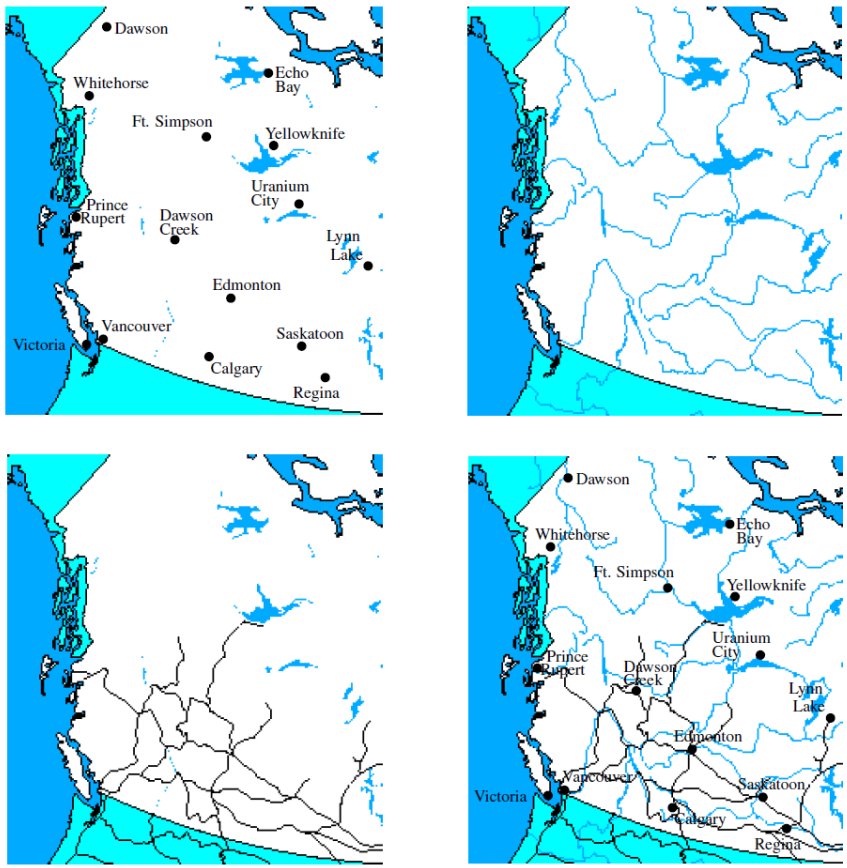
# Geometria obliczeniowa

Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

# Przecinanie się odcinków

W systemach informacji geograficznej mapy są przechowywane jako zbiór warstw, gdzie każda warstwa zawiera inne dane, np. położenie miast, sieć dróg, sieć rzek itp.



Użytkownicy systemu GIS mogą wybrać do zobrazowania jedną z map tematycznych lub nałożyć kilka warstw. Gdy dwie lub więcej warstwy map tematycznych jest pokazanych razem, to częścią wspólną nałożenia są wyjątkowo ciekawe miejsca.

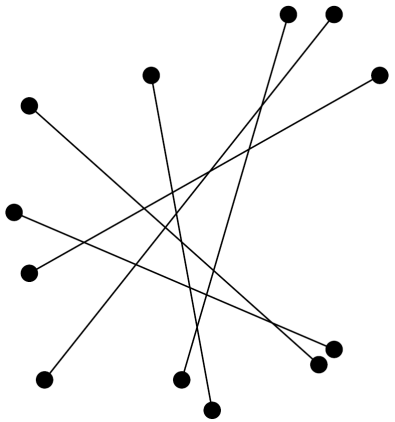
Przykładem może być nałożenie warstwy dróg i rzek. Wówczas przydatne jest wyraźne zaznaczenie ich przecięcia.

Załóżmy, że sieci, które chcemy nałożyć są reprezentowane przez zbiór odcinków.

Problemem, który będziemy chcieli rozwiązać to znalezienie wszystkich przecięć między odcinkami z jednego zbioru i odcinkami z drugiego zbioru.

Odcinki będziemy interpretować jako odcinki domknięte czyli koniec jednego odcinka leżący na drugim odcinku będzie się liczył jako przecięcie.

Dla uproszczenia umieścimy odcinki obu zbiorów w jednym zbiorze i obliczymy wszystkie przecięcia w tym zbiorze.



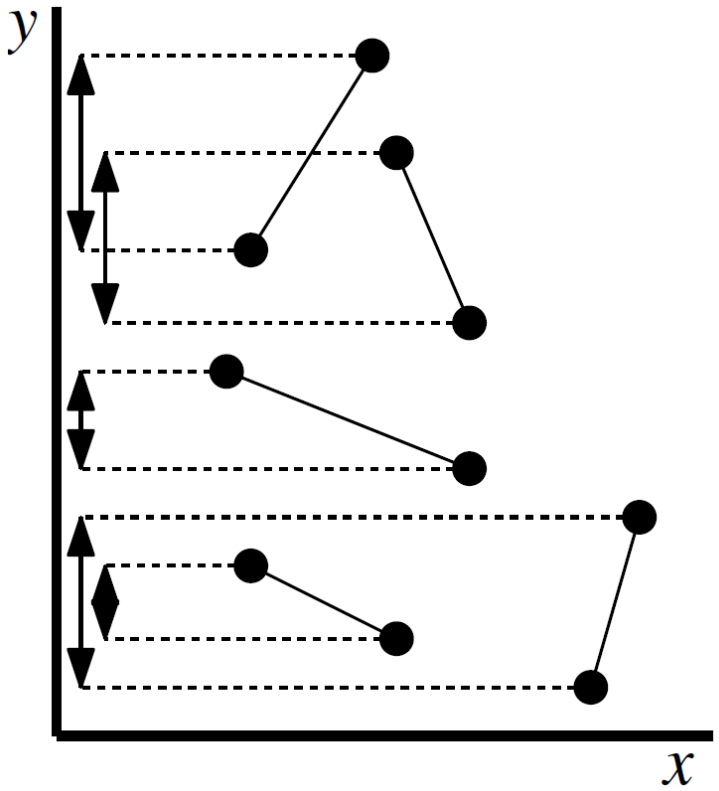
Niech $S$ będzie zbiorem $n$ domkniętych odcinków na płaszczyźnie.

Algorytm siłowy wymaga czasu $\mathcal{O}(n^2)$, ale w praktyce większość odcinków przecina zero lub kilka innych odcinków, więc sumaryczna liczba punktów przecięcia jest znacznie mniejsza niż kwadratowa.

Szukamy algorytmu, który będzie szybszy w takich sytuacjach czyli algorytmu, którego czas będzie zależał nie tylko od liczby odcinków, ale także od liczby punktów przecięcia. Taki algorytm nazywa się algorytmem wrażliwym na wynik (ang. output-sensitive algorithm).

Jak uniknąć sprawdzania wszystkich par odcinków?

Zauważmy, że odcinki, które znajdują się blisko siebie są kandydatami do przecięcia w przeciwieństwie do odcinków, które leżą daleko od siebie.



Zdefiniujmy $y$-przedział odcinka jako jego rzut prostopadły na oś $y$.

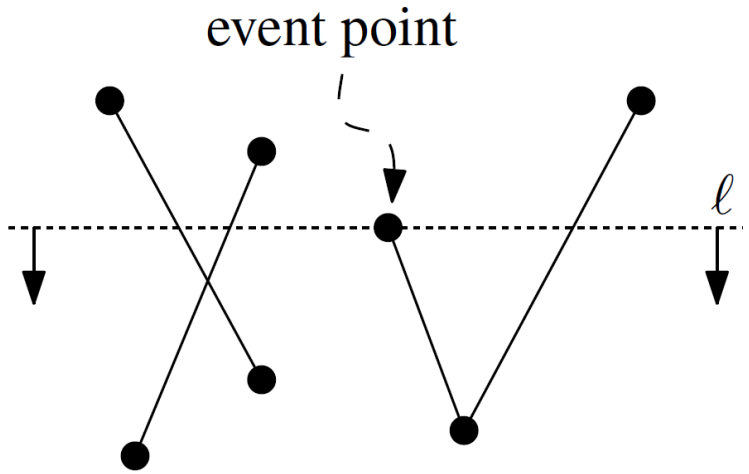
Gdy $y$-przedziały pary odcinków nie zachodzą na siebie to nie mogą się przecinać. Musimy zatem sprawdzać tylko pary odcinków, których $y$-przedziały zachodzą na siebie, tzn. istnieje pozioma prosta przecinająca oba odcinki.

W celu znalezienia par odcinków podejrzanych o przecięcie wyobraźmy sobie prostą $\ell$ przesuwającą się w dół po płaszczyźnie. Prosta startuje z pozycji powyżej wszystkich odcinków.

Przesuwamy prostą $\ell$ w dół i obserwujemy wszystkie odcinki przecinające ją.

Ten typ algorytmu nazywany jest algorytmem zamiatania (ang. plane sweep algorithm), prosta zamiatająca $\ell$ nazywa się miotłą (ang. sweep line). Stanem miotły nazywamy zbiór odcinków przecinających ją.

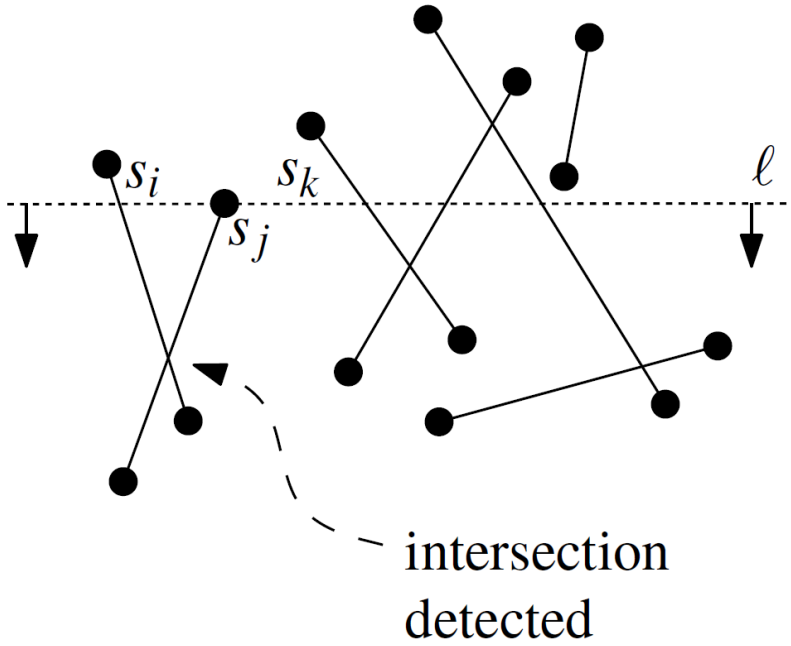
Stan miotły zmienia się, gdy miotła przemieszcza się w dół, ale tylko w punktach szczególnych potrzebna jest aktualizacja stanu. Punkty te nazywamy punktami zdarzeń (ang. event points). Naszymi punktami zdarzeń będą końce odcinków oraz punkty przecięcia odcinków.



Chwile, w których miotła osiąga punkt zdarzeń są jedynymi chwilami, w których algorytm coś robi, tzn. aktualizuje stan miotły i wykonuje testy przecięć.

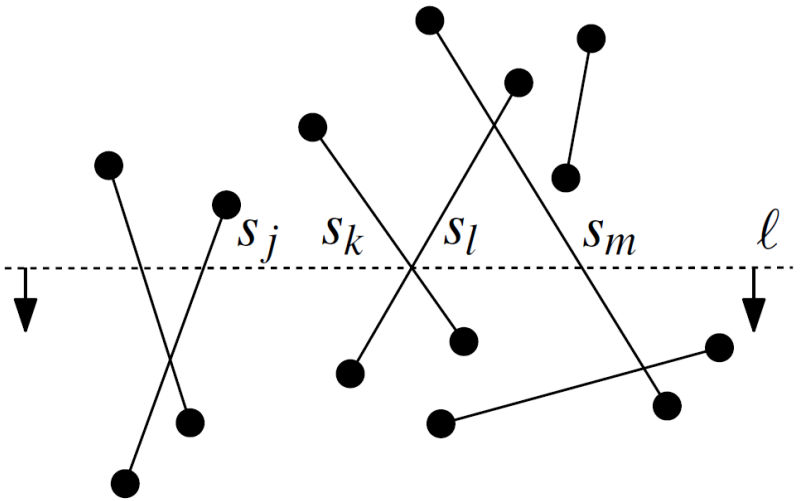
W celu zredukowania liczby testów przecięcia odcinków zawartych w stanie porządkujemy je od lewej do prawej tak, jak przecinają miotłę. Będziemy sprawdzać odcinki tylko wtedy kiedy sąsiadują one w porządku poziomym, tzn. będziemy badać nowy odcinek wraz z odcinkiem bezpośrednio na lewo i prawo od jego górnego końca.

Gdy punkt zdarzeń jest górnym końcem odcinka, to pojawia się nowy odcinek przecinający miotłę. Dla tego odcinka sprawdzamy przecięcie z jego dwoma sąsiadami wzdłuż miotły.

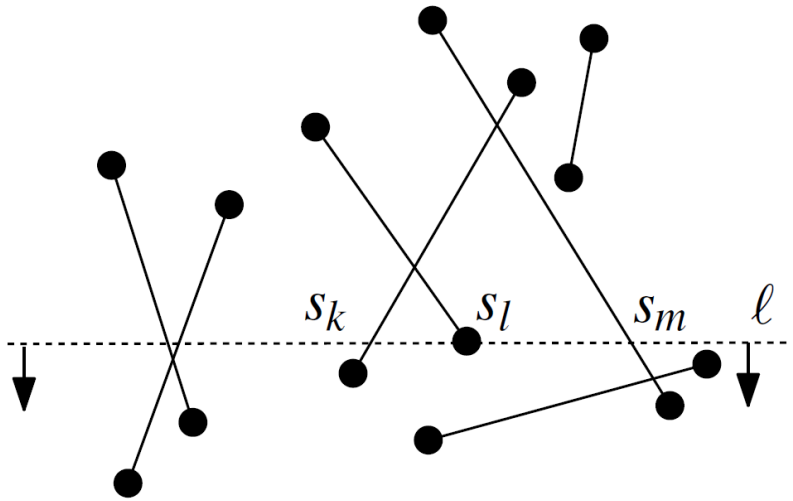



Jeśli znajdziemy przecięcie poniżej miotły, to znaleźliśmy nowy punkt zdarzeń. Przecięcia powyżej miotły zostaly już wcześniej wykryte, więc je pomijamy.

Gdy punkt zdarzeń jest przecięciem, dwa odcinki, które przecinają się zamieniają się kolejnością. Każdy z nich zyskuje (co najwyżej) jednego nowego sąsiada, z którym sprawdza się przecięcie.



Gdy punkt zdarzeń jest dolnym końcem odcinka, to jego dwaj sąsiedzi stają się teraz sąsiadujący i trzeba sprawdzić ich przecięcie.



W naszym algorytmie będziemy potrzebować struktur danych do przechowywania zdarzeń oraz stanu miotły.

W naszym algorytmie będziemy potrzebować struktur danych do przechowywania zdarzeń oraz stanu miotły.

Od kolejki $\mathcal{Q}$ przechowującej zdarzenia wymagamy następujących operacji:

- dodawania zdarzenia,
- usuwania i zwracania z kolejki kolejnego pojawiającego się zdarzenia,
- sprawdzania czy dane zdarzenie jest już w kolejce.

Definiujemy porządek $\prec$ na punktach zdarzeń następująco:

$$p \prec q \iff p_y > q_y \vee (p_y = q_y \wedge p_x < q_x).$$

Punkty zdarzeń przechowujemy w zrównoważonym drzewie BST uporządkowanym zgodnie z porządkiem $\prec$.

Z każdym punktem zdarzeń $p$ w $\mathcal{Q}$ będziemy przechowywać odcinek zaczynający się w $p$, tzn. odcinek, którego górnym końcem jest $p$.

Wydobycie następnego zdarzenia i wstawienie zdarzenia wymagają czasu $\mathcal{O}(\log m)$, gdzie $m$ jest liczbą zdarzeń w $\mathcal{Q}$.

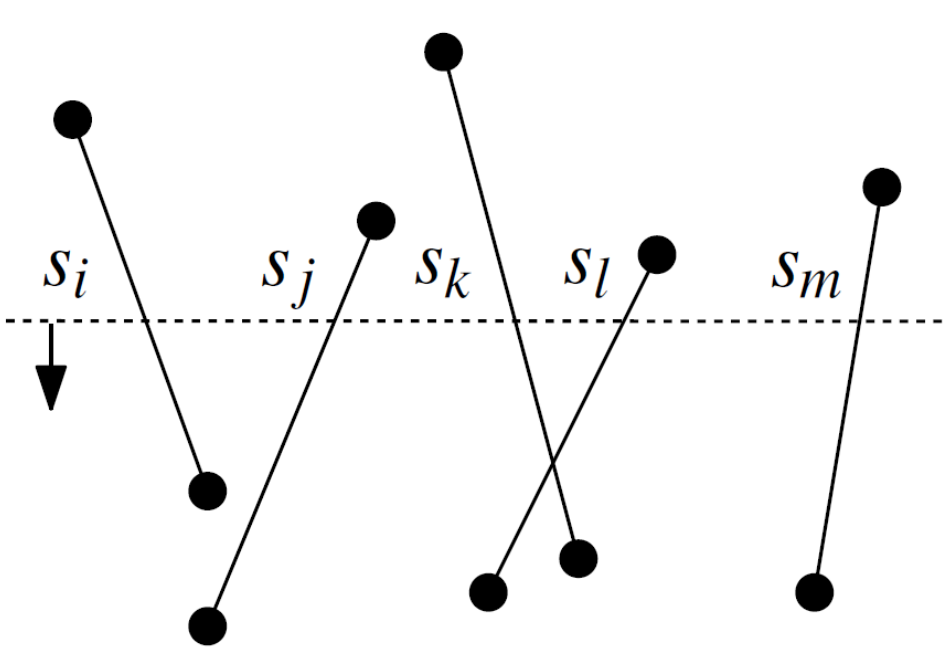
Od struktury $\mathcal{T}$ przechowującej stany (uporządkowany ciąg odcinków przecinających miotłę) wymagamy następujących operacji:

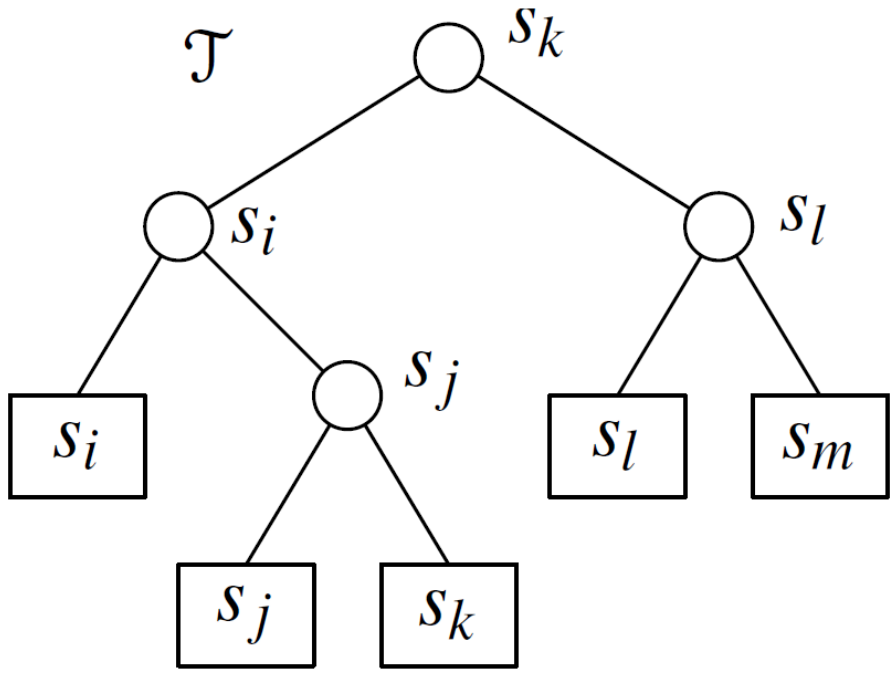
- wyciągnięcie sąsiadów danego odcinka,
- wstawianie odcinka do $\mathcal{T}$,
- usuwanie odcinka z $\mathcal{T}$.

Do reprezentowania $\mathcal{T}$ używamy zrównoważonego drzewa BST.

W drzewie przechowujemy odcinki przecinające miotłę uporządkowane w liściach. Uporządkowanie odcinków od lewej do prawej wzdłuż miotły odpowiada uporządkowaniu liści w $\mathcal{T}$ od lewej do prawej.

W węzłach wewnętrznych przechowujemy odcinek ze skrajnie prawego liścia w jego lewym poddrzewie.



Załóżmy, że wyszukujemy w $\mathcal{T}$ odcinek bezpośrednio na lewo od pewnego punktu $p$ leżącego na miotle.

W każdym węźle wewnętrznym $v$ sprawdzamy czy $p$ leży na lewo lub prawo od odcinka pamiętanego w $v$. W zależności od wyniku schodzimy do lewego lub prawego poddrzewa $v$, ostatecznie kończąc w liściu.

Albo znaleziony liść albo liść bezpośrednio na lewo od niego przechowuje odcinek, którego szukaliśmy.

W podobny sposób możemy znaleźć odcinek bezpośrednio na prawo od $p$ lub odcinek zwierający $p$.

**Algorithm** FINDINTERSECTIONS($S$)

*Input.* A set $S$ of line segments in the plane.

*Output.* The set of intersection points among the segments in $S$, with for each intersection point the segments that contain it.

1. Initialize an empty event queue $\mathcal{Q}$. Next, insert the segment endpoints into $\mathcal{Q}$; when an upper endpoint is inserted, the corresponding segment should be stored with it.
2. Initialize an empty status structure $\mathcal{T}$.
3. **while** $\mathcal{Q}$ is not empty
4. **do** Determine the next event point $p$ in $\mathcal{Q}$ and delete it.
5. HANDLEEVENTPOINT($p$)

HANDLEEVENTPOINT($p$)

1. Let $U(p)$ be the set of segments whose upper endpoint is $p$; these segments are stored with the event point $p$. (For horizontal segments, the upper endpoint is by definition the left endpoint.)
2. Find all segments stored in $\mathcal{T}$ that contain $p$; they are adjacent in $\mathcal{T}$. Let $L(p)$ denote the subset of segments found whose lower endpoint is $p$, and let $C(p)$ denote the subset of segments found that contain $p$ in their interior.
3. **if** $L(p) \cup U(p) \cup C(p)$ contains more than one segment
4. **then** Report $p$ as an intersection, together with $L(p)$, $U(p)$, and $C(p)$.
5. Delete the segments in $L(p) \cup C(p)$ from $\mathcal{T}$.
6. Insert the segments in $U(p) \cup C(p)$ into $\mathcal{T}$. The order of the segments in $\mathcal{T}$ should correspond to the order in which they are intersected by a sweep line just below $p$. If there is a horizontal segment, it comes last among all segments containing $p$.
7. ($*$ Deleting and re-inserting the segments of $C(p)$ reverses their order. $*$)
8. **if** $U(p) \cup C(p) = \emptyset$
9. **then** Let $s_l$ and $s_r$ be the left and right neighbors of $p$ in $\mathcal{T}$.
10. FINDNEWEVENT($s_l, s_r, p$)
11. **else** Let $s'$ be the leftmost segment of $U(p) \cup C(p)$ in $\mathcal{T}$.
12. Let $s_l$ be the left neighbor of $s'$ in $\mathcal{T}$.
13. FINDNEWEVENT($s_l, s', p$)
14. Let $s''$ be the rightmost segment of $U(p) \cup C(p)$ in $\mathcal{T}$.
15. Let $s_r$ be the right neighbor of $s''$ in $\mathcal{T}$.
16. FINDNEWEVENT($s'', s_r, p$)

FINDNEWEVENT($s_l, s_r, p$)

1. **if** $s_l$ and $s_r$ intersect below the sweep line, or on it and to the right of the current event point $p$, and the intersection is not yet present as an event in $\mathcal{Q}$
2. **then** Insert the intersection point as an event into $\mathcal{Q}$.

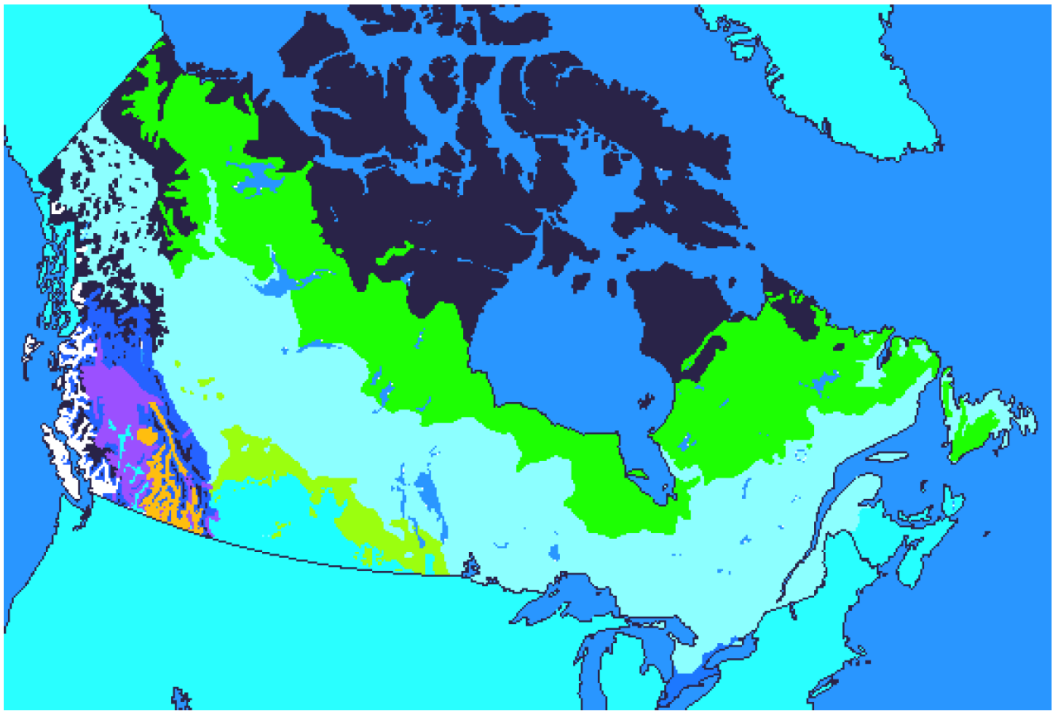
FINDNEWEVENT($s_l, s_r, p$)

1. **if** $s_l$ and $s_r$ intersect below the sweep line, or on it and to the right of the current event point $p$, and the intersection is not yet present as an event in $\mathcal{Q}$
2. **then** Insert the intersection point as an event into $\mathcal{Q}$.

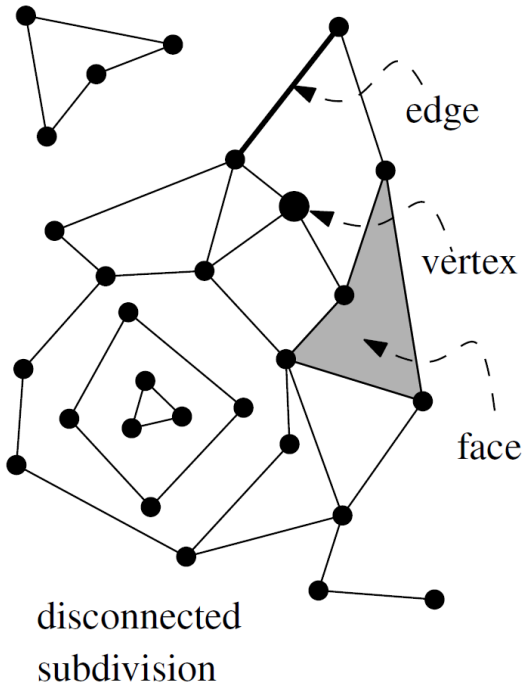
Niech $S$ będzie zbiorem $n$ odcinków na płaszczyźnie. Wszystkie punkty przecięcia w $S$, z odcinkami związanymi z każdym z nich można podać w czasie $\mathcal{O}(n \log n + I \log n)$, z pamięcią $\mathcal{O}(n)$, gdzie $I$ jest liczbą punktów przecięcia.

# Nałożenie dwóch podziałów

Większość map ma strukturę bardziej złożoną niż zbiór odcinków. Najczęściej są one podziałami płaszczyzny na etykietowane regiony, np. mapa lasów w Kanadzie może być podziałem na regiony z etykietami: sosnowy, zrzucający liście, brzozowy, mieszany itp.

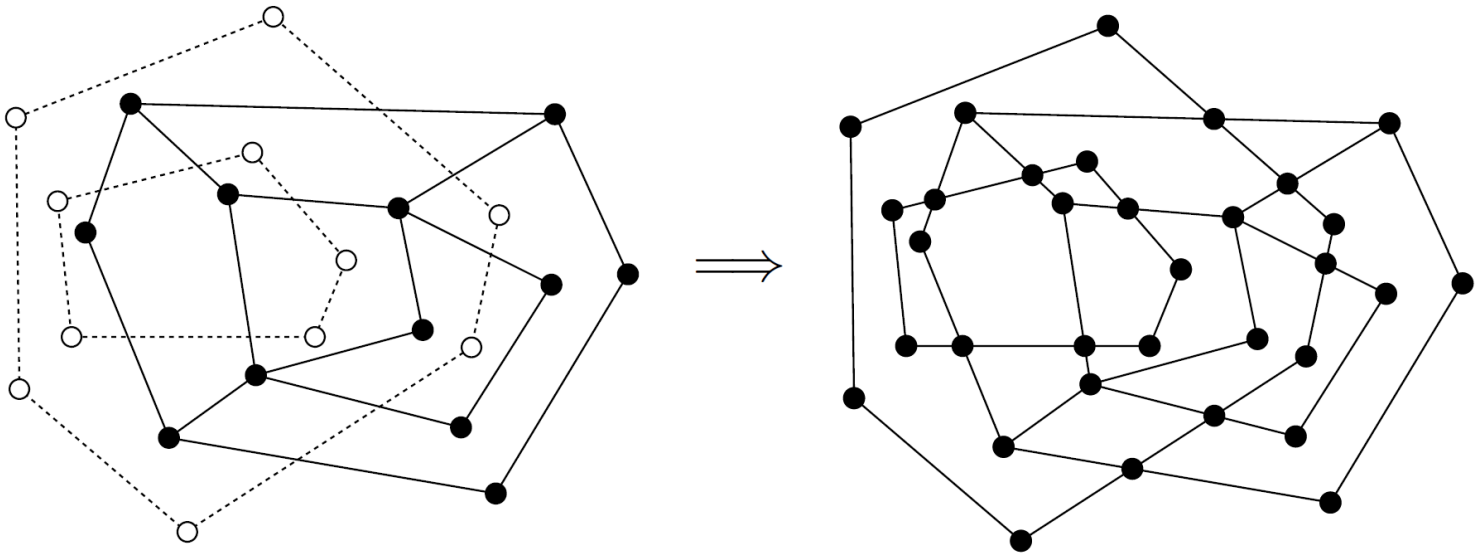


Mapy, które będziemy brać pod uwagę będą podziałami planarnymi (ang. planar subdivision).




Aby algorytm nakładania działał w sposób efektywny do przechowywania podziałów używać będziemy podwójnie łączonej listy krawędzi.

Nałożenie dwóch podziałów $\mathcal{S}_1, \mathcal{S}_2$ definiujemy jako podział taki, że istnieje ściana $f$ w tym podziale $\iff$ istnieją ściany $f_1$ w $\mathcal{S}_1$ i $f_2$ w $\mathcal{S}_2$, dla których $f$ jest maksymalnym spójnym podzbiorem $f_1 \cap f_2$. Taki podział oznaczamy przez $O(\mathcal{S}_1, \mathcal{S}_2)$.



Problem nakładania map polega na obliczeniu podwójnie łączonej listy krawędzi dla $O(\mathcal{S}_1, \mathcal{S}_2)$ z danych list dla $\mathcal{S}_1$ i $\mathcal{S}_2$.

Ponadto chcemy, aby każda ściana w $O(\mathcal{S}_1, \mathcal{S}_2)$ była etykietowana etykietami ścian w $\mathcal{S}_1$ i $\mathcal{S}_2$, które je zawierają. W ten sposób mamy dostęp do informacji o własnościach pamiętanych dla tych ścian.

Problem nakładania map polega na obliczeniu podwójnie łączonej listy krawędzi dla $O(\mathcal{S}_1, \mathcal{S}_2)$ z danych list dla $\mathcal{S}_1$ i $\mathcal{S}_2$.

Ponadto chcemy, aby każda ściana w $O(\mathcal{S}_1, \mathcal{S}_2)$ była etykietowana etykietami ścian w $\mathcal{S}_1$ i $\mathcal{S}_2$, które je zawierają. W ten sposób mamy dostęp do informacji o własnościach pamiętanych dla tych ścian.

Aby wyznaczyć nakładanie się map najpierw kopiujemy podwójnie łączone listy krawędzi $\mathcal{S}_1$, $\mathcal{S}_2$ do nowej podwójnie łączonej listy krawędzi. Następnie wykonujemy dwa kroki:

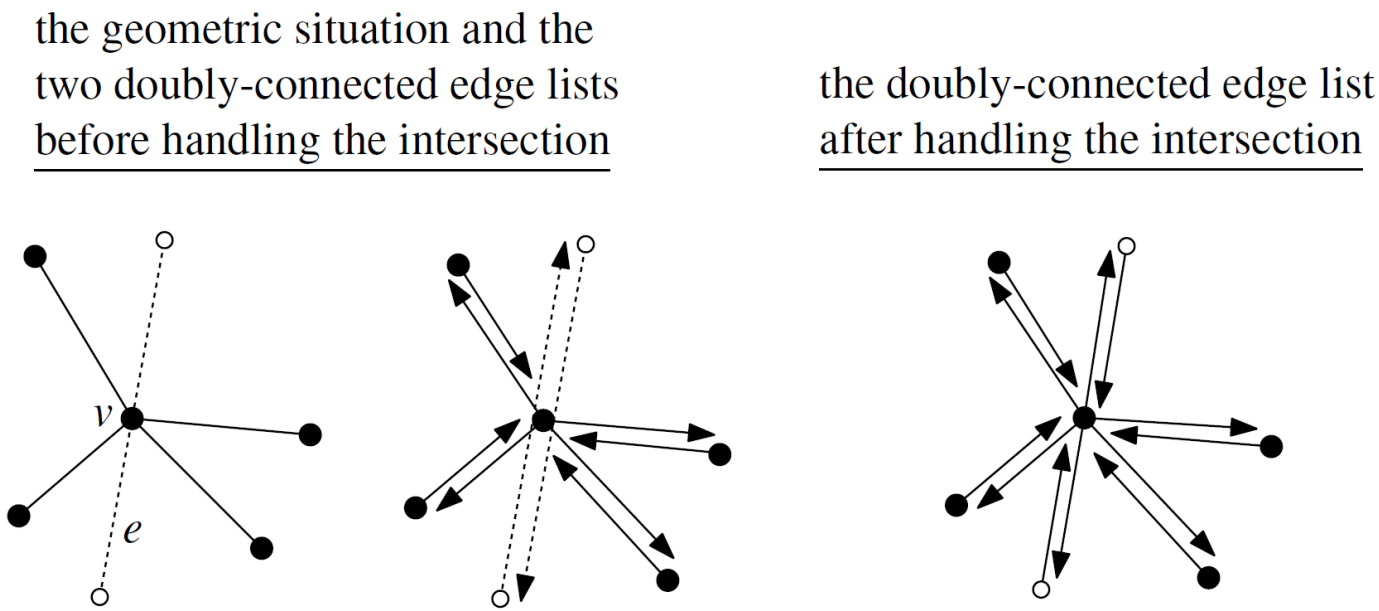
- obliczamy przecięcia w nowej liście,
- tworzymy rekordy nowych ścian.

Pierwszy krok wykonujemy używając algorytmu przecinania się odcinków.

W strukturze $\mathcal{D}$ początkowo umieszczamy kopię podwójnie łączonej listy krawędzi dla $\mathcal{S}_1$ i $\mathcal{S}_2$. Podczas zamiatania przekształcamy $\mathcal{D}$ w poprawną podwójnie łączoną listę krawędzi dla $O(\mathcal{S}_1, \mathcal{S}_2)$, tzn. rekordy wierzchołka i półkrawędzi, bo rekordy ścian utworzymy w drugim kroku.

W strukturze stanu $\mathcal{T}$ utrzymujemy krzyżujące się wskaźniki między krawędziami w $\mathcal{T}$ a odpowiadającymi im rekordami półkrawędzi w $\mathcal{D}$. W ten sposób mamy dostęp do części $\mathcal{D}$, która powinna być zmieniona, gdy napotkamy punkt przecięcia.

W punkcie zdarzeń aktualizujemy struktury $\mathcal{T}$ i $\mathcal{D}$ jak w algorytmie przecięć odcinków. Jeśli zdarzenie dotyczy krawędzi z obu podziałów, to musimy dokonać lokalnych zmian w $\mathcal{D}$, aby połączyć podwójnie łączone listy krawędzi dwóch pierwotnych podziałów w punkcie przecięcia. W tym celu musimy rozpatrzyć różne przypadki, np.

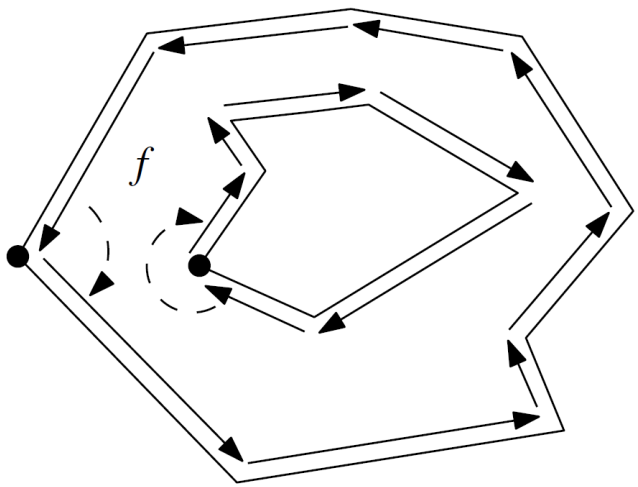


W drugim kroku naszego algorytmu tworzymy rekordy ścian. W tym celu będziemy badać cykle jakie tworzą półkrawędzie stworzone w poprzednim kroku.

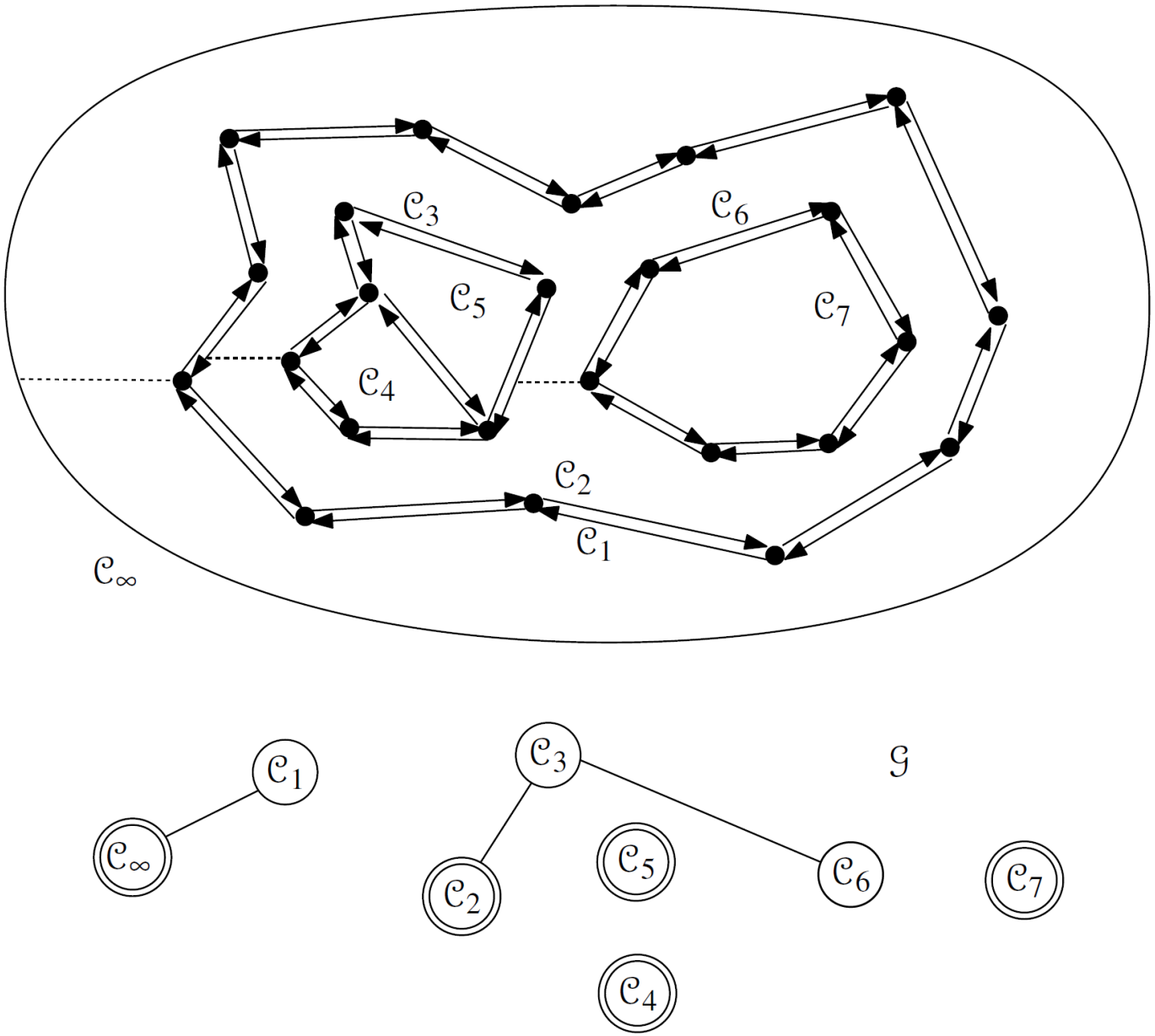
W drugim kroku naszego algorytmu tworzymy rekordy ścian. W tym celu będziemy badać cykle jakie tworzą półkrawędzie stworzone w poprzednim kroku.

W jaki sposób poznać czy cykl jest na brzegu zewnętrznym czy brzegu dziury w ścianie?

Możemy to określić patrząc na skrajnie lewy wierzchołek cyklu lub w przypadku niejednoznacznym na najniższy ze skrajnie lewych wierzchołków. Bierzemy półkrawędzie incydentne z naszym wierzchołkiem i obliczamy kąt pomiędzy nimi. Jeśli kąt jest mniejszy od $180^\circ$, to cykl jest na brzegu zewnętrznym, w przeciwnym przypadku jest to brzeg dziury.

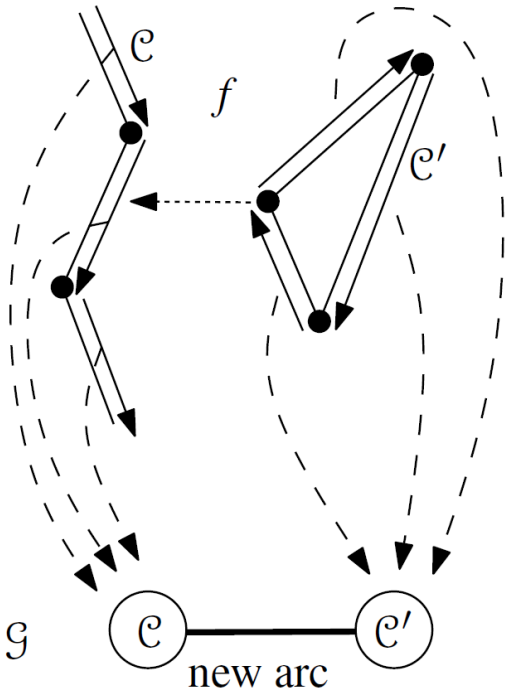


Aby zdecydować, które cykle brzegowe ograniczają te same ściany tworzymy graf $\mathcal{G}$.
Dla każdego cyklu istnieje wierzchołek w $\mathcal{G}$. Jest też jeden wierzchołek dla urojonego zewnętrznego brzegu nieograniczonej ściany.
Między dwoma cyklami istnieje krawędź $\iff$ jeden z cykli jest brzegiem dziury a drugi cykl ma półkrawędź bezpośrednio na lewo od skrajnie lewego wierzchołka cyklu tej dziury. Jeśli nie ma takiej półkrawędzi, to wierzchołek łączymy z wierzchołkiem urojonego brzegu.


$\mathcal{C}_3$
$\mathcal{C}_6$
$\mathcal{C}_5$
$\mathcal{C}_7$
$\mathcal{C}_4$
$\mathcal{C}_2$
$\mathcal{C}_1$
$\mathcal{C}_\infty$
$\mathcal{C}_1$
$\mathcal{C}_3$
$\mathcal{G}$
$\mathcal{C}_\infty$
$\mathcal{C}_5$
$\mathcal{C}_6$
$\mathcal{C}_7$
$\mathcal{C}_2$
$\mathcal{C}_4$

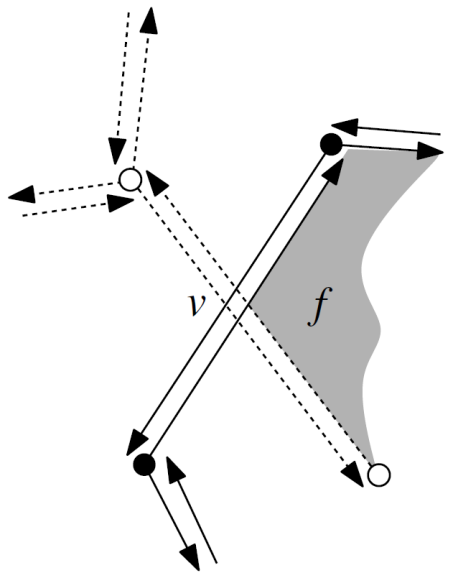
Jak możemy skonstruować graf $\mathcal{G}$?

W algorytmie zamiatania dla przecięć odcinków zawsze szukaliśmy odcinków bezpośrednio na lewo od punktu zdarzeń. Zatem informacja, której potrzebujemy do konstrukcji $\mathcal{G}$ jest określana podczas zamiatania.



Pozostało nam etykietowanie ścian naszego podziału. Niech $f$ będzie ścianą, którą etykietujemy i rozważmy dowolny wierzchołek $v$ tej ściany.

Jeśli $v$ jest przecięciem krawędzi $e_1$ z $\mathcal{S}_1$ i $e_2$ z $\mathcal{S}_2$, to patrzymy na wskaźniki ściany incydentnej odpowiednich półkrawędzi odpowiadających $e_1$ i $e_2$.



Jeśli $v$ nie jest przecięciem, ale jest wierzchołkiem powiedzmy $\mathcal{S}_1$, to znamy tylko ścianę z $\mathcal{S}_1$ zawierającą $f$. Musimy znaleźć ścianę w $\mathcal{S}_2$, która zawiera $v$.

W tym celu musimy dla każdego wierzchołka z $\mathcal{S}_1$ mieć informację, która ściana w $\mathcal{S}_2$ go zawiera (i odwrotnie). Do wyznaczenia tych informacji możemy stworzyć algorytm wykorzystujący paradygmat zamiatania. (Można zmodyfikować zamiatanie obliczające przecięcia).

**Algorithm** MAPOVERLAY($\mathcal{S}_1, \mathcal{S}_2$)
*Input.* Two planar subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$ stored in doubly-connected edge lists.
*Output.* The overlay of $\mathcal{S}_1$ and $\mathcal{S}_2$ stored in a doubly-connected edge list $\mathcal{D}$.

1. Copy the doubly-connected edge lists for $\mathcal{S}_1$ and $\mathcal{S}_2$ to a new doubly-connected edge list $\mathcal{D}$.

2. Compute all intersections between edges from $\mathcal{S}_1$ and $\mathcal{S}_2$ with the plane sweep algorithm of Section 2.1. In addition to the actions on $\mathcal{T}$ and $\mathcal{Q}$ required at the event points, do the following:

   - Update $\mathcal{D}$ as explained above if the event involves edges of both $\mathcal{S}_1$ and $\mathcal{S}_2$. (This was explained for the case where an edge of $\mathcal{S}_1$ passes through a vertex of $\mathcal{S}_2$.)
   - Store the half-edge immediately to the left of the event point at the vertex in $\mathcal{D}$ representing it.

3. ($*$ Now $\mathcal{D}$ is the doubly-connected edge list for $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$, except that the information about the faces has not been computed yet. $*$)
4. Determine the boundary cycles in $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$ by traversing $\mathcal{D}$.
5. Construct the graph $\mathcal{G}$ whose nodes correspond to boundary cycles and whose arcs connect each hole cycle to the cycle to the left of its leftmost vertex, and compute its connected components. (The information to determine the arcs of $\mathcal{G}$ has been computed in line 2, second item.)
6. **for** each connected component in $\mathcal{G}$
7. **do** Let $\mathcal{C}$ be the unique outer boundary cycle in the component and let $f$ denote the face bounded by the cycle. Create a face record for $f$, set $\mathit{OuterComponent}(f)$ to some half-edge of $\mathcal{C}$, and construct the list $\mathit{InnerComponents}(f)$ consisting of pointers to one half-edge in each hole cycle in the component. Let the $\mathit{IncidentFace}()$ pointers of all half-edges in the cycles point to the face record of $f$.
8. Label each face of $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$ with the names of the faces of $\mathcal{S}_1$ and $\mathcal{S}_2$ containing it, as explained above.

Niech $\mathcal{S}_1$ będzie podziałem planarnym o złożoności $n_1$, a $\mathcal{S}_2$ będzie podziałem o złożoności $n_2$ oraz niech $n = n_1 + n_2$. Nałożenie $\mathcal{S}_1$ i $\mathcal{S}_2$ można obliczyć w czasie $\mathcal{O}(n \log n + k \log n)$, gdzie $k$ jest złożonością nałożenia.