Univerzita Karlova
Přírodovědecká fakulta


# Algoritmy počítačové kartografie

## Úloha 1: Geometrické vyhledávání bodu

Adam Kulich, Markéta Růžičková, Eliška Sieglová
N-GKDPZ

Praha 2023

# 1 Zadání

Vstup: *Souvislá polygonová mapa n polygonů* $\{P_1,...,P_n\}$, *analyzovaný bod q.*
Výstup: $P_i$, $q \in P_i$.
Nad polygonovou mapou implementujete Ray Crossing Algorithm pro geometrické vyhledání incidujícího polygonu obsahující zadaný bod *q*.
Nalezený polygon graficky zvýrazněte vhodným způsobem (např. vyplněním, šrafováním, blikáním). Grafické rozhraní vytvořte s využitím frameworku QT.
Pro generování nekonvexních polygonů můžete navrhnout vlastní algoritmus či použít existující geografická data (např. mapa evropských států).
Polygony budou načítány z textového souboru ve Vámi zvoleném formátu. Pro datovou repretentaci jednotlivých polygonů použijte špagetový model.

**Hodnocení:**

| **Krok** | **Hodnocení** |
|---|---|
| Detekce polohy bodu rozlišující stavy uvnitř, vně, na hranici polygonu. | 10b |
| *Analýza polohy bod (uvnitř/vně) metodou Winding Number* | *+5b* |
| *Ošetření singulárního případu u Ray Algorithm: bod leží na hraně polygonu.* | *+5b* |
| *Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.* | *+2b* |
| *Zvýraznění všech polygonů pro oba výše uvedené singulární případy.* | *+3b* |
| **Max celkem:** | **25b** |

Čas zpracování: 1 týden.

# 2 Údaje o bonusových úlohách + body

**Algorithm 1** Body

**if** povinná úloha && všechny bonusové úlohy && psané v LaTeXu **then**
30 bodů?
**end if**

# 3 Popis a rozbor problému + vzorce

## 3.1 Úvod do problému

V této úloze se zaměřujeme na vyřešení problém zvaného *Point in Polygon* či *Point Location Problem*, tedy zjišťujeme, zda se zadaný bod nachází uvnitř, na hraně, nebo vně polygonu. S tímto problémem se často setkáme ve výpočetní geometrii a zejména v mnoha geoinformačních úlohách potřebujeme naleznout jeho řešení. Jedním příkladem může být, že máme zadaný bod, a zadané polygony regionů (např. států), a snažíme se přijít na to, ve kterém státě bod leží.

Princip řešení problému *Point in Polygon* se skládá ze dvou procedur: lokální a globální. Lokální procedura se zabývá polohou řešeného bodu *q* vůči jednomu konkrétnímu polygonu, a jejím výsledkem je, že bod leží uvnitř, vně, či na hraně tohoto polygonu. Do globální procedury poté vstupují všechny polygony v datasetu, a jejím výstupem je, že bod leží uvnitř jednoho mnohoúhelníku, vně všech polygonů, či na hraně/v uzlu dvou či více polygonů.

Techniky řešení tohoto problému se dělí na ty, které problém řeší u konvexních polygonů, a ty, které problém řeší u nekonvexních polygonů. Pro konvexní polygony se úloha Point in Polygon řeší algoritmy Ray Crossing Method (paprsková metoda, více v další části), či Half-Plane Test.

Half-Plane test porovnává řešený bod *q* s hranou polygonu, kdy řeší, zda se pokaždé nachází ve stejné polorovině. Pokud se pokaždé bod nachází ve stejné polorovině, pak leží uvnitř polygonu. Toto se dá aplikovat například

na triangulační síť.

Řešení úlohy u konvexních polygonů je značně jednodušší, nicméně v praxi se s setkáme spíše s polygony nekonvexními, kterými jsme se zabývali i v této úloze. Pro nalezení řešení pro nekonvexní mnohoúhelníky se využívají například algoritmy Ray Crossing Algorithm (paprskový algoritmus), či Winding Number Algorithm (metoda ovíjení).

## 3.2 Ray Crossing Algorithm

V případě paprskového algoritmu se ze zadaného bodu $q$ vede polopřímka $r$ (analogie paprsku, anglicky ray), často rovnoběžná s jednou z hlavních os, a počítá se, kolikrát protne hranu polygonu. Tyto průsečíky značíme písmenem $k$. Pokud je počet průsečíků sudý, bod leží vně polygonu, pokud je počet průsečíků lichý, bod leží uvnitř.

$$k \begin{cases} 1 & q \in P, \\ 0 & q \notin P. \end{cases} \tag{1}$$

Tento algoritmus je o poznání rychlejší než Winding Number (ten bude vysvětlen později), ale jeho problémem jsou případy, kdy bod leží buď na hraně, či ve vertextu polygonu. Tyto singularity se v kódu musí ošetřit zvlášť.
První zmínka o tomto algoritmu je z roku 1962 (Shimrat, 1962), a dnes je stále jedním z nejpoužívanějších algoritmů pro řešení problému Point in polygon.

## 3.3 Winding Number

Metoda Winding Number (metoda ovíjení) spočívá ve spočtení počtu ovinutí polygonu okolo bodu pomocí sčítání úhlů. Hodnota winding number (značené $\Omega$) je suma všech rotací měřená proti směru hodinových ručiček a udává vztah bodu a polygonu. Hodnoty $\Omega$ mohou nabývat hodnot násobků $2\pi$, nebo 0. V případě, že je hodnota winding number nenulová, bod leží uvnitř polygonu.

$$\Omega(q, P) = \begin{cases} 1 & q \in P, \\ 0 & q \notin P. \end{cases} \tag{2}$$

Úhel $\Omega$ je počítán na základě úhlu mezi dvěma směrovými vektory $\vec{u}$ a $\vec{v}$, kdy $\vec{u}$ spojuje dva body hrany, a $\vec{v}$ spojuje bod $q$ s prvním bodem hrany.

$$\vec{u} = (x_{p_{i+1}} - x_{p_i}, y_{p_{i+1}} - y_{p_i}) \tag{3}$$

$$\vec{v} = (x_q - x_{p_i}, y_q - y_{p_i}) \tag{4}$$

Hodnota $\Omega$ je závislá i na směru rotace. Úhly orientované proti směru hodinových ručiček jsou počítány záporně, úhly orientované po směru hodinových ručiček zase kladně. Toho se prakticky docílí tak, že přímka $p(p_i, p_{i+1})$, tedy přímka procházející hranou polygonu, dělí rovinu $\sigma$ na poloroviny $\sigma_l$ (levá polorovina) a $\sigma_r$. Pokud bod $q$ leží v polorovině $\sigma_l$, hodnota se přičítá, pokud v polorovině $\sigma_r$, hodnota se odčítá. Proto když procházíme body proti směru hodinových ručiček, $\Omega$ nám vyjde jako kladné číslo.

$$\Omega(q, P) = \frac{1}{2\pi} \sum_{i=1}^{n} \omega(p_i, q, p_{i+1}) \tag{5}$$

Nevýhodou tohoto algoritmu je jeho složitost zaprvé kvůli nutnosti předzpracování dat, a zadruhé kvůli časové náročnosti opakovaného výpočtu úhlu. Také kvůli práci s desetinnými čísly (při výpočtu úhlů) je možné dojít k chybě kvůli zaokrouhlování. Také je potřeba ošetřit některé singulární případy, například když bod leží ve vertexu polygonu, nebo když jej počítáme nad nejednoduchými polygony.

# 4 Popisy algoritmů formálním jazykem

V této kapitole jsou vloženy pseudokódy zpracovávaných algoritmů.

**Algorithm 2** Ray Crossing Algorithm s ošetřením singularit

**for** každý polygon **do**
  inicializace $kr = 0$ a $kl = 0$ ▷ průsečíky napravo a nalevo od bodu $q$
  posun souřadnicového systému do bodu $q$
  **for** každá hrana polygonu **do**
    **if** jeden z bodů hrany je totožný s $q$ **then**
      bod leží na vertexu polygonu
    **end if**
    **if** $y$ bodu $q$ leží mezi y-souřadnicemi dvou bodů hrany **then**
      spočti průsečík $k$
      **if** průsečík $k$ je napravo od bodu $q$ **then**
        $kr = kr + 1$
      **else if** průsečík $k$ je nalevo od bodu $q$ **then**
        $kl = kl + 1$
      **else if** průsečík je v bodě $q$ **then**
        bod leží na hraně
      **end if**
    **end if**
  **end for**
  **if** počet hran není dělitelný dvěma **then**
    bod leží uvnitř polygonu
  **else if** počet hran je dělitelný dvěma **then**
    bod leží vně polygonu
  **end if**
**end for**

**Algorithm 3** Winding Number Algorithm s ošetřením singularit

**for** každý polygon **do**
  inicializace $\Omega = 0$
  **for** každá hrana polygonu **do** ▷ hranou je myšlená dvojice bodů $p_i$, $p_{i+1}$
    **if** jeden z bodů $p_i$, $p_{i+1}$ je totožný s $q$ **then**
      bod leží na vertexu polygonu
    **end if**
    urči úhel $\omega$ svíraný vektory $\vec{u} = (x_{p_{i+1}} - x_{p_i}, y_{p_{i+1}} - y_{p_i})$ a $\vec{v} = (x_q - x_{p_i}, y_q - y_{p_i})$
    **if** $\omega = \pi$ **then**
      bod leží na hraně polygonu
    **end if**
    **if** $q$ leží v $\sigma_l$ **then**
      $\Omega = \Omega + \omega$
    **else**
      bod leží v pravé polorovině, $\Omega = \Omega - \omega$
    **end if**
    **if** $|\Omega| = 2\pi$ **then**
      bod leží v polygonu
    **else**
      bod neleží v polygonu
    **end if**
  **end for**
**end for**

# 5 Problematické situace a jejich rozbor + ošetření v kódu

## 5.1 Algoritmus Ray crossing

U tohoto algoritmu bylo nutné ošetřit následující problematické situace:

1. Hrana polygonu leží na polopřímce: Pokud je hrana polygonu rovnoběžná se směrem, kterým je vedena polopřímka (tedy v tomto případě s osou x), algoritmus nebude fungovat správně pro body, ležící na stejné souřadnici y. Tato situace je ošetřena tak, že algoritmus vybírá pouze průsečíky v horní nebo spodní polorovině od polopřímky. Ošetření této situace se objevuje v Algoritmu 1 na řádku $5^{??}$.

2. Bod leží na hraně polygonu: V takovém případě je možné nalézt průsečík s osou y po redukci na bod $q$ v bodě $q$, tedy v počátku souřadnicových os. V Algoritmu 2 je tento případ ošetřen na řádcích 14 a $15^2$. V případě, že dojde k této situaci se objeví v aplikaci varovná zpráva a dojde k zabarvení obou polygonů, které mají společnou tuto hranu.

3. Bod leží ve vrcholu polygonu: Tento problematický případ lze snadno ověřit při redukci soustavy souřadnic na bod q. Pokud v jakémkoliv bodu polygonu platí po redukci $x = y = 0$, pak bod $q$ leží v tomto bodu. V Algoritmu 2 je tento případ ošetřen na řádku $5^2$. V případě, že dojde k této situaci se objeví v aplikaci varovná zpráva a dojde k zabarvení všech polygonů, které mají společný tento bod.

## 5.2 Algoritmus Winding number

U tohoto algoritmu bylo nutné ošetřit následující problematické situace:

1. Polygon má vodorovnou hranu: Pokud má polygon vodorovnou hranu, která tedy není orientovaná směrem nahoru ani dolů, algoritmus v běžné podobě nebude fungovat. Pro tento případ byla vypracována vlastní podmínka, která spočítá, zda bod leží v levé nebo pravé polorovině, ve smyslu "nad"nebo "pod"hranou.

2. Bod leží na hraně polygonu: Tento případ pozná algoritmus tak, že v libovolném okamžiku se bude úhel, který svírají dva body polygonu při pohledu z bodu $q$ rovnat $\phi = \pi$. Za této situace se objeví v aplikaci varovná zpráva a dojde k zabarvení všech polygonů, které mají společný tento bod. V Algoritmu 4 je tento problém ošetřen na řádkách 8-9[3].

3. Bod leží ve vrcholu polygonu: Stejně jako u algoritmu Ray Crossing algoritmus Winding Number počítá pozice bodů v polygonu vůči zadanému bodu $q$. Stejným způsobem proto lze určit, zda má některý z bodů v polygonu stejné souřadnice. V případě, že dojde k této situaci, se objeví v aplikaci varovná zpráva a dojde k zabarvení všech polygonů, které mají společný tento bod. V Algoritmu 4 je tento problém ošetřen na řádkách 4-5[3].

4. Číslo $\pi$: Protože funkce $arccos()$ v jazyce Python není schopna vyhodnotit úhel tak, aby vrátila přesně číslo $\pi$, algoritmus musí počítat s $\pi$ zaokrouhleným (v tomto případě) na dvě desetinná místa. Jak při rozpoznávání bodu ležícího na hraně, tak při určování polygonu, ve kterém leží bod $q$ se tento problém vyskytuje.

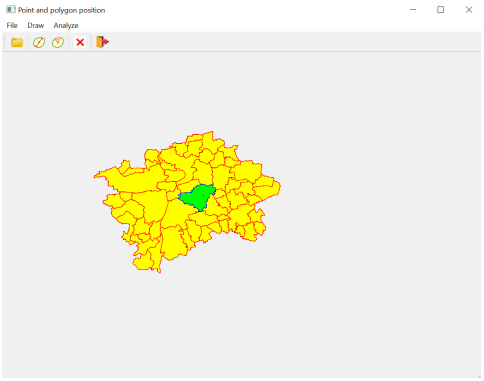
# 6 Vstupní data, formát vstupních dat, popis

Vstupní data pro analýzu jsou data pražských městských částí ve formátu shapefile.

# 7 Výstupní data, formát výstupních dat, popis

Výstupem programu aplikace umožňující načtení všech polygonů v datasetu a bodu, který uživatel určí kliknutím. Program v grafickém rozhraní QT zvýrazní polygon, ve kterém se analyzovaný bod nachází.

# 8 Printscreen vytvořené aplikace




# 9 Dokumentace: popis tříd, datových položek, jednotlivých metod

## 9.1 Třída Algorithms

Třída Algorithms se nachází ve scriptu algorithms.py a v této třídě jsou definovány metody getPolygonPositionR a getPolygonPositionW. Funkce getPolygonPositionR vrací zda se bod nachází vně/uvnitř/na hraně/ve vertexu polygonu pomocí paprskového algoritmu, funkce getPolygonPositionW pak to samé s využitím metody ovíjení.

## 9.2 Třída Draw

Tato třída se nachází ve scriptu draw.py a jejími parametry jsou self.__q (objekt typu QPointF, bod se dvěma souřadnicemi x a y, jehož poloha se bude vyšetřovat), self.__pol (objekt typu QPolygonF, který se bude vyšetřovat), self.__pols (seznam polygonů), self.polsNew. Tato třída má definované metody grafického rozhraní na vykreslování vstupů, výsledků, a interakce uživatel-grafické rozhraní. V této třídě jsou definovány metody mousePressEvent (mění polohu bodu $q$ na základě toho, kam uživatel klikne), input (načte data polygonů), paintEvent (umožňuje uživateli nakreslit bod), switchSource, getPoint (vrací objekt QPointF) a getPolygon (vrací seznam polygonů).

## 9.3 Třída Load

Tato třída se nachází ve skriptu load.py. V této třídě dochází k načtení dat ve formátu shapefile pomocí knihovny geopandas, a získání x a y souřadnic bodů, které tvoří polygony.

## 9.4 Třída Ui_Mainform

Tato třída se nachází ve scriptu mainform.py, který slouží ke spouštění programu a jednotlivých algoritmů. Tato třída byla navržena v rozhraní QT.

# 10 Závěr, možné či neřešené problémy, náměty na vylepšení

V rámci první úlohy z předmětu Algoritmy počítačové kartografie byl řešen problém Point in Polygon pomocí paprskového algoritmu a metody ovíjení. Kód byl psán v jazyce Python v prostředí PyCharm s využitím grafického rozhraní QT. V rámci obou algoritmů byly ošetřeny singulární případy, kdy bod leží na hraně polygonu či na jednom z vertexů.

Výstupem je aplikace, která analyzuje polohu bodu vůči vstupní vrstvě polygonů. V prvním kroku je třeba nahrát data. Jedná se o data pražských městských částí. Následně uživatel kliknutím myši vykreslí bod, jehož polohu chce analyzovat. Uživatel má možnost si zvolit, který z algoritmů chce pro analýzu využít.

K možným vylepšením: Stávající algoritmus je nastaven pro načítání jednoho konkrétního datasetu. Tento je přiložen k algoritmu. Vylepšení aplikace, které se nabízí, je umožnit uživateli nahrání vlastních prostorových dat (přímo z aplikace, nikoli díky změnám ve zdrojovém kódu) a upravit algoritmus načítání dat tak, aby se libovolná data v

aplikaci vhodně zobrazila.

Bylo by také možné dodat uživateli možnost vykreslovat vlastní polygony.

# 11 Seznam literatury

Bayer, Tomáš. 2023. "Point Location Problem." Praha, February 23. https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk3_new.pdf.

Heckbert, Paul S. 1994. Graphics Gems IV. Morgan Kaufmann.

Huang, Chong-Wei, and Tian-Yuan Shih. 1997. "On the Complexity of Point-in-Polygon Algorithms." Computers & Geosciences 23 (1): 109–18. https://doi.org/10.1016/S0098-3004(96)00071-4.

Ye, Yuan, Fan Guangrui, and Ou Shiqi. 2013. "An Algorithm for Judging Points Inside or Outside a Polygon." In 2013 Seventh International Conference on Image and Graphics, 690–93. https://doi.org/10.1109/ICIG.2013.140.