Matematyka Dyskretna

Projekt Grupowy

Wyznaczanie optymalnej trasy drona przy użyciu Algorytmu A*

Grupa 6:

Bartłomiej Guś (297415)

PW

HK

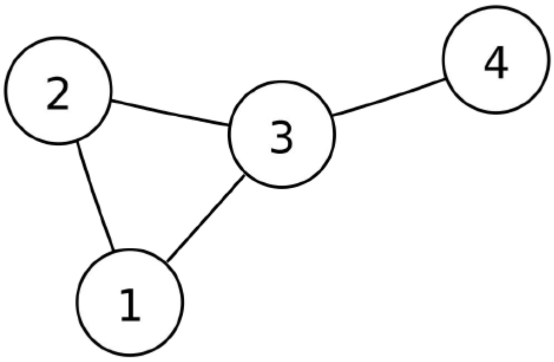
Warszawa 2023

Spis treści

1. Wprowadzenie teoretyczne

Wyznaczanie optymalnej trasy transportu to jeden z podstawowych celów działań logistycznych. Planując optymalną ścieżkę skraca się długość trasy do pokonania, a co za tym idzie – oszczędza się czas i pieniądze. Analizując problem dysponujemy punktem początkowym trasy oraz jej punktem końcowym, a także punktami przez które możliwy jest przejazd wraz z kosztem przejazdu przez nie. Rozwiązaniem problemu jest trasa z punktu początkowego do punktu końcowego o jak najmniejszym koszcie. Narzędziami pozwalającymi na wyznaczanie optymalnej trasy są algorytmy oparte o grafy.

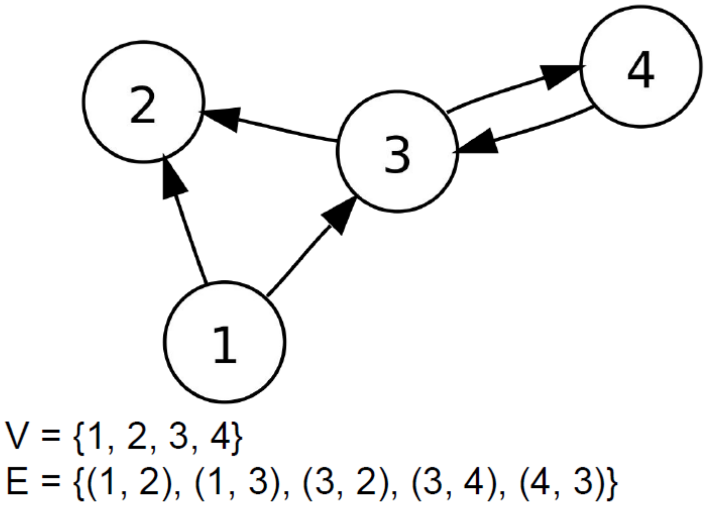
Graf nieskierowany *G* składa się z dwóch zbiorów: niepustego zbioru wierzchołków *V* (vertex) i zbioru dwuelementowych podzbiorów *V* zwanego zbiorem krawędzi *E* (edge), które mogą łączyć dwa wierzchołki. W przypadku grafu nieskierowanego połączenia miedzy wierzchołkami działają w obydwu kierunkach, jeżeli wierzchołek 1 jest połączony z wierzchołkiem 2 to wierzchołek 2 jest połączony z wierzchołkiem 1.



$$V = \{1, 2, 3, 4\}$$
$$E = \{\{1, 2\}, \{2, 3\}, \{3, 1\}, \{3, 4\}\}$$

Rys 1. Przykład grafu nieskierowanego

Graf skierowany *G* składa się z dwóch zbiorów: niepustego zbioru wierzchołków *V* (vertex) i zbioru par uporządkowanych ze zbioru *V* zwanego zbiorem krawędzi *E* (edge), które mogą łączyć dwa wierzchołki. W przypadku grafu kierowanego połączenia miedzy wierzchołkami działają tylko w jednym kierunku i kierunek jest oznaczony strzałką.



Rys 2. Przykład grafu skierowanego

Grafy mogą być uzupełnione o różne dodatkowe informacje. Na przykład:

- waga krawędzi (informacja o odległości lub trudności dostania się z jednego wierzchołka do drugiego),

- parametry wierzchołka (np. kolor, współrzędne kartezjańskie lub geograficzne).

Reprezentacja grafu, jako zbiór wierzchołków i krawędzi nie zawsze jest najlepszym sposobem na przedstawienie grafu. Inne warte uwagi sposoby to:

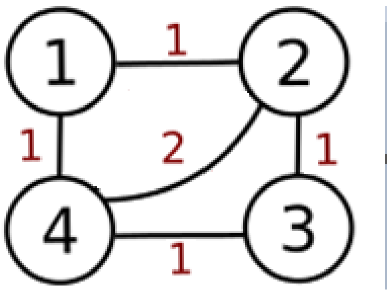
- macierz sąsiedztwa,

- listy sąsiedztwa,

- macierz incydencji.

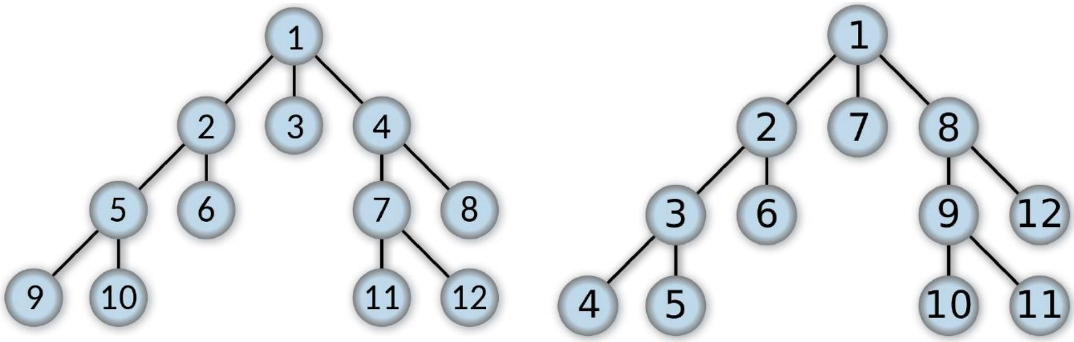
W wybranym algorytmie jako reprezentację grafu użyto macierzy sąsiedztwa, dlatego warto o niej wspomnieć. Macierz sąsiedztwa jest to struktura danych zawierająca informacje na temat połączeń między wierzchołkami oraz koszcie dostania się z jednego do drugiego. Numery wierszy i kolumn odpowiadają numerom wierzchołków, wyraz z *i*–tego wiersza i *j*-tej kolumny odpowiada wartości kosztu trasy z *i*-tego wierzchołka do *j*-tego. Gdy wyraz jest równy 0 oznacza to, że nie ma krawędzi pomiędzy tymi wierzchołkami.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix}$$



Rys 3. Przykład reprezentacji grafu za pomocą macierzy sąsiedztwa

Operacje jakie można wykonywać na grafie to przeszukiwanie, czyli przechodzenie grafu odwiedzając każdy wierzchołek w usystematyzowany sposób w celu zebrania potrzebnych informacji. Rozróżniamy przeszukiwanie wszerz i w głąb. Przeszukiwanie wszerz polega na odwiedzeniu wszystkich wierzchołków, osiągalnych z danego wierzchołka. Przeszukiwanie w głąb polega z kolei na przechodzeniu po krawędziach do kolejnych wierzchołków grafu tak głęboko jak tylko się da a następnie cofa się jeżeli skończyły się wszystkie krawędzie wierzchołka, z którego został odwiedzony.



Rys 4. Przeszukiwanie wszerz (lewy graf) i w głąb (prawy graf)

Grafy mają swoje zastosowanie w wielu zagadnieniach. W wybranym algorytmie grafy pomagają w planowaniu trasy, ale oprócz tego znajdują zastosowanie w:

- problem komiwojażera,

- reprezentacja danych,

- teoria gier,

- reprezentacja automatów,

- problem chińskiego listonosza,

- problem kojarzenia małżeństw.

2. Omówienie algorytmu

Algorytm A* to algorytm grafowy, czyli algorytm rozwiązujący problem przedstawiony przy użyciu pojęcia grafu. Problemem rozwiązywanym za pomocą tego algorytmu jest wyznaczanie najkrótszej ścieżki pomiędzy dwoma wierzchołkami grafu, a uściślając planowanie optymalnej trasy pomiędzy dwoma punktami.

Założenia algorytmu:

- dany jest punkt startowy i końcowy oraz macierz sąsiedztwa określająca możliwe połączenia wierzchołków i ich koszt,

- wybór wierzchołka do którego algorytm będzie szedł w następnym kroku jest determinowany przez funkcję *f(x)* - wybierany jest wierzchołek o najmniejszej wartości *f(x),*

- *f(x)* = *g(x)* + *h(x),* gdzie:

- $g(x)$ - droga od startu do x

- $h(x)$ - heurystyka szacująca drogę od x do mety np. Funkcja stała ≡ 0 lub odległość euklidesowa

- co krok algorytmu wartości f i g są aktualizowane.

Wymagania algorytmu:

- żeby algorytm znalazł trasę , trasa musi istnieć,

- żeby algorytm znalazł optymalną trasę:

  - trasa musi istnieć,

  - heurystyka h musi być optymistyczna/dopuszczalna (*admissible*) - czyli nigdy nie zawyżać wartości względem rzeczywistego kosztu dotarcia do celu, innymi (bardziej matematycznymi) słowy: $\forall x,\ h(x) \leq h^*(x)$, gdzie $h^*(x)$ - rzeczywisty optymalny koszt dotarcia do celu

Wynikiem działania algorytmu jest optymalna trasa zapisana za pomocą wierzchołków grafu przez które przechodzi oraz minimalny koszt związany z przejściem trasy.

3. Implementacja i zastosowanie

W projekcie zaimplementowano algorytm A* w celu wyznaczania optymalnej trasy drona w przestrzeni trójwymiarowej. Poszczególne elementy grafu reprezentują :

- Wierzchołki – punkty w przestrzeni trójwymiarowej, które zostały wcześniej wgrane do mapy,
- Krawędzie – bezkolizyjne trasy pomiędzy danymi punktami przestrzeni,
- Wagi krawędzi – koszt przemieszczenia się drona po danej trasie (krawędzi grafu).

W przestrzeni wyznaczono 10 punktów między którymi może poruszać się dron. Algorytm zadziała zarówno dla grafów skierowanych jak i nieskierowanych.

Danymi wejściowymi są:

- punkt startu,

- punkt mety,

- współrzędne wierzchołków,

- macierz sąsiedztwa.

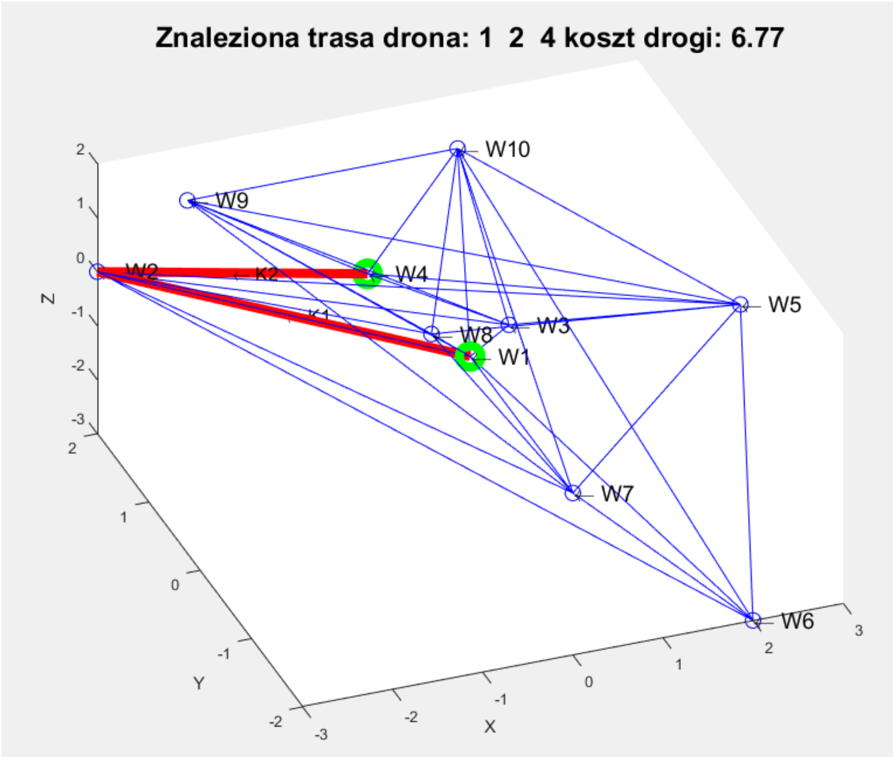
1	[0,0,0]
2	[-3,2,0]
3	[1,1,-1]
4	[0,2,-1]
5	[3,0,0]
6	[2,-2,-3]
7	[0,-2,0]
8	[-1,-1,2]
9	[-2,2,1]
10	[1,2,1]

	1	2	3	4	5	6	7	8	9	10
1	0	3.6100	0	0	0	4.1200	2	0	0	0
2	3.6100	0	4.2400	3.1600	6.3200	7.0700	0	0	0	0
3	1.7300	4.2400	0	1.4100	2.4500	0	0	0	0	2.2400
4	0	0	1.4100	0	3.7400	0	0	0	2.8300	2.2400
5	0	0	0	0	0	3.7400	3.6100	4.5800	5.4800	3
6	4.1200	7.0700	0	0	0	0	3.6100	0	0	5.7400
7	2	5	0	0	0	3.6100	0	2.4500	4.5800	0
8	2.4500	4.1200	0	0	0	0	2.4500	0	3.3200	3.7400
9	3	0	3.7400	2.8300	0	0	0	0	0	3
10	2.4500	0	0	0	3	0	4.2400	3.7400	0	0

Tab. 1. Współrzędne wierzchołków i macierz sąsiedztwa

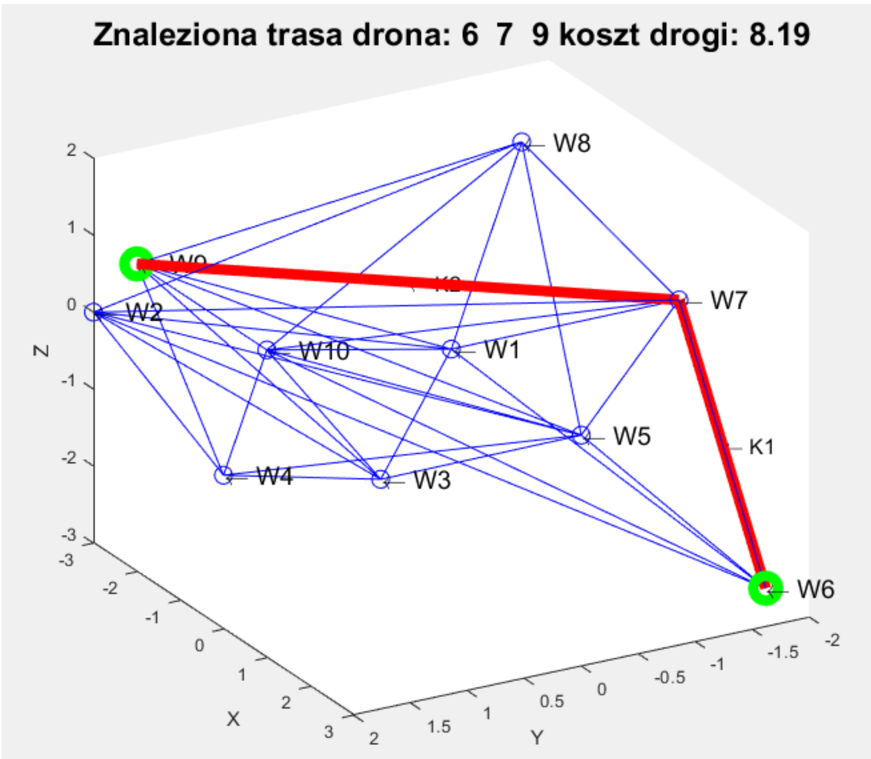
Wynikiem działania algorytmu jest wyznaczona optymalna trasa składająca się z odwiedzonych wierzchołków oraz koszt przejścia trasy. Ścieżka oznaczona jest czerwonymi krawędziami.

a) Wyznaczenie trasy z punktu 1 do punktu 4



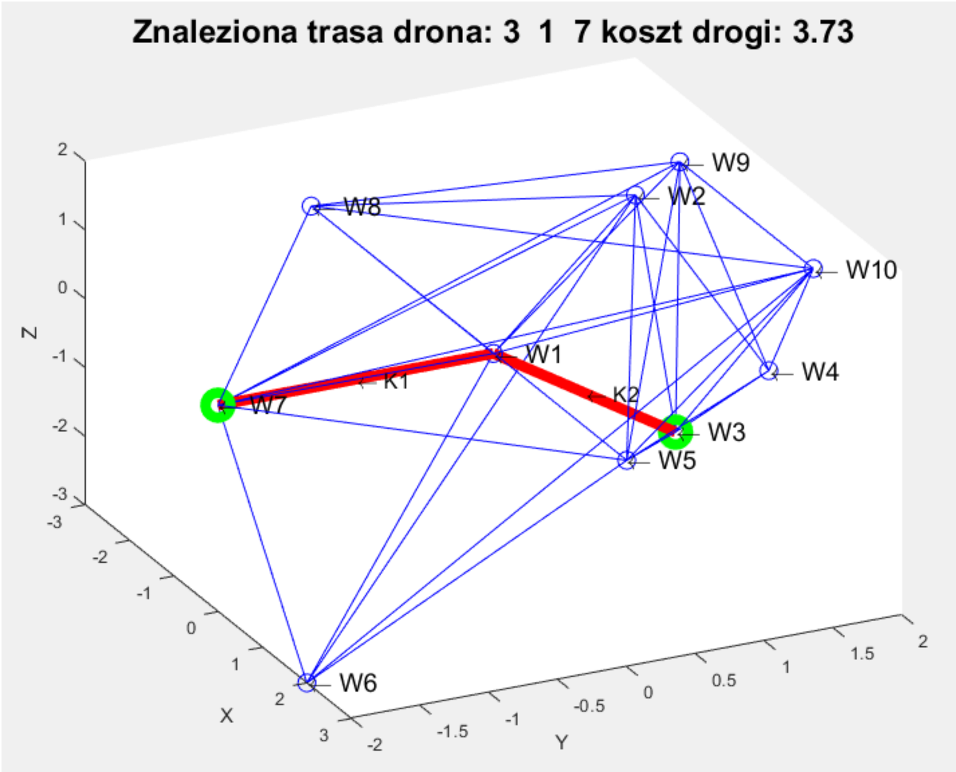
Wykres 1. Optymalna trasa z W1 do W4

b) Wyznaczenie trasy z punktu 6 do punktu 9



Wykres 2. Optymalna trasa z W6 do W9

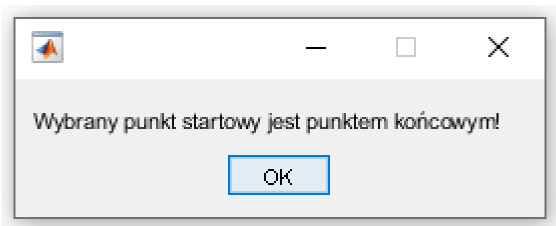
c) Wyznaczenie trasy z punktu 3 do punktu 7



Wykres 3. Optymalna trasa z W3 do W7

4. Omówienie wyników

Algorytm działa poprawnie, znajduje optymalną trasę dla dowolnie wybranych dwóch punktów. Uwzględnia kierunki krawędzi, czyli działa nie tylko dla grafów nieskierowanych, ale również dla grafów skierowanych. Dzięki temu możemy zaimplementować różny koszt lotu drona z punktu A do punktu B oraz z punktu B do punktu A. Może być to przydatne akurat w podanym zastosowaniu ze względu na zmieniający się kierunek wiatru i mniejsze opory drona przy locie w jednym kierunku niż w kierunku przeciwnym. Posiada również obsługę wyjątków. Gdy podamy jednakowy punkt startowy i końcowy pojawi a się komunikat:



Rys 4. Obsługa wyjątków

Dla przykładowych punktów startowych i mety wyznaczono optymalne trasy i ich koszt:

a) dla punktu startowego 1 i punktu mety 4 wyznaczono trasę [1 2 4] o koszcie 6.77,
b) dla punktu startowego 6 i punktu mety 9 wyznaczono trasę [6 7 9] o koszcie 8.19,
c) dla punktu startowego 3 i punktu mety 7 wyznaczono trasę [3 1 7] o koszcie 3.73.

Zaimplementowany algorytm rozwiązuje problem planowania optymalnej trasy drona, a tym samym pozwala oszczędzić czas i koszty związane z pokonywaniem trasy.

5. Kod źródłowy

```
clear;
clc;

plik_wejsciowy = "1";
sciezka = plik_wejsciowy+".mat";
load(sciezka);

rozpatrywane = [Dane.p_start];
przyszedlZ = [];
% droga od startu do aktualnego wierzcholka
g = [];
% f = g + h //h - heurystyka
f = [];

if Dane.p_start == Dane.p_meta
    msgbox('Wybrany punkt startowy jest punktem końcowym!');
    return
end

for i=1:size(Dane.wierzcholki,1)
    przyszedlZ(i)=0;
    g(i) = inf;
    f(i) = inf;
end

g(Dane.p_start) = 0;
f(Dane.p_start) = getDistance(Dane.wierzcholki{Dane.p_start},Dane.wierzcholki{Dane.p_meta});

while size(rozpatrywane,1)~=0
    tmp = inf;
    f_rozp = f(rozpatrywane);
    minF = min(f_rozp);
    indexMin = find(f==minF);
    for i=1:size(indexMin,2)
       if ~isempty(find(rozpatrywane==indexMin(i)))
           x = indexMin(i);
       end
    end
    if x==Dane.p_meta

       [sciezka, sciezka_do_napisu, kosztDrogi] = 
zrekonstruujTrase(przyszedlZ,Dane.p_meta,Dane.p_start,Dane.macierz_sasiedztwa);
       narysujWyniki(Dane,sciezka,sciezka_do_napisu, kosztDrogi);
       break;
    end
    rozpatrywane(rozpatrywane==x) = [];
    for i=1:size(Dane.macierz_sasiedztwa,2)
       if Dane.macierz_sasiedztwa(x,i)~=0
          tym_g = g(x) + Dane.macierz_sasiedztwa(x,i);
          if tym_g < g(i)
             przyszedlZ(i) = x;
             g(i) = tym_g;
             f(i) = g(i) + getDistance(Dane.wierzcholki{i},Dane.wierzcholki{Dane.p_meta});
             if isempty(rozpatrywane)
                czyYrozpatrywane = false;
             else
                czyYrozpatrywane = rozpatrywane==i;
             end
             if ~czyYrozpatrywane
                rozpatrywane = [rozpatrywane;i];
             end
          end
       end
    end
```

```
end


function distance = getDistance(w1,w2)
    distance = sqrt((w2(1)-w1(1))^2+(w2(2)-w1(2))^2+(w2(3)-w1(3))^2);
end


function narysujWyniki(Dane,sciezka,sciezka_do_napisu,kosztDrogi)
wspolczynnik = 100;
iteratorKrawedzi = 1;

for i=1:size(Dane.wierzcholki,1)
   if (i == Dane.p_start) || (i == Dane.p_meta)

plot3(Dane.wierzcholki{i}(1),Dane.wierzcholki{i}(2),Dane.wierzcholki{i}(3),'o','Color','g','MarkerSize'
,14,'LineWidth',6);
   else

plot3(Dane.wierzcholki{i}(1),Dane.wierzcholki{i}(2),Dane.wierzcholki{i}(3),'o','Color','b','MarkerSize'
,10);
   end
   text(Dane.wierzcholki{i}(1),Dane.wierzcholki{i}(2),Dane.wierzcholki{i}(3),"\leftarrow
W"+num2str(i),'FontSize',14);
   hold on;
end

for i=1:size(Dane.macierz_sasiedztwa,1)
   for j=1:size(Dane.macierz_sasiedztwa,2)
       if Dane.macierz_sasiedztwa(i,j) ~= 0
          temp_wektor = [abs(int64((Dane.wierzcholki{i}(1)-
Dane.wierzcholki{j}(1))*wspolczynnik)),abs(int64((Dane.wierzcholki{i}(2)-
Dane.wierzcholki{j}(2))*wspolczynnik)),abs(int64((Dane.wierzcholki{i}(3)-
Dane.wierzcholki{j}(3))*wspolczynnik))];
          liczbaElementow = max(temp_wektor);
          x_lin = linspace(Dane.wierzcholki{i}(1),Dane.wierzcholki{j}(1),liczbaElementow);
          y_lin = linspace(Dane.wierzcholki{i}(2),Dane.wierzcholki{j}(2),liczbaElementow);
          z_lin = linspace(Dane.wierzcholki{i}(3),Dane.wierzcholki{j}(3),liczbaElementow);
          krawedzZnaleziona = false;
          for k=1:size(sciezka,1)
             if sciezka{k}(1) == i && sciezka{k}(2) == j
                 krawedzZnaleziona = true;
             end
          end
          if krawedzZnaleziona == true
            plot3(x_lin,y_lin,z_lin,'Color','r','LineWidth',6)

text(x_lin(int64(liczbaElementow/2)),y_lin(int64(liczbaElementow/2)),z_lin(int64(liczbaElementow/2)),"\
leftarrow K"+num2str(iteratorKrawedzi),'FontSize',12);
            iteratorKrawedzi = iteratorKrawedzi+1;
          else
            plot3(x_lin,y_lin,z_lin,'Color','b','MarkerSize',6);
          end
          hold on;
       end
   end
end

title("Znaleziona trasa drona: " + num2str(sciezka_do_napisu) +" koszt drogi:
"+num2str(kosztDrogi),'FontSize',18);
xlabel('X');
ylabel('Y');
zlabel('Z');
hold off;
end


function [sciezka_do_rysunku,sciezka,kosztDrogi] =
zrekonstruujTrase(przyszedlZ,p_meta,p_start,macierz_sasiedztwa)
kosztDrogi = 0;
odwroconaSciezka=[p_meta];
index = p_meta;
while przyszedlZ(index)~=p_start
    kosztDrogi = kosztDrogi + macierz_sasiedztwa(przyszedlZ(index),odwroconaSciezka(end));
```

```
    odwroconaSciezka=[odwroconaSciezka przyszedlZ(index)];
    index = przyszedlZ(index);
end
kosztDrogi = kosztDrogi + macierz_sasiedztwa(p_start,odwroconaSciezka(end));
odwroconaSciezka = [odwroconaSciezka p_start];
sciezka = flip(odwroconaSciezka);
sciezka_do_rysunku = {};
for i=1:(size(sciezka,2)-1)
    if isempty(sciezka_do_rysunku)
        sciezka_do_rysunku = {[sciezka(i) sciezka(i+1)]};
    else
        sciezka_do_rysunku{end+1,1} = [sciezka(i) sciezka(i+1)];
    end
end
disp("Znaleziona  trasa przez algorytm to: "+num2str(sciezka));
disp("Koszt drogi: "+num2str(kosztDrogi));
```

6. Spis rysunków

7. Spis tabel

8. Spis wykresów

9. Bibliografia

[1] Duszak P.: Grafy i A*, Warszawa 2022

[2] https://pl.wikipedia.org/wiki/Graf_(matematyka)

[3] https://pl.wikipedia.org/wiki/Algorytm_A*