VÝPOČETNÍ GEOMETRIE

Petr Ryšavý

15. května 2025

Katedra počítačů, FEL, ČVUT

HLEDÁNÍ NEJBLIŽÍCH BODŮ V ROVINĚ

Příklad

UVA 10245 - The Closest Pair Problem

Hledání nejbližších bodů v rovině

Definice *Nechť $S \subset \mathbb{R}^2$ je množina bodů v rovině. Nalezněte $x, y \in S$ takové, že $x \neq y$ a platí*

$$a, b \in S : \|a - b\|_2 \geq \|x - y\|_2.$$

Naivní řešení

Naivní řešení

- Naivní řešení spočívá v iterování všech dvojic v $\mathcal{O}(n^2)$, kde n je počet bodů v S.

Řešení v $\mathcal{O}(n \log n)$

- Pokud bychom hledali nejbližší body v 1D, stačilo by seřadit body a projít seznam v $\mathcal{O}(n \log n)$
- Udělejme stejný trik ve 2D - seřaďme body podle x souřadnice
- Rozdělme je metodou *divide-and-conquer* podle svislé přímky v souřadnici x_d
- Nejbližší dvojice bodů je buď v levé polovině, v pravé . . . nebo poblíž dělicí přímky

Dvojice bodů poblíž dělicí hranice?

- Nechť je δ minimum ze vzdáleností dvojic bodů zleva a zprava
- Stačí nám projít body, které jsou v intervalu $\langle x_d - \delta, x_d + \delta \rangle$
- Pro každý bod spočteme vzdálenost k několika dalším

Pseudokód

function CLOSEST-PAIR(S)
    $P_x \leftarrow$ SORT-BY-X(S)
    $P_y \leftarrow$ SORT-BY-Y(S)
    return CLOSEST-PAIR-RECURSIVE(P_x, P_y)
end function

function CLOSEST-PAIR-RECURSIVE(P_x, P_y)
    $n \leftarrow$ LENGHT(P_x)
    if $n \leq 3$ **then return** CLOSEST-PAIR-BRUTE-FORCE(P_x)
    $x_d \leftarrow P_x[\lfloor \frac{n}{2} \rfloor].x$
    $\delta_l \leftarrow$ CLOSEST-PAIR-RECURSIVE($P_x[0 : \lfloor \frac{n}{2} \rfloor], [p \in P_y \mid p.x \leq x_d]$)
    $\delta_r \leftarrow$ CLOSEST-PAIR-RECURSIVE($P_x[\lfloor \frac{n}{2} \rfloor :], [p \in P_y \mid p.x > x_d]$)
    $\delta = \min\{\delta_l, \delta_r\}$
    $P'_y \leftarrow [p \in P_y \mid |p.x - x_d| < \delta]$
    for i **do**$\{0, 1, \ldots,$ LENGHT$(P'_y) - 1\}$
        for j **do**$\{i + 1, i + 2 \ldots, \min\{i + 7,$ LENGHT$(P'_y) - 1\}\}$
            $\delta \leftarrow \min\{\delta, \text{dist}(P'_y[i], P'_y[j])\}$
        end for
    end for

Asymptotická složitost

- Potřebujeme ukázat, že *merge* je v čase $\mathcal{O}(n)$

Asymptotická složitost

- Potřebujeme ukázat, že *merge* je v čase $\mathcal{O}(n)$
- Můžeme si rozdělit okolí bodu na stejně velké čtverce o velikosti $\frac{\delta}{2}$
- V každém čtverci je maximálně jeden bod (jinak by minimum zleva a prava nebylo δ
- Stačí nám projít tedy maximálně 7 bodů na vod z pásu $\langle x_d - \delta, x_d + \delta \rangle$
- Podle mistrovské věty (podobně jako merge sort nebo quick sort) je pak čas běhu

$$\mathcal{O}(n \log(n)).$$

KONVEXNÍ OBÁLKA

Příklad

UVA 218 - Moth Eradication

Hledání konvexní obálky

Definice *Nechť $S \subset \mathbb{R}^2$ je množina bodů v rovině.* ***Konvexní obálka*** *množiny S je nejmenší možná konvexní množina taková, že obsahuje S*

Neformálně řečeno, jde o natažený gumový provaz, který obepíná všechny body S.

Naivní řešení

Naivní řešení

- Začneme bodem nejvíce vlevo, ten je určitě v obálce
- Půjdeme bod po bodu a budeme hledat vždy bod, který musí být z konvexní obálky
- Podobné navíjení provázku na body (někdy se tomuto říká *provázkový algoritmus*)
- Běží až $\mathcal{O}(n^2)$

Zametací přímka

- Seřadíme body podle x-ové souřadnice
- Tvoříme „horní“ a „dolní“ obálku - potkají se v bodu nejvíce vlevo a nejvíce vpravo
- Uvažme horní obálku - přidáme vždy další bod a kontrolujeme, zda jde o konvexní útvar
- Případně vyhodíme existující bod
- Horní obálka se musí stáčet vpravo

Příklad

Jak ověřit zda je trojice bodů orientovaná vpravo

- Pomůže lineární algebra
- Uvažme trojici bodů p_{-2}, p_{-1}, p
- Nechť $(x_1, y_1) = p_{-1} - p_{-2}$ a nechť $(x_2, y_2) = p - p_{-1}$.
- Trojice je levotočivá (a ne kolineární), pokud

$$x_1 y_2 - x_2 y_1 > 0.$$

- (V následujícím slide je jiná, ale obdobná podmínka).

Pseudokód (Andrew's Monotone Chain)

```
function CONVEX-HULL(S)
    P ← SORT-LEXICOGRAPHICALLY(S)
    l ← empty stack                                          ▷ Dolní obálka
    for p doP
        while LENGHT(l) ≥ 2 ∧ CROSS(l[−2], l[−1], p) ≤ 0 do POP(l)
        end while
        PUSH(p, l)
    end for
    POP(l)                                  ▷ Abychom neměli poslední bod dvakrát
    u ← empty stack                                          ▷ Horní obálka
    for each p in REVERSE(P) do
        while LENGHT(u) ≥ 2 ∧ CROSS(u[−2], u[−1], p) ≤ 0 do POP(u)
        end while
        PUSH(p, u)
    end for
    POP(u)                                  ▷ Abychom neměli poslední bod dvakrát
    return CONCATENATE(l,u)
end function
function CROSS(o, a, b)
    return (a.x − o.x) · (b.y − o.y) − (a.y − o.y) · (b.x − o.x)
end function
```

References

- Tim Roughgarden's online courses,
  `http://theory.stanford.edu/~tim/videos.html`
- Tomáš Valla - Průvodce labyrintem algoritmů,
  `https://pruvodce.ucw.cz/static/pruvodce.pdf`
- `https://www.geeksforgeeks.org/convex-hull-monotone-chain-algorithm/`
- Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). Competitive Programming 3. Lulu Independent Publish.

Děkuji za pozornost.

Čas na otázky!