intruz**: maksymalna prędkość $v_{\max} = +\infty$,
  zna strategię mobilych agentów.

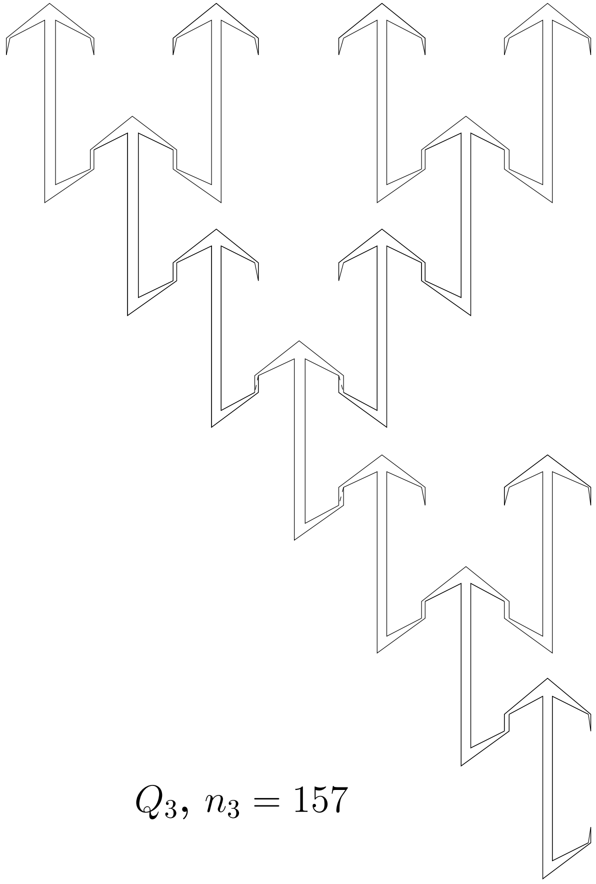
Oszacowanie dolne rzędu $\Omega(\log n)$ /Suzuki *et al.* 1998/



Q_1, $n_1 = 13$



Q_2, $n_2 = 49$

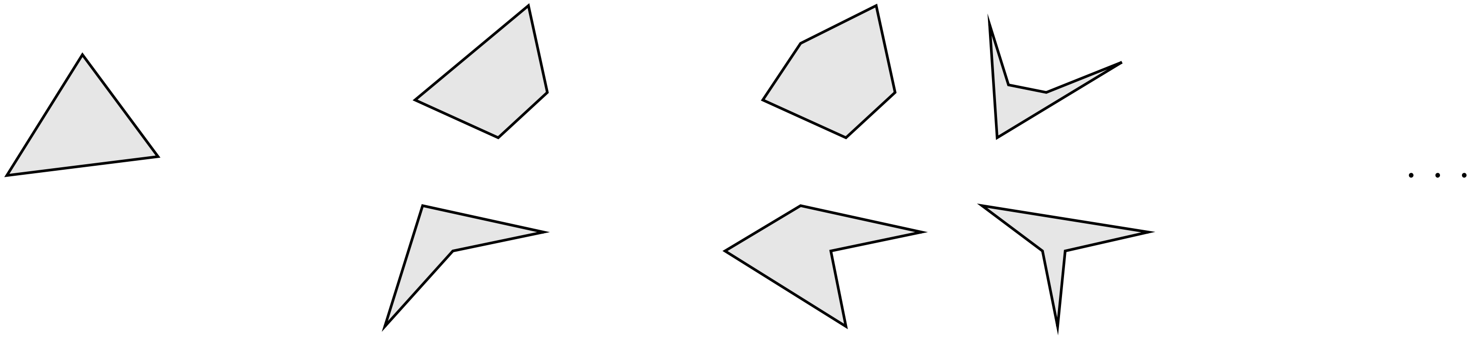


Q_3, $n_3 = 157$

- Wielokąt Q_i ma $n_i = 3^i + 10 \cdot \frac{3^i - 1}{2}$ wierzchołków.
- Wielokąt Q_i potrzebuje $i + 1 = \Omega(\log n)$ mobilnych agentów.

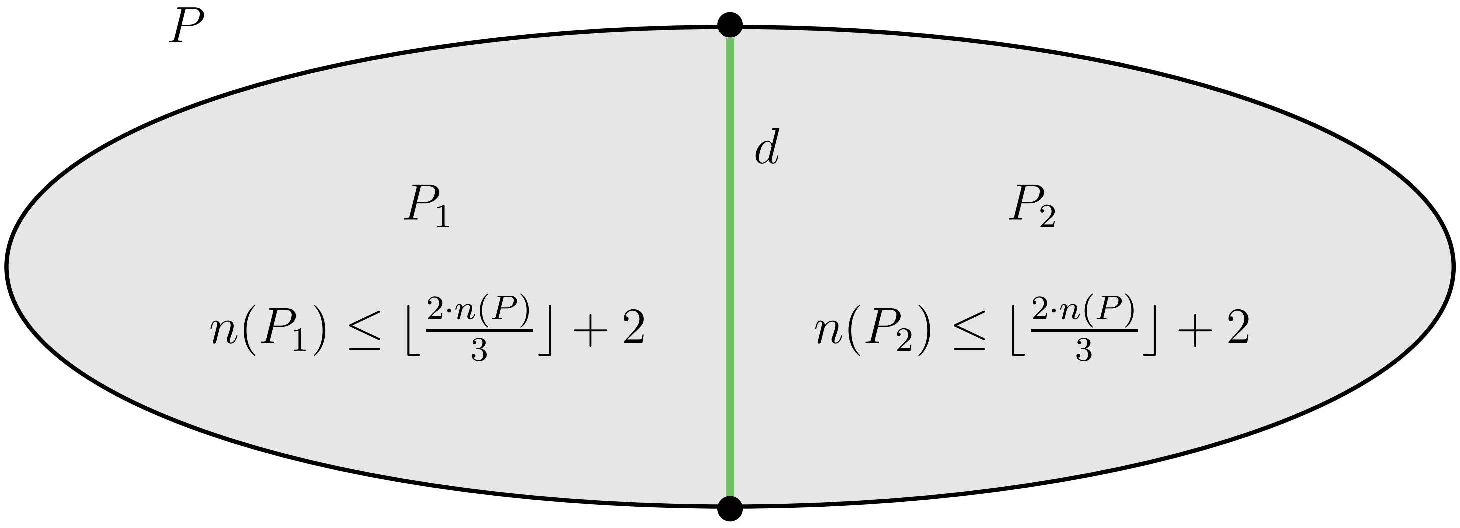
Oszacowanie górne rzędu $O(\log n)$ /Guibas *et al.* 1999/

- Wielokąt o $3, \ldots, 6$ wierzchołkach może być przeszukany przez jednego agenta.



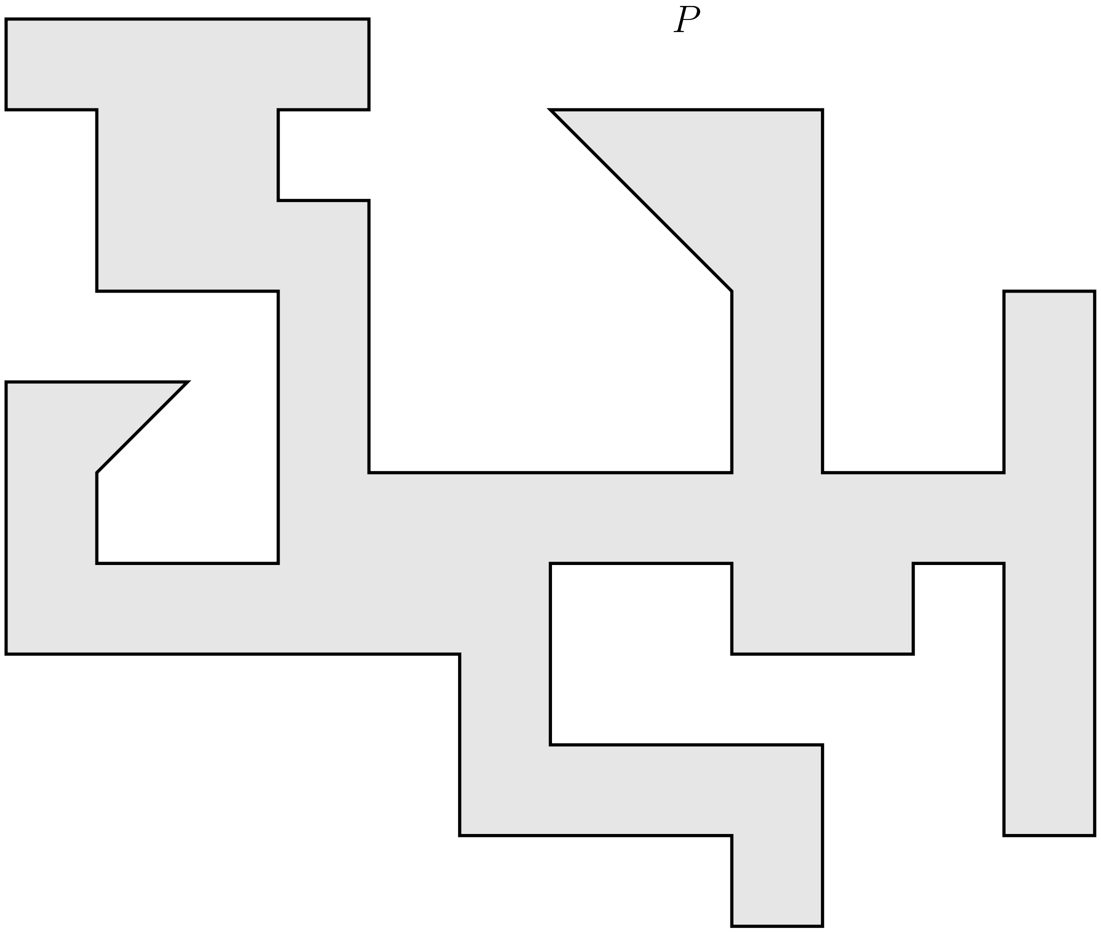
Oszacowanie górne rzędu $O(\log n)$ /Guibas *et al.* 1999/

- Wielokąt o $3, \ldots, 6$ wierzchołkach może być przeszukany przez jednego agenta.



- $n \geq 7 \ldots$

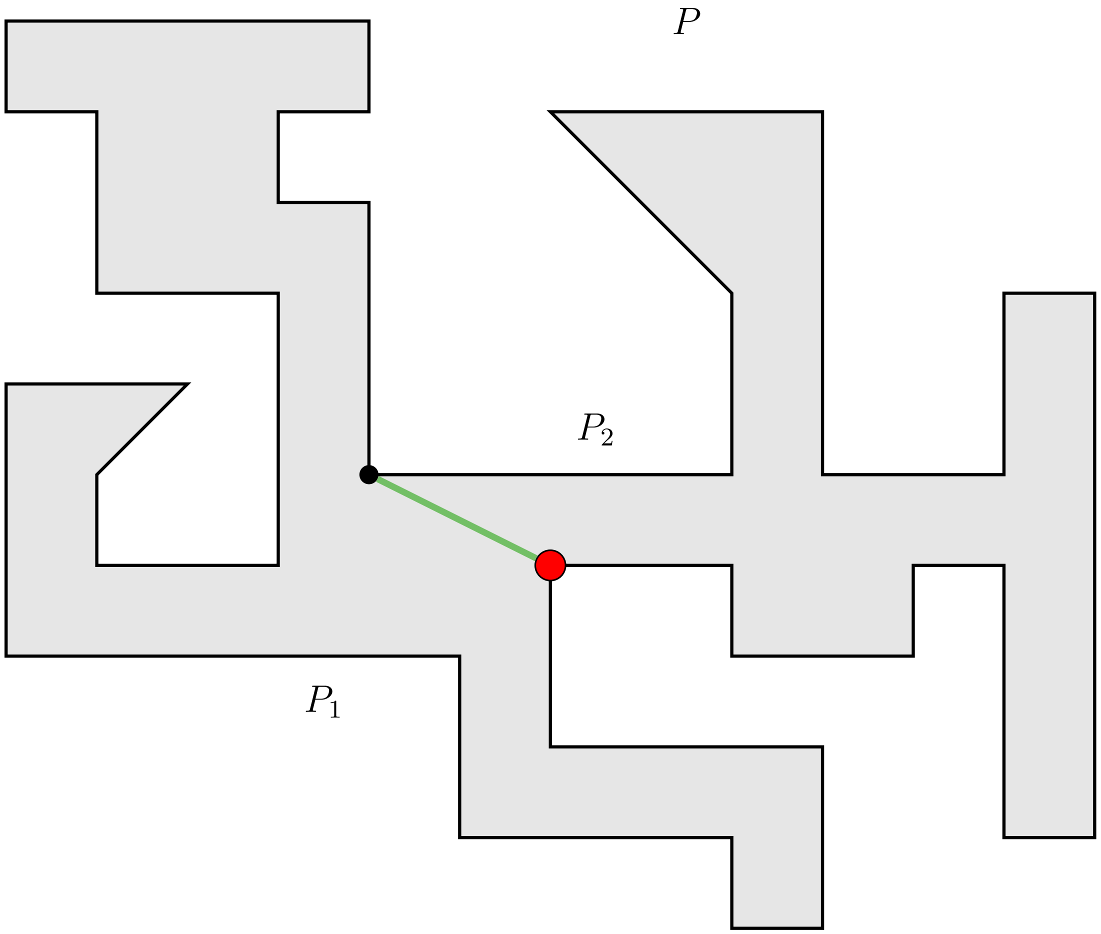
Twierdzenie 4.6. (Chazelle 1982)
W dowolnym n-wierzchołkowym wielokącie prostym P, $n \geq 4$, istnieje (wewnętrzna) przekątna d, która dzieli wielokąt na dwie cześci P_1 i P_2, z których każda ma co najwyżej $\lfloor \frac{2n}{3} \rfloor + 2$ wierzchołków.



wielokąt	liczba wierzchołków	liczba agentów
P	40	0

$n(P) = 40$

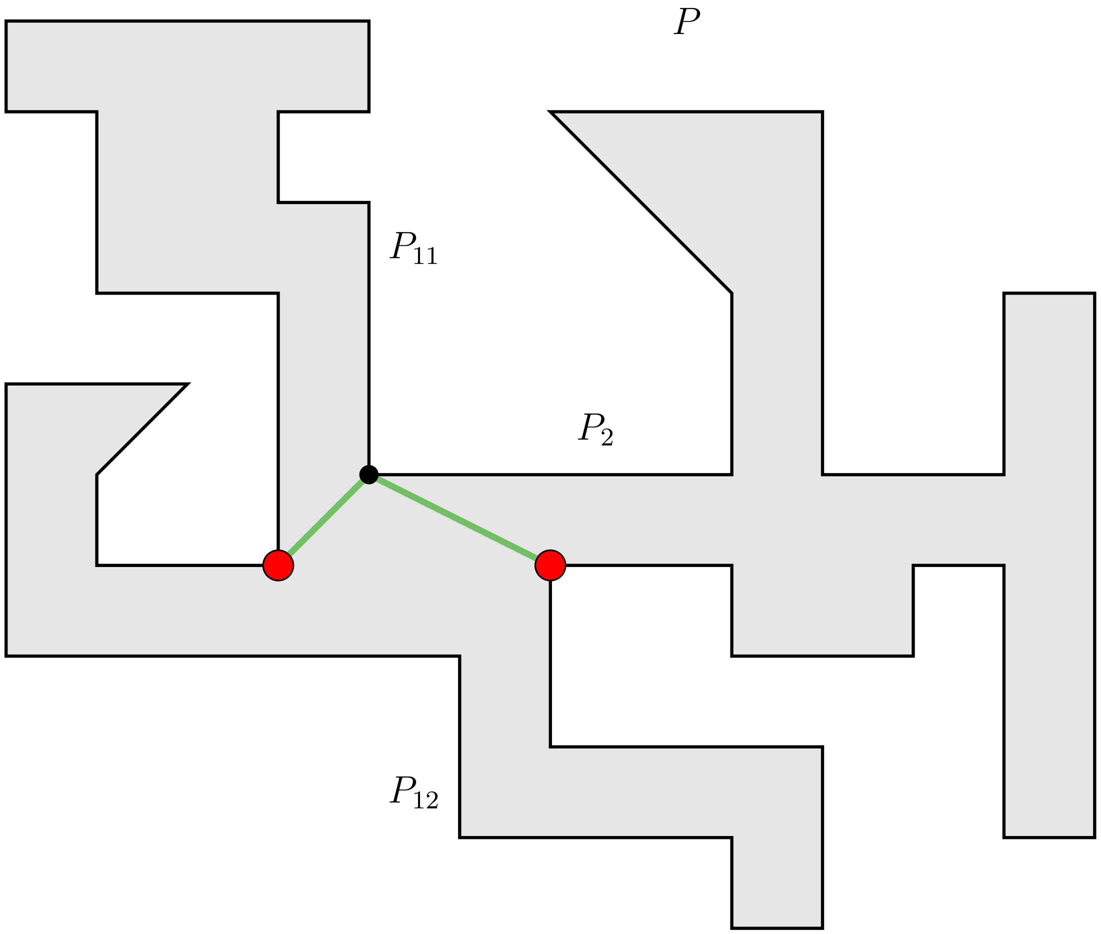
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1

$n(P) = 40$

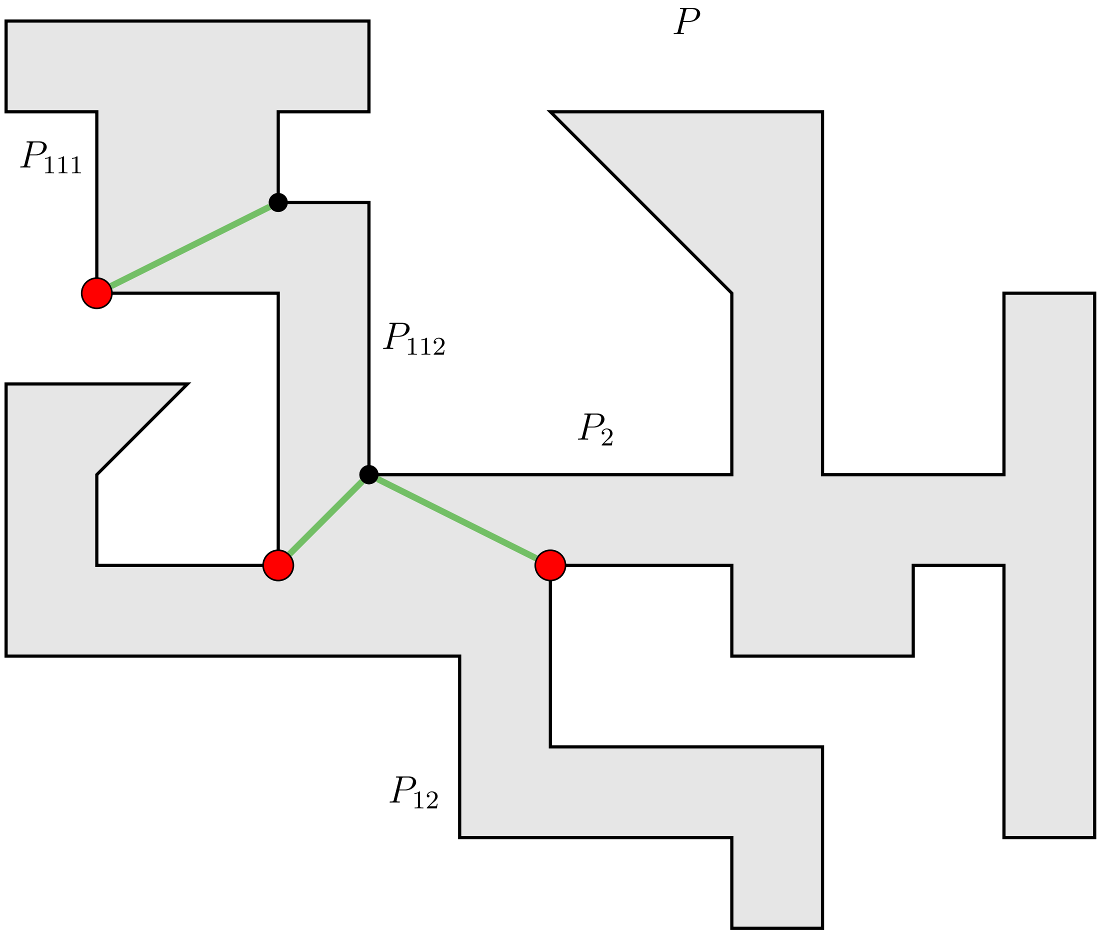
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2

$n(P) = 40$

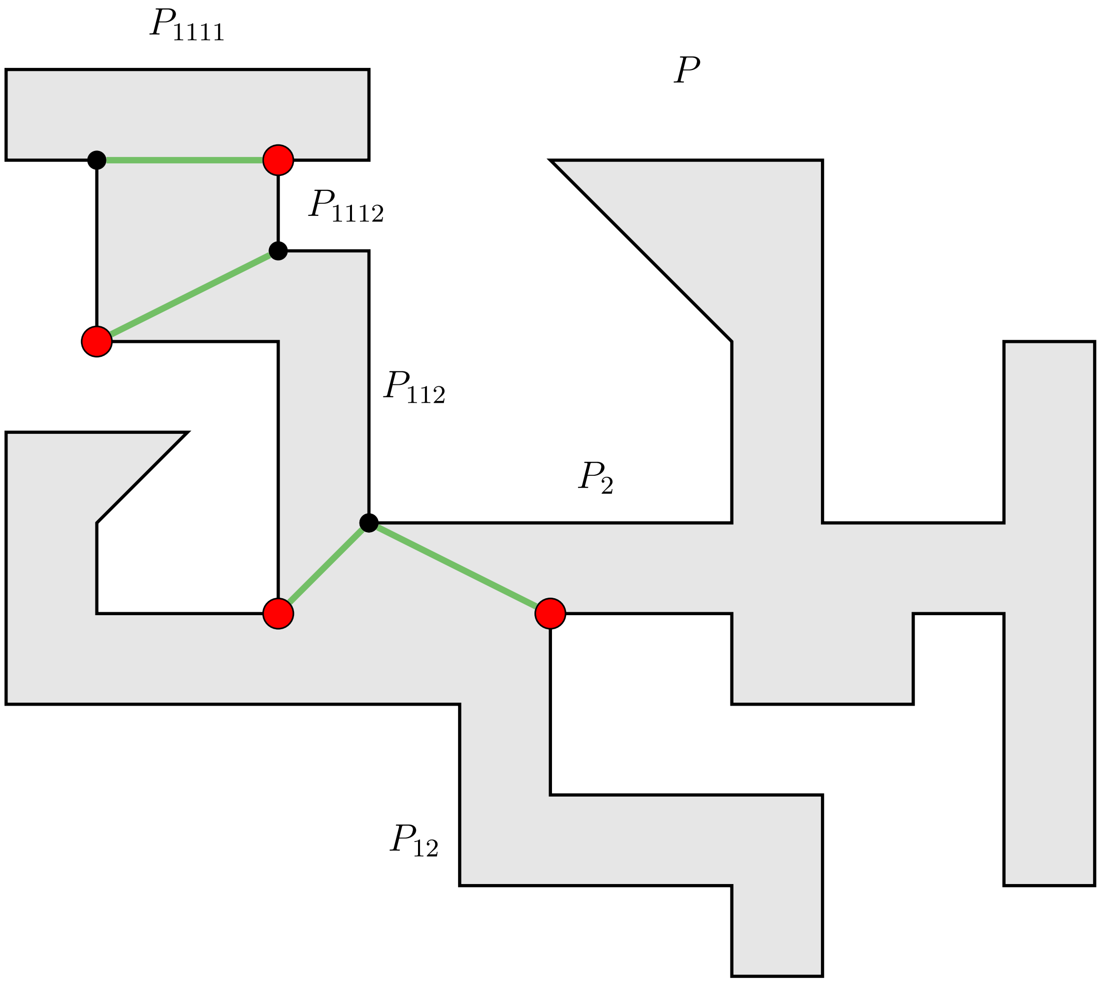
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3

$n(P) = 40$

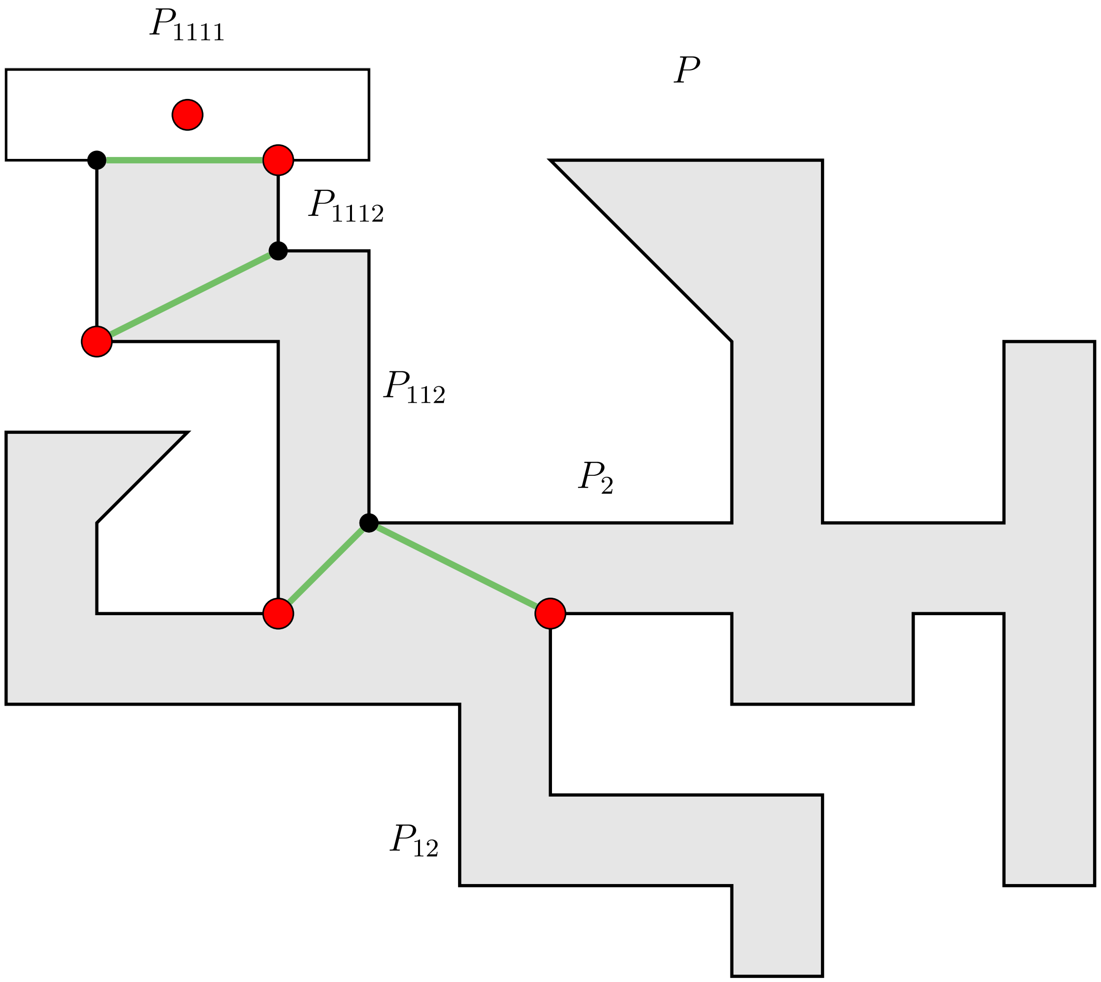
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3
P_{1111}	6	4
P_{1112}	4	4

$$n(P) = 40$$

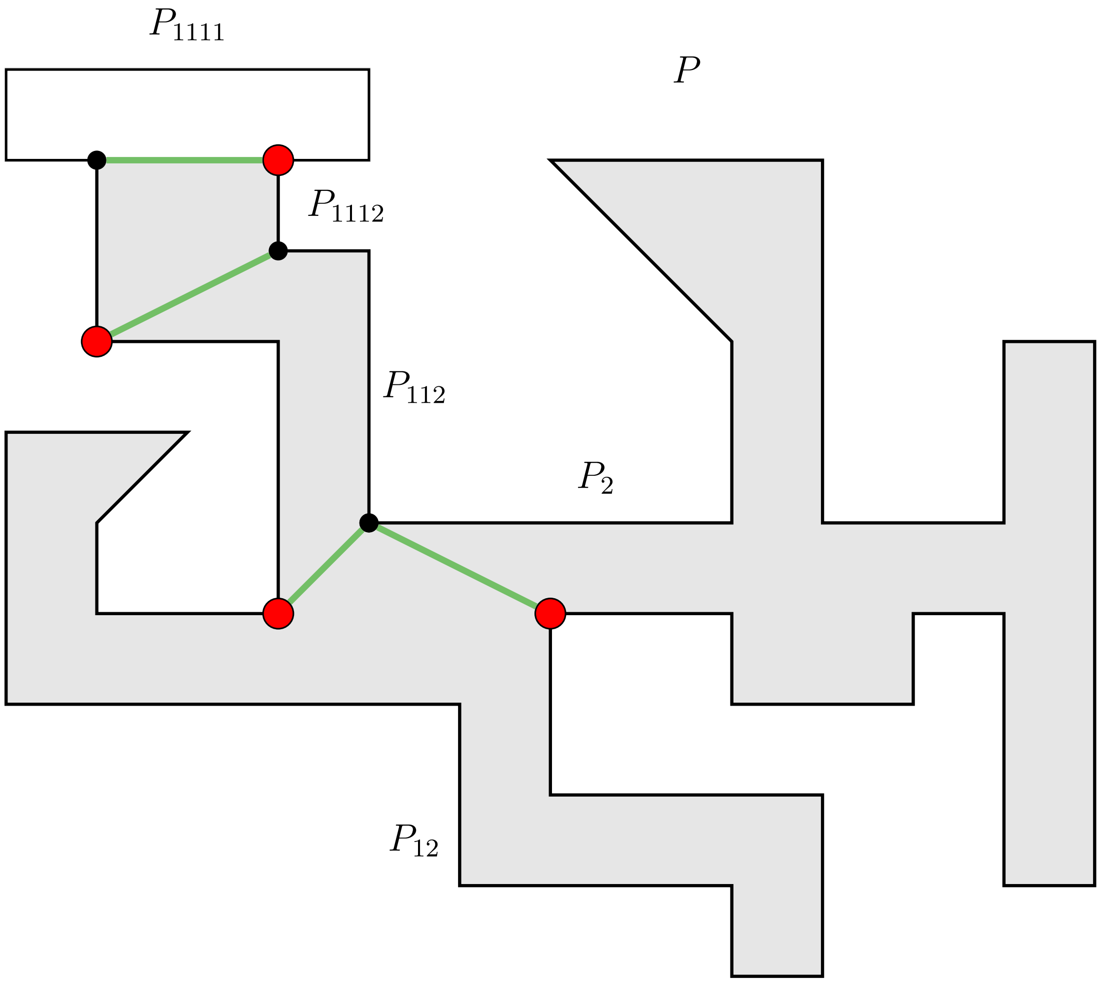
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3
P_{1111}	6	4
P_{1112}	4	4
P_{1111}	6	5

$n(P) = 40$

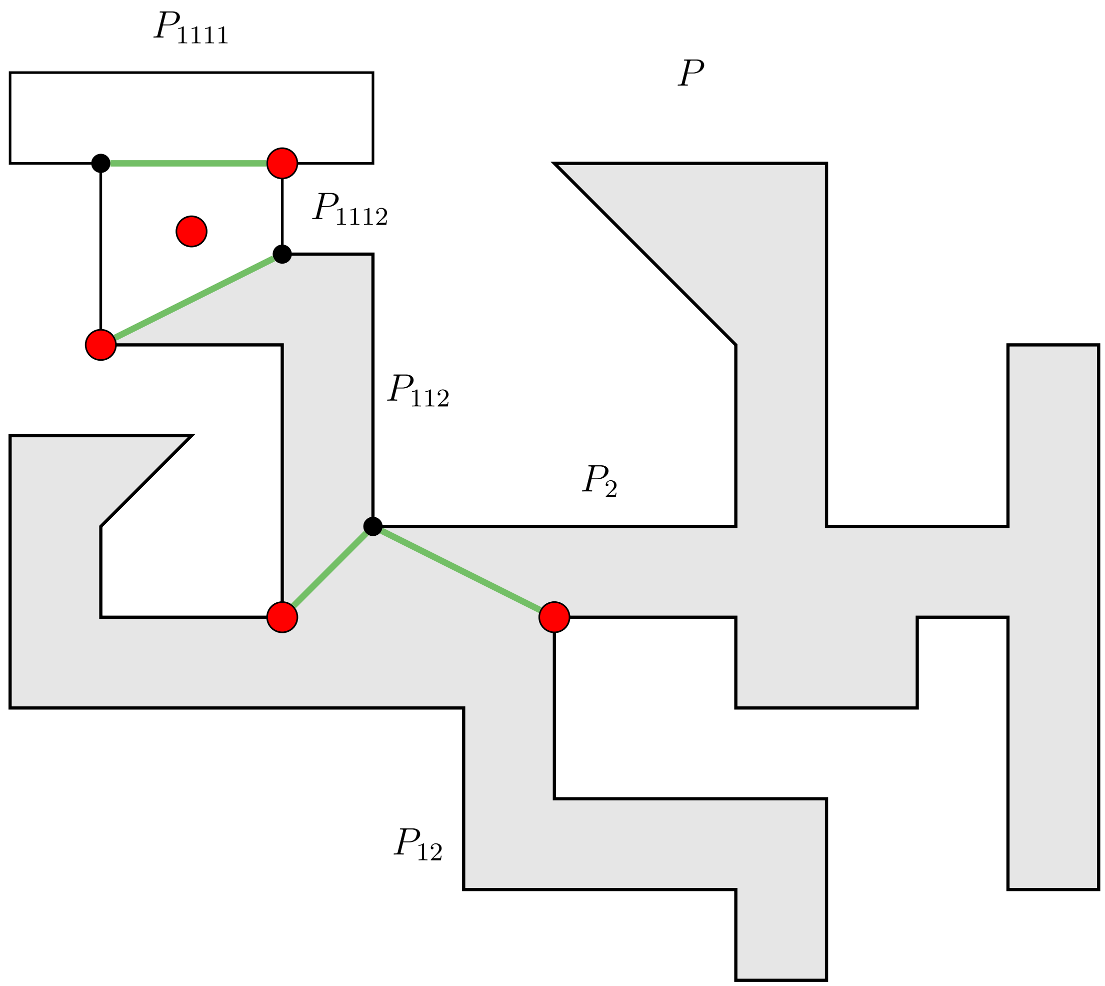
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3
P_{1111}	6	4
P_{1112}	4	4

$$n(P) = 40$$

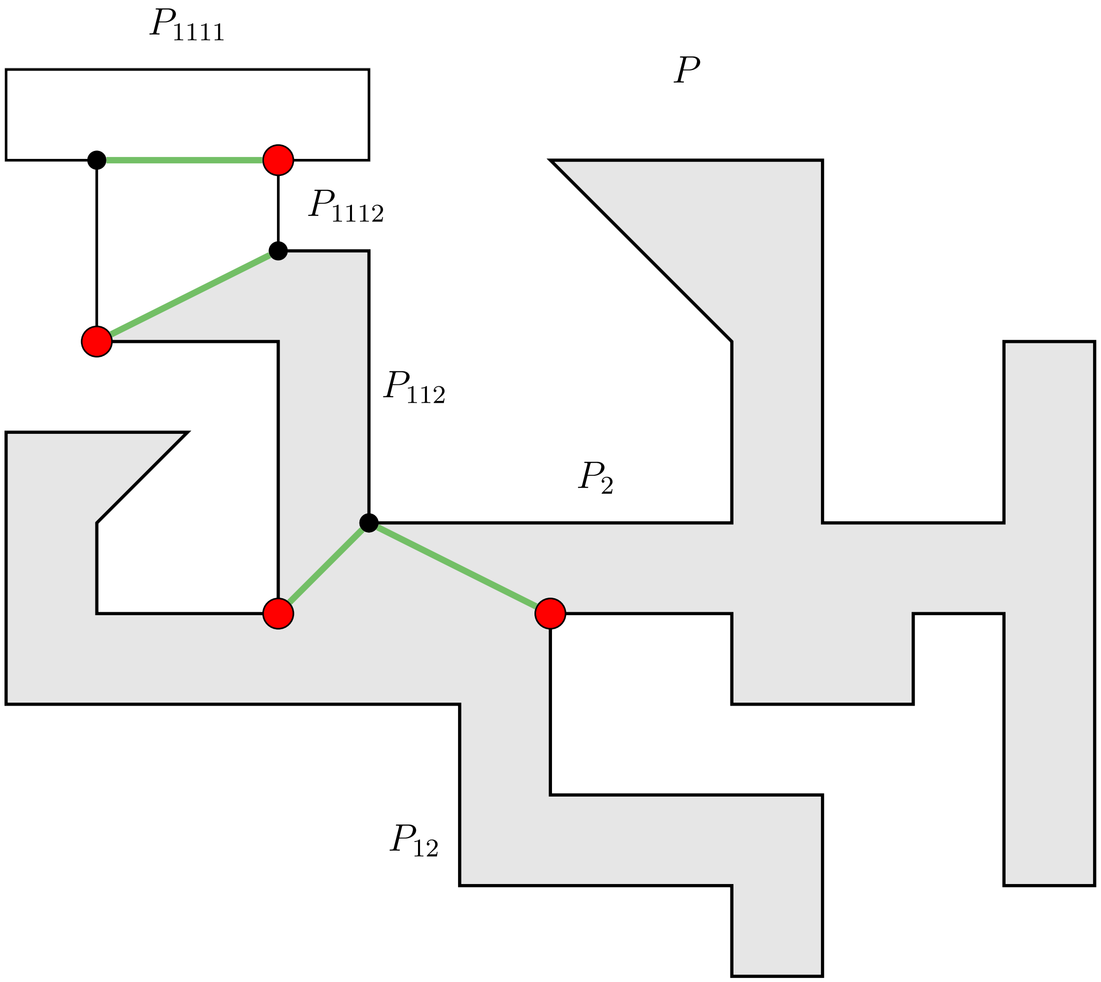
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3
P_{1111}	6	4
P_{1112}	4	4
P_{1112}	4	5

$$n(P) = 40$$

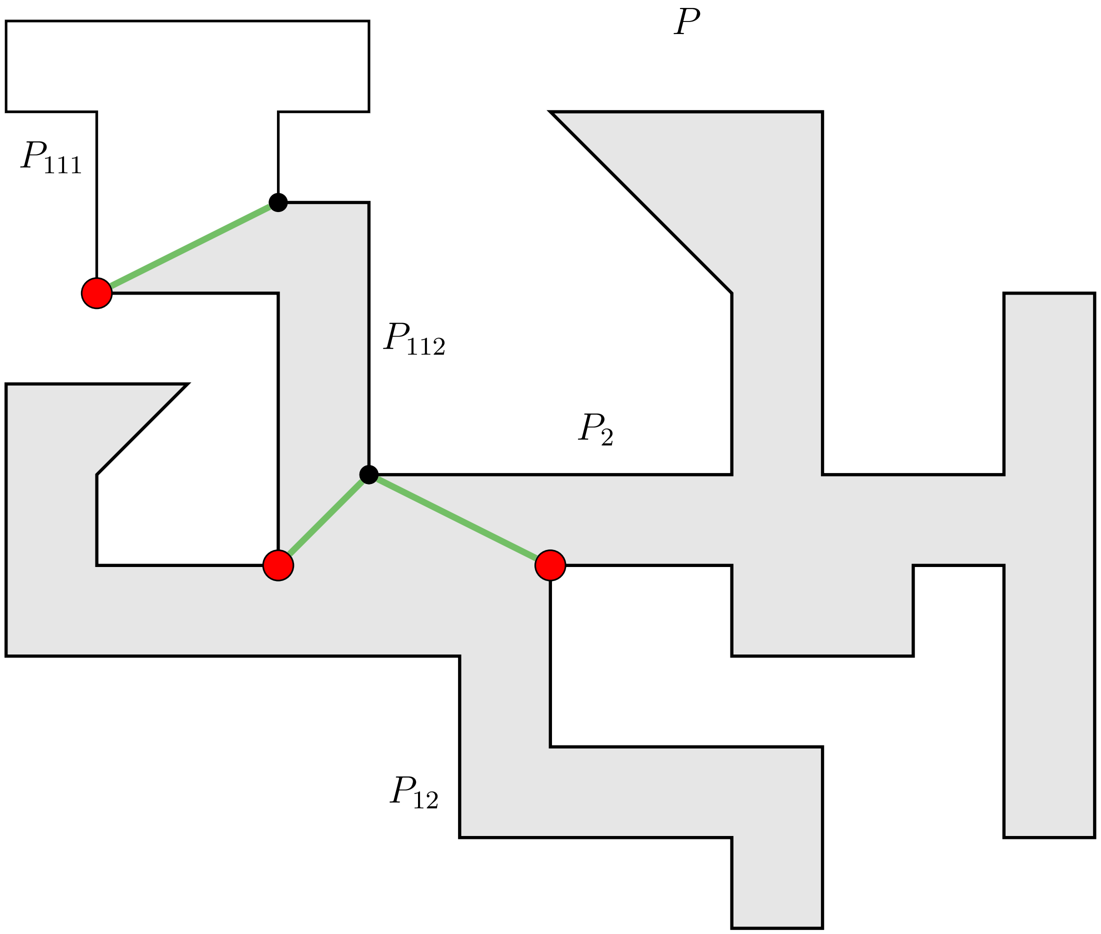
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3
P_{1111}	6	4
P_{1112}	4	4

$n(P) = 40$

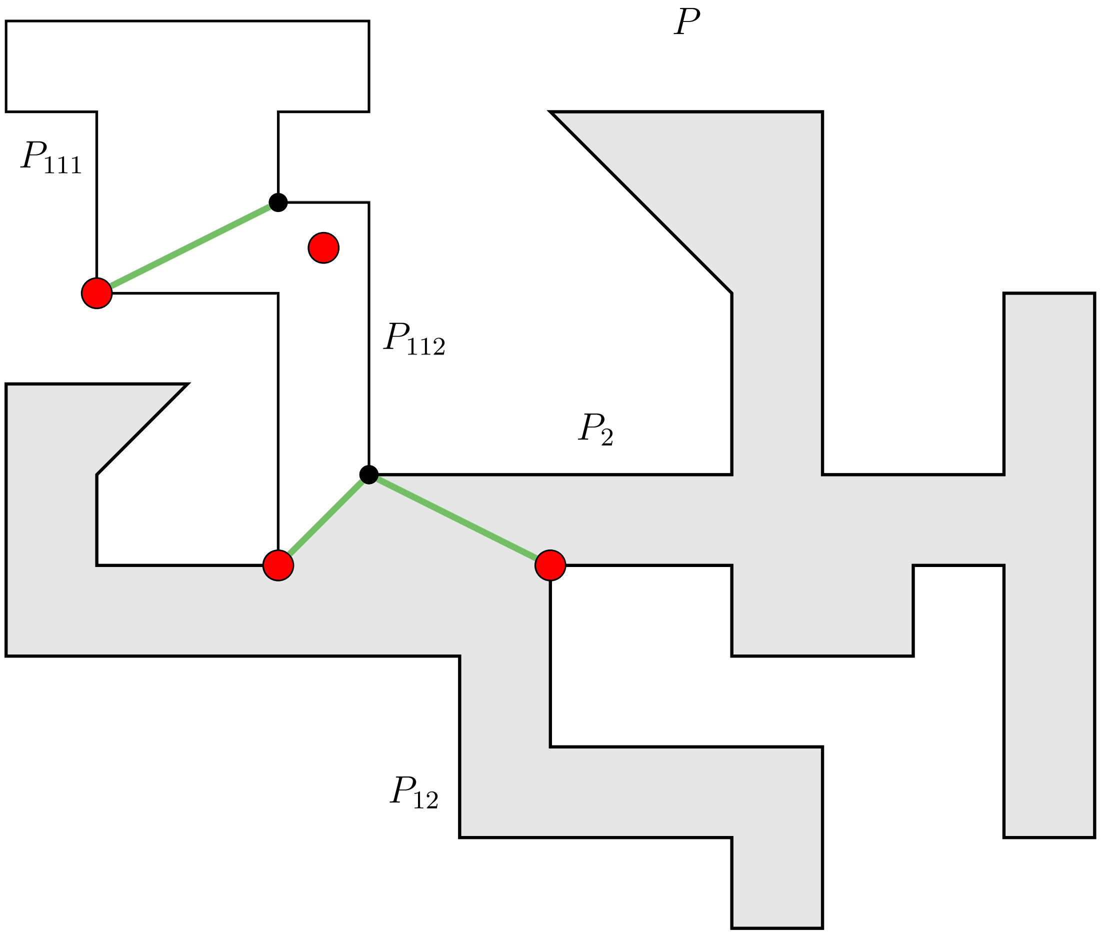
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3

$n(P) = 40$

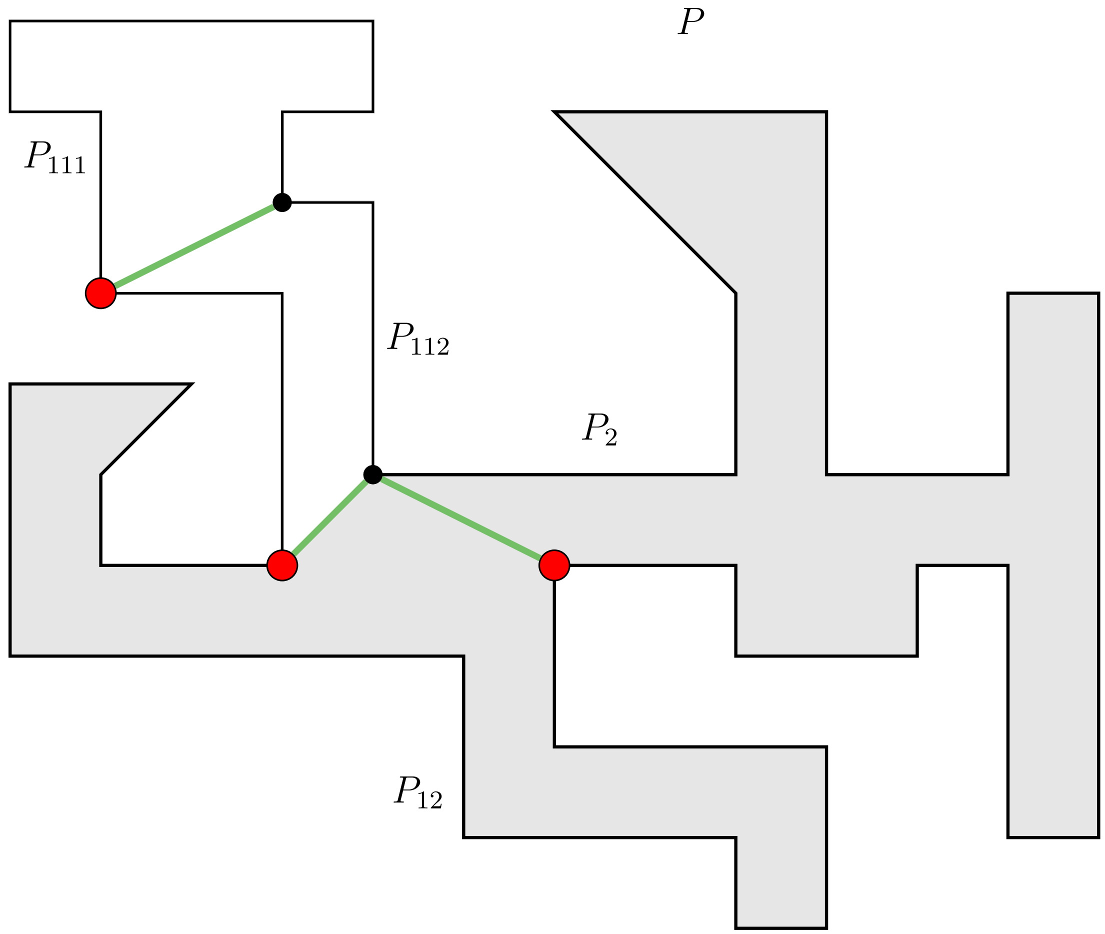
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3
P_{112}	4	4

$$n(P) = 40$$

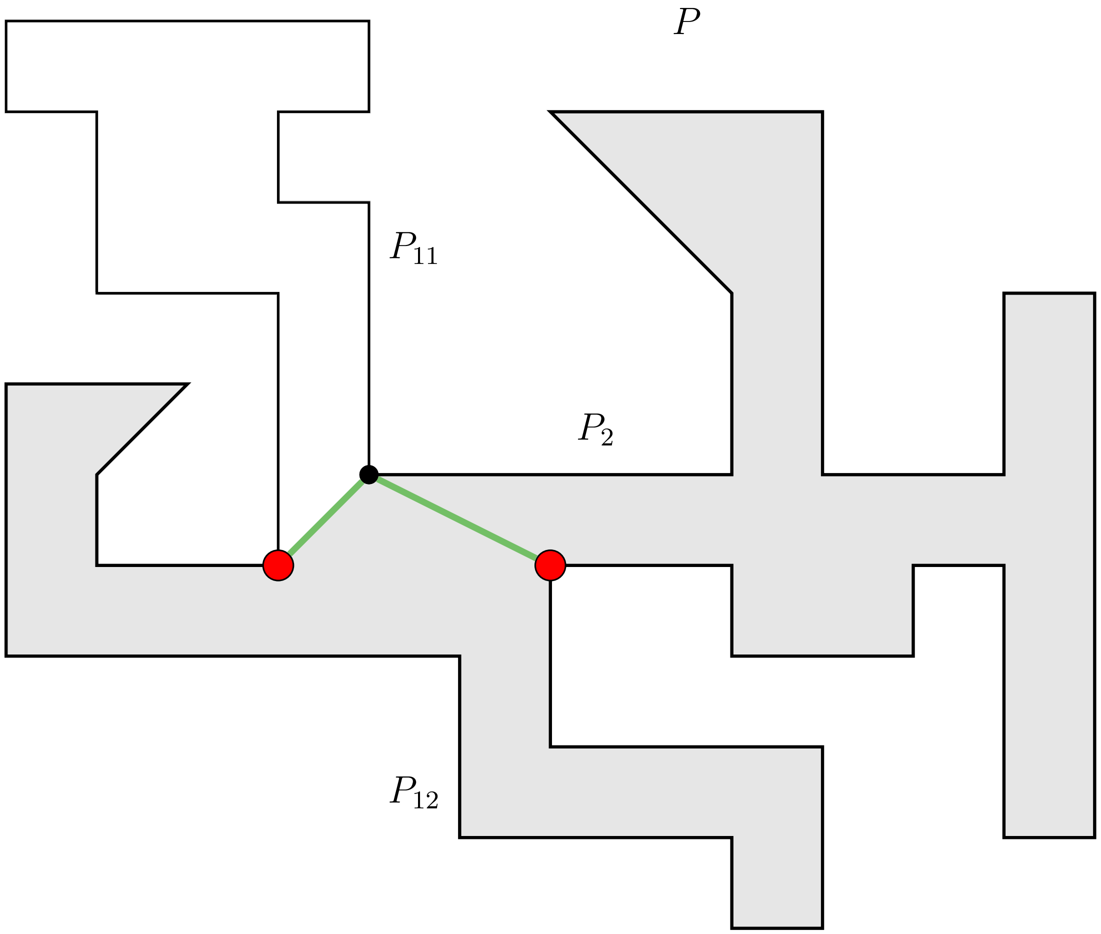
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{111}	8	3
P_{112}	6	3

$$n(P) = 40$$

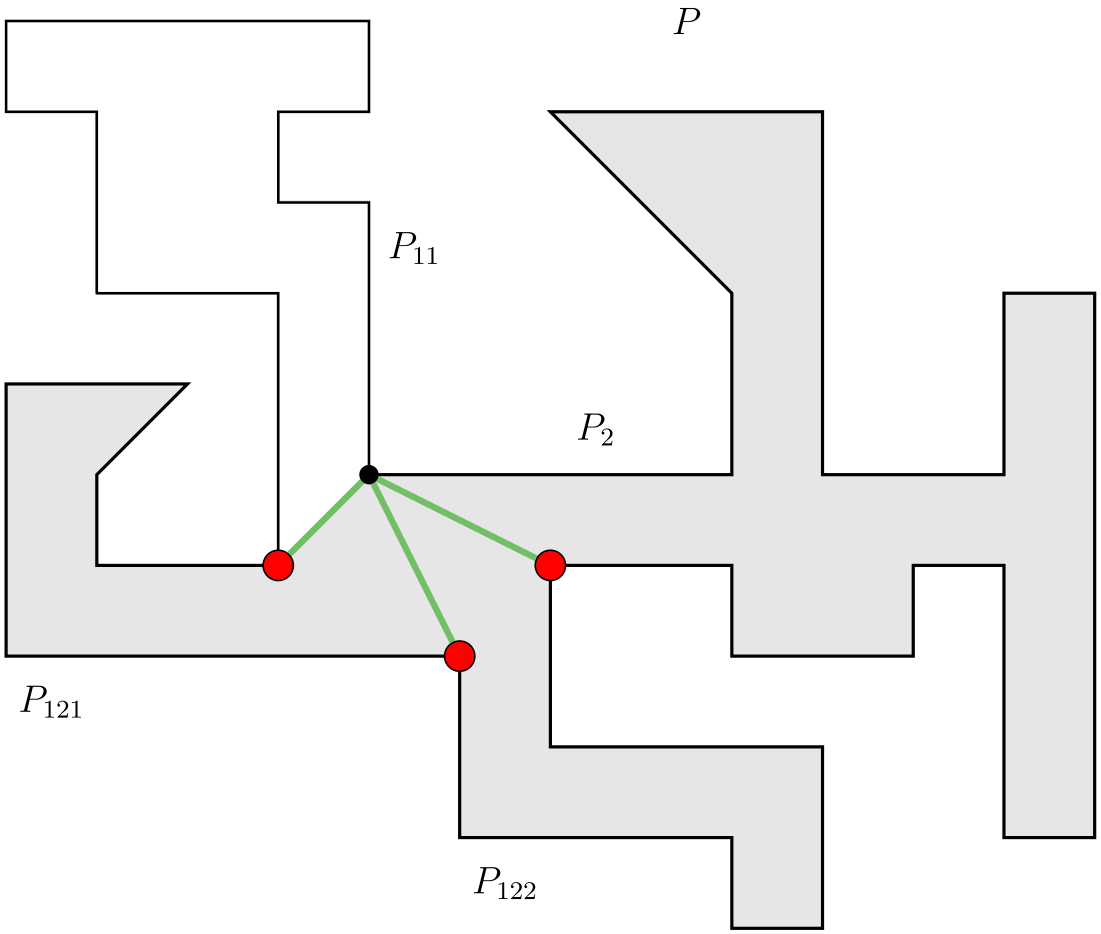
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2

$$n(P) = 40$$

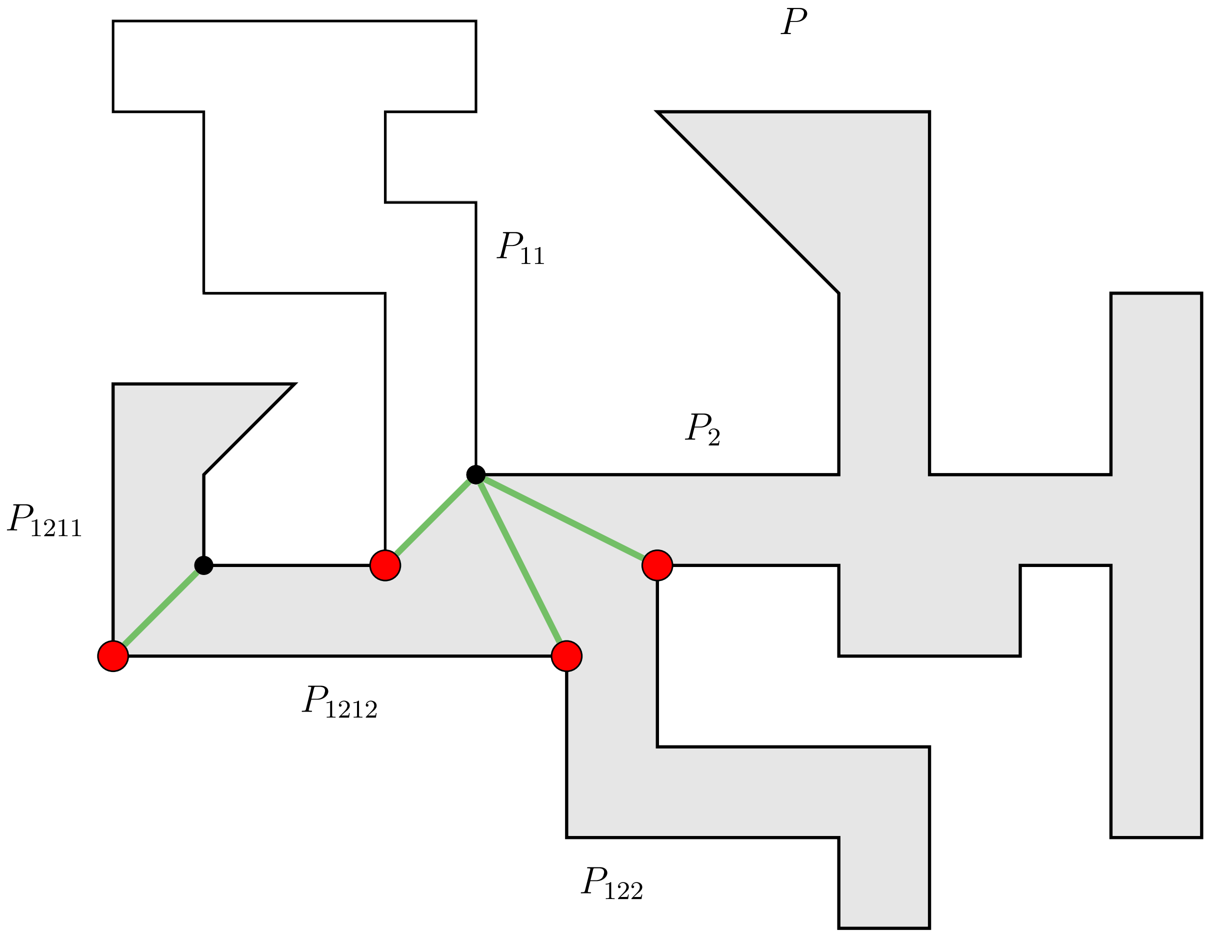
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3

$n(P) = 40$

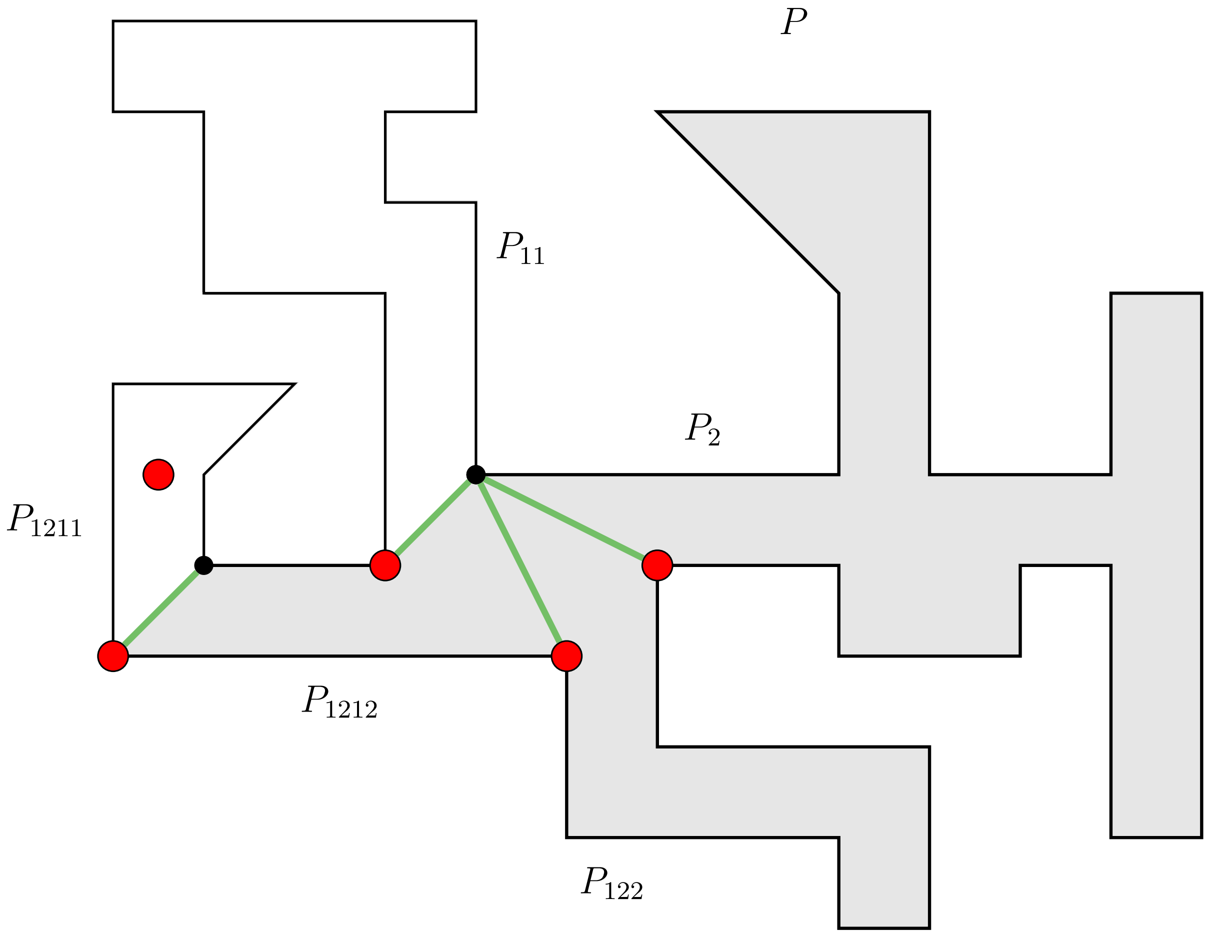
$\log_2 40 = 5,\ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1211}	5	4
P_{1212}	5	4

$$n(P) = 40$$

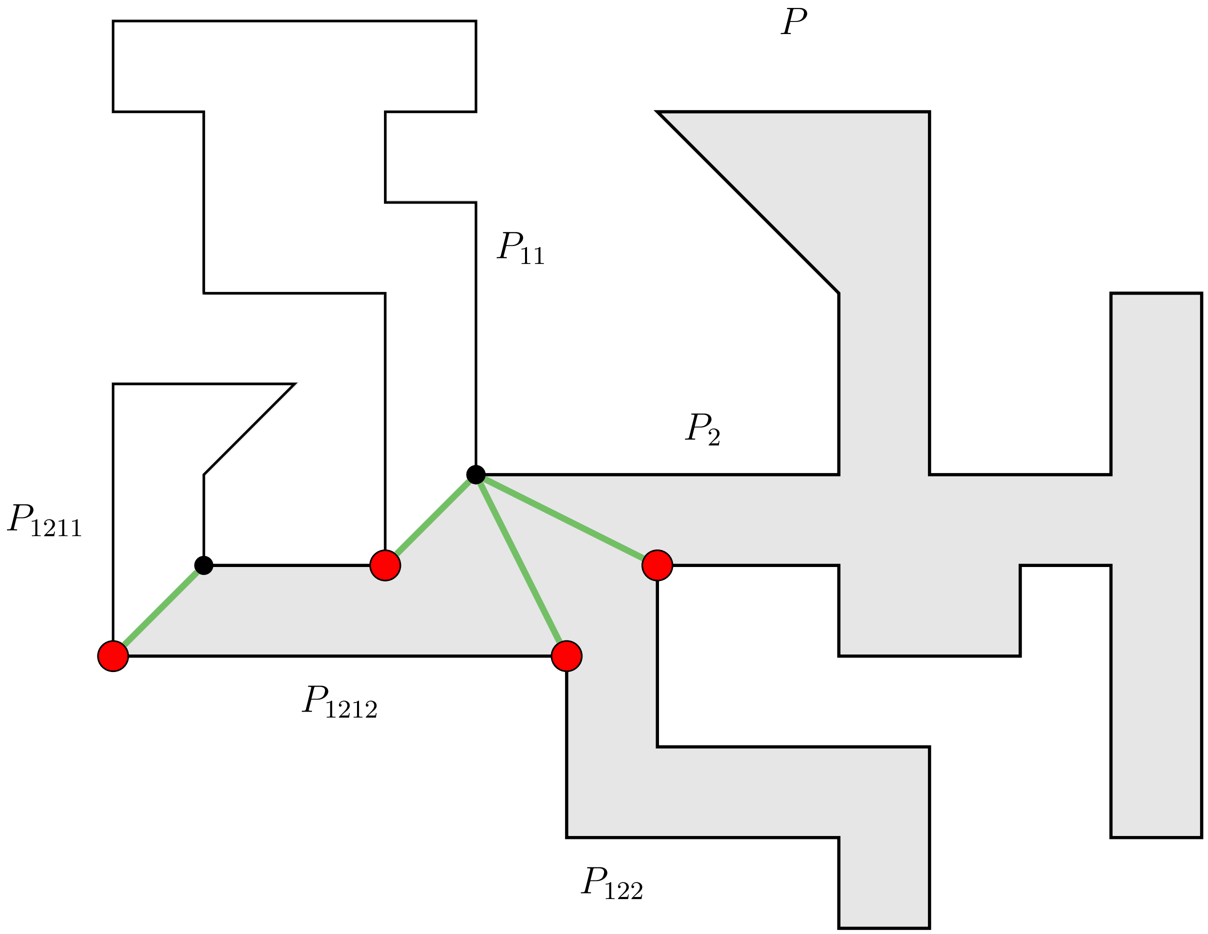
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1211}	5	4
P_{1212}	5	4
P_{1211}	5	5

$$n(P) = 40$$

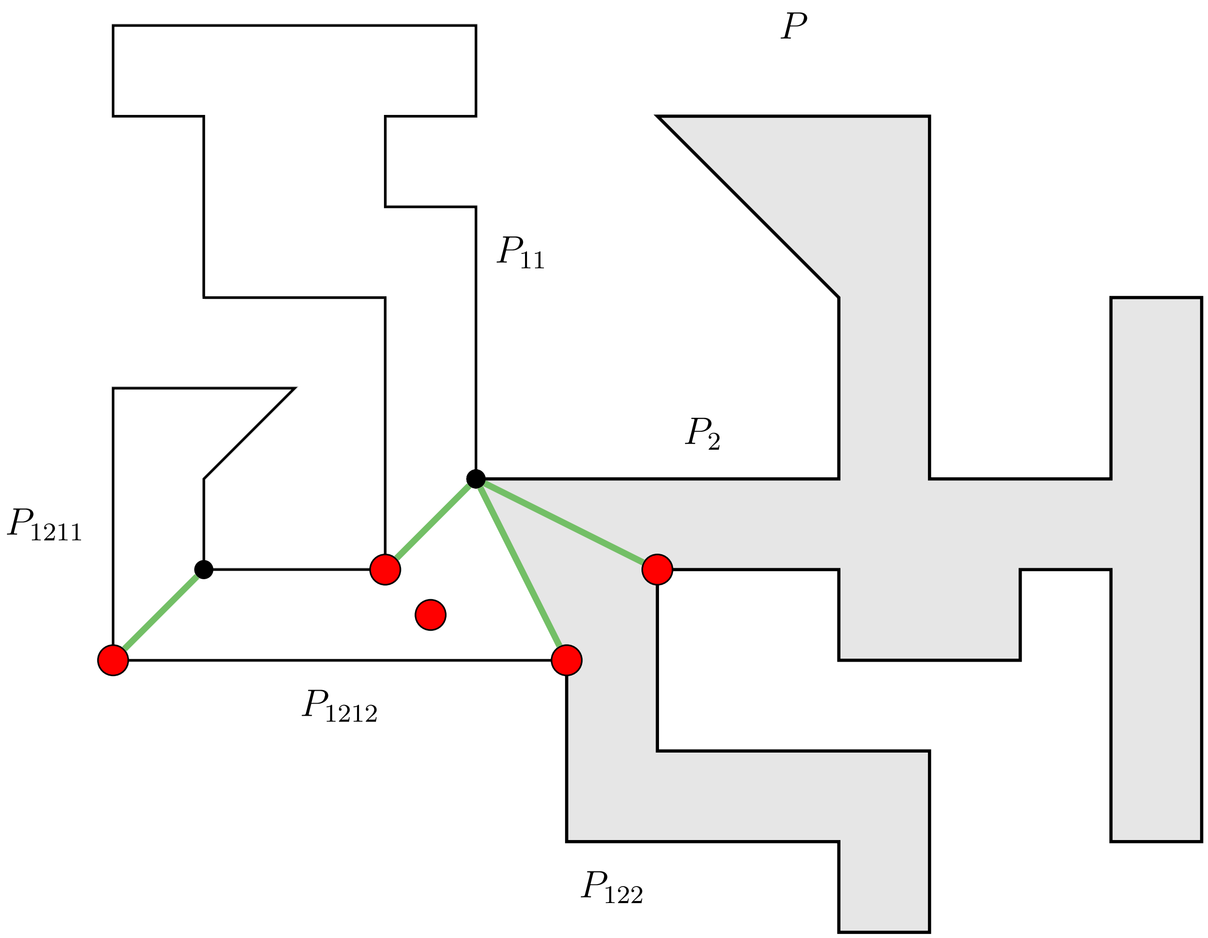
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1211}	5	4
P_{1212}	5	4

$n(P) = 40$

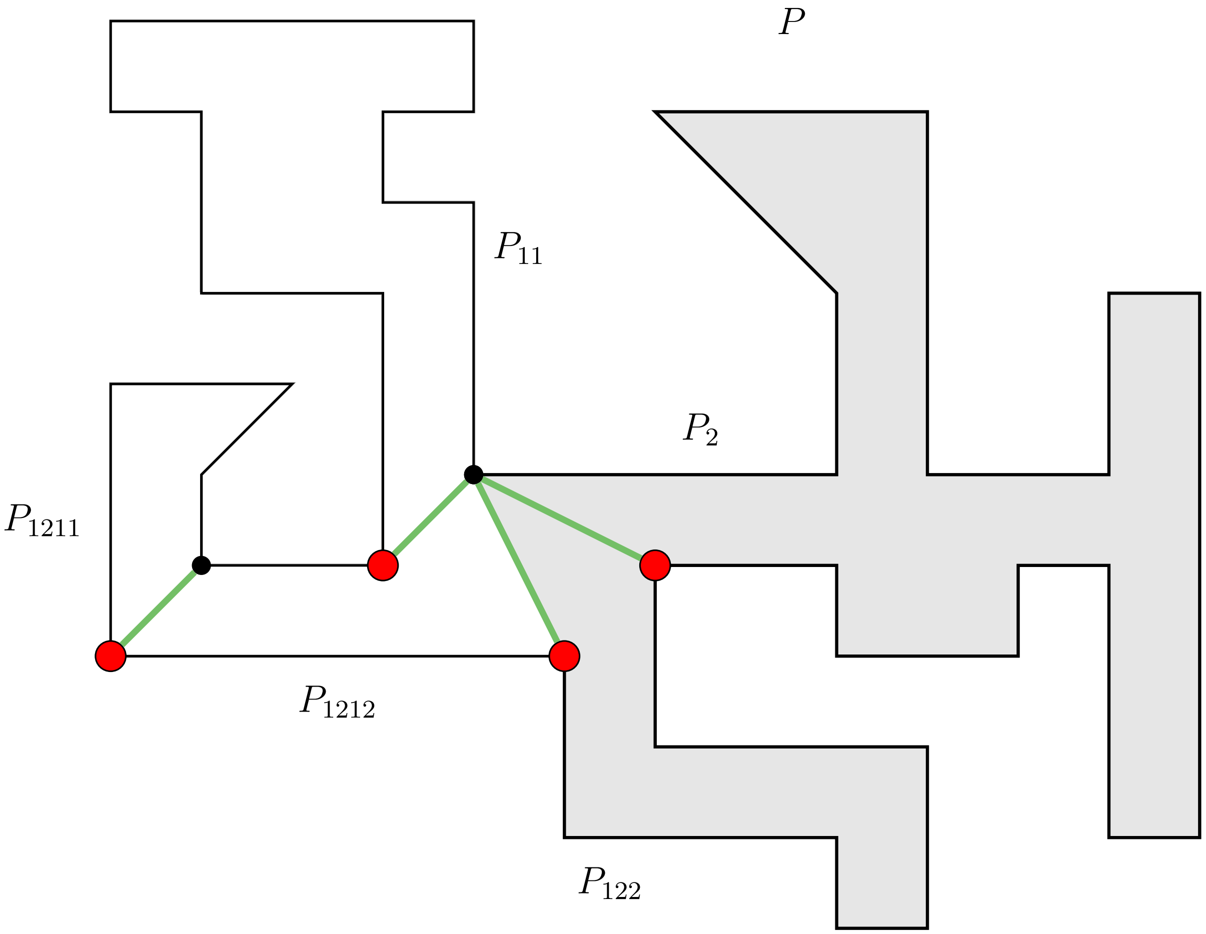
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1211}	5	4
P_{1212}	5	4
P_{1212}	5	5

$n(P) = 40$

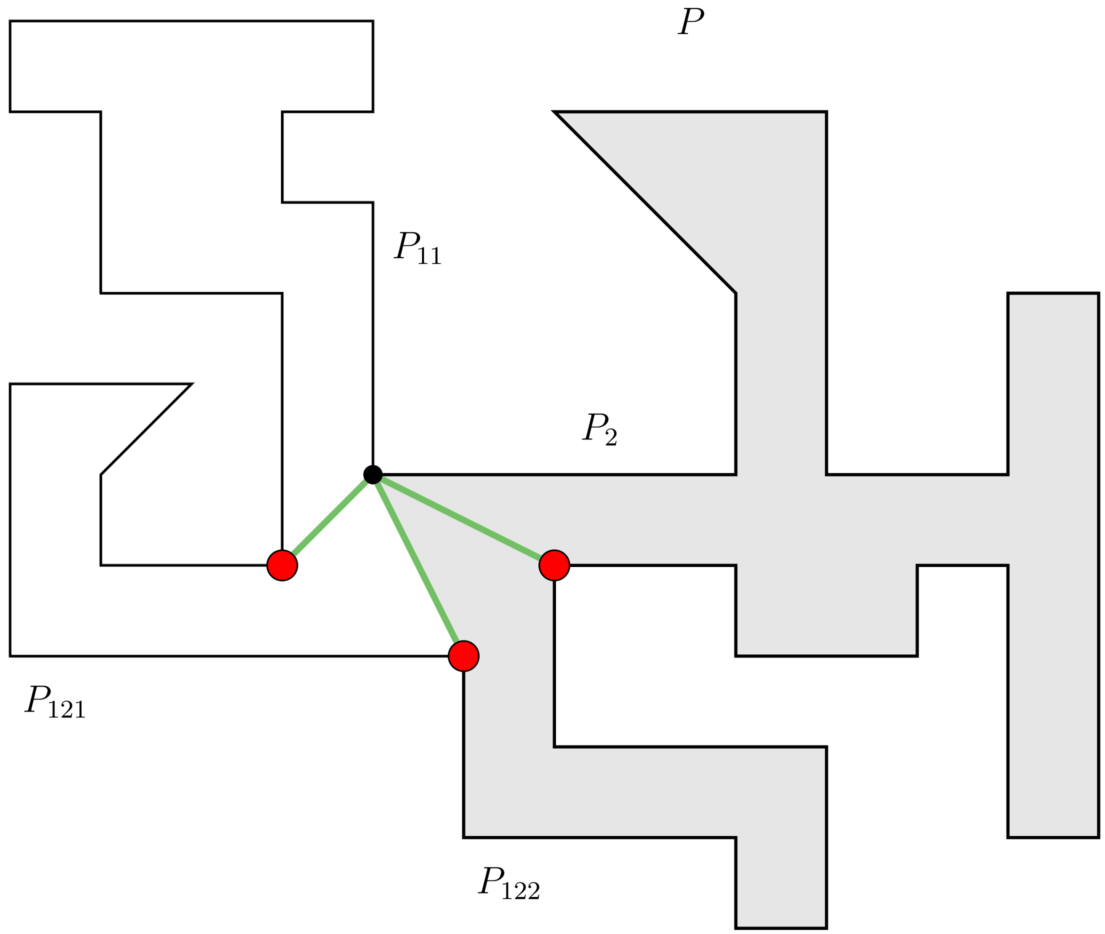
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1211}	5	4
P_{1212}	5	4

$$n(P) = 40$$

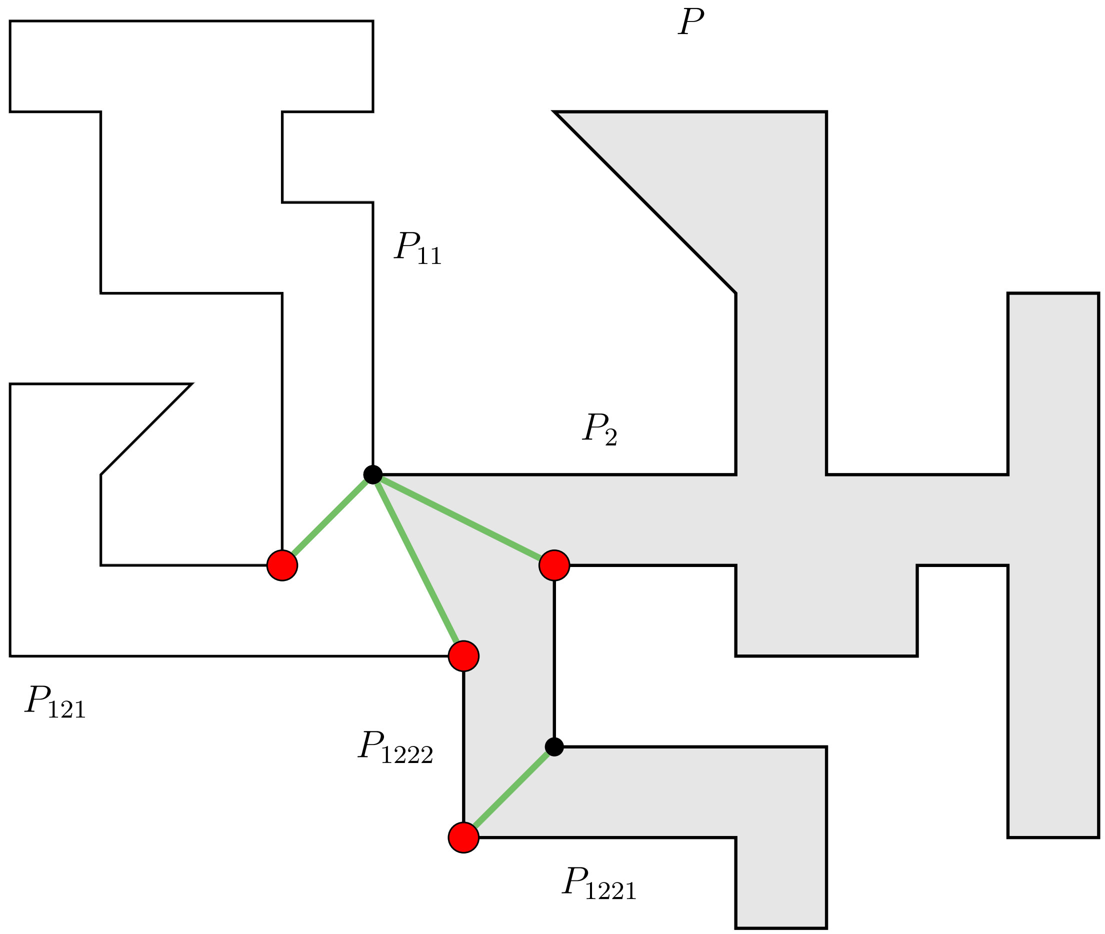
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3

$n(P) = 40$

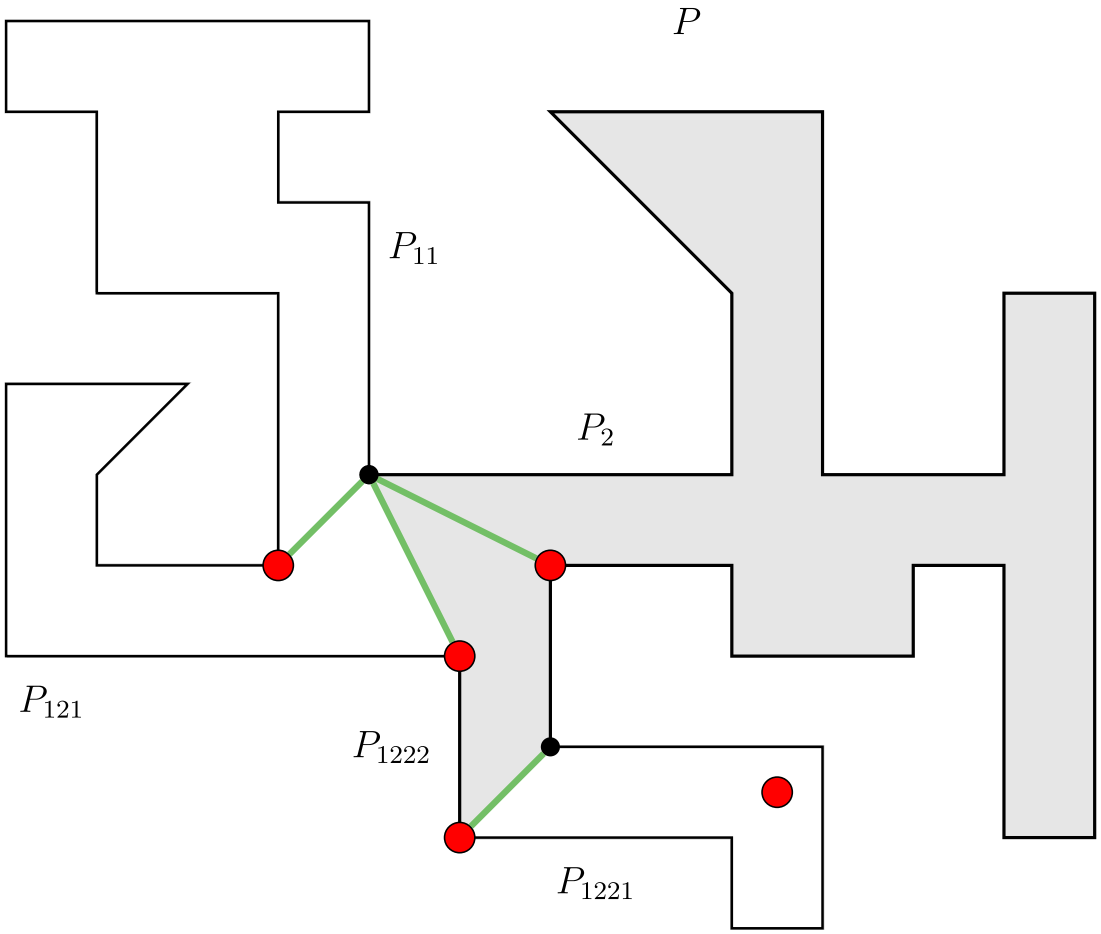
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1221}	6	4
P_{1222}	5	4

$$n(P) = 40$$

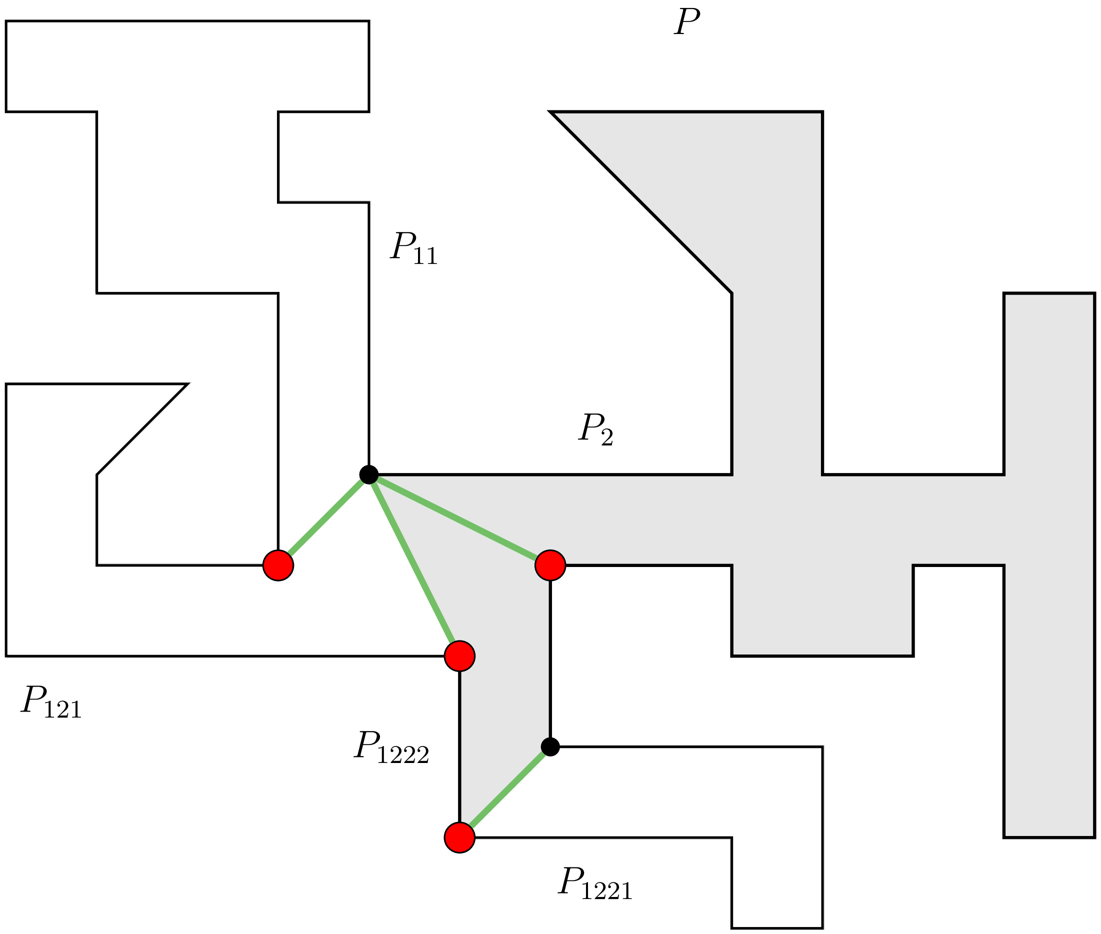
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1221}	6	4
P_{1222}	5	4
P_{1221}	6	5

$$n(P) = 40$$

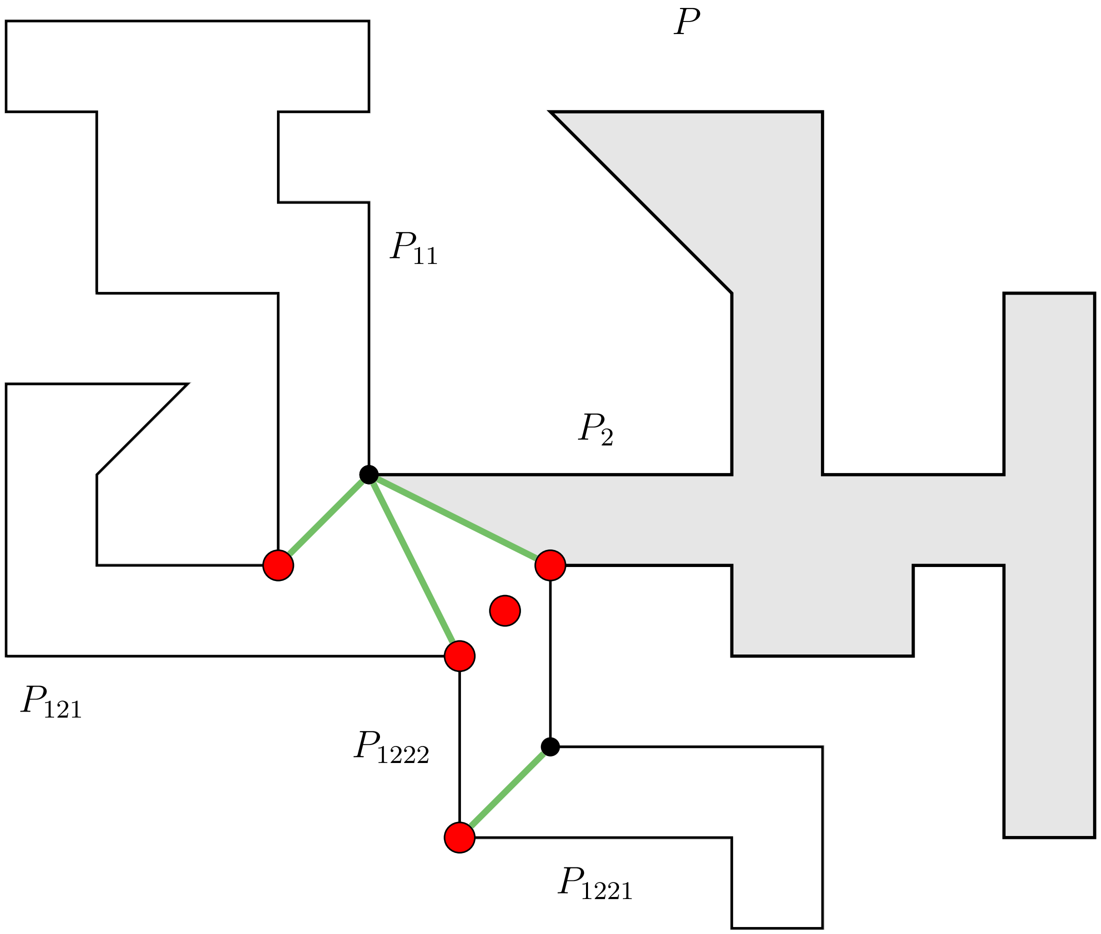
$$\log_2 40 = 5, \dots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1221}	6	4
P_{1222}	5	4

$$n(P) = 40$$

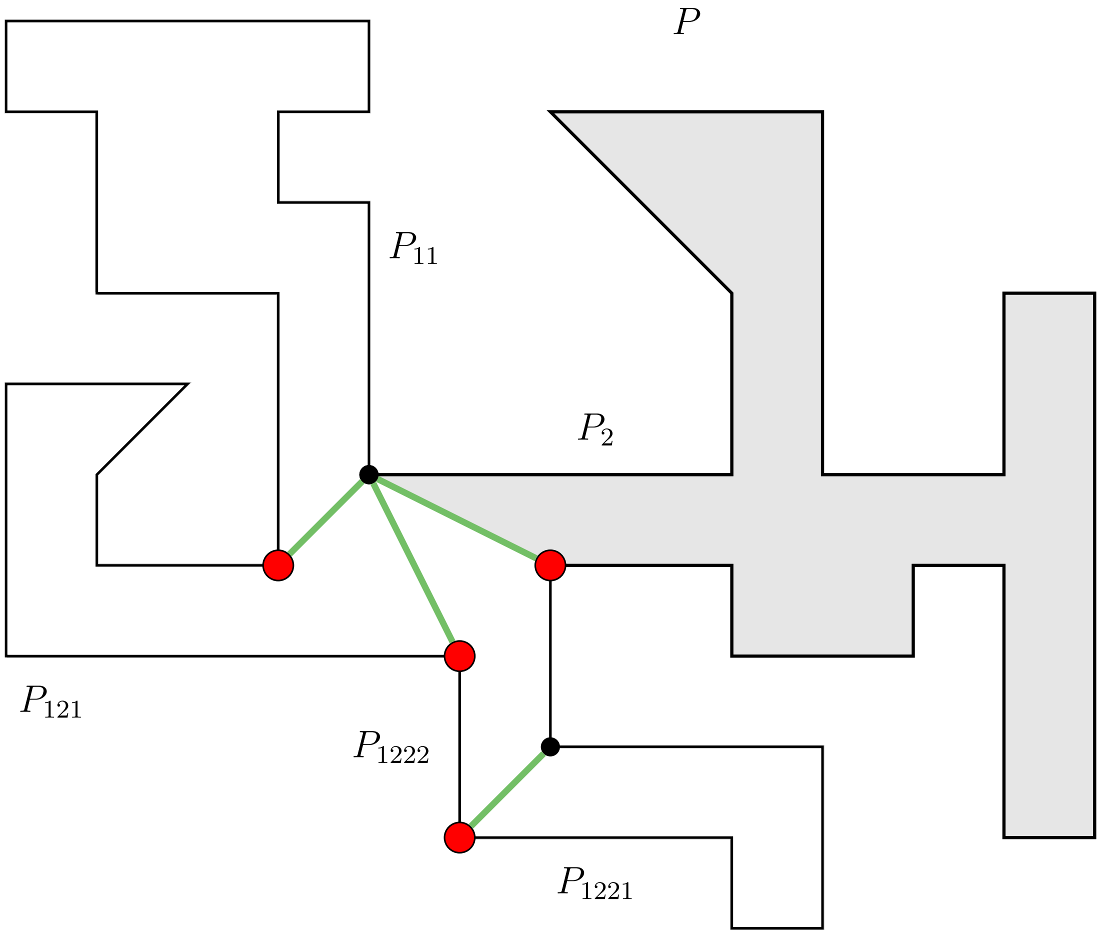
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1221}	6	4
P_{1222}	5	4
P_{1222}	5	5

$$n(P) = 40$$

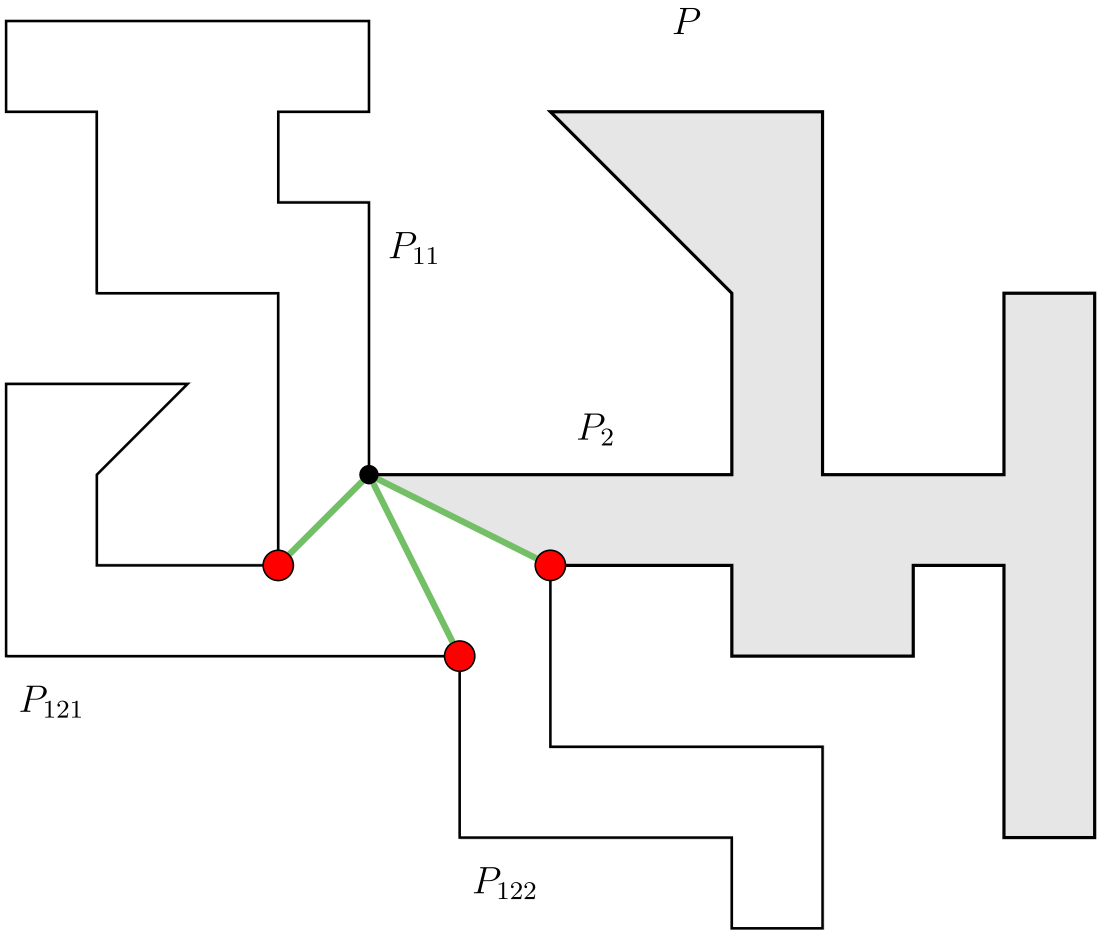
$$\log_2 40 = 5, \ldots$$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3
P_{1221}	6	4
P_{1222}	5	4

$n(P) = 40$

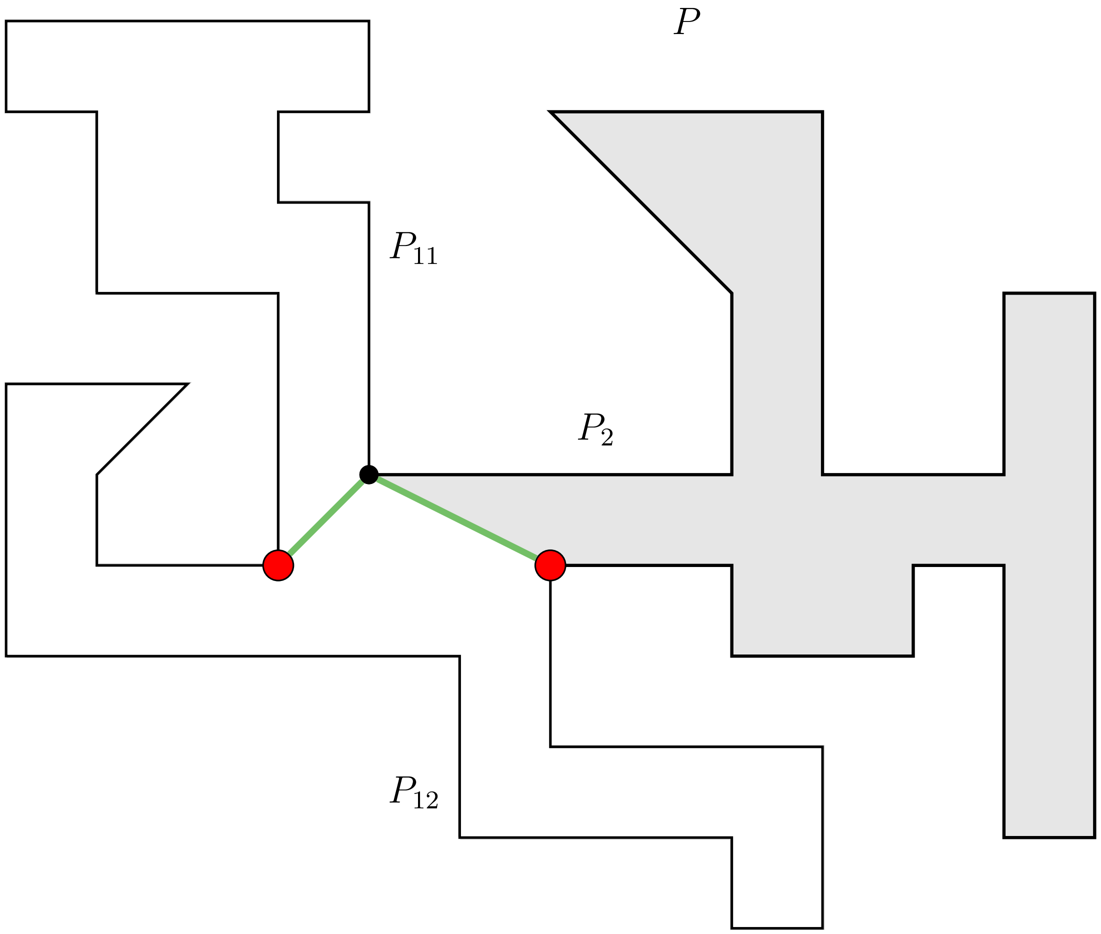
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2
P_{121}	8	3
P_{122}	9	3

$n(P) = 40$

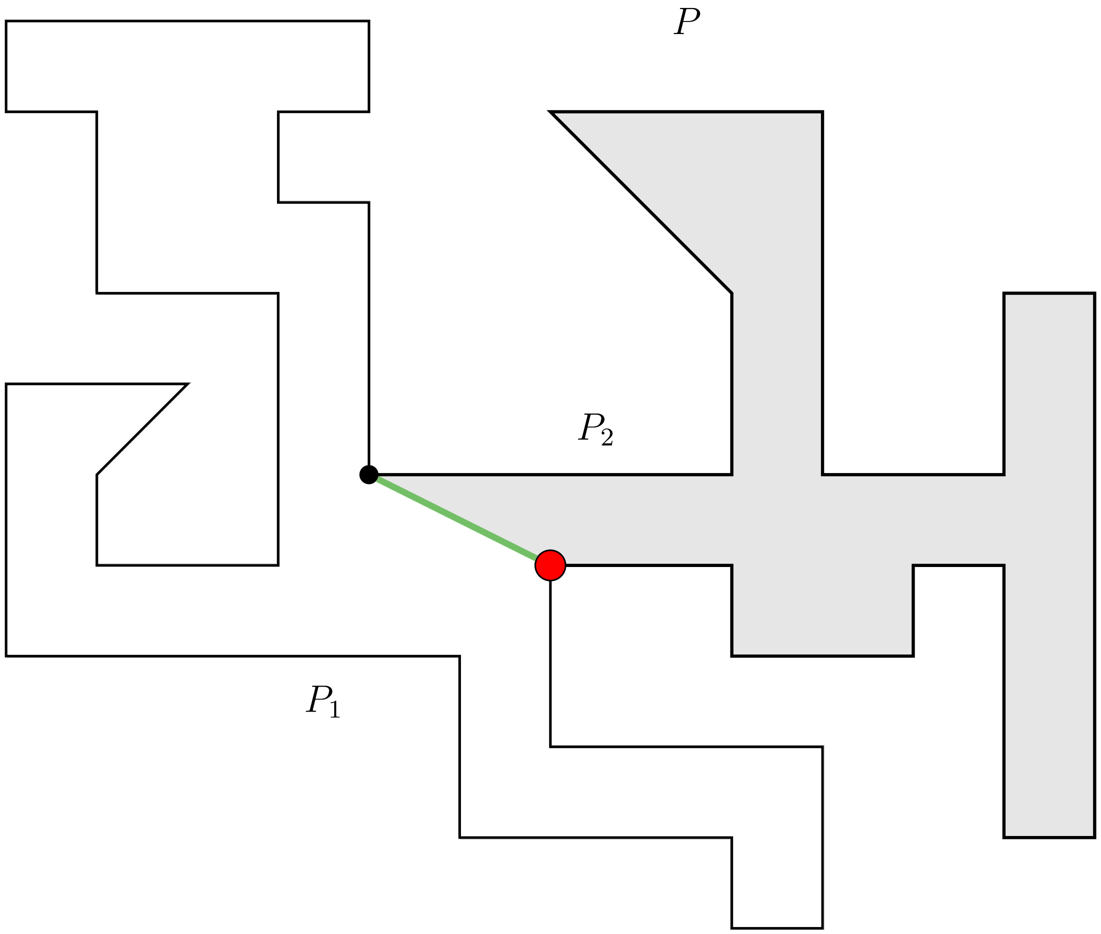
$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1
P_{11}	12	2
P_{12}	15	2

$n(P) = 40$

$\log_2 40 = 5, \ldots$



wielokąt	liczba wierzchołków	liczba agentów
P	40	0
P_1	25	1
P_2	17	1

$$n(P) = 40$$

$$\log_2 40 = 5, \ldots$$

Oszacowanie górne rzędu $O(\log n)$ /Guibas *et al.* 1999/

- Wielokąt o $3, \ldots, 6$ wierzchołkach może być przeszukany przez jednego agenta.
- Niech d będzie przekątną, której istnienie gwarantuje twierdzenie Chazelle'a. Umieśćmy jednego agenta A w końcu tej przekątnej.
- Wysyłamy pozostałych agentów do przeszukania najpierw wielokąta P_1, a następnie wielokąta P_2. (Umiejscowienie agenta A na przekątnej d gwarantuje, że intruz nie przebiegnie z P_1 do P_2 (i na odwrót) niezauważony.)

$$X(n) = \begin{cases} 1 & \text{dla } 3 \leq n \leq 6; \\ X(\lfloor \frac{2n}{3} \rfloor + 2) + 1 & \text{w przeciwnym wypadku.} \end{cases}$$

Można wykazać, że rozwiązaniem powyższej zależności jest $X(n) = O(\log n)$.

Twierdzenie 4.7. (Guibas *et al.* 1999)
$\Theta(\log n)$ *mobilnych agentów czasami potrzeba, ale zawsze wystarcza, aby przeszukać dowolny n-wierzchołkowy wielokąt prosty.*

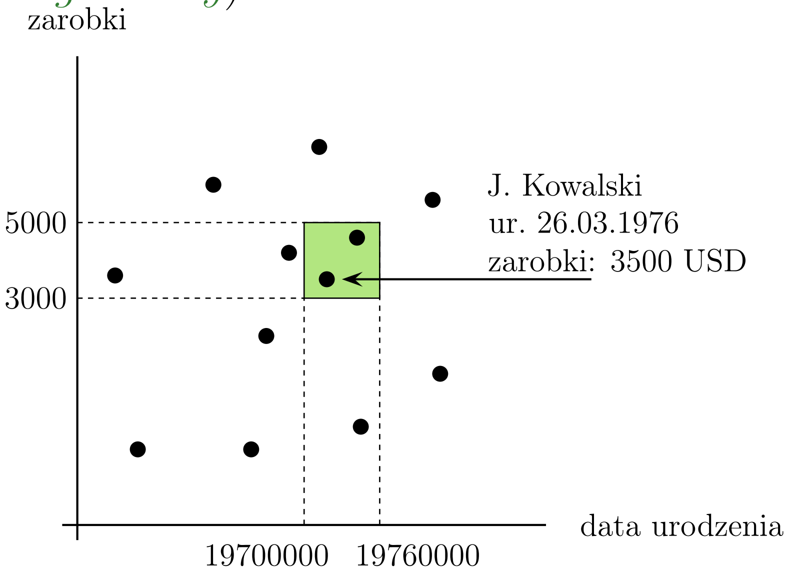
- W przypadku wielokątów prostych z tzw. *dziurami*: $X(n,h) = \Theta(\sqrt{h} + \log n)$.

5. PRZESZUKIWANIE OBSZARÓW PROSTOKĄTNYCH

Niech dany będzie zbiór punktów $S \subset \mathbb{R}^d, d \geq 1$. Chcemy wyznaczyć taki podzbiór $S' \subseteq S$ punktów, że każdy z elementów $p \in S'$ mieści się w zadanym d-wymiarowym prostopadłościanie $R = [x_1^1, x_2^1] \times [x_1^2, x_2^2] \times \cdots \times [x_1^d, x_2^d]$, $x_1^i, x_2^i \in \mathbb{R}$, $x_1^i \leq x_2^i$, $i = 1, \ldots, d$. W geometrii obliczeniowej takie zapytanie nazywa się zapytaniem o *obszar prostokątny* (*obszar ortogonalny*).

M. de Berg *et al.*
Geometria obliczeniowa
rozdziały 5 i 10, WNT (2007)

Problem. Mając daną bazę danych pracowników, podać wszystkich pracowników urodzonych w latach 1970-1976, którzy zarabiają między 3000 a 5000 USD miesięcznie.



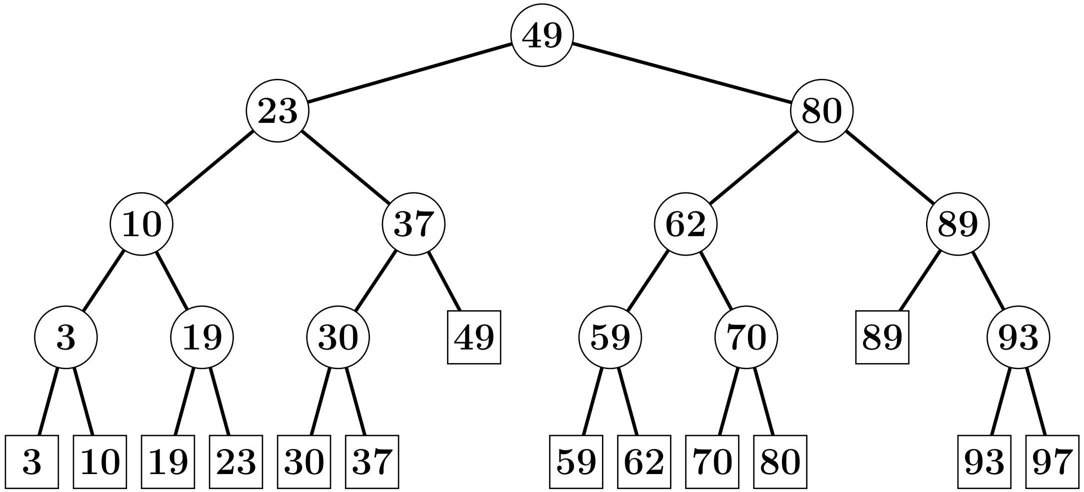
Rozwiązanie. Jeśli jesteśmy zainteresowani odpowiedzią na zapytanie dotyczące d pól rekordów, przekształcamy rekordy w punkty d-wymiarowej przestrzeni. Zapytanie o wszystkie rekordy, których pola leżą między poszczególnymi wartościami, zamienia się wtedy w zapytanie dotyczące wszystkich punktów wewnątrz d-wymiarowego prostopadłościanu o bokach równoległych do osi.

5.1 PRZESZUKIWANIE OBSZARÓW JEDNOWYMIAROWYCH

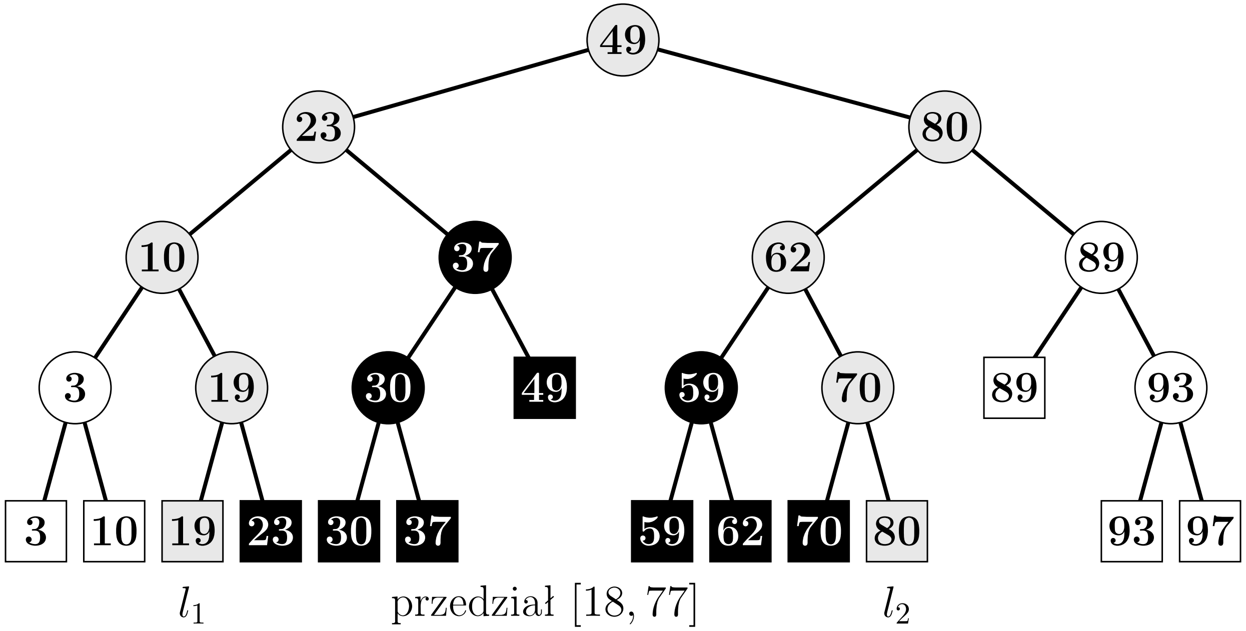
Problem. *Niech S będzie danym zbiorem punktów na prostej rzeczywistej $\mathbb{R}$.*

Wejście: *Przedział zapytania $R = [x_1, x_2]$.*

Wyjście: *Wszystkie punkty z $S \cap R$.*



Punkty/dane przechowywane są w strukturze zrównoważonego drzewa przeszukiwań binarnych $\mathcal{T}$. Liście drzewa $\mathcal{T}$ pamiętają punkty S, a węzły wewnętrzne $\mathcal{T}$ przechowują wartości dzielące: lewe poddrzewo węzła v zawiera wszystkie punkty mniejsze lub równe `klucz`(v) (wartość pamiętana w węźle v), a prawe poddrzewo zawiera wszystkie punkty większe od `klucz`(v).

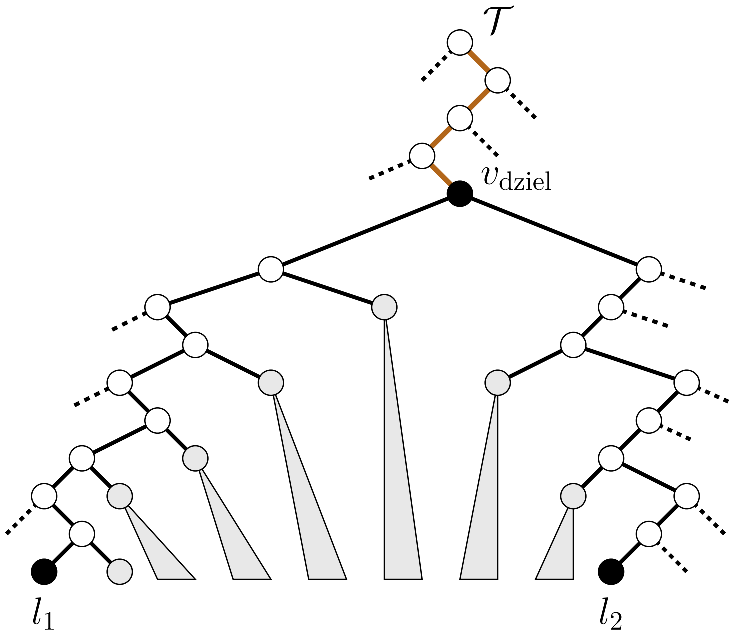


Idea algorytmu zapytania o przedział $[x_1, x_2]$

- Poszukujemy liści o kluczach x_1 i x_2 w drzewie $\mathcal{T}$. Niech l_1 i l_2 będą liśćmi, w których kończy się przeszukiwanie.
- Wówczas punkty z przedziału $[x_1, x_2]$ są punktami pamiętanymi w liściach między l_1 i l_2 oraz, być może, punktami pamiętanymi w l_1 i l_2.

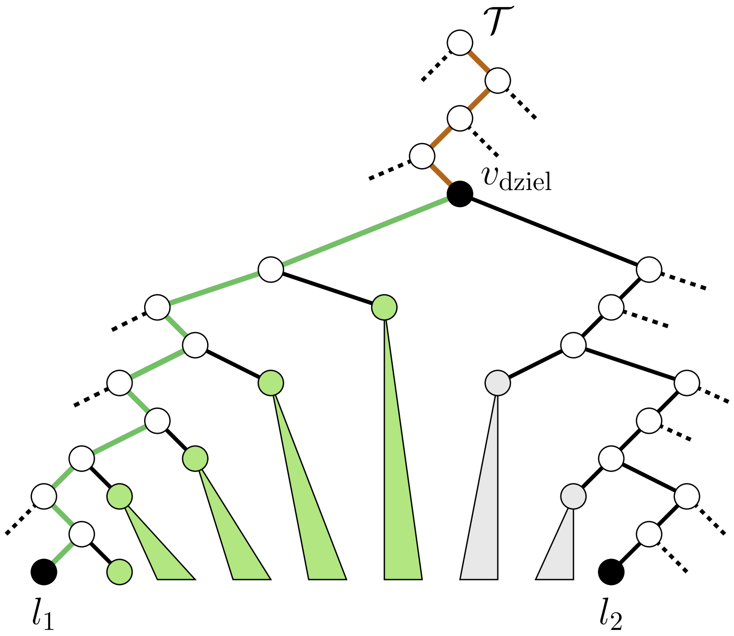
Zatem interesują nas liście pewnych poddrzew między ścieżkami przeszukiwań do l_1 i l_2. Dokładniej, poddrzewa te są zakorzenione w węzłach, które są między ścieżkami przeszukiwań i których rodzice są na ścieżce przeszukiwań.

- Aby znaleźć te węzły, poszukujemy najpierw węzła v_{dziel}, w którym ścieżki do x_1 i x_2 „rozchodzą się" (*najniższy wspólny przodek* dla l_1 i l_2).
- Zaczynając od v_{dziel}, idziemy dalej ścieżką poszukiwania x_1. W każdym węźle, z którego ścieżka skręca w lewo, wyliczamy wszystkie liście z jego prawego poddrzewa.
- Podobnie podążamy ścieżką poszukiwania x_2 i wyliczamy liście w lewym poddrzewie węzłów, w którym ścieżka skręca w prawo.
- Na koniec sprawdzamy punkty pamiętane w liściach l_1 i l_2.



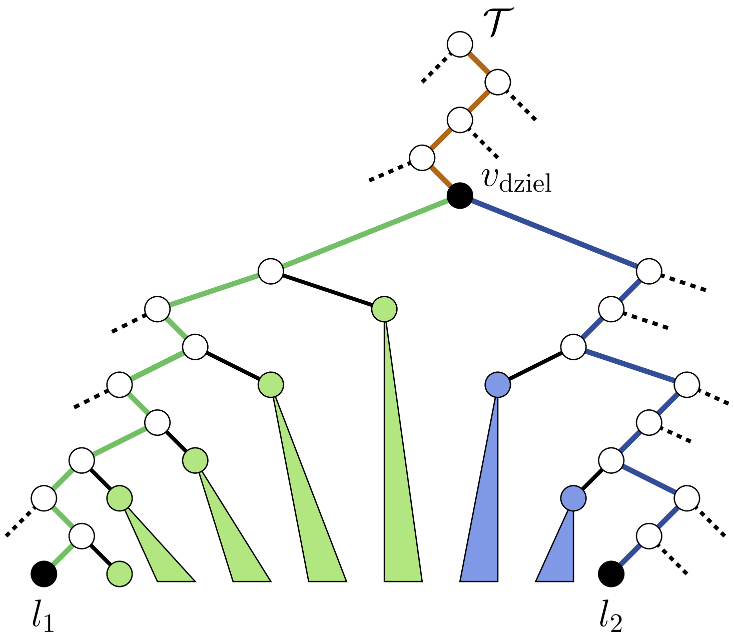
Zatem interesują nas liście pewnych poddrzew między ścieżkami przeszukiwań do l_1 i l_2. Dokładniej, poddrzewa te są zakorzenione w węzłach, które są między ścieżkami przeszukiwań i których rodzice są na ścieżce przeszukiwań.

- Aby znaleźć te węzły, poszukujemy najpierw węzła v_{dziel}, w którym ścieżki do x_1 i x_2 „rozchodzą się” (*najniższy wspólny przodek* dla l_1 i l_2).
- Zaczynając od v_{dziel}, idziemy dalej ścieżką poszukiwania x_1. W każdym węźle, z którego ścieżka skręca w lewo, wyliczamy wszystkie liście z jego prawego poddrzewa.
- Podobnie podążamy ścieżką poszukiwania x_2 i wyliczamy liście w lewym poddrzewie węzłów, w którym ścieżka skręca w prawo.
- Na koniec sprawdzamy punkty pamiętane w liściach l_1 i l_2.



Zatem interesują nas liście pewnych poddrzew między ścieżkami przeszukiwań do l_1 i l_2. Dokładniej, poddrzewa te są zakorzenione w węzłach, które są między ścieżkami przeszukiwań i których rodzice są na ścieżce przeszukiwań.

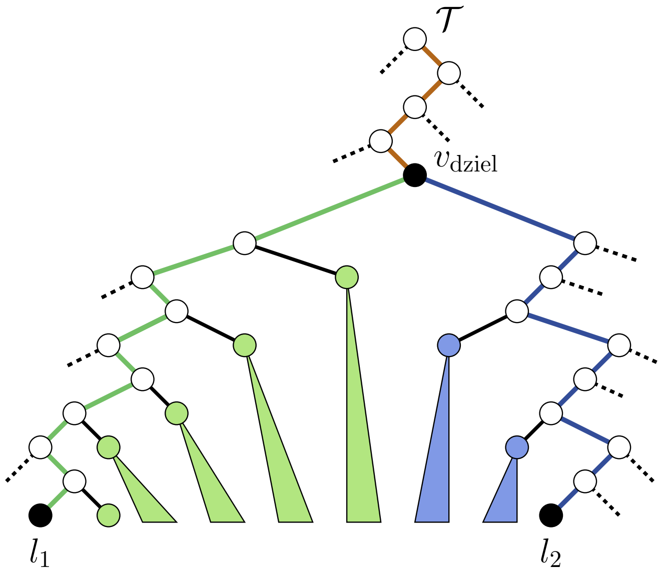
- Aby znaleźć te węzły, poszukujemy najpierw węzła v_{dziel}, w którym ścieżki do x_1 i x_2 „rozchodzą się” (*najniższy wspólny przodek* dla l_1 i l_2).
- Zaczynając od v_{dziel}, idziemy dalej ścieżką poszukiwania x_1. W każdym węźle, z którego ścieżka skręca w lewo, wyliczamy wszystkie liście z jego prawego poddrzewa.
- Podobnie podążamy ścieżką poszukiwania x_2 i wyliczamy liście w lewym poddrzewie węzłów, w którym ścieżka skręca w prawo.
- Na koniec sprawdzamy punkty pamiętane w liściach l_1 i l_2.



Lemat 5.1. *Powyższy algorytm wylicza dokładnie te i tylko te punkty z S, które leżą w obszarze zapytania $[x_1, x_2]$.*

↳ Należy wykazać, że każdy ze zgłaszanych punktów leży w obszarze $[x_1, x_2]$.

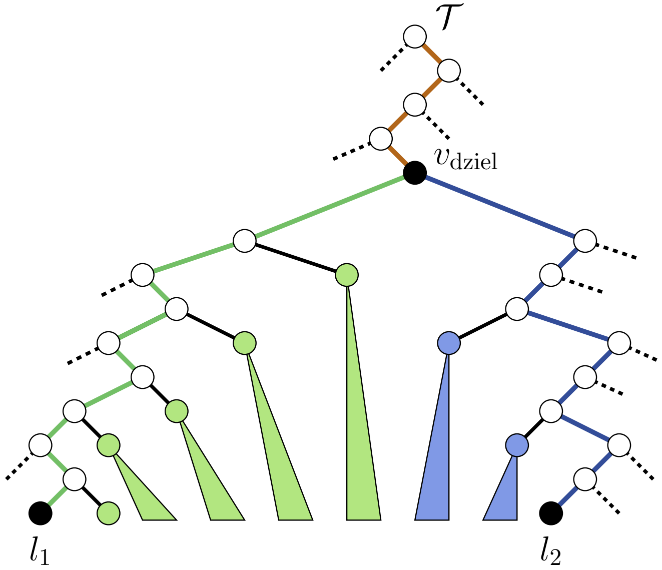
↳ Należy wykazać, że każdy punkt $\in S$, który należy do obszaru $[x_1, x_2]$, zostanie zgłoszony.



Lemat 5.1. *Powyższy algorytm wylicza dokładnie te i tylko te punkty z S, które leżą w obszarze zapytania $[x_1, x_2]$.*

↳ Należy wykazać, że każdy ze zgłaszanych punktów leży w obszarze $[x_1, x_2]$.

↳ Należy wykazać, że każdy punkt $\in S$, który należy do obszaru $[x_1, x_2]$, zostanie zgłoszony.



Stwierdzenie 5.2.
Niech S będzie zbiorem n punktów na prostej rzeczywistej $\mathbb{R}$. Zbiór S można zapamiętać w zrównoważonym drzewie poszukiwań binarnych, które korzysta z pamięci rzędu $O(n)$ i ma czas konstrukcji $O(n \log n)$, tak że punkty w obszarze zapytania można podać w czasie $O(k + \log n)$, gdzie k jest liczbą podawanych punktów.

↳ Liczba węzłów wewnętrznych każdego drzewa binarnego jest mniejsza od liczby liści, a zatem wyliczenie wszystkich liści danego poddrzewa wymaga czasu liniowego względem liczby k podawanych punktów.

↳ Jako że drzewo jest zrównoważone, liczba odwiedzanych węzłów przy poszukiwaniu x_1 i x_2 wynosi $O(\log n)$.

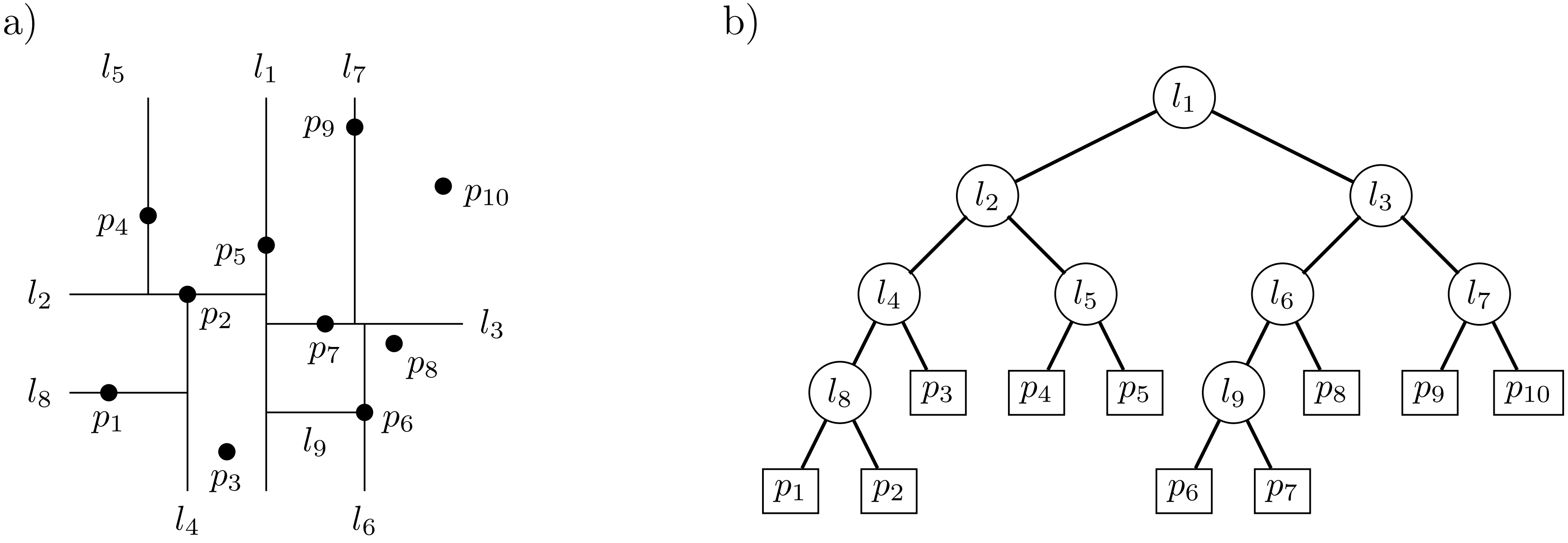
5.2 ZAPYTANIA W PRZESTRZENI DWYWYMIAROWEJ: KD-DRZEWA

Problem. (Przeszukiwanie obszarów ortogonalnych)

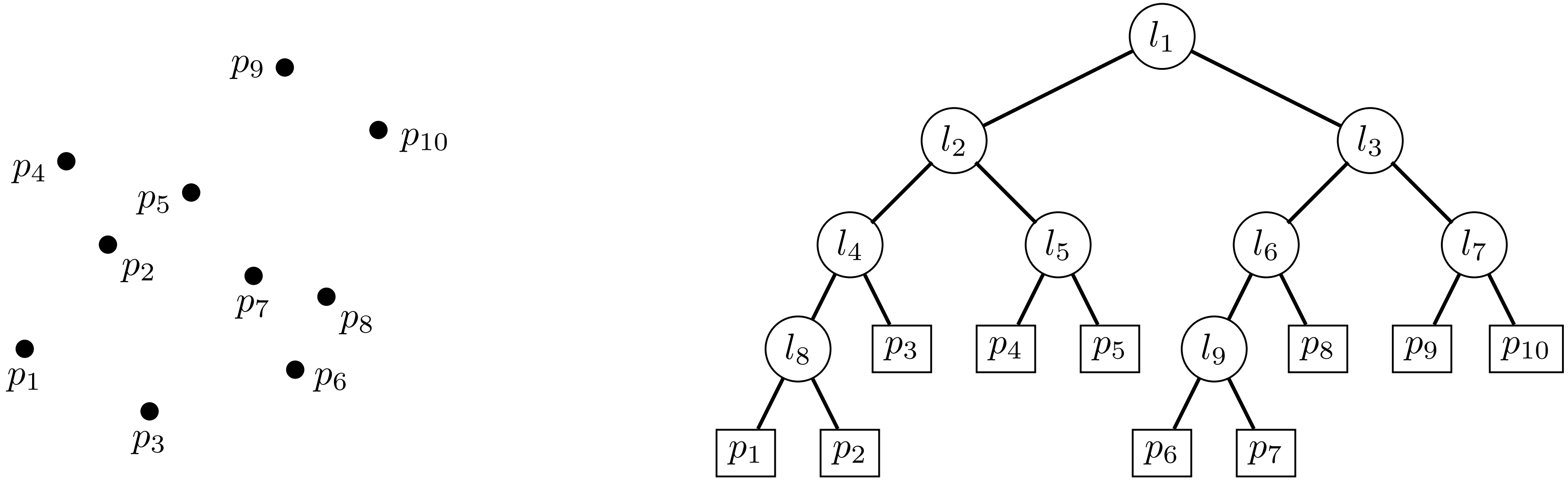
Niech S będzie danym zbiorem punktów na płaszczyźnie rzeczywistej $\mathbb{R}^2$.

Wejście: *Obszar zapytania $R = [x_1, x_2] \times [y_1, y_2]$.*

Wyjście: *Wszystkie punkty z S, które należą do R.*

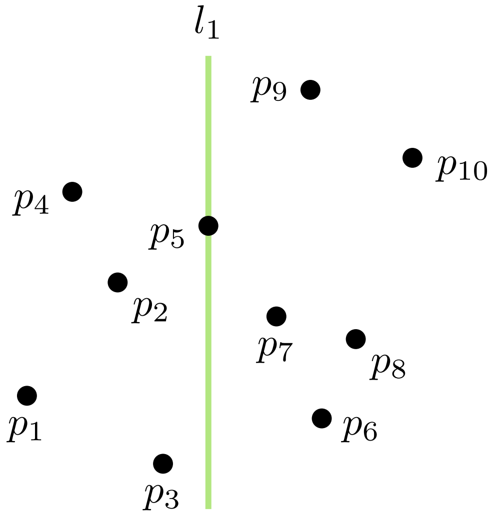


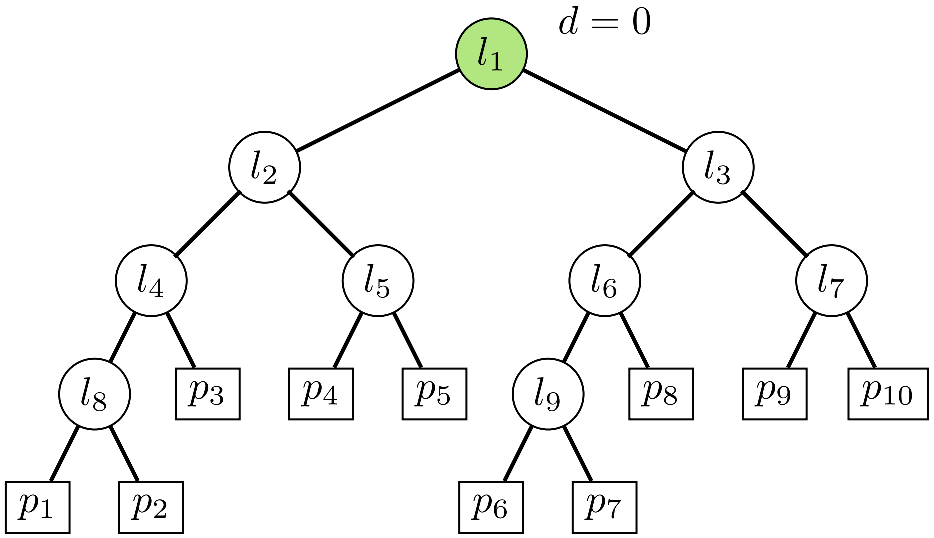
(a) Podział płaszczyzny i (b) odpowiadające mu kd-drzewo.



Idea konstrukcji kd-drzewa dla zbioru $S \subset \mathbb{R}^2$

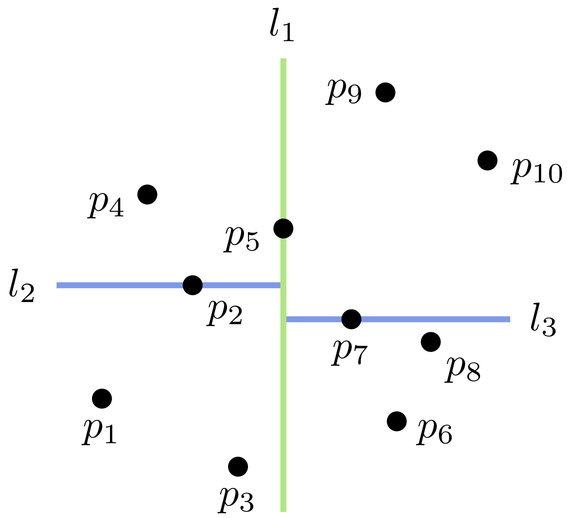
- Dodatkowy parametr d na wejściu (początkowo $d = 0$).
- Jeśli S zawiera tylko jeden punkt, zwróć liść pamiętający ten punkt.
- W przeciwnym wypadku, jeśli d jest parzyste, podziel S na dwa zbiory S_1 i S_2 pionową prostą l przechodzącą przez medianę współrzędnych x punktów z S, gdzie S_1 zawiera punkty na lewo lub na prostej l, a S_2 zawiera punkty na prawo od prostej l.
- $\ldots$

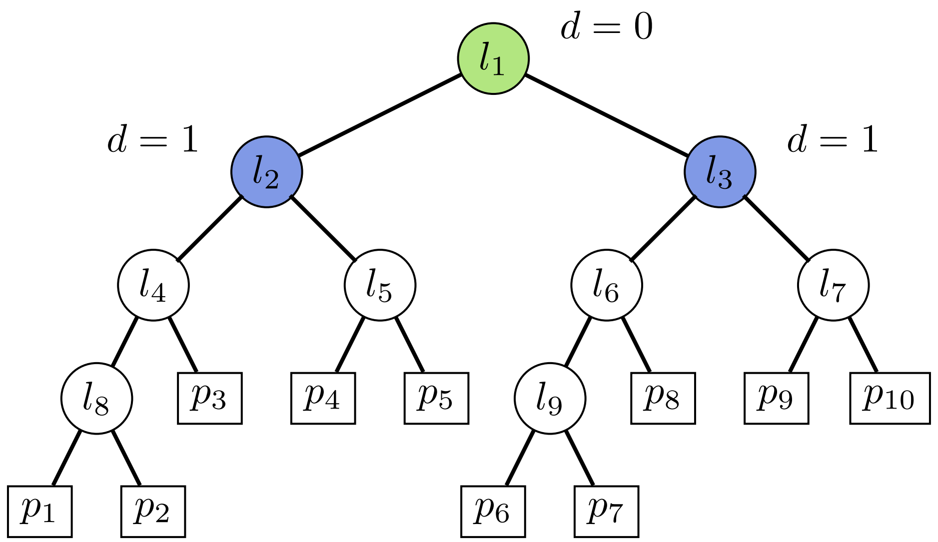




Idea konstrukcji kd-drzewa dla zbioru $S \subset \mathbb{R}^2$

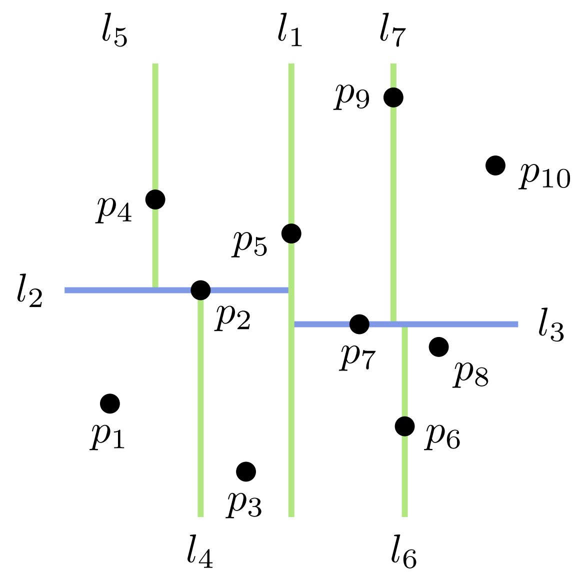
- Dodatkowy parametr d na wejściu (początkowo $d = 0$).
- Jeśli S zawiera tylko jeden punkt, zwróć liść pamiętający ten punkt.
- W przeciwnym wypadku, jeśli d jest parzyste, podziel S na dwa zbiory S_1 i S_2 pionową prostą l przechodzącą przez medianę współrzędnych x punktów z S, gdzie S_1 zawiera punkty na lewo lub na prostej l, a S_2 zawiera punkty na prawo od prostej l.
- ...

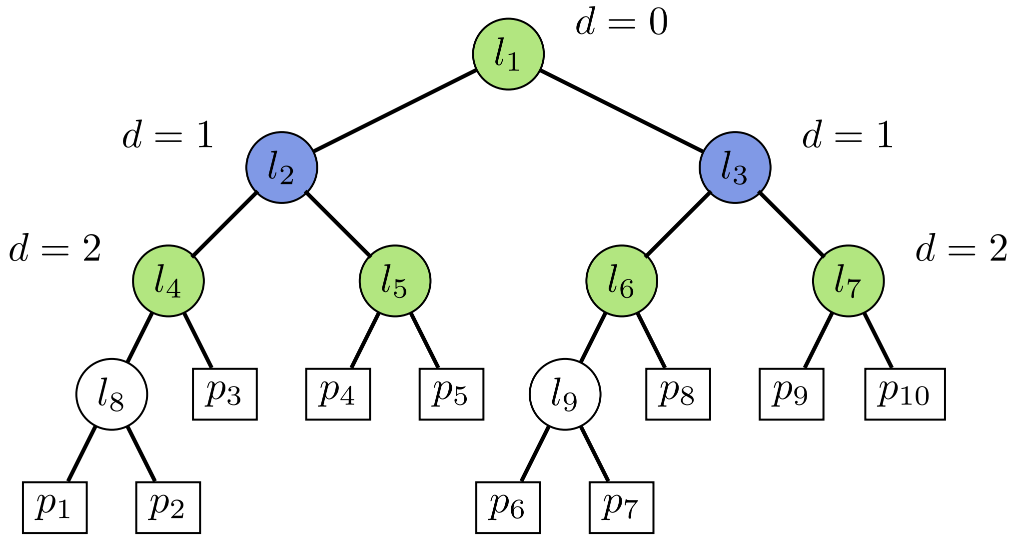




Idea konstrukcji kd-drzewa dla zbioru $S \subset \mathbb{R}^2$

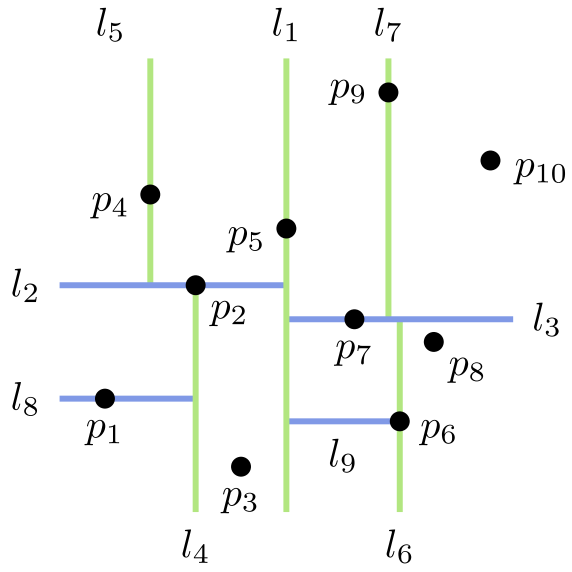
- …
- W przeciwnym wypadku, jeśli d jest nieparzyste, podziel S na dwa zbiory S_1 i S_2 poziomą prostą l przechodzącą przez medianę współrzędnych y punktów z S, gdzie S_1 zawiera punkty poniżej lub na prostej l, a S_2 zawiera punkty powyżej prostej l.
- Rekurencyjnie wyznacz kd-drzewa T_1 dla S_1 oraz T_2 dla S_2 z parametrem $d+1$.
- Zwróć korzeń v (z prostą l), z T_1 jako jego lewym synem, a T_2 – prawym.

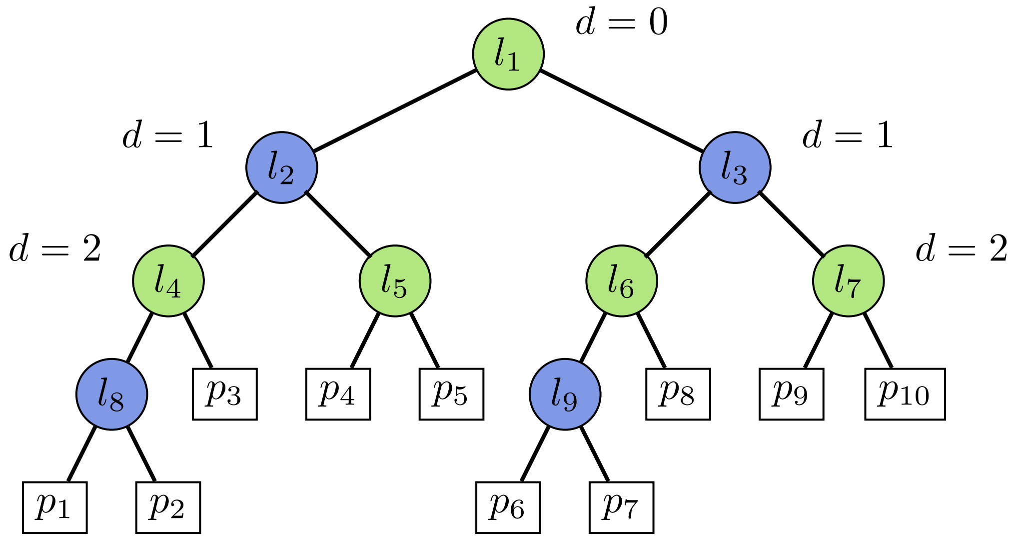




Idea konstrukcji kd-drzewa dla zbioru $S \subset \mathbb{R}^2$

- . . .
- W przeciwnym wypadku, jeśli d jest nieparzyste, podziel S na dwa zbiory S_1 i S_2 poziomą prostą l przechodzącą przez medianę współrzędnych y punktów z S, gdzie S_1 zawiera punkty poniżej lub na prostej l, a S_2 zawiera punkty powyżej prostej l.
- Rekurencyjnie wyznacz kd-drzewa T_1 dla S_1 oraz T_2 dla S_2 z parametrem $d+1$.
- Zwróć korzeń v (z prostą l), z T_1 jako jego lewym synem, a T_2 – prawym.





Idea konstrukcji kd-drzewa dla zbioru $S \subset \mathbb{R}^2$

- …
- W przeciwnym wypadku, jeśli d jest nieparzyste, podziel S na dwa zbiory S_1 i S_2 poziomą prostą l przechodzącą przez medianę współrzędnych y punktów z S, gdzie S_1 zawiera punkty poniżej lub na prostej l, a S_2 zawiera punkty powyżej prostej l.
- Rekurencyjnie wyznacz kd-drzewa T_1 dla S_1 oraz T_2 dla S_2 z parametrem $d+1$.
- Zwróć korzeń v (z prostą l), z T_1 jako jego lewym synem, a T_2 – prawym.

Złożoność czasowa konstrukcji

- Podział zbioru S nas S_1 i S_2, w szczególności wyznaczenie mediany: $O(n)$.
  - ↳ Rozwiązanie prostsze: wystarczy wstępnie posortować zbiór S po współrzędnej x i po y. Wówczas zbiór wejściowy S jest przekazywany do procedury w postaci dwóch posortowanych list, jednej po współrzędnej x, a drugiej po współrzędnej y.
- Rekurencyjne wyznaczenie kd-drzew dla zbiorów S_1 i S_2: $2 \cdot T(n/2)$.

Otrzymujemy

$$T(n) = \begin{cases} O(1) & \text{jeśli } n = 1, \\ 2 \cdot T(n/2) + O(n) & \text{w przeciwnym wypadku,} \end{cases}$$

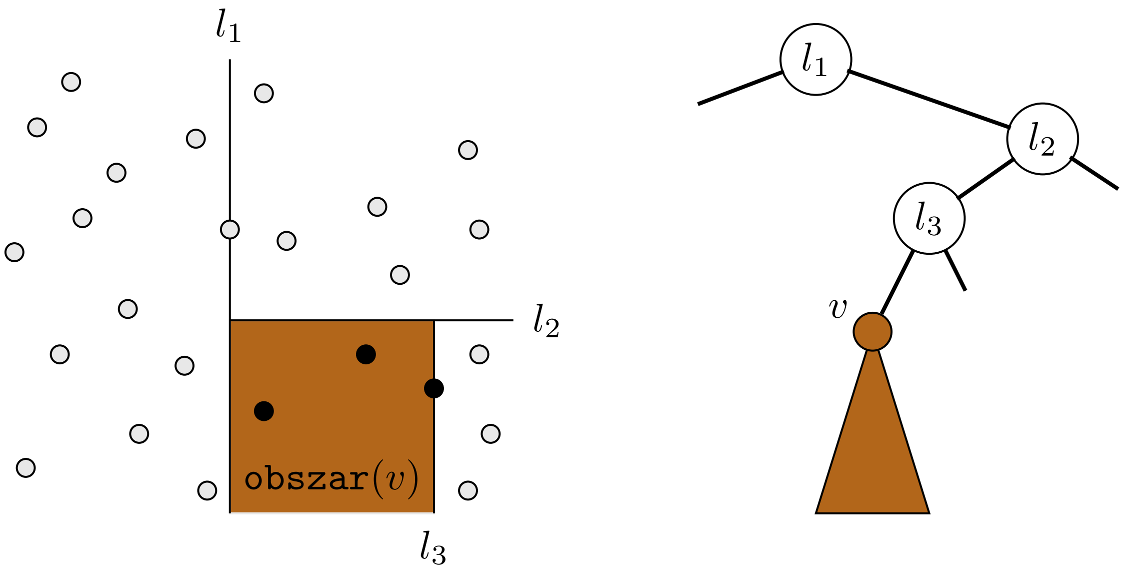
którego rozwiązaniem jest $T(n) = O(n \log n)$.

Lemat 5.3. *Dla zbioru n-punktów na płaszczyźnie można w czasie $O(n \log n)$ skonstruować dwuwymiarowe kd-drzewo, które korzysta z pamięci rzędu $O(n)$.*

↳ Pamięć $O(n)$ – bo drzewo binarne, w którym każdy liść pamięta inny punkt.

Idea algorytmu zapytań

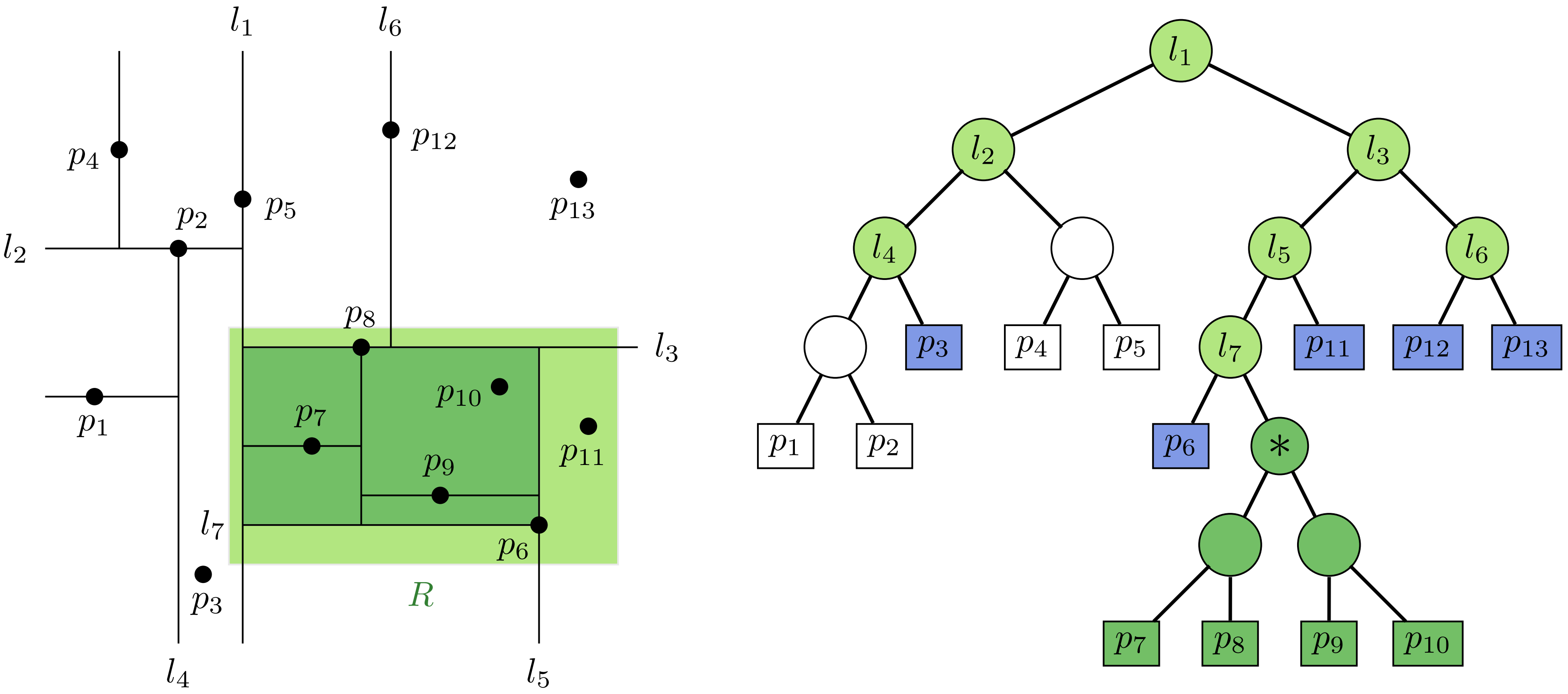
Niech $\texttt{obszar}(v)$ oznacza obszar odpowiadający poddrzewu o korzeniu v.



Odpowiedniość między poddrzewami a obszarami na płaszczyźnie.

- Dla danego obszaru zapytania R musimy przeszukać drzewo zakorzenione w v wtedy i tylko wtedy, gdy R przecina $\texttt{obszar}(v)$, tj. $R \cap \texttt{obszar}(v) \neq \emptyset$.

- A zatem przechodzimy kd-drzewo odwiedzając tylko te węzły v, których obszar `obszar`(v) przecinany jest przez obszar zapytania R, a ponadto:
  - ↳ gdy obszar `obszar`(v) jest całkowicie zawarty w prostokącie zapytania, musimy podać wszystkie punkty pamiętane w tym poddrzewie;
  - ↳ gdy dotrzemy do liścia (w inny niż w/w sposób), musimy sprawdzić, czy punkt pamiętany w tym liściu należy do obszaru R i, jeśli tak, zgłosić go.

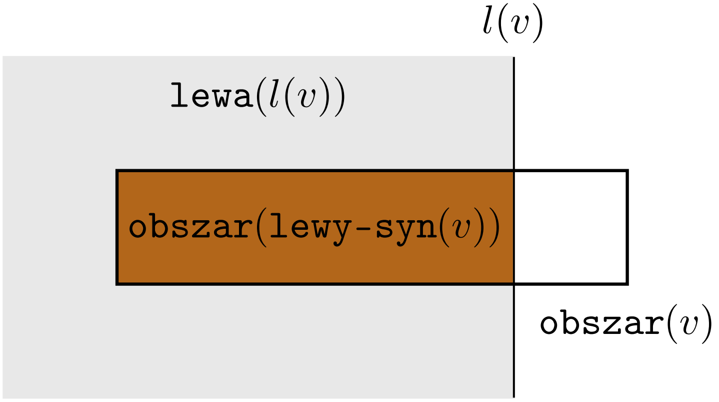


Wszystkie zielone węzły są odwiedzane, natomiast zgłaszane są jedynie liście p_6, p_{11} oraz te ciemnozielone.

/**Uwaga** – Przykładowe drzewo nie jest zrównoważonym kd-drzewem./

- A zatem przechodzimy kd-drzewo odwiedzając tylko te węzły v, których obszar `obszar`(v) przecinany jest przez obszar zapytania R, a ponadto:
  - ↳ gdy obszar `obszar`(v) jest całkowicie zawarty w prostokącie zapytania, musimy podać wszystkie punkty pamiętane w tym poddrzewie;
  - ↳ gdy dotrzemy do liścia (w inny niż w/w sposób), musimy sprawdzić, czy punkt pamiętany w tym liściu należy do obszaru R i, jeśli tak, zgłosić go.

- Zapytanie o przecięcie obszaru R i obszaru odpowiadający pewnemu węzłowi.
  - ↳ Możemy w fazie przetwarzania wstępnego obliczyć dla każdego węzła v jego `obszar`(v) i zapamiętać go.
  - ↳ Można też w trakcie wywołania rekurencyjnego utrzymywać aktualny obszar używając prostych pamiętanych w węzłach wewnętrznych.



Na przykład:

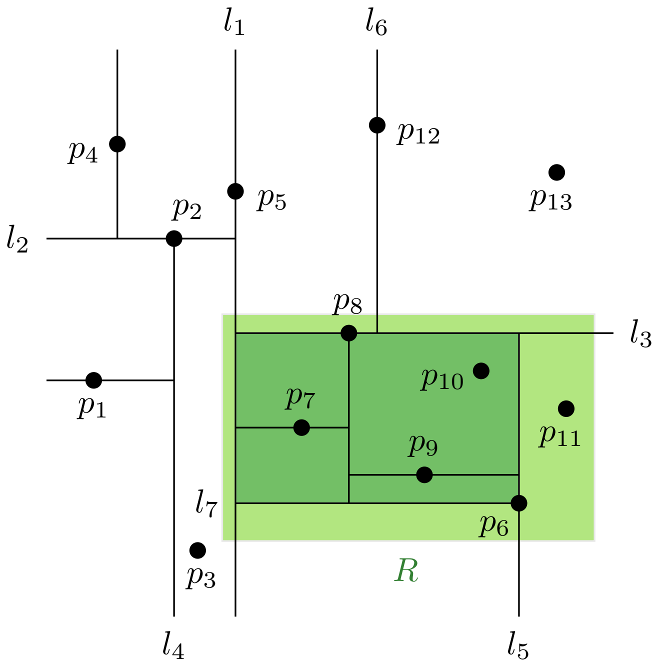
$$\texttt{obszar}(\texttt{lewy-syn}(v)) = \texttt{obszar}(v) \cap \texttt{lewa}(l(v)),$$

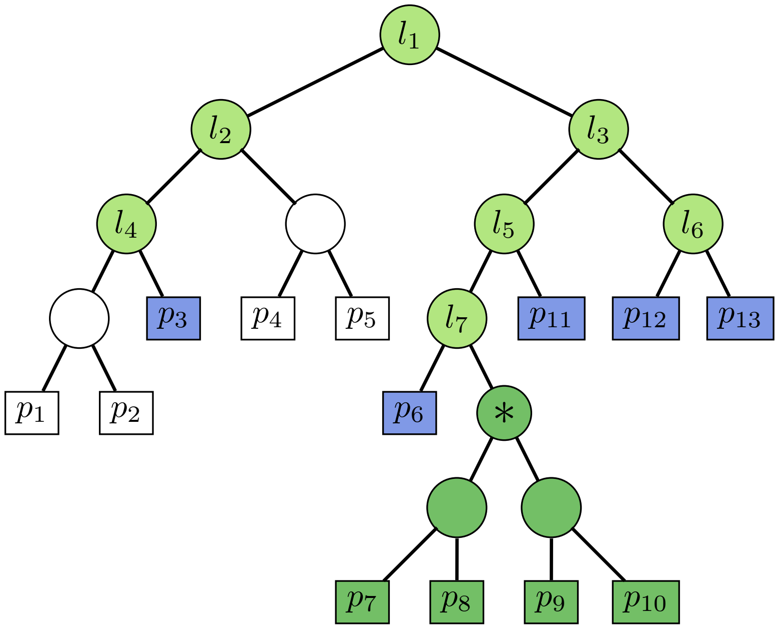
gdzie głębokość węzła v jest parzysta, $l(v)$ jest prostą dzielącą pamiętaną w v, `lewa`$(l(v))$ jest półpłaszczyzną na lewo od $l(v)$, włącznie z $l(v)$.

Lemat 5.4. *Zapytanie o prostokąt o bokach równoległych do osi można wykonać w kd-drzewie przechowujących n punktów należących do płaszczyzny $\mathbb{R}^2$ w czasie $O(\sqrt{n} + k)$, gdzie k jest liczbą podawanych punktów.*

↳ Przypadek 1: `obszar`$(v) \subseteq$ prostokąt zapytania R.

Czas przejścia poddrzewa i podania punktów pamiętanych w jego liściach jest liniowy względem liczby punktów, co wynika z tego, że kd-drzewo jest drzewem binarnym. Zatem całkowity czas potrzebny do przejścia tych poddrzew wynosi $O(k)$, gdzie k jest całkowitą liczbą podawanych punktów.

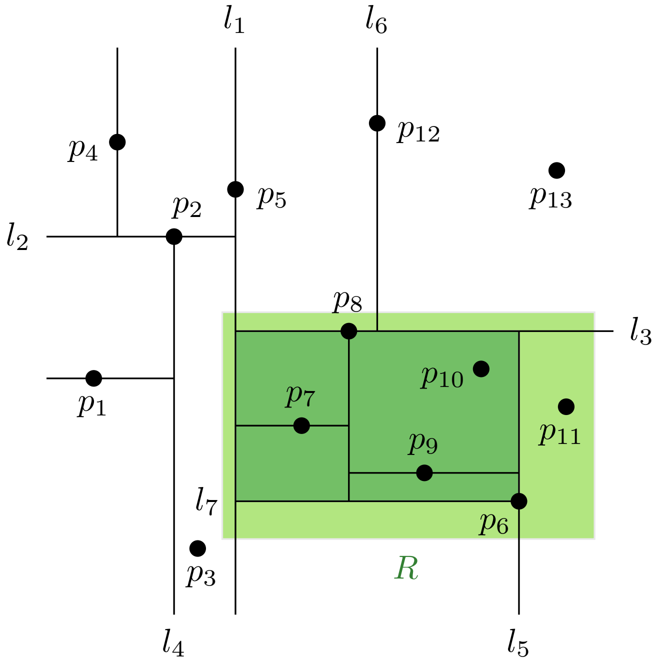


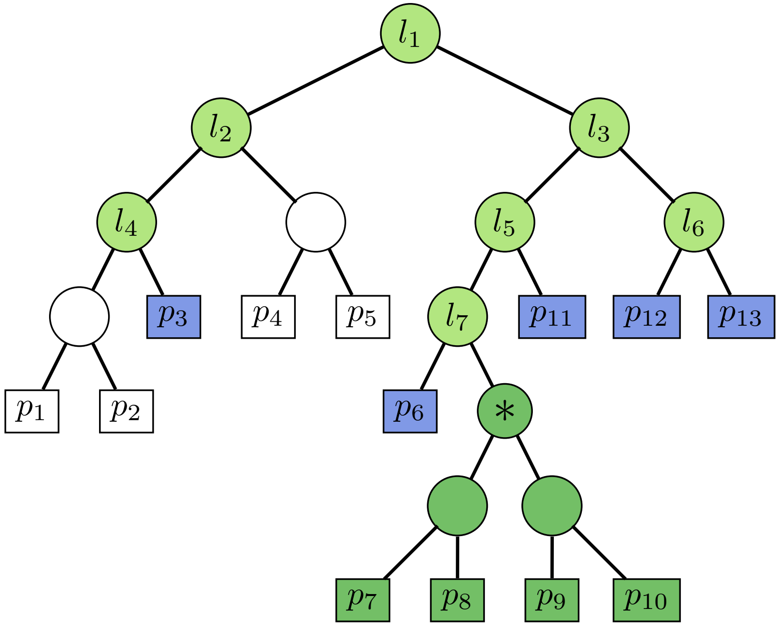


Szacujemy liczbę ciemnozielonych węzłów.

↳ Przypadek 2: `obszar`$(v) \not\subseteq$ prostokąt zapytania R.

Należy oszacować liczbę węzłów $X(n)$, które odwiedzone są przez algorytm zapytań, a które nie są w tych w/w przechodzonych poddrzewach.

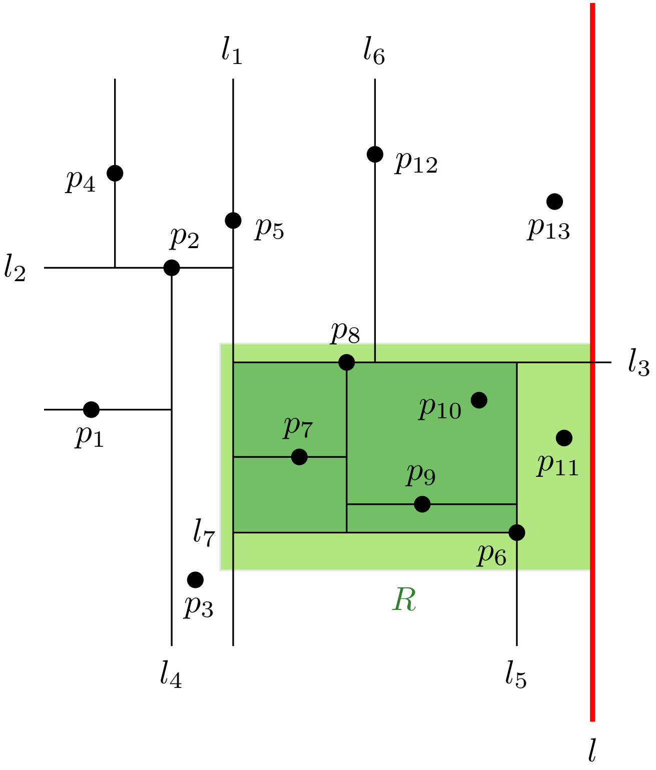


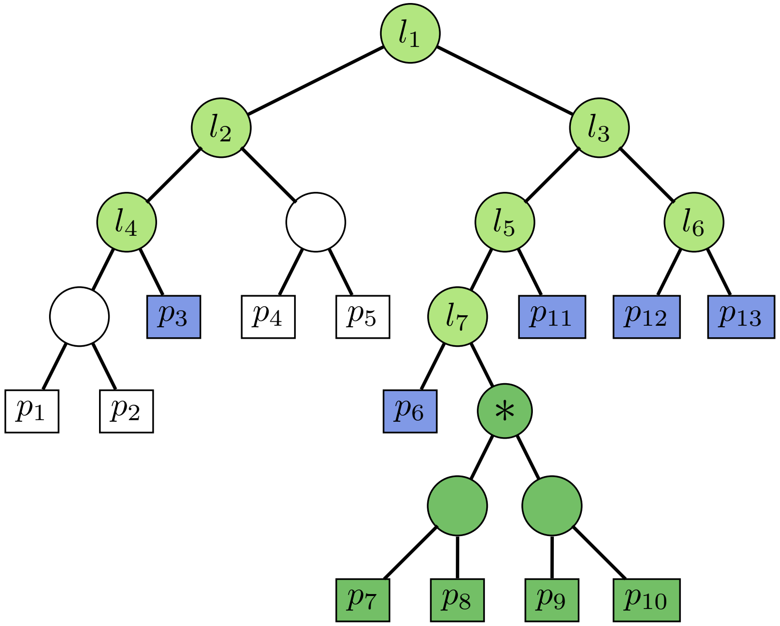


Szacujemy liczbę jasnozielonych lub niebieskich węzłów.

↳ Przypadek 2: `obszar`$(v) \not\subseteq$ prostokąt zapytania R.

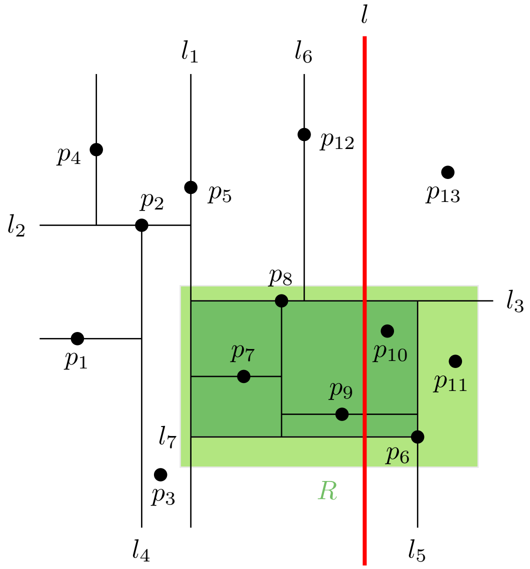
Należy oszacować liczbę węzłów $X(n)$, które odwiedzone są przez algorytm zapytań, a które nie są w tych w/w przechodzonych poddrzewach.

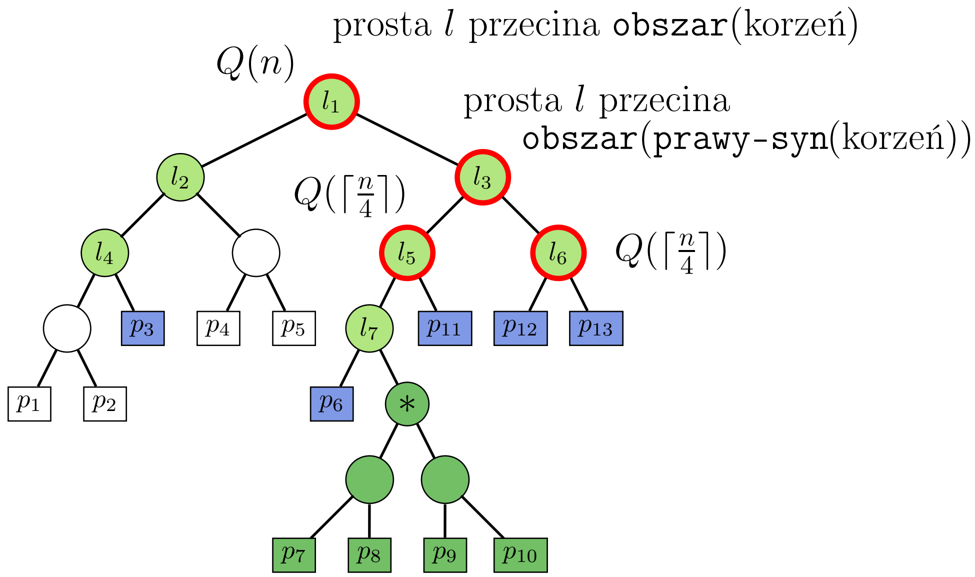




Szacujemy liczbę jasnozielonych lub niebieskich węzłów.

Aby oszacować $X(n)$, rozważamy liczbę obszarów $Q(n)$ przecinanych przez dowolną prostą pionową l. W ten sposób otrzymamy górne ograniczenie na liczbę obszarów przecinanych przez lewą i prawą krawędź obszaru zapytania R.

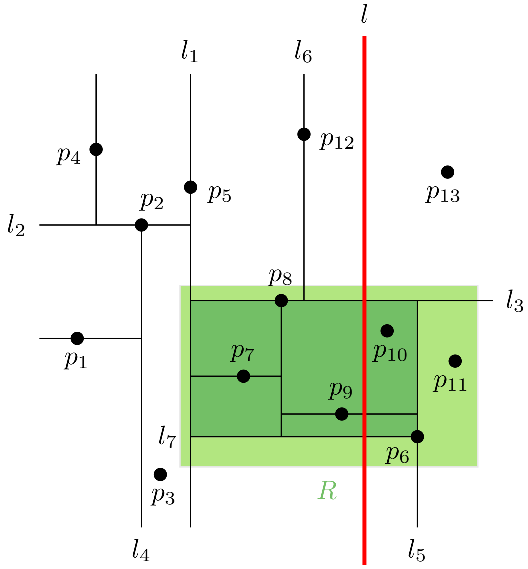


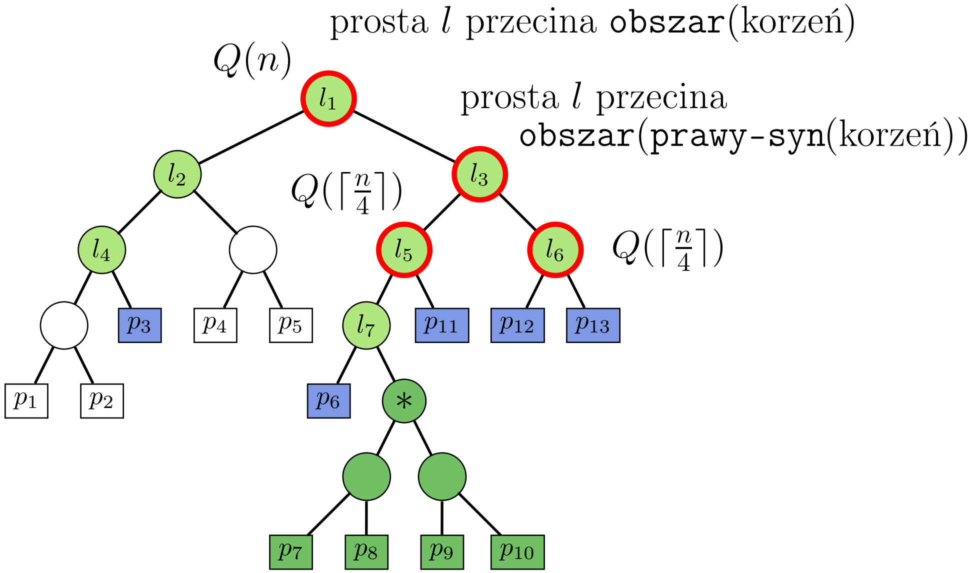


Niech l będzie dowolną pionową prostą i niech $Q(n)$ będzie liczbą przecinanych obszarów przez l w kd-drzewie przechowującym n-punktów na płaszczyźnie, którego korzeń zawiera pionową prostą dzielącą.

$$Q(n) = \begin{cases} O(1) & \text{jeśli } n = 1; \\ 2 + 2 \cdot Q(\lceil n/4 \rceil) & \text{w przeciwnym wypadku.} \end{cases}$$

Rozwiązaniem tego równania jest $Q(n) = O(\sqrt{n})$.





Niech l będzie dowolną pionową prostą i niech $Q(n)$ będzie liczbą przecinanych obszarów przez l w kd-drzewie przechowującym n-punktów na płaszczyźnie, którego korzeń zawiera pionową prostą dzielącą.

$$Q(n) = \begin{cases} O(1) & \text{jeśli } n = 1; \\ 2 + 2 \cdot Q(\lceil n/4 \rceil) & \text{w przeciwnym wypadku.} \end{cases}$$

Rozwiązaniem tego równania jest $Q(n) = O(\sqrt{n})$.

- ▷ W analogiczny sposób można udowodnić, że liczba obszarów przecinanych przez dowolną prostą poziomą jest także rzędu $O(\sqrt{n})$.

A zatem całkowita liczba obszarów przecinanych przez brzeg prostokąta zapytania jest również rzędu $O(\sqrt{n})$.

Twierdzenie 5.5. (Bentley 1975) *Kd-drzewo dla n-elementowego zbioru* $S \subset \mathbb{R}^2$ *punktów wymaga* $O(n)$ *pamięci i można je zbudować w czasie* $O(n \log n)$*. Zapytanie o prostokątny obszar zapytania dla kd-drzewa wymaga czasu rzędu* $O(\sqrt{n}+k)$*, gdzie* k *jest liczbą zgłaszanych punktów.*

Twierdzenie 5.5. (Bentley 1975) *Kd-drzewo dla n-elementowego zbioru* $S \subset \mathbb{R}^2$ *punktów wymaga* $O(n)$ *pamięci i można je zbudować w czasie* $O(n \log n)$*. Zapytanie o prostokątny obszar zapytania dla kd-drzewa wymaga czasu rzędu* $O(\sqrt{n}+k)$*, gdzie* k *jest liczbą zgłaszanych punktów.*

Wielowymiarowe kd-drzewa

- W korzeniu zbiór punktów dzielony jest względem pierwszej współrzędnej tych punktów. W dzieciach korzenia podział ten dokonywany jest względem drugiej współrzędnej. W węzłach o głębokości dwa – względem trzeciej współrzędnej.
- . . . aż do głębokości $d-1$, na której dzielimy względem ostatniej współrzędnej.
- Na głębokości d – podział znowu względem pierwszej współrzędnej, itd.
- Rekursja zatrzymuje się, gdy pozostał tylko jeden punkt, który zapamiętywany jest w liściu.
- Ponieważ d-wymiarowe kd-drzewo dla zbioru n-punktów jest drzewem binarnym o n liściach, więc korzysta ono z pamięci rzędu $O(n)$.
- Czas konstrukcji wynosi $O(n \log n)$. (Zakładamy, że d jest stałą.)
- Można pokazać, że czas zapytania jest rzędu $O(n^{1-1/d} + k)$.

5.3 DWUWYMIAROWE DRZEWA OBSZARÓW

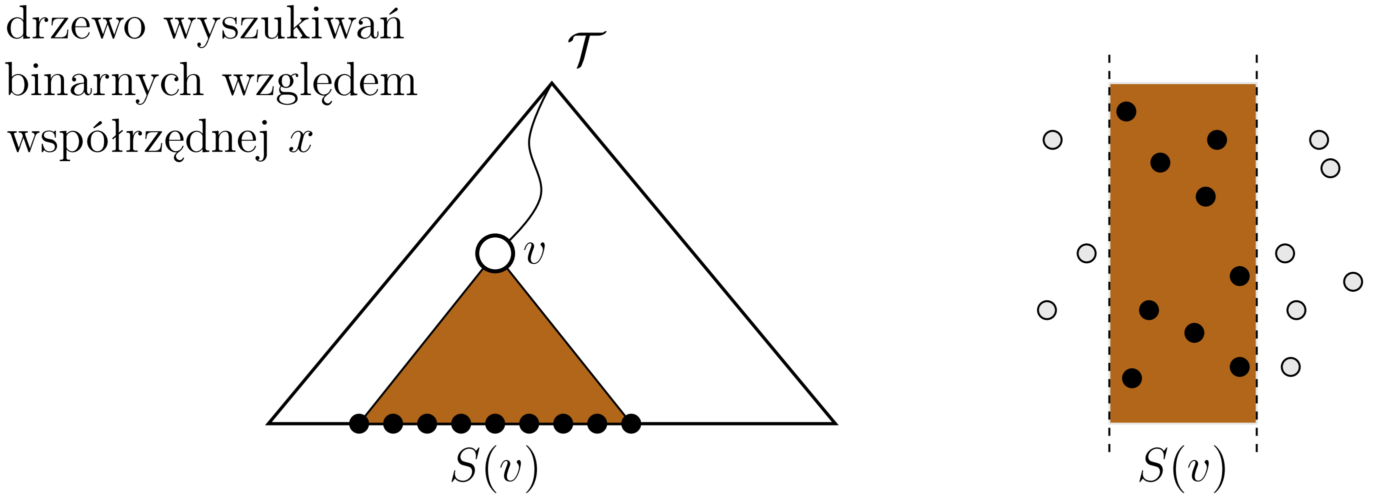
Problem. (Przeszukiwanie obszarów prostokątnych)

Niech S będzie danym zbiorem punktów na płaszczyźnie rzeczywistej $\mathbb{R}^2$.

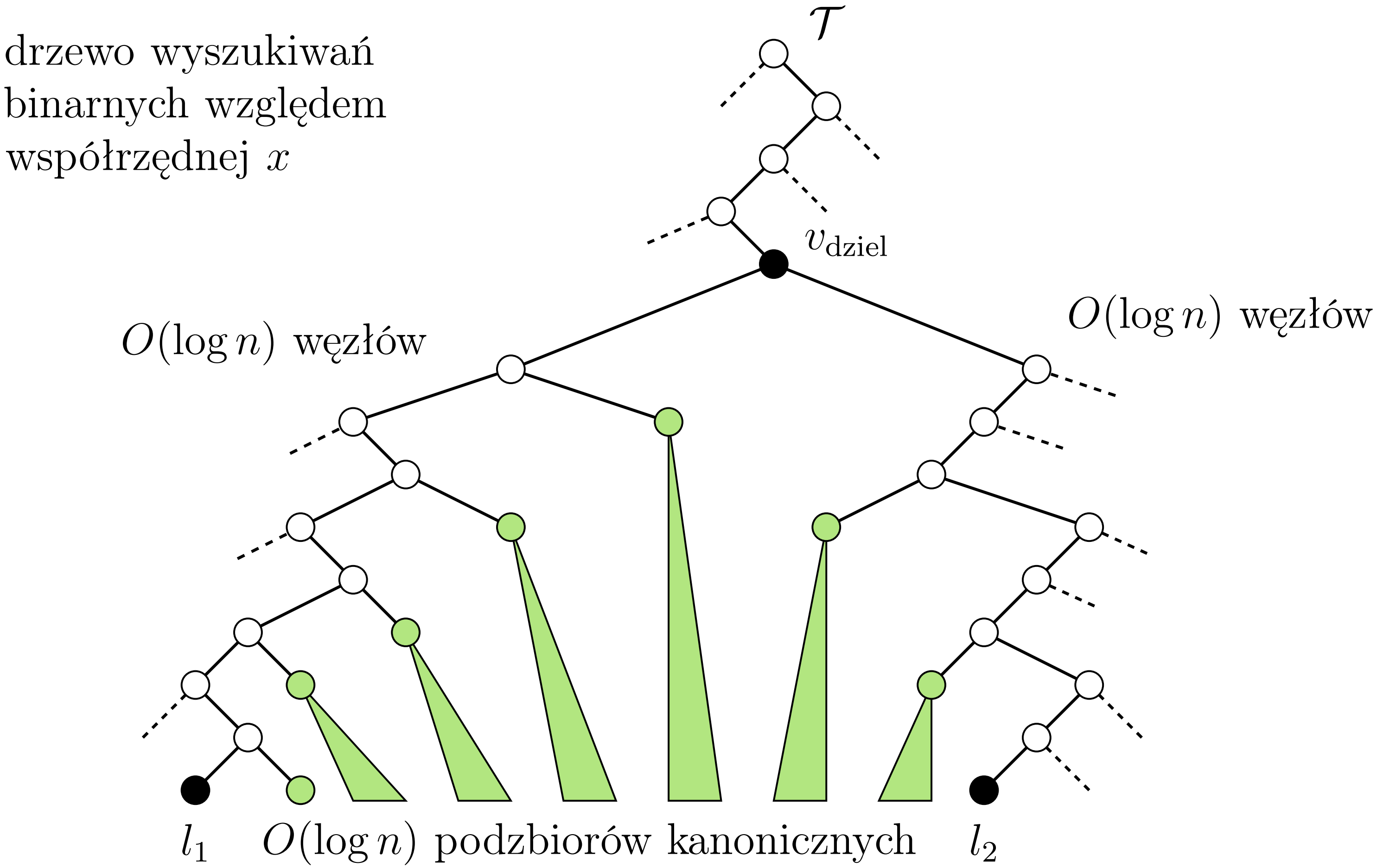
Wejście: *Obszar zapytania $R = [x_1, x_2] \times [y_1, y_2]$.*

Wyjście: *Wszystkie punkty z S, które należą do R.*

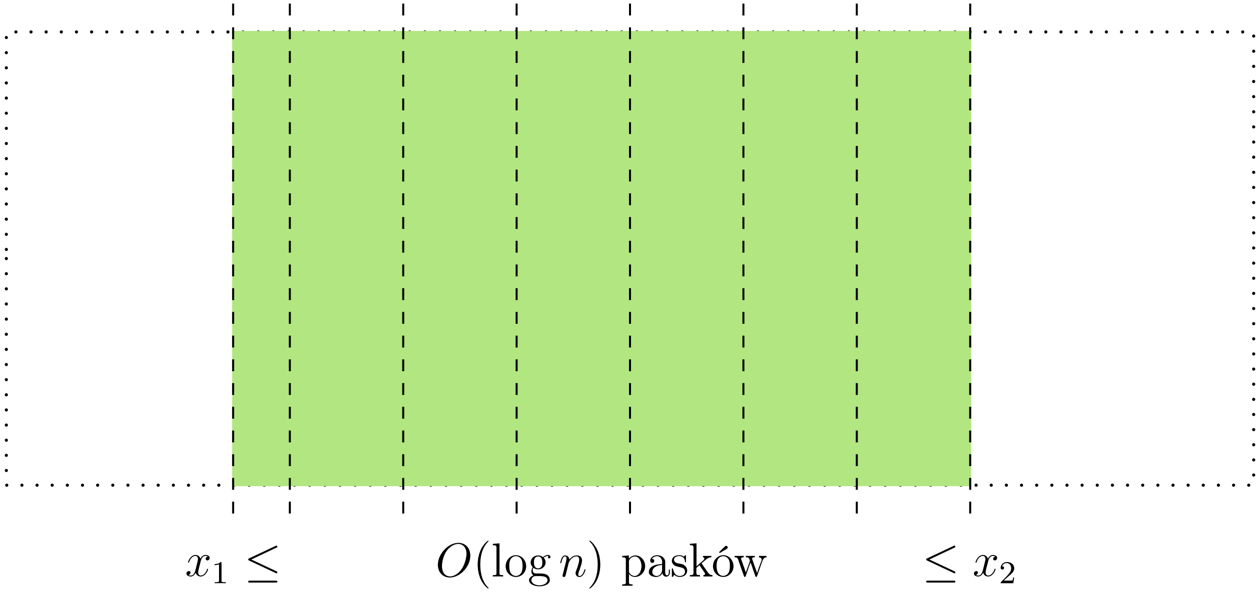
Rozważmy zrównoważone drzewo $\mathcal{T}$ wyszukiwań binarnych przechowujące w liściach wszystkie punkty z S, o porządku wyznaczonym przez odcięte punktów.



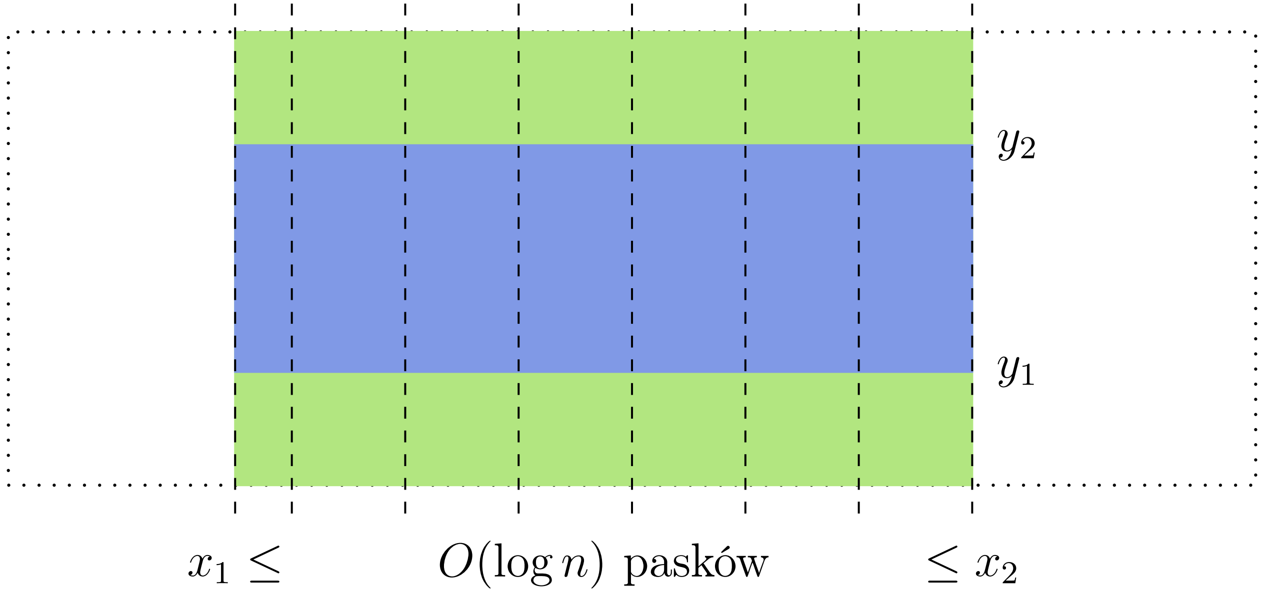
Podzbiór $S(v)$ punktów pamiętany w liściach poddrzewa zakorzenionego w danym wierzcołku v drzewa $\mathcal{T}$ nazywany jest *podzbiór kanonicznym* wierzchołka v.



- Podzbiór punktów, których współrzędna x leży w jednowymiarowym obszarze zapytania $[x_1, x_2]$ można wyrazić jako sumę $O(\log n)$ rozłącznych podzbiorów kanonicznych (pasków) w drzewie $\mathcal{T}$. Są to zbiory $S(v)$ wezłów v, które są korzeniami odpowiednich poddrzew na ścieżkach poszukiwań x_1 oraz x_2.



- Podzbiór punktów, których współrzędna x leży w jednowymiarowym obszarze zapytania $[x_1, x_2]$ można wyrazić jako sumę $O(\log n)$ rozłącznych podzbiorów kanonicznych (pasków) w drzewie $\mathcal{T}$. Są to zbiory $S(v)$ wezłów v, które są korzeniami odpowiednich poddrzew na ścieżkach poszukiwań x_1 oraz x_2.

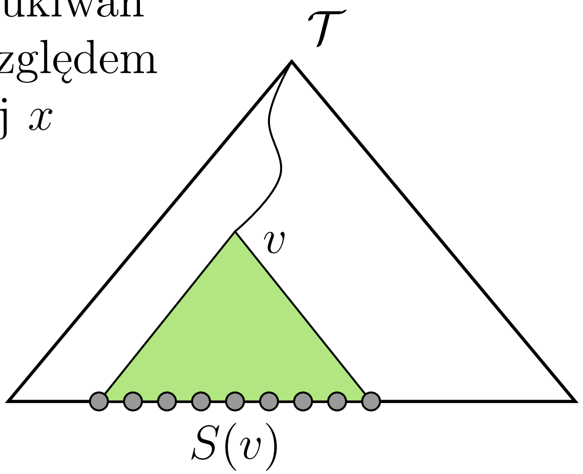


- Jeśli dla dowolnego $S(v_i)$ dostępne jest drzewo wyszukiwań binarnych względem współrzędnej y, wówczas, wykonując jednowymiarowe zapytanie na tym drzewie, jesteśmy w stanie znaleźć w czasie $O(\log |S(v_i)| + k_{v_i})$ wszystkie k_{v_i} punktów z $S(v_i)$, których współrzędna y mieści się w przedziale $[y_1, y_2]$.

Dwupoziomowa struktura danych zwana **drzewem obszarów**.

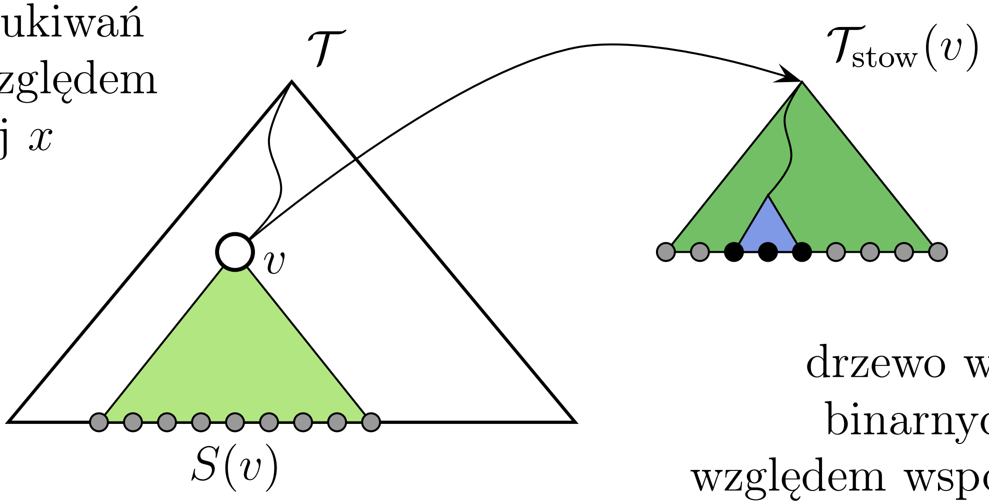
- Główne drzewo jest zrównoważonym drzewem przeszukiwan binarnych $\mathcal{T}$ zbudowanym względem współrzędnej x punktów z S.

drzewo wyszukiwań
binarnych względem
współrzędnej x



Dwupoziomowa struktura danych zwana **drzewem obszarów**.

- Główne drzewo jest zrównoważonym drzewem przeszukiwan binarnych $\mathcal{T}$ zbudowanym względem współrzędnej x punktów z S.
- Dla każdego węzła wewnętrznego lub liścia v w drzewie $\mathcal{T}$, podzbiór kanoniczny $S(v)$ jest pamiętany w zrównoważonym drzewie przeszukiwań binarnych $\mathcal{T}_{\text{stow}}(v)$ względem współrzędnej y punktów (tzw. *struktura stowarzyszona z v*); węzeł v pamięta wskaźnik do drzewa $\mathcal{T}_{\text{stow}}(v)$.

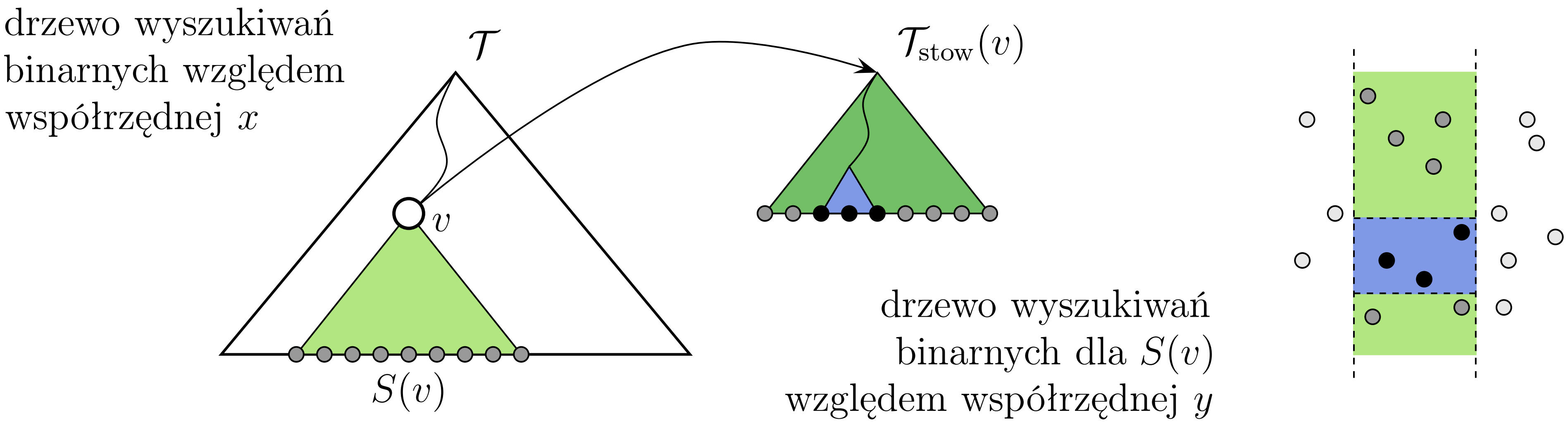


drzewo wyszukiwań
binarnych względem
współrzędnej x

drzewo wyszukiwań
binarnych dla $S(v)$
względem współrzędnej y

Dwupoziomowa struktura danych zwana **drzewem obszarów**.

- Główne drzewo jest zrównoważonym drzewem przeszukiwan binarnych $\mathcal{T}$ zbudowanym względem współrzędnej x punktów z S.
- Dla każdego węzła wewnętrznego lub liścia v w drzewie $\mathcal{T}$, podzbiór kanoniczny $S(v)$ jest pamiętany w zrównoważonym drzewie przeszukiwań binarnych $\mathcal{T}_{\text{stow}}(v)$ względem współrzędnej y punktów (tzw. *struktura stowarzyszona z v*); węzeł v pamięta wskaźnik do drzewa $\mathcal{T}_{\text{stow}}(v)$.


drzewo wyszukiwań binarnych względem współrzędnej x

drzewo wyszukiwań binarnych dla $S(v)$ względem współrzędnej y

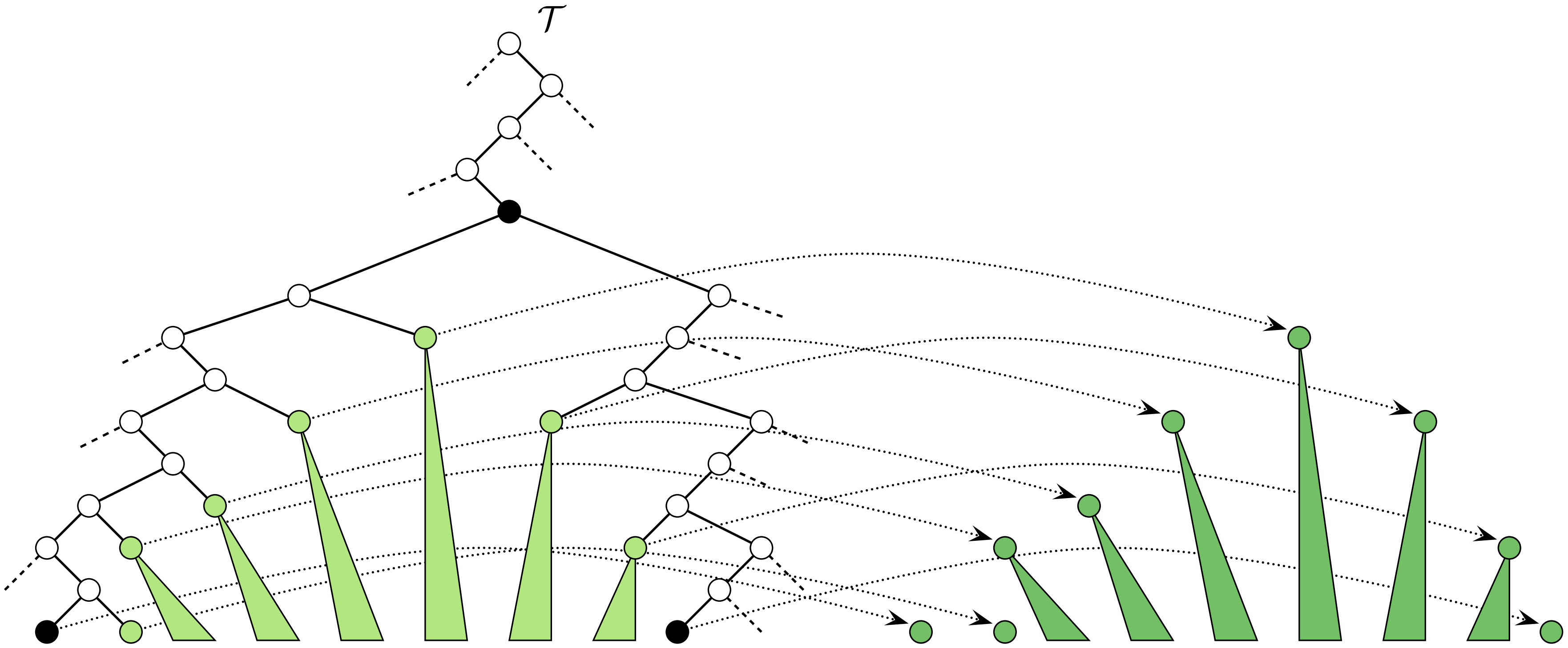
- Na dowolne poddrzewo (zbiór kanoniczny) drzewa $\mathcal{T}_{\text{stow}}(v)$ można patrzeć jak poziomy wycinek pionowego paska $S(v)$.

Idea algorytmu zapytań

- Algorytm wybiera $O(\log n)$ struktur stowarzyszonych łącznie zawierających wszystkie punkty, których współrzędna x leży w obszarze $[x_1, x_2]$.

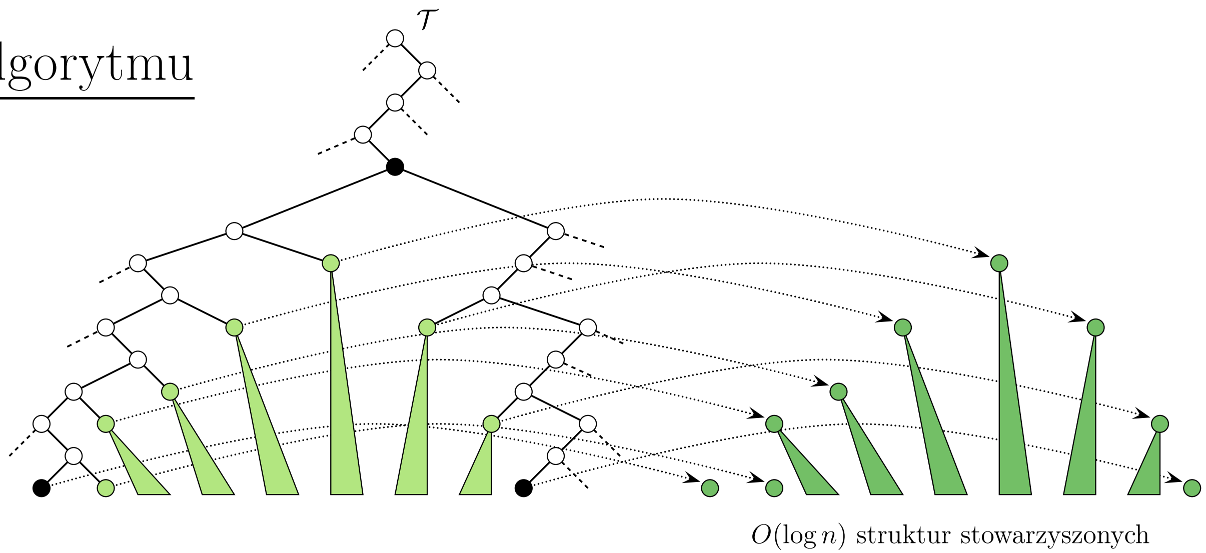
  /Jednowymiarowy algorytm zapytań na drzewie $\mathcal{T}$./

- Z tych podzbiorów zgłaszane są tylko te punkty, których współrzędna y leży w obszarze $[y_1, y_2]$. /Jednowymiarowy algorytm zapytań na drzewach $\mathcal{T}_{\text{stow}}$./



$O(\log n)$ struktur stowarzyszonych

Złożoność czasowa algorytmu



W każdym odwiedzanym węźle v drzewa $\mathcal{T}$ sprawdzamy, dokąd należy dalej iść, oraz być może wywołujemy (jednowymiarowy) algorytm zapytań na drzewie $\mathcal{T}_{\text{stow}}(v)$ o czasię rzędu $O(\log n + k_v)$, gdzie k_v jest liczbą zgłaszanych punktów w wywołaniu dla węzła v. Tym samym całkowity czas zużyty we wszystkich odwiedzonych węzłach wynosi $\sum_v O(\log n + k_v)$.

↳ $\sum_v k_v = k$, gdzie k jest liczbą wszystkich podanych punktów.

↳ Ścieżki poszukiwań x_1 i x_2 w drzewie $\mathcal{T}$ mają długość rzędu $O(\log n)$.

Otrzymujemy zatem $\sum_v O(\log n) = O(\log^2 n)$.

Wniosek 5.6. *W dwuwymiarowym drzewie obszarów przechowującym n-punktów płaszczyzny $\mathbb{R}^2$ zapytanie o prostokąt $[x_1, x_2] \times [y_1, y_2]$ (o bokach równoległych do osi) wymaga czasu $O(\log^2 n + k)$, gdzie k jest liczbą zgłaszanych punktów.*

Idea algorytmu budowy dwuwymiarowego drzewa obszarów

Wstępne przetwarzanie: zbiór wejściowy S reprezentowany jest przez parę (S_x, S_y).

S_x – posortowany zbiór S punktów wejściowych względem współrzędnej x.
S_y – posortowany zbiór S punktów wejściowych względem współrzędnej y.

- ▶ Zbuduj drzewo wyszukiwań binarnych $\mathcal{T}_{\text{stow}}$ dla zbioru S_y rzędnych punktów z S. W liściach $\mathcal{T}_{\text{stow}}$ pamiętaj nie tylko współrzędne y punktów z S_y, ale i odpowiadające im punkty.
- ▶ Jeśli wejściowy zbiór S zawiera tylko jeden punkt, to stwórz liść pamiętający ten punkt i zwiąż z nim $\mathcal{T}_{\text{stow}}$.
- ▶ W przeciwnym wypadku podziel zbiór S na dwa (prawie) równoliczne podzbiory S_1 i S_2, gdzie podzbiór S_1 zawiera punkty o odciętych mniejszych lub równych $x_{\text{środ}}$, medianie współrzędnych x, a podzbiór S_2 zawiera punkty o odciętych większych od $x_{\text{środ}}$.
- ▶ Wywołaj rekurencyjnie budowę drzewa dla zbiorów S_1 oraz S_2, otrzymując odpowiednio poddrzewa T_1 i T_2, o korzeniach v_1 i v_2.
- ▶ Zwróć węzeł/korzeń v pamiętający $x_{\text{środ}}$, o lewym synu v_1, prawym – v_2, związawszy uprzednio $\mathcal{T}_{\text{stow}}$ z v.

Idea algorytmu budowy dwuwymiarowego drzewa obszarów

Wstępne przetwarzanie: zbiór wejściowy S reprezentowany jest przez parę (S_x, S_y).

S_x – posortowany zbiór S punktów wejściowych względem współrzędnej x.
S_y – posortowany zbiór S punktów wejściowych względem współrzędnej y.

- ▶ Zbuduj drzewo wyszukiwań binarnych $\mathcal{T}_{\text{stow}}$ dla zbioru S_y rzędnych punktów z S. W liściach $\mathcal{T}_{\text{stow}}$ pamiętaj nie tylko współrzędne y punktów z S_y, ale i odpowiadające im punkty.
  - ↳ Zbiór S_y jest posortowany, a zatem „łącząc w pary”, od dołu, można skonstruować drzewo $\mathcal{T}_{\text{stow}}$ w czasie liniowym od rozmiaru S.
- ▶ …

Idea algorytmu budowy dwuwymiarowego drzewa obszarów

Wstępne przetwarzanie: zbiór wejściowy S reprezentowany jest przez parę (S_x, S_y).

S_x – posortowany zbiór S punktów wejściowych względem współrzędnej x.
S_y – posortowany zbiór S punktów wejściowych względem współrzędnej y.

- ▶ Zbuduj drzewo wyszukiwań binarnych $\mathcal{T}_{\text{stow}}$ dla zbioru S_y rzędnych punktów z S. W liściach $\mathcal{T}_{\text{stow}}$ pamiętaj nie tylko współrzędne y punktów z S_y, ale i odpowiadające im punkty.
  - ↳ Zbiór S_y jest posortowany, a zatem „łącząc w pary”, od dołu, można skonstruować drzewo $\mathcal{T}_{\text{stow}}$ w czasie liniowym od rozmiaru S.
- ▶ ...

Otrzymujemy następujące równanie rekurencyjne

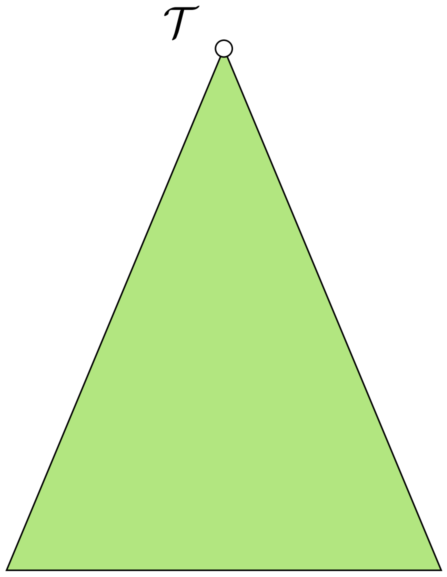
$$T(n) = \begin{cases} O(1) & \text{jeśli } n = 1; \\ 2 \cdot T(n/2) + O(n) & \text{w przeciwnym wypadku,} \end{cases}$$

którego rozwiązaniem jest $T(n) = O(n \log n)$. A zatem czas konstrukcji drzewa, mając na uwadze wykonane wcześniej przetwarzanie wstępne, wynosi $O(n \log n)$.

Złożoność pamięciowa algorytmu

Lemat 5.7. *Drzewo obszarów dla zbioru n punktów na płaszczyźnie wymaga pamięci rzędu $O(n \log n)$.*

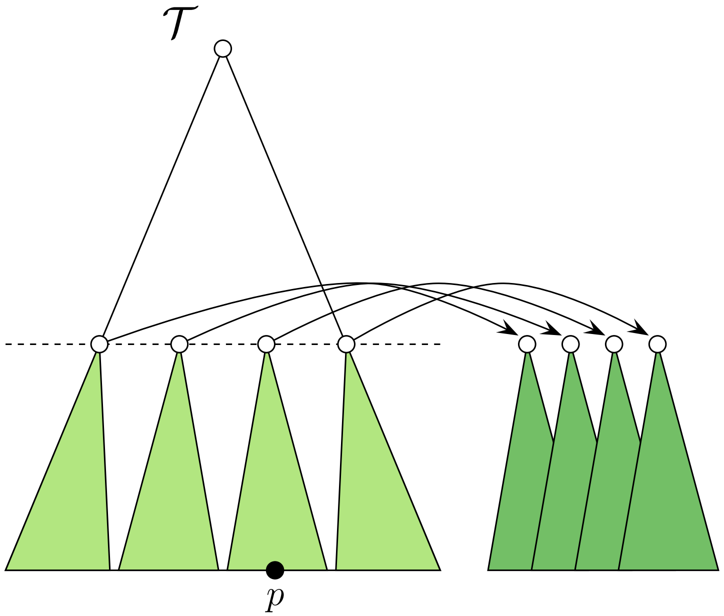
- Drzewo $\mathcal{T}$ potrzebuje pamięci rzędu $O(n)$, gdyż mamy n punktów.



Złożoność pamięciowa algorytmu

Lemat 5.7. *Drzewo obszarów dla zbioru n punktów na płaszczyźnie wymaga pamięci rzędu $O(n \log n)$.*

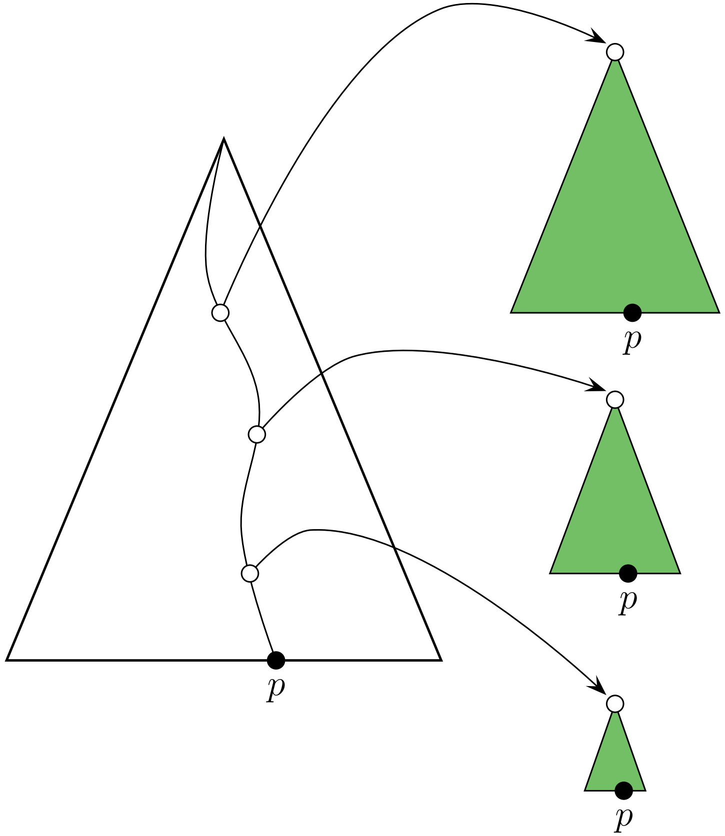
- Drzewo $\mathcal{T}$ potrzebuje pamięci rzędu $O(n)$, gdyż mamy n punktów.
- Struktury stowarzyszone:
  - ↳ dla wszystkich węzłów na tej samej głębokości drzewa $\mathcal{T}$ punkt p pamiętany jest dokładnie w jednej strukturze stowarzyszonej;
  - ↳ struktury stowarzyszone na danej głębokości drzewa $\mathcal{T}$ pokrywają wszystkie punkty, stąd wykorzystują one łącznie $O(n)$ pamięci (na danej głębokości);



Złożoność pamięciowa algorytmu

Lemat 5.7. *Drzewo obszarów dla zbioru n punktów na płaszczyźnie wymaga pamięci rzędu $O(n \log n)$.*

- Drzewo $\mathcal{T}$ potrzebuje pamięci rzędu $O(n)$, gdyż mamy n punktów.
- Struktury stowarzyszone:
  - ↳ dla wszystkich węzłów na tej samej głębokości drzewa $\mathcal{T}$ punkt p pamiętany jest dokładnie w jednej strukturze stowarzyszonej;
  - ↳ struktury stowarzyszone na danej głębokości drzewa $\mathcal{T}$ pokrywają wszystkie punkty, stąd wykorzystują one łącznie $O(n)$ pamięci (na danej głębokości);
  - ↳ głębokość drzewa $\mathcal{T}$ wynosi $O(\log n)$, a zatem całkowity rozmiar pamięci wymaganej dla struktur stowarzyszonych ograniczony jest przez $O(n \log n)$. □



Złożoność pamięciowa algorytmu

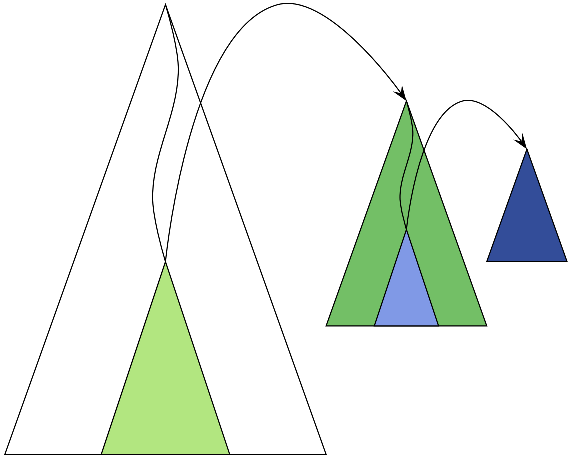
Lemat 5.7. *Drzewo obszarów dla zbioru n punktów na płaszczyźnie wymaga pamięci rzędu $O(n \log n)$.*

- Drzewo $\mathcal{T}$ potrzebuje pamięci rzędu $O(n)$, gdyż mamy n punktów.
- Struktury stowarzyszone: $O(n \log n)$.

Otrzymujemy w konsekwencji:

Twierdzenie 5.8. (m.in. Bentley 1979; Lueker 1978)
Niech S będzie zbiorem n punktów na płaszczyźnie $\mathbb{R}^2$. Drzewo obszarów dla S używa $O(n \log n)$ pamięci i można je zbudować w czasie $O(n \log n)$. Punkty S leżące w prostokątnym obszarze zapytania można wyznaczyć w czasie $O(\log^2 n + k)$, gdzie k jest liczbą zgłaszanych punktów.

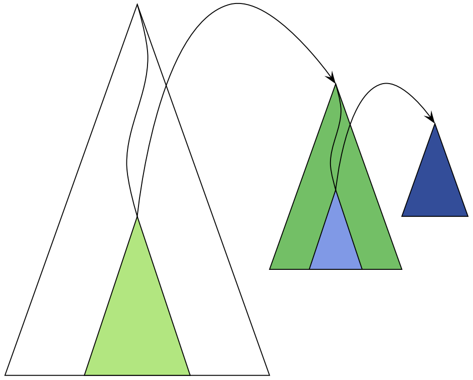
3.4 WIELOWYMIAROWE DRZEWA OBSZARÓW



- Tworzymy zrównoważone drzewo wyszukiwań binarnych względem pierwszej współrzędnej danych punktów.
- W drzewie tym kanoniczny podzbiór $S(v)$ węzła v składa się z punktów pamiętanych w liściach drzewa zakorzenionego w v.
- Dla każdego węzła v tworzymy strukturę stowarzyszoną $\mathcal{T}_{\text{stow}}$ będącą $(d-1)$-wymiarowym drzewem obszarów dla punktów z $S(v)$, ograniczonych do ich $d-1$ współrzędnych.
- $\mathcal{T}_{\text{stow}}$ budowane jest rekurencyjnie; rekursja zatrzymuje się, gdy pozostaniemy z punktami ograniczonymi do ich ostatniej współrzędnej.

Idea algorytmu zapytań

- Używamy drzew pierwszego poziomu do wyznaczenia $O(\log n)$ węzłów, których kanoniczne podzbiory łącznie zawierają wszystkie punkty, których współrzędne są we właściwym przedziale.
- W stosunku do kanonicznych podzbiorów zadawane są dalej zapytania o odpowiedni obszar w odpowiadajacych im strukturach drugiego poziomu;
- W każdej takiej strukturze wybranych zostaje $O(\log n)$ kanonicznych podzbiorów. Tym samym, mamy w sumie $O(\log^2 n)$ kanonicznych podzbiorów w strukturach drugiego poziomu, które łacznie zawierają wszystkie punkty, których pierwsze i drugie współrzędne leżą we właściwym obszarze.
- Następnie struktury trzeciego poziomu przechowujące te kanoniczne podzbiory są pytane o obszar dla trzeciej współrzędnej, itd., aż dotrzemy do drzew jednowymiarowych.



Idea algorytmu zapytań

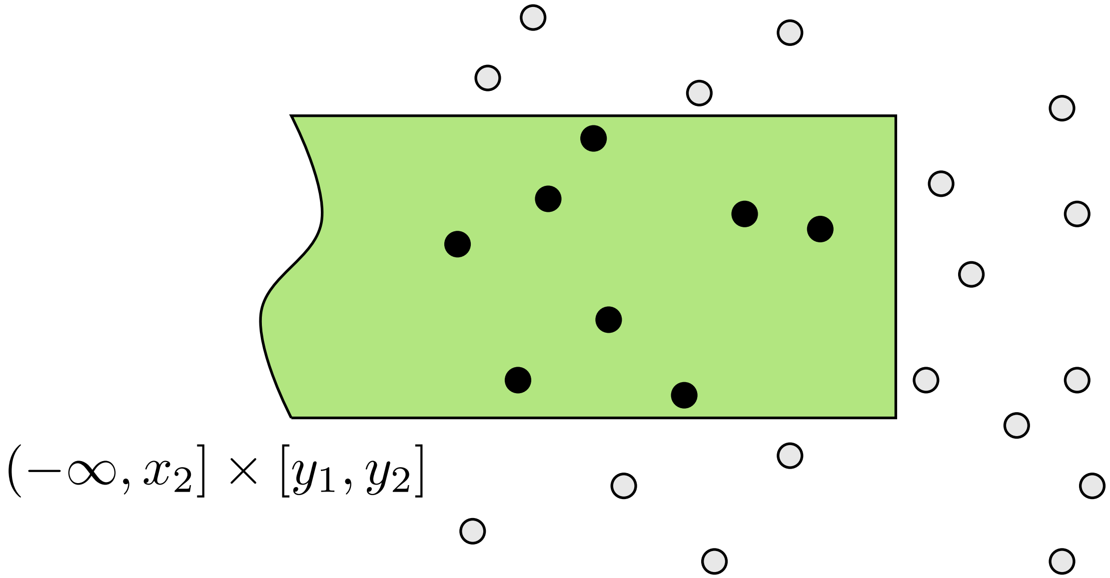
- Używamy drzew pierwszego poziomu do wyznaczenia $O(\log n)$ węzłów, których kanoniczne podzbiory łącznie zawierają wszystkie punkty, których współrzędne są we właściwym przedziale.
- W stosunku do kanonicznych podzbiorów zadawane są dalej zapytania o odpowiedni obszar w odpowiadajacych im strukturach drugiego poziomu;
- W każdej takiej strukturze wybranych zostaje $O(\log n)$ kanonicznych podzbiorów. Tym samym, mamy w sumie $O(\log^2 n)$ kanonicznych podzbiorów w strukturach drugiego poziomu, które łacznie zawierają wszystkie punkty, których pierwsze i drugie współrzędne leżą we właściwym obszarze.
- Następnie struktury trzeciego poziomu przechowujące te kanoniczne podzbiory są pytane o obszar dla trzeciej współrzędnej, itd., aż dotrzemy do drzew jednowymiarowych.

Twierdzenie 5.9. (m.in. Bentley, 1979; Lueker, 1978)
Niech S będzie zbiorem n punktów w d-wymiarowej przestrzeni $\mathbb{R}^d$, $d \geq 2$. Wówczas drzewo obszarów dla S używa $O(n \log^{d-1} n)$ pamięci i może być zbudowane w czasie $O(n \log^{d-1} n)$. Punkty S leżące w prostokątnym obszarze zapytania można wyznaczyć w czasie $O(\log^d n + k)$, gdzie k jest liczbą zgłaszanych punktów.

- *Kaskadowanie cząstkowe*: czas zapytania rzędu $O(\log^{d-1} n + k)$.

5.5 DRZEWA PRZESZUKIWAŃ PRIORYTETOWYCH

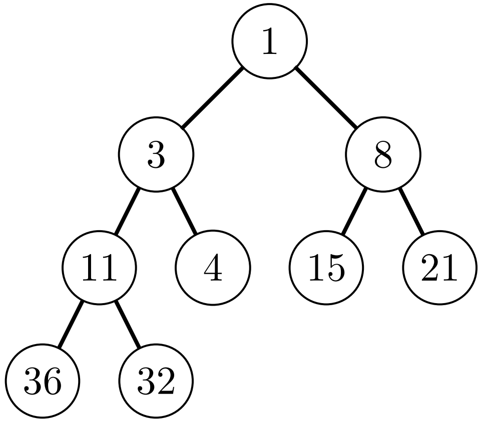
Zastosowanie przy zapytaniach o dwuwymiarowy obszar ortogonalny R, z którego jeden brzeg jest nieograniczony, tj. gdy $R = (-\infty, x_2) \times [y_1, y_2]$.



5.5 DRZEWA PRZESZUKIWAŃ PRIORYTETOWYCH

Zastosowanie przy zapytaniach o dwuwymiarowy obszar ortogonalny R, z którego jeden brzeg jest nieograniczony, tj. gdy $R = (-\infty, x_2) \times [y_1, y_2]$.

Kopiec jednowymiarowy

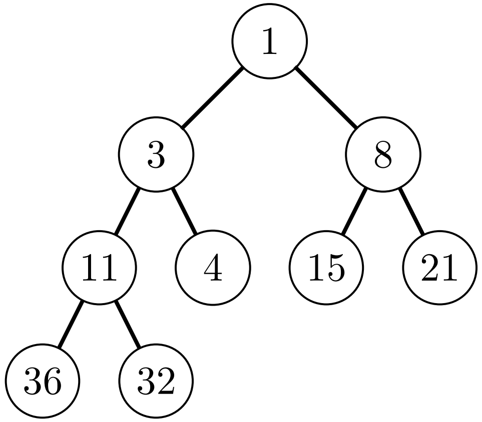


- ▶ Kopiec jest drzewem binarnym, w którym korzeń pamięta punkt ze zbioru o minimalnym kluczu (wartości).
- ▶ Reszta zbioru jest podzielona na dwa podzbiory o prawie równym rozmiarze i te podzbiory pamiętywane są rekurencyjnie w ten sam sposób.

5.5 DRZEWA PRZESZUKIWAŃ PRIORYTETOWYCH

Zastosowanie przy zapytaniach o dwuwymiarowy obszar ortogonalny R, z którego jeden brzeg jest nieograniczony, tj. gdy $R = (-\infty, x_2) \times [y_1, y_2]$.

Kopiec jednowymiarowy

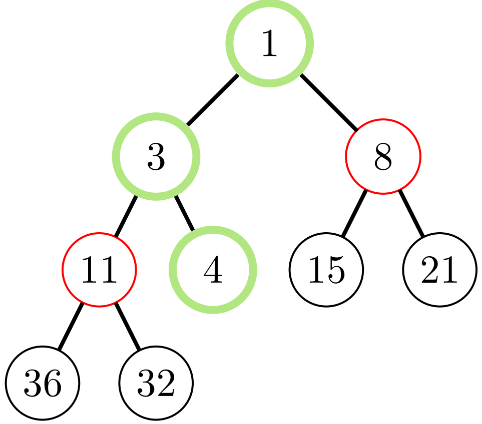


- Kopiec jest drzewem binarnym, w którym korzeń pamięta punkt ze zbioru o minimalnym kluczu (wartości).
- Reszta zbioru jest podzielona na dwa podzbiory o prawie równym rozmiarze i te podzbiory pamiętywane są rekurencyjnie w ten sam sposób.

Obserwacja. Zapytanie o jednowymiarowy obszar $(-\infty, x_2]$ można wykonać w czasie rzędu rzędu $O(1 + k)$, gdzie k jest liczbą zgłaszanych punktów.

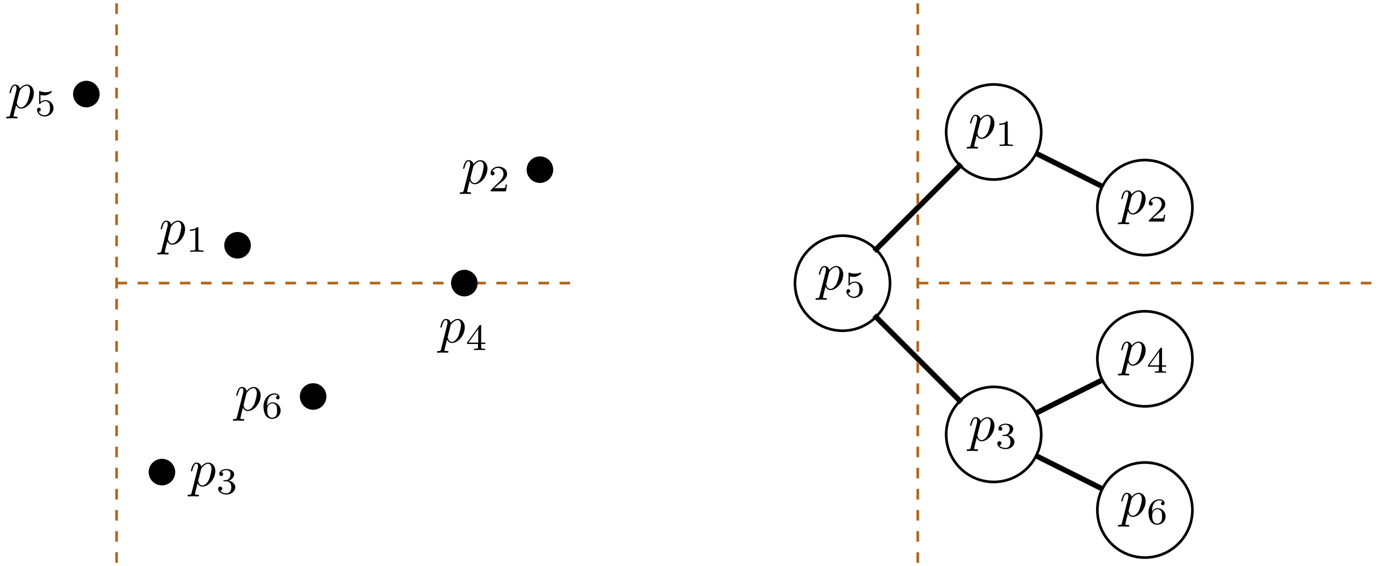
Idea algorytmu zapytania w kopcu jednowymiarowym

- Odwiedzanie węzłów drzewa w głąb: gdy odwiedzamy węzeł v, sprawdzamy, czy jego wartość leży w przedziale $(-\infty, x_2]$.
  - ↳ Jeśli tak, to podajemy punkt i kontynuujemy przeszukiwanie w obu synach węzła v. W przeciwnym razie przerywamy przeszukiwanie w tej części drzewa.



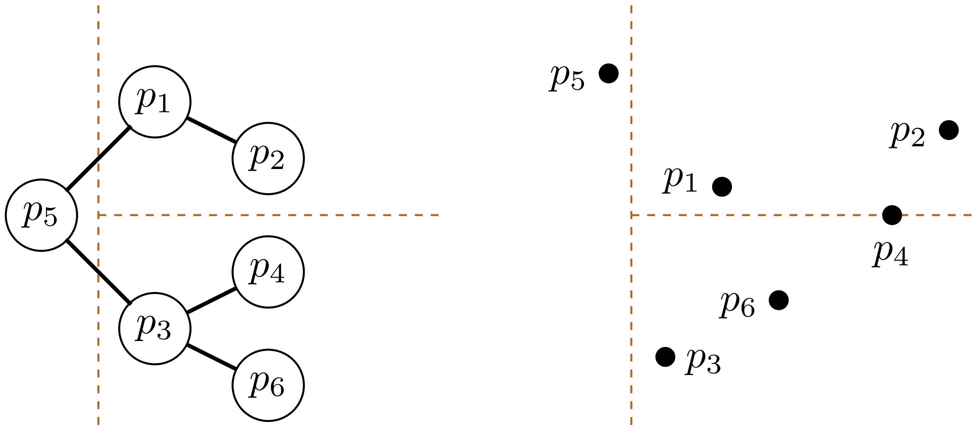
Przykład. Poszukujemy punktów z przedziału $(-\infty, 5]$ w powyższym drzewie. Wówczas odwiedzamy i podajemy węzły/punkty $1, 3$ oraz 4; odwiedzamy także węzły 8 i 11, ale w nich przeszukiwanie zostaje przerwane.

Drzewo przeszukiwań priorytetowych



Ilustracja idei.

- Punkt p_5 ma najmniejszą współrzędną x – zatem stanowi on korzeń drzewa.
- Pozostałe punkty dzielone są „na pół” względem współrzędnej y.
  - ↳ Punkty p_3, p_4 i p_6 mają mniejszą współrzędną y od punktów p_1 i p_2. Pamiętane są w dolnym (lewym) poddrzewie, którego korzeniem jest p_3 – punkt ten ma najmniejszą współrzędną x spośród punktów p_3, p_4 i p_6.
  - ↳ Punkty p_1 i p_2 pamiętane są w górnym (prawym) poddrzewie, którego korzeniem jest p_1 ($\mathrm{x}(p_1) < \mathrm{x}(p_2)$).



Drzewo przeszukiwań priorytetowych dla zbioru punktów S.

(Zakładamy, że wszystkie punkty mają różne odcięte i rzędne.)

- Jeśli $S = \emptyset$, to drzewo przeszukiwań binarnych jest pustym liściem.
- W przeciwnym przypadku, niech p_{min} będzie punktem w zbiorze S o najmniejszej współrzędnej x. Niech y_{med} będzie medianą współrzędnych y pozostałych punktów i niech

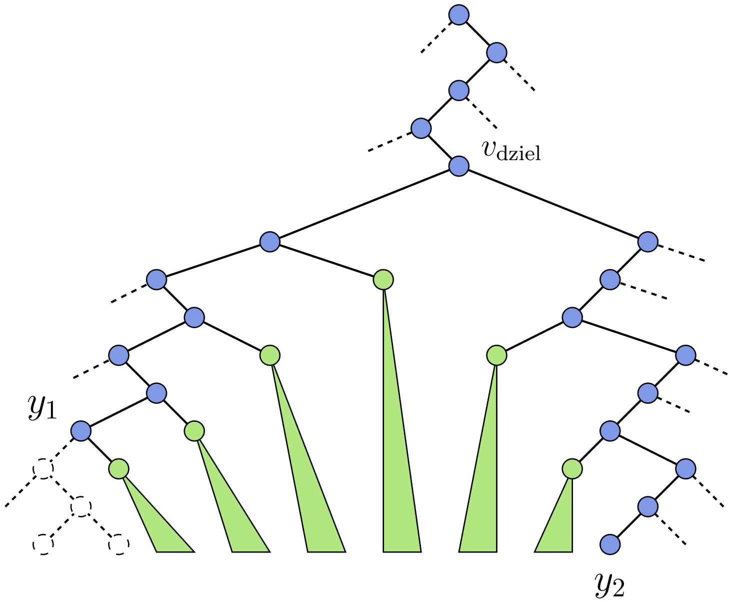
$$S_{\text{poniżej}} := \{p \in S \setminus \{p_{\mathrm{min}}\} \ : \ \mathrm{y}(p) \leq y_{\mathrm{med}}\}$$
$$\text{oraz } S_{\text{powyżej}} := \{p \in S \setminus \{p_{\mathrm{min}}\} \ : \ \mathrm{y}(p) > y_{\mathrm{med}}\}.$$

Drzewo przeszukiwań priorytetowych składa się z korzenia v, w którym są pamiętane punkt $p(v) := p_{\mathrm{min}}$ oraz wartość $y(v) := y_{\mathrm{med}}$, lewe poddrzewo v jest drzewem przeszukiwań priorytetowych dla zbioru $S_{\text{poniżej}}$, a prawe poddrzewo jest drzewem przeszukiwań priorytetowych dla $S_{\text{powyżej}}$.

Czas konstrukcji: $O(n \log n)$.

Idea algorytmu zapytań dla obszaru $(-\infty, x_2] \times [y_1, y_2]$

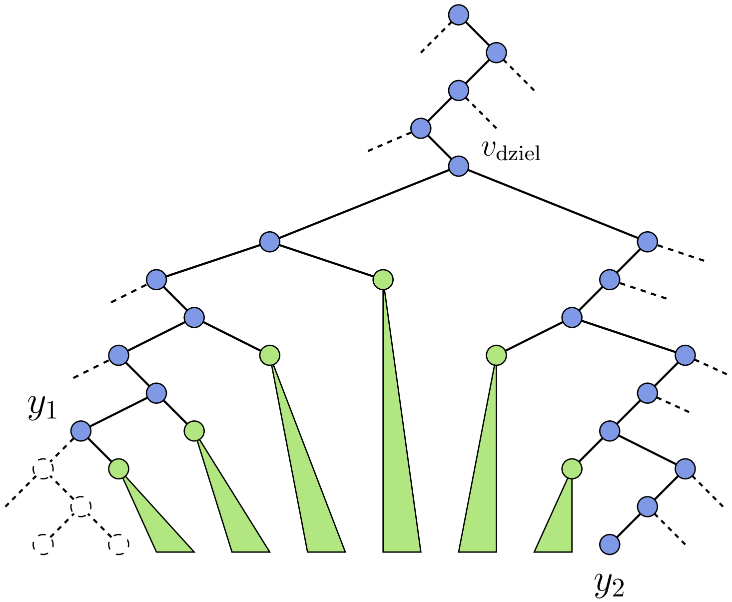
- Poszukujemy w drzewie wartości y_1 i y_2. Interesują nas węzły na ścieżkach poszukiwań π_1 i π_2 wartości y_1 i y_2, a także poddrzewa pomiędzy tymi ścieżkami.
  - ↳ Dla każdego z węzła v na ścieżkach poszukiwań y_1 i y_2 sprawdzamy, czy odpowiadający mu punkt $p(v)$ należy do obszaru zapytania.
  - ↳ Dla każdego poddrzewa T pomiędzy ścieżkami π_1 i π_2 wywołujemy poniższą procedurę `ReportInSubtree`.



Idea algorytmu zapytań dla obszaru $(-\infty, x_2] \times [y_1, y_2]$

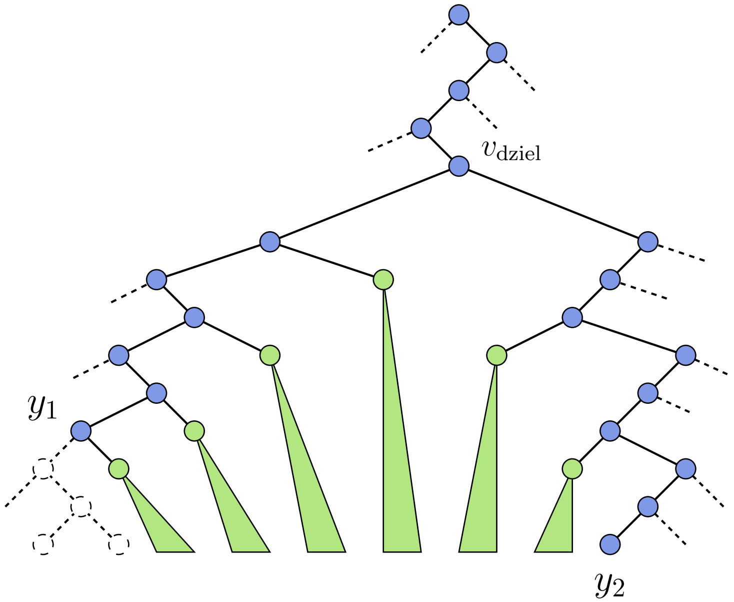
- Poszukujemy w drzewie wartości y_1 i y_2. Interesują nas węzły na ścieżkach poszukiwań π_1 i π_2 wartości y_1 i y_2, a także poddrzewa pomiędzy tymi ścieżkami.
  - ↳ Dla każdego z węzła v na ścieżkach poszukiwań y_1 i y_2 sprawdzamy, czy odpowiadający mu punkt $p(v)$ należy do obszaru zapytania.
  - ↳ Dla każdego poddrzewa T pomiędzy ścieżkami π_1 i π_2 wywołujemy poniższą procedurę `ReportInSubtree`.

    Rzędne punktów przechowywanych w T należą do przedziału $[y_1, y_2]$, zatem wystarczy sprawdzić tylko odciętą tych punktów.



Idea algorytmu zapytań dla obszaru $(-\infty, x_2] \times [y_1, y_2]$

- Poszukujemy w drzewie wartości y_1 i y_2. Interesują nas węzły na ścieżkach poszukiwań π_1 i π_2 wartości y_1 i y_2, a także poddrzewa pomiędzy tymi ścieżkami.
  - ↳ Dla każdego z węzła v na ścieżkach poszukiwań y_1 i y_2 sprawdzamy, czy odpowiadający mu punkt $p(v)$ należy do obszaru zapytania.
  - ↳ Dla każdego poddrzewa T pomiędzy ścieżkami π_1 i π_2 wywołujemy poniższą procedurę `ReportInSubtree`.



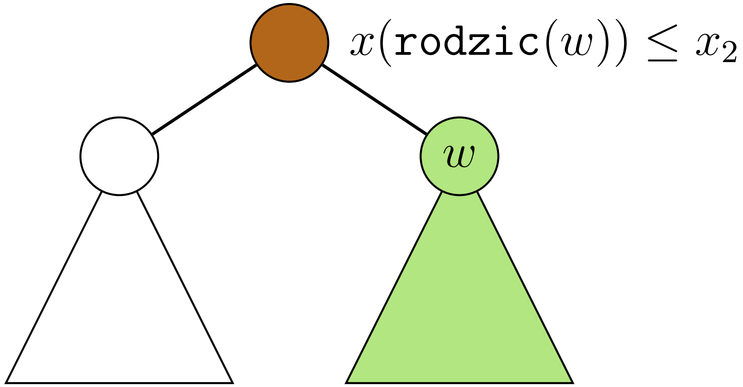
ReportInSubtree(v, x_2)

Wejście. Korzeń poddrzewa przeszukiwań priotytetowych i wartość x_2.
Wyjście. Wszystkie punkty w poddrzewie o odciętej nie większej od x_2.

```
1. if $v \neq$ null oraz x($p(v)$) $\leq x_2$
2.   then Podaj $p(v)$.
3.          ReportInSubtree(lewy-syn($v$), $x_2$)
4.          ReportInSubtree(prawy-syn($v$), $x_2$)
```

Lemat 5.10. *Procedura* ReportInSubtree(v, x_2) *podaje w czasie rzędu* $O(1 + k_v)$ *wszystkie punkty w poddrzewie zakorzenionym w węźle* v*, których odcięta jest nie większa od* x_2*, gdzie* k_v *jest liczbą podawanych punktów.*

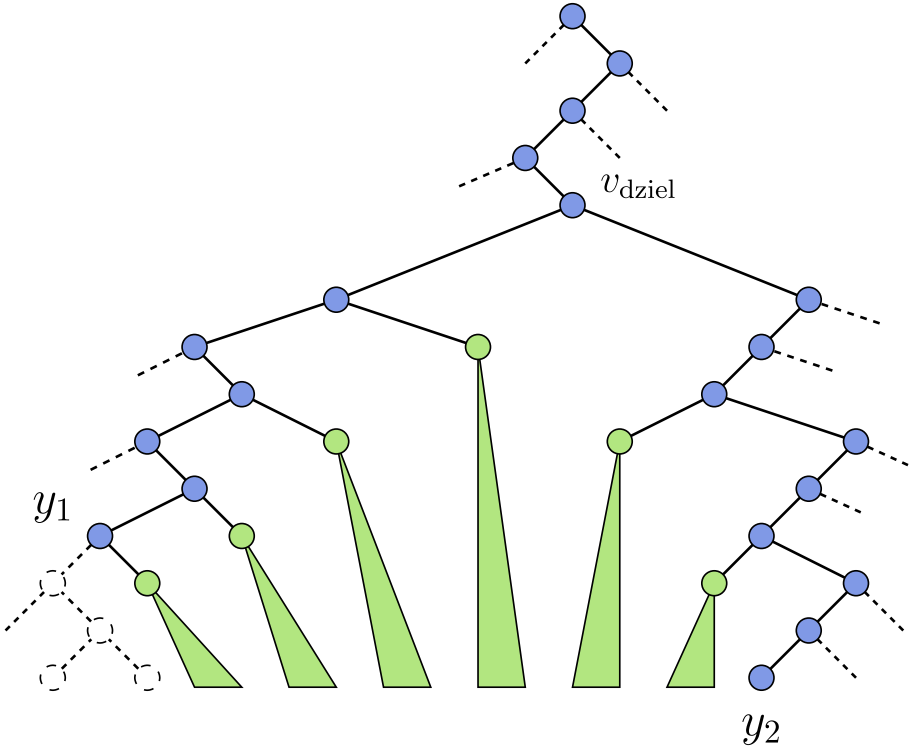
- Zgłoszone zostają wszystkie i tylko te punkty, które leżą w obszarze zapytania.



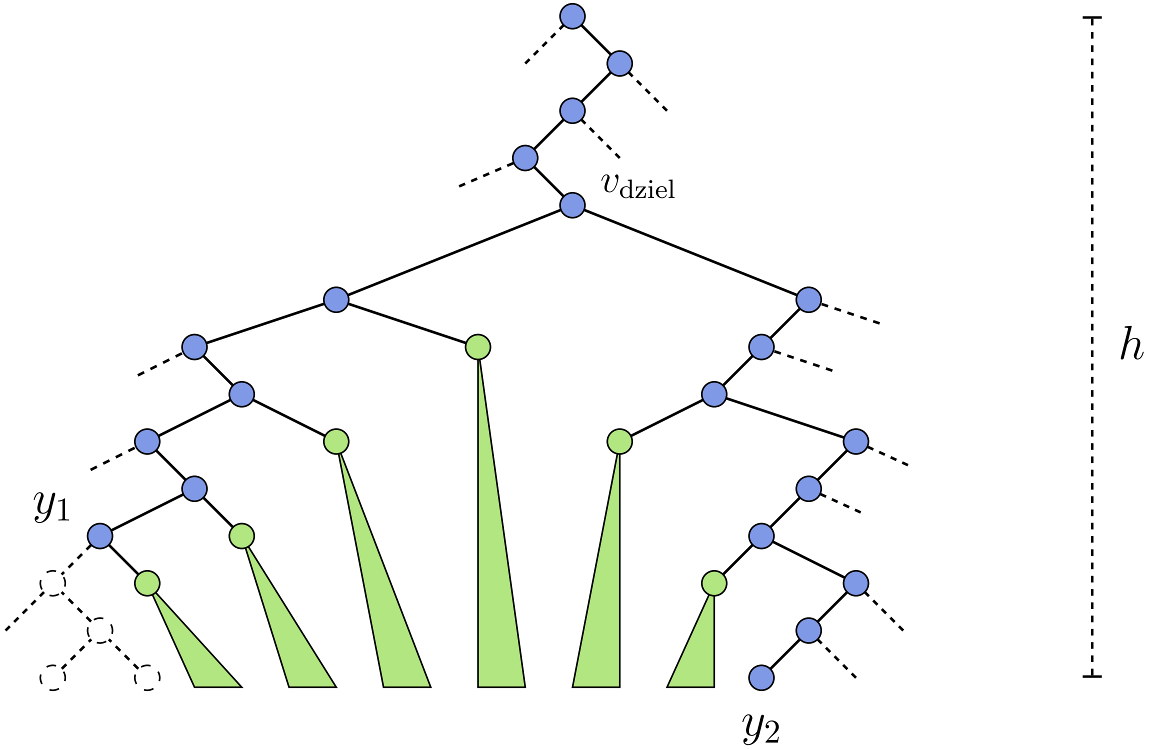
- Złożoność czasowa procedury.
  - ↳ Gdy odwiedzamy węzeł $w \neq v$, musieliśmy wcześniej podać punkt pamiętany przez jego rodzica.
  - ↳ Liczba odwiedzonych węzłów $w \neq v$ jest nie większa niż podwojona liczba zgłoszonych punktów, a zatem jest ona rzędu $O(k_v)$.
  - ↳ W konsekwencji otrzymujemy $O(1 + k_v)$.

Lemat 5.11.

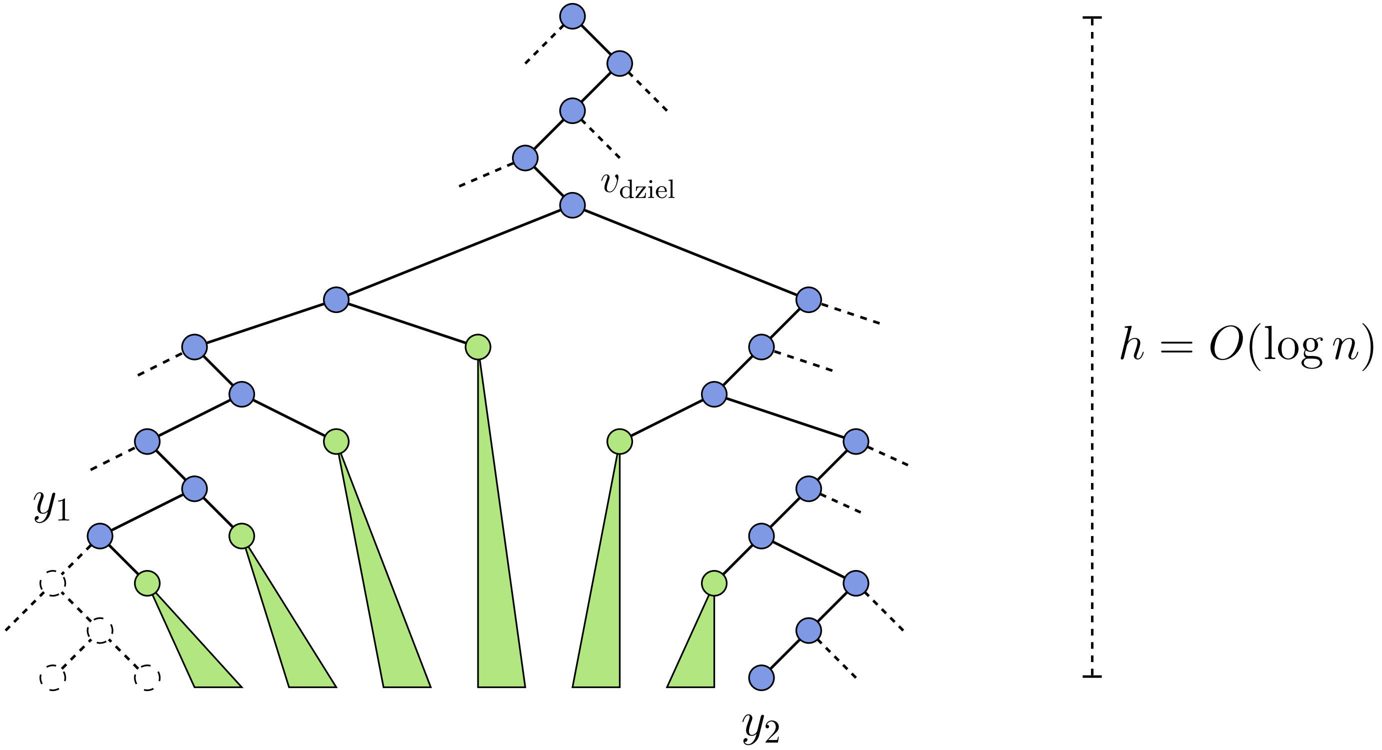
Algorytm podaje wszystkie punkty z obszaru zapytania $(-\infty, x_2] \times [y_1, y_2]$ *w czasie rzędu* $O(h + k)$, *gdzie* h *jest głębokością drzewa, a* k *liczbą podawanych punktów.*



- ▶ Każdy punkt zgłaszany przez algorytm leży w obszarze zapytania.
- ▶ Każdy punkt p z obszaru zapytania zostanie zgłoszony.



- Złożoność czasowa.
  - ↳ Czas liniowy względem liczby węzłów na ścieżkach poszukiwań y_1 oraz y_2.
  - ↳ Głębokość drzewa wynosi h, a zatem całkowita liczba węzłów na ścieżkach poszukiwań jest rzędu $O(h)$.
  - ↳ W konsekwencji, mając na uwadze lemat 4.10, czas wymagany przez wszystkie wywołania ReportInSubtree jest rzędu $O(h + k)$.



Dla zbioru n punktów na płaszczyźnie drzewo przeszukiwań priorytetowych jest drzewem binarnym o głębokości $O(\log n)$ i wykorzystywanej pamięci rzędu $O(n)$.

Twierdzenie 5.12. (McCreight 1985)
Drzewo przeszukiwań priorytetowych dla zbioru n punktów na płaszczyźnie, które używa $O(n)$ pamięci, można zbudować w czasie $O(n \log n)$. Wykorzystując to drzewo można podać wszystkie punkty z obszaru zapytania $(-\infty, x_2] \times [y_1, y_2]$ w czasie $O(\log n + k)$, gdzie k jest liczbą podawanych punktów.

5.6 OGÓLNY ZBIÓR PUNKTÓW

Niech $a|b$ oznacza liczbę złożoną z dwóch liczb rzeczywistych a i b. Zdefiniujmy następujący *porządek leksykograficzny*: dla dwóch liczb złożonych $(a|b)$ i $(a'|b')$ zachodzi

$$(a|b) < (a'|b') \;\Leftrightarrow\; a < a' \text{ lub } (a = a' \text{ i } b < b').$$

5.6 OGÓLNY ZBIÓR PUNKTÓW

Niech $a|b$ oznacza liczbę złożoną z dwóch liczb rzeczywistych a i b. Zdefiniujmy następujący *porządek leksykograficzny*: dla dwóch liczb złożonych $(a|b)$ i $(a'|b')$ zachodzi

$$(a|b) < (a'|b') \;\Leftrightarrow\; a < a' \text{ lub } (a = a' \text{ i } b < b').$$

Załóżmy, że mamy dany zbiór S zawierający n różnych punktów na płaszczyźnie. Rozważmy zbiór S' określony następująco:

$$S' = \{((p_x|p_y), (p_y|p_x)) \;:\; (p_x, p_y) \in S\}.$$

Jako że punkty w S są różne, pierwsze współrzędne dowolnych punktów z S' są także różne; to samo pozostaje prawdą dla drugiej współrzędnej.

5.6 OGÓLNY ZBIÓR PUNKTÓW

Niech $a|b$ oznacza liczbę złożoną z dwóch liczb rzeczywistych a i b. Zdefiniujmy następujący *porządek leksykograficzny*: dla dwóch liczb złożonych $(a|b)$ i $(a'|b')$ zachodzi

$$(a|b) < (a'|b') \;\Leftrightarrow\; a < a' \text{ lub } (a = a' \text{ i } b < b').$$

Załóżmy, że mamy dany zbiór S zawierający n różnych punktów na płaszczyźnie. Rozważmy zbiór S' określony następująco:

$$S' = \{((p_x|p_y), (p_y|p_x)) \;:\; (p_x, p_y) \in S\}.$$

Jako że punkty w S są różne, pierwsze współrzędne dowolnych punktów z S' są także różne; to samo pozostaje prawdą dla drugiej współrzędnej. Przekształćmy teraz obszar zapytania $R = [x_1, x_2] \times [y_1, y_2]$ w obszar R' następująco:

$$R' = [(x_1| - \infty), (x_2| + \infty)] \times [(y_1| - \infty), (y_2| + \infty)].$$

Lemat 5.13. *Zachodzi* $p \in R \;\Leftrightarrow\; p' \in R'$.

BuildKdTree(S, d)

Wejście. Zbiór punktów S i aktualna głębokość d.
Wyjście. Korzeń kd-drzewa przechowującego zbiór S.

```
if $S$ zawiera tylko jeden punkt
   then return liść pamiętający ten punkt
   else if $d$ jest parzyste
      then Podziel $S$ na dwa zbiory $S_1$ i $S_2$ pionową prostą $l$ przechodzącą przez medianę
              współrzędnych $x$ punktów z $S$, gdzie $S_1$ zawiera punkty na lewo lub na prostej $l$,
              a $S_2$ zawiera punkty na prawo od prostej $l$.
      else Podziel $S$ na dwa zbiory $S_1$ i $S_2$ poziomą prostą $l$ przechodzącą przez medianę
              współrzędnych $y$ punktów z $S$, gdzie $S_1$ zawiera punkty poniżej lub na prostej $l$,
              a $S_2$ zawiera punkty powyżej prostej $l$.
         lewy-syn$(v)$ := BuildKdTree$(S_1, d + 1)$;
         prawy-syn$(v)$ := BuildKdTree$(S_2, d + 1)$;
         Stwórz wierzchołek $v$ pamietający prostą $l$, uczyń $v_{\text{lewy}}$ i $v_{\text{prawy}}$ jego lewym
         i prawym dzieckiem, odpowiednio.
return $v$.
```

KdTreeQuery(v, R)

Wejście. Korzeń kd-(pod)drzewa oraz obszar zapytania R.
Wyjście. Wszystkie punkty w liściach poniżej v, które leżą w obszarze R.

```
if $v$ jest liściem
  then Podaj punkt $p$ pamiętany w $v$, o ile $p \in R$.
  else if obszar(lewy-syn($v$)) jest całkowicie zawarty w $R$
         then ReportSubtree(lewy-syn($v$))
         else if obszar(lewy-syn($v$)) przecina $R$
           then KdTreeQuery(lewy-syn($v$), $R$)
       if obszar(prawy-syn($v$)) jest całkowicie zawarty w $R$
         then ReportSubtree(prawy-syn($v$))
         else if obszar(prawy-syn($v$)) przecina $R$
           then KdTreeQuery(prawy-syn($v$), $R$)
```

ReportSubtree jest procedurą, która przechodzi poddrzewo zakorzenione w danym węźle i wylicza wszystkie punkty pamiętane w jego liściach.

FindSplitNode$(\mathcal{T}, x_1, x_2)$

Wejście. Drzewo $\mathcal{T}$ i dwie wartości $x_1, x_2 \in \mathbb{R}$, $x_1 \leq x_2$.
Wyjście. Węzeł $v_{\rm dziel}$, w którym rozchodzą się ścieżki w poszukiwaniu x_1 i x_2;
lub liść, w którym obie ścieżki kończą się.

```
1. $v :=$ korzeń$(\mathcal{T})$;
2. while $v$ nie jest liściem oraz $(x_2 \leq$ klucz$(v)$ lub $x_1 >$ klucz$(v))$ do
   2.1 if $x_2 \leq$ klucz$(v)$ then $v :=$ lewy-syn$(v)$;
   2.2     else $v :=$ prawy-syn$(v)$;
3. Zwróć $v$.
```

1DRangeQuery($\mathcal{T}$, $[x_1, x_2]$)

Wejście. Drzewo $\mathcal{T}$ i przedział $[x_1, x_2]$.
Wyjście. Wszystkie punkty z $\mathcal{T}$, które leżą w przedziale $[x_1, x_2]$.

```
$v_{\text{dziel}}$ := FindSplitNode($\mathcal{T}, x_1, x_2$);
if $v_{\text{dziel}}$ jest liściem
  then Sprawdź, czy klucz($v_{\text{dziel}}$) musi być podany ($\in [x_1, x_2]$).
  else /* Idź ścieżką do $l_1$ i wyliczaj punkty w poddrzewach na prawo od ścieżki.*/
    $v$ := lewy-syn($v_{\text{dziel}}$);
    while $v$ nie jest liściem do
      if $x_1 \leq$ klucz($v$)
      then ReportSubtree(prawy-syn($v$));
              $v$ := lewy-syn($v$);
      else $v$ := prawy-syn($v$);
    Sprawdź, czy punkt pamiętany w liściu $v$ $(= l_1)$ musi być podany.
    Analogiczne postępowanie dla ścieżki poszukiwań $x_2$:
     – wylicz punkty w poddrzewach na lewo od ścieżki;
     – sprawdź, czy wartość na końcu ścieżki (liść) musi być podana.
```

2DRangeQuery$(\mathcal{T}, [x_1, x_2] \times [y_1, y_2])$

Wejście. Dwuwymiarowe drzewo obszarów $\mathcal{T}$ i obszar $[x_1, x_2] \times [y_1, y_2]$.
Wyjście. Wszystkie punkty z $\mathcal{T}$, które leżą w obszarze zapytania.

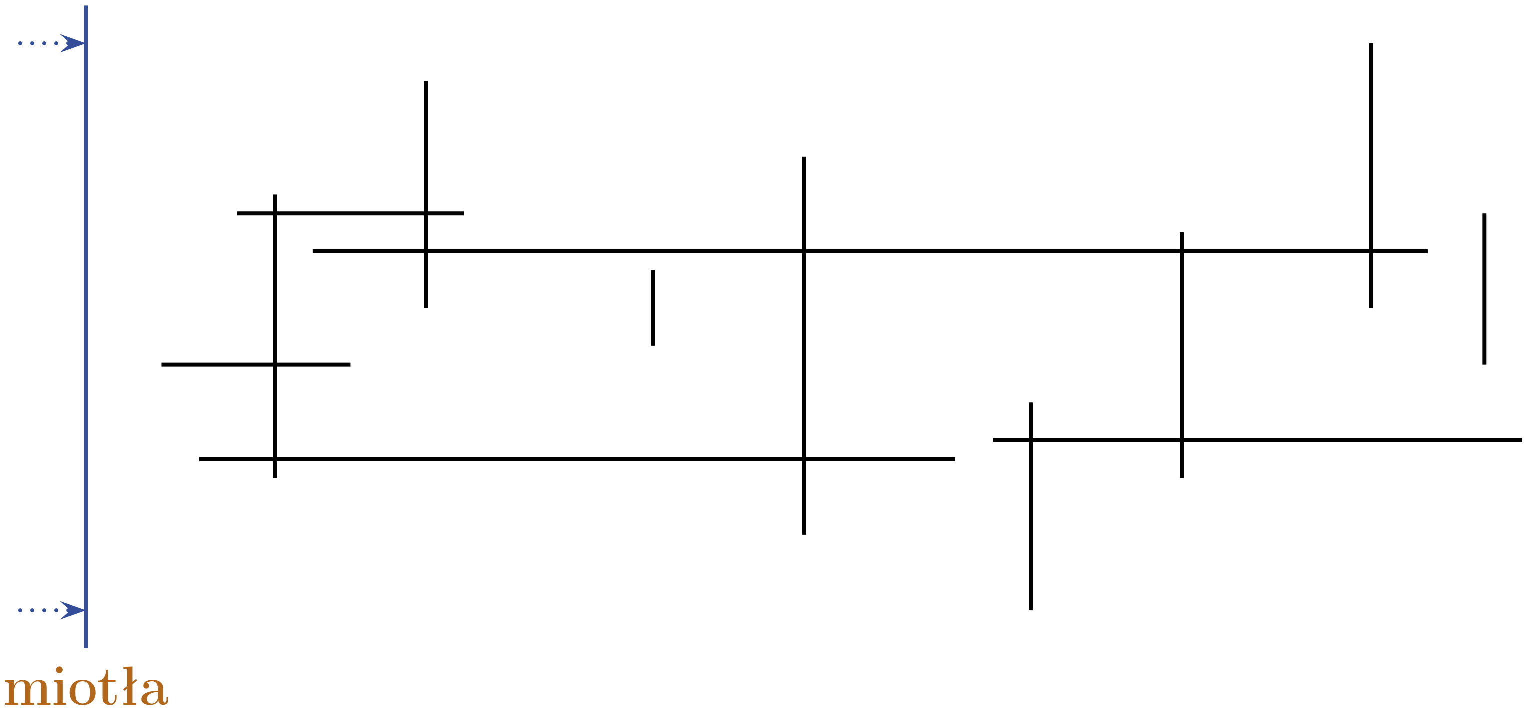
```
$v_{\text{dziel}}$ := FindSplitNode($\mathcal{T}, x_1, x_2$);
if $v_{\text{dziel}}$ jest liściem
  then Sprawdź, czy punkt pamietany w $v_{\text{dziel}}$ musi być podany.
  else /* Idź ścieżką w poszukiwaniu $x_1$ i wywołuj 1DRangeQuery
                dla poddrzew stowarzyszonych na prawo od ścieżki.*/
     $v$ := lewy-syn($v_{\text{dziel}}$);
     while $v$ nie jest liściem do
        if $x_1 \leq$ klucz($v$)
        then 1DRangeQuery($\mathcal{T}_{\text{stow}}$(prawy-syn($v$)), $[y_1, y_2]$);
               $v$ := lewy-syn($v$);
        else $v$ := prawy-syn($v$);
     Sprawdź, czy punkt pamiętany w liściu $v$ musi być podany.
     Analogiczne postępowanie dla ścieżki poszukiwań $x_2$.
     /* Idź ścieżką w poszukiwaniu $x_2$ i wywołuj 1DRangeQuery dla poddrzew stowarzyszonych
        na lewo od ścieżki i sprawdź, czy punkt pamiętany na końcu ścieżki musi być podany. */
```

QuerySearchTree$(\mathcal{T}, (-\infty, x_2] \times [y_1, y_2])$

Wejście. Korzeń drzewa przeszukiwań priotytetowych i obszar zapytania.
Wyjście. Wszystkie punkty w obszarze zapytania.

```
1. Wyszukaj $y_1$ oraz $y_2$ w kopcu $\mathcal{T}$. Niech $v_{\text{dziel}}$ będzie węzłem, w którym ścieżki poszukiwań $y_1$ oraz $y_2$ rozchodzą
   się, i niech $v_1$ oraz $v_2$ będą węzłami, w których skończyło się przeszukiwanie odpowiednio $y_1$ oraz $y_2$. (Jeśli w
   drzewie brak jest którejś z wartości, to odpowiedni wierzchołek końcowy ścieżki jest liściem).
2. for każdy węzeł $v$ na ścieżce poszukiwań $y_1$ oraz $y_2$ do
3.     if $p(v) \in (-\infty, x_2] \times [y_1, y_2])$ then podaj $p(v)$
4. for każdy węzeł $v$ na ścieżce z $v_{\text{dziel}}$ do $v_1$ do
5.     if w węźle $v$ ścieżka poszukiwania $y_1$ idzie w lewo
6.         then ReportInSubtree(prawy-syn$(v), x_2)$
7. for każdy węzeł $v$ na ścieżce z $v_{\text{dziel}}$ do $v_2$ do
8.     if w węźle $v$ ścieżka poszukiwania $y_2$ idzie w prawo
9.         then ReportInSubtree(lewy-syn$(v), x_2)$
```

6. TECHNIKA ZAMIATANIA (na płaszczyźnie)

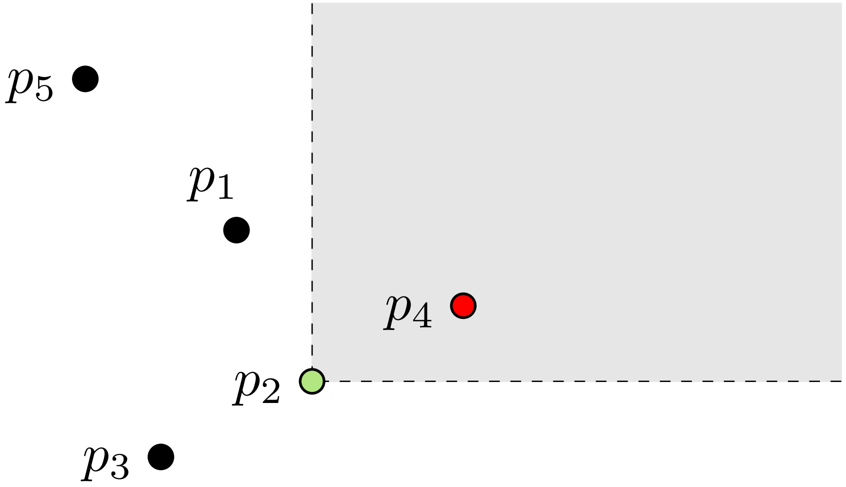


- ▶ Idea algorytmu zamiatania prostą polega na przesuwaniu pionowej prostej – *miotły* – po płaszczyźnie z lewa na prawo (z góry na dół).
- ▶ Podczas zamiatania utrzymywane są dodatkowe informacje: *status* miotły.
- ▶ Status miotły zmienia się w *punktach zdarzeń*.

6.1 PUNKTY DOMINUJĄCE

Dla dwóch punktów $p, q \in \mathbb{R}^2$ mówimy, że p *dominuje* nad punktem q, ozn. $q \prec p$, jeśli $\mathrm{x}(q) \leq \mathrm{x}(p)$ oraz $\mathrm{y}(q) \leq \mathrm{y}(p)$. Punkt $p \in S$ jest *elementem maksymalnym* (*maksimum*), jeśli nie istnieje żaden punkt $q \in S \setminus \{p\}$ taki, że $p \prec q$.

F.P. Preparata, M.I. Shamos
Geometria obliczeniowa – wprowadzenie
rozdział 4.1.3, Helion (2003)

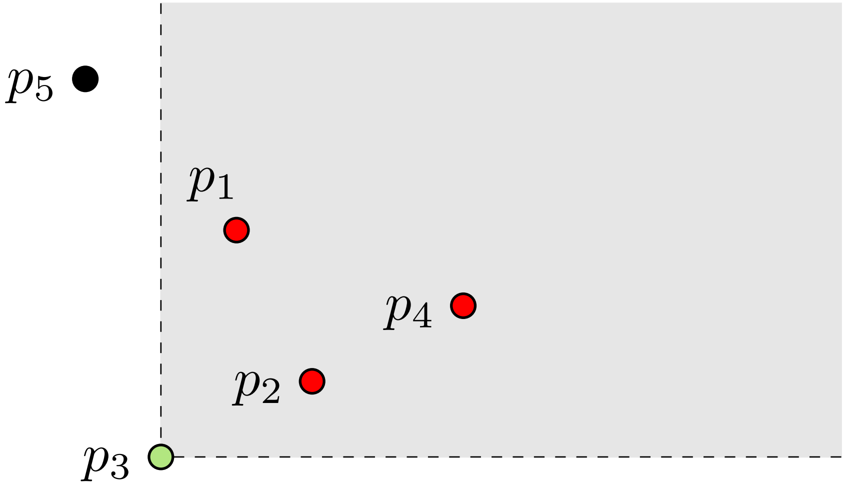


Przykład. Punkt p_2 jest zdominowany przez punkt p_4, a punkt p_3 jest zdominowany przez punkty p_1, p_2, p_4. Natomiast punkty p_1, p_4, p_5 są punktami maksymalnymi.

6.1 PUNKTY DOMINUJĄCE

Dla dwóch punktów $p, q \in \mathbb{R}^2$ mówimy, że p *dominuje* nad punktem q, ozn. $q \prec p$, jeśli $\mathrm{x}(q) \le \mathrm{x}(p)$ oraz $\mathrm{y}(q) \le \mathrm{y}(p)$. Punkt $p \in S$ jest *elementem maksymalnym* (*maksimum*), jeśli nie istnieje żaden punkt $q \in S \setminus \{p\}$ taki, że $p \prec q$.

F.P. Preparata, M.I. Shamos
Geometria obliczeniowa – wprowadzenie
rozdział 4.1.3, Helion (2003)

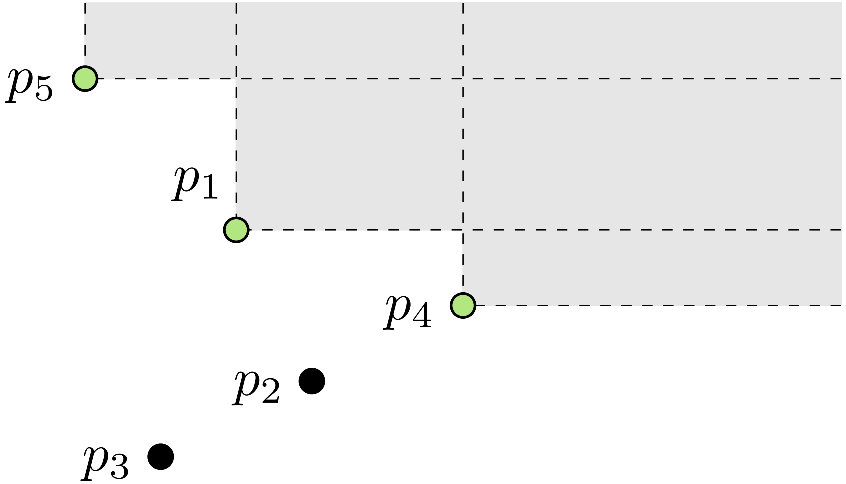


Przykład. Punkt p_2 jest zdominowany przez punkt p_4, a punkt p_3 jest zdominowany przez punkty p_1, p_2, p_4. Natomiast punkty p_1, p_4, p_5 są punktami maksymalnymi.

6.1 PUNKTY DOMINUJĄCE

Dla dwóch punktów $p, q \in \mathbb{R}^2$ mówimy, że p *dominuje* nad punktem q, ozn. $q \prec p$, jeśli $\mathrm{x}(q) \leq \mathrm{x}(p)$ oraz $\mathrm{y}(q) \leq \mathrm{y}(p)$. Punkt $p \in S$ jest *elementem maksymalnym* (*maksimum*), jeśli nie istnieje żaden punkt $q \in S \setminus \{p\}$ taki, że $p \prec q$.

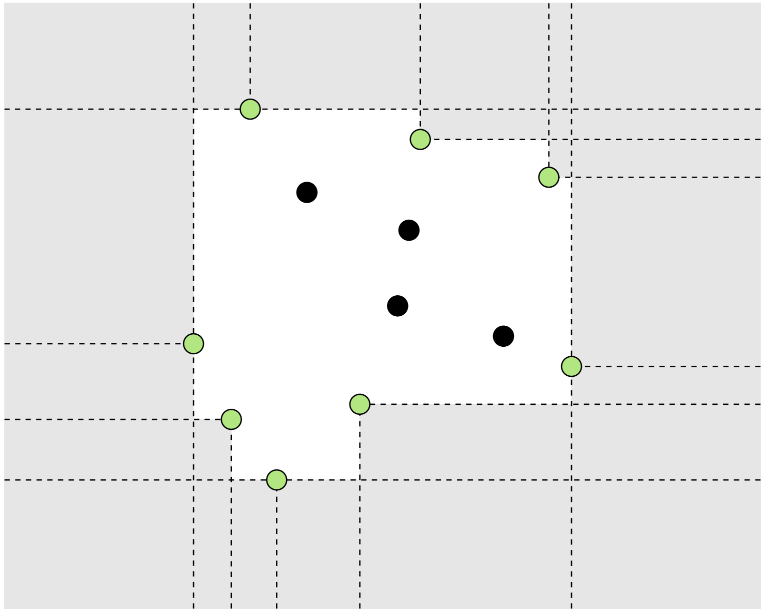
F.P. Preparata, M.I. Shamos
Geometria obliczeniowa – wprowadzenie
rozdział 4.1.3, Helion (2003)



Przykład. Punkt p_2 jest zdominowany przez punkt p_4, a punkt p_3 jest zdominowany przez punkty p_1, p_2, p_4. Natomiast punkty p_1, p_4, p_5 są punktami maksymalnymi.

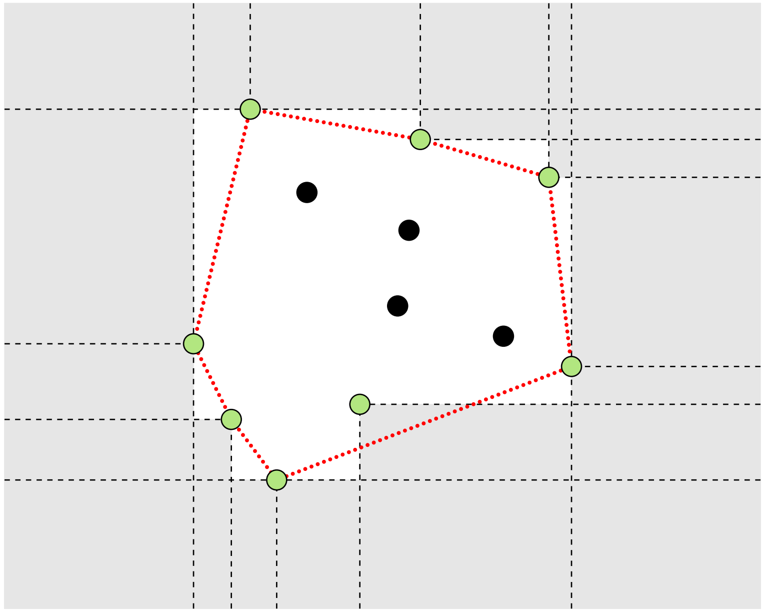
6.1 PUNKTY DOMINUJĄCE

Dla dwóch punktów $p, q \in \mathbb{R}^2$ mówimy, że p *dominuje* nad punktem q, ozn. $q \prec p$, jeśli $\mathrm{x}(q) \leq \mathrm{x}(p)$ oraz $\mathrm{y}(q) \leq \mathrm{y}(p)$. Punkt $p \in S$ jest *elementem maksymalnym* (*maksimum*), jeśli nie istnieje żaden punkt $q \in S \setminus \{p\}$ taki, że $p \prec q$.

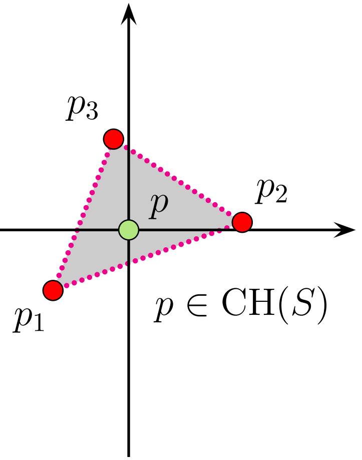


Dla zbioru $S \subset \mathbb{R}^2$, przypisując poszczególnym współrzędnym punktów znaki $+$ i $-$, można zdefiniować cztery problemy wyznaczenia maksimum.

Problem. *Dla danego zbioru punktów $S \subset \mathbb{R}^2$ wyznacz wszystkie jego punkty maksymalne, po wszystkich czterech możliwych przypisaniach znaków.*

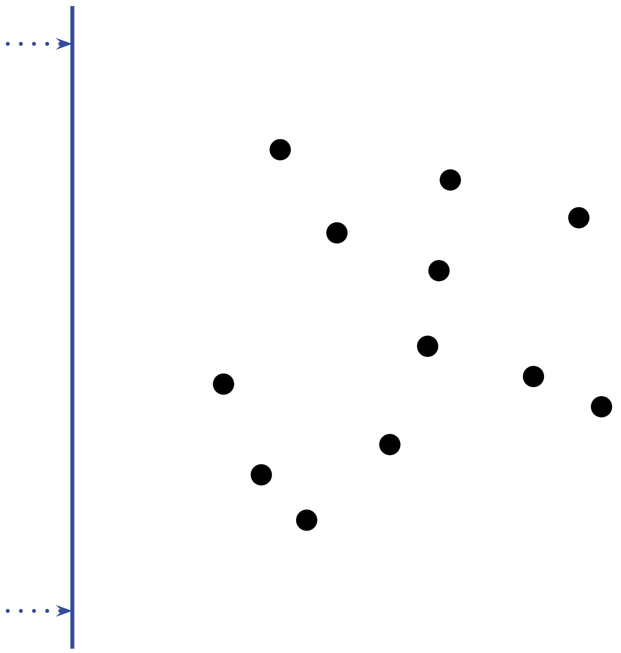


Twierdzenie 6.1. (Bentley, Kung, Schkolnick, Thompson, 1978)
Element zbioru należący do otoczki wypukłej tego zbioru jest elementem maksymalnym w przynajmniej jednym przypisaniu znaków współrzędnych.



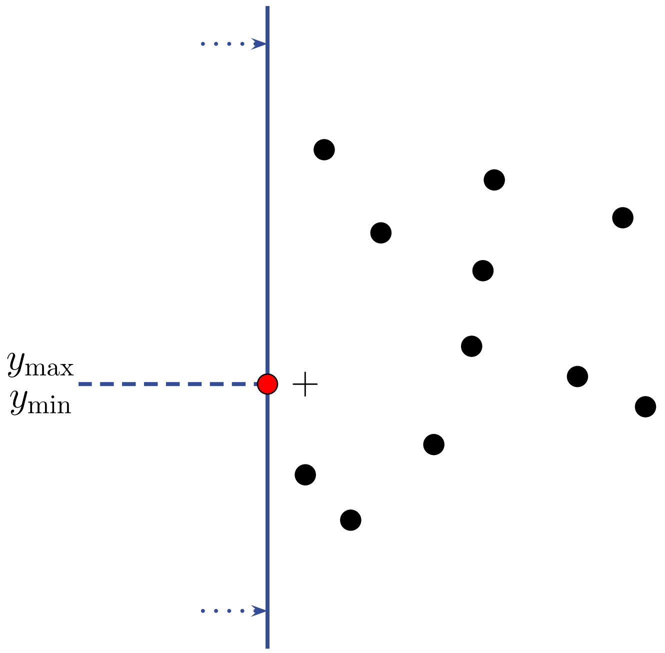
Twierdzenie 6.1. (Bentley, Kung, Schkolnick, Thompson, 1978) *Element zbioru należący do otoczki wypukłej tego zbioru jest elementem maksymalnym w przynajmniej jednym przypisaniu znaków współrzędnych.*

Dowód. Załóżmy przez sprzeczność, że istnieje punkt $p \in S$ otoczki $\mathrm{CH}(S)$, który nie jest punktem maksymalnym dla żadnego z przypisań znaków. Rozważmy układ współrzędnych o początku w punkcie p. Wówczas z założenia istnieją przynajmniej trzy punkty $p_1, p_2, p_3 \in S \setminus \{p\}$ dominujące nad p i takie, że p należy do otoczki wypukłej $\mathrm{CH}(\{p_1, p_2, p_3\})$ – ale że $\mathrm{CH}(\{p_1, p_2, p_3\}) \subseteq \mathrm{CH}(S)$, otrzymujemy sprzeczność, że p jest punktem otoczki. ☐



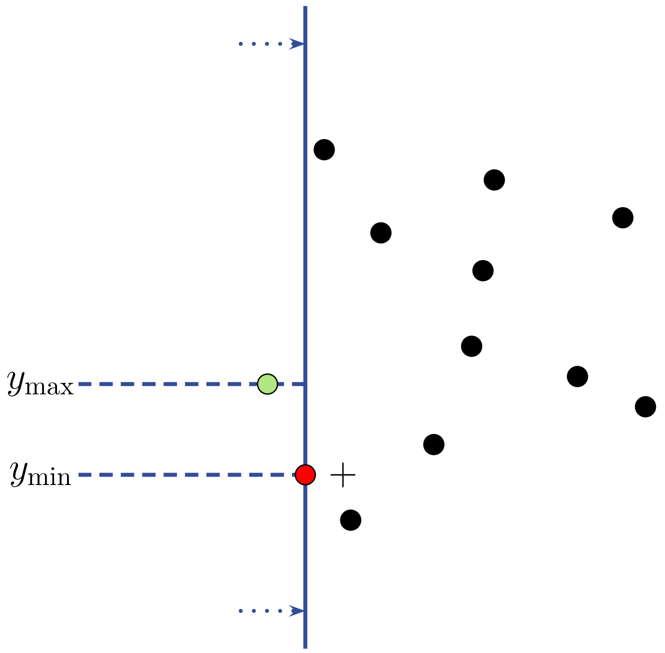
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



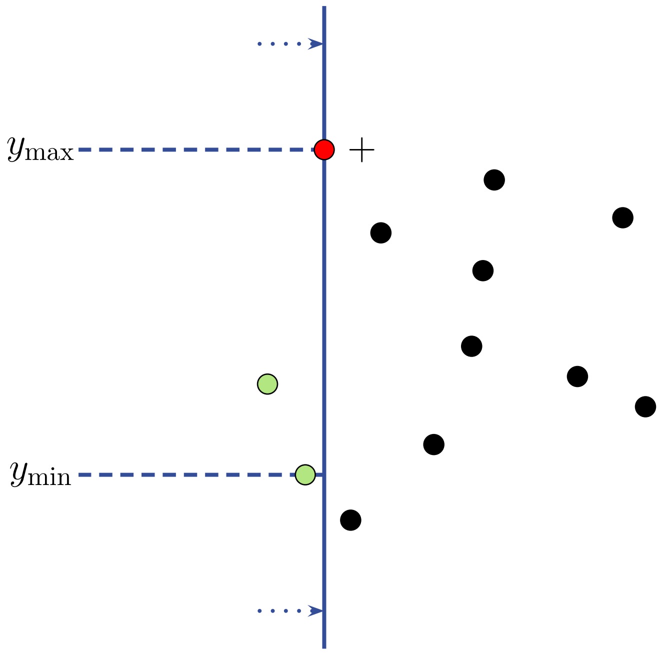
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



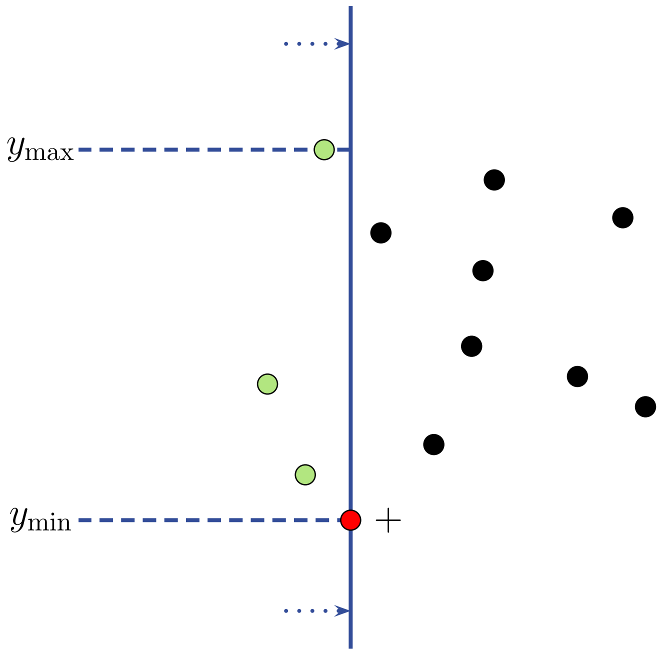
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



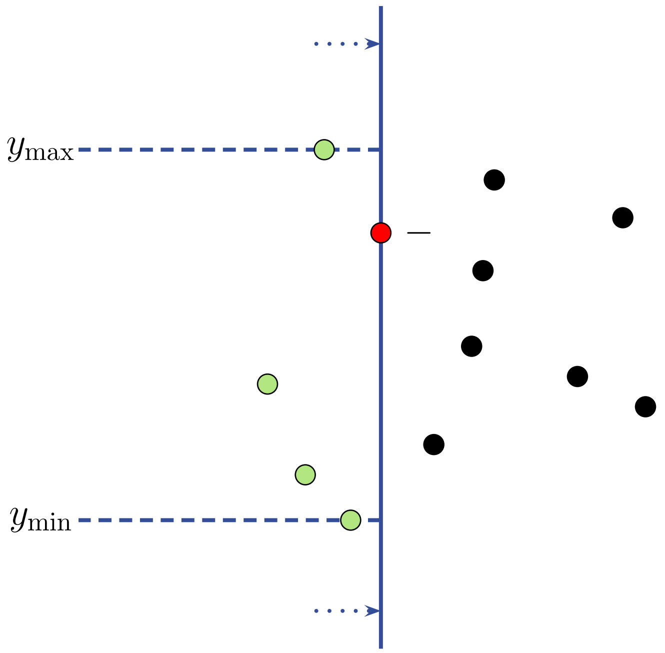
Idea algorytmu /Kung, Luccio, Preparata 1975/

- ▶ Przesuwamy (wirtualną) miotłę z lewa na prawo.
- ▶ Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- ▶ Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- ▶ Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- ▶ W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



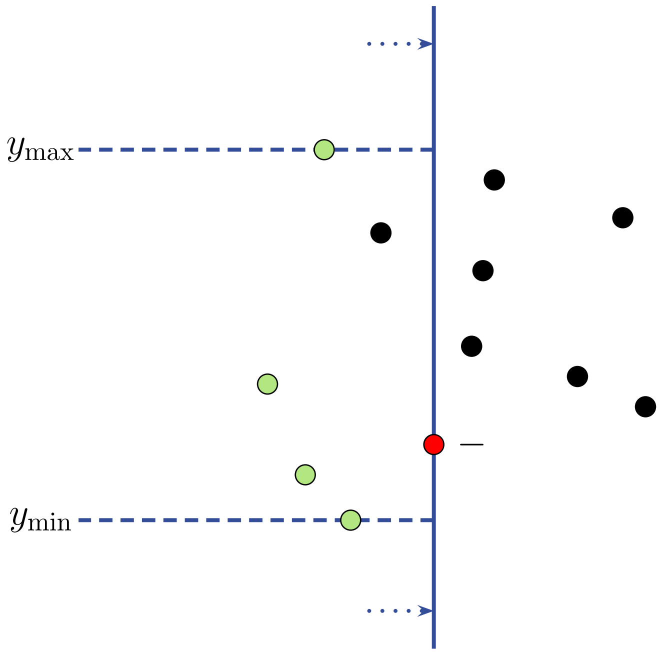
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



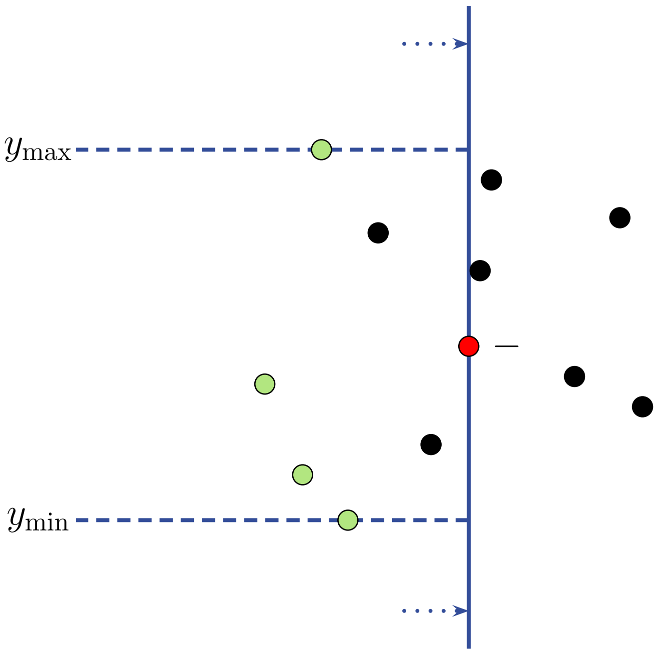
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



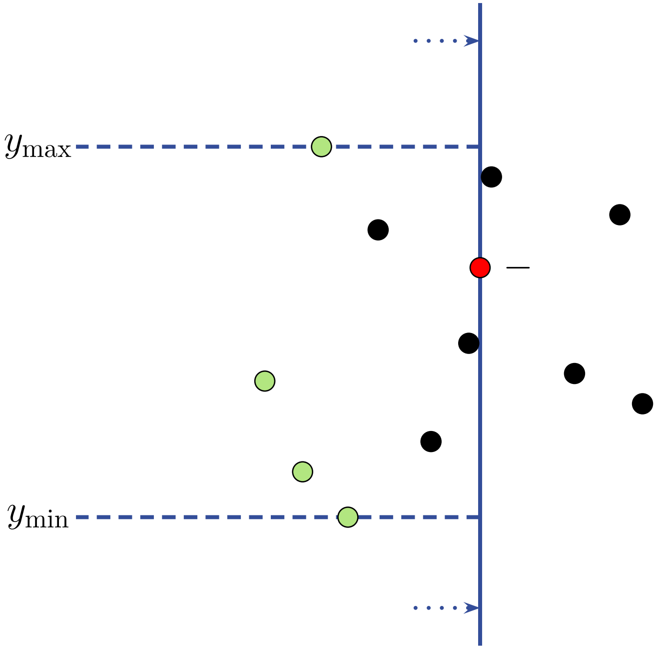
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



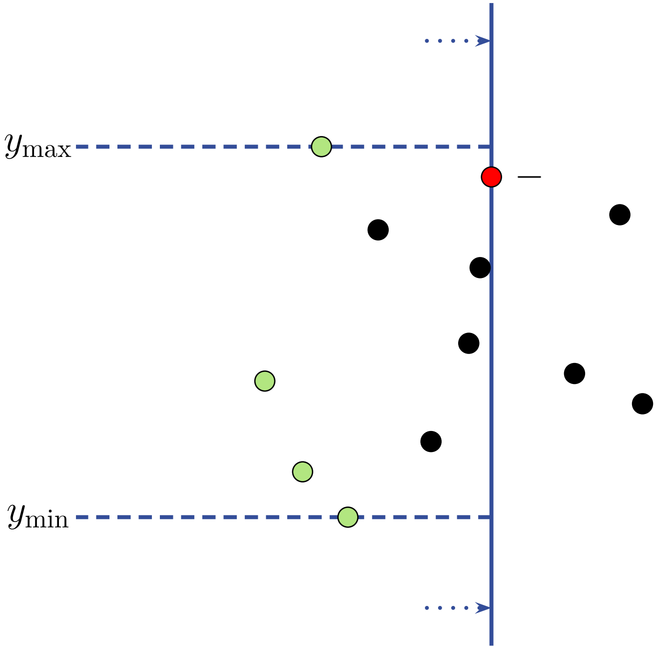
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



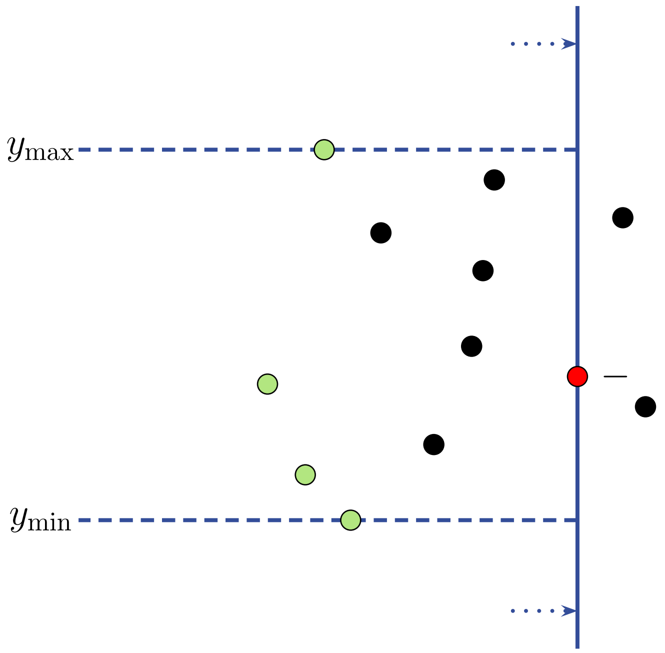
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



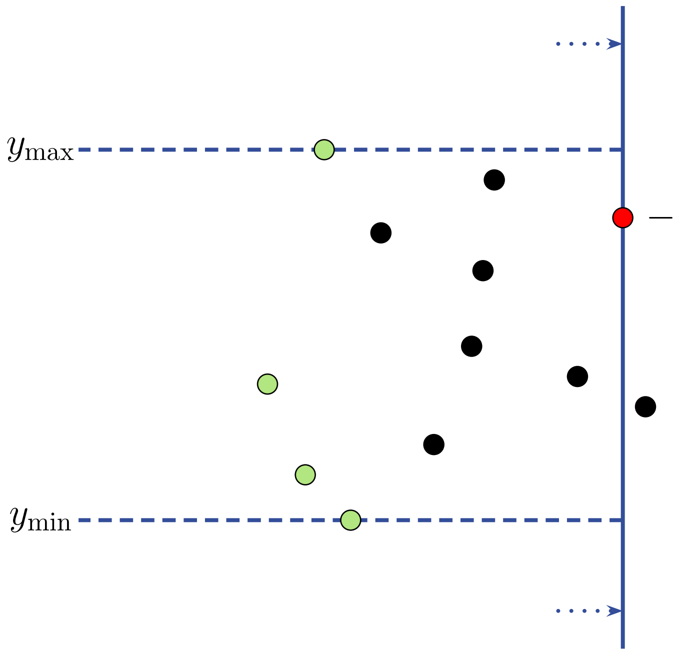
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



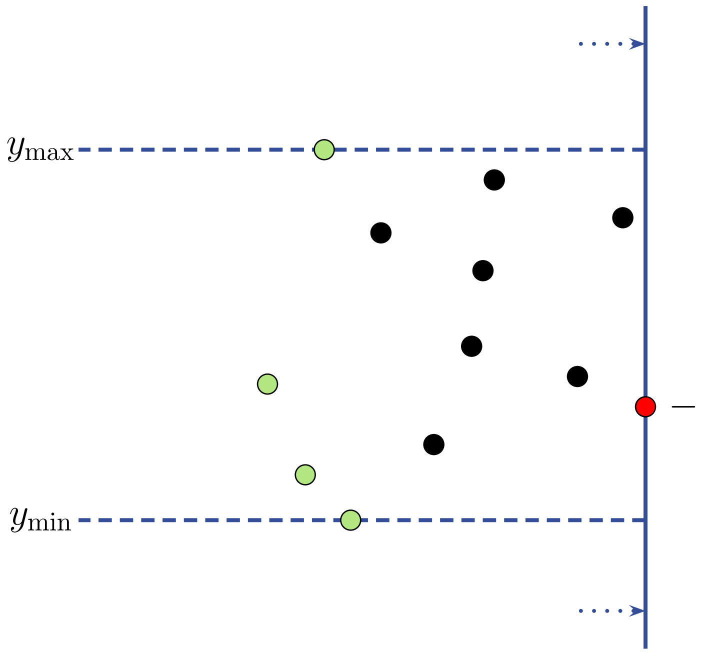
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



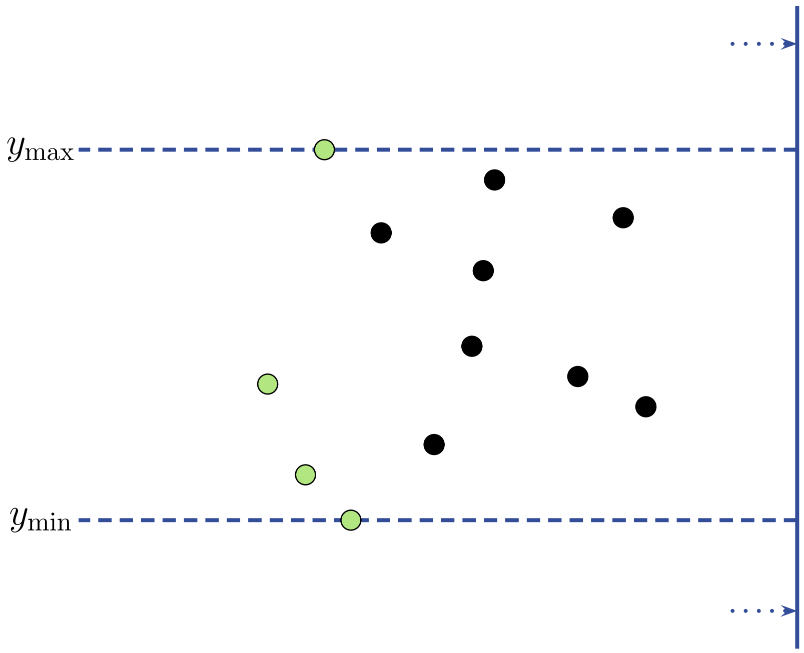
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



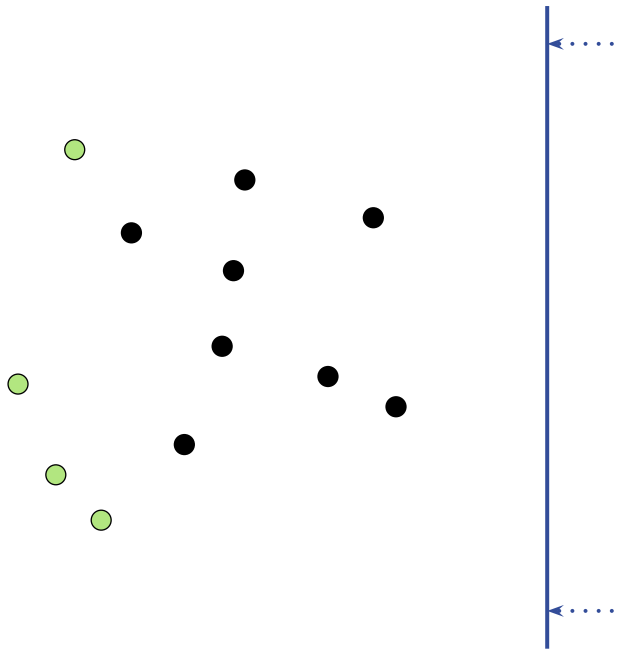
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



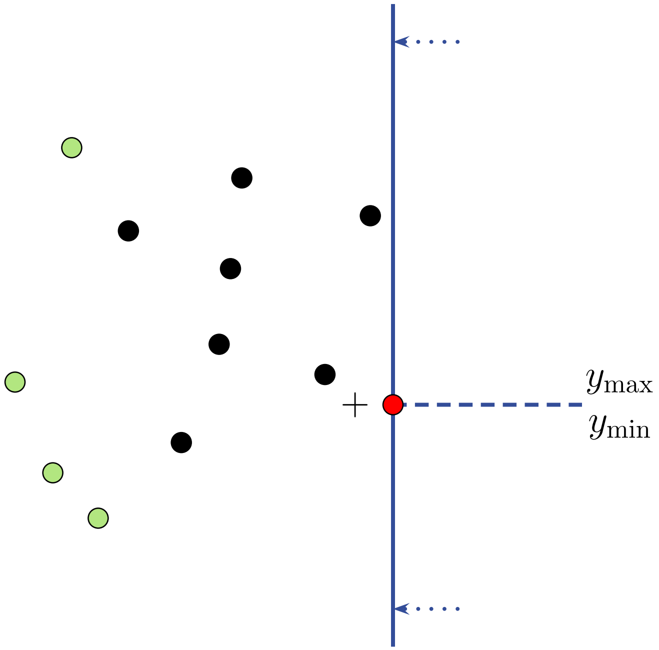
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



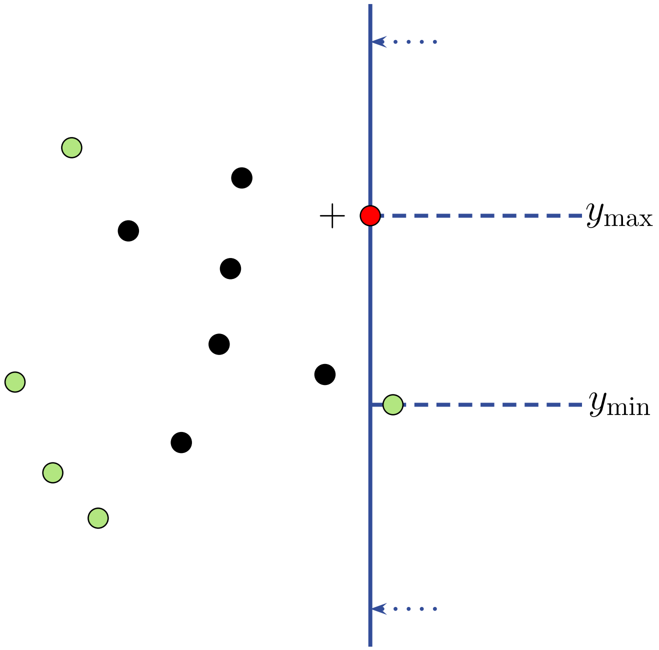
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



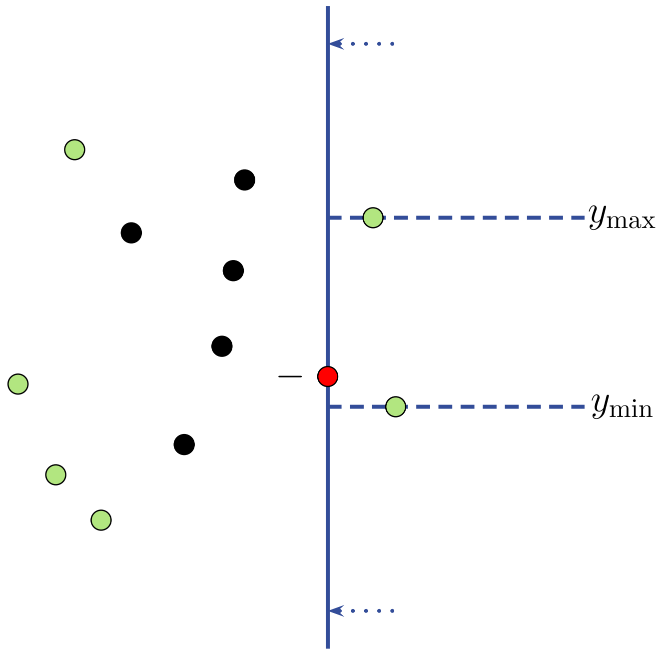
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



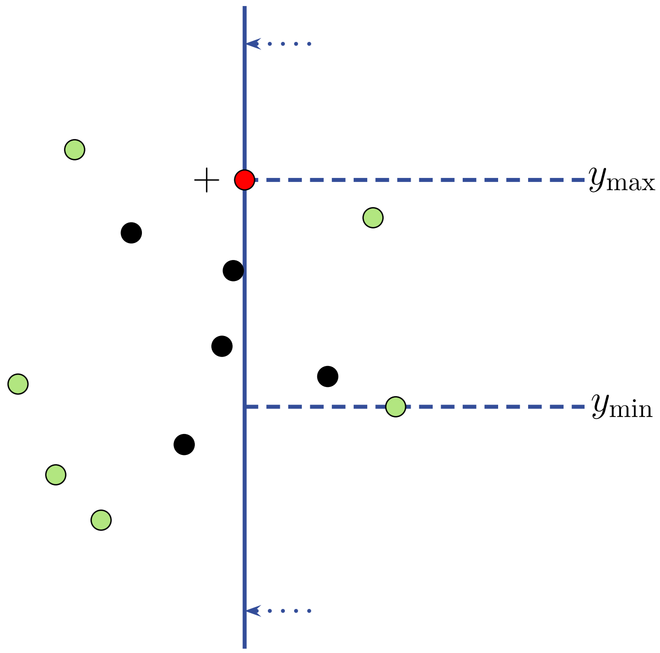
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



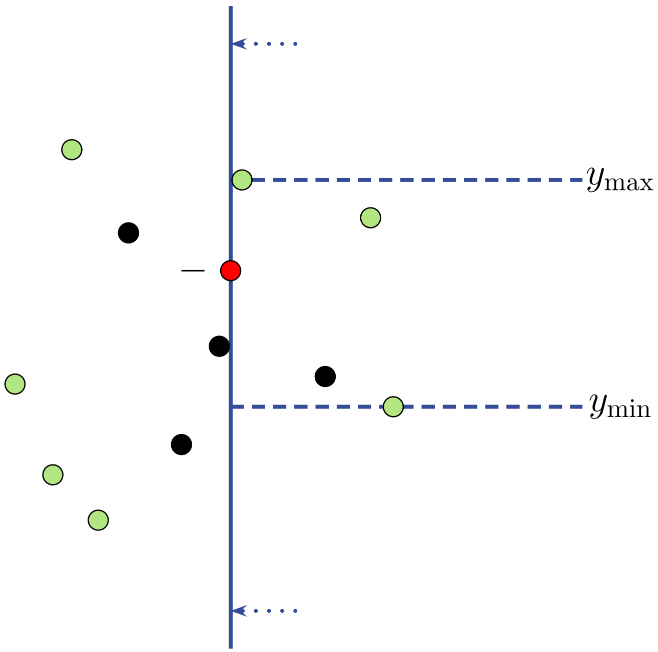
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



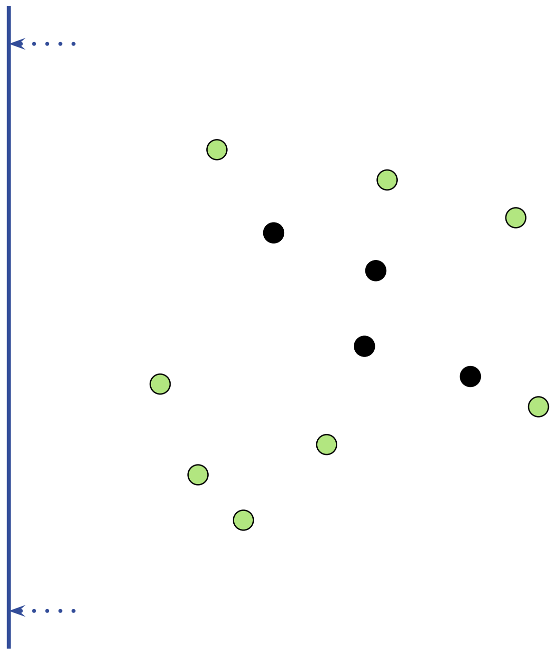
Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej $\boldsymbol{y}$ spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



Idea algorytmu /Kung, Luccio, Preparata 1975/

- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.



Idea algorytmu /Kung, Luccio, Preparata 1975/

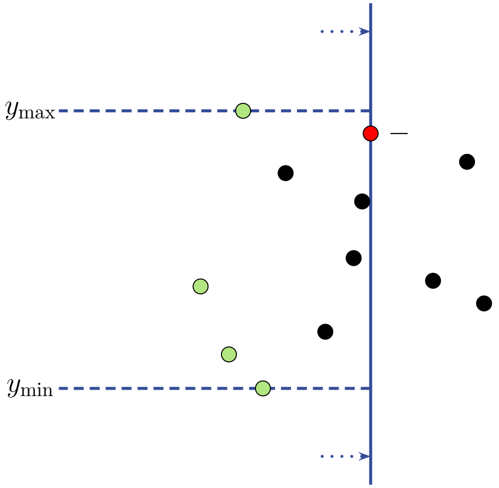
- Przesuwamy (wirtualną) miotłę z lewa na prawo.
- Punktami zdarzeń są punkty ze zbioru wejściowego w kolejności rosnących współrzędnych x.
- Statusem prostej jest minimalna i maksymalna wartość współrzędnej y spośród wszystkich punktów na lewo od lub na miotle
- Jeśli status prostej ulega zmianie, zgłaszane są odpowiednie punkty.
- W analogiczny sposób wszystkie punkty zostają przeglądnięte z prawa na lewo.

DetectMaxima(S)

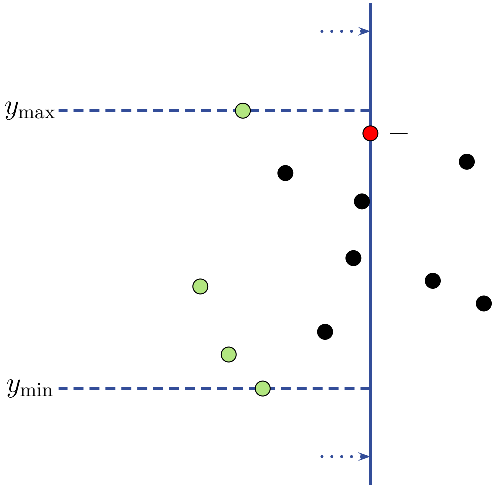
Wejście. Zbiór n punktów $S \subset \mathbb{R}^2$.

Wyjście. Zbiór D elementów maksymalnych.

```
Posortuj $S$ względem współrzędnej $x$, otrzymując punkty $p_1, p_2, \ldots, p_n$.
$D := \{p_1\}$; $y_{\max} := y_{\min} = \mathrm{y}(p_1)$.
for $i = 2$ to $n$ do
    if $\mathrm{y}(p_i) > y_{\max}$ then $D := D \cup \{p_i\}$; $y_{\max} := \mathrm{y}(p_i)$.
    if $\mathrm{y}(p_i) < y_{\min}$ then $D := D \cup \{p_i\}$; $y_{\min} := \mathrm{y}(p_i)$.
$D := D \cup \{p_n\}$; $y_{\max} = y_{\min} = \mathrm{y}(p_n)$.
for $i = n - 1$ to $2$ do
    if $\mathrm{y}(p_i) > y_{\max}$ then $D := D \cup \{p_i\}$; $y_{\max} := \mathrm{y}(p_i)$.
    if $\mathrm{y}(p_i) < y_{\min}$ then $D := D \cup \{p_i\}$; $y_{\min} := \mathrm{y}(p_i)$.
return $D$.
```



```
DETECTMAXIMA($S$)
Wejście. Zbiór $n$ punktów $S \subset \mathbb{R}^2$.
Wyjście. Zbiór $D$ elementów maksymalnych.
 1. Posortuj $S$ względem współrzędnej $x$, otrzymując punkty $p_1, p_2, \ldots, p_n$.
 2. $D := \{p_1\}$; $y_{\max} := y_{\min} = \mathrm{y}(p_1)$.
 3. for $i = 2$ to $n$ do
 4.     if $\mathrm{y}(p_i) > y_{\max}$ then $D := D \cup \{p_i\}$; $y_{\max} := \mathrm{y}(p_i)$.
 5.     if $\mathrm{y}(p_i) < y_{\min}$ then $D := D \cup \{p_i\}$; $y_{\min} := \mathrm{y}(p_i)$.
 6. $D := D \cup \{p_n\}$; $y_{\max} = y_{\min} = \mathrm{y}(p_n)$.
 7. for $i = n-1$ to $2$ do
 8.     if $\mathrm{y}(p_i) > y_{\max}$ then $D := D \cup \{p_i\}$; $y_{\max} := \mathrm{y}(p_i)$.
 9.     if $\mathrm{y}(p_i) < y_{\min}$ then $D := D \cup \{p_i\}$; $y_{\min} := \mathrm{y}(p_i)$.
10. return $D$.
```



Poprawność podejścia.

- ▶ Niezmiennikiem algorytmu w wierszach 2-5 jest „*Wszystkie maksima dla przypisań $(-,+)$ oraz $(-,-)$ na lewo od miotły M zostały zgłoszone*”, natomiast niezmiennikiem w wierszach 6-9 jest „*Wszystkie maksima dla przypisań $(+,+)$ oraz $(+,-)$ na prawo od miotły M zostały zgłoszone*”.

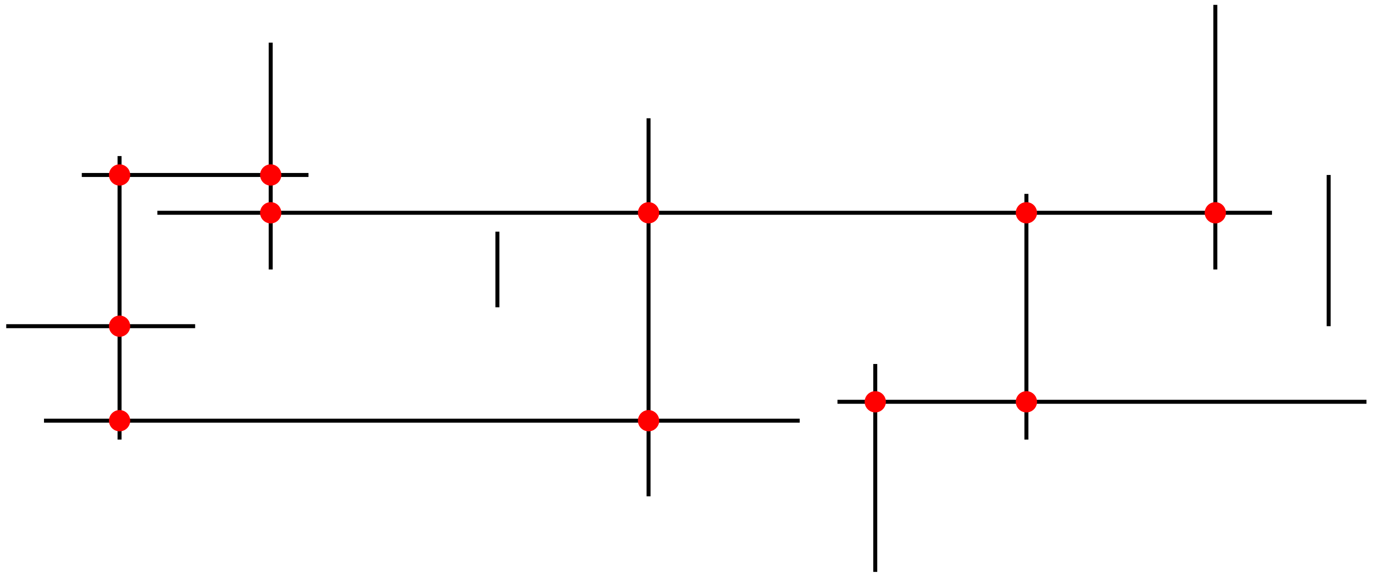
Złożoność obliczeniowa algorytmu.

- ▶ Mając na uwadze wstępne posortowanie po odciętej: czas rzędu $O(n \log n)$.
- ▶ Zapamiętanie $y_{\min}$ oraz $y_{\max}$: pamięć rzędu $O(1)$.

6.2 Punkty przecięć

Dla danego zbioru poziomych i pionowych odcinków $S = H \cup V$ na płaszczyźnie wyznacz wszystkie punkty przecięć odcinków z S.

M. Smid
Computing intersections in a set of horizontal and vertical line segments
`http://people.scs.carleton.ca/~michiel/teaching.html` [23.03.2015]

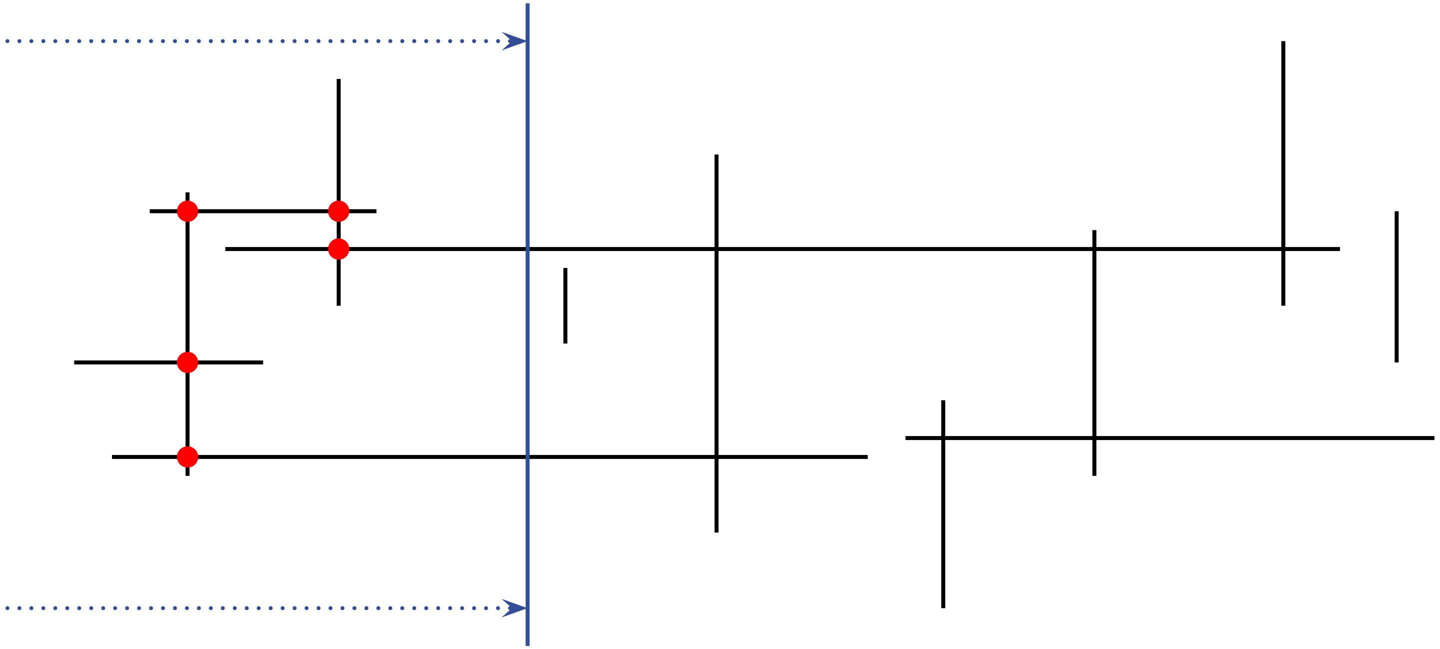


Założenie. Końce odcinków poziomych oraz odcinki pionowe mają różne odcięte.

6.2 PUNKTY PRZECIĘĆ

Dla danego zbioru poziomych i pionowych odcinków $S = H \cup V$ na płaszczyźnie wyznacz wszystkie punkty przecięć odcinków z S.

Niezmiennik. Wszystkie przecięcia na lewo od M zostały zgłoszone.



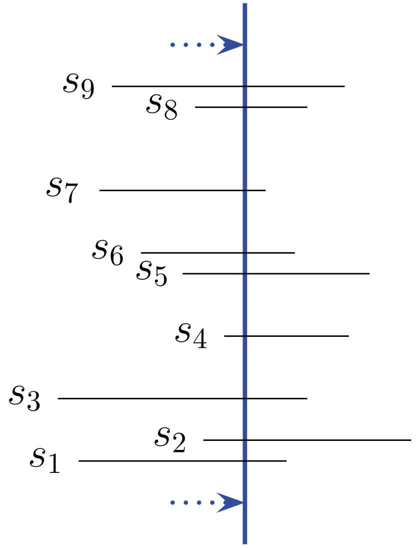
6.2 PUNKTY PRZECIĘĆ

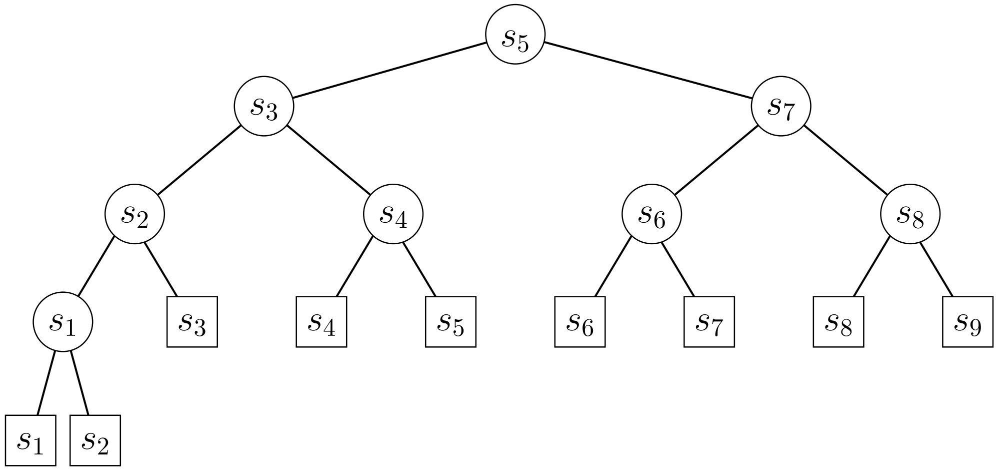
Dla danego zbioru poziomych i pionowych odcinków $S = H \cup V$ na płaszczyźnie wyznacz wszystkie punkty przecięć odcinków z S.

Niezmiennik. Wszystkie przecięcia na lewo od M zostały zgłoszone.

Status miotły. Poziome odcinki przecinane przez miotłę.

Status prostej przechowywany jest w zrównoważonym drzewie przeszukiwań; odcinki (w liściach) posortowane są względem współrzędnej y.





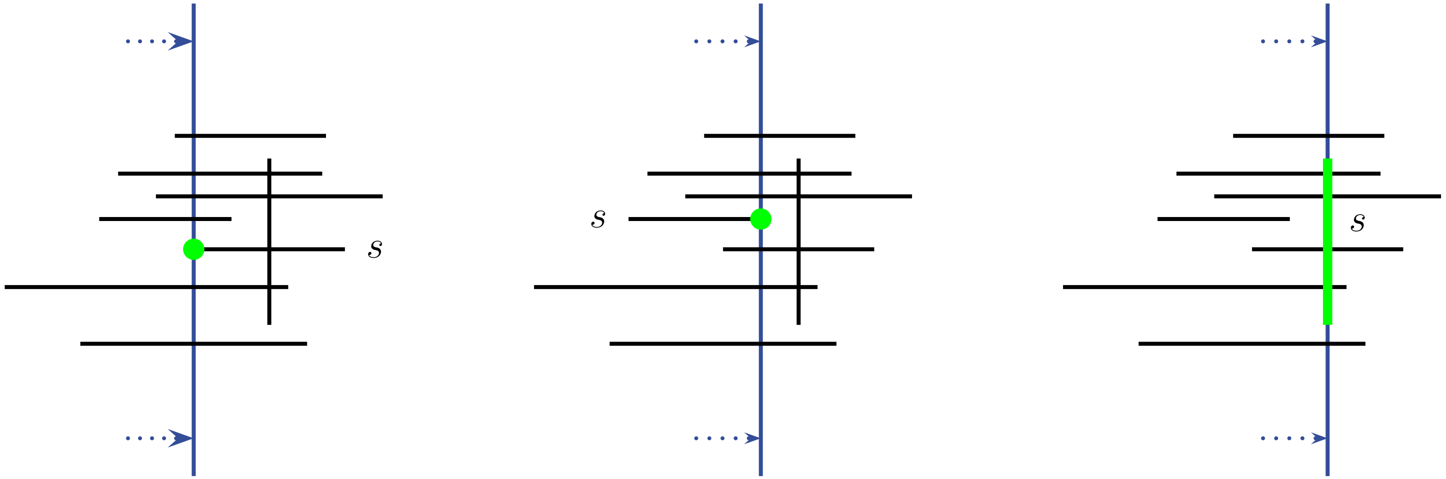
6.2 PUNKTY PRZECIĘĆ

Dla danego zbioru poziomych i pionowych odcinków $S = H \cup V$ na płaszczyźnie wyznacz wszystkie punkty przecięć odcinków z S.

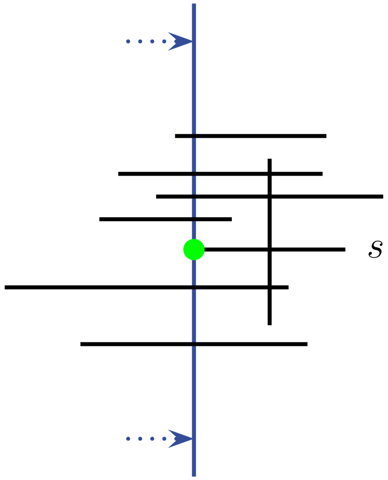
Niezmiennik. Wszystkie przecięcia na lewo od M zostały zgłoszone.

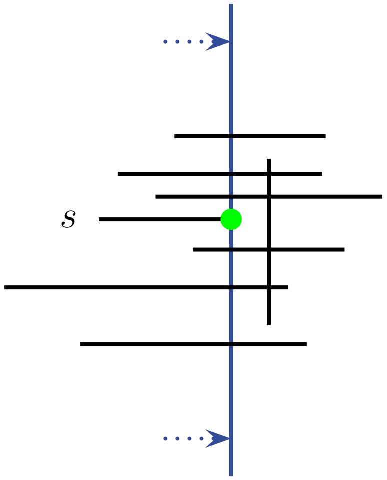
Status miotły. Poziome odcinki przecinane przez miotłę.

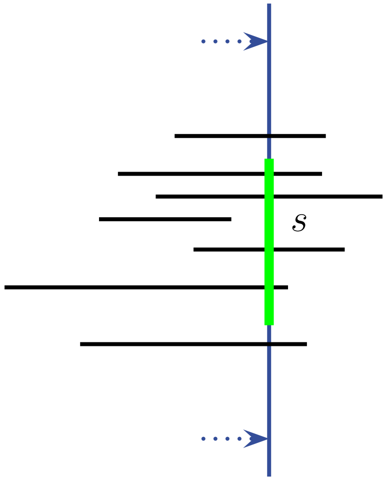
Status prostej przechowywany jest w zrównoważonym drzewie przeszukiwań; odcinki (w liściach) posortowane są względem współrzędnej y.



Punkty zdarzeń. Końce odcinków poziomych i odcinki pionowe.

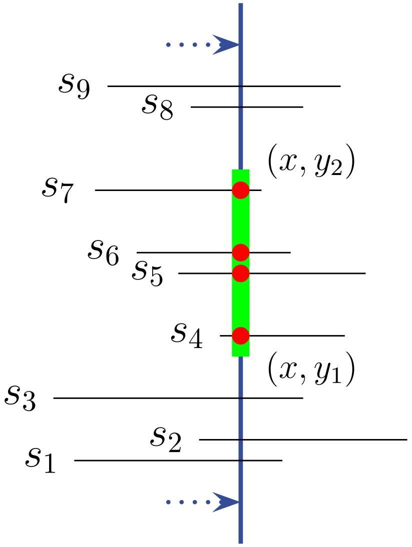


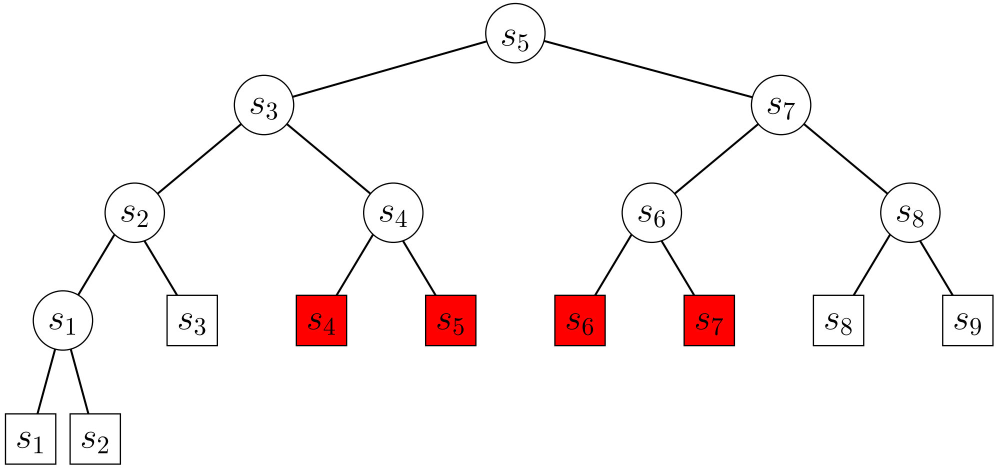




Zmiana statusu. Możliwe są trzy sytuacje:

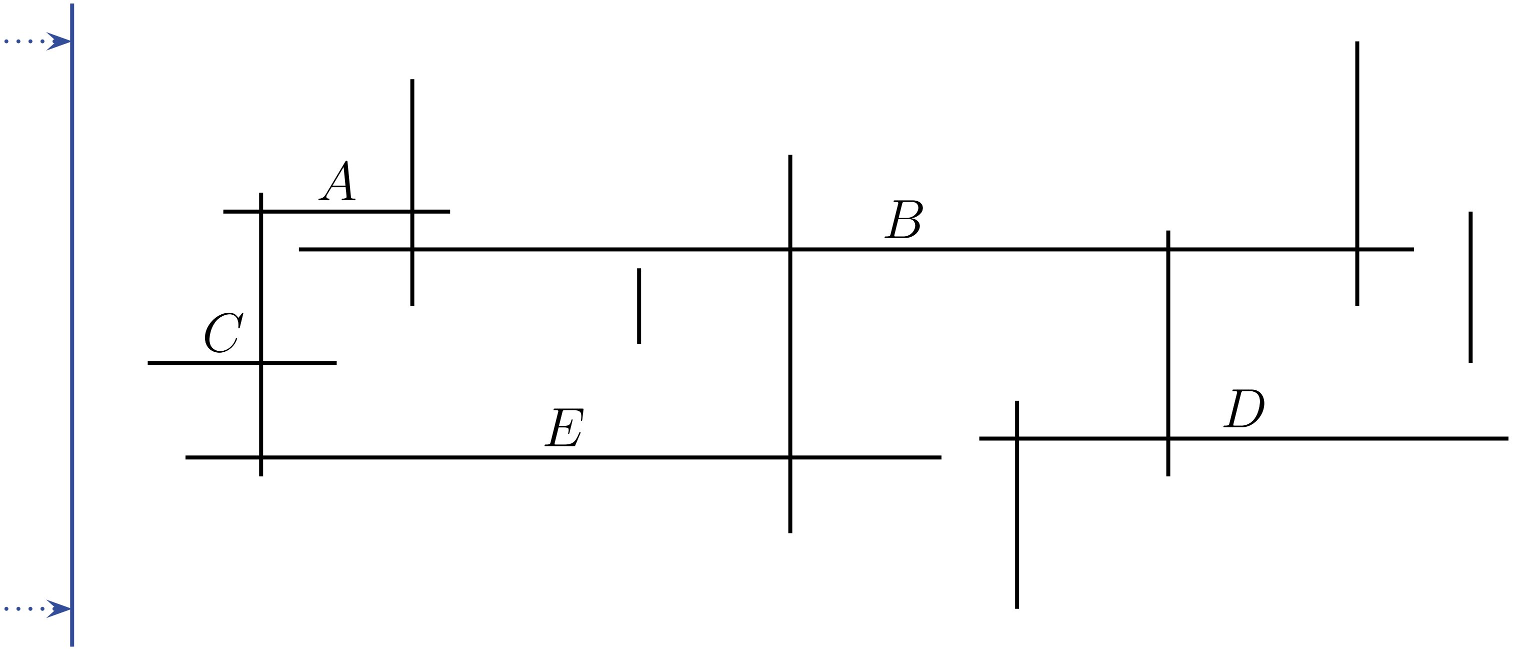
1. Miotła napotyka lewy koniec poziomego odcinka s.
   Miotła zaczyna przecinać s, a zatem odcinek s musi zostać dodany do (zrównoważonego drzewa) statusu miotły.
2. Miotła napotyka prawy koniec poziomego odcinka s.
   Miotła przestaje przecinać s, a zatem odcinek s musi zostać usunięty ze (zrównoważonego drzewa) statusu miotły.
3. Miotła napotyka pionowy odcinek $s = [(x, y_1), (x, y_2)]$.
   Zauważmy, że poziome odcinki przecinające s muszą w tej chwili przecinać także miotłę M, a zatem, aby wyznaczyć wszystkie te przecięcia, wystarczy w zrównoważonym drzewie statusu prostej wyznaczyć te poziome odcinki, których współrzędne y należą do przedziału $[y_1, y_2]$.

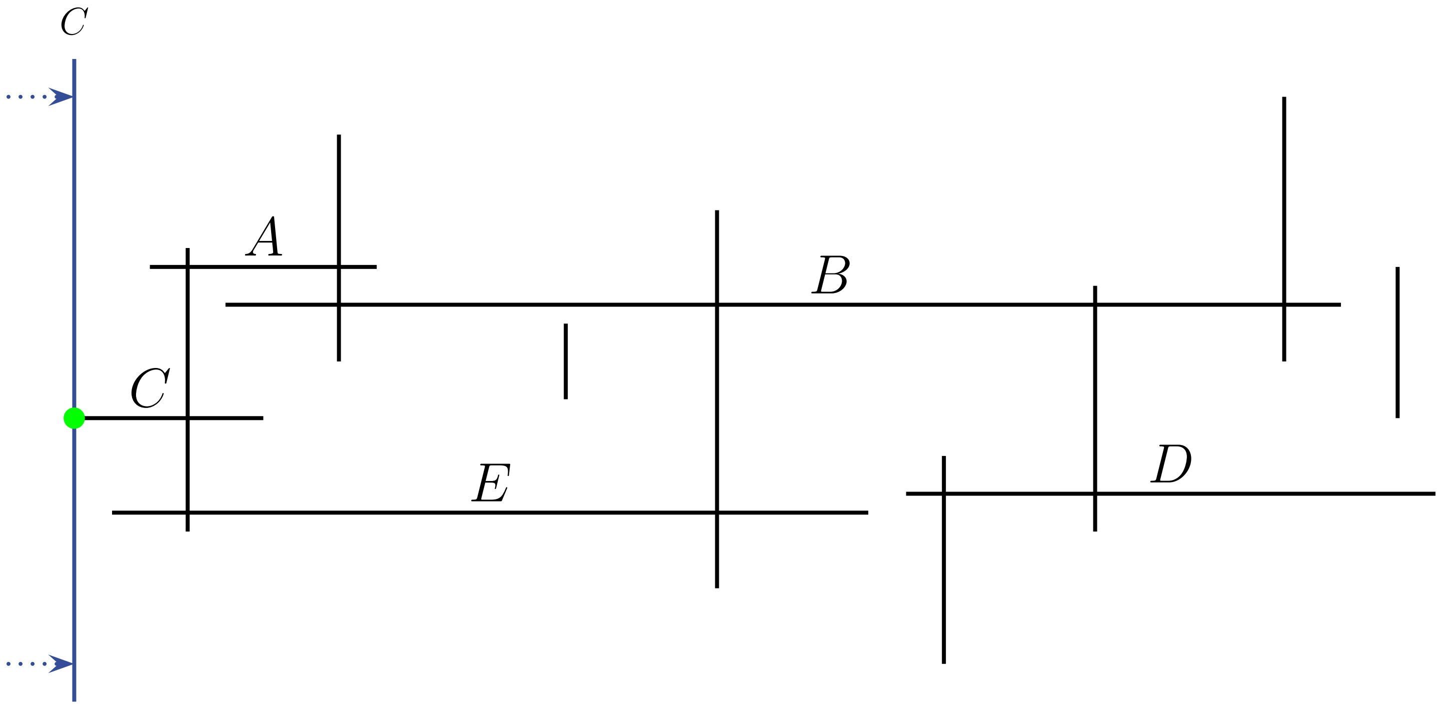


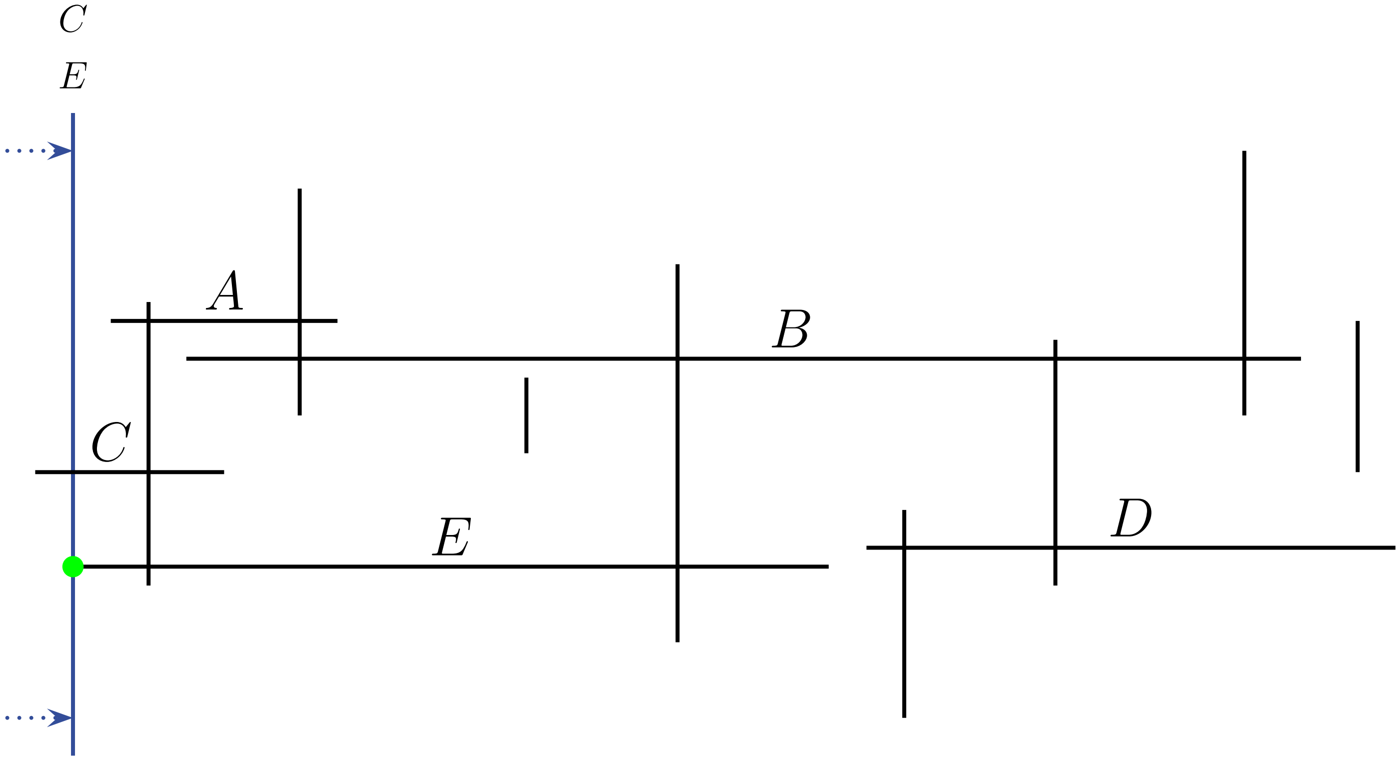


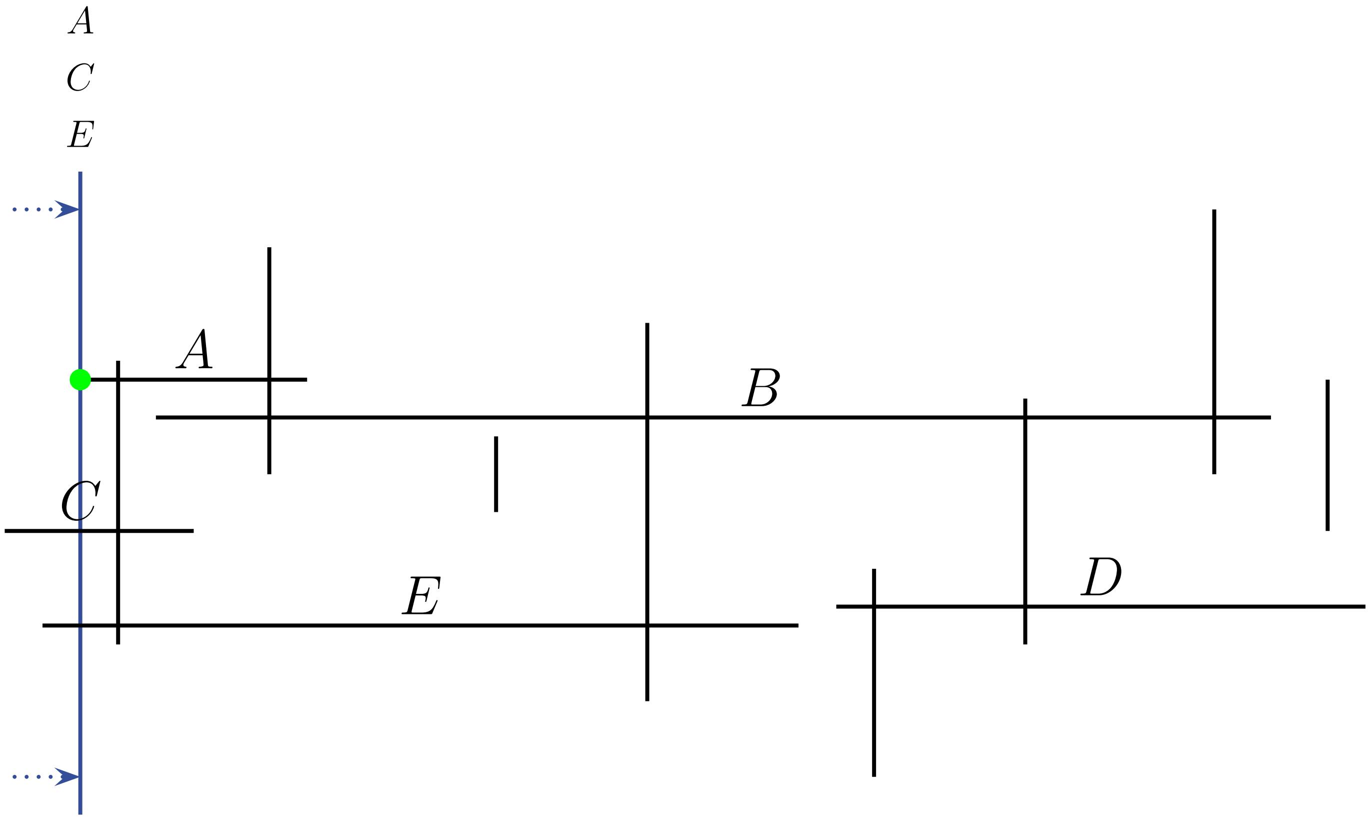
Zmiana statusu. Możliwe są trzy sytuacje:

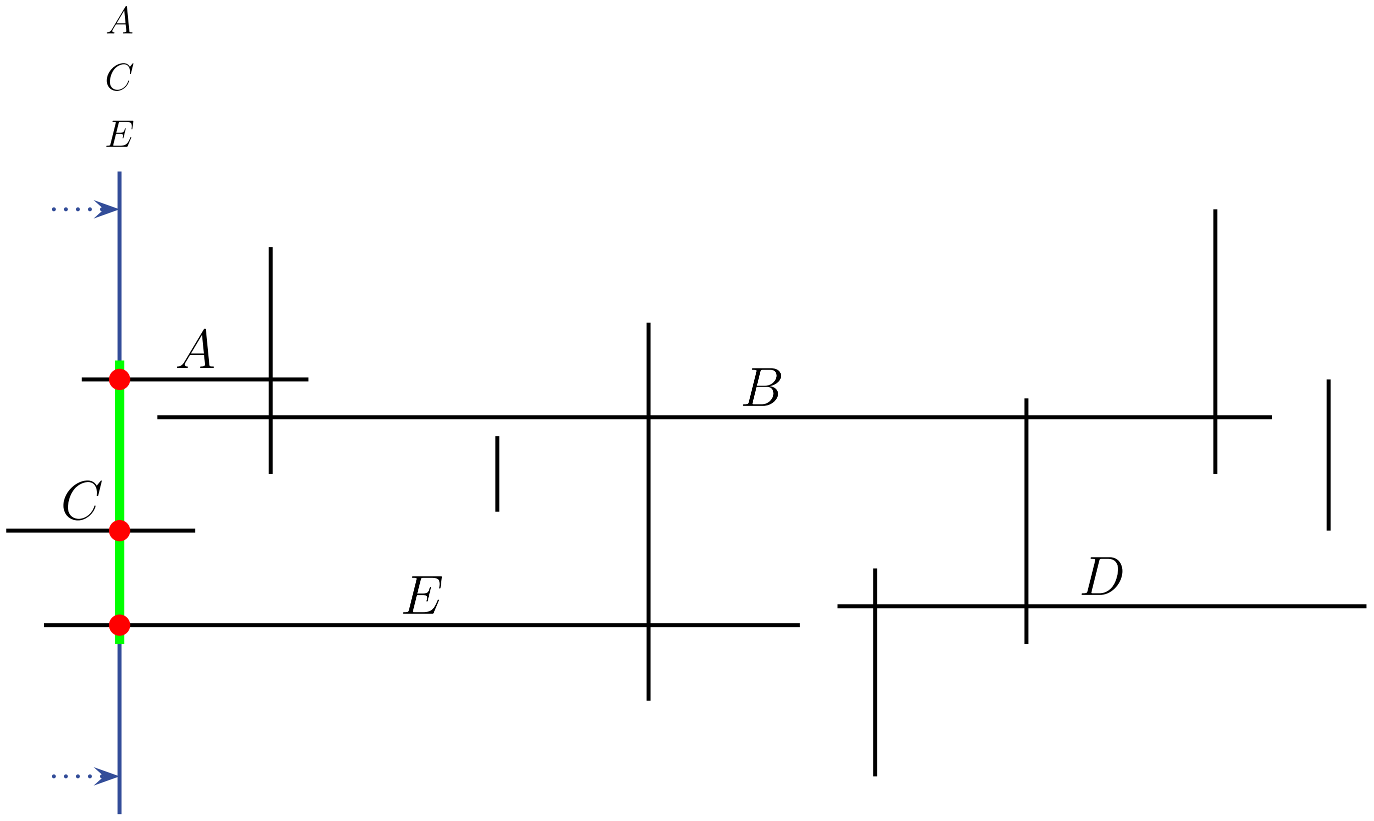
1. Miotła napotyka lewy koniec poziomego odcinka s.
   Miotła zaczyna przecinać s, a zatem odcinek s musi zostać dodany do (zrównoważonego drzewa) statusu miotły.
2. Miotła napotyka prawy koniec poziomego odcinka s.
   Miotła przestaje przecinać s, a zatem odcinek s musi zostać usunięty ze (zrównoważonego drzewa) statusu miotły.
3. Miotła napotyka pionowy odcinek $s = [(x, y_1), (x, y_2)]$.
   Zauważmy, że poziome odcinki przecinające s muszą w tej chwili przecinać także miotłę M, a zatem, aby wyznaczyć wszystkie te przecięcia, wystarczy w zrównoważonym drzewie statusu prostej wyznaczyć te poziome odcinki, których współrzędne y należą do przedziału $[y_1, y_2]$.

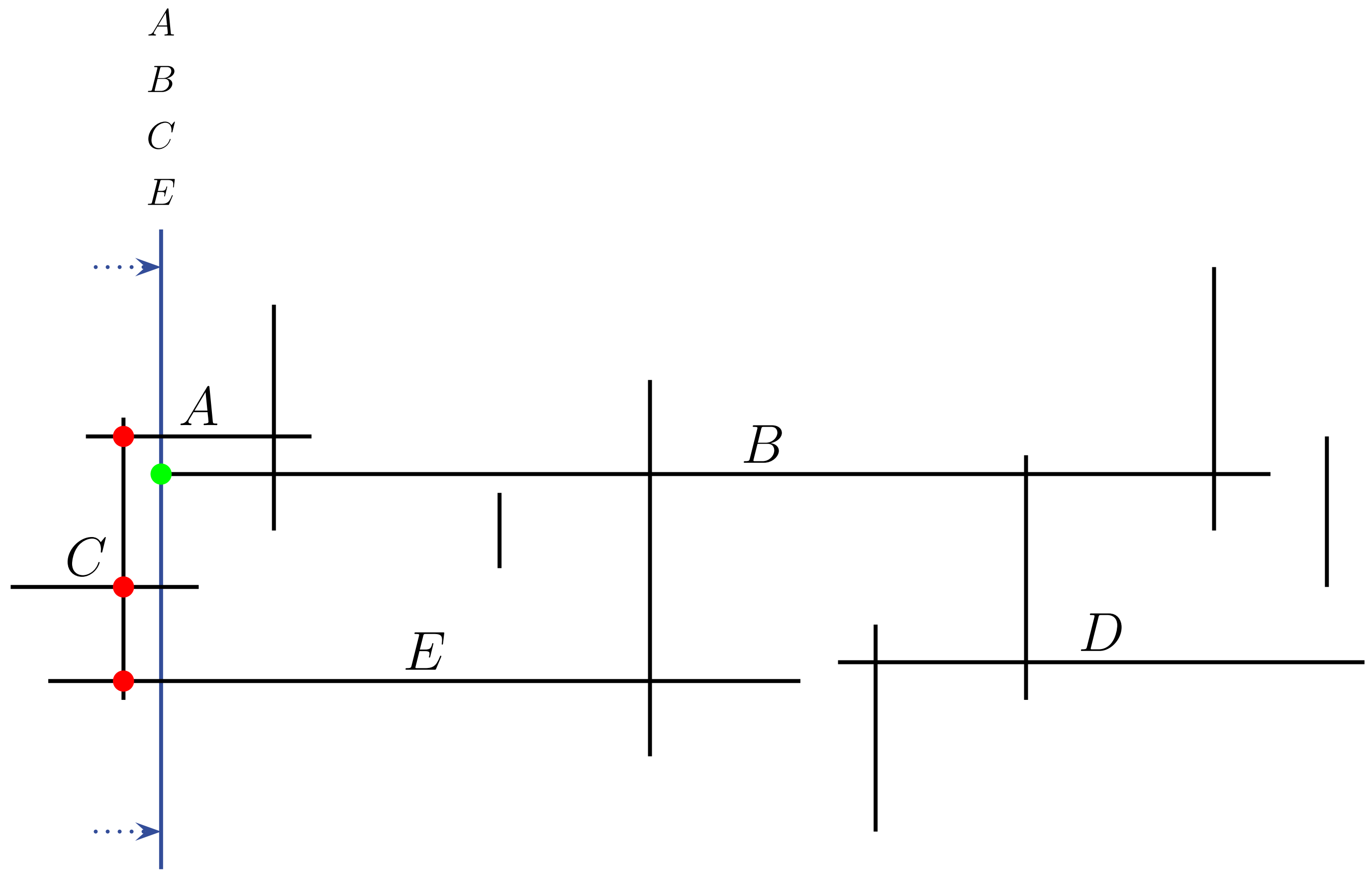

A
B
C
D
E

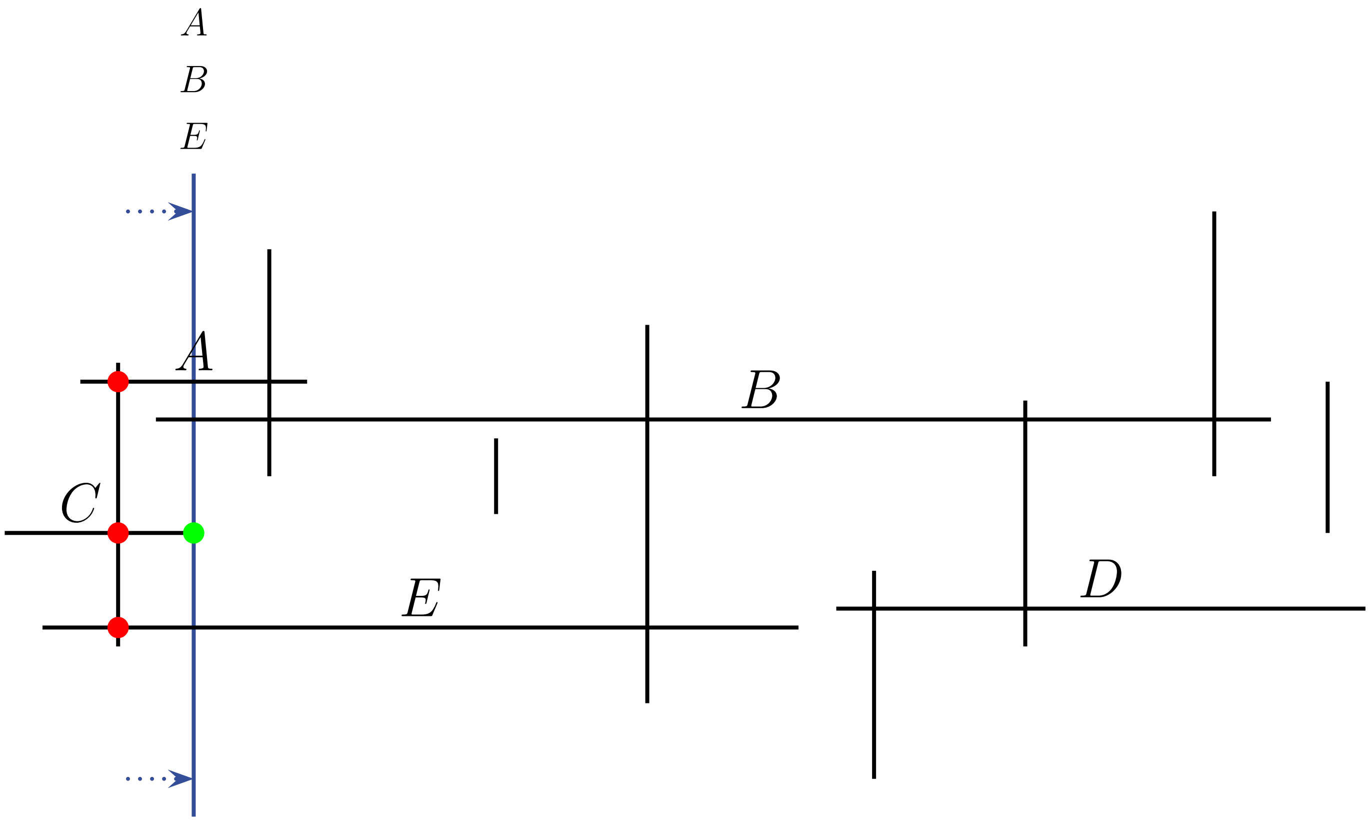

C
A
B
C
E
D

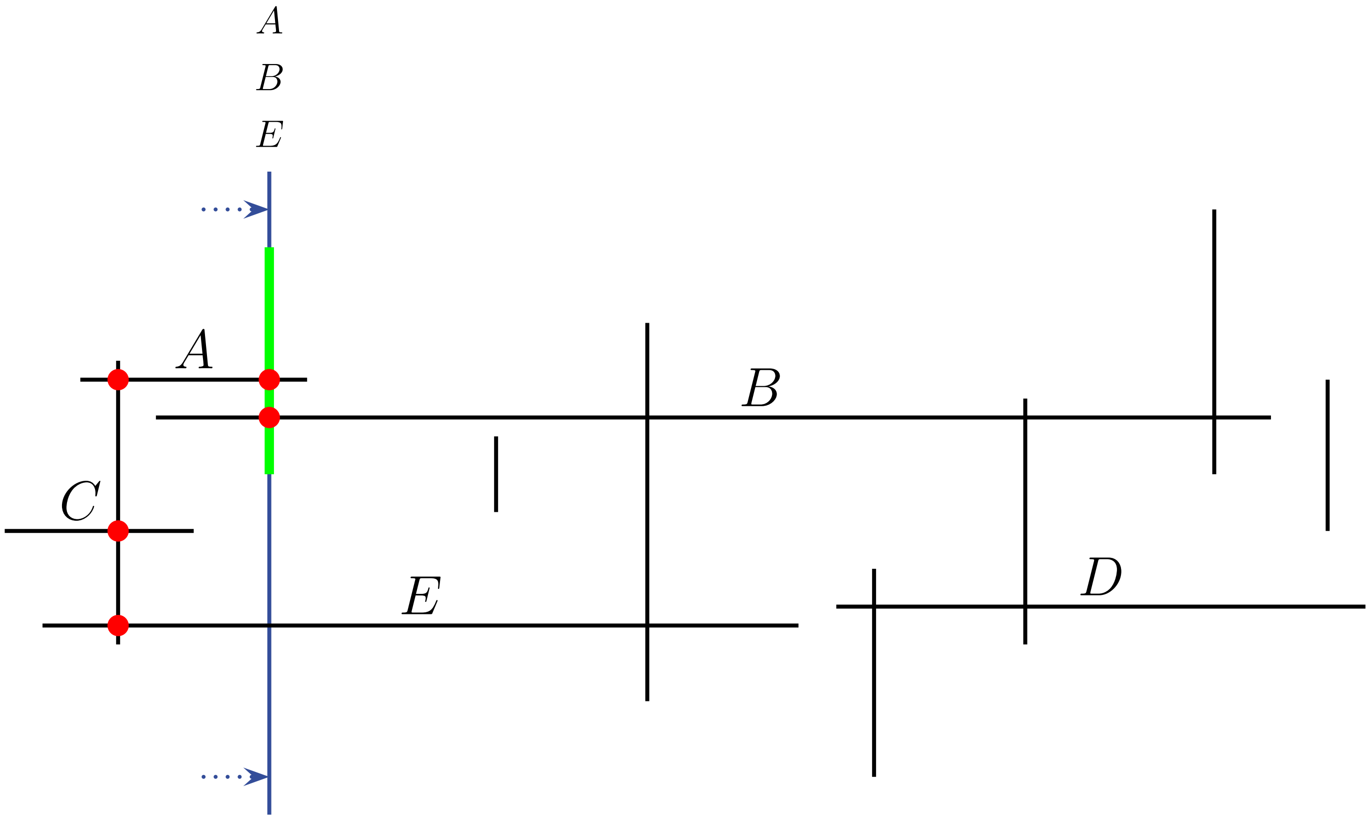

C
E
A
B
C
E
D

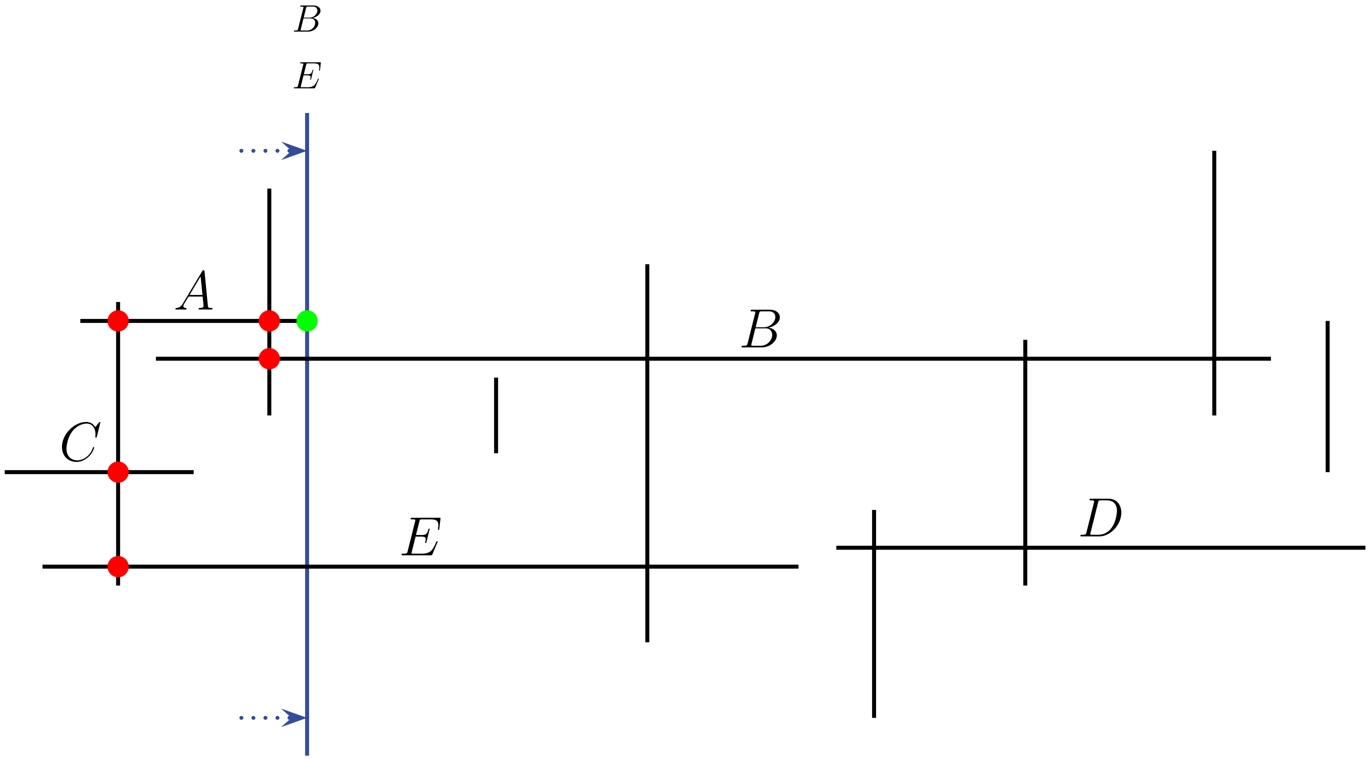

A
C
E
A
B
C
E
D

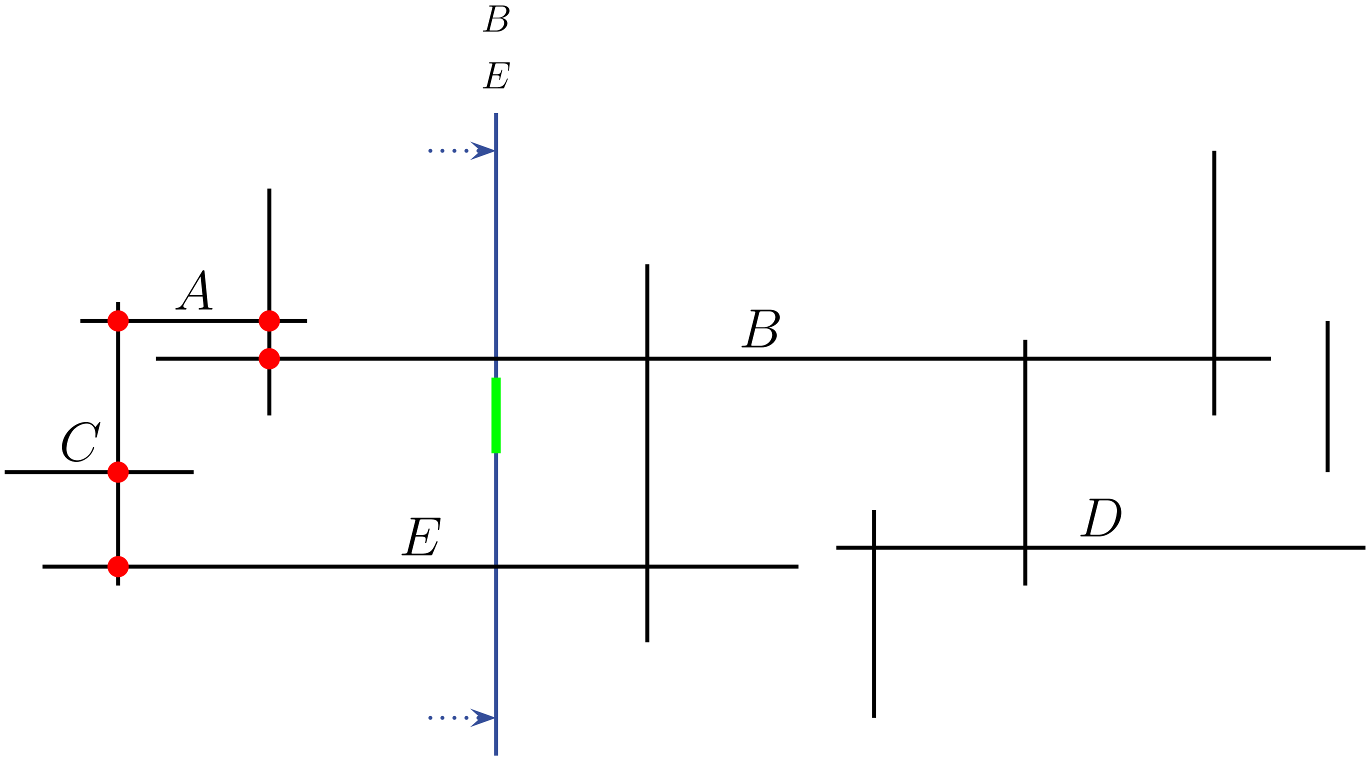

A
C
E
A
B
C
E
D

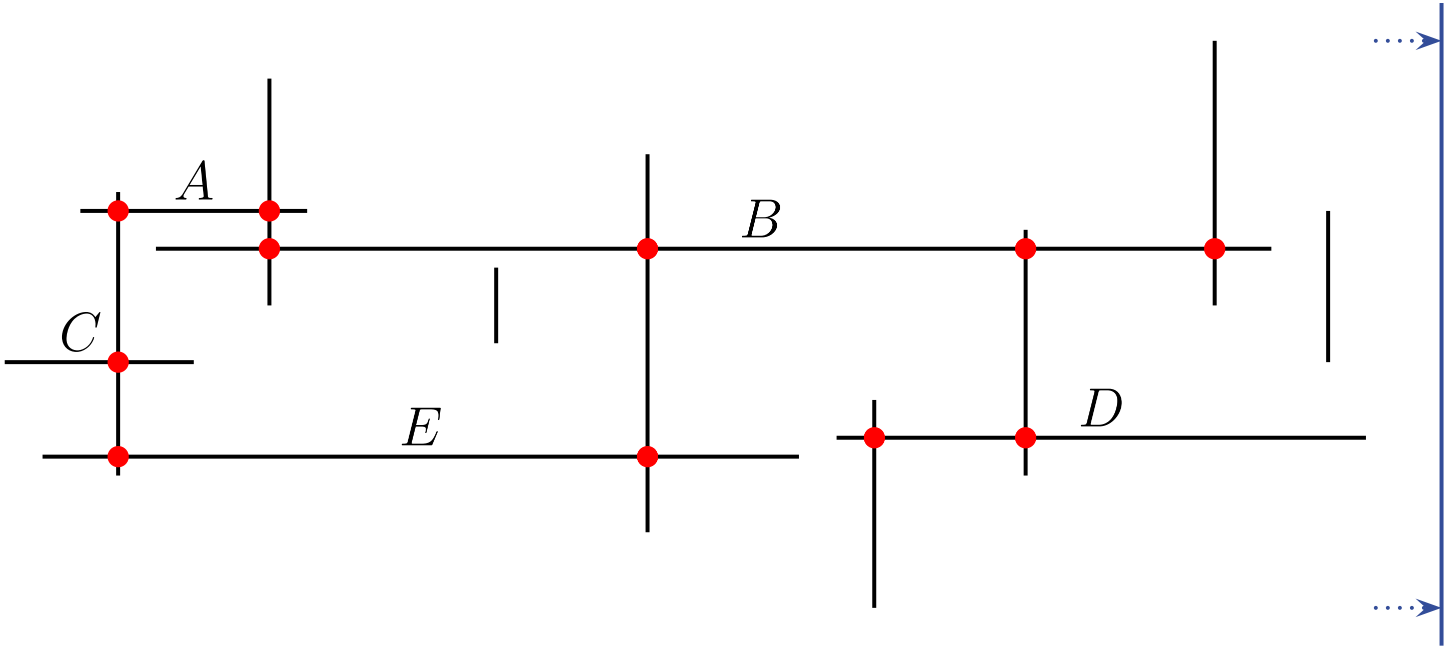

A
B
C
E
A
B
C
E
D


A
B
E
A
B
C
E
D


A
B
E
A
B
C
E
D

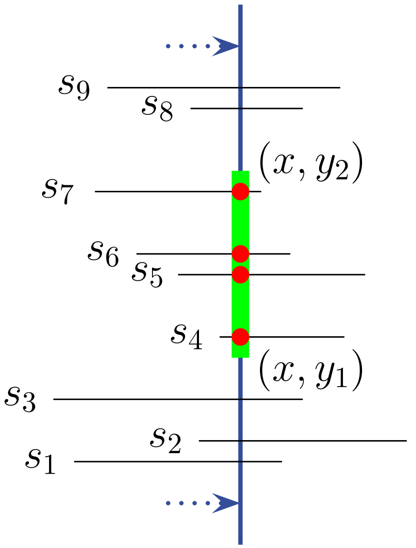

B
E
A
B
C
E
D


B
E
A
B
C
E
D

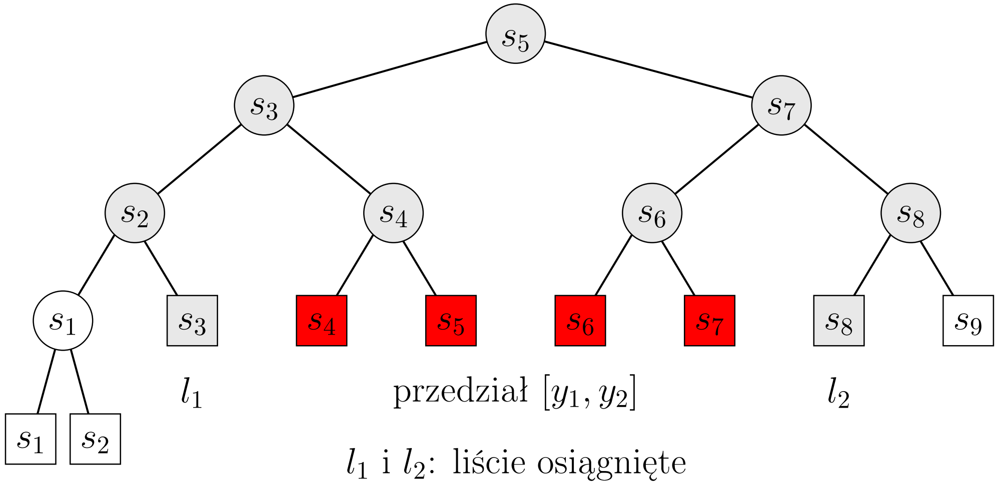

A
B
C
D
E

Algorytm SweepCrossings

```
0. Posortuj zbiór $S^\bullet := \overline{H} \cup V$ względem odciętej, gdzie $\overline{H}$ jest zbiorem końców
   poziomych odcinków z $H$.                                   /Wstępne przetwarzanie./
1. Niech $\mathcal{T}$ będzie pustym zrównoważonym drzewem binarnym.
                                        /Drzewo $\mathcal{T}$ przechowuje status miotły./
2. while $S^\bullet \neq \emptyset$ do
  2.1 Usuń pierwszy element $v \in S^\bullet$.
  2.2 if ($v$ jest lewym końcem odcinka $s$) then dodaj $s$ do $\mathcal{T}$.
  2.3   else if ($v$ jest prawym końcem odcinka $s$) then usuń $s$ z $\mathcal{T}$.
  2.4      else 1DRangeQuery($\mathcal{T}$, $[y_1, y_2]$), gdzie $v = [(x, y_1), (x, y_2)]$.
```



1DRangeQuery($\mathcal{T}$, $[y_1, y_2]$)



l_1 i l_2: liście osiągnięte
w poszukiwaniu y_1 i odpowiednio y_2

Algorytm SweepCrossings

```
0. Posortuj zbiór $S^\bullet := \overline{H} \cup V$ względem odciętej, gdzie $\overline{H}$ jest zbiorem końców
   poziomych odcinków z $H$.                                   /Wstępne przetwarzanie./
1. Niech $\mathcal{T}$ będzie pustym zrównoważonym drzewem binarnym.
                                               /Drzewo $\mathcal{T}$ przechowuje status miotły./
2. while $S^\bullet \neq \emptyset$ do
   2.1 Usuń pierwszy element $v \in S^\bullet$.
   2.2 if ($v$ jest lewym końcem odcinka $s$) then dodaj $s$ do $\mathcal{T}$.
   2.3   else if ($v$ jest prawym końcem odcinka $s$) then usuń $s$ z $\mathcal{T}$.
   2.4     else 1DRangeQuery($\mathcal{T}$, $[y_1, y_2]$), gdzie $v = [(x, y_1), (x, y_2)]$.
```

Analiza poprawności podejścia

- Zachowywanie niezmiennika wynika z konstrukcji algorytmu, tj. wiersza 2.4.

Analiza złożoności obliczeniowej algorytmu

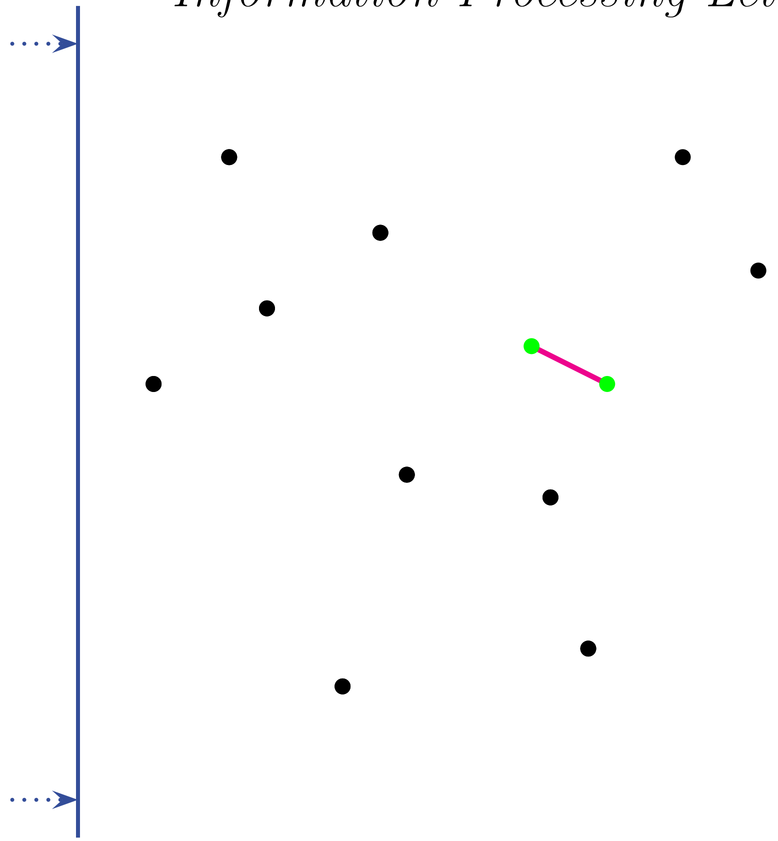
- Przetwarzanie wstępne wymaga czasu rzędu $O(n \log n)$.
- Czas działania algorytmu: $O(n \log n) + A(n)$, gdzie $A(n)$ jest czasem zużytym przez wszystkie wywołania 1DRangeQuery.
- Status miotły (zrównoważone drzewo binarne): pamięć rzędu $O(n)$.

Twierdzenie 6.2. (Smid 2003) Algorytm SweepCrossings wyznacza wszystkie przecięcia w czasie rzędu $O(n \log n + k)$, gdzie n jest liczbą odcinków, a k jest liczbą przecięć; zużycie pamięci jest rzędu $O(n)$.

6.3 Para najbliższych punktów

Dla danego zbioru punktów $S \subset \mathcal{E}^2$ wyznaczyć parę najbliższych punktów.

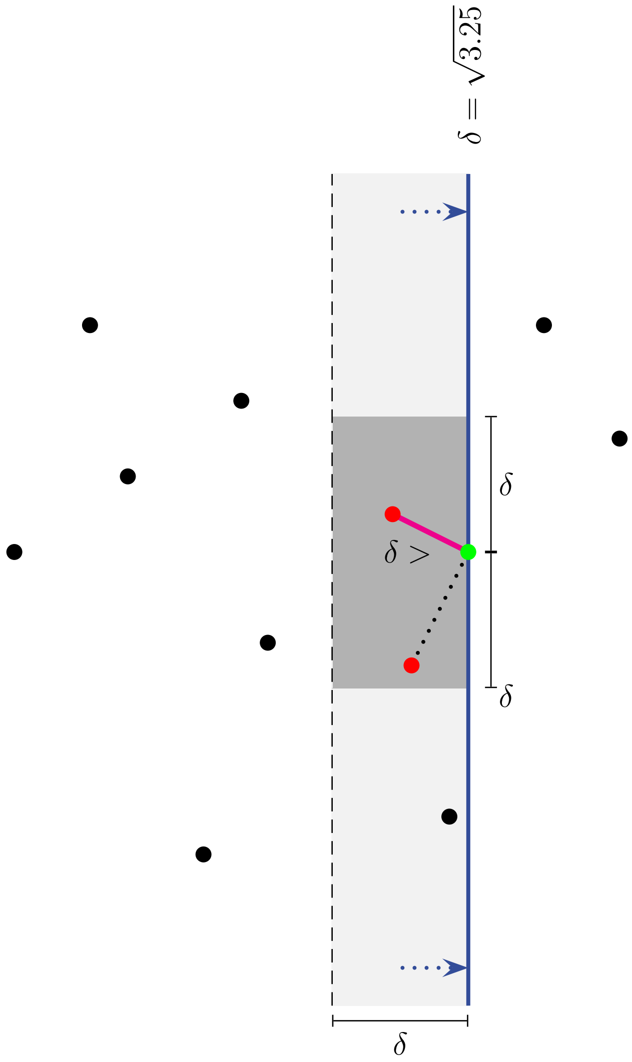
K. Hinrichs, J. Nievergelt, P. Schorn (???)
Plane-sweep solves the closest pair problem elegantly
Information Processing Letters 26(5), 255-261 (1988)



Założenie. Odcięte punktów wejściowych są różne.

Idea algorytmu

- ▶ Miotła M przesuwa się z lewo na prawo pamiętając najmniejszą znalezioną do tej pory odległość δ.
- ▶ Status miotły stanowi zbiór punktów na lewo od miotły w odległości co najwyżej δ i zmienia się on przy napotkaniu kolejnych punktów.
- ▶ Napotykając nowy punkt $p = (x, y)$, sprawdzana jest odległość p do punktów ze statusu miotły leżących „nie za daleko” od p.



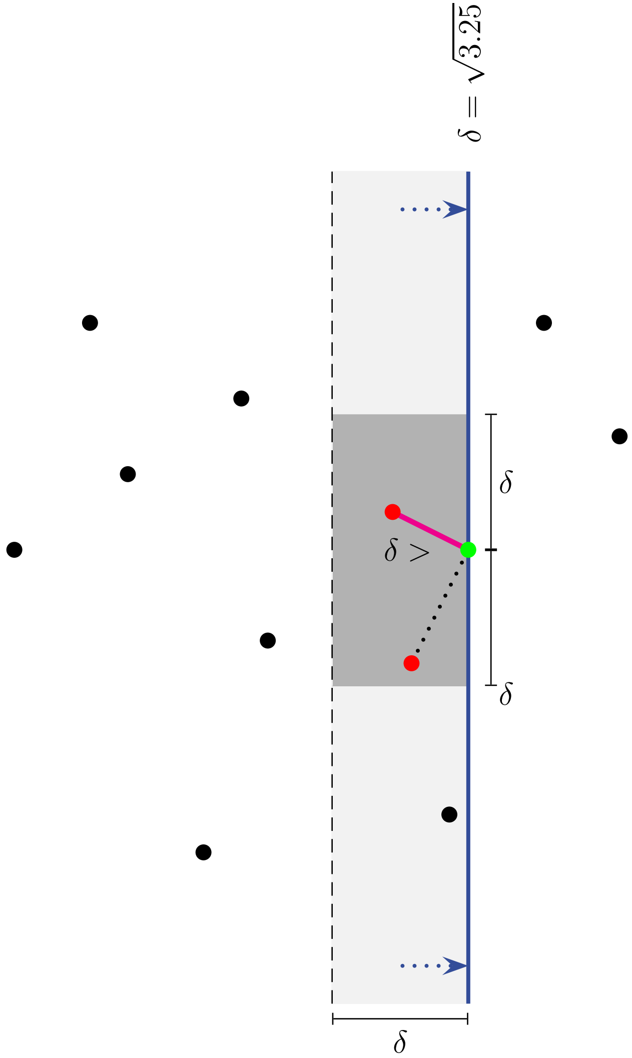
Niezmiennik.
Wartość δ jest najmniejszą odległością spośród punktów na lewo od miotły M.

> Jeśli M przesunie się po całym obszarze, niezmiennik zagwarantuje, że wyznaczona zostanie para najbliższych punktów.

Status miotły.
Punkty na lewo od miotły i w jej δ-otoczeniu.

> Na miotłę można patrzeć jak na pionowy pasek o szerokości δ ograniczony przez dwie pionowe proste. Status miotły przechowywany jest w zrównoważonym drzewie przeszukiwań; punkty posortowane są względem współrzędnej y.

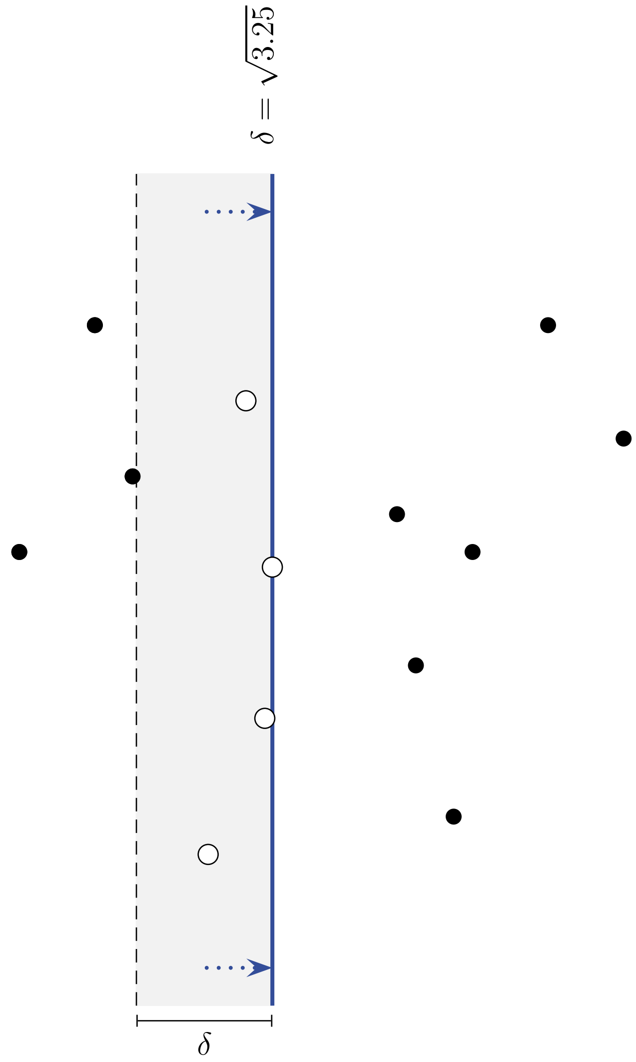
Punkty zdarzeń.
Punkty z S posortowane względem odciętej.



Zmiana statusu.

Miotła napotyka nowy punkt $p = (x, y)$.

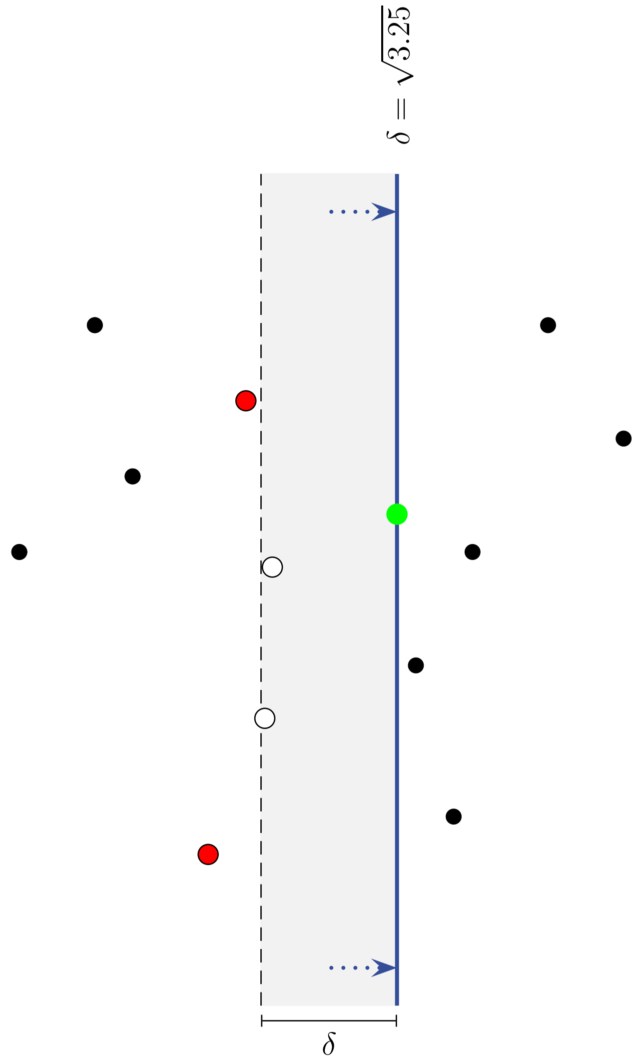
1. Usuwamy ze statusu miotły wszystkie te punkty, które nie leżą w jej δ-otoczeniu.
2. Sprawdzana jest odległość p do punktów ze statusu miotły, których rzędna mieści się w przedziale $[y - \delta, y + \delta]$.
3. Jeśli któraś odległości wyznaczonych w (2) jest mniejsza od dotychczasowej δ, to δ ulega zmianie na najmniejszą z nich. Zapamiętujemy również parę punktów realizującą tę najmniejszą odległość.
4. Wstawiamy p do statusu miotły.



Zmiana statusu.

Miotła napotyka nowy punkt $p = (x, y)$.

1. Usuwamy ze statusu miotły wszystkie te punkty, które nie leżą w jej δ-otoczeniu.
2. Sprawdzana jest odległość p do punktów ze statusu miotły, których rzędna mieści się w przedziale $[y - \delta, y + \delta]$.
3. Jeśli któraś odległości wyznaczonych w (2) jest mniejsza od dotychczasowej δ, to δ ulega zmianie na najmniejszą z nich. Zapamiętujemy również parę punktów realizującą tę najmniejszą odległość.
4. Wstawiamy p do statusu miotły.



Zmiana statusu.

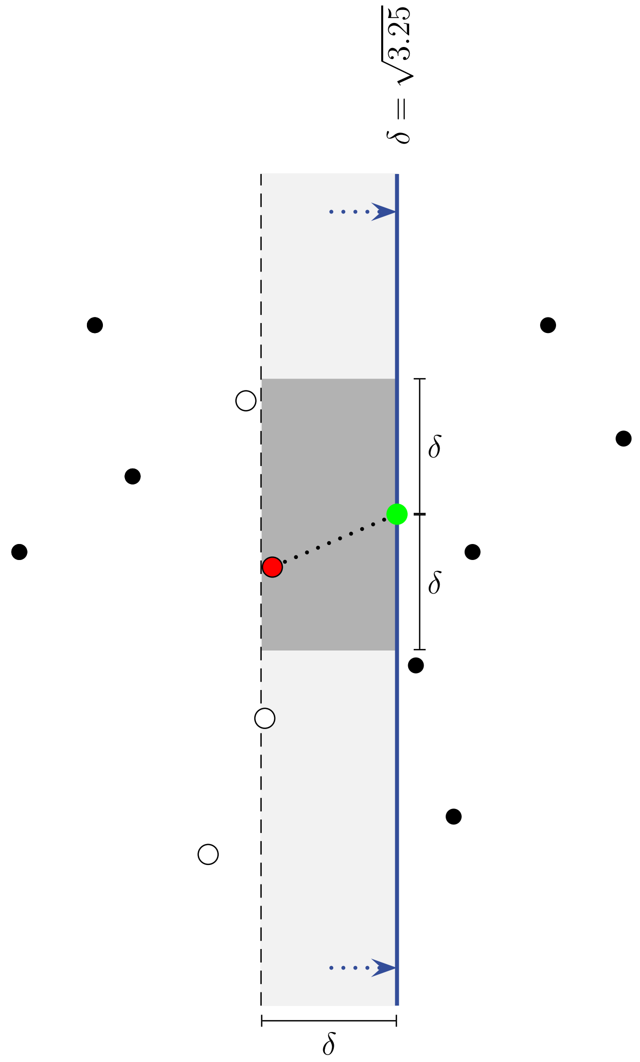
Miotła napotyka nowy punkt $p = (x, y)$.

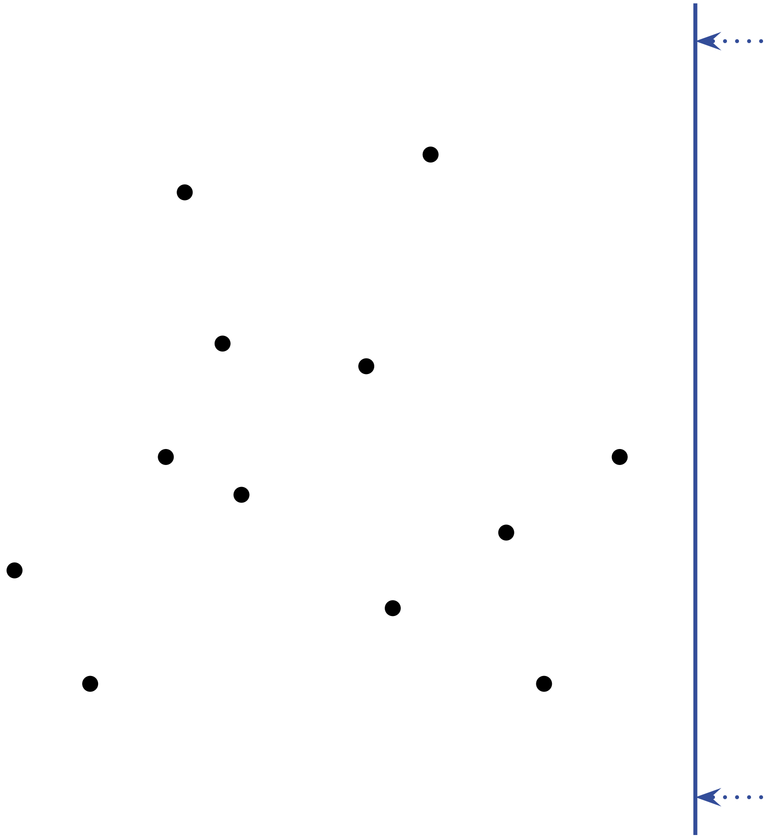
1. Usuwamy ze statusu miotły wszystkie te punkty, które nie leżą w jej δ-otoczeniu.

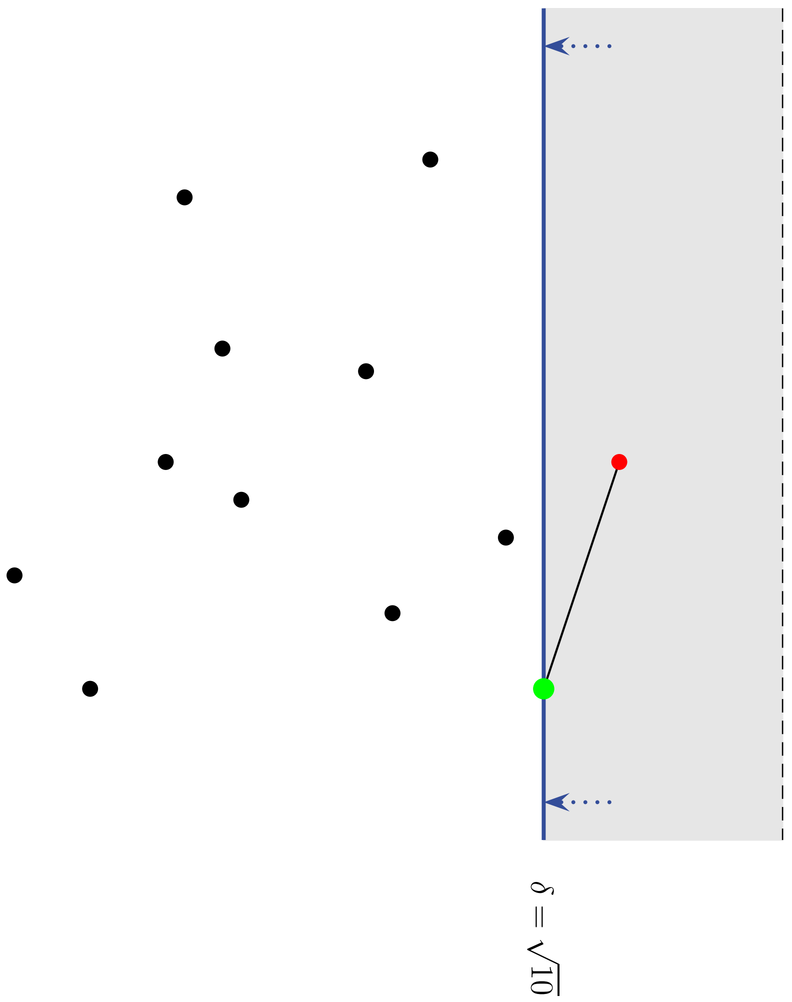
2. Sprawdzana jest odległość p do punktów ze statusu miotły, których rzędna mieści się w przedziale $[y - \delta, y + \delta]$.

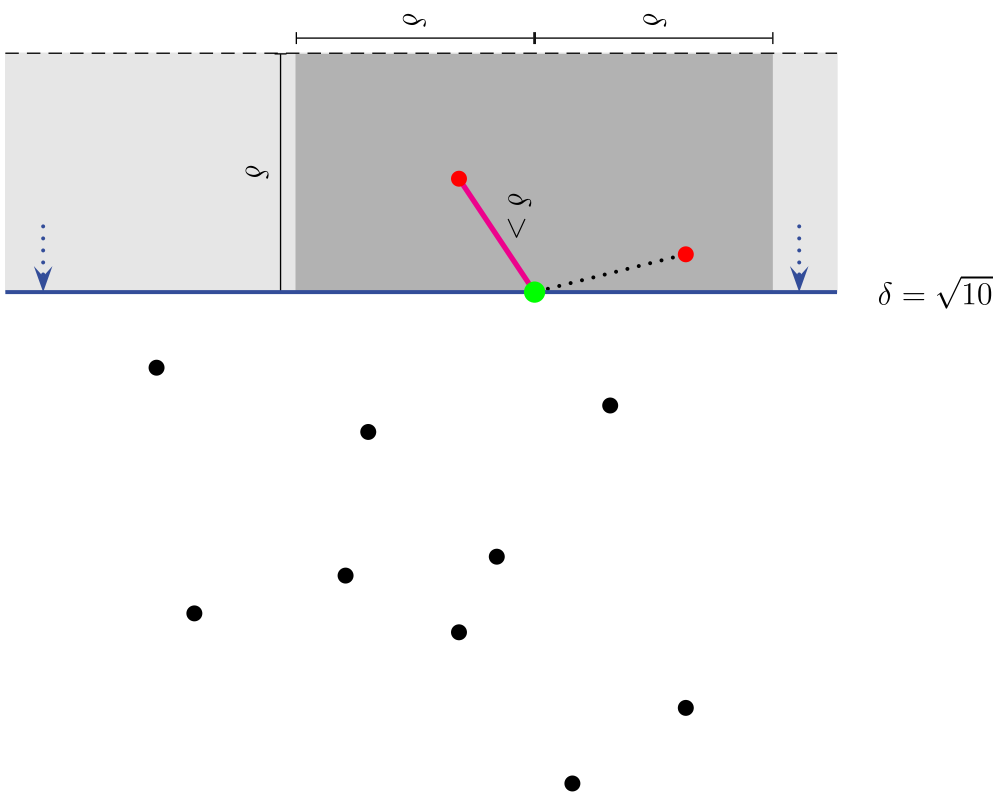
3. Jeśli któraś odległości wyznaczonych w (2) jest mniejsza od dotychczasowej δ, to δ ulega zmianie na najmniejszą z nich. Zapamiętujemy również parę punktów realizującą tę najmniejszą odległość.

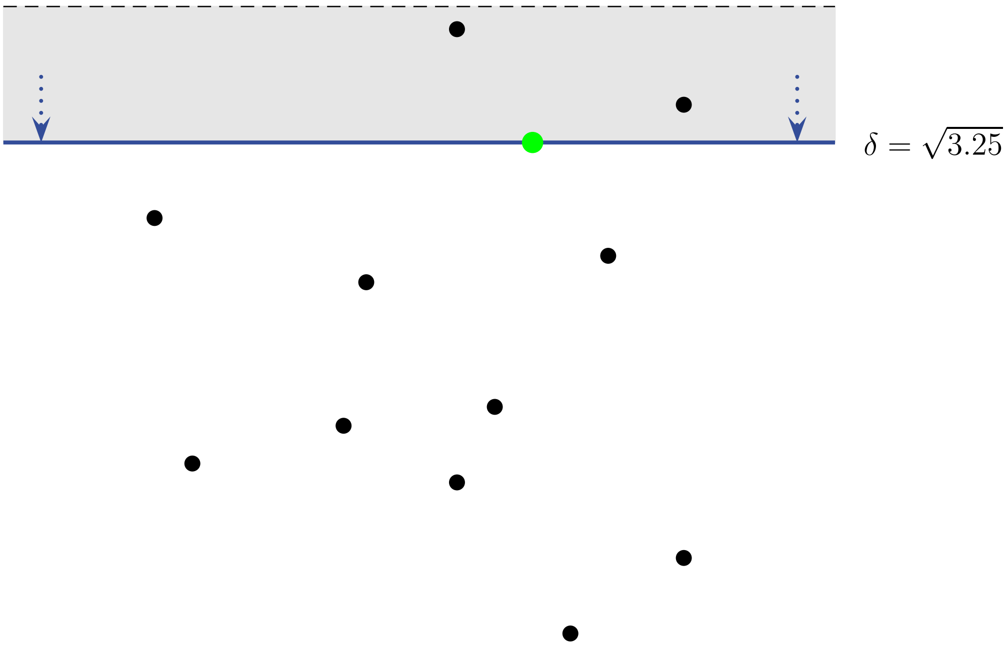
4. Wstawiamy p do statusu miotły.

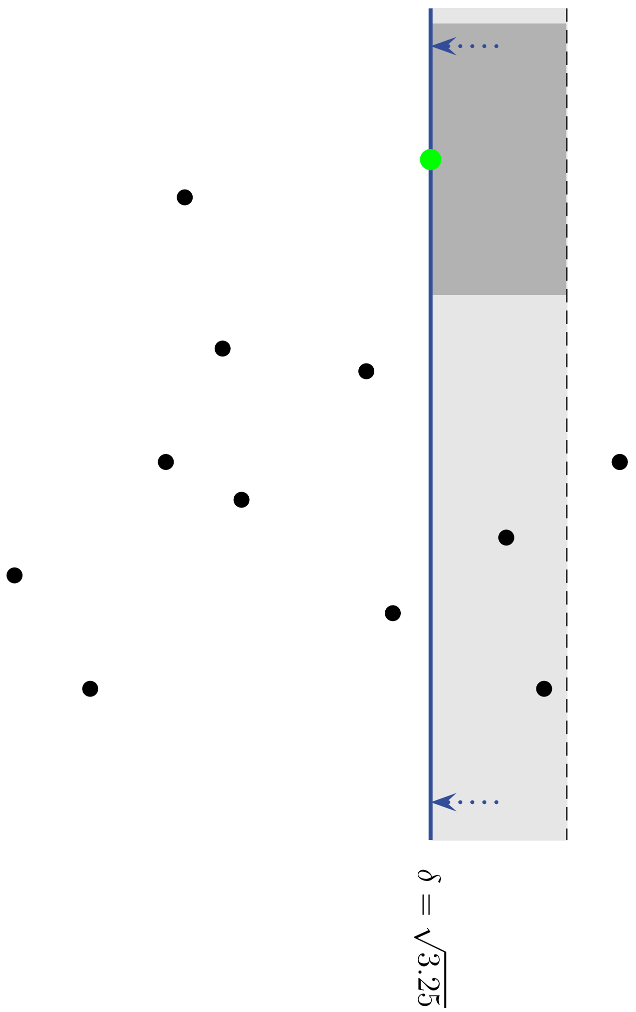


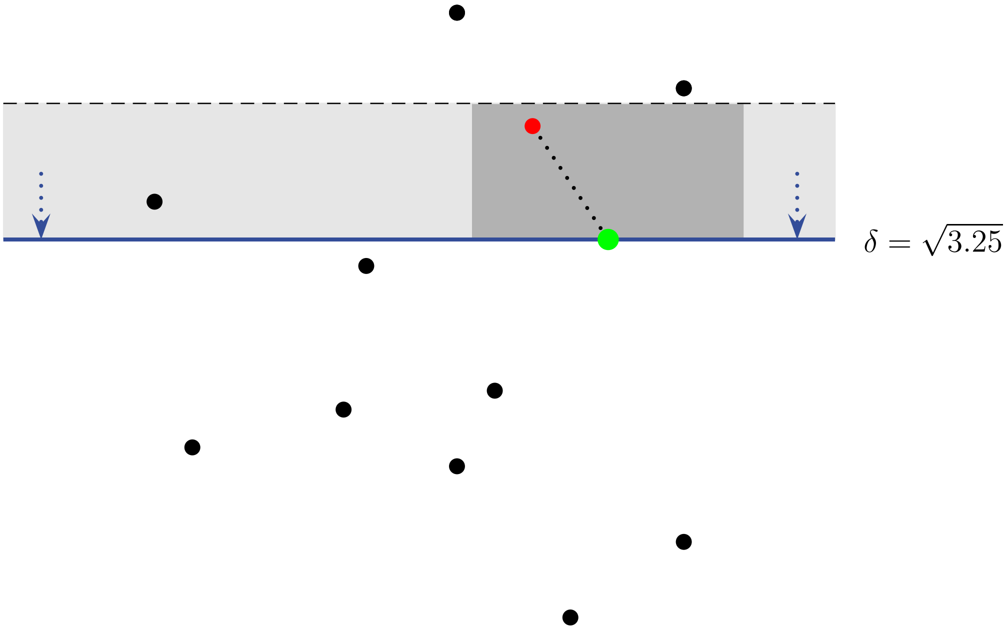


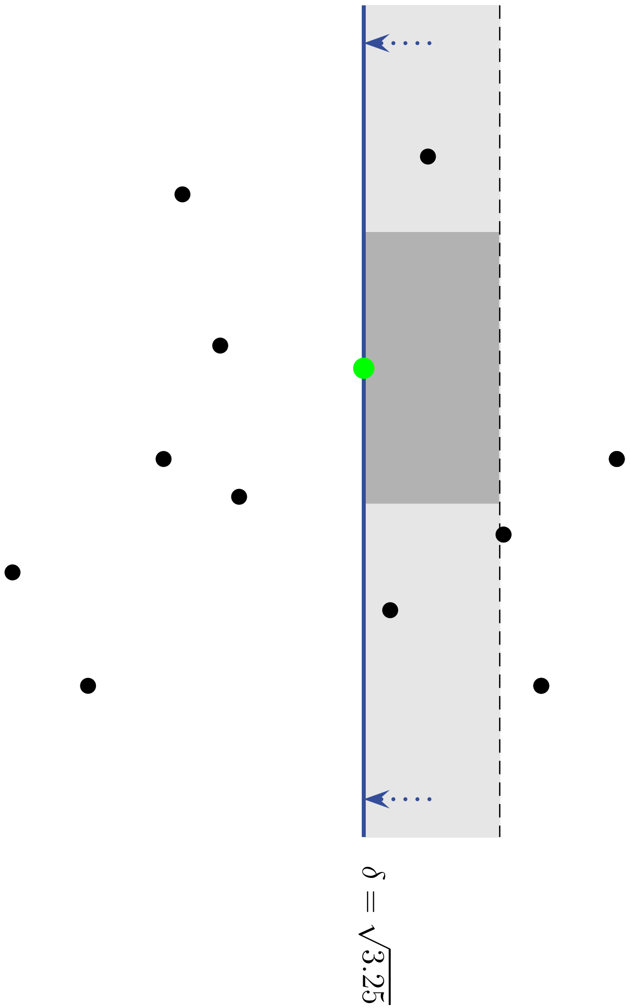

$\delta = \sqrt{10}$


δ
δ
δ
$< \delta$
$\delta = \sqrt{10}$

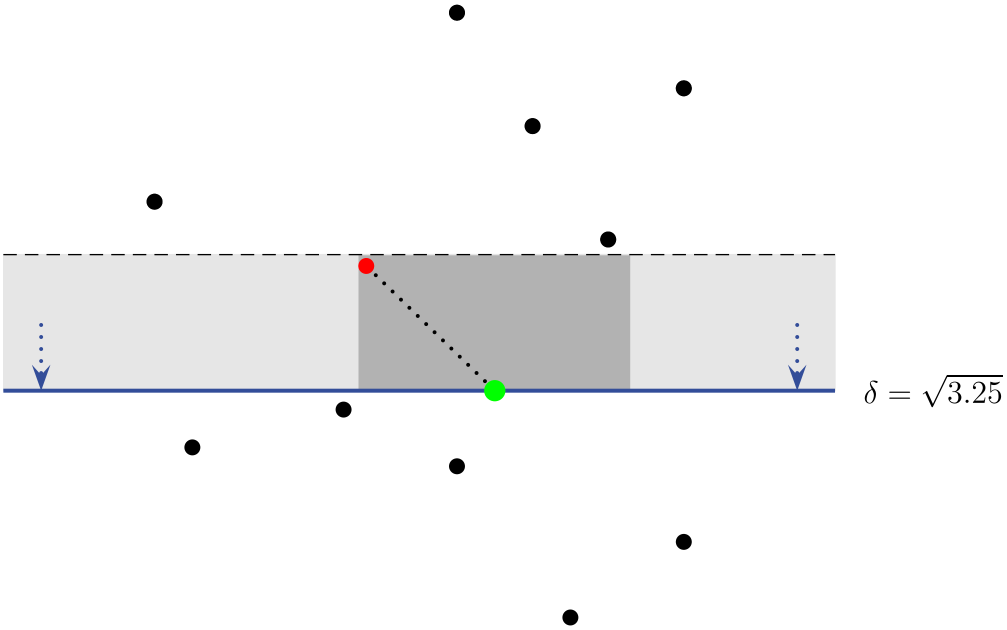

$\delta = \sqrt{3.25}$

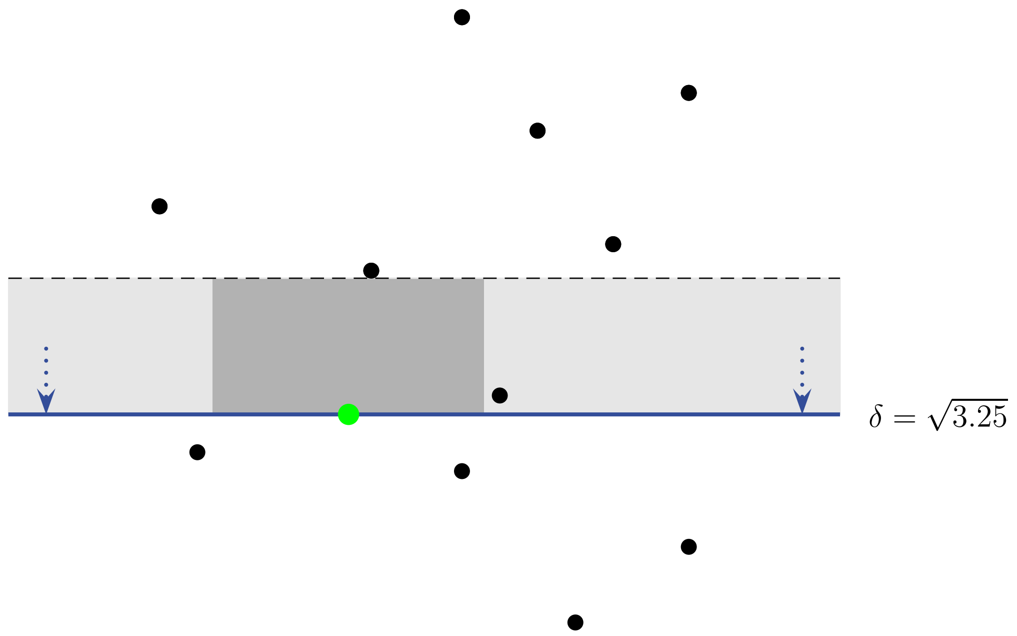

$\delta = \sqrt{3.25}$

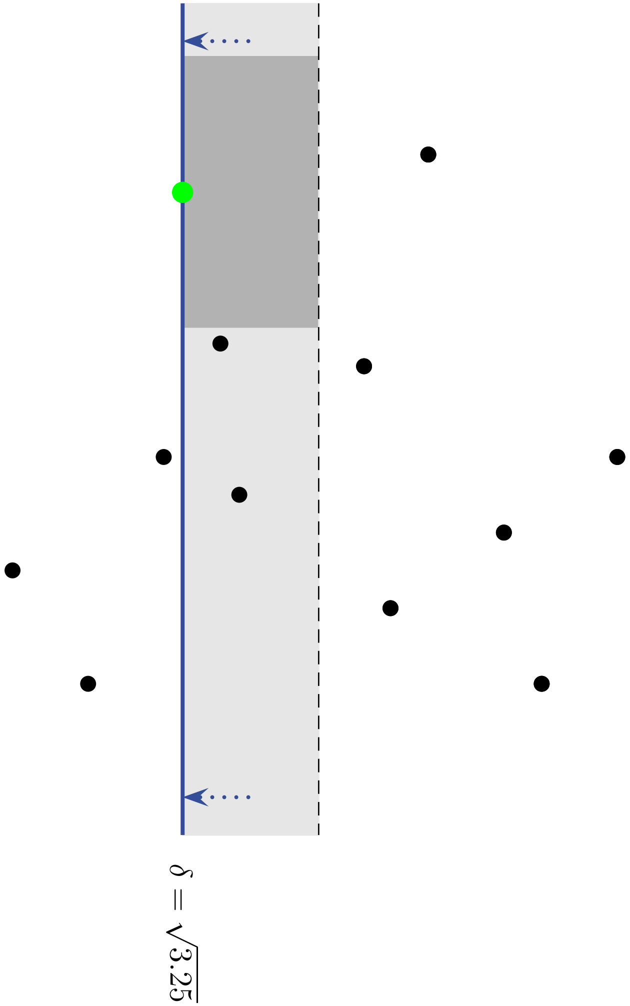

$\delta = \sqrt{3.25}$

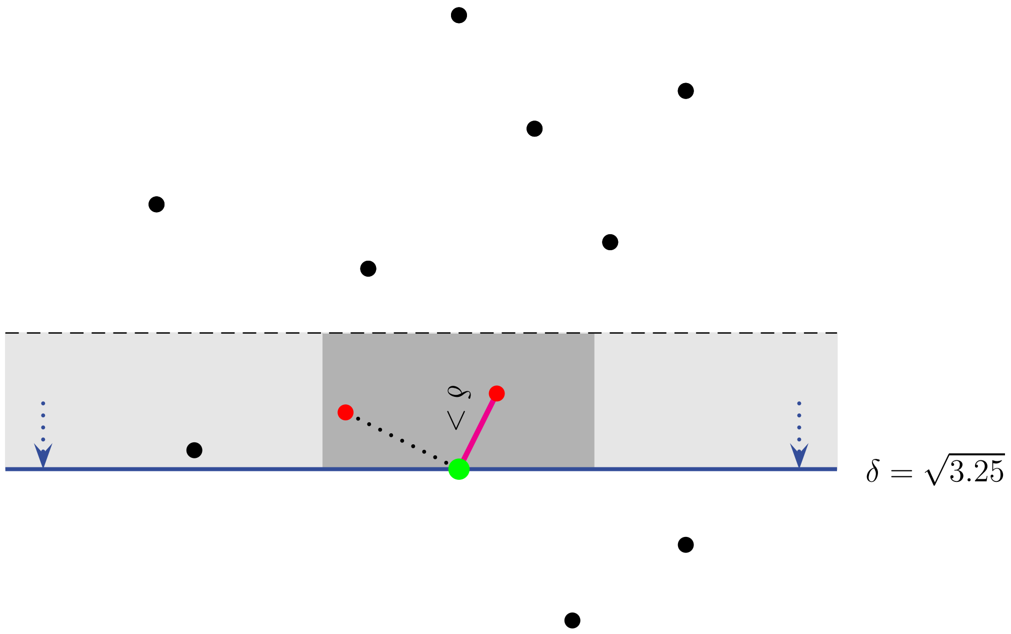


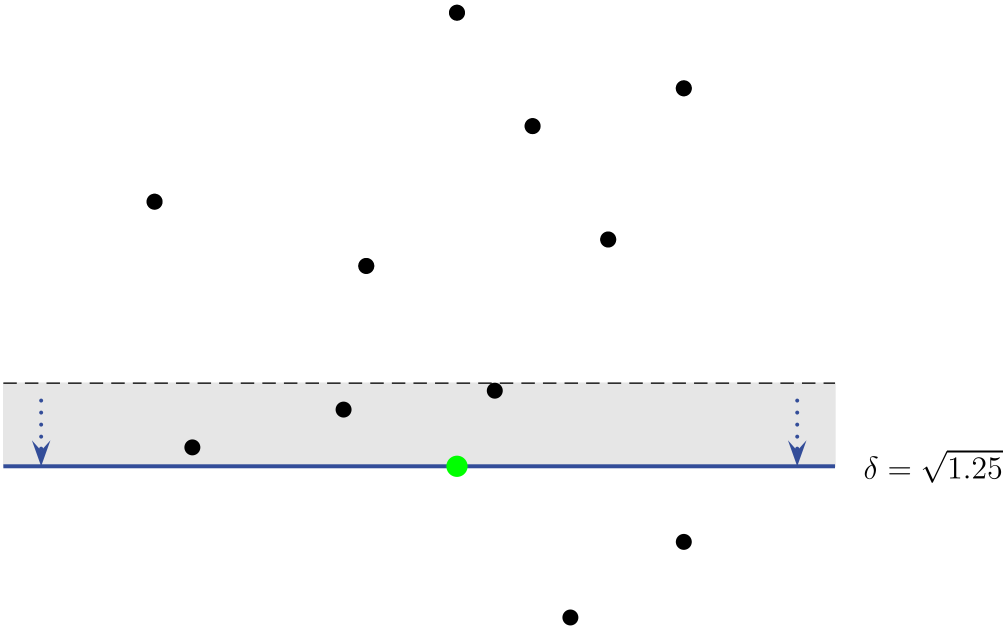
$\delta = \sqrt{3.25}$

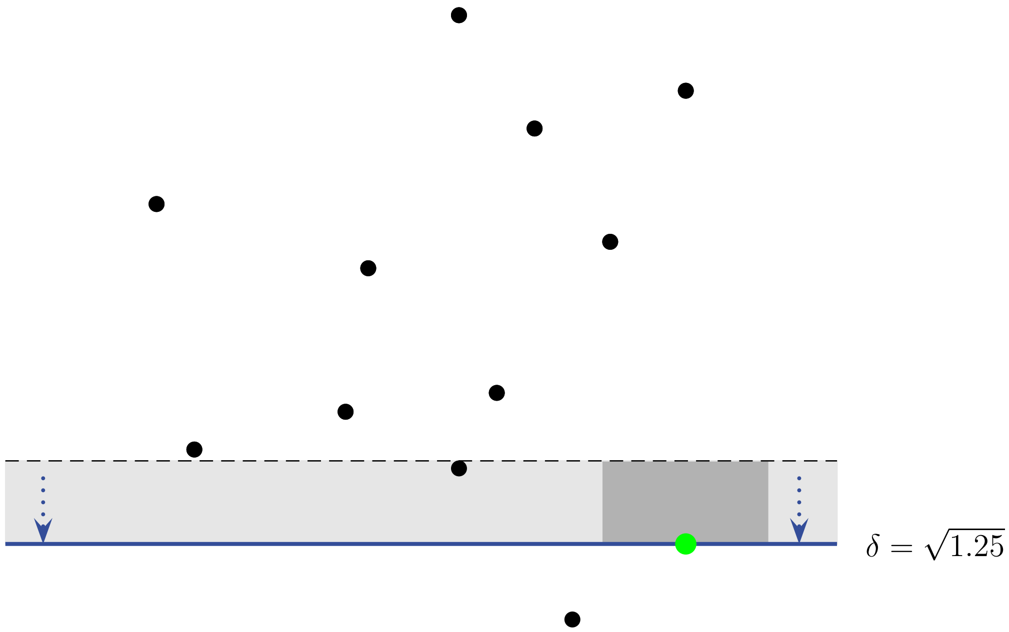

$\delta = \sqrt{3.25}$

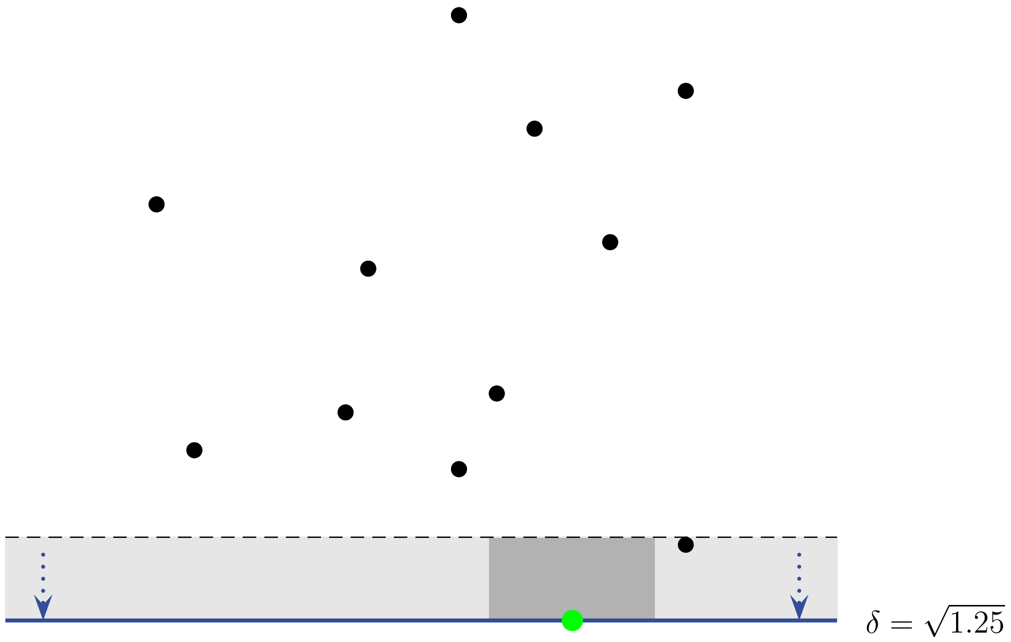

$\delta = \sqrt{3.25}$

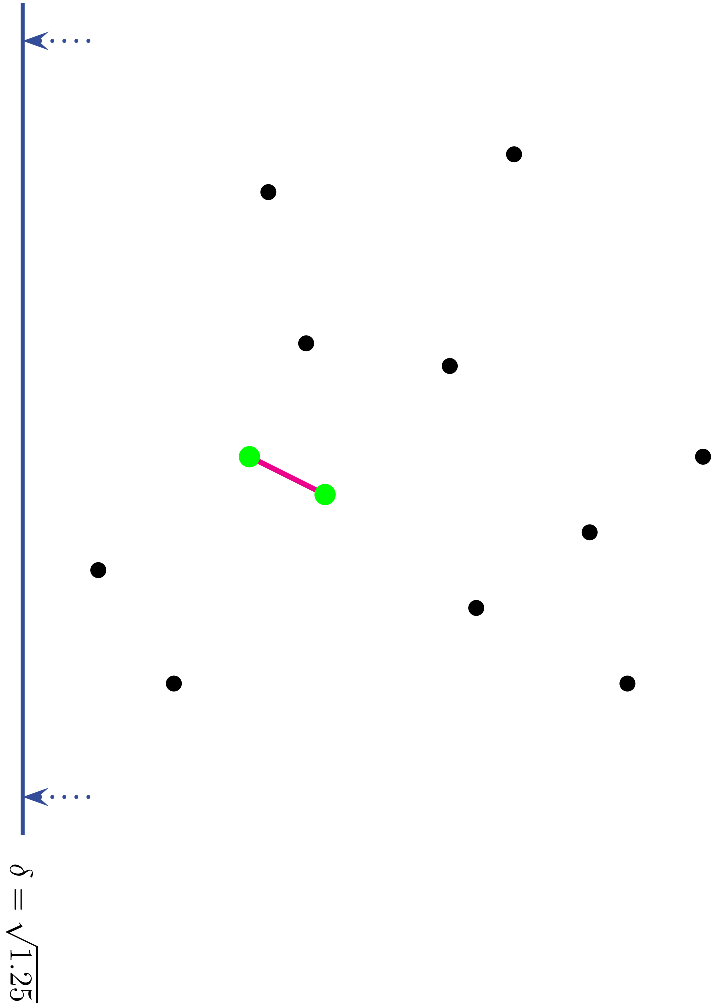

$\delta = \sqrt{3.25}$

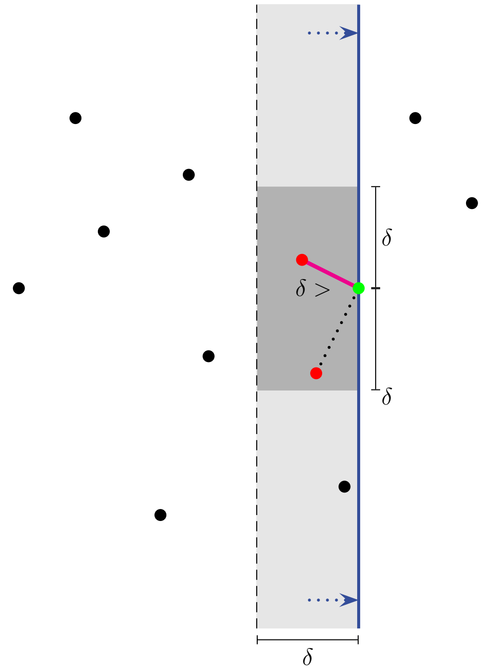

$< \varrho$
$\delta = \sqrt{3.25}$


$\delta = \sqrt{1.25}$


$\delta = \sqrt{1.25}$


$\delta = \sqrt{1.25}$


$\delta = \sqrt{1.25}$



Analiza poprawności podejścia

- ▶ Prawdziwość niezmiennika w przypadku bazowym – tj. gdy rozpatrujemy dwa pierwsze punkty p_1 p_2 i ustalamy $\delta = d_{\mathcal{E}}(p_1, p_2)$ – wynika z definicji.
- ▶ Rozpatrzyliśmy już k punktów, zachowując prawdziwość niezmiennika, tzn. δ przechowuje najmniejszą z odległości dla $\{p_1, p_2, \ldots, p_k\} = S_k$.
- ▶ Nowe zdarzenie/punkt $p_{k+1} = (x, y)$.
  - ↳ Minimalna odległość w zbiorze $\{p_1, p_2, \ldots, p_k, p_{k+1}\} = S_{k+1}$:

$$\min(\delta, d_{\mathcal{E}}(p_{k+1}, S_k)), \text{ gdzie } d_{\mathcal{E}}(p_{k+1}, S_k) = \min_{p \in S_k} d_{\mathcal{E}}(p_{k+1}, p).$$

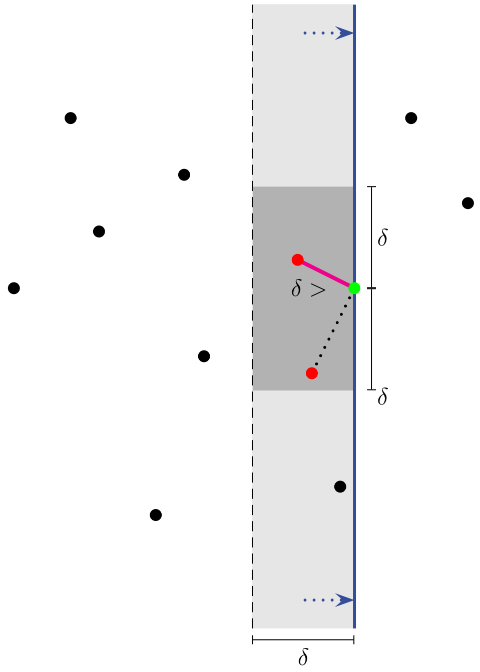
  - ↳ Jeśli $p \in S_k$ oraz $p \notin [x - \delta, x] \times [y - \delta, y + \delta]$, to $d_{\mathcal{E}}(p_{k+1}, p) > \delta$.
  - ↳ A zatem należy sprawdzić jedynie $d_{\mathcal{E}}(p_{k+1}, p)$ dla $p \in [x - \delta, x] \times [y - \delta, y + \delta]$.

Algorytm SweepClosest

Wejście. Zbiór punktów S na płaszczyźnie.

Wyjście. Para najbliższych punktów z S.

```
0. Posortuj zbiór $S$ względem współrzędnej $x$,
   otrzymując zbiór $S = \{p_1, \ldots, p_n\}$.
1. $\delta = d_{\mathcal{E}}(p_1, p_2)$; zapamiętaj parę $(p_1, p_2)$.
2. Wstaw $p_1$ i $p_2$ do pustego zrównoważonego drzewa przeszukiwań $\mathcal{T}$
   o porządku względem współrzędnej $y$.
3. lewy:=1; prawy:=3.
4. while (prawy $\leq n$) do
   4.1 while $\big(\mathrm{x}(p_{\text{lewy}}) < \mathrm{x}(p_{\text{prawy}}) - \delta\big)$ do
           usuń $p_{\text{lewy}}$ z drzewa $\mathcal{T}$;
           lewy:=lewy+1;
   4.2 Niech $S_\delta$ :=1DRangeQuery$\big(\mathcal{T}, [\mathrm{y}(p_{\text{prawy}}) - \delta, \mathrm{y}(p_{\text{prawy}}) + \delta]\big)$.
   4.3 $\delta := \min(\delta, d_{\mathcal{E}}(p_{\text{prawy}}, S_\delta))$; zapamiętaj parę realizującą minimum.
   4.4 Wstaw $p_{\text{prawy}}$ do drzewa $\mathcal{T}$; prawy:=prawy+1.
5. Zwróć parę realizującą $\delta$.
```

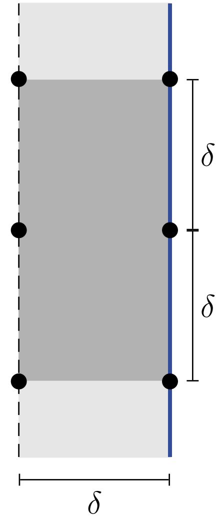


Algorytm SweepClosest

Wejście. Zbiór punktów S na płaszczyźnie.

Wyjście. Para najbliższych punktów z S.

```
0. Posortuj zbiór $S$ względem współrzędnej $x$,
   otrzymując zbiór $S = \{p_1, \ldots, p_n\}$.
1. $\delta = d_{\mathcal{E}}(p_1, p_2)$; zapamiętaj parę $(p_1, p_2)$.
2. Wstaw $p_1$ i $p_2$ do pustego zrównoważonego drzewa przeszukiwań $\mathcal{T}$
   o porządku względem współrzędnej $y$.
3. lewy:=1; prawy:=3.
4. while (prawy $\leq n$) do
   4.1 while $\big(\mathrm{x}(p_{\mathrm{lewy}}) < \mathrm{x}(p_{\mathrm{prawy}}) - \delta\big)$ do
             usuń $p_{\mathrm{lewy}}$ z drzewa $\mathcal{T}$;
             lewy:=lewy+1;
   4.2 Niech $S_\delta$ :=1DRangeQuery$\big(\mathcal{T}, [\mathrm{y}(p_{\mathrm{prawy}}) - \delta, \mathrm{y}(p_{\mathrm{prawy}}) + \delta]\big)$.
   4.3 $\delta := \min(\delta, d_{\mathcal{E}}(p_{\mathrm{prawy}}, S_\delta))$; zapamiętaj parę realizującą minimum.
   4.4 Wstaw $p_{\mathrm{prawy}}$ do drzewa $\mathcal{T}$; prawy:=prawy+1.
5. Zwróć parę realizującą $\delta$.
```



Analiza złożoności obliczeniowej

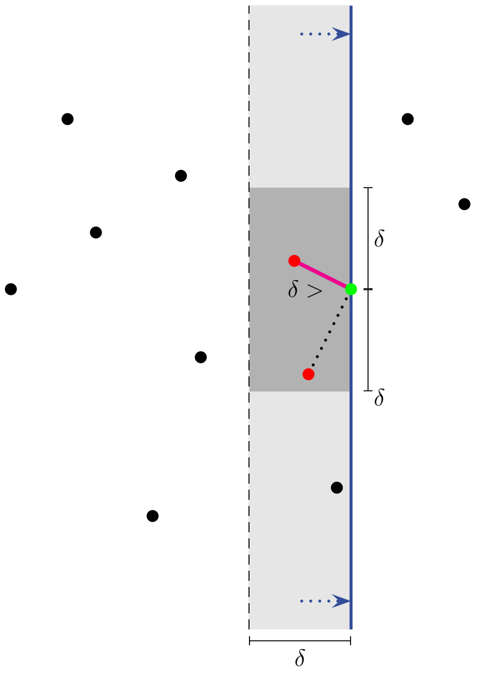
- ▶ Złożoność czasowa jest rzędu $O(n \log n)$.
  - ↳ Czas 1DRangeQuery jest rzędu $O(\log n) + |S_\delta| = O(\log n)$, bo $|S_\delta| \leq 6$.
- ▶ Złożoność pamięciowa jest rzędu $O(n)$.
  - ↳ Status prostej (zrównoważone drzewo binarne): pamięć rzędu $O(n)$.

Algorytm SweepClosest

Wejście. Zbiór punktów S na płaszczyźnie.

Wyjście. Para najbliższych punktów z S.

```
0. Posortuj zbiór $S$ względem współrzędnej $x$,
   otrzymując zbiór $S = \{p_1, \ldots, p_n\}$.
1. $\delta = d_{\mathcal{E}}(p_1, p_2)$; zapamiętaj parę $(p_1, p_2)$.
2. Wstaw $p_1$ i $p_2$ do pustego zrównoważonego drzewa przeszukiwań $\mathcal{T}$
   o porządku względem współrzędnej $y$.
3. lewy:=1; prawy:=3.
4. while (prawy $\leq n$) do
   4.1 while $\big(\mathrm{x}(p_{\mathrm{lewy}}) < \mathrm{x}(p_{\mathrm{prawy}}) - \delta\big)$ do
           usuń $p_{\mathrm{lewy}}$ z drzewa $\mathcal{T}$;
           lewy:=lewy+1;
   4.2 Niech $S_\delta$ :=1DRangeQuery$\big(\mathcal{T}, [\mathrm{y}(p_{\mathrm{prawy}}) - \delta, \mathrm{y}(p_{\mathrm{prawy}}) + \delta]\big)$.
   4.3 $\delta := \min(\delta, d_{\mathcal{E}}(p_{\mathrm{prawy}}, S_\delta))$; zapamiętaj parę realizującą minimum.
   4.4 Wstaw $p_{\mathrm{prawy}}$ do drzewa $\mathcal{T}$; prawy:=prawy+1.
5. Zwróć parę realizującą $\delta$.
```

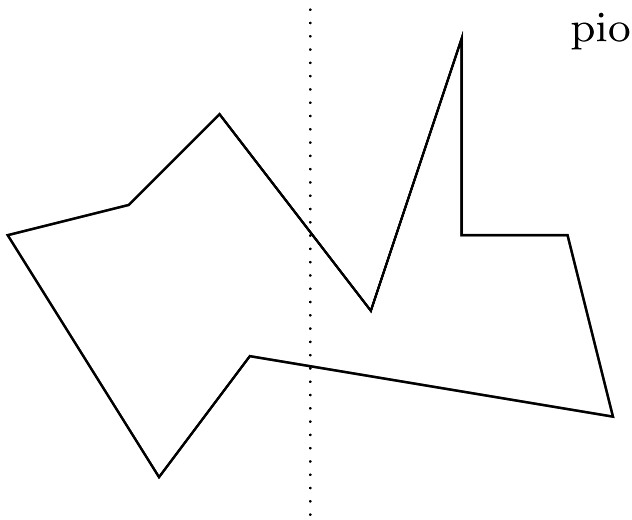


Twierdzenie 6.3. (Hinrichs, Nievergelt, Schorn 1988)

Algorytm SweepClosest *wyznacza najbliższą parę w zbiorze n punktów na płaszczyźnie w czasie $O(n \log n)$, używając pamięci rzędu $O(n)$.*

6.4 TRIANGULACJA WIELOKĄTA MONOTONICZNEGO

Wielokąt prosty nazywamy *monotonicznym względem prostej l*, jeśli dla dowolnej prostej l' prostopadłej do l przecięcie wielokąta z l' jest spójne (jest odcinkiem, punktem, lub zbiorem pustym). Wielokąt, który jest monotoniczny względem osi x, nazywamy *x-monotonicznym*.



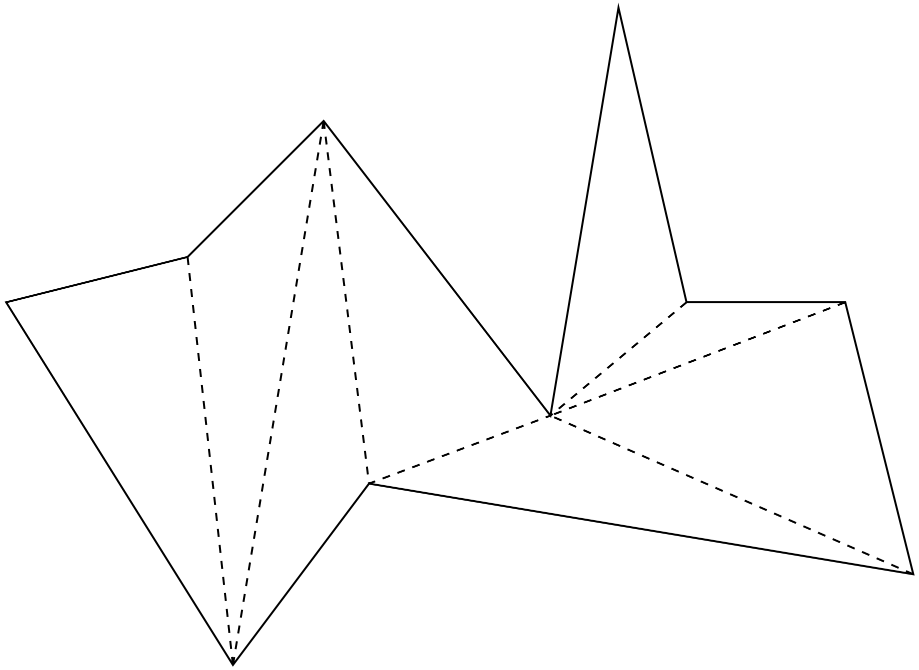
Przecięcie z dowolną pionową prostą jest spójne.

W wielokącie x-monotonicznym, jeśli idziemy wzdłuż łańcucha wierzchołków – dolnego lub górnego – ze skrajnego lewego wierzchołka do skrajnego prawego wierzchołka, to zawsze poruszamy się w prawo lub pionowo, nigdy w lewo.

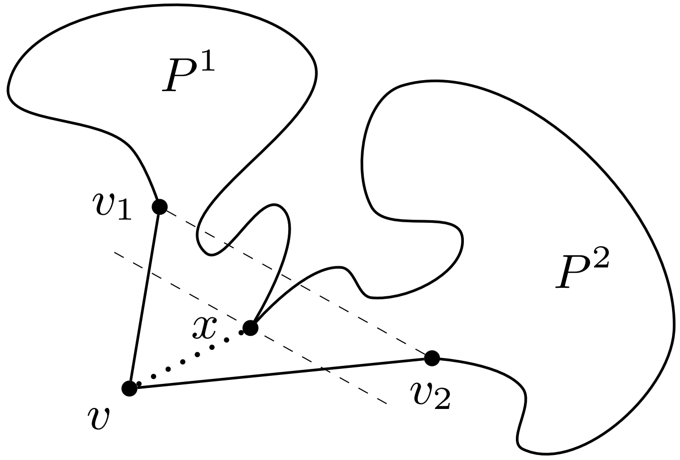
Założenie. Wielokąty *ściśle* monotoniczne: odcięte wierzchołków są różne.

Problem. *Podzielić ściśle x-monotoniczny wielokąt na trójkąty przez dodanie wewnętrznych nieprzecinających się przekątnych łączących wierzchołki wielokąta.*

M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf
Geometria obliczeniowa, rozdział 3, WNT (2007)

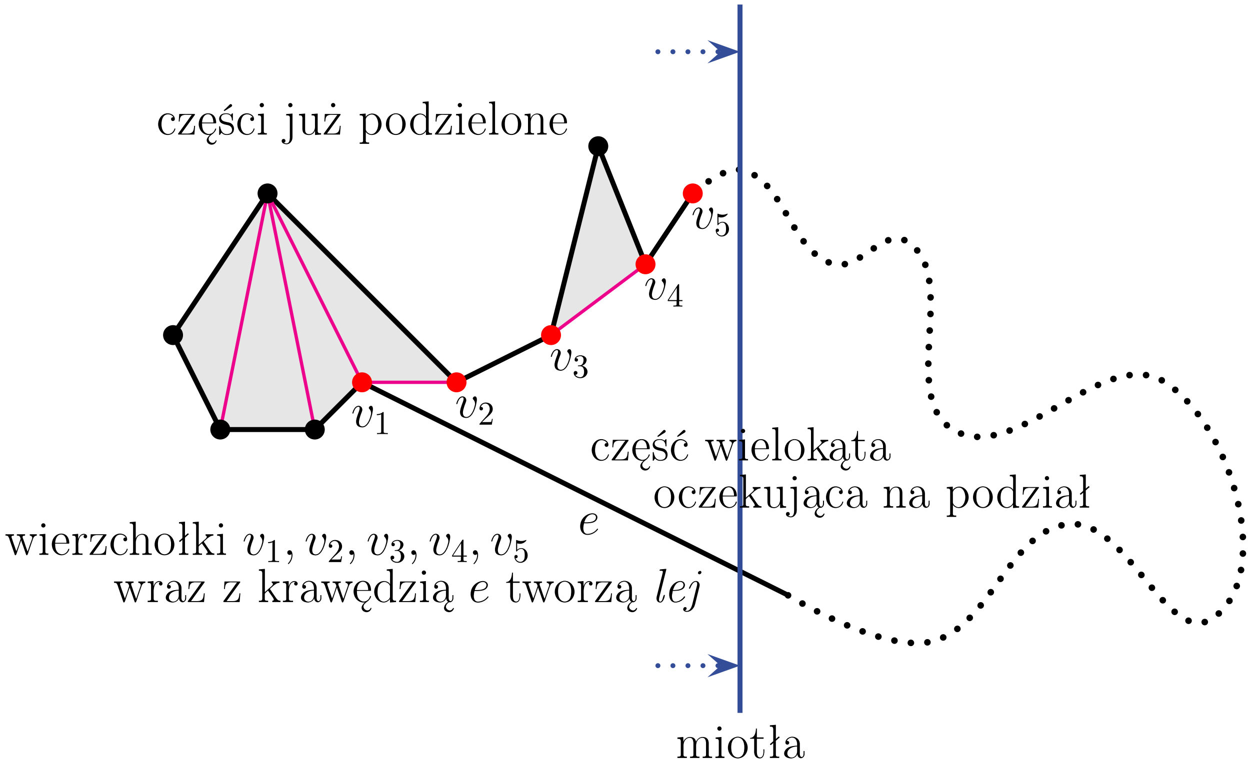


Problem. *Podzielić ściśle x-monotoniczny wielokąt na trójkąty przez dodanie wewnętrznych nieprzecinających się przekątnych łączących wierzchołki wielokąta.*



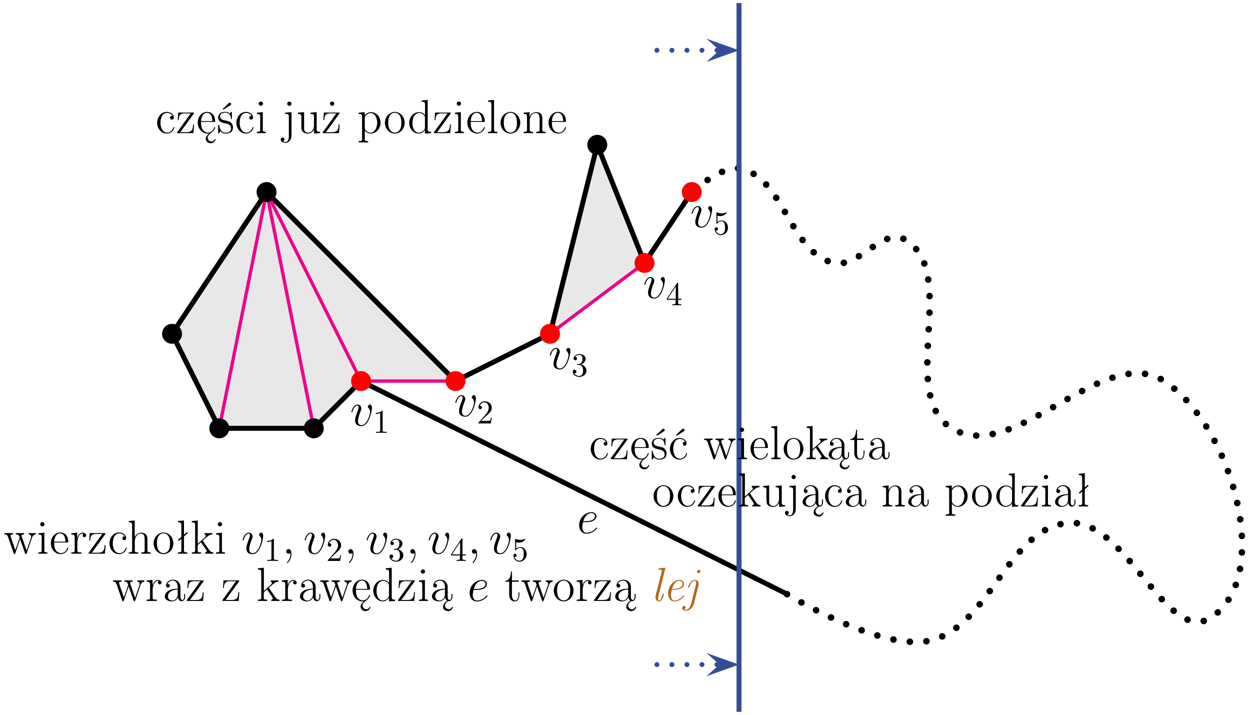
Twierdzenie 6.4. (m.in. Meisters 1975) *Dowolny wielokąt prosty o n wierzchołkach można podzielić na $n-2$ trójkąty przez dodanie $n-3$ wewnętrznych nieprzecinających się przekątnych o wierzchołkach będących wierzchołkami wielokąta.*

↳ Dowolny wielokąt prosty ($n \geq 4$) ma przynajmniej dwa tzw. *ucha*.
Dowód indukcyjny po liczbie wierzchołków (patrz ilustracja).



Idea algorytmu zamiatania

- Miotła przesuwa się z lewo na prawo.
- Status miotły stanowi zbiór wierzchołków na stosie tworzących tzw. *lej*.
- Napotykając nowy wierzchołek v, w zależności od położenia v względem leja, wierzchołek ten jest albo odkładany na stos, powiększając lej, albo cześć wierzchołków jest zdejmowana ze stosu i zgłaszane są odpowiednie przekątne.

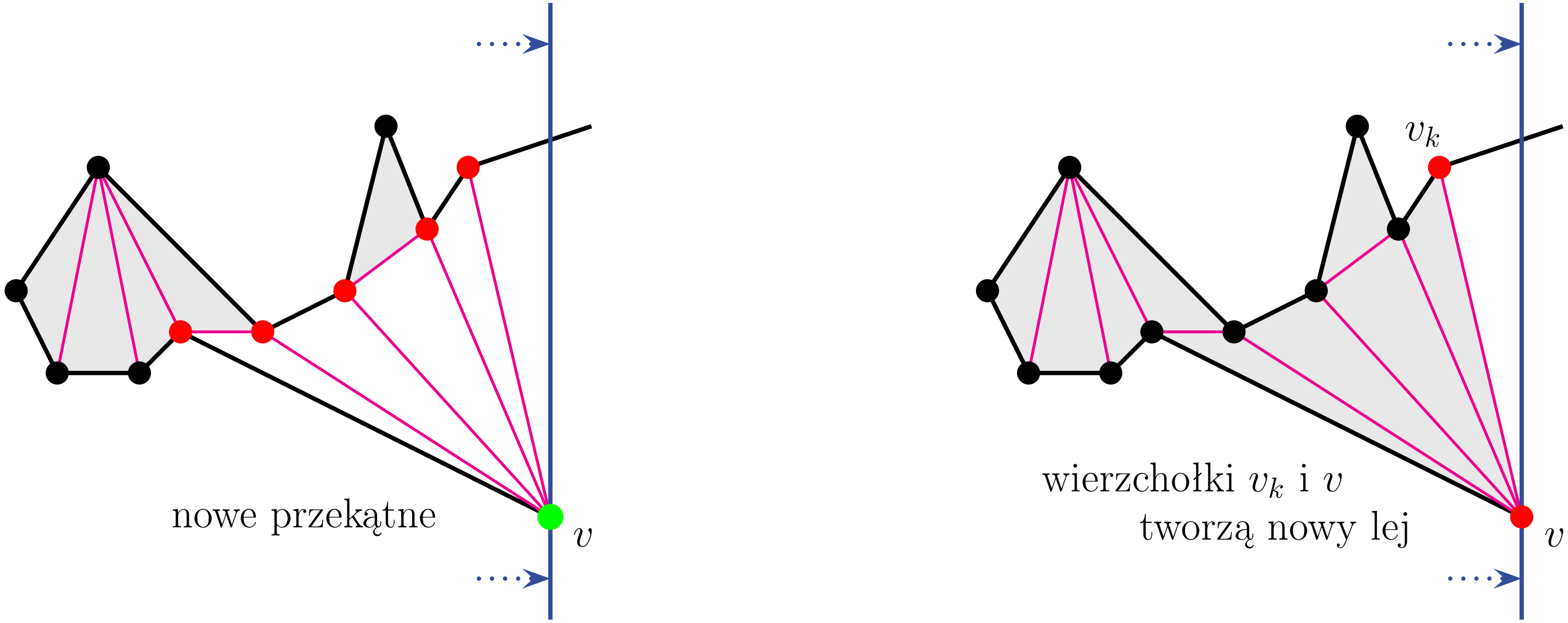


Status miotły. Wierzchołki na stosie.

Niezmiennik. Wierzchołki $v_1, \ldots, v_k$ na stosie tworzą tzw. *lej*, a wielokąt na lewo od łańcucha $v_1, \ldots, v_k$ jest już podzielony na trójkąty..

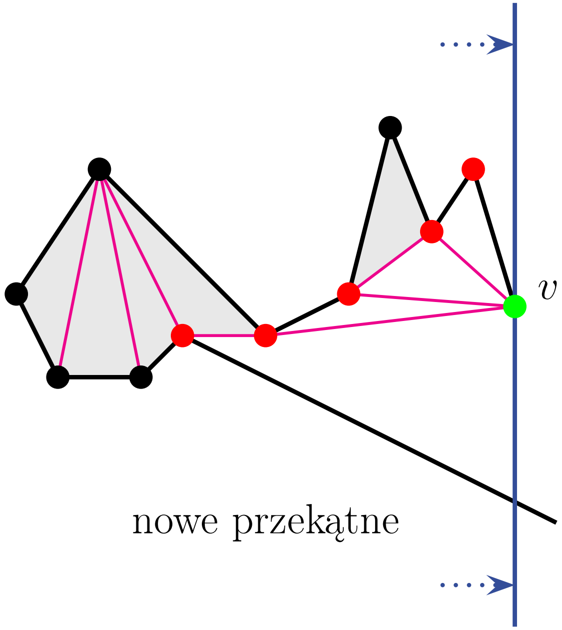
- Wierzchołek v_1, który jest na dnie stosu tworzy kąt wypukły z incydentną krawędzią i kolejnym wierzchołkiem v_2 ze stosu.
- Pozostałe wierzchołki $v_2, \ldots, v_k$ ze stosu należą do tego samego łańcucha (dolnego lub górnego) i są wklęsłe.

Punkty zdarzeń. Wierzchołki P posortowane względem współrzędnej x.

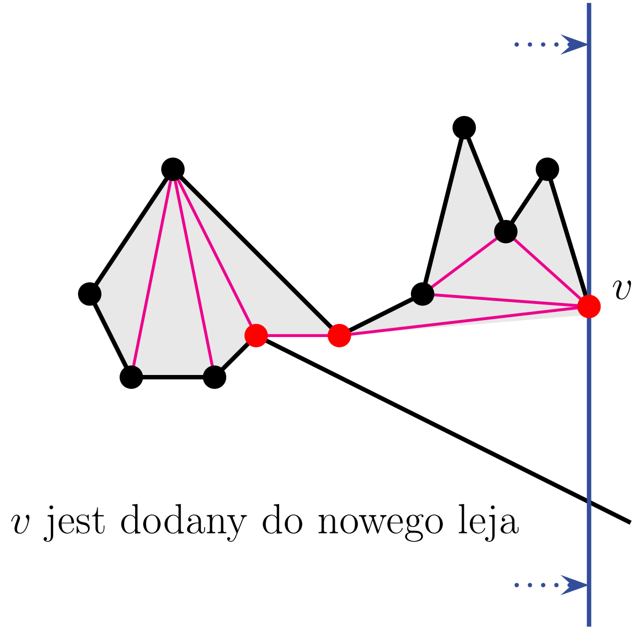


Zmiana statusu. Miotła napotyka nowy wierzchołek v.

1. Jeśli wierzchołek v leży na przeciwległym łańcuchu do wierzchołków $v_2, \ldots, v_k$ ze stosu, tzn. jest prawym końcem krawędzi ograniczającej lej, wówczas możemy dodać przekątne z v do wszystkich wierzchołków znajdujących się na stosie, za wyjątkiem tego z dna stosu (czyli dodajemy przekątne postaci $\{v, v_i\}$, $i = 2, \ldots, k$). Wierzchołki $v_1, \ldots, v_k$ zostają zdjęte ze stosu i włożone są jedynie v_k i v. Niezmiennik – mając na uwadze x-monotoniczność wielokąta oraz położenie v względem $v_2, \ldots, v_k$ – zostaje zachowany.



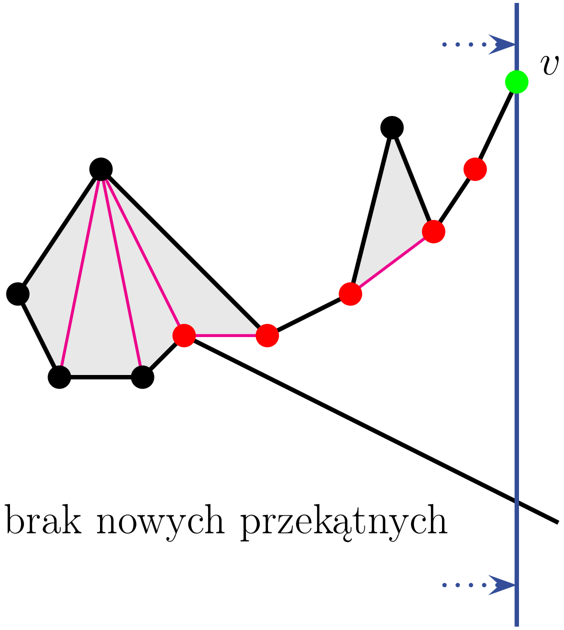
nowe przekątne



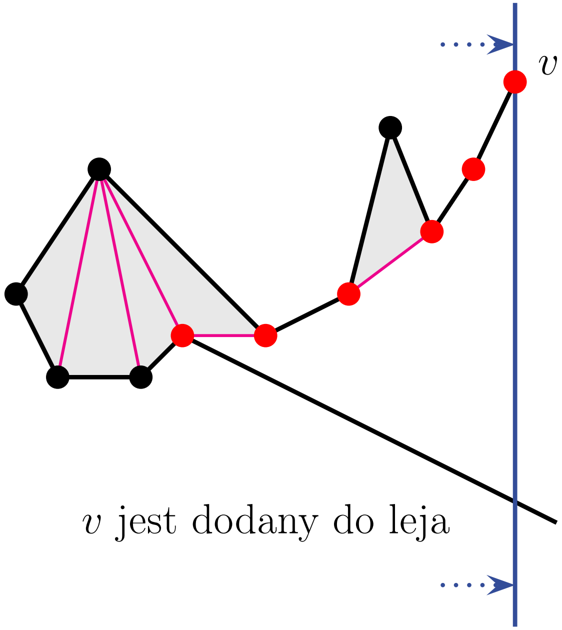
v jest dodany do nowego leja

Zmiana statusu. Miotła napotyka nowy wierzchołek v.

2. Jeśli wierzchołek v leży na tym samym łańcuchu, co wierzchołki $v_2, \ldots, v_k$, wówczas możemy nie być w stanie poprowadzić przekątnych z v do wszystkich wierzchołków ze stosu. Ale te wierzchołki, z którymi możemy połączyć v, są kolejne i znajdują się na szczycie stosu. Zatem zdejmujemy je i dodajemy przekątne, aż nie będzie to możliwe. Wierzchołek v wkładany jest na stos. Zachowanie niezmiennika wynika z niemożliwości dodania dalszych przekątnych.



brak nowych przekątnych



v jest dodany do leja

Zmiana statusu. Miotła napotyka nowy wierzchołek v.

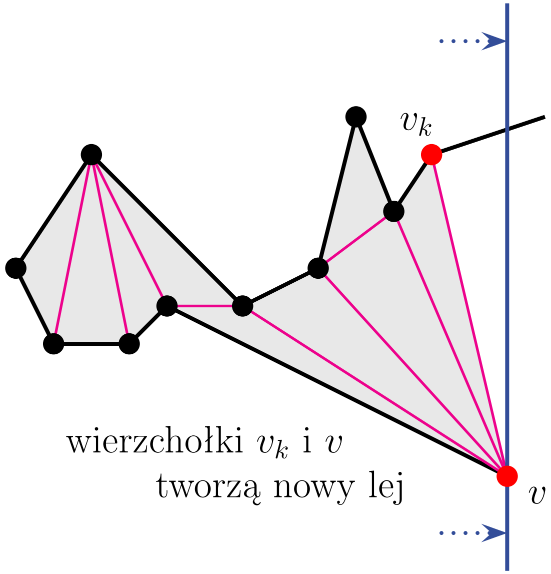
2. Jeśli wierzchołek v leży na tym samym łańcuchu, co wierzchołki $v_2, \ldots, v_k$, wówczas możemy nie być w stanie poprowadzić przekątnych z v do wszystkich wierzchołków ze stosu. Ale te wierzchołki, z którymi możemy połączyć v, są kolejne i znajdują się na szczycie stosu. Zatem zdejmujemy je i dodajemy przekątne, aż nie będzie to możliwe. Wierzchołek v wkładany jest na stos. Zachowanie niezmiennika wynika z niemożliwości dodania dalszych przekątnych.

Algorytm SweepTriangulation

Wejście. Zbiór wierzchołków wielokąta $P = \{v_1, \ldots, v_n\}$ na płaszczyźnie, $n \geq 3$. P jest posortowany względem odciętej.

Wyjście. Podział P na trójkąty.

```
1. Inicjuj pusty stos $S$ i włóż na niego $v_1$ i $v_2$.
2. for $i := 3$ to $n - 1$ do
3.   if ($v_i$ oraz wierzchołek na szczycie stosu są na różnych łańcuchach)
4.     then Zdejmij wszystkie wierzchołki ze stosu $S = (v'_1, \ldots, v'_k)$.
           Zgłoś wszystkie przekątne $\{v_i, v'_j\}$, $j = 2, \ldots, k$.
           Włóż $v'_k$ i $v_i$ na stos.
5.     else Zdejmij wierzchołek $v'_k$ ze stosu $S = (v'_1, \ldots, v'_k)$.
          Zdejmuj pozostałe wierzchołki tak długo, aż odpowiednie przekątne
          $\{v_i, v'_j\}$ są wewnątrz wielokąta; zgłaszaj te przekątne.
          Włóż ostatnio zdjęty wierzchołek na stos.
          Włóż $v_i$ na stos.
6. Dodaj przekątne z $v_n$ do wszystkich wierzchołków na stosie, oprócz pierwszego i ostatniego.
```



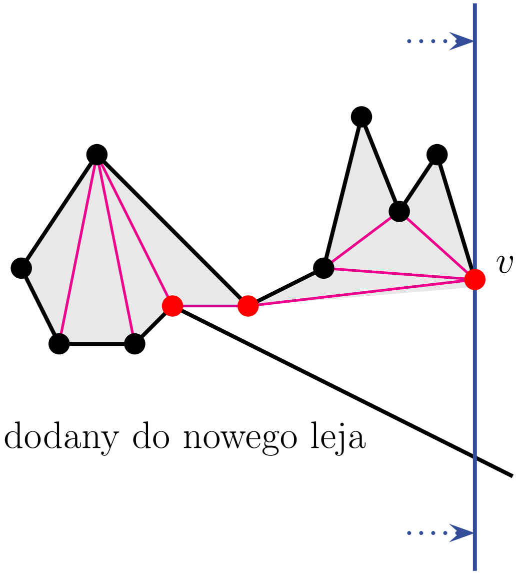
wierzchołki v_k i v tworzą nowy lej

Algorytm SweepTriangulation

Wejście. Zbiór wierzchołków wielokąta $P = \{v_1, \ldots, v_n\}$ na płaszczyźnie, $n \geq 3$. P jest posortowany względem odciętej.

Wyjście. Podział P na trójkąty.

```
Inicjuj pusty stos $S$ i włóż na niego $v_1$ i $v_2$.
for $i := 3$ to $n - 1$ do
  if ($v_i$ oraz wierzchołek na szczycie stosu są na różnych łańcuchach)
    then Zdejmij wszystkie wierzchołki ze stosu $S = (v'_1, \ldots, v'_k)$.
         Zgłoś wszystkie przekątne $\{v_i, v'_j\}$, $j = 2, \ldots, k$.
         Włóż $v'_k$ i $v_i$ na stos.
    else Zdejmij wierzchołek $v'_k$ ze stosu $S = (v'_1, \ldots, v'_k)$.
       Zdejmuj pozostałe wierzchołki tak długo, aż odpowiednie przekątne
       $\{v_i, v'_j\}$ są wewnątrz wielokąta; zgłaszaj te przekątne.
       Włóż ostatnio zdjęty wierzchołek na stos.
       Włóż $v_i$ na stos.
Dodaj przekątne z $v_n$ do wszystkich wierzchołków na stosie, oprócz pierwszego i ostatniego.
```



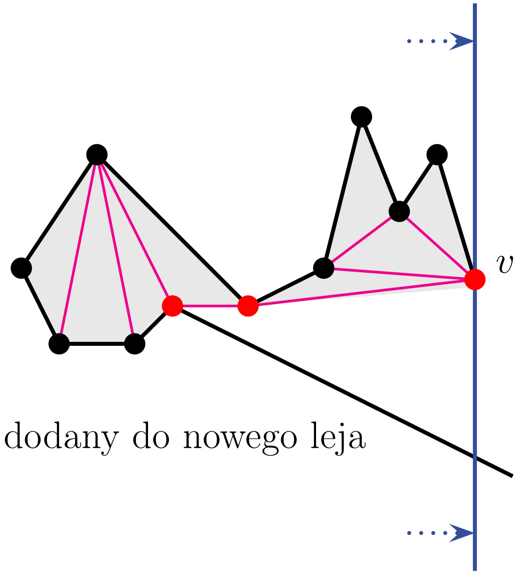
v jest dodany do nowego leja

Algorytm SweepTriangulation

Wejście. Zbiór wierzchołków wielokąta $P = \{v_1, \ldots, v_n\}$ na płaszczyźnie, $n \geq 3$. P jest posortowany względem odciętej.

Wyjście. Podział P na trójkąty.

```
1. Inicjuj pusty stos $S$ i włóż na niego $v_1$ i $v_2$.
2. for $i := 3$ to $n - 1$ do
3.   if ($v_i$ oraz wierzchołek na szczycie stosu są na różnych łańcuchach)
4.     then Zdejmij wszystkie wierzchołki ze stosu $S = (v'_1, \ldots, v'_k)$.
          Zgłoś wszystkie przekątne $\{v_i, v'_j\}$, $j = 2, \ldots, k$.
          Włóż $v'_k$ i $v_i$ na stos.
5.     else Zdejmij wierzchołek $v'_k$ ze stosu $S = (v'_1, \ldots, v'_k)$.
         Zdejmuj pozostałe wierzchołki tak długo, aż odpowiednie przekątne
         $\{v_i, v'_j\}$ są wewnątrz wielokąta; zgłaszaj te przekątne.
         Włóż ostatnio zdjęty wierzchołek na stos.
         Włóż $v_i$ na stos.
6. Dodaj przekątne z $v_n$ do wszystkich wierzchołków na stosie, oprócz pierwszego i ostatniego.
```



v jest dodany do nowego leja

Analiza złożoności obliczeniowej

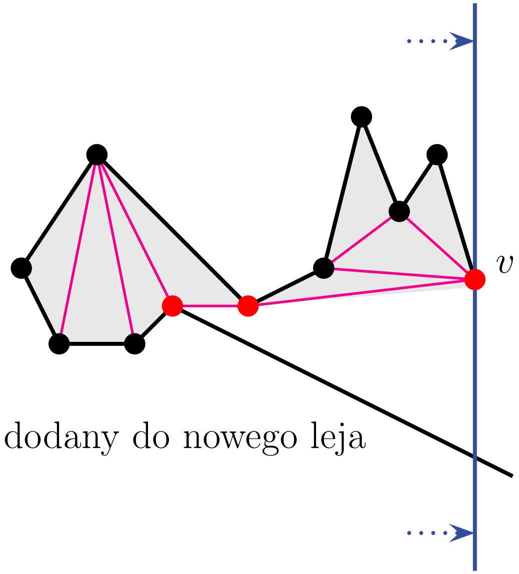
- Złożoność czasowa rzędu $O(n)$.
  - ↳ Pętla **for** wykonywana jest $n - 3$ razy, a jedno wywołanie może wymagać czasu $O(n)$. Ale przy każdym obiegu pętli wstawiane są dwa wierzchołki. Zatem całkowita liczba wstawień, włączając dwa wstawienia w wierszu 1, wynosi $2n - 4$. Ponieważ liczba usunięć nie przewyższa liczby wstawień, całkowity czas wszystkich wykonań pętli **for** wynosi $O(n)$.
- Złożoność pamięciowa: rozmiar stosu rzędu $O(n)$.

Algorytm SweepTriangulation

Wejście. Zbiór wierzchołków wielokąta $P = \{v_1, \ldots, v_n\}$ na płaszczyźnie, $n \geq 3$. P jest posortowany względem odciętej.

Wyjście. Podział P na trójkąty.

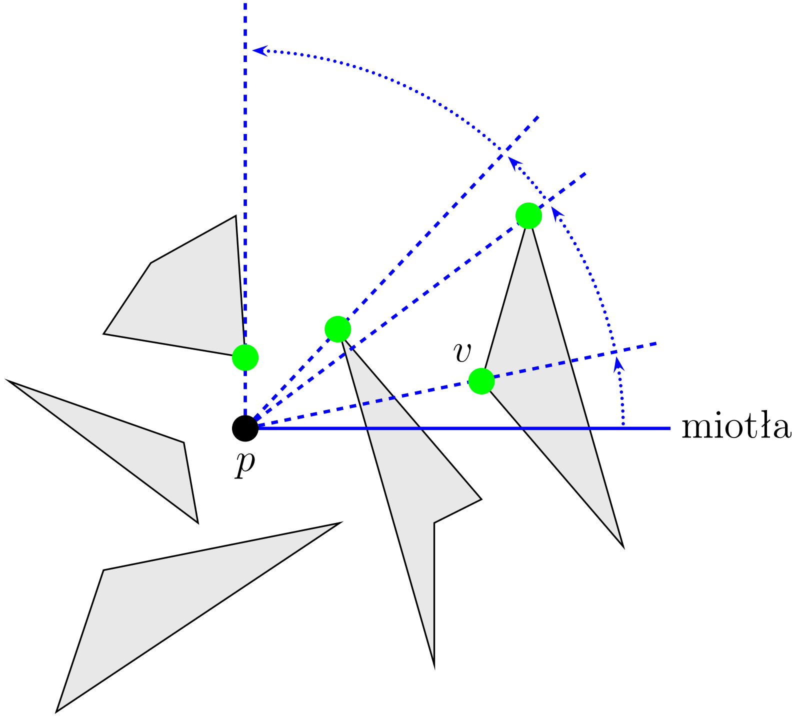
```
1. Inicjuj pusty stos $S$ i włóż na niego $v_1$ i $v_2$.
2. for $i := 3$ to $n - 1$ do
3.   if ($v_i$ oraz wierzchołek na szczycie stosu są na różnych łańcuchach)
4.     then Zdejmij wszystkie wierzchołki ze stosu $S = (v'_1, \ldots, v'_k)$.
           Zgłoś wszystkie przekątne $\{v_i, v'_j\}$, $j = 2, \ldots, k$.
           Włóż $v'_k$ i $v_i$ na stos.
5.     else Zdejmij wierzchołek $v'_k$ ze stosu $S = (v'_1, \ldots, v'_k)$.
          Zdejmuj pozostałe wierzchołki tak długo, aż odpowiednie przekątne
          $\{v_i, v'_j\}$ są wewnątrz wielokąta; zgłaszaj te przekątne.
          Włóż ostatnio zdjęty wierzchołek na stos.
          Włóż $v_i$ na stos.
6. Dodaj przekątne z $v_n$ do wszystkich wierzchołków na stosie, oprócz pierwszego i ostatniego.
```



v jest dodany do nowego leja

Twierdzenie 6.5. (Garey, Johnson, Preparata, Tarjan 1978)
Wielokąt ściśle x-monotoniczny o n wierzchołkach można podzielić na trójkąty przez dodanie nieprzecinających się wewnętrznych przekątnych łączących wierzchołki wielokąta w czasie i pamięci rzędu $O(n)$.

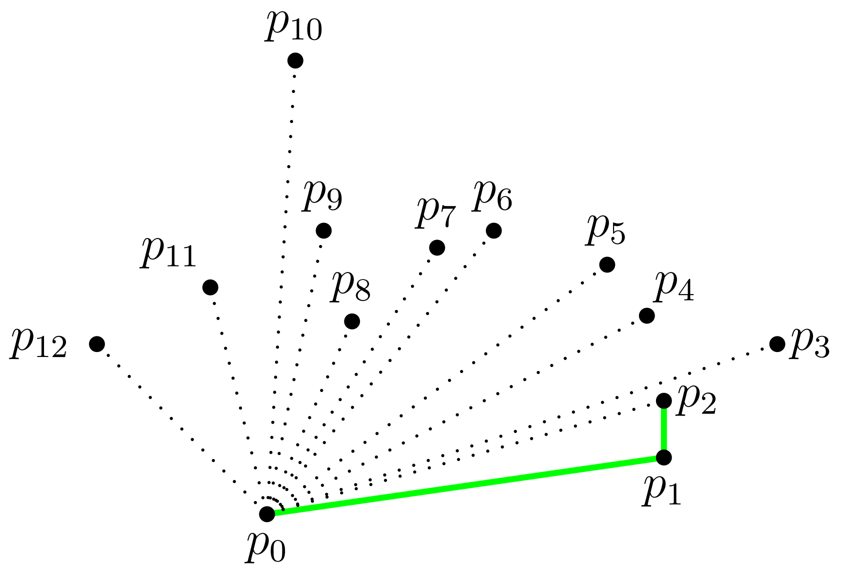
TECHNIKA ZAMIATANIA OBROTOWEGO (na płaszczyźnie)



- Idea algorytmu zamiatania obrotowego polega na obracaniu półprostej – *miotły* – po płaszczyźnie wokół ustalonego punktu.
- Podczas zamiatania utrzymywane są dodatkowe informacje: *status* miotły.
- Status miotły zmienia się w *punktach zdarzeń*.

6.5 Problem otoczki wypukłej — skan Grahama

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

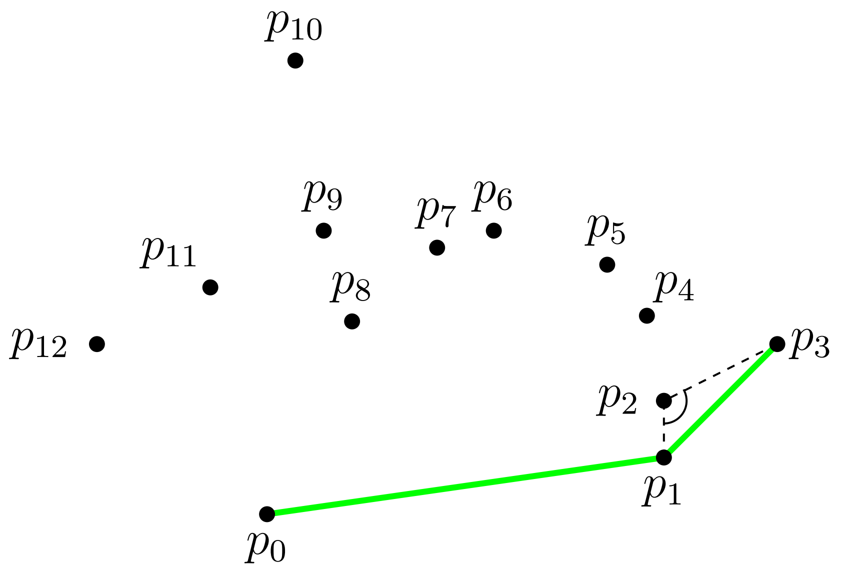


Idea skanu Grahama (1972)

- Obracamy półprostą wokół najniższego punktu.
- Punktami zdarzeń są kolejno napotykane wierzchołki.
- Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

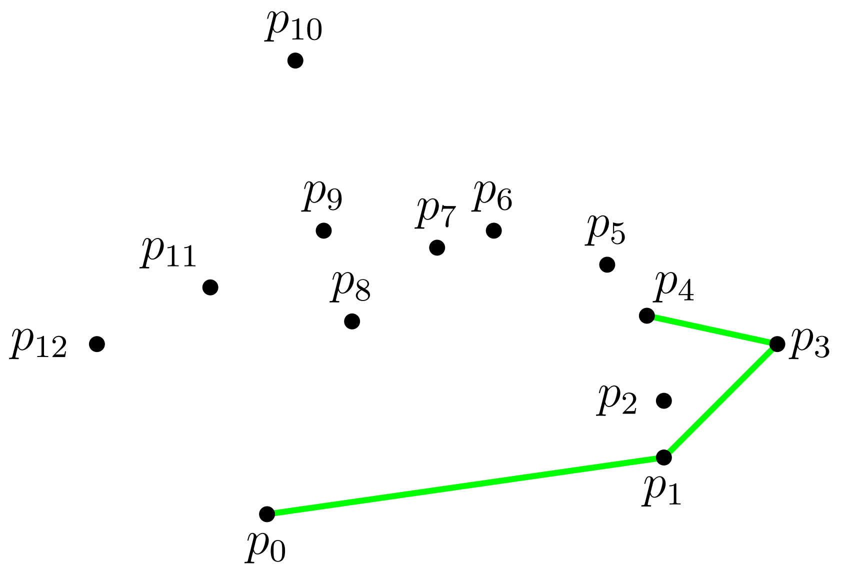


Idea skanu Grahama (1972)

- Obracamy półprostą wokół najniższego punktu.
- Punktami zdarzeń są kolejno napotykane wierzchołki.
- Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

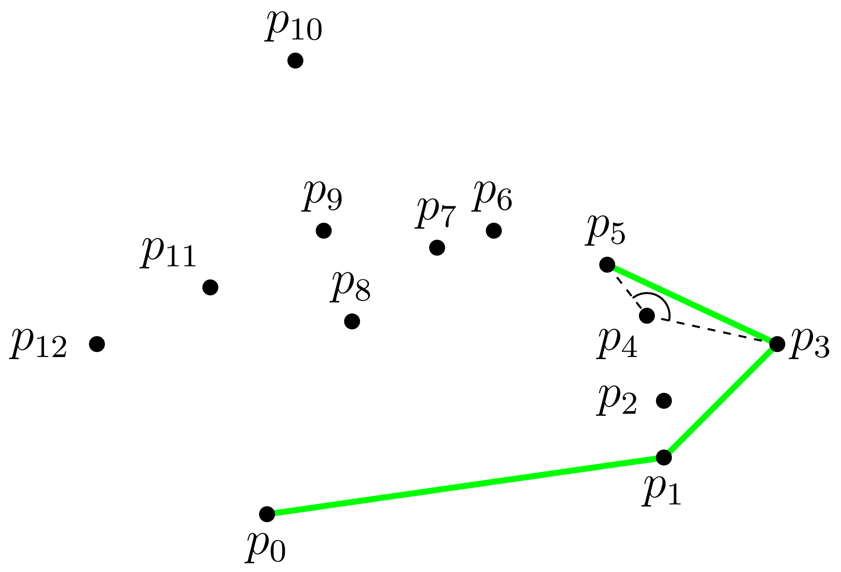


Idea skanu Grahama (1972)

- ▶ Obracamy półprostą wokół najniższego punktu.
- ▶ Punktami zdarzeń są kolejno napotykane wierzchołki.
- ▶ Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- ▶ Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

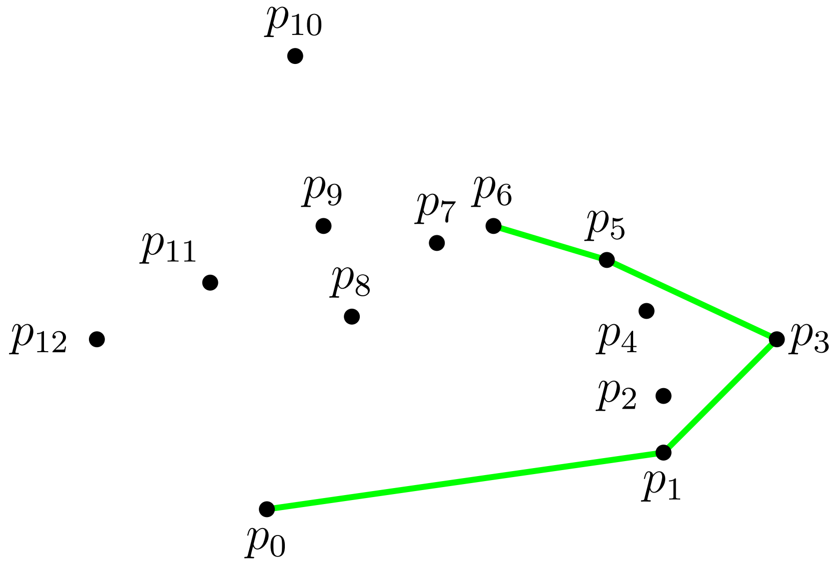


Idea skanu Grahama (1972)

- Obracamy półprostą wokół najniższego punktu.
- Punktami zdarzeń są kolejno napotykane wierzchołki.
- Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

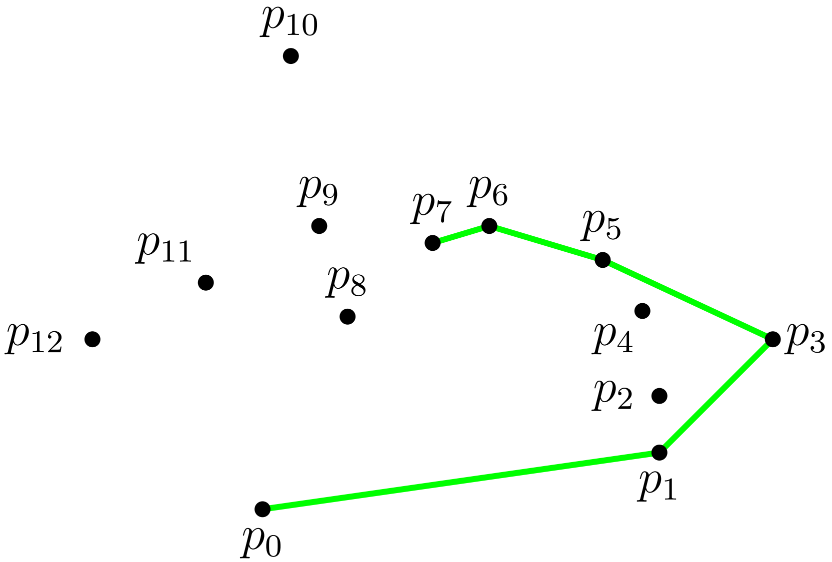


Idea skanu Grahama (1972)

- Obracamy półprostą wokół najniższego punktu.
- Punktami zdarzeń są kolejno napotykane wierzchołki.
- Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

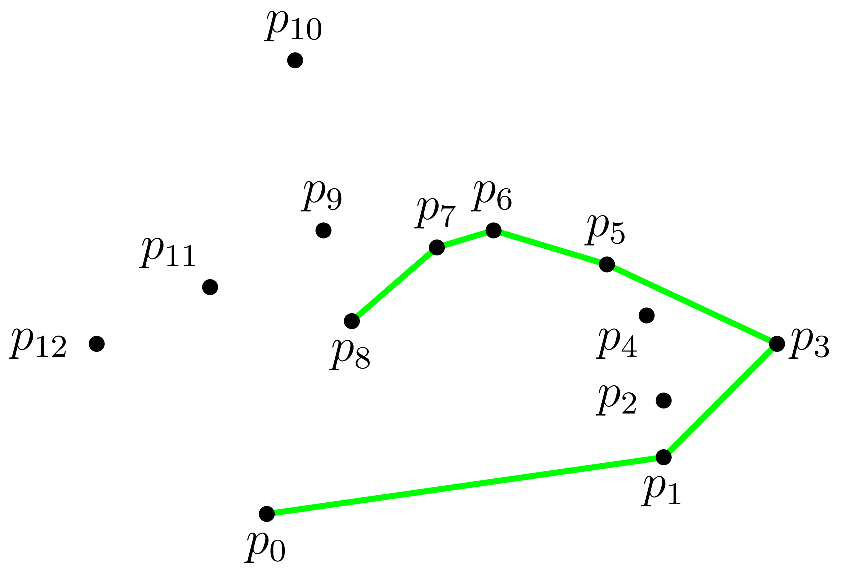


Idea skanu Grahama (1972)

- ▶ Obracamy półprostą wokół najniższego punktu.
- ▶ Punktami zdarzeń są kolejno napotykane wierzchołki.
- ▶ Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- ▶ Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

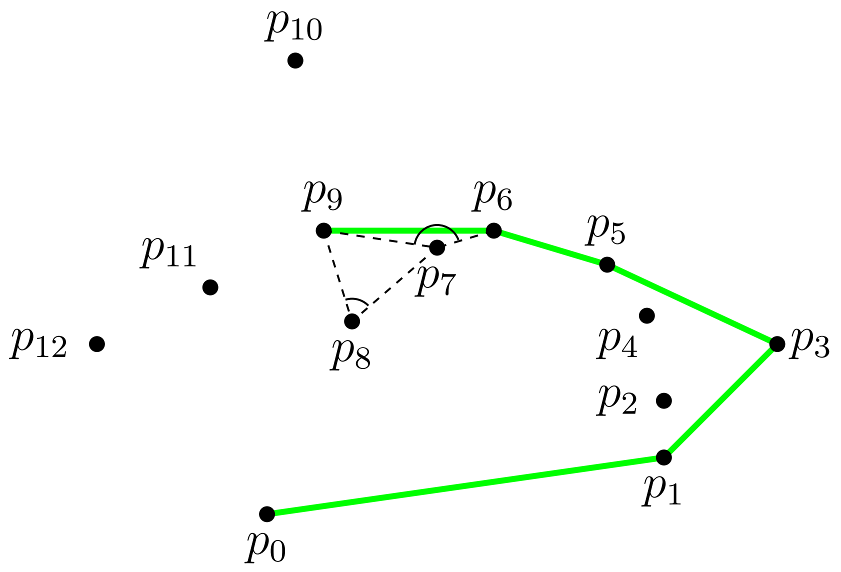


Idea skanu Grahama (1972)

- Obracamy półprostą wokół najniższego punktu.
- Punktami zdarzeń są kolejno napotykane wierzchołki.
- Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

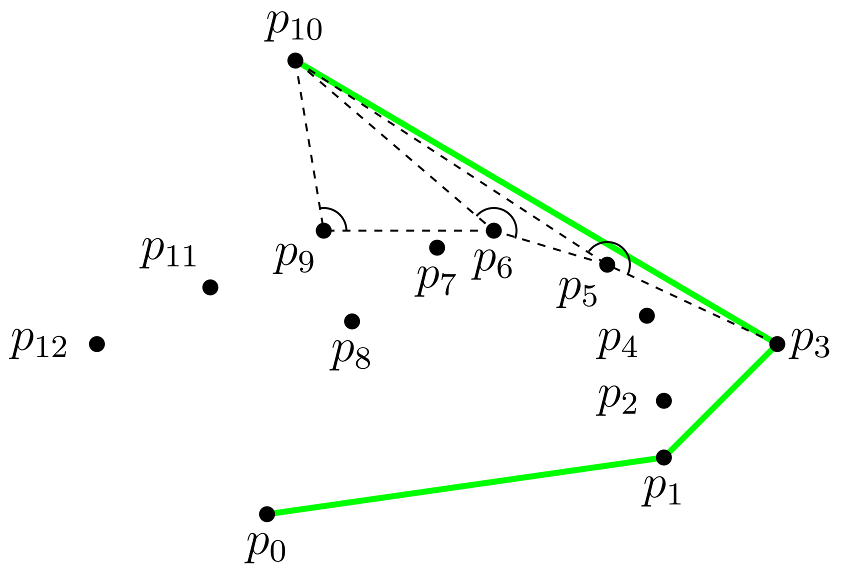


Idea skanu Grahama (1972)

- ▶ Obracamy półprostą wokół najniższego punktu.
- ▶ Punktami zdarzeń są kolejno napotykane wierzchołki.
- ▶ Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- ▶ Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

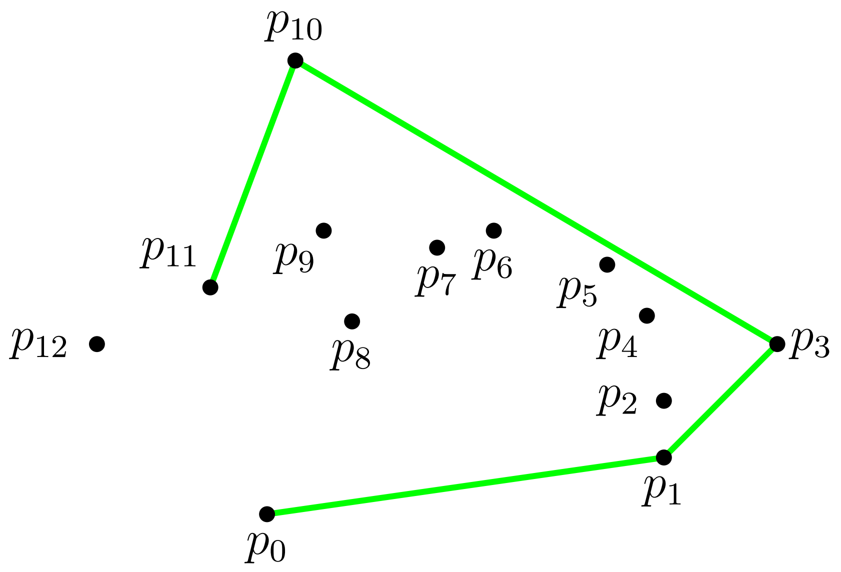


Idea skanu Grahama (1972)

- Obracamy półprostą wokół najniższego punktu.
- Punktami zdarzeń są kolejno napotykane wierzchołki.
- Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

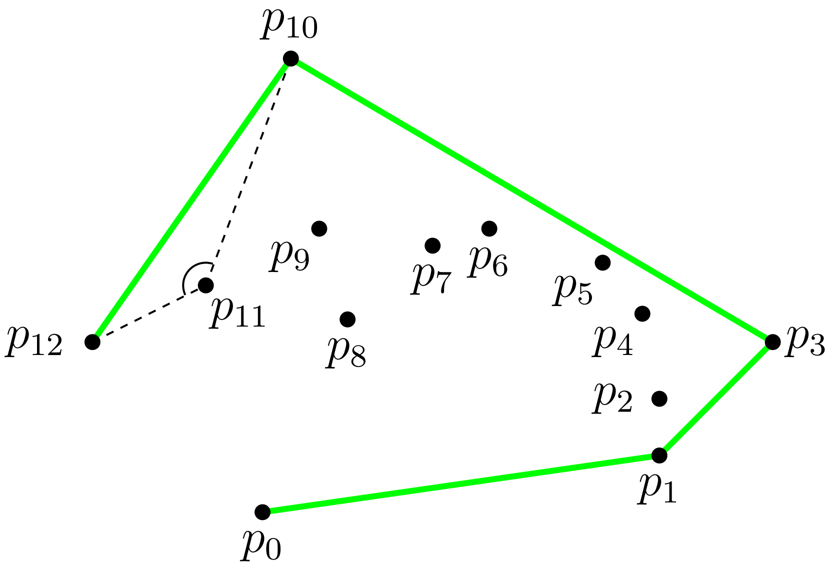


Idea skanu Grahama (1972)

- ▶ Obracamy półprostą wokół najniższego punktu.
- ▶ Punktami zdarzeń są kolejno napotykane wierzchołki.
- ▶ Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- ▶ Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.

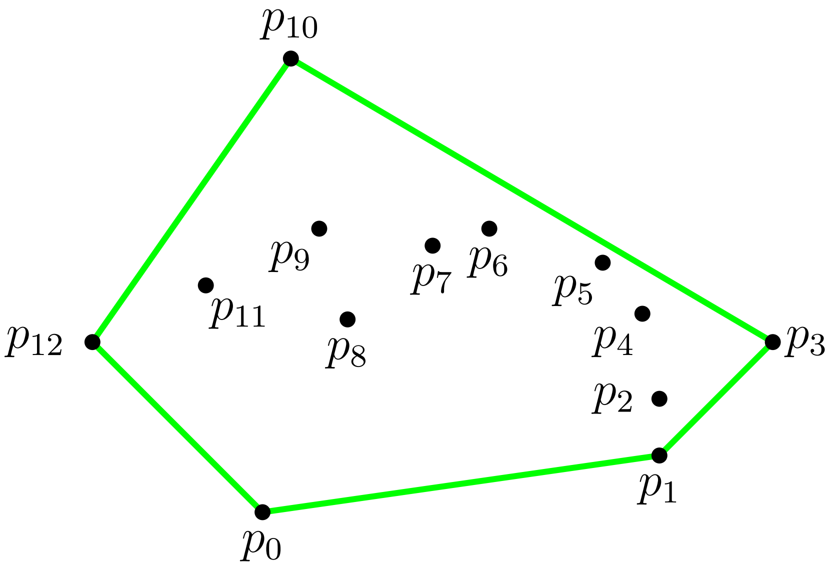


Idea skanu Grahama (1972)

- ▶ Obracamy półprostą wokół najniższego punktu.
- ▶ Punktami zdarzeń są kolejno napotykane wierzchołki.
- ▶ Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- ▶ Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.

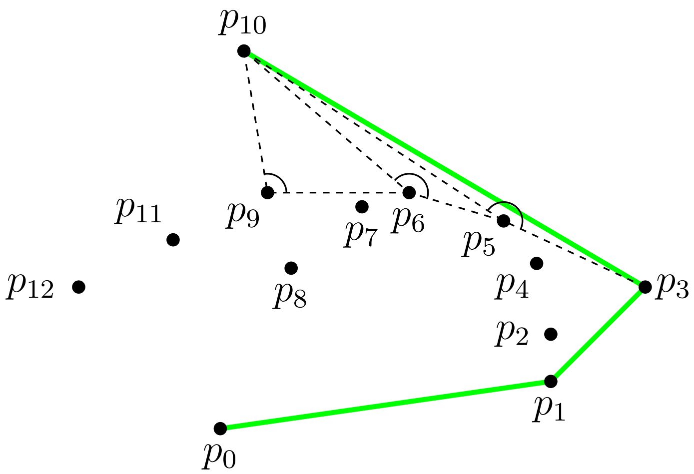
6.5 PROBLEM OTOCZKI WYPUKŁEJ — SKAN GRAHAMA

Otoczka wypukła zbioru punktów $S \subset \mathbb{R}^2$, ozn. $\mathrm{CH}(S)$, to najmniejszy wielokąt wypukły P taki, że każdy punkt ze zbioru S należy do wielokąta P.



Idea skanu Grahama (1972)

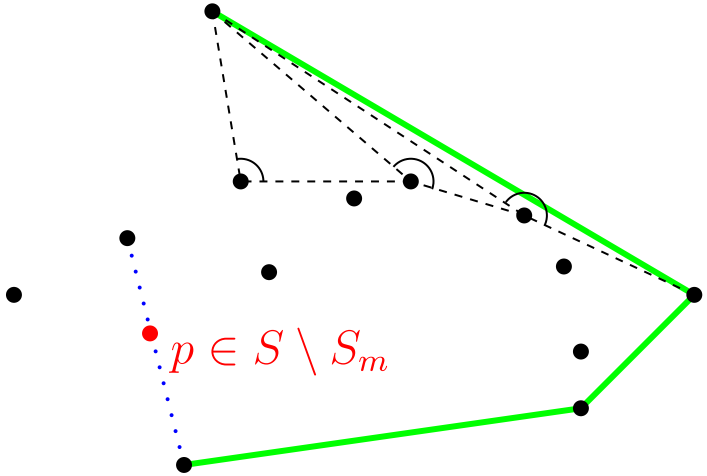
- Obracamy półprostą wokół najniższego punktu.
- Punktami zdarzeń są kolejno napotykane wierzchołki.
- Status miotły przechowywany jest na stosie, który zawiera wierzchołki tworzące otoczkę wypukłą dla wierzchołków do tej pory przetworzonych.
- Jeśli przetwarzany wierzchołek nie tworzy skrętu w prawo ze szczytem stosu oraz z elementem bezpośrednio pod nim, stos musi być uaktualniony.



```
GRAHAMSCAN($S$)
Wejście. Zbiór $n$ punktów $S \subset \mathbb{R}^2$.
Wyjście. Otoczka wypukła CH($S$) zbioru $S$.
1. Niech $p_0$ będzie punktem w $S$ o najmniejszej współrzędnej $y$; w przypadku
   remisu – pierwszym takim punktem z lewej.
2. Niech $p_1, p_2, \ldots, p_m$ będą pozostałymi punktami z $S \setminus \{p_0\}$, posortowanymi
   ze względu na współrzędną kątową względem $p_0$ w kierunku przeciwnym do
   ruchu wskazówek zegara; jeśli więcej niż jeden punkt ma taką samą współ-
   rzędna kątową, usuń wszystkie takie punkty za wyjątkiem punktu położonego
   najdalej od $p_0$.
3. PUSH($p_0$, STOS); PUSH($p_1$, STOS); PUSH($p_2$, STOS);
4. for $i = 3$ to $m$ do
5.     while kąt utworzony przez punkty $p_i$ ,TOP(STOS)
               oraz NEXT-TO-TOP(STOS) nie stanowi skrętu w prawo
6.         do POP(STOS);
7.     PUSH($p_i$, STOS);
8. return STOS.
```

Analiza poprawności algorytmu

- Dla $i = 2, 3, \dots, m$, niech $S_i := \{p_1, p_2, \dots, p_i\}$.
- Otoczki wypukłe CH(S_m) oraz CH(S) są tymi samymi otoczkami.

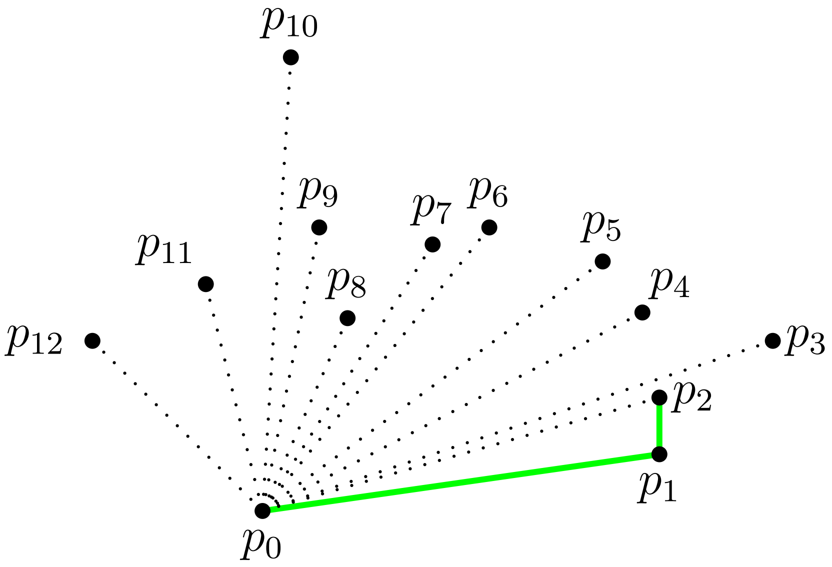


Punkt p nie należy do otoczki CH(S).

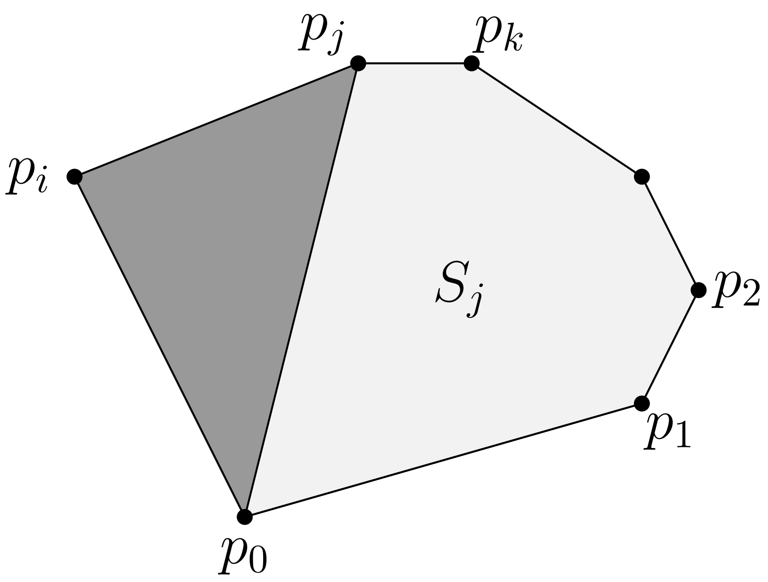
Analiza poprawności algorytmu

- Dla $i = 2, 3, \ldots, m$, niech $S_i := \{p_1, p_2, \ldots, p_i\}$.
- Otoczki wypukłe $\mathrm{CH}(S_m)$ oraz CH(S) są tymi samymi otoczkami.

Niezmiennik. Na początku każdej iteracji pętli **for** w wierszach 4-7 STOS zawiera, patrząc od dołu, wszystkie punkty otoczki $\mathrm{CH}(S_{i-1})$ w kolejności odwrotnej do ruchu wskazówek zegara i tylko te punkty.

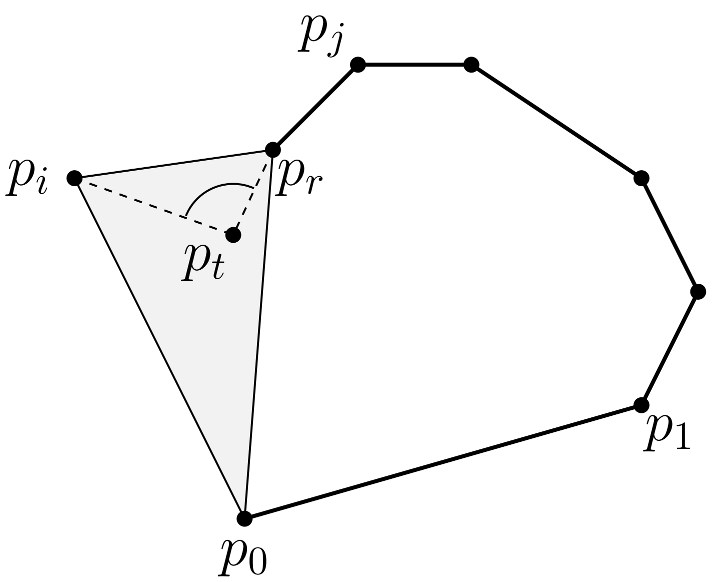


- Niezmiennik jest spełniony przy pierwszym wykonaniu wiersza 4.



- ► Niech p_j będzie punktem na wierzchołku Stosu po zakończeniu pętli **while** w wierszach 5-6, ale przed włożeniem p_i na Stos w wierszu 7, i niech p_k będzie punktem poniżej p_j na Stosie.

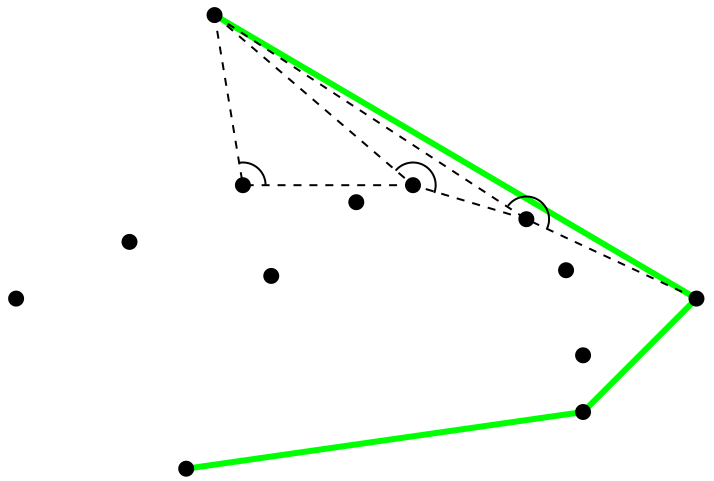
- ► Kąt $\angle p_i p_j p_k$ stanowi skręt w prawo, a tym samym po włożeniu p_i na Stos, stos ten zawiera dokładnie wierzchołki $\mathrm{CH}(S_j \cup \{p_i\})$.



- ► Rozważmy dowolny punkt p_t, który został zdjęty ze stosu podczas i-tej iteracji pętli **for**, i niech p_r będzie punktem tuż poniżej p_t na STOSIE w chwili, kiedy p_t był zdejmowany (p_r to być może p_j).

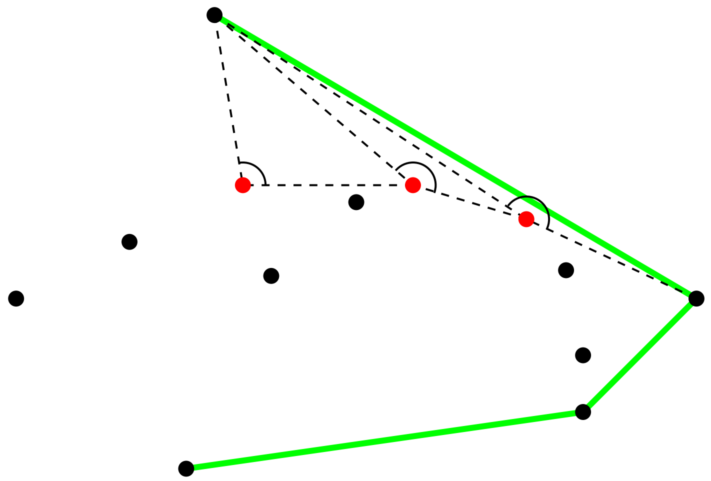
- ► Kąt $\angle p_i p_t p_r$ nie jest skrętem w prawo, punkt p_t leży wewnątrz trójkąta $p_0 p_r p_i$ albo na jego brzegu. Jako że $p_0, p_r, p_i \in S_i$, zatem $\mathrm{CH}(S_i) = \mathrm{CH}(S_i \setminus \{p_t\})$. Jako że p_t był dowolnym punktem zdjętym ze stosu w wierszu 6, otrzymujemy, że $\mathrm{CH}(S_i) = \mathrm{CH}(S_i \setminus \{P_i\})$, gdzie P_i jest zbiorem punktów zdjętych ze STOSU podczas i-tej iteracji pętli `for`. A skoro $S_i \setminus P_i = S_j \cup \{p_i\}$, otrzymujemy

$$\mathrm{CH}(S_i) = \mathrm{CH}(S_j \cup \{p_i\}).$$



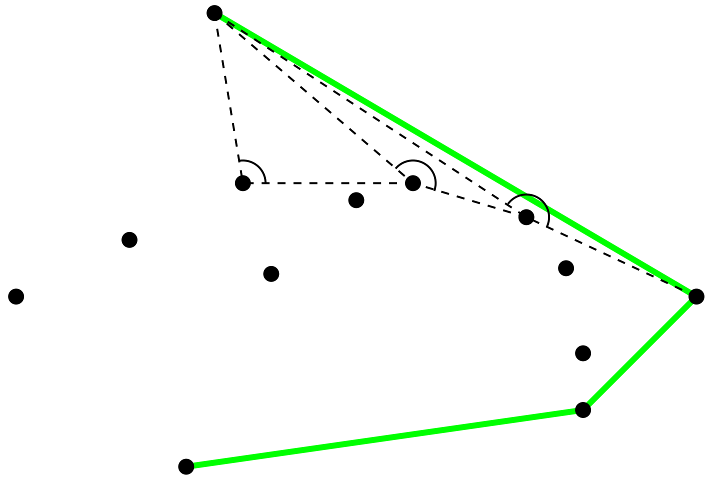
Analiza złożoności czasowej algorytmu

- Wyznaczenie p_0: czas rzędu $\Theta(n)$.
- Sortowanie punktów oraz usunięcie współliniowych punktów: $O(n \log n)$.
- Wstawienie p_0, p_1, p_2: czas $O(1)$.
- Pętla **for** w wierszach 4-7 wykonywana jest $m \leq n - 3$ razy.
  Przy pominięciu czasu dla zagnieżdżonej pętli **while**: czas $O(n)$.



Analiza złożoności czasowej algorytmu

- Wyznaczenie p_0: czas rzędu $\Theta(n)$.
- Sortowanie punktów oraz usunięcie współliniowych punktów: $O(n \log n)$.
- Wstawienie p_0, p_1, p_2: czas $O(1)$.
- Pętla **for** w wierszach 4-7 wykonywana jest $m \leq n - 3$ razy.
  Przy pominięciu czasu dla zagnieżdżonej pętli **while**: czas $O(n)$.
- Całkowity czas wykonywania pętli **while**: $O(n)$.
  - ↳ Każdy punkt wkładany jest na stos dokładnie raz, a na każdą operację PUSH przypada co najwyżej jedna operacja POP. A zatem wykonanych zostaje co najwyżej $m - 2$ operacji POP.
  - ↳ W każdym przebiegu pętli **while** wykonuje się jedna operacja POP, a zatem tych przebiegów może być co najwyżej $m - 2 \leq n - 3$.



Analiza złożoności czasowej algorytmu

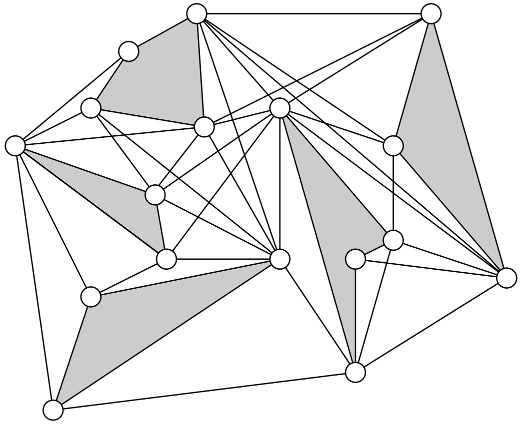
- Wyznaczenie p_0: czas rzędu $\Theta(n)$.
- Sortowanie punktów oraz usunięcie współliniowych punktów: $O(n \log n)$.
- Wstawienie p_0, p_1, p_2: czas $O(1)$.
- Pętla **for** w wierszach 4-7 wykonywana jest $m \leq n - 3$ razy.
  Przy pominięciu czasu dla zagnieżdżonej pętli **while**: czas $O(n)$.
- Całkowity czas wykonywania pętli **while**: $O(n)$.
  - ↳ Każdy punkt wkładany jest na stos dokładnie raz, a na każdą operację PUSH przypada co najwyżej jedna operacja POP. A zatem wykonanych zostaje co najwyżej $m - 2$ operacji POP.
  - ↳ W każdym przebiegu pętli **while** wykonuje się jedna operacja POP, a zatem tych przebiegów może być co najwyżej $m - 2 \leq n - 3$.

Twierdzenie 6.6. Skan Grahama wyznacza otoczkę wypukłą n punktów na płaszczyźnie w czasie rzędu $O(n \log n)$; skan ten używa pamięci liniowej.

6.6* OBLICZANIE GRAFU WIDZIALNOŚCI NA PŁASZCZYŹNIE

Dla danego zbioru S rozłącznych wielokątów na płaszczyźnie *graf widzialności* $G_{\text{widz}}(S)$ jest grafem, którego wierzchołkami są wierzchołki wielokątów z S, a dwa wierzchołki v_1 i v_2 połączone są krawędzią wtedy i tylko wtedy, gdy widzą się nawzajem, tzn. odcinek $\overline{v_1 v_2}$ nie przecina wnętrza żadnego z wielokątów z S.

M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf
Geometria obliczeniowa, rozdział 15, WNT (2007)



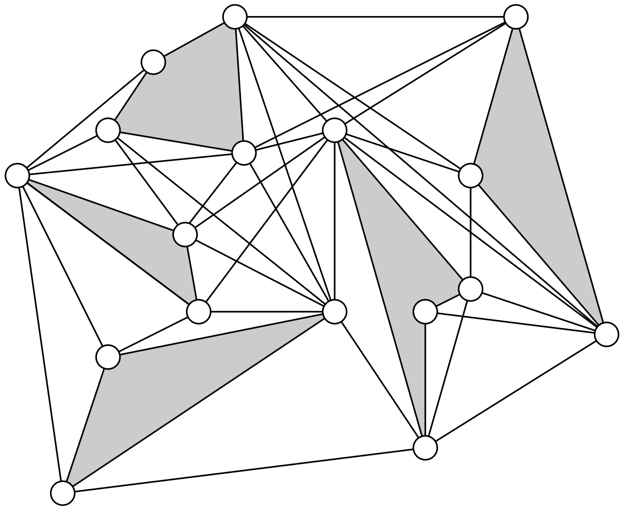
Problem. (Graf widzialności)

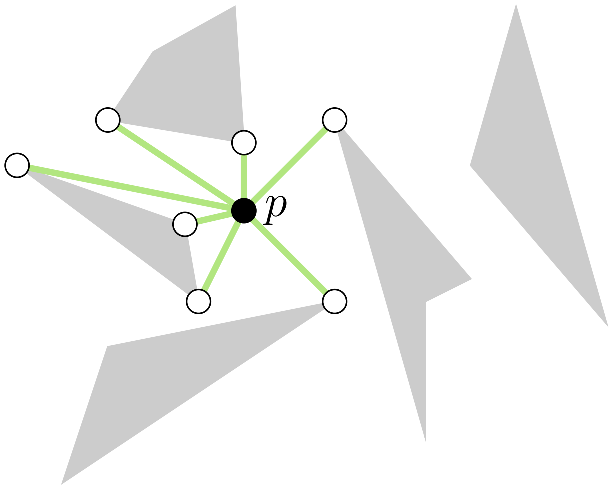
Wejście: *Zbiór S rozłącznych wielokątów na płaszczyźnie $\mathbb{R}^2$.*

Wyjście: *Graf widzialności $G_{\text{widz}}(S)$.*

Algorytm naiwny 1

- Dla każdej pary wierzchołków sprawdzić, czy łączący je odcinek przecina jakikolwiek wielokąt z S. /Złożoność czasowa: $O(n^3)$./





Algorytm naiwny 2

- Dla każdego wierzchołka v wyznacz zbiór wierzchołków widzialnych z v.

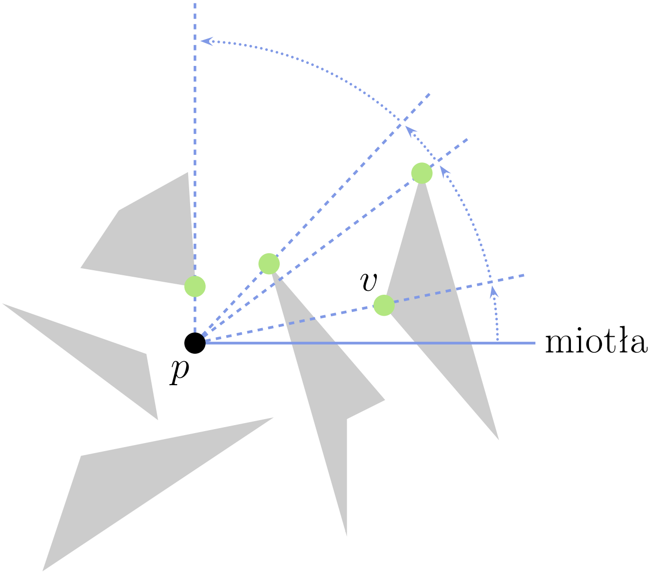
Problem. (Widzialność z punktu p)

Wejście: *Zbiór S rozłącznych wielokątów na $\mathbb{R}^2$ oraz punkt $p \in \mathbb{R}^2$.*

Wyjście: *Zbiór wierzchołków wielokątów widzialnych z p.*

OBLICZANIE WIERZCHOŁKÓW WIDZIALNYCH Z DOWOLNEGO PUNKTU

- Jeśli chcemy sprawdzić, czy jeden wyszczególniony wierzchołek v jest widzialny z p, wówczas w najgorszym przypadku potrzebujemy czasu liniowego.

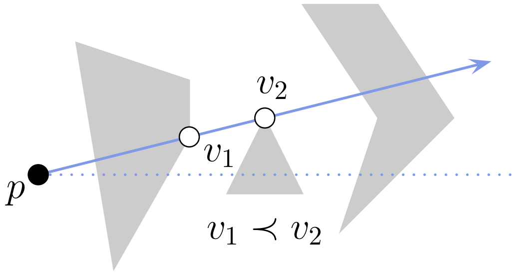


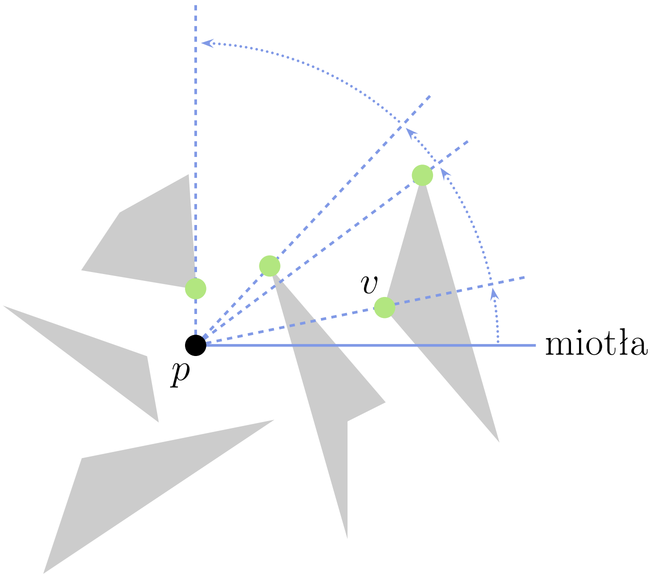
Idea algorytmu zamiatania biegunowego

- Miotła o początku w punkcie p obraca się przeciwnie do ruchu wsk. zegara.
- Status miotły stanowi zbiór przecinanych odcinków i zmienia się on przy rozpatrywaniu kolejnych wierzchołków.
- Napotykając nowy wierzchołek v, sprawdzane jest, czy odcinek $\overline{pv}$ przecinany jest przez krawędź przechowywaną w statusie.
- Ze statusu usuwane są krawędzie incydentne do v, które leżą po stronie zgodnej z ruchem wskazówek zegara, i dodawane te, które leżą po stronie przeciwnej do ruchu wskazówek zegara.

Porządek $\prec$ wierzchołków.

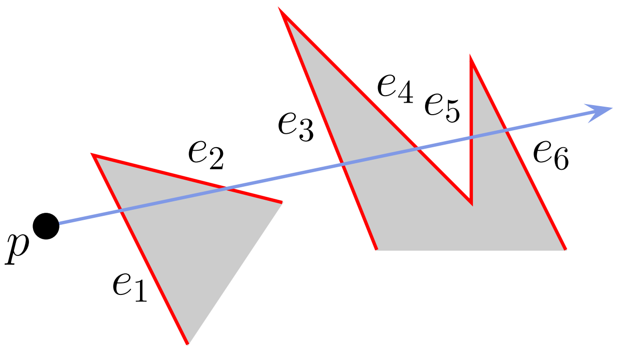
Wierzchołki posortowane są w *porządku biegunowym* względem punktu p; jeśli wierzchołki v_1 i v_2 tworzą ten sam kąt, wówczas $v_1 \prec v_2$ wtedy i tylko wtedy, gdy v_1 leży bliżej punktu p.

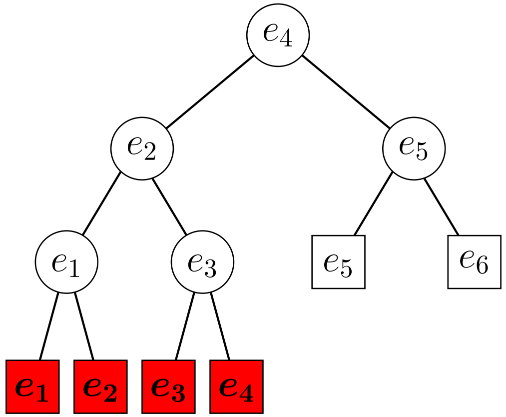




Punkty zdarzeń.

Wierzchołki wielokątów ze zbioru S
posortowane w porządku biegunowym względem p.





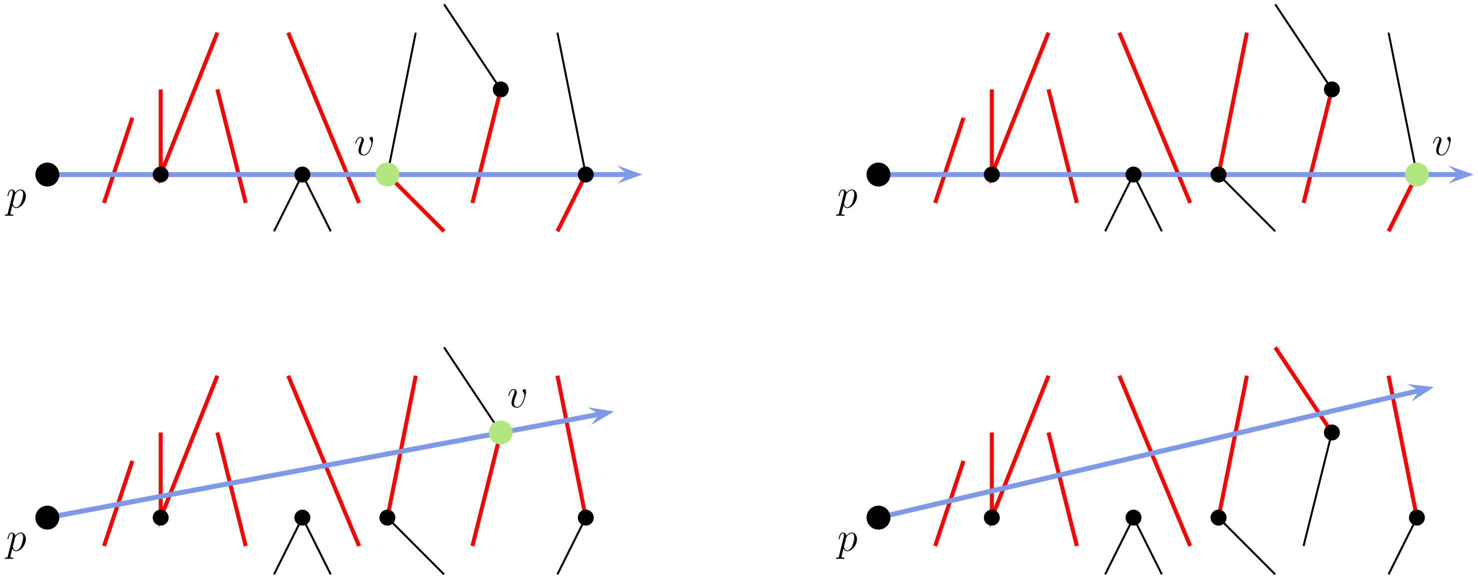
Status miotły. Krawędzie przecinane przez miotłę.

Podczas badania wierzchołków w porządku biegunowym względem p utrzymujemy krawędzie wielokątów przecinane przez miotłę M w zrównoważonym drzewie wyszukiwań binarnych $\mathcal{T}$.

↳ Liście przechowują krawędzie w kolejności przecinania ich przez półprostą M patrząc od jej początku p.

↳ Węzły wewnętrzne wskazują na krawędzie będące prawymi skrajnymi w lewym poddrzewie.

↳ Krawędzie, których oba końce leżą na M, nie muszą być pamiętane.

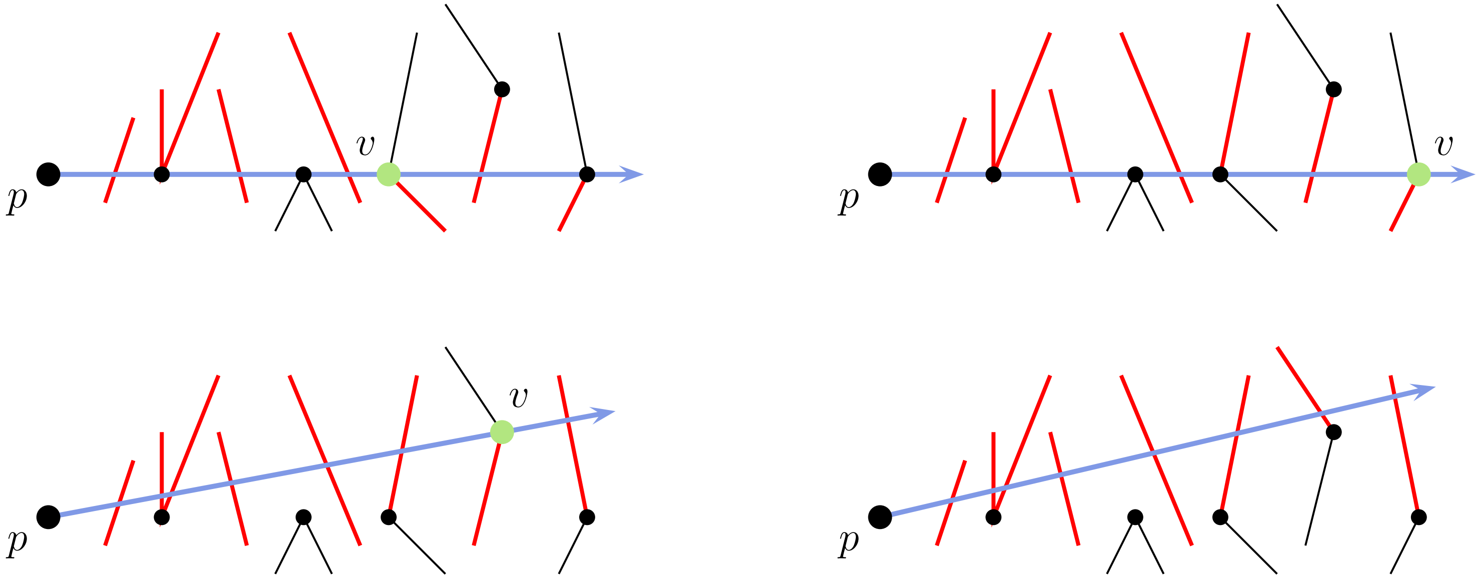


Zmiana statusu. Miotła napotyka nowy wierzchołek v.

↳ Sprawdzenie widzialności między p i v.

↳ Usunięcie ze statusu krawędzi wielokąta incydentnych z v, które leżą po stronie półprostej z p do v zgodnej z ruchem wskazówek zegara.

↳ Wstawienie do statusu krawędzi wielokąta incydentnych z v, które leżą po stronie półprostej z p do v przeciwnej do ruchu wskazówek zegara.

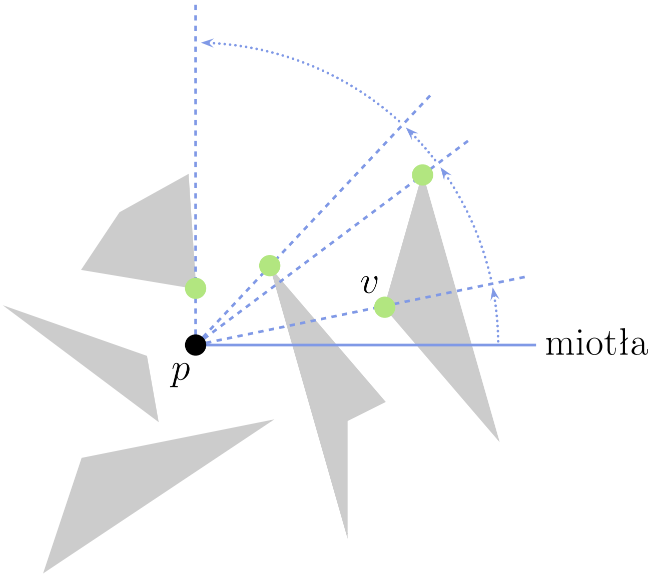


Niezmiennik.

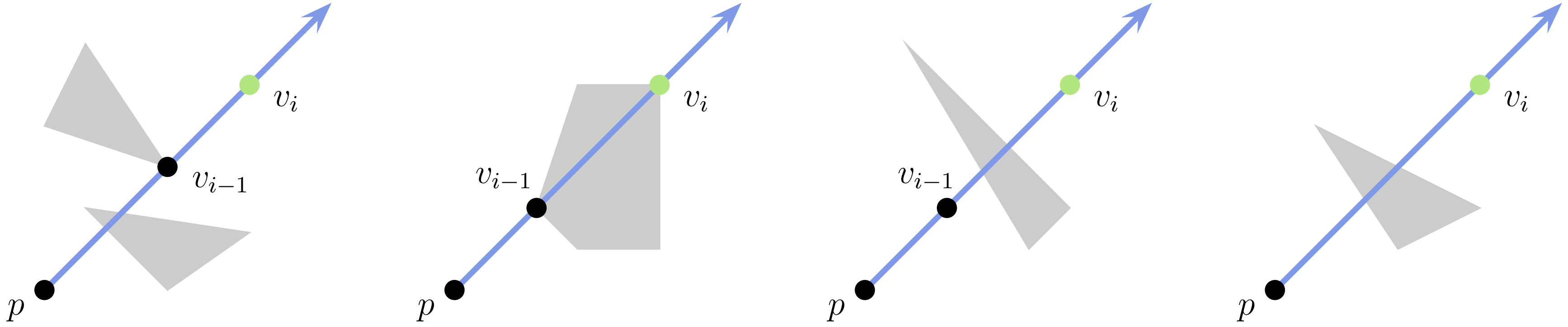
Napotykając nowy wierzchołek v, status na odcinku $\overline{pv}$ zawiera krawędzie *właściwie* przecinane przez miotłę oraz te, które leżą po stronie przeciwnej do ruchu wskazówek zegara, o jednym końcu na odcinku $\overline{pv}$; natomiast na odcinku $\overline{p(+\infty)}$ status zawiera krawędzie *właściwie* przecinane przez miotłę oraz te, które leżą po stronie zgodnej do ruchu wskazówek zegara, o jednym końcu na odcinku $\overline{p(+\infty)}$.

↳ Niezmiennik ten gwarantuje poprawne wyznaczenie widzialności pomiędzy v a kolejnym rozpatrywanym wierzchołkiem.

Idea algorytmu

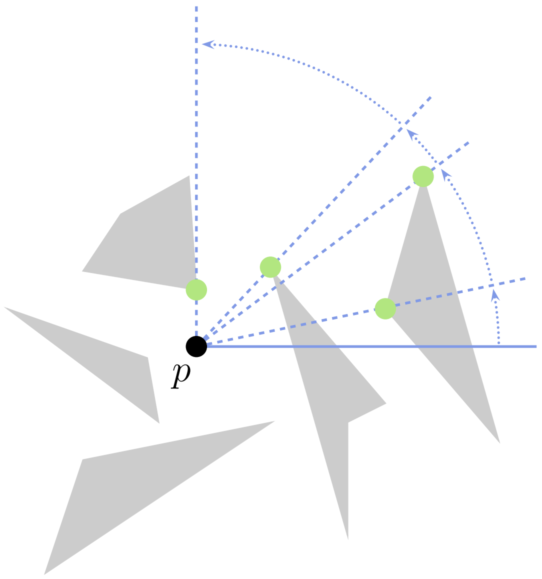


- Niech $v_1, \ldots, v_n$ będzie listą wierzchołków (wszystkich wielokątów) posortowanych biegunowo, tzn. w kierunku przeciwnym do ruchu wskazówek zegara i zgodnie z kątami, jakie półproste od p do każdego z wierzchołków tworzą z dodatnią osią x; porządek wierzchołków o tym samym kącie determinowany jest przez ich odległość do p.
- Wyznaczamy krawędzie wielokątów, które są właściwie przecinane przez poziomą półprostą l o początku w p i zapamiętujemy je w zrównoważonym drzewie wyszukiwań $\mathcal{T}$ w porządku, w jakim są przecinane przez l.
- Dla każdego z wierzchołków $v_1, v_2, \ldots, v_n$ sprawdzamy, czy p widzi v_i.
  - ↳ Usuwamy z $\mathcal{T}$ krawędzie wielokąta incydentne z v_i, które leżą po stronie półprostej z p do v_i zgodnej z ruchem wskazówek zegara.
  - ↳ Wstawiamy do $\mathcal{T}$ krawędzie wielokąta incydentne z v_i, które leżą po stronie półprostej z p do v_i przeciwnej do ruchu wskazówek zegara.



Sprawdzanie widzialności pomiędzy p i v_i

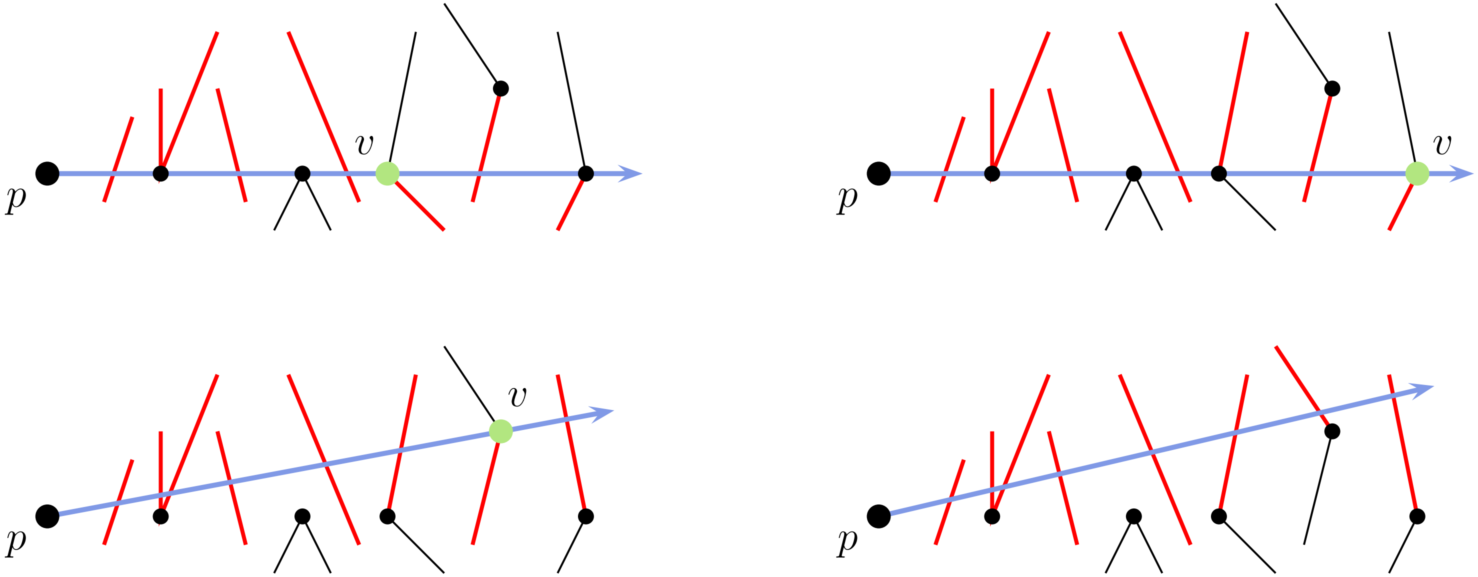
- Jeśli na odcinku $\overline{pv_i}$ leży jakiś inny wierzchołek – spośród wszystkich tych wierzchołków rozważmy wierzchołek w najbliższy v_i. Wówczas, z porządku sortowania, zachodzi $w = v_{i-1}$. A zatem:
  - ↳ Jeśli wierzchołek v_{i-1} nie jest widzialny z p, to v_i też nie.
  - ↳ W przeciwnym wypadku, jeśli v_{i-1} jest widzialny, to zablokowanie widzialności z p do v_i możliwe jest tylko w dwóch przypadkach:
    - $*$ odcinek $\overline{v_{i-1}v_i}$ leży wewnątrz wielokąta;
    - $*$ odcinek $\overline{v_{i-1}v_i}$ przecina (właściwie) krawędź z $\mathcal{T}$.
- Jeśli odcinek $\overline{pv_i}$ przecinany jest we *właściwy* sposób przez jakąś krawędź przechowywaną w statusie $\mathcal{T}$, wówczas p i v_i nie widzą się.
- W przeciwnym wypadku v_i jest widzialny z p.



Analiza złożoności obliczeniowej

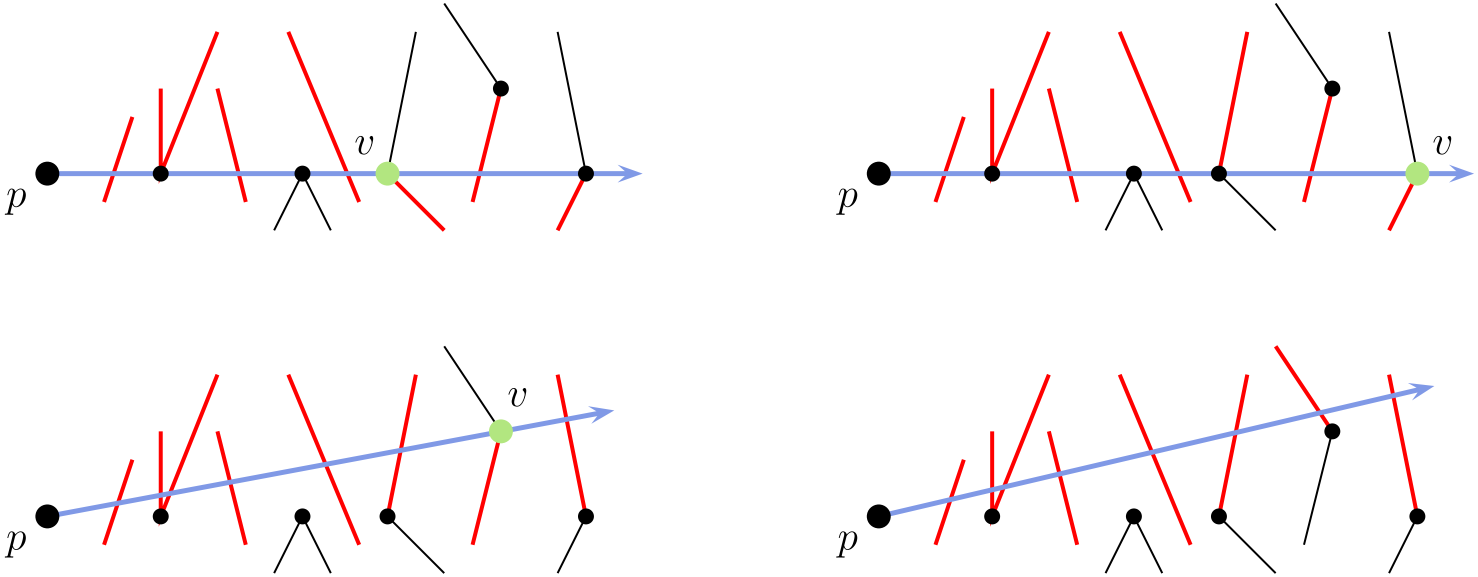
- ▶ Złożoność czasowa rzędu $O(n \log n)$.
  - ↳ Sortowanie w porządku biegunowym: $O(n \log n)$.
  - ↳ Sprawdzenie widzialności pomiędzy p i v_i wymaga $O(1)$ operacji na zrównoważonym drzewie wyszukiwań $\mathcal{T}$, które zajmują czas $O(\log n)$ oraz $O(1)$ geometrycznych testów przeprowadzanych w czasie $O(1)$.
- ▶ Złożoność pamięciowa: rozmiar drzewa $\mathcal{T}$ rzędu $O(n)$.

Analiza poprawności podejścia



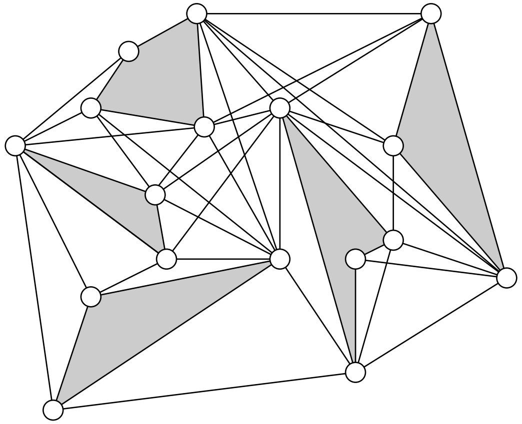
- Poprawność rozstrzygnięcia, czy p widzi v_i, wynika z obserwacji poczynionych przy omawianiu procedury $\textsc{Visible}(p, v_i)$.
- Poprawność drzewa $\mathcal{T}$, tzn. zachowywanie niezmiennika: indukcja.

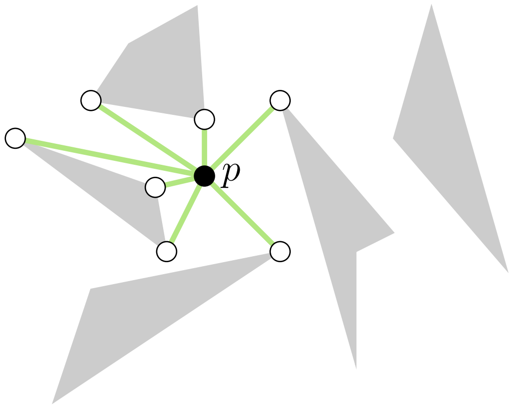
Analiza poprawności podejścia



- Poprawność rozstrzygnięcia, czy p widzi v_i, wynika z obserwacji poczynionych przy omawianiu procedury $\textsc{Visible}(p, v_i)$.
- Poprawność drzewa $\mathcal{T}$, tzn. zachowywanie niezmiennika: indukcja.

Twierdzenie 6.7. (Lee 1978)
Dla danego zbioru rozłącznych wielokątów na płaszczyźnie oraz punktu p problem wyznaczenia wierzchołków widzialnych z p można rozwiązać w czasie rzędu $O(n \log n)$ i pamięci rzędu $O(n)$, gdzie n jest liczbą wierzchołków wszystkich wielokątów.





Algorytm $\textsc{VisibilityGraph}(S)$

Wejście. Zbiór S rozłącznych wielokątów na płaszczyźnie.

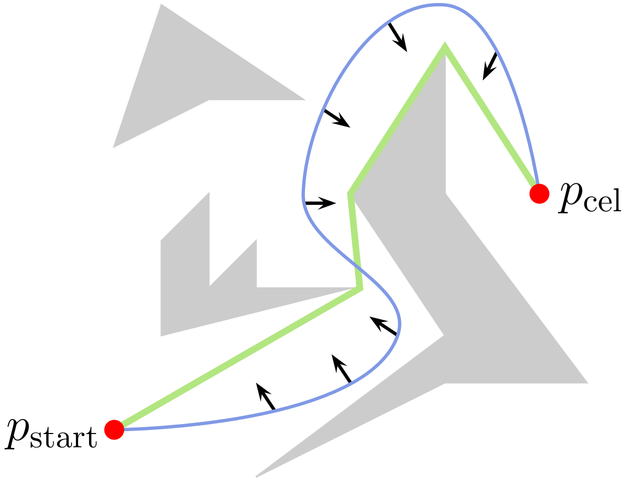
Wyjście. Graf widzialności $G_{\mathrm{widz}}(S)$.

```
1. Inicjuj graf $G = (V, E)$, gdzie $V$ jest zbiorem wszystkich wierzchołków wielokątów z $S$ i $E = \emptyset$.
2. for each $v \in V$ do
   2.1 $W := \textsc{VisibleVertices}(v, S)$.
   2.2 for each $w \in W$ do $E := E \cup \{v, w\}$.
3. return $G$.
```

Twierdzenie 6.8. (Lee 1978)

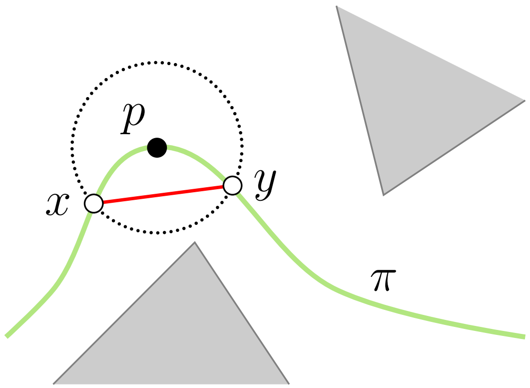
Graf widzialności zbioru rozłącznych wielokątów na płaszczyźnie można wyznaczyć w czasie $O(n^2 \log n)$, gdzie n jest liczbą wierzchołków wszystkich wielokątów.

Problem najkrótszej ścieżki dla robota punktowego



Rozważmy robota punktowego r poruszającego się pośród zbioru S rozłącznych wielokątów prostych na płaszczyźnie. Wielokąty z S nazywane są *przeszkodami*. Przeszkody są zbiorami otwartymi,* a zatem robot może stykać się z nimi. Mamy dane położenie początkowe p_{start} i końcowe p_{cel}, o których zakładamy, że nie znajdują się wewnątrz którejś z przeszkód (czyli znajdują się w tzw. *przestrzeni swobodnej*). Zadanie polega na wyznaczeniu najkrótszej ścieżki z p_{start} do p_{cel}, która nie przecina wnętrza żadnej z przeszkód.

* Przeszkody muszą być zbiorami otwartymi, bo w przeciwnym wypadku – z wyjątkiem sytuacji, w której robot może poruszać się do celu w linii prostej – najkrótsza ścieżka nie istniałaby, gdyż zawsze, przesuwając ścieżkę bliżej przeszkody, możliwe byłoby jej skrócenie.



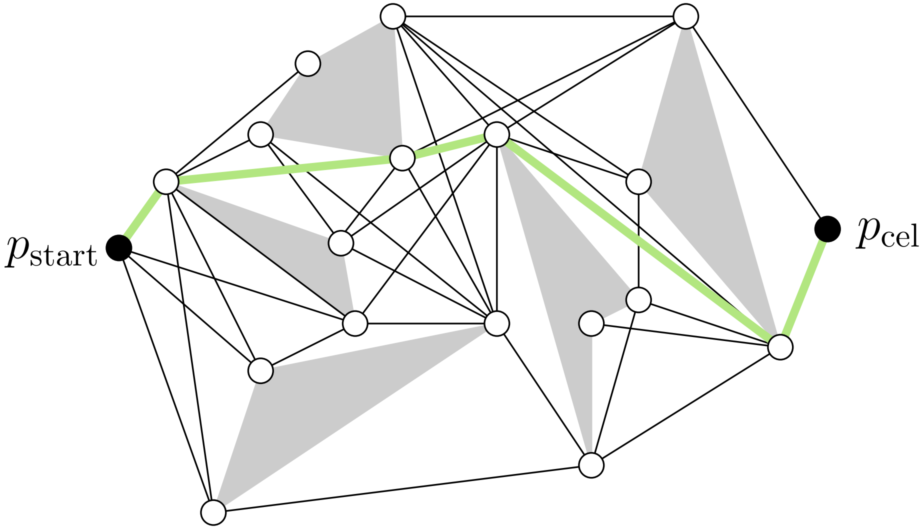
Lemat 6.9. *Dowolna najkrótsza ścieżka między* p_{start} *i* p_{cel} *wśród zbioru* S *rozłącznych przeszkód wielokątnych jest ścieżką wielokątną o wewnętrznych wierzchołkach będących wierzchołkami przeszkód z* S*.*

↳ W przeciwnym wypadku, jako że przeszkody są zbiorami otwartymi, możliwe byłoby skrócenie ścieżki.

Lemat 6.9. *Dowolna najkrótsza ścieżka między* p_{start} *i* p_{cel} *wśród zbioru* S *rozłącznych przeszkód wielokątnych jest ścieżką wielokątną o wewnętrznych wierzchołkach będących wierzchołkami przeszkód z* S*.*

↳ W przeciwnym wypadku, jako że przeszkody są zbiorami otwartymi, możliwe byłoby skrócenie ścieżki.

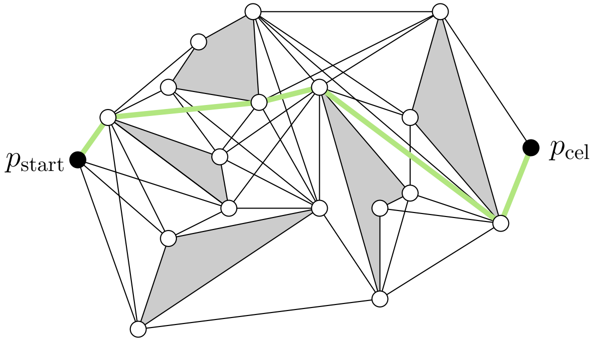
Wniosek 6.10. *Najkrótsza ścieżka między punktami* p_{start} *i* p_{cel} *wśród zbioru* S *rozłącznych przeszkód wielokątnych składa się z krawędzi grafu widzialności* $G_{\text{widz}}(S^*)$*, gdzie* $S^* = S \cup \{p_{\text{start}}, p_{\text{cel}}\}$*.*



Lemat 6.9. *Dowolna najkrótsza ścieżka między p_{start} i p_{cel} wśród zbioru S rozłącznych przeszkód wielokątnych jest ścieżką wielokątną o wewnętrznych wierzchołkach będących wierzchołkami przeszkód z S.*

↳ W przeciwnym wypadku, jako że przeszkody są zbiorami otwartymi, możliwe byłoby skrócenie ścieżki.

Wniosek 6.10. *Najkrótsza ścieżka między punktami p_{start} i p_{cel} wśród zbioru S rozłącznych przeszkód wielokątnych składa się z krawędzi grafu widzialności $G_{\text{widz}}(S^*)$, gdzie $S^* = S \cup \{p_{\text{start}}, p_{\text{cel}}\}$.*



Algorytm $\textsc{ShortestPath}(S, p_{start}, p_{\text{cel}})$

Wejście. Zbiór S rozłącznych wielokątnych przeszkód na płaszczyźnie i dwa punkty p_{start} i p_{cel} w przestrzeni swobodnej.

Wyjście. Najkrótsza bezkolizyjna ścieżka łącząca p_{start} i p_{cel}.

1. $G_{\text{widz}} := \textsc{VisibilityGraph}(S \cup \{p_{start}, p_{\text{cel}}\}$.
2. Przydziel każdej krawędzi $\{u, v\}$ w G_{widz} wagę $d_{\mathcal{E}}(u, v)$.
3. Użyj algorytmu Dijkstry do obliczenia najkrótszej ścieżki między wierzchołkami p_{start} i p_{cel} w grafie G_{widz}.

Twierdzenie 6.11. (Lee 1978) *Najkrótsza ścieżka między dwoma punktami wśród zbioru rozłącznych wielokątnych przeszkód, mających w sumie n wierzchołków, może być wyznaczona w czasie rzędu $O(n^2 \log n)$.*

↳ Krok 1: $O(n^2 \log n)$.
↳ Krok 2: $O(n^2)$.
↳ Krok 3: $O(n \log n + k)$, gdzie $k = O(n^2)$ jest liczbą krawędzi w grafie.

7. ALGORYTMY APROKSYMACYJNE

Algorytmy aproksymacyjne znajdują zastosowanie w przypadku, kiedy czas działania algorytmu dokładnego jest zbyt dużego rzędu: zazwyczaj w przypadku NP-trudnych problemów optymalizacyjnych.

Literatura (m.in.):

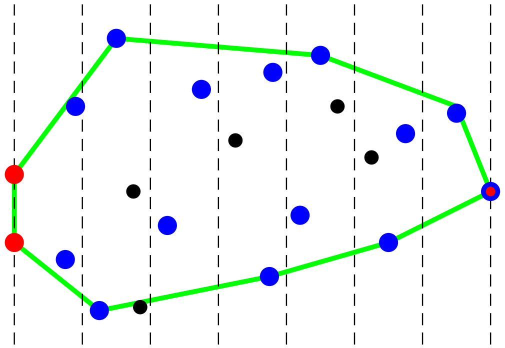
[CLRS05] T. H. Cormen, C. E. Leiserson, R.L. Rivest, C. Stein
Wprowadzenie do algorytmów
rozdziały „NP-zupełność” oraz „Algorytmy aproksymacyjne”
Wydawnictwa naukowo-Techniczne (2005)

[V05] U.V. Vazirani
Algorytmy aproksymacyjne
rozdziały 1, 5, 9 oraz 11
PWN (2005)

[HP06] S. Har-Peled
Wykład „*CS 473g Algorithms*” (2006)
`http://sarielhp.org/teach/notes/algos/`

7.1 Przybliżona otoczka wypukła

- Znajdujemy najmniejszą i największą wartość współrzędnej x.
- Tworzymy pomiędzy tymi dwiema wartościami k pasków/koszy równej szerokości, w których rozmieszczamy n punktów danego zbioru S.
- W każdym pasku znajdujemy (co najwyżej) dwa punkty mające najmniejszą i największa wartość współrzędnej y.
- Wybieramy także dwa punkty ze skrajnymi wartościami współrzędnej x; jeśli istnieje więcej takich punktów, wybieramy te z najmniejszą i największą wartością y.

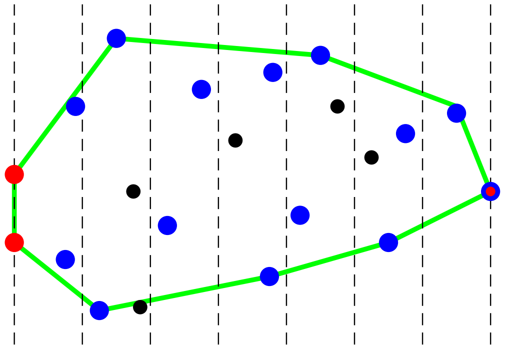


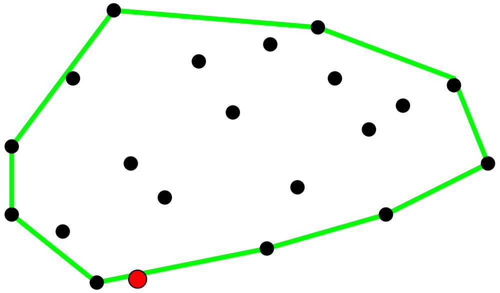
- Niech S^* będzie otrzymanym zbiorem punktów (czas konstrukcji $O(n)$). Zauważmy, że $|S^*| \leq 2k+4$ i punkty z S^* są prawie posortowane ze względu na x, tzn. brak posortowania może wystąpić jedynie w obrębie danego paska/kosza, z których każdy zawiera co najwyżej 4 punkty.

- Tym samym w czasie stałym można posortować każdy z pasków, w konsekwencji otrzymując posortowany zbiór S^* w czasie $O(k)$.

- Znajdujemy otoczkę wypukłą zbioru S^* w czasie $O(k)$.

Twierdzenie 7.1. (Bentley, Faust, Preparata 1982)
Konstrukcja przybliżonej otoczki wypukłej wymaga czasu rzędu $O(n+k)$.





Jeden z punktów znajduje się poza przybliżoną otoczką.

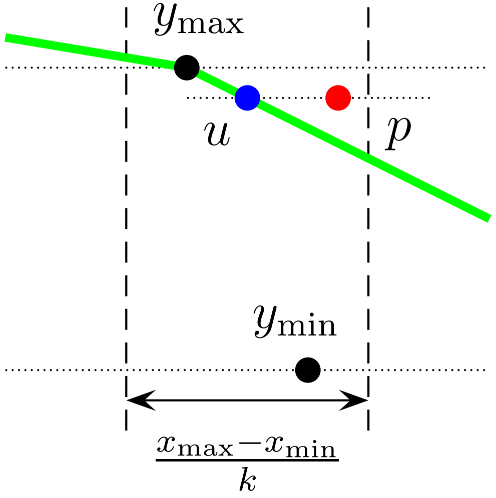
- Niech S^* będzie otrzymanym zbiorem punktów (czas konstrukcji $O(n)$). Zauważmy, że $|S^*| \leq 2k+4$ i punkty z S^* są prawie posortowane ze względu na x, tzn. brak posortowania może wystąpić jedynie w obrębie danego paska/kosza, z których każdy zawiera co najwyżej 4 punkty.

- Tym samym w czasie stałym można posortować każdy z pasków, w konsekwencji otrzymując posortowany zbiór S^* w czasie $O(k)$.

- Znajdujemy otoczkę wypukłą zbioru S^* w czasie $O(k)$.

Twierdzenie 7.1. (Bentley, Faust, Preparata 1982)
Konstrukcja przybliżonej otoczki wypukłej wymaga czasu rzędu $O(n+k)$.

Twierdzenie 7.2. (Bentley, Faust, Preparata 1982)
Dowolny punkt $p \in S$, który nie należy do przybliżonej otoczki wypukłej, znajduje się w odległości co najwyżej $(x_{\max} - x_{\min})/k$ od tej otoczki.



Dowód. Rozważmy pasek, do którego należy punkt p będący poza otoczką przybliżoną. Z definicji i wyboru zbioru S^* mamy, że

$$y_{\min} \leq \mathrm{y}(p) \leq y_{\max}.$$

Jeśli u jest przecięciem otoczki przybliżonej z prostą poziomą przechodzącą przez p, długość odcinka $\overline{pu}$ ogranicza z góry odległość p od otoczki, a sama z kolei jest z góry ograniczona przez szerokość paska równą $(x_{\max} - x_{\min})/k$. ☐

7.2 WSPÓŁCZYNNIK APROKSYMACJI

Załóżmy, że mamy do czynienia z problemem optymalizacyjnym (minimalizacji lub maksymalizacji), w którym każde rozwiązanie ma dodatni koszt.

Definicja. Mówimy, że algorytm aproksymacyjny dla danego problemu ma *współczynnik aproksymacji* $\rho(n)$, jeśli dla dowolnych danych wejściowych rozmiaru n koszt A zwracanego rozwiązania szacuje się przez koszt OPT rozwiązania optymalnego z dokładnością do czynnika $\rho(n)$:

$$\max\left(\frac{A}{\mathrm{OPT}}, \frac{\mathrm{OPT}}{A}\right) \leq \rho(n).$$

Algorytm o współczynniku aproksymacji $\rho(n)$ jest algorytmem *$\rho(n)$-aproksymacyjnym*.

- ▶ Dla problemu maksymalizacji zachodzi $O < A \leq \mathrm{OPT}$, a współczynnik $\frac{\mathrm{OPT}}{A}$ określa, ile razy koszt rozwiązania optymalnego jest większy od kosztu rozwiązania przybliżonego.

- ▶ Dla problemu minimalizacji zachodzi $O < \mathrm{OPT} \leq A$, a współczynnik $\frac{A}{\mathrm{OPT}}$ określa, ile razy koszt rozwiązania przybliżonego jest większy od kosztu rozwiązania optymalnego.

7.3 PAKOWANIE SKRZYŃ

Problem. *Dany jest zbiór n przedmiotów o rozmiarach $a_1, \ldots, a_n \in [0, 1]$. Należy znaleźć sposób zapakowania tych przedmiotów do jak najmniejszej liczby skrzyń o rozmiarze jednostkowym.*

Algorytm FIRSTFIT /m.in. J. D. Ullman 1971/

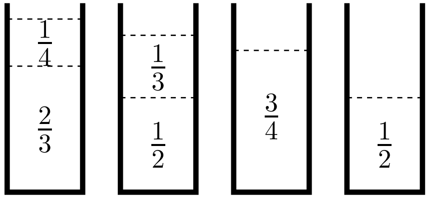
- Przeglądając przedmioty w kolejności, dodawaj je do pierwszego pojemnika na liście B (początkowo pustej), w którym jest jeszcze wystarczająco „miejsca”. Jeżeli nie można przedmiotu dodać do żadnego z pojemników, to dodaj nowy pojemnik do listy i włóż przedmiot właśnie do niego.

Przykład. Rozważmy ciąg sześciu elementów $\frac{2}{3}, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{3}, \frac{1}{2}$.

↳ Pierwszy element $\frac{2}{3}$ zostanie włożony do pierwszej skrzyni B_1.

↳ Jako że $\frac{2}{3} + \frac{1}{2} > 1$, drugi element $\frac{1}{2}$ zostanie włożony do drugiej skrzyni B_2.

↳ ...



↳ Otrzymane rozwiązanie: $B_1 = \{\frac{2}{3}, \frac{1}{4}\}$, $B_2 = \{\frac{1}{2}, \frac{1}{3}\}$, $B_3 = \{\frac{3}{4}\}$, $B_4 = \{\frac{1}{2}\}$.

↳ Optymalne rozwiązanie: $B_1 = \{\frac{2}{3}, \frac{1}{3}\}$, $B_2 = \{\frac{1}{2}, \frac{1}{2}\}$, $B_3 = \{\frac{3}{4}, \frac{1}{4}\}$.

Twierdzenie 7.3. [V05] *Współczynnik aproksymacji* FIRSTFIT *wynosi* 2.

Dowód. Zauważmy, że jeśli po zakończeniu algorytm ten używa m skrzyń, to co najmniej $m-1$ z nich jest zapełnionych więcej niż do połowy – w przeciwnym wypadku, przedmioty z jakichś dwóch skrzyń B_i i B_j, $i < j$, byłyby wszystkie upakowane w skrzyni B_i, a nie w dwóch. A zatem

$$\frac{m-1}{2} < \sum_{i=1}^{n} a_i, \text{ czyli}$$

$$m - 1 < 2\sum_{i=1}^{n} a_i$$

Ponieważ suma rozmiarów przedmiotów jest ograniczeniem dolnym na rozwiązanie optymalne OPT, tj. $\sum_{i=1}^{n} a_i \leq \mathrm{OPT}$, zatem

$$m - 1 < 2 \cdot \mathrm{OPT}, \text{ czyli } m \leq 2 \cdot \mathrm{OPT}.$$

□

- G. Dósa, J. Sgall (STACS'13)
  - ↳ Istnieje nieskończenie wiele instancji, dla których $\mathrm{FF}(I) \geq \lfloor \frac{17}{10} \cdot \mathrm{OPT}(I) \rfloor$.
  - ↳ $\mathrm{FF}(I) \leq \lfloor \frac{17}{10} \cdot \mathrm{OPT}(I) \rfloor$.

Przykład. Rozważmy następujący ciąg elementów ($\varepsilon > 0$ jest dowolnie małe):

↳ 6 elementów postaci $(\frac{1}{6} - 2\varepsilon)$;
↳ 6 elementów postaci $(\frac{1}{3} + \varepsilon)$;
↳ oraz 6 elementów postaci $(\frac{1}{2} + \varepsilon)$.

Algorytm FIRSTFIT użyje 10 pojemników: jeden zostanie wypełniony przez przedmioty rozmiaru $(\frac{1}{6} - 2\varepsilon)$, trzy przez $(\frac{1}{3} + \varepsilon)$, a kolejne sześć przez $(\frac{1}{2} + \varepsilon)$. Tymczasem rozwiązanie optymalne używa 6 pojemników, umieszczając w nich po jednym z przedmiotów każdego rozmiaru.

↳ Powyższą instancję możemy „zwielokrotnić": podać po $12, 18, \ldots$ przedmiotów każdego rozmiaru, odpowiednio dobierając $\varepsilon > 0$.
↳ Otrzymamy w ten sposób całą rodzinę przykładów dla których rozwiązanie zwracane przez APPROXBINPACKING jest $\frac{5}{3}$ razy gorsze od optymalnego.

Modyfikacja przykładu (Johnson, Demers, Ullman, Graham 1974):

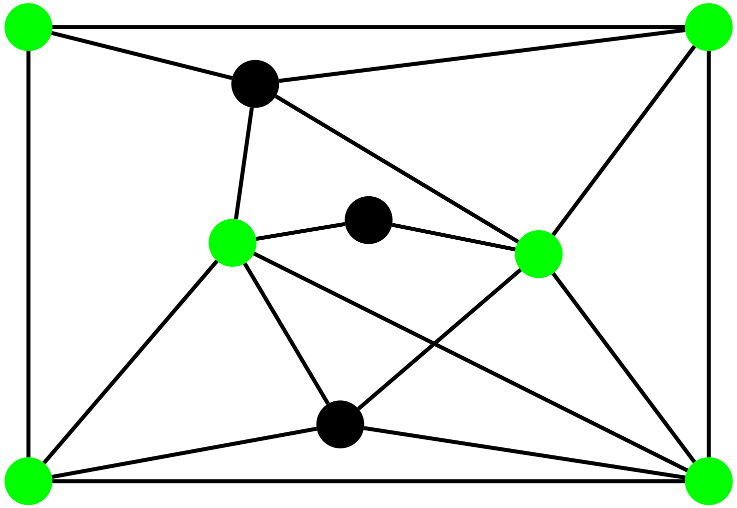
- Istnieją instancje, dla których $\mathrm{FF}(I) > \frac{17}{10} \cdot \mathrm{OPT}(I) - 8$.

Twierdzenie 7.4. [V05] *Dla każdego $\varepsilon > 0$ nie istnieje wielomianowy algorytm o współczynniku $3/2 - \varepsilon$ dla problemu pakowania skrzyń, o ile $\mathrm{P} \neq \mathrm{NP}$.*

7.4 PROBLEM POKRYCIA WIERZCHOŁKOWEGO

Pokryciem wierzchołkowym grafu nieskierowanego $G = (V, E)$ jest podzbiór $V' \subseteq V$ taki, że jeśli $\{u, v\} \in E$, to albo $u \in V'$, albo $v \in V'$ (albo obydwa); liczba $|V'|$ wierzchołków to *rozmiar* pokrycia.

Problem. *Dla danego grafu $G = (V, E)$ wyznaczyć pokrycie wierzchołkowe o najmniejszym rozmiarze.*



Pokrycie wierzchołkowe rozmiaru 6.

7.4 PROBLEM POKRYCIA WIERZCHOŁKOWEGO

Pokryciem wierzchołkowym grafu nieskierowanego $G = (V, E)$ jest podzbiór $V' \subseteq V$ taki, że jeśli $\{u, v\} \in E$, to albo $u \in V'$, albo $v \in V'$ (albo obydwa); liczba $|V'|$ wierzchołków to *rozmiar* pokrycia.

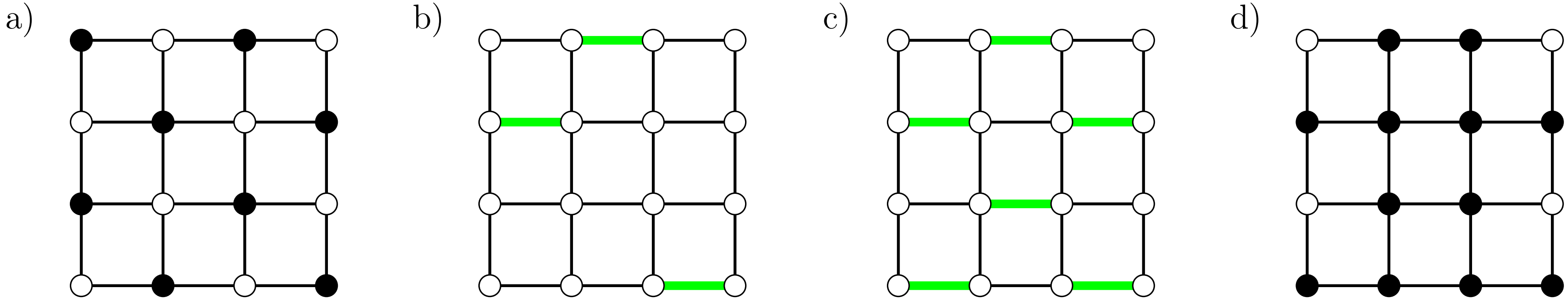
Problem. *Dla danego grafu $G = (V, E)$ wyznaczyć pokrycie wierzchołkowe o najmniejszym rozmiarze.*

Algorytm APPROXVERTEXCOVER($G = (V, E)$) /m.in. Gavril 1974/

```
1. $A := \emptyset$;
2. $E' := E$;
3. while $E' \neq \emptyset$ do
        3.1 Wybierz dowolną krawędź $\{u, v\} \in E'$;
        3.2 $A := A \cup \{u, v\}$;
        3.3 Usuń z $E'$ wszystkie krawędzie incydentne z $u$ lub $v$;
4. return $A$.
```

Przypomnijmy, że *skojarzeniem* w grafie $G = (V, E)$ nazywamy taki podzbiór krawędzi $M \subseteq E$, że żadne dwie krawędzie z M nie mają wspólnego końca.

Przykład.



(a) Optymalne pokrycie wierzchołkowe składa się z 8 elementów. (b-c) Skojarzenie składające się z trzech krawędzi oraz jego uzupełnienie do maksymalnego skojarzenia. (d) Otrzymane rozwiązanie o 12 wierzchołkach.

Poprawność algorytmu.

- Zbiór A pokrywa wszystkie krawędzie, ponieważ usuwane są zawsze tylko te krawędzie, których jeden z końców został dodany do A, a pętla w algorytmie wykonywana jest tak długo, aż każda z krawędzi E zostanie pokryta przez jakiś wierzchołek ze zbioru A.

Złożoność czasowa.

- Czas działania powyższego algorytmu jest wielomianowy; jeśli użyjemy listy sąsiedztwa do reprezentacji grafu, można osiągnąć czas rzędu $O(|V| + |E|)$.

Twierdzenie 7.5. (m.in. Gavril 1974)
Współczynnik aproksymacji ApproxVertexCover *wynosi* 2.

Dowód. Niech M oznacza zbiór krawędzi wybranych w kroku 3.1. Zauważmy, że żadne dwie krawędzie z M nie mają wspólnego wierzchołka – ponieważ za każdym wyborem krawędzi e w kroku 3.3 usuwane są z E' wszystkie krawędzie z nią incydentne – a tym samym M tworzy skojarzenie w grafie G. Jako że w rozwiązaniu optymalnym problemu pokrycia wierzchołkowego żaden z wierzchołkow nie może pokrywać dwóch krawędzi ze skojarzenia M, zachodzi $|M| \leq \mathrm{OPT}$. Jako że $|A| = 2|M|$, otrzymujemy $|A| \leq 2\mathrm{OPT}$. ☐

„Trudny" przypadek dla ApproxVertexCover: pełne dwudzielne grafy $K_{n,n}$.

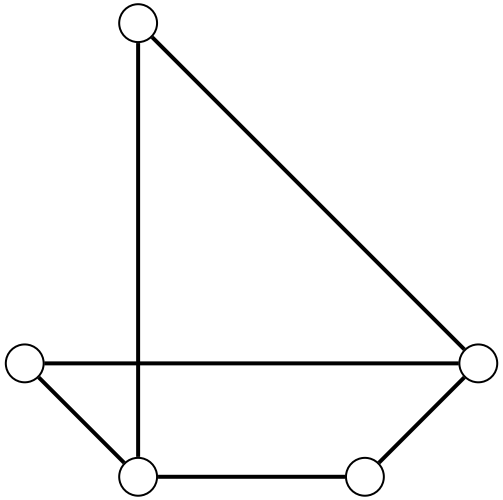
- I. Dinur, S. Safra: On the hardness of approximating minimum vertex cover
  Annals of Mathematics 162 (1), 439-485 (2005)

- S. Khot, O. Regev: Vertex cover might be hard to approximate to within $2-\epsilon$
  Journal of Computer and System Sciences 74 (3), 335-349 (2008)

- Algorytm o współczynniku $2 - \Theta(\frac{1}{\sqrt{\log n}})$
  G. Karakostas: A better approximation ratio for the vertex cover problem
  Journal ACM Transactions on Algorithms 5(4) #41 (2009)

7.5 PROBLEM KOMIWOJAŻERA (TSP)

Problem. *Dla grafu pełnego* $G = (V, E, w)$ *o funkcji kosztu* $w\colon E \to \mathcal{R}^+ \cup \{0\}$ *znaleźć najtańszy cykl przechodzący przez każdy z wierzchołków dokładnie raz.*



CYKL HAMILTONA

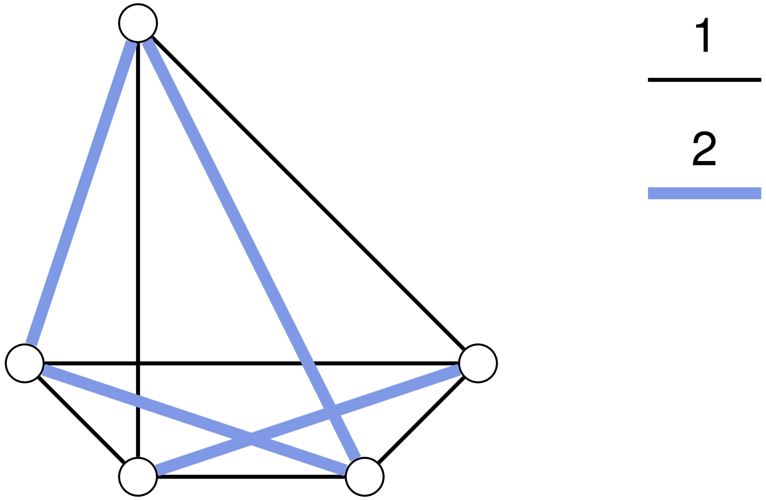
$\propto_{wielomianowo}$

PROBLEM KOMIWOJAŻERA

▶ *Problem komiwojażera jest problemem NP-trudnym.* (Garey, Johnson 1979)

7.5 PROBLEM KOMIWOJAŻERA (TSP)

Problem. *Dla grafu pełnego* $G = (V, E, w)$ *o funkcji kosztu* $w\colon E \to \mathcal{R}^{+} \cup \{0\}$ *znaleźć najtańszy cykl przechodzący przez każdy z wierzchołków dokładnie raz.*



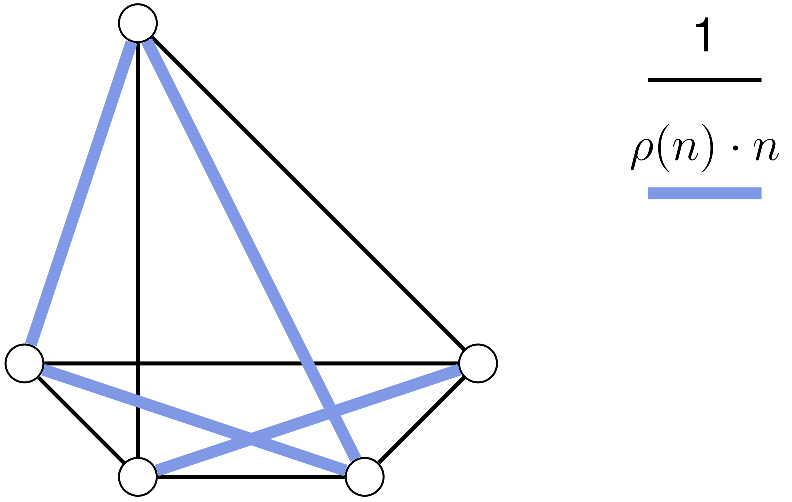
CYKL HAMILTONA

$\propto_{wielomianowo}$

PROBLEM KOMIWOJAŻERA

- ▶ *Problem komiwojażera jest problemem NP-trudnym.* (Garey, Johnson 1979)

OBSERWACJA. *Dla żadnej funkcji $\rho(n)$ obliczalnej w czasie wielomianowym nie istnieje algorytm $\rho(n)$-aproksymacyjny dla TSP, o ile $\mathrm{P} \neq \mathrm{NP}$. $(\rho(n) > 1)$*

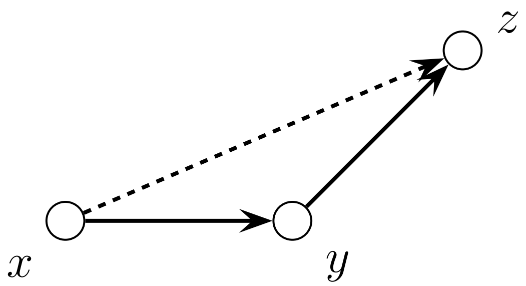


↳ Każdy cykl niezawierający oryginalnej krawędzi grafu wejściowego G ma koszt przynajmniej $\rho(n) \cdot n + (n-1) > \rho(n) \cdot n$.

↳ Jeśli zwracane rozwiązanie ma koszt $\leq \rho(n) \cdot n$, to G jest hamiltonowski.

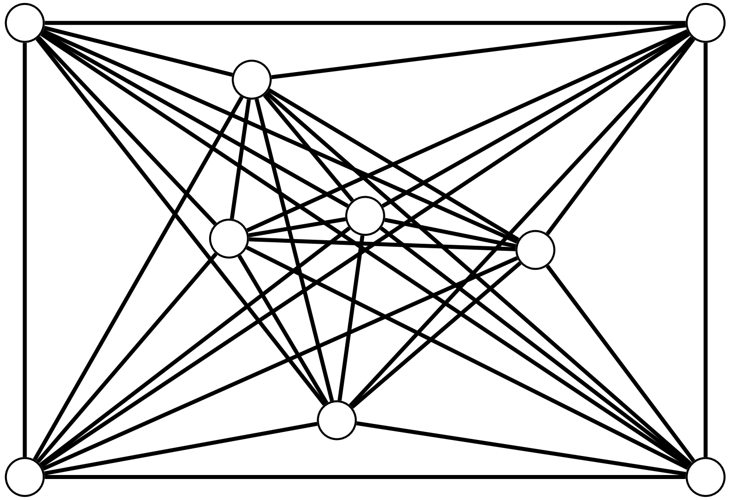
METRYCZNY PROBLEM KOMIWOJAŻERA

Ograniczamy się do grafów, w których spełniona jest *nierówność trójkąta*; problem nadal pozostaje NP-trudny.



$$w(x,z) \leq w(x,y) + w(y,z)$$

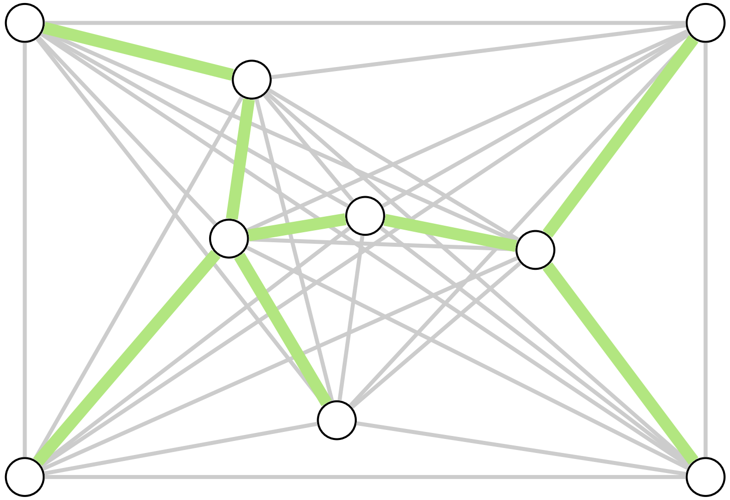
Twierdzenie 7.6. (Christofides 1976)
Dla metrycznego TSP istnieje algorytm $\frac{3}{2}$-przybliżony.



Algorytm ApproxTSP

1. Znajdź minimalne drzewo spinające T grafu G.
2. Znajdź najtańsze doskonałe skojarzenie M w zbiorze wierzchołków nieparzystego stopnia w T. Dodaj M do drzewa T; otrzymany graf jest grafem eulerowskim.
3. Znajdź cykl Eulera C_E w tym grafie.
4. Zwróć trasę C, która odwiedza wszystkie wierzchołki grafu G w kolejności ich wystąpień w cyklu C_E.

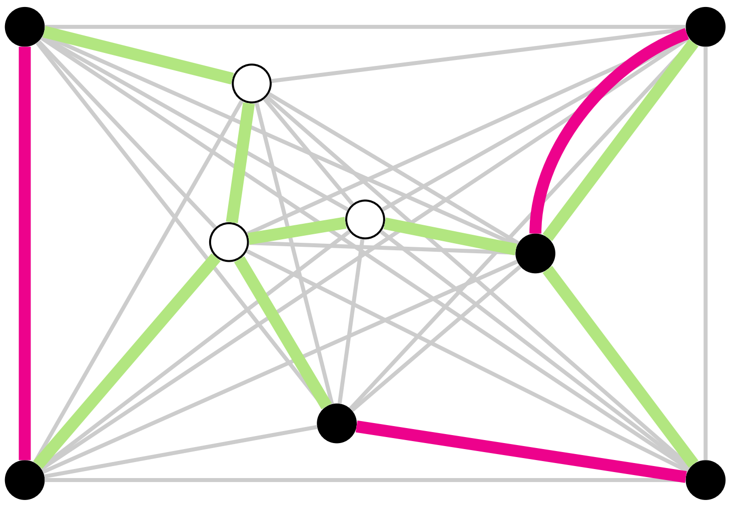
- Algorytm ApproxTSP jest algorytmem wielomianowym, gdyż najtańsze skojarzenie można znaleźć w czasie wielomianowym.



Algorytm ApproxTSP

1. Znajdź minimalne drzewo spinające T grafu G.
2. Znajdź najtańsze doskonałe skojarzenie M w zbiorze wierzchołków nieparzystego stopnia w T. Dodaj M do drzewa T; otrzymany graf jest grafem eulerowskim.
3. Znajdź cykl Eulera C_E w tym grafie.
4. Zwróć trasę C, która odwiedza wszystkie wierzchołki grafu G w kolejności ich wystąpień w cyklu C_E.

- Algorytm ApproxTSP jest algorytmem wielomianowym, gdyż najtańsze skojarzenie można znaleźć w czasie wielomianowym.

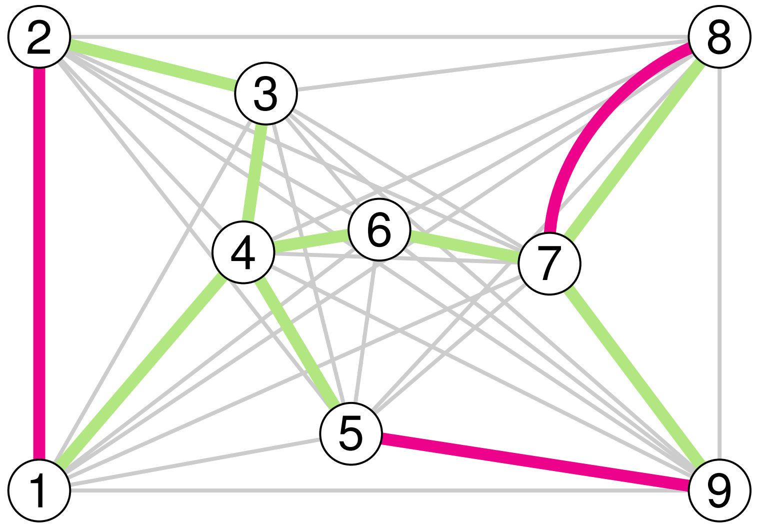


Algorytm ApproxTSP

1. Znajdź minimalne drzewo spinające T grafu G.
2. Znajdź najtańsze doskonałe skojarzenie M w zbiorze wierzchołków nieparzystego stopnia w T. Dodaj M do drzewa T; otrzymany graf jest grafem eulerowskim.
3. Znajdź cykl Eulera C_E w tym grafie.
4. Zwróć trasę C, która odwiedza wszystkie wierzchołki grafu G w kolejności ich wystąpień w cyklu C_E.

- ▶ Algorytm ApproxTSP jest algorytmem wielomianowym, gdyż najtańsze skojarzenie można znaleźć w czasie wielomianowym.

cykl Eulera: 1-2-3-4-6-7-8-7-9-5-4-1



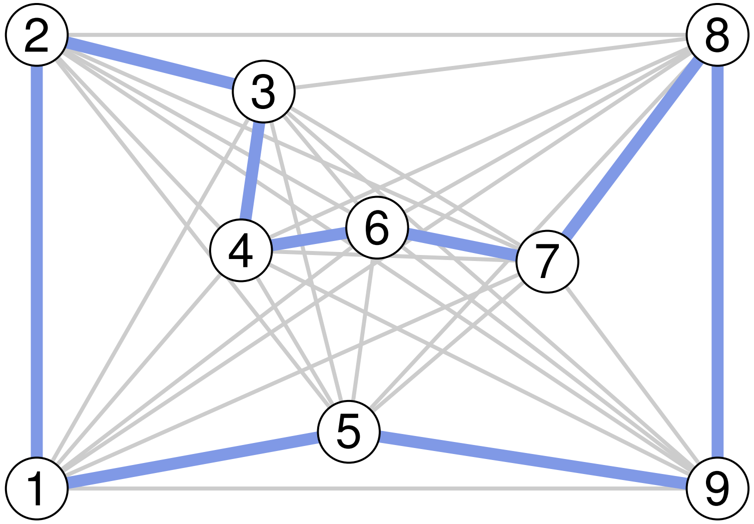
Algorytm ApproxTSP

1. Znajdź minimalne drzewo spinające T grafu G.
2. Znajdź najtańsze doskonałe skojarzenie M w zbiorze wierzchołków nieparzystego stopnia w T. Dodaj M do drzewa T; otrzymany graf jest grafem eulerowskim.
3. Znajdź cykl Eulera C_E w tym grafie.
4. Zwróć trasę C, która odwiedza wszystkie wierzchołki grafu G w kolejności ich wystąpień w cyklu C_E.

- ▶ Algorytm ApproxTSP jest algorytmem wielomianowym, gdyż najtańsze skojarzenie można znaleźć w czasie wielomianowym.

cykl Eulera: 1-2-3-4-6-7-8-7-9-5-4-1

cykl Hamiltona:1-2-3-4-6-7-8-9-5-1



Algorytm ApproxTSP

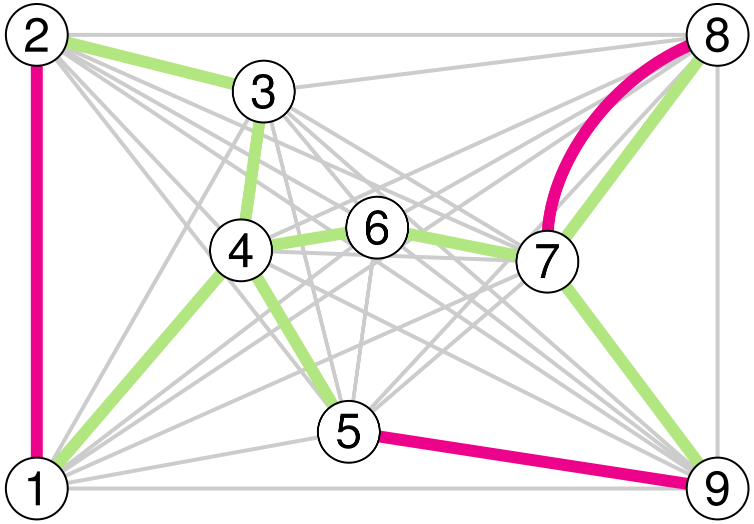
1. Znajdź minimalne drzewo spinające T grafu G.
2. Znajdź najtańsze doskonałe skojarzenie M w zbiorze wierzchołków nieparzystego stopnia w T. Dodaj M do drzewa T; otrzymany graf jest grafem eulerowskim.
3. Znajdź cykl Eulera C_E w tym grafie.
4. Zwróć trasę C, która odwiedza wszystkie wierzchołki grafu G w kolejności ich wystąpień w cyklu C_E.

- Algorytm ApproxTSP jest algorytmem wielomianowym, gdyż najtańsze skojarzenie można znaleźć w czasie wielomianowym.

cykl Eulera: 1-2-3-4-6-7-8-7-9-5-4-1

cykl Hamiltona:1-2-3-4-6-7-8-9-5-1

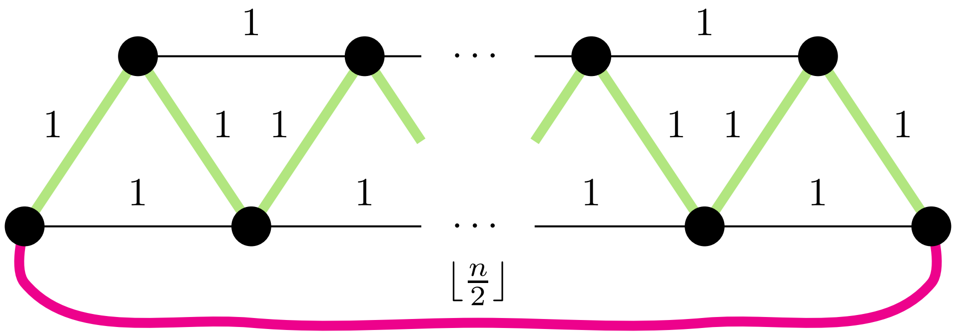
$$\begin{aligned}\mathsf{koszt}(C) &= \mathsf{koszt}(T) + \mathsf{koszt}(M)\\ &\leq \mathsf{koszt}(\mathrm{OPT}) + \frac{\mathsf{koszt}(\mathrm{OPT})}{2}\\ &= \tfrac{3}{2}\cdot \mathsf{koszt}(\mathrm{OPT})\end{aligned}$$



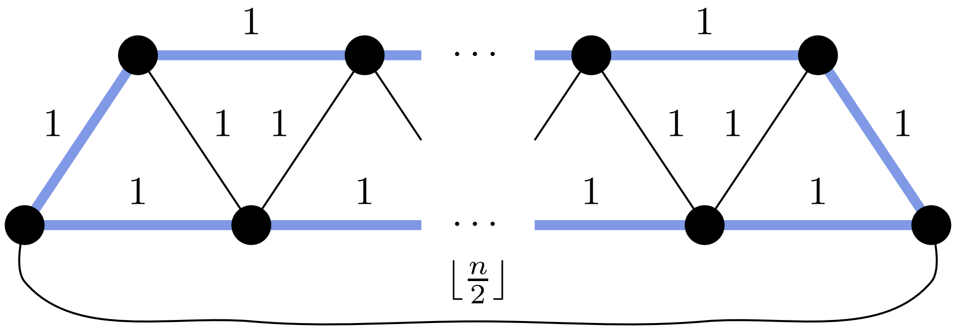
Algorytm ApproxTSP

1. Znajdź minimalne drzewo spinające T grafu G.
2. Znajdź najtańsze doskonałe skojarzenie M w zbiorze wierzchołków nieparzystego stopnia w T. Dodaj M do drzewa T; otrzymany graf jest grafem eulerowskim.
3. Znajdź cykl Eulera C_E w tym grafie.
4. Zwróć trasę C, która odwiedza wszystkie wierzchołki grafu G w kolejności ich wystąpień w cyklu C_E.

- Algorytm ApproxTSP jest algorytmem wielomianowym, gdyż najtańsze skojarzenie można znaleźć w czasie wielomianowym.

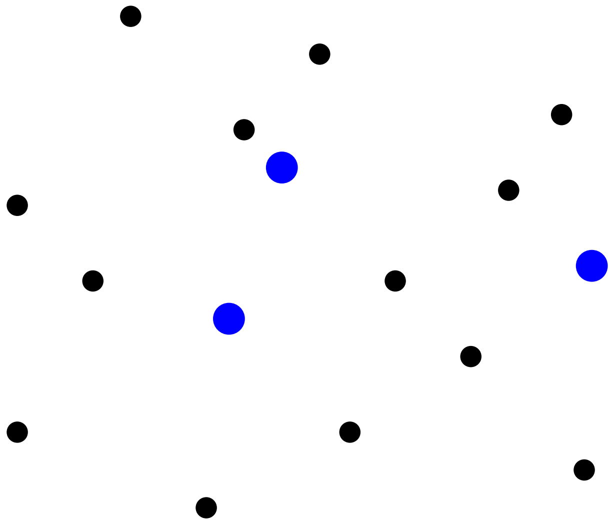


Trudny przypadek. Trudnym przypadkiem dla ApproxTSP2 jest n-wierzchołkowy graf z powyższego rysunku dla n nieparzystego (wszystkie krawędzie za wyjątkiem jednej mają wagę równą 1). Zielonymi krawędziami zaznaczone jest minimalne drzewo spinające, które algorytm znajduje w kroku (1). W drzewie tym są dwa wierzchołki stopnia nieparzystego. Dodając do drzewa krawędź łączącą te wierzchołki (najtańsze skojarzenie), otrzymujemy trasę komiwojażera o koszcie $(n-1) + \lfloor \frac{n}{2} \rfloor$, podczas gdy optymalna trasa ma koszt n.



7.6 PROBLEM LOKALIZACJI SZPITALI /Problem k-centrum na $\mathbb{R}^2$/

Dany jest zbiór $P = \{p_1, \ldots, p_n\}$ miast oraz odległości między każdą parą miast. Należy wybrać k (≥ 1) miast, w których wybudowane zostaną szpitale. Chcemy zminimalizować największą odległość między miastem a najbliższym mu szpitalem.



$k = 3$

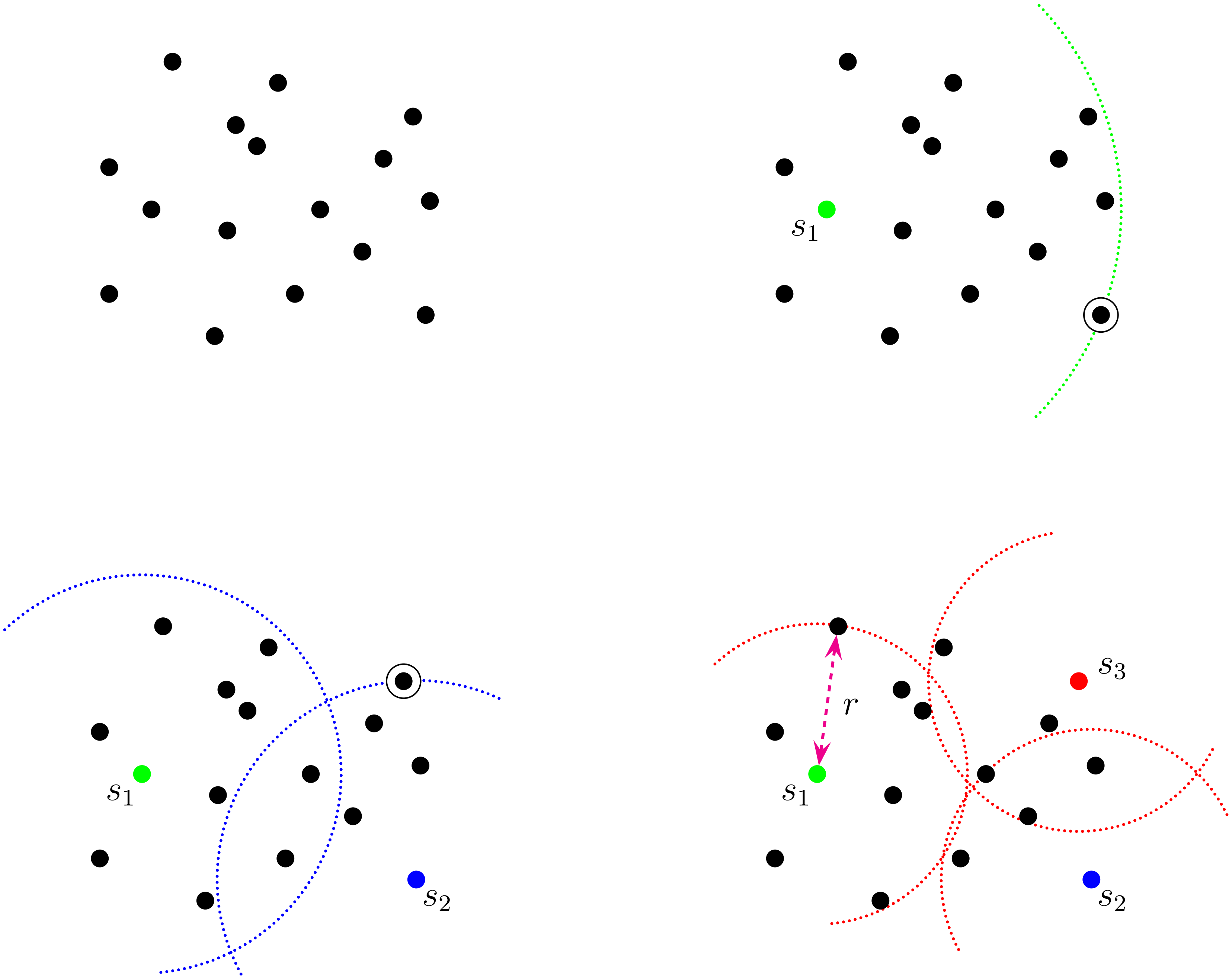
7.6 PROBLEM LOKALIZACJI SZPITALI /Problem k-centrum na $\mathbb{R}^2$/

Dany jest zbiór $P = \{p_1, \ldots, p_n\}$ miast oraz odległości między każdą parą miast. Należy wybrać k (≥ 1) miast, w których wybudowane zostaną szpitale. Chcemy zminimalizować największą odległość między miastem a najbliższym mu szpitalem.

Algorytm FURTHESTFIRST$(P; k)$ /Gonzalez 1985; Hochbaum, Shmoys 1985/

```
1. Wybierz dowolny punkt $s_1 \in P$ jako lokalizację dla szpitala.
   $S := \{s_1\}$.
2. for $j := 2$ to $k$ do
         Niech $s_j \in P$ będzie punktem najdalszym od $s_1, \ldots, s_{j-1}$.
         $S := S \cup \{s_j\}$.
3. return $S$.
```

FURTHESTFIRST można zaimplementować tak, aby działał w czasie $O(nk)$.


s_1
s_1
s_2
r
s_3
s_1
s_2

Twierdzenie 7.7. (Gonzalez 1985; Hochbaum, Shmoys 1985)
FurthestFirst *ma współczynnik aproksymacji równy* 2.

- W ogólnym metrycznym przypadku problem jest trudny do aproksymacji dla dowolnego $2 - \epsilon$, $\epsilon > 0$, o ile P≠NP (m.in. Gonzalez 1985).

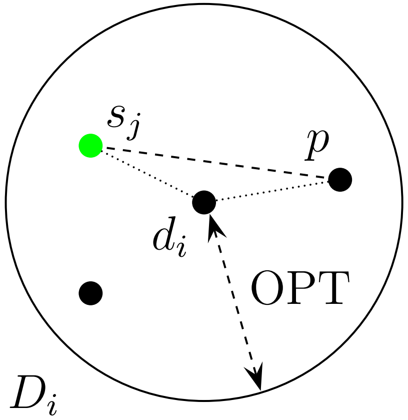
Niech r_j będzie odległością najdalszego punktu z P od punktów $s_1, \ldots, s_j$, $j = 1, \ldots, k$. Z konstrukcji algorytmu mamy, że dla $j = 1, \ldots, k-1$, r_j jest odległością s_{j+1} od punktów $s_1, \ldots, s_j$, a $r = r_k$ jest rozwiązaniem zwracanym przez FurthestFirst, tzn. r jest odległością najdalszego punktu od zbioru $S = \{s_1, \ldots, s_k\}$ do najbliższego szpitala z S.

> Zauważmy, że w każdej iteracji algorytm FurthestFirst dodaje nowe lokalizacje, oraz odległość punktów do najbliższego szpitala nie zwiększa się, a zatem, w szczególności, odległość najdalszego punktu do dotychczas wyznaczonych lokalizacji nie zwiększa się. Tym samym:
>
> $$r_1 \geq r_2 \geq \cdots \geq r_k = r.$$

Lemat. *Zachodzi* $r \leq 2\mathrm{OPT}$, *gdzie* OPT *jest maksymalną odległością punktu* $p \in P$ *do szpitala w optymalnym rozwiązaniu dla* P.

Dowód. Niech d_i, $i = 1, \ldots, k$, będzie optymalnym rozwiązaniem, tj. optymalną lokalizacją szpitali; niech D_i oznacza koło o promieniu OPT i środku w d_i.

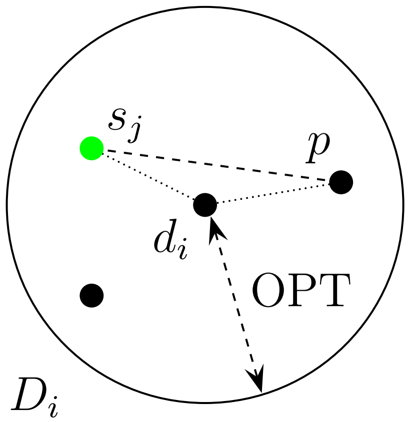
↳ Jeśli każde koło D_i zawiera jakiś punkt s_j (z przybliżonego rozwiązania), wówczas – z nierówności trójkąta i z faktu, że $D_1, \ldots, D_k$ pokrywają wszystkie punkty z P – mamy, że każdy punkt z $p \in P$ znajduje się w odległości co najwyżej 2OPT od jakiegoś punktu z S, a zatem $r \leq 2\mathrm{OPT}$.

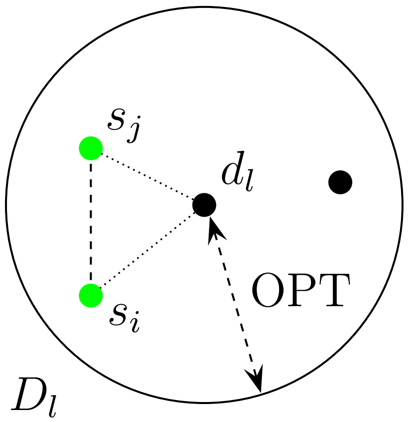


□

Dowód. Niech d_i, $i = 1, \ldots, k$, będzie optymalnym rozwiązaniem, tj. optymalną lokalizacją szpitali; niech D_i oznacza koło o promieniu OPT i środku w d_i.

↳ Jeśli każde koło D_i zawiera jakiś punkt s_j (z przybliżonego rozwiązania), wówczas – z nierówności trójkąta i z faktu, że $D_1, \ldots, D_k$ pokrywają wszystkie punkty z P – mamy, że każdy punkt z $p \in P$ znajduje się w odległości co najwyżej 2OPT od jakiegoś punktu z S, a zatem $r \leq 2\text{OPT}$.

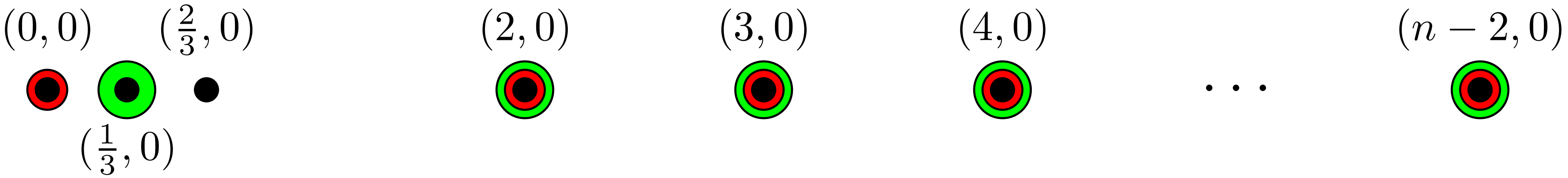




↳ W przeciwnym wypadku, jako że $D_1, \ldots, D_k$ pokrywają wszystkie punkty z P, istnieją s_i, s_j, $i < j$, oraz D_l takie, że $s_i, s_j \in D_l$. Ponieważ $r_j = \text{dist}(s_j, \{s_1, \ldots, s_{j-1}\}) \leq \text{dist}(s_i, s_j)$ oraz $r \leq r_j$, otrzymujemy $r \leq \text{dist}(s_i, s_j)$. Tym samym, z nierówności trójkąta:

$$r \leq r_j \leq \text{dist}(s_i, s_j) \leq \text{dist}(s_i, d_l) + \text{dist}(s_j, d_l) \leq 2\text{OPT}.$$

$\square$



🟢 – rozwiązanie optymalne o koszcie $\frac{1}{3}$

🔴 – możliwe rozwiązanie o koszcie $\frac{2}{3}$ zwrócone przez algorytm

Trudny przypadek. $P = \{(0,0), (\frac{1}{3},0), (\frac{2}{3},0), (2,0), (3,0), \dots, (n-2,0)\}$ i $k = n-2$.

Twierdzenie 7.8. (Gonzalez 1985)
Dla każdego $\varepsilon > 0$ nie istnieje wielomianowy algorytm o współczynniku $\sqrt{3}-\varepsilon$ dla problemu k-centrum na płaszczyźnie, o ile $\mathrm{P} \neq \mathrm{NP}$.

Twierdzenie 7.9. (Gonzalez 1985)
Dla każdego $\varepsilon > 0$ nie istnieje wielomianowy algorytm o współczynniku $2-\varepsilon$ dla problemu k-centrum w przestrzeni trójwymiarowej, o ile $\mathrm{P} \neq \mathrm{NP}$.

7.7* PROBLEM k-CENTRUM W GRAFACH

Niech $G = (V, E, w)$ będzie nieskierowanym ważonym grafem pełnym, w którym wagi krawędzi spełniają nierówność trójkąta. Niech k będzie dodatnią liczbą całkowitą. Dla dowolnego zbioru $S \subseteq V$ i wierzchołka $v \in V$, niech $w(v, S)$ będzie kosztem najtańszej krawędzi łączącej v z wierzchołkiem z S.

Problem.
Znajdź $S \subseteq V$ taki, że $|S| = k$ oraz $\max_{v \in V} w(v, S)$ jest możliwie najmniejsze.

Posortujmy krawędzie G w kolejności niemalejących wag, tzn. $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$, $m = |E|$. Niech $G_i = (V, E_i)$, gdzie $E_i = \{e_1, e_2, \ldots, e_i\}$.

- Problem k-centrum jest równoważny problemowi znalezienia najmniejszego i takiego, że graf G_i ma zbiór dominujący rozmiaru co najwyżej k. *Zbiór dominujący* w grafie G to taki podzbiór $S \subseteq V$, że każdy wierzchołek w $V \setminus S$ ma sąsiada w S.
- Niech i^* będzie najmniejszym takim indeksem. Wówczas $\mathrm{OPT} = w(e_{i^*})$ jest kosztem optymalnego k-centrum.

Kwadratem grafu G nazywamy graf o tym samym zbiorze wierzchołków i zawierający krawędź $\{v_1, v_2\}$ wtedy i tylko wtedy, gdy odległość – mierzona jako liczba krawędzi na najkrótszej ścieżce – między wierzchołkami u i v jest nie większa niż 2 i $v_1 \neq v_2$; kwadrat grafu G oznaczany jest przez G^2.

Algorytm ApproxCentrum$(G = (V, E, w); k)$

1. Skonstruuj grafy $G_1^2, G_2^2, \ldots, G_m^2$.
2. W każdym z grafów G_i^2 znajdź maksymalny zbiór niezależny I_i.
3. Niech j będzie najmniejszym indeksem takim, że $|I_j| \leq k$.
4. **return** I_j.

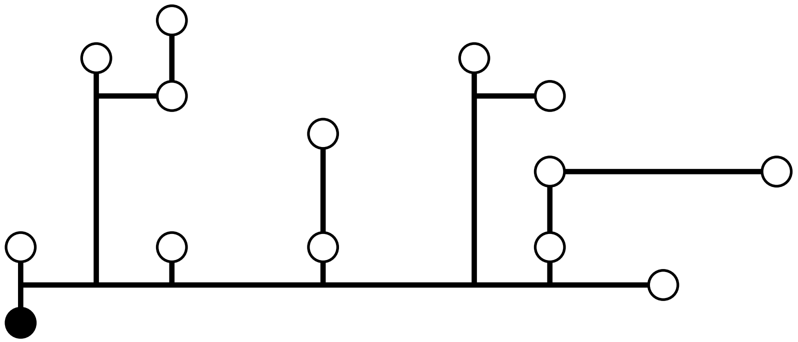
Twierdzenie 7.10. (Hochbaum, Shmoys 1985) *Algorytm* ApproxCentrum *jest algorytmem 2-aproksymacyjnym dla metrycznego problemu k-centrum.*

Złożoność czasowa.

- Konstrukcja grafów G_i^2 zajmuje czas wielomianowy. Zauważmy, że w kroku 2 interesuje nas dowolny maksymalny zbiór niezależny, który może być wyznaczony w sposób zachłanny (i wielomianowy). Kroki 3-4: czas wielomianowy.

7.8 PROBLEM RSA (NAJKRÓTSZA PROSTOKĄTNA GAŁĄŹ STEINERA)

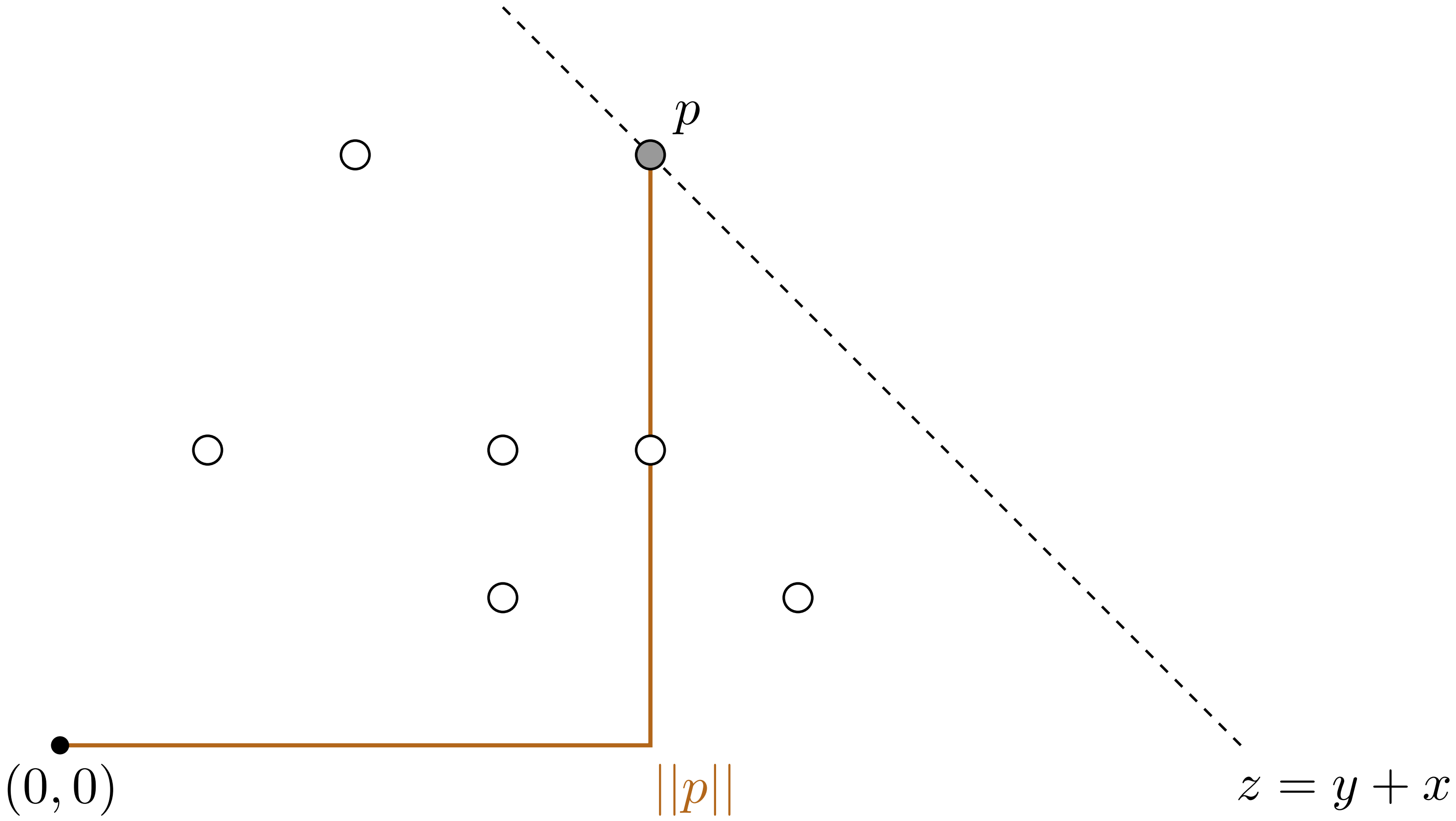
S. K. Rao, P. Sadayappan, F. K. Hwang, P. W. Shor
The rectilinear steiner arborescence problem
Algorithmica 7(1-6), 277-288 (1992)



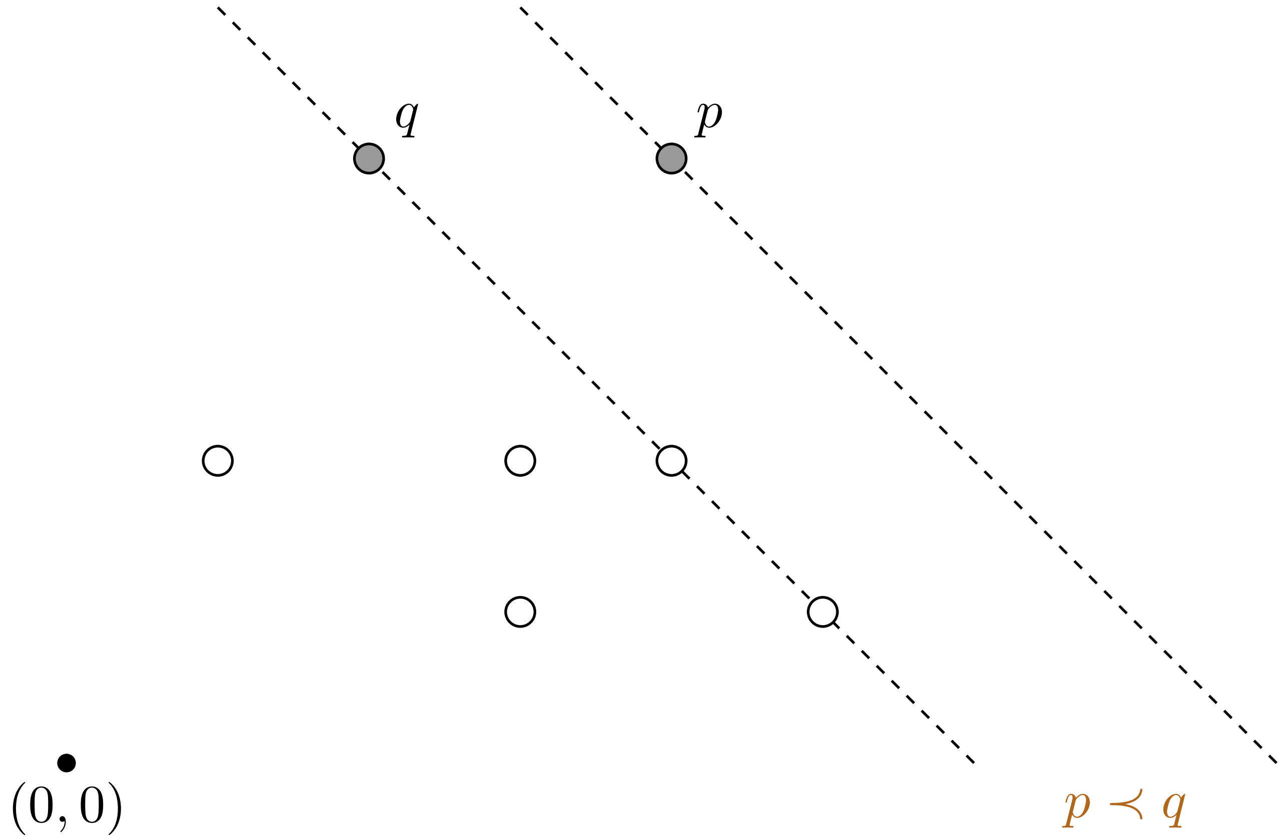
Najkrótsza prostokątna gałąź Steinera

Problem. (Rectilinear Steiner Arborescence) *Dla zbioru* $P \cup \{r\}$ *punktów z pierwszej ćwiartki płaszczyzny* $\mathbb{R}^2$ *takich, że* r *dominuje nad wszystkimi punktami ze zbioru* P*, wyznaczyć najkrótszą prostokątną gałąź Steinera dla* P *o korzeniu* r.

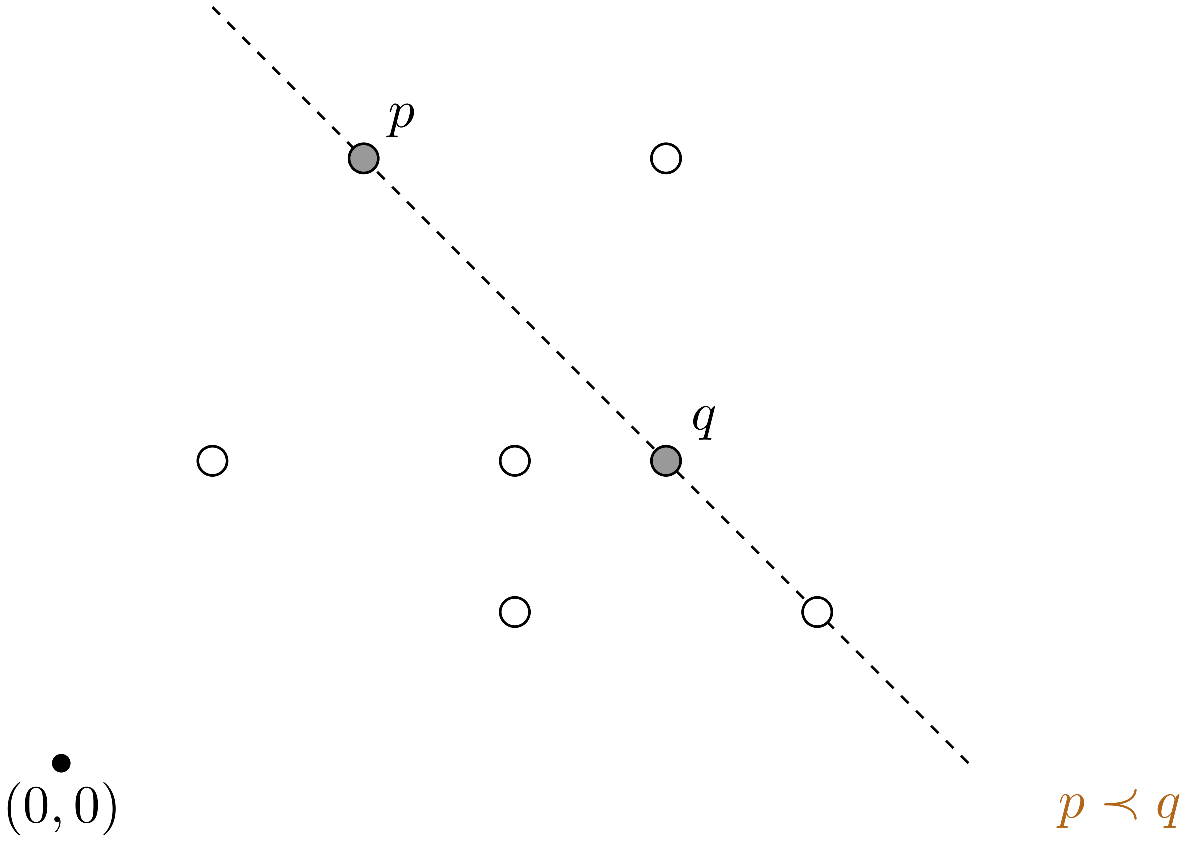
- *Koszt* prostokątnej gałęzi Steinera to suma długości jej krawędzi.
- Problem RSA jest problemem NP-trudnym (Shi, Su 2006).



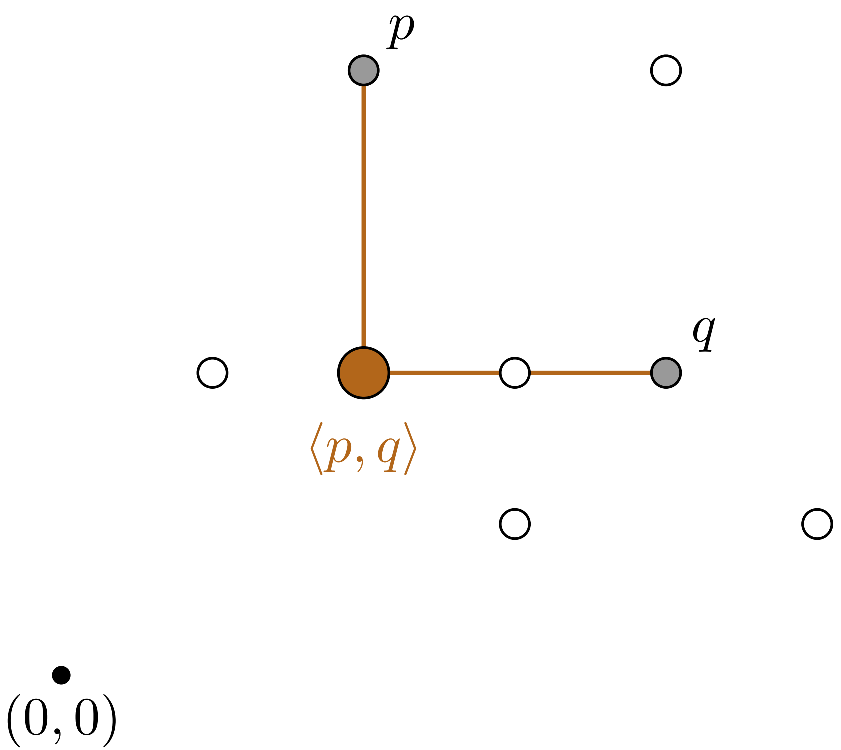
- $||p|| \doteq \mathrm{x}(p) + \mathrm{y}(p)$
- Porządek: $p \prec q$, jeśli $||q|| < ||p||$ lub $||q|| = ||p||$ oraz $\mathrm{x}(p) < \mathrm{x}(q)$
- Scalanie: $\langle p, q\rangle \doteq (\min\{\mathrm{x}(p), \mathrm{x}(q)\}, \min\{\mathrm{y}(p), \mathrm{y}(q)\})$
- $koszt(\langle p, q\rangle) \doteq |\mathrm{x}(p) - \mathrm{x}(q)| + |\mathrm{y}(p) - \mathrm{y}(q)|$



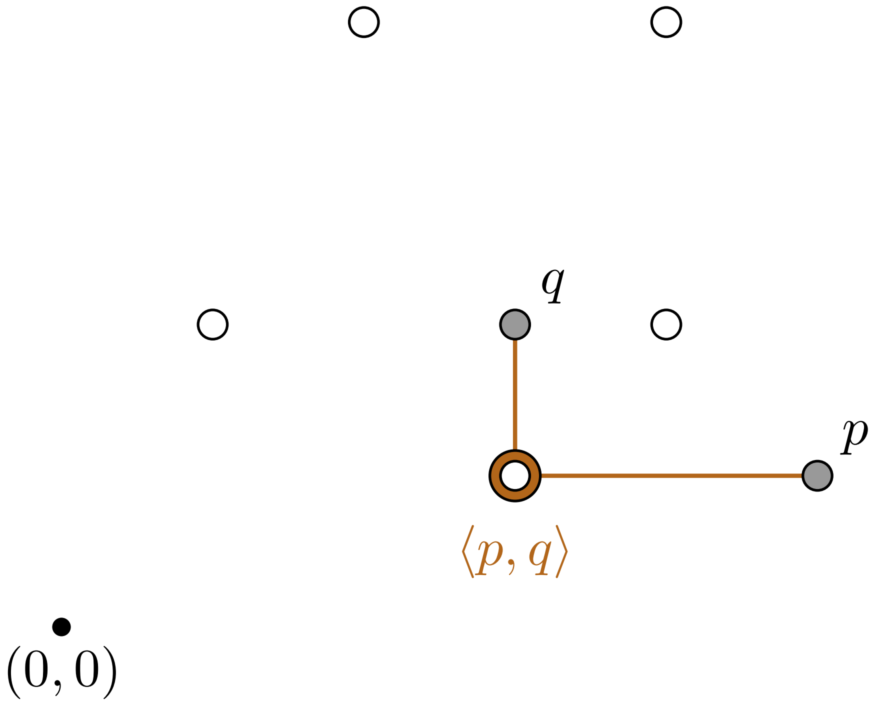
- $||p|| \doteq \mathrm{x}(p) + \mathrm{y}(p)$
- Porządek: $p \prec q$, jeśli $||q|| < ||p||$ lub $||q|| = ||p||$ oraz $\mathrm{x}(p) < \mathrm{x}(q)$
- Scalanie: $\langle p, q \rangle \doteq (\min\{\mathrm{x}(p), \mathrm{x}(q)\}, \min\{\mathrm{y}(p), \mathrm{y}(q)\})$
- $koszt(\langle p, q \rangle) \doteq |\mathrm{x}(p) - \mathrm{x}(q)| + |\mathrm{y}(p) - \mathrm{y}(q)|$



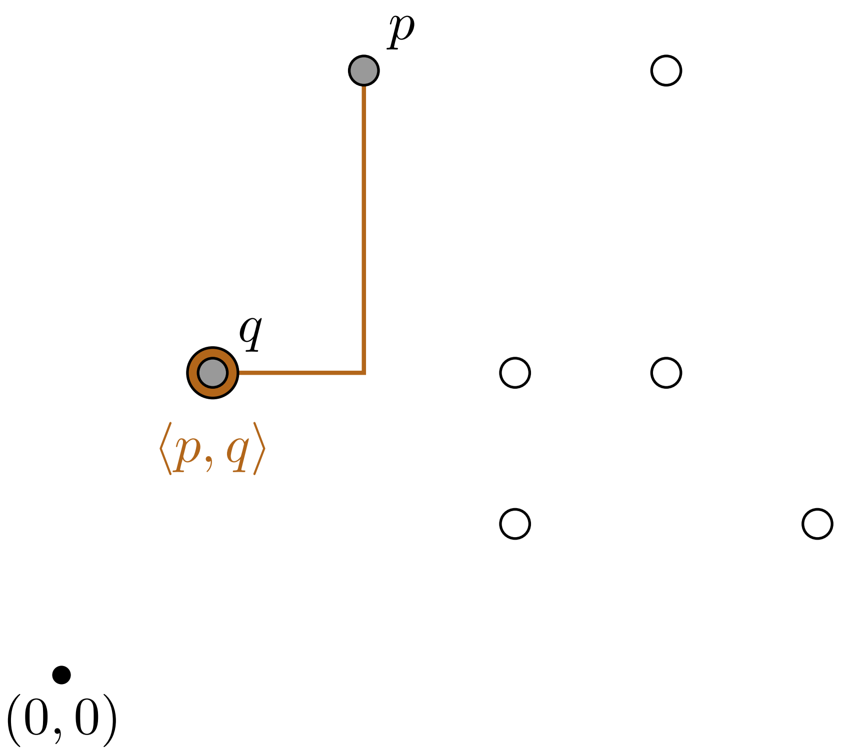
- $||p|| \doteq \mathrm{x}(p) + \mathrm{y}(p)$
- Porządek: $p \prec q$, jeśli $||q|| < ||p||$ lub $||q|| = ||p||$ oraz $\mathrm{x}(p) < \mathrm{x}(q)$
- Scalanie: $\langle p, q\rangle \doteq (\min\{\mathrm{x}(p), \mathrm{x}(q)\}, \min\{\mathrm{y}(p), \mathrm{y}(q)\})$
- $koszt(\langle p, q\rangle) \doteq |\mathrm{x}(p) - \mathrm{x}(q)| + |\mathrm{y}(p) - \mathrm{y}(q)|$



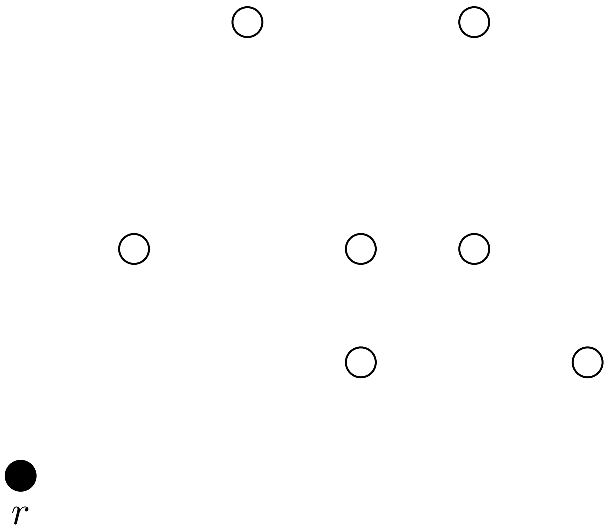
- $||p|| \doteq \mathrm{x}(p) + \mathrm{y}(p)$
- Porządek: $p \prec q$, jeśli $||q|| < ||p||$ lub $||q|| = ||p||$ oraz $\mathrm{x}(p) < \mathrm{x}(q)$
- Scalanie: $\langle p, q\rangle \doteq (\min\{\mathrm{x}(p), \mathrm{x}(q)\}, \min\{\mathrm{y}(p), \mathrm{y}(q)\})$
- $koszt(\langle p, q\rangle) \doteq |\mathrm{x}(p) - \mathrm{x}(q)| + |\mathrm{y}(p) - \mathrm{y}(q)|$



- $||p|| \doteq \mathrm{x}(p) + \mathrm{y}(p)$
- Porządek: $p \prec q$, jeśli $||q|| < ||p||$ lub $||q|| = ||p||$ oraz $\mathrm{x}(p) < \mathrm{x}(q)$
- Scalanie: $\langle p, q\rangle \doteq (\min\{\mathrm{x}(p), \mathrm{x}(q)\}, \min\{\mathrm{y}(p), \mathrm{y}(q)\})$
- $koszt(\langle p, q\rangle) \doteq |\mathrm{x}(p) - \mathrm{x}(q)| + |\mathrm{y}(p) - \mathrm{y}(q)|$



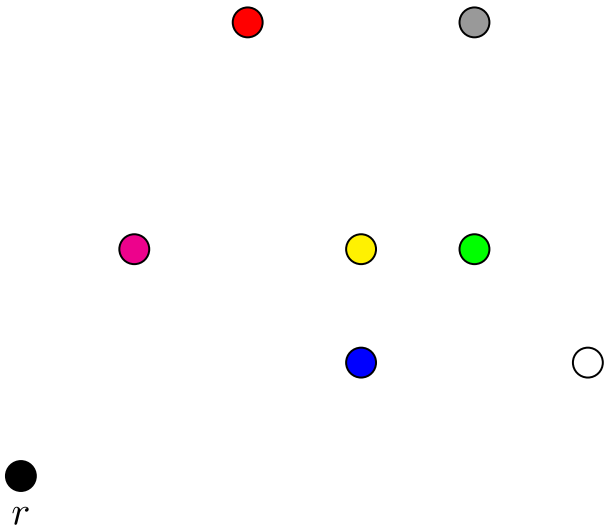
- ▶ $||p|| \doteq \mathrm{x}(p) + \mathrm{y}(p)$
- ▶ Porządek: $p \prec q$, jeśli $||q|| < ||p||$ lub $||q|| = ||p||$ oraz $\mathrm{x}(p) < \mathrm{x}(q)$
- ▶ Scalanie: $\langle p, q\rangle \doteq (\min\{\mathrm{x}(p), \mathrm{x}(q)\}, \min\{\mathrm{y}(p), \mathrm{y}(q)\})$
- ▶ $koszt(\langle p, q\rangle) \doteq |\mathrm{x}(p) - \mathrm{x}(q)| + |\mathrm{y}(p) - \mathrm{y}(q)|$



Algorytm ApproxRSA

```
1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z $Q$, że $\langle p, q \rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q' \rangle : p', q' \in Q\}$.
   2.2 Zastąp $p$ i $q$ w $Q$ przez $\langle p, q \rangle$, a następnie dodaj do $G$ wierzchołek $\langle p, q \rangle$ wraz odpowiednimi krawędziami z $p$ do $\langle p, q \rangle$ oraz z $q$ do $\langle p, q \rangle$.
3. Zwróć gałąź $G$.
```

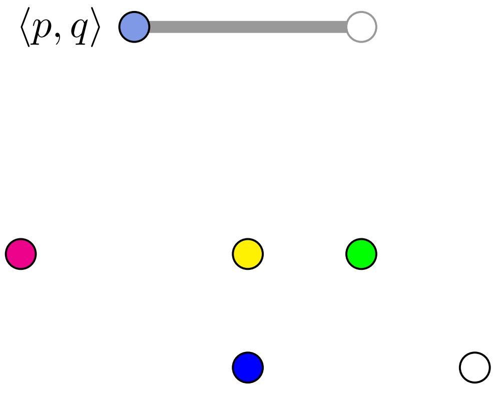
Q: ● ● ● ○ ● ● ● ●



Algorytm ApproxRSA

```
1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z $Q$, że $\langle p, q \rangle$ jest
       elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q' \rangle : p', q' \in Q\}$.
   2.2 Zastąp $p$ i $q$ w $Q$ przez $\langle p, q \rangle$, a następnie dodaj do $G$ wierzchołek $\langle p, q \rangle$
       wraz odpowiednimi krawędziami z $p$ do $\langle p, q \rangle$ oraz z $q$ do $\langle p, q \rangle$.
3. Zwróć gałąź $G$.
```

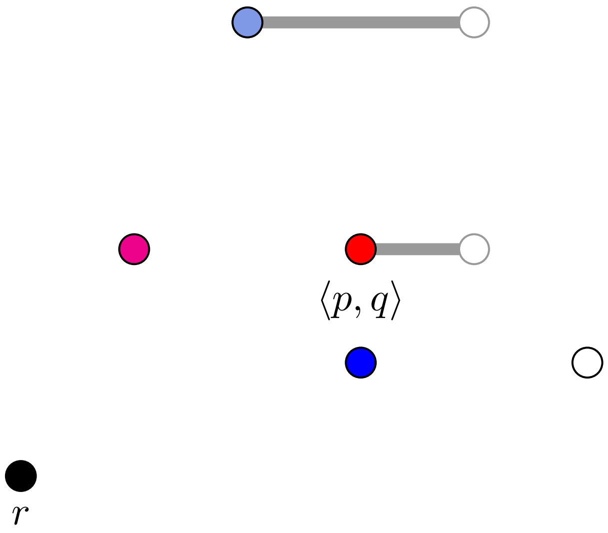
Q: ● ● ○ ● ● ● ●



Algorytm ApproxRSA

```
1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z $Q$, że $\langle p, q\rangle$ jest
       elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q'\rangle : p', q' \in Q\}$.
   2.2 Zastąp $p$ i $q$ w $Q$ przez $\langle p, q\rangle$, a następnie dodaj do $G$ wierzchołek $\langle p, q\rangle$
       wraz odpowiednimi krawędziami z $p$ do $\langle p, q\rangle$ oraz z $q$ do $\langle p, q\rangle$.
3. Zwróć gałąź $G$.
```

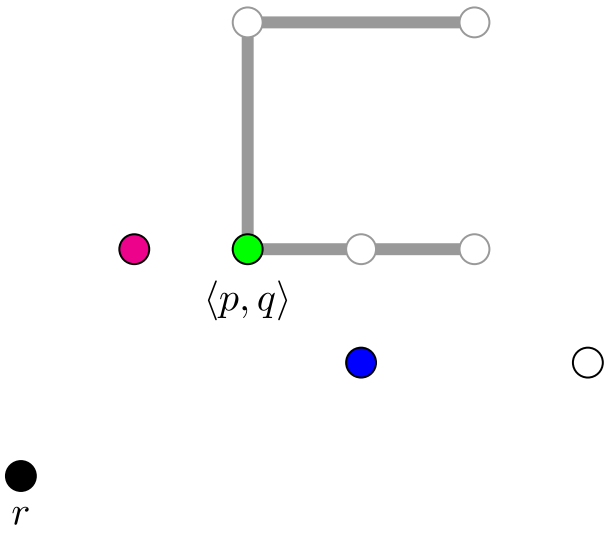
Q: ● ○ ● ● ● ●



Algorytm ApproxRSA

```
1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z $Q$, że $\langle p, q\rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q'\rangle : p', q' \in Q\}$.
   2.2 Zastąp $p$ i $q$ w $Q$ przez $\langle p, q\rangle$, a następnie dodaj do $G$ wierzchołek $\langle p, q\rangle$ wraz odpowiednimi krawędziami z $p$ do $\langle p, q\rangle$ oraz z $q$ do $\langle p, q\rangle$.
3. Zwróć gałąź $G$.
```

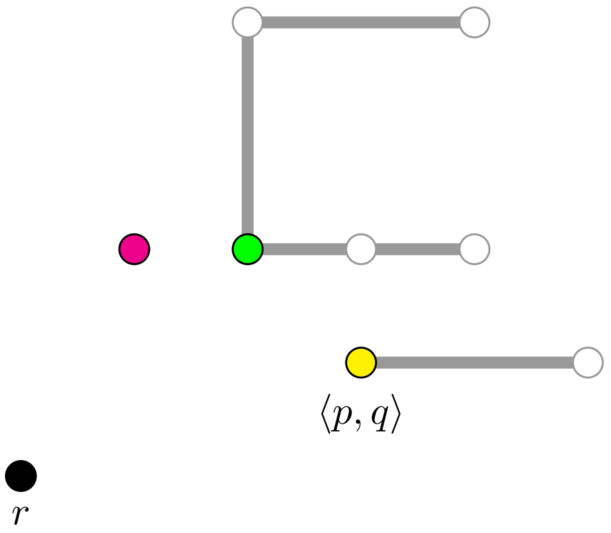
Q: ○ ● ● ● ●



Algorytm ApproxRSA

1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   - 2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z Q, że $\langle p, q\rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q'\rangle : p', q' \in Q\}$.
   - 2.2 Zastąp p i q w Q przez $\langle p, q\rangle$, a następnie dodaj do G wierzchołek $\langle p, q\rangle$ wraz odpowiednimi krawędziami z p do $\langle p, q\rangle$ oraz z q do $\langle p, q\rangle$.
3. Zwróć gałąź G.

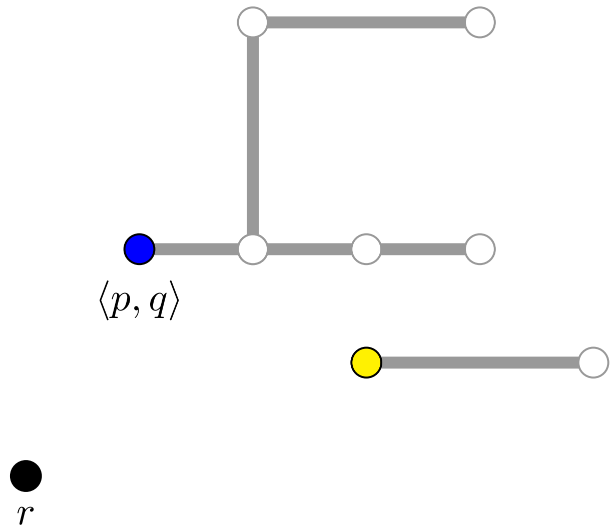
Q: ● ● ● ●



Algorytm ApproxRSA

```
1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z $Q$, że $\langle p, q\rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q'\rangle : p', q' \in Q\}$.
   2.2 Zastąp $p$ i $q$ w $Q$ przez $\langle p, q\rangle$, a następnie dodaj do $G$ wierzchołek $\langle p, q\rangle$ wraz odpowiednimi krawędziami z $p$ do $\langle p, q\rangle$ oraz z $q$ do $\langle p, q\rangle$.
3. Zwróć gałąź $G$.
```

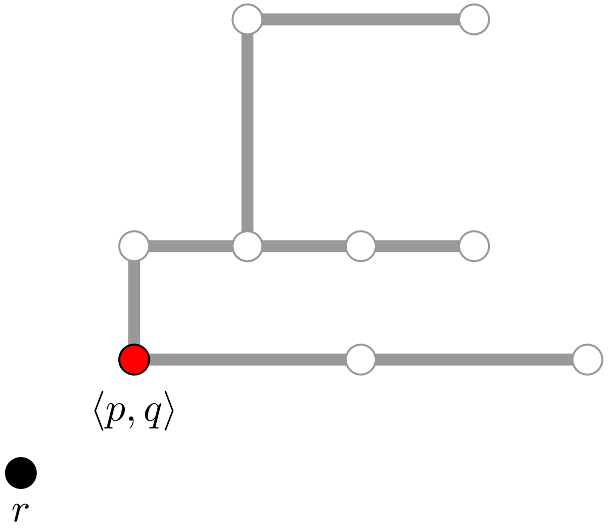
Q: 🟡 🔵 ⚫



Algorytm ApproxRSA

1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   - 2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z Q, że $\langle p, q\rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q'\rangle : p', q' \in Q\}$.
   - 2.2 Zastąp p i q w Q przez $\langle p, q\rangle$, a następnie dodaj do G wierzchołek $\langle p, q\rangle$ wraz odpowiednimi krawędziami z p do $\langle p, q\rangle$ oraz z q do $\langle p, q\rangle$.
3. Zwróć gałąź G.

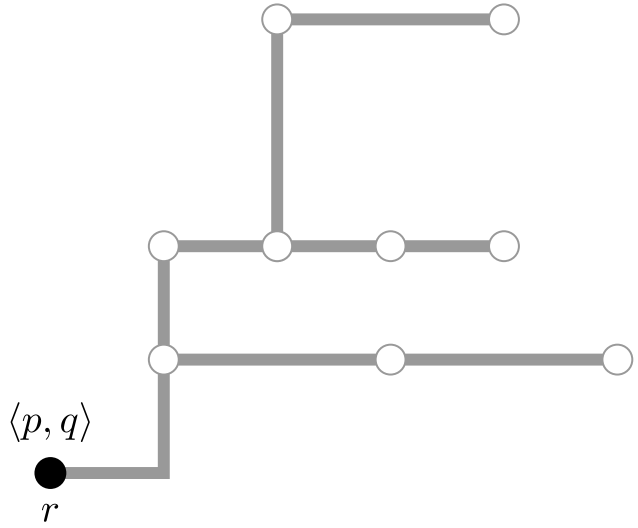
Q: ● ●



Algorytm ApproxRSA

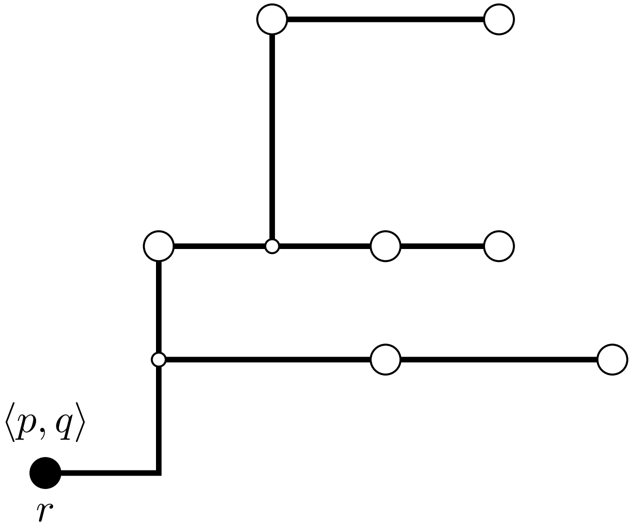
```
1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z $Q$, że $\langle p, q\rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q'\rangle : p', q' \in Q\}$.
   2.2 Zastąp $p$ i $q$ w $Q$ przez $\langle p, q\rangle$, a następnie dodaj do $G$ wierzchołek $\langle p, q\rangle$ wraz odpowiednimi krawędziami z $p$ do $\langle p, q\rangle$ oraz z $q$ do $\langle p, q\rangle$.
3. Zwróć gałąź $G$.
```

Q: ●



Algorytm ApproxRSA

1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   - 2.1 Niech $(p, q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z Q, że $\langle p, q\rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p', q'\rangle \colon p', q' \in Q\}$.
   - 2.2 Zastąp p i q w Q przez $\langle p, q\rangle$, a następnie dodaj do G wierzchołek $\langle p, q\rangle$ wraz odpowiednimi krawędziami z p do $\langle p, q\rangle$ oraz z q do $\langle p, q\rangle$.
3. Zwróć gałąź G.



Algorytm ApproxRSA

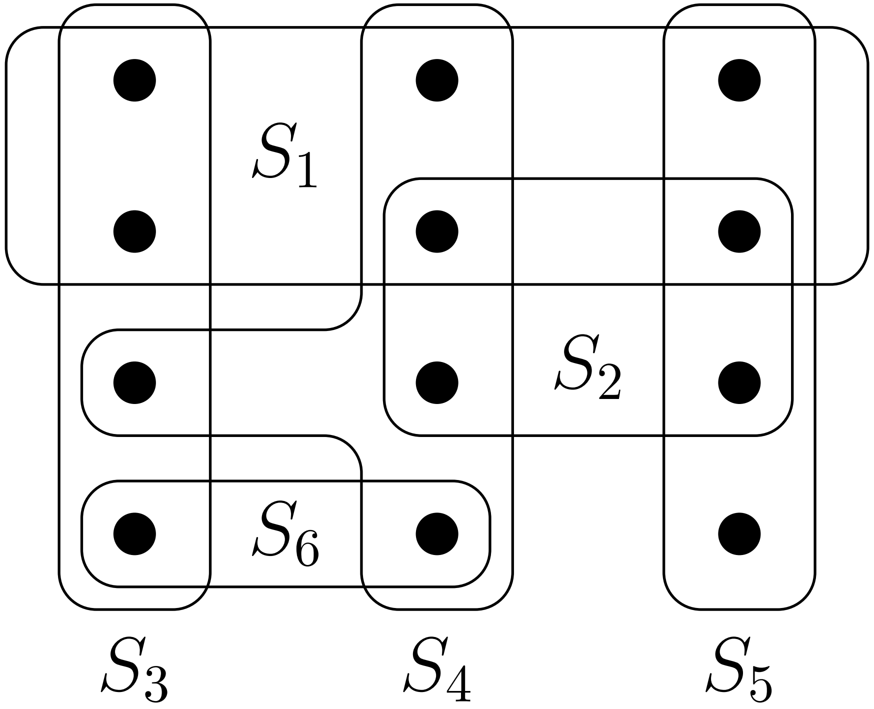
1. $Q := P \cup \{r\}$ oraz $G := (Q, \emptyset)$.
2. Dopóki $|Q| > 1$:
   2.1 Niech $(p,q) \in Q \times Q$ $(p \prec q)$ będzie taką parą punktów z Q, że $\langle p,q\rangle$ jest elementem najmniejszym (w porządku $\prec$) w zbiorze $\{\langle p',q'\rangle : p',q' \in Q\}$.
   2.2 Zastąp p i q w Q przez $\langle p,q\rangle$, a następnie dodaj do G wierzchołek $\langle p,q\rangle$ wraz odpowiednimi krawędziami z p do $\langle p,q\rangle$ oraz z q do $\langle p,q\rangle$.
3. Zwróć gałąź G.

Twierdzenie 7.11. (Rao, Sadayappan, Hwang, Shor 1992)
Współczynnik aproksymacji ApproxRSA *jest równy* 2.

- ▶ Algorytm ApproxRSA jest algorytmem wielomianowym; można go zaimplementować tak, aby działał w czasie liniowo-lagarytmicznym (zamiatanie).

7.9 PROBLEM POKRYCIA ZBIORU

Dla danego n-elementowego uniwersum U, rodziny jego podzbiorów $\mathcal{S}$ oraz funkcji kosztu $c\colon \mathcal{S} \to \mathcal{R}^+$ należy znaleźć najtańszą podrodzinę $\mathcal{X} \subseteq \mathcal{S}$ pokrywającą U.



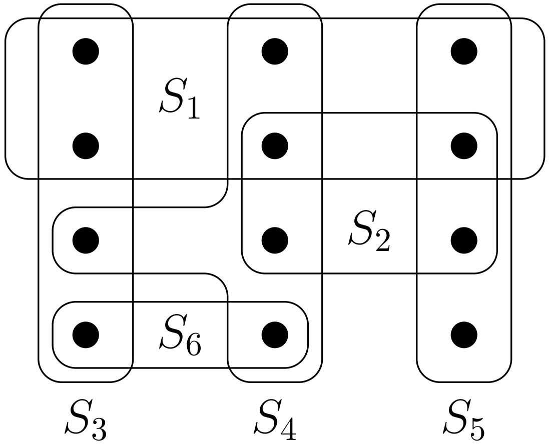
Przykład problemu pokrycia zbioru, gdzie uniwersum U składa się z 12 elementów, a rodzina $\mathcal{S} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$; funkcja kosztu jest stała i równa 1 dla każdego ze zbiorów. Minimalne pokrycie zbioru to $\mathcal{X} = \{S_3, S_4, S_5\}$.

Stała funkcja kosztu

Algorytm ApproxSetCover

```
1. $\mathcal{X} := \emptyset$, $T := U$.
2. while ($T$ jest niepusty) do
        Znajdź zbiór $S$, który pokrywa największą liczbę elementów w $T$.
        $\mathcal{X} := \mathcal{X} \cup \{S\}$.
        $T := T \setminus S$.
3. return $\mathcal{X}$.
```

→ Wielomianowość algorytmu wynika z konstrukcji.



ApproxSetCover znajduje pokrycie rozmiaru 4, wybierając kolejno S_1, S_4, S_5 i S_3.

Twierdzenie 7.12. (Johnson 1974; Lovasz 1975; Stein 1974)
Współczynnik aproksymacji ApproxSetCover *wynosi* $O(\ln n)$.

Dowód. Oznaczmy przez T_i zbiór niepokrytych elementów w i-tej iteracji; dla $i = 1$ mamy $T_1 = U$. Zauważmy, że pokrycie optymalne $\mathcal{X}_{\mathrm{OPT}}$ o koszcie OPT zawsze pokrywa T_i używając nie więcej niż $\mathrm{OPT} = k$ zbiorów. W $(i+1)$-szym kroku ApproxSetCover zawsze wybiera zbiór S_{i+1} zawierający największą liczbę elementów ze zbioru T_i. Rozmiar $|S_{i+1}|$ tego zbioru wynosi przynajmniej $|T_i|/k$ – w przeciwnym wypadku nie ma możliwości pokrycia $T_i \subseteq U$ rodziną o k zbiorach, a tym samym pokrycia U, co prowadzi do sprzeczności z byciem optymalnym pokryciem przez $\mathcal{X}_{\mathrm{OPT}}$. A zatem otrzymujemy, że

$$|T_{i+1}| \leq |T_i| - |T_i|/k = (1 - \frac{1}{k}) \cdot |T_i|,$$

a w konsekwencji, na mocy indukcji, zachodzi

$$|T_i| \leq (1 - \frac{1}{k})^i \cdot n.$$

Algorytm zatrzymuje się, jeśli $|T_i| = 0 < 1$. ...

Twierdzenie 7.12. (Johnson 1974; Lovasz 1975; Stein 1974)
Współczynnik aproksymacji APPROXSETCOVER *wynosi* $O(\ln n)$.

Dowód. ...

Zauważmy, że dla $M = k \ln n + 1$ mamy

$$|T_M| \leq (1 - \frac{1}{k})^M \cdot n \leq e^{-\frac{1}{k} \cdot M} \cdot n = e^{-\frac{k \ln n + 1}{k}} \cdot n < e^{-\frac{k \ln n}{k}} \cdot n = 1,$$

jako że $1 - x \leq e^{-x}$ dla $x \geq 0$, a tym samym, dla $M = k \ln n + 1$ zachodzi $|T_M| = 0$. W konsekwencji algorytm zawsze zatrzymuje się po wykonaniu co najwyżej $k \ln n + 1$ iteracji, zwracając tym samym pokrycie rozmiaru co najwyżej $k \ln n + 1 = |\mathrm{OPT}| \cdot O(\ln n)$. ☐

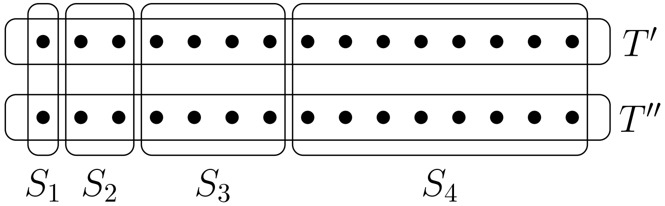
Twierdzenie 7.12. (Johnson 1974; Lovasz 1975; Stein 1974)
Współczynnik aproksymacji ApproxSetCover *wynosi* $O(\ln n)$.

Dowód. ...

Zauważmy, że dla $M = k \ln n + 1$ mamy

$$|T_M| \leq (1 - \frac{1}{k})^M \cdot n \leq e^{-\frac{1}{k} \cdot M} \cdot n = e^{-\frac{k \ln n + 1}{k}} \cdot n < e^{-\frac{k \ln n}{k}} \cdot n = 1,$$

jako że $1 - x \leq e^{-x}$ dla $x \geq 0$, a tym samym, dla $M = k \ln n + 1$ zachodzi $|T_M| = 0$. W konsekwencji algorytm zawsze zatrzymuje się po wykonaniu co najwyżej $k \ln n + 1$ iteracji, zwracając tym samym pokrycie rozmiaru co najwyżej $k \ln n + 1 = |\mathrm{OPT}| \cdot O(\ln n)$. □



Trudny przypadek. Universum składa się z $2(2^k - 1) = 2^{k-1} - 2$ elementów, a rodzina podzbiorów $\mathcal{S}$ składa się z rozłącznych zbiorów $S_1, \ldots, S_i, \ldots, S_k$ o odpowiednio 2^i elementach oraz dwóch zbiorów T' i T'', z których każdy pokrywa połowę elementów z każdego zbiorów S_i. Algorytm zachłanny wybierze kolejno zbiory $S_k, S_{k-1}, \ldots, S_1$, podczas gdy optymalne pokrycie składa się z dwóch zbiorów T' i T''. Współczynnik aproksymacji wynosi $k/2 = \Theta(\log_2 n)$.

Dowolna funkcja kosztu

Algorytm **ApproxSetCover2** /Chvátal 1979/

```
1. $\mathcal{X} := \emptyset$, $C := \emptyset$.
2. while $C \neq U$ do
       Znajdź najmniej kosztowny zbiór $S$, tzn. zbiór o najmniejszej wartości $\frac{c(S)}{|S \setminus C|}$.
       $\mathcal{X} := \mathcal{X} \cup \{S\}$.
       $C := C \cup S$.
3. return $\mathcal{X}$.
```

Algorytm można zaimplementować tak, aby działał w czasie wielomianowym.

Dowolna funkcja kosztu

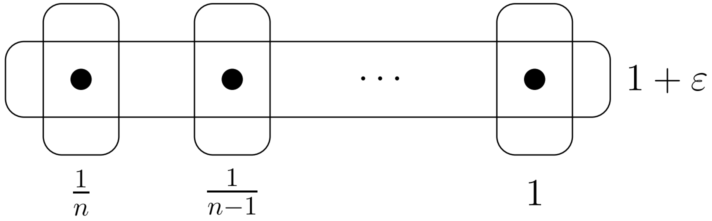
Algorytm ApproxSetCover2 /Chvátal 1979/

```
1. $\mathcal{X} := \emptyset$, $C := \emptyset$.
2. while $C \neq U$ do
       Znajdź najmniej kosztowny zbiór $S$, tzn. zbiór o najmniejszej wartości $\frac{c(S)}{|S \setminus C|}$.
       $\mathcal{X} := \mathcal{X} \cup \{S\}$.
       $C := C \cup S$.
3. return $\mathcal{X}$.
```

Algorytm można zaimplementować tak, aby działał w czasie wielomianowym.

Twierdzenie 7.13. (Chvátal 1979) *Współczynnik aproksymacji* ApproxSetCover2 *jest równy* H_n*, gdzie* $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$*, a zatem jest rzędu* $O(\log n)$*.*

Trudny przypadek.



Algorytm zachłanny wykonany na tej instancji wypisze pokrycie składające się z n singletonów, ponieważ w każdej interacji jeden z singletonów jest najmniej kosztowny. Łączny koszt tego pokrycia to $1 + \frac{1}{2} + \cdots + \frac{1}{n} = H_n$. Koszt optymalnego pokrycia wynosi $1 + \varepsilon$.

8. PROGRAMOWANIE DYNAMICZNE

Etapy programowania dynamicznego:

1. Scharakteryzowanie struktury optymalnego rozwiązania.
2. Rekurencyjne zdefiniowanie kosztu optymalnego rozwiązania.
3. Obliczenie kosztu optymalnego rozwiązania metodą wstępującą.
4.* Skonstruowanie optymalnego rozwiązania na podstawie wyników wcześniejszych obliczeń.

Aby programowanie dynamiczne było skuteczne:

- Liczba wszystkich podproblemów, które trzeba rozwiązać, musi być niewielka.
- Musi być możliwe określenie takiej kolejności wyliczania przykładów, aby w każdym momencie znane były rozwiązania wszystkich mniejszych podproblemów potrzebnych do rozwiązania podproblemu aktualnie rozważanego.

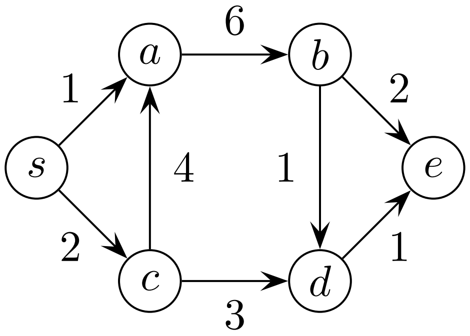
Przykłady:

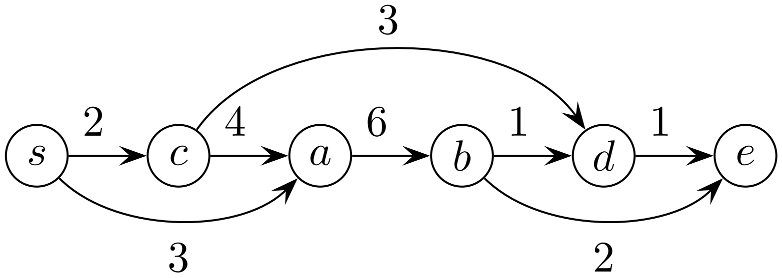
- najdłuższy wspólny podciąg;
- mnożenie ciągu macierzy;
- optymalne drzewo poszukiwań binarnych.

8.1 NAJKRÓTSZA ŚCIEŻKA W GRAFACH DAG

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.1, PWN (2012)

Problem. *Dla danego acyklicznego ważonego grafu skierowanego* $G = (V, A, w)$*, ozn. DAG, oraz wierzchołków* $v_1, v_2 \in V$ *wyznacz najkrótszą ścieżkę łączącą* v_1 *i* v_2*.*



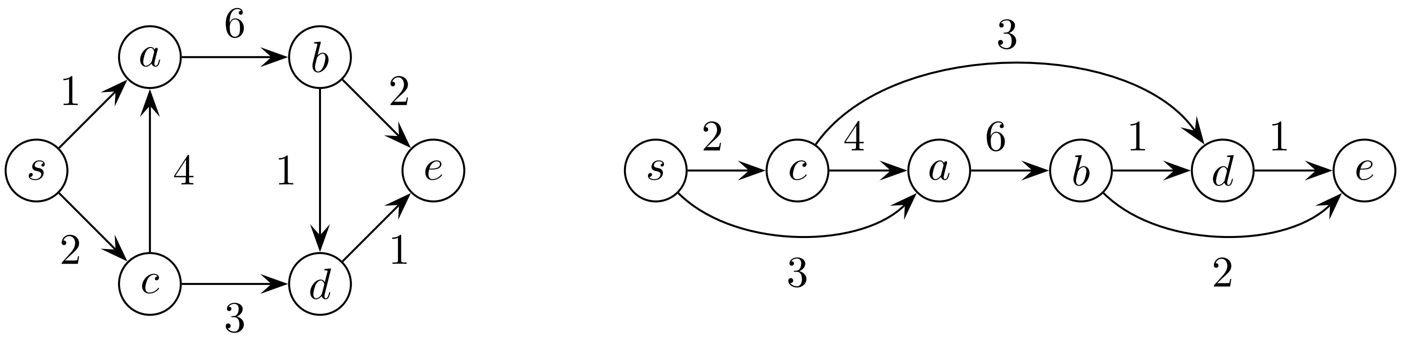


$$\mathrm{dist}(s, d) = \min\{\mathrm{dist}(s, b) + 1, \mathrm{dist}(s, c) + 3\}$$

8.1 NAJKRÓTSZA ŚCIEŻKA W GRAFACH DAG

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.1, PWN (2012)

Problem. *Dla danego acyklicznego ważonego grafu skierowanego* $G = (V, A, w)$, *ozn. DAG, oraz wierzchołków* $v_1, v_2 \in V$ *wyznacz najkrótszą ścieżkę łączącą* v_1 *i* v_2*.*



$$\mathrm{dist}(s, d) = \min\{\mathrm{dist}(s, b) + 1, \mathrm{dist}(s, c) + 3\}$$

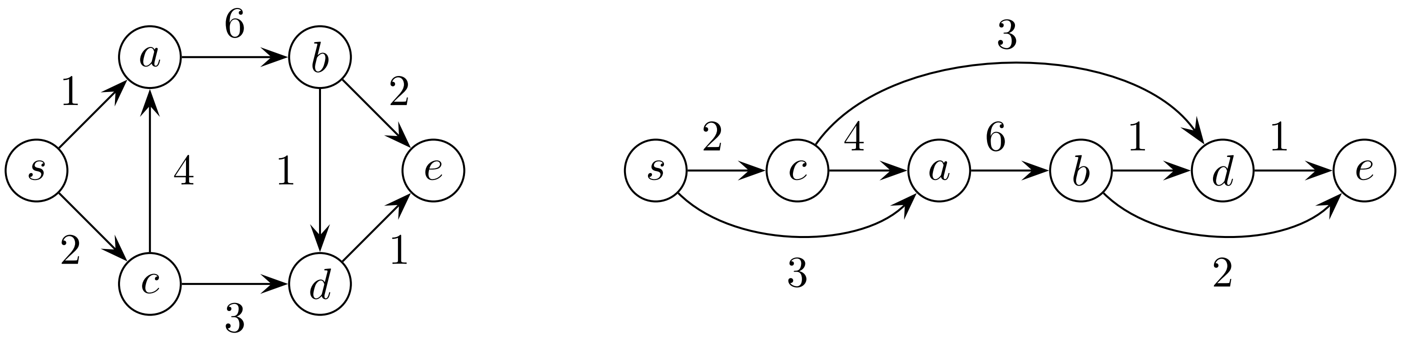
Algorytm

```
1. for each $v \in V$ do $\mathsf{dist}(v) := \infty$.
2. $\mathsf{dist}(s, s) := 0$.
3. for each $v \in V \setminus \{s\}$ in linearized order, do
       $\mathsf{dist}(s, v) := \min_{(u,v) \in A}\{\mathsf{dist}(s, u) + w((u, v))\}$.
```

8.1 Najkrótsza ścieżka w grafach DAG

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.1, PWN (2012)

Problem. *Dla danego acyklicznego ważonego grafu skierowanego $G = (V, A, w)$, ozn. DAG, oraz wierzchołków $v_1, v_2 \in V$ wyznacz najkrótszą ścieżkę łączącą v_1 i v_2.*



$$\mathrm{dist}(s, d) = \min\{\mathrm{dist}(s, b) + 1, \mathrm{dist}(s, c) + 3\}$$

Uwaga. Algorytmu opiera się na rozwiązywaniu podproblemów ze zbioru

$$\{\mathsf{dist}(s, v) : v \in V\},$$

w kolejności od „lewej do prawej”, zaczynając od „najmniejszego”, tj. od $\mathsf{dist}(s, s)$.

8.2 NAJDŁUŻSZY PODCIĄG ROSNĄCY

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.2, PWN (2012)

Problem. *Wyznacz najdłuższy podciąg rosnący ciągu liczb* $x_1, \ldots, x_n$.

Np. dla ciągu $5, 2, 8, 6, 3, 6, 9, 7$ rozwiązaniem jest podciąg $2, 3, 6, 9$.

8.2 NAJDŁUŻSZY PODCIĄG ROSNĄCY

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.2, PWN (2012)

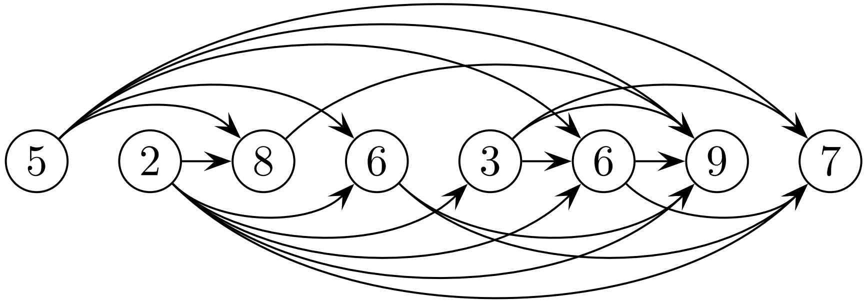
Problem. *Wyznacz najdłuższy podciąg rosnący ciągu liczb* $x_1, \ldots, x_n$.

Np. dla ciągu $5, 2, 8, 6, 3, 6, 9, 7$ rozwiązaniem jest podciąg $2, 3, 6, 9$.

Idea rozwiązania

- Zdefiniujmy graf skierowany $G = (V, A)$, którego wierzchołki odpowiadają liczbom x_i, a dwa wierzchołki x_i i x_j są połączone łukiem (x_i, x_j) wtedy i tylko wtedy, gdy $i < j$ oraz $x_i < x_j$.

5 2 8 6 3 6 9 7



- Graf G jest DAG.
- *Istnieje wzajemna jednoznaczność pomiędzy najdłuższą ścieżką w* G *a najdłuższym podciągiem rosnącym ciągu* $x_1, \ldots, x_n$.

8.2 NAJDŁUŻSZY PODCIĄG ROSNĄCY

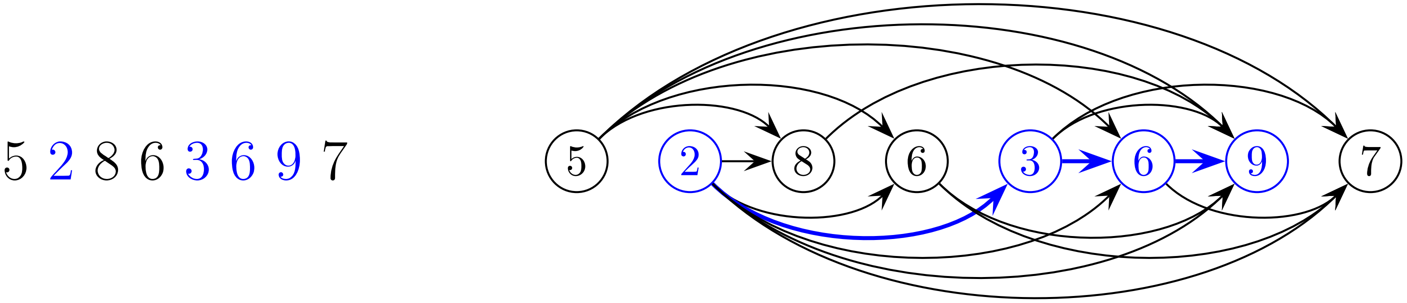
S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.2, PWN (2012)

Problem. *Wyznacz najdłuższy podciąg rosnący ciągu liczb* $x_1, \ldots, x_n$.

Np. dla ciągu $5, 2, 8, 6, 3, 6, 9, 7$ rozwiązaniem jest podciąg $2, 3, 6, 9$.

Idea rozwiązania

- Zdefiniujmy graf skierowany $G = (V, A)$, którego wierzchołki odpowiadają liczbom x_i, a dwa wierzchołki x_i i x_j są połączone łukiem (x_i, x_j) wtedy i tylko wtedy, gdy $i < j$ oraz $x_i < x_j$.



- Graf G jest DAG.
- *Istnieje wzajemna jednoznaczność pomiędzy najdłuższą ścieżką w* G *a najdłuższym podciągiem rosnącym ciągu* $x_1, \ldots, x_n$.

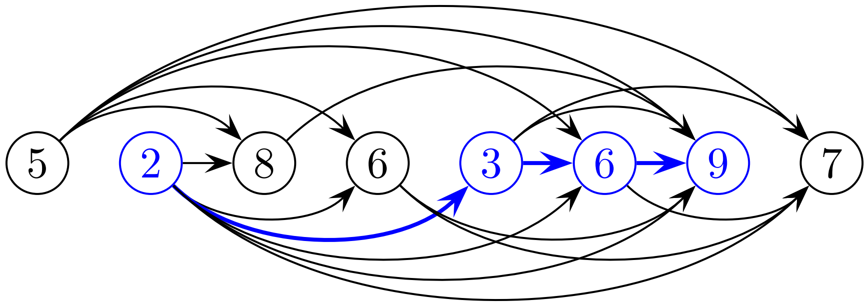
8.2 NAJDŁUŻSZY PODCIĄG ROSNĄCY

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.2, PWN (2012)

Problem. *Wyznacz najdłuższy podciąg rosnący ciągu liczb* $x_1, \ldots, x_n$.

Np. dla ciągu $5, 2, 8, 6, 3, 6, 9, 7$ rozwiązaniem jest podciąg $2, 3, 6, 9$.

5 2 8 6 3 6 9 7



Algorytm /zwraca tylko długość najdłuższego podciągu rosnącego/
↳ length(v): długość najdłuższej ścieżki kończącej się w wierzchołku v.

```
1. length($x_1$) := 0.
2. for $j$ := 2 to $n$ do length($x_j$) := $1 + \max\{$length$(x_i) : (i, j) \in A\}$.
3. return $\max_{j \in \{1,\ldots,n\}}\{$length$(x_j)\}$.
```

Aby rozwiązać problem najdłuższego podciągu rosnącego, zgodnie z ideą programowania dynamicznego, zdefiniowaliśmy:

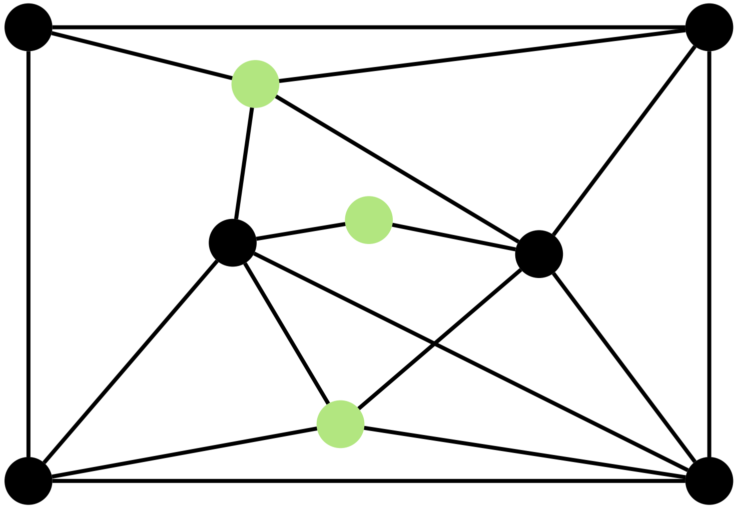
- zbiór $\left\{\mathsf{length}(v) : v \in V\right\}$ podproblemów do rozwiązania;
- uporządkowanie tych podproblemów (kolejność ich rozwiązywania);
- oraz relację, która pozwala na rozwiązanie każdego podproblemu korzystając z rozwiązań „mniejszych” podproblemów, tzn. tych podproblemów, które znajdują się wcześniej w uporządkowaniu.

Uwaga. *Programowanie dynamiczne zawsze wiąże się ze zdefiniowaniem odpowiedniego DAG podproblemów, choć graf ten często nie jest podawany* explicite*.*

8.3 NAJWIĘKSZY ZBIÓR NIEZALEŻNY

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.7, PWN (2012)

Podzbiór wierzchołków $S \subseteq V$ grafu $G = (V, E)$ jest zbiorem *niezależnym* wtedy i tylko wtedy, gdy żadne dwa wierzchołki z S nie są sąsiednie w G.

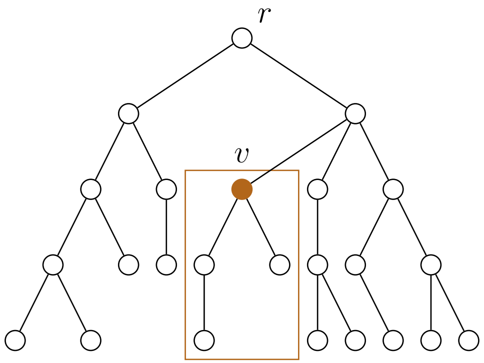


(Maksymalny) Zbiór niezależny mocy 3.

Problem. *Dla danego drzewa wyznaczyć największy zbiór niezależny.*

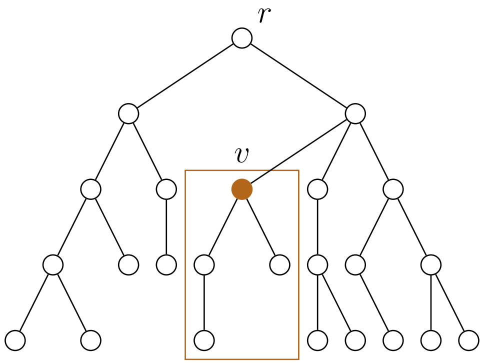
Niech r będzie korzeniem drzewa $T = (V, E)$. Wówczas z dowolnym wierzchołkiem v można związać poddrzewo zakorzenione w v, co automatycznie sugeruje postać podproblemu:

$I(v)$ = rozmiar największego zbioru niezależnego w poddrzewie o korzeniu w v.



Niech r będzie korzeniem drzewa $T = (V, E)$. Wówczas z dowolnym wierzchołkiem v można związać poddrzewo zakorzenione w v, co automatycznie sugeruje postać podproblemu:

$I(v)$ = rozmiar największego zbioru niezależnego w poddrzewie o korzeniu w v.



Idea algorytmu /metoda programowania dynamicznego/

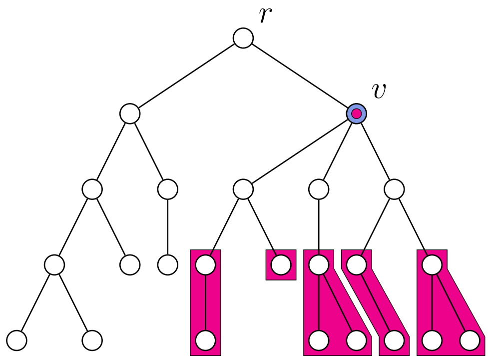
- Jeśli v jest liściem, wówczas $I(v) = 1$.
- Przyjmijmy, że znamy rozwiązania dla wszystkich potomków wierzchołka v. Wówczas $I(v)$ można określić rekurencyjnie następująco:

$$I(v) = \max\left\{1 + \sum\nolimits_{\text{wnuk } w \text{ wierzchołka } v} I(w), \sum\nolimits_{\text{potomek } w \text{ wierzchołka } v} I(w)\right\}.$$

- Rozwiązaniem jest $I(r)$.
- Złożoność czasowa: $2 \cdot \sum_{v \in V} \deg(v) \cdot O(1) = O(n)$.

Niech r będzie korzeniem drzewa $T = (V, E)$. Wówczas z dowolnym wierzchołkiem v można związać poddrzewo zakorzenione w v, co automatycznie sugeruje postać podproblemu:

$I(v)$ = rozmiar największego zbioru niezależnego w poddrzewie o korzeniu w v.



Idea algorytmu /metoda programowania dynamicznego/

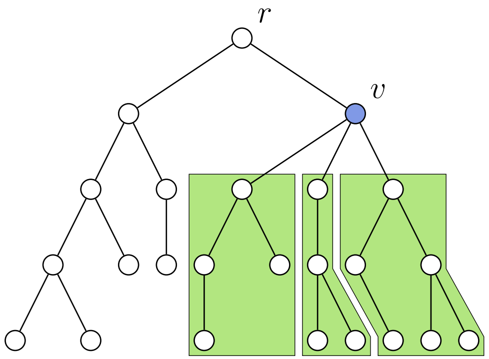
- Jeśli v jest liściem, wówczas $I(v) = 1$.
- Przyjmijmy, że znamy rozwiązania dla wszystkich potomków wierzchołka v. Wówczas $I(v)$ można określić rekurencyjnie następująco:

$$I(v) = \max\left\{1 + \sum_{\text{wnuk } w \text{ wierzchołka } v} I(w), \sum_{\text{potomek } w \text{ wierzchołka } v} I(w)\right\}.$$

- Rozwiązaniem jest $I(r)$.
- Złożoność czasowa: $2 \cdot \sum_{v \in V} \deg(v) \cdot O(1) = O(n)$.

Niech r będzie korzeniem drzewa $T = (V, E)$. Wówczas z dowolnym wierzchołkiem v można związać poddrzewo zakorzenione w v, co automatycznie sugeruje postać podproblemu:

$I(v)$ = rozmiar największego zbioru niezależnego w poddrzewie o korzeniu w v.

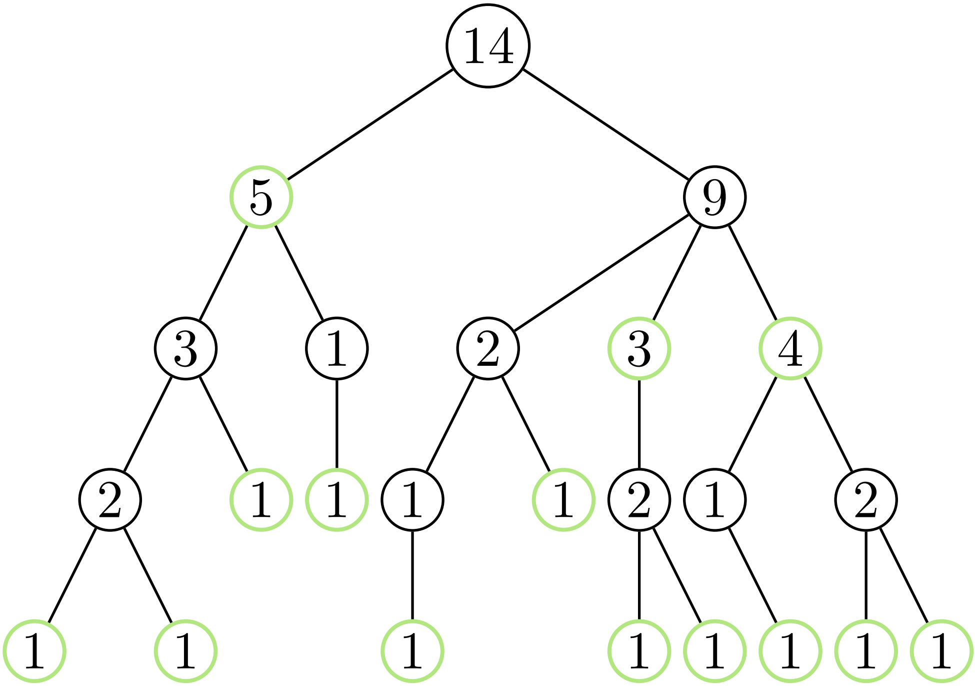


Idea algorytmu /metoda programowania dynamicznego/

- Jeśli v jest liściem, wówczas $I(v) = 1$.
- Przyjmijmy, że znamy rozwiązania dla wszystkich potomków wierzchołka v. Wówczas $I(v)$ można określić rekurencyjnie następująco:

$$I(v) = \max\left\{1 + \sum_{\text{wnuk } w \text{ wierzchołka } v} I(w), \sum_{\text{potomek } w \text{ wierzchołka } v} I(w)\right\}.$$

- Rozwiązaniem jest $I(r)$.
- Złożoność czasowa: $2 \cdot \sum_{v \in V} \deg(v) \cdot O(1) = O(n)$.



$$I(14) = \max\{1 + (3 + 1 + 2 + 3 + 4), 5 + 9\} = \max\{14, 14\} = 14$$

8.4 PROBLEM WSZYSTKICH NAJKRÓTSZYCH ŚCIEŻEK

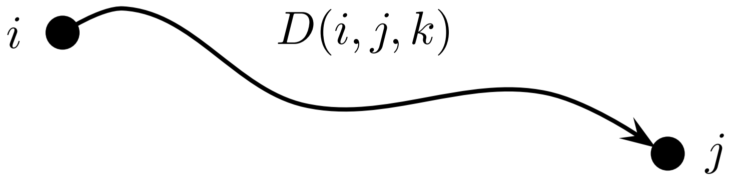
Problem. *Dla danego skierowanego grafu ważonego* $G = (V, E, w)$ *wyznaczyć najkrótsze ścieżki pomiędzy wszystkimi parami wierzchołków.*

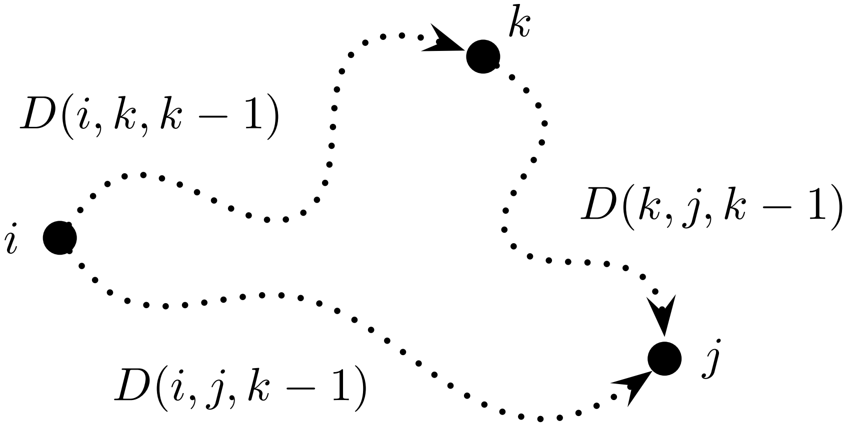
- Wyznaczenie najkrótszych ścieżek po kolei dla każdego z wierzchołów: algorytm Bellmana-Forda, złożoność rzędu $|V| \cdot O(|V| \cdot |E|) = O(|V|^2 \cdot |E|)$.
- Algorytm Dijkstra o złożoności $O(|V| \log |V| + |E|)$ może być zastosowany jedynie w grafach o dodatnich wagach.

8.4 PROBLEM WSZYSTKICH NAJKRÓTSZYCH ŚCIEŻEK

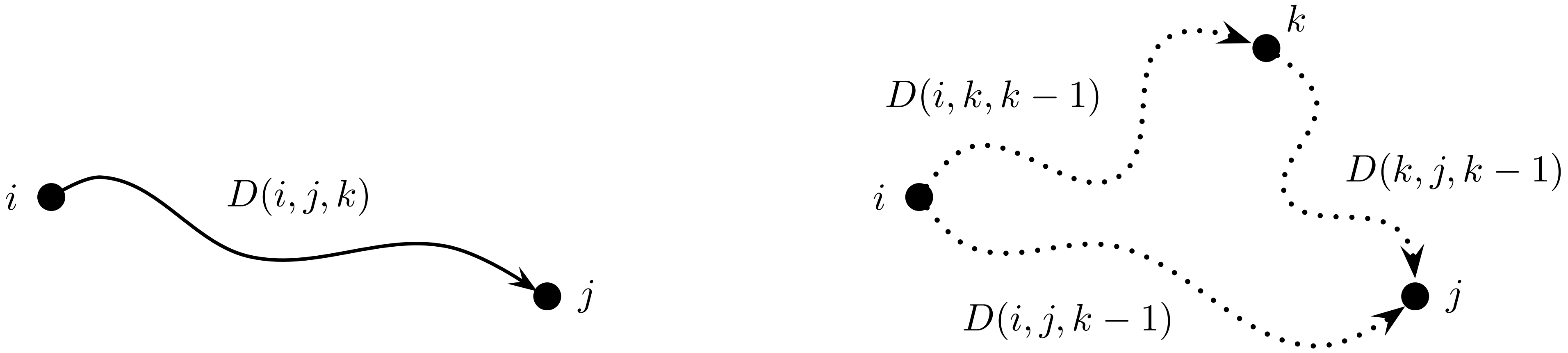
Problem. *Dla danego skierowanego grafu ważonego* $G = (V, E, w)$ *wyznaczyć najkrótsze ścieżki pomiędzy wszystkimi parami wierzchołków.*

- Wyznaczenie najkrótszych ścieżek po kolei dla każdego z wierzchołów: algorytm Bellmana-Forda, złożoność rzędu $|V| \cdot O(|V| \cdot |E|) = O(|V|^2 \cdot |E|)$.
- Algorytm Dijkstra o złożoności $O(|V| \log |V| + |E|)$ może być zastosowany jedynie w grafach o dodatnich wagach.





Obserwacja. *Dla dowolnych dwóch wierzchołków* $i, j \in V = \{1, \ldots, n\}$, *najkrótsza ścieżka pomiędzy* i *oraz* j *składa się z pewnej liczby wierzchołków pomiędzy* i *oraz* j. *A zatem możemy stopniowo obliczać najkrótsze ścieżki, tj. dopuszczając tylko* k *wierzchołków pośrednich* $1, \ldots, k$.



Algorytm Floyd'a-Warshall'a (1959/1962)

- Ponumerujmy wierzchołki liczbami od 1 do n. Niech $D(i,j,k)$ oznacza najkrótszą ścieżkę pomiędzy wierzchołkami i oraz j, która przechodzi co najwyżej przez wierzchołki $1, 2, \ldots, k$.
- Wartość $D(i,j,0)$ można określić z definicji:

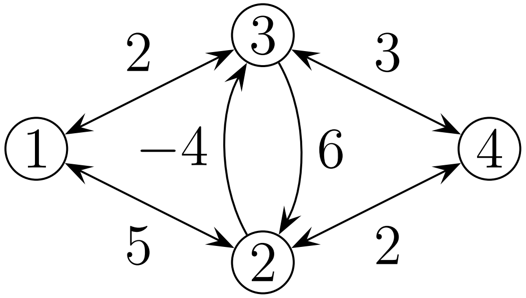
$$D(i,j,0) = \begin{cases} w(i,j) & \text{jeśli } (i,j) \in E; \\ \infty & \text{w przeciwnym wypadku.} \end{cases}$$

- Wartość $D(i,j,k)$, $k \geq 1$, można określić rekurencyjnie następująco:

$$D(i,j,k) = \min\{D(i,j,k-1), D(i,k,k-1) + D(k,j,k-1)\}.$$

- Złożoność czasowa: $O(n^3)$.

Przykład



$D(i,j,0)$	1	2	3	4
1	∞	5	2	∞
2	5	∞	-4	2
3	2	6	∞	3
4	∞	2	3	∞

$D(i,j,1)$	1	2	3	4
1	∞	5	2	∞
2	5	10	-4	2
3	2	6	4	3
4	∞	2	3	∞

$D(i,j,2)$	1	2	3	4
1	10	5	1	7
2	5	10	-4	2
3	2	6	2	3
4	7	2	-2	4

$$\begin{aligned}
\text{Na przykład: } D(2,2,1) &= \min\{D(2,2,0), D(2,1,0) + D(1,2,0)\} \\
&= \min\{+\infty, 5+5\} = 10; \\
D(1,3,2) &= \min\{D(1,3,1), D(1,2,1) + D(2,3,1)\} \\
&= \min\{2, 5+(-4)\} = 1; \\
D(3,3,2) &= \min\{D(3,3,1), D(3,2,1) + D(2,3,1)\} \\
&= \min\{+\infty, 6+(-4)\} = 2; \\
D(4,1,2) &= \min\{D(4,1,1), D(4,2,1) + D(2,1,1)\} \\
&= \min\{+\infty, 2+5\} = 7;
\end{aligned}$$

Przykład (cont)

$D(i,j,3)$	1	2	3	4
1	3	5	1	4
2	-2	2	-4	-1
3	2	6	2	3
4	0	2	-2	1

$D(i,j,4)$	1	2	3	4
1	3	5	1	4
2	-2	1	-4	-1
3	2	5	1	3
4	0	2	-2	1

Na przykład:
$$\begin{aligned}
D(1,1,3) &= \min\{D(1,1,2), D(1,3,2)+D(3,1,2)\}\\
&= \min\{10, 1+2\} = 3;\\
D(1,4,3) &= \min\{D(1,4,2), D(1,3,2)+D(3,4,2)\}\\
&= \min\{7, 1+3\} = 4;\\
D(2,2,3) &= \min\{D(2,2,2), D(2,3,2)+D(3,2,2)\}\\
&= \min\{10, (-4)+6\} = 2;\\
D(2,4,3) &= \min\{D(2,4,2), D(2,3,2)+D(4,2,2)\}\\
&= \min\{2, (-4)+3\} = -1;\\
D(3,2,4) &= \min\{D(3,2,3), D(3,4,3)+D(4,2,3)\}\\
&= \min\{6, 3+2\} = 5.
\end{aligned}$$

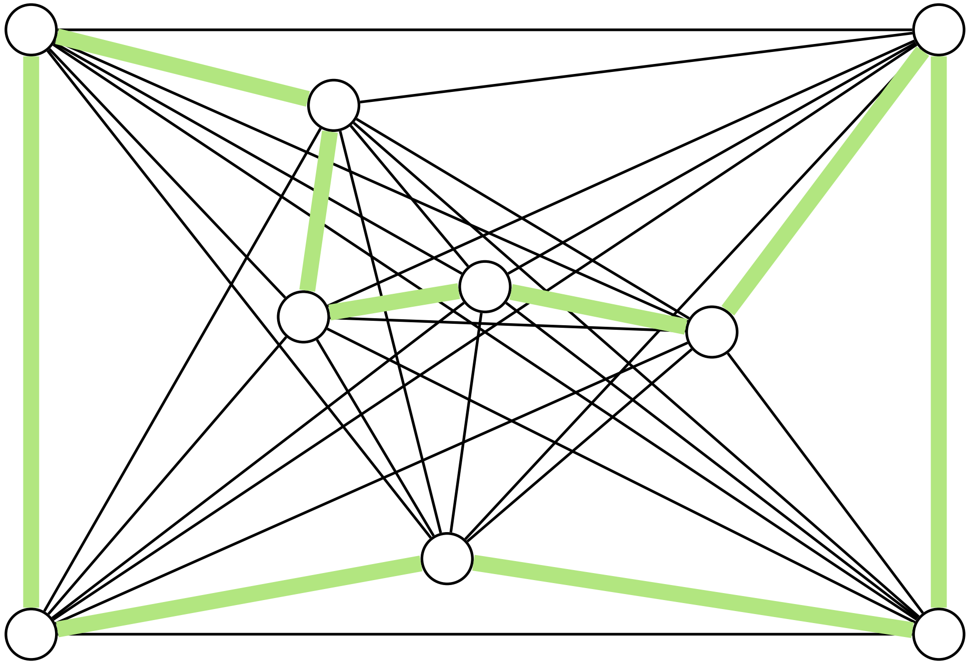
8.5 PROBLEM KOMIWOJAŻERA (W GRAFACH)

M. Held, R. M. Karp, A dynamic programming approach to sequencing problems
J. for the Society for Industrial and Applied Mathematics 10(1), 196-210 (1962)

Problem. *Dla danego ważonego skierowanego grafu pełnego* $G = (V, A, w)$ *znajdź cykl Hamiltona* $C \subseteq G$ *o najmniejszym koszcie.*

↳ *Koszt* cyklu C: suma wag jego krawędzi.

↳ Problem NP-trudny.

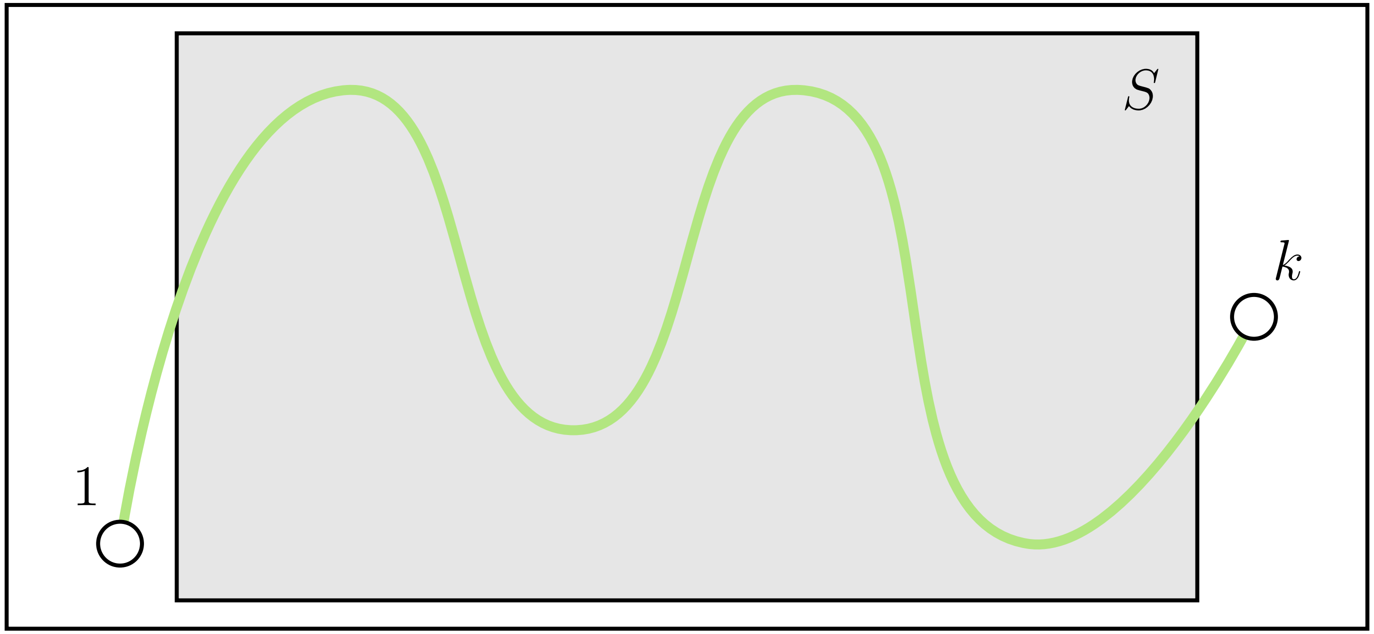


Ważony graf $G = (V, A, w)$; tutaj długość łuku (u, v) równa się długości łuku (v, u).

Idea algorytmu /Held, Karp, Bellman 1962/

- Algorytm opiera się na własności, że dowolna ścieżka zawarta w najkrótszej ścieżce jest sama najkrótszą ścieżką.
- Załóżmy, że wierzchołki grafu ponumerowane są liczbami $1, \ldots, n$.
- Dla podzbioru $S \subseteq \{1, \ldots, n\}$ wierzchołków definiujemy:

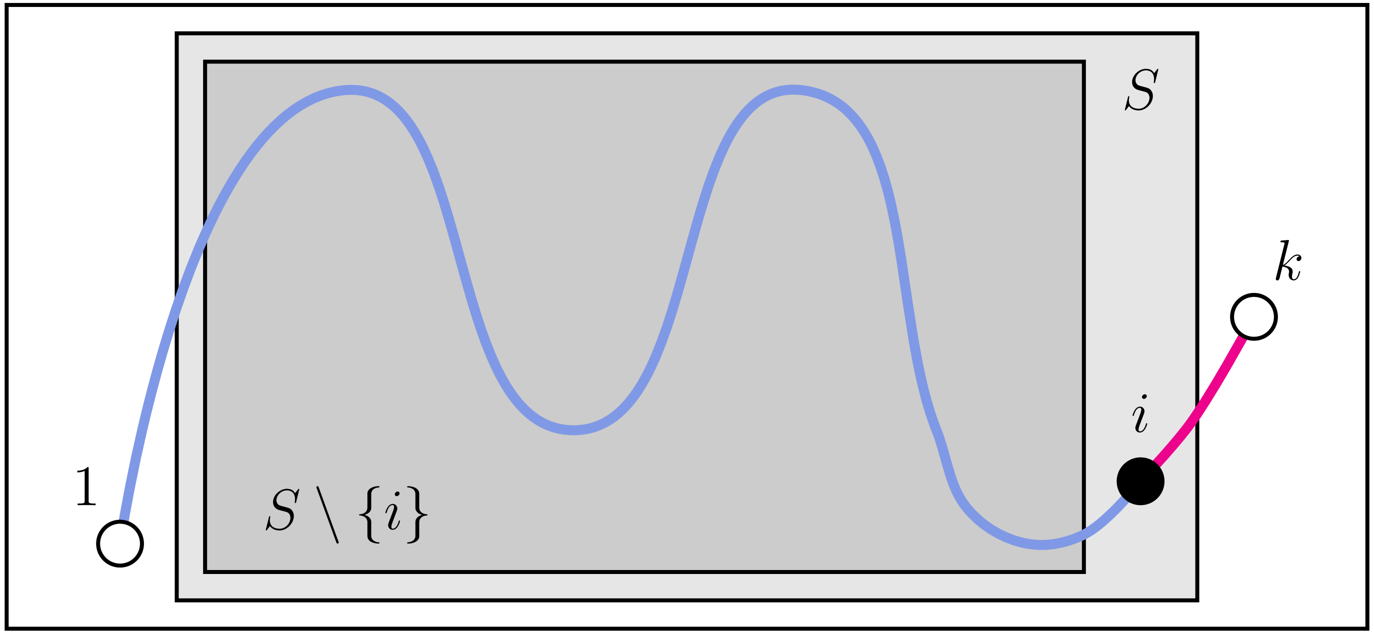
$P(S, k)$ – najkrótsza ścieżka łącząca
wierzchołek 1 z wierzchołkiem k,
która odwiedza po drodze tylko wierzchołki z S.



Idea algorytmu /Held, Karp, Bellman 1962/

- Algorytm opiera się na własności, że dowolna ścieżka zawarta w najkrótszej ścieżce jest sama najkrótszą ścieżką.
- Załóżmy, że wierzchołki grafu ponumerowane są liczbami $1, \dots, n$.
- Dla podzbioru $S \subseteq \{1, \dots, n\}$ wierzchołków definiujemy:

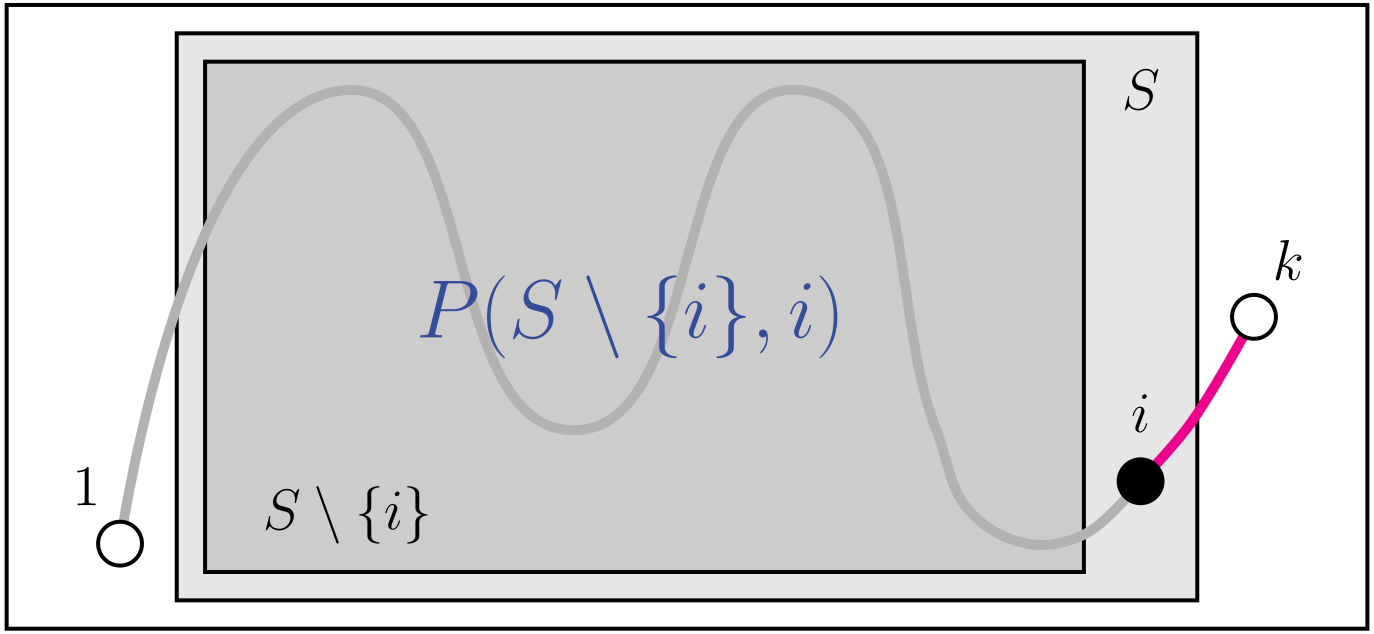
$P(S, k)$ – najkrótsza ścieżka łącząca
wierzchołek 1 z wierzchołkiem k,
która odwiedza po drodze tylko wierzchołki z S.



Idea algorytmu /Held, Karp, Bellman 1962/

- Algorytm opiera się na własności, że dowolna ścieżka zawarta w najkrótszej ścieżce jest sama najkrótszą ścieżką.
- Załóżmy, że wierzchołki grafu ponumerowane są liczbami $1, \ldots, n$.
- Dla podzbioru $S \subseteq \{1, \ldots, n\}$ wierzchołków definiujemy:

  $P(S, k)$ – najkrótsza ścieżka łącząca wierzchołek 1 z wierzchołkiem k, która odwiedza po drodze tylko wierzchołki z S.



Idea algorytmu /Held, Karp, Bellman 1962/

- Algorytm opiera się na własności, że dowolna ścieżka zawarta w najkrótszej ścieżce jest sama najkrótszą ścieżką.
- Załóżmy, że wierzchołki grafu ponumerowane są liczbami $1, \ldots, n$.
- Dla podzbioru $S \subseteq \{1, \ldots, n\}$ wierzchołków definiujemy:

  $P(S, k)$ – najkrótsza ścieżka łącząca
  wierzchołek 1 z wierzchołkiem k,
  która odwiedza po drodze tylko wierzchołki z S.

$$\mathsf{koszt}(P(S, k)) = \min_{i \in S} \mathsf{koszt}(P(S \setminus \{i\}, i)) + w(i, k)$$

- Rozwiązanie jest wyznaczone przez $k \in \{2, \ldots, n\}$, które minimalizuje sumę

$$\mathsf{koszt}(P(\{2, \ldots, n\} \setminus \{k\}, k)) + w(k, 1).$$

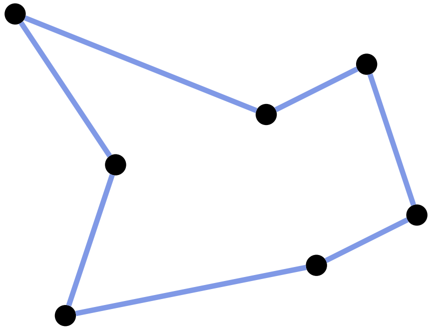
- Złożoność czasowa: $O(n^2 \cdot 2^n)$.
- Złożoność pamięciowa: $O(n \cdot 2^n)$.

8.6 Bitoniczny problem komiwojażera

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein
Wprowadzenie do algorytmów, problem 15-1, WNT (2005)

Euklidesowy problem komiwojażera polega na wyznaczeniu najkrótszego cyklu (zamkniętej ścieżki), który trawersuje wszystkie n danych punktów na płaszczyźnie.

- Problem NP-trudny.
- Programowanie dynamiczne: $O(n^2 \cdot 2^n)$.

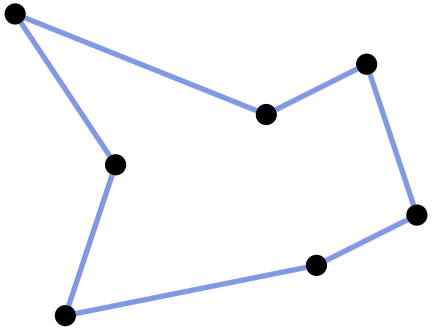


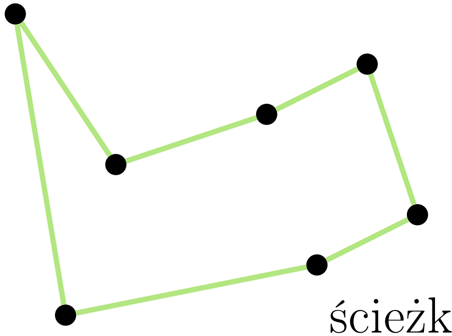
8.6 BITONICZNY PROBLEM KOMIWOJAŻERA

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein
Wprowadzenie do algorytmów, problem 15-1, WNT (2005)

Euklidesowy problem komiwojażera polega na wyznaczeniu najkrótszego cyklu (zamkniętej ścieżki), który trawersuje wszystkie n danych punktów na płaszczyźnie.

- Problem NP-trudny.
- Programowanie dynamiczne: $O(n^2 \cdot 2^n)$.





ścieżka bitoniczna

Bitoniczna ścieżka zamknięta jest to ścieżka, która zaczyna się w skrajnie lewym punkcie, biegnie następnie ciągle w prawo aż do punktu skrajnie prawego, po czym przebiega w lewo do punktu początkowego.

Problem. *Dla danego zbioru $P = \{p_1, \ldots, p_n\}$ punktów na płaszczyźnie znajdź najkrótszą zamknietą ścieżkę bitoniczną.*

Przyjmijmy, że wszystkie punkty mają różne współrzędne x oraz że są posortowane względem tej współrzędnej.

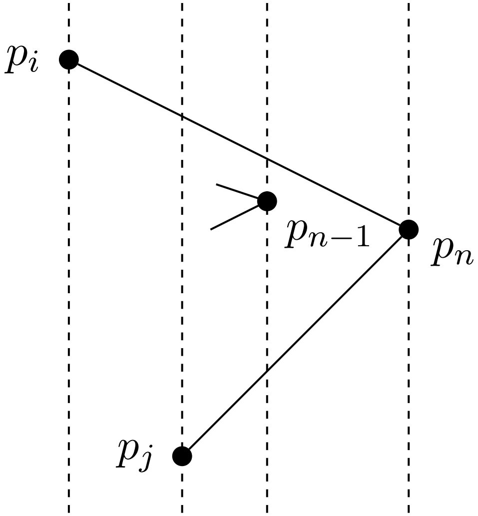
Idea algorytmu /wyznaczanie kosztu/ (cont.)

- Zauważmy, że *w dowolnym rozwiązaniu (zamkniętej ścieżce bitonicznej) punkty p_{n-1} i p_n muszą być ze sobą połączone*.

Przyjmijmy, że wszystkie punkty mają różne współrzędne x oraz że są posortowane względem tej współrzędnej.

Idea algorytmu /wyznaczanie kosztu/ (cont.)

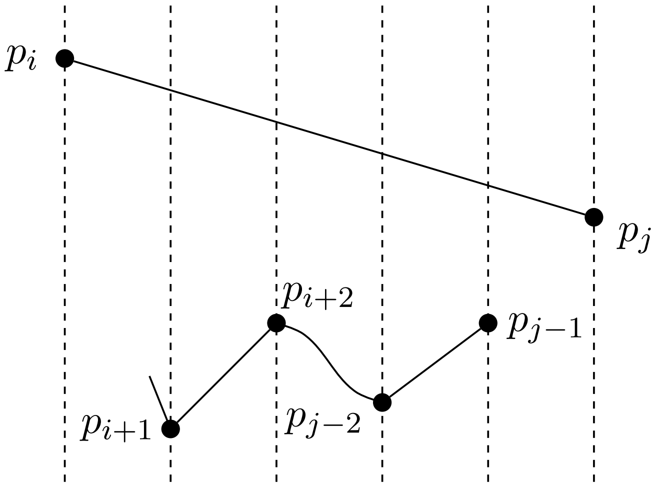
- Zauważmy, że *w dowolnym rozwiązaniu (zamkniętej ścieżce bitonicznej) punkty p_{n-1} i p_n muszą być ze sobą połączone*.



Dowód. W przeciwnym wypadku, jeśli p_i oraz p_j są sąsiadami p_n, $i, j \neq n-1$, wówczas ścieżka trawersuje wierzechołki w kolejności p_i, p_n, p_j, p_{n-1}. Ale że odcięte punktów p_i oraz p_j są mniejsze od współrzędnej x punktu p_{n-1}, ścieżka ta nie spełnia warunku bycia bitoniczną — sprzeczność. $\square$

Idea algorytmu /wyznaczanie kosztu/ (cont.)

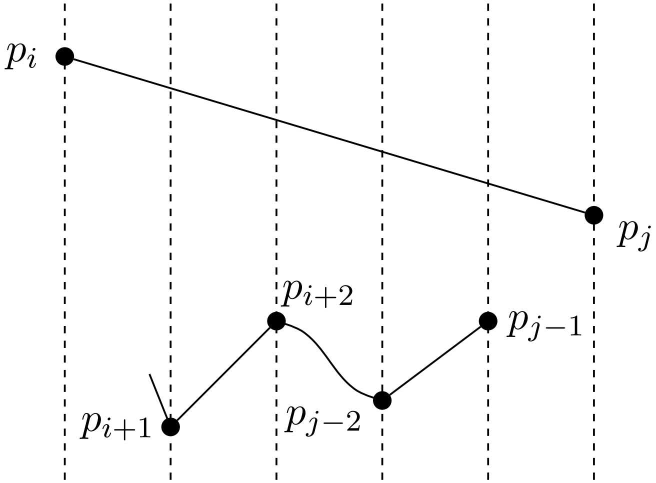
- Zdefiniujmy $B(j)$, $1 \leq j \leq n$, jako najkrótszą niezamkniętą bitoniczną ścieżkę łączącą wierzchołki p_{j-1} oraz p_j, trawersującą wszystkie wierzchołki $p_1, \ldots, p_j$.



- *Jeśli p_i jest sąsiadem wierzchołka p_j, $i < j$, w optymalnym rozwiązaniu $B(j)$, wówczas wierzchołki $p_{i+1}, p_{i+2}, \ldots, p_{j-1}$ muszą występować kolejno na ścieżce $B(j)$,* tzn. ścieżka $B(j)$ składa się z odcinka $\overline{p_i p_j}$, ścieżki $B(i+1)$ oraz ścieżki $p_{i+1}, p_{i+2}, \ldots, p_{j-1}$. /Wynika to z bitoniczności podścieżki $p_{i+1}, \ldots, p_{j-1}$./

Idea algorytmu /wyznaczanie kosztu/ (cont.)

- Zdefiniujmy $B(j)$, $1 \leq j \leq n$, jako najkrótszą niezamkniętą bitoniczną ścieżkę łączącą wierzchołki p_{j-1} oraz p_j, trawersującą wszystkie wierzchołki $p_1, \ldots, p_j$.



- *Jeśli p_i jest sąsiadem wierzchołka p_j, $i < j$, w optymalnym rozwiązaniu $B(j)$, wówczas wierzchołki $p_{i+1}, p_{i+2}, \ldots, p_{j-1}$ muszą występować kolejno na ścieżce $B(j)$,* tzn. ścieżka $B(j)$ składa się z odcinka $\overline{p_i p_j}$, ścieżki $B(i+1)$ oraz ścieżki $p_{i+1}, p_{i+2}, \ldots, p_{j-1}$. /Wynika to z bitoniczności podścieżki $p_{i+1}, \ldots, p_{j-1}$./
- Aby wyznaczyć $B(j)$, wystarczy sprawdzić, która z wartości $B(i+1)$, $i < j-1$, wraz z kosztem ścieżki $p_{i+1}, p_{i+2}, \ldots, p_{j-1}$ i odcinka $\overline{p_i p_j}$, jest najmniejsza.
- Rozwiązaniem jest $B(n) + \overline{p_{n-1} p_n}$.

Algorytm

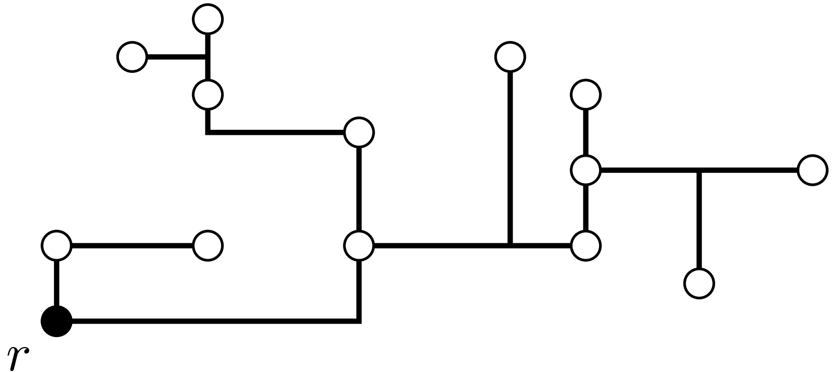
```
1. $B(2) := \mathrm{dist}_{\mathcal{E}}(p_1, p_2)$;
2. for $j := 3$ to $n$ do
  2.1      $\min := \infty$;
  2.2      $\mathrm{suma} := 0$;
  2.3      for $i := j - 2$ downto $1$ do
    2.3.1        $d := B(i+1) + \mathrm{dist}_{\mathcal{E}}(p_i, p_j) + \mathrm{suma}$;
    2.3.2        if $d < \min$ then $\min := d$;
    2.3.3        $\mathrm{suma} := \mathrm{suma} + \mathrm{dist}_{\mathcal{E}}(p_i, p_{i+1})$;
  2.4      $B(j) := \min$;
```

► Złożoność algorytmu jest rzędu $O(n^2)$.

8.7 PROBLEM RSA (RECTILINEAR STEINER ARBORESCENCE)

J.-M. Ho, M. T. Ko, T. H. Ma, T. Y. Sung
Algorithms for rectilinear optimal multicast trees
Lecture Notes in Computer Science 650, 106-115 (1992)

Niech r będzie punktem należącym do pierwszej ćwiartki Q_1 płaszczyzny $\mathbb{R}^2$ i niech $P \subset Q_1$ będzie zbiorem punktów, które są zdominowane przez r, tzn. dla dowolnego punktu $p \in P$ zachodzi $\mathrm{x}(r) \leq \mathrm{x}(p)$ oraz $\mathrm{y}(r) \leq \mathrm{y}(p)$. *Prostokątna gałąź Steinera* dla zbioru P o korzeniu r jest to takie prostokątne drzewo Steinera T dla zbioru $P \cup \{r\}$, w którym dla każdego punktu $p \in P$ długość (unikalnej) ścieżki w drzewie T łączącej p i r wynosi $(\mathrm{x}(p) - \mathrm{x}(r)) + (\mathrm{y}(p) - \mathrm{y}(r))$.

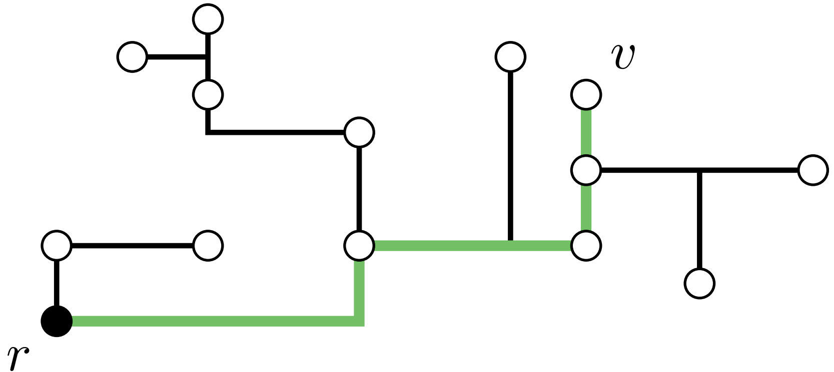


Prostokątne drzewo Steinera

8.7 PROBLEM RSA (RECTILINEAR STEINER ARBORESCENCE)

J.-M. Ho, M. T. Ko, T. H. Ma, T. Y. Sung
Algorithms for rectilinear optimal multicast trees
Lecture Notes in Computer Science 650, 106-115 (1992)

Niech r będzie punktem należącym do pierwszej ćwiartki Q_1 płaszczyzny $\mathbb{R}^2$ i niech $P \subset Q_1$ będzie zbiorem punktów, które są zdominowane przez r, tzn. dla dowolnego punktu $p \in P$ zachodzi $\mathrm{x}(r) \leq \mathrm{x}(p)$ oraz $\mathrm{y}(r) \leq \mathrm{y}(p)$. *Prostokątna gałęź Steinera* dla zbioru P o korzeniu r jest to takie prostokątne drzewo Steinera T dla zbioru $P \cup \{r\}$, w którym dla każdego punktu $p \in P$ długość (unikalnej) ścieżki w drzewie T łączącej p i r wynosi $(\mathrm{x}(p) - \mathrm{x}(r)) + (\mathrm{y}(p) - \mathrm{y}(r))$.



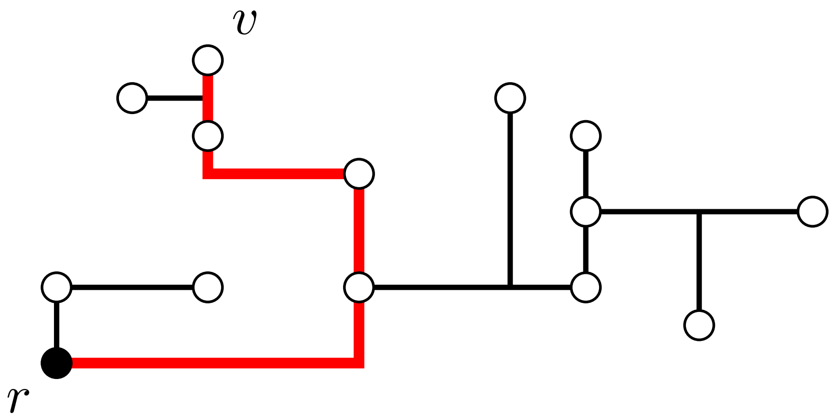
Prostokątne drzewo Steinera

$$|\pi(r, v)| = (\mathrm{x}(p) - \mathrm{x}(r)) + (\mathrm{y}(p) - \mathrm{y}(r))$$

8.7 PROBLEM RSA (RECTILINEAR STEINER ARBORESCENCE)

J.-M. Ho, M. T. Ko, T. H. Ma, T. Y. Sung
Algorithms for rectilinear optimal multicast trees
Lecture Notes in Computer Science 650, 106-115 (1992)

Niech r będzie punktem należącym do pierwszej ćwiartki Q_1 płaszczyzny $\mathbb{R}^2$ i niech $P \subset Q_1$ będzie zbiorem punktów, które są zdominowane przez r, tzn. dla dowolnego punktu $p \in P$ zachodzi $\mathrm{x}(r) \leq \mathrm{x}(p)$ oraz $\mathrm{y}(r) \leq \mathrm{y}(p)$. *Prostokątna gałęź Steinera* dla zbioru P o korzeniu r jest to takie prostokątne drzewo Steinera T dla zbioru $P \cup \{r\}$, w którym dla każdego punktu $p \in P$ długość (unikalnej) ścieżki w drzewie T łączącej p i r wynosi $(\mathrm{x}(p) - \mathrm{x}(r)) + (\mathrm{y}(p) - \mathrm{y}(r))$.



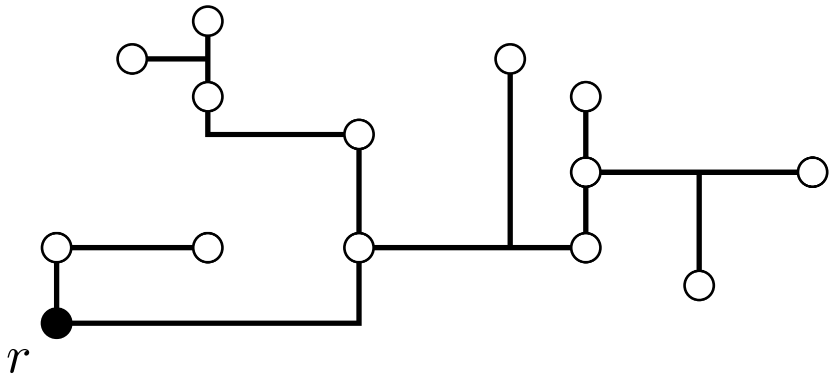
Prostokątne drzewo Steinera

$$|\pi(r,v)| \neq (\mathrm{x}(p) - \mathrm{x}(r)) + (\mathrm{y}(p) - \mathrm{y}(r))$$

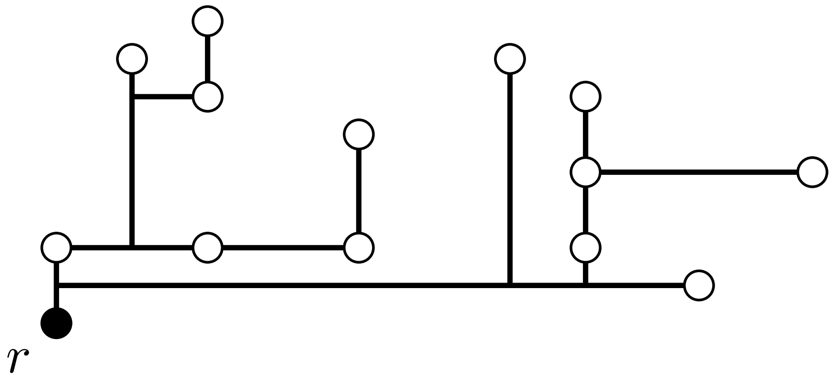
8.7 Problem RSA (Rectilinear Steiner Arborescence)

J.-M. Ho, M. T. Ko, T. H. Ma, T. Y. Sung
Algorithms for rectilinear optimal multicast trees
Lecture Notes in Computer Science 650, 106-115 (1992)

Niech r będzie punktem należącym do pierwszej ćwiartki Q_1 płaszczyzny $\mathbb{R}^2$ i niech $P \subset Q_1$ będzie zbiorem punktów, które są zdominowane przez r, tzn. dla dowolnego punktu $p \in P$ zachodzi $\mathrm{x}(r) \leq \mathrm{x}(p)$ oraz $\mathrm{y}(r) \leq \mathrm{y}(p)$. *Prostokątna gałęź Steinera* dla zbioru P o korzeniu r jest to takie prostokątne drzewo Steinera T dla zbioru $P \cup \{r\}$, w którym dla każdego punktu $p \in P$ długość (unikalnej) ścieżki w drzewie T łączącej p i r wynosi $(\mathrm{x}(p) - \mathrm{x}(r)) + (\mathrm{y}(p) - \mathrm{y}(r))$.



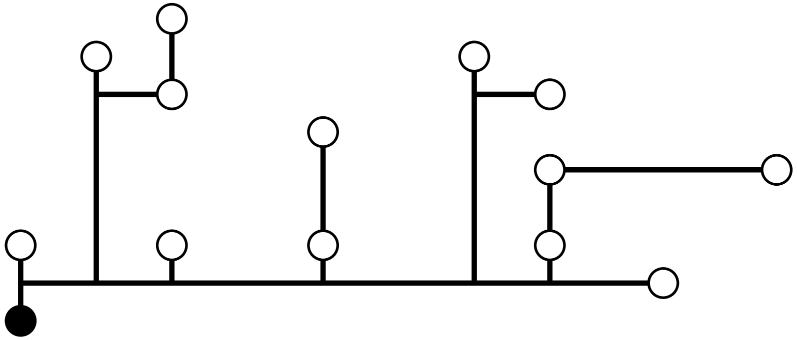
Prostokątne drzewo Steinera



Prostokątna gałąź Steinera

8.7 PROBLEM RSA (RECTILINEAR STEINER ARBORESCENCE)

J.-M. Ho, M. T. Ko, T. H. Ma, T. Y. Sung
Algorithms for rectilinear optimal multicast trees
Lecture Notes in Computer Science 650, 106-115 (1992)



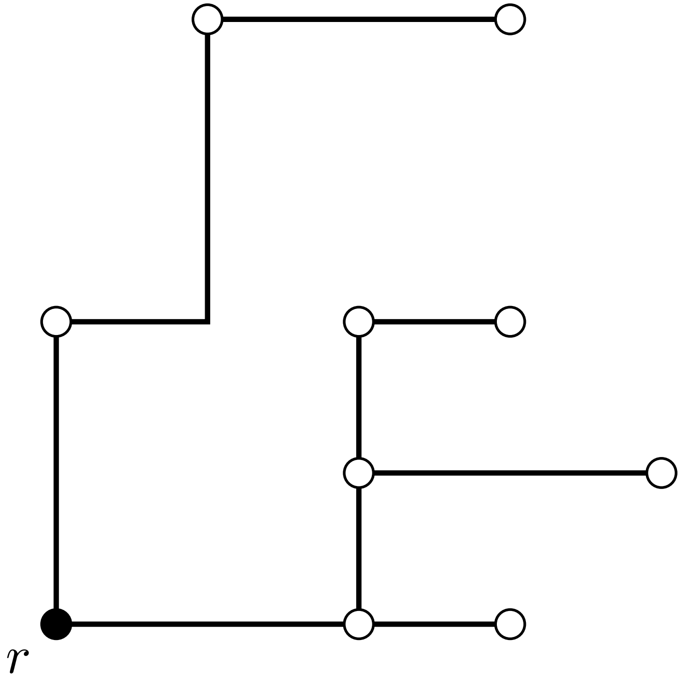
Najkrótsza prostokątna gałąź Steinera

Problem. (Rectilinear Steiner Arborescence) *Dla zbioru $P \cup \{r\}$ punktów z pierwszej ćwiartki płaszczyzny $\mathbb{R}^2$ takich, że r dominuje nad wszystkimi punktami ze zbioru P, wyznaczyć najkrótszą prostokątną gałąź Steinera dla P o korzeniu r.*

- *Koszt* prostokątnej gałęzi Steinera to suma długości jej krawędzi.
- Problem RSA jest problemem NP-trudnym (Shi, Su 2006).

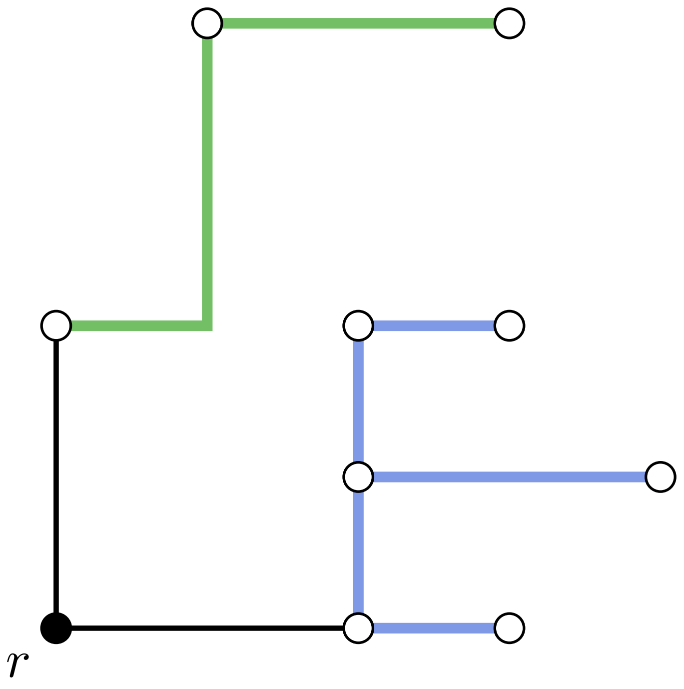
Idea algorytmu /wyznaczanie kosztu/

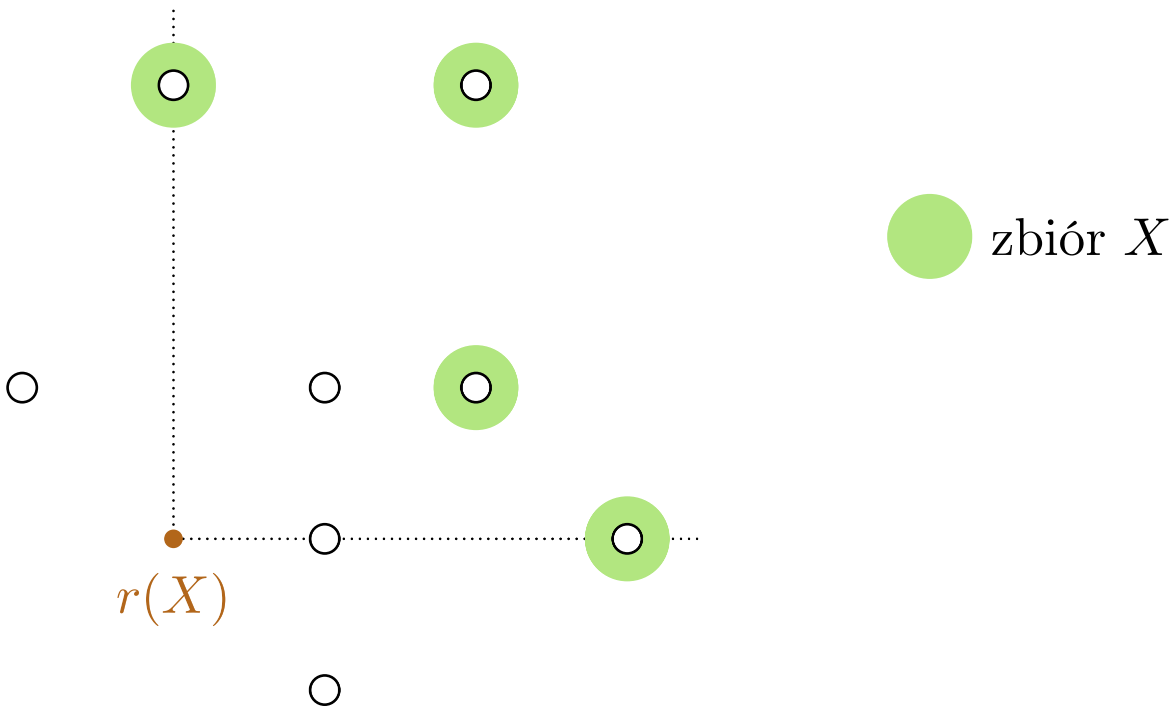
- Algorytm opiera się na własności, że najkrótsza prostokątna gałąź Steinera dla zbioru P o korzeniu r jest sumą dwóch najkrótszych prostokątnych gałęzi Steinera dla pewnej partycji zbioru P, dodając do tego koszt ich połączeń do korzenia r.



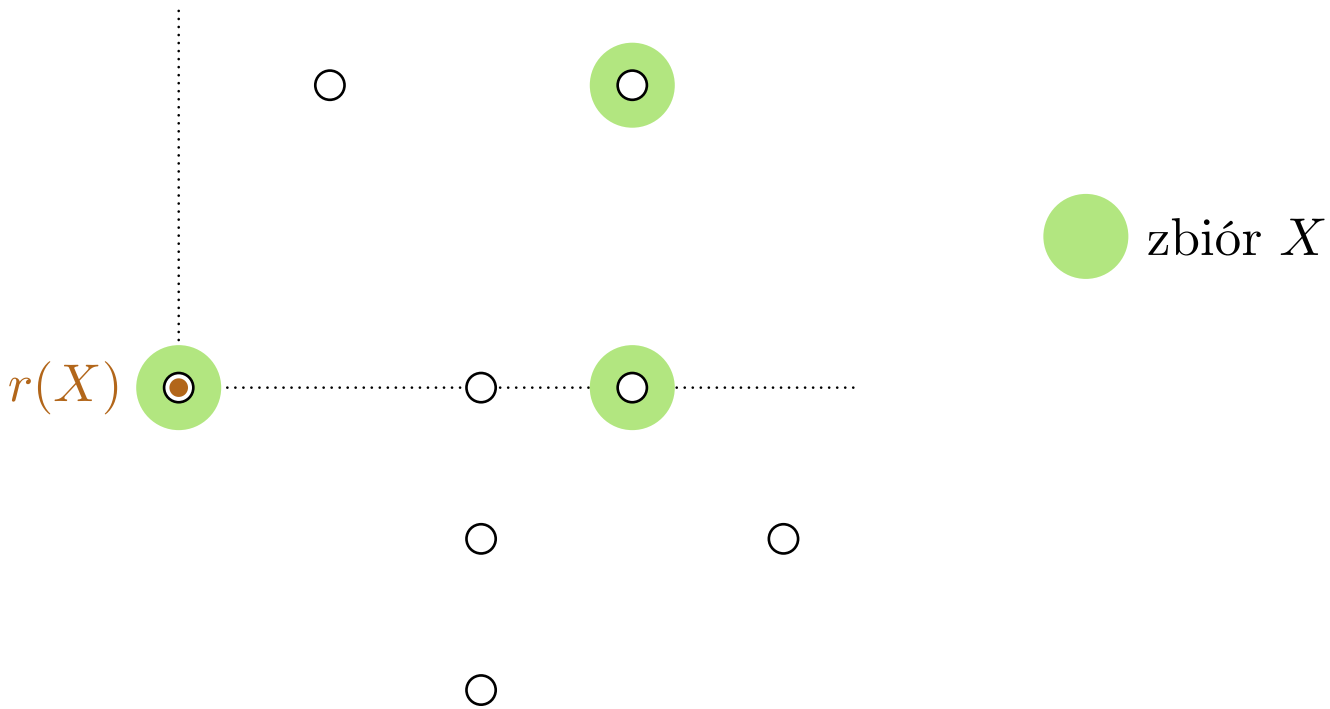
Idea algorytmu /wyznaczanie kosztu/

- Algorytm opiera się na własności, że najkrótsza prostokątna gałąź Steinera dla zbioru P o korzeniu r jest sumą dwóch najkrótszych prostokątnych gałęzi Steinera dla pewnej partycji zbioru P, dodając do tego koszt ich połączeń do korzenia r.

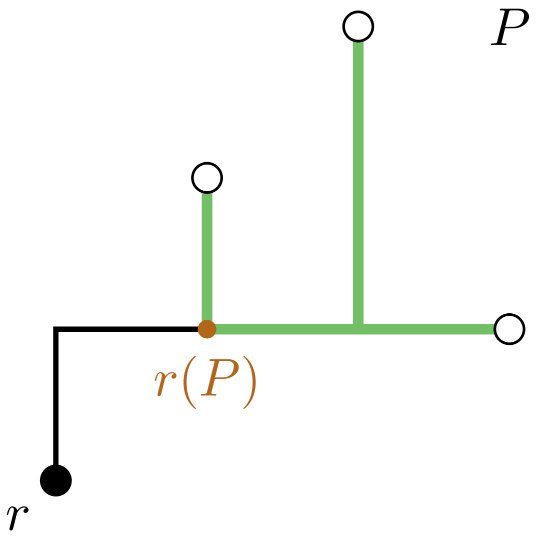




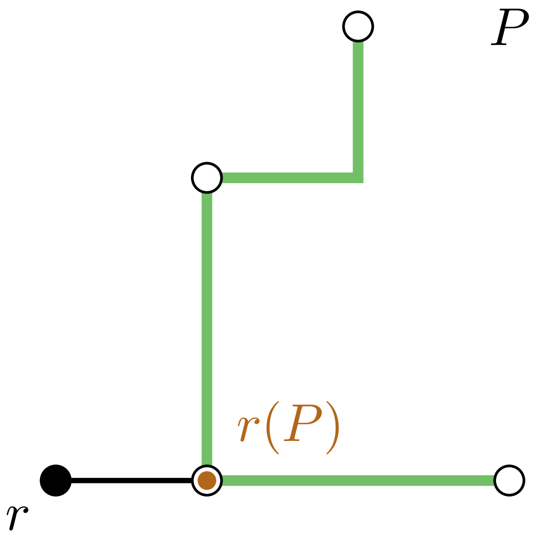
$r(X) = (x_{\min}, y_{\min})$, gdzie $x_{\min} = \min_{p \in X} \mathrm{x}(p)$ oraz $y_{\min} = \min_{p \in X} \mathrm{y}(p)$



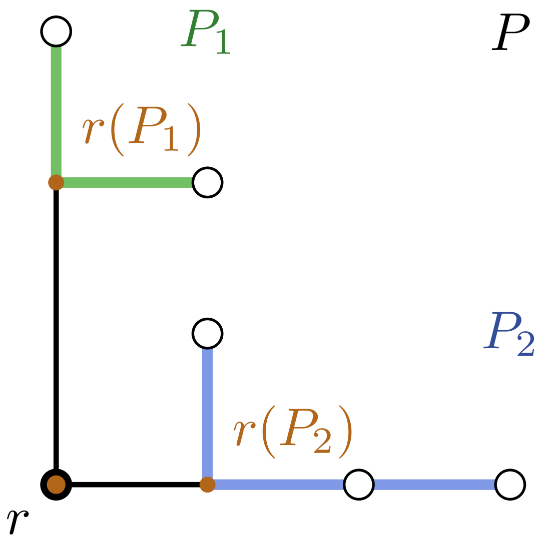
$r(X) = (x_{\min}, y_{\min})$, gdzie $x_{\min} = \min_{p \in X} \mathrm{x}(p)$ oraz $y_{\min} = \min_{p \in X} \mathrm{y}(p)$



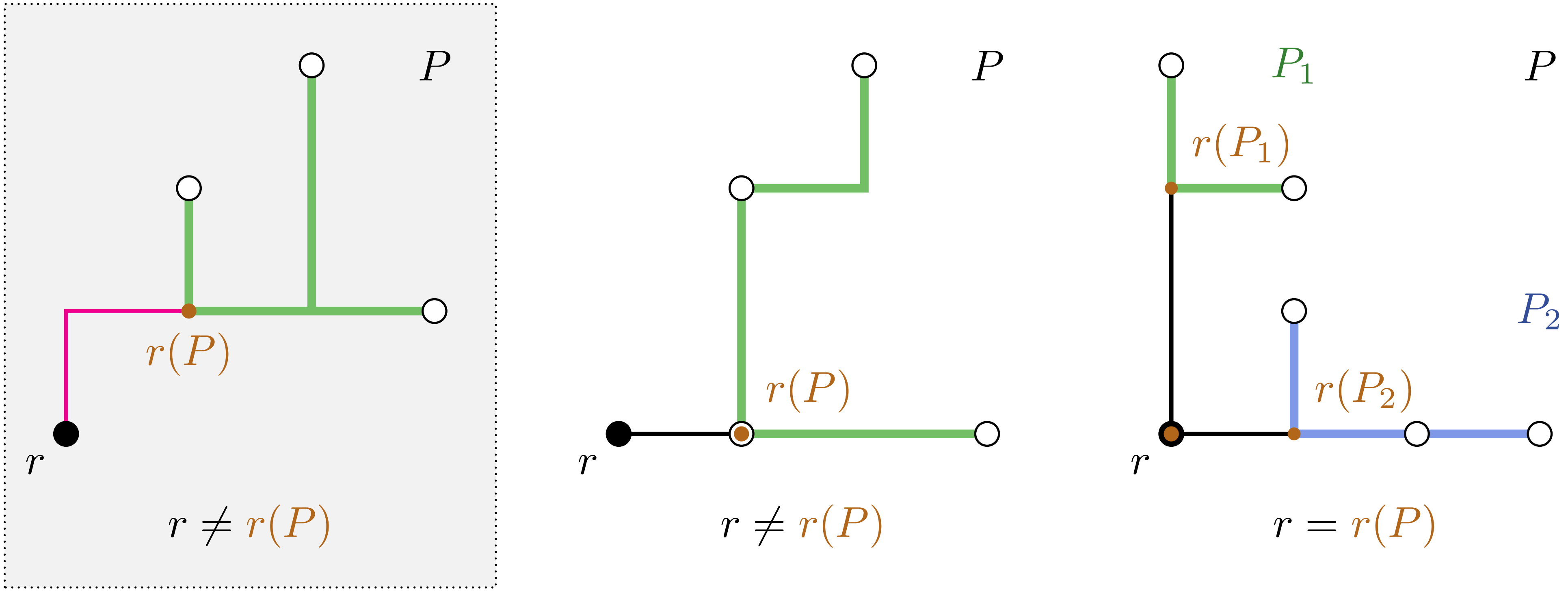
$r \neq r(P)$



$r \neq r(P)$



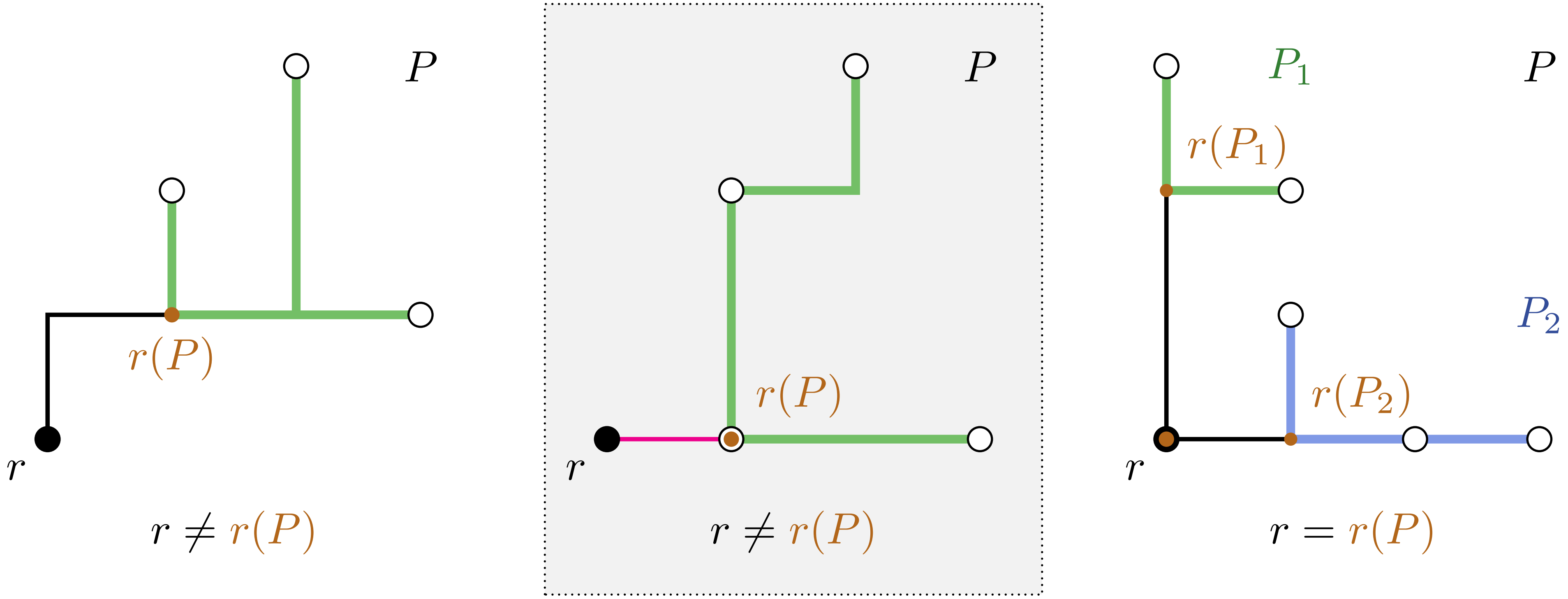
$r = r(P)$



$$\mathrm{OPT}(P) = \mathrm{OPT}^*(P) + d(r, r(P))$$

$\mathrm{OPT}^*(X)$ – najkrótsza prostokątna gałąź Steinera dla X o korzeniu $r(X)$

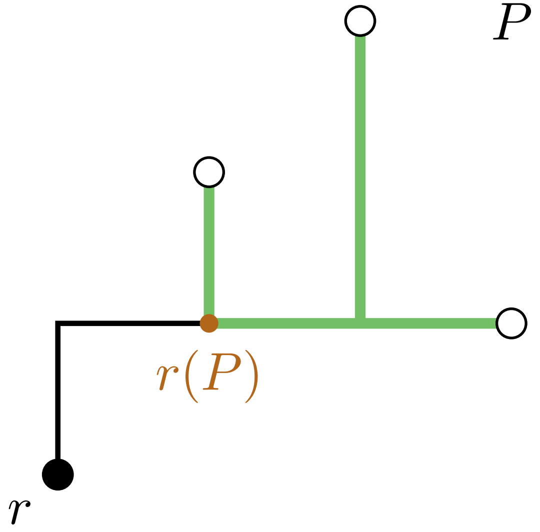
$d(r, r(X)) = (\mathrm{x}(r(X)) - \mathrm{x}(r)) + (\mathrm{y}(r(X)) - \mathrm{y}(r))$



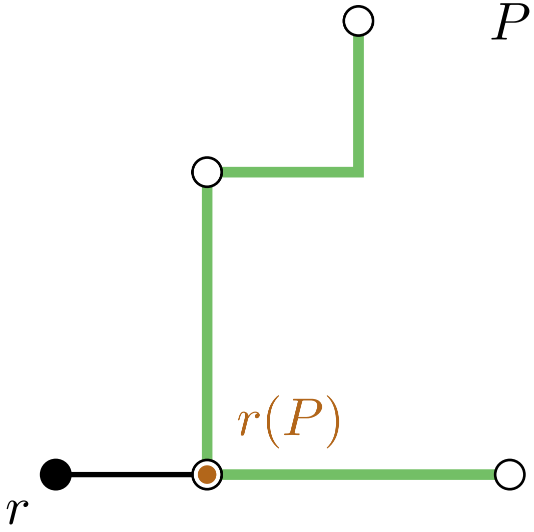
$$\mathrm{OPT}(P) = \mathrm{OPT}^*(P) + d(r, r(P))$$

$\mathrm{OPT}^*(X)$ – najkrótsza prostokątna gałąź Steinera dla X o korzeniu $r(X)$

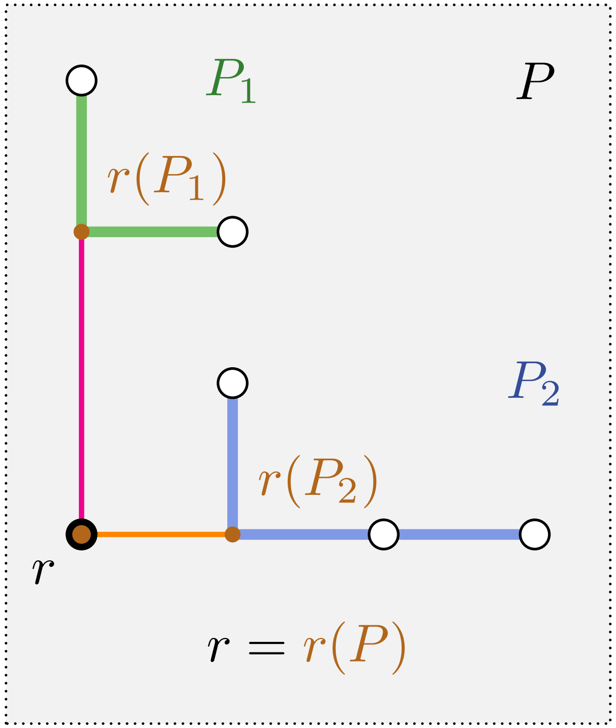
$d(r, r(X)) = (\mathrm{x}(r(X)) - \mathrm{x}(r)) + (\mathrm{y}(r(X)) - \mathrm{y}(r))$



$r \neq r(P)$



$r \neq r(P)$

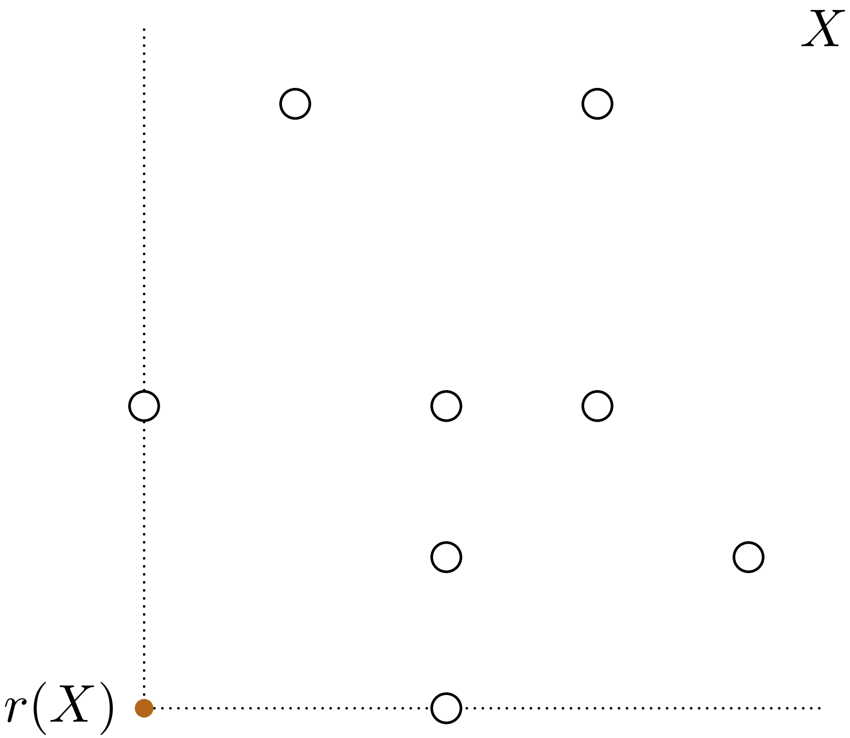


$r = r(P)$

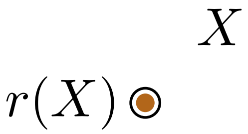
$$\mathrm{OPT}(P) = \mathrm{OPT}^*(P_1) + d(r, r(P_1)) + \mathrm{OPT}^*(P_2) + d(r, r(P_2))$$

$\mathrm{OPT}^*(X)$ – najkrótsza prostokątna gałąź Steinera dla X o korzeniu $r(X)$

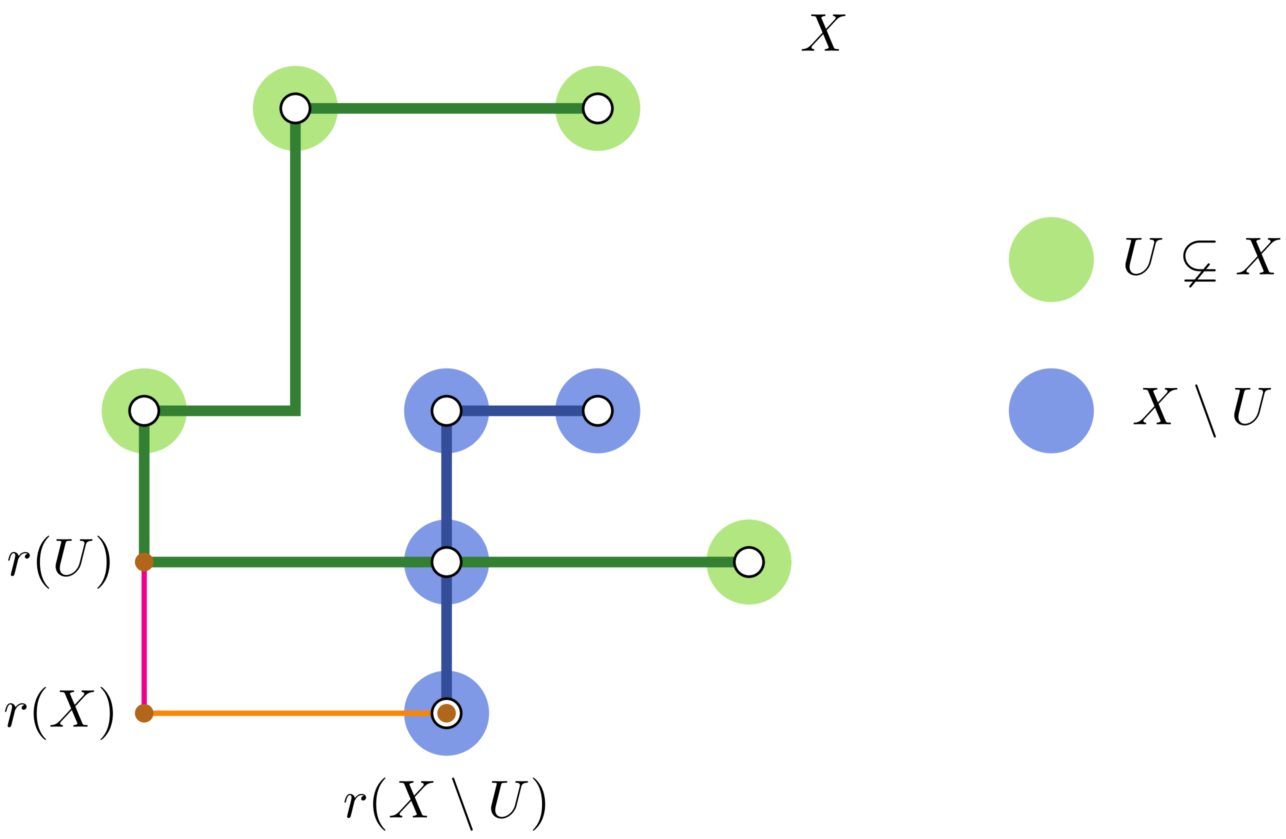
$d(r, r(X)) = (\mathrm{x}(r(X)) - \mathrm{x}(r)) + (\mathrm{y}(r(X)) - \mathrm{y}(r))$



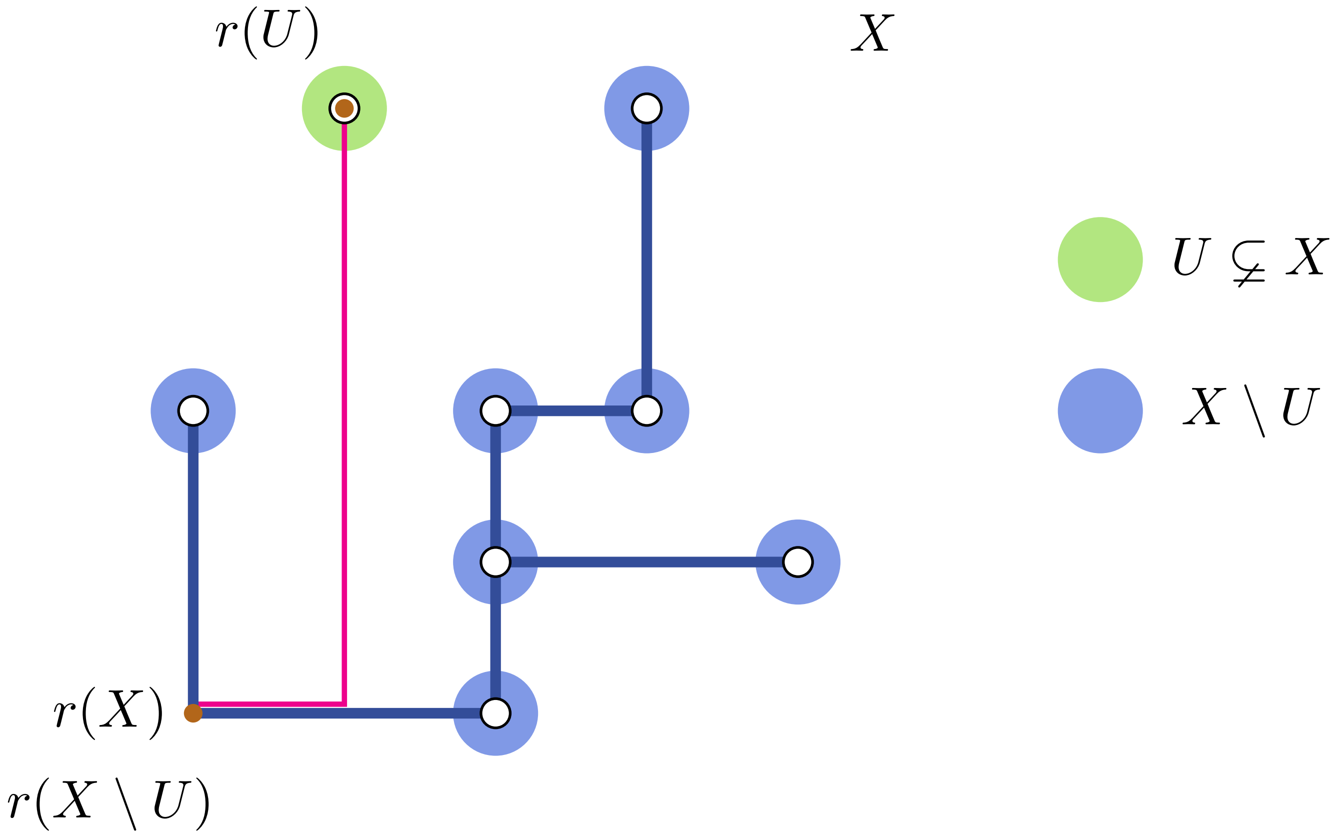
$$\mathrm{OPT}^*(X) = \begin{cases} 0 & \text{jeśli } |X| = 1; \\ \min_{U \subsetneq X} \mathrm{OPT}^*(U) + d(r(X), r(U)) & \\ \qquad\qquad + \mathrm{OPT}^*(X \setminus U) + d(r(X), r(X \setminus U)) & \text{w przeciwnym wypadku.} \end{cases}$$



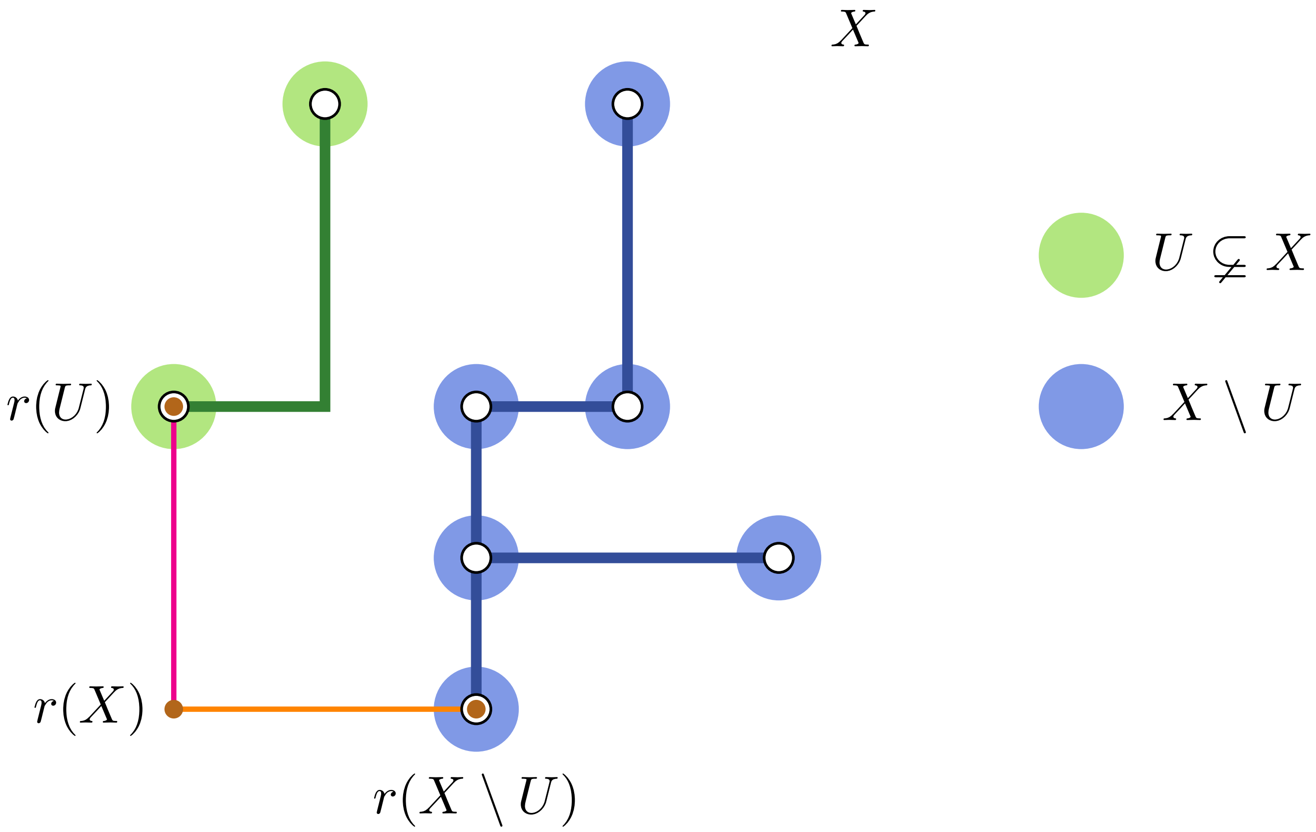
$$\mathrm{OPT}^*(X) = \begin{cases} 0 & \text{jeśli } |X| = 1; \\ \min_{U \subsetneq X} \mathrm{OPT}^*(U) + d(r(X), r(U)) & \\ \qquad + \mathrm{OPT}^*(X \setminus U) + d(r(X), r(X \setminus U)) & \text{w przeciwnym wypadku.} \end{cases}$$



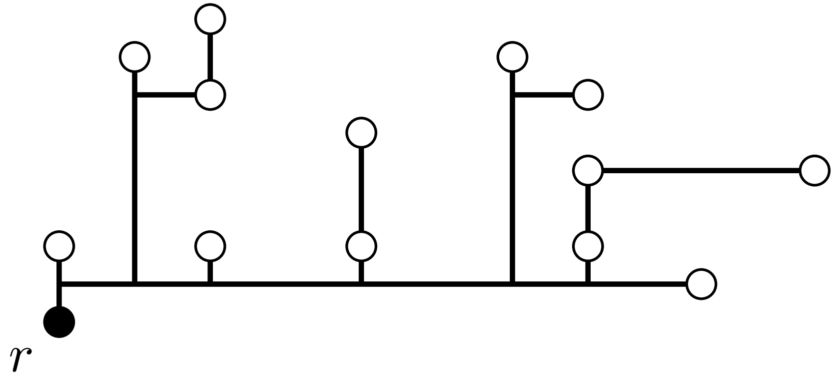
$$\mathrm{OPT}^*(X) = \begin{cases} 0 & \text{jeśli } |X| = 1; \\ \min_{U \subsetneq X} \mathrm{OPT}^*(U) + d(r(X), r(U)) & \\ \qquad + \mathrm{OPT}^*(X \setminus U) + d(r(X), r(X \setminus U)) & \text{w przeciwnym wypadku.} \end{cases}$$



$$\mathrm{OPT}^*(X) = \begin{cases} 0 & \text{jeśli } |X| = 1; \\ \min_{U \subsetneq X} \mathrm{OPT}^*(U) + d(r(X), r(U)) & \\ \qquad + \mathrm{OPT}^*(X \setminus U) + d(r(X), r(X \setminus U)) & \text{w przeciwnym wypadku.} \end{cases}$$



$$\mathrm{OPT}^*(X) = \begin{cases} 0 & \text{jeśli } |X| = 1; \\ \min_{U \subsetneq X} \mathrm{OPT}^*(U) + d(r(X), r(U)) & \\ \qquad + \mathrm{OPT}^*(X \setminus U) + d(r(X), r(X \setminus U)) & \text{w przeciwnym wypadku.} \end{cases}$$



Twierdzenie 8.1. (Ho, Ko, Ma, Sung 1992) *Problem najkrótszej gałęzi Steinera dla zbioru n punktów pierwszej ćwiartki płaszczyzny można rozwiązać w czasie* $O(3^n)$.

- ▶ Przetwarzanie wstępne (wyznaczenie $r(X), X \subseteq P$): $O(n \cdot 2^n) = O(3^n)$.
- ▶ $\sum_{i=1}^{n} \left[\binom{n}{i} \cdot 2^i \cdot O(1) \right] = O(3^n)$.

8.8 PROBLEM PLECAKOWY

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.4, PWN (2012)

Dany jest zbiór przedmiotów $X = \{a_1, \ldots, a_n\}$, gdzie przedmiot $a_i \in X$ ma *rozmiar* $s(a_i) \in \mathbb{Z}^+$ oraz *wartość* $v(a_i) \in \mathbb{Z}^+$.

Przedmiot a_i	Rozmiar $s(a_i)$	Wartość $v(a_i)$
1	6	5
2	3	1
3	4	4
4	2	2

Ograniczenie $S = 10$.

Problem. (Problem plecakowy)
Dla danej liczby $S \in \mathbb{Z}^+$, zwanej pojemnością plecaka*, wyznacz najbardziej wartościowy zbiór przedmiotów o łącznym rozmiarze nie przekraczającym S.*

8.8 PROBLEM PLECAKOWY

S. Dasgupta, Ch. Papadimitriou, U.V. Vazirani
Algorytmy, rozdział 6.4, PWN (2012)

Dany jest zbiór przedmiotów $X = \{a_1, \dots, a_n\}$, gdzie przedmiot $a_i \in X$ ma *rozmiar* $s(a_i) \in \mathbb{Z}^+$ oraz *wartość* $v(a_i) \in \mathbb{Z}^+$.

Przedmiot a_i	Rozmiar $s(a_i)$	Wartość $v(a_i)$
1	6	5
2	3	1
3	4	4
4	2	2

Ograniczenie $S = 10$.

Problem. (Problem plecakowy)
Dla danej liczby $S \in \mathbb{Z}^+$, zwanej pojemnością plecaka, *wyznacz najbardziej wartościowy zbiór przedmiotów o łącznym rozmiarze nie przekraczającym S.*

Zachłanne podejście

- Wybierając przedmioty w kolejności malejącego stosunku wartości do rozmiaru, możemy otrzymać rozwiązanie dowolnie „odległe” od optymalnego.

Przedmiot a_i	Rozmiar $s(a_i)$	Wartość $v(a_i)$	$v(a_i)/s(a_i)$
1	1	2	2
2	x	x	1

Ograniczenie $S = x$.

Idea rozwiązania #1 /największa możliwa wartość plecaka/

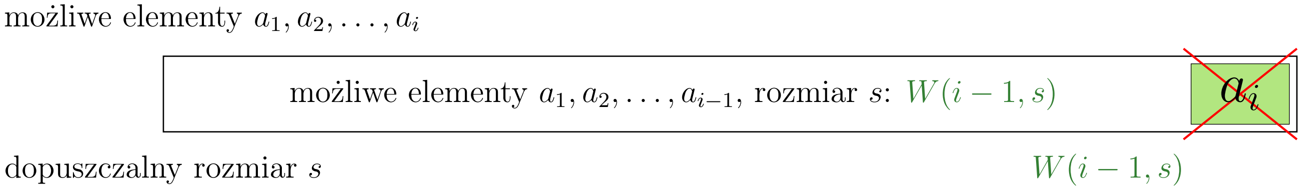
- Dla $i \in \{1, \ldots, n\}$ i $s \in \{1, \ldots, S\}$, niech $W(i, s)$ oznacza **największą wartość** osiągalną przez elementy ze zbioru $\{a_1, \ldots, a_i\}$, przy założeniu, że suma rozmiarów nie przekracza s.
- Wartość $W(1, s)$ można określić z definicji:

$$W(1, s) = \begin{cases} 0 & \text{dla } s < s(a_1); \\ v(a_1) & \text{w przeciwnym wypadku.} \end{cases}$$

Idea rozwiązania #1 /największa możliwa wartość plecaka/

- Dla $i \in \{1, \dots, n\}$ i $s \in \{1, \dots, S\}$, niech $W(i, s)$ oznacza **największą wartość** osiągalną przez elementy ze zbioru $\{a_1, \dots, a_i\}$, przy założeniu, że suma rozmiarów nie przekracza s.
- Wartość $W(i, s)$ można określić rekurencyjnie następująco:

↳ Jeśli a_i jest za duże $(s(a_i) > s)$, to nie możemy włożyć a_i: $W(i-1, s)$.

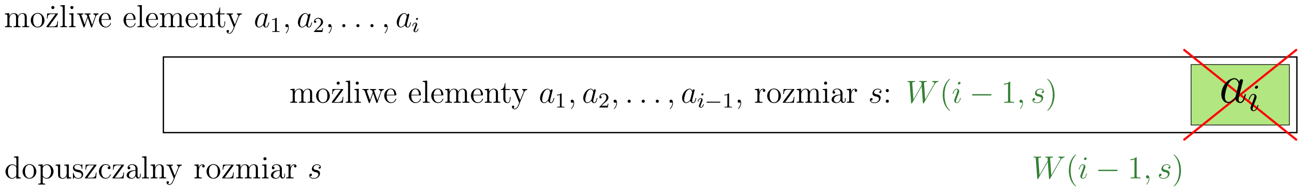


możliwe elementy $a_1, a_2, \dots, a_i$

możliwe elementy $a_1, a_2, \dots, a_{i-1}$, rozmiar s: $W(i-1, s)$ $\xcancel{a_i}$

dopuszczalny rozmiar s $W(i-1, s)$

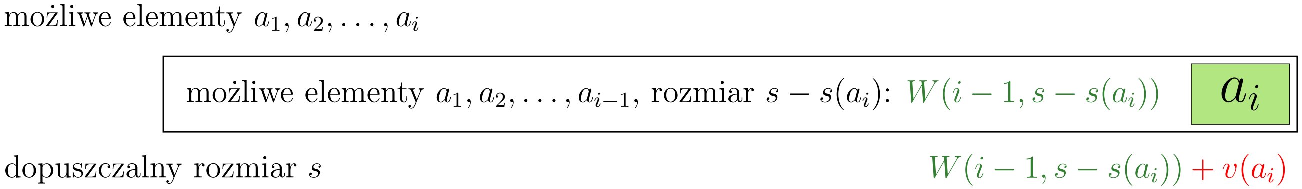
↳ Jeśli rozmiar $s(a_i) \leq s$, to wybieramy lepsze z rozwiązań: ‘z’ lub ‘bez’ a_i.



możliwe elementy $a_1, a_2, \dots, a_i$

możliwe elementy $a_1, a_2, \dots, a_{i-1}$, rozmiar s: $W(i-1, s)$ $\xcancel{a_i}$

dopuszczalny rozmiar s $W(i-1, s)$



możliwe elementy $a_1, a_2, \dots, a_i$

możliwe elementy $a_1, a_2, \dots, a_{i-1}$, rozmiar $s - s(a_i)$: $W(i-1, s - s(a_i))$ a_i

dopuszczalny rozmiar s $W(i-1, s - s(a_i)) + v(a_i)$

Idea rozwiązania #1 /największa możliwa wartość plecaka/

- Dla $i \in \{1, \ldots, n\}$ i $s \in \{1, \ldots, S\}$, niech $W(i, s)$ oznacza **największą wartość** osiągalną przez elementy ze zbioru $\{a_1, \ldots, a_i\}$, przy założeniu, że suma rozmiarów nie przekracza s.

- Wartość $W(1, s)$ można określić z definicji:

$$W(1, s) = \begin{cases} 0 & \text{dla } s < s(a_1); \\ v(a_1) & \text{w przeciwnym wypadku.} \end{cases}$$

- Wartość $W(i, s)$ można określić rekurencyjnie następująco:

$$W(i, s) = \begin{cases} W(i-1, s) & \text{jeśli } s < s(a_i); \\ \max\{W(i-1, s), W(i-1, s - s(a_i)) + v(a_i)\} & \\ & \text{w przeciwnym wypadku.} \end{cases}$$

- Rozwiązaniem jest zatem $W(n, S)$.

Przykład

a_i	$s(a_i)$	$v(a_i)$
1	6	5
2	3	1
3	4	4
4	2	2

$S = 10$

	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	5	5	5	5	5
2	0	0	0	1	1	1	5	5	5	6	6
3	0	0	0	1	4	4	5	5	5	6	9
4	0	0	2	2	4	4	6	6	7	7	$\underline{\mathbf{9}}$

$$\begin{aligned}
W(2,3) = [s(2) = 3 \leq 3] \; &= \\
&= \max\Big\{W(1,3), W(1,3-3)+1\Big\} \\
&= \max\Big\{W(1,3), W(1,0)+1\Big\} = \max\Big\{0, 0+1\Big\} = 1; \\
W(3,7) = [s(3) = 4 \leq 7] \; &= \\
&= \max\Big\{W(2,7), W(2,7-4)+4\Big\} \\
&= \max\Big\{W(2,7), W(2,3)+4\Big\} = \max\Big\{5, 1+4\Big\} = 5.
\end{aligned}$$

Rozwiązanie: $W(4,10)$=9.

Przykład

a_i	$s(a_i)$	$v(a_i)$
1	6	5
2	3	1
3	4	4
4	2	2

$S = 10$

	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	5	5	5	5	5
2	0	0	0	1	1	1	5	5	5	6	6
3	0	0	0	1	4	4	5	5	5	6	9
4	0	0	2	2	4	4	6	6	7	7	$\underline{\mathbf{9}}$

$$\begin{aligned} W(2,3) = [s(2) = 3 \leq 3] &= \\ &= \max\{W(1,3), W(1,3-3)+1\} \\ &= \max\{W(1,3), W(1,0)+1\} = \max\{0, 0+1\} = 1; \\ W(3,7) = [s(3) = 4 \leq 7] &= \\ &= \max\{W(2,7), W(2,7-4)+4\} \\ &= \max\{W(2,7), W(2,3)+4\} = \max\{5, 1+4\} = 5. \end{aligned}$$

Rozwiązanie: $W(4,10){=}9$.

- Algorytm ten ma złożoność rzędu $O(nS)$ i jest algorytmem *pseudowielomianowym*, tzn. jego czas działania zależy wielomianowo do rozmiaru wejścia, w którym wszystkie liczby – stanowiące część wejścia – zakodowane są unarnie, tj. do zapisu liczby k używamy k bitów, a nie $\lfloor \log_2 k \rfloor + 1$. W ogólnym przypadku problem plecakowy jest problemem NP-trudnym.

Idea rozwiązania #2 /najmniejszy rozmiar plecaka/

- Dla $1 \leq i \leq n$ oraz $0 \leq v \leq V = \sum_{i=1}^{n} v(a_i)$, niech $A(i, v)$ oznacza **najmniejszy rozmiar** tego spośród wszystkich podzbiorów zbioru $\{a_1, \ldots, a_i\}$, którego wartość elementów wynosi v; jeśli taki podzbiór nie istnieje, $A(i, v) := \infty$.

$$A(i, v) = \min\{ \textstyle\sum_{a \in X'} s(a) : \sum_{a \in X'} v(a) = v \text{ oraz } X' \subseteq \{a_1, \ldots, a_i\}\}$$

- Wartość $A(i, 0) = 0$ dla $i \geq 1$.
- Wartość $A(1, \cdot)$ można określić z definicji:

$$A(1, v) = \begin{cases} s(a_1) & \text{dla } v(a_1) = v; \\ \infty & \text{w przeciwnym wypadku.} \end{cases}$$

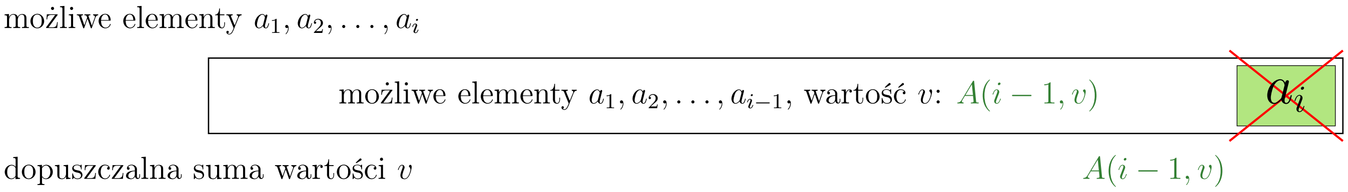
Idea rozwiązania #2 /najmniejszy rozmiar plecaka/

- Dla $1 \leq i \leq n$ oraz $0 \leq v \leq V = \sum_{i=1}^{n} v(a_i)$, niech $A(i, v)$ oznacza **najmniejszy rozmiar** tego spośród wszystkich podzbiorów zbioru $\{a_1, \ldots, a_i\}$, którego wartość elementów wynosi v; jeśli taki podzbiór nie istnieje, $A(i, v) := \infty$.

$$A(i, v) = \min\left\{ \sum_{a \in X'} s(a) : \sum_{a \in X'} v(a) = v \text{ oraz } X' \subseteq \{a_1, \ldots, a_i\}\right\}$$

- Wartość $A(i, v)$, $i \geq 2$, można określić rekurencyjnie następująco:

↳ Jeśli wartość $v(a_i)$ jest za duża, to nie możemy włożyć a_i: $A(i-1, v)$.

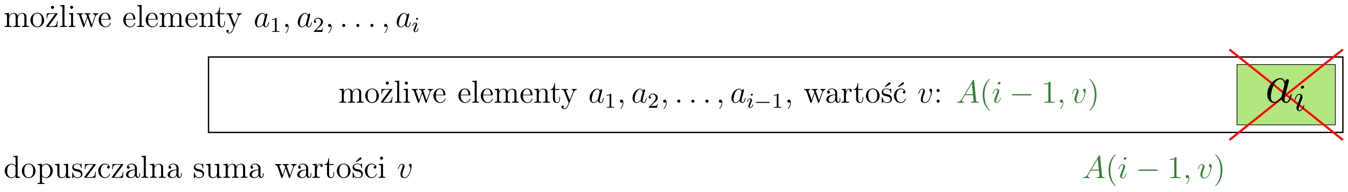


możliwe elementy $a_1, a_2, \ldots, a_i$

możliwe elementy $a_1, a_2, \ldots, a_{i-1}$, wartość v: $A(i-1, v)$ $\cancel{a_i}$

dopuszczalna suma wartości v $A(i-1, v)$

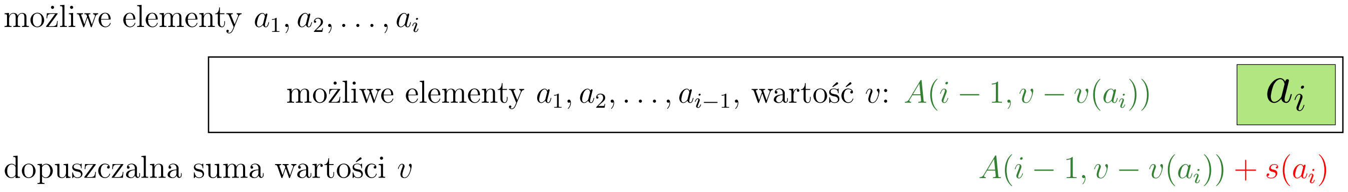
↳ Jeśli wartość $v(a_i) \leq v$, to wybieramy lepsze z rozwiązań: ‘z’ lub ‘bez’ a_i.



możliwe elementy $a_1, a_2, \ldots, a_i$

możliwe elementy $a_1, a_2, \ldots, a_{i-1}$, wartość v: $A(i-1, v)$ $\cancel{a_i}$

dopuszczalna suma wartości v $A(i-1, v)$



możliwe elementy $a_1, a_2, \ldots, a_i$

możliwe elementy $a_1, a_2, \ldots, a_{i-1}$, wartość v: $A(i-1, v-v(a_i))$ a_i

dopuszczalna suma wartości v $A(i-1, v-v(a_i)) + s(a_i)$

Idea rozwiązania #2 /najmniejszy rozmiar plecaka/

- Dla $1 \leq i \leq n$ oraz $0 \leq v \leq V = \sum_{i=1}^{n} v(a_i)$, niech $A(i,v)$ oznacza **najmniejszy rozmiar** tego spośród wszystkich podzbiorów zbioru $\{a_1,\ldots,a_i\}$, którego wartość elementów wynosi v; jeśli taki podzbiór nie istnieje, $A(i,v) := \infty$.

$$A(i,v) = \min\{\sum_{a\in X'} s(a) : \sum_{a\in X'} v(a) = v \text{ oraz } X' \subseteq \{a_1,\ldots,a_i\}\}$$

- Wartość $A(i,0) = 0$ dla $i \geq 1$.
- Wartość $A(1,\cdot)$ można określić z definicji:

$$A(1,v) = \begin{cases} s(a_1) & \text{dla } v(a_1) = v; \\ \infty & \text{w przeciwnym wypadku.} \end{cases}$$

- Wartość $A(i,v)$, $i \geq 2$, można określić rekurencyjnie następująco:

$$A(i,v) = \begin{cases} A(i-1,v) & \text{jeśli } v < v(a_i); \\ \min\{A(i-1,v), A(i-1,v-v(a_i)) + s(a_i)\} & \\ & \text{w przeciwnym wypadku.} \end{cases}$$

- Rozwiązaniem jest największe v takie, że $A(n,v) \leq S$.

Przykład

a_i	$s(a_i)$	$v(a_i)$
1	6	5
2	3	1
3	4	4
4	2	2

$S = 10$

	0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	∞	∞	∞	∞	6	∞	∞	∞	∞	∞	∞	∞
2	0	3	∞	∞	∞	6	9	∞	∞	∞	∞	∞	∞
3	0	3	∞	∞	4	6	9	∞	∞	10	13	∞	∞
4	0	3	2	5	4	6	6	8	11	$\underline{\mathbf{10}}$	13	12	15

$$\begin{aligned}
A(3,5) = [v(3) = 4 \le 5] &= \\
&= \min\Big\{A(2,5), A(2,5-4)+4\Big\} \\
&= \min\Big\{A(2,5), A(2,1)+4\Big\} \\
&= \min\Big\{6, 3+4\Big\} = 6; \\
A(4,6) = [v(4) = 2 \le 6] &= \\
&= \min\Big\{A(3,6), A(3,6-2)+2\Big\} \\
&= \min\Big\{A(3,6), A(3,4)+2\Big\} \\
&= \min\Big\{9, 4+2\Big\} = 6.
\end{aligned}$$

Rozwiązanie: 9, gdyż $A(4,9) = 10 = \max\{v : A(4,v) \le 10\}$.

- Algorytm ten ma złożoność $O(nV)$ i jest algorytmem pseudowielomianowym.

8.9 Definiowanie podproblemów przy programowaniu dynamicznym

1. Wejściem jest $\{x_1, x_2, \ldots, x_n\}$, a podproblemem $\{x_1, x_2, \ldots, x_i\}$.

$$\boxed{x_1\ x_2\ x_3\ x_4\ x_5\ x_6}\ x_7\ x_8\ x_9\ x_{10}$$

Liczba podproblemów jest wtedy rzędu $O(n)$.

2. Wejściem jest $\{x_1, x_2, \ldots, x_n\}$, a podproblemem $\{x_i, x_{i+1}, \ldots, x_j\}$. $/O(n^2)/$

$$x_1\ x_2\ \ \boxed{x_3\ x_4\ x_5\ x_6\ x_7}\ x_8\ x_9\ x_{10}$$

3. Wejściem jest $(\{x_1, \ldots, x_n\}, W)$, a podproblemem $(\{x_1, \ldots, x_i\}, W')$. $/O(nW)/$

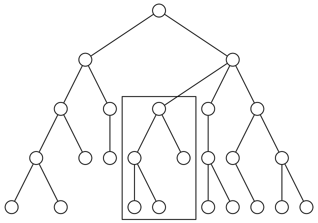
$$\boxed{x_1\ x_2\ x_3\ x_4\ x_5\ x_6}\ x_7\ x_8\ x_9\ x_{10} \quad \text{oraz} \quad W' \leq W, \quad \text{gdzie } W', W \in \mathbb{N}$$

4. Wejściem są $x_1, \ldots, x_n$ i $y_1, \ldots, y_m$, a podproblemami $x_1, \ldots, x_i$ i $y_1, \ldots, x_j$. $/O(nm)/$

$$\boxed{x_1\ x_2\ x_3\ x_4\ x_5\ x_6}\ x_7\ x_8\ x_9\ x_{10}$$

$$\boxed{y_1\ y_2\ y_3\ y_4}\ y_5\ y_6\ y_7\ x_8$$

5. Dane wejściowe są ukorzenionym drzewem; podproblemem jest poddrzewo. $/O(n)/$



8.10 KILKA UWAG O CZASIE I PAMIĘCI

1. Złożoność czasowa algorytmu opartego na programowaniu dynamicznym:

- ω(#liczba krawędzi w DAG podproblemów).
- Czas inicjalizacji pól tablicy, w której przechowujemy rozwiązania.

2. Często nie wszystkie wiersze odpowiedniej tablicy są wyliczane/zmieniane; czasami do wyznaczenia kolejnego wiersza/kolumny wystarczy znać wartości tylko poprzedniego wiersza/kolumny.

- Technika „spamiętywania”: tworzona jest tablica, która pamięta tylko wartości podproblemów do tej pory wyznaczanych (np. tablica z haszowaniem).
- Zamiast przechowywać całą tablicę, można przechowywać tylko dwa kolejne wiersze/kolumny.

9. SCHEMATY APROKSYMACYJNE

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Wprowadzenie do algorytmów, WNT (2004)

O.H. Ibarra, C.E. Kim
Fast approximation algorithms for the knapsack and sum of subset problems
Journal of the Association for Computing Machinery, 22(4), 463-468 (1975)

V.V. Vazirani
Algorytmy aproksymacyjne (Polish Edition: M. Mucha), WNT (2005)

Definicja. Niech Π będzie NP-trudnym problemem optymalizacyjnym z funkcją celu f. Mówimy, że algorytm A jest *schematem aproksymacyjnym* dla Π, jeżeli dla wejścia (I, ε) — gdzie I jest instancją dla Π (wejściem), a $\varepsilon > 0$ jest parametrem opisującym dopuszczalny błąd — algorytm ten zwraca rozwiązanie S takie, że:

- $f(I, S) \leq (1 + \varepsilon) \cdot \mathrm{OPT}$, jeśli Π jest problemem minimalizacji;
- $f(I, S) \geq (1 - \varepsilon) \cdot \mathrm{OPT}$, jeśli Π jest problemem maksymalizacji.

A zatem, o ile $\mathrm{P} \neq \mathrm{NP}$, to z teoretycznego punktu widzenia istnienie wielomianowego schematu aproksymacyjnego jest w wypadku NP-trudnych problemów optymalizacyjnych najlepszą możliwą sytuacją.

Definicja. Mówimy, że schemat aproksymacyjny A jest *wielomianowym schematem aproksymacyjnym*, w skrócie *PTAS*, jeżeli dla dowolnego ustalonego $\varepsilon > 0$ czas działania algorytmu A jest wielomianowy ze względu na rozmiar wejścia I. W przypadku, jeśli czas działania A jest wielomianowy zarówno ze względu na rozmiar wejścia I jak i $1/\varepsilon$, algorytm A nazywamy *w pełni wielomianowym schematem aproksymacyjnym*, w skrócie *FPTAS*. Skrót (F)PTAS pochodzi od angielskiego terminu (*fully*) *polynomial time approximation scheme*.

Np. schemat aproksymacyjny o czasie działania $O(n^{1/\varepsilon})$ jest PTAS, podczas gdy schemat o czasie $O(1/\varepsilon^n)$ już nie; analogicznie, PTAS z czasem działania $O(n^{1/\varepsilon})$ nie jest FPTAS, podczas gdy PTAS o czasie $O(n^2/\varepsilon^2)$ już jest.

- Problem plecakowy
- Problem sumy zbioru
- Najwiekszy zbiór niezależny w grafach UDG
- ...

9.1 PROBLEM PLECAKOWY

Dany jest zbiór przedmiotów $X = \{a_1, \ldots, a_n\}$, gdzie przedmiot $a_i \in X$ ma *rozmiar* $s(a_i) \in \mathbb{Z}^+$ oraz *wartość* $v(a_i) \in \mathbb{Z}^+$.

Przedmiot a_i	Rozmiar $s(a_i)$	Wartość $v(a_i)$
1	6	5
2	3	1
3	4	4
4	2	2

Ograniczenie $S = 10$.

Problem. (Problem plecakowy)
Dla danej liczby $S \in \mathbb{Z}^+$, zwanej pojemnością plecaka, *wyznacz najbardziej wartościowy zbiór przedmiotów o łącznym rozmiarze nie przekraczającym S.*

Zachłanne podejście

- Wybierając przedmioty w kolejności malejącego stosunku wartości do rozmiaru, możemy otrzymać rozwiązanie dowolnie „odległe” od optymalnego.

Przedmiot a_i	Rozmiar $s(a_i)$	Wartość $v(a_i)$	$v(a_i)/s(a_i)$
1	1	2	2
2	x	x	1

Ograniczenie $S = x$.

Programowanie dynamiczne /najmniejszy rozmiar plecaka/

- Dla $1 \leq i \leq n$ oraz $0 \leq v \leq V = \sum_{i=1}^{n} v(a_i)$, niech $A(i,v)$ oznacza **najmniejszy rozmiar** tego spośród wszystkich podzbiorów zbioru $\{a_1, \ldots, a_i\}$, którego wartość elementów wynosi v; jeśli taki podzbiór nie istnieje, $A(i,v) := \infty$.

$$A(i,v) = \min\{\, \textstyle\sum_{a \in X'} s(a) : \sum_{a \in X'} v(a) = v \text{ oraz } X' \subseteq \{a_1, \ldots, a_i\}\}$$

- Wartość $A(i,0) = 0$ dla $i \geq 1$.
- Wartość $A(1,\cdot)$ można określić z definicji:

$$A(1,v) = \begin{cases} s(a_1) & \text{dla } v(a_1) = v; \\ \infty & \text{w przeciwnym wypadku.} \end{cases}$$

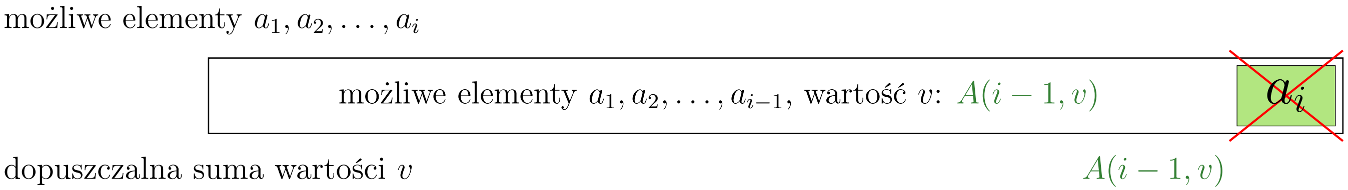
Programowanie dynamiczne /najmniejszy rozmiar plecaka/

- Dla $1 \leq i \leq n$ oraz $0 \leq v \leq V = \sum_{i=1}^{n} v(a_i)$, niech $A(i,v)$ oznacza **najmniejszy rozmiar** tego spośród wszystkich podzbiorów zbioru $\{a_1, \ldots, a_i\}$, którego wartość elementów wynosi v; jeśli taki podzbiór nie istnieje, $A(i,v) := \infty$.

$$A(i,v) = \min\{ \textstyle\sum_{a \in X'} s(a) : \sum_{a \in X'} v(a) = v \text{ oraz } X' \subseteq \{a_1, \ldots, a_i\}\}$$

- Wartość $A(i,v)$, $i \geq 2$, można określić rekurencyjnie następująco:

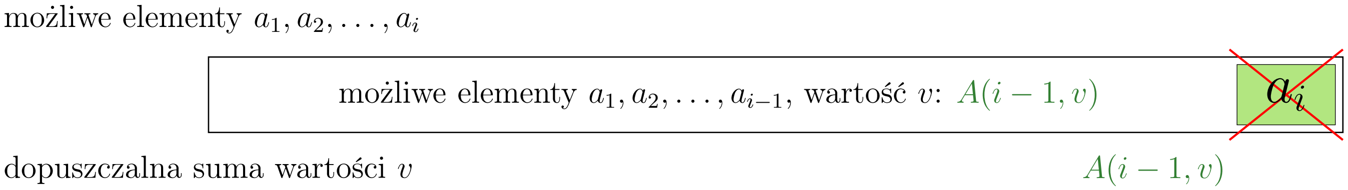
↳ Jeśli wartość $v(a_i)$ jest za duża, to nie możemy włożyć a_i: $A(i-1,v)$.


możliwe elementy $a_1, a_2, \ldots, a_i$

możliwe elementy $a_1, a_2, \ldots, a_{i-1}$, wartość v: $A(i-1,v)$ $\cancel{a_i}$

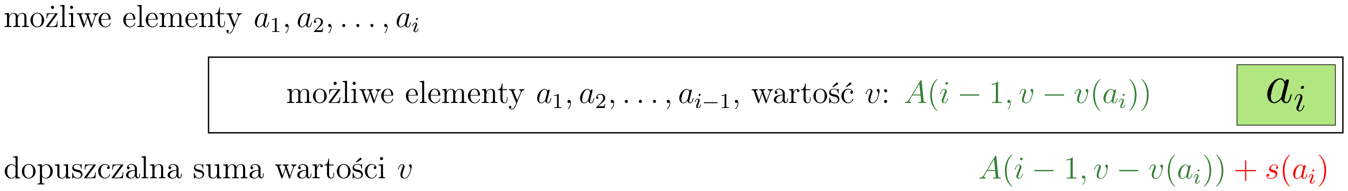
dopuszczalna suma wartości v $A(i-1,v)$

↳ Jeśli wartość $v(a_i) \leq v$, to wybieramy lepsze z rozwiązań: ‘z’ lub ‘bez’ a_i.


możliwe elementy $a_1, a_2, \ldots, a_i$

możliwe elementy $a_1, a_2, \ldots, a_{i-1}$, wartość v: $A(i-1,v)$ $\cancel{a_i}$

dopuszczalna suma wartości v $A(i-1,v)$


możliwe elementy $a_1, a_2, \ldots, a_i$

możliwe elementy $a_1, a_2, \ldots, a_{i-1}$, wartość v: $A(i-1, v - v(a_i))$ a_i

dopuszczalna suma wartości v $A(i-1, v - v(a_i)) + s(a_i)$

Programowanie dynamiczne /najmniejszy rozmiar plecaka/

- Dla $1 \leq i \leq n$ oraz $0 \leq v \leq V = \sum_{i=1}^{n} v(a_i)$, niech $A(i, v)$ oznacza **najmniejszy rozmiar** tego spośród wszystkich podzbiorów zbioru $\{a_1, \ldots, a_i\}$, którego wartość elementów wynosi v; jeśli taki podzbiór nie istnieje, $A(i, v) := \infty$.

$$A(i, v) = \min\{\textstyle\sum_{a \in X'} s(a) : \sum_{a \in X'} v(a) = v \text{ oraz } X' \subseteq \{a_1, \ldots, a_i\}\}$$

- Wartość $A(i, 0) = 0$ dla $i \geq 1$.
- Wartość $A(1, \cdot)$ można określić z definicji:

$$A(1, v) = \begin{cases} s(a_1) & \text{dla } v(a_1) = v; \\ \infty & \text{w przeciwnym wypadku.} \end{cases}$$

- Wartość $A(i, v)$, $i \geq 2$, można określić rekurencyjnie następująco:

$$A(i, v) = \begin{cases} A(i-1, v) & \text{jeśli } v < v(a_i); \\ \min\{A(i-1, v), A(i-1, v - v(a_i)) + s(a_i)\} & \\ & \text{w przeciwnym wypadku.} \end{cases}$$

- Rozwiązaniem jest największe v takie, że $A(n, v) \leq S$; złożoność $O(nV)$.

Algorytm KNAPSACKFPTAS(X, S, ε)

```
1. Dla danego $\varepsilon > 0$, niech $K = \frac{\varepsilon P}{n}$.
2. Dla każdego przedmiotu $a_i$, niech $v'(a_i) := \lfloor \frac{v(a_i)}{K} \rfloor$.
3. Korzystając z programowania dynamicznego, znajdź najwartościowszy zbiór $C$
   dla problemu plecakowego z wartościami $v'(a_i)$.
4. return $C$.
```

Idea rozwiązania

- Gdyby zyski odpowiadające przedmiotom były małe, tzn. ograniczone były przez wielomianową funkcję rozmiaru n, to algorytm oparty na programowaniu dynamicznym byłby algorytmem wielomianowym.
- Ignorujemy zatem końcowe bity liczb $v(a_i)$, a liczba ignorowanych bitów zależy od ε. Dzięki temu rozmiar nowych wartości jest wielomianowy ze względu na n i $1/\varepsilon$, a znalezione rozwiązanie — w czasie wielomianowym ze względu na n i $1/\varepsilon$ — jest o wartości co najmniej $(1 - \varepsilon) \cdot \mathrm{OPT}$.
- Zauważmy, że jeśli dla jakiegoś elementu $a \in X$ zachodzi $s(a) > S$, wówczas element ten nie może tworzyć rozwiązania optymalnego, a zatem możemy założyć, że dla każdego elementu $a \in X$ zachodzi $s(a) \leq S$, a w konsekwencji $\mathrm{OPT} \geq P = \max_{a \in X} v(a)$.

Lemat 9.1. *Zachodzi* $\text{zysk}(C) = \sum_{a \in C} v(a) \geq (1-\varepsilon) \cdot \text{OPT}$.

Dowód. Niech C_{OPT} będzie optymalnym zbiorem przedmiotów dla problemu plecakowego z wartościami $v(a_i)$. Zauważmy, że

$$K \geq v(a) - K \cdot \lfloor \frac{v(a)}{K} \rfloor \geq v(a) - K \cdot v'(a), \text{ a zatem } K \cdot v'(a) \geq v(a) - K.$$

W konsekwencji

$$K \cdot \sum_{a \in C_{OPT}} v'(a) \geq \sum_{a \in C_{\text{OPT}}} v(a) - |C_{\text{OPT}}| \cdot K \geq \sum_{a \in C_{\text{OPT}}} v(a) - nK$$

co implikuje, że

$$\begin{array}{rcll} \text{zysk}(C) = \sum_{a \in C} v(a) & \geq & K \cdot \sum_{a \in C} v'(a) & \geq K \cdot \sum_{a \in C_{\text{OPT}}} v'(a) \\ & \geq & \sum_{a \in C_{\text{OPT}}} v(a) - nK & = \text{OPT} - \varepsilon P \\ & \geq & (1-\varepsilon) \cdot \text{OPT}, & \end{array}$$

gdzie ostatnia nierówność wynika z tego, że $\text{OPT} \geq P$. □

Twierdzenie 9.2. [Ibarra, Kim 1975] *Algorytm* KnapsackFPTAS *jest w pełni wielomianowym schematem aproksymacyjnym dla problemu plecakowego.*

Dowód. Z lematu 9.1 wynika, że rozwiązanie znajdowane przez algorytm jest odpowiedniej wartości. Algorytm działa w czasie $O(n^2 \lfloor \frac{P}{K} \rfloor) = O(n^2 \lfloor \frac{n}{\varepsilon} \rfloor)$, czyli w czasie wielomianowym ze względu na n i $1/\varepsilon$. □

9.2 PROBLEM SUMY ZBIORU

Problem. *Dla danego zbioru* $S = \{x_1, \ldots, x_n\}$ *liczb naturalnych i całkowitej liczby* t *znaleźć podzbiór zbioru* S *o jak największej sumie nie przekraczającej* t.

- ▶ Problem sumy zbioru może być zredukowany do problemu pakowania plecaka.
  - ↳ Wartość $v(a_i)$ przedmiotu a_i oraz jego rozmiar $s(a_i)$ ustalone zostają jako x_i.
  - ↳ Ograniczeniem na rozmiar S plecaka jest t.
  - ↳ Optymalne rozwiązanie tak sformułowanego problemu plecakowego stanowi optymalne rozwiązanie problemu sumy zbioru
- ▶ Problem sumy zbioru jest problemem NP-trudnym.
  - ↳ NP-trudność sumy zbioru nie wynika z NP-trudności problemu plecakowego, gdyż powyższa redukcja redukuje problem sumy zbioru do problemu plecakowego, a nie na odwrót. A zatem to NP-trudność problemu plecakowego wynika z NP-trudności sumy zbioru.
- ▶ Na mocy powyżej redukcji możemy zastosować schemat aproksymacyjny dla problemu plecakowego.

9.3 PROBLEM SUMY ZBIORU [2]

Idea FPTAS oparta jest na konstruowaniu przybliżonych sum częściowych,

- ▶ Dla danej listy L oraz liczby x, lista $L \oplus x$ jest listą powstałą przez dodanie do każdego elementu listy L liczby x.
- ▶ $\textsc{Scal}(L, L')$ scala dwie posortowane listy w czasie $\Theta(|L|+|L'|)$ w posortowaną listę długości $|L| + |L'|$ i usuwa powtórzenia.

Algorytm EXACTSUBSETSUM

```
1. $L_0 := \{0\}$;
2. for $i := 1$ to $n$ do
       $L_i := \textsc{Scal}(L_{i-1}, L_{i-1} \oplus x_i)$;
       usuń z listy $L_i$ elementy większe od $t$;
3. return $\max L_n$.
```

Zauważmy, że rozmiar listy L_n może wynieść 2^n. Zatem czas działania algorytmu jest w ogólnym przypadku wykładniczy, chociaż w szczególnych przypadkach wielomianowy: np. gdy t lub wszystkie elementy zbioru S są wielomianowe ze względu na n. Poprawność algorytmu wynika bezpośrednio z konstrukcji, tj. łatwo zauważyć, że zachodzi $L_i \subseteq P_i$, gdzie P_i jest zbiorem wszystkich możliwych sum podzbiorów zbioru $\{x_1, \ldots, x_i\}$.

FPTAS dla problemu sumy zbioru opiera się na *skracaniu* listy przez ustalony parametr δ, tj. usuwaniu elementów z listy tak, aby w otrzymanej liście dla każdego usuniętego y istniało jego tzw. δ-*przybliżenie*.

Definicja. Dla dwóch dodatnich liczb rzeczywistych y i z mówimy, że z jest δ-*przybliżeniem* y, jeśli $(1-\delta)\cdot y \leq z \leq y$ (równoważne: $z \leq y$ oraz $\frac{y-z}{y} \leq \delta$).

Niech $L = \{y_1, \ldots, y_m\}$ będzie posortowaną listą, a δ zadanym parametrem. Poniższa procedura TRIM(L, δ) usuwa z listy L kolejne elementy, które mogą być δ-przybliżone przez pozostałe.

Procedura TRIM(L, δ)

```
1. $L_{\text{out}} := \{y_1\}$;
2. tmp $:= y_1$;
3. for $i := 2$ to $m$ do
       if (tmp $< (1-\delta)\cdot y_i$) then
           dołącz $y_i$ na koniec $L_{\text{out}}$;
           tmp $:= y_i$;
4. return $L_{\text{out}}$.
```

Rozważmy teraz zmodyfikowany algorytm ExactSubsetSum.

Algorytm SubsetSumFPTAS

Zakładamy, że zbiór $S = \{x_1, x_2, \dots, x_n\}$ jest posortowany.

```
1. $\delta := \frac{\varepsilon}{n}$; $L_0 := \{0\}$;
2. for $i := 1$ to $n$ do
        $E_i :=$ Scal($L_{i-1}, L_{i-1} \oplus x_i$);
        $L_i :=$ Trim($E_i, \delta$);
        usuń z listy $L_i$ elementy większe od $t$;
3. return max$L_n$.
```

Twierdzenie 9.3. [???] *Algorytm* SubsetSumFPTAS *jest w pełni wielomianowym schematem aproksymacyjnym dla problemu sumy zbioru.*

Dowód. Niech P_i będzie zbiorem wszystkich możliwych sum podzbiorów zbioru $\{x_1, \ldots, x_i\}$. Zauważmy, że z definicji listy L_i zachodzi $L_i \subseteq P_i$. Zatem wartość zwracana w kroku 3 jest sumą pewnego podzbioru zbioru S.

Pozostaje pokazać, że $(1-\varepsilon) \cdot \mathrm{OPT} \leq \max L_n$, tzn. że zwracane rozwiązanie jest dostatecznie bliskie optymalnego, oraz że czas działania jest wielomianowy ze względu na n, $\log t$ oraz $1/\varepsilon$.

- *Jeśli $y \in E_i$ oraz $y \leq t$, to istnieje $z \in L_i$ takie, że $(1-\delta) \cdot y \leq z \leq y$. Zatem dla każdego $y \in P_i$ takiego, że $y \leq t$, istnieje $z \in L_i$ takie, że*

$$(1-\delta)^i \cdot y \leq z \leq y.$$

  Niech zatem $\mathrm{OPT} \leq t$ będzie optymalnym rozwiązaniem problemu sumy zbioru. Z powyższej zależności otrzymujemy, że istnieje $z \in L_n$ takie, że

$$(1-\varepsilon/n)^n \cdot \mathrm{OPT} \leq z \leq \mathrm{OPT}.$$

  Ponieważ dla funkcji $f(x) = (1-\varepsilon/x)^x$ zachodzi $f'(x) > 0$, funkcja $(1-\varepsilon/n)^n$ rośnie wraz z n, a zatem dla $n \geq 1$ mamy $(1-\varepsilon) \leq (1-\varepsilon/n)^n$, a stąd

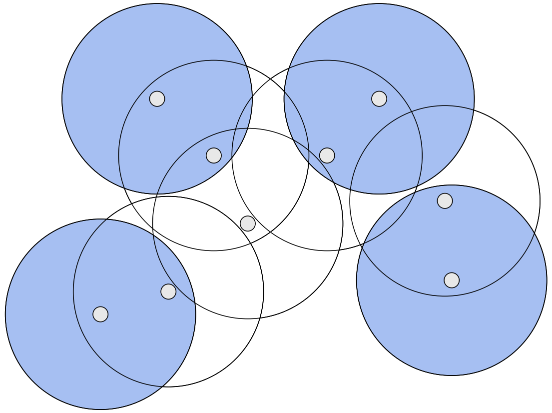
$$(1-\varepsilon) \cdot \mathrm{OPT} \leq z \leq \mathrm{OPT}.$$

- Złożoność czasowa jest rzędu $O(\frac{n^2 \log t}{\varepsilon})$, ponieważ zachodzi $|L_i| \leq \frac{n \ln t}{\varepsilon}$. □

9.4 NAJWIĘKSZY ZBIÓR NIEZALEŻNY W GRAFACH UD

Definicja. Mówimy, że graf $G = (V, E)$ jest *unit disk grafem*, ozn. *UDG*, wtedy i tylko wtedy, kiedy istnieje odwzorowanie $g\colon V \to \mathbb{R}^2$, zwane *reprezentacją geometryczną*, takie, że $\{u, v\} \in E \iff \mathrm{dist}_{\mathcal{E}}(g(u), g(v)) \leq 2$.

Znaczna część prac dotyczących problemów optymalizacyjnych w grafach UDG dotyczy przypadku, kiedy dana jest reprezentacja geometryczna. Jednakże sam problem wyznaczenia reprezentacji geometrycznej danego grafu UDG jest problemem NP-trudnym [Breu, Kirkpatrick 1998], a tym samym wariant problemu, w którym graf UDG zadany jest bez reprezentacji, różni się zasadniczo od tego, w którym jest ona dana.



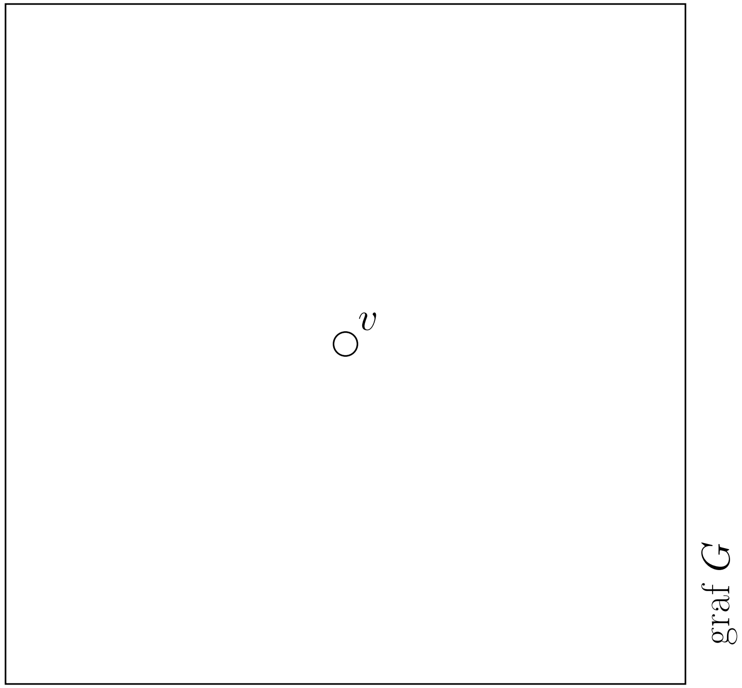
Przypomnijmy, że podzbiór $S \subseteq V$ grafu $G = (V, E)$ jest zbiorem *niezależnym* wtedy i tylko wtedy, gdy żadne dwa wierzchołki z S nie są sąsiednie w G.

Problem. *Dla danego grafu UDG wyznaczyć największy zbiór niezależny.*

9.4 NAJWIĘKSZY ZBIÓR NIEZALEŻNY W GRAFACH UD

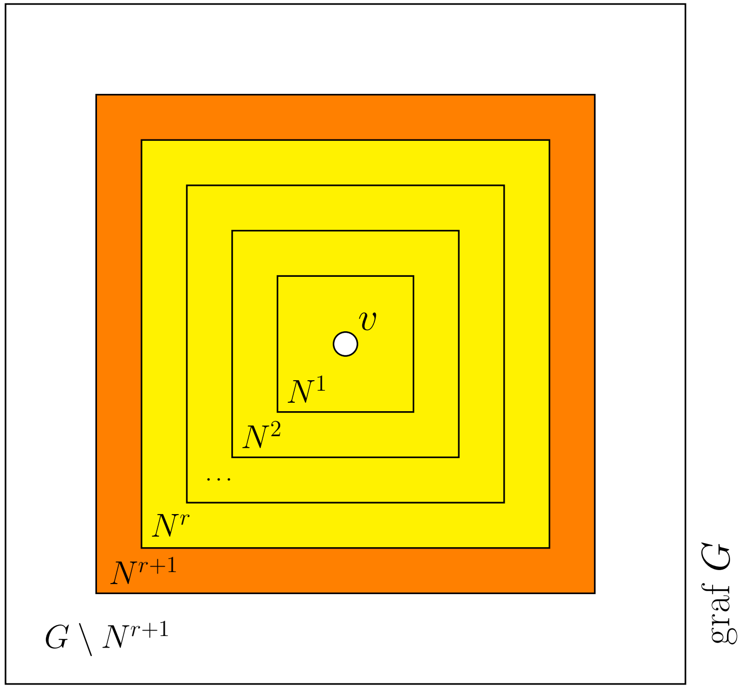
Twierdzenie 9.4. [Nieberg al. '03] *Istnieje PTAS rozwiązujący problem największego zbioru niezależnego w grafach UD* ***bez danej*** *reprezentacji na płaszczyźnie.*

- $N^r(v) = \{w \in V : \mathrm{dist}_G(v,u) \leq r\}$
- I_r – największy zbiór niezależny w $G[N^r(v)]$
- $|I_1| < (1-\varepsilon) \cdot |I_2|$
  $|I_2| < (1-\varepsilon) \cdot |I_3|$
  $\ldots$
- $|I_r| \geq (1-\varepsilon) \cdot |I_{r+1}|$, $r = \mathrm{const}(\varepsilon)$
- $I^* :=$ $(1-\varepsilon)$-przybliżenie OPT w $G[V \setminus N^{r+1}(v)]$
- Rozwiązanie: $I = I_r \cup I^*$



9.4 NAJWIĘKSZY ZBIÓR NIEZALEŻNY W GRAFACH UD

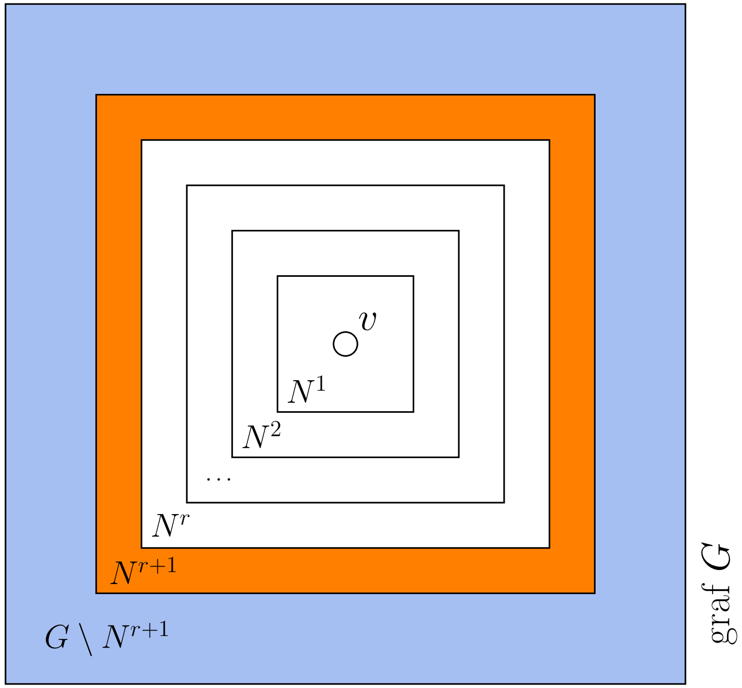
Twierdzenie 9.4. [Nieberg al. '03] *Istnieje PTAS rozwiązujący problem największego zbioru niezależnego w grafach UD* ***bez danej*** *reprezentacji na płaszczyźnie.*



- $N^r(v) = \{w \in V : \mathrm{dist}_G(v,u) \leq r\}$
- I_r – największy zbiór niezależny w $G[N^r(v)]$
- $|I_1| < (1-\varepsilon) \cdot |I_2|$
  $|I_2| < (1-\varepsilon) \cdot |I_3|$
  $\dots$
- $|I_r| \geq (1-\varepsilon) \cdot |I_{r+1}|$, $r = \mathrm{const}(\varepsilon)$
- $I^* :=$ $(1-\varepsilon)$-przybliżenie OPT w $G[V \setminus N^{r+1}(v)]$
- Rozwiązanie: $I = I_r \cup I^*$

9.4 NAJWIĘKSZY ZBIÓR NIEZALEŻNY W GRAFACH UD

Twierdzenie 9.4. [Nieberg al. '03] *Istnieje PTAS rozwiązujący problem największego zbioru niezależnego w grafach UD* ***bez danej*** *reprezentacji na płaszczyźnie.*



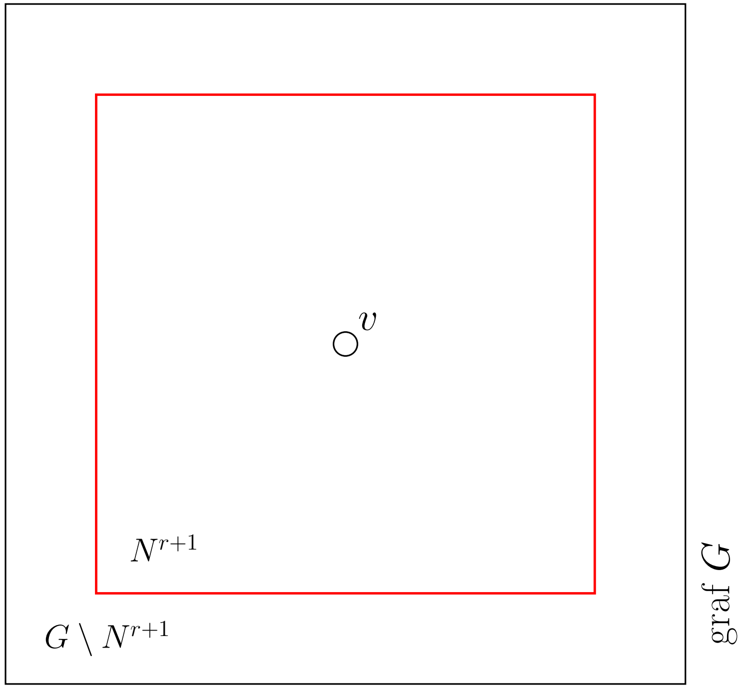
- $N^r(v) = \{w \in V : \mathrm{dist}_G(v,u) \leq r\}$
- I_r – największy zbiór niezależny w $G[N^r(v)]$
- $|I_1| < (1-\varepsilon) \cdot |I_2|$
  $|I_2| < (1-\varepsilon) \cdot |I_3|$
  $\dots$
- $|I_r| \geq (1-\varepsilon) \cdot |I_{r+1}|$, $r = \mathrm{const}(\varepsilon)$
- $I^* :=$ $(1-\varepsilon)$-przybliżenie OPT w $G[V \setminus N^{r+1}(v)]$
- Rozwiązanie: $I = I_r \cup I^*$

9.4 NAJWIĘKSZY ZBIÓR NIEZALEŻNY W GRAFACH UD

Twierdzenie 9.4. [Nieberg al. '03] *Istnieje PTAS rozwiązujący problem największego zbioru niezależnego w grafach UD* ***bez danej*** *reprezentacji na płaszczyźnie.*

$$\begin{aligned}\mathrm{OPT} &\leq \mathrm{OPT}(G[N^{r+1}]) + \mathrm{OPT}(G[V \setminus N^{r+1}]) \\ &\leq |I_{r+1}| + \tfrac{1}{1-\varepsilon} \cdot |I^*| \\ &\leq \tfrac{1}{1-\varepsilon} \cdot |I_r| + \tfrac{1}{1-\varepsilon} \cdot |I^*| \\ &= \tfrac{1}{1-\varepsilon} \cdot |I|,\end{aligned}$$

a zatem $(1-\varepsilon) \cdot \mathrm{OPT} \leq |I|$.



- $|I_r| \geq (1-\varepsilon) \cdot |I_{r+1}|$, $r = \mathrm{const}(\varepsilon)$
- $I^* :=$ $(1-\varepsilon)$-przybliżenie OPT w $G[V \setminus N^{r+1}(v)]$
- Rozwiązanie: $I = I_r \cup I^*$

10. ALGORYTMY RANDOMIZOWANE

Algorytmy randomizowane są to algorytmy, które podczas wykonywania kolejnych kroków podejmują losowe decyzje (korzystają z losowych liczb). Owa losowość może mieć wpływ zarówno na czas działania takich algorytmów jak i na poprawność zwracanego rozwiązania.

R. Motwani, P. Raghavan
Randomized Algorithms, rozdziały 1.1-1.2 oraz 10.2.1
Cambridge University Press (1995).

S. Har-Peled
Wykład „*CS 473g Algorithms*" (2006)
`http://sarielhp.org/teach/notes/algos/`

N. Alon, R. Yuster, U. Zwick
Color-coding, *Journal of the ACM* 42, 844-856 (1995)

10. ALGORYTMY RANDOMIZOWANE

Algorytmy randomizowane są to algorytmy, które podczas wykonywania kolejnych kroków podejmują losowe decyzje (korzystają z losowych liczb). Owa losowość może mieć wpływ zarówno na czas działania takich algorytmów jak i na poprawność zwracanego rozwiązania.

Definicja 10.1. Algorytm *Monte Carlo* jest to algorytm randomizowany, który może zwrócić niepoprawne/nieoptymalne rozwiązanie, ale poprzez wielokrotne wywoływanie algorytmu prawdopodobieństwo błędu może być sukcesywnie zmniejszane.

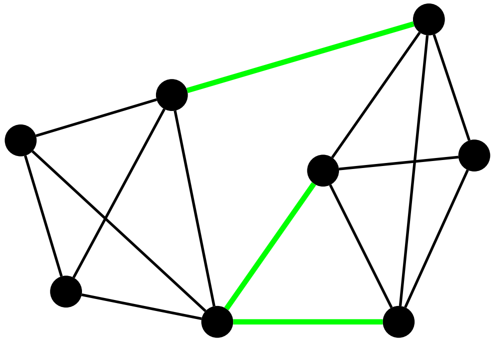
Przykłady: RANDOMMINCUT oraz RANDOMLONGPATH.

Definicja 10.2. Algorytm *Las Vegas* jest to algorytm randomizowany, który *zawsze* zwraca poprawne/optymalne rozwiązanie, natomiast jego czas działania może różnić się podczas poszczególnych wywołań.

Przykłady: QUICKSORT oraz RANDOMIZEDCLOSESTPAIR.

10.1 PROBLEM NAJMNIEJSZEGO ROZCIĘCIA

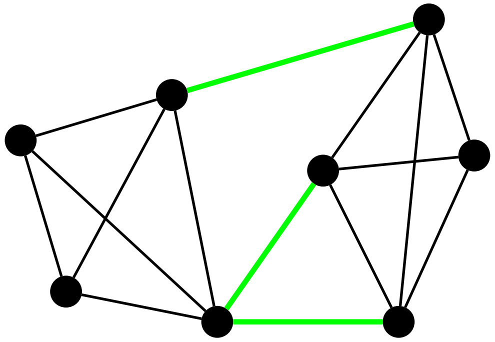
Niech $G = (V, E)$ będzie spójnym (multi)grafem o n wierzchołkach i m krawędziach. *Zbiorem rozspajającym* grafu G nazywamy zbiór krawędzi, których usunięcie powoduje, że powstały graf jest niespójny. *Rozcięcie* jest to zbiór rozspajający, którego żaden podzbiór właściwy nie jest zbiorem rozspajającym.



Najmniejsze rozcięcie.

10.1 PROBLEM NAJMNIEJSZEGO ROZCIĘCIA

Niech $G = (V, E)$ będzie spójnym (multi)grafem o n wierzchołkach i m krawędziach. *Zbiorem rozspajającym* grafu G nazywamy zbiór krawędzi, których usunięcie powoduje, że powstały graf jest niespójny. *Rozcięcie* jest to zbiór rozspajający, którego żaden podzbiór właściwy nie jest zbiorem rozspajającym.



Najmniejsze rozcięcie.

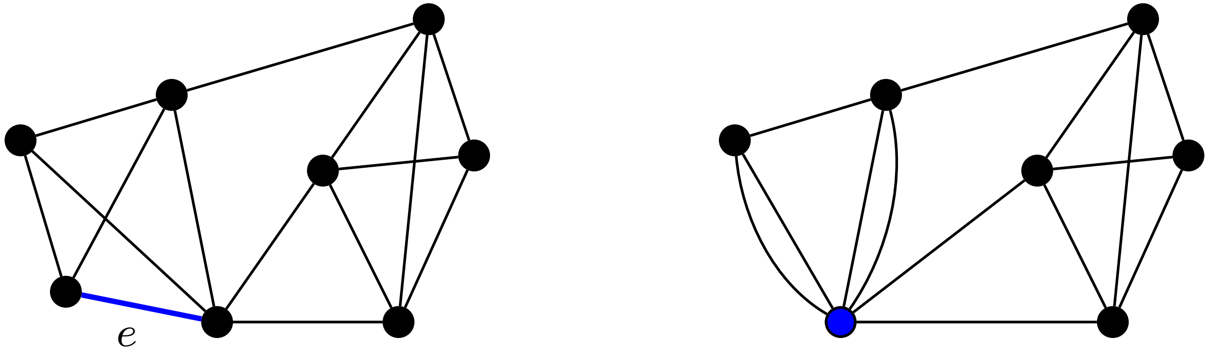
Problem.
Dla danego spójnego multigrafu $G = (V, E)$ znaleźć najmniejsze rozcięcie.

Zauważmy, że moc najmniejszego rozcięcia wynosi co najwyżej $\delta(G)$, tzn. jest nie większa niż minimalny stopień $\delta(G)$ grafu G.

Operacja ***ściągnięcia grafu*** $G = (V, E)$ ***wzdłuż krawędzi*** $e \in E$

Niech $e = \{x, y\}$ będzie krawędzią grafu G. Graf $G/\{e\} = (V', E')$ powstaje przez scalenie wierzchołków x i y w jeden wierzchołek v i usunięcie powstałych ewentualnie pętli. Formalnie:

$$V' = \big(V \setminus \{x, y\}\big) \cup \{v\};$$
$$E' = \big(E \setminus \big\{e' \mid e' \cap \{x, y\} \neq \emptyset\big\}\big) \cup \big\{\{v, v'\} \mid v' \in N(x) \cup N(y), v' \neq x, y\big\}.$$

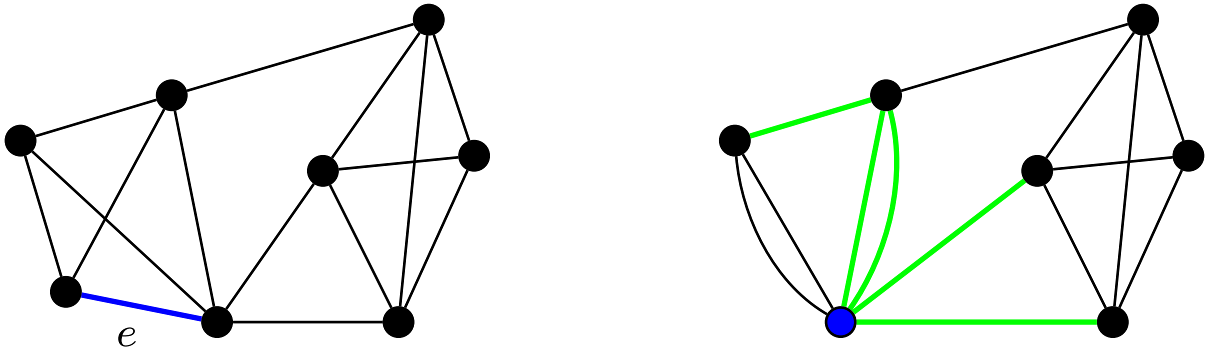


Operacja ściągnięcia wzdłuż krawędzi e.

Operacja ***ściągnięcia grafu*** $G = (V, E)$ ***wzdłuż krawędzi*** $e \in E$

Niech $e = \{x, y\}$ będzie krawędzią grafu G. Graf $G/\{e\} = (V', E')$ powstaje przez scalenie wierzchołków x i y w jeden wierzchołek v i usunięcie powstałych ewentualnie pętli. Formalnie:

$$V' = \big(V \setminus \{x, y\}\big) \cup \{v\};$$
$$E' = \big(E \setminus \{e' \mid e' \cap \{x, y\} \neq \emptyset\}\big) \cup \big\{\{v, v'\} \mid v' \in N(x) \cup N(y), v' \neq x, y\big\}.$$



Rozcięcie w grafie $G/\{e\}$ a rozcięcie w grafie G.

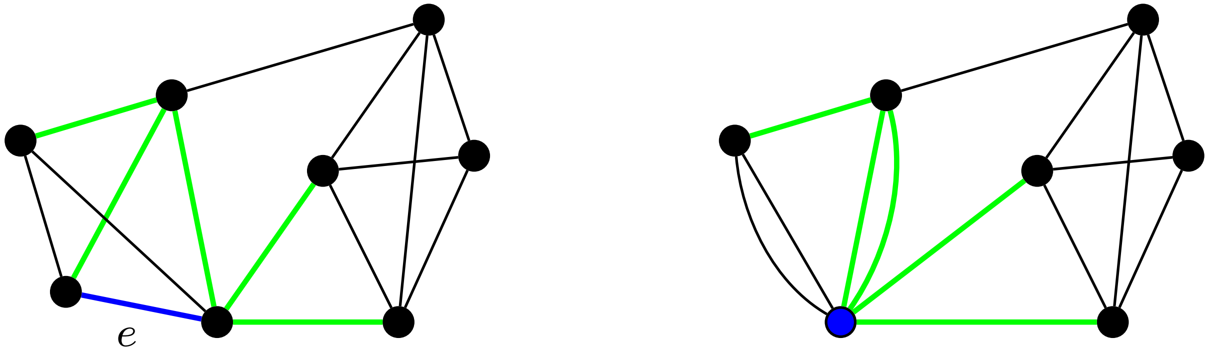
Zauważmy, że – z definicji grafu $G/\{e\}$ – dowolnemu rozcięciu w grafie $G/\{e\}$ o mocy c odpowiada rozcięcie tej samej mocy w grafie G.

Obserwacja 10.3. *Rozmiar najmniejszego rozcięcia w grafie G jest nie większy niż rozmiar najmniejszego rozcięcia w grafie $G/\{e\}$.*

Operacja ***ściągnięcia grafu*** $G = (V, E)$ ***wzdłuż krawędzi*** $e \in E$

Niech $e = \{x, y\}$ będzie krawędzią grafu G. Graf $G/\{e\} = (V', E')$ powstaje przez scalenie wierzchołków x i y w jeden wierzchołek v i usunięcie powstałych ewentualnie pętli. Formalnie:

$$V' = \big(V \setminus \{x, y\}\big) \cup \{v\};$$
$$E' = \big(E \setminus \{e' \mid e' \cap \{x, y\} \neq \emptyset\}\big) \cup \big\{\{v, v'\} \mid v' \in N(x) \cup N(y), v' \neq x, y\big\}.$$



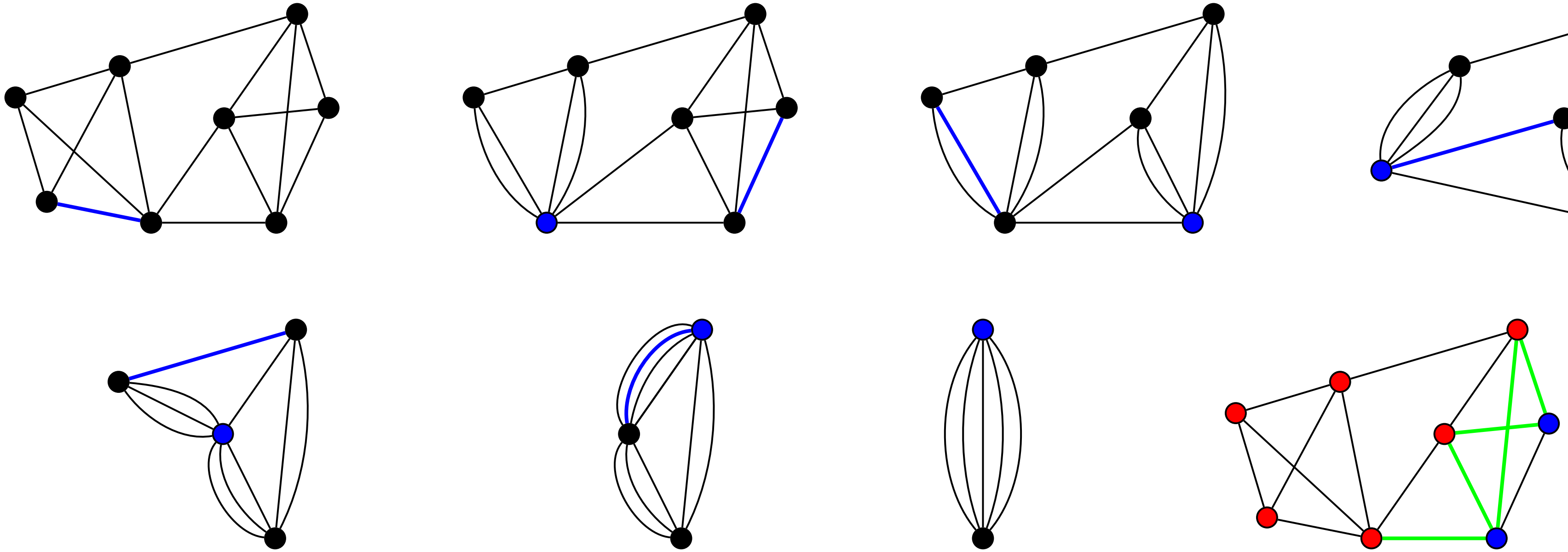
Rozcięcie w grafie $G/\{e\}$ a rozcięcie w grafie G.

Zauważmy, że – z definicji grafu $G/\{e\}$ – dowolnemu rozcięciu w grafie $G/\{e\}$ o mocy c odpowiada rozcięcie tej samej mocy w grafie G.

Obserwacja 10.4. *Rozmiar najmniejszego rozcięcia w grafie G jest nie większy niż rozmiar najmniejszego rozcięcia w grafie $G/\{e\}$.*

Idea algorytmu

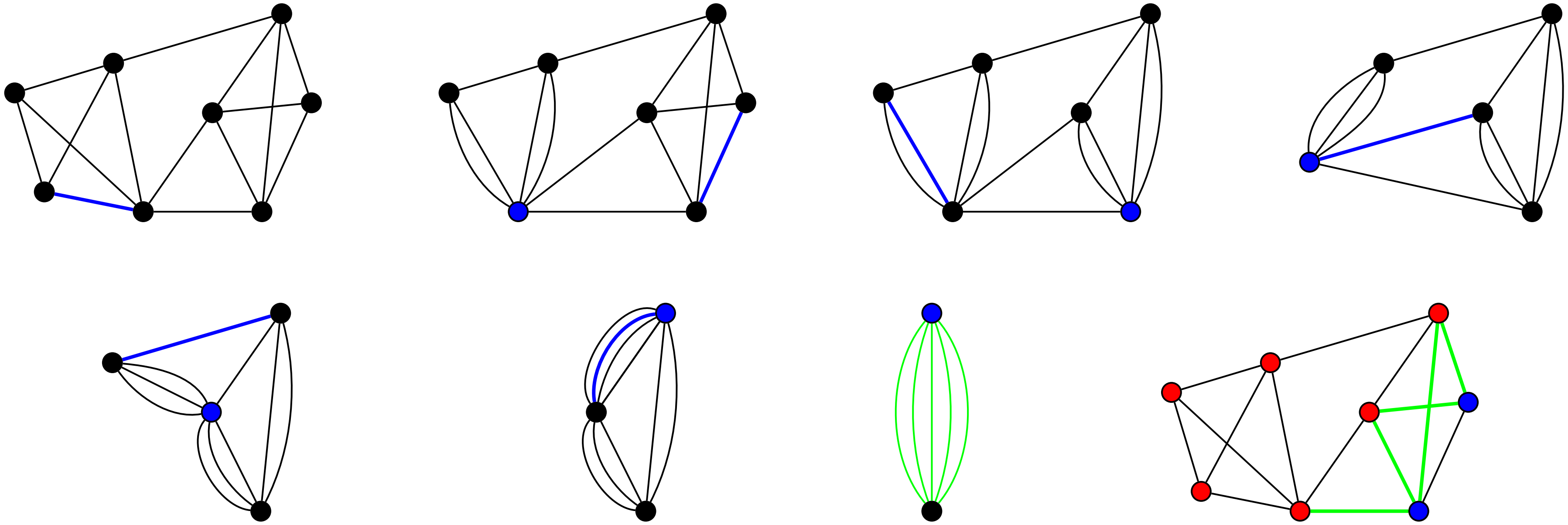
Główną ideą algorytmu jest powtarzanie losowych operacji ściągnięcia, otrzymując tym samym coraz mniejszy graf, dla którego to – na pewnym etapie – można już policzyć najmniejsze rozcięcie w sposób „naiwny”.



Kolejne operacje ściągnięcia wzdłuż krawędzi i otrzymane rozcięcie rozmiaru 5 w grafie G.

Idea algorytmu

Główną ideą algorytmu jest powtarzanie losowych operacji ściągnięcia, otrzymując tym samym coraz mniejszy graf, dla którego to – na pewnym etapie – można już policzyć najmniejsze rozcięcie w sposób „naiwny”.



Kolejne operacje ściągnięcia wzdłuż krawędzi i otrzymane rozcięcie rozmiaru 5 w grafie G.

Obserwacja 10.5. *Niech* $e_1, e_2, \ldots, e_{n-2} \in E$ *będą krawędziami, które nie należą do minimalnego rozcięcia grafu* $G = (V, E)$ *oraz takimi, że graf* $G' = G/\{e_1, \ldots, e_{n-2}\}$ *ma dwa wierzchołki. Wówczas (multi)krawędzie w* G *odpowiadające (multi)krawędziom w* G' *tworzą minimalne rozcięcie w* G*.*

Algorytm $\textsc{RandomMinCut}(G)$

```
1. $G_0 := G$.
2. $i := 0$;
3. while $G_i = (V_i, E_i)$ ma więcej niż dwa wierzchołki do
       wybierz losowo krawędź $e_i \in E_i$;
       $G_{i+1} := G_i/\{e_i\}$;
       $i := i + 1$;
4. Niech $C^*$ będzie rozcięciem w grafie $G$,
   które odpowiada krawędziom w grafie $G_{n-2}$.
5. return $C^*$.
```

Algorytm RandomMinCut(G)

```
1. $G_0 := G$.
2. $i := 0$;
3. while $G_i = (V_i, E_i)$ ma więcej niż dwa wierzchołki do
       wybierz losowo krawędź $e_i \in E_i$;
       $G_{i+1} := G_i/\{e_i\}$;
       $i := i + 1$;
4. Niech $C^*$ będzie rozcięciem w grafie $G$,
   które odpowiada krawędziom w grafie $G_{n-2}$.
5. return $C^*$.
```

- Jako że operację ściągnięcia można wykonać w czasie $O(n)$, złożoność algorytmu RandomMinCut jest rzędu $O(n^2)$.

- Z definicji operacji ściągnięcia wynika, że zwróci on poprawne rozwiązanie, tj. rozcięcie w wejściowym grafie G.

Lemat 10.6. *Jeśli w n-wierzchołkowym grafie $G = (V, E)$ najmniejsze rozcięcie jest rozmiaru k, to $|E| \geq \frac{kn}{2}$.*

Dowód. Zauważmy, że w grafie G stopień każdego wierzchołka wynosi przynajmniej k – w przeciwnym wypadku krawędzie incydentne do wierzchołka o stopniu mniejszym niż k tworzyłyby rozcięcie rozmiaru $< k$ i otrzymalibyśmy sprzeczność z definicją k jako rozmiaru najmniejszego rozcięcia. W konsekwencji w grafie G mamy przynajmniej $\frac{\sum_{v \in V} \deg(v)}{2} \geq \frac{nk}{2}$ krawędzi. □

Lemat 10.7. *Prawdopodobieństwo, że losowo wybrana krawędź $e \in E$ należy do najmniejszego (ustalonego) rozcięcia C w grafie $G = (V, E)$, jest nie większe niż $\frac{2}{n}$.*

Dowód. Z lematu 10.6 wynika, że graf G ma przynajmniej $kn/2$ krawędzi. Dokładnie k z nich należy do rozcięcia C, a zatem prawdopodobieństwo wyboru jednej z nich jest nie większe niż $\frac{k}{kn/2} = \frac{2}{n}$. □

Lemat 10.8. *Algorytm* RANDOMMINCUT *zwraca najmniejsze rozcięcie z prawdopodobieństwem* $\geq \frac{2}{n(n-1)}$.

Dowód. Niech C będzie (ustalonym) najmniejszym rozcięciem w grafie G. Niech $\mathcal{E}_i$ będzie zdarzeniem polegającym na *niewybraniu* krawędzi należącej do rozcięcia C w i-tej iteracji algorytmu. Z obserwacji 10.5 wynika, że RANDOMMINCUT zwróci rozcięcie C, jeśli zajdą wszystkie zdarzenia $\mathcal{E}_1, \ldots, \mathcal{E}_{n-2}$. Z lematu 10.6 wynika, że prawdopodobieństwo zajścia zdarzenia $\mathcal{E}_1$ wynosi co najmniej $1 - \frac{2}{n}$. Analogicznie, zakładając, że zaszło zdarzenie $\mathcal{E}_1$, prawdopodobieństwo zajścia zdarzenia $\mathcal{E}_2$ wynosi przynajmniej $1 - \frac{2}{n-1}$, gdyż graf G_1 ma przynajmniej $k(n-1)/2$ krawędzi (patrz dowód lematu 10.6), tzn.:

$$\mathrm{Prob}(\mathcal{E}_2 | \mathcal{E}_1) \geq 1 - \frac{2}{n-1}.$$

Idąc dalej, w i-tej iteracji graf G_{i-1} ma $n-i+1$ wierzchołków i – pod warunkiem zajścia zdarzenia $\cap_{j=1}^{i-1} \mathcal{E}_j$ – rozmiar najmniejszego rozcięcia nadal wynosi k, a tym samym liczba krawędzi w grafie G_{i-1} wynosi przynajmniej

$$|E_{i-1}| \geq \frac{k(n-i+1)}{2}.$$

W konsekwencji:

$$\mathrm{Prob}(\mathcal{E}_i | \cap_{j=1}^{i-1} \mathcal{E}_j) \geq 1 - \frac{2}{n-i+1}. \quad (\ldots)$$

Lemat 10.8. *Algorytm* RANDOMMINCUT *zwraca najmniejsze rozcięcie z prawdopodobieństwem* $\geq \frac{2}{n(n-1)}$.

Dowód. (...)

Korzystając ze wzoru na prawdopodobieństwo warunkowe otrzymujemy:

$$\begin{aligned}\mathrm{Prob}(\cap_{i=1}^{n-2}\mathcal{E}_j) \ &= \ \mathrm{Prob}(\mathcal{E}_1)\cdot\mathrm{Prob}(\mathcal{E}_2|\mathcal{E}_1)\cdot\mathrm{Prob}(\mathcal{E}_3|\mathcal{E}_1\cap\mathcal{E}_2)\cdot\ldots\\ &\ldots\cdot\mathrm{Prob}(\mathcal{E}_n|\mathcal{E}_1\cap\ldots\cap\mathcal{E}_{n-1}) \geq \prod_{i=1}^{n-2}(1-\tfrac{2}{n-i+1}) = \tfrac{2}{n(n-1)}.\end{aligned}$$ □

Lemat 10.8. *Algorytm* RANDOMMINCUT *zwraca najmniejsze rozcięcie z prawdopodobieństwem* $\geq \frac{2}{n(n-1)}$.

Dowód. (...)

Korzystając ze wzoru na prawdopodobieństwo warunkowe otrzymujemy:

$$\begin{aligned} \mathrm{Prob}(\cap_{i=1}^{n-2}\mathcal{E}_j) &= \mathrm{Prob}(\mathcal{E}_1) \cdot \mathrm{Prob}(\mathcal{E}_2|\mathcal{E}_1) \cdot \mathrm{Prob}(\mathcal{E}_3|\mathcal{E}_1 \cap \mathcal{E}_2) \cdot ... \\ & ... \cdot \mathrm{Prob}(\mathcal{E}_n|\mathcal{E}_1 \cap ... \cap \mathcal{E}_{n-1}) \geq \prod_{i=1}^{n-2}(1 - \tfrac{2}{n-i+1}) = \tfrac{2}{n(n-1)}. \quad \square \end{aligned}$$

Otrzymujemy w konsekwencji następujące twierdzenie.

Twierdzenie 10.9. [Karger 1993] RANDOMMINCUT *w czasie* $O(|V|^2)$ *znajduje minimalne rozcięcie w grafie* $G = (V, E)$ *z prawdopodobieństwem rzędu* $\Omega(\frac{1}{|V|^2})$.

Lemat 10.8. *Algorytm* RANDOMMINCUT *zwraca najmniejsze rozcięcie z prawdopodobieństwem* $\geq \frac{2}{n(n-1)}$.

Dowód. (...)

Korzystając ze wzoru na prawdopodobieństwo warunkowe otrzymujemy:

$$\begin{aligned}\mathrm{Prob}(\cap_{i=1}^{n-2}\mathcal{E}_j) &= \mathrm{Prob}(\mathcal{E}_1) \cdot \mathrm{Prob}(\mathcal{E}_2|\mathcal{E}_1) \cdot \mathrm{Prob}(\mathcal{E}_3|\mathcal{E}_1 \cap \mathcal{E}_2) \cdot ... \\ & ... \cdot \mathrm{Prob}(\mathcal{E}_n|\mathcal{E}_1 \cap ... \cap \mathcal{E}_{n-1}) \geq \prod_{i=1}^{n-2}(1 - \tfrac{2}{n-i+1}) = \tfrac{2}{n(n-1)}. \quad \square\end{aligned}$$

Otrzymujemy w konsekwencji następujące twierdzenie.

Twierdzenie 10.9. [Karger 1993] RANDOMMINCUT *w czasie* $O(|V|^2)$ *znajduje minimalne rozcięcie w grafie* $G = (V, E)$ *z prawdopodobieństwem rzędu* $\Omega(\frac{1}{|V|^2})$.

Modyfikacja algorytmu

Obserwacja: prawdopodobieństwo sukcesu w pierwszych t iteracjach RANDOMMINCUT maleje wraz ze zmniejszaniem się grafu. Modyfikacja: poszczególne wywołania algorytmu są tym częstsze, im mniej wierzchołków ma wejściowy graf.

Twierdzenie 10.10. [Karger, Stein 1996]
Istnieje randomizowany algorytm o czasie działania $O(|V|^2 \log |V|)$, *który znajduje minimalne rozcięcie w grafie* $G = (V, E)$ *z prawdopodobieństwem rzędu* $\Omega(\frac{1}{\log |V|})$.

Poprawność z dużym prawdopodobieństwem

Definicja 10.11. *Mówimy, że algorytm zwraca prawidłowe rozwiązanie* z dużym prawdopodobieństwem, *jeśli prawdopodobieństwo zwrócenia* *błędnego* *rozwiązania jest nie większe niż* $\frac{1}{n^r}$ *dla pewnego* $r \in \mathbb{R}^+$.

Problem rozcięcia raz jeszcze

Załóżmy teraz, że dla danego grafu $G = (V, E)$ wywołamy algorytm RANDOM-MINCUT N razy (*amplifikacja*), za każdym razem dokonując niezależnych losowań krawędzi. Z niezależności każdej z prób wynika, że prawdopodobieństwo, iż każda z nich zwróci błędne rozwiązanie, wynosi co najwyżej

$$\left(1 - \frac{2}{n(n-1)}\right)^N,$$

gdzie $n = |V|$, a tym samym prawdopodobieństwo sukcesu wynosi przynajmniej

$$1 - \left(1 - \frac{2}{n(n-1)}\right)^N \geq 1 - e^{-2N/n(n-1)}.$$

(Korzystamy z nierówności $1 + x \leq e^x$ dla $x \in \mathbb{R}$.)

Poprawność z dużym prawdopodobieństwem

Definicja 10.11. *Mówimy, że algorytm zwraca prawidłowe rozwiązanie* z dużym prawdopodobieństwem, *jeśli prawdopodobieństwo zwrócenia błędnego rozwiązania jest nie większe niż $\frac{1}{n^r}$ dla pewnego $r \in \mathbb{R}^+$.*

Problem rozcięcia raz jeszcze

(...) W szczególności, jeśli $N = \binom{n}{2} \cdot \lceil \ln n \rceil$, wówczas prawdopodobieństwo sukcesu naszego algorytmu w ciągu N niezależnych prób wynosi przynajmniej

$$1 - e^{-2N/n(n-1)} = 1 - e^{-\lceil \ln n \rceil} \geq 1 - e^{-\ln n} = 1 - \frac{1}{n},$$

a tym samym prawdopodobieństwo porażki jest mniejsze od $\frac{1}{n}$.

Twierdzenie 10.12. [Karger 1993]
Istnieje randomizowany algorytm o czasie działania $O(|V|^4 \log |V|)$ znajdujący z dużym prawdopodobieństwem najmniejsze rozcięcie w grafie $G = (V, E)$.

Poprawność z dużym prawdopodobieństwem

Definicja 10.11. *Mówimy, że algorytm zwraca prawidłowe rozwiązanie z dużym prawdopodobieństwem, jeśli prawdopodobieństwo zwrócenia błędnego rozwiązania jest nie większe niż* $\frac{1}{n^r}$ *dla pewnego* $r \in \mathbb{R}^+$.

Problem rozcięcia raz jeszcze

(...) W szczególności, jeśli $N = \binom{n}{2} \cdot \lceil \ln n \rceil$, wówczas prawdopodobieństwo sukcesu naszego algorytmu w ciągu N niezależnych prób wynosi przynajmniej

$$1 - e^{-2N/n(n-1)} = 1 - e^{-\lceil \ln n \rceil} \geq 1 - e^{-\ln n} = 1 - \frac{1}{n},$$

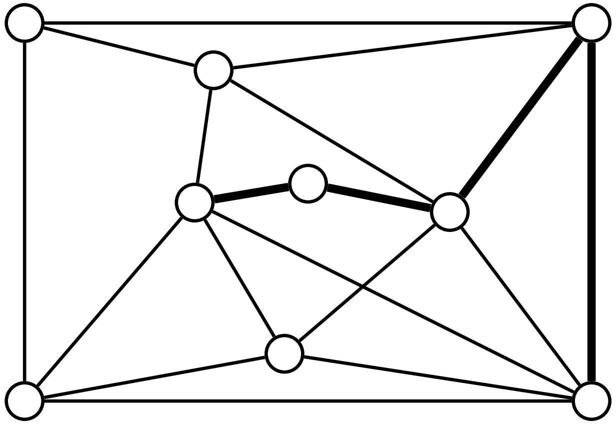
a tym samym prawdopodobieństwo porażki jest mniejsze od $\frac{1}{n}$.

Twierdzenie 10.12. [Karger 1993]
Istnieje randomizowany algorytm o czasie działania $O(|V|^4 \log |V|)$ *znajdujący z dużym prawdopodobieństwem najmniejsze rozcięcie w grafie* $G = (V, E)$.

Twierdzenie 10.13. [Karger, Stein 1996]
Istnieje randomizowany algorytm o czasie działanie $O(|V|^2 \log^3 |V|)$ *znajdujący z dużym prawdopodobieństwem najmniejsze rozcięcie w grafie* $G = (V, E)$.

10.2 PROBLEM DŁUGIEJ ŚCIEŻKI

Problem. *Dla danego n-wierzchołkowego grafu* $G = (V, E)$ *i liczby całkowitej* k *znajdź w grafie* G *ścieżkę długości* k *(o ile taka istnieje).*

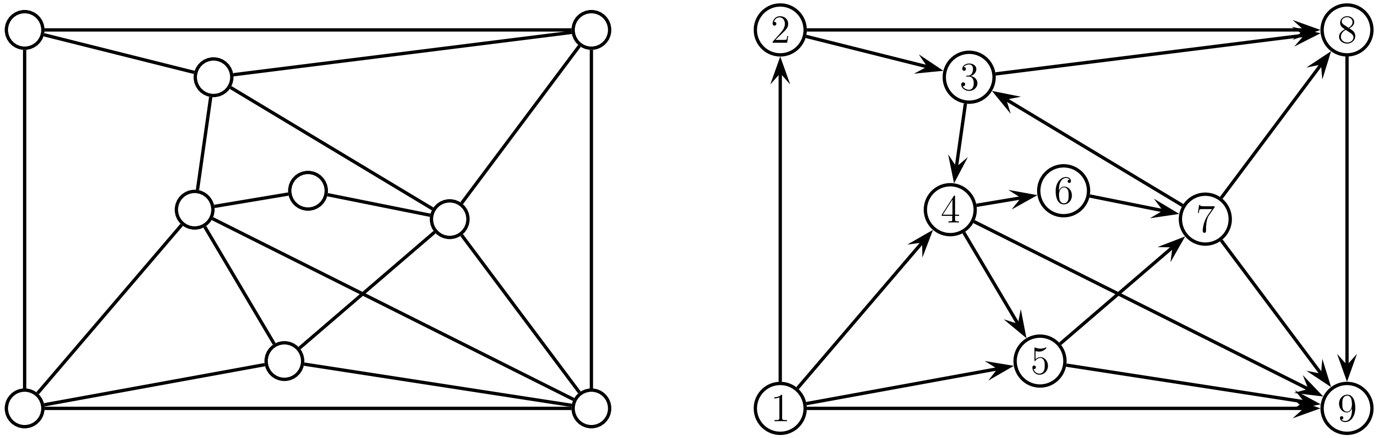


Ścieżka długości $k = 4$.

Dla $k = |V| - 1$ problem ten jest równoważny problemowi znalezienia ścieżki Hamiltona w grafie G, który jest problemem NP-zupełnym, a zatem problem długiej ścieżki jest przynajmniej tak trudny jak problem znalezienia ścieżki Hamiltona.

Wielomianowy algorytm dla $k = O(\frac{\log n}{\log \log n})$

Idea algorytmu opiera się na wykorzystaniu faktu, że dla spójnego skierowanego acyklicznego grafu (DAG) potrafimy wyznaczyć najdłuższą ścieżkę w czasie wielomianowym, a dokładnie w czasie rzędu $O(m)$, gdzie m jest liczbą krawędzi.



Przekształcenie grafu $G = (V, E)$ w spójny acykliczny graf skierowany $\vec{G}$.

- ▶ Dowolnie ponumeruj wierzchołki grafu różnymi liczbami $1, \ldots, |V|$.
- ▶ Skieruj wszystkie krawędzie zgodnie z wartościami w wierzchołkach tak, aby wierzchołek początkowy każdej skierowanej krawędzi miał mniejszą wartość od wartości wierzchołka końcowego.

Wielomianowy algorytm dla $k = O(\frac{\log n}{\log \log n})$

Idea algorytmu opiera się na wykorzystaniu faktu, że dla spójnego skierowanego acyklicznego grafu (DAG) potrafimy wyznaczyć najdłuższą ścieżkę w czasie wielomianowym, a dokładnie w czasie rzędu $O(m)$, gdzie m jest liczbą krawędzi.

Algorytm RandomLongPath(G, k)

```
1. Niech $p = \frac{2}{(k+1)!}$.
2. for $i := 1$ to $\lceil \frac{1}{p} \rceil$ do
3.     Przypisz losową permutację zbioru $\{1, \ldots, n\}$ wierzchołkom grafu $G$.
4.     Używając tej permutacji skonstruuj spójny skierowany acykliczny graf $\vec{G}$.
5.     Znajdź najdłuższą ścieżkę $P$ w $\vec{G}$ /programowanie dynamiczne/.
6.     if (długość ścieżki $P \geq k$) then return True;
7.         else return False.
```

Lemat 10.14. *Jeśli graf G zawiera ścieżkę P długości k, to ścieżka P będzie ścieżką skierowaną w $\vec{G}$ z prawdopodobieństwem $p = \frac{2}{(k+1)!}$.*

Dowód. Ścieżka P długości k będzie skierowaną ścieżką $\vec{G}$ wtedy i tylko wtedy, gdy jej $k+1$ wierzchołków będzie miało przypisane wartości w sposób (ściśle) rosnący lub (ściśle) malejący. Mając na uwadze, że wszystkie permutacje $k+1$ wierzchołków są jednakowo prawdopodobne i tylko dwie z nich dają ścieżkę skierowaną w $\vec{G}$, ścieżka P będzie ścieżką skierowaną w $\vec{G}$ z prawdopodobieństwem $p = \frac{2}{(k+1)!}$. □

Wniosek 10.15. *Jeśli graf G zawiera ścieżkę P długości k, to $\vec{G}$ zawiera ścieżkę skierowaną długości k z prawdopodobieństwem co najmniej $p = \frac{2}{(k+1)!}$.*

Twierdzenie 10.16. (Alon, Yuster, Zwick 1995)
Algorytm RandomLongPath *znajduje w czasie wielomianowym ścieżkę o długości* $k = O(\frac{\log n}{\log\log n})$ *z prawdopodobieństwem przynajmniej* $1 - \frac{1}{e}$.

Dowód. Niech $t = \lceil \frac{1}{p} \rceil$, gdzie $p = \frac{2}{(k+1)!}$. Załóżmy, że w n-wierzchołkowym grafie $G = (V, E)$ istnieje ścieżka długości k.

↳ Jako że każda z t iteracji jest niezależna oraz mając na uwadze wniosek 10.15, prawdopodobieństwo porażki, tzn. tego, że graf $\vec{G}$ nie posiada skierowanej ścieżki o długości k, wynosi co najwyżej

$$\text{Prob(porażki we wszystkich } t \text{ iteracjach)} \leq (1-p)^t \leq e^{-pt} \leq \frac{1}{e}.$$

(Korzystamy z nierówności $1 + x \leq e^x$ dla każdego $x \in \mathbb{R}$.)

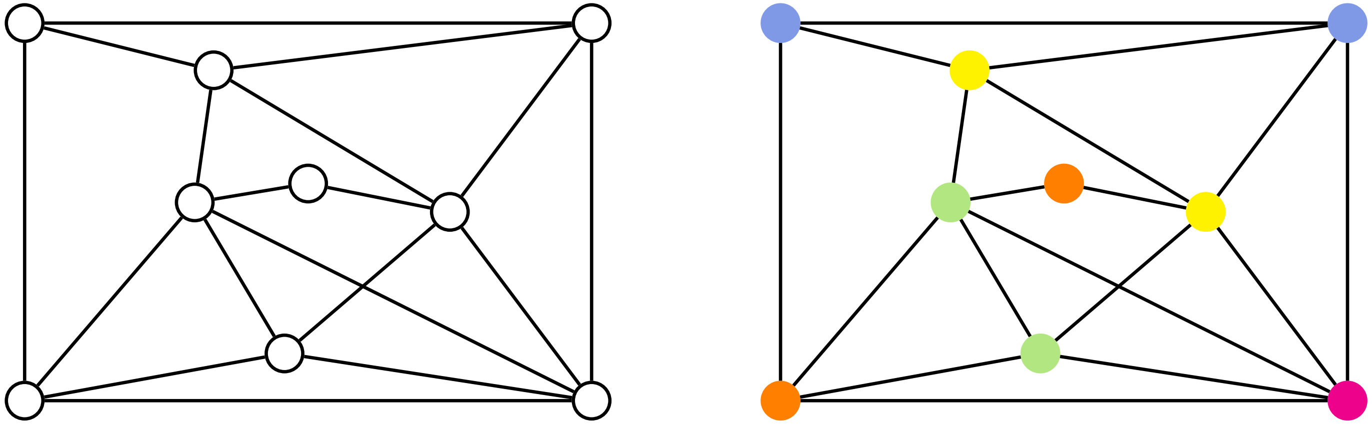
↳ Czas działania $T(n)$ algorytmu wynosi

$$T(n) = t \cdot O(m) = \frac{(k+1)!}{2} \cdot O(m), \text{ gdzie } m = |E|.$$

Zachodzi $(k+1)! \leq (k+1)^k$.
Podstawiając $k = \frac{c \log n}{\log\log n}$, otrzymujemy czas rzędu $O(n^c m)$. □

Wielomianowy algorytm dla $k = O(\log n)$

Idea algorytmu opiera się na losowym przypisaniu wartości ze zbioru $\{1, \ldots, k+1\}$ wierzchołkom wejściowego grafu $G = (V, E)$, a tak otrzymane przypisanie traktowane jest jako $(k+1)$-pokolorowanie wierzchołkowe grafu G.



Niech $G = (V, E)$ będzie spójnym grafem i niech $c \colon V \to \{1, \ldots, k+1\}$ będzie pokolorowaniem jego wierzchołków $k+1$ kolorami. Mówimy, że ścieżka π w grafie G jest *kolorowa*, jeśli wszystkie jej wierzchołki są różnego koloru.

Lemat 10.17. *Dla danego grafu $G = (V, E)$ i dowolnego $(k+1)$-pokolorowania jego wierzchołków, korzystając z programowania dynamicznego, można znaleźć kolorową ścieżkę długości k — o ile istnieje — w czasie $O(2^k km)$, gdzie $m = |E|$.*

Wielomianowy algorytm dla $k = O(\log n)$

Idea algorytmu opiera się na losowym przypisaniu wartości ze zbioru $\{1, \ldots, k+1\}$ wierzchołkom wejściowego grafu $G = (V, E)$, a tak otrzymane przypisanie traktowane jest jako $(k+1)$-pokolorowanie wierzchołkowe grafu G.

Algorytm RandomLongPath2(G, k)

```
Niech $p := \frac{\sqrt{2\pi(k+1)}}{e^{k+1}}$.
for $i := 1$ to $\lceil \frac{1}{p} \rceil$ do
    Pokoloruj losowo wierzchołki grafu $k+1$ kolorami.
    Znajdź najdłuższą kolorową ścieżkę $P$ w $G$ (programowanie dynamiczne).
    if (długość ścieżki $P \geq k$) then return True;
        else return False.
```

Lemat 10.18.
Niech P będzie ścieżką długości k w grafie $G = (V, E)$. Wówczas zachodzi

$$\mathrm{Prob}(P \textit{ jest kolorowa}) \approx \frac{\sqrt{2\pi(k+1)}}{e^{k+1}}.$$

Dowód. Po losowym przypisaniu pokolorowaniu wierzchołków grafu $k+1$ kolorami, ścieżka P jest ścieżką kolorową, jeśli wszystkie jej wierzchołki otrzymały różne kolory. Wszystkich możliwych pokolorowań jest $(k+1)^{k+1}$, z czego tylko $(k+1)!$ czynią ścieżkę P kolorową. Korzystając teraz ze wzoru Stirlinga ($n! \approx (\frac{n}{e})^n\sqrt{2\pi n}$), otrzymujemy:

$$\begin{aligned}\mathrm{Prob}(P \text{ jest kolorowa}) &= \tfrac{(k+1)!}{(k+1)^{k+1}} \\ &\approx \frac{(\frac{k+1}{e})^{k+1}\sqrt{2\pi(k+1)}}{(k+1)^{k+1}} = \frac{\sqrt{2\pi(k+1)}}{e^{k+1}}.\end{aligned}$$

□

Twierdzenie 10.19. (Alon, Yuster, Zwick 1995)
Algorytm RandomLongPath2 *znajduje w czasie wielomianowym ścieżkę o długości* $k = O(\log n)$ *z prawdopodobieństwem przynajmniej* $1 - \frac{1}{e}$.

Dowód. Niech $t = \lceil \frac{1}{p} \rceil$. Załóżmy, że w n-wierzchołkowym grafie $G = (V, E)$ istnieje ścieżka długości k.

↳ Jako że każda z t iteracji jest niezależna oraz mając na uwadze lemat 10.18, prawdopodobieństwo porażki wynosi co najwyżej:

$$\text{Prob}(\text{porażki we wszystkich } t \text{ iteracjach}) \leq (1-p)^t \leq e^{-pt} \leq \frac{1}{e}.$$

↳ Czas działania $T(n)$ algorytmu wynosi

$$T(n) = t \cdot O(2^k km) = O(e^{k+1} \cdot 2^k km), \text{ gdzie } m = |E|.$$

Dla $k = c \log n$ otrzymujemy czas $T(n)$ rzędu:

$$\begin{aligned} T(n) &= O(e^{c \log n + 1} \cdot 2^{c \log n} \cdot c \log n \cdot m) \\ &= O(e^{\log 2n^c} \cdot 2^{\log n^c} \cdot c \log n \cdot m) \\ &= O((2n^c)^{\log e} \cdot n^c \cdot c \log n \cdot m) = O(cn^{4c} m), \end{aligned}$$

który jest czasem wielomianowym dla dowolnej stałej c. □