Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Težji projekt pri Matematičnem programiranju

Timen Bobnar

Ljubljana, 2024

Povzetek

V ravnini so dane točke A_1, A_2, ... A_n. Enostavna sklenjena lomljenka je krivulja brez samopresečišč, ki je sestavljena iz daljic A_1A_2, A_2A_3,...,A_nA_1.

Zanimala nas bosta dva problema, in sicer:

Problem 1:

Zanimalo nas bo ali neko zaporedje točk res predstavlja enostavno sklenjeno lomljenko.

Problem 2:

Če imamo enostavno sklenjeno lomljenko in še neko poljubno točko, nas bo zanimalo, ali ja ta točka v notranjosti enostavne sklenjene lomljenke ali ne.

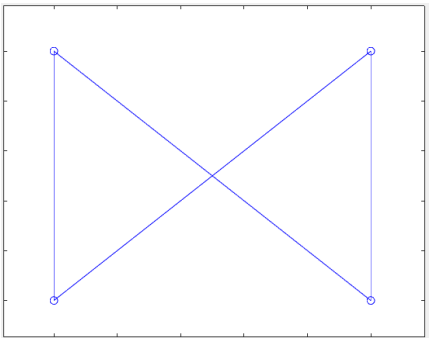
Problem 1

Navodilo:

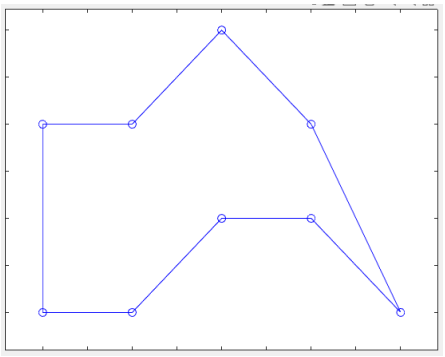
Sestavi funkcijo, ki ugotovi, ali je zaporedje točk A_1, A_2, ... A_n res dana sklenjena lomljenka brez samopresečišč.

Primeri:

Primer, ki ima samo presečišče, torej ni enostavna sklenjena lomljenka.



Primer, ki nima samo presečišča, torej je enostavna sklenjena lomljenka.



Teoretična osnova:

Da rešimo naš problem moramo za vsako daljico preveriti ali se seka s kakšno od ostalih daljic naše sklenjene lomljenke.

Presečišče se lahko zgodi le na naslednje načine. Vzemimo 2 daljici prva daljica je definirana s točkama A_1 in A_2. Druga daljica pa z B_1 in B_2.

- Dve premici se sekata natanko tedaj, ko veljata naslednja pogoja.
    1. Če imamo daljico A_1A_2 in pogledamo orientacijo točk B_1 ter B_2 glede na daljico A_1A_2, sta orientaciji nasprotno predznačeni.
    2. Če imamo daljico B_1B_2 in pogledamo orientacijo točk A_1 ter A_2 glede na daljico B_1B_2, sta orientaciji nasprotno predznačeni.
- Dve daljici se tudi lahko sekata natanko tedaj, ko eno izmed krajišč prve daljice leži na drugi daljici.

Pogoj 1:

Prvo vprašanje, ki se pojavi je, kako pridobimo orientacijo točke glede na daljico.

Pri tem si lahko pomagamo z vektorskim produktom, ki ima formulo:

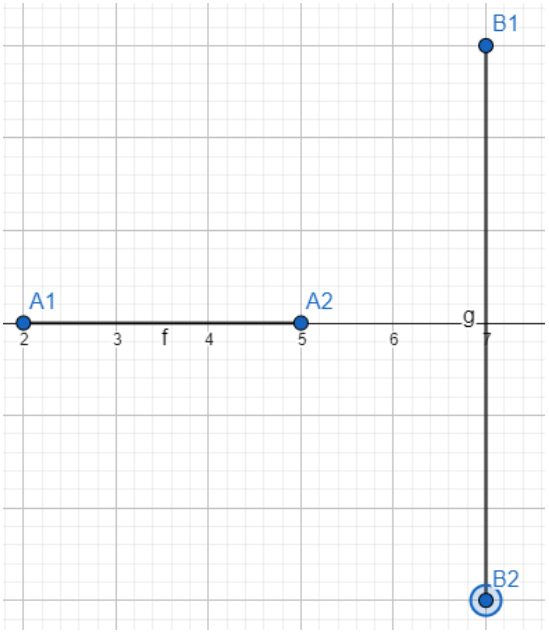
$$|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| * \sin\rho$$

Torej če imamo daljico A_1A_2 in poljubno točko C, potem definiramo vektor $\vec{a} \equiv \overrightarrow{A1A2}$ in vektor $\vec{b} \equiv \overrightarrow{A1C}$. Če stvari tako definiramo, potem bo vektorski produkt pozitiven, ko bo točka C na levi strani premice, negativen, ko je ne desni strani in 0, ko sta vektorja $\vec{a}$ in $\vec{b}$ vzporedna.

Sedaj si oglejmo kako deluje pogoj 1.

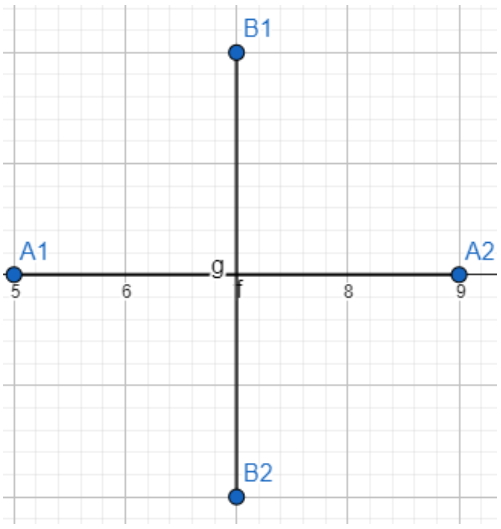
Imamo 3 možne situacije:

1. Situacija:



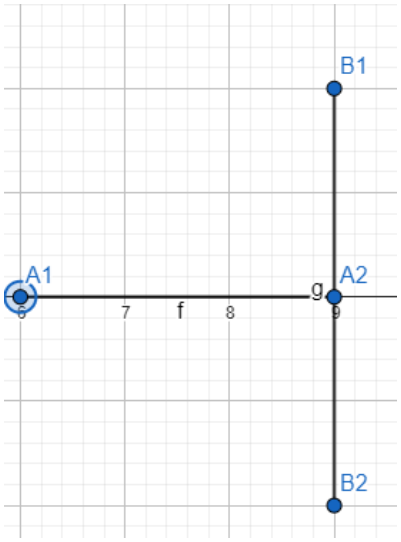
To je primer ko velja le en pogoj. B_1 je orientirana pozitivno glede na daljico A_1A_2,točka B_2 pa negativno. Hkrati sta toči A_1 in A_2 obe orientirani negativno glede na premico B_1B_2. Daljici se torej ne sekata.

2. Situacija



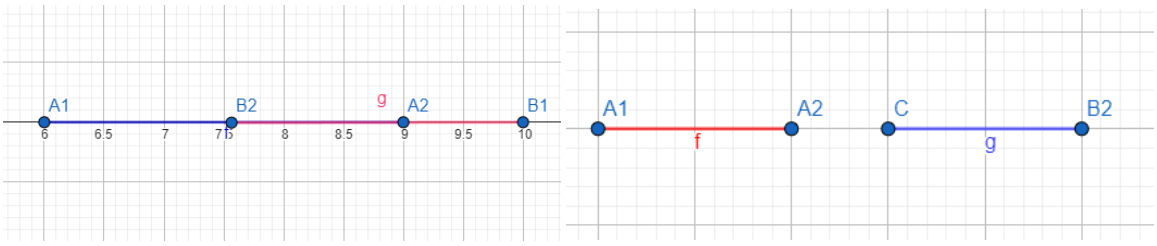
To je primer ko veljata oba pogoja. B_1 je orientirana pozitivno glede na daljico A_1A_2, točka B_2 pa negativno. Hkrati sta točki A_1 in A_2 nasprotno orientirani glede na premico B_1B_2. Torej se daljici sekata.

3. Situacija



Tu pridemo do situacije ko velja prvi pogoj. Ko preverimo drugi pogoj, pa pridemo do situacije, da je A_1 negativno orientiran, ter da A_2 laži na premici med B_1 in B_2. Ta problem je rešljiv že samo s pregledamo orientacij, vendar ga bomo zaradi enostavnejše kode združili z naslednjim problemom (ali točka leži na premici).

4. Situacija



Tu pridemo do situacije, ko so vse orientacije 0. In si z orientacijami ne moremo pomagati da ocenimo ali se daljici sekata ali ne. Oba primera imata vse orientacije enake 0, vendar prvi ima presečišče drugi pa ne.

Zadnje dva problema bomo oba rešili tako, da preverimo ali katero od krajišč leži na drugi daljici. Torej pogoj 2.

Pogoj 2:

Spomnimo se, da pogoj 2 izvedemo, če je ena od orientacij enaka 0.

Pri pogoju 2 si lahko pomagamo z preprosto idejo, ki jo izpeljemo iz algebraične definicije premice. Imejmo premico A_1A_2 ter točko C. Sedaj lahko definiramo:

$$\vec{C} = \overrightarrow{A1} + \lambda * \overrightarrow{\mathrm{A1A2}}$$

Iz formule sledi, da v primeru, ko je $\lambda \in [0,1]$, je točka C na premici A_1A_2.

Lambda je torej enaka:

$$\lambda = \frac{\vec{C} - \overrightarrow{A1}}{\overrightarrow{\mathrm{A1A2}}} = \frac{\vec{C} - \overrightarrow{A1}}{\overrightarrow{A2} - \overrightarrow{A1}}$$

Lambde bomo posebej izračunali za x in y koordinate tako, da bomo pisali $\lambda x \; in \; \lambda y$.

V splošnem primeru sledi, da ko sta $\lambda x \; in \; \lambda y$ enaki ter iz intervala [0,1], da točka leži na daljici.

Vendar imamo nekaj problemov.

Problemi:

Problemi nastopijo ko so naše daljice vodoravne ali navpične. V tem primeru je ena od $\lambda x \; ali \; \lambda y$ nedefinirana, saj bomo delili z 0.

Recimo da je λx nedefinirana. To pomeni, da je premica navpična, saj se *x* koordinata začetne ter končne točke ne spremeni. Iz tega sledi, da če hočemo, da naša točka leži na daljici mora izpolnjevati dva pogoja.

1. Kot smo omenili je premica navpična, torej imata končna in začetna točka isto x koordinato, zato jo mora imeti tudi točka C, če hočemo da leži na nje.
2. Ker je premica navpična, se lahko naša točka premika le med $\min(A1_y, A2_y)$ in $\max(A1_y, A2_y)$, če hočemo da leži na daljici.

Podobno velja, če je λy nedefinirana.

Implementacija v Matlab-u

Nalogo smo implementirali, tako da smo definirali nekaj različnih funkcij.

- Prva funkcija je orientacija oziroma implementacija prvega pogoja. Ta nam pove, kako je točka orientirana glede na neko daljico.

```
function or = orientacija(p, q, r)
    % gledamo kako je r orientiran glede na premico pq
    % 0 --> na isti premici
    % 1 --> negativna smer
    % 2 --> pozitivna mser
    or = (q(1) - p(1)) * (r(2) - p(2)) - (q(2) - p(2)) * (r(1) - p(1));
    %vektorski produkt
    if or == 0
        or = 0; % na isti premici
    elseif or > 0
        or = 1; % negativna smer
    elseif or < 0
        or = 2; % pozitivna smer
    end
end
```

V funkciji *or* izračunamo preko vektorskega produkta.

- Druga funkcija je funkcija z imenom na premici oziroma implementacija drugega pogoja. Ta ugotovi ali se neka točka nahaja na daljici.

```
function lezi = na_premici(p, q, r)
% Preveri, ali je točka q na premici pr


lezi=false;
lamda1=(q(1)-p(1)) / (r(1) - p(1));
lamda2=(q(2)-p(2)) / (r(2) - p(2));%izračunamo labde, ki pove kje se nahaja točka q glede na premico pr q=p+lamda*(r-q)

if isnan(lamda2) || isnan(lamda1) %se zgodi ko je kakšna od lamnd nedefinirana / premica je navprična ali vodoravna
    lezi = (q(1) <= max(p(1), r(1)) && q(1) >= min(p(1), r(1)) && q(2) <= max(p(2), r(2)) && q(2) >= min(p(2), r(2)));
    return;% lahko se lepo pokaže s skico kvadrata
end

if lamda2==lamda1 && 0<=lamda1 && lamda1<=1 %če se lambdi ujemata in sta med 0 in 1 potem točka res leži na daljici
   lezi=true;
end
```

Lamde izračunamo po definiciji premice. V prvem pogoju rešujemo tako imenovan Problem pri pogoju dva iz teoretične osnove. V drugem pogoju pa samo sledimo pogoju 2, ki pravi, da če sta lamdi enaki in iz intervala [0,1], potem točka leži na premici.

- Tretja funkcija se imenuje ali_se_sekajo. Ta funkcija prejme 4 točke oziroma 2 daljici in vrne true če se daljici sekata in false sicer. To naredi tako, da najprej preveri pogoje orientacije:

```
function resitev = ali_se_sekajo(p1, q1, p2, q2)
    % preveri ali se premice p1-q1 in p2-q2 sekajo
    o1 = orientacija(p1, q1, p2);
    o2 = orientacija(p1, q1, q2);
    o3 = orientacija(p2, q2, p1);
    o4 = orientacija(p2, q2, q1);

    % splošno
    if o1 ~= o2 && o3 ~= o4
        resitev = true;
        return;
    end
```

V primeru označenim s splošno preverjamo pogoj:

Dve premici se sekata natanko tedaj ko veljata naslednja pogoja.

1. Če imamo daljico A_1A_2 in pogledamo orientacijo točk B_1 ter B_2 glede na daljico A_1A_2. Sta orientaciji nasprotno predznačeni.
2. Če imamo daljico B_1B_2 in pogledamo orientacijo točk A_1 ter A_2 glede na daljico B_1B_2. Sta orientaciji nasprotno predznačeni.

V primeru, ko pa je kakšna od orientacije enaka 0 uporabimo funkcijo na_premici:

```
if o1 == 0 && na_premici(p1, p2, q1)
    resitev = true;
    return;
end
if o2 == 0 && na_premici(p1, q2, q1)
    resitev = true;
    return;
end
if o3 == 0 && na_premici(p2, p1, q2)
    resitev = true;
    return;
end
if o4 == 0 && na_premici(p2, q1, q2)
    resitev = true;
    return;
end
```

Pri teh pogojih preverimo ali točka, ki je imela orientacijo 0 leži na daljici.

- Četrta funkcija se imenuje lomljenka. V tej funkciji se sprehodimo čez vse možne kombinacije daljic naše enostavne sklenjene lomljenke.

```
function odg1 = lomljenka(tocke)
    % preverimo ali je res zaporedje točk sklenjena lomljenka.
    n = length(tocke);
    if n < 3
        odg1 = false; %moramo imete več kot 3 točke da ima stvar smisu
        return;
    end
```

Najprej preverimo ali je sploh možno sestaviti lomljenko. Če imamo manj kot 3 točke to ni možno, saj dve daljici ne moreta sestaviti lomljenke.

```
if tocke(1,:) == tocke(end,:)
    odg1 = false;
    return;
    %n=n-1;% namesto zgornjih 2h vrstic lahko vstavimo to in kodo
    %spremenimo tako da lahko našo lomljenko podamo na dva načina in
    %sicer: prva in zadnja točka sta enaki ali pa ne
end

if (tocke(1,1) ~= tocke(end,1))||(tocke(1,2) ~= tocke(end,2))
    tocke=[tocke;tocke(1,:)];
end
```

V tem delu najprej preverimo ali je zadnja točka enaka prvi, če sta enaki to ni sklenjena lomljenka. Sicer pa dodamo na konec prvo točko za lažje risanje in obravnavo.

```
n=n-1;%da ne dobimo napake zaradi zadnjega skup
%potreben je en popravek ko zadnja črta leži na

for i = 1:n
    for j = (i + 2):n %i+2 zato da preskočimo e
        if i ~= j
            %gremo z mod da se lahko zavrtimo
            p1 = tocke(i, :);
            q1 = tocke(i+1 , :);
            p2 = tocke(j, :);
            q2 = tocke(j+1 , :);
            if ali_se_sekajo(p1, q1, p2, q2)
                odg1 = false; % se seka
                return;
            end
        end
    end
end
odg1 = true; % se ne seka
```

V tem delu najprej *n* zmanjšamo za ena, da ne pride do prekrivanj v zanki. Zunanja zanka nam bo torej opisala vse segmente lomljenke razen zadnjega. Druga zanka pa vse segmente od i+2-gega naprej. Opazimo, da pri preverjanju preskočimo en segment in sicer $A_{i+1}A_{i+2}$, ta segment ustvari samo presečišče natanko tedaj, ko je A_{i+2} na daljici A_iA_{i+1}, kar pa že tako ali tako preverimo.

```
%zadnja točka lezi med prvima dvema ali druga točka lezi med prvo in
%zadnjo
if na_premici(tocke(1,:), tocke(end-1,:), tocke(2,:)) || na_premici(tocke(1,:), tocke(2,:), tocke(end-1,:))
    odg1= false;
end
```

Ker smo zadnjo daljico izpustili, moramo preveriti še njo. Preverimo pogoj, zadnja točka leži med prvima dvema ali druga točka leži med prvo in zadnjo. Če je to res, potem to ni enostavna sklenjena lomljenka.

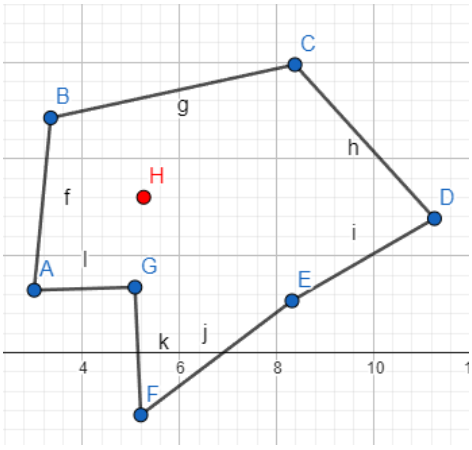
Problem 2

Navodilo:

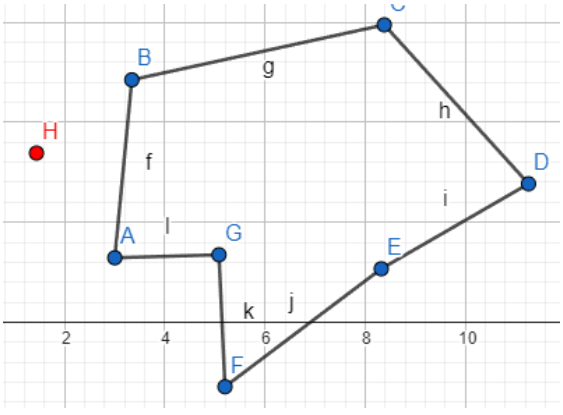
Sestavi funkcijo, ki ugotovi, ali se dana točka T nahaja v notranjosti enostavne sklenjene lomljenke A_1A_2, A_2A_3,...,A_nA_1 ali ne.

Primeri:

Primer, ko je točka v notranjosti enostavne sklenjene lomljenke:

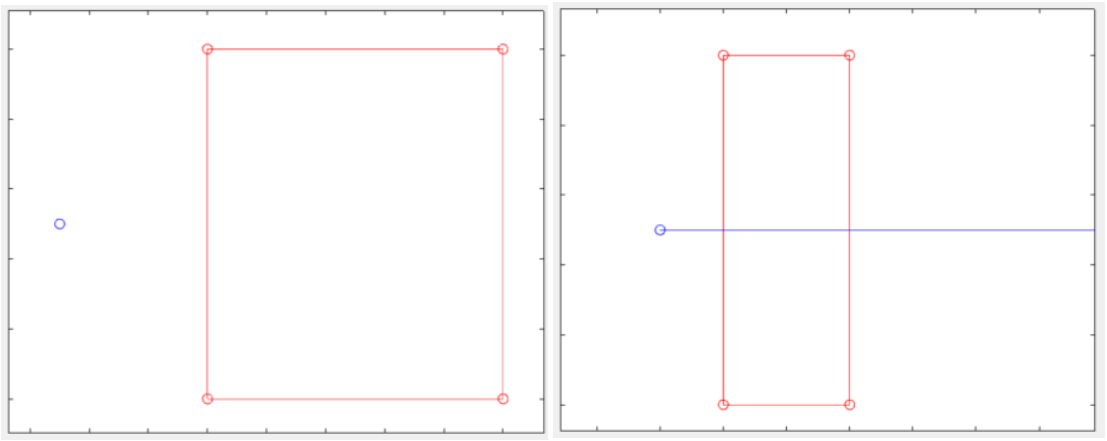


Primer ko je točka izven enostavne sklenjene lomljneke:

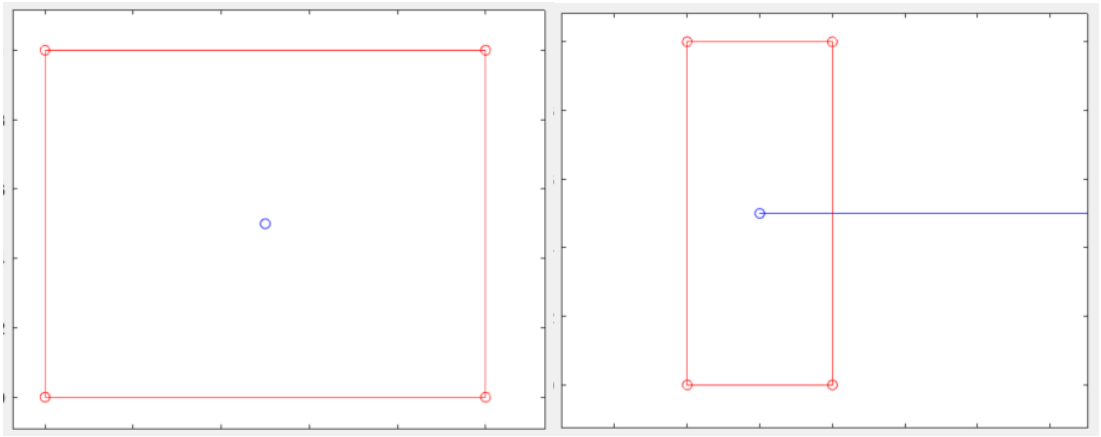


Teoretična osnova

Teoretična osnova za našo rešitev prihaja iz Ray casting aloritma. Ray casting algoritem sloni na matematičnem ozadju iz teorije grafov in topologije. Tu bomo samo uporabili idejo, ki pravi, če imamo poljubno točko ter enostavno sklenjeno lomljenko. Iz poljubne točke potegnemo premico (ni važno v katero smer). Naša premica sedaj seka enostavno sklenjeno lomljenko v sodo oziroma liho mnogo točkah. V primeru, da premica seka lomljenko sodo krat pomeni, da točka ni vsebovana. Če premica seka lomljenko liho krat pomeni, da je točka vsebovana v lomljenki.



Opazimo, da je točka res zunaj in, če preštejemo presečišča jih je sodo.



Tu opazimo, da je točka v notranjosti, in kot pričakovano je število presečišč liho.

Uporabili bomo tudi vse kar je bilo povedano v teoretični osnovi za prvi problem.

Implementacija v Matlab-u

Nalogo smo implemenirali z uporabo različnih funkcij. Večina je enakih kot v prejšnjem poglavju in jih bomo zato samo našteli:

1. Orientacija
2. na_premici
3. ali_se_sekajo

Definirali smo še eno novo funkcijo, ki preveri ali točka leži v lomljenki ali ne. Podano imamo naše točke, ki predstavljajo lomljenko in točko *p* ki predstavlja našo poljubno točko.

```
if (tocke(1,1) ~= tocke(end,1))||(tocke(1,2) ~= tocke(end,2))
    tocke=[tocke;tocke(1,:)];
end

plot(tocke(:,1),tocke(:,2),'ro-')
hold on
plot([p(1)]',[p(2)]','bo-')
% naredimo točko nikje zelo daleč stran da lahko nato vlečemo povezave
extreme = [inf, p(2)];

count = 0;%preštejemo vsa presečišča

n=n+1;

i = 1;
```

Našo lomljenko zapremo tako, da dodamo še eno točko. In *n*(števec daljic) povečamo za ena, da bomo preverili vse daljice. Ustvarimo tudi točko extreme, ki jo potrebujemo, da ustvarimo daljico od točke *p* do neskončnosti v neko smer.

```
i = 1;
while true && i<n
    naslednji = i+1;

    %preverimo ali se sekajo premice tocka(i)--tocka(i+1) ter premica
    % p--etreme
    if ali_se_sekajo(tocke(i,:), tocke(naslednji,:), p, extreme)
        %če so premice vsporedne preverimo če se točka p nahaja na
        %premici tocka(i)--tocka(i+1) če se nahajav na premici vrnemo
        %true
        if orientacija(tocke(i,:), p, tocke(naslednji,:)) == 0
            notri = na_premici(tocke(i,:), p, tocke(naslednji,:));
            return
        end
        %če se premice le sekajo števec povečamo
        count = count + 1;
    end

    i = naslednji;
    if i == 1
        break
    end
end
```

Za vsako daljico naše lomljenke preverimo ali se seka z našo umetno daljico od *p* do neskončno. Če se seka povečamo števec za 1. Preverimo pa tudi ali se točka *p* nahaja na daljici, to storimo le, ko je orientacija enaka 0, če se izkaže da je točka *p* res na neki daljici naše lomljenke, lahko takoj odgovorimo, da naša točka leži znotraj lomljenke.

```
%kot pravi Ray-casting aloritem če je števec liho vrnemo
%true(točka je notri) sicer vrnemo false
if mod(count, 2) == 1
    notri=1;
end
```

Preverimo ali je število presečišč liho ali sodo in podamo odgovor.

Viri

- Point in polygon, https://en.wikipedia.org/wiki/Point_in_polygon , 25.2.2024
- How to chech if a given point lies liside or outside a polygon?, https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/ , 25.2.2024
- Nekatere ideje so bile pridobljene s pomočjo ChatGPT-ja
- Nekatere ideje so bile razvite skupaj s kolegi