Dokumentacja projektu

obliczanie grafu widoczności

Algorytmy Geometryczne

Bartosz Nowak i Iwo Szczepaniak

Jupyter Notebook Anaconda, Python 3.9.12

1) Wstęp

Zadaniem, które należy wykonać jest stworzenie i wizualizacja grafu widoczności. Do tego konieczne są:

- interfejs umożliwiający zadanie przeszkód i wizualizujący je
- algorytm tworzący graf widoczności
- wizualizacja poszczególnych kroków algorytmu

Menu

2) Część techniczna

Program składa się z pliku głównego w Python Notebook, do którego zaimportowano szereg funkcji i struktur danych w języku Python. Kod źródłowy znajduje się w repozytorium na Githubie. Aplikację można otworzyć w wersji webowej oraz z IDE (np. VS Code):

- geometria.py – plik konfiguracyjny, w którym zapisana jest obsługa rysowania figur, linii, punktów, i zapis do Plot'ów
- constants.py – przechowująca stałe wykorzystywane w innych podprogramach
- graph.py – tu przechowywane są implementacje klas Vertex, Edge i Graph, wraz z funkcjami przypisanymi do tych klas
- trig.py – szereg funkcji obsługujących przecinanie odcinków, sortowanie, sprawdzanie czy krawędź należy do wnętrza figury itp.
- visible_vertices.py – funkcja zwracająca listę wierzchołków widocznych z zadanego wierzchołka
- visibility_graph.py – korzystając z visible_vertices tworzy graf widoczności
- utilities.py – zamienia zaimplementowane klasy Vertex, Edge i Graph na LinesCollection lub PointsCollection(potrzebne do wizualizacji)
- shortest_path.py – korzystając z klas Vertex i Graph znajduje minimalną ścieżkę od punktu start do end w grafie

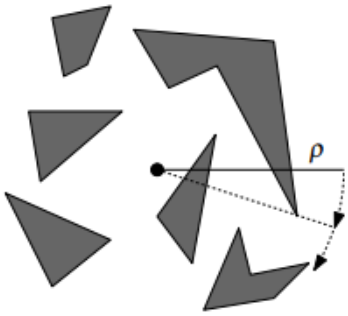
3) Część użytkownika

Aby zobaczyć działanie algorytmu należy uruchomić main_function.ipynb. W pierwszym bloku należy zaimportować funkcje, które umożliwią poprawne działanie reszty programu. Następnie, należy wprowadzić figury, które mają być przeszkodami klikając „Dodaj figurę”(najlepiej po narysowaniu figury na chwilę przełączyć się na „Dodaj linię” i z powrotem na „Dodaj figurę”), oraz dwa punkty - startowy i końcowy za pomocą przycisku „Dodaj punkt”. Dzięki temu po uruchomieniu ostatniego bloku pokaże się nam wizualizacja powstawania grafu widoczności(kolor szary) oraz najkrótsza ścieżka w nim(kolor zielony). Dodatkowo wyróżnione na zielono będą punkty start i end. Wizualizacja zakłada, że jeśli użytkownik wprowadził figurę niedomkniętą, to nie trafił on w wierzchołek i miała być ona domknięta.

4) Sprawozdanie

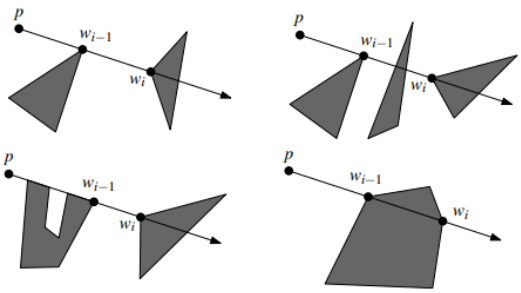
Graf widoczności to graf złożony z pewnej liczby wierzchołków oraz krawędzi łączących wierzchołki „widzące się wzajemnie". Wierzchołki „widzą się wzajemnie", jeśli krawędź łącząca te wierzchołki nie przekracza żadnej z figur zadanych przez użytkownika. Wierzchołkami grafu widoczności są wierzchołki figur oraz punkt początkowy i punkt końcowy. Krawędziami grafu widoczności są także odcinki, stanowiące boki poszczególnych wielokątów. Definiowanie grafu widoczności polega na znalezieniu wszystkich par wierzchołków, które widzą się wzajemnie. Naiwne podejście zatem dawałoby rozwiązanie n^3(dla każdego wierzchołka sprawdzenie n^2 możliwości).

Istnieje jednak sposób na poprawienie tej złożoności poprzez zastosowanie obrotowego zamiatania. Stanem jest w tym przypadku uporządkowany ciąg krawędzi przecinanych przez półprostą, a zdarzeniami są wierzchołki figur(oraz punkty start i end). Stan jest reprezentowany przez drzewiastą strukturę EdgeSet zaimplementowaną w trig.py.
Zamiatanie odbywa się dzięki skierowanej w dodatnim kierunku półprostej x i przebiega w kierunku przeciwnym do ruchu wskazówek zegara. Jeśli wierzchołek widoczny – dodajemy nową krawędź do listy widocznych krawędzi. Następnie przechodzimy do kolejnego wierzchołka i usuwamy krawędzie których półprosta już nie przecina oraz dodajemy nowe które przecina.



Rys 1.

Co jednak, gdy wierzchołki są w linii? Na szczęście sortowanie wierzchołków, gdy są na linii zwraca te bliższe jako pierwsze, więc rozważając odcinek w_i możemy wykorzystać wiedzę o wierzchołku w_{i-1}.

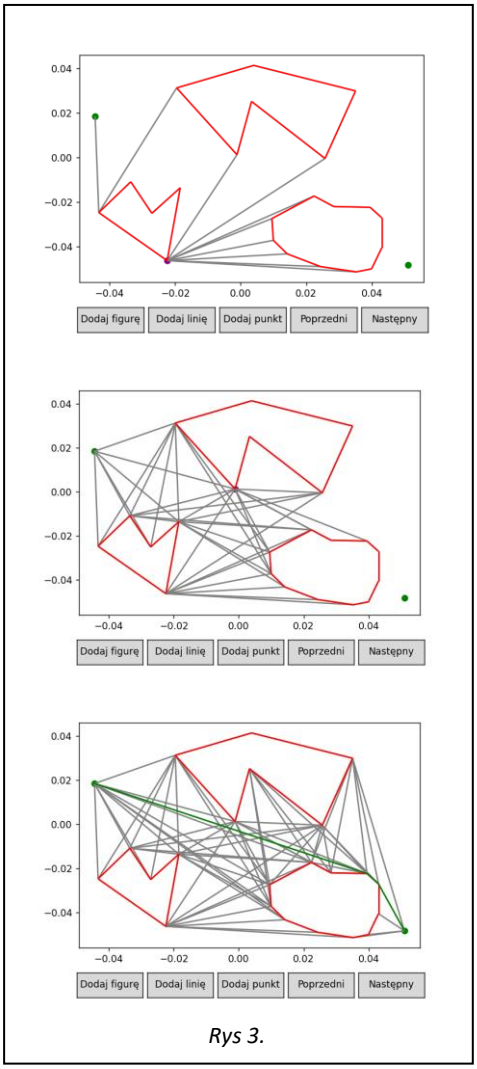


Rys 2.

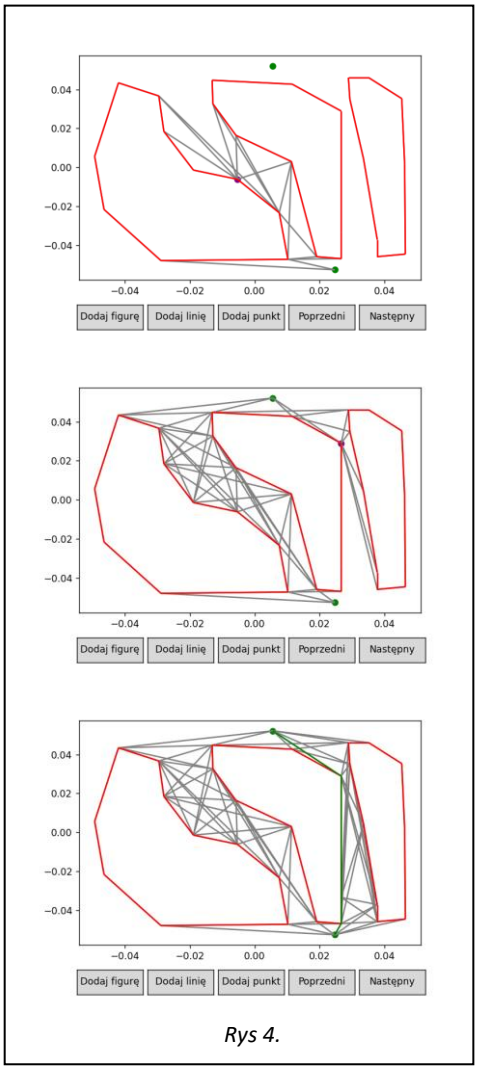
Zauważmy, że jeśli w_{i-1} nie jest widzialny, to w_i nie może być widzialny. Żeby w_i był widzialny, w_{i-1} musi być widzialny, ale nie daje to gwarancji widoczności w_i. Gdy w_{i-1} jest widoczny, to w_i może być niewidoczny na dwa sposoby – albo odcinek w_{i-1} w_i jest wnętrzem figury do której należą oba te wierzchołki, albo między nimi znajduje się figura, która odcinek w_{i-1} w_i przecina.

Wykorzystując tą wiedzę, jesteśmy w stanie stworzyć algorytm zwracający krawędzie widziane z danego wierzchołka. Tak więc, żeby otrzymać graf widoczności jedyne czego potrzeba to wykonać tę samą procedurę dla każdego wierzchołka. Dzięki temu podejściu redukujemy naiwne n^2 możliwości do nlogn, co daje sumaryczną złożoność algorytmu n^2logn. Ostatnim krokiem jest znalezienie najkrótszej ścieżki w grafie – na przykład poprzez wykorzystanie algorytmu Dijkstry, który ma pesymistyczną złożoność n^2logn, więc złożoność algorytmu pozostaje n^2logn.

Testowanie działania algorytmu



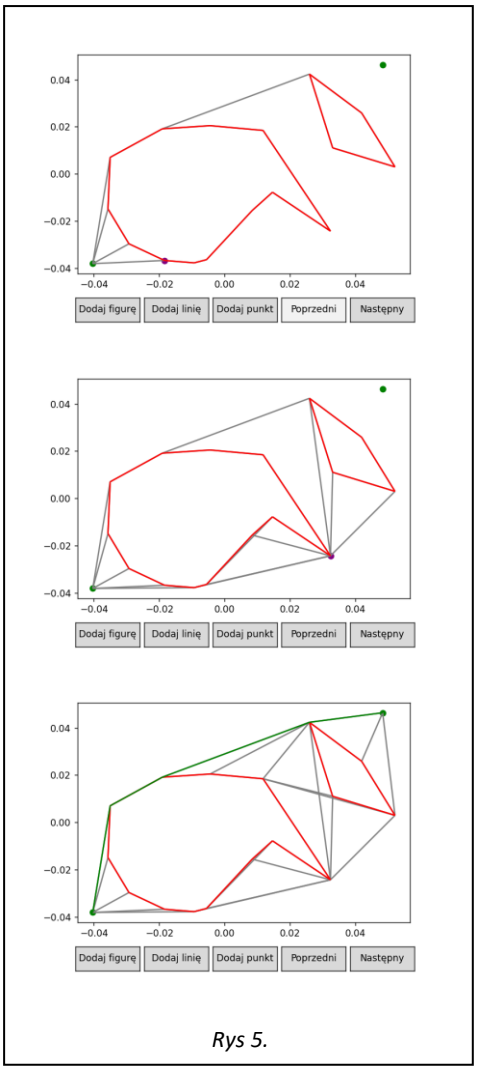
Rys 3.



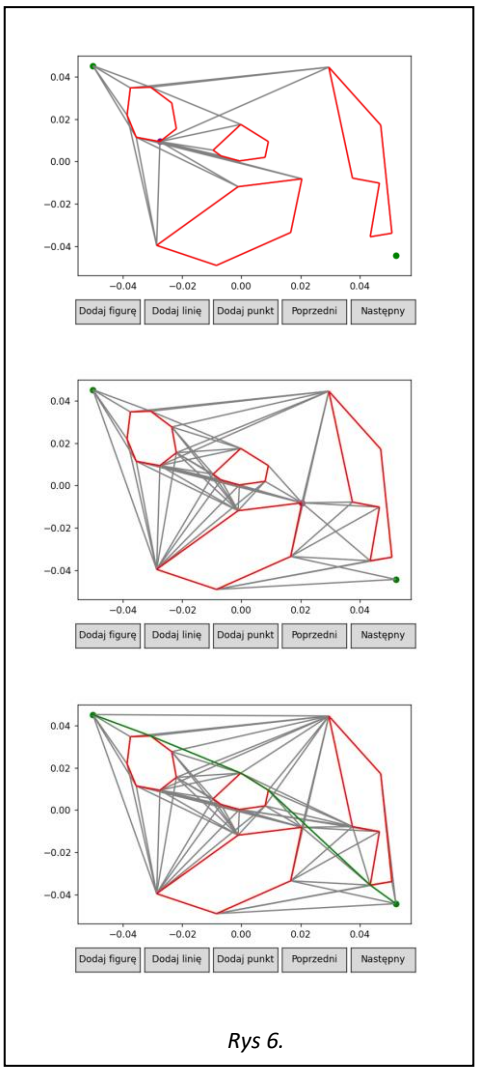
Rys 4.

We wszystkich przypadkach zastosowano to samo kolorowanie:

- na czerwono wprowadzone figury
- na szaro powstający graf widoczności
- na zielono najkrótsza droga w grafie widoczności oraz wierzchołki początkowy oraz końcowy
- na fioletowo aktualnie przetwarzany wierzchołek



Rys 5.



Rys 6.

Dla wszystkich wprowadzonych figur wizualizacja zachowuje się zgodnie z przewidywaniami.

Bibliografia

- Geometria obliczeniowa. Algorytmy i zastosowania - M. Berg, M. Kreveld, M. Overmars, O.Schwarzkopf
- https://www.science.smith.edu/~istreinu/Teaching/Courses/274/Spring98/Projects/Philip/fp/algVisibility.htm
- https://sj.umg.edu.pl/sites/default/files/ZN501.pdf
- https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/