Obliczanie nałożenia dwóch podziałów 2D

Algorytmy geometryczne

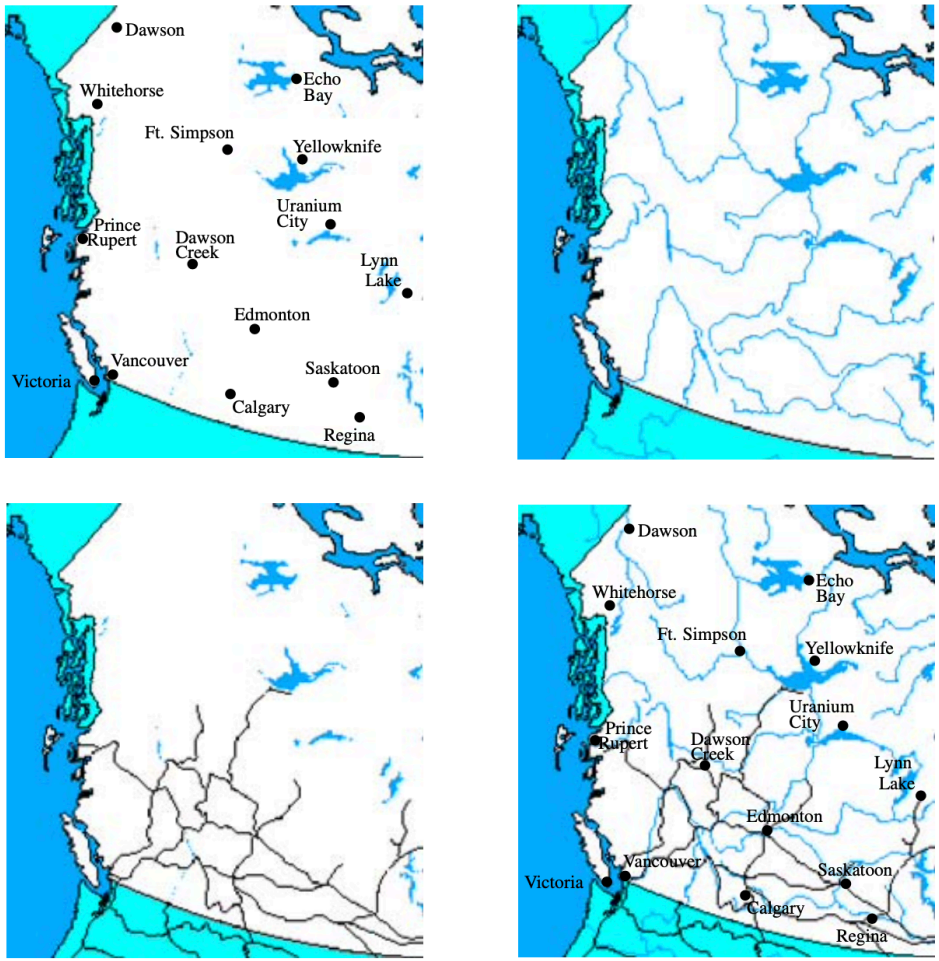
Michał Szafarczyk Piotr Czarnik

9 stycznia 2022

Spis treści

Opis problemu 1

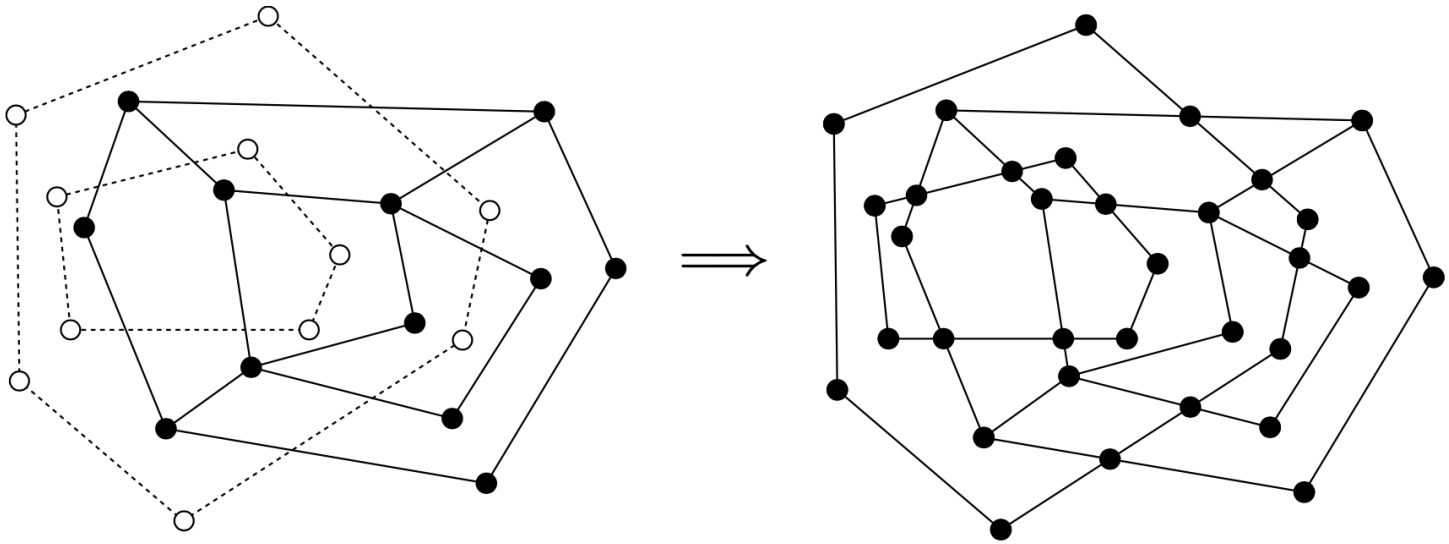
W celu uniknięcia nadmiaru informacji na mapie, a jednocześnie z zamiarem maksymalizacji przekazywanej ilości informacji, mapy dzieli się na warstwy, z których każda przedstawia inny rodzaj informacji (np. położenie rzek, miast i torów kolejowych).



Rysunek 1: Mapy miast, rzek, kolei i ich nałożenie w zachodniej Kanadzie.

Opis problemu 2

Mając takie różne podziały płaszczyzny (mapy) można chcieć uzyskać informację z wielu warstw równocześnie (np. gdzie występują przecięcia rzek i torów tam należy zbudować wiadukty). Aby taką informację uzyskać należy obliczyć nałożenie dwóch podziałów płaszczyzny, wynikiem czego uzyska się jeden podział, zawierający równocześnie informacje z obu podziałów.



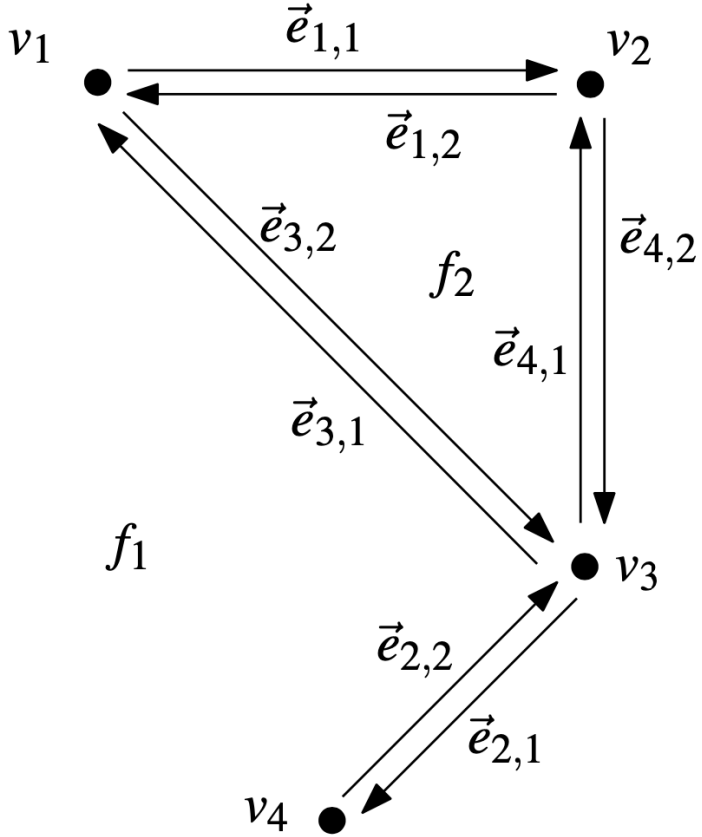
Rysunek 2: Przykładowe przekształcenie dwóch podziałów płaszczyzny w jeden.

Podwójnie łączona lista krawędzi *DCEL* 1

Podwójnie łączona lista krawędzi (doubly connected edge list, w skrócie *DCEL*) to struktura służąca do przechowywania podziału płaszczyzny za pomocą grafu planarnego. Charakteryzuje się tym, że z każdą krawędzią związane są dwie półkrawędzie. Na strukturę składają się trzy rodzaje rekordów:

- wierzchołek (*VertexRecord*): współrzędne x, y; wskaźnik do dowolnej incydentnej krawędzi *incidentalEdge*.
- ściana (*FaceRecord*): wskaźnik do dowolnej półkrawędzi otaczającej ścianę *outerEdge*; tablica wskaźników do półkrawędzi, które znajdują się wewnątrz ściany *innerEdges*.
- półkrawędź (*HalfEdgeRecord*): wskaźnik do wierzchołka, z którego półkrawędź wychodzi *beginning*; wskaźnik do bliźniaczej półkrawędzi *twinEdge*; wskaźnik do kolejnej półkrawędzi w cyklu *nextEdge*; wskaźnik do poprzedniej półkrawędzi w cyklu *previousEdge*; wskaźnik do incydentnej ściany (otaczanej przez dany cykl) *incidentalFace*.

Podwójnie łączona lista krawędzi *DCEL* 2



Rysunek 3: Przykładowa reprezentacja grafu za pomocą *DCEL*.

Przedstawienie kilku zależności w *DCEL* z rys. 3:

$$v_1.incidentalEdge = e_{1,1}$$

$$v_2.incidentalEdge = e_{1,2}$$

$$e_{1,2}.twinEdge = e_{1,1}$$

$$e_{1,1}.twinEdge = e_{1,2}$$

$$e_{1,2}.nextEdge = e_{3,2}$$

$$e_{1,2}.prevEdge = e_{4,1}$$

$$e_{1,2}.incidentalFace = f_2$$

$$f_2.outerEdge = e_{1,2}$$

Struktura zdarzeń

Struktura zdarzeń jest implementowana przez zrównoważone binarne drzewo wyszukiwań (w tym przypadku AVL). Zawiera zdarzenia, w których miotła się zatrzyma i zawiera jedynie zdarzenia (punkty) znajdujące się poniżej miotły. W węzłach przechowywane są obiekty typu *QueueRecord*:

- współrzędne punktu reprezentującego zdarzenie x, y.
- wskaźniki *left*, *right*, *parent*.
- rodzaj zdarzenia *type* (początek krawędzi, koniec krawędzi, przecięcie krawędzi).
- tablica krawędzi U, dla których punkt jest górnym końcem.
- tablica krawędzi L, dla których punkt jest dolnym końcem.
- tablica krawędzi C, które zawierają punkt w swoim wnętrzu.

W drzewie został wprowadzony następujący porządek:

$$p_1 \prec p_2 \Leftrightarrow p_{1y} > p_{2y} \vee (p_{1y} = p_{2y} \wedge p_{1x} < p_{2x})$$

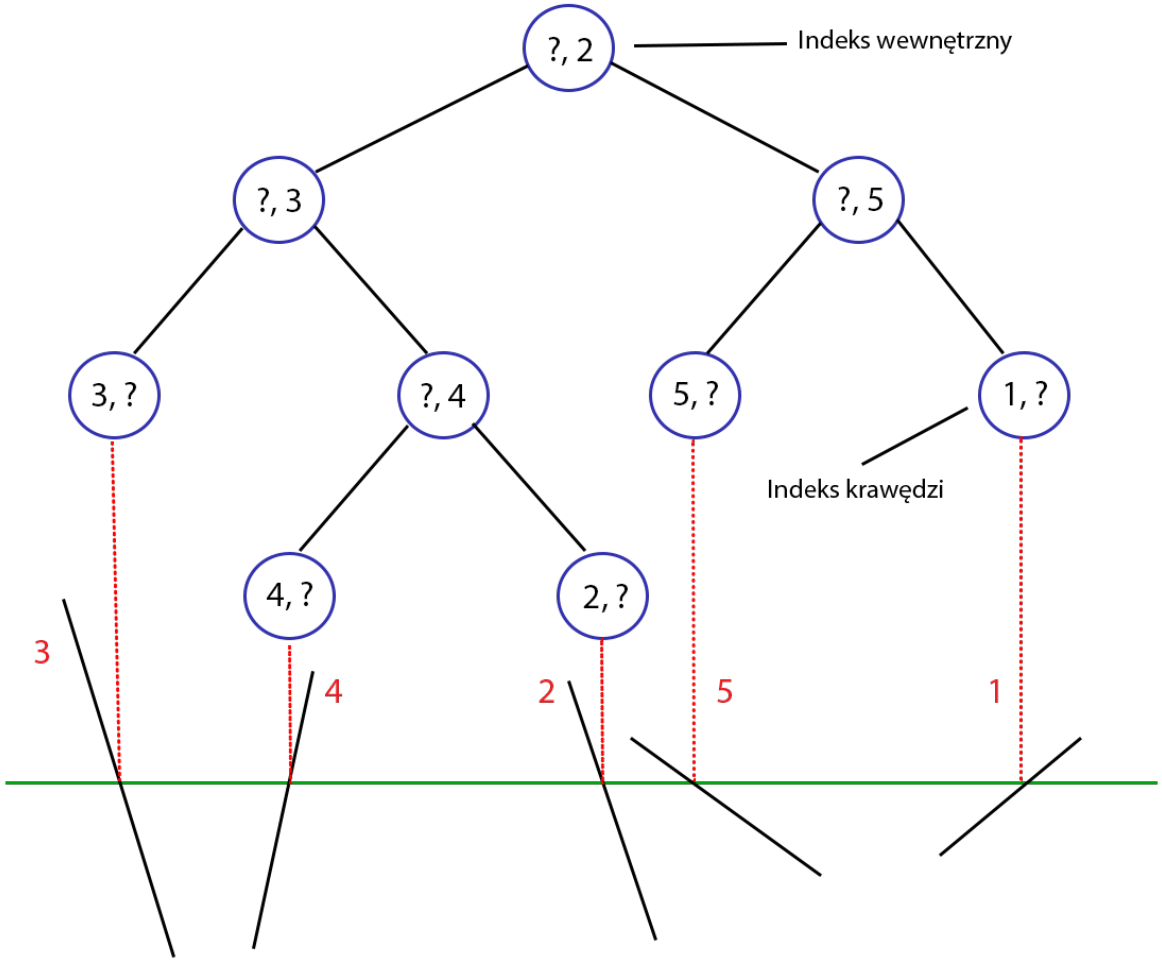
Struktura stanu miotły 1

Struktura stanu miotły implementowana jest przez zrównoważone binarne drzewo wyszukiwań (w tym przypadku AVL). Wykorzystywane jest do przechowywania kolejności krawędzi, w jakiej przecinają miotłę. W tym celu w liściach trzymane są indeksy krawędzi przecinających miotłę, a pozostałe wierzchołki służą do nawigowania po tym drzewie.
W każdym wierzchołku miotły trzymany jest obiekt typu *StateRecord*:

- indeks krawędzi *edgeIndex*, jeśli obiekt jest liściem.
- indeks krawędzi na lewo i maksymalnie na prawo *insideNodeIndex*, jeśli obiekt nie jest liściem.
- wskaźniki *left*, *right*, *parent*.

StateRecord umożliwia działanie zarówno jako liść, jak i wierzchołek wewnętrzny drzewa. Wierzchołki pośrednie przechowują indeks krawędzi w liściu na lewo i maksymalnie na prawo w drzewie.

Struktura stanu miotły 2



Rysunek 4: Przykładowe drzewo stanu miotły. Na zielono zaznaczono aktualną pozycję miotły, na czerwono indeksy krawędzi.

Algorytm Compute-Map-Overlay

Pseudokod głównego algorytmu programu, który oblicza nałożenie dwóch podziałów płaszczyzny 2D.

Compute-Map-Overlay(D_1, D_2)

Wejście: Dwie podwójnie łączone listy krawędzi (DCEL)
D_1 i D_2 reprezentujące podziały płaszczyzny.
Wyjście: DCEL D reprezentująca nałożenie podziałów
reprezentowanych przez dwie wejściowe DCEL D_1 i D_2.

1 $D =$ Merge-DCEL(D_1, D_2)
2 Convert-to-Planar-Representation(D)
3 Fix-Faces(D)
4 **return** D

Algorytm MERGE-DCEL

Pseudokod algorytmu scalającego dwie podwójnie łączone listy krawędzi w jedną.

MERGE-DCEL(D_1, D_2)

1 Utwórz nową pustą DCEL D.
2 Przepisz każdą krawędź z D_1 i D_2 do D, jeżeli się nie powtarza.
3 Przepisz każdy wierzchołek z D_1 i D_2 do D, jeżeli się nie powtarza. Jeżeli istnieją dwa takie same wierzchołki w D_1 i D_2, przepisz jeden z nich i przepnij wszystkie incydentne krawędzie do tego wierzchołka.
4 **return** D

Algorytm Convert-to-Planar-Representation 1

Pseudokod algorytmu przekształcającego podział płaszczyzny uzyskany po scaleniu dwóch *DCEL* do poprawnej, planarnej postaci *DCEL*.

Convert-to-Planar-Representation(D)

Zainicjalizuj poniższe struktury:

1. Q - Kolejka zdarzeń. Początkowo zawiera zdarzenia, będące punktami znalezionymi podczas scalania dwóch DCEL.
2. T - Struktura stanu miotły. Początkowo jest puste.
3. ME - Tablica indeksów krawędzi przyporządkowanych w następujący sposób: po podziale krawędzi, jej indeks jest przypisywany tej nowo powstałej krawędzi, która jest położona niżej. Początkowo każdej krawędzi odpowiada jej własny indeks.
4. IP - Tablica z punktami przecięć. Początkowo zawiera wszystkie punkty przecięć znalezione podczas scalania dwóch DCEL.

Algorytm Convert-to-Planar-Representation 2

5 Pobierz pierwsze zdarzenie z Q i dodaj do T wszystkie krawędzie incydentne do tego zdarzenia (punktu).
6 **while** Q nie jest puste
7 Niech p będzie kolejnym zdarzeniem wyjętym z Q.
8 Przyporządkuj właściwe indeksy w ME wszystkim krawędziom zawierającym punkt p w swoim wnętrzu.

Algorytm Convert-to-Planar-Representation 3

9 **if** jakaś krawędź przechodzi przez lub kończy się w punkcie p i jest przechowywana w zdarzeniu p

10 Usuń z T wszystkie krawędzie, które zawierały lub kończyły się w punkcie p i były przechowywane w zdarzeniu p. Dla ostatniej usuniętej krawędzi znajdź lewego i prawego sąsiada.

11 Dodaj do Q ewentualne przecięcie znalezionych sąsiadów.

12 **else** przez punkt p może przechodzić jedna krawędź, która nie została zapisana w zdarzeniu p. Może tak się stać, gdy w punkcie p zaczyna się wiele krawędzi.

13 Nawigując po drzewie T spróbuj usunąć krawędź, która przechodzi przez punkt p.

14 Jeżeli usunięto krawędź, to dodaj ją do zbioru krawędzi przechodzących przez punkt p oraz dodaj p do zbioru IP

Algorytm CONVERT-TO-PLANAR-REPRESENTATION 4

15 **if** punkt p to przecięcie krawędzi e_1 i e_2
16 HANDLE-INTERSECTION-A(D, p, e_1, e_2)
17 **elseif** krawędź e przechodzi przez punkt p
18 HANDLE-INTERSECTION-B(D, p, e)
19 **else** punkt p jest początkiem krawędzi
20 **if** w p zaczyna się tylko jedna krawędź
21 Dodaj tę krawędź do drzewa T.
22 Znajdź lewego i prawego sąsiada tej krawędzi i dodaj ich ewentualne przecięcie do kolejki Q.

Algorytm Convert-to-Planar-Representation 5

23 **elseif** w p zaczyna się lub przecina się więcej niż jedna krawędź

24 Dodaj w odpowiedniej kolejności (od lewej do prawej) nowe krawędzie do T.

25 Znajdź lewego sąsiada nowo dodanej krawędzi znajdującej się najbardziej na lewo i dodaj ich ewentualne przecięcie do Q. Zrób to samo dla prawego sąsiada krawędzi znajdującej się najbardziej na prawo.

26 **return** IP

Algorytm Handle-Intersection-A

Pseudokod algorytmu naprawiającego D, gdy krawędzie e_1 i e_2 przecinają się w punkcie p.

Handle-Intersection-A(D, p, e_1, e_2)

1. Podziel krawędzie e_1 i e_2 względem punktu przecięcia. Dla każdej części utwórz dwie nowe półkrawędzie (łącznie osiem nowych półkrawędzi) i połącz je odpowiednio w bliźniacze pary (pole *twinEdge*).
2. Dodaj po jednej półkrawędzi z bliźniaczej pary do D.
3. Napraw wskaźniki dla nowych półkrawędzi na końcach krawędzi e_1, e_2.
4. Jeśli któraś krawędzi e_1 lub e_2 była krawędzią incydentną dla jakiegoś wierzchołka, ustaw odpowiednie wskaźniki dla nowych półkrawędzi.
5. Połącz odpowiednie półkrawędzie wokół punktu przecięcia w kolejności wyznaczonej przez współczynnik nachylenia krawędzi wraz z informacją, w której ćwiartce układu współrzędnych znajduje się pólkrawędź.

Algorytm Handle-Intersection-B

Pseudokod algorytmu naprawiającego D, gdy krawędź e przechodzi przez punkt p.

Handle-Intersection-B(D, p, e)

1. Podziel krawędź e na dwie części względem punktu p, utwórz dla każdej nowej krawędzi po dwie półkrawędzie i połącz je w bliźniacze pary (pole *twinEdge*).
2. Napraw wskaźniki dla nowych półkrawędzi na końcach krawędzi e.
3. Dodaj do D nowe półkrawędzie, mające początek w wierzchołku p.
4. Jeżeli któraś półkrawędź krawędzi e była półkrawędzią incydentną dla jakiegoś wierzchołka końcowego, to napraw ten wskaźnik.
5. Znajdź odpowiednie półkrawędzie incydentne do wierzchołka p, między które zostaną podpięte nowe półkrawędzie.

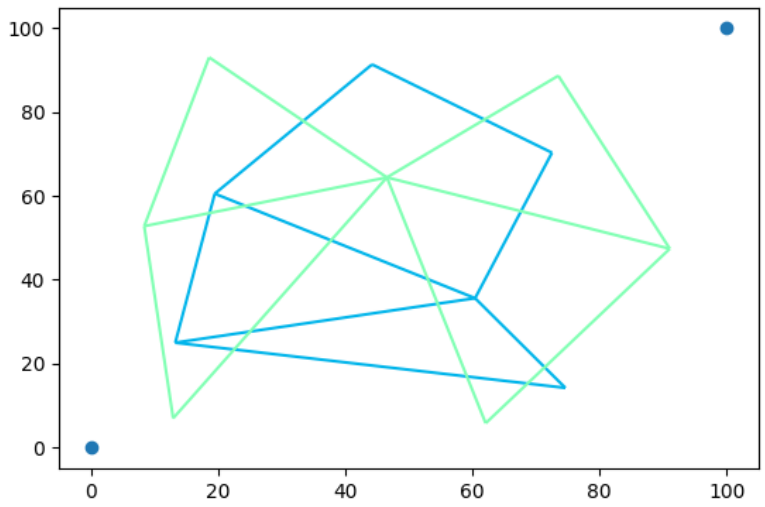
Algorytm Fix-Faces

Pseudokod algorytmu naprawiającego ściany w D.

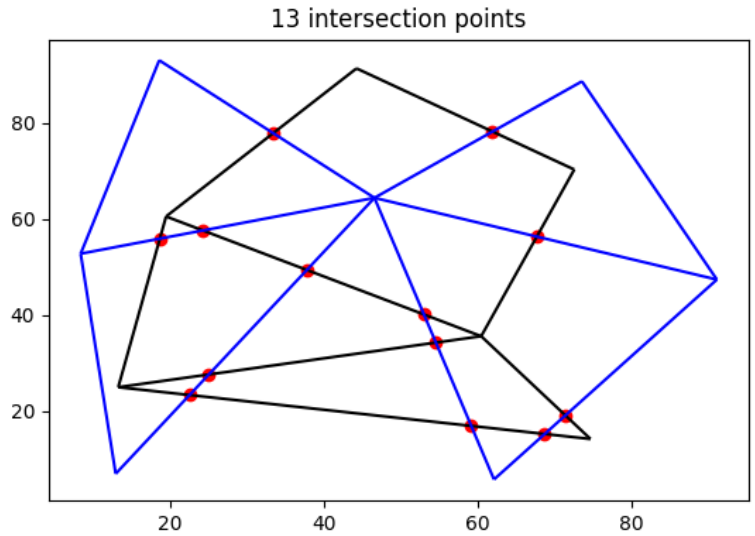
Fix-Faces(D)

1. Przejdź algorytmem DFS po wszystkich cyklach i zaindeksuj każdy cykl.
2. Zainicjalizuj graf nieskierowany G, w którym wierzchołki to znalezione cykle, a krawędź u-v oznacza, że cykl zewnętrzny u, ograniczający dziurę w ścianie, lub cykl wewnętrzny u leży bezpośrednio na lewo od cyklu zewnętrznego v.
3. Uzupełnij krawędzie w grafie G, ponownie zamiatając wierzchołki z D.
4. Na podstawie grafu G utwórz rekordy ścian i połącz dziury z cyklami ograniczającymi ściany.
5. Ponownie przejdź algorytmem DFS po wszystkich cyklach i uzupełnij informacje o ścianach w przechadzanych cyklach.
6. Dla każdej ściany uzupełnij brakujące informacje na podstawie cykli zawierających tę ścianę.

Wizualizacje 1

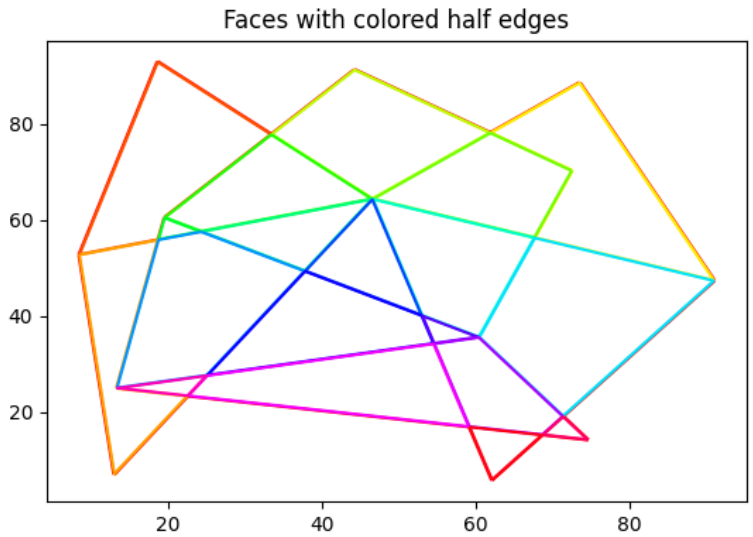


Rysunek 5: Wprowadzone dwa podziały płaszczyzny.

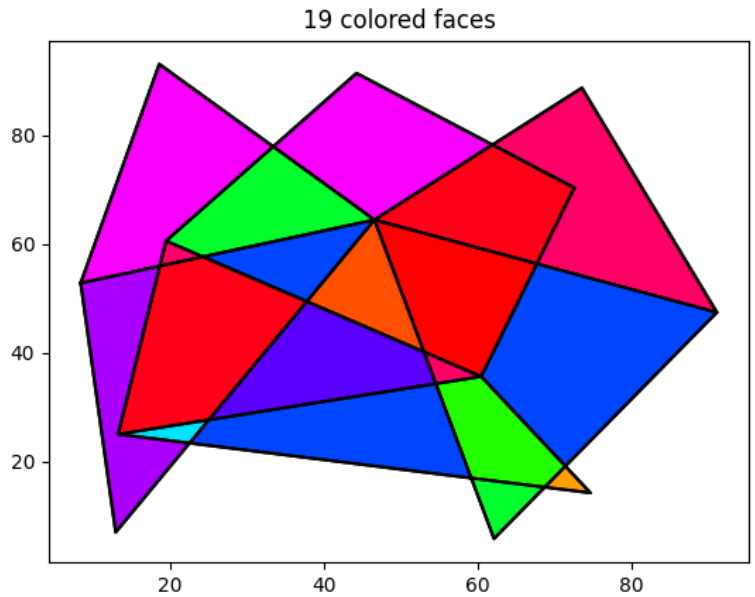


Rysunek 6: Znalezione punkty przecięć z rys. 5.

Wizualizacje 2

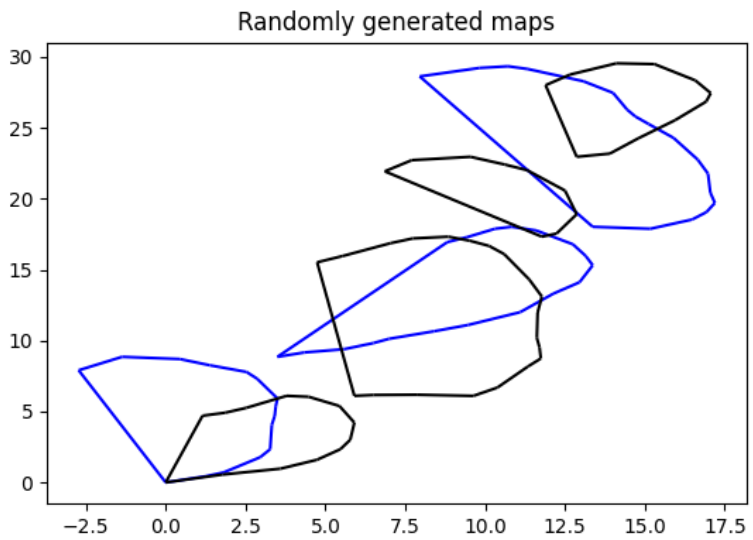


Rysunek 7: Oznaczone kolorami półkrawędzie dla ścian z rys. 5.

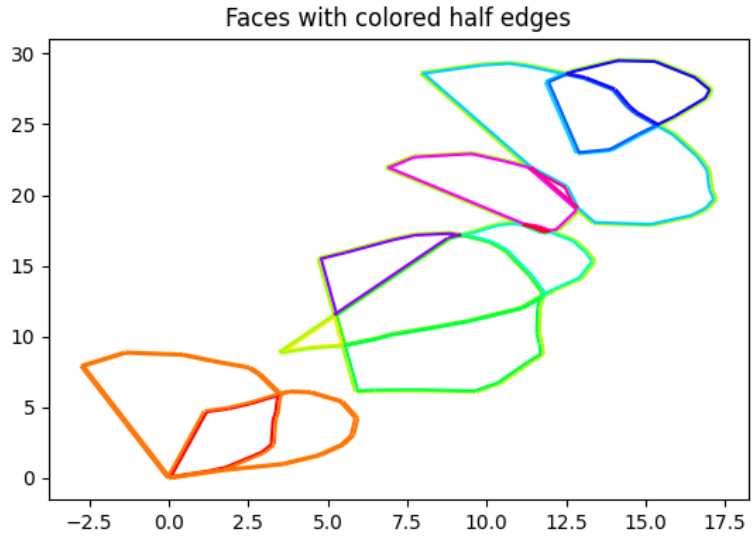


Rysunek 8: 19 znalezionych ścian dla podziału z rys. 5.

Wizualizacje 3



Rysunek 9: Losowo zadane podziały płaszczyzny.



Rysunek 10: Pokolorowanie półkrawędzi przystających do ścian z rys. 9.

Bibliografia

[1] Berg et al. (2000) *Computational Geometry: Algorithms and Applications*, Springer.