# Geometria obliczeniowa

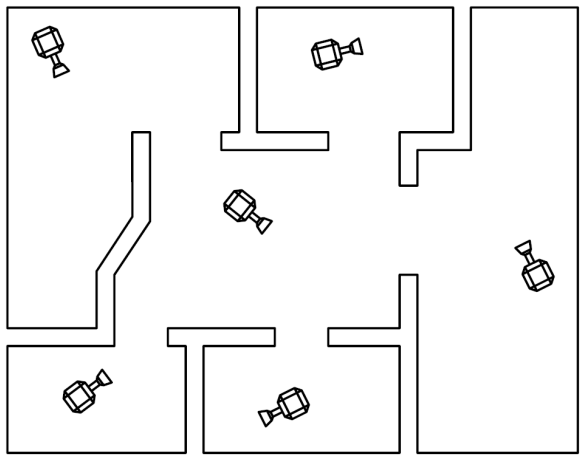
Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

# Problem galerii sztuki

Załóżmy, że chcemy rozmieścić w galerii sztuki kamery. Ponieważ łatwiej jest skupić uwagę na kilku ekranach niż na wielu, więc chcemy aby liczba kamer była tak mała, jak to możliwe. Nie możemy mieć też zbyt mało kamer, ponieważ każdy fragment galerii musi być widoczny dla co najmniej jednej z nich.

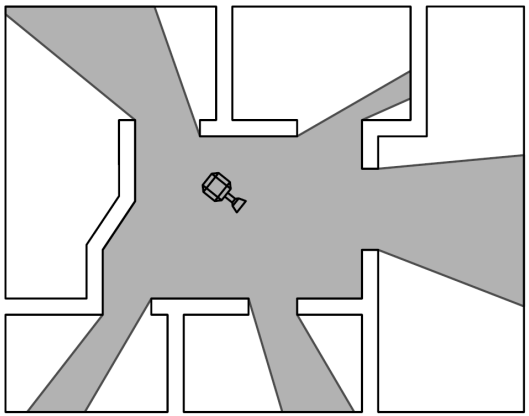
Jest to tak zwany problem galerii sztuki. Został on sformułowany w 1973 r. przez Victora Klee.



Galeria oczywiście jest trójwymiarowa, ale plan podłogi daje nam wystarczająco dużo informacji, aby ulokować kamery. Zatem galerię będziemy modelować za pomocą wielokąta na płaszczyźnie.

Dodatkowo ograniczymy się do wielokątów prostych (ang. simple polygons), tzn. obszarów otoczonych przez pojedynczy, domknięty, wielokątny łańcuch, który nie przecina się ze sobą. Czyli nie dopuszczamy obszarów z dziurami.

Kamera widzi te punkty w wielokącie, z którymi można ją połączyć otwartym odcinkiem leżącym we wnętrzu wielokąta.



Liczbę kamer dla danego wielokąta będziemy wyrażać w zależności od liczby wierzchołków wielokąta.

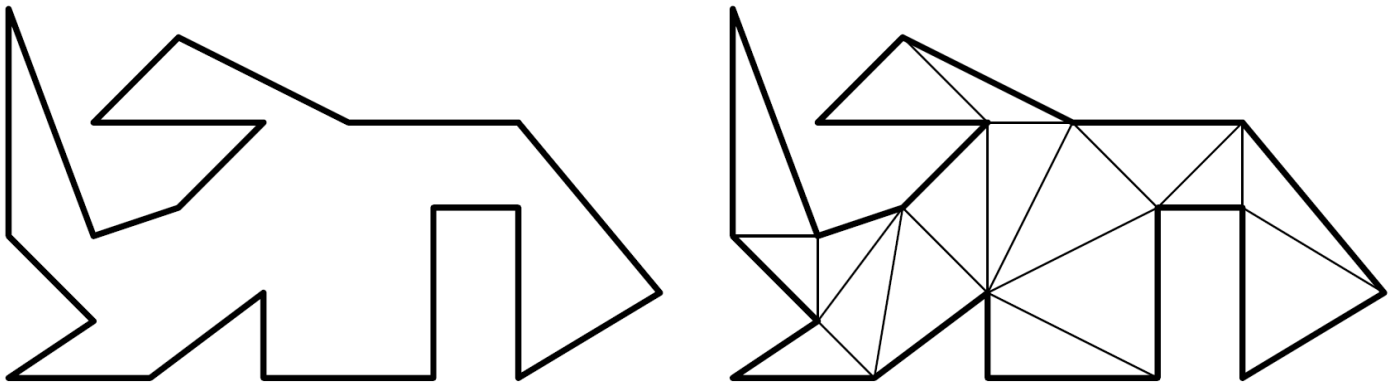
Nawet gdy dwa wielokąty mają taką samą liczbę wierzchołków to jeden może być łatwiejszy do ochrony niż drugi, np. wielokąt wypukły można zawsze ochronić za pomocą jednej kamery.

Aby uniknąć niespodzianek będziemy rozpatrywać przypadek pesymistyczny.

Dobrze by było znaleźć minimalną liczbę kamer dla danego wielokąta, ale niestety jest to problem NP-trudny.

Niech $\mathcal{P}$ będzie wielokątem prostym o $n$ wierzchołkach. Będziemy rozkładać $\mathcal{P}$ na kawałki, które są łatwe do ochrony czyli trójkąty.

Robimy to rysując przekątne między parami wierzchołków. Przekątna jest otwartym odcinkiem, który łączy dwa wierzchołki wielokąta $\mathcal{P}$ i leży we wnętrzu $\mathcal{P}$.



Rozkład wielokąta na trójkąty przez maksymalny zbiór nieprzecinających się przekątnych nazywamy triangulacją wielokąta (ang. triangulation).

Triangulacje nie są zwykle jednoznaczne.

Kiedy mamy już triangulację $\mathcal{T}_\mathcal{P}$ wielokąta $\mathcal{P}$ możemy umieścić kamerę w każdym trójkącie.

Triangulacje nie są zwykle jednoznaczne.

Kiedy mamy już triangulację $\mathcal{T}_\mathcal{P}$ wielokąta $\mathcal{P}$ możemy umieścić kamerę w każdym trójkącie.

Ale czy triangulacja zawsze istnieje? Odpowiedź jest twierdząca.

**Twierdzenie.** Każdy prosty wielokąt można striangulować, a każda triangulacja prostego wielokąta o $n$ wierzchołkach składa się z dokładnie $n - 2$ trójkątów.

Z twierdzenia wynika, że każdy prosty wielokąt o $n$ wierzchołkach można ochronić za pomocą $n - 2$ kamer.

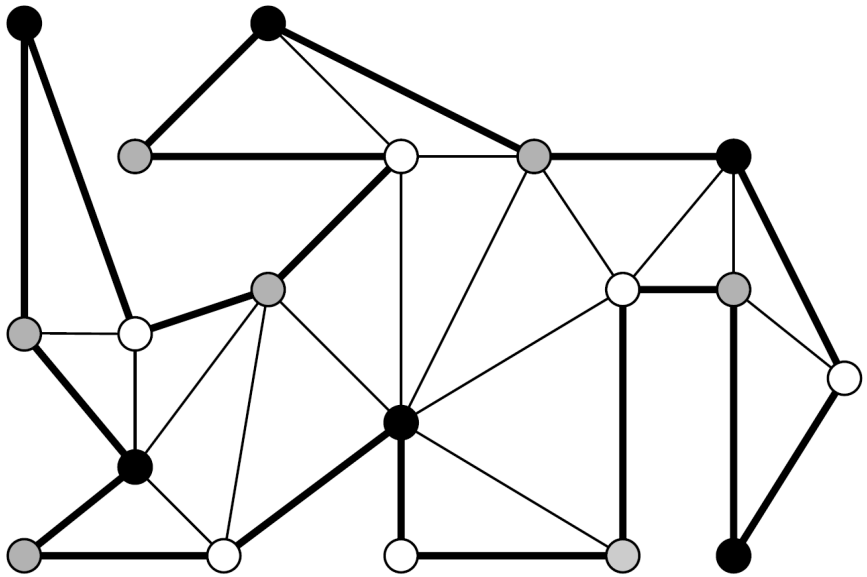
Zauważmy jednak, że umieszczenie kamery na przekątnej sprawia, że ochrania ona dwa trójkąty. Zatem umieszczając kamery na dobrze wybranych przekątnych może zmniejszyć liczbę kamer do około $n/2$.

Umieszczenie kamer w wierzchołkach jest jeszcze lepsze, ponieważ wierzchołek może być incydentny z wieloma trójkątami, a kamera w tym wierzchołku strzeże je wszystkie.

W jaki sposób wybrać wierzchołki, w których umieścić kamery?

Niech $\mathcal{T}_\mathcal{P}$ będzie triangulacją wielokąta $\mathcal{P}$. Wybierzemy podzbiór wierzchołków $\mathcal{P}$ taki, że każdy trójkąt w $\mathcal{T}_\mathcal{P}$ ma co najmniej jeden wybrany wierzchołek i umieścimy kamery w wybranych wierzchołkach.

Aby wybrać ten podzbiór przydzielamy każdemu wierzchołkowi z $\mathcal{P}$ kolor: biały, szary lub czarny. Będziemy tak kolorować, aby dowolne dwa wierzchołki połączone krawędzią lub przekątną miały różne kolory. Nazywa się to 3-kolorowaniem striangulowanego wielokąta.



Mając pokolorowane wierzchołki wielokąta wybieramy najmniejszy, jednobarwny zbiór wierzchołków i umieszczamy w nich kamery. Będzie ich co najwyżej $\lfloor n/3 \rfloor$.

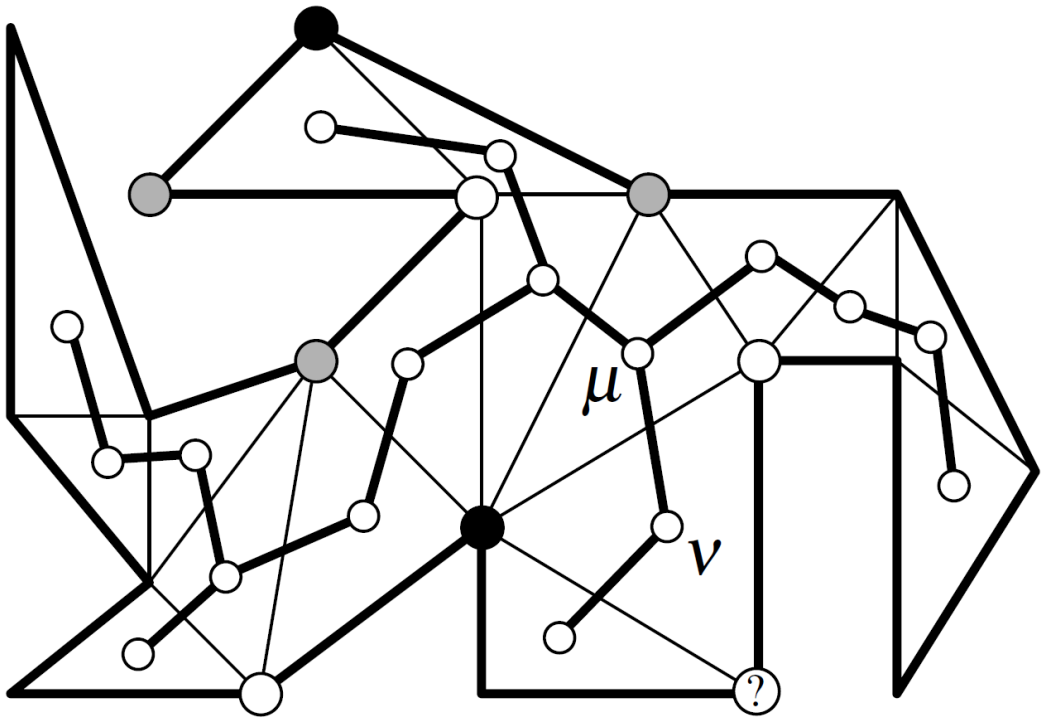
Mając pokolorowane wierzchołki wielokąta wybieramy najmniejszy, jednobarwny zbiór wierzchołków i umieszczamy w nich kamery. Będzie ich co najwyżej $\lfloor n/3 \rfloor$.

Czy 3-kolorowanie zawsze istnieje? Odpowiedź ponownie jest twierdząca.

Aby się o tym przekonać spójrzmy na tak zwany graf dualny (ang. dual graph) triangulacji $\mathcal{T}_{\mathcal{P}}$.

Graf dualny $G(\mathcal{T}_\mathcal{P})$ ma węzeł dla każdego trójkąta w $\mathcal{T}_\mathcal{P}$. Niech $t(v)$ oznacza trójkąt odpowiadający węzłowi $v$.

Między dwoma węzłami $v$ i $\mu$ istnieje krawędź gdy $t(v)$ i $t(\mu)$ rozdziela przekątna. Zatem krawędź w $G(\mathcal{T}_\mathcal{P})$ odpowiada przekątnej w $\mathcal{T}_\mathcal{P}$.

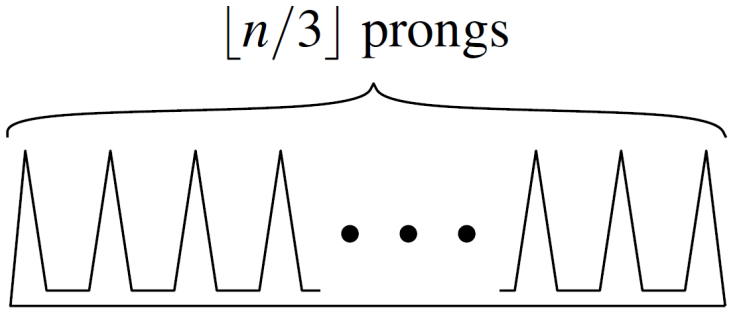


Graf dualny jest drzewem, więc do jego 3-kolorowania można wykorzystać przeszukiwanie grafu w głąb.

Zaczynamy od dowolnego wierzchołka grafu $G(\mathcal{T}_\mathcal{P})$. Trzy wierzchołki odpowiedniego trójkąta kolorujemy na biało, szaro i czarno. Przypuśćmy, że osiągamy węzeł $v$ przychodząc z węzła $\mu$. Trójkąty $t(v)$ i $t(\mu)$ rozdziela przekątna.
Ponieważ wierzchołki $t(\mu)$ zostały już pomalowane, więc tylko jeden wierzchołek $t(v)$ został do pokolorowania. Jest tylko jeden kolor, który pozostał dla naszego wierzchołka.

**Twierdzenie (o galerii sztuki).** Dla prostego wielokąta o $n$ wierzchołkach $\lfloor n/3 \rfloor$ kamer jest czasem koniecznych i zawsze wystarczających, aby każdy punkt wielokąta mógł być widoczny z co najmniej jednej kamery.

Przykład wielokąta, dla którego konieczne jest użycie $\lfloor n/3 \rfloor$ kamer.

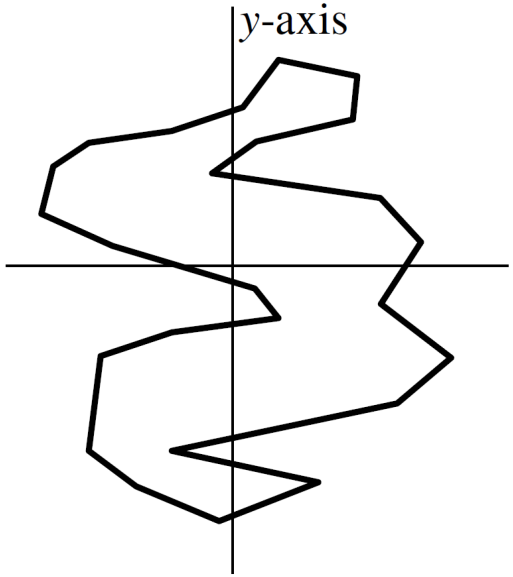


# Podział wielokąta na części monotoniczne

W celu triangulacji wielokąta prostego będziemy go dzielić na części, które są łatwiejsze do striangulowania. Pomimo, że wielokąty wypukłe są bardzo proste do striangulowania, to znalezienie podziału wielokąta na wielokąty wypukłe jest równie trudne co sama triangulacja naszego wielokąta prostego.

Prosty wielokąt nazywamy monotonicznym względem prostej $\ell$, jeśli dla dowolnej prostej $\ell'$ prostopadłej do $\ell$ przecięcie wielokąta z $\ell'$ jest spójne.

Wielokąt, który jest monotoniczny względem osi $y$ nazywamy $y$-monotonicznym.



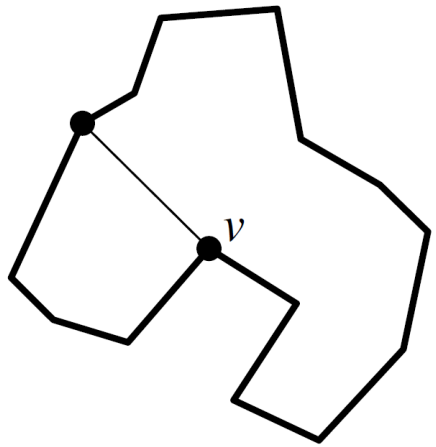

Wielokąt $y$-monotoniczny ma własność: jeśli idziemy z najwyższego do najniższego wierzchołka wzdłuż jego lewego (lub prawego) łańcucha brzegu, to zawsze poruszamy się w dół lub poziomo, ale nigdy w górę.

Jak rozłożyć wielokąt prosty na $y$-monotoniczne części?

Załóżmy, że idziemy z najwyższego do najniższego wierzchołka wielokąta prostego po lewym lub prawym brzegu. Wierzchołek gdzie kierunek zmienia się z kierunku w dół na kierunek do góry lub z kierunku do góry na kierunek w dół nazywamy wierzchołkiem zwrotu (ang. turn vertex).

Aby podzielić wielokąt prosty na $y$-monotoniczne części musimy wyeliminować wierzchołki zwrotu. W tym celu będziemy dodawać przekątne.

Jeśli w wierzchołku zwrotu $v$ obie incydentne krawędzie idą w dół, a wnętrze wielokąta lokalnie leży powyżej $v$, to musimy wybrać przekątną, która wychodzi z $v$ do góry.



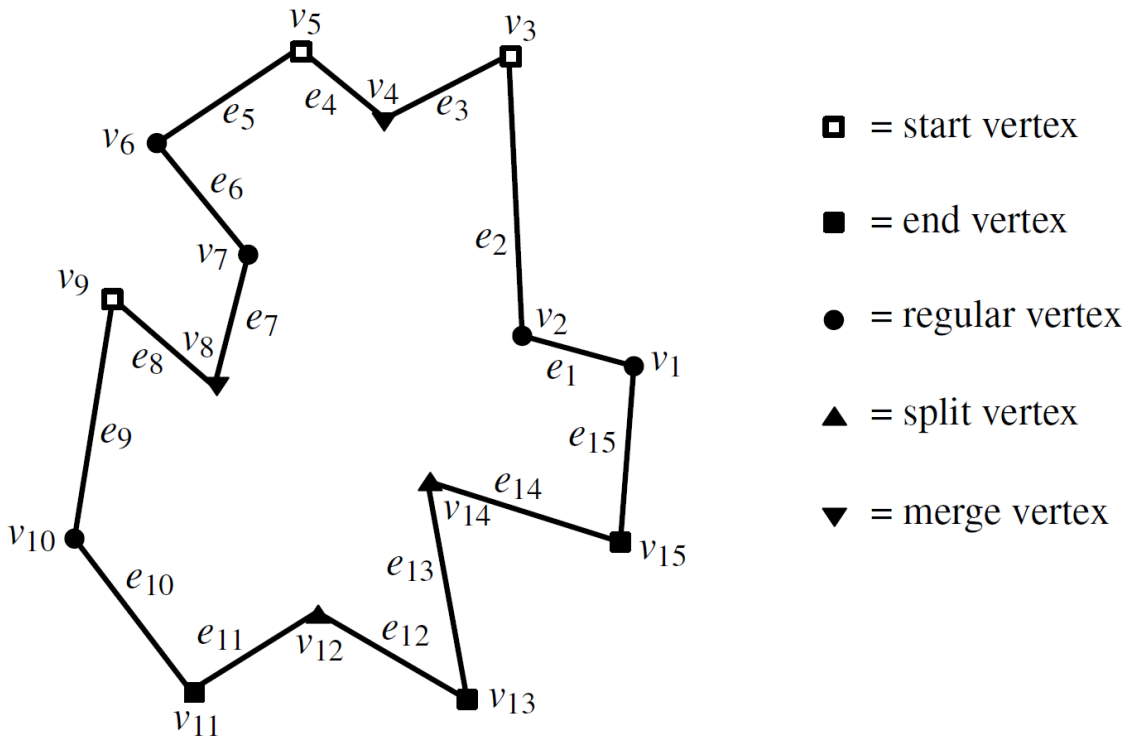

Jeśli obie incydentne krawędzie wierzchołka zwrotu wychodzą w górę, a wnętrze wielokąta lokalnie leży poniżej niego, to wybieramy przekątną wychodzącą w dół.

Punkt $p$ jest poniżej punktu $q$, gdy $p_y < q_y$ lub ($p_y = q_y$ i $p_x > q_x$).
Punkt $p$ jest powyżej punktu $q$, gdy $p_y > q_y$ lub ($p_y = q_y$ i $p_x < q_x$).

W wielokącie wyróżniamy pięć rodzajów wierzchołków:

- wierzchołek początkowy (ang. start vertex) jest to wierzchołek, którego dwaj sąsiedzi leżą poniżej niego, a kąt wewnętrzny w nim jest mniejszy od $\pi$,
- wierzchołek dzielący (ang. split vertex) jest to wierzchołek, którego dwaj sąsiedzi leżą poniżej niego, a kąt wewnętrzny w nim jest większy od $\pi$,
- wierzchołek końcowy (ang. end vertex) jest to wierzchołek, którego dwaj sąsiedzi leżą powyżej niego, a kąt wewnętrzny w nim jest mniejszy od $\pi$,

- wierzchołek łączący (ang. merge vertex) jest to wierzchołek, którego dwaj sąsiedzi leżą powyżej niego, a kąt wewnętrzny w nim jest większy od $\pi$,
- wierzchołek prawidłowy (ang. regular vertex) jest to wierzchołek, który nie jest żadnym z pozostałych rodzajów.



Z każdego wierzchołka dzielącego będziemy dodawać przekątne w górę, a z wierzchołka łączącego w dół. Do tego celu wykorzystamy technikę zamiatania.

Niech $v_1, v_2, \ldots, v_n$ będzie numeracją wierzchołków wielokąta prostego $\mathcal{P}$ w kierunku przeciwnym do ruchu wskazówek zegara oraz niech $e_1, e_2, \ldots, e_n$ będzie zbiorem krawędzi $\mathcal{P}$, gdzie $e_i = \overline{v_i v_{i+1}}$ dla $1 \leqslant i < n$ i $e_n = \overline{v_n v_1}$.

Algorytm zamiatania będzie przesuwać miotłę $\ell$ w dół po płaszczyźnie. Punktami zdarzeń będą wierzchołki wielokąta. Podczas zamiatania nie będą tworzone nowe punkty zdarzeń.

Kolejka zdarzeń $\mathcal{Q}$ będzie kolejką priorytetową, gdzie priorytetem wierzchołka jest jego współrzędna $y$. Jeśli dwa wierzchołki mają taką samą współrzędną $y$, to ten bardziej na lewo ma wyższy priorytet.

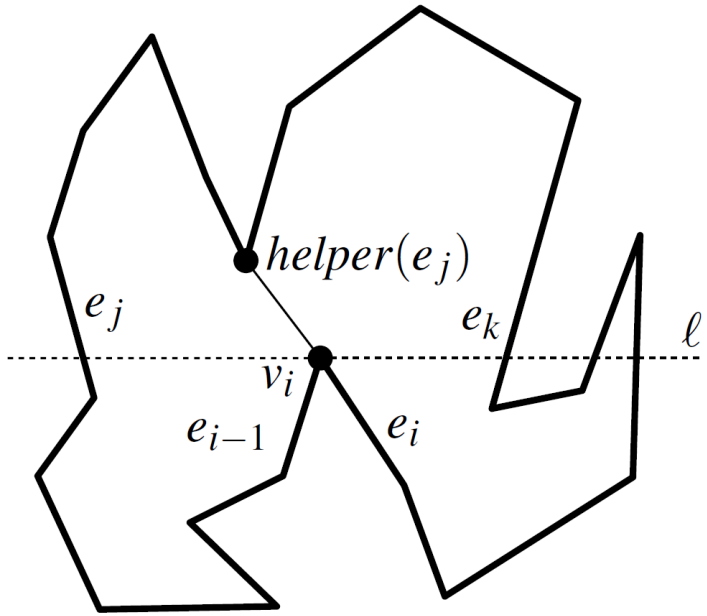
Ponieważ podczas zamiatania nie będziemy tworzyć nowych punktów zdarzeń, więc zamiast kolejki priorytetowej możemy użyć posortowanej tablicy, w której porządek ustalony jest tak jak priorytet w kolejce priorytetowej.

Kolejka zdarzeń $\mathcal{Q}$ będzie kolejką priorytetową, gdzie priorytetem wierzchołka jest jego współrzędna $y$. Jeśli dwa wierzchołki mają taką samą współrzędną $y$, to ten bardziej na lewo ma wyższy priorytet.

Ponieważ podczas zamiatania nie będziemy tworzyć nowych punktów zdarzeń, więc zamiast kolejki priorytetowej możemy użyć posortowanej tablicy, w której porządek ustalony jest tak jak priorytet w kolejce priorytetowej.

Przypuśćmy, że miotła dociera do wierzchołka dzielącego $v_i$. Niech $e_j$ będzie krawędzią na miotle bezpośrednio na lewo od $v_i$, a $e_k$ krawędzią na miotle bezpośrednio na prawo od $v_i$.

Wierzchołek $v_i$ możemy połączyć z najniższym wierzchołkiem między $e_j$ i $e_k$, leżącym powyżej $v_i$. Jeśli nie ma takiego wierzchołka, to możemy połączyć $v_i$ z górnym końcem $e_j$ lub z górnym końcem $e_k$. Wierzchołek ten nazywamy pomocnikiem krawędzi $e_j$ (ang. helper) i oznaczamy $helper(e_j)$.



Dokładna definicja pomocnika jest następująca: jest to najniższy wierzchołek powyżej miotły taki, że poziomy odcinek łączący ten wierzchołek z $e_j$ leży wewnątrz $\mathcal{P}$.

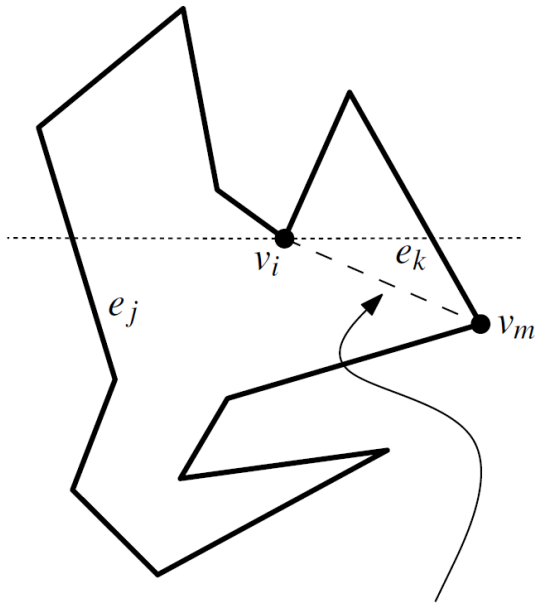
Dokładna definicja pomocnika jest następująca: jest to najniższy wierzchołek powyżej miotły taki, że poziomy odcinek łączący ten wierzchołek z $e_j$ leży wewnątrz $\mathcal{P}$.

Przypuśćmy teraz, że miotła spotyka wierzchołek łączący $v_i$. Ponownie niech $e_j$ i $e_k$ będą odpowiednio krawędziami na miotle bezpośrednio na lewo i prawo od $v_i$.

Chcemy połączyć $v_i$ z najwyższym wierzchołkiem poniżej miotły między $e_j$ i $e_k$. Oczywiście nie znamy najwyższego wierzchołka poniżej miotły, gdy spotykamy $v_i$. Ale możemy taki wierzchołek znaleźć później.

Gdy natkniemy się na $v_m$, który zastąpi $v_i$ w roli pomocnika $e_j$, to jest to wierzchołek, którego szukaliśmy.

Ilekroć będziemy zastępować pomocnika pewnej krawędzi sprawdzamy czy stary pomocnik jest wierzchołkiem łączącym. Jeśli tak, to dodajemy przekątną między starym i nowym pomocnikiem. Może się zdarzyć, że pomocnik nigdy nie jest zastępowany poniżej $v_i$. W tym przypadku możemy połączyć $v_i$ z dolnym końcem $e_j$.



W naszym podejściu musimy mieć możliwość znajdowania krawędzi na lewo od każdego wierzchołka. Do tego celu wykorzystujemy dynamiczne drzewo BST $\mathcal{T}$, w którym w liściach pamiętamy krawędzie $\mathcal{P}$ przecinające miotłę. Porządek liści od lewej do prawej odpowiada porządkowi krawędzi od lewej do prawej. Z każdą krawędzią w $\mathcal{T}$ pamiętamy jej pomocnika.

Drzewo $\mathcal{T}$ i pomocnicy pamiętani z krawędziami tworzą stan algorytmu.

Listę wierzchołków i krawędzi wielokąta $\mathcal{P}$ pamiętamy jako podwójnie łączoną listę krawędzi $\mathcal{D}$. Obliczone przekątne będą dodawane do tej listy.

**Algorithm** MAKEMONOTONE($\mathcal{P}$)
*Input.* A simple polygon $\mathcal{P}$ stored in a doubly-connected edge list $\mathcal{D}$.
*Output.* A partitioning of $\mathcal{P}$ into monotone subpolygons, stored in $\mathcal{D}$.

1. Construct a priority queue $\mathcal{Q}$ on the vertices of $\mathcal{P}$, using their $y$-coordinates as priority. If two points have the same $y$-coordinate, the one with smaller $x$-coordinate has higher priority.
2. Initialize an empty binary search tree $\mathcal{T}$.
3. **while** $\mathcal{Q}$ is not empty
4. **do** Remove the vertex $v_i$ with the highest priority from $\mathcal{Q}$.
5. Call the appropriate procedure to handle the vertex, depending on its type.

**Algorithm** MAKEMONOTONE($\mathcal{P}$)
*Input.* A simple polygon $\mathcal{P}$ stored in a doubly-connected edge list $\mathcal{D}$.
*Output.* A partitioning of $\mathcal{P}$ into monotone subpolygons, stored in $\mathcal{D}$.
1. Construct a priority queue $\mathcal{Q}$ on the vertices of $\mathcal{P}$, using their $y$-coordinates as priority. If two points have the same $y$-coordinate, the one with smaller $x$-coordinate has higher priority.
2. Initialize an empty binary search tree $\mathcal{T}$.
3. **while** $\mathcal{Q}$ is not empty
4. **do** Remove the vertex $v_i$ with the highest priority from $\mathcal{Q}$.
5. Call the appropriate procedure to handle the vertex, depending on its type.

HANDLESTARTVERTEX($v_i$)
1. Insert $e_i$ in $\mathcal{T}$ and set $helper(e_i)$ to $v_i$.

HANDLEENDVERTEX($v_i$)
1. **if** $helper(e_{i-1})$ is a merge vertex
2. **then** Insert the diagonal connecting $v_i$ to $helper(e_{i-1})$ in $\mathcal{D}$.
3. Delete $e_{i-1}$ from $\mathcal{T}$.

HANDLESPLITVERTEX($v_i$)

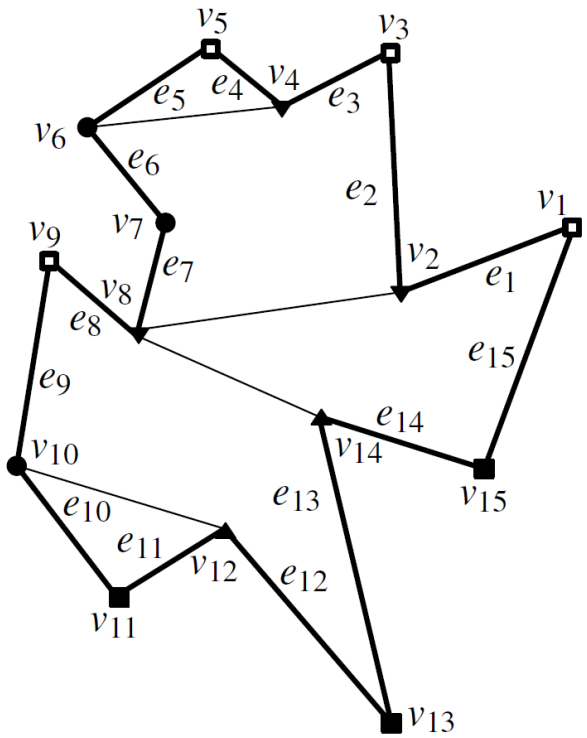
1. Search in $\mathcal{T}$ to find the edge $e_j$ directly left of $v_i$.
2. Insert the diagonal connecting $v_i$ to $helper(e_j)$ in $\mathcal{D}$.
3. $helper(e_j) \leftarrow v_i$
4. Insert $e_i$ in $\mathcal{T}$ and set $helper(e_i)$ to $v_i$.

HANDLEMERGEVERTEX($v_i$)

1. **if** $helper(e_{i-1})$ is a merge vertex
2. **then** Insert the diagonal connecting $v_i$ to $helper(e_{i-1})$ in $\mathcal{D}$.
3. Delete $e_{i-1}$ from $\mathcal{T}$.
4. Search in $\mathcal{T}$ to find the edge $e_j$ directly left of $v_i$.
5. **if** $helper(e_j)$ is a merge vertex
6. **then** Insert the diagonal connecting $v_i$ to $helper(e_j)$ in $\mathcal{D}$.
7. $helper(e_j) \leftarrow v_i$

HANDLEREGULARVERTEX($v_i$)

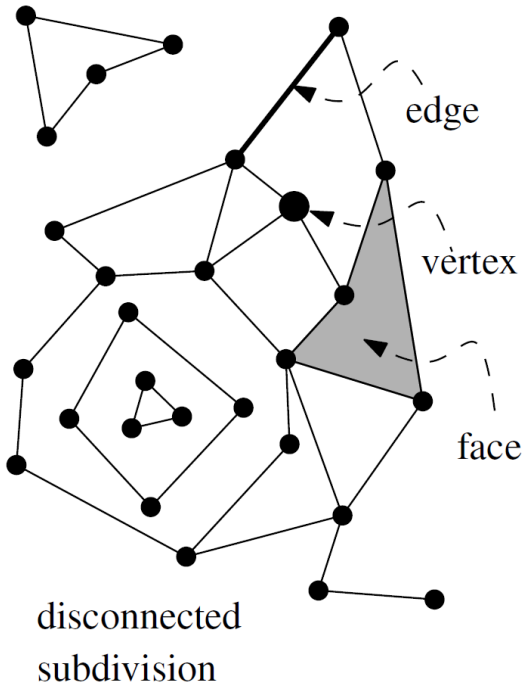
1. **if** the interior of $\mathcal{P}$ lies to the right of $v_i$
2. **then if** $helper(e_{i-1})$ is a merge vertex
3. **then** Insert the diagonal connecting $v_i$ to $helper(e_{i-1})$ in $\mathcal{D}$.
4. Delete $e_{i-1}$ from $\mathcal{T}$.
5. Insert $e_i$ in $\mathcal{T}$ and set $helper(e_i)$ to $v_i$.
6. **else** Search in $\mathcal{T}$ to find the edge $e_j$ directly left of $v_i$.
7. **if** $helper(e_j)$ is a merge vertex
8. **then** Insert the diagonal connecting $v_i$ to $helper(e_j)$ in $\mathcal{D}$.
9. $helper(e_j) \leftarrow v_i$




Prosty wielokąt o $n$ wierzchołkach można podzielić w czasie $\mathcal{O}(n \log n)$ na $y$-monotoniczne wielokąty za pomocą algorytmu używającego $\mathcal{O}(n)$ pamięci.

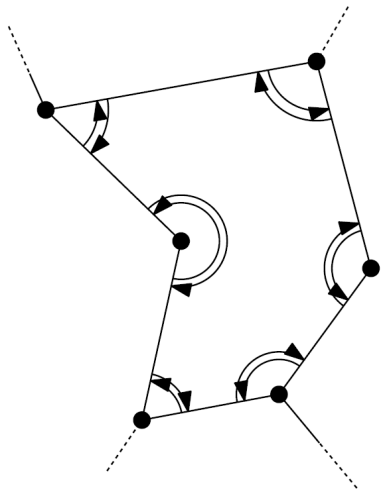
# Podwójnie łączona lista krawędzi

Do przechowywania podziałów planarnych możemy wykorzystać podwójnie łączona lista krawędzi (ang. doubly-connected edge list).

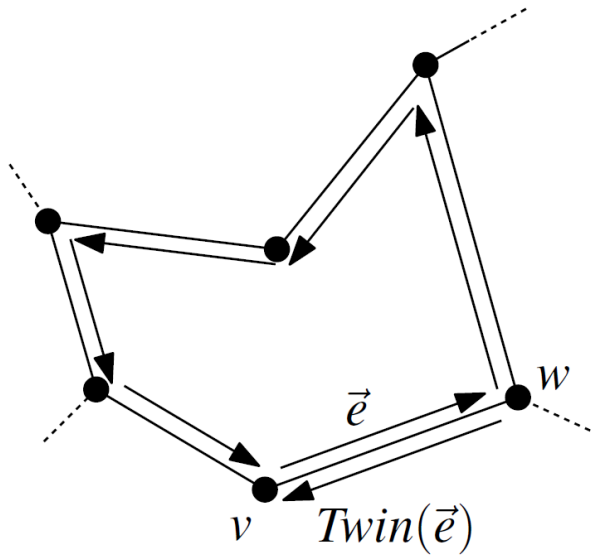


Lista ta zawiera rekord dla każdej ściany, krawędzi i wierzchołka podziału. Poza informacjami geometrycznymi i topologicznymi każdy rekord może pamiętać dodatkowe informacje. Dodatkowa informacja nazywana jest również informacją o własnościach (ang. attribute information).

Każda krawędź będzie zawierać wskaźnik na następną krawędź w ścianie. Aby móc przejść w odwrotną stronę będzie również zawierać wskaźnik na poprzednią krawędź w ścianie.



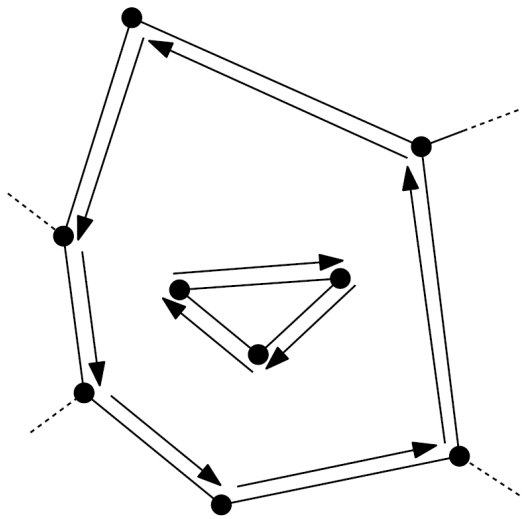
Krawędź należy do co najwyżej dwóch ścian, więc będziemy potrzebować dla każdej krawędzi dwóch par wskaźników. Wygodniej jest reprezentować krawędzie przez dwie półkrawędzie (ang. half-edges). Dwie półkrawędzie dla jednej krawędzi nazywamy bliźniakami (ang. twins).



Półkrawędzie orientujemy w ten sposób, że ściana, którą ogranicza leży po jej lewej stronie dla obserwatora kroczącego wzdłuż krawędzi. W ten sposób obchodzimy ścianę przeciwnie do ruchu wskazówek zegara.

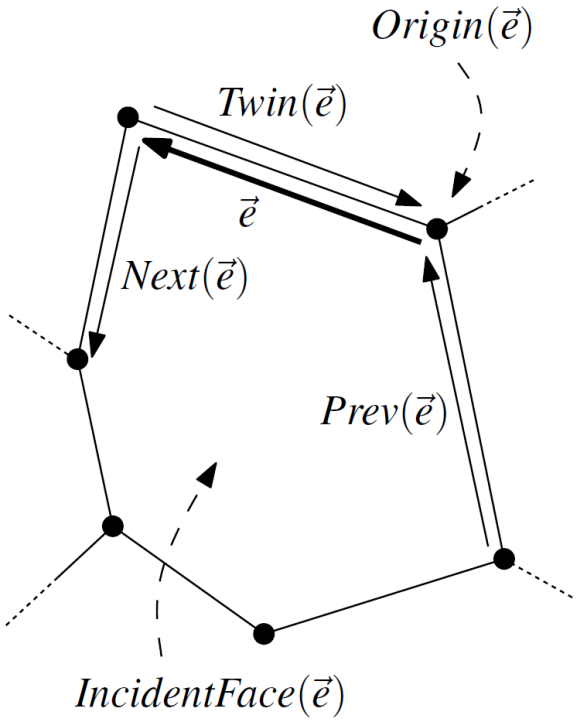
Półkrawędzie są zorientowane, zatem mają początek i koniec.

W przypadku dziur w ścianie chcemy, aby i one leżały po lewej stronie dla obserwatora, więc orientujemy je tak, że obchodzimy ścianę zgodnie z ruchem wskazówek zegara.



Podwójnie łączona lista krawędzi składa się z:

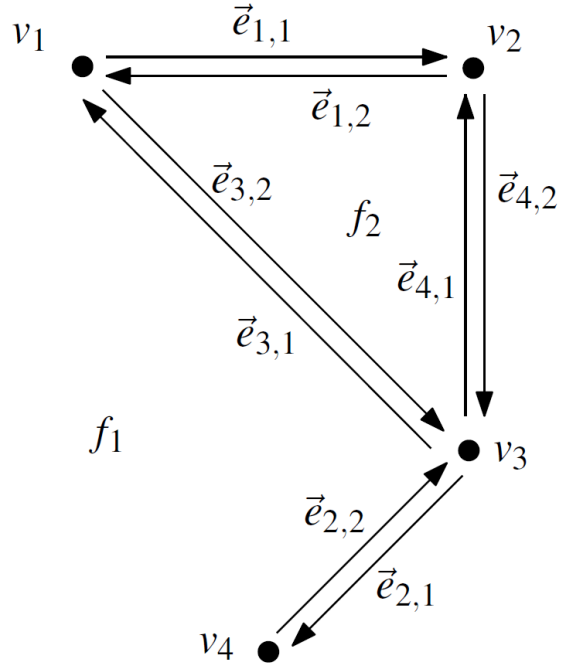
- rekordu wierzchołka – współrzędne wierzchołka $v$, wskaźnik na krawędź incydentną, która ma $v$ jako początek,
- rekordu ściany – wskaźnik do półkrawędzi na jej zewnętrznym brzegu (składowa zewnętrzna), listę, która dla każdej dziury zawiera wskaźnik do półkrawędzi na jej brzegu,
- rekordu półkrawędzi – wskaźnik do jej początku, wskaźnik do półkrawędzi bliźniaczej, wskaźnik do ściany incydentnej, wskaźnik do następnej i poprzedniej półkrawędzi na brzegu ściany incydentnej.



| Vertex | Coordinates | IncidentEdge |
|---|---|---|
| $v_1$ | $(0,4)$ | $\vec{e}_{1,1}$ |
| $v_2$ | $(2,4)$ | $\vec{e}_{4,2}$ |
| $v_3$ | $(2,2)$ | $\vec{e}_{2,1}$ |
| $v_4$ | $(1,1)$ | $\vec{e}_{2,2}$ |

| Face | OuterComponent | InnerComponents |
|---|---|---|
| $f_1$ | **nil** | $\vec{e}_{1,1}$ |
| $f_2$ | $\vec{e}_{4,1}$ | **nil** |

| Half-edge | Origin | Twin | IncidentFace | Next | Prev |
|---|---|---|---|---|---|
| $\vec{e}_{1,1}$ | $v_1$ | $\vec{e}_{1,2}$ | $f_1$ | $\vec{e}_{4,2}$ | $\vec{e}_{3,1}$ |
| $\vec{e}_{1,2}$ | $v_2$ | $\vec{e}_{1,1}$ | $f_2$ | $\vec{e}_{3,2}$ | $\vec{e}_{4,1}$ |
| $\vec{e}_{2,1}$ | $v_3$ | $\vec{e}_{2,2}$ | $f_1$ | $\vec{e}_{2,2}$ | $\vec{e}_{4,2}$ |
| $\vec{e}_{2,2}$ | $v_4$ | $\vec{e}_{2,1}$ | $f_1$ | $\vec{e}_{3,1}$ | $\vec{e}_{2,1}$ |
| $\vec{e}_{3,1}$ | $v_3$ | $\vec{e}_{3,2}$ | $f_1$ | $\vec{e}_{1,1}$ | $\vec{e}_{2,2}$ |
| $\vec{e}_{3,2}$ | $v_1$ | $\vec{e}_{3,1}$ | $f_2$ | $\vec{e}_{4,1}$ | $\vec{e}_{1,2}$ |
| $\vec{e}_{4,1}$ | $v_3$ | $\vec{e}_{4,2}$ | $f_2$ | $\vec{e}_{1,2}$ | $\vec{e}_{3,2}$ |
| $\vec{e}_{4,2}$ | $v_2$ | $\vec{e}_{4,1}$ | $f_1$ | $\vec{e}_{2,1}$ | $\vec{e}_{1,1}$ |



# Triangulacja wielokątów monotonicznych

Załóżmy, że wielokąt $\mathcal{P}$ jest ściśle $y$-monotoniczny czyli że jest $y$-monotoniczny i nie zawiera poziomych krawędzi. W takim wielokącie zawsze idziemy w dół, gdy przechodzimy po lewym lub prawym łańcuchu brzegu $\mathcal{P}$.

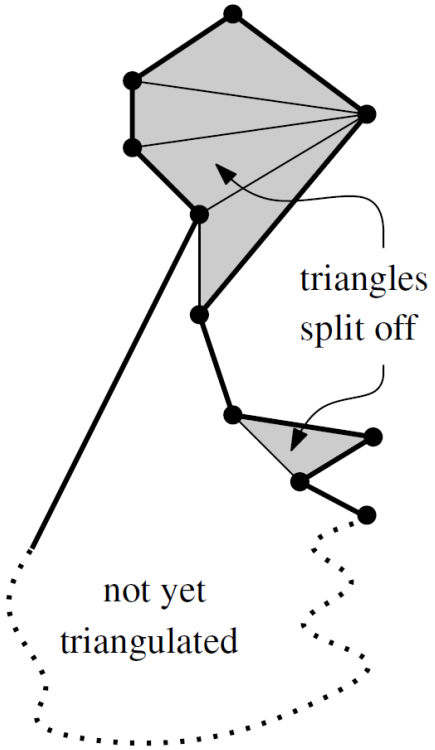
# Triangulacja wielokątów monotonicznych

Załóżmy, że wielokąt $\mathcal{P}$ jest ściśle $y$-monotoniczny czyli że jest $y$-monotoniczny i nie zawiera poziomych krawędzi. W takim wielokącie zawsze idziemy w dół, gdy przechodzimy po lewym lub prawym łańcuchu brzegu $\mathcal{P}$.

Algorytm będzie badać wierzchołki w kolejności malejących współrzędnych $y$. Jeśli dwa wierzchołki mają taką samą współrzędną $y$, to wierzchołek bardziej na lewo jest badany pierwszy.

W algorytmie potrzebować będziemy stosu $\mathcal{S}$. Początkowo stos będzie pusty. W trakcie działania algorytmu będzie zawierać wierzchołki wielokąta, które zostały napotkane, ale mogą jeszcze mieć więcej incydentnych przekątnych.

Gdy badamy wierzchołek dodajemy tak wiele przekątnych z tego wierzchołka do wierzchołków na stosie jako tylko jest to możliwe (podejście zachłanne).



Wierzchołki, które zbadano, ale nie zostały oddzielone (wierzchołki na stosie) są na brzegu tej części $\mathcal{P}$, którą trzeba jeszcze striangulować.

Najniższy z tych wierzchołków, który jest ostatnim napotkanym znajduje się na szczycie stosu. Drugi najniższy jest drugi na stosie itd.

Część $\mathcal{P}$, którą trzeba jeszcze striangulować i leży powyżej ostatniego napotkanego wierzchołka ma kształt lejka odwróconego do góry nogami. Jeden brzeg lejka składa się z części krawędzi wielokąta $\mathcal{P}$, a drugi brzeg jest łańcuchem składającym się z wklęsłych wierzchołków (kąt wewnętrzny wynosi co najmniej $180^\circ$). Tylko najwyższy wierzchołek, który jest na dnie stosu, jest wypukły.

Podczas badania wierzchołka mamy dwa przypadki. Następny badany wierzchołek $v_j$ leży na:

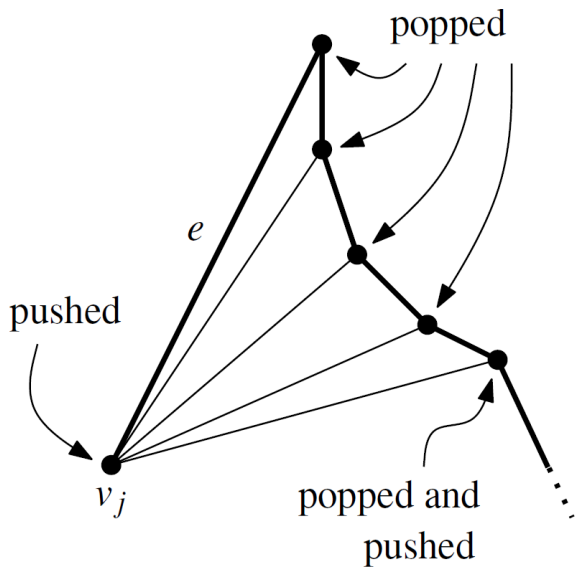
- przeciwległym łańcuchu co wklęsłe wierzchołki ze stosu,
- tym samym łańcuchu co wklęsłe wierzchołki ze stosu.

Podczas badania wierzchołka mamy dwa przypadki. Następny badany wierzchołek $v_j$ leży na:

- przeciwległym łańcuchu co wklęsłe wierzchołki ze stosu,
- tym samym łańcuchu co wklęsłe wierzchołki ze stosu.

W pierwszym przypadku $v_j$ musi być dolnym końcem pojedynczej krawędzi $e$ ograniczającej lejek. Dodajemy przekątne z $v_j$ do wszystkich wierzchołków znajdujących się na stosie, oprócz ostatniego (tego na dnie stosu). Ostatni wierzchołek na stosie jest górnym wierzchołkiem krawędzi $e$.

Wierzchołek $v_j$ oraz wierzchołek, który był na szczycie stosu (w tej chwili są one połączone krawędzią) pozostają częścią jeszcze niestriangulowanego wielokąta, więc wkładamy je na stos.

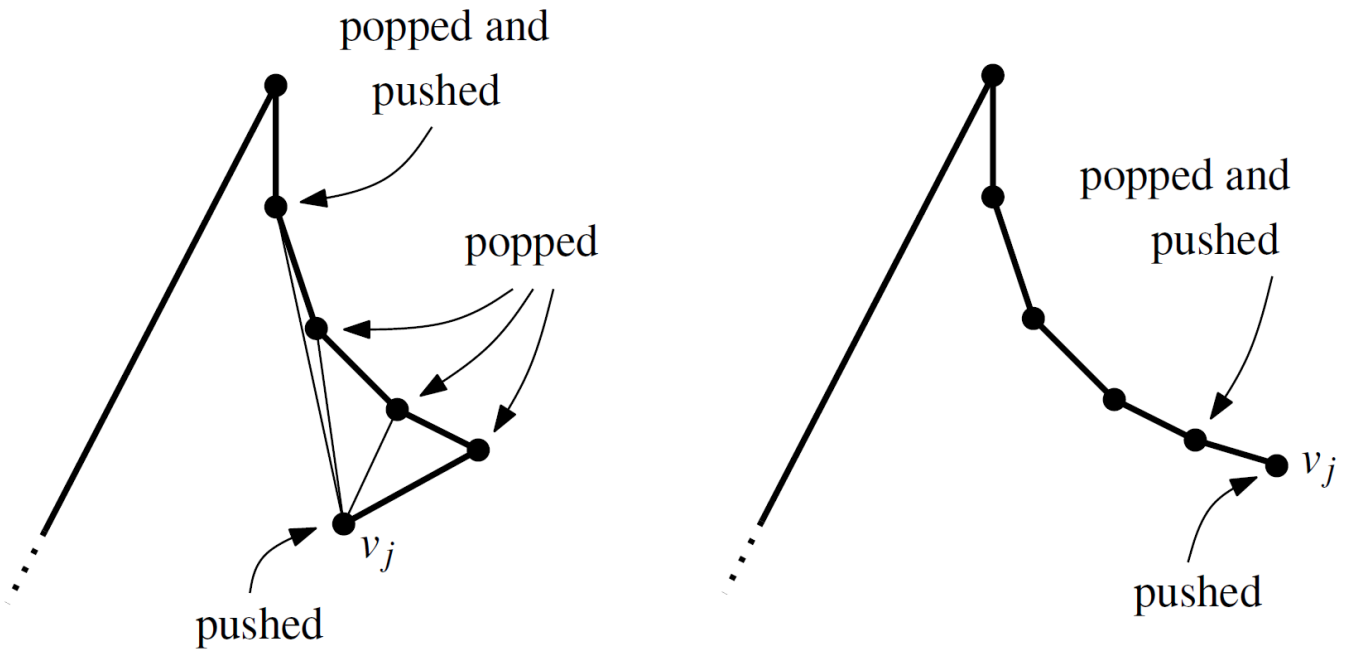


W drugim przypadku, tzn. $v_j$ jest na tym samym łańcuchu co wklęsłe wierzchołki ze stosu, możemy nie być w stanie poprowadzić przekątnych z $v_j$ do wszystkich wierzchołków ze stosu.

Mimo to, wszystkie wierzchołki, z którymi możemy połączyć $v_j$ są kolejne i znajdują się na szczycie stosu. Zatem zdejmujemy wierzchołek ze szczytu. Wierzchołek ten jest już połączony z $v_j$ krawędzią wielokąta $\mathcal{P}$.

Zdejmujemy wierzchołki ze stosu i łączymy je z $v_j$ dopóki nie spotkamy wierzchołka dla którego nie jest to możliwe. Sprawdzenie czy możemy dodać przekątną z $v_j$ do $v_k$ możemy zrobić patrząc na $v_j$, $v_k$ i poprzedni wierzchołek, który został zdjęty.

Gdy znajdziemy wierzchołek, z którym nie możemy połączyć $v_j$ wkładamy ostatni zdjęty wierzchołek z powrotem na stos. Jest nim albo ostatni wierzchołek, do którego dodano przekątną, albo sąsiad $v_j$ na brzegu $\mathcal{P}$. Na koniec wkładamy $v_j$ na stos.



**Algorithm** TRIANGULATEMONOTONEPOLYGON($\mathcal{P}$)

*Input.* A strictly $y$-monotone polygon $\mathcal{P}$ stored in a doubly-connected edge list $\mathcal{D}$.

*Output.* A triangulation of $\mathcal{P}$ stored in the doubly-connected edge list $\mathcal{D}$.

1. Merge the vertices on the left chain and the vertices on the right chain of $\mathcal{P}$ into one sequence, sorted on decreasing $y$-coordinate. If two vertices have the same $y$-coordinate, then the leftmost one comes first. Let $u_1, \ldots, u_n$ denote the sorted sequence.
2. Initialize an empty stack $\mathcal{S}$, and push $u_1$ and $u_2$ onto it.
3. **for** $j \leftarrow 3$ **to** $n-1$
4. **do if** $u_j$ and the vertex on top of $\mathcal{S}$ are on different chains
5. **then** Pop all vertices from $\mathcal{S}$.
6. Insert into $\mathcal{D}$ a diagonal from $u_j$ to each popped vertex, except the last one.
7. Push $u_{j-1}$ and $u_j$ onto $\mathcal{S}$.
8. **else** Pop one vertex from $\mathcal{S}$.
9. Pop the other vertices from $\mathcal{S}$ as long as the diagonals from $u_j$ to them are inside $\mathcal{P}$. Insert these diagonals into $\mathcal{D}$. Push the last vertex that has been popped back onto $\mathcal{S}$.
10. Push $u_j$ onto $\mathcal{S}$.
11. Add diagonals from $u_n$ to all stack vertices except the first and the last one.

Na początku zakładaliśmy, że wielokąt jest ściśle $y$-monotoniczny, a algorytm podziału na $y$-monotoniczne części mógł tworzyć poziome krawędzie. Okazuje się, że nie jest to problem o ile wierzchołki z taką samą współrzędną $y$ rozpatrujemy od lewej do prawej.

Prosty wielokąt o $n$ wierzchołkach można striangulować w czasie $\mathcal{O}(n \log n)$ z pomocą algorytmu używającego $\mathcal{O}(n)$ pamięci.

Na początku zakładaliśmy, że wielokąt jest ściśle $y$-monotoniczny, a algorytm podziału na $y$-monotoniczne części mógł tworzyć poziome krawędzie. Okazuje się, że nie jest to problem o ile wierzchołki z taką samą współrzędną $y$ rozpatrujemy od lewej do prawej.

Prosty wielokąt o $n$ wierzchołkach można striangulować w czasie $\mathcal{O}(n \log n)$ z pomocą algorytmu używającego $\mathcal{O}(n)$ pamięci.

A co wielokątami z dziurami?
W algorytmie dzielącym na $y$-monotoniczne części nie korzystaliśmy z faktu, że wielokąt jest prosty, więc możemy go użyć do wielokątów z dziurami.